Eerik Saarinen

LOGIIKKAOHJELMALOHKOJEN MÄÄRITTELYOHJE JA KIRJASTOINTI SIEMENS TIA-PORTAL YMPÄRISTÖSSÄ

Sähkö- ja automaatiotekniikan koulutusohjelma

2018

LOGIIKKAOHJELMALOHKOJEN MÄÄRITTELYOHJE JA KIRJASTOINTI SIEMENS TIA-PORTAL YMPÄRISTÖSSÄ

Saarinen, Eerik
Satakunnan ammattikorkeakoulu
Sähkö- ja automaatiotekniikan koulutusohjelma
tammikuu 2018
Sivumäärä: 49
Liitteitä: 3

Asiasanat: ohjelmointi, dokumentointi, ohjelmoitavat logiikat, määrittely, digitaaliset kirjastot

---

Opinnäytetyön tavoitteina olivat suunnitella määrittelyohje logiikkaohjelman ohjelmalohkoille, suunnitella ja toteuttaa ohjetta noudattaen suoran moottoriohjauksen ohjelmalohko sekä luoda ohjeistusta Siemens:n TIA-Portaalin kirjastointiominaisuuksille.

Tarve työlle tuli yritykseltä ja heidän halusta yhtenäistää suunnittelijoidensa logiikkaohjelmasuunnittelua. Ohjelmalohkojen määrittelyohjeen avulla on tarkoitus suunnittelijat saada tekemään yhtenäisempää logiikkaohjelmaa. Kirjastoinnin avulla on tarkoitus tallentaa yleiskäyttöiset ohjelmalohkot tulevia projekteja varten, koska samanlaiset ohjelmaelementit toistuvat projektista toiseen.

Määrittelyohjetta pohjustettiin logiikkaohjelmointiin ja niiden suunnitteluun liittyvillä standardeilla sekä muulla kirjallisuudella. Kerätyn pohjamateriaalin avulla suunniteltiin määrittelyohje, jota käytettiin hyödyksi ohjelmalohkon kehitysprosessissa. Näiden ohella tutkittiin Siemens:n TIA-Portaalin tarjoamia kirjastointityökaluja ja luotiin ohjeistus niiden käyttämiseen.

Määrittelyohjetta apuna käyttäen on mahdollista suunnitella yleiskäyttöisiä ohjelmalohkoja, jotka voidaan kirjastoida tulevaa käyttöä varten. Määrittelyohje yhtenäistää yrityksen suunnittelijoiden ohjelmalohkojen suunnittelua, jolloin logiikkaohjelman sisältö on helpommin jokaisen suunnittelijan ymmärrettävissä ja suunnittelijoiden suunnittelemat ohjelmalohkot ovat paremmin yhteensopivia toistensa kanssa. Työssä toteutettu suoran moottoriohjauksen ohjelmalohko toimii yrityksen suunnittelijoille esimerkkinä, miltä määrittelyohjeen mukaisesti toteutettu ohjelmalohko näyttää ja mitä se sisältää. Yleiskäyttöiset ohjelmalohkot säästävät logiikkaohjelmasuunnitteluun kuluvaa aikaa ja tehostavat siten ohjelmatoimituksia.

DEFINITION GUIDELINES FOR PLC FUNCTION BLOCKS AND USE OF LIBRARIES IN SIEMENS TIA-PORTAL ENVIRONMENT

Saarinen, Eerik
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Electrical and Automation Engineering
January 2018
Number of pages: 49
Appendices: 3

Keywords: programming, documentation, programmable logics, definition, digital libraries

---

The purposes of this thesis were to design definition guidelines for function blocks of the plc program, design and implement direct-on-line motor starter program block which complies with the guidelines and create instructions for using the library functions of the Siemens' TIA-portal software.

The need for this thesis came from the company and their will to standardize designing of the plc program. The goal of the definition guideline is to get program designers to make more uniform plc program. By using libraries, the goal is to save function blocks for the future projects, because similar program elements are repeated from one project to another.

The basis for definition guideline was studied from plc programming related international standards and other literature. The definition guideline was designed based on studied literature and was used in development process of function block. In addition to these, the tools for managing libraries in Siemens' TIA-Portal software were studied and the instructions for using the libraries were created.

Using the defining guidelines is possible to design reusable function blocks, which can be stored in library for later use. The defining guidelines uniform function block design of engineers therefore contents of the plc program is easier to understand by all company engineers. In addition, the function blocks have better compatibility with each other. The function block of direct-on-line motor starter is an example for the engineers of the company. An example block demonstrates how are the definition guidelines visible in practical implementation. Reusable function blocks decrease the time used in designing of the plc program and thus make software deliveries more efficient.

SISÄLLYS

1 JOHDANTO 5
2 MÄÄRITTELYN TAUSTAA 6
3 OHJELMALOHKON MÄÄRITTELY 11
3.1 Lohkon ja instanssien nimeämisestä 12
3.2 Yleisiä ohjeita koodin rakentamiseen 12
3.3 Kohteiden nimeäminen 14
3.4 Notaatiosta 16
3.5 Kommentointi 17
4 OHJELMALOHKON KEHITYS 19
4.1 Suunnittelu 19
4.2 Toteutus ja testaus 22
5 OHJELMALOHKON KIRJASTOINTI 26
5.1 Tyyppiin liitettävän dokumentaation käsittely 27
5.2 Ohjeita projektikirjaston käyttöön 31
5.2.1 Ohjelmalohkon vienti projektikirjastoon 33
5.2.2 Kirjastoidun ohjelmalohkon muokkaus 35
5.2.3 Ohjelmalohkon julkaiseminen 37
5.3 Ohjeita globaalin kirjaston käyttöön 38
5.3.1 Uuden globaalin kirjaston luominen 38
5.3.2 Ohjelmalohkon vienti tai päivittäminen globaaliin kirjastoon 42
5.3.3 Ohjelmalohkon tuonti globaalista kirjastosta 44
5.4 Versionumeroinnista 45
6 JATKOKEHITYKSESTÄ 47
LÄHTEET 49
LIITTEET

1 JOHDANTO

Opinnäytetyön tilaajana oli ulvilalainen sähkö ja automaatiosuunnitteluun keskittyvä Piir-Group Oy. Yritys tarjoaa kokonaisvaltaista projektiosaamista ja ratkaisuja teollisuuden sähkö- ja automaatiosuunnitteluun. Yrityksen asiakkaina ovat muun muassa elintarvike-, metalli-, prosessi-, lämpövoima-, ja puunjalostusteollisuuden yritykset. (PIIR-GROUP Oy:n etusivu, 2017)

Työn aiheena oli suunnitella määrittelyohje logiikkaohjelman ohjelmalohkoille ja selvittää Siemens:n TIA Portal ympäristön kirjastointiominaisuuksia. Määrittelyohje toimii ohjeena yleiskäyttöisen ohjelmalohkon suunnittelua varten. Noudattamalla määrittelyohjetta ohjelmalohkon pitäisi olla suunniteltu siten, että se on yleiskäyttöinen ja sillä on tarvittava dokumentaatio. Samalla varmistetaan, että ohjelmalohko on sopiva kirjastoitavaksi tulevia projekteja varten. Työn aikana toteutettiin esimerkkinä toimiva suoran moottoriohjauksen ohjelmalohko, joka on suunniteltu määrittelyohjetta noudattaen.

Työn aihe tuli yrityksen halusta yhtenäistää logiikkaohjelmakoodin suunnittelua suunnittelijoiden kesken. Lisäksi ajatuksena oli saada alulle yrityksen sisäisen ohjelmalohkokirjaston rakentaminen TIA Portal-ympäristöön. Aiheet nousivat esille työskennellessäni yrityksessä kesäaikana.

Toimintatapojen yhtenäistäminen yrityksen sisällä tuo tehokkuutta logiikkaohjelman suunnitteluun. Määrittelyohjeen on tarkoitus yhtenäistää suunnittelijoiden ohjelmalohkosuunnittelua ja auttaa suunnittelemaan yleiskäyttöisiä ohjelmalohkoja. Ohjelmalohkojen kirjasto säästää aikaa tulevissa projekteissa, kun käytettävissä on valmiita testattuja ohjelmalohkoja. Projekteissa toistuvat usein samanlaiset logiikkaohjelman elementit, jolloin kirjastoidut yleiskäyttöiset ohjelmalohkot ovat todella hyödyllisiä.

2 MÄÄRITTELYN TAUSTAA

Koska työn tarkoituksena on yhtenäistää logiikkaohjelmakoodin suunnittelua, on oltava jonkinlainen määrittelyohje, jossa sovitaan yhteisistä käytännöistä. Tässä työssä keskitytään ohjelmalohkojen suunnitteluun, toisaalta ohjeistusta voidaan soveltuvilta osin käyttää ohjeena myös logiikkaohjelmalohkojen ulkopuolelle luotavaan koodiin. Ohjelmalohkot ovat logiikkaohjelman rakennusosasia, mitkä sisältävät itsenäisenä instanssina suoritettavan toiminta-algoritmin. Samaa algoritmin runkoa voidaan monistaa usealle instanssille logiikkaohjelmassa ja ohjelmamuutoksia voidaan hallita keskitetysti. Määrittelyohjeeseen on kirjattu asioita, joita ohjelmalohkon dokumentaatioon tulee sisältyä ja ohjeita siihen, miten ohjelmakoodia olisi hyvä rakentaa, jotta se pysyy yhtenäisempänä sekä selkeänä ja luettavana.

Suoraan valmista ja joka tilanteeseen sopivaa ohjeistusta koodin ja ohjelmalohkojen rakentamiseen ei ole olemassa, koska mahdollisia käytäntöjä toteutustapoja on monia. Lisäksi jokaisella suunnittelijalla omat mieltymyksensä, eli tapoja on lähes yhtä monta kuin on logiikkaohjelmakoodin kirjoittajiakin. Lähes kaikki sovellukset voidaan toteuttaa monella eri tavalla ja jokainen niistä voi olla aivan yhtä oikea tapa. Kuitenkin yrityksen sisällä toimintatapoja on hyvä yhtenäistää, jotta sovellusten kehitykseen saadaan parempaa tehokkuutta ja samoja ohjelmakomponentteja voidaan käyttää projektista toiseen.

Standardi SFS-EN 61131-3 määrittelee ohjelmoitavissa logiikoissa käytettävän kielen syntaksin ja semantiikan, mutta ei ohjeista siinä, miten kieltä hyödynnetään rakennettaessa logiikkaohjelmasovelluksia. Standardi SFS-EN 61131-8 antaa suuntaviivoja juuri siihen, miten standardoitua logiikkaohjelmakieltä olisi hyvä soveltaa. Lisäksi standardissa on esitetty mittareita järjestelmän laadun mittaamiseen, millä voidaan arvioida järjestelmän kykenevyyttä, saavutettavuutta, käytettävyyttä ja muunneltavuutta.

Logiikkaohjelmointistandardista SFS-EN 61131-3 on uusin painos julkaistu 2013, joka sisälsi uudistuksia ja lisäyksiä esimerkiksi datatyyppeihin ja tyyppimuutoksiin, ja kaiken muun lisäksi uutena asiana standardiin on liitetty olio-ohjelmoinnin periaatteet.

Valitettavasti uutta painosta standardista SFS-EN 61131-8 logiikkaohjelmakielen soveltamisesta ei ole vielä SESKO:n toimesta julkaistu vastaamaan uusinta logiikkaohjelmointikielen standardin painosta, vaan työn tiedot perustuvat vuonna 2003 julkaistuun jo melko vanhaan versioon. Kuitenkin vuoden 2017 marraskuussa IEC on julkaissut uuden teknisen raportin, joka on yhteensopiva aiemman version kanssa ja laajentaa sitä logiikkaohjelmastandardiin tulleiden uusien asioiden pohjalta. Valitettavasti en saanut päivitettyä teknistä raporttia käsiini opinnäytetyön teon aikana.

Standardi SFS-EN 61131-8 antaa melko laveat suuntaviivat logiikkaohjelmien toteuttamiseen ja standardissa mainitaankin, että myös muita kuin tässä standardissa käytettyjä tapoja ja käytäntöjä voidaan käyttää. Standardin linjaus antaa paljon vapauksia sovellusten toteutukseen, mikä on hyvä mutta samaan aikaan huono asia. Vapaus aiheuttaa helposti epäyhtenäisyyttä eri suunnittelijoiden toteutusten kesken, mutta kannustaa samalla vapaaseen ja innovatiiviseen ajatteluun. Liian tarkat rajaukset ohjelmasuunnittelussa saattavat tappaa suunnittelijoiden mielenkiinnon luoda täysin uutta ja kehittää vanhaa. Tärkeimmät ja eniten standardeissa korostuvat asiat liittyvät koodin uudelleenkäytettävyyteen, rakenteisuuteen ja paketointiin, koska näiden periaatteiden noudattaminen edesauttavat ohjelmakoodin luotettavuutta, huollettavuutta, helppolukuisuutta ja laajennettavuutta. Ohjelmakoodin kirjoittamiseen kuluu vähemmän aikaa, yritysten projektit tehostuvat ja järjestelmäsuunnittelijoille jää enemmän aikaa keskittyä järjestelmän toimintojen suunnitteluun, jopa luomaan kokonaan uudenlaisia sovelluksia. Standardien on myös tarkoitus yhtenäistää eri logiikkavalmistajien ratkaisuja, jotta logiikkaohjelmakoodi olisi samanlaista valmistajasta riippumatta ja helpommin siirrettävissä eri valmistajien kehitysympäristöjen välillä. Käytännössä koodin siirtäminen kehitysympäristöstä toiseen on vielä toistaiseksi melko hankalaa, ja toisaalta täydellistä yhteensopivuutta standardi ei vaadi.

Automaatiojärjestelmän toimitukseen kuluu selkeitä peräkkäisiä, mutta keskenään limittäisiä vaiheita esisuunnittelusta järjestelmän luovutukseen. Ohjelmalohkon suunnitteluun voidaan soveltaa samanlaista lähestymiskulmaa. Ohjelmalohkolla on tietynlaiset vaatimukset, jotka sen pitää toteuttaa, lisäksi pitää huomioida miten näiden vaatimusten toteutuminen testataan ohjelmalohkon ollessa valmis. Vaatimusten perusteella ohjelmalohkolle kirjoitetaan tarkka toiminnallinen kuvaus, josta selviää lohkon toiminta ja ominaisuudet. Ohjelmalohkon kuvauksen ollessa valmis, voidaan lähteä

toteuttamaan lohkoa ohjelmakoodin tasolla. Ohjelmalohkoa testataan ja sen toimintaa verrataan alkuperäisiin vaatimuksiin, jotta voidaan todeta lohkon haluttu toiminta. Logiikkaohjelmointiin liittyvä standardikin onneksi vaatii, että kehitysympäristössä pitää olla työkalut ohjelman testausta ja monitorointia varten. Tämä mahdollistaa ohjelman testaamisen virtuaaliympäristöissä mahdollisimman pitkälle, jopa oikeaa järjestelmää vastaavassa simulaatiomallissa, ennen sen käyttöönottoa oikeassa järjestelmässä. Ohjelmalohko otetaan käyttöön automaatiojärjestelmän yhteydessä, jossa voi vielä ilmetä uusia asioita ja lohkoon voi joutua tekemään pieniä muutoksia. Käyttöönoton jälkeen ohjelmalohko voidaan ajatella olevan valmis luovutettavaksi ja toimivan niin kuin pitääkin. Ohjelmalohko voidaan halutessa tallettaa myöhempää käyttöä varten. Toisaalta ohjelmakoodista on vielä pitkänkin ajan jälkeen mahdollista löytää pientä korjausta vaativia osia, vaikka nämä tulisi saada korjattua jo käyttöönotossa.

Työn tavoitteena on pyrkiä yhtenäistämään eri suunnittelijoiden tekemää logiikkakoodia, jotta samanlaiset asiat toteutettaisiin samantyylisellä tavalla suunnittelijasta riippumatta. Näin suunnittelija ymmärtää helpommin toisen suunnittelijan luomaa koodia, koska suuntaviivat ovat kaikille samat. PLCopen on valmistaja- ja tuoteriippumaton maailmanlaajuinen yhdistys, jonka tarkoituksena on ratkaista logiikkaohjelmointiin liittyviä asioita ja tukea kansainvälisten standardien käyttöä alalla. PLCopen on luonut teknisen raportin suositeltavista käytännöistä koodin kirjoittamisessa. Raportin tarkoituksena on antaa ohjeita millä tavoin ohjelmakoodin rakenteita ja tyylejä voidaan yhtenäistää, mutta jokainen organisaatio voi soveltaa ohjeita mielensä mukaan ja dokumentoida oman tyylioppaansa. Keskisuuret yritykset ovat olleet hyvinkin kiinnostuneita PLCopen:n koodaussuositusten käytöstä, koska heillä ei välttämättä ole sellaista vielä käytössä. Raportin suositusten perustuessa logiikkaohjelmointistandardiin on pelkästään hyvä, että standardia otetaan käyttöön yrityksissä ympäri maailmaa. (PLCopen etusivu, 2017)

Ohjelman objektien nimeäminen on hyvinkin vapaata. Nimeämiseen on ohjeistusta esimerkiksi käytettäviin muuttujien etuliitteisiin liittyen, mutta muutoin objektien nimet voivat olla lähes mitä tahansa. Kuitenkin nimeäminen olisi hyvä olla projektista toiseen yhtenäistä, esimerkiksi vakioituna termejä voitaisiin käyttää lohkosta toiseen, kun objektilla on samantapainen tehtävä. Kun jotakin termiä päätetään käyttää esimerkiksi tietynlaiseen tehtävään luodun muuttujan nimessä, tullaan tätä samaa nimeä

myös käyttämään tulevaisuudessa suunniteltavissa uusissa ohjelmalohkoissa. Standardi SFS-EN 81346 esittelee erilasia jäsentelykäytäntöjä, joilla voidaan jäsennellä laajat ja monimutkaisetkin kokonaisuudet. Jokaiselle projektin fyysiselle tai ei-fyysiselle kohteelle voidaan antaa yksilöllinen viitetunnus, joka muodostuu puumaisesta rakenteesta. Standardin avulla on mahdollista luoda erilaisia uudelleenkäyttöisiä moduuleja, joista voidaan helposti rakentaa yhtenäisiä suurempia kokonaisuuksia projektista toiseen. Tämä tehostaa huomattavasti tulevia projekteja, kun järjestelmän osasta on valmis testattu moduuli, joka sisältää kaiken tarvittavan. Moduuleja voidaan yhdistellä kokonaisuuksiksi ja järjestelmä rakentuu valmiiksi huomattavasti nopeammin, kuin vain yhteen sovellukseen räätälöidyllä järjestelmällä, jossa ei ole pohdittu järjestelmän osien uudelleenkäytettävyyttä. Kun tarkoituksena on luoda yleiskäyttöisiä projektista toiseen käytettäviä ohjelmalohkoja, näiden nimeämiseen viitetunnusstandardi ei oikein sovellu, eikä ole kovin havainnollinen. Ohjelmalohkon toimintaa kuvaava lyhyehkö nimi kertoo havainnollisemmin lohkon tarkoituksesta, kuin usean kirjaimen sarja. Sähköalaan liittyvissä standardeissa käytettävää termistöä on mahdollista etsiä esimerkiksi sivustolta electropedia.org, joka on IEC:n ylläpitämä termitietokanta.

