

Petteri Mäkelä

PLC-ohjelmointi Structured Text -ohjelmointikielellä

SISÄLTÖ

SISÄLTÖ	2
1 Johdanto	4
2 PLC-ohjelmointi.....	5
2.1 IEC 61131-3 standardi	5
2.2 POU	5
2.3 Muuttujat	5
2.4 IEC 61131-3 ohjelmointikielet.....	7
2.5 ST-ohjelmointikieli	8
2.6 Tulot ja lähdöt.....	10
2.7 Ohjelman suoritus	11
3 TwinCAT 3	12
3.1 TwinCAT-projektin tekeminen	12
3.2 Esimerkkiohjelma	15
3.3 Visualisointi	18
4 Simulaattori	22
4.1 Tulot ja lähdöt.....	23
4.2 Simulaattorin asennus.....	25
4.3 TwinCAT-projektin tekeminen simulaattoria varten	26
5 Sekvenssiohjaukset	28
5.1 Mooren kone	28
5.2 Mealyn kone.....	28
5.3 PLC-ohjelmointi ja tilakoneet.....	29
5.4 Sekvenssiohjauksen tekeminen	29
5.5 Harjoitus sekvenssiohjauksesta	31
5.6 Laskuri	35
5.7 Esimerkki laskurista	36
5.8 Harjoituksia	37
6 Ajastimet	38
6.1 TON-ajastin	38
6.2 Harjoitus TON-ajastimesta	39

6.3	Ajastimet sekvenssiohjauksessa.....	43
6.4	Harjoitus ajastimesta sekvenssiohjauksessa	44
7	Funktiot ja toimilohkot	47
7.1	Funktio	47
7.2	Oman toimilohkon määrittely.....	51
7.3	Harjoitus	52
7.4	Tehtäviä	55

1 Johdanto

Tämän oppaan tavoitteena on antaa lukijalle perustiedot ohjelmoitavien logiikoiden ohjelmoinnista Structured Text (ST) -ohjelmointikielellä. Oletuksena on, että lukija osaa ohjelmoinnin perusteet jollakin muulla ohjelmointikielellä (esimerkiksi C, Python, Java tai C#) ja hänellä on käsitys automaation perusteista.

Oppaan esimerkit on tehty Beckhoffin TwinCAT 3 -kehitysympäristöllä ja esimerkkien käyttöliittymissä hyödynnetään TwinCAT3:n visualisointia. Koska TwinCATin ohjelmointiympäristö perustuu osittain CoDeSysin ohjelmointiympäristöön, on esimerkit tehtävissä myös CoDeSysillä. Tämä ei koske kuitenkaan simulaatio-ohjelmia hyödyntäviä esimerkkejä, joissa tarvitaan Beckhoffin ADS-protokollaa.

Luvussa 2 esitellään lyhyesti IEC 61131-3 -standardin mukaiset ohjelmointikielet sekä PLC-ohjelman rakenne, muuttujat ja toiminta. Luvussa 3 opastetaan TwinCAT-ympäristön käyttö esimerkkien avulla.

Luvussa 4 kerrotaan, miten sekvenssiohjaus tehdään ST-ohjelmointikielellä. Luvussa 5 esitellään ajastimet ja kerrotaan, miten tehdään ajastimia hyödyntävä sekvenssiohjaus. Ohjelman jakaminen funktioihin ja toimilohkoihin on kuvattu luvussa 6.

2 PLC-ohjelmointi

2.1 IEC 61131-3 standardi

Standardi IEC 61131 "Programmable controllers -Part 3: Programming languages" koskee ohjelmoitavien logiikoiden ohjelmointikieliä. Useimpien PLC-toimittajien ohjelmointiympäristöt tukevat IEC 61131 -standardia.

2.2 POU

Standardin IEC 61131 mukaan ohjelma koostuu itsenäisistä osista, joita kutsutaan Program Organization Uniteiksi (POU).

Program Organization Uniteita (POU) on kolmea tyyppiä

- Program eli ohjelma
- Function (funktio)
- Function block (toimilohko)

Program on varsinainen pääohjelma. Programissa määritellään muun muassa kuinka fyysiset tulot ja lähdöt linkitetään muuttujiin.

Funktio vastaa perinteisten ohjelmointikielten aliohjelmia. Funktiolle voidaan antaa parametreja. Funktiolla ei ole omaa muistia tilatiedoille, eikä se muista mitään edelliseltä suorituskerralta. Funktio tuottaa aina saman ulostulon samoilla parametreilla.

Toimilohkolla (function block) on sisäinen muisti. Ulostulon arvo riippuu parametrien lisäksi funktion sisäisten muuttujien tilasta.

2.3 Muuttujat

Muuttujat sisältävät POU:ssa käsiteltävän datan (declaration part). Muuttujat määritellään POU:n alussa. Muuttujalle annetaan nimi ja tietotyyppi. Muuttujan nimi yksilöi

muuttujan. Muuttujan tietotyyppi kertoo, mitä arvoja muuttuja voi saada. IEC 61131-3 -standardissa on määritelty useita tietotyypppejä kokonaislukuja, liukulukuja, merkkijonoja sekä päivämäärää ja aikaa varten. Alla on muutamia esimerkkejä yleisimmin käytetyistä tietotyypeistä.

- BOOL – (TRUE / FALSE)
- Bittijonot
 - BYTE – 8 bit (1 byte)
 - WORD – 16 bit (2 byte)
 - DWORD – 32 bit (4 byte)
 - LWORD – 64 bit (8 byte)
- Kokonaisluvut
 - SINT – signed short integer (1 byte)
 - INT – signed integer (2 byte)
 - DINT – signed double integer (4 byte)
 - LINT – signed long integer (8 byte)
 - USINT – Unsigned short integer (1 byte)
 - UINT – Unsigned integer (2 byte)
 - UDINT – Unsigned double integer (4 byte)
 - ULINT – Unsigned long integer (8 byte)
- Liukuluvut
 - REAL – (4 byte)
 - LREAL – (8 byte)
- Aika
 - TIME – (4 byte). Esimerkki vakiosta T#5m90s15ms
 - LTIME – (8 byte). Nanosekunnin resoluutio, T#5m90s15ms542us15ns
- Päivämäärä
 - DATE
 - LDATE
- Merkit ja merkkijonot
 - CHAR
 - STRING

Muuttujat voivat olla myös rakenteista tyyppiä, kuten taulukoita, tietueita, funktiota ja toimilohkoja.

Muuttujat esitellään VAR-alkuisen lohkon sisällä näkyvyysalueensa mukaan. VAR-lohkon sisällä määritellään POU:n paikalliset muuttujat. VAR_INPUT-, VAR_OUTPUT- ja VAR_IN_OUT-muuttujia käytetään sisään- ja ulostuloparametrien yhteydessä. VAR_GLOBAL-, VAR_EXTERNAL- ja VAR_ACCESS-määritteitä käytetään globaalien muuttujien yhteydessä. Alla on esimerkki erään toimilohkon muuttujien määrittelystä.

```
FUNCTION_BLOCK VILKKUVALO
VAR_INPUT
    start : BOOL;
    reset : BOOL;
    flashTime : TIME;
    numberOfFlashes : INT;
END_VAR
VAR_OUTPUT
    Q : BOOL; // sequence ready
    lightOn : BOOL;
END_VAR
VAR // local variables
    timer : TON;
    step : INT;
    counter : INT;
END_VAR
```

Ohjelmakoodi on omassa osassaan (code part). Ohjelma kirjoitetaan IEC 61131-3 standardissa määritellyillä ohjelmointikielillä.

2.4 IEC 61131-3 ohjelmointikielet

IEC 61131-3 -standardi määrittelee viisi ohjelmointikieltä. Tekstimuotoisia kieliä ovat Instruction list (IL) ja Structured text (ST). Graafisia symboleita käyttäviä ohjelmointikieliä ovat Ladder diagram (LD) eli tikapuukaavio, Function block diagram (FBD) ja Sequential function chart (SFC).

Instruction list eli käskylistaus on matalimman tason kieli ja se muistuttaa konekielistä ohjelmointia (assembler). Kaikki muut IEC 61131-3 voidaan kääntää IL-muotoon. IL-ohjelmat ovat vaikeita ymmärtää koodin lukijalle, ja IL-kielen käyttö on harvoin perusteltua.

Structured text (ST) on korkean abstraktiotason kieli ja se muistuttaa ”tavallisia” ohjelmointikieliä, kuten C ja Pascal. ST-ohjelma koostuu lauseista ja lausekkeista.

Ladder diagram (LD) on yleisin ohjelmoitavien logiikoiden ohjelmointikieli. LD:n tausta on sähkötekniikassa ja se perustuu virran kulun kuvaamiseen relelogiikkaohjauksessa. LD Sopii parhaiten yksinkertaisiin ohjauksiin, mutta sillä on mahdollista tehdä myös suuria ohjausjärjestelmiä.

Function block diagram (FBD) on graafinen ohjelmointikieli, joka muistuttaa digitaalitekniikassa käytettyjä IC-piirien piirikaavioita. FBD-ohjelmoinnissa yhdistetään toimilohkoja toisiinsa. Toimilohkot koostuvat sisäänmenoista, itse lohkoista ja ulostuloista. Nimitys Function Block Diagram saattaa olla hieman harhaanjohtava, sillä Function Blockeja eli toimilohkoja käytetään myös kaikissa muissa IEC 61131-3 - ohjelmointikielissä. FBD-ohjelma etenee samalla tavalla kuin tikapuuohjelma eli vasemmalta oikealle ja ylhäältä alas.

Sequential Function Chart voidaan tehdä graafisena tai tekstimuotoisena. SFC:n avulla voidaan jakaa suuri ohjelma pienempiin ja helpommin hallittaviin osiin. SFC-ohjelma koostuu askelista (tiloista) ja siirtymisistä näiden tilojen välillä.

PLC-ohjelmoinnissa yhdistetään usein eri IEC 61131-3 -standardin ohjelmointikieliä. Esimerkiksi pääohjelma saatetaan esittää toimilohkodiagrammina ja muut osat ST-kielillä.

2.5 ST-ohjelmointikieli

Tässä luvussa esitellään ST-ohjelmointikielestä vain tärkeimmät piirteet, jotka eroavat muista ohjelmointikielistä.

ST-ohjelma koostuu lauseista, kuten muutkin ohjelmointikieliset. Lauseen perässä on (yleensä) puolipiste.

Sijoituslause on samanlainen kuin Pascal-ohjelmointikielessä, eli siinä on kaksoispiste ja yhtäsuuruusmerkki:

`A := B + 10; // sijoitetaan B + 10 muuttujan A arvoksi`

Yhden rivin **kommentti** alkaa merkinnällä `//`. Usean rivin kommentti on muotoa:

`(* tämä on kommentti *)`

Operaattorit ovat enimmäkseen samoja kuin muissa ohjelmointikielissä. Potenssiin korotus on kuitenkin `**`. Negaatio (vastakohta) on `NOT`. Ja-operaattori on `AND (&)` ja tai-operaattori on `OR`. Jakojäännös on `MOD`. Yhtäsuuruutta verrataan yhdellä yhtäsuuruusmerkillä `(=)` ja erisuuruutta verrataan operaattorilla `<>`.

Ehtolause on muotoa:

```
IF startMotor AND NOT iMotlrunning THEN
    qMotlstart := TRUE;
END_IF
```

Ketjutettu ehtolause on muotoa:

```
IF a <= 10 THEN
    a := 1;
ELSIF a <= 20 THEN
    a := 2;
ELSE
    a := 3;
END_IF
```

Switch-lauseen vastine ST-ohjelmointikielessä on `CASE OF` -lause:

```

CASE a OF
  1:
    b := 10;
  2:
    b := 20;
  3:
    b := 30;
  ELSE
    b := 0;
END_CASE

```

Viiden alkion kokonaislukutaulukko määritellään näin (muuttujien määrittelyosiossa):

```

VAR
  values: ARRAY [0..5] OF INT;
END_VAR

```

While-lause näyttää tältä:

```

i := 0;
WHILE i < 10 DO
  values[i] := i;
END_WHILE

```

Vastaava **For-lause** on seuraavaa muotoa:

```

FOR i := 0 TO 5 BY 1 DO
  values[i] := i;
END_FOR

```

2.6 Tulot ja lähdöt

Ohjelmoitavan logiikan ulkoisista liitännöistä käytetään yleisesti termejä tulo ja lähtö. Tuloporttien kautta logiikka saa tietoa järjestelmän tilasta, ja lähtöporttien kautta se voi ohjata järjestelmää.

Tulot ovat kytkennän osa, jolla PLC:lle viedään tietoa (kytkimet, painonapit, anturit, näppäimistö, hiiri, ja kosketusnäyttö). Lähdöt ovat kytkennän osa, jolla PLC antaa tietoa ulkomaailmaan (moottorien ja venttiilien ohjaus, merkkivalot ja näytöt)

2.7 Ohjelman suoritus

Ohjelmakierron aluksi PLC lukee tuloihin liitettyjen antureiden, kytkimien ja lähettimien välittämät tiedot sisäisiin muistipaikkoihin. Tämän jälkeen suoritetaan ohjelmakierros. Ohjelmakierron lopuksi tieto välittyy lähtöyksiköistä ohjausväylän kautta toimilaitteille.

