OULUN AMMATTIKORKEAKOULU

Niko Kautto

ETHERCAT-SIMULAATIO

ETHERCAT-SIMULAATIO

Niko Kautto
Opinnäytetyö
Kevät 2021
Sähkö- ja automaatiotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Sähkö- ja automaatiotekniikka, Automaatio

Tekijä: Niko Kautto
Opinnäytetyön nimi: EtherCAT-simulaatio
Työn ohjaajat: Harri Aaltonen ja Manne Tervaskanto
Työn valmistumislukukausi ja -vuosi: Kevät 2021 Sivumäärä: 58

Tämän työn tavoitteena oli perehtyä Beckhoffin TwinCAT3-ohjelmointiympäristöön sijoittuvaan TE1111-lisäosaan ja sen ominaisuuksiin sekä EtherCAT-väylän ominaisuuksiin. Työssä käsiteltiin myös IEC 61131-3 -standardia ja sen tarjoamia ohjelmointikieliä ja toteutettiin esimerkkisovellus hyödyntäen käsiteltyjä asioita.

Työ toteutettiin perehtymällä Beckhoffin materiaaleihin heidän tarjoamistaan komponenteista ja käytiin tarkemmin läpi teoriaa komponenttien lähellä olevista asioista käyttäen monia eri lähteitä. IEC 61131-3 -standardin käsittelyllä perusteltiin ohjelmointiin liittyvän kielen valinta ja käytiin läpi hieman olio-ohjelmoinnin periaatteita. Ohjelmointiesimerkki toteutettiin yhdellä koneella virtuaaliympäristöä hyödyksi käyttäen, koska fyysisiä laitteita ei ollut työn tekohetkellä tarjolla.

Työn tuloksena saatiin kokonaisuus, jonka avulla lukija saa näkemyksen EtherCAT-väylästä, TwinCAT3-ohjelmasta ja IEC 61131-3 -standardeista. Työn avulla lukija osaa myös käyttää TwinCAT3-sovellusta ohjelman luomiseen sekä käyttää TE1111-komponentin tuomia ominaisuuksia simulointiin.

Asiasanat: datasähkö, simulointi, TE1111, TwinCAT3, IEC 61131-3, EtherCAT

ABSTRACT

Oulu University of Applied Sciences
Electrical and Automation Engineering, Automation Engineering

---

Author(s): Niko Kautto
Title of thesis: EtherCAT Simulation
Supervisor(s): Harri Aaltonen and Manne Tervaskanto
Term and year when the thesis was submitted: Spring 2021
Number of pages: 58

The main goal with this project was to get familiar with Beckhoff's TwinCAT3 programming environment belonging TE1111 add-on and its features with EtherCAT bus. Project also included insight to IEC 61131-3 standards, the programming languages provided by it and implemented example program utilizing discussed topics.

The work was carried out by familiarizing oneself with the materials provided by Beckhoff on the components they developed and reviewing the theory of things close to the components in more detail using many different sources. The discussion of IEC 61131-3 standard justifies the choice of programming language and includes a bit of the basic principles of object-oriented programming. The programming example of the work was implemented on one personal computer using the help of virtual machine environment. Virtual machine was used in project mainly because physical equipment was not available by the time the work was done.

As a result the reader gets an overview of work was set that it allows reader to tell about EtherCAT bus, TwinCAT3 program, IEC 61131-3 standards, use the TwinCAT3 application to create program and will be able to use as well as the features provided by the TE1111 component in simulation.

Keywords: PLC, Simulation, TE1111, TwinCAT3, IEC 61131-3, EtherCAT

SISÄLLYS

SANASTO

Bitti	Binäärinen numero
CODESYS	Ohjelmointiympäristö
Ethernet	Standardisoitu tapa yhdistää laitteita langallisesti verkkoon
EPU EtherCAT	Processing Unit
ESC EtherCAT	Slave Controller
FMMU	Fieldbus Memory Management Unit
FPGA	Field-programmable gate array
HMI	Human Machine Interface
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
Kehys	Lähetetyn tietopaketin kokonaisuus
PC	Personal Computer
PLC	Programmable Logic Controller
POU	Program Organization Unit
Redundanssi	Tiedon ylimäärä järjestelmän toiminnan varmistamiseksi
RJ45	Ethernet-liitäntä
Solmu	Yksi laite isossa verkossa
Synkronointi	Tietokoneen ja prosessin välinen samanaikaisuus
TCP	Transmission Control Protocol
Topologia	Yhdistettyjen laitteiden verkko
TwinCAT	The Windows Control and Automation Technology
UDP	User Datagram Protocol

1 JOHDANTO

Opinnäytetyön tavoitteena oli saada käsitys, miten Beckhoff toteuttaa erilaisia järjestelmiä hyödyntäen EtherCAT-väylää, ja perehtyä TwinCAT3-ohjelmaan sisältyvään TE1111-komponenttiin, jota käytetään fyysisten laitteiden simuloinnissa. Työssä haluttiin käydä myös tarkemmin läpi eri IEC 61131-3 ohjelmointikielivaihtoehtoja ja opetella olio-ohjelmoinnin periaatteita Struct Text -kielen avulla.

Aluksi työssä käsitellään EtherCAT-kommunikointia, jonka Beckhoff on rakentanut Ethenet-protokollaa hyväksikäyttäen ja esitellään EtherCAT-väylän rakennetta. Väylän esittelyn jälkeen käsitellään sitä, miten simulaatio TE1111-komponentilla toimii ja käydään läpi teoreemaa toiminnan takana. Komponentista kerrotaan myös mahdollisuudet erilaisista kytkennöistä ja eri kokonaisuuksista, joita käyttäjä voi rakentaa. Komponentin käsittelyn jälkeen kerrotaan IEC 61131-3 -standardin mukaisista kielistä ja esitellään niiden vahvuuksia verraten tekstuaalisia ja graafisia kieliä toisiinsa. IEC 61131-3 -standardin kielistä otettiin käyttöön Structured Text -kieli, jota olio-ohjelmointi tukee parhaiten standardin vaihtoehdoista ja käytiin läpi olio-ohjelmoinnin periaatteita. Läpikäytyjä aiheita hyödynnettiin lopuksi esimerkkiin, joka toteutettiin TwinCAT3-ohjelmalla. Ohjelmaesimerkkiin sisältyy aluksi esimerkki siitä, miten projekti aloitettiin TwinCAT3:lla ja ohjelmaesimerkin kautta siirryttiin linkityksiin ja vaatimuksiin, joita virtuaaliympäristön ylösajaminen edellytti TwinCAT3:n yhteydessä.

Työ toteutettiin ilman fyysistä laitteistoa käyttäen hyväksi TwinCAT3:n tarjoamia ohjelmistokomponenttien ilmaisia lisenssejä ja virtuaalikonetta. Kirjoitus toteutettiin käyttäen hyväksi Beckhoffin tarjoamaa materiaalia sen komponenteista ja TwinCAT3-ohjelmasta sekä hyödynnettiin erilaisia lähteitä aiheisin liittyen.

2 ETHERCAT

EtherCAT on Beckhoff Automation -nimisen yrityksen kehittämä väylätekniikka, jota käytetään monenlaisiin PLC-ohjauksiin sekä reaaliaikaisiin ohjausjärjestelmiin. EtherCAT on peräisin kenttäväylästä nimeltä LightBus, minkä Beckhoff kehitti jo 1980-luvun alussa vastaamaan kasvaviin kaistanleveysongelmiin eri laitteiden välillä. EtherCAT itsessään esiteltiin 2003 nykyisessä muodossaan ja sen oikeudet luovutettiin samalla EtherCAT Technology Groupille, joka on vastuussa väylätekniikan kehityksestä sekä sen IEC 61158 -standardeista. (1.) Standardi IEC 61158 on valmistajien yleinen säädös, joka pitää automaatiojärjestelmien ja kenttäväylien yhteen liittämisen ideologian samanlaisena kommunikoinnin helpottamiseksi (2, s. 15).

2.1 EtherCAT-kommunikointi

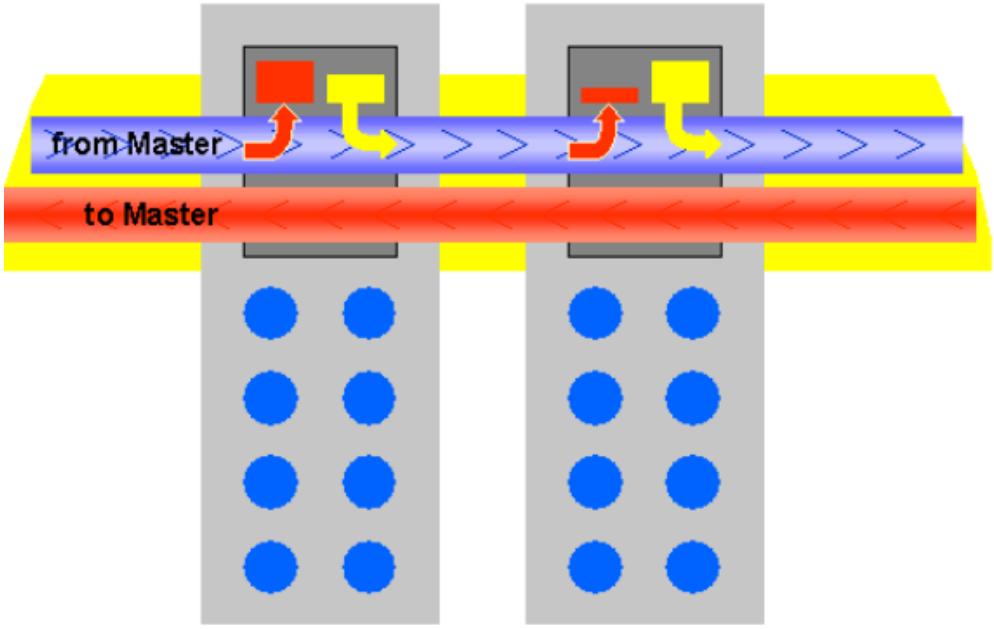
EtherCAT käyttää monien muiden protokollien tavoin Ethernet-protokollaa omana pohjanaan, mutta muokkaa sen viestintää yksinkertaisempaan suuntaan karsimalla Ethernet-protokollan ominaisuuksia. Ethernet-protokolla on erittäin hyvä tapa ja yleisesti paljon käytetty varsinkin tietokoneiden tiedon välitykseen. Teollisuusautomaatiossa tarvitaan kuitenkin nopeaa kommunikointia, jonka takia monet valmistajat ovat joutuneet muokkaamaan protokollaa. Tyypillinen Ethernet-kommunikointi tapahtuu TCP/IP- tai UPD/IP:n välityksellä. TCP/IP-kommunikoinnissa lähettävä laite laittaa kyselyn viestiä vastaanottavalle laitteelle ja vahvistuksen saadessaan se alkaa jakamaan tietoa toisen laitteen kanssa. TCP/UDP-kommunikoinnissa lähettävä laite lähettää tietoa määritettyyn osoitteeseen eikä varmista tiedon saapumista kohteeseen, kuten TCP/IP-kommunikoinnissa. Moni valmistaja on valinnut UDP/IP-yhteyden omaan protokollaansa, koska sen tiedonsiirto on nopeampaa ja vähemmän kuormittavaa kuin TCP/IP-tiedonsiirron.

Täyden Ethernet-tuen ansiosta väylä pystyy kommunikoimaan samanaikaisesti monen eri protokollan kanssa, kuten yleisen HTTP-protokollan kanssa ja teollisuusprotokollan Profinetin kanssa (3, s. 12). Yhteensopivuus EtherCAT- ja Ehternet-kommunikointia käyttävien laitteiden välillä on hyvä, koska reaaliaikainen tiedon lähetys onnistuu helposti Ethernet-protokollan avulla PC maailmaan. Verkkosivuille reaaliaikainen tiedon lähettäminen onnistuu samankaltaisella protokollalla helposti. Lyhin Ethernet-kehys on 84 bitin kokoinen ja pieni sovellus, joka pyörii PLC:lla ja voi olla

esimerkiksi 4 bitin kokoinen. Normaaliin Ethernet-kehykseen sisältyvät PLC-ohjauksen lisäksi esimerkiksi TCP osan segmentit, jotka EtherCAT-protokolla jättää pois paketista, kun se liikuttaa tietoa omassa väylässä. Ethernet-protokolla ilman muutoksia lähettää siis paljon turhaa tietoa, mikä hidastaa väyläliikennettä. Ethernetin lähettämän kehyksen suuruuden pienentyessä 84 bitistä 4 bittiin saadaan samassa suhteessa vähennettyä aikaa, joka kuluu tiedonvälitykseen toimilaitteen ja PLC:n välille. (3, s. 10.)

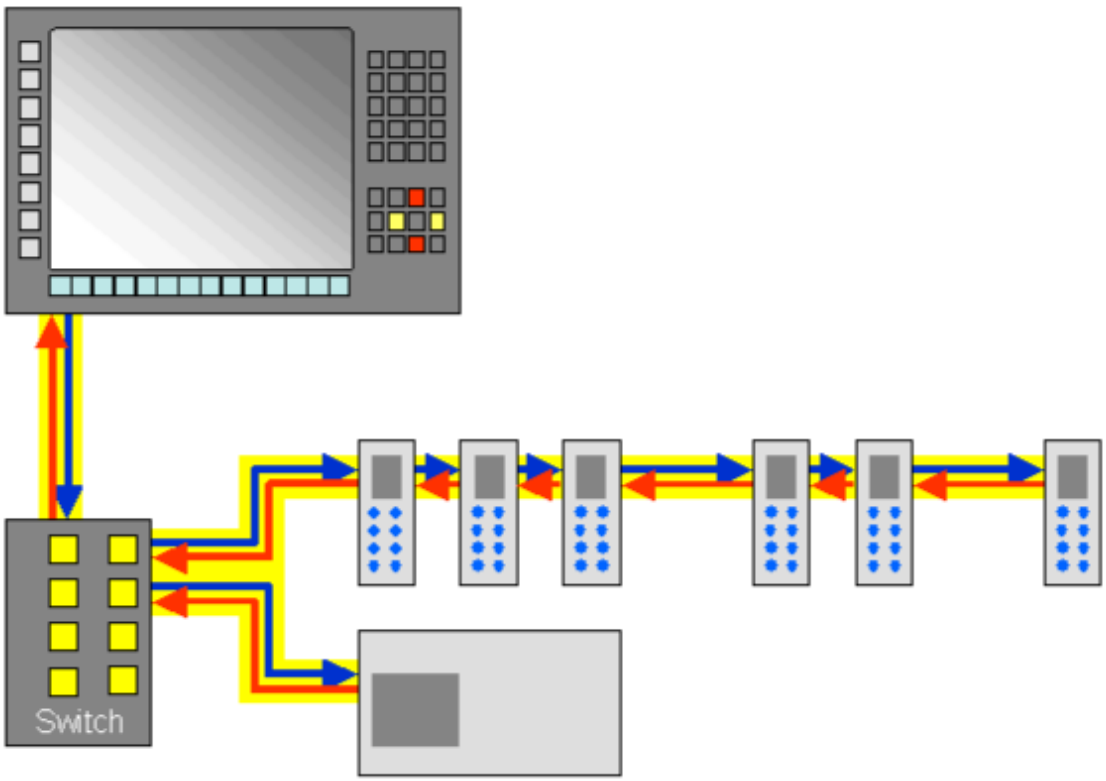
2.1.1 EtherCAT-tekniikka

EtherCAT käyttää ratkaisua, jossa paketteja ei vastaanoteta, tulkita ja kopioida enää erikseen vaan jokaisessa väylän laitteessa on FMMU, joka lukee kyseiselle laitteelle tarkoitetun viestin sekä tiedot. Tiedon kulku ei pysähdy laitteeseen vaan jokainen toimilaite lisää väylässä liikkuvaan pakettiin oman osansa, joka päätyy takaisin tietoa lähettävään laitteeseen. (3, s. 10.) Tämä nopeuttaa järjestelmää verrattuna tavalliseen Ethernet-malliin, koska tyypillistä vahvistusta ei tarvita, vaan jokaisessa laitteessa on oma FMMU-yksikkö jakamassa takaisin liikkuvaan kehykseen tiedot ilman päätelaitteen tulkintaa. Päätelaite siis vastaanottaa valmiiksi muunnellun kehyksen jokaiselta laitteelta kytketyssä järjestelmässä. (Kuva 1.)



KUVA 1. EtherCAT-kehyksen liike (3, s. 10)

EtherCAT käyttää IEEE 802.3 -Ethernet-standardia, joten käyttöönottoon ei tarvita erillistä tai valmistajan omaa liitäntää. Järjestelmään yhteensopivat laitteet eivät tarvitse ulkoisia kytkimiä, koska jokaisessa laitteessa on kaksi RJ45-porttia (4). Tämä tekee kentän suunnittelusta ja rakentamisesta huomattavasti halvempaa, kun yhdistetään laitteisiin erilaisia teollisuus-PC:tä, joissa on aina vakiona RJ45-liitäntä yhdistämistä varten. Erilaisten HMI-laitteiden liittäminen järjestelmään tulee myös olemaan helpompaa (kuva 2).



KUVA 2. EtherCAT-arkkitehtuuri (3, s. 11)

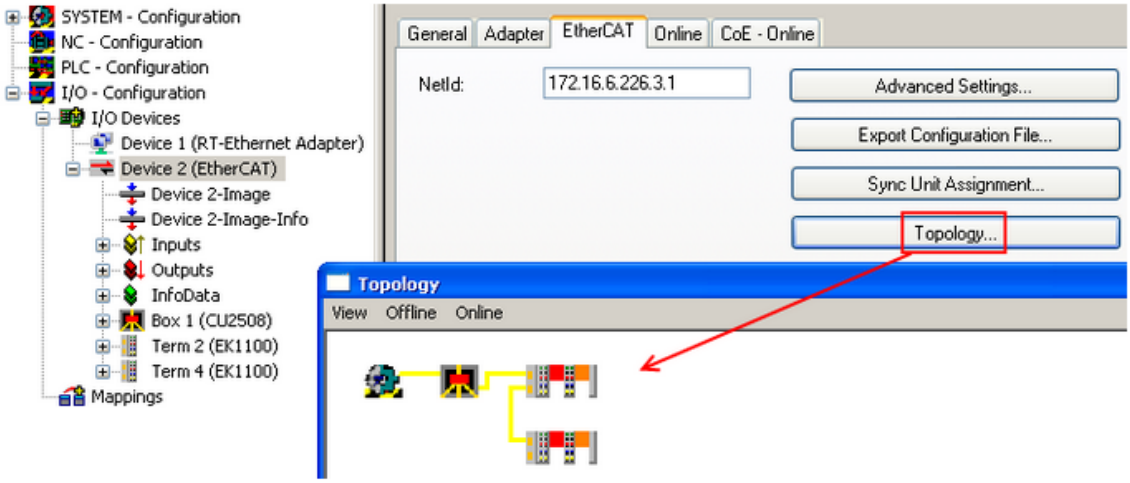
2.1.2 EtherCAT-protokolla

EtherCAT-protokolla on optimoitu prosessidataa varten ja se lähetetään suoraan muokatussa Ethernet kehyksessä. Datan lähetysjärjestys on riippumaton kentällä olevien fyysisten laitteiden kytkennöistä ja jokaisen laitteen välillä on mahdollista kommunikoida tarpeen vaatiessa. Kytkennät eivät ole rajoitettuja sisäiseen aliverkkoon vaan EtherCAT voi pakata tiedot UDP/IP-muotoon ja tätä kautta jokainen verkossa oleva Ethernet-prokollaa käyttävä laite voi kommunikoida järjestelmän kanssa. Paketti tarvitsee muuttaa vain yhdessä asemassa, jonka jälkeen se pystyy liikkumaan Ethernet-protokollaa käyttävien laitteiden välillä. (3, s. 12.) Paketin muuttaminen vain yhdessä laitteessa rasittaa verkkoa vähän ja sitä kautta tiedonsiirron hidastuminen on erittäin vähäistä, mikä mahdollistaa kriittisten laitteiden tiedon siirron monelle eri protokollan laitteelle ilman ylimääräistä hitautta.

