TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Informaatika aluste õppetool

Praktilised meetodid programmeeritava loogikakontrolleri tarkvara testimiseks

Magistritöö

Üliõpilane: Indrek Ploompuu
Üliõpilaskood: 132301 IAPM
Juhendaja: Ants Torim

Tallinn
2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

---------- ----------

(*kuupäev*) (*allkiri*)

Annotatsioon

Programmeeritava loogikakontrolleri tarkvara testimine on seotud paljude probleemidega. Testimine võib olla raskendatud vajaliku riistvara puudumise tõttu, samas võivad riistvaraga testimisel esilekerkivad tarkvaravead põhjustada ohtu seadmetele, keskkonnale või inimesele. Testitavaid protsesse võib olla inimese poolt raske jälgida. Käesoleva töö eesmärk on välja pakkuda praktilised testimismeetodid, mida oleks võimalik kasutada programmeeritava loogikakontrolleri tarkvara testimiseks igasugustes probleemsetes olukordades. Meetodid peavad võimaldama teste läbi viia nii riistvaraga koos kui ilma, nii käsitsi kui automaatselt.

Töös analüüsitud valdkonna senises praktikas kasutatavad testimismeetodid katavad ühiselt enamuse testimisel esilekerkivatest probleemsetest olukordadest, kuid mitte kõiki. Ülejäänud olukordade katmiseks pakutakse töös välja uus testimismeetod, mis võimaldab ka automaattestide läbiviimist. Kirjeldatud testimismeetodid koos lisatud soovituste ja näpunäidetega on tarkvaraarendajate poolt praktikas kergesti kasutusele võetavad.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 84 leheküljel, 4 peatükki, 17 joonist, 17 tabelit.

Abstract

Testing software for a Programmable Logic Controller can be related to many problems. It can be difficult because of missing hardware. Testing with hardware can potentially be hazardous for the equipment, environment and people in case of emerging software faults. The process under test could be difficult to observe by humans. The goal of this thesis is to find or propose practical testing methods that could be used to test the software for a Programmable Logic Controller in all kinds of problematic situations. The methods should allow conducting software tests with or without hardware, manually or automatically.

The testing methods used by the problem domain of this thesis in practice together cover most of the problematic situations that emerge during testing, but not all of them. For the situations not covered by these methods a new method is proposed that covers everything, including conducting automated tests. The described testing methods along with some suggestions and tips can be easily put to practice by software developers.

The thesis is in Estonian and contains 84 pages of text, 4 chapters, 17 figures, 17 tables.

Lühendite ja mõistete sõnastik

PLC	*Programmable Logic Controller* Programmeeritav loogikakontroller
IO	*Input-Output* Sisendid ja väljundid
DI	*Digital Input* Sigitaalsisend
DO	*Digital Output* Digitaalväljund
AI	*Analog Input* Analoogsisend
AO	*Analog Output* Analoogväljund
SCADA	*Supervisory Control and Data Acquisition* Järelevalve ja andmete kogumine – PLC protsesside juhtimiseks inimese poolt
LD	*Ladder Diagram* Redel-loogika – PLC tarkvara programmeerimiskeel
FBD	*Function Block Diagram* Funktsiooni-plokk-skeem – PLC tarkvara programmeerimiskeel
PC	*Personal Computer* Personaalarvuti

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

1.1 Taust ja probleem

Tööstusautomaatika või ka laiemalt lihtsalt automaatika all mõeldakse üldiselt süsteeme, mille abil juhitakse riistvaraseadmeid nagu pumpasid, klappe, hüdraulikat või servosi. Automaatikasüsteem nõuab tavaliselt minimaalset inimesepoolset sekkumist, aga võib olla ka täiesti automatiseeritud. Lihtsamates automaatikasüsteemides kasutatakse riistvara juhtimiseks releeautomaatikat või eelprogrammeeritud kontrollereid, mille kasutamiseks pole tarkvara vaja luua, neid tuleb vaid seadistada. Keerulisemates automaatikasüsteemides kasutatakse aga programmeeritavaid loogikakontrollereid (PLC) – kontrollereid, mille tarkvara on võimalik programmeerida, et riistvara soovitud viisil tööle panna.

Ilma vigadeta tarkvara pole olemas. See väide kehtib ka PLC tarkvara kohta. Seetõttu on tarvis PLC tarkvara kvaliteedi tõstmiseks seda ka testida. Testimise käigus kontrollitakse, kas juba loodud tarkvara töötab vastavalt nõuetele.

Kui üldiselt on tarkvaraarendajad harjunud teste läbi viima tarkvara käivitamise ja testolukordade käsitsi või automatiseeritud läbimängimise teel, siis PLC tarkvara testimisel võib see olla komplitseeritud, kuna PLC tarkvara on osa automaatikasüsteemist, mis hõlmab endas ka riistvara. Harjumuspärased meetodid võivad riistvara kahjustada. Eksisteerib oht inimestele, keskkonnale. Samuti on vaja tarkvara testida ka siis, kui riistvarale puudub üldse ligipääs või testitavad protsessid on inimese poolt raskesti juhitavad või jälgitavad. Ka PLC tarkvara iseärasused võivad testimisel takistuseks saada. Seetõttu pole kõiki olukordi võimalik tuntud meetodeid kasutades läbi mängida.

PLC tarkvara testimine selle käivitamise teel on hädavajalik aga riistvaraga seotud probleemidest sõltumata. Tarkvaraarendajatel on vaja konkreetseid ja praktilisi meetodeid, millega selliseid teste läbi viia juba arendamise käigus. Samuti on vaja automaatikasüsteeme

testida testijatel, vahel ka tellijal. Mida konkreetsem ja läbimõeldum on testimismetoodika, seda parem mõju ka tarkvara kvaliteedile.

Käesolev töö püüab neid lahendusi välja pakkuda aastatel 2002-2015 tööstusautomaatika valdkonnas töö autori poolt omandatud isiklike kogemuste põhjal, analüüsides ka teiste sama ala tarkvaraarendajate kogemusi ning kirjanduses levinud praktikaid.

1.2 Ülesande püstitus

Lühidalt võib käesoleva töö eesmärgid sõnastada järgmiselt:

1. Anda ülevaade PLC tarkvara testimisega seotud probleemidest ja iseärasustest
2. Leida praktilised meetodid PLC tarkvara testimiseks, mis sobiksid kasutamiseks ka keerulistes riistvaralistes süsteemides
3. Leida meetod ka PLC tarkvara automaattestide korraldamiseks
4. Pakkuda välja näpunäiteid PLC tarkvara testimisprotsessi hõlbustamiseks ja tarkvara kvaliteedi tõstmiseks
5. Rõhuasetus on praktilisusel, et välja pakutud meetodeid oleks tarkvaraarendajatel lihtne PLC tarkvara arendamise juures osaliselt või tervikuna kasutada

Töö uurimisküsimuseks on leida PLC tarkvara testimismeetod, mis võimaldaks teste läbi viia ilma kedagi või midagi ohustamata, riistvaraga või ilma, ka täpsete ja kiirete protsesside korral ning vajadusel täiesti automaatselt.

1.3 Metoodika

Laias laastus saab käesoleva töö valmimisprotsessi jagada järgmisteks töö autori poolt läbiviidavateks alamprotsessideks:

- Autori isikliku kogemuse analüüsimine tööstusautomaatika tarkvara valdkonnas testimisega seotud probleemide tuvastamiseks
- Probleemidele lahenduse leidmise vajalikkuse analüüsimine
- Varem autori enda poolt kasutatud testimismeetodite analüüsimine
- Antud valdkonna praktikate analüüsimine

- Tarkvara nõuete esitamise ja testimise üldiste põhimõtete uurimine
- Autori isiklikest töödest näidete väljavalimine töös kasutamiseks
- Analüüsi tulemusena leitud testimismeetodite läbiproovimine
- Uue testimismeetodi kirjeldamine
- Lõpptulemuse formuleerimine

1.4 Ülevaade tööst

Käesolev töö on jaotatud kaheks osaks.

Esimeses osas (peatükk 2) kirjeldatakse PLC tarkvara toimimisviisi ning selle arendamist, tutvustatakse tarkvara testimise temaatikat üldisemalt ning kirjeldatakse PLC tarkvara testimisega seotud probleeme. Probleemide kirjeldus annab ülevaate töö poolt otsitava testimismeetodi omaduste vajalikkusest. Ühtlasi seatakse raamid töö mahule ning laiendatakse töö eesmärke antud töö osas tutvustatud temaatika kontekstis. Esimeses osas antakse ülevaade ka PLC tarkvara testimise teadaolevatest praktikatest ning kirjeldatakse PLC arenduseks üldisemal tasandil sobivat tarkvaraarendusprotsessi metoodikat.

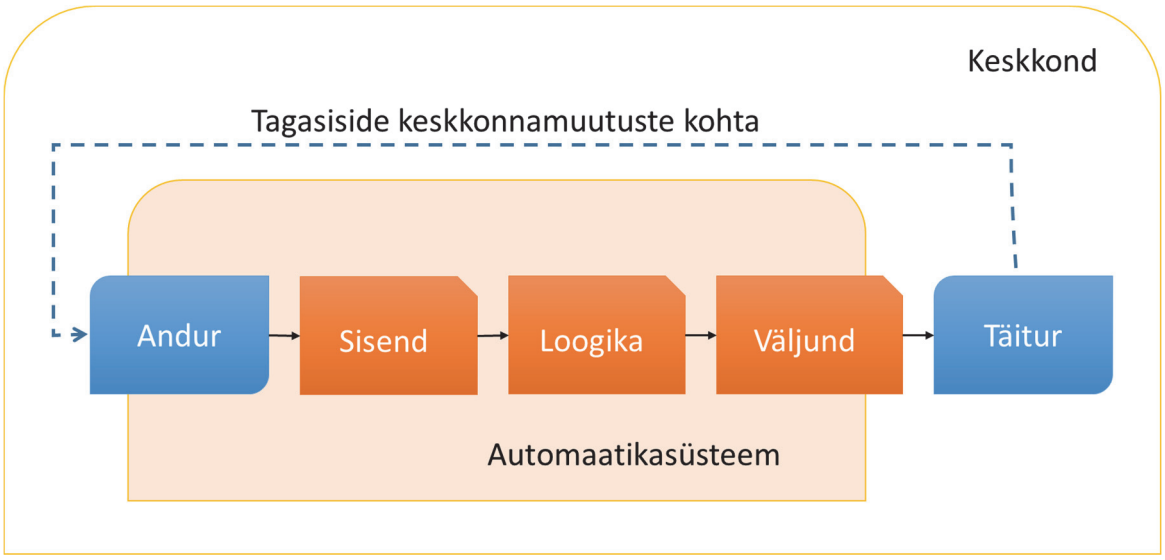
Teises osas (peatükk 3) kirjeldatakse kõigepealt ühte näidisprojekti. Näidisprojekti testjuhtusid illustratsioonina kasutades kirjeldatakse siis PLC tarkvara testimise erinevaid meetodeid vastavalt senisele praktikale, analüüsitakse nende sobivust töö eesmärkide täitmise kontekstis ning näidatakse töö poolt otsitava testimismeetodi vajalike omaduste mittetriviaalsust. Seejärel pakutakse välja uus testimismeetod, mis sobib ka teiste meetoditega testimata jäänud testjuhtude korral kasutamiseks ning automaattestide läbiviimiseks. Näidatakse, et uuel meetodil on olemas kõik omadused, mis on töö poolt otsitava testimismeetodi jaoks vajalikud. Lisaks tuuakse välja mitmeid tarkvara kvaliteeti tõstvad ja testimist hõlbustavad näpunäiteid, mida PLC tarkvara arendamisel jälgida.

2. Taust ja üldised põhimõtted

2.1 Üldised mõisted

2.1.1 Tööstusautomaatika seos keskkonnaga

Tööstusautomaatika juhtimissüsteem on lihtne sisend-väljund reaalajasüsteem. Sisendid on ühendatud anduritega, mille abil süsteem tajub keskkonda. Väljundid on ühendatud täituritega, mille abil süsteem mõjutab keskkonda. Automaatikasüsteemi ülesanne on muuta väljundeid vastavalt tema sees asuvale loogikale ning sisenditele ning teha seda reaalajas.



Joonis 1. Automaatikasüsteemi ja keskkonna seos

Tööstusautomaatika, automaatika, automaatikasüsteem, juhtimissüsteem ja ka lihtsalt juhtimine on kõik terminid, mida kasutatakse tihti segadusttekitavalt sünonüümidena. Nende all mõeldakse enamasti ainult neid seadmeid ja tarkvara, mille eesmärk on teisi riistvaraseadmeid juhtida. Juhitavad seadmed ise selle termini alla tavaliselt ei kuulu. Seetõttu loetakse automaatikasüsteemi sisse enamasti ka andurid, aga täitureid mitte. Andurid on vajalikud ainult automaatikasüsteemi jaoks, samas kui täitureid on põhimõtteliselt võimalik

kasutada ka käsitsi ehk mitteautomaatselt. See seos on näha ka joonisel 1. Joonisel näha olevaid nooli nimetatakse signaalideks.

Selle töö autori kogemus PLC tarkvara arendamisel on põhiliselt seotud Eesti toiduainetööstusega. Seetõttu on käsitletavad probleemid ja lahendused eelkõige sobivad selle valdkonnaga. Samas, reaalsuses on tööstusautomaatika üsna universaalne, mistõttu kõlbab sarnaseid lahendusi kasutada ka väga paljudes teistest tööstusharudes.

2.1.2 Testimise definitsioon

Testimist võib defineerida mitmeti. Selles töös käsitletakse testimist eelkõige kui tarkvara käivitamist vigade leidmise ning nõuetele vastavuse kontrollimise eesmärgil. Rõhuasetus on sõnal käivitamine ning seda eelkõige kontekstis, kus arendaja soovib näha, kuidas tema loodud programm käitub. Töö ei keskendu formaalsele verifitseerimisele või valideerimisele, vaid praktilistele meetoditele testimise läbiviimiseks.

Laiemas mõttes võib testimist käsitleda kui tarkvara omaduste hindamise protsessi, mis hõlmab endas kogu projekti elutsüklit ning kõiki osapooli (ka tellijat), kuid see mõiste käesoleva töö raamidesse ei mahu.

2.2 PLC riistvara tööpõhimõte

PLC ehk programmeeritav loogikakontroller on digitaalne tööstuslik arvuti, mida kasutatakse automaatikasüsteemides sisendite teisendamiseks väljunditeks. PLC jooksutab programmi, millel on ligipääs lihtsale digitaalsele liidesele, mis seotakse füüsilise maailmaga ehk riistvaraseadmetega läbi sisendite ja väljundite (IO). Riistvaraliselt on PLC enamasti tavaarvutist kompaktsem, lihtsam ning keskkonna suhtes vastupidavam. Kuna PLC peab olema äärmiselt töökindel, on riistvara arhitektuur tehtud võimalikult lihtsaks. [1]

PLC puhul räägitakse üldiselt kahte tüüpi IO-dest: digitaalsetest ja analoogsetest. Kokku on seega neli viisi PLC suhtlemiseks keskkonnaga: digitaalsisendid (DI), digitaalväljundid (DO), analoogsisendid (AI), analoogväljundid (AO).

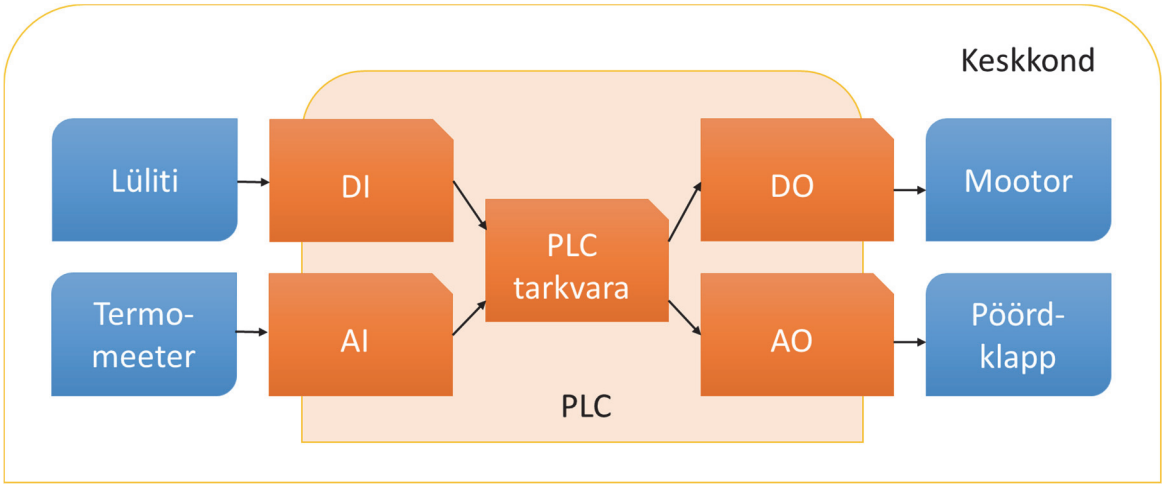
Digitaalsed on need IO-d, mis on diskreetse iseloomuga, ehk nii-öelda käima-seisma tüüpi, näiteks lüliti või relee. Nii keskkonnas kui PLC tarkvaras käsitletakse neid ühtemoodi –

midagi kas on või ei ole, `true` või `false`. Riistvaraseadmete jaoks tähendab digitaal-signaal üldiselt kas pinge olemasolu või selle puudumist elektrijuhtmes.

