Kuutioiden pakkaus - Teoria

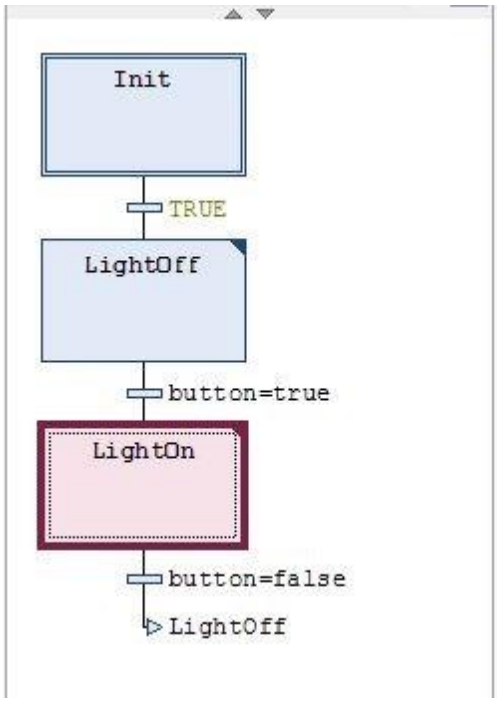
Oppimistavoitteet

- SFC (Sequential Flow Chart) kielen perusteet
- Opitaan analysoimaan sovellusta siten, että yleistajuisen tekstin ja/tai kuvien avulla jäsennetään ratkaistava ongelma sellaisiin kokonaisuuksiin, joista löytyy suoraviivainen yhteys ohjelmoijan käsitteisiin. (Tämä on keskeinen taito kaikessa ohjelmistotuotannossa, eikä vain PLC ohjelmoinnissa.)
- Mitä ohjelmistosuunnittelulla tavoitellaan ja minkälainen on hyvin suunniteltu ohjelmisto? Yleisiä tavoitteita on luettavuus, ylläpidettävyys, testattavuus, skaalattavuus. Keskeisin suunnittelukeino tavoitteiden saavuttamiseksi on modularisointi.
- Harjoituksessa suoritetaan ohjelmiston modularisointi siten, että useammassa paikassa toistuva toiminnallisuus paketoidaan toimilohkoon (function block). Sovelluksessa on 2 samalla tavalla toimivaa liukuhihnaa. Toisin sanoen, luodaan oma toimilohkotyyppi. Tämä parantaa useita ohjelmiston laatuominaisuuksia: luettavuus, ylläpidettävyys, testattavuus, skaalattavuus.
- Saavutettua modularisointia ja testattavuutta hyödynnetään suorittamalla toimilohkon yksikkötestaus.
- Ohjelmistosuunnittelu suoritetaan vähemmän yksityiskohtaisten ohjeiden mukaan kuin edellisessä harjoituksessa.
- Lisäksi harjoitellaan valvomo-ohjelmointia. (Varsinaista valvomosuunnittelua tehdään vasta kuumavesitankki sovelluksen yhteydessä.)

SFC-kielen perusteet

Ohjelmistotuotannossa pyritään ensin yleistajuisen tekstin ja/tai kuvien avulla jäsentämään ratkaistava ongelma sellaisiin kokonaisuuksiin, joista löytyy suoraviivainen yhteys ohjelmoijan käsitteisiin. Yksi tällainen käsite on *tila (state)*. Tilojen avulla mallinnettu järjestelmä on vain yhdessä tilassa kullakin ajanhetkellä – kyseistä tilaa nimitetään aktiiviseksi tilaksi. Jokaisessa tilassa on oma toiminnallisuus ja sitä vastaava ohjelmakoodi (”state active” koodi), jota suoritetaan vain silloin kun kyseinen tila on aktiivinen. PLC ohjelmistojen tapauksessa kyseinen koodi suoritetaan alusta loppuun jokaisella PLC syklillä. Tämän lisäksi voidaan määritellä koodinpätkiä, jotka suoritetaan vain kerran silloin kun kyseiseen tilaan tullaan (”state entry” koodi) tai kun kyseisestä tilasta poistutaan (”state exit” koodi).