2.2 EtherCAT-topologia

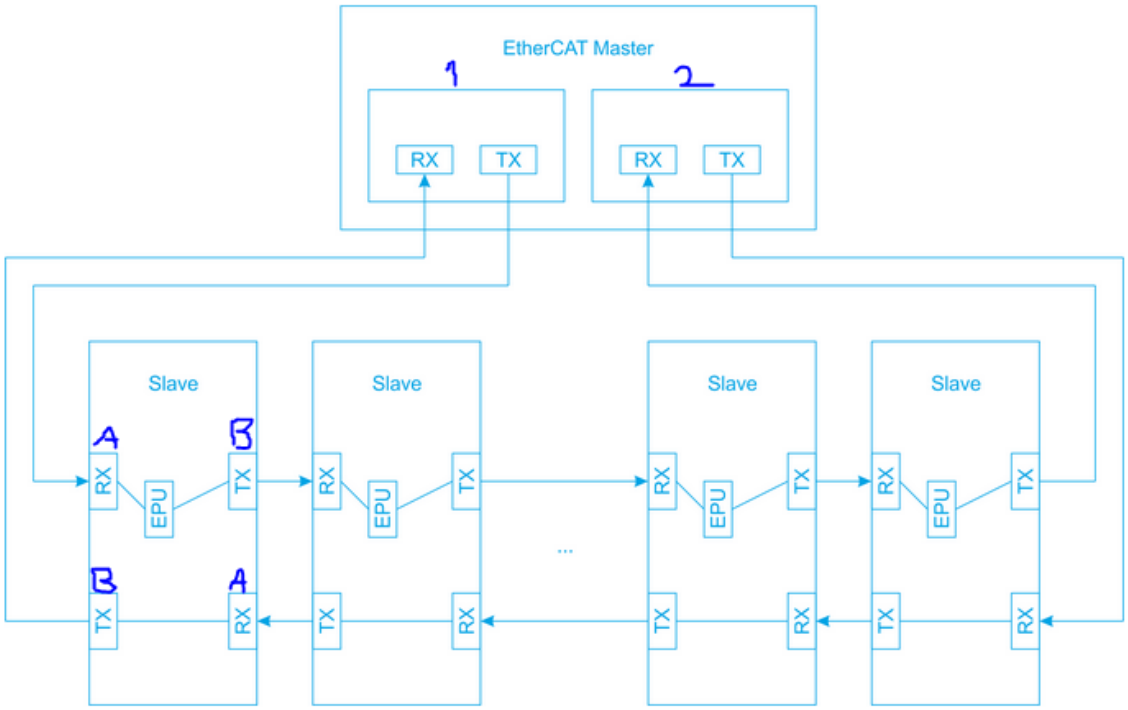
EtherCAT:in monipuolisuus tulee vastaan myös topologiassa. Verkkorakennetta suunniteltaessa ei tarvitse välittää miten koneet on sijoitettu verkossa, koska EtherCAT ei ole rajoitettu kytkinten määrän tai niiden kaskadiin kytkemisen kannalta (5, s. 5). Kaskadikytkentä tässä tapauksessa tarkoittaa esimerkiksi laitteen säätöä, jossa sisäkkäisten toimilaitteiden määrää ei ole rajoitettu kytkennässä, joka palauttaa asetusarvon säädön ensimmäiselle laitteelle. Arvot kaskadikytkentään tulevat eri toimilaitteilta samaan aikaan. Tämän tyylinen verkon rakenne vapauttaa erilaiset kytkentämahdollisuuden EtherCAT:ssä.

Verkko on rakennettavissa mihin tahansa muotoon, koska EtherCAT-verkossa isäntälaite jakaa osoitteet erikseen kaikille orjalaitteille käyttäjän halutessa. Virtuaaliseksi tai kuvitteelliseksi kytkentämalliksi voidaan todeta linjatopologia, koska isäntälaite määrää orjalaitteiden järjestyksen. Topologian vapaudessa voi olla myös haittoja muille kuin suunnittelijalle. Järjestelmä toimii hyvin ilman muutoksia, mutta uuden komponentin lisääminen järjestelmään voi aiheuttaa ylimääräistä työtä. Osoiterakenne täytyy pitää loogisessa muodossa, koska se helpottaa järjestelmän käyttöä sen luovutuksen jälkeen. Beckhoffin TwinCAT3-ohjelma antaa hyvän visualisoinnin siitä, montako laitetta topologiassa on, joten virheiden riski on pieni (kuva 3).



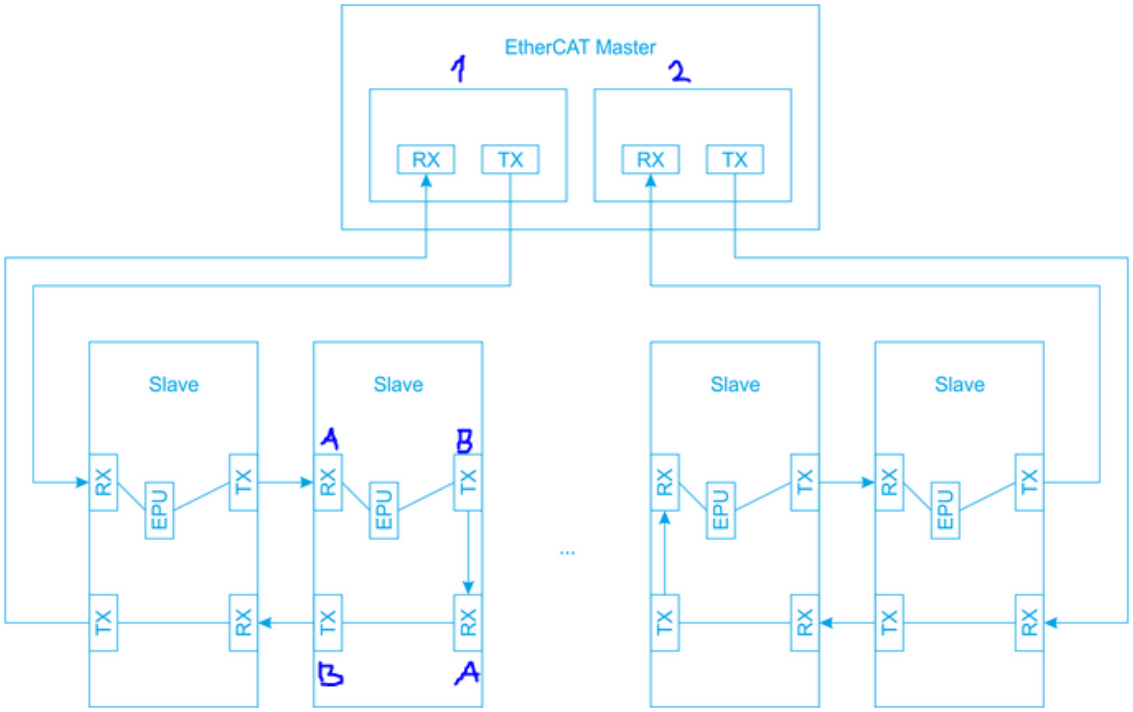
KUVA 3. TwinCAT3-topologia kuva (6)

Jokaiseen EtherCAT-orjayksikköön sisältyy prosessointiyksikkö (EPU), jonka kautta kehykset järjestelmässä kulkevat. Yksikön toimintaan sisältyy rekisterin luku-, muisti- ja datanprosessointielementtejä. Redundanssia voidaan hyödyntää vain järjestelmissä, joissa on käytetty ympyrä topologiaa. EtherCAT-isäntälaite tarvitsee tuolloin toisen verkkoadapterin lähettämään täysin identtistä kehystä toiseen kertaan. Esimerkiksi portille A tuleva signaali voi vaihtaa tietoja prosessointiyksikön kautta mennessään lähtevää porttia B päin, mutta toiseen suuntaan tullessaan portista B portille A tietoa ei tutkita prosessointiyksikössä. (Kuva 4.) Ellei käytetyssä ympyrätopologiassa ole kaapelivikaa, ensimmäinen sovitin lähettää kehyksen, joka näyttää identtiseltä saapuessaan toiselle sovittimelle. Kytkennöissä täytyy ottaa huomioon, että jokainen tulo ja lähtö on oikeaan suuntaan, koska kahden sovittimen järjestelmä ei toimi oikein kytkentöjen ollessa väärin. (7.)



KUVA 4. Ehjä kommunikointi isäntä- ja orjalaitteiden välillä (8)

Vikatilanteen ilmentyessä ympyrätopologia hylätään ja sen fyysinen redundanssi häviää. Ohjelmallinen redundanssi kuitenkin säilyy kahden sovittimen ansiosta. Ensimmäinen sovitin lähettää dataa viimeiseen tuloporttiin asti ja kaapelivian sattuessa järjestelmä tunnistaa viallisen laitteen. Kehys, joka lähetetään ensimmäisestä laitteesta, luetaan samalla tavalla kuin ehjässä järjestelmässä prosessointiyksikön avulla ja välitetään samalta laitteelta takaisin ensimmäiselle sovittimelle. Toinen sovitin toimii tässä tapauksessa samalla tavalla. Ainoa ero sovittimien toiminnassa on se, että toinen sovitin lukee tiedot orjalaitteilta vasta paluumatkalla toisin kuin ensimmäinen, joka ottaa tiedon ensin vastaan. (Kuva 5.) Kumpikin sovitin siis kerää edelleen identtisestä kehyksestä dataa, mutta tällä kertaa kummallakin sovittimella on tehtävä täyttää isäntälaitteen tarvitsemat tiedot (7). Redundanssia voi hyödyntää prosesseissa, joissa jokin osa järjestelmästä on paikassa, johon on hankala mennä vaihtamaan komponenttia tai prosessi pitäisi saada lopetettua ennen komponenttien tai kaapelin vaihtoa. Redundanssin käyttäminen voi auttaa siis kustannuksissa sekä turvallisuudessa, kun vika päästään korjaamaan normaalia vastaavissa olosuhteissa prosessin keskeytyksen jälkeen.



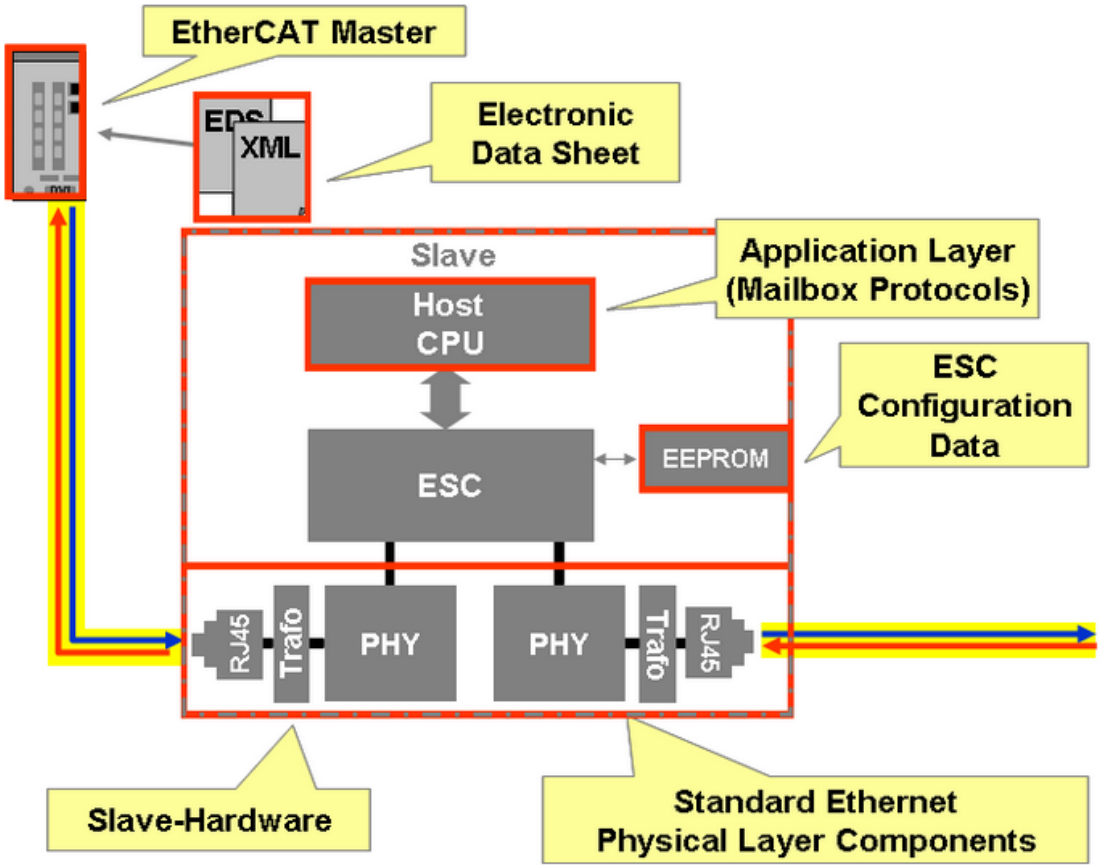
KUVA 5. Vikatilanteeseen reagointi isäntä- ja orjalaitteiden välillä

2.3 Hajautetut kellot

EtherCAT-terminologiassa hajautetut kellot viittaavat loogiseen verkkoon, jossa yhteensopivat laitteet kommunikoivat keskenään samaan aikaan. Hajautetut kellot pystyvät synkronoimaan ajan hyvinkin lyhyellä aikavälillä käyttäen apuna EtherCAT:in Ethernet-protokollaa. Jokaisen laitteen täytyy siis tukea hajautettujen kellojen järjestelmää ja isosta järjestelmästä valitaan yksi orjalaite, jonka mukaan muut laitteet synkronoidaan. Toisin sanoen valittu orjalaite kertoo koko järjestelmälle ajan, jota muut laitteet yrittävät tavoitella. Järjestelmän isäntälaite lähettää määritellyn kyselyviestin orjalaitteille lyhyin väliajoin varmistaakseen, että orjalaitteiden kellot ovat pysyneet määrätyissä rajaarvoissa.

Kaikki laitteet seuraavat valitun orjalaitteen synkronointitietoja, jotka välittyvät laitteiden välillä kehyksen avulla. Topologiaan täytyy myös olla määritettynä oikea järjestys orjalaitteille synkronoinnin toimivuuden takia. Orjalaitteen, josta muut laitteet ottavat mallia, täytyy olla topologiassa ensimmäisenä, koska mallilaitetta edeltävät laitteet eivät muutoin synkronoidu oikein. (8.)

Jokaiseen hajautettuja kelloja tukevaan orjalaitteeseen sisältyy elektroninen komponentti, kuten FPGA, joka pystyy käsittelemään yksinkertaisia toimintoja. EtherCAT-järjestelmässä komponenttia kutsutaan nimellä ESC. (Kuva 6.) Komponentin hoitamat toiminnot voivat olla esimerkiksi digitaalisten tulojen ja lähtöjen käsittelyssä mukana, mutta pääasiassa sen tehtävä on huolehtia kellotoiminnoista ja sen tehtävistä (8).



KUVA 6. ESC-komponentin toiminta järjestelmässä (8)

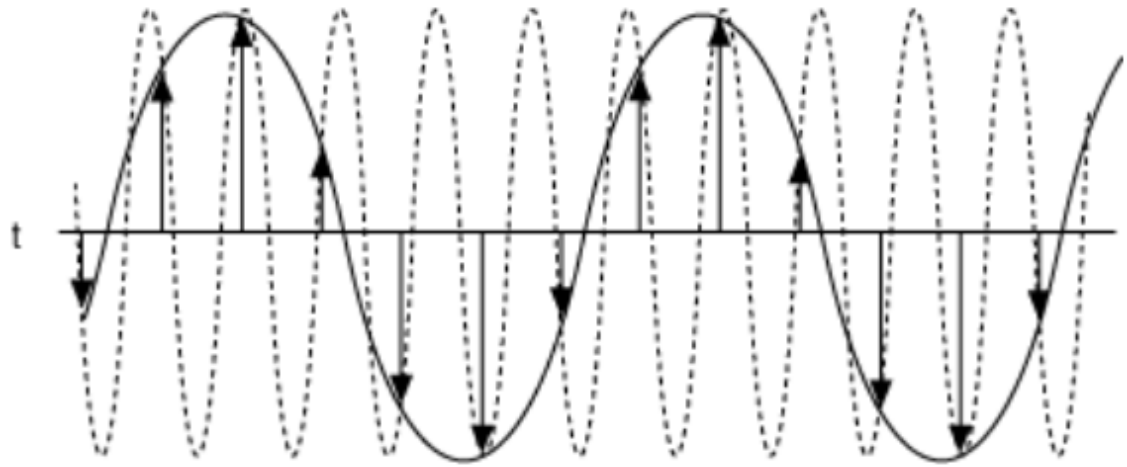
3 TWINCAT3 TE1111 LISÄOSA

TwinCAT3 TE1111 on simulaattorikomponentti TwinCAT3 -sovellusohjelmointialustalla, joka kopioi käyttäjän määrittämän fyysisen laitteen konfiguraation kytkennät peilaamaan todellista konfiguraatiota.

3.1 Shannonin ja Nyquistin teoreema

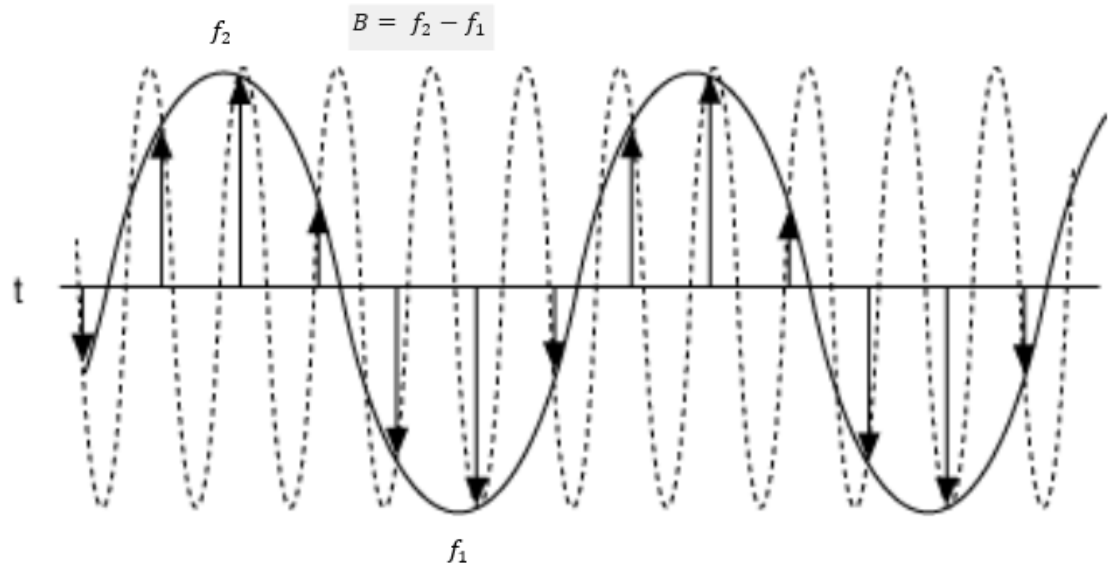
Simulaattoriominaisuus tarjoaa käyttäjälle mahdollisuuden kuvata oikeaa järjestelmää, mutta simuloidun ohjelman täytyy toimia kaksi kertaa nopeammin kuin ohjausjärjestelmän, koska muuten simulaattorin viesti ei välity oikein ohjausjärjestelmälle. Tässä kohdassa tarvitaan Shannonin ja Nyquistin teoreemaa havainnollistamaan taajuuksien muutoksia. (9, s.9.) Puhutaan siis signaalien käsittelystä, jossa valitaan tietyt ajankohdat jatkuvasta aikasignaalista ja käsitellään niitä Shannonin ja Nyquistin teoreeman avulla. Yleisesti teoreemaa kutsutaan näytteenottolauseeksi, johon sisältyvät esimerkiksi kaistanrajoitussignaalit, spektrit ja TwinCAT3:n tapauksessa sinisignaalit, joiden absoluuttinen arvo on isompi kuin nolla. Lause viittaa siihen, että signaalista voidaan löytää riittävän suuri näytteenottotaajuus sen täydelliseen uudelleenrakentamiseen. (10.)

Nyquistin teoreemassa todetaan, että analoginen signaali täytyy näytteistää vähintään kaksi kertaa suuremmalla taajuudella verrattuna näytteenottotaajuuteen uuden signaalin luomiseksi alkuperäistä vastaavaksi. Ellei näytteenottotaajuus ole tarpeeksi suuri, signaali laskostuu matalammille taajuuksille eikä siten vastaa alkuperäistä korkeampaa taajuutta ja sitä kautta se voi näkyä myös alkuperäistä signaalia tutkittaessa vääristävänä tekijänä. (11.) Esimerkiksi 5 MHz siniaaltoa, jota näytteistetään liian hitaalla analogia/digitaali-muuntimella 6 MS/s, ei voida pitää pätevänä, koska signaali laskostuu. S/s tarkoittaa muuntimen tapauksessa näytteenottoja sekunnissa. (Kuva 7.)