PLC:n ohjelmakierron ymmärtäminen saattaa olla joskus vaikeaa tavanomaiseen ohjelmointiin totuneelle ohjelmoijalle. Ohjelmakierroksessa ohjelma toistetaan kokonaisuudessaan esimerkiksi 10 ms välein. Tuloille ja lähdöille ei tapahdu mitään ohjelmakierroksen suorituksen aikana. PLC-ohjelmoinnissa lähdön tila muuttuu vasta, kun ohjelmakierros päättyy. Ohjelman eteneminen vaiheesta toiseen (sekvenssi) täytyy tehdä tilakoneen avulla.

3 TwinCAT 3

Tässä oppaassa käytetään työkaluna Beckhoffin TwinCAT 3 -ohjelmistoa. TwinCAT sisältää ajonaikaisen järjestelmän ohjauksien suorittamiseen sekä ohjelmien kehitysympäristön. TwinCAT-ohjelmia voidaan ajaa Beckhoffin ohjainten lisäksi myös PC:ssä.

Harjoituksien tekemiseen voi käyttää myös muita ympäristöjä. Esimerkiksi Codesys-työkalu on ulkoasultaan samankaltainen kanssa.

Tässä oppaassa keskitytään vain Structured Text -ohjelmoinnin perusteisiin käyttäen simulaatio-ohjelmia oikeiden ohjattavien laitteiden sijaan. Tämän takia TwinCAT-ympäristöstä kerrotaan vain se, mikä välttämätöntä PC:ssä ajettavien ohjelmien tekemiseen.

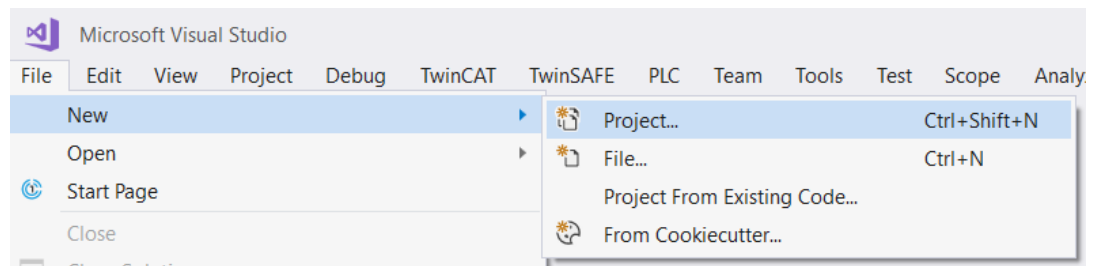
TwinCAT-ohjelma on maksuton opiskelukäytössä ja se on ladattavissa Beckhoffin sivulta www.beckhoff.com. TwinCAT asennetaan Visual Studio -ohjelman yhteyteen. Tässä oppaassa ei ole asennusohjetta, sillä asennus saattaa muuttua Visual Studion tai TwinCATin version muuttuessa.

Tässä luvussa opetellaan TwinCAT-ohjelman käyttöä esimerkkisovelluksen avulla. Samalla testataan TwinCAT-asennuksen toiminta.

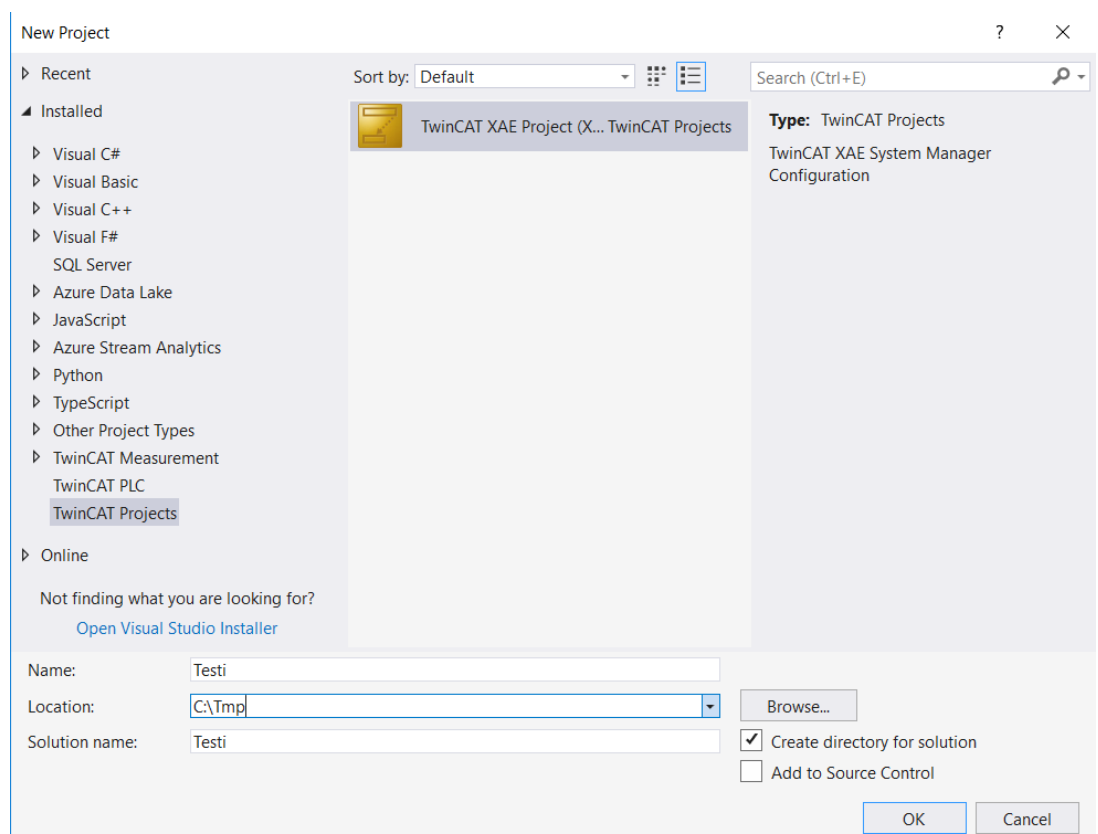
3.1 TwinCAT-projektin tekeminen

Tee uusi TwinCAT-projekti seuraavien ohjeiden mukaan.

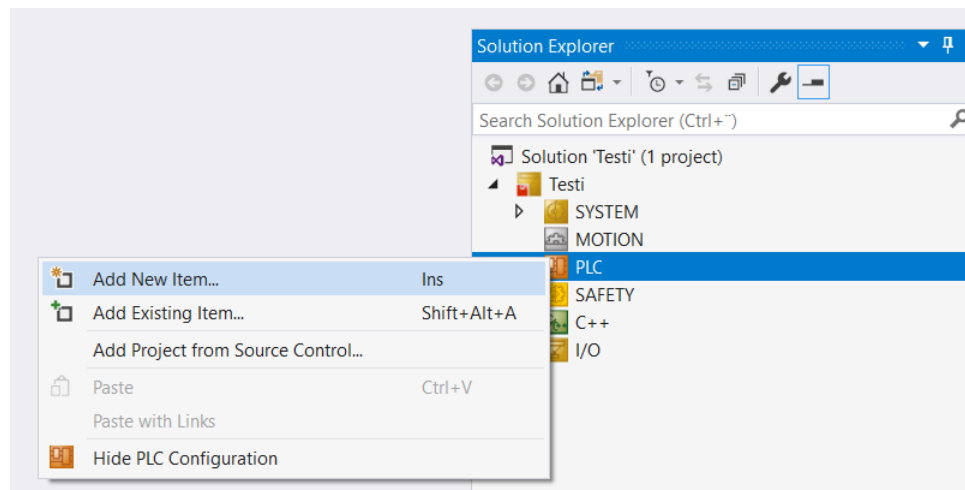
1. Käynnistä Visual Studio
2. Valitse Visual Studiosta File->New Project



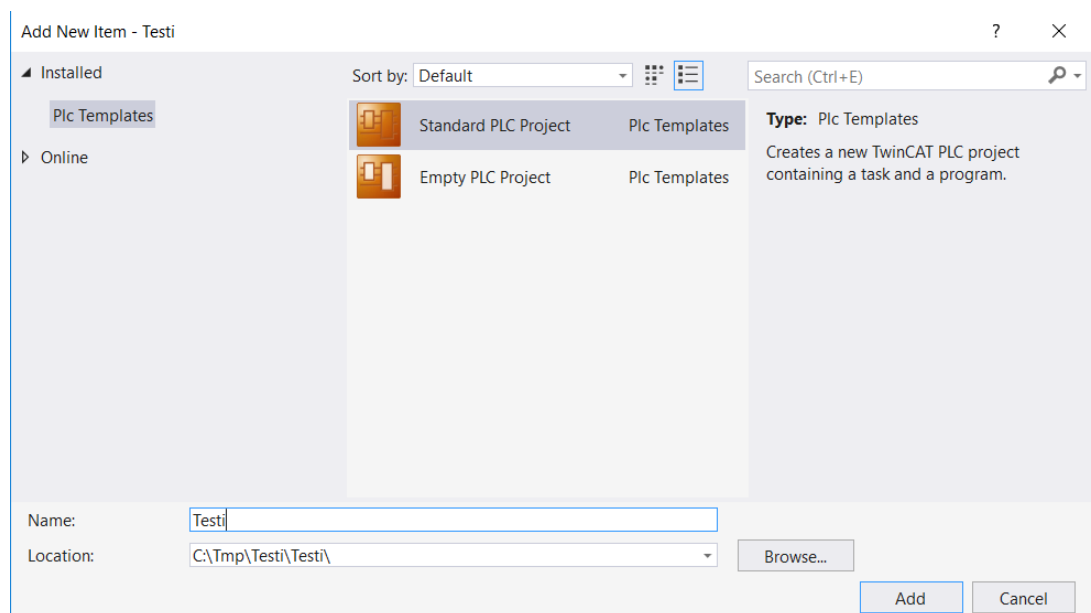
3. Valitse TwinCAT Projects ja TwinCAT XAE Project (XML format). Anna projektille nimi ja tallennuspaikka.



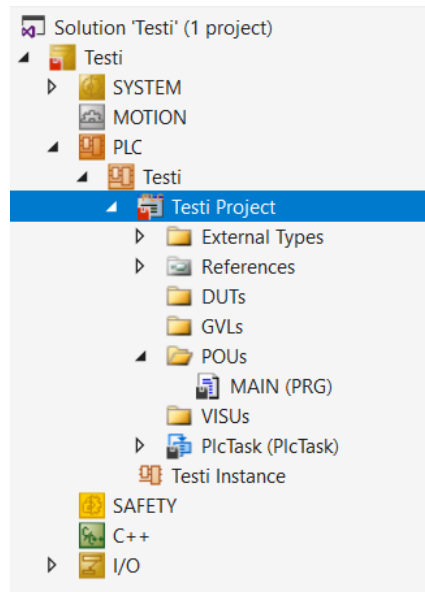
4. TwinCAT-projektin sisälle perustetaan vielä PLC-projekti. Klikkaa hiiren oikealla Solution Explorer -ikkunasta kohtaa PLC ja valitse Add New Item.



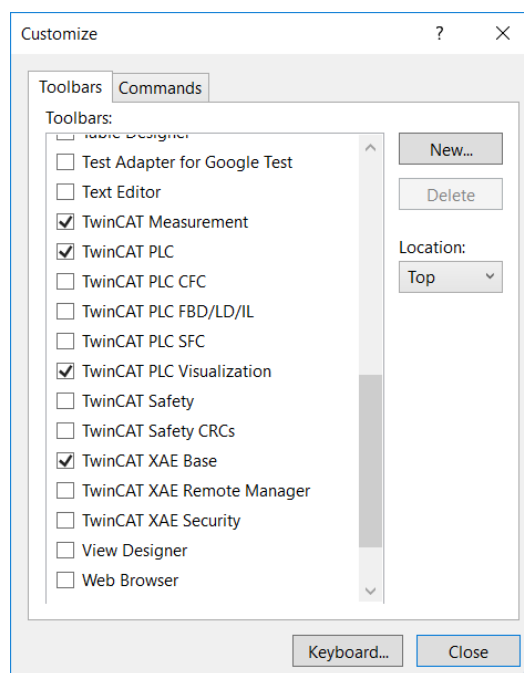
5. Valitse PLC-Templates kohdasta Standard PLC Project. Anna projektille nimi.



6. Projektin rakenne Solution Explorerissa näyttää nyt tältä:



7. Seuraavaksi muutetaan työkaluvalikon asetuksia, että tarvittavat työkalut ovat paremmin näkyvissä. Valitse Tools-valikosta Customize. Lisää Customize ikkunaan ruksit kohtiin TwinCAT Measurement, TwinCAT PLC, TwinCAT PLC Visualization ja TwinCAT XAE Base.



3.2 Esimerkkiohjelma

Tehdään seuraavaksi lyhyt esimerkkiohjelma TwinCAT-ympäristön testausta varten.