Tarkoituksena oli luoda ohjelmalohko suoraa moottoriohjausta varten, joten tutustuin SFS-käsikirja 16:ta, joka esittelee vakiosovelluksia enintään 1000V moottorikäytöille. Käsikirjassa käsitellään moottorilähtöjen vaatimuksia, toimintoja ja esimerkkejä niiden toteutuksista. Käsikirjaan on tiivistetty useamman standardin vaatimukset, jotka koskevat moottorikäyttöjä. Käsikirjan esimerkit eivät ole velvoittavia, vaan sovelluksia voidaan rakentaa myös muilla tavoilla, kunhan standardien vaatimuksia noudatetaan. Käsikirjaa apuna käyttäen on mahdollista helposti luoda yrityksen sisäinen vakioitu toimintatapa, jolla moottorilähtöjä suunnitellaan ja toteutetaan. Käsikirjan esittämät vaatimukset antavat hyvää pohjaa logiikkaohjelmalohkon toiminnallisille vaatimuksille etenkin koneturvallisuuteen liittyvissä asioissa. Ohjausjärjestelmän esimerkit koostuvat kaksiosaisesta älykkäästä järjestelmästä. Ohjausjärjestelmästä lähetetään moottoriohjauskeskukselle sarjaliikennettä hyödyntäen ohjaussanomat, ja paluuviestinä moottorikeskukselta saadaan tilasanomat. Sanamuotoiset sanomat puretaan moottorikeskuksen etäyksikössä ja keskuksen moottorien ohjaukset suoritetaan sanoman sisällön mukaisesti. Esimerkin toiminta vastaa väylään liitetyn taajuusmuuttajan tai moottoriohjaimen ohjaamista, mikä ei ole tämän työn ydinsisältöä.

Opinnäytteen tilaajalla ei aikaisemmin ole ollut selkeää ohjeistusta tai yhteisiä sovittuja toimintatapoja logiikkaohjelmakoodin suunnittelusta ja sen toteuttamisesta. Tutustuin yrityksen aikaisempiin projekteihin etsimällä niistä tunnusomaisia elementtejä saadakseni kuvan siitä, miten logiikkaohjelmia on aikaisemmin rakennettu. Ohjelmien suunnittelu on nojautunut tähän saakka yksittäisten suunnittelijoiden ammattitaitoon ja näkemykseen. Uskon yrityksien ja suunnittelijoiden ottavan uudenlaiset asiat vastaan paremmin, kun heillä on mahdollisuus vaikuttaa tuleviin uudistuksiin tai muutoksiin. Esittelin heille suunnittelemani määrittelyohjeen, perustelin tekemiäni valintoja ja pyysin heiltä kommentteja määrittelystä. Päivitin määrittelyohjetta saamieni kommenttien pohjalta ja lähdin sen pohjalta kirjoittamaan esimerkkilohkon kuvausta. Pyysin suunnittelijoita tutkimaan ja kommentoimaan myös ohjelmalohkon kuvausta saadakseni selville, olemmeko keskenämme samoilla linjoilla.

Ohjelmalohkon määrittelyohjetta rakennettaessa on sovellettu näitä lähteitä sekä omaa kokemustani, opintojeni aikana oppimiani asioita ja hyväksi havaittuja toimintatapoja. Tarkoituksena on kirjoittaa määrittelyohje yleiskäyttöisen ohjelmalohkon näkökulmasta, joten ohjeessa on painotettu enemmän näihin olennaisesti liittyviä asioita. Ohjelmalohkon ympärillä olevaan ohjelmaan voidaan myös soveltaa näitä ohjeita, mutta näiden lisäksi voidaan tarvita lisämäärittelyitä.

Uudelleenkäytettävällä ohjelmalohkolla on oltava hyvä ajantasainen dokumentaatio ja hyvä olla ohjeistusta siitä, miten sellainen suunnitellaan ja toteutetaan. Tällöin jokainen ohjelmalohkoa käyttävä tai kehittävä suunnittelija pystyy nopeasti ymmärtämään lohkon toiminnan. Tämä mahdollistaa vanhojen kirjastoitujen ohjelmalohkojen käytön uusissa projekteissa, jolloin logiikkaohjelman rakentaminen nopeutuu huomattavasti. Ohjelmakoodin testaukseen ja virheiden etsimiseen kuluva aika lyhenee jo valmiiksi testatun ja toimivan koodin vuoksi. Hyvä dokumentaatio on myös edellytys ohjelmalohkolle tehtävälle tulevaisuuden kehitykselle.

3 OHJELMALOHKON MÄÄRITTELY

Määrittelyn ohjelma-arkkitehtuurisena ajatuksena on, että logiikkaohjelma koostuu puumaisessa rakenteessa olevista järkevän kokoisista ohjelmalohkoista. Ohjelmalohkojen rakenteen olisi hyvä seurata järjestelmän fyysistä kokoonpanoa. Ohjelman rakentuessa pienemmistä kokonaisuuksista sen luettavuus on parempi. Laiteohjauslohko sisältää laitteen toiminnan suorittavan toiminta-algoritmin. Laiteohjauslohkon sisällä on toimilaiteohjauslohko, jonka tehtävänä on ohjata logiikan HW:hen liitettyä toimilaitetta juuri sillä tavalla kuin laiteohjauslohko haluaa sitä ohjattavan. Toimilaitelohko toimii laiteohjauslohkon orjana ja suorittaa toimilaitteen ohjauksen vain silloin kuin laiteohjauslohko niin haluaa. Laiteohjauslohkolla voi olla erilaisia toimintatiloja, mutta toimilaiteohjauslohkolla ei tarvitse olla tietoa laitteen eri käyttötiloista.

Yleiskäyttöisen ohjelmalohkon suunnittelu aloitetaan sille asetettavien vaatimusten selvittämisellä. On selvitettävä, miten lohko liittyy ylempään lohkoon, logiikan tuloihin ja lähtöihin sekä millaisia toimintoja tai ominaisuuksia siltä vaaditaan. Parhaassa tapauksessa käytettävissä on kuvaus suunniteltavaa lohkoa ylemmän tason lohkon rajapinnasta ja sen toiminnasta. Lohkon vaatimuksia laadittaessa tulee pohtia, miten vaatimuksien täyttyminen todetaan ja testataan.

Ohjelmalohkon vaatimusten ollessa selvillä, voidaan lähteä suunnittelemaan lohkon kuvausta. Kuvaukseen sisältyy toiminnankuvaus, lohkon liitynnät ja parametrit. Toiminnankuvauksessa esitetään, miten kaikki lohkolle asetetut vaatimukset toteutetaan, miten ohjelmalohko toimii ja mitä ominaisuuksia se sisältää. Myös toiminnankuvausta kirjoittaessa tulee pohtia, miten jokainen lohkon toiminta ja ominaisuus testataan, jotta lohko toimii oikein myös virhetilanteissa. Ohjelmalohkon liityntä tulee kuvata yksityiskohtaisesti. Listattuna tulee olla lohkon tulot, lähdöt ja parametrit sekä niiden nimet ja tarkemmat kuvaukset, joita käytetään koodin myös kommenteissa. Lohkon liityntä määritellään ja kirjoitetaan myös koodissa seuraavassa järjestyksessä: SW-liitynnät, HW-liitynnät ja parametrit. Ohjelmointiympäristön ominaisuudet voivat vaikuttaa siihen, kuinka hyvin liitynnän järjestystä pystyy noudattamaan, tällöin käytetään järjestystä liityntätyypin sisällä. Ohjelmalohkon kuvauksen yhteyteen kirjataan myös lyhy-

esti ohjelmalohkon käyttötarkoitus, josta selviää nopeasti vilkaisemalla lohkon käyttökohteet. Kiteytetysti ohjelmalohkon yhteyteen liitettävästä dokumentaatiosta tulisi löytyä seuraavat asiat:

1. Ohjelmalohkon käyttötarkoitus
2. Ohjelmalohkon kuvaus
    a. Toiminnankuvaus
    b. Lohkon tulot
    c. Lohkon lähdöt
    d. Lohkon tulo-lähdöt
    e. Lohkon parametrit
        i. Ulkoiset
        ii. Sisäiset

3.1 Lohkon ja instanssien nimeämisestä

Yleiskäyttöinen ohjelmalohko nimetään hyvin lohkoa kuvaavalla ytimekkäällä nimellä ja joissakin tilanteissa apuna voidaan käyttää tunnettuja lyhenteitä. Eli ohjelmalohkokirjastoon vietävää ohjelmalohkoa ei nimetä käyttäen viitetunnusstandardia. Kun ohjelmalohkosta luodaan suoritettava instanssi logiikkaohjelmaan, voidaan se nimetä viitetunnusstandardin nimeämiskäytäntöä käyttäen. Yleiskäyttöisen ohjelmalohkon instanssi usein kannattaakin nimetä vastaavalla tavalla kuin projektin fyysisen laitteen dokumentaatiossa. Nimeämisessä käytetään yleisesti alalla käytössä olevia termejä.

3.2 Yleisiä ohjeita koodin rakentamiseen

Tarkoituksena on yhtenäistää yrityksen suunnittelijoiden kirjoittamaa ohjelmakoodia ja näitä ohjeita noudattamalla yhtenäisyyttä viedään hieman eteenpäin.

Ohjelmalohkon erilaiset tilat ja häiriöt paketoidaan käyttäjän määrittelemään datatyyppiin. Lisäksi siihen voidaan liittää SW puolen liityntöjä ja erilaisia parametreja. Paketoimalla muuttujat käyttäjän määrittelemään datatyyppiin, yksinkertaistetaan ja selkeytetään lohkon liityntää.

Ohjelmalohkosta ei saa olla suoria absoluuttisia osoituksia sen ulkopuolelle. Kaikki tarvittava tieto tuodaan ja viedään lohkon sisälle ja ulos liityntöjen kautta. Ohjelmalohkoissa ei käytetä ulkoisia muuttujia, jotta lohkon uudelleenkäytettävyys säilyy projektista toiseen. Pyritään siihen, että lohkot eivät vaatisi jotakin tiettyä globaalia muuttujaa, ei edes liitynnän kautta tuotavana.

Kaikki muuttujat tulee alustaa tilanteeseen sopivalla tavalla. Usein ohjelmointiympäristö tekee alustuksen automaattisesti, mutta erikoistapauksissa muuttuja voidaan alustaa poikkeavalla arvolla.

Muuttujien muistialueet eivät saa mennä päällekkäin, jotta vältytään vääränlaisen datan lukemiselta ja kirjoittamiselta.

Jos funktion suorituksessa tapahtuu virhe, tulee virheellisen tuloksen käyttö estää, jotta vältytään virheellisiltä toiminnoilta.

Ohjelmalohkoa ei kutsuta itsensä sisällä, koska rekursiivinen käsittely voi aiheuttaa helposti järjestelmävirheen, eikä ole kaikilla kehitysalustoilla tuettuna.

Ohjelmalohkosta voidaan poistua vain yhdestä kohtaa. Jos ohjelmasta halutaan poistua useammasta kohdasta koodia, voidaan hypätä RETURN-avainsanan sisältävälle riville. Kuitenkin ohjelmalohkon koodissa RETURN-avainsana esiintyy vain kerran.

Input, output, input-output muuttujat taikka parametrit tulee käsitellä niiden toiminnan mukaisesti. Input luetaan kerran syklillä ja sitä ei kirjoiteta. Output kirjoitetaan ainakin kerran syklillä. Input-output sekä luetaan, että kirjoitetaan koodissa syklillä.

Logiikkaohjelmointistandardinkin mukaisesti koodi on vahvasti tyypitettyä, joten jokaisen muuttujan pitää olla tyypitetty joksikin tietotyypiksi. Valitaan kuhunkin tilanteeseen sopiva tietotyyppi ja muunnetaan se tarvittaessa toiseen sopivampaan tietotyyppiin. Muuttujan tietotyypin pitää muuttua sen mukaan, miten sitä käsitellään. Jos datassa ei ole negatiivisia arvoja, käytetään tällöin etumerkitöntä tietotyyppiä, kuten

unsigned integer (uint). Tällä pystytään estämään negatiivisista arvoista johtuvat ongelmat joidenkin toimintojen tai funktioiden kohdalla. Tarvittaessa voidaan luoda oma tilanteeseen sopivampi tyyppi.

Muuttujien tyyppimuunnokset pitää olla yksiselitteisiä eli eksplisiittisiä, jotta kehitysympäristön koodikääntäjän ei tarvitse arvata operaatiossa käytettäviä tietotyyppejä. Yksiselitteisillä tyyppimuunnoksilla varmistetaan se, että muuttujien operaatiosta tulee tulos, jonka koodinkirjoittaja on halunnut.

Hyppykomentoja ei käytetä LADDER ja FBD ohjelmointikielillä. Tekstipohjaisissa kielissä koodissa ei saa olla hyppyjä taaksepäin, poikkeuksena tähän ovat kuitenkin silmukat.

Ohjelmalohkon instanssia kutsutaan vain yhden kerran ohjelmasyklillä. Tästä voidaan erikoistapauksissa poiketa, jos lohkon toiminta on sellainen, että sitä on tarve kutsua useita kertoja yhdellä ohjelmasyklillä.

Käytetään TEMP-muuttujaa, kun muuttujaa tarvitaan vain ohjelmalohkon suorituksen ajan.

Ohjelmalohkon liitynnässä voi olla tulo, lähtö tai tulo-lähtö liityntöjä enintään noin kymmenen kutakin tyyppiä. Jos liityntöjä tulisi enemmän, kannattaa ne paketoida käyttäjän määrittelemään datatyyppiin.

Jos muuttujan arvoa ei tarvitse päästä muuttamaan kuin esimerkiksi käyttöönotossa, voidaan muuttuja pitää ohjelmalohkon sisäisenä muuttujana. Lohkon sisällä aseteltavat parametrit kerätään koodin alkuun.

3.3 Kohteiden nimeäminen

Ohjelmakoodin kohteiden (esim. funktio, ohjelmalohko ja muuttuja), nimeämisessä pyritään siihen, että käytetään yleisesti käytössä olevia tai jo käytettyjä termejä. Kun

aikaisemmin toteutetussa lohkossa on päätetty käyttää sisältöä kuvaavaa nimeä muuttujalle, käytetään tätä samaa muuttujanimeä uudessa suunniteltavassa lohkossa, jos muuttujien sisältöjen ominaisuudet ovat lähellä toisiaan. Kohteen nimi tulee kuvata sisältöään ja samalla nimi on jo itsessään kommentoiva.

Nimeämisessä käytetään englannin kieltä, kansallisten erikoiskirjainten aiheuttaminen ongelmien välttämiseksi.

Kohteiden nimissä ei saa käyttää mitään ohjelmakoodin syntaksiin liittyviä avainsanoja, koska tämä voi aiheuttaa virheitä koodiin ja ongelmia koodin kääntäjälle.

Etuliitteettömät kohteiden nimet kirjoitetaan tyylillä UpperCamelCase ja etuliitteelliset kirjoitetaan tyylillä lowerCamelCase, jotta nimen luettavuus pysyy hyvänä.

Ohjelmalohkon sisäiset muuttujat eivät saa olla samanlaisia jonkin globaalin muuttujan kanssa eikä samaa muuttujanimeä saa käyttää kahdella eri kirjasinkoolla esim. Muuttuja ja muuttuja. Tällä vältytään epäselvistä tilanteista, viitataanko koodissa globaaliin vai lohkon paikalliseen muuttujaan.

Käytetään keskimäärin maksimissaan 15 merkkiä pitkiä kohteiden nimiä, mutta erikoistapauksissa voidaan maksimissaan käyttää 25 merkkiä pitkiä. Kohteen nimen tulisi olla vähintään 3 merkkiä pitkä, mutta mielellään yli 8 merkkiä. Poikkeuksena vähimmäisvaatimukseen on esim. silmukoiden indeksit. Kohteen nimellä tulee olla sopivasti mittaa, jotta sillä on edes mahdollisuus kertoa merkityksellistä tietoa kohteen sisällöstä.