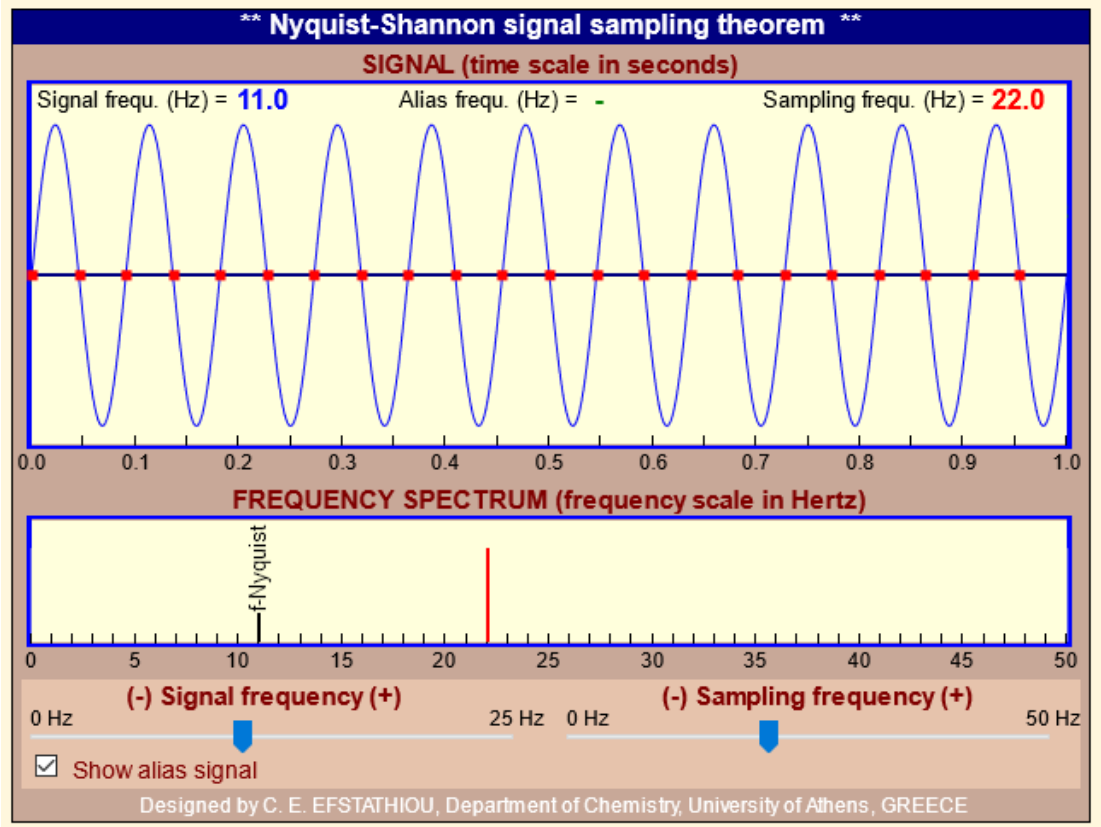
KUVA 7. 5 MHz siniaaltoon laskostunut 1MHz signaali (10)

Analogista signaalia käsitellessä analoginen signaali $s(t)$ voidaan siis muodostaa uudestaan suuremmalla taajuudella kuin $fs > 2B$, jossa B on kaistanleveys. Kaistanleveydellä tarkoitetaan signaalien käsittelyssä ylemmän ja alemman rajataajuuden erotusta. Kaistanleveys B saadaan siis vähentämällä taajuuden huippuarvo f_2 ja minimiarvo f_1 toisistaan. (Kuva 8.) Nyquistin tapauksessa puhutaan yleisesti myös Nyquistin taajuudesta, jota voidaan suoraan verrata maksimitaajuuteen signaalissa (12).



KUVA 8. Kaistanleveys B

Täytyy myös ottaa huomioon mahdollinen erikoistapaus. Kun käsitellään signaalia $s(t)$ taajuudella fs ja siitä uudelleen rakennettu signaali on muotoa $fn = \frac{fs}{2}$, kaikki $s(t)$ korkeammat signaalit kuin fn aliasoituvat nollan ja fn väliin (12). Tapaus on harvinainen, koska häiriön takia signaali ei ole yleensä täysin nollassa ja monitoroidessa pääsee dataa läpi, vaikka asetetut taajuuden arvot olisivat oikeat kuvaamaan tilannetta. Tyypillinen vastaan tuleva tilanne voi olla, että näytteenotto näyttäisi välillä nollaa ja välillä jotakin muuta arvoa hyppien kummankin välillä tasaiseen tahtiin. Tilanteen havainnollistaminen kuvalla on tärkeää ymmärtämisen kannalta signaalin käsittelyssä, joten asetetulla signaalin taajuudella 11 Hz ja näytteenottotaajuudella 22 Hz päästään haettuun tulokseen (kuva 9).

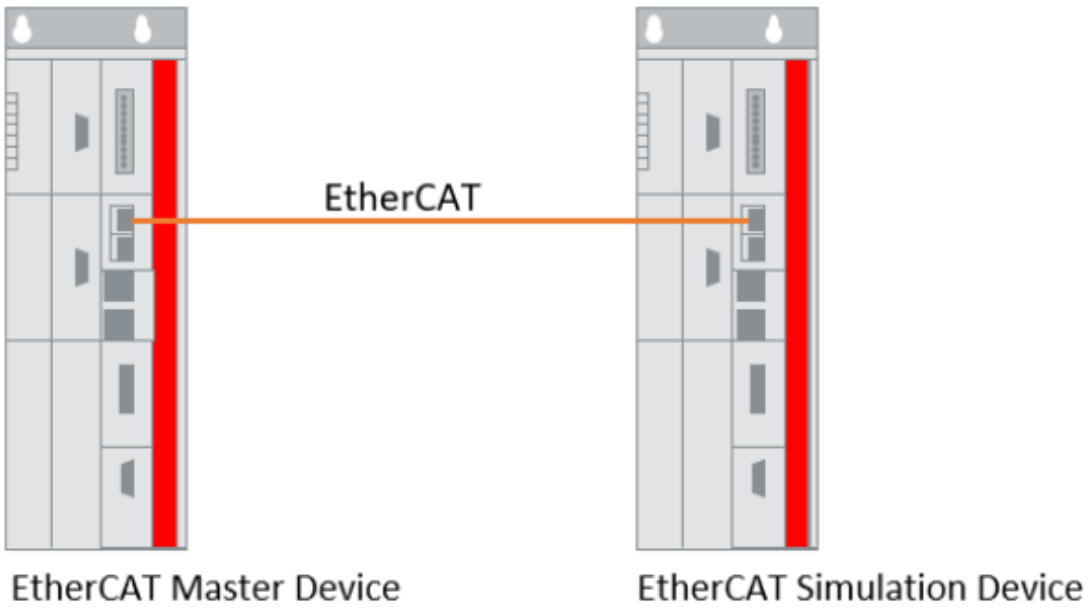


KUVA 9. Tasan kaksinkertainen näytteenottotaajuus signaaliin nähden (13)

Shannonin teoreemassa käytetään täsmälleen samanlaista ajattelua digitaalisen signaalin näytteen käsittelyssä kuin Nyquistin teoreemassa. Digitaalinen signaali on päivitettävä tutkittaessa vähintään kaksi kertaa nopeammin kuin luotavan signaalin kaistanleveys. Esimerkkinä voidaan siis käyttää samanlaisia arvoja kuin Nyquistin teoreemassa, mutta toiseen suuntaan. Käyttäen (kuva 4) arvoja ja tutkittaessa niitä toiseen suuntaan voidaan todeta, että näytteenottotaajuus 6 MS/s ei ollut riittävä muodostamaan kopiota alkuperäisestä 5 MHz:n signaalista. (10.)

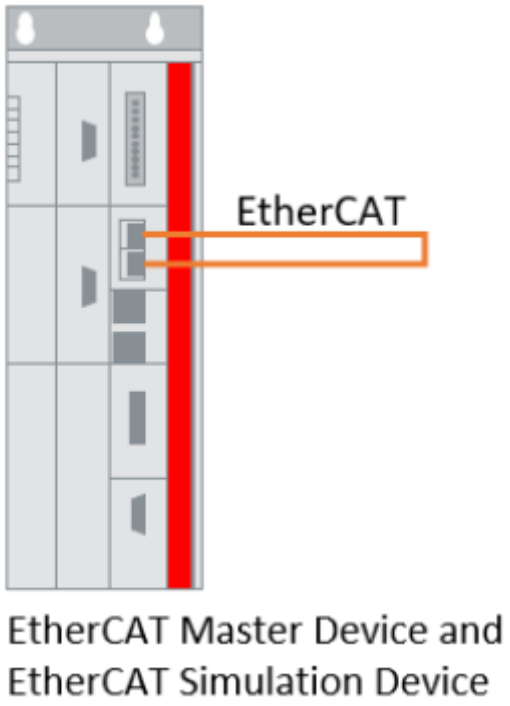
3.2 Kaapelointivaihtoehdot

Kaapeloinnissa on monia erilaisia tapoja rakentaa EtherCAT-väylä monipuolisien kaapelointivaihtoehtojen ansiosta. Yleisin tapa on yhdistää EtherCAT-päälaite simulointilaitteeseen, mikä onnistuu yhdellä kaapelilla. Päälaite sisältää alkuperäisen projektin, jolla laitteistoa halutaan ohjata ja simulointilaitteeseen on laitettu sisälle luonnollisesti simulointiohjelma (kuva 10).



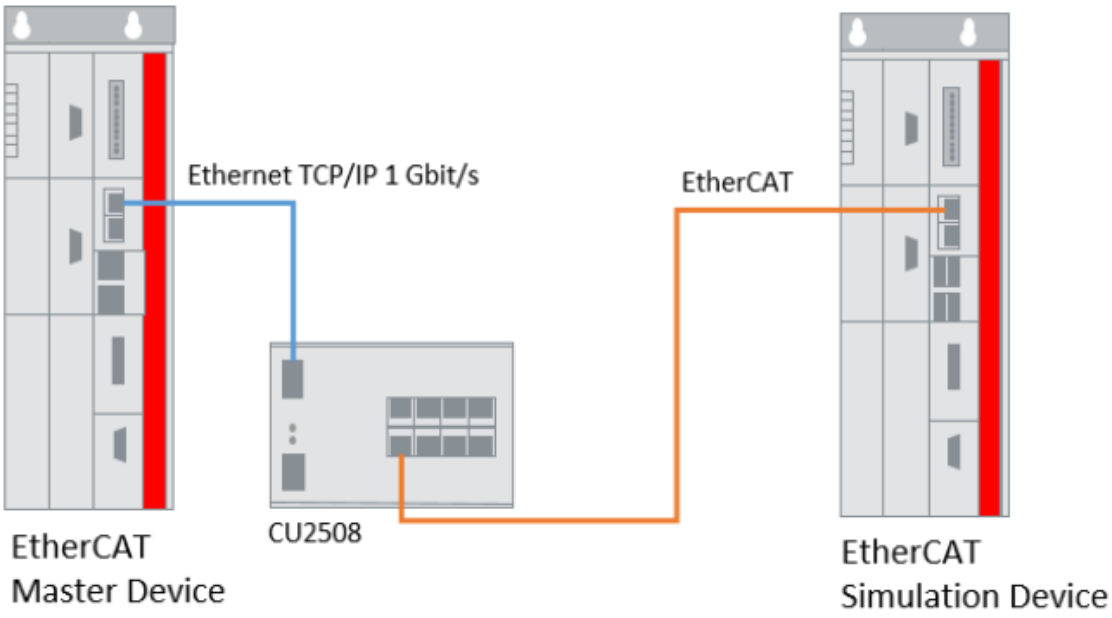
KUVA 10. tyypillinen kytkentä päälaitteen ja simulointilaitteen välillä (9, s.9)

Ohjelmien testausta varten voidaan käyttää kahta samanlaista tietokonetta, joissa ainoa edellytys on kaksi Ethernet-porttia fyysisiä kytkentöjä varten (kuva 11). Tästä ominaisuudesta on hyötyä järjestelmän suunnittelussa, koska käyttäjän ei tarvitse tilata tai odottaa fyysisten laitteiden saapumista paikalle. On erittäin harvinaista, että tietokoneen emolevyssä on valmiina kaksi tai useampi Ethernet-liitäntä, joten ratkaisuna toimii myös adapterin käyttö. USB-Ethernet adapteri sopii tähän tarkoitukseen ja on kustannustehokas ratkaisu.



KUVA 11. Takaisinkytkentä päälaitteeseen (9, s.10)

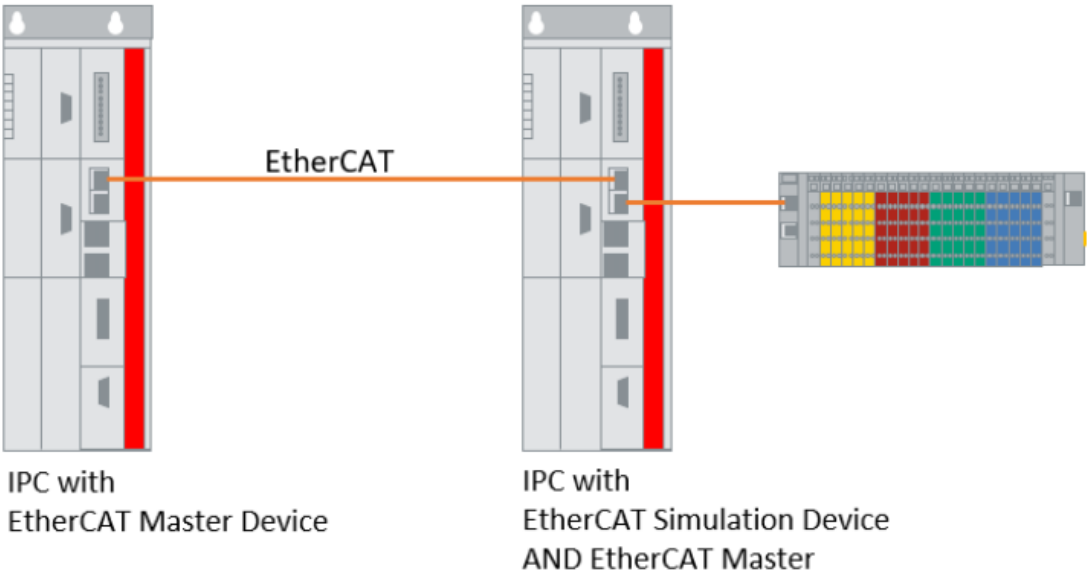
Beckhoff tarjoaa käyttäjälle mahdollisuuden käyttää CU2508-porttikerrointa simulaattorin kanssa, mutta käyttö on mahdollista pelkästään päälaitteelta simulointilaitteeseen päin (kuva 12). Simulointilaitteeseen täytyy luoda sama määrä simuloituja laitteita riippuen siitä, kuinka monta fyysistä kytkentää simulointilaitteeseen halutaan kytkeä. (9, s.10.) Porttikertojaa käytetään säästämään kustannuksissa ja monesti se on valinta, kun valmiiseen järjestelmään lisätään uusia laitteita. Kertojan käyttäminen säästää kaapelointikuluissa sekä lisää järjestelmän modulaarisuutta sen tuomilla lisäkytkentäpaikoilla. (14.)



KUVA 12. Porttikertojan esimerkkikytkentä (9, s.11)

Simulointilaitteen puoli ei tue porttikertojaa, jos sitä käytetään järjestelmässä orjalaitteena. Tiedon siirtyessä orjalaitteelta porttikertojaan päin tieto on puutteellista, koska porttikertoja ei sisälly orjalaitteen luomaan lähetettävään tiedostoon. Porttikertoja CU2508 ei osaa lukea kehystä, jonka väärin kytketty orjalaite lähettää. Porttikertoja osaa lukea viestin silloin, kun järjestys on päälaite, porttikertoja ja orjalaite, mutta lähetettäessä kehyksen orjapuolelta kehys häviää hierarkian toteutumattomuuden takia. Kehyksen hävitessä päälaitteen laitetila ja orjalaskuri eivät näytä oikeita lukuja ja orjalaitteen näkymättömyyden takia päälaite voi myös näyttää virheellisiä arvoja mittauksissa. (9, s.11.)

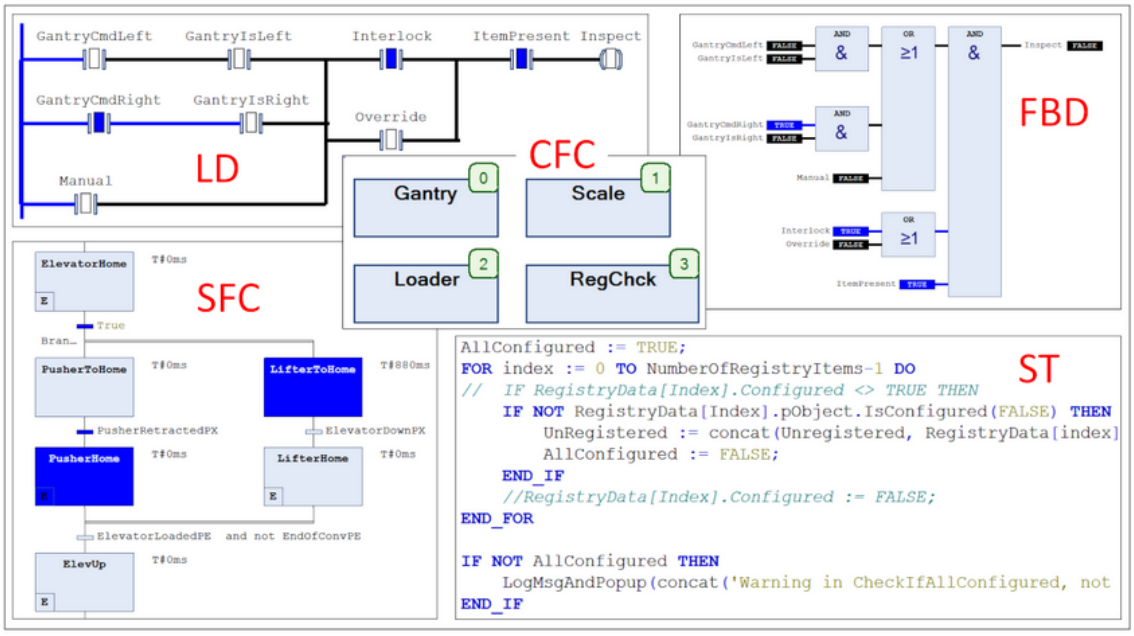
Käyttäjän halutessa simulointilaitteen kytkentä onnistuu samalla ideologialla (kuva 10). Käyttäjä voi luoda yhdelle laitteelle projektin, johon sisältyy pää- ja simulointilaite. Tällä tavalla voidaan luoda monen suuruisia simulointiympäristöjä ja voidaan keskittyä haluttuun osa-alueeseen. Järjestelmässä voi olla valmiina fyysisiä kytkentöjä ja niitä voidaan hyödyntää simuloinnin kanssa. (kuva 13.) Käyttäjän täytyy tässäkin tapauksessa muistaa, että verkkoliitäntöjä on tarpeeksi EtherCAT väylän käyttöä varten (9, s.11).



KUVA 13. Sekoitettu järjestelmä (9, s.11)

4 IEC 61131-3 JA OLIO-OHJELMOINTI

IEC 61131-3 -standardi on yksi osio IEC 61131 -standardista, johon sisältyy ohjelmoitavien logiikoiden ohjelmointikielet ja rakenteet (15). Standardi sisältää viisi eri ohjelmointikieltä Ladder diagram (LD), Function block diagram (FBD), Structured text (ST), Instruction list (IL) ja Sequential function chart (SFC) (kuva 14). Edellä mainitut kielet on suunniteltu määrittämään toimivia kokonaisuuksia ja monet komponenttien valmistajat ovat adoptoineet nämä standardit omia tuotteitaan tukemaan. Vaikka IEC 61131 -standardi kokonaisuutena määrittelee sovittuja rakenteita ja kokonaisuuksia ohjelmiin ja komponentteihin, se ei kuitenkaan määrittele loogisten komponenttien sisältöä eri valmistajien tarjoamissa ohjelmissa. Kokonaisuutena eri valmistajien työkaluilla ohjelmointi helpottuu paljon, koska käyttäjä voi luottaa sovittuihin standardeihin ja niiden olemassaoloon. IEC 61131-3:n ansiosta myös siirtyminen eri logiikkaohjelmiin on helpompaa suunnittelijan kannalta. (16.)