1. Avaa POUs-kansion alta pääohjelma MAIN. Pääohjelma on valmiiksi Structured Text -muodossa.
2. Esittele muuttujat. Ohjelman esittelyosassa kerrotaan, että kyseessä on ohjelma (PROGRAM), jonka nimi on MAIN. Tämän jälkeen on VAR-lohko paikallisten muuttujien esittelyä varten. Muuttujien esittelyt tulee VAR- ja END_VAR-avainsanojen väliin. TwinCAT lisää muuttujien esittelyt automaattisesti ohjelmaa kirjoitettaessa, mutta voit kirjoittaa myös muuttujien esittelyt itse alla olevan mallin mukaan.

```
PROGRAM MAIN
VAR
    start: BOOL;
    stop:  BOOL;
    light: BOOL;
END_VAR
```

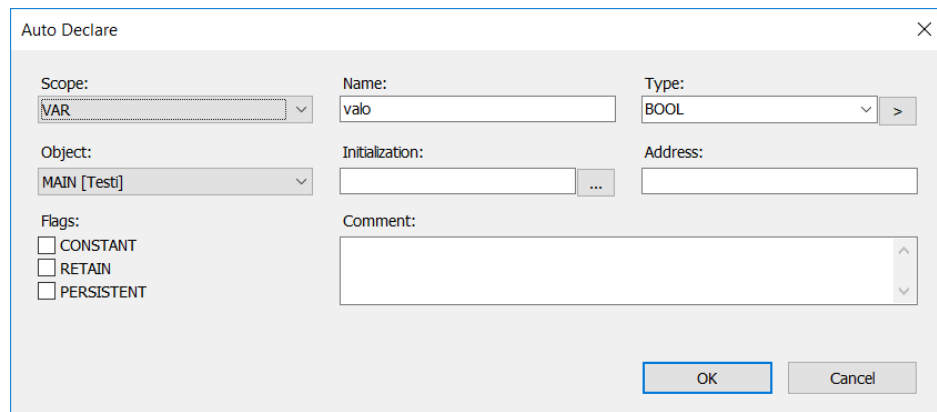
Muuttujien esittely on muotoa nimi: tyyppi;

3. Kirjoita ohjelma alla olevan mallin mukaan. Ohjelma käynnistää valon, kun painetaan start-painiketta. Valo sammuu stop-painikkeesta.

```
IF start THEN
    light := TRUE;
END_IF

IF stop THEN
    light := FALSE;
END_IF
```

Huomaa sijoitusoperaation muoto. Sijoitusoperaatiossa on puolipiste yhtäsuuruusmerkin lisäksi. Ellei uutta muuttujaa ollut jo esitelty, avaa TwinCAT ikkunan, jossa muuttujan tyyppi voidaan määritellä.

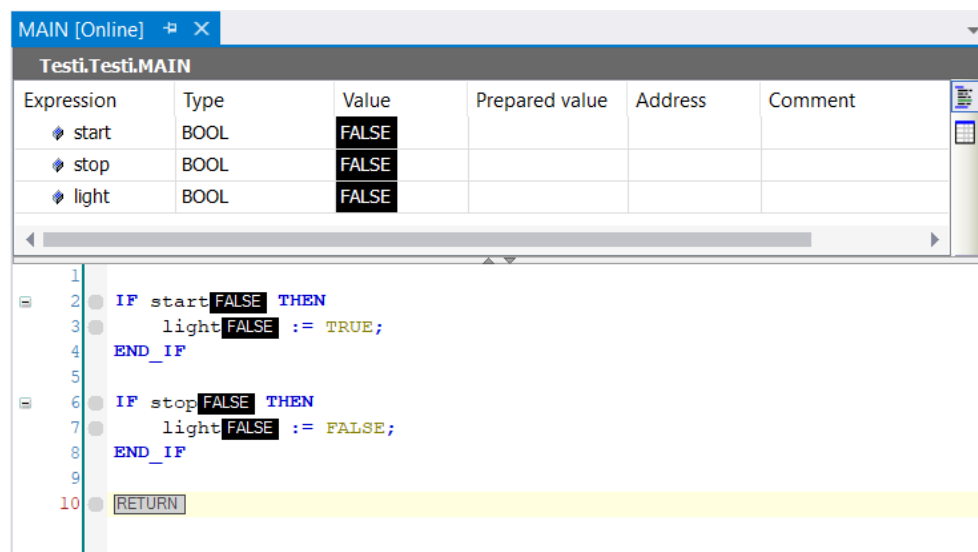


The 'Auto Declare' dialog box is shown with the following fields:

- Scope:** VAR
- Name:** valo
- Type:** BOOL
- Object:** MAIN [Testi]
- Initialization:** (empty field with a dropdown arrow)
- Address:** (empty field)
- Flags:**
 - ☐ CONSTANT
 - ☐ RETAIN
 - ☐ PERSISTENT
- Comment:** (empty text area)

Buttons: OK, Cancel

4. Käännä ohjelma valitsemalla Build-valikosta Build Solution. Korjaa mahdolliset virheet.
5. Aja ohjelma. Valitse ensin TwinCAT Active Configuration. Vastaa kysymyksiin OK.
6. Valitse PLC-valikosta ensin Login ja sitten Start. Ohjelma on nyt ajossa ja siinä näkyvät muuttujien arvot:



The screenshot shows the TwinCAT Variable Manager and Ladder Editor. The Variable Manager table is as follows:

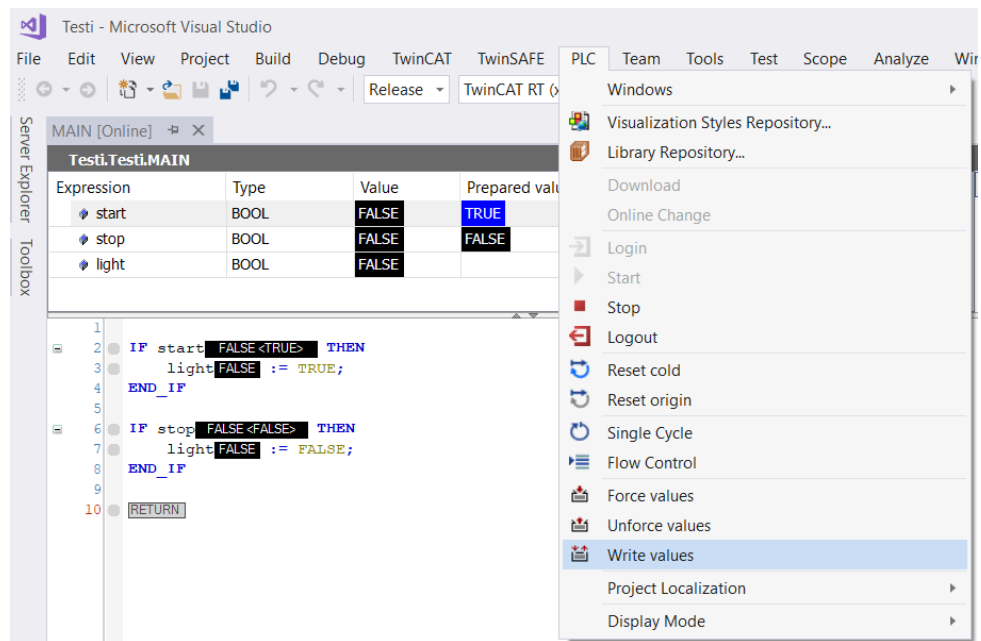
Expression	Type	Value	Prepared value	Address	Comment
start	BOOL	FALSE			
stop	BOOL	FALSE			
light	BOOL	FALSE			

The Ladder Editor shows the following code:

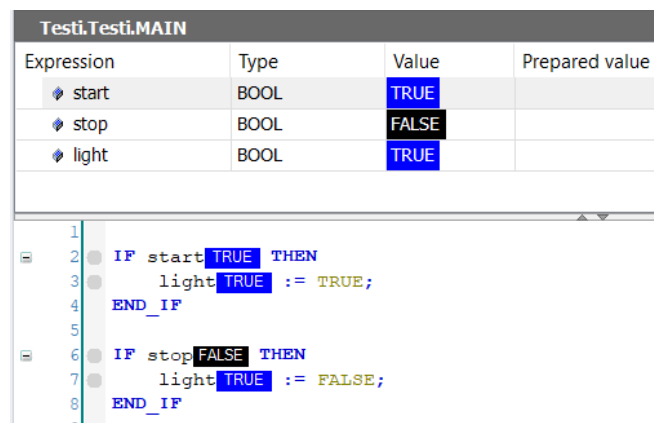
```

1  IF start FALSE THEN
2    light FALSE := TRUE;
3  END_IF
4
5
6  IF stop FALSE THEN
7    light FALSE := FALSE;
8  END_IF
9
10 RETURN
  
```

7. Muuta tulojen start ja stop arvoja. Tämä tapahtuu klikkaamalla kenttään Prepared value kyseisen muuttujan kohdalla. Valitse seuraavaksi PLC-valikosta Write Values.



Write Values päivittää Prepared value -kentässä määritellyn arvon muuttujalle. Muutos näkyy muuttujalistassa ja ohjelmassa.

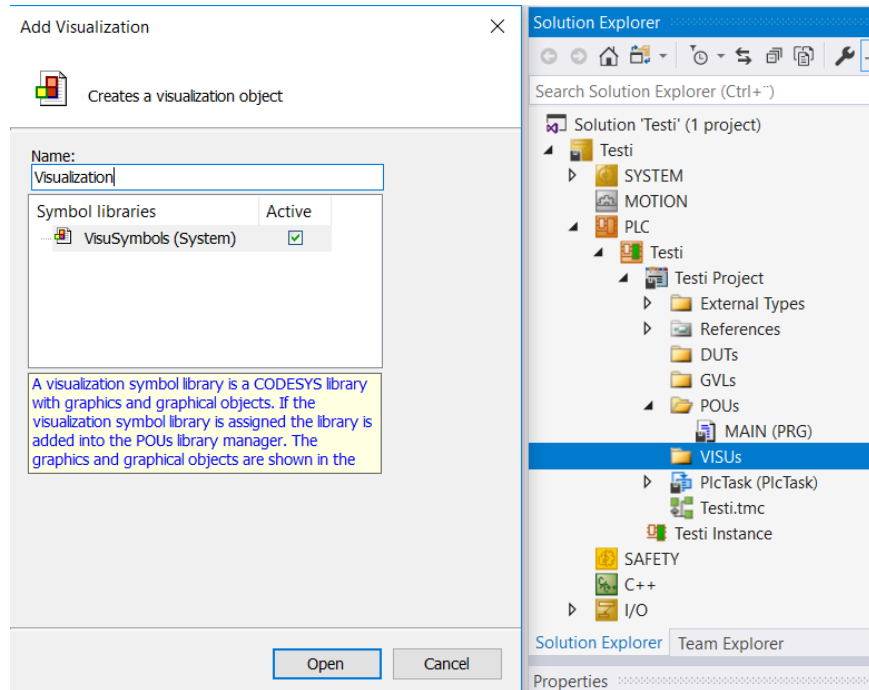


8. Lopeta ohjelma valitsemalla PLC-valikosta Logout. Tämän jälkeen voit tehdä taas muutoksia ohjelmaan.

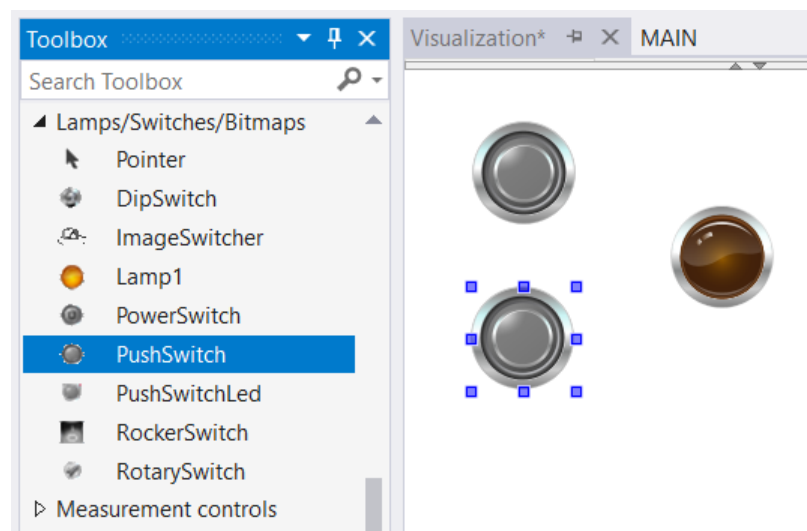
3.3 Visualisointi

Tehdään seuraavaksi ohjelmalle yksinkertainen käyttöliittymäpaneeli, jossa on start- ja stop-painikkeet sekä merkkivalo.

1. Klikkaa hiiren oikealla painikkeella VISUs-kansiota Solution Explorerissa näkyvästä projektista. Valitse Add ja Visualization. Anna näyttöön tulevassa ikkunassa visualisoinnille nimi ja rastita VisuSymbols.

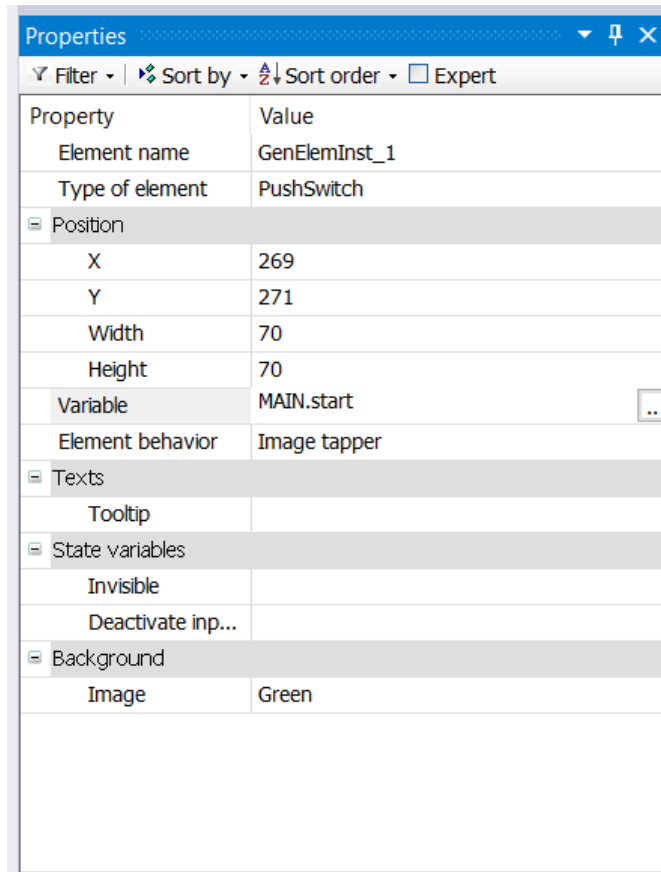


2. Lisää visualisointiin kaksi Push switch -painiketta ja yksi lamppu. Listan käyttöliittymäkomponenteista saa näkyviin valitsemaalle View-valikosta Toolbox.