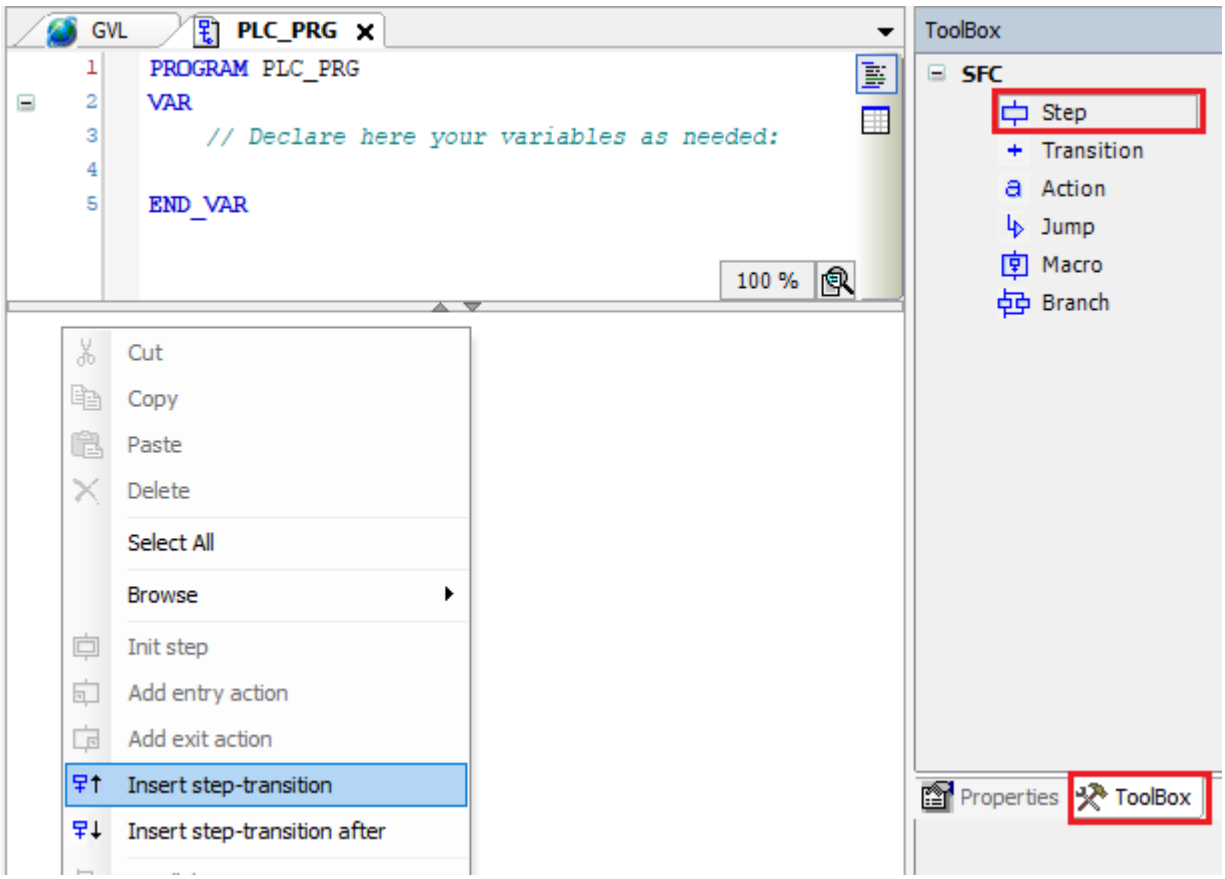
Geneerinen tilakone (”state machine” tai ”state chart”) määrittelee transitiot tilojen välillä. Transitiolla on alkutila A ja lopputila B ja transitioehto (guard condition). Jos tila A on aktiivinen ja ehto toteutuu, suoritetaan tilan A mahdollinen exit koodi, vaihdetaan B aktiiviseksi tilaksi ja suoritetaan mahdollinen tilan B entry koodi. Ehto voi olla mikä tahansa lauseke, jonka työkalu pystyy evaluoimaan tosi/epätosi. Esimerkki lampun tilakoneesta alla (kuva 1).