KUVA 14. IEC 61131-3 -ohjelmointikielien kirjoitustavat (20)

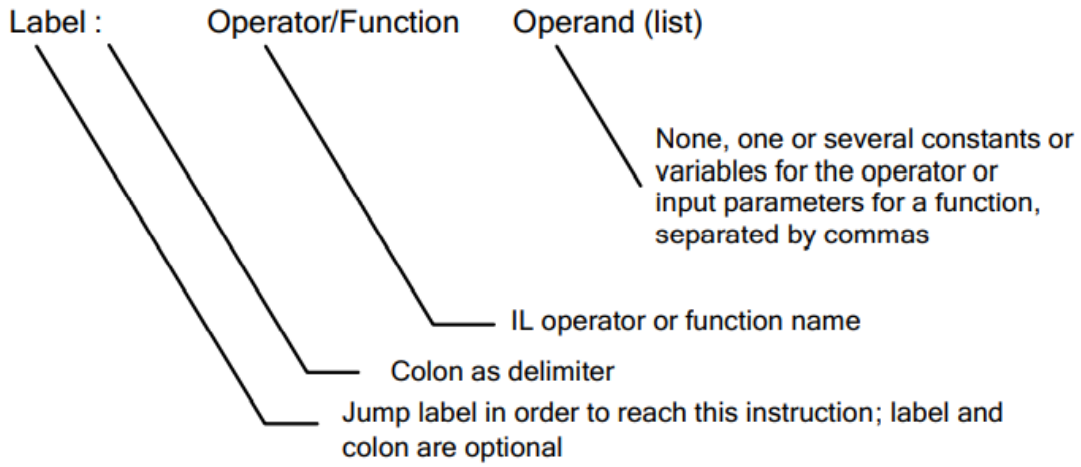
4.1 IEC 61131-3 -ohjelmointikielet

Ohjelmointikielet jaetaan PLC-ohjelmoinnin tapauksessa kahteen eri luokkaan, joihin sisältyvät graafiset ja tekstuaaliset kielet. Tekstuaalisiin kieliin sisältyy Instruction list (IL), Structured text (ST) ja Sequential function chart (SFC). Graafisiin kieliin sisältyy Ladder diagram (LD), Function block

diagram (FBD) ja Sequential function chart (SFC). Näistä määrityksistä voidaan huomata, että (SFC) kuuluu kumpaankin alueeseen ja tätä kieltä voidaan toteuttaa käyttäjän valitsemalla tavalla. Tekstuaaliset kielet luokitellaan korkeamman tason ohjelmointikieliin ja niihin sisältyy monenlaisia kombinaatiota lausekkeista, jotka ohjaavat määritettyjä muuttujia ajon aikana. Graafisilla kielillä halutut toiminnot tehdään graafisilla elementeillä ja liitännät elementtien välillä kuvaavat tiedon kulkua toimilohkojen välillä. (17, s.99.) Kielien erottaminen eri osa-alueisiin ei kuitenkaan poista niiden yhteensopivuutta. Käyttäjä pystyy monessa tapauksessa yhdistelemään erilaisia komponentteja tai ohjelmoida toiminnallisuuksia graafisten toimilohkojen sisään käyttäen tekstuaalista kieltä. Ohjelmointikielien käyttötarkoitus ja toiminta on siis täsmälleen samanlainen ja valinta, mitä kieltä ohjelmoija käyttää, on hänen tai ohjelmointityötä tilaavan yrityksen päätettävissä. Vaikka ohjelmointikielet ovat pohjimmiltaan samanlaisia, silti korkeamman tason tekstuaalisilla ohjelmointikielillä käyttäjä voi luoda monimutkaisempia toiminallisuuksia ja omia graafisia toimilohkoja, mikä tekee tekstuaalisista kielistä kokonaisuutena monipuolisempia.

4.1.1 Instruction List (IL)

Instruction list sivuaa läheltä konekieltä ja sitä käytetään monesti yleisenä kielenä eri kielien välisiin käännöksiin. IL on linjakeskeinen kieli, joka on täsmälleen yhdellä rivillä suoritettava komento suoraan PLC:lle. (17, s.100.) Tunniste kommenttiriville ei ole pakollinen, mutta tehtäessä hyppyjä koodin suoritusjärjestykseen ne ovat välttämättömiä. IL:n koodia voi myös kommentoida aina, kun tyhjää tilaa on tarjolla IEC 61131-3 -standardin mukaan kommenttien käyttö on jokaisen muun kielen kanssa samanlainen sulkuja ja tähtiä lauseen alussa tai lauseen lopussa. (17, s.101.) Käyttäjä voi kirjoittaa haluamansa tekstin määritettyyn muotoon, mutta merkit eivät saa olla kiinni edellisissä määrityksissä IL:ssä. IL ei määritä funktioille ja muuttujille tiettyä muotoa, vaan se on käyttäjän määritettävissä. (Kuva 15.)



KUVA 15. IL:n rakenne (17, s.100)

Funktioiden kutsuminen tapahtuu IL:ssä kutsumalla funktion nimeä ja oikeat parametrit liitetään funktion perään erottamalla ne pilkulla toisistaan. Parametrit voidaan määrittää funktiota kutsuttaessa := merkillä, mutta IL pystyy lukemaan sen omia operaattoreita myös ilman edellä mainittua merkintää. Ensimmäinen funktion parametri täytyy kutsua ennen kuin itse funktiota kutsutaan, koska IL:n tapauksessa ensimmäinen funktion kutsu kohdistuu toiseen määritettyyn parametriin funktiossa. (17, s.109.) IL:n koodin kulku on siis suoraviivaista hyppyjä huomioon ottamatta. Funktioon määritellään sisälle muuttujat ja niiden datatyypit. Funktio suorittaa IL:n tarjoamien komentojen avulla tai käyttäjän kutsumien funktioiden avulla määritetyt toiminnallisuudet (kuva 16.)

```
VAR RETAIN
   Direction:      BOOL;          (* Current direction up or down *)
END_VAR
```

Muuttujan määritys

```
(* System running for the first time after power on? Yes: Reset the end signal *)
(* activated by the last shutdown *)
LD      MRStart                   (* The first call? *)
R       EndSignal                 (* Yes: Reset the warning signal *)
JMPC    ResCount

(* Not the first call! *)
JMP     Arrive
```

Ohjelman kulku IL muodossa

KUVA 16. Muuttujan määritys ja ohjelma IL:ssä (17, s. 114)

4.1.2 Structured Text (ST)

Structured Text -kieltä kutsutaan korkeamman tason ohjelmointikieleksi, koska se ei käytä pelkästään matalan tason koneisiin keskittyviä operaattoreita, vaan se tarjoaa suuren valikoiman lauseita,

jotka pystyvät ilmaisemaan monimutkaisia toiminnallisuuksia hyvinkin pakatulla tavalla. ST:n etuja verrattuna IL:iin ovat hyvin pakattu muotoilu koodin rakenteessa, selkeä rakenne lauselohkoissa ja tehokkaat rakenteet ohjaamaan komentoja. Haittapuolena ST:ssä on hankalampi pääsy suoraan konekoodiin, koska ST:n koodi täytyy käydä läpi kääntäjän kautta. Kääntäjän tarkoitus on kääntää ihmisen luettava koodi koneen luettavaksi. Koodin ajaminen on raskaampaa ST muodossa ja tämä vaatii enemmän suoritustehoa laitteelta, missä koodia ajetaan. (17, s.116.) ST:n algoritmi on jaettu useisiin eri vaiheisiin, joita kutsutaan lauseiksi. Lauseita käytetään ST:ssä laskentaan, arvojen kirjoittamiseen, ohjauksiin ja komentovirran kontrollointiin (kuva 17.) ST-kieli on verrattavissa PASCAL- ja C-kieliin tietokonemaailmassa. ST:n ohjelmarakenne koostuu monista lausunnoista, jotka erotellaan toisistaan puolipilkun avulla ohjelmarakenteessa toisin kuin IL:ssä, jossa siirtyminen seuraavalle riville tarkoittaa seuraavaan toimintoon siirtymistä. ST:ssä lausuntoja voidaan venyttää monen rivin pituiseksi. Kommentointi tapahtuu ST:ssä samalla tavalla kuin IL:ssä suluilla ja tähtimerkinnöillä, joiden väliin käsky kirjoitetaan. (17, s.116.)

Keyword	Description	Example	Explanation
:=	Assignment	d := 10;	Assignment of a calculated value on the right to the identifier on the left.
	Call of an FB [a]	FBName(Par1:=10, Par2:=20, Par3:=>Res);	Call of another POU of type FB including its parameters. ":=" for input parameters, "=>" for output parameters
RETURN	Return	RETURN;	Leave the current POU and return to the calling POU
IF	Selection	IF d < e THEN f:=1; ELSIF d=e THEN f:=2; ELSE f:= 3; END_IF;	Selection of alternatives by means of Boolean expressions.
CASE	Multi-selection	CASE f OF 1: g:=11; 2: g:=12; ELSE g:=FunName(); END_CASE;	Selection of a statement block, depending on the value of the expression "f".

FOR	Iteration (1)	FOR h:=1 TO 10 BY 2 DO
		f[h/2] := h;
		END_FOR;
			Multiple loop of a statement block with a start and end condition and an increment value.

WHILE	Iteration (2)	WHILE m > 1 DO
		n := n / 2;
		END_WHILE;
			Multiple loop of a statement block with end condition at the beginning.

REPEAT	Iteration (3)	REPEAT
		i := i*j;
		UNTIL i < 10000
		END_REPEAT;
			Multiple loop of a statement block with end condition at the end.

EXIT	End of loop	EXIT;	Premature termination of an iteration statement
;	Dummy statement	; ;	

KUVA 17. ST-kielen lauserakenteita (17, s.117)

ST-kieleen ei erikseen sisälly käskyjä, joilla voitaisiin siirtyä koodissa eri paikkoihin, vaan kaikki mahdolliset siirtymät voidaan ohjelmoida esimerkiksi IF-rakenteen kautta. PLC:n kannalta tämä on hyödyllistä, koska käyttäjän määrittäessä oikeat ehdot IF-lausekkeeseen voidaan välttää turhia kiertoja PLC:n prosessointiyksiköltä. Vikatilanteen tullessa koodiin määritetty ehto siirtää ohjelman vikatilaan suoraan ilman, että sen tarvitsee käydä koko koodirivin kierto loppuun. (17, s.117.)

Funktioihin ja niistä koostuviin kokonaisuuksiin viittaaminen tapahtuu suluissa olevalla argumenttiluettelolla riippuen siitä, onko funktiolla parametrejä. Parametrille arvoa annettaessa se määritetään := merkillä, kun taas lähtevälle parametrille annetaan arvo käyttäen merkintää =>. Parametrien

määritysjärjestyksellä ei ole väliä, koska jätettäessä parametrit pois funktio käyttää oletusalkuarvoa kyseiselle tietotyypille tai määrittää sen edellisen kierron perusteella uudestaan samaksi. (17, s.123.) Myös oikean parametrin kutsuminen suoraan on mahdollista ST:ssä ja IL:n tavoin oikeat arvot välitetään ohjelmassa eteenpäin pilkuin erotettuna (kuva 18).

```
FUNCTION_BLOCK FbType
  VAR_INPUT
    VarIn: INT, VarH: INT := 1;
  END_VAR
  VAR_OUTPUT VarOut: INT := 5;
  END_VAR

  IF (VarIn > VarH) THEN
    VarOut := 10;
  END_IF;
END_FUNCTION_BLOCK
```

```
Call of FB FbName:
...
VAR FbName: FbType; RES: INT;
END_VAR

FbName();
(* FbName.VarOut == 5 *)
FbName(VarIn := 3, VarOut=>Res);
(* Alternative call: FbName(3, 1, Res *);
(* FbName.VarOut == 10,
copied to RES *)
...
```

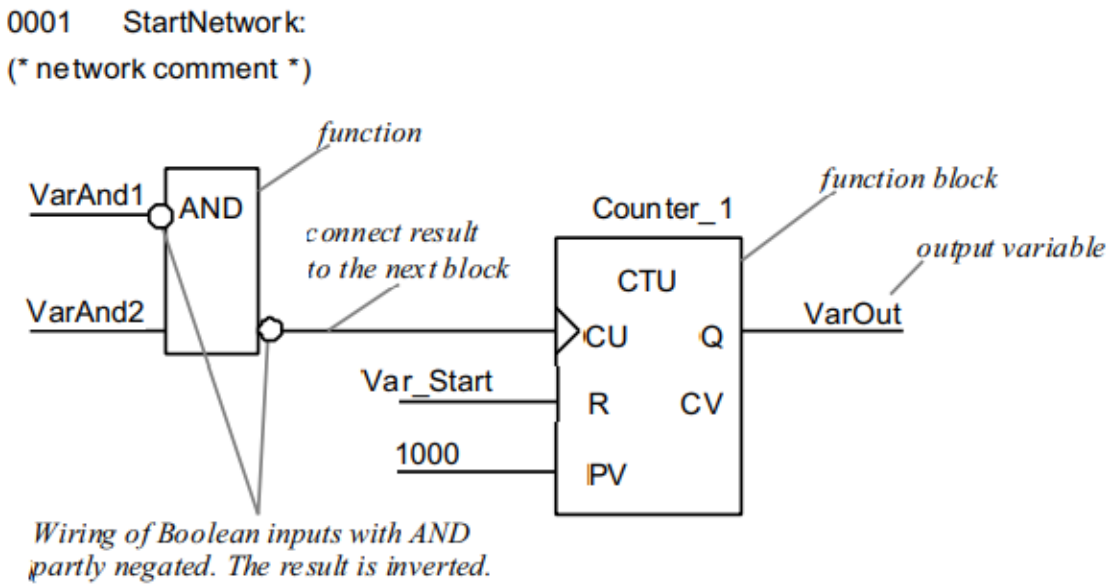
KUVA 18. Vasemmalla toimilohko nimeltään FbType ja oikealla toimilohkon kutsunta ohjelmassa (17, s.123)

4.1.3 Function Block Diagram (FBD)

Function block diagram (FBD) tulee alun perin kenttäprosessoinnin puolelta, jossa kokonais- ja liukulukusignaalien käsittely on tärkeää. FBD on siis kehittynyt kenttäprosessointiin erikoistuneesta kielestä vakiintuneeksi kieleksi PLC-ohjelmointiin. FBD esitetään kolmessa osassa, joihin kuuluu esitys-, ilmoitus- ja koodiosa. Esittelyosaan kuuluu aloitus ja lopetus samalla tavalla kuin tekstuaalisissa kielissä, mikä tarkoittaa POU:n alku- ja loppuosaa. Ilmoitusosa voi olla tekstuaalinen tai graafinen ja moni ohjelma tarjoaa kummallekin tuen. Koodiosa on jaettu erilaisiin verkkoihin, jotka auttavat tekemään koodin rakenteesta selkeämmän. Jokainen verkko tai verkkojen joukko määritellään joko aakkosnumeerisen tunnisteen avulla tai allekirjoittamattoman desimaaliluvun avulla. Yleensä ohjelmointijärjestelmä luo verkoille numerojärjestyksessä jonkin luvun määrittämään sen sijaintia koodissa. (17, s.134.) Verkon jakaminen ja nimeäminen on tärkeää virheiden kannalta, koska ohjelma osaa kertoa käyttäjälle vian sijainnin samalla tavalla kuin tekstuaalisessa kielessä. Virhe tulee näkyviin ohjelmassa tietylle riville.

FBD sisältää graafisia elementtejä, jotka ovat suorakulmaisia laatikoita. Graafiset elementit asetellaan kuvaamaan ohjauksen suuntaisia virtauslausekkeita, jotka ehtojen mukaan päästävät viestejä

läpi. Graafisiin elementteihin voi halutessaan jättää avoimia paikkoja ja se ei haittaa ohjelman toimintaa. Graafiset elementit tuovat myös mahdollisuuden kääntää elementtiin syötettyjä arvoja päinvastoin. Käännettyjä arvoja kuvataan graafisesti pallolla. (Kuva 19.)



KUVA 19. FBD-verkon elementit (17, s.137)

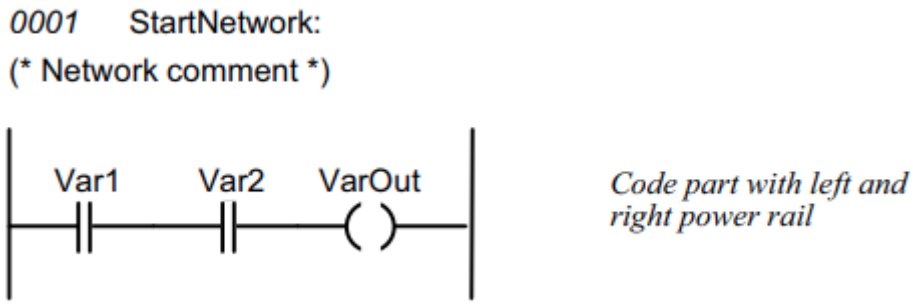
Graafisiin elementteihin FBD:ssä liittyvät myös kytkennät objektien välillä ja ne ovat aina pysty- tai vaakaviivoja. FBD ei ota vastaan minkäänlaisia kytkentöjä, joissa sisääntulo tai ulostulo on kytketty samaan kytkentäpisteeseen. Rajoitus sisääntulon ja ulostulon kytkennälle on tehty vähentämään epäselvyyksiä ohjelmarakenteessa. Yhteyden jaolla objektien välillä voi helposti jakaa samanlaista tuloarvoa monelle eri objektille, jotka tarvitsevat sitä. (Kuva 20.)

Graphical object	Name	Explanation	Ex. [a]
	Horizontal connection	The horizontal connection copies the value written on the left-hand side to the right-hand side.	all
	Vertical connection with horizontal connections.	The horizontal connection copies the value written on the left-hand side to all places on the right-hand side (splitting).	[0004]
	forbidden connection ("wired OR")		

KUVA 2. Erilaiset kytkentämahdollisuudet FBD:ssa (17, s.139)

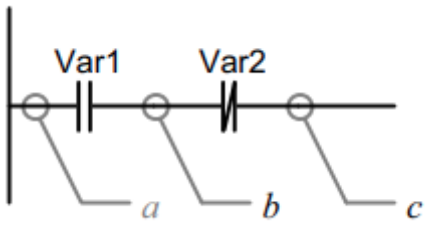
4.1.4 Ladder Diagram (LD)

Ladder diagram (LD) on ideologialtaan samanlainen kuin FBD, mutta yksinkertaisemmassa muodossa. LD:tä tyypillisesti käytetään sähkömekaanisten relejärjestelmien ohjaamiseen ja se kertoo virran liikkeen verkon läpi POU:lle vasemmalta oikealle. LD on suunniteltu kuvaamaan ja ohjaamaan Boolean-signaaleja. Boolean-signaaleissa numero 1 tarkoittaa, että jokin arvo on tosi ja numerolla 0 se on epätosi. FBD:n tavoin LD käyttää verkkoja erottelemaan toiminnallisuuksia ja ohjelma käsitellään normaalissa tilanteessa ylhäältä alaspäin, ellei käyttäjä ole määrittänyt muita ehtoja. (17, s.147.) Verkot kytketään LD:ssa kuvitteelliseen virtakiskoon, joka sijaitsee ohjelmarakenteessa ohjelman vasemmassa laidassa. Vasemmalta puolelta ohjelmalle tulee Boolean-arvoja 1 kokoaikaisesti ja ohjelmaa ohjataan erilaisilla ehdoilla, jotka päästää arvoa 1 läpi käyttäjän määrittelemään aikaan. (Kuva 21.)



KUVA 21. LD kielen rakenne (17, s.148)

LD:ssa verkkojen toiminta perustuu graafiseen asetteluun ja muuttujien liitäntöihin. Kytkennät tehdään muuttujien välillä sarjaan- tai rinnankytkennöillä (17, s.148). Ohjelmoinnissa tiedonkulku määritellään Boolean-arvoina ja graafisina elementteinä ovat kytkimet, jotka päästävät virran läpi tai eivät päästä sitä läpi. Avoin kytkin päästää graafisessa elementissä virran läpi ja suljettu ei päästä virtaa läpi. (Kuva 22.)



KUVA 3. LD Avoin kytkin Var1 ja suljettu kytkin Var2 (17, s.149)

Kytkimet, jotka ovat avoimia tai suljettuja, määrittävät ehdot tiedon liikkumiselle verkossa ja ne ovat niille asetettujen tilojen perusteella auki tai kiinni. Itse ehto-osia ei muuteta ohjelmassa, vaan ne säilyvät samanlaisina. Muuttujan arvot määritetään ohjelmassa keloilla ja muuttujan nimi kirjoitetaan yleisesti kelan yläpuolelle ohjelmaan. Myös keloilla on LD:ssä Boolean-muuttujatyyppi, joten datatyypit eivät muutu tiedonsiirrossa. (Kuva 23.)