Kohteen nimi ei saa alkaa numerolla tai alaviivalla. Nimessä käytetään englanninkielen aakkostoa ja numeroita. Tarvittaessa voidaan käyttää joitakin erikoismerkkejä ASCII 7-bittisestä merkistöstä. Kansallisten erikoisaakkosten ja erikoisten merkkien käyttäminen tuottaa usein ongelmia ohjelmointiympäristöissä.

3.4 Notaatiosta

Lohkon liityntään tulevat muuttujat erotetaan lohkon sisäisistä muuttujista etuliitteillä:

- Tulot, i_
- Lähdöt, o_
- Tulo-lähdöt, io_
- Parametrit, p_

Lohkon liitynnän muuttujat ovat helposti erotettavissa lohkon sisäisistä muuttujista, lisäksi koodin kirjoittaminen nopeutuu.

Koska logiikkaohjelmakoodi on vahvasti tyypitettyä, liitetään muuttujan nimen etuliitteeksi lyhenne datatyypistä taulukon 1 mukaisesti. Kun datatyyppinä on käyttäjän määrittelemä datatyyppi (user defined datatype), käytetään näistä yhteisesi etuliitettä udt.

Taulukko 1 Datatyyppien etuliitteet

Datatyyppi	Etuliite
BOOL	b
SINT	si
INT	i
DINT	di
LINT	li
USINT	usi
UINT	ui
UDINT	udi
ULINT	uli
REAL	r
LREAL	lr
TIME	tim
LTIME	ltim
DATE	dt
LDATE	ldt
TIME_OF_DAY / TOD	tod

LTIME_OF_DAY / TOD	ltod
DATE_AND_TIME / DT	dt
LDATE_AND_TIME / DT	ldt
STRING	str
WSTRING	wstr
CHAR	c
WCHAR	wc
BYTE	by
WORD	w
DWORD	dw
LWORD	lw

Esimerkiksi lohkon reaaliluku-tyyppinen tulomuuttuja on edellä mainitun perusteella: i_rVariable ja lohkon sisäinen integer-tyyppinen muuttuja iVariable.

Kun ohjelmalohkon häiriöitä halutaan eritellä hälytyksiksi ja varoituksiksi, erotetaan nämä toisistaan etuliitteillä: Hälytykset (Alarms) a_ ja varoitukset (Warnings) w_. Häiriöt listataan myös tässä järjestyksessä esimerkiksi käyttäjän luomaan datatyyppiin. Pääsääntönä voidaan pitää, että hälytystason häiriö lopettaa ohjauksen, mutta varoitus ei lopeta ohjausta, vaan toimii ennakoivana ilmoituksena.

3.5 Kommentointi

Ohjelmakoodin sisältö itsessään pitäisi olla itseään kommentoivaa, esimerkiksi muuttujien nimet tulee nimetä sisältöään kuvaavasti. Yksinkertaistettuna ohjelmakoodin yhteyteen kirjoitetun kommentin tulisi vastata kysymykseen: Mikä on koodin tarkoitus?

Kun kirjoitetaan tekstipohjaista ohjelmakoodia, kommentoidaan käyttäen yksirivisiä kommentteja, koska rajatun kommentin käyttö lisää riskiä kirjoitusvirheiden aiheuttamille ongelmille ja saattaa rajata tarpeellista koodia vahingossa suoritettavan osan ulkopuolelle. Pääsääntönä kommentti kirjoitetaan ennen koodilohkoa esim. IF .. THEN .. ELSE IF .. ELSE, koska jokaisen koodirivin kommentointi ei ole välttämätöntä.

Graafisissa ohjelmointikielissä käytetään ohjelmointiympäristön tarjoamia kommenttikenttiä.

Kommentointi tehdään ensisijaisesti suomeksi tai englanniksi. Projektista ja ohjelmointiympäristön riippuen kommentit voi olla muullakin kielellä tai kommentit on käännetty useammalle kielelle. Tekstipohjaisissa ohjelmointikielissä kommentointikielenä on suositeltavinta käyttää englantia.

Valmiissa tai kirjastoon siirrettävässä ohjelmalohkossa ei saa olla ohjelmakoodia kommenttien sisällä, jotta tahalliset tai tahattomat muutokset kommentteihin eivät aiheuta muutoksia lohkon toimintaan.

4 OHJELMALOHKON KEHITYS

Opinnäytetyön yhtenä tarkoituksena oli suunnitella ja toteuttaa määrittelyohjeeseen pohjautuvana esimerkkinä suoran moottoriohjauksen logiikkaohjelmalohko. Ohjelmalohkolla on tarkoitus havainnollistaa määrittelyn vaikutusta, mitä asioita lohkolta vaaditaan käytännössä ja miten ohje vaikuttaa toteutukseen.

4.1 Suunnittelu

Ohjelmalohkon suunnittelu pystyi alkamaan kunnolla vasta kun määrittelyohje oli valmistunut. Kun määrittelyn eteneminen ja ohjeistus olivat selvillä, kävimme yrityksen suunnittelijoiden kanssa yhdessä lävitse, minkälaisia vaatimuksia ohjelmalohkolle asetetaan.

HW-liityntöjä varten pyysin yritykseltä yleisesti käytettyjä moottoriohjauksen piirikaaviota. Pyrin etsimään piirikaavioita, joissa olisi mahdollisimman monipuolisesti erilaisia komponentteja, joiden tietoja mahdollisesti tuodaan logiikalle. Toisena lähteenä käytin SFS-käsikirja 16 moottorilähtöjen mallipiirikaavioita. Lohkon HW-liityntään tulee toimilaitteen tilatietojen lisäksi tiedot myös moottoriohjauskeskuksen häiriöistä, koska ne ovat olennaisena osana moottorin ohjauskytkentää. Ohjelmalohkoon on helppo rakentaa hieman laajempi liityntä kuin aina on edes tarvetta. Tällöin samaa lohkoa voidaan käyttää sekä yksinkertaisissa että monimutkaisissa projekteissa.

Ohjelmalohkon ympärille ei ollut laiteohjauslohkoa tai projektia, joten SW-liitynnöille ei ollut aivan yhtä selkeitä suuntaviivoja kuin HW:n liitynnälle. Suunnittelun perusajatuksena kuitenkin on, että tarkoituksena on suunnitella ohjelmalohko, joka toimii toisen lohkon orjana. Lohkon SW-liityntä suunnitellaan laiteohjauslohkoa varten, jotta laiteohjauslohko voi ohjata toimilaitelohkoa ja saa siltä haluamansa tiedot sen toiminnasta.

Ohjelmalohkolle kirjattiin seuraavia vaatimuksia:

- Ohjaa kontaktoreilla ohjattavaa moottoria
- Moottoria voidaan ohjata kahteen suuntaan ja hoidettava suunnanvaihtotilanteet

- Voidaan käyttää myös yksisuuntaisissa ohjauksissa
- Saatava tietoa moottorin tiloista ja virheistä.
- Ohjataan jatkuvalla ohjauksella
- Kehitysalustana S7-1200-sarjan CPU

Näiden vaatimusten lisäksi kirjattiin liitynnän vaatimukset.
HW-liityntä:

- Takaisinkytkentätiedot molemmista käyntisuunnista kontaktoreilta tai moottorin anturilta.
- Moottorisuojakytkimen, lämpöreleen ja mahdollisesti oikosulkusuojan tilatiedot.
- Turvakytkimen tilatieto
- Moottorin termistorien releen tilatieto.
- Moottorikeskusvika-apureleen tilatieto
- Kenttävika-apureleen tilatieto.
- Hätäseispiirin tilatieto.
- Moottorien kontaktorien ohjaukset yksi lähtö per suunta.
- Moottorin jarrun ohjaus.

SW-liityntä:

- Liityntä laiteohjauslohkolta
    - Moottorin ajo eteen
    - Moottorin suunnan valinta taakse.
    - Häiriöiden kuittaus
- Liityntä laiteohjauslohkolle
    - Vikadiagnostiikka ja moottorin tilatiedot.

Vielä ennen kuvauksen suunnitteluun ja kirjoittamiseen siirtymistä, pohdin minkälaisia testejä ohjelmalohkolle tulisi tehdä, jotta kirjattujen vaatimusten toteutuminen voidaan todeta.

Kun vaatimukset olivat selvillä, ohjelmalohkon kuvausta oli mahdollisuus lähteä suunnittelemaan määrittelyohjeiden mukaisesti. Ohjelmalohkon kuvaus on tarkasteltavissa

liitteessä 1. Aloitin lohkon suunnittelun kirjoittamalla sen toiminnankuvauksen. Toiminnankuvauksen kirjoittamisen yhteydessä pohditaan laajasti ja tarkasti lohkon toimintaa, jolloin saattaa herätä tarpeita esimerkiksi uusille tiloille tai parametreille. Toiminnankuvauksen kirjoittamisen jälkeen oli hyvä siirtyä kuvaamaan ohjelmalohkon liitynnät, tilat ja parametrit.

Lohkolla on kaksi ajokäskyä, joista ensimmäisen ollessa päällä moottoria ajetaan eteen ja molempien ollessa päällä moottoria ajetaan taakse. Toiminta estää epäselvät tilanteet, missä moottoria mahdollisesti ohjattaisiin molempiin suuntiin samanaikaisesti. Toteutuva suunta voisi olla epävakaa ja riippua siitä, miten koodi suoritetaan logiikassa.

Moottori voidaan käynnistää viiveellä moottorin jarrun avautumisen jälkeen, jos esimerkiksi jarrun avautumisessa on viivettä. Jarrun kiinnikytkeytymiselle voi asettaa viivettä, jos ohjattavana on laite, jossa on suuria massahitauksia tai jarrua ei ole välttämättä tarvetta kytkeä heti kiinni moottorin sähkönsyötön poistuessa.

Ohjelmalohkon liityntää suunnitellessa tausta-ajatuksena oli, että lohkon tilalle voitaisiin sijoittaa esimerkiksi taajuusmuuttajaa ohjaava lohko. Eli lohkojen liitynnöillä tulisi olemaan yhteisiä elementtejä, jolloin laiteohjauslohkon sisälle voidaan valita toimilaitteesta riippuen suora moottoriohjaus- tai taajuusmuuttajalohko.

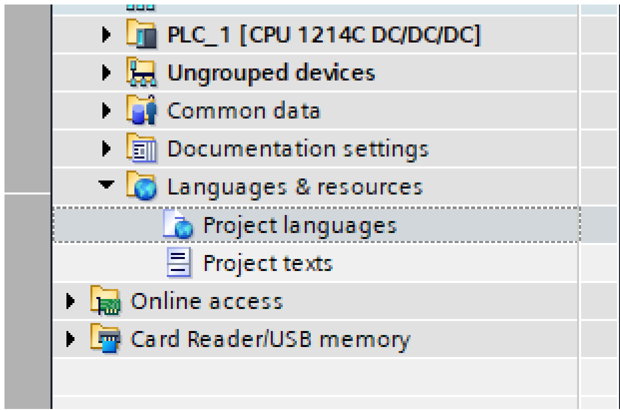
Ohjelmalohkon tilat ja häiriöt on paketoitu käyttäjän määrittelemään datatyyppiin eli UDT:hen. Samanlaiset asiat on hyvä paketoida, jolloin niitä on helpompi käsitellä ja lohkon liityntä pysyy siistimpänä.

Suunnanvaihtoviiveen jätin ulkoiseksi parametriksi, jotta sitä on mahdollista muuttaa myös logiikkaohjelmaa ajettaessa. Muut parametrit on tarkoitus hakea kohdilleen käyttöönotossa ja muokata logiikkaohjelmakoodiin. Toiminnankuvauksen kirjoittamisen yhteydessä myös pohdin ja kirjasin ylös testejä, joita pitää suorittaa toimivuuden toteamiseksi.

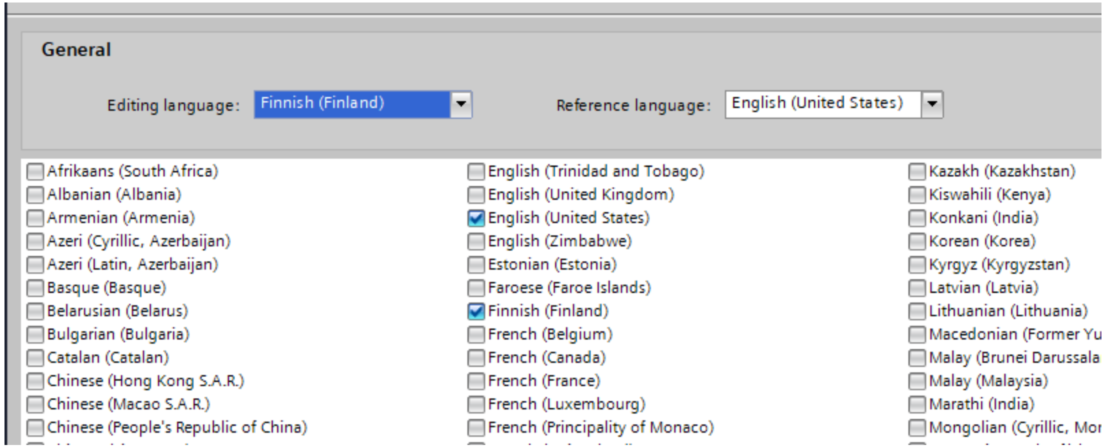
4.2 Toteutus ja testaus

Ohjelmalohko toteutettiin Siemens:n TIA Portal-kehitysympäristössä ja ohjelmistosta oli käytössä V14 SP1 Update 2 versio. Ohjelmalohkon toimintaa simuloitiin Siemens:n S7-PLCSIM ohjelmistolla ja sen V14 versiolla.

Ohjelmalohkon toteutus alkoi uuden projektin luomisella ja sen konfiguroinnilla. Koska ohjelmalohkon kommentit on tarkoitus kirjoittaa suomen kielellä, on projektin kieliasetuksista muutettava muokkauskieleksi suomi. Kieliasetusten ollessa oikein, kommenttikenttiin kirjoitettava teksti tallentuu suoraan oikean kielen alle. Kehitysympäristön toiminta perustuu projektikohtaiseen tietokantaan, mihin projektin tekstit tallentuvat kieliasetusten mukaisesti omiin taulukoihinsa. Projektin tekstejä on mahdollista muokata ja siirrellä keskitetysti jälkeenpäin, mutta oikeat kieliasetukset säästävät ylimääräiseltä työltä. Kuvassa 1 projektipuusta löytyvät kieliasetukset ja tekstityökalu, sekä kuvassa 2 projektiin valitut kieliasetukset.

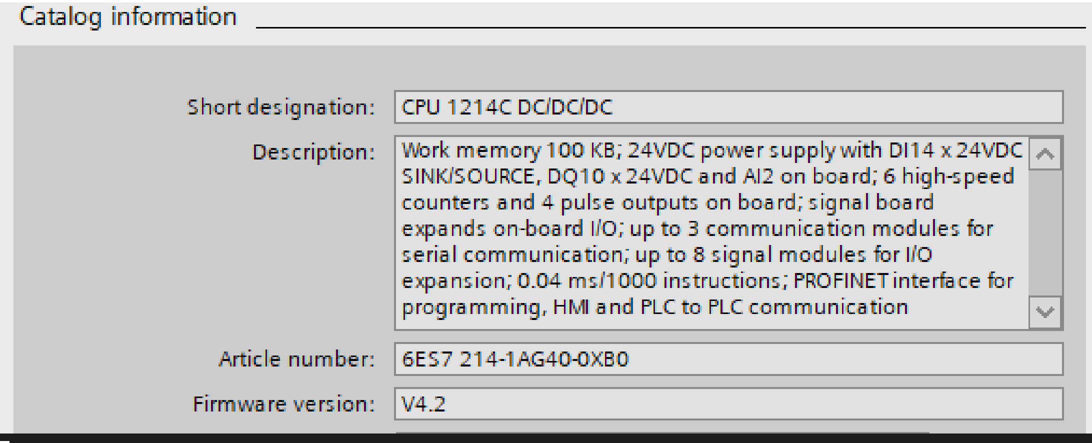


Kuva 1 Projektin kieliasetukset projektipuussa



Kuva 2 Projektiin valitut kieliasetukset

Ohjelmalohkon vaatimusten mukaisesti kehitysalustaksi valittiin Siemens:n S7-1200-perheen logiikka, jonka tarkemmat tiedot kuvassa 3. Logiikkaperheen sisällä samanlaiset ohjelmointitoiminnot ovat käytettävissä, jolloin voidaan valita lähes mikä tahansa CPU:n malli perheen sisällä. Sen verran CPU:n valintaan tässä kohtaa tulee kiinnittää huomiota, että CPU mallin simulointi on tuettuna, jotta ohjelmakoodia päästään helposti testaamaan. S7-1200-perheen logiikassa pitää olla version 4.0 firmware tai uudempi, jotta simulointi on mahdollista S7-PLCSIM:llä. Päädyin valitsemaan koulussa yleisesti käytetyn CPU:n, koska tiedän sen simuloinnin olevan tuettuna simulointiympäristössä.



Kuva 3 CPU:n tarkemmat tiedot

Kun kehitysalusta on valittu, voidaan luoda uusi Function Block (FB)-tyyppinen ohjelmalohko sekä kaksi kuvauksen mukaista tietotyyppiä. Tietotyypit on hyvä luoda valmiiksi ennen kuin aloittaa lohkon liitynnän kirjoittamisen. Tietotyyppien ja lohkon liitynnän kirjoittamisessa on hyvä käyttää apuna Exceliä. Kopioimalla ohjelmalohkon kuvauksen Word-dokumentin taulukoista tiedot Exceliin ja edelleen Excelistä TIA Portal:iin säästyy huomattavasti aikaa kirjoittamiselta ja vältytään kirjoitusvirheiltä.