Analoog-IO-d on pideva iseloomuga, näiteks temperatuur või kaugus. Analoog-IO-sid käsitletakse PLC tarkvaras kui numbreid ning seda nimetust kasutatakse sellest hoolimata, et need numbrid ise on digitaalsed. Riistvaraseadmete jaoks tähendab analoog-signaal tavaliselt mingi elektrilise parameetri mõõtmist või mõjutamist juhtmes. Näiteks võib analoogsignaal olla 4-20 mA voolutugevust või 0-10 V pinget. See tähendab, et kahe seadme vahel tekitatakse vooluring, milles üks muudab, teine mõõdab kokkulepitud elektrilist parameetrit.

Kokkuvõtlikult toimib IO-süsteem järgmisel põhimõttel (vaata ka joonis 2):

- Andur tajub keskkonda ning muundab mõõdetud väärtuse mingiks elektriliseks signaaliks. Näiteks lüliti lülitamisel pingestatakse juhe (DI), temperatuuri mõõtmisel muutub takistus (AI).
- Sisend mõõdab seda elektrilist signaali ning muundab selle digitaalseks, näiteks pinge olemasolu loogiliseks üheks (DI), mõõdetud takistuse täisarvuliseks numbriks (AI).
- PLC tarkvara töötleb seisendist tulnud digitaalset signaali ning edastab tulemuse väljundile uue digitaalse signaalina.
- Väljund muudab digitaalse signaali elektriliseks, näiteks pingestatakse juhe loogilise ühe olemasolul (DO) või hoitakse juhtmes kindlat voolutugevust vastavalt täisarvulisele numbrile (AO).
- Täitur reageerib sellele elektrilisele signaalile. Näiteks pingestatud juhtme korral käivitatakse mootor (DO), voolutugevusest sõltuvalt reguleeritakse klappi (AO).



Joonis 2. PLC sisendite ja väljundite ühendumise põhimõte

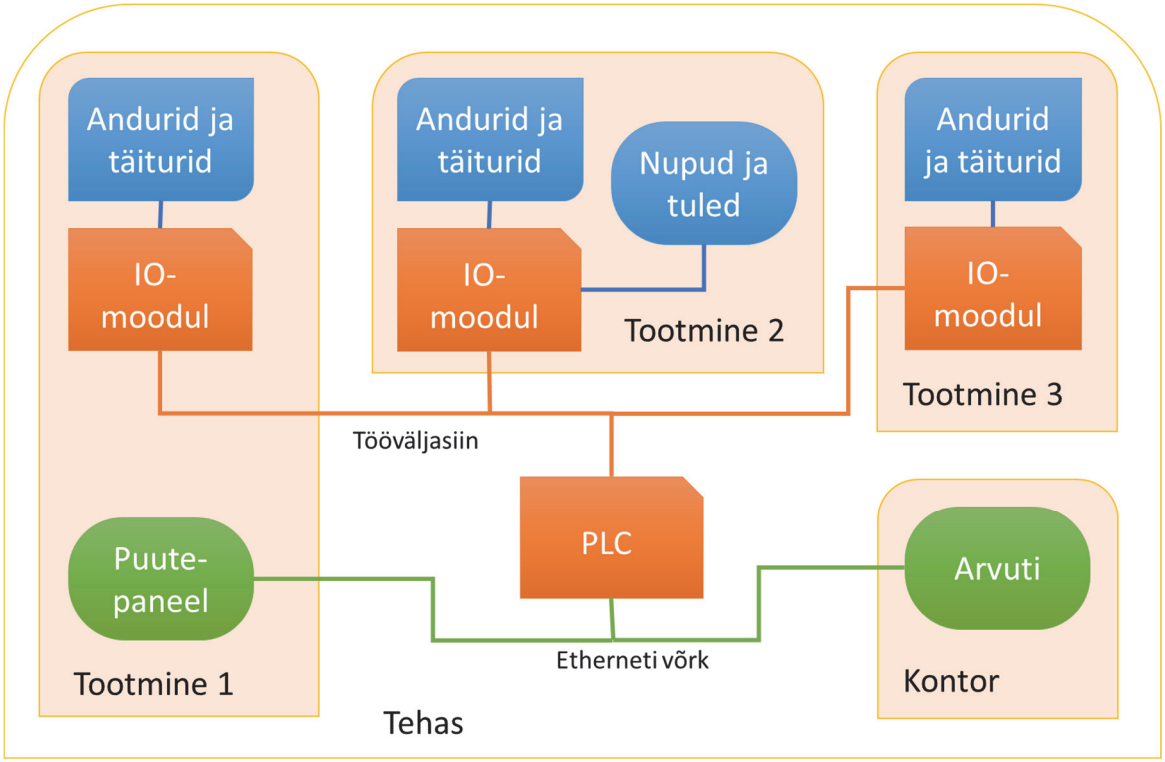
Lihtsamate automaatikasüsteemide puhul ongi PLC-ga samas korpuses mingi kindel kogus IO-sid. Riistvaralises mõttes tähendab see klemmlaudu, mille külge saab juhtmeid kinnitada. Elektrisignaalid nendes klemmides konverteeritakse automaatselt digitaalseteks signaalideks PLC tarkvara jaoks ja vastupidi. Selliseid süsteeme kasutatakse siis, kui juhitavaid seadmeid pole palju ning need asuvad kõik võrdlemisi lähestikku. Kui seadmeid on rohkem, saab mõningaid PLC mudeleid laiendada lisa-IO-moodulitega. Tinglikult öeldes saab klemmlauda sel viisil pikendada.

Keerulisemate automaatikasüsteemide korral PLC küljes olevatest IO-dest aga ei piisa, sest ühte kohta kõigi signaaljuhtmete kokku vedamine ei pruugi olla otstarbekas. Sellisel juhul kasutatakse jagatud või hajutatud lähenemist, kus PLC paikneb ühes kohas, kuid IO-moodulid laiali tööstusseadmete läheduses. PLC ja IO-moodulite vahel on sel juhul kasutuses mõni digitaalne kommunikatsiooniliides andmete vahetamiseks (nn tööväljasiin, näiteks *ProfiBus*, *DeviceNet*, *AS-i*, *CAN* või *Ethernet*). See andmevahetus lisab signaalide levimisele küll teatavat viivitust, aga muus osas toimib kõik sarnaselt olukorrale, kus IO-d on PLC korpusesse integreeritud. Selline hajus lahendus on näha ka allpool joonisel 3.

Tavaliselt on üks osa PLC IO-st eraldatud ka kasutajaliidese jaoks. Lihtsama automaatika-süsteemi puhul võivad selleks olla nupud ja tuled, keerulisemate süsteemide puhul kasutatakse aga juba graafilist/elektroonilist kasutajaliidest puutepaneeli, arvuti või tahvli näol, mida tuntakse ka SCADA lühendi all. Kasutajaliidese puhul kasutatakse kolme põhilist lähenemist: [2]

- Üks suur kasutajaliides PLC küljes või lähedal, kust juhitakse kogu protsessi
- Eraldiseisvad kasutajaliidesed, mis on hajutatud vastavalt tootmisprotsessi vajadustele
- Internetipõhine lähenemine, kus PLC pole enam otseühenduses kasutajaliidesega, vaid serveritega. See uus lähenemine on tihedalt seotud nn asjade interneti (*IoT*) arengutega.

Paljudes reaalsetes süsteemides kombineeritakse erinevaid lähenemisi. Näiteks ka joonisel 3 toodud automaatikasüsteemis, kus esimeses tootmise osas on puutepaneel, teises osas füüsilised nupud ja tuled, kolmandas osas pole üldse kasutajaliidest ning kontoris on arvutipõhine kasutajaliides.



Joonis 3. Tüüpilise hajusa automaatikasüsteemi ülesehitus

Ettevõtteid, mis erinevaid PLC-põhiseid lahendusi toodavad, on palju. Näiteks Siemens, Mitsubishi, Omron, Beckhoff, Rockwell Automation, Toshiba. Suur osa nendest ettevõttest toodab ise ka andureid ja täitureid, lisaks ka kasutajaliidese lahendusi.

Käesoleva töö autor on eelkõige kokku puutunud Siemensi ja Omroni PLC-de tarkvara arendamisega. Käesoleva töö näited on toodud Omroni PLC platvormi baasil, sest Siemensi kõrval kasutatakse Omronit Eestis võrdlemisi palju, samas on Omroni kohta eestikeelsete näidete leidmine Eesti turu väiksuse tõttu keerulisem, mis annab käesolevale tööle võimaluse ka seda tühimikku täita. Erisused suuremate platvormide vahel on olemas, kuid käesoleva töö kontekstis mitte märkimisväärsed.

2.3 PLC tarkvara tööpõhimõte

PLC tarkvarakeskkonna üldpõhimõte on sama nagu laialt levinud arvutitel – protsessor täidab masinkoodis ette antud käske ning opereerib mäluga.

2.3.1 PLC tarkvara mälu

PLC mälu jaotatakse tavaliselt piirkondadeks või registriteks. Sõltuvalt PLC tootja poolsest lahendusest võib eristada järgmisi mälupiirkondi: [3]

- Programmiloogika mälu, kus hoitakse programmi, mida PLC protsessor täidab
- Mittepüsiv andmemälu, mille sisu kaob voolu kadumisel
- Püsiv andmemälu, mille sisu säilib ka voolu kadumisel
- IO ühenduste mälu, mille kaudu toimub interaktsioon IO-signaalide saatmiseks ja vastuvõtmiseks
- Taimerite, loendurite mälu
- Automaatselt jagatav mälu, mida PLC või selle arenduskeskkond ise jaotab näiteks funktsiooniplokkide või muude süsteemide kasutuseks vastavalt vajadusele
- Struktureeritud andmemälu või andmeblokkide mälu, mida arendaja ise jaotab piirkondadeks vaastavavalt vajadusele
- Mäluühiku ehk -pesa suurusest sõltuv mälu, näiteks bitimälu, baidimälu, sõnamälu (kaks baiti)

Ühele mäluühikule ehk mälupesale otseviitamine toimub programmis enamasti läbi kolmest osast koosneva viitamissüsteemi:

- Mälupiirkond või register
- Mäluühiku suurus
- Mälu aadress ehk järjekorranumber selles piirkonnas või registris

Kaks näidet PLC mälule viitamisest:

- `H20.7` – Omroni PLC, „H“ viitab püsivale mälupiirkonnale, „20“ viitab järjekorras 20-ndale sõnale, „.7“ viitab selle sõna 8-ndale bitile ehk kahendmuutujale (*boolean*).
- `PIW500` – Siemensi PLC, „I“ viitab sisendite mälupiirkonnale, „P“ viitab perifeersele sisendite mälupiirkonnale (ühendatud tööväljasiini kaudu), „W“ tähendab ühte sõna ning „500“ viitab selle mälupiirkonna 500-ndale baidile ehk 250-ndale sõnale.

Need mäluviidad ei määratle mälupesas olevate andmete kuju. Sõltuvalt PLC toest ja mälupesa suurusest või järjestikkuste pesade arvust võivad andmed olla kahendmuutujad, täisarvud, pikad täisarvud, ujukomaga arvud, ASCII-formaadis tähed, jne. Andmete tüüp

määratletakse kasutaja jaoks muutujanimede kasutuselevõtuga ning programmi jaoks vastavate käskude kasutamisega.

Muutujate kasutamine tähendab PLC tarkvara puhul seda, et kirjeldatakse ära, mis nime ja andmetüübiga mõni konkreetne mälupesa on. See tähendab muuhulgas, et programmikoodis hoitakse mälupesa viidet, mitte muutuja nime. Muutuja mälupesa muutudes jääb programmikoodis salvestatud mälupesa seega alles ning kaotab lihtsalt nime. Muutujad on mõeldud ainult arendaja töö hõlbustamiseks, nende kasutamine pole kohustuslik.

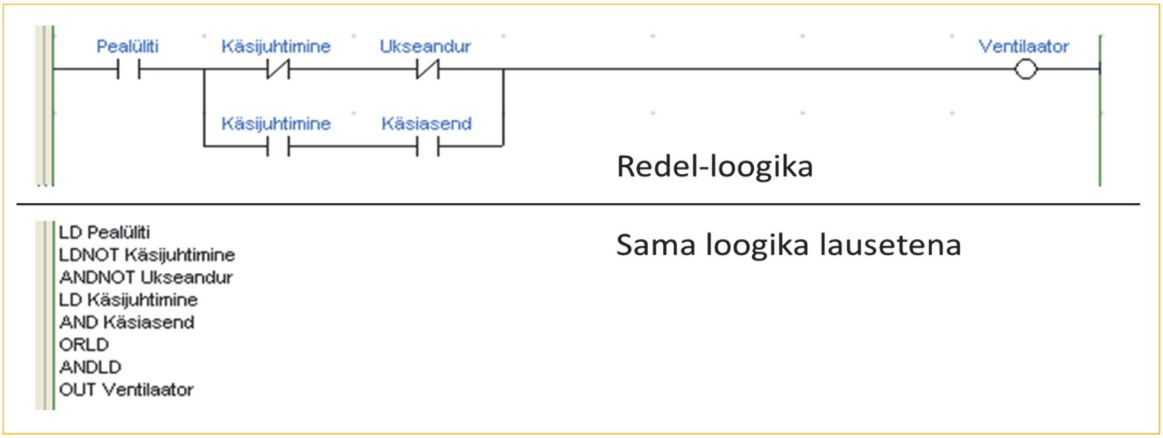
Mälu võib sõltuvalt PLC mudeli hinnast olla piiratud ressurss. Mälu maht võib ulatuda mõnest kilobaidist mõne megabaidini.

2.3.2 PLC tarkvara programmeerimiskeel ja struktuur

IEC 1131-3 on rahvusvaheline standard PLC tarkvara programmeerimiskeelte jaoks. Seal pakutakse välja mitu programmeerimiskeelt, mida PLC tootjad üldiselt ka realiseerivad. See võimaldab ühel arendajal mitme tootja PLC peale tarkvara luua ning iga projekti jaoks sobiva programmeerimiskeele valida PLC tootjast sõltumata. Kolm levinumat keelt on järgmised: [4]

- *Ladder diagram* (LD) – redel-loogika. See meetod pakub graafilise liidese, kus PLC loogika on kujutatud sarnaselt relee-loogikale. Sisendid on vasakul, väljundid paremal. Nende vahele paigutatud jooned on justkui juhtmed, mida mööda signaal liigub sisendist väljundisse. Virtuaalsed lülitid lubavad signaalil liikuda ainult sinna, kuhu vaja. Redel-loogika meetod on levinuim PLC programmeerimise meetod.
- *Function Block Diagram* (FBD) – funktsiooni-plokk-skeem. See meetod pakub samuti graafilise liidese, mis erineb redel-loogikast selle poolest, et lülitite asemel kasutatakse signaali juhtimiseks loogilisi tehteid teostavaid funktsiooniblokke.
- *Structured Text* (ST) – struktureeritud tekst. See meetod on kõige sarnasem traditsioonilisele tarkvaraarendusele ning on ühtlasi ka kõige võimsam, võimaldades sisendeid töödelda ja väljundeid mõjutada tekstipõhiste käskude abil. Käskude valik ja keelestruktuuride tase sõltub väga PLC tootjast ning konkreetsest mudelist, ulatudes lihtsatest loogikakäskudest kuni Basicu või Pascali sarnase keeleni välja. Selle meetodi puhul on erisused eri tootjate puhul märkimisväärsed.
- *Instruction List* (IL) – lausete nimekiri. Samuti tekstipõhine programmeerimiskeel, kuid oluliselt lihtsam kui ST.

Tüüpilist PLC programmi on võimalik struktuuriliselt jagada erinevateks osadeks (sektsioonideks, plokkideks), millest igaüks võib kasutada ka erinevat programmeerimiskeelt. LD ja FBD skeeme on võimalik automaatselt konverteerida IL-ks ehk lausete nimekirjaks. LD ja vastavate IL lausete võrdlust võib näha joonisel 4.