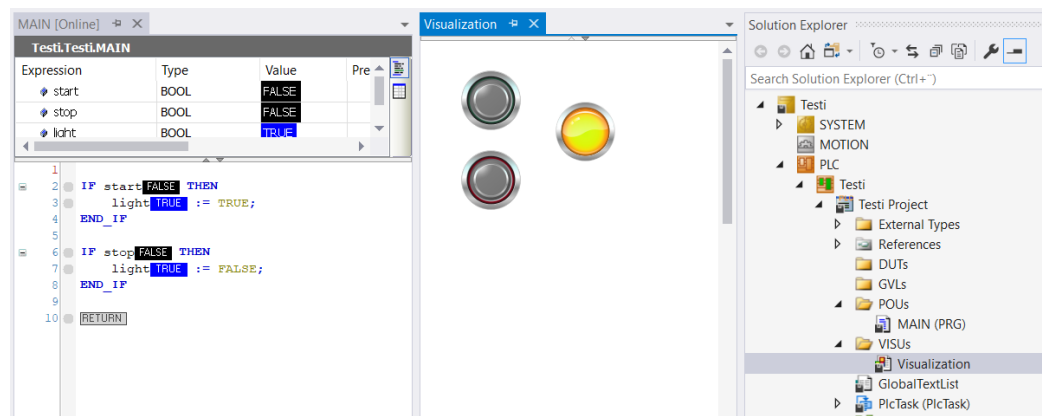


3. Muuta painikkeiden ja lampun asetuksia ja kytke ne MAIN-ohjelman muuttujiin Properties-ikkunasta. Muuta Push switch -painikkeen Element behaviour arvoon Image tapper. Näin painike ei jää pohjaan. Kytke käyttöliittymäpainike

MAIN-ohjelman muuttujaan kohdassa Variable. Voit myös muuttaa painikkeen väriä kohdassa Background -> Image. Kytke painikkeet muuttujiin MAIN.start ja MAIN.stop sekä lamppu muuttujaan MAIN.light.



4. Käännä ohjelma valitsemalla Build-valikosta Build Solution. Korjaa mahdolliset virheet.
5. Aja ohjelma valitsemalla ensin TwinCAT Active Configuration. Vastaa kysymyksiin OK. Valitse sitten PLC-valikosta ensin Login ja sitten Start. Elleivät MAIN-ohjelma ja visualisointi ole jo näkyvissä, avaa tiedostot Solution Explorerista. Saat molemmat ikkunat yhtä aikaa näkyviin valitsemalla Window-valikosta New vertical tab group.

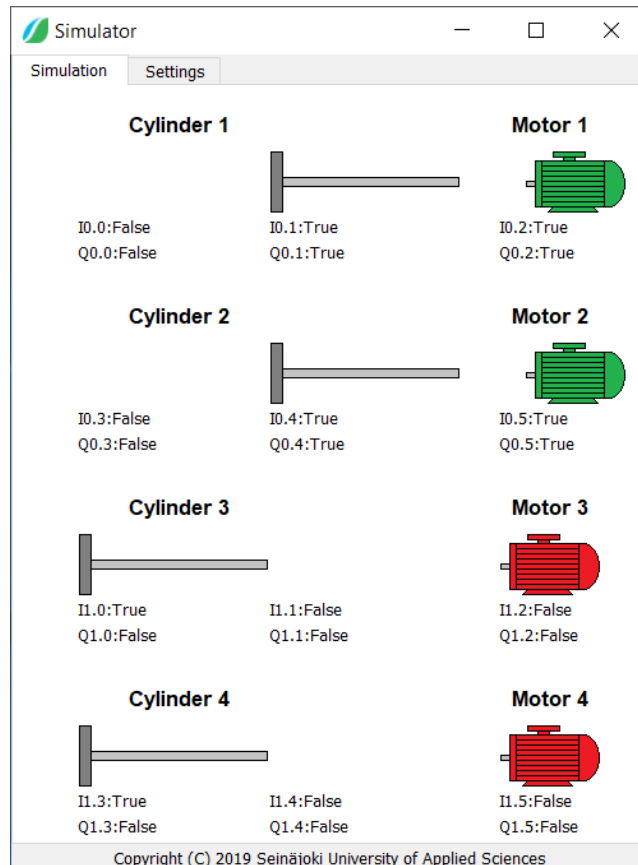


Kokeile start- ja stop-painikkeita. Muuttujien arvojen vaihtuminen näkyy MAIN-ohjelman ikkunassa.

6. Lopeta ohjelma valitsemalla PLC-valikosta Logout.

4 Simulaattori

PLC-ohjelmoinnin harjoittelua varten on tehty simulaattoriohjelma, joka on ladattavissa osoitteesta <https://github.com/SeAMKedu/ads-simulator> . Simulaattorin käyttöliittymä on alla olevassa kuvassa.



Simulaattori on erillinen Python-sovellus, joka kommunikoi TwinCATilla tehdyn PLC-ohjelman kanssa TwinCAT ADS-protokollan avulla. Simulaattorissa on neljä moottoria ja neljä sylinteriä, joita voi ajaa oikealle ja vasemmalle.

Simulaattorin tarkoitus on mallintaa ohjattavaa laitetta. Simulaattorin avulla voidaan harjoitella esimerkiksi sekvenssiä, jossa moottoreita ja sylintereitä käytetään tietyssä järjestyksessä. Esimerkiksi porausaseman ohjausta voitaisiin mallintaa seuraavalla sekvenssillä:

1. Käynnistä moottori 1, kun Start-painiketta on painettu.
2. Kun moottori on käynnistynyt, aja sylinteri 1 oikealle.
3. Kun sylinteri 1 on saapunut oikealle, odota kolme sekuntia

4. Kun kolme sekuntia on kulunut, aja sylinteri 1 vasemmalle.
5. Kun sylinteri 1 on saapunut vasemmalle, pysäytä moottori 1.
6. Kun moottori 1 on pysähtynyt, sytytä merkkivalo.

4.1 Tulot ja lähdöt

Simulaattorin yhteydessä käytettävään TwinCAT-ohjelmaan määritellään symboliset muuttujat moottorien ja sylintereiden tuloja ja lähtöjä varten. Symbolinen muuttuja on kytketty jotakin tuloa tai lähtöä vastaavaan osoitteeseen.

Tulot on määritelty VAR_INPUT-osassa:

```
VAR_INPUT
  iCyl1minus AT %IX0.0: BOOL;
  iCyl1plus  AT %IX0.1: BOOL;
  iCyl2minus AT %IX0.3: BOOL;
  iCyl2plus  AT %IX0.4: BOOL;
  iCyl3minus AT %IX1.0: BOOL;
  iCyl3plus  AT %IX1.1: BOOL;
  iCyl4minus AT %IX1.3: BOOL;
  iCyl4plus  AT %IX1.4: BOOL;
  iMot1running AT %IX0.2: BOOL;
  iMot2running AT %IX0.5: BOOL;
  iMot3running AT %IX1.2: BOOL;
  iMot4running AT %IX1.5: BOOL;
END_VAR
```

Vastaavasti lähdöt ovat VAR_OUTPUT-osassa:

```

VAR _OUTPUT
    qCyl1toMinus AT %QX0.0: BOOL;
    qCyl1toPlus AT %QX0.1: BOOL;
    qCyl2toMinus AT %QX0.3: BOOL;
    qCyl2toPlus AT %QX0.4: BOOL;
    qCyl3toMinus AT %QX1.0: BOOL;
    qCyl3toPlus AT %QX1.1: BOOL;
    qCyl4toMinus AT %QX1.3: BOOL;
    qCyl4toPlus AT %QX1.4: BOOL;
    qMot1start AT %QX0.2: BOOL;
    qMot2start AT %QX0.5: BOOL;
    qMot3start AT %QX1.2: BOOL;
    qMot4start AT %QX1.5: BOOL;
END _VAR

```

Simulaattorissa kaikki tulot ja lähdöt ovat BOOL-tyyppisiä. Moottori 1 käynnistetään asettamalla muuttuja qMot1start arvoon TRUE. Moottori ei ole käynnissä heti. Kiihdytys täyteen kierrosnopeuteen vie pari sekuntia. Vasta tämän jälkeen tulo iMot1Running saa arvon TRUE.

Vastaavasti sylinteri 1 ajetaan oikealle asettamalla lähtö qCyl1toMinus arvoon TRUE. Kun sylinteri 1 on saapunut parin sekunnin päästä oikealle, saa muuttuja iCyl1plus arvon TRUE.

Tulojen ja lähtöjen määrittelyissä on tämän kaltaisia merkintöjä: %IX0.0 ja %QX0.0. Tässä kirjain I tarkoittaa tuloa ja kirjain Q lähtöä. Kirjain X tarkoittaa, että osoite on bittimuodossa (arvo voidaan sijoittaa BOOL-muuttujaan). Jos symbolina olisi esimerkiksi kirjain W, olisi osoitteessa tilaa sana verran. Merkintä 0.0 tarkoittaa muistipaikan osoitetta.

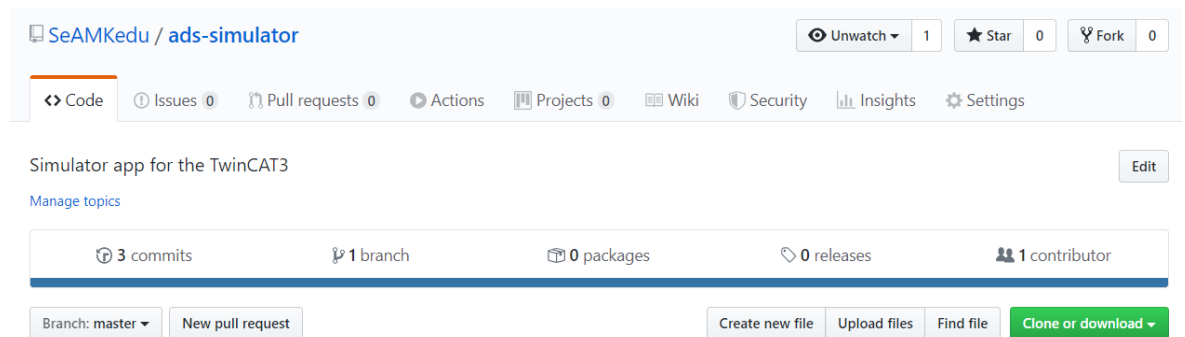
Muistiosoitteita voitaisiin käyttää suoraan ohjaamisessa. On kuitenkin parempi määrittellä kutakin tuloa ja lähtöä varten muuttujat, kuten tässä esimerkissä on tehty. Muistiosoitteet eivät ole kuitenkaan oleellisia simulaattorin käytön kannalta. Muistiosoitteita tarvitaan oikeasti vasta sitten, kun ohjelman muuttujat kytketään I/O-korttien oikeisiin tuloihin ja lähtöihin.

4.2 Simulaattorin asennus

Simulaattori on ladattavissa osoitteesta <https://github.com/SeAMKedu/ads-simulator> lähdekoodineen. Samalta sivulta löytyy myös ohjeita simulaattorin käyttämiseen.

Alla on ohjeet simulaattorin asentamiseen:

1. Asenna Python 3.x, ellei sitä ole jo asennettu.
2. Kopioi simulaattorin lähdekoodi itsellesi haluamaasi hakemistoon. Valitse painike Clone or download ja sen jälkeen Download zip.



3. Pura zip-paketti ja kopioi hakemisto ads-simulator-master haluamaasi paikkaan. Avaa komentokehote (Command prompt) ja navigoi sillä tähän hakemistoon.

```
M:\>C:
C:\>cd C:\temp\ads-simulator-master
```

4. Asenna simulaattorin tarvitsemat kirjastot pyads ja PyQt5 komentokehoteissa:
 pip install pyads pyqt5 --user
5. Simulaattori voidaan käynnistää nyt komentokehoteesta näin:
 py simulator.py
 Simulaattori antaa kuitenkin vielä tässä vaiheessa virheilmoituksen, koska TwinCAT-ohjelmaa ei ole käynnistetty.

4.3 TwinCAT-projektin tekeminen simulaattoria varten

Tee seuraavaksi TwinCAT-ohjelma alla olevien ohjeiden mukaan.