Ohjelmalohkon tuloissa on liityntöjä, joihin kytketään erilaisten suojalaitteiden tilatietoja. Tulojen nimien mukaisesti niiden arvolla tosi, ajatellaan suojalaitteen olevan kunnossa. Suojalaitteiden tilatiedot rakennetaan toimimaan näin päin, koska häiriö voidaan todeta helposti signaalin puuttumisena. Ohjelmalohkon käyttö mahdollistaa tarvitsemattomien liityntöjen kytkemättä jättämisen. Ohjelmalohkoon voidaan kytkeä vain kulloinkin tarvittavat liitynnät, jolloin osa liitynnöistä saattaa jäädä kytkemättä.

Ohjelmalohko toimii normaalisti, mutta esimerkiksi toteutettavassa lohkossa suojalaitteiden tilatietojen kytkemättä jättäminen hankaloittaa lohkon testaamista ja käyttöä. Ongelma on ratkaistavissa asettamalla tämänlaisten tulojen oletusarvo todeksi. Ohjelmalohkojen yhteydessä on mahdollista syöttää haluttu alustusarvo muuttujille. Muuttujat alustetaan oletusarvoilla logiikkaohjelman suorituksen alkaessa. Jos tuloon ei kytketä mitään, ohjelmalohko käyttää asetettua oletusarvoa, muutoin lohko käyttää tuloon kytketyn muuttujan arvoa. Ominaisuus säästää koodinkirjoittamiseen kuluvaa aikaa, kun kytkemättä jätettyihin tuloihin ei tarvitse turhaan kytkeä aina arvon tosi sisältävää muuttujaa. Kuvassa 4 on muuttujia, joille asetin oletusarvoksi tosi.

	Name	Data type	Default value
5	i_bSafetyCircuitOk	Bool	true
6	i_bShortCircuitProtOk	Bool	true
7	i_bOverloadProtOk	Bool	true
8	i_bThermistorProtOk	Bool	true
9	i_bSafetySwitchClosed	Bool	true

Kuva 4 Suojalaitteiden tulomuuttujia

Liitynnän valmistumisen jälkeen kirjoitin ohjelmalohkon koodia palanen kerrallaan. Kun ohjelmalohkoa rakentaa ominaisuus kerrallaan, pystyy ominaisuuden toiminnan myös yleensä testaamaan simuloinnilla välittömästi. Ohjelmakoodin virheet ja puutteet löytyvät heti, jolloin ne tulee poistettua mahdollisimman aikaisessa vaiheessa. Lisäksi ohjelmalohkossa on vähemmän testattavaa, kun se on valmis lopullista testausta varten.

Koska ohjelmalohkon häiriöt on paketoitu yhteen UDT:hen, on niiden monipuolinen käsittely helpompaa. Häiriöjoukkoa on mahdollista käsitellä esimerkiksi tavumitassa, jolloin lohkossa tiedetään olevan häiriö päällä, kun jokin häiriötavuista on arvoltaan suurempi kuin nolla. Useissa tapauksissa tämänlainen käsittely vähentää kirjoitettavaa koodia ja parantaa ohjelmalohkon laajennettavuutta tulevaisuutta varten.

Häiriöitä käsitellään järjestelmäfunktiolla, jotta UDT:n data saadaan siirrettyä tavutaulukkoon. Datan siirrossa tavutaulukkoon voi tapahtua virhe, jolloin tavutaulukkoon siirrettyjä arvoja ei saa käyttää ohjelmakoodissa. Tavutaulukon tiedon käyttäminen estetään, kun järjestelmäfunktio antaa virhekoodin. Esimerkiksi satunnaisella logiikan

ohjelmakierron syklillä tapahtuu datan siirrossa virhe, jolloin virheellisellä datalla ei tehdä minkäänlaisia jatko-operaatiota, mutta seuraavalla syklillä datan siirto toivottavasti onnistuu ja tarvittavat operaatiot voidaan suorittaa. Jos siirrossa tapahtuvia virheitä ei testattaisi ohjelmakoodissa, voisivat ne aiheuttaa ylimääräisiä häiriöilmoituksia ja samalla laitteen pysähtymisiä.

Testasin lopuksi ohjelmalohkon kokonaisuudessaan, jotta se toteuttaa kaikki vaatimukset ja toiminnankuvauksessa määritellyt toiminnat. Toiminnan testauksen lisäksi tein lohkolle testejä, jotka olin kirjannut ylös lohkon vaatimusten perusteella sekä kuvauksen suunnittelun yhteydessä. Todetessani lohkon testatuksi ja oikein toimivaksi, tarkistin vielä ohjelmalohkon sisällön, että se noudattaa määrittelyohjetta. Viimeiseksi tarkistin, että ohjauslohkon kuvaus vastaa toteutettua ohjelmalohkoa, jotta dokumentaatio ja toteutus varmasti vastaavat toisiaan.

Suoran moottoriohjauksen ohjelmalohko on nyt valmis kirjastoon siirtämistä varten. Lohkoa ei julkaista kirjastoon kuitenkaan valmiina versiona, koska sitä ei ole vielä testattu tai käyttöönotettu minkään projektin yhteydessä.

5 OHJELMALOHKON KIRJASTOINTI

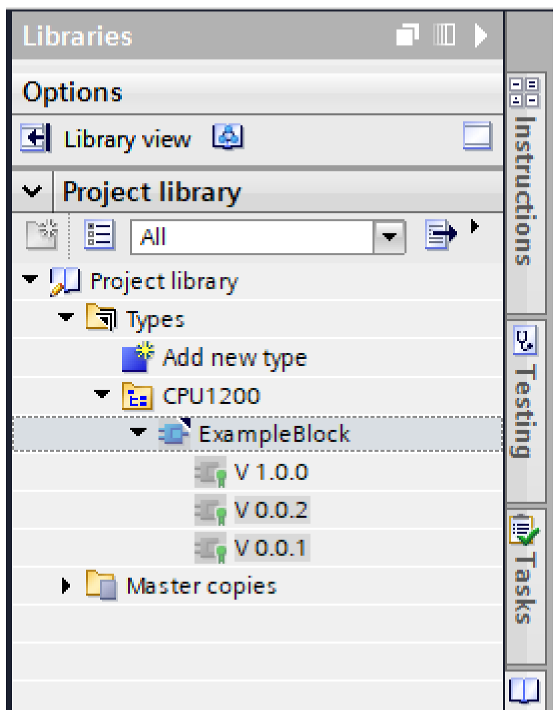
Tarkoituksena oli selvittää, millaisia kirjastointiominaisuuksia Siemens:n TIA Portal-kehitysympäristö sisältää. Se toimii sovelluskehitysympäristönä Siemens:n laitteille. Ympäristöön on yhdistetty logiikka-, liikeohjaus- ja käyttöliittymäsovellusten kehitys kaikki samaan ohjelmistoon.

Ympäristö tarjoaa myös työkalut kirjaston luontia ja sen hallintaa varten, mikä on kehitysympäristön yhtenä vaatimuksena standardissa. Kirjastoon voidaan tallettaa monenlaisia erilaisia komponentteja, kuten järjestelmäkonfiguraatioita, monitorointitaulukoita, PLC datatyyppejä, ohjelmalohkoja ja erilaisia visualisointikomponentteja.

Siemens on jakanut kirjastoitavat kohteet kahteen alaluokkaan: tyypit (Types) ja pääkopiot (Master Copies). Tyyppeihin voidaan tallettaa esimerkiksi ohjelmalohkoja ja visualisointikomponentteja. Pääkopioihin voidaan tallettaa esimerkiksi useiden eri tyyppien laajempia kokonaisuuksia, valmiita järjestelmäkonfiguraatioita ja vaikka valmiita hälytysluokkia.

Kehitysympäristössä on kahdenlaisia kirjastoja: projektikirjasto ja globaalit kirjastot. Projektikirjasto on nimensä mukaisesti projektikohtainen. Globaaleista kirjastoista löytyy Siemens:n omia kirjastoja, mutta mahdollista on luoda myös yrityksen sisällä käytettäviä kirjastoja.

Projektikirjaston (kuvassa 5) avulla voidaan hallita keskitetysti projektissa yleisesti käytössä olevia tyyppejä. Kaikki tyyppiin tehtävät muutokset tulevat voimaan kaikissa projektipuussa olevissa tyypin ilmentymissä.



Kuva 5 Projektikirjaston sivupalkki

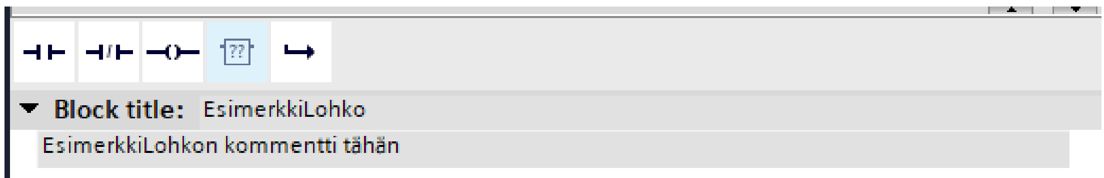
Globaaleihin kirjastoihin voidaan tallettaa kirjastoitavaksi päätettyjä ohjelmalohkoja. Globaalit kirjastot voidaan jakaa verkkolevyn avulla tai käyttämällä Siemensi:n palvelinohjelmistoa. Tyyppejä voidaan kopioida globaalista kirjastosta projektikirjastoon ja käyttää uudelleen projektista toiseen.

5.1 Tyyppiin liitettävän dokumentaation käsittely

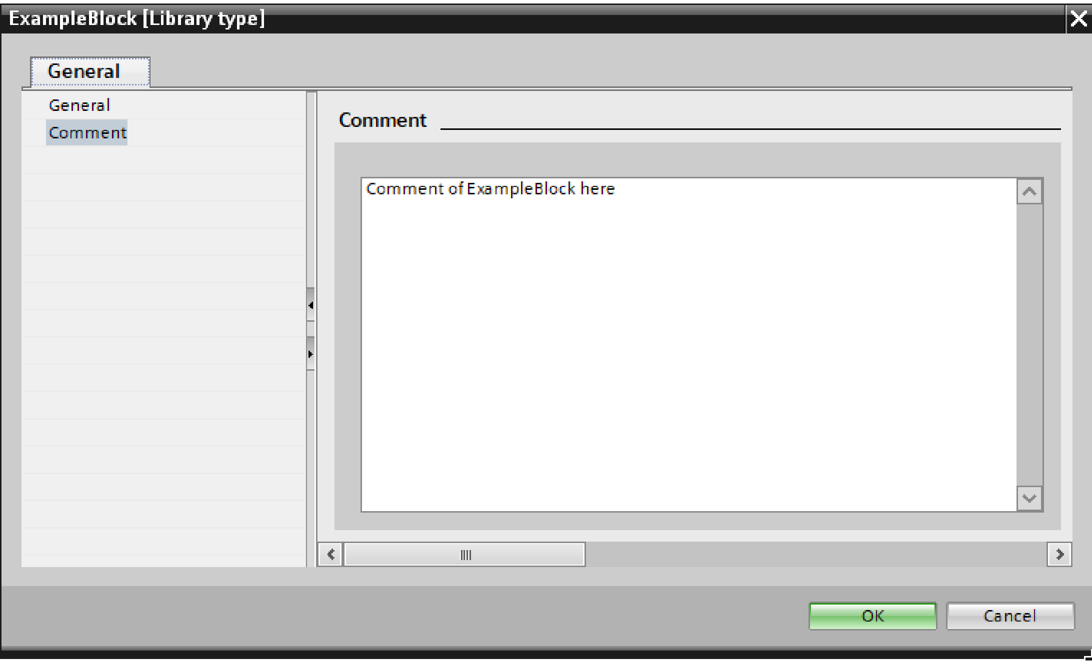
TIA Portal ei valitettavasti tarjoa muita työkaluja tyyppien dokumentaation hallintaan kuin tyypin yhteydessä olevat kommenttikentät, eikä kirjastoon ei ole mahdollistaa liittää erillisiä dokumentteja tyypin kuvauksesta. Kehitysympäristön kommenttikentät ovat täysin tekstipohjaisia, eivätkä tue laajoja tekstin muotoiluja.

Vaikka työkalut eivät ole parhaat mahdolliset, kuitenkin dokumentaatio on parasta liittää tyypin yhteyteen. Jos tyypin kuvaus olisi erillisenä dokumenttina jossakin muussa sijainnissa, dokumentaation päivitys helposti unohtuu tyypin päivityksen yhteydessä.

Tyypin yhteyteen liitettävälle dokumentaatiolle on kaksi mahdollista kommenttikenttää, mutta vain toinen kommentti on luettavissa, kun tyyppi on kirjastoitu. Tyypin dokumentaatio voidaan viedä näihin molempiinkin, jolloin kommentit ovat nähtävissä sekä ohjelmakoodin alussa olevassa kommenttikentässä (kuvassa 6), että kirjastossa olevan tyypin kommenttikentässä (kuvassa 7). Ohjelmakoodin alussa olevaa kommenttia ei pysty lukemaan kirjastossa olevan tyypin ominaisuuksista.



Kuva 6 Ennen ohjelmakoodia oleva kommenttikenttä

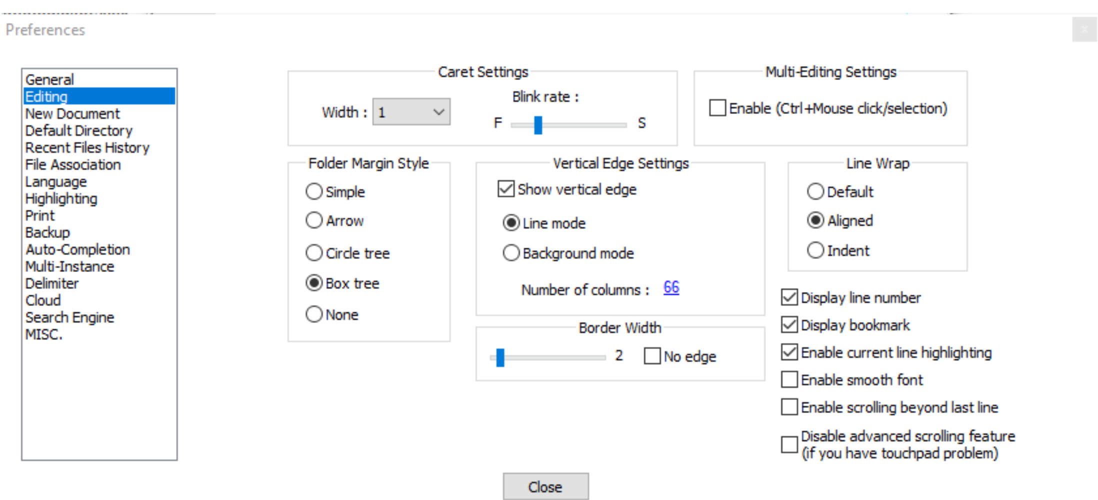


Kuva 7 Kirjastoidun tyypin kommenttikenttä

Tekstin kirjoittaminen ja muokkaaminen on melko hankalaa kirjastoidun tyypin kommenttikentässä, joten suositeltavampaa on käyttää ulkoista tekstieditoria. Käytin suoran moottoriohjauksen ohjelmalohkon dokumentaation muotoilun apuna Notepad++:aa, joka on avoimen lähdekoodin monipuolinen tekstieditori. Notepad++:ssa on monia toimintoja, jotka helpottavat ja nopeuttavat tekstin kirjoittamista ja muotoilua verrattuna esimerkiksi hyvin yksinkertaiseen Windows:n Notepad:iin.