Joonis 4. Redel-loogika võrdlus samaväärse lausete nimekirjaga PLC tarkvaras

PLC programmi struktureerimisel kasutatakse lisaks sektsioonidele ka funktsiooniplokke. [5] Neid võib lihtsustatult jaotada kolme kategooriasse:

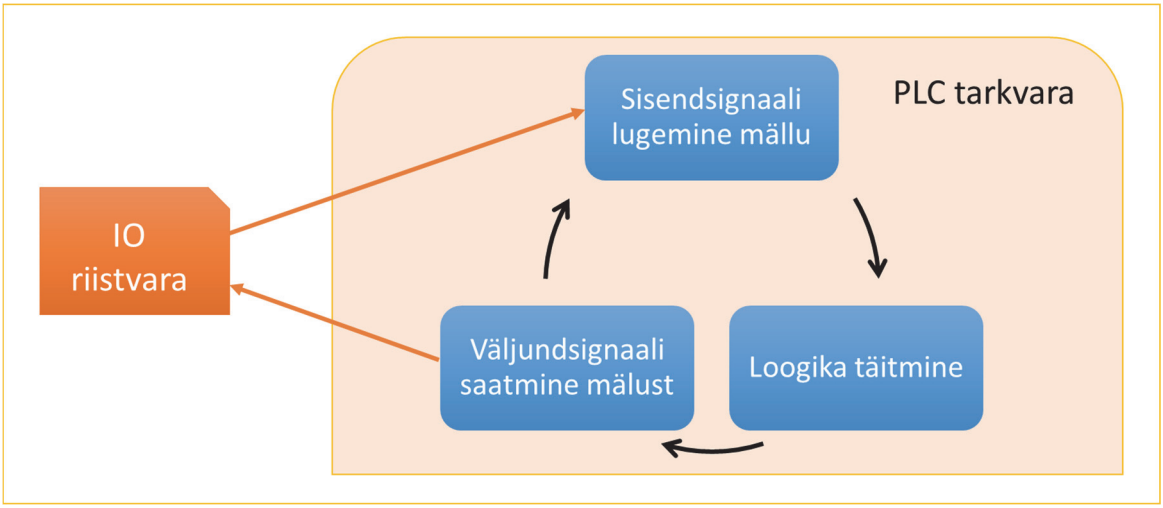
- Tavalised funktsiooniplokid võimaldavad mingit loogikat käivitada vaid vajadusel.
- Sisendite ja väljunditega funktsiooniplokid mõjutavad ploki väljundeid vastavalt sisenditele.
- Andmetega funktsiooniplokid mõjutavad samuti ploki väljundeid vastavalt sisenditele, kuid neile on määratud ka eraldi mälupiirkond, kus hoitakse ühe objekti andmeid mõnevõrra sarnaselt objektorienteeritud lähenemisele.

PLC nimetuse alla loetakse vahel ka arvutisüsteeme, kus seadmete juhtloogikat jooksutatakse mõnes levinumas keeles nagu C++ või Java. Selliseid lahendusi käesolev töö ei vaatle.

2.3.3 PLC tarkvara töötsükkel

PLC tarkvara lihtsustatud töötsükkel on järgmine (vt ka joonis 5): [6]

- Sisendsignaalide lugemine riistvarast vastavatesse mälupesadesse
- Loogika täitmine ühe korra alates algusest kuni lõpuni
- Väljundsignaalide edastamine riistvarale vastavalt mälupesadele



Joonis 5. PLC tarkvara töötsükkel

Seda tsüklit korratakse regulaarselt, üldiselt nii kiiresti kui võimalik. Kahe tsükli alustamise vahelist aega mõõdetakse millisekundites, sobiv väärtus sõltub väga konkreetse projekti vajadustest. Teatud piirides annab kahe tsükli alustamise vahelist aega ka käsitsi määrata.

2.3.4 PLC tarkvara arendamise tööriistad

Üldiselt on iga PLC tootja loonud oma seadmete programmeerimiseks sobiva arenduskeskkonna. Enamus neist on üksteisega sarnased, pakkudes võimalusi automaatika-süsteemi struktuuri kirjeldamiseks ja seadistamiseks, PLC komponentide parameetrite muutmiseks ning programmiloogika kirjutamiseks. Arenduskeskkonna põhiülesannete hulka kuulub ka valmisarendatud tarkvara koodi kompileerimine ja laadimine PLC mällu ning töötava PLC loogika diagnoosimine.

Olulisemad tööriistad tarkvara arenduse ja diagnostika seisukohalt Omroni PLC puhul: [7]

- Koodi kontrollimine – tööriistad, mis kontrollivad programmi loogika semantilist korrektsust.
- Reaalajas muutmine – tööriistad, mis võimaldavad programmi loogikat muuta ilma PLC tööd peatamata.
- Mälu jälgimine – tööriistad, millega saab töötava PLC mälupesade väärtusi lugeda ja mõjutada.
- Mälupesade kasutuse analüüs – tööriistad, millega saab jälgida mälupiirkondade koormatust ning mälupesade ristikasutust.

- Reaalajas loogika vaatamine – tööriistad, mis võimaldavad programmiloogika vaates visuaalselt mälupesade väärtusi kuvada. LD ning FBD meetodite puhul värvitakse ära need jooned, mida mööda loogiline signaal justkui liigub.
- Simulaatorid – tööriistad, mis võimaldavad PLC tarkvara käivitada ilma seda riistvarasse laadimata.
- Kasutajaliidese ehk SCADA loomise vahendid – tööriistad, mille abil saab käsitsi luua liideseid nii automaatikasüsteemi juhtimiseks kui diagnostikaks.

2.4 Tarkvara nõuete esitamine

Nagu varem kirjeldatud, eksisteerib automaatikasüsteem selleks, et mõjutada keskkonda (vt jaotis 2.1.1). See osa keskkonnast, mida süsteem tajub ning mõjutab, on selle konkreetse süsteemi rakendusvaldkond – valdkond, milles elab ja tegutseb süsteemi tellija. Selleks, et oleks võimalik edukalt süsteemi arendada, on vaja võimalikult hästi mõista rakendus-valdkonna taustsüsteemi ning konkreetseid probleeme, mida süsteem lahendama peab.

Oluline on eristada probleemi ja selle lahendust. Probleem paikneb rakendusvaldkonnas, selle lahendus aga süsteemis. Probleem vastab küsimusele „mis“, lahendus küsimusele „kuidas“. Väga tihti keskendutakse eelkõige lahenduse väljamõtlemisele, sest seda on vaja niikuinii teha. Seetõttu juhtubki, et loodud tarkvara ei vasta rakendusvaldkonna vajadustele ning tuleb kõrvale heita või ümber teha. Süsteemi rakendusvaldkonnast kui probleemi kontekstist on vaja aru saada enne, kui asuda probleemi lahendama läbi tarkvara disaini ja realisatsiooni.

Rakendusvaldkonnale olulise tähelepanu pööramine tähendab seda, et tuleb kirjutada täpseid kirjeldusi. Kui automaatikasüsteemile keskenduda kui sisend-väljund-süsteemile, võib tekkida kiusatus piirduda süsteemiprotsesside kirjeldamisega – kuidas sisenditest saavad väljundid. Süsteemi rakendusvaldkond on kõik see, mis paikneb teiselpool neid sisendeid ja väljundeid – selle kirjeldamine annab täpse seletuse, mis on probleem, miks midagi vaja on, kes on asjaga seotud, jne. On välja pakutud head näpunäited selliste kirjelduste vormistamiseks. [8]

- Kirjelduste jaoks tuleb mõisted ja terminid määratleda kas inimkeeles mitteformaalse nimetusena või formaalse definitsioonina. Oluline on, et need nimetused ja definitsioonid kirjeldaksid mingit nähtust või objekti võimalikult üheselt, et neid kasutavad teised kirjeldused oleksid konkreetsed ning selgelt mõistetavad. Jälgida

tuleb, et selliste nimetuste ja definitsioonide osakaal kogu kirjelduses ei oleks liiga suur. Tööstusautomaatikas on selliseid taaskasutatavaid mõisteid palju, sest tihti opereeritakse mitmete sarnaselt toimivate objektidega (pump, klapp, silinder). Nimetusele või definitsioonile viidates tuleks kasutada mingit selget märgistust, mis aitaks eristada neid teistest üldise tähendusega sõnadest.

- Kirjeldused tuleb tükeldada mõistlikult, et nad ei sisaldaks endas üleliigset informatsiooni, mis konkreetse probleemi lahenduse jaoks vajalik ei ole, ega ka liiga vähe informatsiooni, mis takistaks probleemi tausta ja probleemi sisu mõistmast. Tarkvara realisatsioonist tuttavat moodulitesse jaotamise põhimõtet tuleb rakendada ka kirjelduste puhul. Tükeldamisele aitab kaasa probleemi ja selle osade ruumilise ning ajalise ulatuse mõistmine ning piiramine.
- Kirjeldus peaks olema ümberlükatav. Kui nimetusi ja definitsioone ei ole enamasti mõistlik ümber lükata, siis ülejäänud kirjeldused peaksid olema selliselt üles ehitatud, et nende vaidlustamine on mõistlik ja lihtne. See võimaldab analüüsida nende tõesust.
- Kirjeldus võib olla ka ligikaudne visand. Kirjelduste väljamõtlemise algfaasis on tihti oluline mingi üldise pildi kirjasaamine, sest põhjalik vormistamine võib olla ennatlik või võib selle töö käigus midagi olulist meelest minna. Samas ei tohiks lõplike kirjelduste hulgas selliste visandite osakaal suureks jääda, sest need on üheseks mõistmiseks enamasti liiga pealiskaudsed. Ideaalis peaks visandid kirjeldamise käigus asendama põhjalike, ümberlükatavate kirjeldustega.

Nõuded on kirjeldused, mis kirjeldavad probleemi. Nõuded ei kirjelda lahendust, vaid nendes olevat informatsiooni kasutatakse lahenduse loomisel. Nõuded kirjeldavad, milliseid omadusi on vaja saavutada. Testimisel kontrollitakse, kas need omadused on saavutatud.

Spetsifikatsioon kirjeldab liidest probleemi ja lahenduse vahel. Sinna kuuluvad ainult sellised elemendid rakendusvaldkonnast, mis saavad tinglikult kuuluda ka realiseeritud süsteemi. Näiteks lause „Operaatori käsu peale peab mahuti tühjaks valguma“ on vajalik ja arusaadav nõue, kuid kui süsteemis pole ühtegi andurit mahuti vedelikutaseme registreerimiseks, ei ole see sobiv spetsifikatsioon. Selle asemel peaks spetsifikatsioonis olema lause „Nupu vajutamine avab mahuti väljalaskeklapi“.

On oluline välja tuua, et tööstusautomaatikas on ka eelmises lõigus välja toodud olukord probleem, mitte lahendus. Probleem üldisemalt on see, et operaatoril on vaja mahuti tühjaks lasta, probleem detailsemalt on see, et nupu vajutamine peab kuidagi väljalaskeklapi avama.

See aga ei ole lahendus. Lahendus on see, et nupp ja väljalaskeklapp ühendatakse automaatikasüsteemiga ning süsteemis paiknev tarkvara registreerib nupuvajutuse signaali ning annab mingi loogika alusel omakorda signaali väljalaskeklapile, et see avaneks. Tööstusautomaatika tarkvaraarendajate jaoks on automaatikasüsteemi rakendusvaldkonnaks seega enamasti tootmisseadmete juhtimine, mitte tootmine. Tootmine on osa taustsüsteemist, sest toodavad seadmed, mida automaatikasüsteem juhib.

See, millised konkreetsed meetodid valida probleemi kirjeldamiseks, sõltub probleemist endast. Lihtsate probleemide puhul piisab ka lihtsatest kirjeldustest, kuid suuremate probleemide puhul tuleb hoolikalt valida sobiv meetod probleemile lähenemiseks koos vastavate skeemide ja tabelitega. Teiste poolt välja pakutud kirjeldusmeetodid on enamasti seotud konkreetsete probleemiraamistikega. Kui probleem või selle osa mahub mingisse raamistikku, võib probleemi analüüsiks ja kirjelduste koostamiseks kasutada raamistikuga kaasnevat meetodit (näiteks Ivar Jacobsoni tuntud kasutusjuhu-põhine meetod).

2.5 Testimise üldised põhimõtted tarkvaraarenduses

Tarkvara testimise põhieesmärk on leida üles vigu mitte ainult programmi realisatsioonis, vaid kogu tarkvara arendusprotsessis tervikuna, võrreldes reaalse tarkvara töö tulemusi seatud nõuetega. Olulisemad näited vigadest, mis PLC tarkvara testimise käigus välja tulla võivad:

- Vead tarkvara realisatsioonis – tarkvara ei käitu nii, nagu disainis ette nähtud, ega võimalda seetõttu nõudeid täita.
- Vead tarkvara disainis – loodud disain ei võimalda seatud nõudeid täita.
- Vead nõuete vormistamises – nõuded on ebatäpsed või ebaselged.
- Vead nõuete funktsionaalsuses – püstitatud nõudeid pole võimalik täita või on nõuete täitmisel saadav tulemus mitterahuldav tööstusprotsessi tehnoloogilises mõttes.
- Vead automaatikaseadmete disainis – komplekteeritud riistvarasüsteem võib olla väga keeruline ning puudujäägid võivad selguda alles reaalsete testide käigus. Näiteks võib olla mootori jaoks valitud ebasobiv sagedusmuundur või võib olla paigaldatud liiga vähe IO-mooduleid.

- Vead automaatikaseadmete paigalduses – komplekteeritud automaatikasüsteemis võib esineda mitmeid ajutisi puudujääke, mis tulevad välja ainult tarkvaraga testimise käigus. Tööväljasiini ning signaalkaablite paigalduse kvaliteet on üks olulisimaid tegureid.
- Vead tootmisseadmete disainis või kvaliteedis – tootmisseadmete disaini- või koostevead ning jõukaablite paigalduse vead võivad selguda samuti alles käivitamise käigus. Selliste vigade tõttu võib olla võimatu nõudeid täita.

Ülalmainitud riistvara puudutavad vead ei ole tinglikult öeldes üldse seotud PLC tarkvaraga vaid automaatikasüsteemi ja selle keskkonnaga kui tervikuga. PLC tarkvara testimine on alati tihedalt seotud automaatikaseadmete ning tootmisseadmete testimisega ka selles mõttes, et tihti tuleb riistvaravigasid lahendada või kompenseerida tarkvaraliselt.

Üks eduka testimise aluseid on põhimõte, et testimist peaks käsitlema kui järjepidevat planeeritud protsessi tarkvara kvaliteedi tõstmiseks. See tähendab, et testimise vajadusega arvestatakse projekti algusest kuni lõpuni. Automaatika- ning tootmisseadmete iseärasusi tuleb arvesse võtta juba nõuete püstitamisel, et need nõuded oleksid ka võimalikult hästi testitavad. Testimist tuleb planeerida varakult, et seadmete iseärasusi saaks testimise kontekstis vajadusel ka tarkvaraarenduse käigus arvesse võtta, et pärast mingi tarkvara osa valmimist oleks see realisatsioon paremini testitav. Testimiseks on vaja varuda aega. Mida etteplaneeritum ning süstemaatilisem on testimine, seda parem on ka tulemus (seda rohkem vigu avastatakse). [9] [10]

Tarkvara testimist võib tinglikult jaotada erinevateks etappideks. Teadlikust jaotamisest on palju kasu, kuna see aitab testimist paremini ja süstemaatiliselt planeerida. Oluline on aga meeles pidada, et igasugune testimine on vajalik tarkvara kvaliteedi tõstmiseks, mitte etappide või plaani täitmiseks. Seetõttu tuleb tihti mitut etappi korraga täita või hilisemast etapist tagasi varasemasse etappi liikuda.

Testimist võib etappideks jaotada ka vastavalt kogu projekti arenduse etappidele, alustades üksikute tarkvaramoodulite testimisest ja moodulite omavahelise koostöö testimisest kuni terviksüsteemi testimiseni. Testimist võib jaotada ka selle järgi, kes testimist läbi viib. [11]

Testimine koosneb üksikute testide läbiviimisest. Testi vaadeldakse kui komplekti ühest või mitmest testjuhust, mille raames programm käivitatakse kindlate sisendandmetega ning seejärel võrreldakse programmi tegelikke väljundandmeid oodatavate väljundandmetega. Üks

olulisemaid testimisprotsessi osasid ongi nende testijuhtude genereerimine. Selleks leitakse konkreetne eesmärk, mida soovitakse täita (nt kontrollida vastavust nõuetele), ning leitakse sobivad sisend- ja väljundandmed, mis võimalikult palju olukordi võimalikult väheste testide arvuga läbi lubaksid proovida. Testide väljamõtlemine on keeruline protsess, milleks võib olla tööstusautomaatikas vaja kaasata ka tellija esindajaid ning eksperte. [12] [13]

Testjuhud ja testimisprotsess tervikuna on soovituslik ka dokumenteerida, eriti kui ka nõuded on sobivalt dokumenteeritud, sest see lihtsustab testi sisend- ja väljundandmete kirjeldamist. Dokumenteerimine aitab kaasa testimise järjepidevusele ning süstemaatilisusele ning võimaldab seeläbi leida rohkem vigu.