Kuva 1: Lampun tilakone

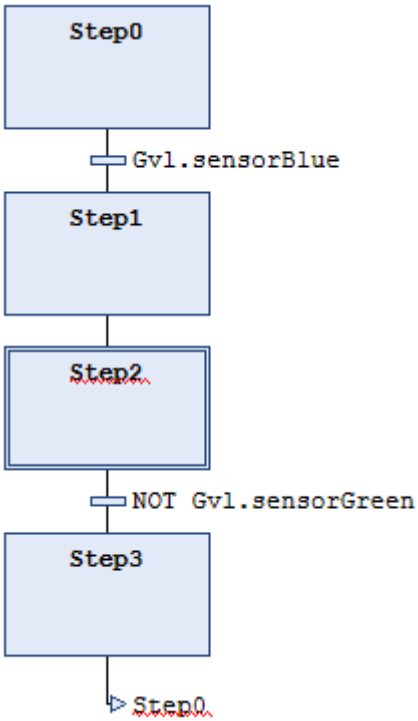
SFC on rajoitettu tilakone, jossa oletetaan, että tilat ovat peräkkäisiä vaiheita, eli transitio on aina seuraavaan tilaan. (Tämän lisäksi SFCssä voi olla rinnakkaisia haaroja, joita ei käsitellä tässä harjoituksessa.) Tämän takia SFC tiloja kutsutaan askeliksi (step). Step entry/active/exit toiminnallisuus voidaan kirjoittaa millä tahansa IEC 61131-3 standardin PLC ohjelmointikielellä, mukaan lukien SFC:llä.

SFC ohjelman laatiminen alkaa raahaamalla ToolBoxista Step PLC_PRG:n koodialueelle. Vaihtoehtoinen tapa on painaa koodialueella oikeaa hiirennäppäintä ja valita ”Insert step-transition”. Esimerkki alla kuvassa 2.



Kuva 2: SFC- kielellä stepin luominen

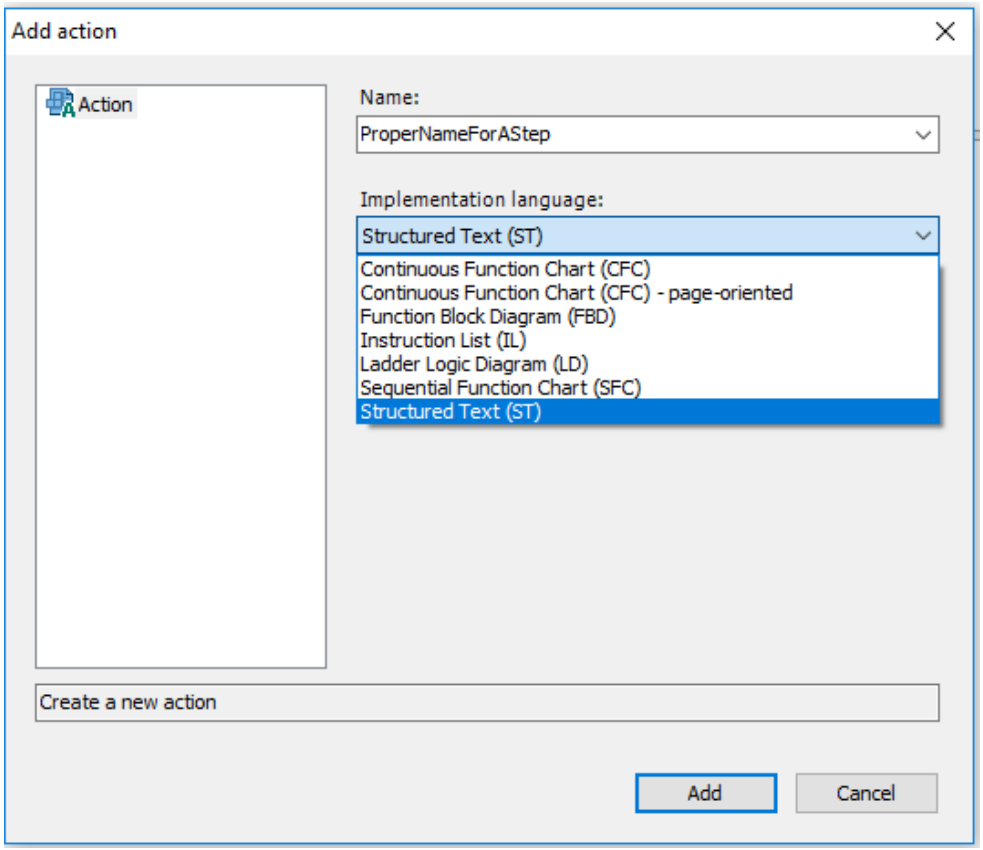
Tilakoneella on aina alustustila ”Init”, jossa järjestelmä on, kun se ensimmäisen kerran käynnistetään. CodeSys määrittää ensimmäisenä koodialueelle asetetun stepin alustus tilaksi (Init step). Alustus tilan tunnistaa kaksoisääriviivoista (kuvan 3 step 2). Alustustilan voi määrittää painamalla sellaista tilaa, joka ei ole alustustila oikealla hiirennäppäimellä ja valitsemalla ”Init step”.