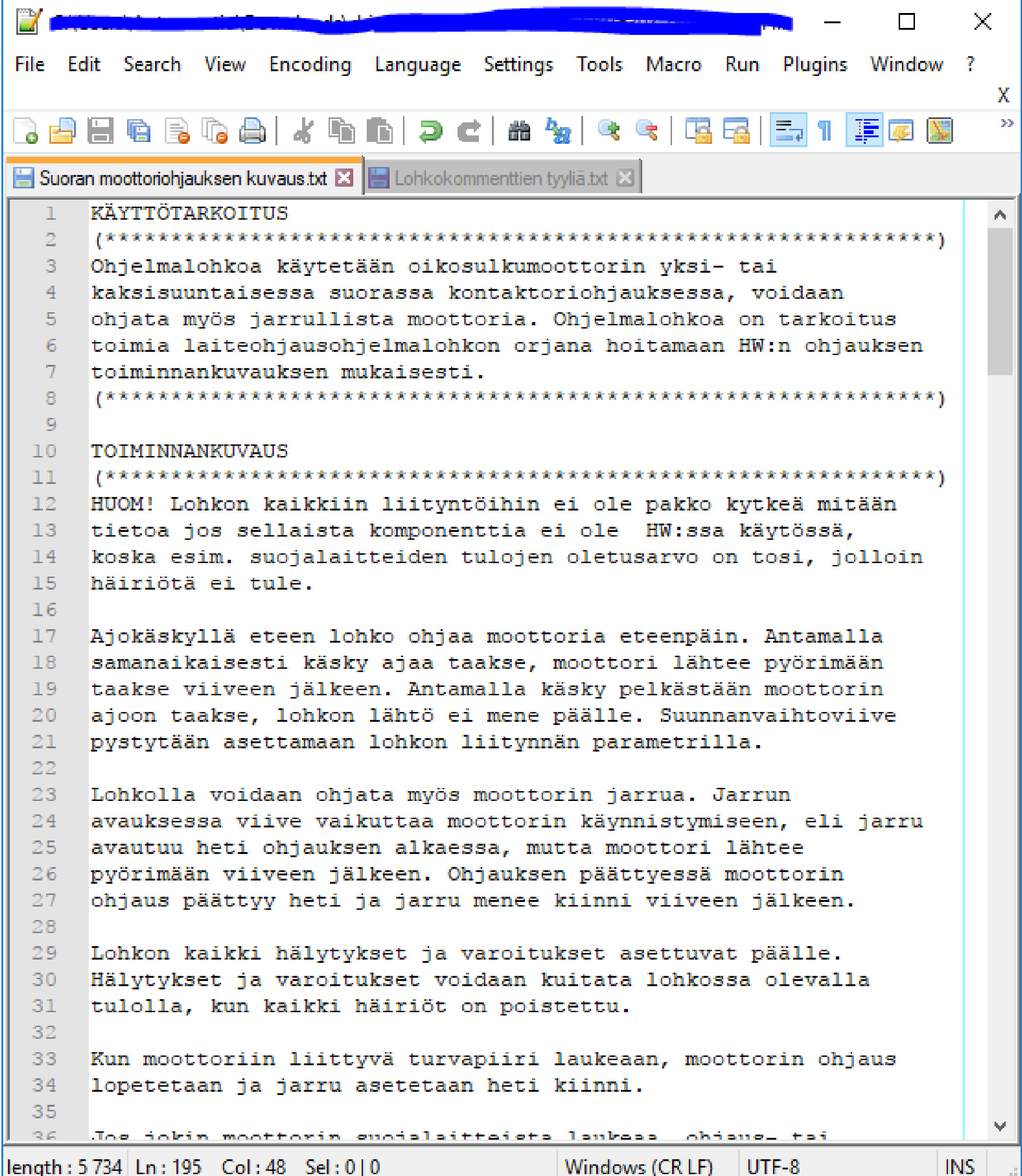
Omaa käyttöä varten kirjoitin ohjelmalohkon dokumentaation Wordia käyttäen, koska siinä voin luoda liityntöjen muuttujille taulukon, joka on huomattavasti havainnollisempi kuin listamainen esitystapa. Lisäksi muuttujien nimet ja kommentit on helpohkosti kopioitavissa taulukosta TIA Portal:iin.

Pohdin ja testailin sopivia muotoiluja tekstidokumentille, jotta se olisi mahdollisimman luettava TIA Portal:ssa. Muotoiluohjeet ovat löydettävissä liitteestä 2. Muotoiluohjeita noudattamalla tekstin tulisi näyttää samalta TIA Portalin kommenttikentässä kuin Notepad++:ssa, kun teksti kopioidaan tekstieditorista kokonaisuudessaan kommenttikenttään. Huomasin 65 merkin rivipituuden olevan sopiva, jolloin teksti rivittyy luontevasti myös TIA Portal:ssa. Ilman tekstin rivittämistä rivipituudet voivat kasvaa todella pitkiksi, jolloin tekstin luettavuus kärsii. Teksti on helppo rivittää Notepad++:n rivitystoiminnolla. Asetetaan Notepad++:n Vertical Edge asetukset kuvan 8 mukaisesti navigoimalla *Settings* -> *Preferences*-valikkoon.



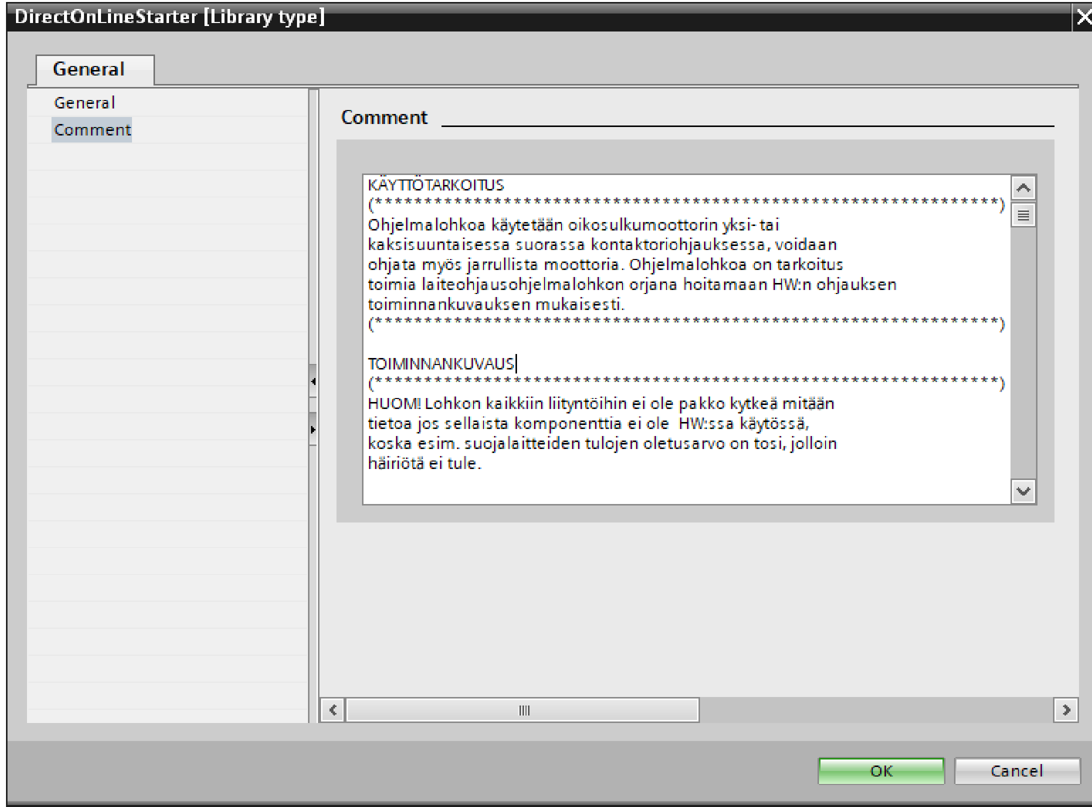
Kuva 8 Notepad++:n Vertical Edge asetukset

Nyt tekstin rivitys onnistuu helposti valitsemalla haluttu teksti ja painamalla näppäinyhdistelmää Ctrl+I. Ohjelmalohkon dokumentaatio rivitettynä kuvassa 9.



Kuva 9 Ohjelmalohkon kuvaus Notepad++:ssa

Nyt ohjelmalohkon kuvaus voidaan kopioida Notepad++:sta lohkon kommenttikenttään. Kuvassa 10 on esimerkki kommenttinäkymästä.



Kuva 10 Kommenttinäkymä

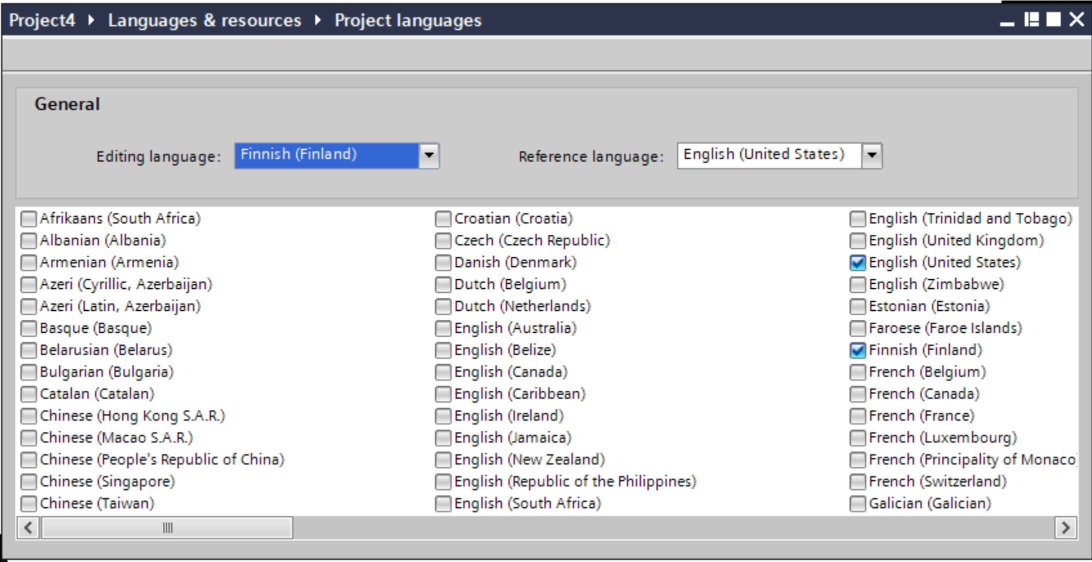
5.2 Ohjeita projektikirjaston käyttöön

Projektikirjasto on jokaisella projektilla omansa. Kun uutta tyyppiä lähdetään kehittämään kirjastoitavaksi, sen kehitys tapahtuu kokonaisuudessaan projektikirjastossa, koska globaalissa kirjastossa ei ole mahdollista muokata tyyppien sisältöä. Projektipuun kirjastoitujen tyyppien ilmentymät ovat aina linkitettyinä projektikirjastoon. Tyyppi ei voi olla linkitettynä globaaliin kirjastoon, vaan kehitysympäristö tuo tyypin automaattisesti projektikirjastoon, jos tyyppi otetaan suoraan käyttöön globaalista kirjastosta.

Kun kirjastossa olevalla tyypillä on riippuvuuksia muista kirjaston tyypeistä, mitään näistä tyypeistä ei voi poistaa ennen kuin kaikki riippuvuudet on poistettu.

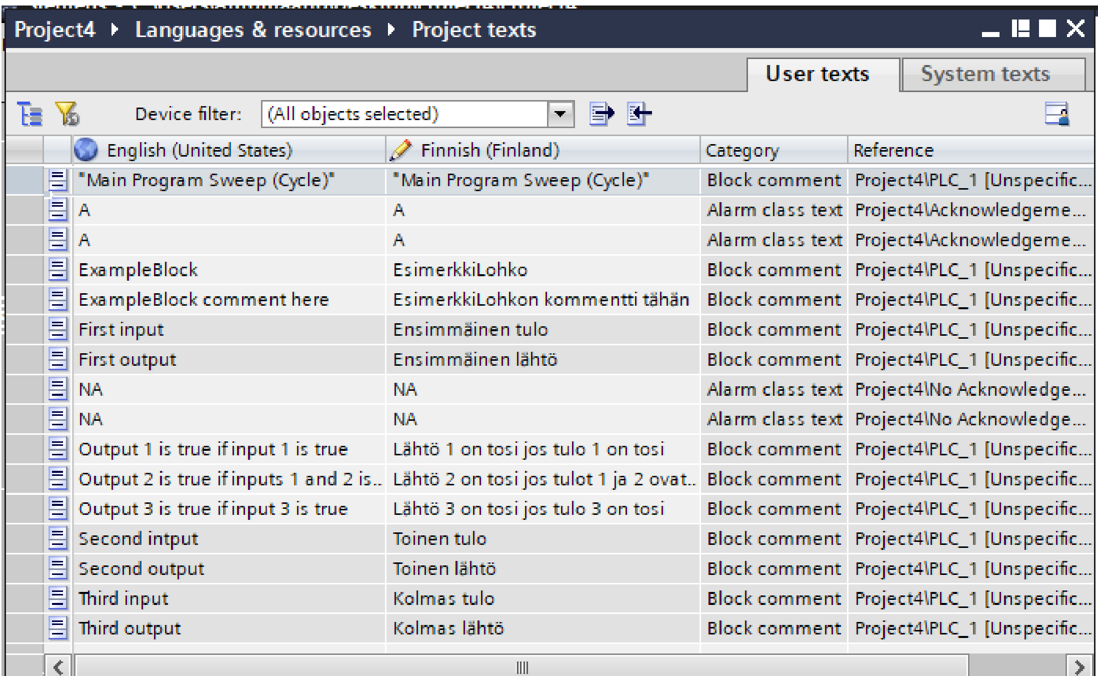
Kun projektikirjastossa olevaa tyyppiä muokataan, pitää tyypille valita sopiva kehitysalusta esimerkiksi projektissa oleva logiikka.

Projektikirjaston kieliasetukset seuraavat projektissa asetettuja kieliä. Projektin muokkauskieleksi tulee asettaa se kieli, jolla ohjelman kommentointi kirjoitetaan kehitysympäristön tarjoamiin kenttiin. Viitekielenä on hyvä käyttää englantia, kuvassa 11 on esimerkki projektikielien asetuksista.



Kuva 11 Esimerkki käytettävistä projektikielistä

Projektin tekstejä voidaan kääntää keskitetysti kohdassa *Project texts*, kuvassa 12 on esimerkki tekstityökalusta.



Kuva 12 Projektin tekstityökalu

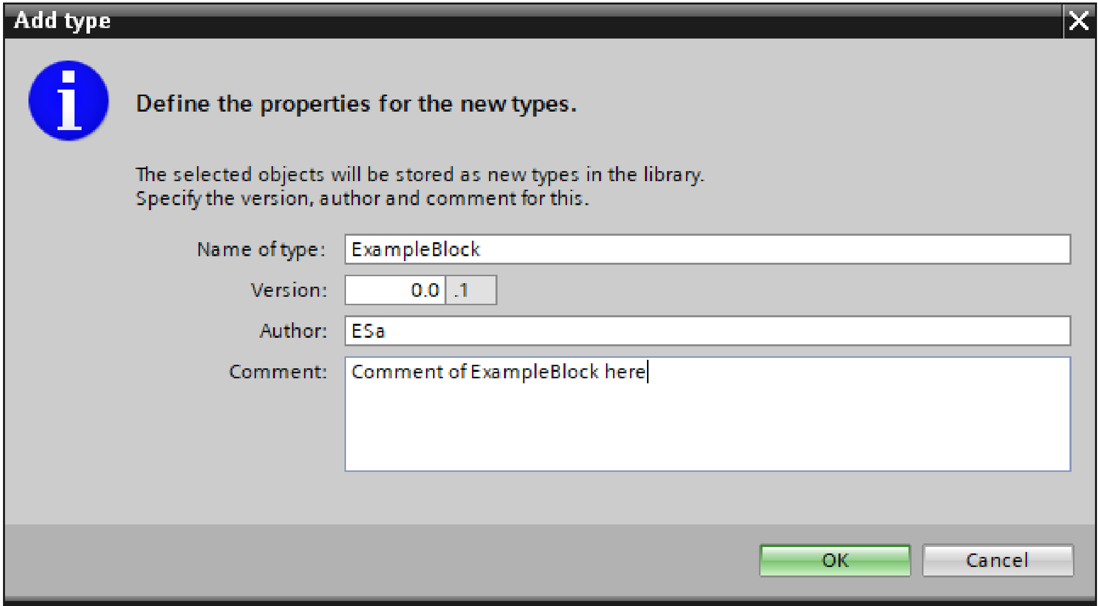
5.2.1 Ohjelmalohkon vienti projektikirjastoon

Ennen kuin lohko voidaan kirjastoida, tulee sillä olla kuvaus. Kuvauksesta tulee selvitä määrittelyohjeessa vaaditut asiat.

Ohjelmalohko on ensimmäiseksi hyvä luoda projektipuuhun. Kun ohjelmalohko on luotu ja sille on dokumentaatio, voidaan se kopioida projektikirjastoon.

Ennen lohkon kirjastoon siirtämistä on hyvä ottaa seuraava asia huomioon. Siemens:n ratkaisussa jokaisella Function Block:lla on nimen lisäksi lohkonumero, joka pitää olla yksilöllinen logiikan sisällä käytettävien ohjelmalohkojen kesken. Kuitenkin kirjastoon voidaan siirtää ohjelmalohkoja samoilla lohkonumeroilla. Jos lohkot ovat kirjastoitu samoilla lohkonumeroilla, pitää projektipuuhun tuotujen lohkojen numerot vaihtaa käsin. Valitettavasti kirjastossa olevista lohkoista ei voi suoraan nähdä millä lohkonumerolla se on tallennettu. Yksi ratkaisu olisi tallettaa jokainen ohjelmalohko eri lohkonumerolla, esimerkiksi kasvattamalla numeroa juoksevasti.

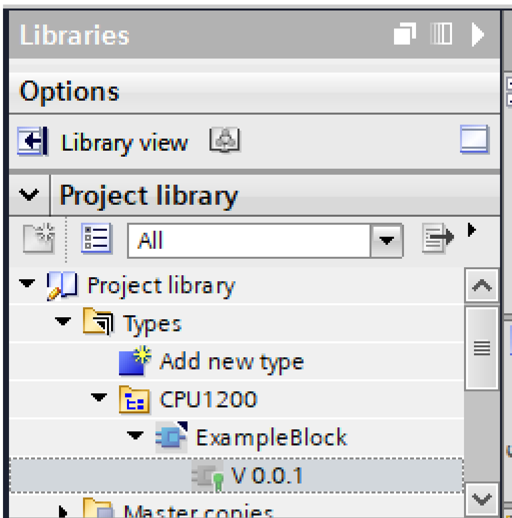
Ohjelmalohkon saa kopioitua projektikirjastoon drag-and-drop-menetelmällä raahaamalla se haluttuun kansioon. Kehitysympäristö kysyy kirjastoitavalle tyypille tarvittavia tietoja, jotka näkyvät kuvassa 13.