Palju rutiinset tööd sisaldavaid või muul põhjusel keerulisi teste on soovituslik läbi viia automatiseeritult, nii et sisendandmeid söödab ette ning väljundandmeid kontrollib üle teine programm. Eriti oluline on selline testimismeetod regressioonitestimisel, kus teste viiakse läbi korduvalt, et välja selgitada, ega tarkvara muudatused pole varem testitud programmis uusi vigu tekitanud. Automaattestimine on väga hea testimismeetod seetõttu, et vähendab oluliselt inimese poolt tekkida võivate metoodiliste või muud sorti vigade hulka, ning aitab kaasa testimise süstemaatilisuse tagamisele. Ühtlasi on see sisuliselt ainus viis, kuidas läbi viia teste, mille nõuetele vastavust on keeruline inimesel endal kontrollida näiteks kiirus- või täpsusnõuete tõttu. [14]

Lisaks testimisele kui tarkvara käivitamisele on tarkvara kvaliteeti võimalik tõsta veel mitmete meetodite abil, näiteks lähtekoodi läbivaatus, formaalne tõestamine, küsitlemine, vastavus standarditele, mudelipõhine testimine, jne. Testimisprotsessi läbiviimise hõlbustamiseks on loodud ka mitmeid standardeid, raamistikke, metoodikaid. Nende meetoditega on soovituslik igal tarkvaraarendusprotsessiga seotud inimesel tutvuda.

2.6 PLC tarkvara testimise probleemid

PLC tarkvara testimisega selle käivitamise teel on seotud mitmed probleemid, mida tarkvaraarenduses tihti ette ei tule. Peamiselt on need probleemid tingitud PLC riistvara olemasolust, kuid väiksemal määral ka arenduskeskkonna ja programmeerimiskeele iseärasustest. Nende probleemide analüüs annab hea ülevaate sellest, miks on käesoleva töö poolt otsitava testimismeetodi eeldatavad omadused head ja vajalikud.

2.6.1 Piiratud arenduskeskkond

PLC tarkvara arenduskeskkonna poolt pakutavad tarkvara arendus- ja silumisvõimalused on levinud arenduskeskkondadega võrreldes üldjuhul piiratumad. Tabel 1 võrdleb mõningate *Eclipse IDE for Java EE* arenduskeskkonna tuntud omaduste olemasolu Omroni PLC arenduskeskkonna *CX-Programmer 9.0* omadustega. [7]

Omadus	**Eclipse**	**CX-Programmer**
Koodi analüüsimine vigade leidmiseks kompileerimisel	Jah	Jah
Lihtsate vigade visuaalne esiletoomine jooksvalt	Jah	Piiratud
Programmi käivitamine	Jah	Jah
Intelligentne koodi lõpetamine (*IntelliSense*)	Jah	Piiratud
Käivitatud programmi peatamine murdepunktidega ning samm-sammuline läbikäimine	Jah	Simulaatoris
Käivitatud programmi muutujate väärtuste lugemine otse programmikoodist	Jah	Jah
Käivitatud programmi loogika muutmine ilma programmi seiskamata	Jah	Jah
Silumis-konsool teadete jaoks	Jah	Ei
Raamistikud automaattestide läbiviimiseks	Jah	Ei
Integratsioon versioonihaldustarkvaraga	Jah	Ei
Laiendatavus pistikprogrammidega	Jah	Ei
Võimalus arenduskeskkonda vahetada	Jah	Ei
Võimalus arendusarvuti platvormi valida	Jah	Ei

Tabel 1. Tarkvara arenduskeskkondade Eclipse Java EE ja Omron CX-Programmer omaduste olemasolu võrdlus

Koodist kompileerimisvigade leidmine ja loetlemine on *CX-Programmeris* olemas, samuti kuvatakse visuaalselt ka mõningaid elementaarseid koodivigu. Samuti on võimalik laadida programm käivitamiseks üles reaalsesse PLCsse või simulaatorisse. Simulaatoris on võimalik ka loogikat pausile panna, samm-sammult läbi käia ning murdepunktidega peatada. Simulaator pole siiski võrreldav näiteks tavalise Java rakenduse käivitamisega testimise eesmärgil, sest simulaator jätab ühendamata kogu PLC IO osa, samas kui Java rakendust on

enamasti võimalik siluda oma loomulikus keskkonnas või täieliku funktsionaalsusega test-keskkonnas.

Puudusi leidub veel. *CX-Programmeris* puudub võimalus programmi poolt midagi konsoolile väljastada. Ainus võimalus töötava programmi jälgimiseks on kasutada mälumuutujate jälgimise vahendeid (vt jaotis 2.3.4). *CX-Programmeri* intelligentne koodi lõpetamine on piiratud mälu muutujanimede väljapakkumisega. Programm töötab ka ainult Windows-platvormiga arvutites.

Need kitsendused puudutavad Omroni PLC tarkvara arenduskeskkonda, kuid kehtivad üldiselt enamuste levinud PLC tootjate kohta. Kuna arendaja on kohustatud kasutama tootja enda poolt valmistatud arenduskeskkonda, tuleb soovitud keskkonna kasutamiseks vahetada ka PLC tootjat.

Oluliseks probleemiks on ka igasuguse automaattestide tegemise raamistiku puudumine.

2.6.2 Tööstusautomaatika kui reaalajasüsteem

Tõsiasi, et tööstusautomaatika süsteemid on reaalajasüsteemid, väärib eraldi välja toomist. See tähendab lihtsustatult, et oluline pole mitte ainult andurite põhjal täiturite mõjutamise loogika, vaid ka selle protsessi ajaline korrektsus.

Mälumuutujate jälgimise kaudu on tihti võimalik saada päris hea ülevaade mingi programmi toimimisest, eriti kui muutujate väärtused kuvatakse otse LD diagrammil. Aga kui selle programmi protsessid toimuvad väga kiiresti, on neid muutusi silmaga raske kui mitte võimatu jälgida. Kiirete protsesside puhul ei ole tihti võimalik tulemust hinnata isegi töötavaid seadmeid jälgides.

Aeg on oluline ka selles mõttes, et tööstusprotsessides on tihti vaja täitureid mõjutada täpselt õigel ajal, ei varem ega hiljem. Süsteemi toimimiseks etteantud ajalisi parameetreid on väga raske üle kontrollida lihtsalt jälgimise teel. Isegi kui PLC arenduskeskkond võimaldaks programmi ajutist peatamist ka väljaspool simulaatorit, ei pruugi see olla riistvaraseadmete tõttu võimalik, sest need jätkavad reaalajas töötamist.

2.6.3 Puuduv riistvara

Kui kogu automaatikasüsteemi riistvara on tehases või osaliselt kasvõi töölaua peal komplekteeritud, on PLC koodi käivitamine ning loogika abil andurite seisundi lugemine ja täiturite mõjutamine täiesti võimalik. Väga tihti on aga vaja PLC programmi testida ilma riistvarata, sest tehase riistvara pole veel paigaldatud, või on vaja tarkvara arendada veel kontoris, kus seda riistvara ei saagi olema.

Lihtsamal juhul tähendab see mõne üksiku seadme puudumist, näiteks mõni andur või mootor. Tarkvara tuleb siis käivitada nii, et loogika toimiks ka nende seadmete puudumisest hoolimata. Kogu riistvarakonfiguratsioon võib saada komplekteeritud alles objekti kasutuselevõtuks. Näiteks võivad olla mõned täiturid kasutuses väljavahetamist vajava automaatikasüsteemi poolt ning need ühendatakse uue süsteemi külge alles ööl enne kasutuselevõttu, välistades igasuguse eelneva testimise. Tihti võivad automaatikasüsteemiga integreerimist vajada keerulised välised kompaktseadmed paljude sisendite ja väljunditega, ilma et seadmed ise varakult testimiseks saadaval oleksid.

Keerulisemal juhul võib puudu olla suurem osa riistvarast, kaasa arvatud PLC ise. Tarkvaraarenduse esimeses faasis on see tavaline. Kuidagi tuleb arendamisel olevat loogikat katsetada ka sellises olukorras. See probleem on üks peamisi põhjuseid, miks tihti jäetakse PLC tarkvara arendus tervenisti automaatikasüsteemi lõppfaasi, mil kogu riistvara on paigaldatud. Tagajärjeks on aga suurem ajakulu ning probleemide hiline avastamine.

2.6.4 Oht automaatikaseadmetele ja täituritele

Testimise käigus võivad vigase loogika tõttu riistvaraseadmed kahjustada saada. Mõned näited on toodud alljärgnevalt:

- Andureid võib kahjustada toode, millel on vigase töötlemise tagajärjel valed mõõtmed. Näiteks pakendamise või lõikamise tulemusena on toode liiga suur.
- Mootor võib vale juhtimise, ülekoormuse või kuivale jäämise tõttu üle kuumeneda või mehaaniliselt puruneda.
- Midagi võib puruneda või deformeeruda liiga kõrge rõhu, temperatuuri või muu parameetri tõttu.

- Midagi võib puruneda vales kohas paiknemise tõttu. Näiteks võib ajutiselt toote juurde liigutatav andur jääda ette konveieri osale. Paljud tööstusprotsessid eeldavad väga hoolikat liigutuste sünkroniseerimist.

Paljud automaatikaseadmed ja täiturid on väga kapriissed ning neile on seatud ranged töötingimused, mida tuleb igal juhul täita. Testimise käigus võib olla nende tingimuste range täitmine raske või lausa välistatud. Tihti on tingimuste hulgas nõudmisi, mida ei olegi võimalik läbi mängida, sest oht seadmetele on liiga suur. Seadmete kahjustamist testimisel tuleb ilmselgelt vältida.

2.6.5 Oht tootele ja keskkonnale

Kõige parem ja kindlam viis näiteks tootmisliinide testimiseks on nende testimine reaalse tootega. Tihti on see aga välistatud liigse kulukuse tõttu. Tooraine võib olla liiga kallis selleks, et selle peal eksperimenteerida. Selliste protsesside puhul on vajalik testimine mingil muul viisil ära teha, et reaalse toorainega testimisel oleks ebaõnnestumise tõenäosus minimaalne.

Testimine võib olla ohtlik ka tootmise keskkonnale. Esiteks võib testimine kahjustada riistvara, mööblit ja muid abivahendeid tootmiskeskkonnas. Näiteks võivad deformeeruda hoidlate seinad või tootmispinnad. Testimine võib ohustada ümbritsevat keskkonda potentsiaalse reostuse näol. Toiduainetööstuses kasutatakse erinevaid kemikaale, mis eriti kõrgetes kontsentratsioonides on keskkonnale ohtlikud – näiteks happed ja alused torustike ja tootmisseadmete pesuks ning desinfitseerimiseks. Testimise käigus esilekerkivate vigade tõttu võivad sellised kemikaalid kergesti keskkonda sattuda.

2.6.6 Oht inimesele

Eraldi tuleb mainida ohtu, mida kujutavad PLC testimisprotsessi käigus esile kerkida võivad tarkvaravead inimese füüsilisele tervisele. Ohustatud on tarkvaraarendajad, tehnikud, elektrikud, operaatorid, kliendi esindajad ning teised inimesed, kes seadmete läheduses viibivad. Mõned näited inimest ohustavatest teguritest:

- Inimene võib füüsiliselt kuskile vahele jääda või millegagi pihta saada. Seadmed võivad töötada ettearvamatul viisil, samuti võib õnnetusi kergesti juhtuda probleemolukordade lahendamisel. Näiteks puudutab see oht protsesse, kus on kasutusel hüdraulika või pneumaatika.

- Inimene võib kokku puutuda kemikaalide või ohtliku tolmuga. Ohtliku aine leke võib olla märkamatu, kuid tõsiste tagajärgedega.
- Inimest võib ohustada veel elekrivool, kõrge temperatuur, kõrge rõhk, suured kõrgused ja paljud teised tegurid.

Testimine tootmiskeskkonnas reaalsete seadmetega on küll vajalik ja efektiivne, kuid võib olla väga ohtlik.

2.6.7 Automaattestide keerulisus

Kõik eelnevates punktides käsitletud probleemid teevad PLC tarkvara automaattestide läbiviimise äärmiselt keeruliseks. Olukorras, kus on oht automaatikaseadmetele, tootele, keskkonnale ja inimestele on peaaegu mõeldamatu teste automaatselt jooksutada. Teisalt oleks mõningaid teste just vaja automaatselt läbi viia (näiteks protsessi keerukuse või kiiruse tõttu), kuid selleks puudub tarkvararaamistik või riistvara.

Ka regressioonitestide jaoks on PLC tarkvara puhul vajadus täiesti olemas. PLC tarkvara muudetakse pärast käiku andmist küll harva, aga kui muudetakse, on seda olulisem veenduda, et juba varem testitud programmi loogikasse vigu ei tekiks, sest vastasel korral võivad tagajärjed olla rasked.

2.7 Ülesande püstituse laiendus PLC tarkvara testimise probleemidest lähtuvalt

Eelmises peatükis välja toodud PLC tarkvaraga seotud testimise probleemidest sõltuvalt võib uuesti üle vaadata ja mõnevõrra laiendada käesoleva töö alguses seatud eesmärke (vt jaotis 1.2). Sellest lähtuvalt otsib käesoleva töö teine osa vastuseid järgmistele küsimustele:

1. Kas eksisteerib ning milline näeks välja PLC tarkvara testimismeetod, millega on võimalik PLC tarkvara testida järgmistes olukordades:
    a. On vaja testida lihtsat loogikat koos komplekteeritud riistvaraseadmetega
    b. Tööstusprotsessid on nii kiired või täpsed, et silmaga pole võimalik jälgida nende toimimise korrektsust
    c. Tööstusprotsessid peavad toimuma täpselt ettenähtud aja piirides

d. PLC-l puudub osaliselt või täielikult ligipääs riistvaraseadmetele
e. Arendajal puudub ligipääs PLC-le
f. Puudub võimalus reaalse toote või toorainega testimiseks
g. Automaatikaseadmed ja/või täiturid võivad saada tarkvaravea tõttu kahjustatud
h. Keskkonnale või tootele eksisteerib oht tarkvaravea tõttu
i. Inimesele eksisteerib oht tarkvaravea tõttu
j. Loogika on keeruline, projekt mahukas
k. On vaja läbi viia automaatteste kõigis eelpoolmainitud olukordades

2. Mida on vaja PLC tarkvara arendusprotsessis soovituslikult arvesse võtta, et välja pakutud testimismeetodite kasutamine oleks efektiivsem ning üldine tarkvara kvaliteet kõrgem?

Vastuste leidmiseks vaadeldakse kõigepealt neid meetodeid, mida kasutatakse antud valdkonnas senise praktika järgi, antakse neile hinnang, ning pakutakse seejärel välja uus meetod, millega on võimalik katta kõik ülal püstitatud küsimused, muuhulgas ka senise praktika puudujäägid.

2.8 Antud valdkonna senine praktika

2.8.1 Olukord vestluste põhjal

Töö autor vestles kokku viie inimesega, kellel on pikaaegsed professionaalsed kogemused PLC programmeerimise alal. Eelkõige andsid need vestlused hea ülevaate Eestis kasutatavatest praktikatest, väiksemal määral ka infot välisriikide kohta. Nendest vestlustest oli selgelt näha, et põhiliselt testitakse PLC tarkvara simulaatoriga või töölaual paikneva PLC-ga ning seejärel juba lõplikus konfiguratsioonis koos riistvaraga.

Simulaatoriga testimist kasutab enamus arendajaid ning see on üks-ühele võrreldav laual paikneva PLC-ga testimisega, sest IO-seadmeid pole kummalgi juhul süsteemile külge ühendatud. PLC loogika toimimiseks vajalikke sisendeid ja mälupesi mõjutatakse käsitsi PLC arendaja poolt pakutavate tööriistadega ning jälgitakse väljundeid ehk tulemusi oma silmaga. Suurem osa realiseeritavaid projekte on nii aeglase iseloomuga, et ei tunta vajadust parema

lahenduse järele. Mõningate PLC platvormide jaoks eksisteerib ka keerulisemaid sisendite mõjutamise tööriistu, kuid neid ei kasutata.

Samas on ka neid arendajaid, kes simulaatoril testimise peale üldse aega ei kuluta. Esimesed testid tehaksegi töötava PLC peal, millele on IO-seadmed külge ühendatud. Sellisel juhul on lõplikus konfiguratsioonis koos riistvaraga tehtavad kasutuselevõtmise testid ainsad tarkvara testid. Põhjustena tuuakse välja aja ja raha kokkuhoidu testimise arvelt. Viidatakse ka sellele, et osa programmist on nagunii kopeeritud mõnest varasemast projektist, ega vaja uuesti testimist. Väidetakse ka, et ainult reaalne kasutuselevõtt näitab süsteemi tegelikku käitumist. Samas on teemat uurinud autorid välja toonud [10], et reaalsuses kulub sellise lähenemise puhul kogu projekti peale rohkem aega kui võiks, sest riistvaraseadmete paigalduse lõppedes võib tarkvara kirjutamine alles algusjärgus olla (ka testimist tuleks lugeda tarkvara kirjutamise osaks) või pole selle tööga riistvara puudumise ettekäändel veel alustatudki.