Kuva 3: alustustila

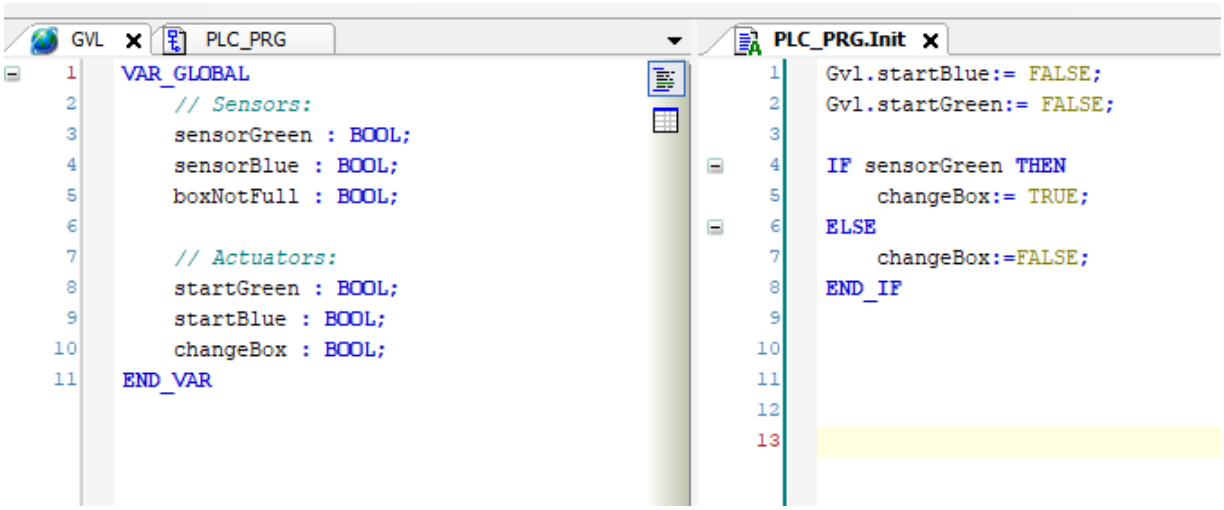
Uusia tiloja voi PLC_PRG:hen lisätä joko raahaamalla ToolBoxista uusia steppejä tai painamalla jotain tilaa oikealla hiirennäppäimellä ja valitsemalla ”Insert Step-Transition” tai ”Insert Step-Transition after”. Erona näillä tavoilla on se, että ”Insert Step-Transition” lisää tilan lisäksi siirtymäehdon, transition. Siirtymäehto voidaan lisätä myös ToolBoxista erikseen. Transitioehtoa voidaan editoida tuplaklikkaamalla. Kuvan 3 esimerkissä tilasta 0 poistutaan, kun Globaalina muuttujana märitetty sininen anturi saa TOSI arvon. Tilasta 1 poistutaan, kun sen sisällä ollut ohjelmakoodi on suoritettu. Tilan 3 jälkeen hypätään takaisin tilaan 0 ”jumpin” avulla.