Kuva 13 Tietojen syöttöikkuna

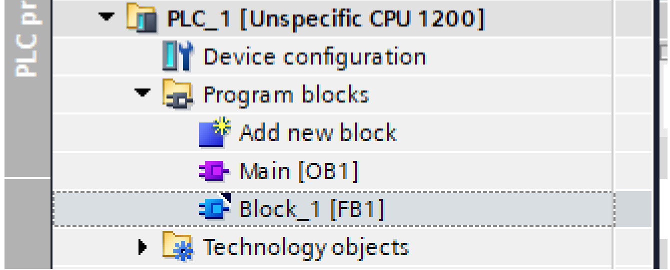
Kirjastotavalle ohjelmalohkolle annetaan kuvauksessa päätetty nimi, versionumero, tekijän nimi ja kommentti. Tässä ikkunassa syötettävä kommentti tallentuu sekä lohkon kirjastokommenttiin, että lohkon ensimmäisen version kommenttiin. Lohkon kirjastokommenttia voidaan muokata myöhemmin, mutta versiokommentti on lukittu versiokohtaisesti.

Kirjastoon on hyvä luoda kansioita tyyppien ryhmittelyä varten. Esimerkissä projektikirjastoon on luotu testauksessa käytettyä CPU-perhettä vastaava kansio. Kuvassa 14 esimerkin ohjelmalohko on viety projektikirjastoon.



Kuva 14 Ohjelmalohko projektikirjastossa

Projektipuuhun luodulle lohkolle ilmestyy yläkulmaan musta nuoli (kuvassa 15), kun lohko on linkitetty kirjastoon.

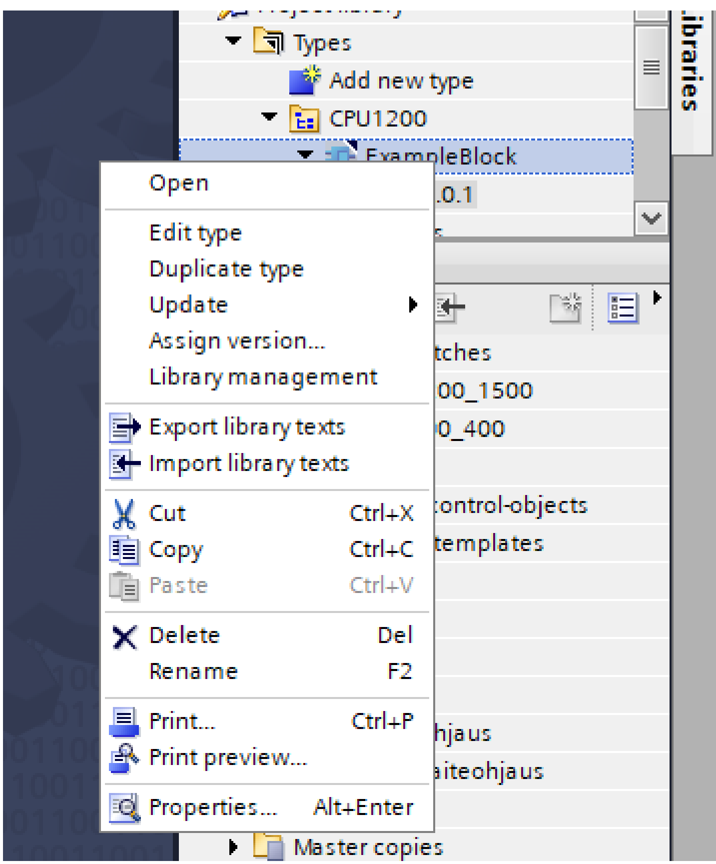


Kuva 15 Projektipuussa oleva kirjastoitu lohko

Kun projektipuun ohjelmalohko on linkitetty projektikirjaston tyyppiin, kaikki kirjastossa lohkoon tehdyt muutokset päivittyvät automaattisesti projektin ohjelmalohkoon.

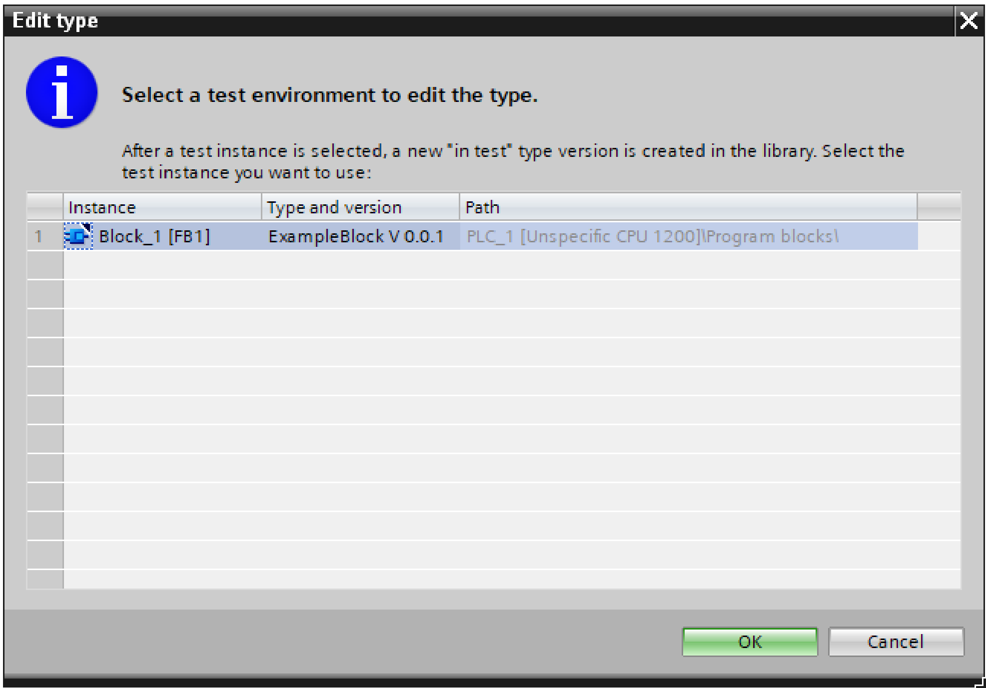
5.2.2 Kirjastoidun ohjelmalohkon muokkaus

Projektikirjastossa olevaa lohkoa pääsee muokkaamaan useampaa eri kautta. Yksi tapa on klikata hiiren oikealla painikkeella lohkoa ja valita *Edit type*, kuten kuvassa 16.



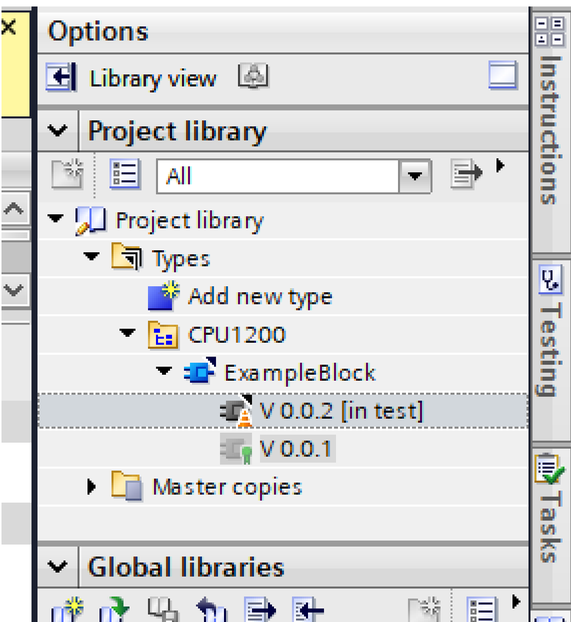
Kuva 16 Tyypin valikko

Kehitysympäristö kysyy käytettävää testiympäristöä (kuvassa 17), valitaan oikea testiympäristö ja klikataan ok.



Kuva 17 Testausympäristön valintaikkuna

Lohko aukeaa muokkaustilassa ja kehitysympäristö luo lohkosta automaattisesti uuden version testaustilassa, josta esimerkki kuvassa 18.

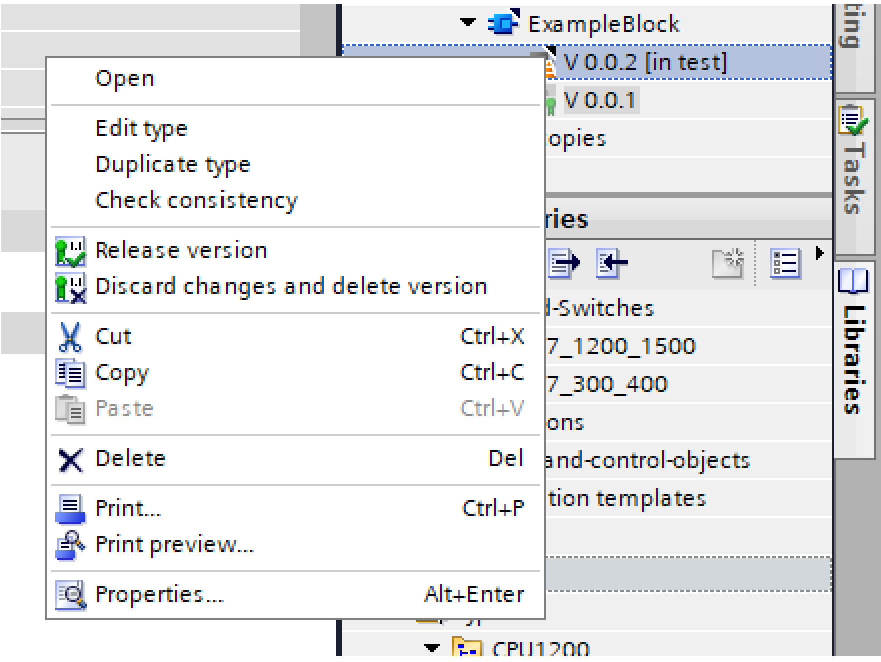


Kuva 18 Projektikirjaston näkymä

Lohkolle voidaan nyt tehdä kaikki tarvittavat muutokset ja lisäykset. Lohkon toimintaa voidaan testata valitussa testiympäristössä, kunhan lohkoa kutsutaan jossakin kohtaa logiikkaohjelmaa.

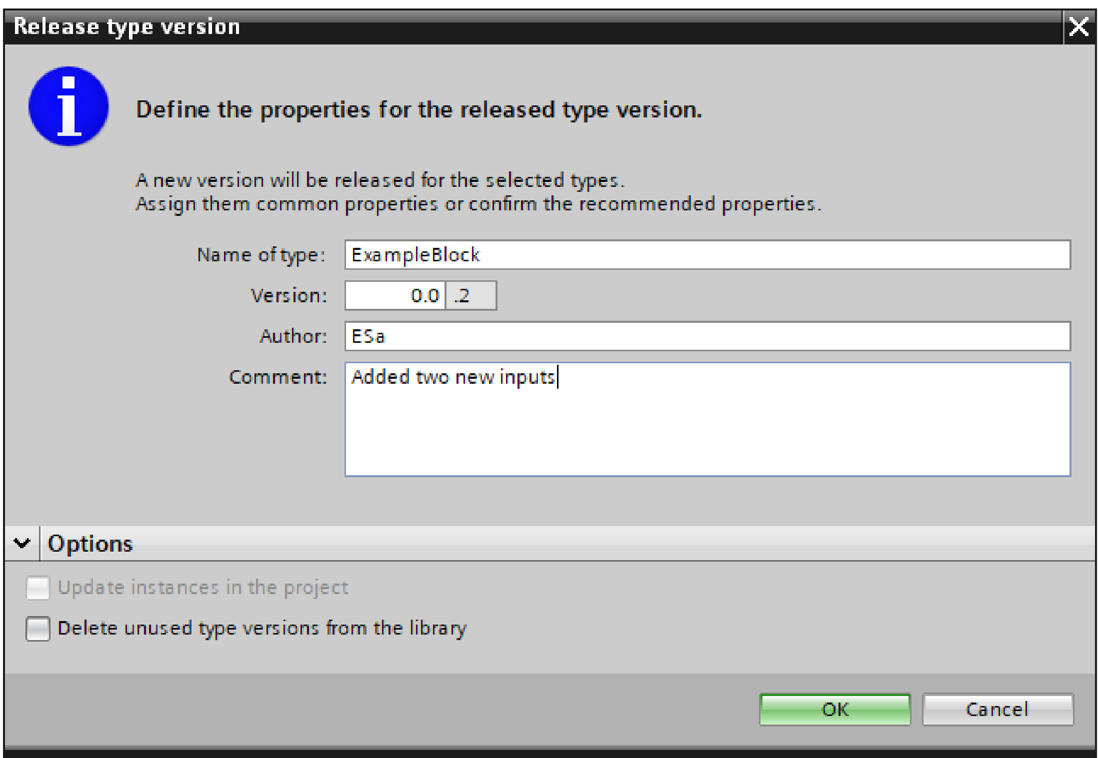
5.2.3 Ohjelmalohkon julkaiseminen

Kun lohko ajatellaan olevan valmis julkaistavaksi, se voidaan julkaista esimerkiksi klikkaamalla hiiren oikealla painikkeella lohkon versiota ja valitsemalla *Release version* kuin kuvassa 19.



Kuva 19 Tyypin version valikko

Kehitysympäristö kysyy tiedot julkaistavaa versiota varten (kuvassa 20). Ikkunassa voidaan muuttaa tyypin nimeä, versiota ja tekijää. Ikkunassa syötettävä kommentti on vain julkaistavaa versiota koskeva kommentti.



Kuva 20 Julkaisun tietojen syöttöikkuna

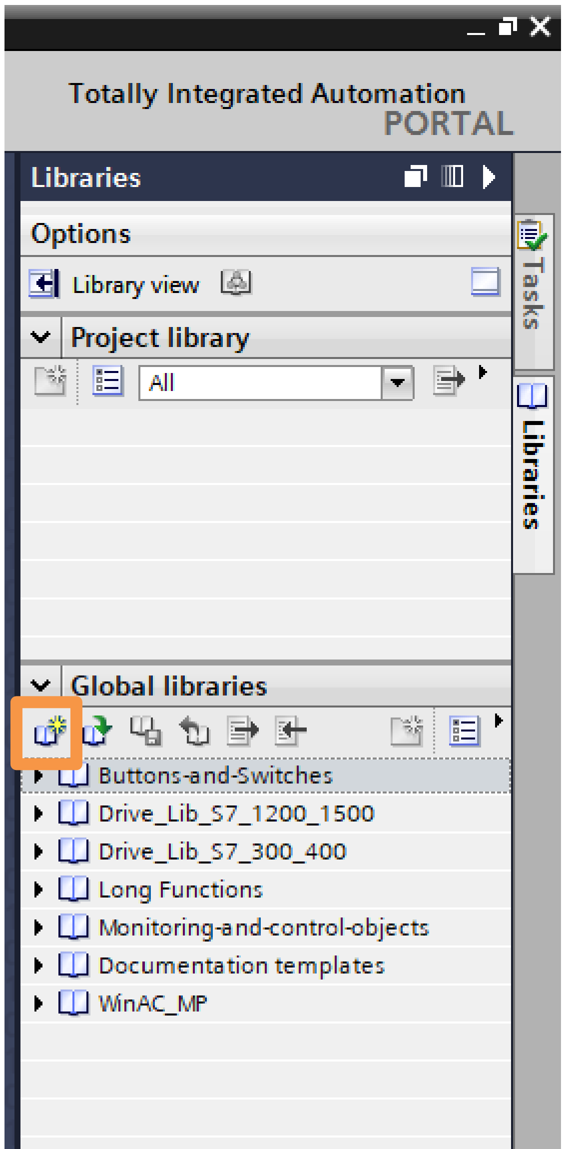
5.3 Ohjeita globaalin kirjaston käyttöön

Globaali kirjasto eroaa projektikirjastosta oleellisesti, koska siellä olevia lohkoja ei pysty suoraan muokkaamaan. Ainoa asia mitä globaalissa olevalle lohkolle voi tehdä, on muokata lohkon kirjastokommenttia. Päästäkseen muokkaamaan lohkon kommenttia, pitää kirjasto olla avattu kirjoitustilassa.

Globaalilla kirjastolla on omat kieliasetuksensa, joka on riippumaton projektin kielestä. Kirjastolle on hyvä asettaa vastaavat kielet käyttöön kuin siellä olevilla lohkoillakin on.

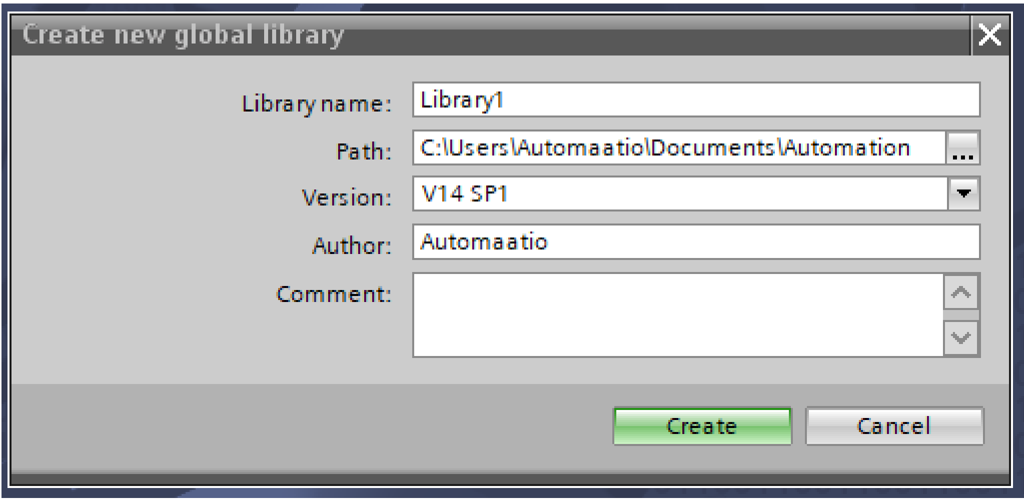
5.3.1 Uuden globaalin kirjaston luominen

Siirrytään ohjelmointiympäristön *Project view*-tilaan ja navigoidaan oikealla sivupalkissa välilehdelle *Libraries*. Uusi globaali kirjasto luodaan painamalla kuvassa 21 korostettua painiketta *Create new global library*.