Enne riistvaraga koos testimise juurde asumist järgivad mõned arendajad seda praktikat, et testivad osa PLC programmist ära koos selliste seadmetega, mida annab kontorisse kaasa võtta. Näiteks kui on vaja kasutusele võtta uus sagedusmuundur või mõne muu seadme kontroller, luuakse selle juhtimiseks PLC programm (*draiver*) ning testitakse see ära kontoris.

Sellist osade kaupa testimist rakendavad üldiselt kõik arendajad ka juba tööstuse juures lõpliku riistvarakonfiguratsiooniga testides. Alustatakse IO-testist, mille käigus veendutakse, et kõik sisendid ja väljundid reageerivad nii nagu ette nähtud ning seadmete sisend- ja väljundsignaalid on ühendatud tarkvaras õigete mäluaadressidega. Seejärel jätkatakse eraldiseisvate, isoleeritud komponentide testimist ning tasapisi jõutakse kogu süsteemi testimiseni selle normaalse käivitamise teel.

Kui tegu on potentsiaalselt ohtliku protsessiga, asendatakse kemikaalid veega, ühendatakse mõned täiturid ajutiselt lahti või püütakse mõnel muul moel riske vähendada. Esialgu proovitakse protsesse läbi käsitsi, hiljem juba automaatselt. Vahel harva kasutatakse ka IO-seadmetena nuppe ja potentsiomeetreid ning teisi käepäraseid vahendeid. On arendajaid, kes püüavad mõnda puuduvat andurit asendada loogika muutmisega, et testimine saaks jätkuda.

Selliste testimismeetodite kasutamist põhjendavad arendajad eelkõige asjaoluga, et need on lihtsasti rakendatavad ning aja- ja rahakulu mõttes soodsad.

Kindlaid meetodeid nõuete kirjeldamiseks ei kasuta vähemalt Eestis põhimõtteliselt mitte keegi. Klient esitab oma soovid sellisel kujul, nagu ise heaks arvab, edaspidised täpsustused

tehakse jooksvalt koosolekute ja muul viisil arutelude teel ning pannakse kirja arendaja poolt juhuslikul viisil, kui üldse. Mõni klient kirjeldab ära isegi *JA-VÕI*-loogika, teine klient vaid üldsõnalise tööpõhimõtte. Kliendi poolt antud lähteülesannete põhjal kindlas vormis nõuete kirjeldusi arendajad ei koosta. Tihti on põhiline nõuete esitamise viis *AutoCad* joonis või mõni muu skeem, kus kujutatakse kõikide tootmisseadmete asukohti ning omavahelisi ühendusi. Mõnel juhul on joonisele lisatud ka automaatikaseadmete, näiteks andurite asukohad.

Suuremate kogemustega arendajate sõnul on näiteks Soomes, Hollandis ja Saksamaal kliendid rohkem ette valmistunud kui tüüpiliselt Eestis, sest nende poolt edasiantavad nõuded on mõnevõrra põhjalikumad. Sakslastel on dokumendid kõige täpsemad ning neid tuleb ka rangemalt järgida (ka vigaseid dokumente).

Nõuete esitamise olukorrale on sarnane ka klientide kaasatus arendus- ja testimisprotsessi. Eesti ettevõtted tahavad kätte saada töötava lõpptulemuse, samas kui eelpool mainitud riikidest pärit kliendid tunnevad rohkem huvi ka arendusprotsessi ja vaheetappide üle. Välisriikide klientide puhul tuleb ette ka tarkvara tüüppõhjade või moodulite läbitestimist ning tellijapoolset kinnitamist. Eesti ettevõtetel selline huvi üldjuhul puudub.

Dokumenteeritud testimist Eesti PLC tarkvara arendajad ei kasuta üldse. Ainsad dokumenteeritud testid on vahel harva need, mida tellija ise nõuab. Ohtlike tööstusprotsesside puhul võib olla klient koostanud väga täpse kasutuselevõtmise programmi, mis ühtlasi toimib ka kasutuselevõtu-testina. Seda aga viivad enamasti läbi ka kliendi enda esindajad. Arendajad ei koosta ise nõuete põhjal dokumenteeritud teste koos testjuhtudega, kuna seda peetakse liiga ajamahukas ja kulukaks.

2.8.2 Olukord kirjanduse põhjal

PLC tarkvara testimist käsitletakse kirjanduses võrdlemisi vähe. Suurem osa meetoditest ja teemadest, mida käsitletakse, on pigem seotud mudelipõhise testimisega või teiste meetoditega, mis ei hõlma endas loogika käivitamist. Palju räägitakse tarkvara testimise põhimõtetest, mis kehtivad tegelikult igasuguse tarkvara kohta (vt jaotis 2.5). Üldiselt eeldatakse sarnaselt tegelikule praktikale, et põhilised PLC tarkvara testimise meetodid on simulaatori kasutamine ning lõplikus riistvarakonfiguratsioonis testimine. [15] [16]

Eraldi välja toomist väärib see, et päris palju juhitakse tähelepanu põhimõttele, et testimist tuleks läbi viia moodulite kaupa ning tarkvara peaks olema samuti modulaarne. [10]

Reaalajasüsteemide ja sardsüsteemide (*Embedded Systems*) valdkonnas on testimisega seotud kirjandust märksa rohkem. Paraku eeldatakse nende süsteemide jaoks mõeldud testimismeetodite kirjeldamisel enamasti oluliselt laiemate võimalustega arvutite olemasolu, mistõttu on neid meetodeid PLC puhul raske või võimatu kasutada. Isegi *ARM Cortex M3* protsessoriga süsteem sisaldab võimalusi, mida PLC puhul realiseerida võimalik pole. [17]

Ka nende süsteemide puhul eeldatakse esmase testimise viisina lihtsat käivitamist. Tähelepanu juhitakse samuti moodulite kaupa sammhaaval testimisele ning võimalusele täiturid näiteks LED-tuledega asendada ning andureid traadijupiga mõjutada. [18] Päris mitmeid meetodeid on välja pakutud just reaalajasüsteemide ajakriitiliste funktsioonide korrektsuse hindamiseks. [19] Suur osa väljapakutud testimismeetoditest on siiski pigem teoreetilist laadi meetodid, mis kirjeldavad väga spetsiifilisi samme kindlat tüüpi probleemide testimiseks, ning neid on üldisesse PLC konteksti raske üle tuua. [20] [21]

Eraldi võib välja tuua ühe reaalajasüsteemide jaoks välja pakutud ent PLC puhul põhimõtteliselt kasutatava testimismeetodi, mis hõlmab endas töötava süsteemi jälgimist läbi selleks loodud erilise riistvara ning süsteemi sisse lisatud tarkvaraliste diagnostika-funktsioonide. Süsteemi toimimist kirjeldavad parameetrid salvestatakse, et hiljem arenduskeskkonnas testitava tarkvara tööd rekonstrueerida ning analüüsida. Välja on pakutud ka puhtalt tarkvarapõhine jälgimismeetod, kuid see pole PLC iseärasuste tõttu käesoleva töö kontekstis kasutatav. [22]

PLC tootjate endi poolt avaldatud kirjanduses käsitletakse testimise kontekstis praktiliselt ainult simuleerimist ja käesolevas töös varasemalt kirjeldatud protsesside jälgimise ning loogika silumise vahendite kasutamist (vt jaotis 2.3.4).

Automaattestimise lahendusi, mis võimaldaksid kogu PLC loogikat automaatse programmiga testida, turul saada ei ole. Ühiktestimisest räägitakse küll, kuid vaid selles kontekstis, et tarkvara tuleks testida moodulite kui ühikute kaupa. [23] Üksikud valmiskujul testimistarkvara lahendused on keskendunud riistvaraseadmete vahelise kommunikatsiooni emuleerimisele või teesklemisele, mis sobib küll näiteks SCADA lahenduste katsetamiseks, mitte aga PLC loogika põhjalikuks testimiseks. [24] Olemas on veel ka teisi valmiskujul testimist hõlbustavaid tarkvarapakette, kuid need on üldjuhul väga kallid. [25]

3. Praktilised testimismeetodid ühe projekti näitel

Selles peatükis analüüsitakse kõigepealt antud valdkonna senise praktika poolt kasutatavaid PLC testimismeetodeid ning leitakse, millistes olukordades neid kasutada saab. Seejärel pakutakse välja uus testimismeetod. Meetodite analüüsi juures kasutatakse illustratsiooniks ühe näidisprojekti kontekstis toodud teste.

3.1 Näidisprojekti kirjeldus

Näidisprojekt on valitud selliselt, et seda kirjeldavate nõuete baasil oleks võimalik luua teste, mis käsitleksid kõiki käesoleva töö eesmärkides väljatoodud olukordi (vt jaotis 2.7). Projekt on valitud võimalikult lihtne, et tähelepanu ei koonduks testimismeetoditelt projektile. Projekt ei ole siin töös kirjeldatud seetõttu ka terviklikult, vaid näidete toomise seisukohalt piisavas ja vajalikus mahus.

Näidisprojekt on osa töö autori poolt realiseeritud projektist Eesti toiduainetööstuse ettevõtte jaoks.

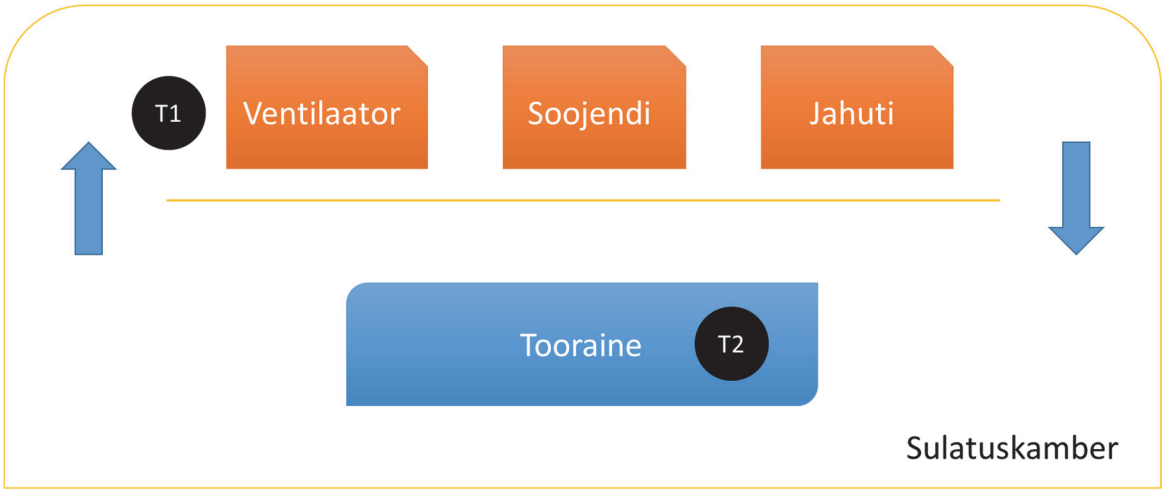
3.1.1 Taust ja lühikirjeldus

Tegemist on sulatuskambriga, mille eesmärk on sulatada liha, kala ning teisi külmutatud toiduainetööstuse tooteid ning neid säilitada pärast sulatust sobivates tingimustes. Varem on klient toodet sulatanud mittekontrollitud viisil ettevõtte tootmishoone koridorides. Vajadus on aga sulatusprotsessi kiirendada ning tagada nõuetekohane tulemus õigeks ajaks.

Selleks on klient ehitanud kambri, kus on lae all kanalis ventilaator õhu tsirkulatsiooniks kambri sees ning kalorifeer kambri õhu soojendamiseks ehk energia lisamiseks. Lisaks on kambris ka külmagregaat kambri õhu jahutamiseks ehk energia eemaldamiseks. Sulatusprotsessi juhtimiseks on mõeldud temperatuuriandurid nii ümbritsevasse ruumi kui

toote sisse. Kliendi soov on, et kõrge õhutemperatuuriga sulatataks toode üles soovitud temperatuurini, misjärel säilitataks sulatatud toodet madalal temperatuuril nii kaua kui vaja.

Klient on ette andnud kõigi seadmete, kaasa arvatud andurite asukoha. Kirjelduse põhjal on koostatud joonis 6, mis ruumi situatsiooni külgvaates visualiseerib. Temperatuuriandurid on joonisel tähistatud T1 ja T2. Siniste nooltega on näidatud õhu tsirkuleerimise suund.



Joonis 6. Näidisprojekti tootmisprotsessi lihtsustatud põhimõtteskeem

3.1.2 Näidisprojekti riistvaralahendus

Sulatuskambri automaatikasüsteemi PLC on lihtne, monoliitne kontroller, millele on sisse ehitatud projekti vajaduste jaoks piisavalt sisendeid ja väljundeid. PLC paikneb kambrile piisavalt lähedal ning kõik automaatikaseadmed ja täiturid on kompaktselt koos, mistõttu pole ühegi tööväljasiini kasutamine vajalik. Protsessi juhtimine käib füüsiliste nuppudega.

Tabelis 2 on nimekiri sulatuskambri PLC külge ühendatud sisenditest ja väljunditest. Need IO-d on PLC tarkvaras seotud igaüks vastavate mälupesadega.

DI	AI	DO	AO
Ukseandur	Kambri õhu temperatuur (T1)	Ventilaatori käiviti	Jahuti võimsus
Ventilaatori termiline kaitse	Toote temperatuur (T2)	Soojendi pooljuht-lüliti	
Sulatusrežiimi nupp		Alarmituli ja sireen	

Säilitusrežiimi nupp		Sulatusrežiimi nupu tuli	
Stopprežiimi nupp		Säilitusrežiimi nupu tuli	
		Stopprežiimi nupu tuli	

Tabel 2. Näidisprojekti lihtsustatud IO-tabel

3.1.3 Näidisprojekti nõuded

Allpool on lihtsustatud kujul kirjeldatud mõned PLC tarkvarale seatud nõuded. Nõuete formuleerimisel pole kasutatud ühelgi probleemiraamistikul põhinevat meetodit, sest probleem on väga lihtne. Igal nõudel on oma ID-kood, mille abil sellele nõudele käesolevas töös viidatakse.

ID	Kirjeldus
N-1	Sulatuskambril on kolm režiimi: sulatus, säilitus, stopp. Korraga tohib sees olla ainult üks režiim. Aktiivset režiimi saab kasutaja käsitsi igal ajahetkel muuta.
N-2	Sulatus- ja säilitusrežiimi ajal ringleb kambris õhk. Õhk ei tohi ringelda stopprežiimi ajal või siis, kui on avatud kambri uks. Õhk ei ringle kambris tegelikkuses ka siis, kui on rakendunud ventilaatori termiline kaitse. Kui see on rakendunud, peab käima alarm.
N-3	Soojendi tohib töötada ainult siis, kui õhk sulatuskambris ringleb.
N-4	Sulatusrežiimi ajal tuleb kambri õhu temperatuur hoida soojendi abil võimalikult ühtlaselt 43°C lähedal. 1h pärast sulatuse algust ei tohi kambri õhu temperatuuri erisus ületada 1°C võrreldes ettenähtud temperatuuriga.
N-5	Soojendi võimsust reguleerib süsteem soojendi sisse- ja väljalülitamisega. Kahe sisselülitamise vaheline aeg peab olema vahemikus 0,1 kuni 2 sekundit.
N-6	Kui sulatusrežiimi ajal tõuseb toote temperatuur üle 2°C, lülitub süsteem automaatselt üle säilitusrežiimi.
N-7	Ükski DO ei tohi olla sisse või välja lülitatud lühemaks ajaks kui 1 sek. Soojendi puhul on selleks piiriks 0,01 sek.

Tabel 3. Näidisprojekti nõuded

3.1.4 Näidisprojekti testid

Käesolevad testijuhud on loodud vastavalt kirjeldatud nõuetele (vt jaotis 3.1.3). Tegemist on näidistestidega, mida arendaja võiks tahta läbi viia, et tarkvarast vigu leida, mistõttu ei moodustu testidest kõikehõlmav täielik testikogu. Testid on kirjeldatud tabelite kujul (vt tabelid 4-9).

Testi ID	T-1
Nõude ID	N-2, N-7
Kirjeldus	Ventilaatori töötamine vastavalt režiimile
Sisendid	Ukseandur pole rakendunud. Ventilaatori termiline kaitse pole rakendunud. Lühikeste vahedega on aktiivsed järgmised režiimid: stopp, sulatus, säilitus, sulatus, stopp.
Oodatavad väljundid	Ventilaatori seisund peab režiimide ajal olema vastavalt: väljas, sees, sees, sees, väljas. Ventilaator ei tohi jääda sisse- või väljalülitatuks lühemaks ajaks kui 1 sek.