Graphical object	Name	Explanation	Ex. [a]
VarName —( )—	Coil	Variable [b] := LeftLink. (Copy the value of the left link to the variable).	[0007]
VarName —(/)—	Negated coil	Variable [b] := NOT LeftLink. (Copy the negated value (change FALSE <-> TRUE) of the left link to the variable).	

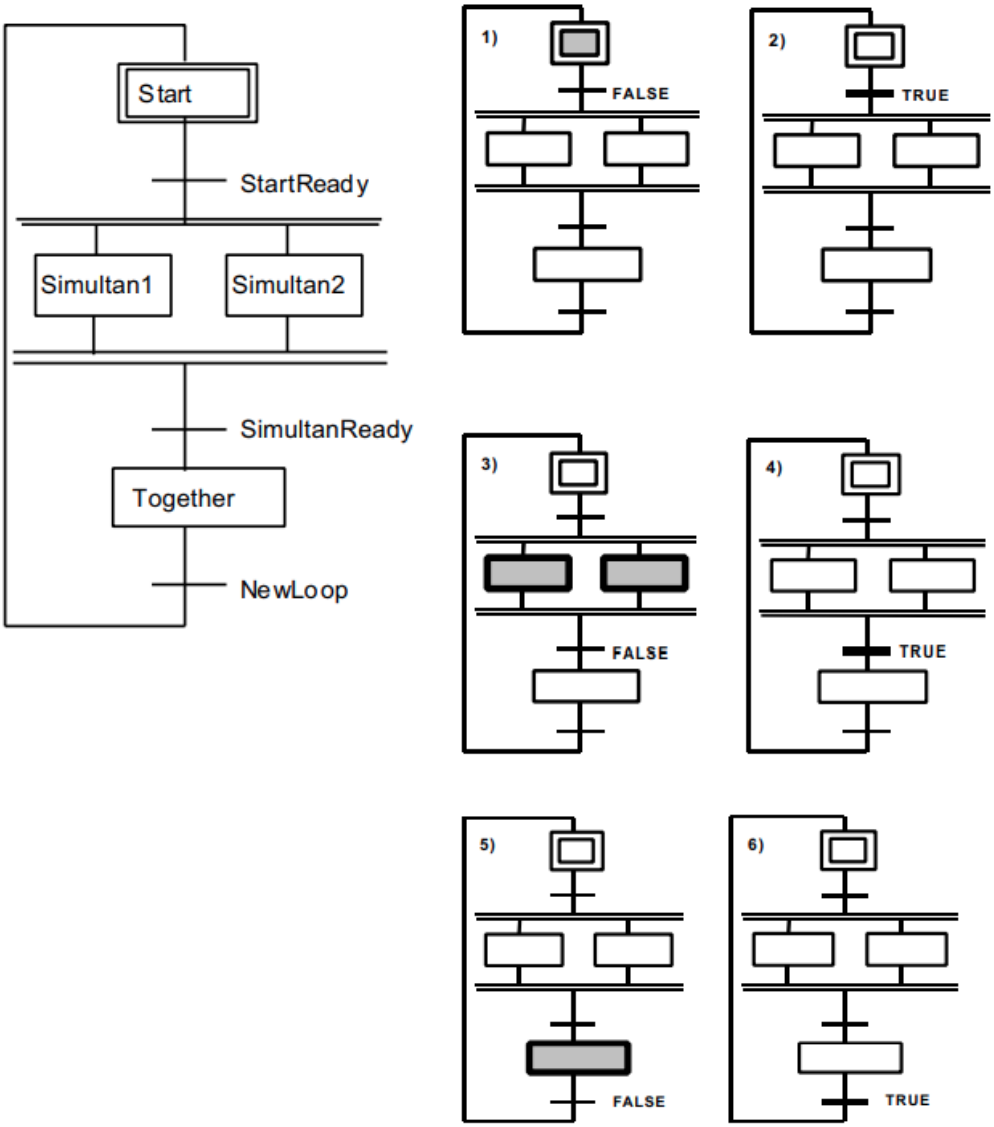
KUVA 23. Kelan graafinen rakenne LD:ssa (17, s.152)

4.1.5 Sequential Funtion Chart (SFC)

Sequential Function Chart (SFC) kehitettiin monimutkaisten ohjelmien hajottamiseksi pienempiin palasiin ja kuvaamaan ohjelman ohjausta paremmin eri palasten välillä. Hyödyntäen SFC:n tarjoamia ominaisuuksia käyttäjän on mahdollista yhdistellä peräkkäisiä ja rinnakkaisia prosesseja yhteen kokonaisuuteen helposti. Prosessien suoritusjärjestys riippuu ohjelmaan määritetyistä staattisista ehdoista sekä sisään- ja ulostulojen dynaamisista ehdoista. Itse ohjelmarakenne voidaan tehdä millä tahansa muulla neljällä IEC 61131-3 -standardissa määritetyllä kielellä. SFC on ensisijaisesti graafinen kieli, mutta myös tekstuaalisen kielen käyttö onnistuu SFC:lla. Kummankin kielen tuki helpottaa tiedon siirtoa eri järjestelmien välillä ja auttaa yksinkertaistamaan ohjelmien varmuuskopiointia. SFC:n vahvuuksien hyödyntämiseen graafinen esitystapa on selkeämpi, koska se havainnollistaa eri vaiheiden riippuvuuksia ja riippumattomuuksia askelten välillä paremmin. (17, s.169.)

Muiden graafisien kielien tavoin SFC rakentuu verkoilla, joiden sisällä on yksi tai useampi verkko toiminnallisuutena. Verkon eri osioita kutsutaan vaiheiksi tai siirtymiksi ja vaiheen täytyy olla joko aktiivinen tai ei-aktiivinen, kun SFC-ohjelmaa suoritetaan. (Kuva 24.) Vaiheen ollessa aktiivinen siihen kuuluvia toimintoja suoritetaan, kunnes vaihe muuttuu passiiviseksi. Päätöksen vaiheen tilan muuttumisesta määrittää siirtymä, joka on graafisessa muotoilussa heti vaiheen alla ohjelmassa.

Siirtymä on Boolean tyyppinen ehto, joka muuttaa edellisen vaiheen passiiviseksi tullessaan todeksi eli arvoksi 1. (17, s.170.)



KUVA 24. SFC eteneminen ohjelmassa (17, s.171)

4.2 Olio-ohjelmoinnin periaatteet

Olio-ohjelmointi luokitellaan korkeamman tason ohjelmointikieliin ja sen periaatteita käytetään apuna monissa ohjelmointikielissä. Olio-ohjelmoinnin ideologia kehitettiin ensimmäisen kerran kiellellä nimeltä Simula, joka valmistui 1960-luvun puolivälissä Norjalaisen Ole-Johan Dahlin ja Kristem Nygaardin toimesta. (18.) Tietokoneteknologian kehittyessä tehokkaammaksi muutkin kielet, kuten

C-kieleen pohjautuva Objective-C, kehitettiin C++ nimiseksi kieleksi. C++ kieleen on lisätty ominaisuuksia mitä normaali C-kieli ei välttämättä tarjoa (19). Logiikkaohjelmoinnin kannalta on tärkeä ymmärtää, että C-kieli on lähimpänä ''konekieltä'' eli binääristä kieltä. Binäärinen kieli koostuu vain symboleista 1 ja 0, joiden avulla kone osaa toimia. Monet toiminnot logiikan ja sille määritetyn ohjelman toiminnassa ovat ennen olleet Boolean tyyppisten tietojen käsittelyä eli tilatietojen 1 tai 0 käyttämistä. Myös ohjelmarakenne on koodirivillä ylhäältä ilman poikkeuksia ja ongelma isoissa sovelluksissa on voinut olla samaan fyysiseen paikkaan kirjoittaminen väärässä kohdassa.

Olio-ohjelmointi koostuu pääasiassa neljästä säännöstä ja ominaisuudesta, jotka ovat kapselointi, abstraktio, perinnöllisyys ja polymorfismi. Normaaliin terminologiaan olio-ohjelmoinnin yhteyteen täytyy tietää komponentit objekti, luokka, menetelmä ja ominaisuus, jotka rakentavat pohjan neljälle säännölle. Luokka on malli tai standardi jollekin toiminnolle tai joukolle siitä, miten objekti toimii.

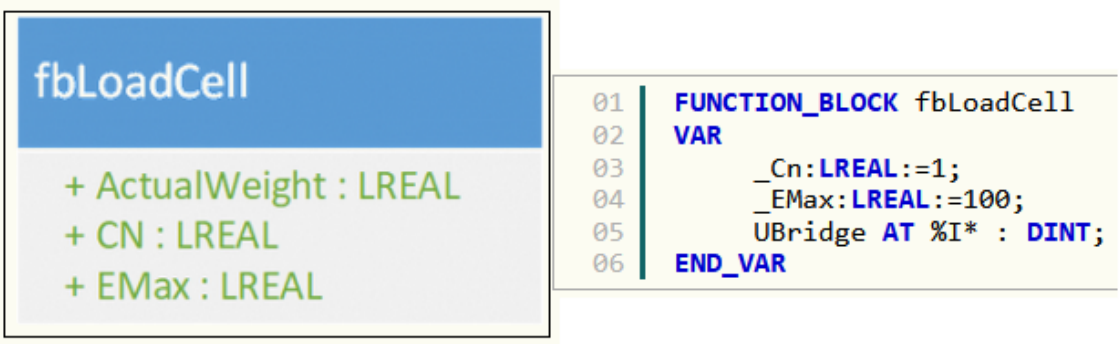
Objekti on ilmentymä luokasta ja objektit ovat työskenteleviä osioita luokasta. Menetelmä voi muokata luokan tilaa ja se pätee jokaiseen luokan instanssiin. Instanssit ovat kuin objekteja, mutta ne laskevat montako kertaa jokin asia tapahtuu määritellyssä luokassa kopiomäärästä riippumatta. (20.) Abstraktion tarkoitus on vähentää näkyvää dataa käyttäjälle ja sen ominaisuudet ovat kapseloinnin kanssa samankaltaisia. PLC-ohjelmoinnissa tieto on jaoteltu valmiiksi toimilohkoihin, joten abstraktiota ei logiikkaohjelmoinnin kannalta ole niin tärkeää käsitellä.

Olio-ohjelmointi tuo logiikkaohjelmointiin muiden kielien käyttämiä hyviä ominaisuuksia, mutta se edellyttää, että käyttäjä siirtyy pääasiassa ST-muodossa kirjoittamiseen. Olioiden tyylisiä tapoja on ollut käytössä jo pitkään logiikkaohjelmoinnissa esimerkiksi toimilohkojen muodossa. Ohjelmointiympäristön kehittäjät ovat lisänneet valmiita toimintoja sovelluksiin, jotka on todettu hyviksi ja toimiviksi palvelun tarjoajien päässä. Tätä kautta käyttäjä voi luottaa toimilohkojen toiminnallisuuksiin ja voi ohjelmaa korjatessaan keskittyä muiden osa-alueiden tarkastamiseen.

4.2.1 Kapselointi

Kapselointi kerää käytettävät ominaisuudet ja menetelmät yhteen objektiin, joka on tyypillisesti toimilohko. Kapseloinnilla referoidaan funktion ominaisuuksia piilottaa tietoa ja ominaisuuksia, joita ei haluta käyttäjän saataville tai ne eivät ole hyödyllisiä käyttäjälle. Kokonaisuutena tämä tarkoittaa

toimilohkojen erotusta käyttöliittymän ja sen toteutuksen välillä. Kapseloinnin hyötyjä ovat tarkat määritykset, kuka pääsee tutkimaan toimilohkon tietoja. Se auttaa suojaamaan tietoja vahingoittumasta ja pitämään koodin puhtaana. (21.) TwinCAT3:ssa käyttäjä pystyy käyttämään normaalia toimilohkoa rakentamaan objektin samalla tavalla kuin vaikkapa C#-kielessä, jossa puhutaan luokista. Ominaisuuksien ja menetelmien avulla toimilohkoon voidaan tehdä sisääntuloporttien kaltaisia toimintoja, joilla päästään käyttämään toimilohkon sisäisiä toimintoja (kuva 25.)

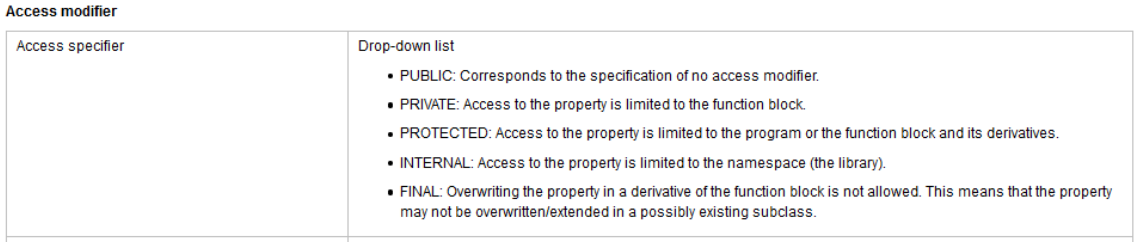


KUVA 25. C# luokka ja TwinCAT3 toimilohko (21)

Piilotettua informaatiota käsitellään kapseloinnissa Getter- ja Setter-toiminnoilla, jotka ovat muiden toimilohkojen tai luokkien luettavissa (22). TwinCAT3 käyttää näistä kahdesta toimintoja, jotka on lyhennetty Get ja Set (23). Get-menetelmää käytetään palauttamaan tietty arvo luokan tai toimilohkon sisällä ja Set-menetelmää käytetään asettamaan tai päivittämään tietty arvo muuttujalle (24). Ohjelmoijat voivat manipuloida toimilohkojen ja luokkien näkyvyyttä pääsynmuokkaajilla, joita on 5 kappaletta TwinCAT3:ssa (kuva 26).

Pääsynmuokkaajat ovat

- Julkinen: päästää kaiken läpi
- Yksityinen: käyttö on suljettu toimilohkoon
- Suojattu: pääsy rajoitettu toimilohkoon ja sen johdannaisiin
- Sisäinen: pääsy rajoitettu kirjastoon
- Lopullinen: toimilohkoa ei saa korvata eikä laajentaa alaluokissa.

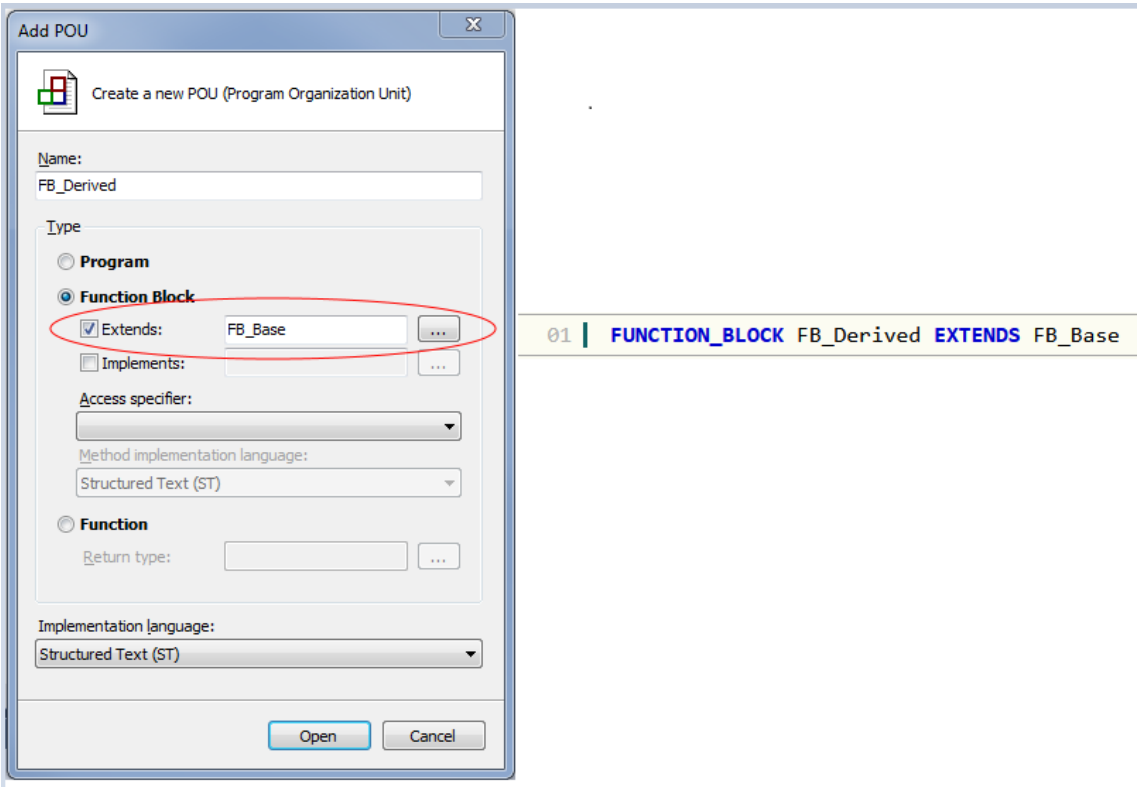


KUVA 26. Pääsynmuokkaajat TwinCAT3:ssa (24)

4.2.2 Perinnöllisyys

Perinnöllisyys määritellään mekanismiksi, jolla toimilohkon perustoiminnallisuudesta saadaan tehtyä laajennettava toiminto. Toimilohko laajennetaan sen isäntälohkosta ja se perii kaikki samat ominaisuudet kuin isäntälohko. Periytymiseen sisältyvät siis kaikki menetelmät, ominaisuudet ja paikalliset muuttujat, joita voidaan laajentaa laajennettuun lohkoon sen omilla elementeillä. Laajennettu toimilohko voi käyttää ominaisuuksia, jotka ovat näkyvillä julkisesti, suojatusti tai sisäisesti toimilohkolle. TwinCAT3:ssa paikalliset muuttujat ovat kuitenkin aina käytettävissä johdetulle toimilohkolle. (25.)

Perinnöllisyyden tärkeimmät ominaisuudet ovat jo luotujen toiminnallisuuksien käyttäminen ilman saman uudelleen kirjoittamista ja toimivien kokonaisuuksien laajentamisessa käyttäjän ei tarvitse muuttaa alkuperäistä toimilohkoa, mikä on hyödyllistä ulkopuolisten viittausten tullessa mukaan ohjelmointiin. Toimilohkoa laajentaessa sen menetelmät ja toiminnallisuus voidaan ylikirjoittaa ilman, että käyttäjä määrittää alkuperäiseen toimilohkoon mitään muutoksia tai lupia. (25.) TwinCAT3:ssa funktioiden laajentaminen tehdään luodessa toimilohkoa tai kirjoittamalla suoraan ST-muodossa laajennus EXTENDS-komennolla (kuva 27). Perinnöllisyys siis laajentaa käyttäjälle mahdollisuudet käyttää uudelleen luotuja toiminnallisuuksia ilman, että käyttäjän tarvitsee huolehtia koodin muuttujien päällekkäisyyksistä tai muuttujien ylikirjoituksesta, jos annettuja työkaluja käytetään oikein.



KUVA 27. Toimilohkon laajennus TwinCAT3:ssa (25)

4.2.3 Polymorfismi

Polymorifsmi-termi tulee kreikkalaisesta sanasta ”monia muotoja” ja ohjelmoinnin kannalta se tarkoittaa yhtä tyyppiä tai toiminnallisuutta, joka voi ottaa monia eri muotoja. Esimerkiksi PLC-ohjelmoinnissa voidaan ohjata kolmea ovea, jotka kaikki toimivat erilaisilla tekniikoilla. Ovet voivat toimia pneumaattisesti, sähköisesti ja hydraulisesti ja niiden kaikkien toimilohkot ovat erilaiset toiminnallisuuksiltaan. Kaikilla toimilohkoilla tulee olemaan kuitenkin saman tyyppiset menetelmät ovien avaamiseksi ja sulkemiseksi. Polymorfismi antaa käyttäjälle mahdollisuuden luoda kolmesta ovityypistä yleisen tyypin ja käyttää tätä yleistä tyyppiä ohjelmoidessa piittaamatta siitä, millä tavalla ovet aukeavat. (26.)