Tilan sisällä tapahtuvaa ohjelmakoodia voidaan määritellä ja muokata lisäämällä Step active. Step active toiminnallisuuden voi lisätä tuplaklikkaamalla tilaa, jolloin valitaan millä kielellä se toteutetaan ja mikä on tilan nimi. Kuutiot tehtävän osalta haluamme Active actionin kielenä käyttää ST-kieltä (Kuva 4). Tilalle voidaan määrittää myös Exit ja entry actionit, mutta näitä käsitellään kuumavesitankki tehtävän yhteydessä.



Kuva 4: Kielen valinta

Tilan sisällä määritetään muuttujien arvot ja mahdolliset ehtolauseista syntyvät dynaamiset muutokset. Esimerkki alla kuvassa 5. Esimerkissä globaalit muuttujat (GVL:ssä olevat muuttujat), jotka ohjaavat sinisen ja vihreän hihnan moottoria alustetaan pois päältä ja isoa laatikkoa vaihtavan hihnan moottorin arvoa muutetaan vihreän hihnan anturin tilan perusteella.



Kuva 5. Muuttujien alustus

Sovelluksen analysointi

Moduuli (module) on ohjelmistosuunnittelun perusyksikkö. Termi on löyhästi määritelty, koska eri ohjelmointikielet tukevat modularisointia eri tavoin. Esimerkiksi oliokielet kuten Java ja C++ tarjoavat luokan modularisoinnin välineeksi. Moduulin keskeisiä ominaisuuksia ovat seuraavat:

- Moduulilla on hyvin määritellyt *riippuvuudet (dependency)* muiden ohjelmiston osien välillä. Jos moduulina on toimilohko, nämä riippuvuudet ovat moduulin sisään- ja ulostulot, jotka pitää osata kytkeä ennen kuin moduulia voidaan käyttää.

- Moduuli yleensä erittelee *rajapinnan (interface)* ja *toteutuksen (implementation)*. Toimilohkon tapauksessa rajapinta ja sen dokumentaatio riittävät siihen, että joku muu voi käyttää lohkoa omissa sovelluksissaan. Vain moduulin ohjelmoijan tarvitsee tuntea sovellus. Tämä periaate on keskeinen, jotta satojen tai tuhansien ohjelmoijien aikaansaannoksia voidaan tehokkaasti hyödyntää jonkun ohjelmistosovelluksen tuottamiseksi.

- Mitä paremmin rajapinta ja riippuvuudet on kuvattu, sitä helpompaa on tehdä sovellukseen muutoksia tai laajennuksia.
    - *Ylläpidettävyys (maintainability)* tarkoittaa sitä, että muutosten tekeminen (testaus mukaan lukien) on helppoa.
    - *Skaalautuvuus (scalability)* tarkoittaa sitä että ohjelmiston kasvattaminen on helppoa, jos ohjelmiston tehtäviä lisätään
    - *Testattavuus (testability)* tarkoittaa sitä, että ohjelmiston osia on helppo testata erikseen (vikojen paikantaminen on huomattavasti helpompaa, jos testattava kohde on pieni). Testattavuus koostuu kahdesta tekijästä, joihin moduulisuunnittelu vaikuttaa *(HUOM: termeille ei ole vakiintuneita käännöksiä, joten termien suomennetut versiot ovat vapaita käännöksiä)*:
        - *ohjattavuus (controllability):* kuinka helposti voidaan asettaa testattavan moduulin sisääntulot
        - *tarkkailtavuus (observability):* kuinka helposti moduulin ulostuloja voidaan havainnoida ja tarkkailla
    - *Luettavuus (readability)* tarkoittaa sitä, että ihminen pystyy helposti hahmottamaan mitä ohjelma tekee. Esimerkiksi monimutkaisen sekvenssiohjauksen pystyy kyllä tekemään siten että kaikki koodi on yhdessä pitkässä tekstitiedostossa, mutta vaiheiden ja vaiheesta toiseen

siirtymislogiikan hahmottaminen on huomattavasti vaikeampaa kuin SFC koodista.