1. Perusta TwinCAT-projekti tavalliseen tapaan.
2. Esittele paikalliset muuttuja VAR-lohkossa alla olevan mallin mukaan.

```
PROGRAM MAIN
VAR
    startMotor: BOOL;
    stopMotor: BOOL;
    cylinderToRight: BOOL;
    cylinderToLeft: BOOL;
END_VAR
```

3. Kopio tulojen määrittelyt (VAR_INPUT) muuttujien määrittelyosaan VAR-lohkon perään. Määrittelyt voi kopioida sivulta <https://github.com/SeAMKedu/ads-simulator> tai alta:

```
VAR_INPUT
    iCyl1minus AT %IX0.0: BOOL;
    iCyl1plus AT %IX0.1: BOOL;
    iCyl2minus AT %IX0.3: BOOL;
    iCyl2plus AT %IX0.4: BOOL;
    iCyl3minus AT %IX1.0: BOOL;
    iCyl3plus AT %IX1.1: BOOL;
    iCyl4minus AT %IX1.3: BOOL;
    iCyl4plus AT %IX1.4: BOOL;
    iMot1running AT %IX0.2: BOOL;
    iMot2running AT %IX0.5: BOOL;
    iMot3running AT %IX1.2: BOOL;
    iMot4running AT %IX1.5: BOOL;
END_VAR
```

4. Kopio lähtöjen määrittelyt (VAR_OUTPUT) samalla tavalla VAR_INPUT-lohkon perään.

```
VAR_OUTPUT
    qCyl1toMinus AT %QX0.0: BOOL;
    qCyl1toPlus AT %QX0.1: BOOL;
    qCyl2toMinus AT %QX0.3: BOOL;
    qCyl2toPlus AT %QX0.4: BOOL;
    qCyl3toMinus AT %QX1.0: BOOL;
    qCyl3toPlus AT %QX1.1: BOOL;
    qCyl4toMinus AT %QX1.3: BOOL;
    qCyl4toPlus AT %QX1.4: BOOL;
    qMot1start AT %QX0.2: BOOL;
    qMot2start AT %QX0.5: BOOL;
    qMot3start AT %QX1.2: BOOL;
    qMot4start AT %QX1.5: BOOL;
END_VAR
```

Muuttujien määrittelyosassa on nyt alla olevat lohkot:

```
PROGRAM MAIN
```

```
VAR [4 lines]
```

```
END_VAR
```

```
VAR_INPUT [12 lines]
```

```
END_VAR
```

```
VAR_OUTPUT [12 lines]
```

```
END_VAR
```

5. Tee testiohjelma alla olevan mallin mukaan.

```
IF startMotor THEN // käynnistä moottori
```

```
    qMotlstart := TRUE;
```

```
END_IF
```

```
IF stopMotor THEN // sammuta moottori
```

```
    qMotlstart := FALSE;
```

```
END_IF
```

```
IF iCyllminus AND cylinderToRight THEN // aja sylinteri oikealle
```

```
    qCylltoPlus := TRUE;
```

```
END_IF
```

```
IF iCyllplus AND cylinderToLeft THEN // aja sylinteri vasemmalle
```

```
    qCylltoPlus := TRUE;
```

```
END_IF
```

Tee myös käyttöliittymä (Visualization), jossa on painikkeet moottorin käynnistämistä ja sammuttamista sekä sylinterin ajamista varten. Kytke painikkeisiin muuttujat startMotor, stopMotor, cylinderToRight ja cylinderToLeft. Lisää käyttöliittymään myös merkkivalot, jotka kertovat moottorin sekä sylinterin tilan (muuttujat iMot1running, iCyl1minus ja iCyl1plus).

6. TwinCAT-ohjelman on oltava käynnissä (run-tilassa) ennen kuin simulaattoria voi ajaa. Käynnistä TwinCAT-ohjelma normaaliin tapaan (Active configuration, Login ja Start).
7. Käynnistä simulaattori antamalla komentokehoteessa py simulator.py.
8. Kokeile TwinCAT-ohjelmaa. Käynnistä moottori ja aja sylinteri ensin oikealle ja sitten vasemmalle.

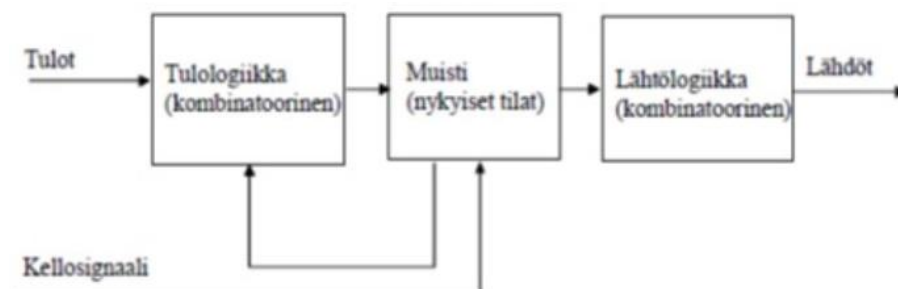
5 Sekvenssiohjaukset

Sekvenssiohjausta käytetään tehtäviin, joissa toiminnot tapahtuvat ajallisesti peräkkäin. Seuraavaan askeleen mennään yleensä edellisen askeleen kautta, kun siirtoehto toteutuu. Sekvenssiohjauksesta käytetään myös nimitystä tilakone.

Tilakoneen voidaan olevan sekvenssilogiikan kytkentä, joka tuottaa halutun lähtösekvenssin tunnetulle tulosekvenssille. Digitaalitekniikassa ja Function Block Diagram -ohjelmoinnissa tilakone tehdään kiikkujen ja kombinatorisen logiikan avulla. Tilakoneita on kahta päätyyppiä. Mooren koneessa kaikki lähdöt riippuvat vain piirin (ohjelman) tilasta. Mealyn koneessa esiintyy myös mahdollisia lähtöjä eli lähtö riippuu sekä piirin tilasta että tuloista.

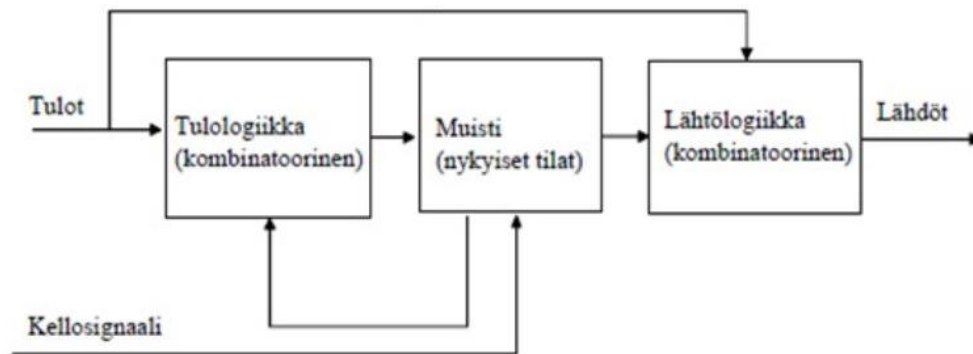
5.1 Mooren kone

Mooren tilakoneessa tulot eivät vaikuta suoraan lähtöön. Lähtöjen tilat riippuvat ainoastaan piirin tilasta. Lähtöjen tilanmuutokset on synkronoitu kellosignaaliin. Huomaa, että PLC-ohjelmoinnin yhteydessä kellojakson ajatellaan olevan ohjelmakierron jaksonaika (esim. 10 ms), eikä prosessorin kellojakson. Tulosignaalit luetaan ohjelmakierron alussa ja lähtösignaalit kirjoitetaan ohjelmakierron lopuksi.



5.2 Mealyn kone

Mealyn tilakoneessa lähtöjen tilat riippuvat nykyisistä tuloista ja piirin sisäisestä tilasta.



5.3 PLC-ohjelmointi ja tilakoneet

PLC-ohjelmoinnissa sekvenssiohjaus kannattaa tehdä Mooren koneella. Toisin sanoen sekvenssi on hyvä tehdä siten, että lähdöt riippuvat vain sekvenssin tilasta. Tulojen on tarkoitus vaikuttaa ainoastaan tilasiirtymiin, ei suoraan lähtöihin. Tämä ohje pätee niin Ladder-, Function Block Diagram- kuin ST-ohjelmointiinkin.

Toki PLC-ohjelman sekvenssin voi toteuttaa myös Mealyn koneena. Mealyn tilakone voidaan tehdä pienemmällä tilojen määrällä kuin Mooren kone. Mealyn koneen toteuttaminen oikein on kuitenkin huomattavasti vaikeampaa kuin Mooren koneen. Lisäksi ohjelmakoodista saattaa tulla vaikeammin ymmärrettävä ja lähtöjen asettaminen oikein kaikissa kombinaatioissa saattaa olla hankalaa. Mooren konetta käyttämällä pyritään siihen, että ohjelmakoodi pysyy yksinkertaisena ja selkeänä.

5.4 Sekvenssiohjauksen tekeminen

ST-ohjelmoinnissa sekvenssi tehdään yleensä int-tyyppisen tilamuuttujan ja case of -lauseen avulla. Sekvenssi kannattaa tehdä käyttäen Mooren konetta: lähtöjen tilat riippuvat ainoastaan ohjelman tilasta, ei suoraan tuloista.

Alla olevassa esimerkissä on Mooren koneen periaatetta noudattava sekvenssi, joka käynnistää kaksi moottoria peräkkäin, kun start-nappia on painettu. Esimerkissä oletetaan, että moottorin käynnistymisessä on viive. Kun lähtö qMotor1 on laitettu päälle, kestää muutaman sekunnin ennen kuin moottori on saavuttanut täyden kierrosnopeuden. Tällöin tulo iMotor1running saa arvon TRUE.

```

// nollataan kaikki lähdöt joka kierroksen alussa
qMotor1 := FALSE;
qMotor2 := FALSE;

CASE step OF
  0:
    // siirrytään alkutilasta tilaan 10, kun startia on painettu
    IF start THEN
      step := 10;
    END_IF
  10:
    // käynnistetään moottori 1
    qMotor1 := TRUE;
    // siirrytään tilaan 20, kun moottori 1 on (hetken päästä)
    // käynnistynyt
    IF iMotor1running THEN
      step := 20;
    END_IF
  20:
    // pidetään moottori 1 käynnissä ja käynnistetään myös
    // moottori 2
    qMotor1 := TRUE;
    qMotor2 := TRUE;
END_CASE

```

Esimerkki on tehty siten, että lähtöjen arvot riippuvat vain tilasta, ei suoraan tuloista. Tämä on toteutettu niin, että lähtöjä kuvaavat muuttujat asetetaan arvoon FALSE jokaisen kierroksen alussa. Lähtö vaihdetaan arvoon TRUE nykyisen tilan mukaan.

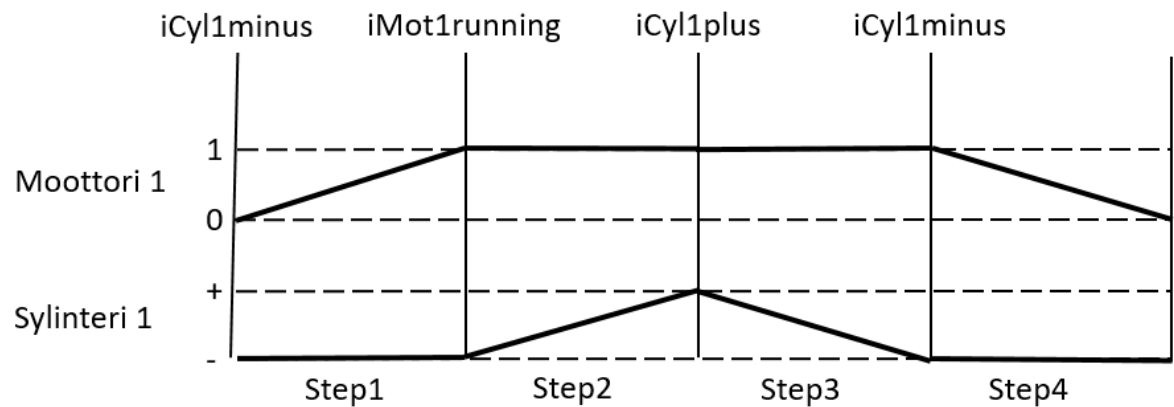
Huomaa, että PLC-ohjelmoinnissa muuttujien arvo säilyy ohjelmakierroksesta toiseen. Kannattaa myös muistaa, että muuttujan arvo kirjoitetaan fyysiseen lähtöön vasta ohjelmakierron lopuksi. Tilassa step=10 moottori 1 ei siis sammu välillä, vaikka muuttuja qMotor1 saakin arvon FALSE ohjelman alussa ja arvon TRUE case-lauseessa. Viimeisin arvo qMotor1 = TRUE jää siis voimaan. On tietenkin makuasia, noudattaako edellä mainittua periaatetta. Tämä on kuitenkin yksi tapa varmistaa, että lähtöjen arvot riippuvat vain tilasta.

Edellisessä esimerkissä tilasiirtymien ehdot (IF-lauseet) ovat case-haarojen sisällä. Uusi tila riippuu siis myös nykyisestä tilasta. Tiettyyn tilaan ei voi siis mennä mistä tahansa tilasta.

5.5 Harjoitus sekvenssiohjauksesta

Tehdään ohjelma, jossa hyödynnetään sylinteri-moottori-simulointiohjelmaa. Esi-merkissä tehdään sekvenssi, joka käynnistää ensin moottorin, ajaa sitten sylinterin oikealle ja takaisin vasemmalle ja pysäyttää lopuksi moottorin.

Ohjauksen askelkaavio on seuraava:



Sekvenssissä on siis seuraavat askeleet:

- Step 1: Käynnistä moottori
- Step 2: Kun moottori on käynnissä, aja sylinteri oikealle
- Step 3: Kun sylinteri on saapunut oikeaan reunaan, aja sylinteri takaisin vasemmalle
- Step 4: Kun sylinteri on saapunut vasempaan reunaan, sammuta moottori

Lisätään vielä Step 5: Kun moottori on sammunut, laitetaan merkkivalo päälle.

Tee uusi TwinCAT-projekti. Tee projektiin visualisointi, jossa on seuraavat käyttöliittymäkomponentit:

- Start-painike
- Stop-painike
- Merkkivalo moottori käynnissä
- Merkkivalot sylinteri vasemmalla ja oikealla
- Merkkivalo sekvenssin päättymiselle

Tee seuraavaksi ohjelmakoodi tässä luvussa olevan mallin mukaan.