Tabel 4. Testi T-1 kirjeldus

Testi ID	T-2
Nõude ID	N-2, N-3
Kirjeldus	Ventilaatori ja soojendi peatamine ukse avamisel
Sisendid	Aktiivne on sulatusrežiim. Ventilaatori termiline kaitse pole rakendunud. Kambri õhu temperatuur on oluliselt alla 43°C. Toote temperatuur on oluliselt alla 2°C. Ukseandur on esialgu väljas, kuid siis rakendatakse.
Oodatavad väljundid	Ventilaator peab olema esialgu sees, samuti peab regulaarselt sisse lülituma soojendi. Pärast ukse avamist peab nii ventilaator kui soojendi olema seiskunud.

Tabel 5. Testi T-2 kirjeldus

Testi ID	T-3
Nõude ID	N-4
Kirjeldus	Sulatustemperatuuri hoidmine

Sisendid	Aktiivne on sulatusrežiim. Ukseandur pole rakendunud, ventilaatori termiline kaitse pole rakendunud. Toote temperatuur on oluliselt alla 2°C.
Oodatavad väljundid	Kambri õhu temperatuur peab tõusma 43°C lähedale. Tunni möödudes ei tohi temperatuur erineda eesmärgist rohkem kui 1°C.

Tabel 6. Testi T-3 kirjeldus

Testi ID	T-4
Nõude ID	N-2, N-3
Kirjeldus	Ventilaatori termilise kaitsme rakendumine
Sisendid	Aktiivne on sulatusrežiim. Ukseandur pole rakendunud. Toote temperatuur on oluliselt alla 2°C. Kambri õhu temperatuur on oluliselt alla 43°C . Alarm ei käi. Ventilaatori termiline kaitse pole esialgu rakendunud, kuid siis rakendub.
Oodatavad väljundid	Esialgu peab soojendi regulaarselt sisse lülituma, pärast termilise kaitse rakendumist peab soojendi olema välja lülitatud ning alarm käima.

Tabel 7. Testi T-4 kirjeldus

Testi ID	T-5
Nõude ID	N-5
Kirjeldus	Soojendi võimsuse reguleerimise vastamine tingimustele
Sisendid	Aktiivne on sulatusrežiim. Ukseandur pole rakendunud. Ventilaatori termiline kaitse pole rakendunud. Toote temperatuur on oluliselt alla 2°C. Kambri õhu temperatuur on oluliselt alla 43°C . PLC tarkvara poolt arvutatud antud hetkel vajalikku soojendi võimsust peab saama tuvastada.
Oodatavad väljundid	Soojendi peab sisse ja välja lülituma selliselt, et ühe sisse-väljalülitus-tsükli korral SeesKestvus / KokkuKestvus * 100 = EttenähtudVõimsusProtsentides (erinevus mitte üle 1%), kusjuures 0,1 sek < KokkuKestvus < 2 sek.

Tabel 8. Testi T-5 kirjeldus

Testi ID	T-6
Nõude ID	N-6, N-1
Kirjeldus	Automaatne sulatuse lõpetamine
Sisendid	Aktiivne on sulatusrežiim. Ukseandur pole rakendunud. Ventilaatori termiline kaitse pole rakendunud. Toote temperatuur on alla 2°C. Toote temperatuur tõuseb pidevalt.
Oodatavad väljundid	Kui toote temperatuur tõuseb temperatuurini 2°C, peab aktiivseks minema säilitusrežiim.

Tabel 9. Testi T-6 kirjeldus

3.2 Lihtne testimine – tarkvara käivitamine tootmistingimustes

Lihtsaim PLC tarkvara testimise viis on PLC tarkvara käivitamine tootmistingimustes koos töötava riistvara, tooraine ning kõige muuga, mis protsessi toimimiseks ette nähtud on. Sellist testi võib nimetada ka kasutuselevõtutestiks, kus proovitakse läbi kogu vajalik funktsionaalsus ning jälgitakse, kas tulemus vastab ootustele. Lihtne testimine on praktikas kasutatavatest testimismeetoditest levinuim.

Sellisel viisil testimise läbiviimine on lihtne, sest selleks pole vaja luua midagi ajutist, mis on vajalik ainult testimise läbiviimiseks. See on nagu lihtsa arvutiprogrammi käivitamine ning kasutusjuhtude läbiproovimine. Enamasti ei koostata sellise testimise jaoks ka dokumenteeritud teste, testide aluseks võetakse otse püstitatud nõuded, olgu needki dokumenteeritud või mitte.

3.2.1 Lihtsa testimisprotsessi kirjeldus

Lihtsa testimise läbiviimiseks tuleb täita järgmised eeldused:

- PLC on varustatud valmiskujul tarkvaraga ning töötab tõrgeteta.
- PLC külge on ühendatud kogu vajalik riistvara koos kaablitega. Signaalid peavad jõudma sisendseadmetest PLC-sse ning PLC-st väljundseadmetesse.

- Täidetud on ka hädavajalikud tootmisalased tingimused, näiteks on varutud toorainet.
- Soovituslikult võiks PLC olla ühendatud ka arenduskeskkonnaga. Arenduskeskkonna abil on võimalik jooksvalt jälgida kõikide kasutatud mälupesade hetkeseisu (IO-de ning protsessimuutujate jälgimiseks).

Kui need eeldused on täidetud, tulebki testimiseks automaatikasüsteemi lihtsalt kasutada ettenähtud viisil, et läbi proovida kogu funktsionaalsus. Kui eksisteerivad dokumenteeritud testid, tuleb need ükshaaval läbi käia. Kui neid pole, kuid on olemas dokumenteeritud nõuded, tuleb testid kohapeal välja mõelda.

Näidisprojekti kontekstis tuleb lihtsaks testimiseks läbi käia kõik kirjeldatud testid (vt jaotis 3.1.4). Näitena on toodud paari testi lihtsa läbiviimise kirjeldus (vt tabelid 10 ja 11).

Testi ID ja pealkiri	T-1. Ventilaatori töötamine vastavalt režiimile
Testi tegevused	Suletakse kambri uks. Eeldatakse, et ventilaatori termiline kaitse pole rakendunud. Vajutatakse režiimi nuppe erinevas järjekorras. Kuulatakse, kas ukse taga kambris hakkab ventilaator mühisema. Kui ventilaatoreid on mitu, jäetakse üks inimene kambrisse olukorda jälgima. Hinnatakse, ega ventilaator ei lähe käima lühemaks ajaks kui 1 sek, kui režiimi nuppe kiiresti vahetatakse. Üritatakse kuulata, ega ventilaatori relee ei klõpsi kiiresti sisse-välja (ei „värise“). Ventilaatori relee väljundi korrektsust üritatakse hinnata ka PLC arenduskeskkonnas vastava mälupesa seisundit silmaga jälgides.
Testi võimalik järeldus	Ventilaator töötab vaid ettenähtud režiimide ajal. Tundub, et ventilaatori minimaalne seisu- või tööaeg on umbes 1 sek. Mingeid iseäralikke relee-klõpsatusi ei ole kuulda.

Tabel 10. Testi T-1 läbiviimise kirjeldus

Testi ID ja pealkiri	T-3. Sulatustemperatuuri hoidmine
Testi tegevused	Viiakse kambrisse külmutatud toode, selle sisse paigaldatakse toote temperatuuriandur. Suletakse uks. Käivitatakse sulatusrežiim. Jälgitakse kambri õhu temperatuuri PLC arenduskeskkonnast või kambrisse kaasa võetud käsi-termomeetriga. Aega mõõdetakse kella vaadates, temperatuuri kõikumist hinnatakse mälu järgi.

Testi võimalik järeldus	Kambri õhu temperatuur tõuseb soovitud tasemele, ega tundu kõikuvat liialt.

Tabel 11. Testi T-3 läbiviimise kirjeldus

Ülejäänud testide läbimine toimub sarnaselt neile näidetele.

3.2.2 Võimalused ja ohud näidisprojektis

Selline testimisviis on hea ja efektiivne mitmel põhjusel:

- Testimise tulemus on vahetult näha ning koheselt hinnatav iga nõuetega kursis oleva inimese, mitte ainult arendaja jaoks.
- Kuna kasutusel pole ajutisi lahendusi või asendusmeetmeid on testimise tulemus lõplik selles mõttes, et kui jõutakse seisuni, kus enam vigu ei leita, võibki süsteemi kasutusele võtta.
- Testimise ettevalmistamise kulu on minimaalne.
- Kui eksisteerib riistvaraga seotud suuri vigu, selguvad need kiiresti. See kehtib eriti toite- või signaalivigade korral.

Samas on näidisprojekti lihtsa testimisega seotud mitmed konkreetsed probleemid:

- Kõiki teste pole üldse võimalik läbi viia:
    - Test T-4 (Ventilaatori termilise kaitsme rakendumine) pole testitav, sest termilist kaitset pole võimalik käsitsi rakendada (mootorit läbi põletada).
    - Test T-5 (Soojendi võimsuse reguleerimine vastavalt tingimustele) pole samuti otseselt testitav. Pooljuht-lüliti lülitumine ei tee piisavalt häält, et seda jälgida, ning kui jälgida LED-tuld selle olemasolul või PLC mälupesa seisundit, pole võimalik silmaga sellise vilkumise intervalli isegi mitte hinnata, rääkimata selle põhjal võimsuse välja arvutamisest. Kõik arvutused oleksid väga ligikaudsed. Sisuliselt on võimalik tuvastada vaid, et midagi toimub.
- Testide tulemused pole piisavalt veenvad, näiteks:
    - Näitena toodud T-1 testimisel pole stopperiga minimaalse ventilaatori seisu- või käiguaja hindamine piisavalt täpne. Ventilaatori hääle tekkimisel ja kadumisel läbi seina või ka kambri sees on ebamäärane viivitus. Releede kuulamine nn „värisemise“ diagnoosimiseks ei pruugi anda soovitud tulemusi.

PLC arenduskeskkonnast kiiret väljundite vilkumist ei näe, sest arenduskeskkond uuendab muutujate väärtusi paar korda sekundis, mis on liiga aeglane.

    - Näitena toodud T-3 lihtne testimine on küllaltki hea ja temperatuuri hoidmise algoritmi testimise ja seadistamise seisukohalt hädavajalik. Samas ei pruugi ka see olla piisav, kuna kõikumise hindamine võib olla liiga ebatäpne. Lisaks võib PLC kaudu mälupesa jälgimist takistada see, et AI mälupesa väärtus on tavaliselt mingis vahemikus täisarv, mis vajab inimloetavaks teisendamist.

- Eksisteerib oht inimesele, keskkonnale, tootele:
    - Sulatuskambri ventilaatorid on väga võimsad ning kambris sees viibimine või isegi avatud ukse kõrval seismine võib olla ohtlik, sest asjad võivad lenduda ning inimesele või ventilaatorile pihta lennata. Lisaks esineb testimisel ka suuri temperatuurikõikumisi, mis koos tuulega võivad kergesti külmetusnähte põhjustada. (T-1, T-2, T-3).
    - T-2 (Ventilaatori ja soojendi peatamine ukse avamisel) läbiviimisel on lisaks see oht, et kui ventilaator jääb seisma, kuid soojendi mitte, võib tekkida tulekahju, mis ohustab nii inimest kui keskkonda.
    - Sulatustemperatuuri ebaõige hoidmine või liiga hiline või varajane sulatuse lõpetamine võib toodet kahjustada (T-3, T-6).
- Eksisteerib oht automaatikaseadmetele ja täituritele.
    - Liiga tihe ventilaatori sisse-välja-klõpsimine võib ventilaatori mootorit kahjustada. Relee „värisemine“ võib lisaks mootorile kahjustada ka releed või DO-moodulit (T-1).
    - Ventilaatori termilise kaitsme rakendumise mitteregistreerimine võib põhjustada soojendi ülekuumenemist ja tulekahju (T-4).
    - Soojendi ebaoptimaalne sisse-väljalülitamine võib soojendit või selle pooljuht-lülitit kahjustada (T-5).

Nagu näha, on lihtsa testimisega töötava riistvara peal isegi primitiivse näidisprojekti põhjalik testimine problemaatiline. Rõhutada tuleb seda, et suur osa probleemidest on riskid, mis võivad realiseeruda vigade esinemise korral. Kui vigu ei esine, siis riskid ei realiseeru. Selleks, et ülalmainitud probleeme vältida ja riske vähendada ning kõik seatud testid edukalt läbi viia, tuleb kasutada täiendavaid testimismeetodeid.

Kui puuduks nii testide kui nõuete dokumentatsioon, poleks ilmselt ka takistused nii suured, sest täpselt seadmata nõuete täitmist on raske kontrollida. Tarkvara kvaliteedi ning testimise efektiivsuse ja ohutuse tõstmiseks on seetõttu põhjalike kirjelduste olemasolu oluline.

Ülalmainitud probleemid kehtivad olukorras, kus on täidetud lihtsa testimise tingimused (vt jaotis 3.2.1). Kui need eeldused pole kõik täidetud, näiteks pole toorainega testimine võimalik, on puudu mõningaid automaatika- või tootmisseadmeid või on vaja tarkvara testida ilma PLC-ta, on probleemide nimekiri oluliselt pikem. Lihtne testimine ei võimalda käesolevas töös otsitavale testimismeetodile seatud nõudmisi seega mitte kuidagi täita.

Kindlasti leidub ka väga triviaalseid projekte, mille puhul selline lihtne testimine on täiesti piisav ning sobiv kõigi testide läbi jooksutamiseks. Valdav enamus projekte on aga keerulisemad.

3.3 Käsitsi rakendatavad meetodid riistvaraga ümber käimiseks

Kui lihtsast testimisest jääb väheseks, võib testimise hõlbustamiseks ning ohtude vähendamiseks kasutada mõningaid käsitsi rakendatavaid meetodeid riistvaraga ümber käimiseks. Kuna valdav enamus lihtsal testimisel kirjeldatud probleemidest (vt jaotis 3.2.2) on riistvara või füüsilise maailmaga seotud, on sellistest meetoditest kindlasti kasu. Neid meetodeid kasutatakse reaalsuses palju.

3.3.1 Toote asendamine

Üks lihtsamaid ja elementaarsemaid viise on andurite petmine ehk füüsilise keskkonna mehaaniline simuleerimine toote asendamise teel. Mõned näited sellisest tegevusest:

- Optiline andur annab signaali, kui tema valguskiire blokeerib toode. Toote asemel võib aga kiire blokeerida käega. See võimaldab näiteks konveierit testida ilma tooteta.
- Piima asemel võib läbi torude pumbata vett, mis vähendab ohtu keskkonnale ning välistab toote riknemise.

Käesoleva töö näidisprojektis saaks sellist meetodit kasutada temperatuurianduri petmiseks. Ühe näidistesti läbi jooksutamise võimalik kirjeldus on toodud alljärgnevalt (vt tabel 12).

Testi ID ja pealkiri	T-6. Automaatne sulatuse lõpetamine
Testi tegevused	Kambrisse viiakse üks klaas jääveega, teine klaas sooja veega. Toote temperatuuriandur asetatakse jäävette ning oodatakse, kuni PLC arenduskeskkonnast on näha, et anduri näit on langenud 0°C juurde. Käivitatakse sulatusrežiim. Nüüd tõstetakse toote temperatuuriandur ümber sooja veega täidetud klaasi. PLC arenduskeskkonnast jälgitakse temperatuuri tõusu.
Testi võimalik järeldus	Toote temperatuuri tõusmisel ligikaudu üle 2°C läheb süsteem üle säilitusrežiimile.

Tabel 12. Testi T-6 läbiviimise kirjeldus

Sellisel testimismeetodil on mitu tugevust lihtsa testimisega võrreldes:

- Vigade ilmnemisel ei seata ohtu toodet.
- Testida on võimalik ka siis, kui tootega testimine pole mingil muul põhjusel võimalik.
- Kui T-6 puhul tegutseda kiiresti, ei jõua kamber väga soojaks minna, st oht tervisele on väiksem.
- Juhul kui petta ka ukseandurit, on võimalik kambri juures vajalikke teste teha nii, et kambri uks on PLC tarkvara arvates suletud, kuid tegelikult lahti, mis lihtsustab liikumist ja kommunikatsiooni.

Samas ei lahenda see meetod enamusi probleeme, mis lihtsa testimisega seotud on. Kuna ventilaatorid töötavad endiselt, pole oht inimesele täielikult kadunud, kahte testi kuuest pole endiselt võimalik läbi viia, jne. See meetod on sobilik kasutada eelkõige selliste protsesside testimisel, kus pole selliseid elemente, mis kedagi või midagi ohustada võiksid, ning toote asendamine millegi muuga on võrdlemisi lihtne.