Kuva 21 Kirjastot sivupalkki

Kehitysympäristö kysyy seuraavaksi kuvan 22 ikkunassa kirjastoon liittyviä asioita.



Kuva 22 Kirjaston tietojen syöttöikkuna

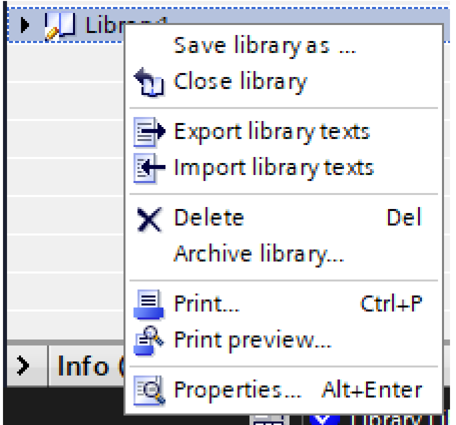
Kirjastolle valitaan sopiva nimi, tallennuskohde, käytettävän kehitysympäristön versio ja kirjaston tekijä. Lisäksi kirjastolle voi antaa jonkin kommentin tai kuvauksen. Jos kirjasto halutaan jakaa monelle käyttäjälle, voidaan tässä kohtaa valita tallennuskohteeksi esimerkiksi verkkolevy. Valitaan *Create*, ja kirjasto ilmestyy muiden globaalien kirjastojen listaan.

Kun globaali kirjasto on luotu, se on oletuksena avoinna muokkaustilassa. Kirjaston kuvakkeen päällä on pieni kynä-kuvake, kun kirjasto on muokkaustilassa.

Kuva 23 Globaalin kirjaston kuvake

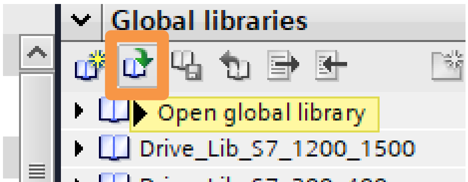
Muokkaustilassa globaaliin kirjastoon voidaan siirtää uusia tyyppejä, muokata tyyppien kommentteja ja kirjaston asetuksia. Kirjasto voi olla avoinna muokkaustilassa vain yhdellä käyttäjällä kerrallaan ja tällöin toiset eivät pysty avaamaan kirjastoa.

Jos globaali kirjasto on esimerkiksi jaettu verkkolevyllä pitää kirjasto avata *Read-only*-tilassa, jotta useat käyttäjät voivat käyttää kirjastoa. Suljetaan muokkaustilassa oleva kirjasto klikkaamalla kirjastoa hiiren oikealla painikkeella ja valitsemalla *Close library*, kuten kuvassa 24.



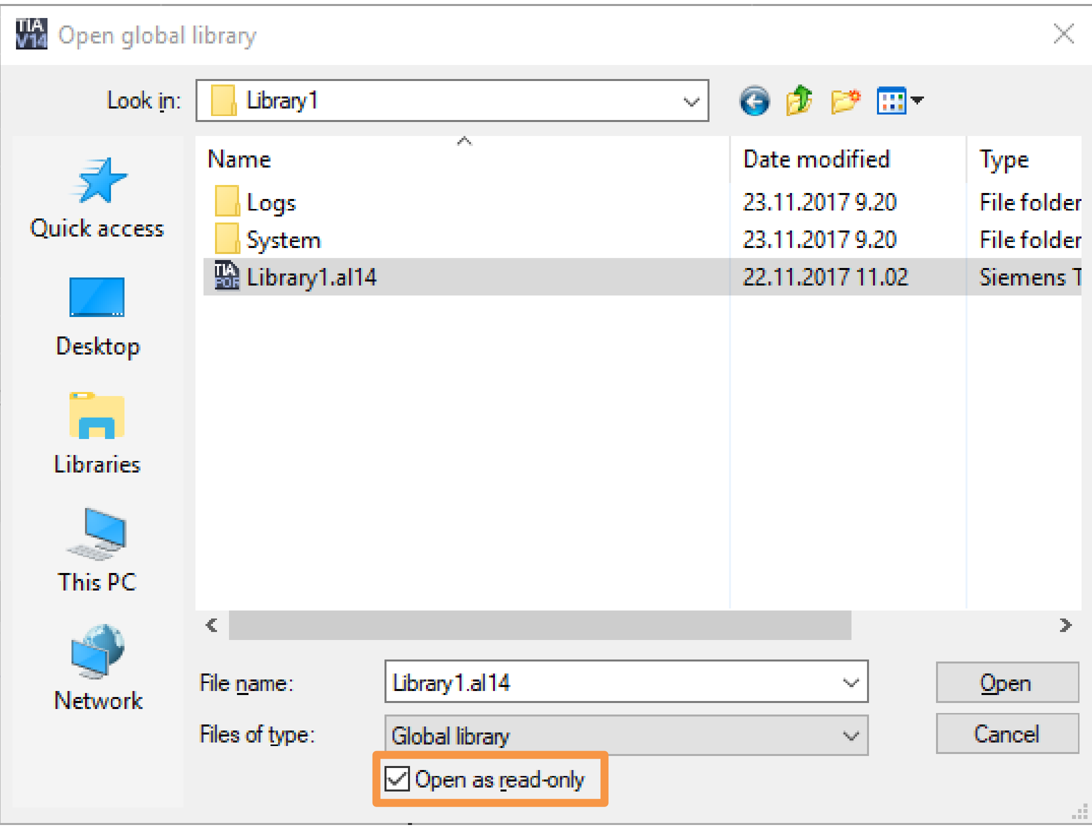
Kuva 24 Kirjaston valikko

Avataan kirjasto uudelleen, esimerkiksi kuvan 25 osoittamasta painikkeesta.



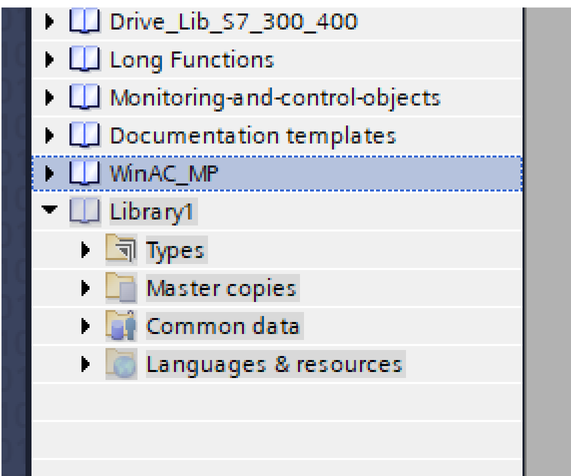
Kuva 25 Kirjaston avaaminen

Haetaan haluttu kirjasto ja avataan se *Read-only*-tilassa.



Kuva 26 Kirjaston avausikkuna

Read-only-tilassa avatun kirjaston kohteet näkyvät sivupalkissa harmaalla taustalla, kuten kuvassa 27.



Kuva 27 Globaali kirjasto avattuna Read-only-tilassa

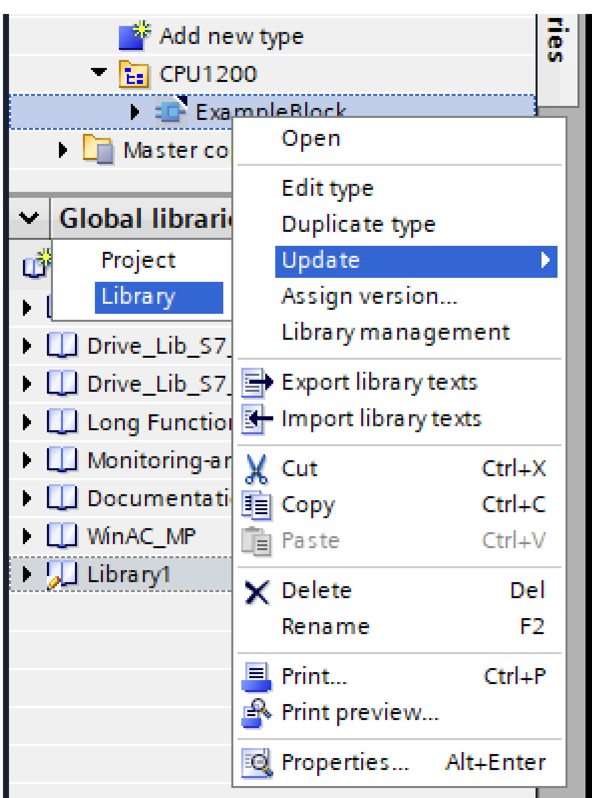
Kirjaston sisältä löytyvät tyypit (Types), pääkopiot (Master copies), yhteinen data (Common Data) ja kielet & resurssit (Languages & Resources).

Yhteinen data sisältää kirjaston lokin, jonne talletetaan tiedot kirjastoon tehdyistä muutoksista.

Kielet & resurssit-osiossa voidaan hallita kirjastossa käytettäviä kieliä, kieliasetukset on hyvä asettaa vastaamaan tyypeissä käytettyjä kieliä. Vaikka globaaliin kirjastoon ei olisi asetettu kaikkia tyypin sisältämiä kieliä käyttöön, käännökset tallentuvat tyypin yhteyteen, eivätkä häviä globaaliin kirjastoon kopioinnin yhteydessä.

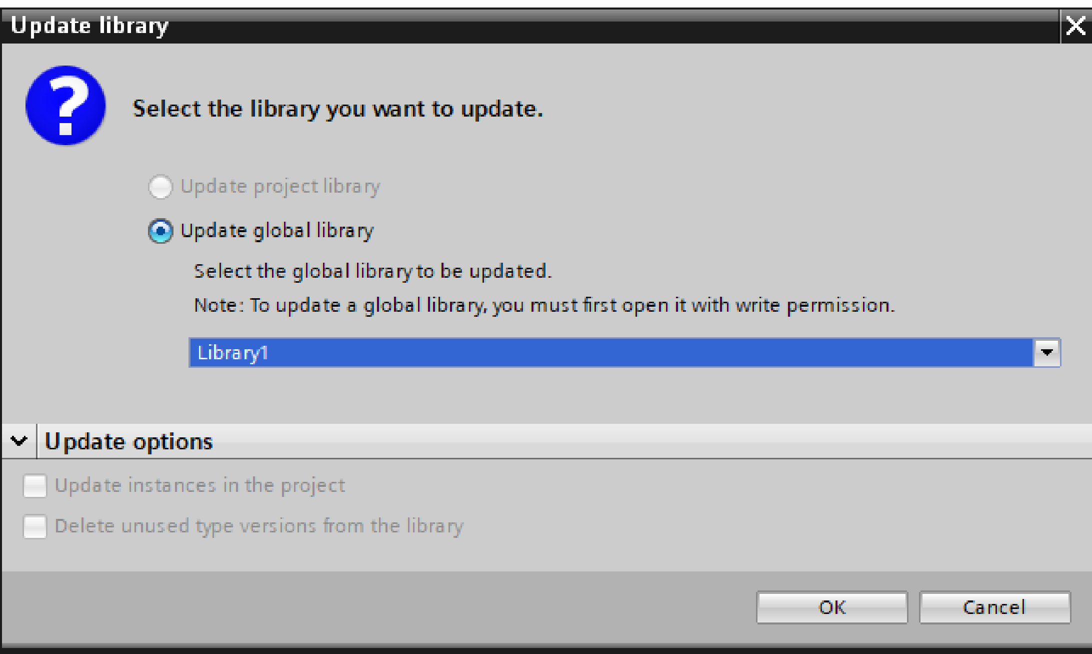
5.3.2 Ohjelmalohkon vienti tai päivittäminen globaaliin kirjastoon

Globaali kirjasto pitää olla avattuna kirjoitustilassa, jonka jälkeen ohjelmalohko voidaan kopioida tai päivittää globaaliin kirjastoon muutamalla eri tavalla. Yksi tapa on klikata hiiren oikealla painikkeella projektikirjastossa olevaa tyyppiä ja valitaan *Update -> Library*, kuten kuvassa 28.



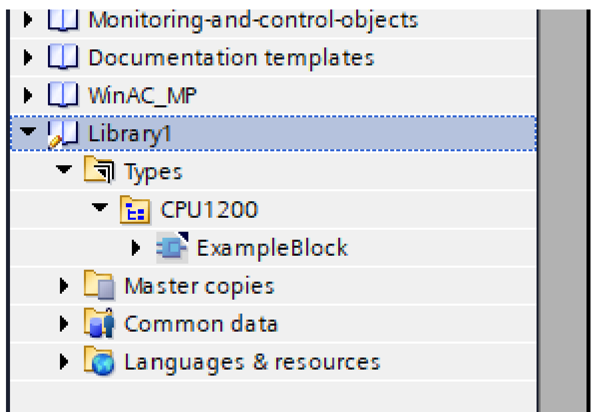
Kuva 28 Tyypin valikko

Kehitysympäristöön aukeaa ikkuna, jossa kysytään mihin kirjastoon lohko halutaan päivittää. Valitaan haluttu globaali kirjasto, kuten kuvassa 29 ja painetaan ok.



Kuva 29 Kirjaston päivitysikkuna

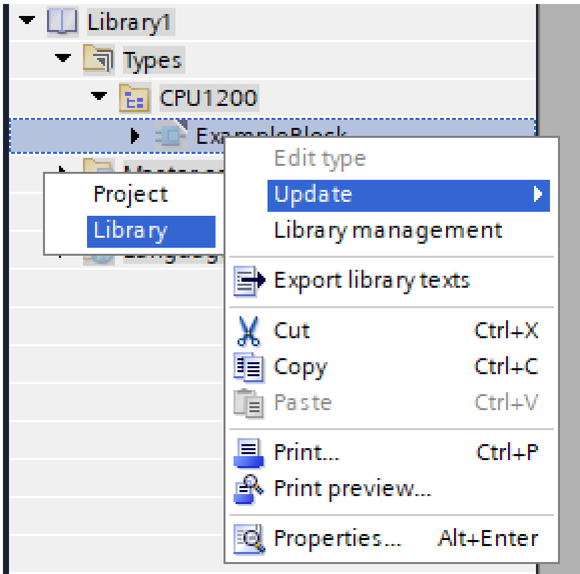
Toiminto kopioi saman kansiorakenteen kuin projektikirjastossa automaattisesti ja kopioi tyypin globaaliin kirjastoon, kuten kuvassa 30.



Kuva 30 Tyyppi globaalissa kirjastossa

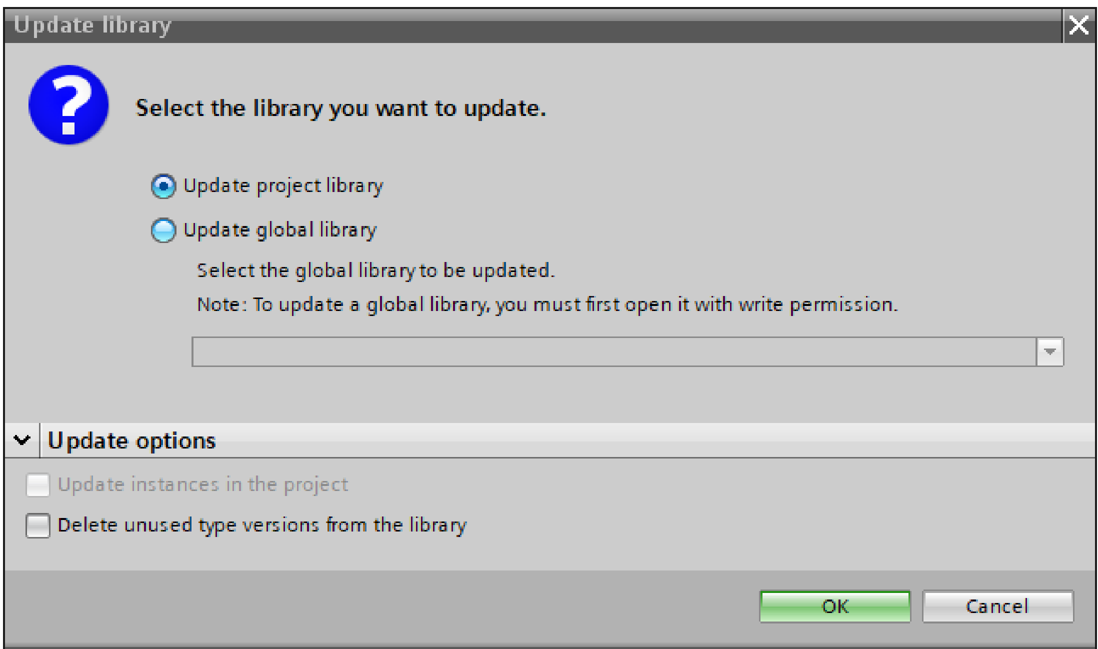
5.3.3 Ohjelmalohkon tuonti globaalista kirjastosta

Ohjelmalohko voidaan tuoda projektikirjastoon samalla tavalla kuin se vietiin globaaliin kirjastoon. Klikataan hiiren oikealla painikkeella globaalissa kirjastossa olevaa ohjelmalohkoa ja valitaan *Update* -> *Library*, kuten kuvassa 31.



Kuva 31 Tyypin valikko globaalissa kirjastossa

Kehitysympäristö kysyy, mihin kirjastoon ohjelmalohko halutaan päivittää, valitaan kohteeksi projektikirjasto, kuten kuvassa 32 ja painetaan ok.



Kuva 32 Kirjaston päivitysikkuna

Toiminto kopioi saman kansiorakenteen kuin globaalissa kirjastossa automaattisesti ja kopioi tyypin projektikirjastoon.