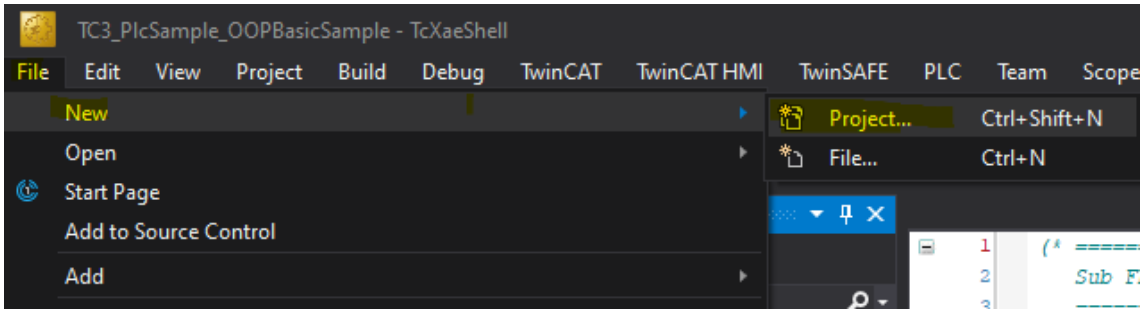
Sekä polymorfismiin että perinnöllisyyteen liittyy kaksi osoitinta, jotka ovat tärkeä ottaa huomioon kumpaakin ominaisuutta käytettäessä. SUPER- ja THIS-osoittimet ovat TwinCAT3:ssa ja muissakin ohjelmointikielissä automaattisesti käytettävissä jokaisessa toimilohkossa tai luokassa. THIS-osoitin viittaa aina omaan itseensä ja SUPER-osoitin viittaa toimilohkon ilmentymään, josta kyseinen toimilohko luotiin. Polymorfismilla tarkoitetaan siis yksittäisien tyyppien tai toimilohkojen kykyä ottaa monia muotoja ja perinnöllisyyden tavoin polymorfismin toiminnallisuuksia voidaan käyttää uudelleen laajentamiseen kuuluvien THIS- ja SUPER-osoittimien avulla. (26.)

5 TWINCAT3-OHJELMOINTI

Beckhoffin TwinCAT3-ohjelmointiympäristö rakentuu Microsoftin Visual Studion päälle ja sen rakennuksessa on käytetty myös CODESYS-ohjelmointiympäristön elementtejä apuna. Seuraavaksi tehdään esimerkkiohjelma käyttäen hyödyksi aikaisemmin työssä käsiteltyjä asioita. TwinCAT3 antaa myös käyttäjän luoda loputtoman määrän seitsemän päivän lisenssejä käyttäjälle ohjelmien ja komponenttien testaamiseen.

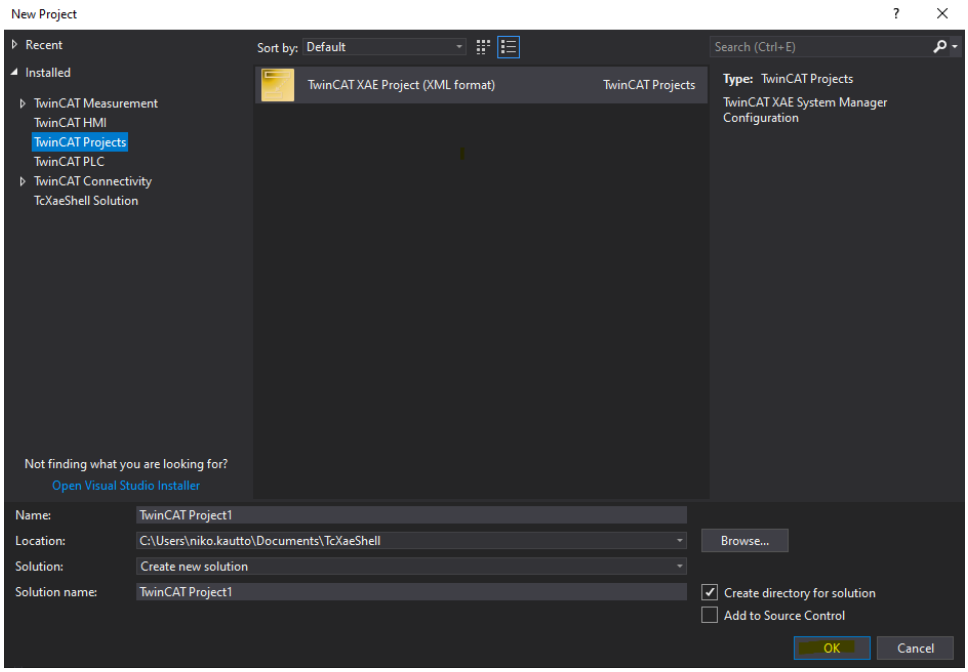
5.1 Projektin luominen

Käyttäjän asennettua TwinCAT3:n ja avattua ohjelman täytyy luoda uusi projekti tai käyttää olemassa olevaa projektia. Käytetään polkua File, New ja Project (kuva 28).



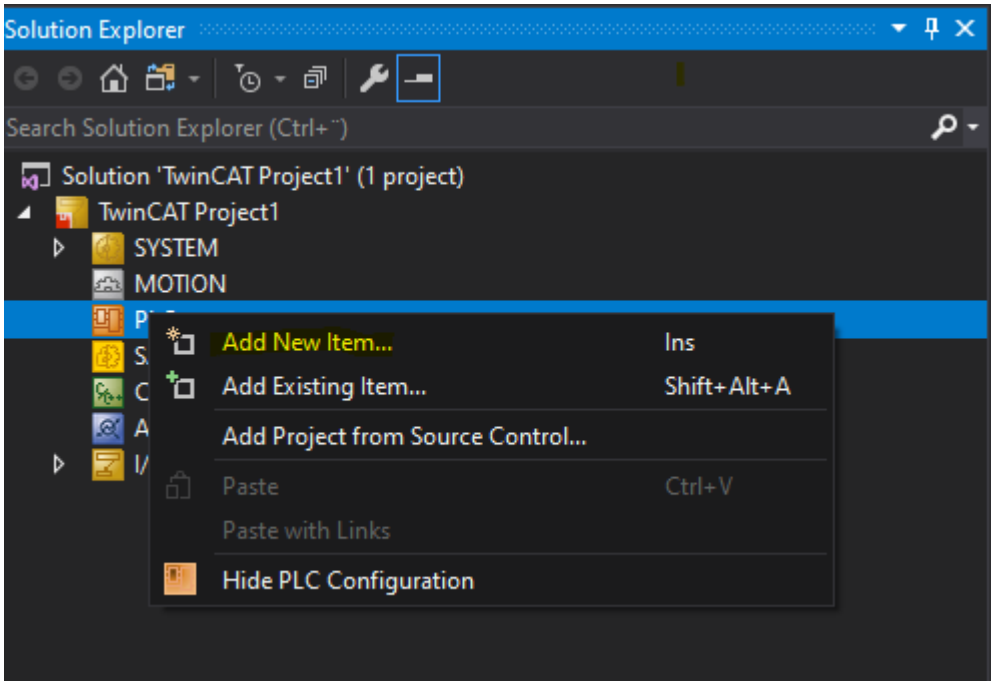
KUVA 28. Uusi TwinCAT3-projekti

Edellisten toimintojen jälkeen avautuu uusi ikkuna, jossa projekti luodaan. Käyttäjä voi määrittää itse haluamansa nimen projektille ja valita sen tiedostosijainnin (kuva 29).



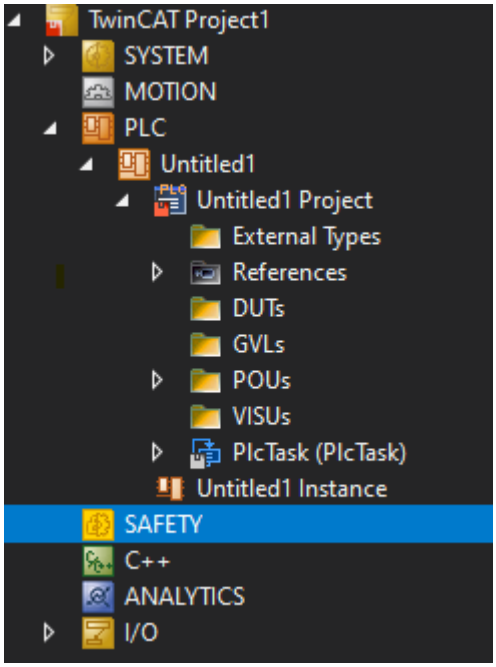
KUVA 29. Projektin tiedot

Projektin luomisen jälkeen käyttäjälle tulee näkyviin osioita, joita voi käyttää halutessaan projektissa. Kaikkiin osioihin voi lisätä sisältöä samalla periaatteella painamalla hiiren oikeaa näppäintä, jonka jälkeen vaiheet ovat samanlaisia kuin projektin luomisessa. Tässä tapauksessa otetaan uusi PLC-projektia varten (kuva 30.)



KUVA 30. Sisällön luominen projektiin

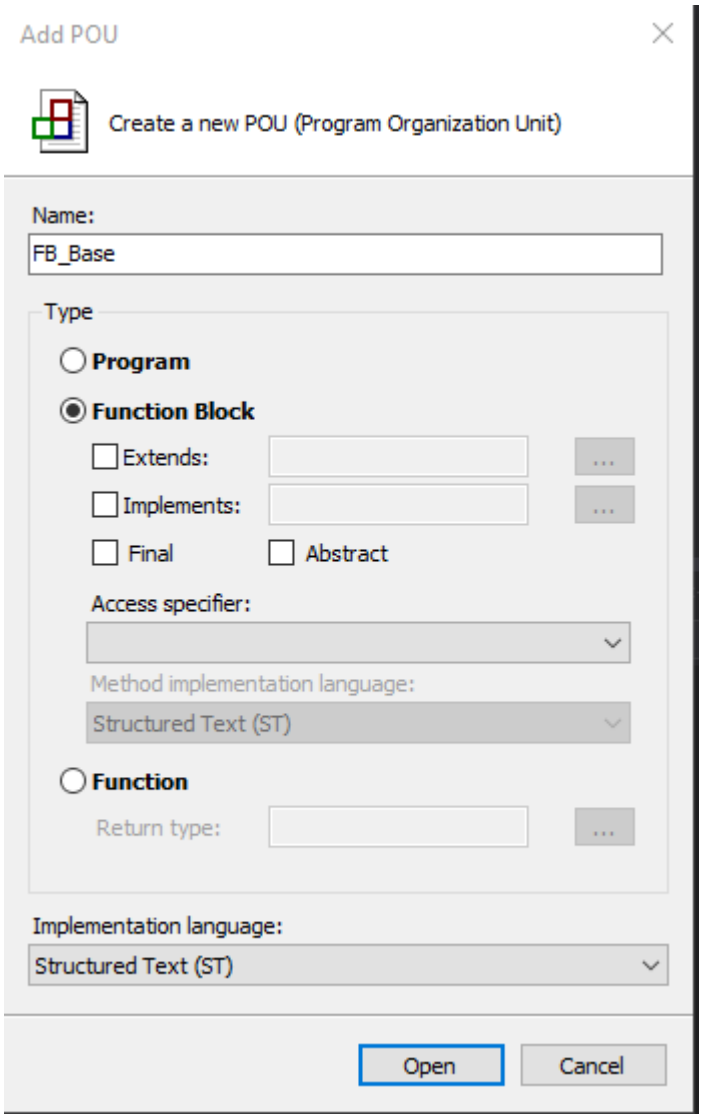
PLC:n luomisen jälkeen TwinCAT3 antaa käyttäjälle valmiiksi jaoteltuja osioita, mutta kansioita ja rakenteita voi lisätä sekä nimetä projektissa oman tarpeen mukaisesti (kuva 31).



KUVA 31. Lisätty PLC TwinCAT3-projektiin

5.2 Ohjelman luominen

Projektin luomisen tavoin ohjelmaan lisätään komponentteja painamalla haluttua kansiota hiiren oikealla painikkeella. Lisätään ohjelmaan uusi POU ja tehdään siitä toimilohko. Toimilohkoa tai muuta toiminnallisuutta lisätessä voi myös valita olio-ohjelmoinnin ominaisuuksia tai pääsynmuokkaajia, mutta niitä ei tarvita tehdessä ensimmäistä toiminnallisuutta. Kieli toimilohkon kirjoitukseen valitaan avatun sivun alalaidasta. (kuva 32.) On myös hyvä nimetä toiminnallisuuden tarkoitus sen nimeen. Toimilohkoa tehdessä on hyvä nimetä se alkukirjaimilla FB ja esimerkiksi ohjelmaa luodessa alkukirjaimet olisivat PRG.



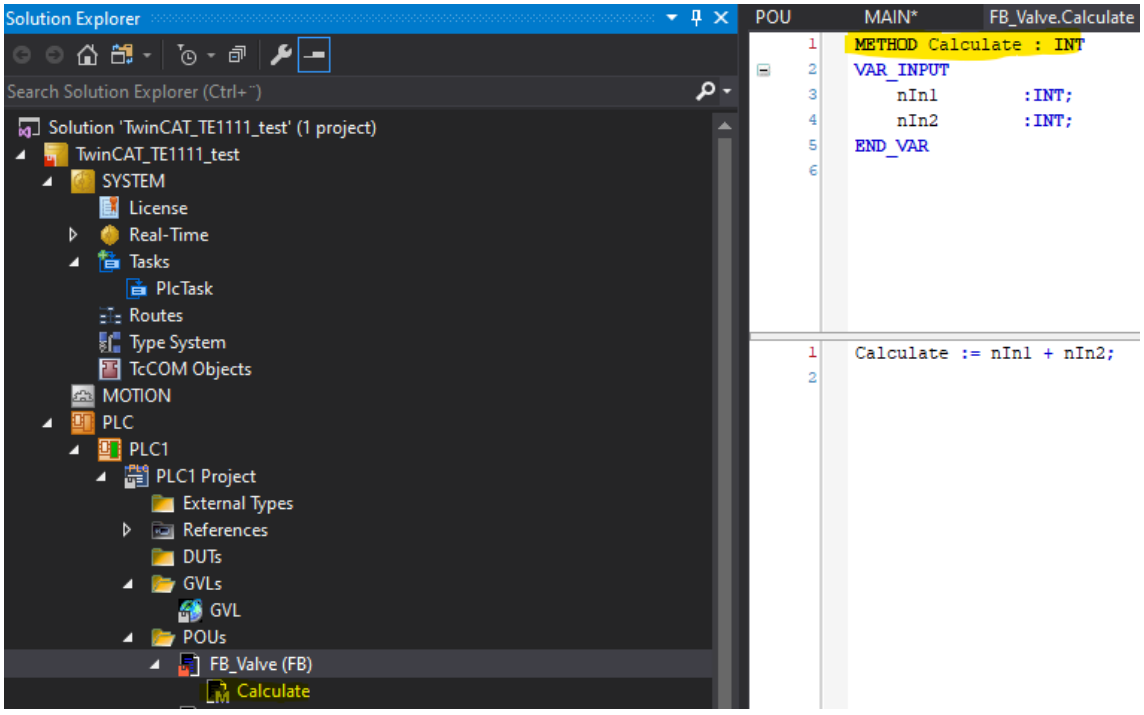
KUVA 32. POU:n luominen

POU:n luomisen jälkeen voidaan toimilohkolle määrittää sisään tulevia, uloslähteviä tai sisäisiä muuttujia riippuen käyttötarkoituksesta ja ellei kaikkia tarvita, toimilohkon määritykset voidaan jättää tyhjäksi tai poistaa. Esimerkissä määritetään toimilohkolle yksi muuttuja, joka määritellään fyysiseksi tuloksi %I* merkinnän avulla. (Kuva 33.)

```
1  FUNCTION_BLOCK FB_Valve
2  VAR_INPUT
3      bInput  AT %I*  :BOOL;
4  END_VAR
5
6  VAR
7  END_VAR
8
```

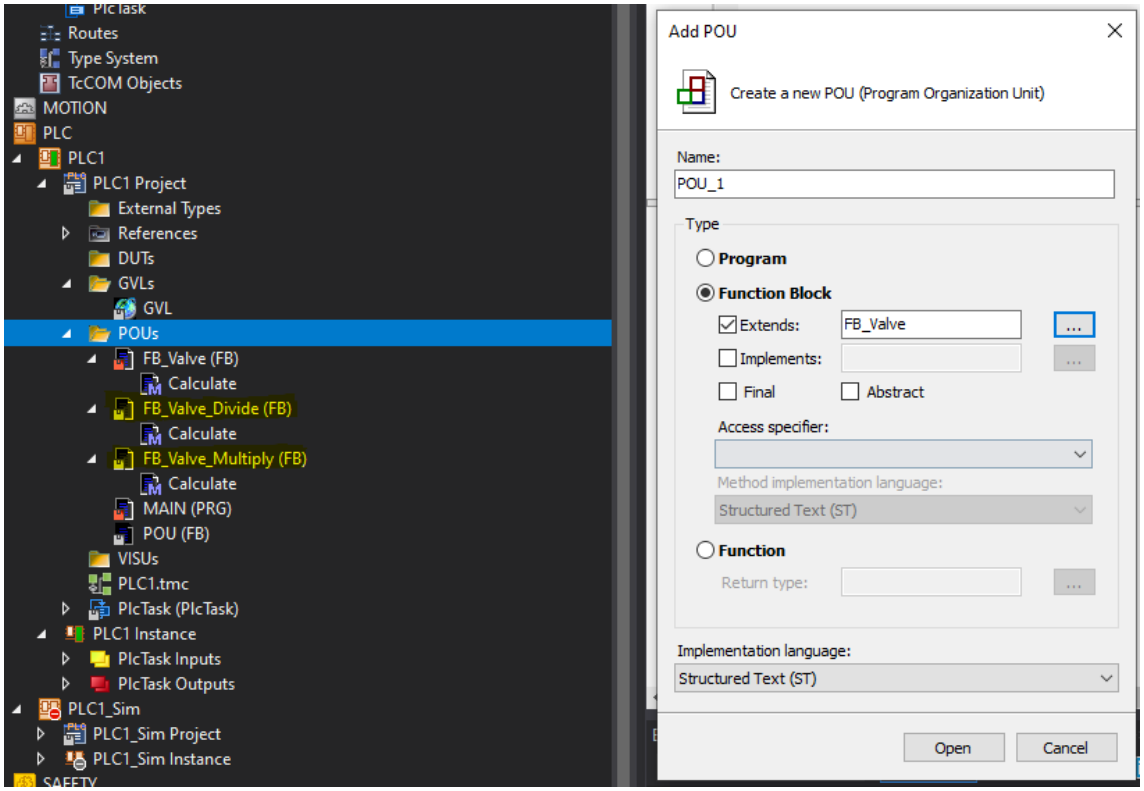
KUVA 33. Fyysisen tulon määritys toimilohkolle

Menetelmä toimilohkolle lisätään tässäkin tapauksessa samalla tavalla kuin aikaisemmatkin lisäykset hiiren oikeaa painiketta painamalla halutun toimilohkon päällä ja valitsemalla menetelmä. Menetelmälle voi määrittää muuttujatyypin ja pääsynmuokkaajan suoraan valikosta, mutta niitä voi muuttaa myös jälkikäteen kirjoittamalla tekstimuodossa komennot menetelmään. Esimerkissä määritetään menetelmän nimeksi Calculate ja lisätään sille muutama Integer-tyyppinen muuttuja. Calculate-lohkolle määritellään myös laskutoimitus (kuva 34.)