- Yksikkötestaus (unit test) tarkoittaa moduulin testaamista erikseen (vastakohtana sille, että testataan ajamalla koko sovellus).
    - Jos sovellus on iso, yksikkötestauksella saavutetaan 2 suurta etua:
        - vikojen paikantaminen on helpompaa, kun testattu koodi on pieni (esim. helpompi etsiä vikaa moduulista, jossa on 100 riviä koodia, kuin koko sovelluksesta, jossa on 10000 riviä koodia
        - helpompi testata moduuli kattavasti, kun testit voidaan laatia huomioiden moduulin rajapinnan kautta tulevien syötteiden kaikki mahdolliset kombinaatiot. Kaupallisen kokoisissa sovelluksissa täydellinen testikattavuus on mahdotonta. Niiltä osin kuin testit eivät kata toiminnallisuutta, loppu testauksesta jää asiakkaalle. On viime kädessä bisnespäätös, kuinka kattavaan testaukseen pyritään. Teollisuusautomaation sovellukset ovat usein turvallisuuskriittisiä, joten ohjelmistotestauksen kattavuuden osoittaminen on olennaista.
    - Yksikkötestaus aiheuttaa lisätyötä:
        - Pitää luoda ympäristö, jossa voidaan muuttaa moduulin sisääntuloja siten, että testit voidaan suorittaa. Ympäristön pitää myös havainnollistaa moduulin ulostuloja siten, että testaaja voi päätellä menikö testit läpi.
        - Monessa tapauksessa yksikkötestauksen hyödyt ovat suurempia kuin sen vaatima ylimääräinen työ. Ylimääräisen työn suuruus riippuu puolestaan siitä, miten testattavaa koodia on tehty
    - Ohjelmistotestaus on ammattitaitoa: AUT koulutusohjelmasta on viime vuosina tehty firmoihin lukuisia diplomitöitä ohjelmistotestaukseen liittyen.

- Testisuunnitelma koostuu joukosta testitapauksia. Testitapauksen tulee täyttää seuraavat kriteerit:
    - Kuka tahansa pätevä ohjelmistoinsinööri voi ohjeiden mukaan suorittaa testin yksikäsitteisesti
    - Testitapauksessa on määritelty haluttu lopputulos, jolloin testaajalle ei jää tulkinnanvaraa siihen, että menikö testi läpi vai ei
    - Tapauksia luodaan kattamaan sekä normaali toiminta että poikkeustilanteet

Ohjelmiston suunnittelu

Huomataan että molemmat liukuhihnat suorittavat täsmälleen samanlaista toiminnallisuutta, mutta hyödyntävät eri laatikon havaitsemisanturia ja liukuhihnan käynnistys/pysäytys toimilaitetta. *Hyvä yleisperiaate on, että pätevästi modularisoidussa ohjelmistossa ei ole tarpeen kopioida samaa koodia useampaan paikkaan.* Tässä tapauksessa luodaan uusi toimilohko nimeltä HihnanOhjaus, joka pudottaa N kpl kuutioita laatikkoon ja sitten pysäyttää hihnan, kunnes se saa ”uudelleenkäynnistä” signaalin, jolloin laskuri nollataan ja liukuhihna käynnistetään.

Jotta toimilohkoa voidaan käyttää molempiin hihnoihin, siitä pitää luoda kaksi instanssia: yksi ylempää hihnaa varten ja toinen alempaa. Lisäksi oikea anturi ja toimilaite pitää kytkeä oikean toimilohkon input/output muuttujiin. Laatikon vaihtaminen ja hihnojen uudelleenkäynnistäminen eivät aiheuta tarvetta kopioida samaa koodia useampaan paikkaan, joten tämä toiminnallisuus voidaan hoitaa ohjelmassa (program).