5.4 Versionumeroinnista

TIA Portal-tarjoaa kirjastointityökalujen yhteydessä tyyppien versioinnin. Tyyppien versionumerot ovat kolmiosaisia X.Y.Z, joista kaksi ensimmäistä osaa voidaan valita vapaasti väillä 0-999. X-osa kuvaa tyypin pääversiota, Y-osa kuvaa tyypin alaversiota ja Z-kuvaa tyypin build-versiota. Ohjelmointiympäristö kasvattaa automaattisesti Z-osaa, kun tyyppi julkaistaan.

Versioinnin lisäksi tyypit voivat olla kirjastossa kolmessa eri tilassa:

1. In work = työn alla (Faceplatet ja HMI:n UDT:t)
2. In test = testauksessa (kaikki muut tyypit, paitsi kohdan yksi tyypit)
3. Released = julkaistu

Tyyppi kehitetään mahdollisimman valmiiksi ja testataan kokonaisuudessaan ennen kuin se julkaistaan valmiina versiona. Kehitysvaiheessa versionumeroinnissa voidaan käyttää apuna kahta viimeisintä versionumeron osaa eli Y.Z, ja pitämällä X-osa arvossa nolla. Kun tyyppi julkaistaan valmiina ja testattuna, asetetaan pääversion (X) arvoksi 1.

Kun tyyppiin tehdään suuria muutoksia, jotka vaikuttavat tyypin toimintaan tai muuttavat lohkon liityntää niin merkittävästi, ettei uudella tyypin versiolla voida suoraan korvata vanhaa versiota, tyypin pääversiota X kasvatetaan yhdellä.

Kun tyyppiin tehdään suurehkoja muutoksia tai ominaisuuden lisäys, ja uudella tyypin versiolla voidaan suoraan korvata sen vanha versio, tyypin alaversiota Y kasvatetaan yhdellä.

Kun tyyppiin tehdään korjauksia ja pieniä muutoksia, ja uudella tyypin versiolla voidaan suoraan korvata sen vanha versio, tyyppi julkaistaan uutena build-versiona ja TIA:n ohjelmointiympäristö kasvattaa automaattisesti build-numeroa yhdellä.

Kun jokin tyyppi riippuu toisista tyypeistä, annetaan jokaiselle tyypille samat X.Y-osien versionumero. Tällöin versionumerosta on suoraan nähtävissä, että tyypit ovat keskenään yhteensopivia. Kun versionumeroa pitää kasvattaa lohkojen yhteensopivuuden osoittamiseksi, kasvatetaan ensisijaisesti alaversion arvoa tarvittavalla määrällä.

6 JATKOKEHITYKSESTÄ

Työssä keskityttiin antamaan ohjeita, miten ohjelmalohkon määrittely ja kehitys etenevät, mitä sen dokumentaatiolta vaaditaan ja miten kirjoitettavaa koodia voidaan yhtenäistää. Työn ulkopuolelle jää pohdittavaksi, miten esimerkiksi todetaan, että ohjelmalohkon dokumentaatio on tarpeeksi kattava. Miten dokumentaation laatua valvotaan? Millä keinoin validoidaan se, että ohjelmalohkon dokumentaatiossa on kaikki, mitä määrittelyssä on kerrottu tarvittavan. Suoran moottoriohjauksen ohjelmalohkon dokumentaatio pyrkii antamaan esimerkkiä dokumentaation tasosta. Ohjelmalohkon kuvauksen valmistumisen jälkeen, on esimerkiksi hyvä antaa kuvaus luettavaksi toiselle suunnittelijalle, ja antaa hänen arvioida sen sisältöä. Usein asian ulkopuolella työskentelevä huomaa täysin uusia asioita, joille itse on tullut jo sokeaksi.

Samantapaiset kysymykset kuin dokumentaatiossa, koskettavat myös ohjelmakoodin testaamista. Nämä jätetään myös työn ulkopuolelle harkittavaksi, koska testauskäytäntöjen rakentaminen on itsessään laaja asia. Millä keinoin todetaan se, että ohjelmalohkon logiikkaohjelmakoodi on rakennettu määrittelyohjeen mukaisesti ja ohjelmakoodi toimii oikein. Ilman testauskäytäntöjä logiikkakoodin testaaminen saattaa vaihdella suurestikin suunnistelijasta riippuen. Testauksen laatuun vaikuttavat suunnittelijan ammattitaito ohjelmakehityksestä sekä näkemys testauksesta. Ohjelmalohkon dynaamista testausta varten pystytään tietysti käyttämään hyväksi ohjelmalohkolle asetettuja vaatimuksia ja sen kuvausta, joihin lohkon ajossa tapahtuvaa toimintaa voidaan verrata. Tulevaisuudessa voidaan myös harkita, voitaisiinko ohjelmakoodin testaamista automatisoida jollakin tavalla. Staattista testausta varten voidaan käyttää apuna määrittelyohjetta, mutta paljon jää myös suunnittelijan tietämyksen varaan logiikkaohjelmointikielen syntaksista.

Dokumentaation tarkastamisesta ja laadusta sekä ohjelmakoodin testaamisesta riittää aihetta jo uuden opinnäytetyön pohjaksi.

Kun määrittelyohjetta tullaan käyttämään käytännössä, ajan saatossa esille tulee varmasti kehitysehdotuksia. Huomataan uusia asioita, joita voisi kirjata määrittelyyn tai

olemassa olevia ohjeita voidaan joutua muuttamaan. Ohjeita pitääkin kehittää jatkuvasti ja joissakin tilanteissa osa ohjeistuksesta saattaa vanhentua kokonaan tai muuttua merkitsemättömäksi.

Kun ratkaisujen vakiointia halutaan viedä pidemmälle, voi olla hyvä mainita ohjelmalohkon yhteydessä, minkälaisten mallipiirikaavioiden yhteydessä sitä on käytetty tai on hyvä käyttää. Tällöin olisi mahdollista rakentaa vakioituja ratkaisuja, joissa käytetyt piirikaaviot ja logiikkaohjelman lohkot ovat samanlaisia projektista toiseen. Yhdistelemällä näitä moduuleita voidaan rakentaa laajempia kokonaisuuksia.

UDT:n käyttäminen ohjelmalohkossa antaa mahdollisuudeksi sen, että sen kautta tehtävää liityntää voidaan laajentaa helpostikin. Lohkon ohjaamista varten kaikki tärkeät asiat ovat samassa paketissa. Lohkojen tilojen lisäksi UDT:hen voisi lisätä lohkon parametrit ja SW ohjausten liitynnän. Näitä toimintoja varten UDT on siirrettävä tulolähtö tyyppiseksi liitynnäksi, jotta UDT:n sisäisiä muuttujia pystytään myös kirjoittamaan.

Ohjelmalohkon dokumentaatiossa voisi olla hyvä mainita käytetyistä testausalustoista. Työn toteutuksessa päätettiin käyttää kansiorakenteeseen perustuvaa jaottelua. Eli S7-1200 ja S7-1500 logiikkaperheen alustoilla testatut lohkot ovat kirjastossa vastaavien kansioiden alla. Toisaalta S7-1200 logiikkaperheen tukemat toiminnot toimivat myös S7-1500 perheen logiikoissa, mutta toiseen suuntaan yhteensopivuudesta ei voi olla varma, koska S7-1500 logiikkaperhe sisältää laajemmat ominaisuudet.

Osa opinnäytteen ulkopuolelle jätetyistä asioista ovat sellaisia, jotka jäivät työn tavoitteiden rajausten ulkopuolelle, ja siksi jätettiin työn ulkopuolelle harkittavaksi. Lisäksi toteutuksen aikana ja sen jälkeen heräsi ajatuksia, joita olisi voinut hyödyntää työn toteutuksessa aivan yhtä hyvin kuin valittuja ratkaisuja. Jatkokehityskohteisiin on hyvä kiinnittää huomiota, jolloin ne tiedostetaan ja niille voidaan lähteä kehittämään uusia toimintatapoja tulevaisuudessa. Ongelmat pitää suunnitella ratkaistavaksi ensin yhdellä tavalla, ennen kuin ratkaisutapaa voidaan lähteä kehittämään paremmaksi.

LÄHTEET

Asmala, H.;Koskinen, K.;Koskela, M.;Mätäsniemi, T.;Soini, A.;Strömman, M.;. . . Valkonen, J. (2005). *Automaatiosovellusten ohjelmistokehitys.* Suomen Automaatioseura ry.

Etusivu: PIIR-GROUP Oy. (20. 12 2017). Noudettu osoitteesta PIIR-GROUP Oy:n verkkosivut: http://www.piir-group.com/

IEC. (30. 11 2017). *IEC 60050 - International Electrotechnical Vocabulary - Welcome*. Noudettu osoitteesta Electropedia: www.electropedia.org

PLCopen. (2016). *Coding Guidelines.* PLCopen.

PLCopen. (29. 11 2017). *PLCopen for efficiency in automation*. Noudettu osoitteesta PLCopen: http://www.plcopen.org/index.html

SFS-käsikirja 16. (2003). *SFS-käsikirja 16, Moottorikeskukset ja ohjelmoitavat ohjaukset. Vakiosovelluksia enintään 1000 V moottorikäytöille.* SFS ry.

SFS-Käsikirja 631-2. (2014). *SFS-Käsikirja 631-2, Ohjelmointi ja dokumentointi.* SFS ry.

Siemens AG. (16. 10 2017). *Simatic Step 7 Professional V14 SP1 System manual.* Noudettu osoitteesta Industry Online Support: https://support.industry.siemens.com/cs/mdm/109747136?c=96094627083&lc=en-WW

OHJELMALOHKON KUVAUS

Ohjelmalohkon käyttötarkoitus

Ohjelmalohkoa käytetään oikosulkumoottorin yksi- tai kaksisuuntaisessa suorassa kontaktoriohjauksessa, voidaan ohjata myös jarrullista moottoria. Ohjelmalohkon on tarkoitus toimia laiteohjausohjelmalohkon orjana hoitamaan HW:n ohjauksen toiminnankuvauksen mukaisesti.

Toiminnankuvaus

Ajokäskyllä eteen lohko ohjaa moottoria eteenpäin. Antamalla samanaikaisesti käsky ajaa taakse, moottori lähtee pyörimään taakse viiveen jälkeen. Antamalla käsky pelkästään moottorin ajoon taakse, lohkon lähtö ei mene päälle. Suunnanvaihtoviive pystytään asettamaan lohkon liitynnän parametrilla.

Lohkolla voidaan ohjata myös moottorin jarrua. Jarrun avauksessa viive vaikuttaa moottorin käynnistymiseen, eli jarru avautuu heti ohjauksen alkaessa, mutta moottori lähtee pyörimään viiveen jälkeen. Ohjauksen päättyessä moottorin ohjaus päättyy heti ja jarru menee kiinni viiveen jälkeen.

Lohkon kaikki hälytykset ja varoitukset asettuvat päälle. Hälytykset ja varoitukset voidaan kuitata lohkossa olevalla tulolla, kun kaikki häiriöt on poistettu.

Kun moottoriin liittyvä turvapiiri laukeaa, moottorin ohjaus lopetetaan ja jarru asetetaan heti kiinni.

Jos jokin moottorin suojalaitteista laukeaa, ohjaus- tai syöttöpiirissä on häiriö, moottorin ohjaus lopetetaan ja jarru menee kiinni viiveen jälkeen.

Lohkolle tuodaan takaisinkytkentätieto, jolla valvotaan moottorin käyntiä eteen tai taakse. Lohkon sisällä voidaan asetella viive, jonka sisällä takaisinkytkentätiedon on tultava. Käynnistymisen yhteydessä olevan viiveen jälkeen takaisinkytkentätiedon poistuessa tulee välittömästi häiriö ja moottorin ohjaus lopetetaan sekä jarru menee kiinni viiveen jälkeen. Ohjattaessa moottoria uuteen suuntaan valvontaviive alkaa

suunnanvaihtoviiveen jälkeen. Jos takaisinkytkentätieto tulee lohkolle eri suunnasta kuin moottoria ohjataan tai tieto tulee, vaikka moottoria ei ohjata, aiheuttavat nämä vastaavat hälytykset.

Ohjelmalohkolla on seuraavien taulukoiden mukaiset tilat ja häiriöt. Ohjelmalohkon tilat on pakattu UDT:hen, jonka nimi on DirectOnLineStarter_States. Ohjelmalohkon häiriöt on pakattu UDT:hen, jonka nimi on DirectOnLineStarter_Alarms.

Ohjelmalohkon tilat:

Tila	Selitys
bAlarm	Laitteessa *hälytys* tason häiriö, laite pysähtyy ja sitä ei voi käynnistää. Hälytystila aiheutuu, jos jokin hälytyslistan hälytyksistä menee päälle
bWarning	Laitteessa *varoitus* tason häiriö, laitetta voidaan käyttää, mutta vaatii käyttäjän toimenpiteitä
bReady	Laite valmiina, odottaa ajokäskyä
bStarting	Laite käynnistymässä
bRunningFwd	Laite käynnissä, eteen
bRunningRev	Laite käynnissä, taakse
bStopping	Laite pysähtymässä

Ohjelmalohkon häiriöt:

Häiriö	Selitys
a_bSafetyCircuit	Turvapiiri lauennut
a_bShortCircuitProt	Oikosulkusuoja lauennut
a_bOverloadProt	Ylikuormitussuoja lauennut
a_bThermistorProt	Termistorisuojaus lauennut
a_bSafetySwitchOpen	Turvakytkin auki
a_bControlCentreFault	Ohjauskeskusvika
a_bFieldFault	Kenttävika
a_bFeedbackFwdFault	Vika takaisinkytkennässä, suunta eteen
a_bFeedbackRevFault	Vika takaisinkytkennässä, suunta taakse
a_bCrossFeedback	Takaisinkytkentätieto tulee ristikkäiseen tuloon

a_bFeedbackNoControl	Takaisinkytkentätieto tulee, vaikka ei ohjausta

Liitynnän kuvaus

Lohkon tulot:

SW-tulot:

SW-tulot	Kommentti
i_bRun	Ajo eteen
i_bReverse	Ajon suunnanvaihto taakse
i_bAcknowledge	Toimilaitteen häiriöiden kuittaus

HW-tulot:

HW-tulot	Kommentti
i_bSafetyCircuitOk	Turvapiiri kunnossa
i_bShortCircuitProtOk	Oikosulkusuojaus kunnossa (esim. moottorinsuoja-kytkin, sulakkeiden tilatieto)
i_bOverloadProtOk	Ylikuormitussuojaus kunnossa (esim. lämpörele)
i_bThermistorProtOk	Termistorisuojaus kunnossa (esim. termistorirele)
i_bSafetySwitchClosed	Turvakytkin kiinni
i_bControlCentreFault	Ohjauskeskusvika
i_bFieldFault	Kenttävika
i_bFeedbackFwd	Takaisinkytkentätieto, suunta eteen
i_bFeedbackRev	Takaisinkytkentätieto, suunta taakse

Lohkon lähdöt:

SW-lähdöt:

SW-lähdöt	Kommentti
o_udtStates	UDT DirectOnLineStarter_States sisältää kaikki lohkon toimintatilat.
o_udtAlarms	UDT DirectOnLineStarter_Alarms sisältää kaikki hälytykset ja varoitukset
o_bAlarm	Lohkossa hälytys päällä
o_bWarning	Lohkossa varoitus päällä

HW-lähdöt:

HW-lähdöt	Kommentti
o_bControlFwd	Ohjaus eteen
o_bControlRev	Ohjaus taakse
o_bBrake	Jarrun ohjaus

Lohkon tulo-lähdöt:

Ei ole.

Lohkon parametrit:

Ulkoiset parametrit:

Parametri	Kommentti
p_timChangeDelay	Suunnanvaihtoviive

Sisäiset parametrit:

Parametri	Kommentti
timRunDelay	Käynnistysviive ohjauskäskyn saapumisen ja jarrun avautumisen jälkeen
timBrakeDelay	Jarrun kiinnikytkentäviive moottorin pysähtyessä
timFeedbackDelay	Takaisinkytkennän monitoroinnin viive käynnistyk-sessä

LIITE 2

Aseta notepad++:n Show vertical edge päälle ja merkkien
määräksi 66 merkkiä. Tekstin saa rivitettyä notepad++:ssa
näppäinyhdistelmällä Ctrl + I. Jos teksti on rivitetty tälle
alueelle sen pitäisi näyttää jotakuinkin samalta TIA:ssa.

Alueen erotin on Structured Text-ohjelmointikielestä
tuttu rajatun kommentin erotinmerkki (*..*)
(**)

TÄMÄ ON OTSIKKO (otsikkoa seuraa erotinrivi)
(**)
Otsikkoon liittyvän aiheen teksti tähän. (alue päättyy
erotinriviin.)

ENSIMMÄINEN VÄLIOTSIKKO
Ensimmäinen kappale (Kappaleet erotetaan uudella rivillä.)

Toinen kappale.

TOINEN VÄLIOTSIKKO
Ensimmäinen kappale (Kappaleet erotetaan uudella rivillä.)

Toinen kappale.
(**)

TOINEN OTSIKKO (otsikkoa seuraa erotinrivi)
(**)
Otsikkoon liittyvän aiheen teksti tähän.

Jos aiheeseen liittyy esim. joitakin listoja tai

taulukkotyyppisiä asioita. Listataan esimerkiksi seuraavasti.

-Ensimmäinen asia

Ensimmäisen asiaan liittyvä tarkennus

-Toinen asia

Toiseen asiaan liittyvä tarkennus

(**)

LIITE 3

Tämä liite julkaistaan vain työn teettäjän versiossa.