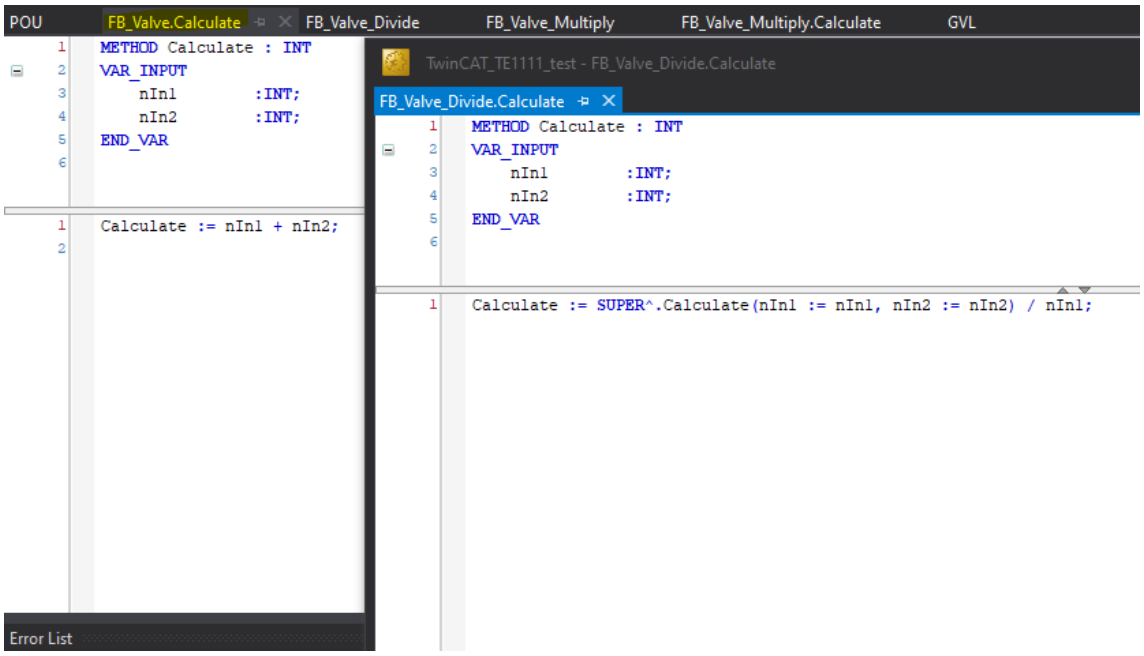
KUVA 34. Menetelmän lisääminen ja määrittely

Pohjana olevan toimilohkon määritykset voidaan laajentaa nyt muille toimilohkoille ja käyttää hyödyksi esimerkiksi valitsemalla Extends ja hakemalla halutun toimilohkon nimi. Esimerkissä tehdään kaksi toimilohkoa laajentamalla ne alkuperäisestä toimilohkosta. (Kuva 35.)



KUVA 35. Toimilohkon laajentaminen

Laajentamisen jälkeen jokainen laajennettu toimilohko säilyttää niiden isäntälohkon ominaisuudet ja sen muuttujan bInput-sisääntulon. Jokaiseen laajennettuun toimilohkoon voi kuitenkin lisätä ominaisuuksia ja menetelmien ei ole pakko olla samanlaisia kuin isäntälohkossa. Samalla nimellä oleva Calculate-toiminnallisuus ottaa muuttujat isännältä, mutta toiminnallisuus on ohjelmoijan päätettävissä. Esimerkiksi FB_Valve_Divide-toimilohkossa käytetään SUPER-osoittajan avulla isäntälohkon Calculate-menetelmää, joka jaetaan osoittajan jälkeen. (Kuva 36.)



KUVA 36. Menetelmien erilaisuus laajennuksissa

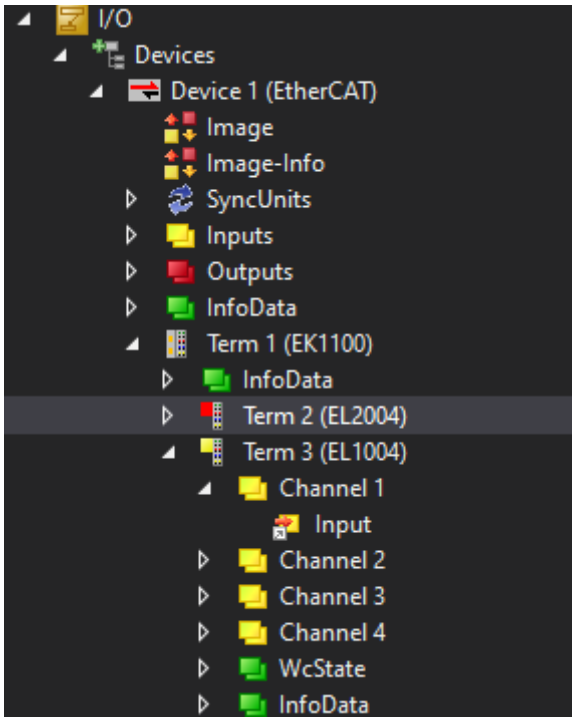
Jokaisen toimilohkon ominaisuuksia voidaan kutsua erikseen ja käyttää niitä niin monta kertaa kuin käyttäjä haluaa. Esimerkissä määritetään arvot nInput1- ja nInput2-muuttujille sekä kutsutaan tehtyjen toimilohkojen ominaisuuksia instanssien avulla. Ohjelmassa kutsutaan jokaisen toimilohkon Calculate-menetelmää ja määritellään menetelmän arvojen nIn1- ja nIn2-suuruudet. (Kuva 37.)

```
PROGRAM MAIN
VAR
//Muuttujat sisääntulo parametreille
    nInput1     :INT :=100;
    nInput2     :INT :=200;

//Instanssit toimilohkoista
    fbBase              : FB_Valve;
    fbSub1              : FB_Valve_Divide;
    fbSub2              : FB_Valve_Multiply;


    nReturn_Valve_base      :INT;
    nReturn_Valve_sub1      :INT;
    nReturn_Valve_sub2      :INT;


nReturn_Valve_base := fbBase.Calculate(nIn1 := nInput1, nIn2 := nInput2); //Yhteenlasku 100 + 200
nReturn_Valve_sub1 := fbSub1.Calculate(nIn1 := nInput1, nIn2 := nInput2); //Kertolasku  100 * 200
nReturn_Valve_sub2 := fbSub2.Calculate(nIn1 := nInput1, nIn2 := nInput2); //Jakolasku (100 + 200) / 200
```

KUVA 37. Muuttujien ja toimilohkojen kutsuminen pääohjelmassa

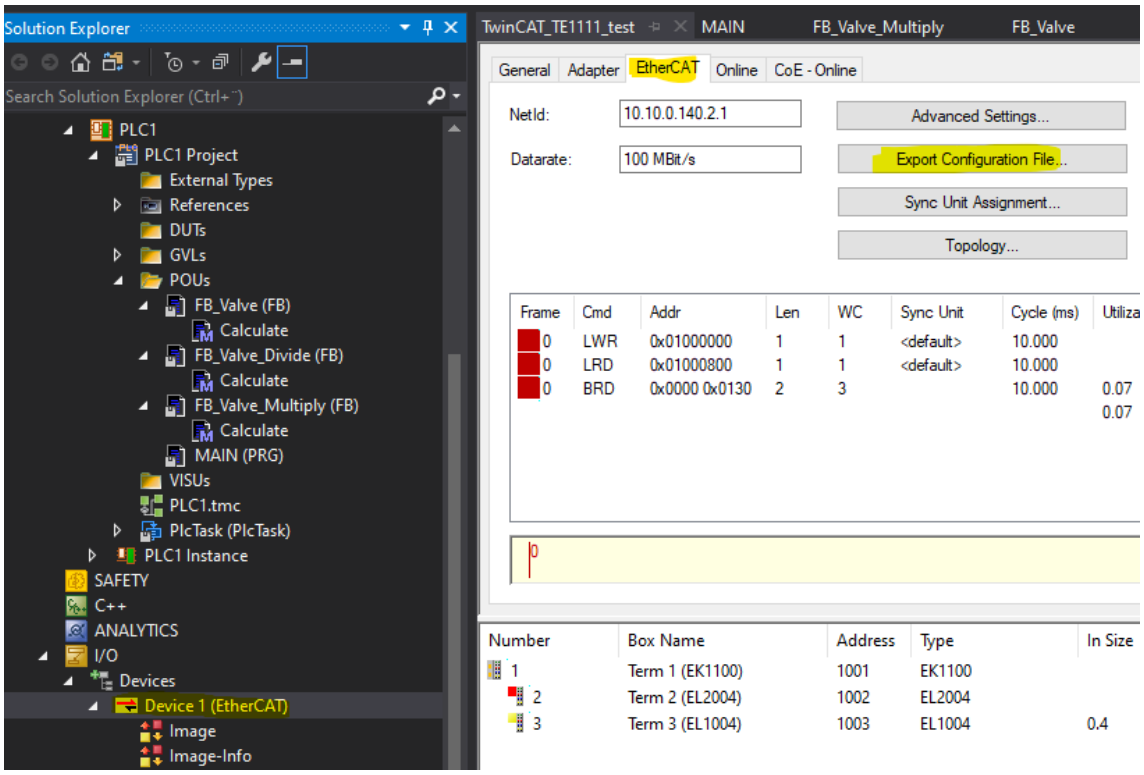
5.3 EtherCAT-isäntä ja sen simulaatio

Halutessa yhdistää luotuja muuttujia kuvitteelliseen tai fyysiseen laitteistoon täytyy valita sopivat toimilaitteet I/O ja Devices-osiosta TwinCAT3:ssa, jossa samaan tyyliin hiiren oikeaa painiketta painamalla löytyy lisäysvaihtoehtoja laitteistolle. Esimerkkiin on valittuna Device 1 EtherCAT-isäntä ja EK1100 EtherCAT-terminaali, johon on lisätty kortit EL2004 ja EL1004 sisään- ja ulostuloja varten. Punainen indikoi tässä tapauksessa ulostuloja ja keltainen sisääntuloja korteille. Jokaiseen kanavaan voi liittää muuttujan ja kanavan ollessa linkitettynä sitä indikoi pieni valkoinen nuoli kuten kanava 1:n alla näkyy. (Kuva 38.)



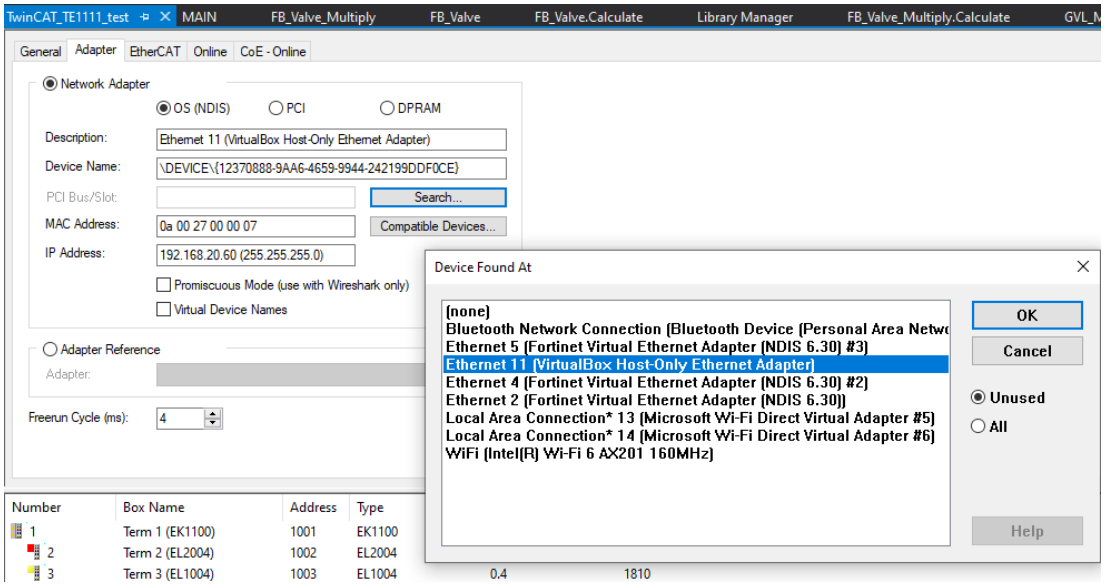
KUVA 38. I/O konfiguraation lisääminen

Simulaatiota varten tarvitaan tehdystä konfiguraatiosta tiedosto, jossa on identtinen kuva konfiguraatiosta. Tiedosto tehdään painamalla tehtyä EtherCAT-isäntälaitetta hiiren vasemmalla painikkeella, jonka jälkeen navigoidaan EtherCAT-välilehdelle ja painetaan Export Configuration File. (Kuva 39.)



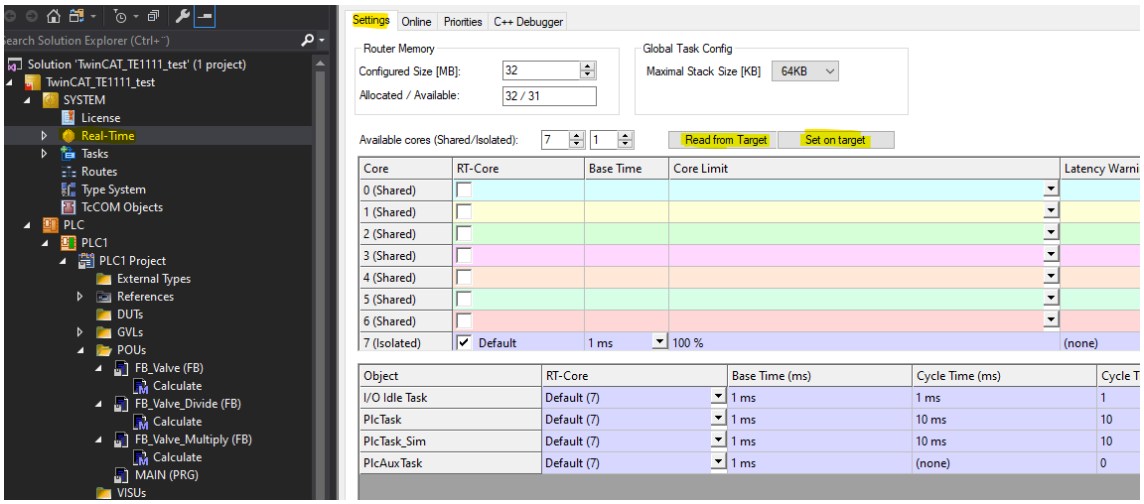
KUVA 39. Device 1:n konfiguraation tallentaminen

Simulaatiota voi ajaa Beckhoffin mukaan vain fyysisillä Ethernet-porteilla, mutta yhdistäminen onnistuu myös virtuaalikoneella, joka luo käyttäjän haluamat portit lokaalin koneen ja virtuaalikoneen välille. Haetaan haluttu portti ja liitetään se EtherCAT-isäntään, joka tulee lukemaan määritettyä porttia. (Kuva 40.) Käyttäjän täytyy itse määrittää portti oikealle IP-osoitealueelle lokaalilla sekä virtuaalikoneella, joka on esimerkissä sijoitettu alueelle 192.168.20.xxx.



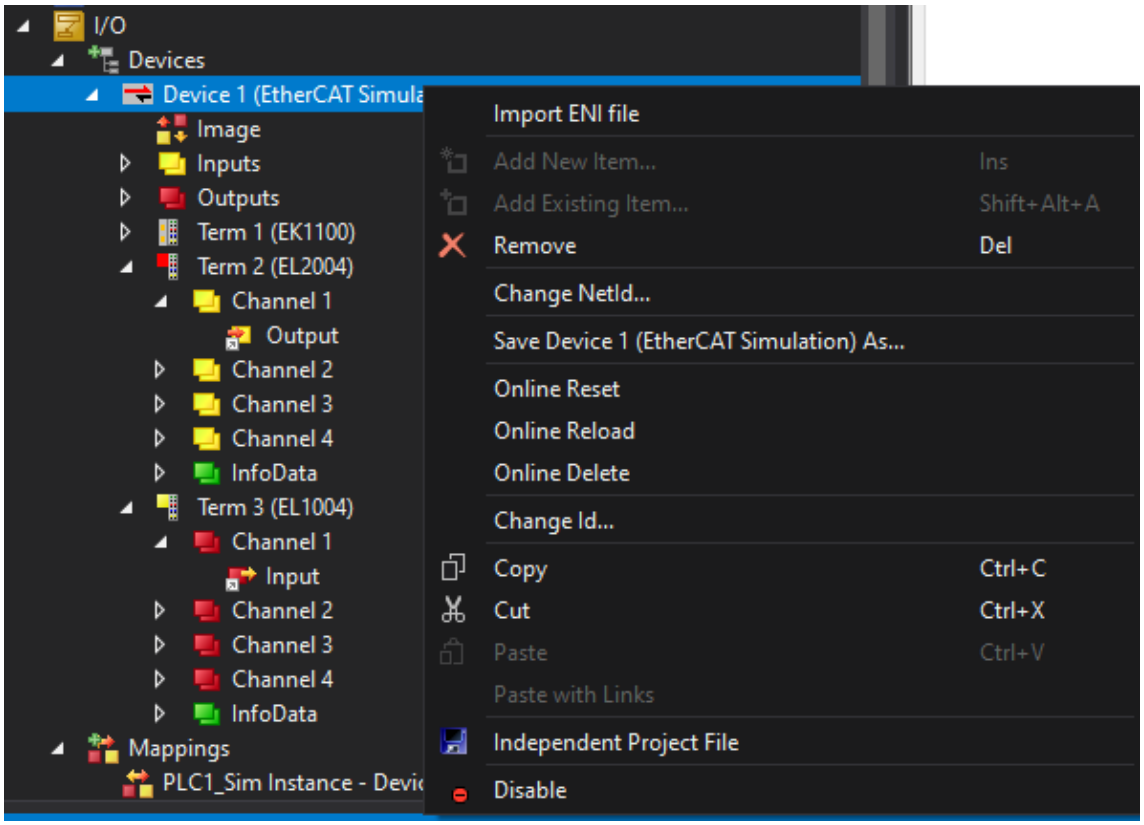
KUVA 40. Ethernet-portin määrittäminen EtherCAT-laitteeseen

Virtuaalikonetta käyttäessä täytyy huomioida se, että TwinCAT3 jakaa oletusasetuksena käytössä olevat prosessoriytimet Windowsin kanssa ja ne täytyy erottaa toisistaan. Ytimien erotus täytyy tehdä paikallisen PC:n sekä virtuaalikoneen puolella virtuaalikoneen toiminnan varmistamiseksi TwinCAT3:n kanssa. TwinCAT3:ssa mennään Real-Time-kohtaan ja Settings-välilehdellä, josta löytyy kohdat Read From Target ja Set on target. Read from Target hakee käyttäjälle suoraan koneen ytimien määrän ja Set on target -kohdassa käyttäjä voi erottaa halutun määrän ytimiä TwinCAT3:n käyttöön. (Kuva 41.) Kuvassa 41 erotetulle ytimelle on määritetty yhden millisekunnin aika, joten simulaattorin puolella aika täytyy puolittaa 500 mikrosekunttiin.



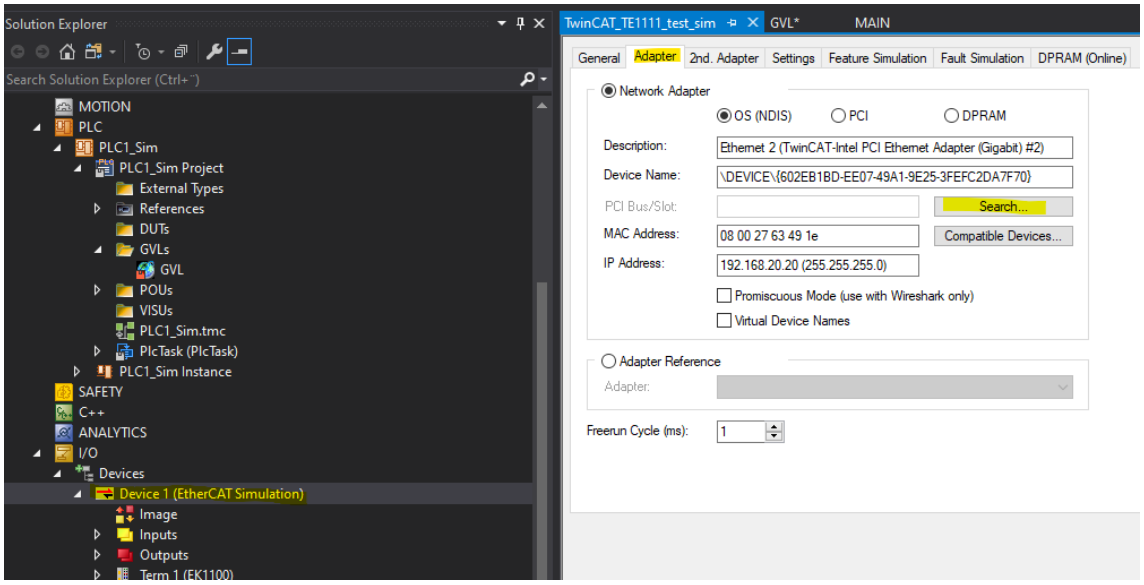
KUVA 41. Ytimien erotus TwinCAT3:n avulla

Virtuaalikoneen ja TwinCAT3:n oikean asennuksen jälkeen tehdään samat toimenpiteet kuin lokaalilla koneella projektin luomiseksi. Projektin luomisen jälkeen lisätään I/O-osioon virtuaalikoneella EtherCAT Simulation terminaali Devices-osioon, johon saadaan peilattua lokaalin koneen konfiguraatio painamalla hiiren oikeaa näppäintä terminaalin päällä ja valitsemalla kuvan mukaisesti Import ENI file -kohta. Tiedoston lisäämisen jälkeen se luo automaattisesti oikeat terminaalit ohjelmaan. (Kuva 42.)