3.3.2 Riistvaraseadmete asendamine

Järgmine võimalus toote asendamise kõrval on asendada riistvaraseadmeid. See tähendab praktikas järgmisi tegevusi:

- Anduri asemel ühendatakse PLC sisendiga näiteks lüliti, potentsiomeeter või pingeregulaator, vastavalt sisendi tüübile. See võimaldab PLC tarkvarale elektriliselt simuleerida riistvaralisi sisendsignaale. Potentsiomeetriga keeratakse just selline „temperatuur“, nagu vaja. Nõudlikemate sisendite puhul kasutatakse signaali-generaatorit.
- Täituri asemel ühendatakse PLC väljundiga näiteks tuli (DO puhul) või multimeeter (AO puhul). See võimaldab jälgida ja mõõta väljundeid.

Käesoleva töö näidisprojektis saaks sellist meetodit kasutada enamuste testide läbiviimiseks. Ühe näidistesti läbijooksutamine võiks toimuda järgnevalt (vt tabel 13).

Testi ID ja pealkiri	T-4. Ventilaatori termilise kaitsme rakendumine
Testi tegevused	Ventilaatori termiline kaitse ja ukseandur asendatakse lülititega, kumbki pole rakendunud. Mõlemad temperatuuriandurid asendatakse potentsiomeetriga, mõlemad on keeratud madala väärtuse peale. Ventilaatori käiviti asemele ühendatakse tuli. Soojendi pooljuht-lüliti asemele ühendatakse tuli. Käivitatakse sulatusrežiim. Jälgitakse ventilaatori tule põlemist ning soojendi tule vilkumist. Nüüd lülitatakse termilise kaitse lülitit. Jälgitakse soojendi tuld. Kuulatakse sireeni.
Testi võimalik järeldus	Soojendi lakkab pärast ventilaatori termilise kaitse rakendumist töötamast. Alarmi sireen ja vilkur käivituvad.

Tabel 13. Testi T-4 läbiviimise kirjeldus

Selline testmeetod lahendab suure osa varem tekkinud probleemidest:

- Kõrvaldab ohu inimestele, keskkonnale, tootele ja seadmetele, võimaldades tarkvara loogikat läbi testida ilma andureid ja täitureid kaasamata, kas osaliselt või täielikult.
- Võimaldab osaliselt või tervikuna testida pooliku riistvarakomplektiga automaatikasüsteemi.
- Võimaldab suuremat mugavust, kuna teste saab läbi viia kontoritingimustes.

Samas ei lahenda see meetod kõiki probleeme:

- Testi T-5 piisavalt täpne läbiviimine pole endiselt võimalik, sest ka tulede vilkumist pole silmaga võimalik adekvaatselt hinnata.

- Endiselt eksisteerib probleem testide tulemuse veenvusega, sest silma ja kõrvaga vaatlemise põhjal pole võimalik igas olukorras kõike näha. Asjaolu, et mootor või käiviti on asendatud põlema süttiva tulega, ei muuda siin palju.
- Selliste ajutiste meetmete rakendamine võib olla väga ajamahukas, vajalik on ka sobiva testriistvara olemasolu, mis võib olla kallis. Suuremahuliste projektide puhul on see meetod laiaulatuslike testide läbiviimiseks sisuliselt välistatud.
- Multimeetri näidu teisendamine inimesele mõistetavaks väärtuseks võib olla tülikas ning segadusttekitav. Sama probleem eksisteerib analoogsisenditega, sest seos näiteks temperatuuri ja takistuse vahel ei pruugi olla kõigi jaoks ilmne.
- PLC puudumisel pole võimalik mitte midagi testida.
- Teste tuleb igal juhul läbi viia käsitsi, automaattestide tegemine on välistatud.

Tegemist on hea testimismeetodiga eelkõige väiksemate projektide puhul. Kasu on kõige suurem, kui seda meetodit kasutada kombineeritult lihtsa testimisega, asendades vaid hädavajalikud seadmed. Näiteks võiks ülaltoodud näites asendada lülitiga ka ainult ventilaatori termilise kaitsme ning testida T-4 üldjoontes läbi lihtsal meetodil. Siiski on vaja veel täiendavaid meetodeid PLC tarkvara testimiseks, et kaetud saaksid kõik esineda võivad olukorrad.

3.4 Diagnostikafunktsioonide lisamine PLC tarkvara sisse

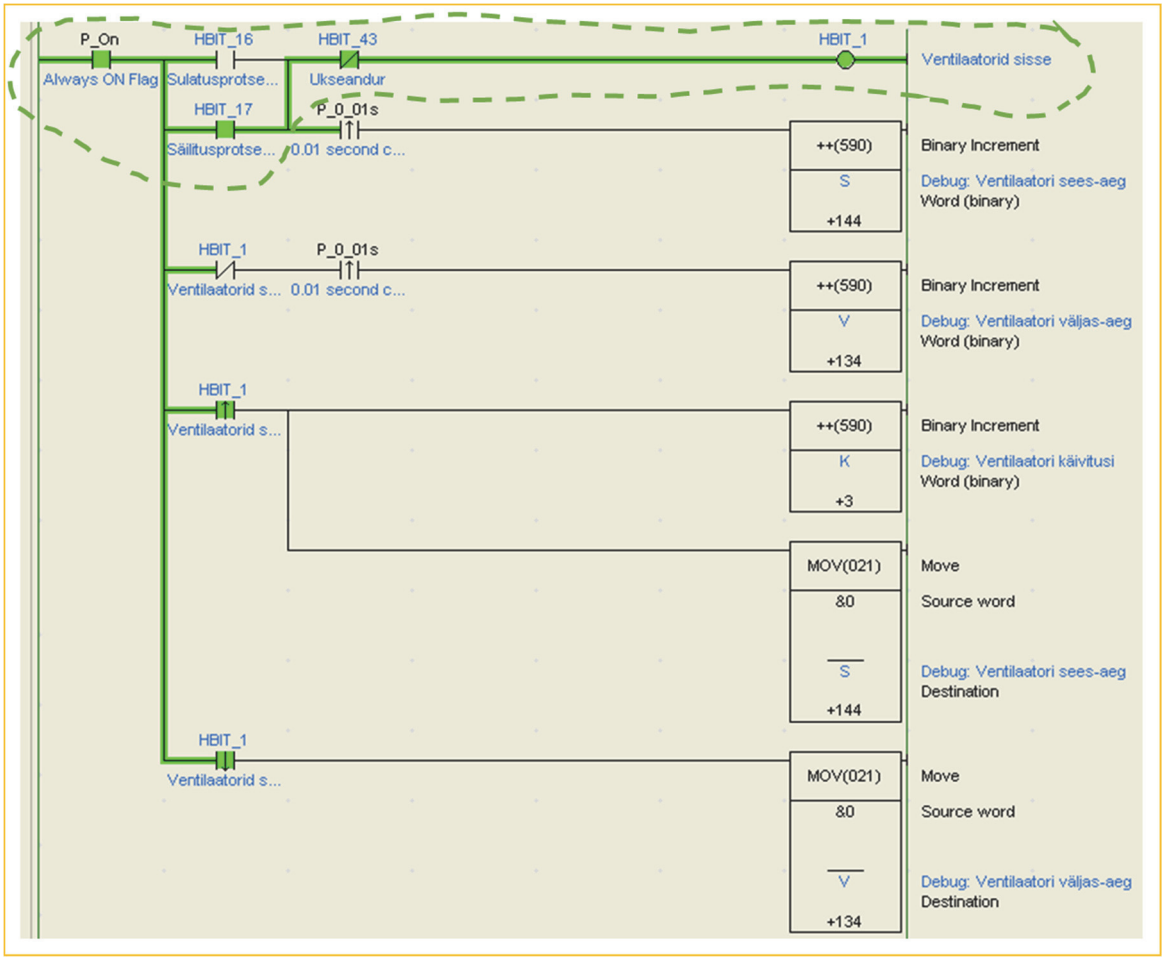
Üks meetod, mida PLC tarkvara testimisprotsessi hõlbustamiseks kasutada võib, on PLC tarkvarasse diagnostikafunktsioonide sisseprogrammeerimine. See tähendab, et loogika vahele lisatakse meetodid või muutujad, mis võimaldavad loogikaprotsesside toimimist paremini jälgida. Mõned näited sellistest funktsioonidest:

- Kriitilistes loogika-sõlmedes muutujate mõjutamine, et oleks hiljem neid muutujaid vaadates võimalik loogika toimimise kohta järeldusi teha. Näiteks võib olla vajalik näha, kas mingi lühiajaline signaal on tarkvara poolt registreeritud, või on vaja näha kiiresti muutuva sisendi väärtust mingi loogilise otsuse tegemise hetkel.
- Tsüklilise protsessi korduste arvu loendamine.
- Hinnangulise tööaja arvutamine (stopper).

Käesoleva töö näidisprojektis saaks sellist meetodit kasutada testi T-1 (Ventilaatori töötamine vastavalt režiimile) läbiviimiseks. Problemaatiline on just nõude N-7 täitmise kontrollimine, et veenduda „värisemise“ puudumises. See on olukord, mis tekib eriti PLC LD ja FBD programmeerimiskeelte puhul väga kergesti, kui ühel tarkvara läbimise tsüklil DO sisse lülitatakse ning järgmisel tsüklil uuesti välja lülitatakse. Sõltuvalt PLC tarkvara töötsükli kestvusest võib väljundi „värisemise“ sagedus olla isegi üle 100 Hz.

Nõude N-7 kontrollimise jaoks võiks PLC tarkvarasse lisada kaks funktsiooni (vt joonis 7).

- Tuleks luua muutuja `K`, mille täisarvulist väärtust suurendada iga kord, kui ventilaatori DO väärtus muutub nullist üheks.
- Tuleks luua muutujad `S` ning `V`, kuhu loendada, mitu sajandiksekundit on ventilaatori DO väärtus null või üks olnud. Sajandiksekundite loendamine ei ole küll täpne viis nii väikeste aegade mõõtmiseks, kuid taimerite kasutamine on oluliselt keerulisem.



Joonis 7. Diagnostikafunktsioonid näidisprojekti PLC loogikas. Testitav loogika on punktiirjoonega märgistatud.

Joonisel 9 kujutatud loogika tundub küll ebamõistlikult pikk ning testitav loogika väga lihtne, kuid meeles tuleb pidada, et T-1 ei testi ainult ühte moodulit vaid süsteemi tervikuna. Vigane tarkvara võib paikneda kuskil mujal. Testi T-1 läbiviimine võiks selle meetodiga käia alljärgnevalt:

Testi ID ja pealkiri	T-1. Ventilaatori töötamine vastavalt režiimile
Testi tegevused	Sisendeid mõjutatakse mõnel eelmistes peatükkides kirjeldatud meetodil. Muudetakse korduvalt aktiivset sulatuskambri režiimi. Ventilaatori käivitumiste arv loetakse üles paberil või peas. Testi lõpus võrreldakse saadud arvu muutuja K väärtusega. Lülitatakse korraks (vähem kui sek) sisse ventilaator (sobiva režiimiga), seejärel tagasi välja. Vaadatakse muutuja S väärtust. Lülitatakse ventilaator uuesti sisse. Nüüd lülitatakse ventilaator korraks (vähem kui 1 sek) välja, seejärel tagasi sisse. Vaadatakse muutuja V väärtust.
Testi võimalik järeldus	Ventilaator töötab vaid ettenähtud režiimide ajal (nõue N-1). Kuna K väärtus vastab testi läbiviija poolt loendatud väärtusele ning S ja V väärtused >= 100 (sajandiksekundit), on nõue N-7 täidetud.

Diagnostikafunktsioonide lisamine on hea meetod mitmel põhjusel:

- Võimaldab saada mingi ülevaate ka suurt täpsust nõudva või kiireloomulise PLC loogika toimimise kohta.
- Võimaldab mingil moel täita seda funktsiooni, mida tarkvaraarenduses tavaliselt silumiskonsoolile trükkimine täidab.

Samas on selle meetodiga palju potentsiaalseid probleeme:

- Diagnostikafunktsioonide programmeerimine võib olla keeruline ning ajamahukas.
- Diagnostikafunktsioonide lisamine või eemaldamine võib tekitada uusi loogikavigu.
- Diagnostikafunktsioonide jätmine töötavasse automaatikasüsteemi võib süsteemi tööd aeglustada või muul moel häirida.
- Korduvate, pikaajaliste või muul moel keeruliste loogikafunktsioonide diagnostika võib olla ebapiisava põhjalikkusega või välistatud. Valitud PLC programmeerimis-keele võimalused ei pruugi olla selleks piisavalt mitmekülgsed.
- Loogika vahele diagnostikafunktsioonide muutmine ei pruugi muudel põhjustel võimalik olla.

- Testi T-5 läbiviimine selle meetodiga on teoreetiliselt küll võimalik, kuid ilmselt ebaotstarbekas, sest vajalik diagnostikafunktsioon oleks oluliselt mahukam testitavast funktsioonist endast ning PLC poolt kasutatav stopperifunktsionaalsus ei pruugi olla piisavalt täpne.

Probleemide hulgast lähtuvalt on seda meetodit mõistlik kasutada vaid erandjuhul väga kiirete või muul moel kriitiliste ent samas väikesemahuliste tootmisprotsesside või nende osade testimisel. Seetõttu kasutatakse seda meetodit reaalsuses väga vähe. Suuremate ja keerulisemate probleemide puhul on vaja rakendada teisi meetodeid.

3.5 PLC simulaatori kasutamine

Simulaatori kasutamine on küsitluse põhjal praktikas väga levinud PLC tarkvara testimise meetod. PLC simulaatori põhilisi omadusi võib kirjeldada järgmiselt: [7]

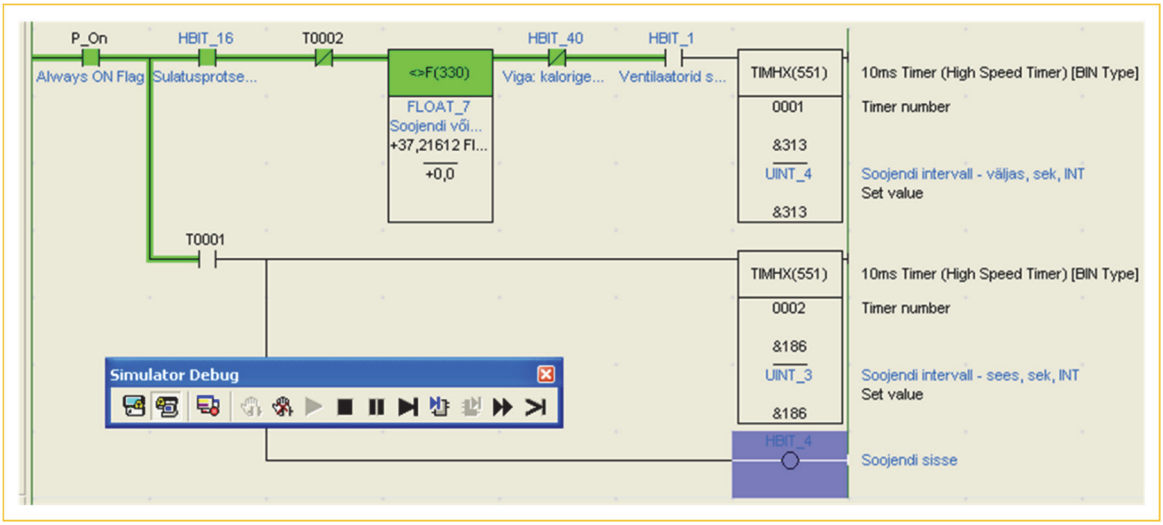
- Simulaator on arenduskeskkonda sisse ehitatud (Omroni *CX-Programmeri* puhul). Simulaatorit käivitades tekitab arenduskeskkond virtuaalse kontrolleri, millesse laetakse kogu testimist vajav PLC tarkvara. Loodud virtuaalne PLC töötab ainult arendusarvutis.
- Simulaator võimaldab loogikat käivitada, pausile panna ning samm-sammult täita.
- PLC tarkvara sisendite, väljundite ja teiste muutujate väärtusi on võimalik jooksvalt jälgida ning käsitsi muuta.

Käesoleva töö näidisprojekti puhul saaks simulaatorit kasutada pea kõigi testide läbiviimisel. Ühe näidistesti läbijooksutamine võiks toimuda alljärgnevalt (vt tabel 14), kasutades IO mälupesade tähistamiseks DI(x), AI(x) ja DO(x) ning protsessisiseste muutujate tähistamiseks M(x). Olukord, kus jälgitakse, et DO(Soojendi pooljuht-lüliti) != true, on toodud joonisel 8.

Testi ID ja pealkiri	T-2. Ventilaatori ja soojendi peatamine ukse avamisel
Testi tegevused	Käivitatakse simulaator. Alguses on kõik DI ja AI väärtused vastavalt 0 või *false*. Soovi korral lisatakse mõnda loogika punkti ka murdepunkt ning täidetakse huvipakkuvat loogikat samm-sammult. • Määratakse AI(T1) = 20 ning AI(T2) = 0.5. • Määratakse DI(Sulatusrežiimi nupp) = *true*.