1. Kopioi simulaattorin tulojen ja lähtöjen määrittelyt VAR-lohkon perään alta.

```

VAR_INPUT
    iCyl1minus AT %IX0.0: BOOL;
    iCyl1plus AT %IX0.1: BOOL;
    iCyl2minus AT %IX0.3: BOOL;
    iCyl2plus AT %IX0.4: BOOL;
    iCyl3minus AT %IX1.0: BOOL;
    iCyl3plus AT %IX1.1: BOOL;
    iCyl4minus AT %IX1.3: BOOL;
    iCyl4plus AT %IX1.4: BOOL;
    iMot1running AT %IX0.2: BOOL;
    iMot2running AT %IX0.5: BOOL;
    iMot3running AT %IX1.2: BOOL;
    iMot4running AT %IX1.5: BOOL;
END_VAR

VAR_OUTPUT
    qCyl1toMinus AT %QX0.0: BOOL;
    qCyl1toPlus AT %QX0.1: BOOL;
    qCyl2toMinus AT %QX0.3: BOOL;
    qCyl2toPlus AT %QX0.4: BOOL;
    qCyl3toMinus AT %QX1.0: BOOL;
    qCyl3toPlus AT %QX1.1: BOOL;
    qCyl4toMinus AT %QX1.3: BOOL;
    qCyl4toPlus AT %QX1.4: BOOL;
    qMot1start AT %QX0.2: BOOL;
    qMot2start AT %QX0.5: BOOL;
    qMot3start AT %QX1.2: BOOL;
    qMot4start AT %QX1.5: BOOL;
END_VAR

```

2. Määrittele paikalliset muuttujat alla olevan mallin mukaan:

```
PROGRAM MAIN
```

```
VAR
```

```

    start: BOOL;
    stop: BOOL;
    endOfSequenceLight: BOOL;
    step: INT; // current state of the sequence

```

```
END_VAR
```

```
VAR_INPUT
```

```

    iCyl1minus AT %IX0.0: BOOL;
    iCyl1plus AT %IX0.1: BOOL;
    iCyl2minus AT %IX0.3: BOOL;

```

3. Tee seuraavaksi pääohjelma. Aseta kaikki tarvittavat lähdöt arvoon FALSE ohjelman alussa.


```
// Nollataan lähdöt
qMotlstart := FALSE;
qCylltoMinus := FALSE;
qCylltoPlus := FALSE;
endOfSequenceLight := FALSE;
```

4. Lisää stop-painikkeen käsittely. Tässä ohjelmassa siirytään tilaan step=0, kun stop-nappia on painettu. Oikeassa tilanteesta pysäytys pitää tehdä hallitusti, usein monen tilan kautta.

```
// tähän lopetustoimenpiteet
IF stop THEN
    step := 0;
END_IF
```

5. Lisää case-lause ja sen sisään case-haarat kutakin tilaa varten.

```
CASE step OF
    0: // alkutila
    10: // Käynnistetään moottori
    20: // Moottori on käynnissä ja sylinteri liikkuu oikealle
    30: // Sylinteri tullut oikealle, vie se vasemmalle
    40: // moottori sammunut
    50: // sekvenssi valmis
END_CASE
```

6. Täydennä tilan 0 ohjelmakoodi mallin mukaan.

```
CASE step OF
    0: // alkutila
        // ehto, jolla siirytään seuraavaan tilaan
        IF start AND iCyllminus AND NOT iMotlrunning THEN
            step := 10;
        END_IF
```

Jos step-muuttujan arvo on 0, jatkuu ohjelman suoritus tästä case-haarasta. Kun case-haara on suoritettu, siirtyy ohjelman suoritus END_CASE-lauseen jälkeiseen lauseeseen. Seuraavia case-haaroja ei siis vielä suoriteta.

Startia painamalla siirytään tilaan 10, jos sylinteri 1 on vasemmalla ja moottori 1 ei ole käynnissä. Huomaa, että, ehdon toteutuminen aiheuttaa vain tilasiirtymän. Mikään lähtö ei mene vielä tässä vaiheessa päälle.

7. Tee seuraavaksi tilat 10 ja 20.

```

10: // Käynnistetään moottori
    // Laitetaan tämän tilan lähdöt päälle
    qMotlstart := TRUE;
    // ehdo seuraavaan tilaan siirtymiseen
    IF iMotlrunning THEN
        step := 20;
    END_IF

20: // Moottori on käynnissä ja sylinteri liikkuu oikealle
    qMotlstart := TRUE;
    qCyl1toPlus := TRUE;
    IF iCyl1plus THEN
        step := 30;
    END_IF

```

Kun step-muuttujan arvo on 10, saa muuttuja qMot1start arvon TRUE. Ohjelma pysyy tässä tilassa useamman ohjelmakierroksen ajan eli niin kauan, kun moottori 1 on kiihtynyt täyteen nopeuteen. Tällöin tulo iMot1running saa arvon TRUE ja ohjelman suoritus siirtyy tilaan 20.

Tilassa 20 moottori 1 pidetään käynnissä ja sylinteri 1 ajetaan oikealle. Tilaan 30 siirrytään sitten, kun sylinteri on saapunut oikeaan reunaan (rajakytkintä kuvaava muuttuja iCyl1plus on TRUE).

8. Tee seuraavaksi loput tilat alla olevan mallin mukaan.

```

30: // Sylinteri tullut oikealle, vie se vasemmalle
   qMotlstart := TRUE;
   qCylltoMinus := TRUE;
   IF iCyllMinus THEN
       step := 40;
   END_IF

40: // moottori sammunut
   IF NOT iMotlrunning THEN
       step := 50;
   END_IF

50: // sekvenssi valmis
   endOfSequenceLight := TRUE;

END_CASE

```

Tilassa 30 sylinteri 1 ajetaan vasemmalle moottorin 1 ollessa edelleen käynnissä. Kun sylinteri 1 on saapunut vasemmalle, siirrytään tilaan 40. Tilassa 40 yhtään lähtöä ei aseteta enää päälle. Koska qMotlstart on asetettu arvoon FALSE ohjelman alussa, moottorin pyörimisnopeus alkaa hidastua. Kun moottori on pysähtynyt, siirrytään tilaan 50.

9. Kokeile valmista ohjelmaa simulaattorin kanssa.

5.6 Laskuri

ST-ohjelmoinnissa ei kannata käyttää FBD- ja Ladder-ohjelmointikielistä tuttuja CTU-, CTD- ja CTUD-toimilohkoja laskurin toteuttamiseen. ST-ohjelmoinnissa laskuri toteutetaan kokonaislukumuuttujalla samalla tavoin kuin ohjelmoinnissa yleensä.

```

VAR
    counter : INT;
END_VAR

```

Laskurin arvoa voidaan lisätä ja vähentää yksinkertaisesti näin:

```

counter := counter + 1;

counter := counter - 1;

```

5.7 Esimerkki laskurista

Muokataan edellistä ohjelmaa siten, että sylinteri ajetaan edestakaisin vasemmalta oikealle kolme kertaa.

1. Esittele ensin laskurimuuttuja VAR-lohkossa.

```

VAR
  counter : INT;
  start: BOOL;
  stop: BOOL;
  endOfSequenceLight: BOOL;
  step: INT; // current state of the sequence
END_VAR

```

2. Muokkaa askeleen step=30 tilasiirtymän ehtoa seuraavasti: Jos laskurin arvo on pienempi kuin kolme, palataan tilaan 20, jossa sylinteri ajetaan oikealle. Muussa tapauksessa siirrytään tilaan 40, jossa moottori on pysähtymässä.

```

20: // Moottori on käynnissä ja sylinteri liikkuu oikealle
    qMotlstart := TRUE;
    qCylltoPlus := TRUE;
    IF iCyllplus THEN
        step := 30;
    END_IF
30: // Sylinteri tullut oikealle, vie se vasemmalle
    qMotlstart := TRUE;
    qCylltoMinus := TRUE;
    IF iCyllMinus THEN
        IF counter < 3 THEN
            step := 20; // palataan askeleeseen 20
        ELSE
            step := 40; // siirrytään askeleeseen 40
        END_IF
    END_IF
40: // moottori sammunut
    IF NOT iMotlrunning THEN
        step := 50;
    END_IF

```

Voit kasvattaa laskuria esimerkiksi tilassa 20 lausella `counter := counter + 1`.

5.8 Harjoituksia

1. Tee ohjaus, jossa käynnistetään ensin moottorit 1 ja 2. Kun moottorit ovat käynnissä, ajetaan sylinterit 1 ja 2 oikealle. Kun sylinterit 1 ja 2 ovat oikealla, käynnistetään moottorit 3 ja 4 ja ajetaan sylinterit 3 ja 4 oikealle. Lopuksi kaikki sylinterit ajetaan yhtä aikaa vasemmalle ja moottorit sammutetaan.

6 Ajastimet

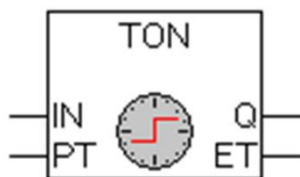
IEC 61131-3 -standardissa on kolme erilaista ajastinta:

3. TON (Timer on-delay): Viivästyttää ulostulon päälle pois laittamista.
4. TOF (Timer off-delay): Viivästyttää ulostulon laittamista pois päältä.
5. TP (Pulse timer): Käytetään tietyn pituisen pulssin generointiin.

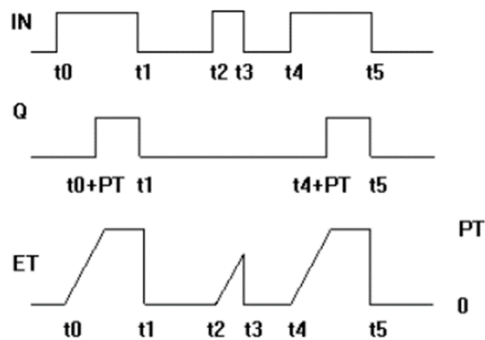
Tässä dokumentissa käsitellään vain TON-ajastin. Periaatteessa kaikki ajastinta vaativat tehtävät ovat tehtävissä sillä.

6.1 TON-ajastin

TON-ajastin viivästyttää ulostulon päälle laittamista. TON-ajastimessa on kaksi tuloa ja kaksi lähtöä.



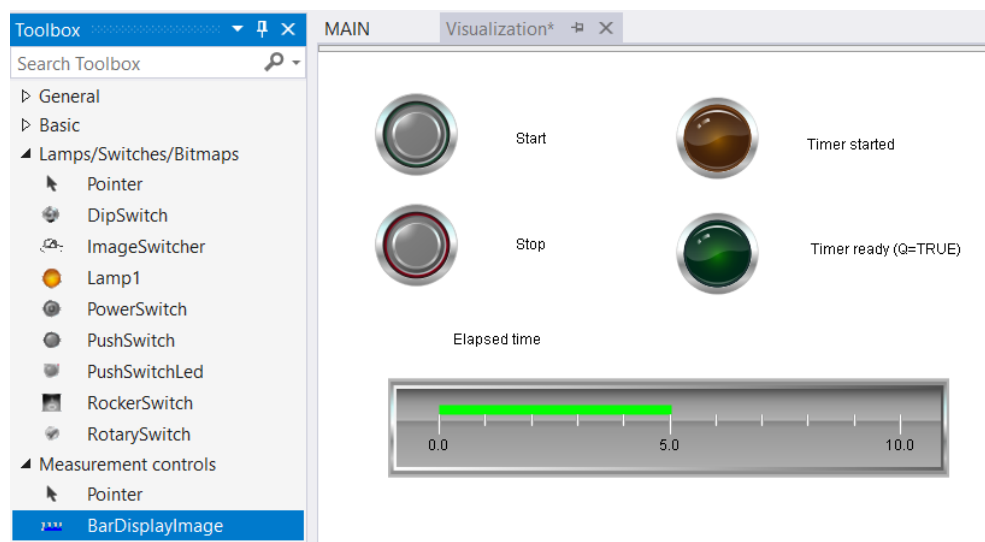
Tulo IN käynnistää ajastimen nousevalla reunalla ja sammuttaa ajastimen laskevalla reunalla. Ajastusaika asetetaan tulossa PT. Viiden sekunnin aika annettaisiin PT-tuloon muodossa T#5s. Lähtö Q on TRUE, kun IN on TRUE, ja aikaa on kulunut vähintään PT:ssä määritellyn ajan verran. ET kertoo kuluneen ajan. Q on aina FALSE, kun IN on FALSE. Kaavio TON-ajastimen toiminnasta on seuraavassa kuviossa.



6.2 Harjoitus TON-ajastimesta

Tehdään ohjelma, joka käynnistää TON-ajastimen, kun start-nappia on painettu. Tällöin syttyy myös keltainen valo. Kun 10 sekuntia on kulunut, keltainen valo sammuu ja vihreä valo syttyy. Kulunut aika näytetään käyttäjälle. Kun käyttäjä painaa stop-nappia, ajastin nollaantuu ja valot sammuvat. Harjoitus tehdään ensin ilman varsinaista sekvenssiä.

1. Tee uusi TwinCAT-projekti luvussa 3 annettujen ohjeiden mukaan.
2. Tee visualisointi alla olevan kuvan mukaan.



Start- ja Stop-painikkeet ovat tyyppiä PushSwitch ja toiminta on Image tapper eli painikkeet eivät jää pohjaan. Elapsed time -näyttö on tyyppiä BarDisplayImage ja sen lukuarvo asetetaan Properties-ikkunan kohdasta Scale.