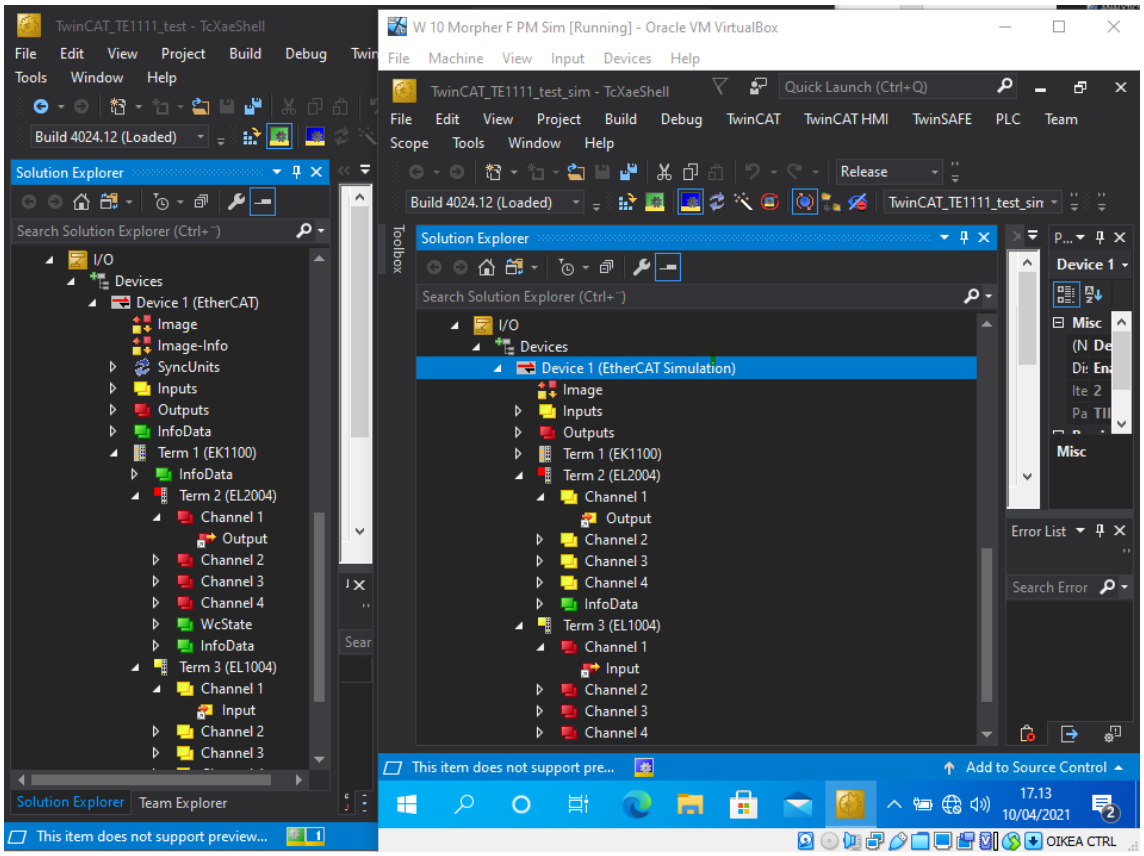
KUVA 42. EtherCAT-isäntälaitteen konfiguraation lisääminen EtherCAT-simulaatiokomponenttiin

Konfiguraation lisäämisen jälkeen EtherCAT-simulaatiolaitteeseen yhdistetään virtuaalikoneen adapteri, joka määritellään sijoittumaan samalle IP-osoitealueelle (kuva 43).



KUVA 43. Ethernet-adapterin lisääminen virtuaalikoneen Device-osioon

Virtuaalikoneen TwinCAT3:n on siis tarkoitus vastata lokaalin koneen tarpeisiin ja sen takia simulaatiolaite Devices-osiossa kääntää sisään- ja ulostulot päinvastoin linkityksiin. TwinCAT3 havainnollistaa näitä kääntöjä väreillä. Lokaalikoneessa indikoivat värit ovat punainen ulostulolle ja sisääntulossa keltainen. Simulaatiopuolella siis linkitetään suora vastaus lokaalilla koneella olevalle järjestelmälle. (Kuva 44.)



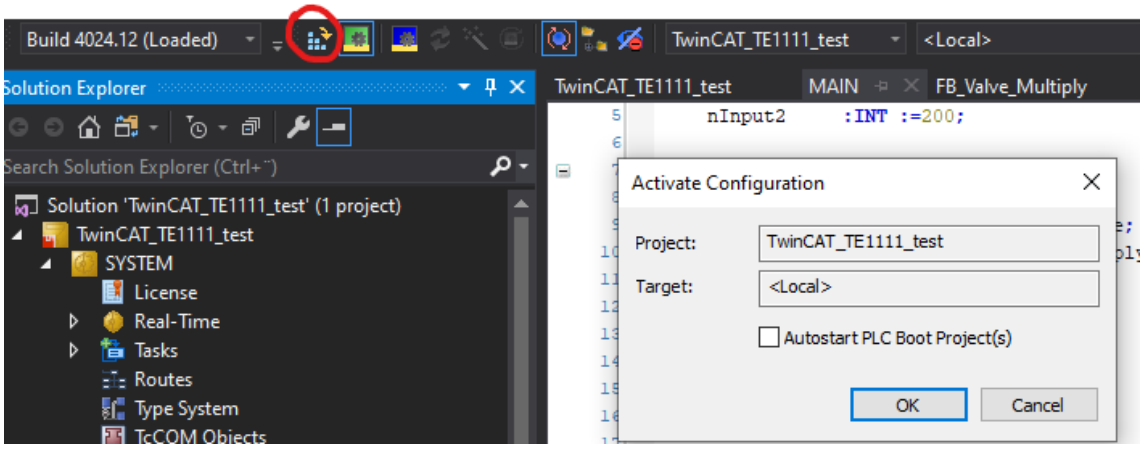
KUVA 44. Vasemmalla EtherCAT-isännän ja oikealla siihen vastaavan simulaation konfiguraatio

Esimerkissä simulaatiopuolelle määritellään pari kappaletta muuttujia, jotka linkitetään I/O-konfiguraatioon vastaamaan lokaalille koneelle (kuva 45).

```
14
15      //Simulaatio puolen ulostulo PLC:lle (Term 2 EL1004 sisääntulo kanava on käännettynä ulostuloksi)
16      bSIM_to_PLC1_1_sim  AT %Q* :BOOL;
17      bSIM_to_PLC1_2_sim  AT %Q* :BOOL;
18      bSIM_to_PLC1_3_sim  AT %Q* :BOOL;
19      bSIM_to_PLC1_4_sim  AT %Q* :BOOL;
20
```

KUVA 45. Simulaation muuttujat

Kummankin TwinCAT3-projektin on oltava aluksi config-tilassa, jonka jälkeen simulaatiopuolen konfiguraatio aktivoidaan ensin ja PLC ajetaan run-tilaan. Simulaatiopuolen ollessa run-tilassa voidaan aktivoida isäntälaitteen puoli samalla tavalla ja ajaa PLC run-tilaan (kuva 46). Ellei konfiguraation aktivointia tehdä, TwinCAT3 ei saa tietoa muutoksista ja virtuaalikone voi kaatua. Kummankin koneen Target-osion täytyy olla lokaalissa tilassa, koska ADS-yhteyden käyttäminen ylikirjoittaa linkitetyt Ethernet-portit ja sen takia vain toisen TwinCAT3:n voi laittaa ajotilaan.



KUVA 46. Konfiguraation aktivointi

Konfiguraation aktivoinnin jälkeen kumpaankin TwinCAT3:een kirjaudutaan sisään ja ne laitetaan ajotilaan. Onnistuneesti ajotilaan pääsemisen jälkeen sekä lokaalista, että simulointipuolelta voidaan muuttaa ulostulojen tiloja ja kummassakin näkyvät muutokset reaaliajassa (kuva 47).

EtherCAT master

TwinCAT_TE1111_test.PLC1.GVL_Main

Expression	Type	Value	Prepared value	Address
bPLC1_to_SIM_1	BOOL	TRUE		%Q*
bPLC1_to_SIM_2	BOOL	FALSE		%Q*
bPLC1_to_SIM_3	BOOL	TRUE		%Q*
bPLC1_to_SIM_4	BOOL	FALSE		%Q*
bSIM_to_PLC1_1	BOOL	TRUE		%I*
bSIM_to_PLC1_2	BOOL	TRUE		%I*
bSIM_to_PLC1_3	BOOL	FALSE		%I*
bSIM_to_PLC1_4	BOOL	FALSE		%I*

EtherCAT simulaatio

TwinCAT_TE1111_test_sim.PLC1_Sim.GVL

Expression	Type	Value	Prepared value	Address
bPLC1_to_SIM_1_sim	BOOL	TRUE		%I*
bPLC1_to_SIM_2_sim	BOOL	FALSE		%I*
bPLC1_to_SIM_3_sim	BOOL	TRUE		%I*
bPLC1_to_SIM_4_sim	BOOL	FALSE		%I*
bSIM_to_PLC1_1_sim	BOOL	TRUE	FALSE	%Q*
bSIM_to_PLC1_2_sim	BOOL	TRUE		%Q*
bSIM_to_PLC1_3_sim	BOOL	FALSE		%Q*
bSIM_to_PLC1_4_sim	BOOL	FALSE		%Q*

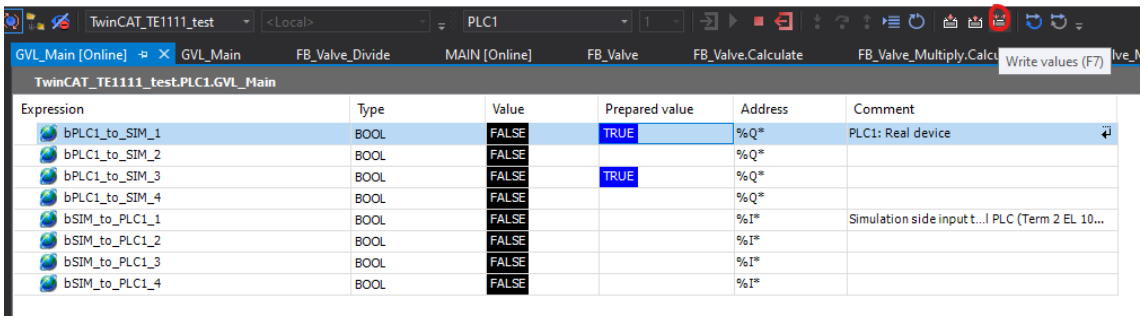
KUVA 47. Isäntä- ja simulaatiolaite Online-tilassa

Yhteyden toimiessa oikein yhdellä simulointilaitteella voidaan vastata monen eri terminaalin toimintaan, jota oikea laite kaipaa sen ohjelman etenemiseen. Simulaation puolelta voidaan siis esimerkiksi kirjoittaa takaisin tietoa ohjelmaan, joka vaatisi tiedon venttiilin aukeamisesta. (Kuva 48.)

```
6   //Simulaatiosta tulevat inputit toimilohkolle
7   fbBase  (bInput TRUE  := GVL_Main.bSIM_to_PLC1_1 TRUE );
8   fbSub1  (bInput TRUE  := GVL_Main.bSIM_to_PLC1_2 TRUE );
9   fbSub2  (bInput FALSE := GVL_Main.bSIM_to_PLC1_3 FALSE);
```

KUVA 48. Simulaation sisääntulot yhdistettynä isäntälaitteen globaaleihin muuttujiin

Muuttujien tiloja voidaan manuaalisesti muuttaa painamalla Prepared values -kohtaan TwinCAT3:n ollessa Online-tilassa. Kuvassa näkyy pelkkiä Boolean-muuttujia, mutta kirjoittaminen on mahdollista esimerkiksi Integer-muuttujan isoille arvoille. Kun haluttu arvo on kirjoitettu oikeaan osioon, voidaan painaa Write values -kohtaa kuvan oikeasta ylänurkasta ja arvot kirjoitetaan muuttujiin (kuva 49).



KUVA 49. Muuttujiin kirjoittaminen Online-tilassa

6 YHTEENVETO

Tämän työn tarkoitus oli perehtyä Beckhoffin TwinCAT3 ohjelmointiympäristön TE1111-ohjelmistokomponentin käyttöön ja opetella siihen liittyvää hyödyllistä tietoa esimerkiksi EtherCAT-väylästä. Työssä kerrotaan ensikertalaiselle perusteita eri tekniikoista, joita Beckhoff käyttää väyläratkaisussaan, sekä eri standardinmukaisista ohjelmointikielistä, joita TwinCAT3 sisältää ja niiden avulla tehtiin esimerkkiohjelma.

Beckhoffin omiin infojärjestelmiin perehtyessä ja tietoa etsiessä huomasin sen, että oikean tiedon löytäminen vaati jo valmiiksi tietyn perustason ohjelmoinnista ja erilaisista komponentteihin liittyvistä ohjeista ollakseen hyödyllisiä aloittelijalle. Varsinkin TE1111-komponentin ohjeissa kaikki on kerrottu tekijälle yleispätevästi ja ohjelmointi- tai toimintoesimerkkejä ei ole ohjeessa suoraan annettu ensikertalaiselle. Ohjeissa sanottiin esimerkiksi, että Ethernet porttien täytyi olla TwinCAT3 Real Time (RT) yhteensopivia, mikä ei ollut vaatimus toimivan ympäristön valmistamiseksi vaan kokoaikaisen ajon vaatimus. Oppiminen tapahtui monien yritysten kautta TwinCAT3-ympäristöä ajaessa ja varsinkin virtuaalikoneelle TwinCAT3:n toimivaksi saaminen oli haastavaa alkuun, koska samalla myös täytyi opetella virtuaaliympäristön ylösajoa.

Isäntäkoneen ja virtuaalikoneen välisissä testauksissa käytin ohjelmointikielenä ST-muotoa, johon halusin lisätä pienen palan olio-ohjelmointia. Testausta varten loin yhden toimilohkon, jossa käytin yhtä fyysistä muuttujaa. Toimilohkoa laajentamalla havainnollistin, miten samannimiset menetelmät ovat käytössä vain laajennuksien sisällä, ellei toisin määritellä ja annoin Calculate-toiminnoille erilaiset yksinkertaiset laskutoimitukset. Isäntälaitteen muuttujien ollessa valmiina linkitin ne konfiguraatioon, jota käytettäisiin myös oikean laitteen määrityksessä ja tallensin kokoonpanon tiedostoon, jolla se siirrettäisiin simulaatiopuolelle. Ohjelman ollessa suhteellisen valmiina oli tehtävä ytimien erotus paikallisella koneella ja virtuaalikoneella TwinCAT3:n toimivuuden varmistamiseksi ja muistaa puolittaa simulaatiopuolen ytimen taajuus verrattuna isäntälaitteeseen. Ytimien erotuksen jälkeen lisäsin simulaatiopuolelle isäntälaitteen konfiguraation, joka toi identtiset terminaalit käännetyillä tuloilla ja lähdöillä simulaatioon. Ethernet-adapterien samalle alueelle määrittämisen jälkeen pystyin testaamaan kummankin koneen TwinCAT3-ajotilaa ja yhteyksien ollessa kunnossa pääsin muuttamaan sisään- ja ulostuloja kummaltakin koneelta. Tulojen toimiessa oikein isäntäkoneen koodin testaaminen on helppoa, koska halutut sisään- ja ulostulot vastaavat simulaation puolelta ilman isäntälaitteen koodin muutoksia.

Kun kokonaisuus oli saatu toimivaksi, työ oli todella opettavainen ja toivon opinnäytetyöstä hyötyä muille käyttäjille, jotka aloittelevat Beckhoffin ympäristöön tutustumista.

LÄHTEET

1. What is EtherCAT and how does it work? Saatavissa: https://dewesoft.com/daq/what-is-ethercat-protocol Hakupäivä 8.2.2021

2. International Standard IEC 61158-2. Saatavissa: https://rpa.energy.mn/wp-content/uploads/2017/03/IEC-61158-3-Data-link-service-definition.pdf Hakupäivä 8.2.2021

3. EtherCAT System Documentation. Saatavissa: https://download.beckhoff.com/download/document/io/ethercat-terminals/ethercatsystem_en.pdf Hakupäivä 8.2021

4. ETHERCAT. Saatavissa: https://www.rtautomation.com/technologies/ethercat/ Hakupäivä 8.2.2021

5. EtherCAT – the Ethernet Fieldbus. Saatavissa: https://www.ethercat.org/pdf/english/ETG_Brochure_EN.pdf Hakupäivä 9.2.2021

6. Setting up: Device EtherCAT win cable redundancy. Saatavissa: https://infosys.beckhoff.com/english.php?content=../content/1033/cu2508/2129825163.html&id= Hakupäivä 9.2.2021

7. Beckhoff Information System: Redundancy mode. Saatavissa: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_io_intro/1446583307.html&id=8240898519565691463 Hakupäivä 9.2.2021

8. Beckhoff Information System: EtherCAT Distributed Clocks. Saatavissa: https://infosys.beckhoff.com/english.php?content=../content/1033/ethercatsystem/2469118347.html&id= Hakupäivä 10.2.2021

9. TE1111 TwinCAT 3 EtherCAT Simulation. Saatavissa: https://download.beckhoff.com/download/document/automation/twincat3/TE1111_TC3_EtherCAT_Simulation_en.pdf Hakupäivä 2.3.2021

10. 10.2 Sampling Theorem. Saatavissa: https://cnx.org/contents/d2CEAGW5@15.4:9lzz-jMl0@9/Sampling-Theorem#uid2 Hakupäivä 2.3.2021

11. Nyquist and Shannon's Sampling Theorems. Saatavissa: https://zone.ni.com/reference/en-XX/help/370524V-01/siggenhelp/fund_nyquist_and_shannon_theorems/ Hakupäivä 2.3.2021

12. Difference between Nyquist rate and Nyquist frequency? Saatavissa: https://dsp.stackexchange.com/questions/26721/difference-between-nyquist-rate-and-nyquist-frequency/26723 Hakupäivä 3.3.2021

13. Signal Sampling: Nyquist – Shannon Theorem. Saatavissa: http://195.134.76.37/applets/AppletNyquist/Appl_Nyquist2.html Hakupäivä 3.3.2021

14. Using Port Multipliers with SATA Host for Higher Performing Storage. Saatavissa: https://www.synopsys.com/designware-ip/technical-bulletin/port-multipliers.html Hakupäivä 4.3.2021

15. IEC 61131-3. Saatavissa: https://en.wikipedia.org/wiki/IEC_61131-3 Hakupäivä 4.3.2021

16. How many IEC 61131-3 languages do I need? Saatavissa: https://www.control-design.com/articles/2018/how-many-iec-61131-3-languages-do-i-need/ Hakupäivä 5.3.2021

17. IEC 61131-3: Programming Industrial Automation Systems. Saatavissa: https://sci-hub.se/10.1007/978-3-642-12015-2 Hakupäivä 5.3.2021

18. A Very Brief History of Object-Oriented Computing. Saatavissa: https://www.cut-the-knot.org/Mset99/OOhistory.shtml Hakupäivä 5.3.2021

19. C Programming Language – Computer Languages. Saatavissa: http://www.btechsmart-class.com/c_programming/C-Computer-Languages.html Hakupäivä 5.3.2021

20. The Four Pillars of Object Oriented Programming. Saatavissa: https://info.keylimeinteractive.com/the-four-pillars-of-object-oriented-programming Hakupäivä 5.3.2021

21. The three pillars of OOP – Encapsulation. Saatavissa: https://www.plccoder.com/encapsulation/ Hakupäivä 24.3.2021

22. Encapsulation – How is Information Hidden via Encapsulation Programming? Saatavissa: https://www.sumologic.com/glossary/encapsulation/ Hakupäivä 24.3.2021

23. Object Interface property. Saatavissa: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_plc_intro/4256479627.html&id= Hakupäivä 24.3.2021

24. Object Property. Saatavissa: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_plc_intro/2530335371.html&id= Hakupäivä 24.3.2021

25. The three pillars of OOP – Inheritance. Saatavissa: https://www.plccoder.com/inheritance/ Hakupäivä 28.3.2021

26. The three pillars of OOP – Polymorphism. Saatavissa: https://www.plccoder.com/polymorphism/ Hakupäivä 28.3.2021