3. Tee seuraavaksi pääohjelma. Aloita määrittelemällä muuttujat seuraavasti:

```

PROGRAM MAIN
VAR
    start: BOOL;
    stop: BOOL;
    yellowlight: BOOL;
    greenlight: BOOL;
    timer: TON;
    elapsedTime: WORD;
END_VAR

```

Muuttujien määrittelyjä ei ole pakko kirjoittaa etukäteen. TwinCAT-ohjelmointiympäristö lisää muuttujien määrittelyt sitä mukaan, kun muuttujia otetaan ohjelmakoodissa käyttöön. Tällöin täytyy tarkistaa, onko ehdotettu muuttujan tyyppi oikein.

4. Tee seuraavaksi varsinainen ohjelmakoodi alla olevan mallin mukaan:

```

// start the timer and set the yellow light on
IF start THEN
    timer(IN := TRUE, PT := T#10S);
    yellowlight := TRUE;
END_IF

// check weather timer is complete
IF timer.Q THEN
    yellowlight := FALSE;
    greenlight := TRUE;
END_IF

// reset timer and turn off the lights
IF stop THEN
    timer.IN := FALSE;
    yellowlight := FALSE;
    greenlight := FALSE;
END_IF

// timer.ET is type of TIME and it must be converted
// to milliseconds by TIME_TO_WORD function
// Seconds are obtained by dividing by 1000
elapsedTime := TIME_TO_WORD(timer.ET) / 1000;

// timer must be called in every cycle
timer();

```

Kun Start-painiketta on painettu, laitetaan keltainen valo päälle ja kutsutaan ajastinta näin:

timer(IN := TRUE, PT := T#10s)

Tässä timer-muuttuja on siis TON-tyyppinen toimilohko. Esimerkissä parametrit on annettu hieman eri tavalla kuin esimerkiksi kutsuttaessa funktiota C-ohjelmointikielessä. ST-ohjelmoinnissa voidaan antaa parametrin nimi toimilohkon kutsussa. Tällöin kaikkia parametreja ei tarvitse antaa. Kun parametrit on nimetty, niitä ei tarvitse antaa myöskään samassa järjestyksessä kuin toimilohkon määrittelyssä. Tässä esimerkissä IN-tulolle on annettu arvo TRUE ja PT-tulolle arvo T#10s (10 sekuntia).

On huomattava, että tämä ajastimen kutsu ei pidä ajastinta päällä muuta kuin sen aikaa, mitä Start-painike on pohjassa. ST-ohjelmoinnissa ajastin on päällä vain niillä ohjelmakierroksilla, joilla sitä on kutsuttu. Tässä esimerkissä kutsutaan ajastinta ohjelman lopussa ilman parametreja lauseella timer();

Jos toimilohkolle ei ole annettu parametreja (tuloja), käytetään viimeksi annettuja tulojen arvoja. Start-painikkeen painamisen jälkeen käytetään siis ohjelman lopussa olevassa timer()-kutsussa tulojen arvoja IN = TRUE ja PT = T#10s. Jos tuloja ei ole annettu vielä ollenkaan, käytetään oletusarvoja.

Itse asiassa start-painikkeen painamisen jälkeinen timer-toimilohkon kutsu on tässä tapauksessa turha, koska ajastinta kutsutaan joka tapauksessa ohjelman lopuksi. Start-painikkeen IF-lause, voidaan kirjoittaa myös näin:

```
// start the timer and set the yellow light on
IF start THEN
    timer.IN := TRUE;
    timer.PT := T#10s;
    yellowlight := TRUE;
END_IF
```

Tässä tapauksessa asetetaan vain timer-toimilohkon tuloiksi IN = TRUE ja PT = T#10s, mutta itse ajastimen kutsu puuttuu. Nämä tulojen arvot ovat kuitenkin voimassa, kun ajastinta kutsutaan ohjelman lopussa lauseella timer();

Ajastimen laukeaminen tarkistetaan lauseella timer.Q (tai timer.Q = TRUE):

```
// check weather timer is complete
IF timer.Q THEN
    yellowlight := FALSE;
    greenlight := TRUE;
END_IF
```

Tällöin laitetaan keltainen valo pois päältä ja vihreä valo päälle.

Ohjelma laitetaan alkutilaan Stop-painikkeella:

```
// reset timer and turn off the lights
IF stop THEN
    timer.IN := FALSE;
    yellowlight := FALSE;
    greenlight := FALSE;
END_IF
```

Tällöin nollataan ajastin ja otetaan merkkivalot pois päältä.

Ohjelman lopussa otetaan ajastimen ET-lähdöstä kulunut aika. Kulunut aika näytetään visualisoinnin BarDisplayImageissa. Tämä käyttöliittymäkomponentti vaatii WORD-tyyppisen muuttujan. TIME-tyyppinen arvo täytyy muuttaa WORD-tyyppiin kutsumalla funktiota TIME_TO_WORD. Vastauksena saadaan kulunut aika millisekunteina, joka pitää jakaa vielä tuhannella, että saadaan sekunteja.

```
// timer.ET is type of TIME and it must be converted
// to milliseconds by TIME_TO_WORD function
// Seconds are obtained by dividing by 1000
elapsedTime := TIME_TO_WORD(timer.ET) / 1000;

// timer must be called in every cycle
timer();
```

Kytke vielä lopuksi visualisoinnin käyttöliittymäobjektit MAIN-ohjelman muuttujiin ja kokeile ohjelman toimintaa.

Edellä olevassa esimerkissä oli oikeastaan kolme erillistä tilaa: alkutila, odotustila ja valmis-tila. Tiloja ja niiden välisiä siirtymiä ei oltu kuitenkaan vielä otettu kunnolla huomioon ohjelmassa. Useampia tiloja sisältävät ohjelmat kannattaa toteuttaa sekvenssiohjauksena.

6.3 Ajastimet sekvenssiohjauksessa

Edellisessä luvussa tutustuttiin jo ajastimen toimintaan. Tuossa esimerkissä tilakoneetta ei vielä toteutettu Mooren koneen periaatetta käyttäen. Tehdään seuraavaksi ajastinesimerkki niin, että lähdöt riippuvat ainoastaan tilasta.

1. Kopioi aiempi ajastinesimerkki pohjaksi. Lisää MAIN-ohjelman muuttujien määrittelyyn muuttuja `step : INT`.

```
VAR
    start: BOOL;
    stop: BOOL;
    yellowlight: BOOL;
    greenlight: BOOL;
    timer: TON;
    elapsedTime: WORD;
    step: INT;
END_VAR
```

2. Muokkaa pääohjelmaa siten, että siinä käytetään CASE-lausetta. Tee ohjelma alla olevan esimerkin mukaan. Nollaa ensin kaikki lähdöt, laita ajastimen IN-tulo arvoon TRUE ja käsittele lopetus.

```
// nollataan lähdöt
yellowlight := FALSE;
greenlight := FALSE;
elapsedTime := 0;

// ajastimen IN-tulo on oletusarvoisesti TRUE
timer.IN := TRUE;

// lopetus
IF stop THEN
    step := 0;
END_IF
```

Ajastin on siis oletusarvoisesti ajastamassa joka kierroksella.

3. Tee seuraavaksi ohjelma loppuun alla olevan mallin mukaan.

```

CASE step OF
  0:
    IF start THEN
      step := 10;
      // käytetään ajastin arvossa FALSE, että ajastus alkaa
      // seuraavalla ohjelmakierroksella
      timer.IN := FALSE;
    END_IF
  10: // starttia painettu, keltainen valo palaa, odotetaan 10 s
    yellowlight := TRUE;
    timer.PT := T#10S;
    elapsedTime := TIME_TO_WORD(timer.ET) / 1000;
    IF timer.Q THEN
      step := 20;
    END_IF
  20: // 10 s kulunut, vihreä valo palaa
    greenlight := TRUE;

END_CASE
// ajastinta kutsutaan joka kierroksella
timer();

```

Ajastimen IN-tulo käytetään arvossa FALSE odotustilaa edeltävässä tilasiirtymässä (tilan 0 IF-lauseessa). Kun ajastimen IN-tulo on käynyt arvossa FALSE tälle kierroksella ja saa taas arvon TRUE seuraavalla kierroksella, alkaa ajastus alusta. Ajastimen lähtö Q saa arvon TRUE, kun 10 sekuntia on kulunut. Tällöin siirrytään tilaan 20.

6.4 Harjoitus ajastimesta sekvenssiohjauksessa

Muokataan aiempaa esimerkkiä, jossa oli moottorin ja sylinterin ohjaus siten, että sylinteri pysähtyy oikealle kolmeksi sekunniksi.

1. Kopioi aiempi esimerkki itsellesi
2. Lisää esimerkin muuttujien määrittelylohkoon TON-ajastin.

```

VAR
    counter : INT;
    start: BOOL;
    stop: BOOL;
    endOfSequenceLight: BOOL;
    step: INT; // current state of the sequence
    timer : TON;
END_VAR

```

3. Aseta ajastimen IN-tulo arvoon TRUE ohjelmakierron alussa.

```

// Nollataan lähdöt
qMotlstart := FALSE;
qCylltoMinus := FALSE;
qCylltoPlus := FALSE;
endOfSequenceLight := FALSE;

// laitetaan ajastimen IN-tulo arvoon TRUE
// oletusarvona joka kierrokselle
timer.IN := TRUE;

```

4. Muokkaa siirtymäehtoa tilasta 20 siten, että ajastimen IN-tulo asetetaan arvoon FALSE. Tällä tavoin ajastin nollautuu ja se lähtee laskemaan aikaa seuraavalla ohjelmakierroksella. Aseta seuraavaksi tilaksi uusi odotustila 25.

```

20: // Moottori on käynnissä ja sylinteri liikkuu oikealle
    qMotlstart := TRUE;
    qCylltoPlus := TRUE;
    IF iCyllplus THEN
        timer.IN := FALSE;
        step := 25;
    END_IF

```

5. Lisää odotustila 25 tilojen 20 ja 30 väliin. Tässä tilassa moottori on edelleen käynnissä. Ajustusajaksi asetetaan tässä kolme sekuntia. Ajastimen lähtö timer.Q saa arvon TRUE, kun 3 sekuntia on kulunut. Tällöin siirrytään seuraavaan tilaan 30.

```

25: // odotustila
   qMotlstart := TRUE;
   timer.PT := T#3S;
   IF timer.Q THEN
       step := 30;
   END_IF

```

6. Lisää CASE-lauseen jälkeen ajastimen kutsu ilman parametreja. Kun parametreja ei ole annettu, kutsutaan ajastinta niillä parametreilla (IN ja PT), jotka ovat olleet voimassa viimeksi. Jos ohjelmakierron aikana on tapahtunut tilasiirtymä tilasta 20 tilaan 25, on timer.IN arvossa FALSE. Muuten timer.IN lähdön arvo on TRUE.

```

END_CASE
// kutsutaan ajastinta joka kierroksella
timer();

```

7 Funktiot ja toimilohkot

PLC-ohjelmat kannattaa jakaa moduuleihin samaan tapaan kuin esimerkiksi C- tai C#-ohjelmat. Jakamalla ohjelma useisiin pienempiin osiin saadaan monia etuja. Ohjelmakoodi on helpommin ymmärrettävissä, jos se jaetaan pienempiin osiin. Samoin ohjelmakoodin uudelleenkäyttö helpottuu.

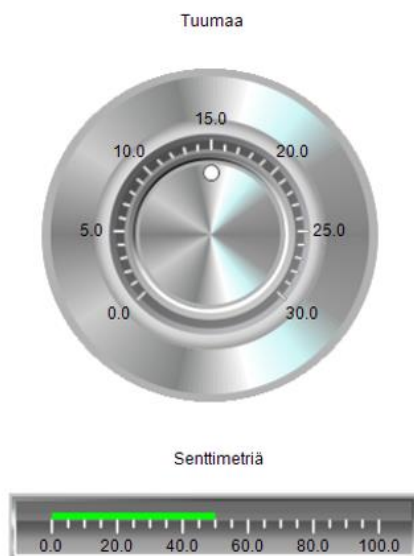
PLC-ohjelmoinnissa aliohjelmat toteutetaan funktioina tai toimilohkoina (function block). Myös olio-ohjelmointi on mahdollista IEC 61131-3 -standardin versiossa 3. Olio-ohjelmointia ei kuitenkaan käsitellä tässä dokumentissa.

Funktioilla ei ole sisäistä tilaa ja funktiot antavat aina saman ulostulon samalla syötteellä. Funktiolla on vain yksi ulostulo (paluuarvo), kuten yleisissä ohjelmointikielessä. Toimilohkolla on taas sisäinen tila (muisti), joten toimilohkon ulostulo riippuu syötteen lisäksi toimilohkon sisäisestä tilasta.

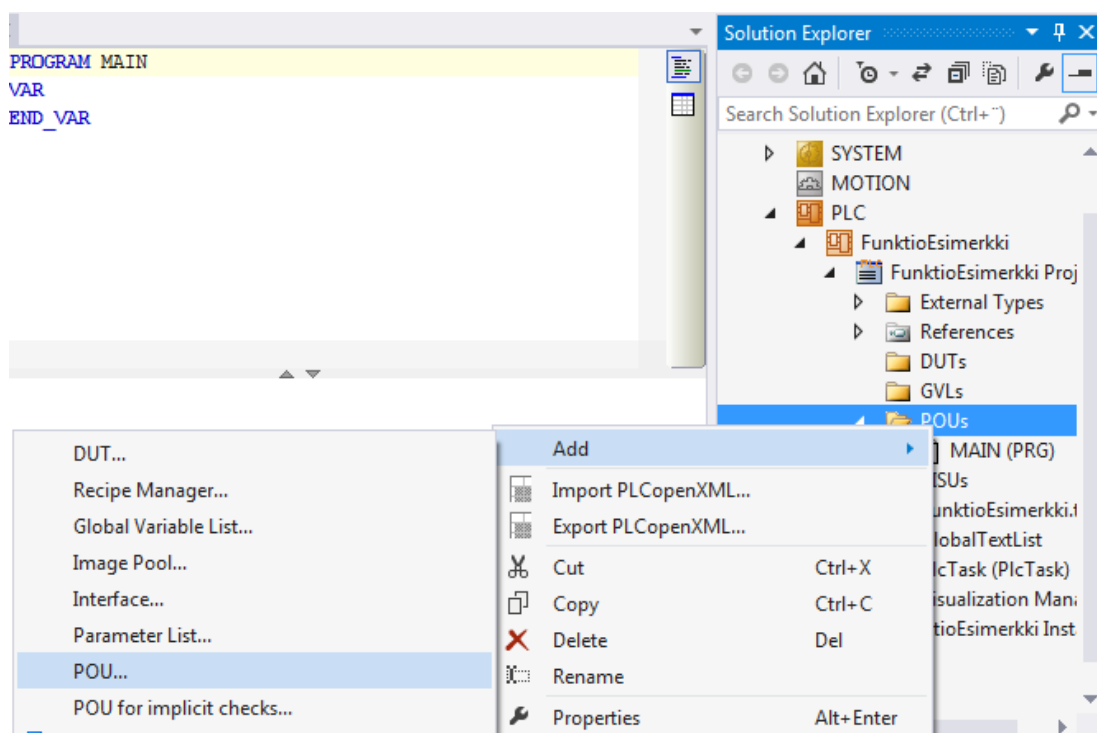
7.1 Funktio

Tehdään seuraavaksi funktio, joka muuntaa tuumat senttimetreiksi.

1. Tee ensin uusi TwinCAT projekti. Tee sitten ohjelmaan käyttöliittymä (visualization) ja lisää siihen käyttöliittymäkontrollit alla olevan mallin mukaan (Potentiometer360 ja BarDisplayImage).



2. Lisää POU-kansioon uusi funktio. Valitse hiiren oikealla näppäimellä POU-kansio, sitten Add ja sitten POU.



3. Täytä pyydytyt tiedot Add POU -ikkunaan alla olevan kuvan mukaan.

Add POU

Create a new POU (Program Organization Unit)

Name:
INCHES_TO_CM

Type

☐ Program

☐ Function Block

☐ Extends: [] ...

☐ Implements: [] ...

Access specifier:
[]

Method implementation language:
Structured Text (ST)

☒ Function

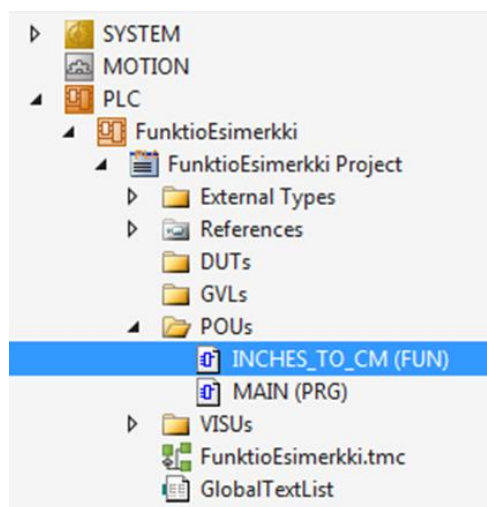
Return type: INT [] ...

Implementation language:
Structured Text (ST)

Open Cancel

- Name: **INCHES_TO_CM**
- Type: **Function**, return type: **INT**
- Implementation language: Structured Text (ST)

Lisätty funktio ilmestyy projektiin POUs-kansion alle.



4. Avaa uusi funktio INCHES_TO_CM POUs-kansioista. Muuttujien määrittelyosassa on kerrottu, että kyseessä on funktio ja sen paluuarvo on INT.

5. Lisää funktiolle parametri (tulo) inches, joka on tyyppiä INT. Funktion parametrit lisätään VAR_INPUT-lohkoon.

```
FUNCTION INCHES_TO_CM : INT
VAR_INPUT
    inches : INT;
END_VAR
VAR
END_VAR
```

Funktio ei ole tässä vaiheessa vielä mukana käännösprosessissa (funktion nimi on harmaana Solution Explorerissa). Funktiota täytyy kutsua ennen kuin se otetaan käännökseen mukaan

6. Avaa pääohjelma ja kutsu siinä funktiota INCHES_TO_CM

```
PROGRAM MAIN
VAR
    tuumat: INT;
    sentit: INT;
END_VAR

sentit := INCHES_TO_CM(tuumat);
```

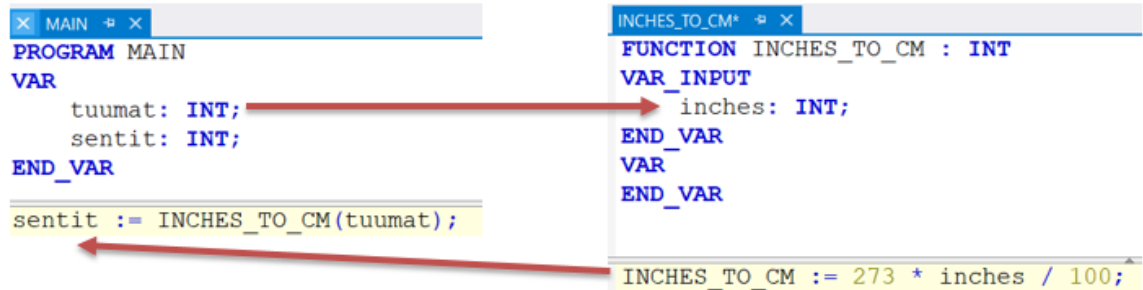
7. Tee funktion INCHES_TO_CM toteutus.

```
FUNCTION INCHES_TO_CM : INT
VAR_INPUT
    inches: INT;
END_VAR
VAR
END_VAR

INCHES_TO_CM := 25.4 * inches / 100;
```

Paluarvo määritellään kopioimalla lopputulos funktion nimeen. Tässä esimerkissä toteutus on tehty kokonaisluvuilla.

Funktion kutsu toimii samalla tavoin kuin C#-ohjelmassa. **Todellinen parametri** tuumat kopioidaan kutsussa **muodolliseen parametriin** inches. Funktion **paluuarvo** kopioidaan muuttujaan sentit.

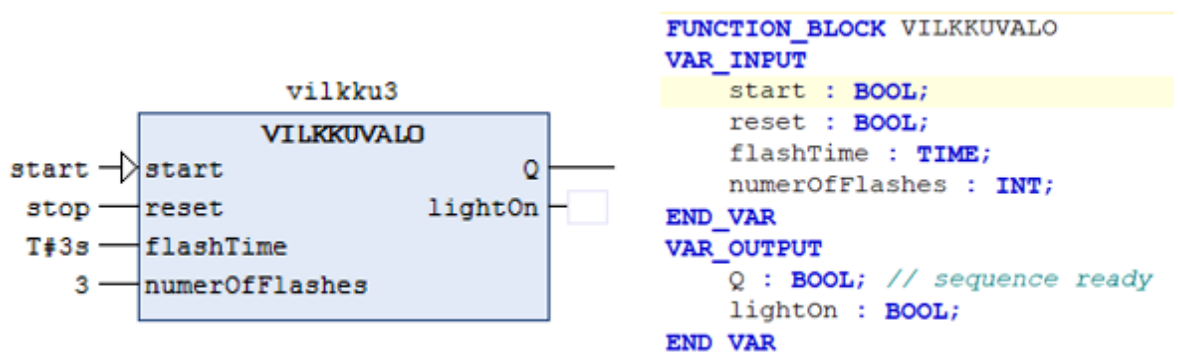


8. Kytke visualisoinnin kontrollit pääohjelman muuttujiin ja testaa ohjelmaa.

7.2 Oman toimilohkon määrittely

Omia toimilohkoja tarvitaan PLC-ohjelmoinnissa useammin kuin omia funktioita. Toimilohkolla voi olla sisäinen tila toisin kuin funktiolla. Kun funktiolla on vain yksi paluuarvo, toimilohkolla voi olla useita ulostuloja.

Toimilohkon tulot määritellään VAR_INPUT-lohkossa ja toimilohkon lähdöt VAR_OUTPUT-lohkossa. Alla olevassa kuvassa on vasemmalla toimilohko VILKKUVALO sillä tavoin, kun se näkyy Function Block Diagram (FBD) -ohjelmointikielellä tehdystä POUsta. Oikealla puolella on vastaavan toimilohkon määrittely.



Toimilohkoa kutsuvassa POUssa (esim. MAIN) toimilohkosta täytyy tehdä instanssi ennen sen käyttöä. Toisin sanoen tehdään muuttuja, jonka tyyppinä on kyseisen toimilohkon nimi.

```
PROGRAM MAIN
VAR
    start: BOOL;
    stop: BOOL;
    lamppu: BOOL;

    vilkku: VILKKUVALO;
END_VAR
```

Toimilohkoa kutsutaan ST-ohjelmointikielellä tehdystä kutsuvasta POUsta näin:

```
vilkku(start := start, reset := stop,
        flashTime := T#3S, numberOfFlashes := 3,
        Q => valmis, lightOn => lamppu);
```

Toimilohkosta voidaan tehdä tarpeen vaatiessa myös useita eri instansseja (muuttujia).

```
vilkku1 : VILKKUVALO;
vilkku2 : VILKKUVALO;
vilkku3 : VILKKUVALO;
END_VAR
```

7.3 Harjoitus

Tehdään seuraavaksi itse määriteltyä toimilohkoa hyödyntävä ohjelma. Ohjelma vilkuttaa kahta merkkivaloa.

1. Tee ensin uusi TwinCAT-projekti. Lisää siihen visualisointi, jossa on kaksi merkkivaloa sekä start- ja stop-painikkeet.
2. Lisää POU-kansioon uusi POU. Anna avautuvaan ikkunaan alla olevat tiedot:
 - Name: VILKKUVALO
 - Type: Function Block
 - Implementation language: Structured Text (ST)

3. Määrittele toimilohkon tulot ja lähdöt sekä paikalliset muuttujat alla olevan mallin mukaan.

```

FUNCTION_BLOCK VILKKUVALO
VAR_INPUT
    start : BOOL;
    reset : BOOL;
    flashTime : TIME;
    numberOfFlashes : INT;
END_VAR
VAR_OUTPUT
    Q : BOOL; // sequence ready
    lightOn : BOOL;
END_VAR
VAR // local variables
    timer : TON;
    step : INT;
    counter : INT;
END_VAR

```

4. Tee toimilohkon toteutus alla olevan mallin mukaan.

```

// reset outputs
Q := FALSE;
lightOn := FALSE;

timer.IN := TRUE; // timer is on by default

IF reset THEN
    step := 0;
    counter := 0;
END_IF

CASE step OF
    0: // initial state
        IF start THEN
            timer.IN := FALSE;
            step := 10;
        END_IF
    10: // light is on
        lightOn := TRUE;
        timer.PT := flashTime;
        IF timer.Q THEN
            counter := counter + 1;
            timer.IN := FALSE;
            step := 20;
        END_IF
    20: // light is off
        timer.PT := T#1S;
        IF timer.Q THEN
            timer.IN := FALSE;
            IF counter < numeberOfFlashes THEN
                step := 10; // go back to step 10
            ELSE
                step := 30; // go to the end state
            END_IF
        END_IF
    30: // sequence complete
        Q := TRUE;

END_CASE
timer();

```

5. Lisää sitten projektiin pääohjelma MAIN. Määrittely muuttujat vilkku1 ja vilkku2, jotka ovat tyyppiä VILKKUVALO. Lisää pääohjelmaan vielä muuttujat start, stop, lamppu1, lamppu2 ja valmis.
6. Tee pääohjelma alla olevan mallin mukaan.

```

vilkku1(start := start, reset := stop,
        flashTime := T#3S, numberOfFlashes := 2,
        lightOn => lamppu1);

vilkku2(start := start, reset := stop,
        flashTime := T#3S, numberOfFlashes := 3,
        lightOn => lamppu2);

valmis := vilkku1.Q AND vilkku2.Q;

```

Huomaa, että edellisessä esimerkissä toimilohkoja vilkku1 ja vilkku2 ajetaan yhtä aikaa. Valojen vilkutussekvenssin tapahtuu siis samaan aikaan.

7.4 Tehtäviä

1. Muuta edellistä vilkkuvaloesimerkkiä siten, että toimilohkot vilkku1 ja vilkku2 ajetaan peräkkäin.
2. Tee moottori-sylinteri-simulaattoria käyttävä ohjaus, jossa hyödynnetään itse määriteltyjä toimilohkoja. Voit ajatella, että toimilohkolla käsitellään yhtä moottori-sylinteriparia (porausliike). Toimilohko sisältää siis sekvenssin, joka käynnistää moottorin, ajaa sylinterin oikealle ja takaisin vasemmalle sekä pysäyttää moottorin.
3. Lisää edelliseen esimerkkiin askel 25, jossa odotetaan, että sylinteri odottaa oikealla kolme sekuntia ennen kuin palaa takaisin vasemmalle.