	• Veendutakse, et mõne aja pärast M(Sulatusrežiim) == *true*. • Veendutakse, et DO(Ventilaatori käiviti) == *true* ning aeg-ajalt DO(Soojendi pooljuht-lüliti) == *true*; • Määratakse DI(Ukseandur) == *true*. • Veendutakse, et DO(Ventilaatori käiviti) == *false* ning DO(Soojendi pooljuht-lüliti) == *false*.
Testi võimalik järeldus	Ukse avanemisel seiskuvad nii ventilaator kui soojendi.

Tabel 14. Testi T-2 läbiviimise kirjeldus



Joonis 8. Näidisprojekti PLC loogika testimine simulaatoris – soojendi väljundi jälgimine

Simulaatori kasutamise tugevused:

- Võimaldab läbi testida praktiliselt igasugust loogikat sobivas tempos.
- Võimaldab teste läbi viia automaatikaseadmeteta, kontorilaua tagant lahkumata.
- Puudub igasugune oht seadmetele või keskkonnale ka kõige raskemate vigade korral.
- Võimaldab loogika toimimist sügavamalt analüüsida seda pausile pannes ning samm-sammult käivitades, kui tootmisprotsess seda võimaldab.

Simulaatori kasutamise puudused:

- Mõningate ajakriitiliste protsesside täielik testimine võib olla keeruline või praktiliselt võimatu, sest puudub võimalus hinnata sündmuste toimumise õigeaegsust.
- Keeruliste tööstusprotsesside puhul võib sisendite ja väljundite haldamine muutuda liialt suureks ülesandeks. Sisendeid võib olla liiga palju, nende väärtusi võib olla vaja muuta liiga tihti selleks, et neid oleks võimalik käsitsi hallata.
- Simuleerimist pole võimalik ühildada riistvaraga isegi osaliselt mitte. Seda ei ole võimalik kasutada koos toote või riistvaraseadmete asendamisega või lihtsa testimisega. Kehtib põhimõte – kas kõik või mitte midagi.
- Käesoleva töö testi T-5 läbiviimisel pole simulaatorist kasu, sest simulaator ei anna juurde mingeid tööriistu, millega oleks võimalik täpseid aegu mõõta.
- Automaattestide läbiviimine pole endiselt võimalik, kuna testimissammud tuleb läbi teha käsitsi.

Ka simulaatoriga testimine pole selline meetod, mis võimaldaks PLC tarkvara igas olukorras põhjalikult testida. Kõik käesolevas töös otsitavale meetodile seatud tingimused pole seega täidetud. Sellegipoolest peaks kõigist käsitletud meetoditest olema simulaatoril testimine esimene meetod, millega testimist alustada. Pärast simulaatoriga testimist saab edasi liikuda teiste meetodite juurde.

3.6 Uus meetod: riistvarast eraldatud loogika mõjutamine välise testimisprogrammi abil

Nagu varasematest meetoditest näha, on keeruliste ja nõudlike tööstusprotsesside testimiseks hädavajalik ilma riistvaraseadmeteta testimine. See on ainus viis, kuidas kõrvaldada oht inimestele, keskkonnale, seadmetele. Käesolevas töös senini käsitletud testimismeetodid pole olnud samas piisavad kõigi olukordade kasutamiseks. Üheks suurimaks pudelikaelaks on ilma riistvarata testimise puhul loogika sisendite käsitsi mõjutamise ning väljundite jälgimise keerukus ja täpsus eriti suuremahuliste või muul põhjusel nõudlike tööstusprotsesside korral.

Käesoleva töö autor pakub PLC tarkvara testimiseks välja uue meetodi, mida pole senises praktikas keegi varem kasutada soovitatud. Selle meetodi põhieesmärk on anda sisendite mõjutamise ja väljundite jälgimise vastutus inimeselt üle programmile. Väljapakutav uus

meetod annab testimisfunktsiooni üle arendaja poolt koostatud tarkvarale, mis paikneb väljaspool PLC keskkonda.

3.6.1 Meetodi kirjeldus

Meetodi tuum seisneb lühidalt järgmises:

- Testimist juhib personaalarvuti (PC) testimisprogramm, mis on arendatud mõnes levinud programmeerimiskeeles (näiteks Java, Python, C#).
- Testimisprogramm on ühendatud kommunikatsiooniliidese kaudu PLC või simulaatoriga.
- PLC IO-seadmeid on võimalik riistvarast jooksvalt kas osaliselt või tervenisti lahti ühendada.
- Testimisprogramm mõjutab ise PLC sisendeid ning jälgib väljundeid.

Meetodil on kolm etappi – PLC ettevalmistamine, testimisprogrammi arendamine ning testide läbiviimine.

3.6.1.1 PLC ettevalmistamine

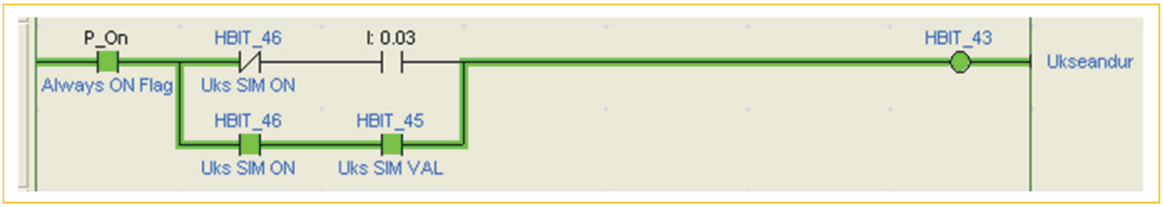
PLC sisenditele vastavad mälupesad ei ole muudetavad ei PLC loogika ega väliste programmide poolt. Simulaatoris on need muudetavad, kuid selleks, et oleks võimalik läbi viia teste ka osalise riistvaraga, et saa simulaatorit kasutada. Väljapakutava meetodi lahendus on sisendid loogikast eraldada vahemuutuja kaudu – `M(Sisend)`. Selle asemel, et loogikas kasutada otse sisendile vastavat mälupesa, näiteks `DI(Sisend)`, tuleb kasutada sellele sisendile eraldatud vahemuutuja mälupesa `M(Sisend)`. Seda vahemuutujat on võimalik siis riistvaraga siduda või mitte siduda. Loogika ise ei „tea“, kas tema sisendsignaal tuleb reaalsest IO-st või selle annab talle ette hoopis mõni programm. Sisuliselt võimaldab see loogikale riistvara olemasolu tarkvaraliselt simuleerida.

Lisaks tuleb iga sisendi kohta kasutusele võtta veel kaks muutujat:

- Sisendi väärtus `M(Sisend SIM VAL)`, mida PLC soovib sisendile määrata – simuleeritud väärtus. Selle muutuja tüüp sõltub sisendi tüübist.

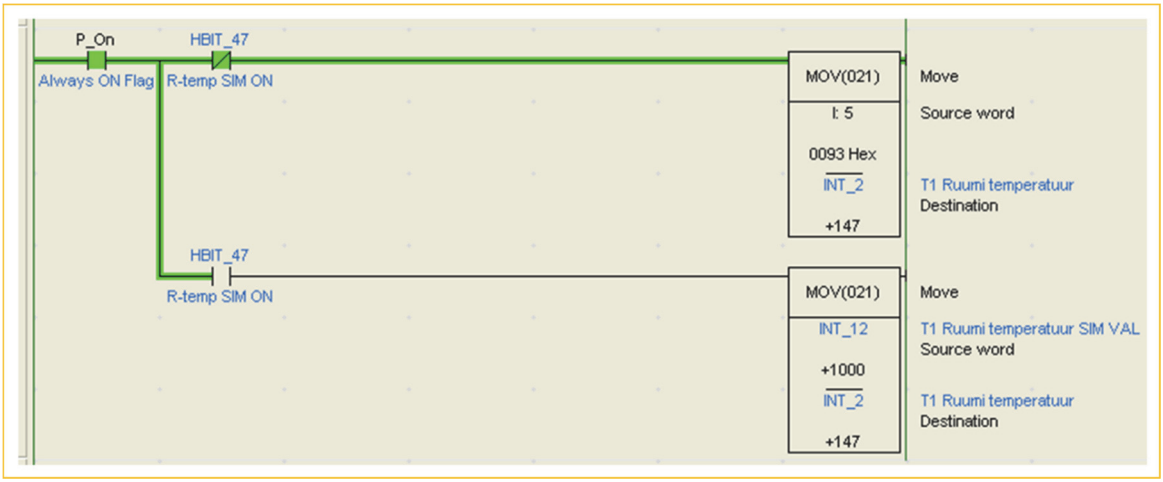
- *Boolean*-tüüpi muutuja M(Sisend SIM ON), millega valitakse, kas muutuja M(Sisend) väärtuseks omistatakse reaalne DI(Sisend) või programmi poolt muudetav muutuja M(Sisend SIM VAL).

Näidisprojekti ukseanduri kui DI eraldamise loogika on kujutatud joonisel 9.



Joonis 9. Näidisprojekti ukseanduri sisendist eraldamise loogika

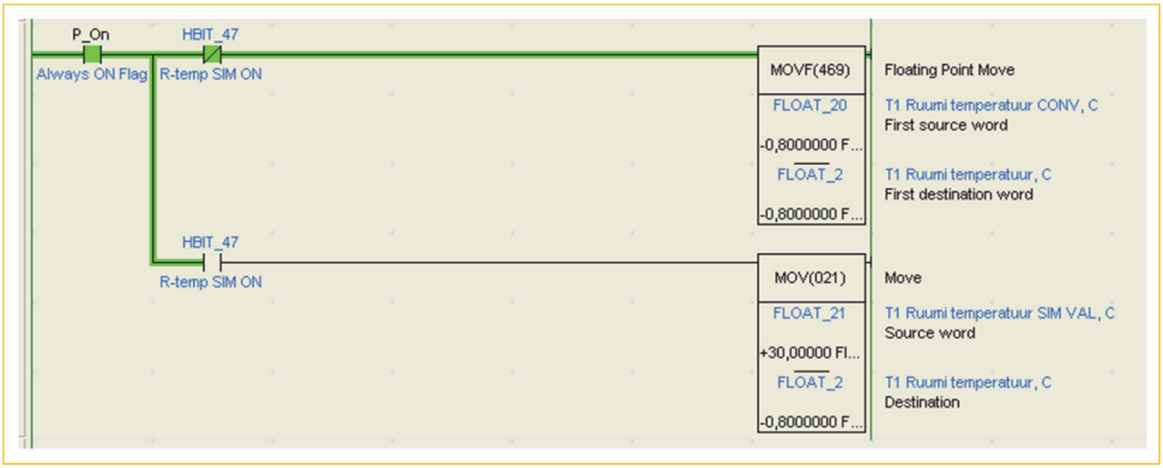
Näidisprojekti temperatuurianduri puhul on sisendi eraldamise põhimõte sama, kuid realisatsioon sõltub sellest, kuidas on AI loogikas kasutusele võetud. Lihtsaim on AI-sid kasutada otse sellisel kujul, nagu neid IO-moodul mälusse salvestab, st kasutades otse AI(Sisend) väärtust. Omroni PLC puhul näiteks teisendatakse AI signaali vahemik 0-20 mA vastavalt täisarvuks vahemikus 0-6000. Selle lihtsa lahenduse puhul on AI eraldamine ülejäänud loogikast kujutatud joonisel 10.



Joonis 10. Näidisprojekti ruumi temperatuurianduri sisendist eraldamise loogika teisendamata analoogsisendi korral

Mõnevõrra keerulisem lähenemine AI kasutamisele on teisendada selle täisarvuline väärtus PLC tarkvara abil inimesele arusaadavale kujule. Näiteks temperatuurianduri poolt väljastatud analoogsignaal teisendatakse näidisprojektis reaalarvuks, mis vastab reaalsele temperatuurile ühikuga *°C*. See teisendus toimub iga sisendi jaoks eraldi, kasutades lihtsaid matemaatilisi

tehteid. Seega, AI(Sisend) teisendatakse ümber ning salvestatakse uude muutujasse nimega M(Sisend CONV), mis on mõeldud PLC loogikas otse kasutamiseks AI(Sisend) väärtuse asemel. Sellise lahenduse korral tuleb AI eraldamiseks ülejäänud loogikast kasutada muutujaid, mille tüübid vastavad teisendatud AI väärtuse tüübile. Näidis on kujutatud joonisel 11.



Joonis 11. Näidisprojekti ruumi temperatuurianduri sisendist eraldamise loogika reaalarvuks teisendatud analoogsisendi korral

Sellised loogikaplokid koos vastavate mälumuutujatega tuleks luua kõigi sisendite jaoks, mida on soov simuleerida. Testimisprogrammiga on võimalik nende muutujate väärtusi mõjutada ning seega PLC loogika sisendi signaali muuta.

Lisaks eelpoolkirjeldatule tuleb PLC-s ette valmistada kommunikatsiooniliidese töö. Omroni PLC-ga on võimalik välistel programmidel suhelda üle FINS protokolli [26]. Füüsilise liidesena saab selleks kasutada *Ethernet* võrku või USB-liidest. Vaikimisi on mõlemad liidesed kasutamiseks lubatud, vaja on sisestada vaid korrektsed võrguseaded. Vastavate liideste ja protokollide valik on eri tootjatel ja PLC mudelitel erinev. [27]

3.6.1.2 Testimisprogrammi arendamine

Nagu varem kirjeldatud, on PLC testimisprogrammi eesmärgiks viia läbi teste, mõjutades PLC sisendeid ja jälgides väljundeid. Selle programmi võib realiseerida arendajale sobivas programmeerimiskeeles sõltumata valitud PLC platvormist, sest programm jookseb eraldi PC, mitte PLC peal.

Selleks, et testimisprogrammil oleks võimalik PLC sisendeid mõjutada ning väljundeid jälgida, peab olema realiseeritud kommunikatsioon programmi ja PLC vahel. Vastavaid

tarkvarafunktsioone ei ole vaja enamasti ise välja mõelda – enamus PLC tootjaid on juba loonud selleks vabalt saadavad teegid või raamistikud, mille abil on oma programmile kommunikatsioonivõime lisamine võrdlemisi lihtne. Standardsete protokollide jaoks on samuti saadaval mitmeid erinevaid ettevõtete või kogukonna poolt arendatud teeke. Halvimal juhul, kui tootja jagab vaid protokolli spetsifikatsiooni, tuleb kommunikatsioon ise realiseerida. Omron näiteks ongi oma FINS protokolli kohta välja andnud vaid põhjaliku spetsifikatsiooni.

Omroni FINS on lihtne protokoll, kus PLC on server ning programm klient. Kogu kommunikatsioon käib lihtsas küsimus-vastus stiilis – klient saadab käsu, server töötleb ning vastab. Käskudega on võimalik PLC mälupesasid lugeda ja kirjutada ükshaaval ning hulgi. Käesolevas töös välja pakutud testimisprogrammi jaoks on see piisav, sest sisendite mõjutamine ning väljundite jälgimine tähendabki praktikas mälupesade lugemist ja kirjutamist.

Kommunikatsiooniteegi funktsioone kasutades on tarkvara arendajal võimalik sobilik testimisprogramm luua väga lihtsaid põhimõtteid järgides:

- Iga PLC sisendi muutujat `M(Sisend SIM ON)` mõjutades saab valida, kas konkreetne sisend on reaalse riistvaraga ühendatud või mitte. See võimaldab hallata kasvõi ainult ühe sisendi väärtust, jättes ülejäänud sisendid IO-seadmete külge.
- Kui `M(Sisend SIM ON) == true`, saab `M(Sisend SIM VAL)` väärtust muutes muuta loogikale edastatava sisendsignaali väärtust.
- Igale väljundile vastava muutuja `DO(Väljund)` või `AO(Väljund)` või suvalise protsessiloogikas kasutatava `M(Muutuja)` väärtust PLC-st lugedes on võimalik jälgida PLC loogika kulgu ning hinnata tulemuse sobivust.

See meetod võimaldab testimisprogrammi liidesena kasutada ükskõik millist liidest. Tulemus võib olla konsooliprogramm, mis sätib ise paika kõik sisendid ning kontrollib väljundeid. Tulemus võib olla ka graafilise kasutajaliidesega, võimaldades testimistegevusi inimesel käsitsi käivitada ning tulemuse hindamiseks vajalikke numbreid silmaga jälgida. Testimisprogramm võib sisaldada graafikuid, sündmuste logisid või teisi analüüsivahendeid. Igal arendajal on võimalik kasutada oma oskustele vastavaid meetmeid.

3.6.2 Näidisrealisatsioon

Käesoleva töö autor realiseeris ise väljapakutud meetodi, et selle kasutusvõimalusi paremini kirjeldada ning analüüsida. Testitava automaatikasüsteemina kasutatakse käesoleva töö näidisprojekti. Realisatsioon hõlmab endas näidisprojekti PLC loogikas muudatuste tegemist, testimisprogrammi disaini ja arendust ning testide läbiviimist.

3.6.2.1 Näidisprojekti PLC loogika muudatused

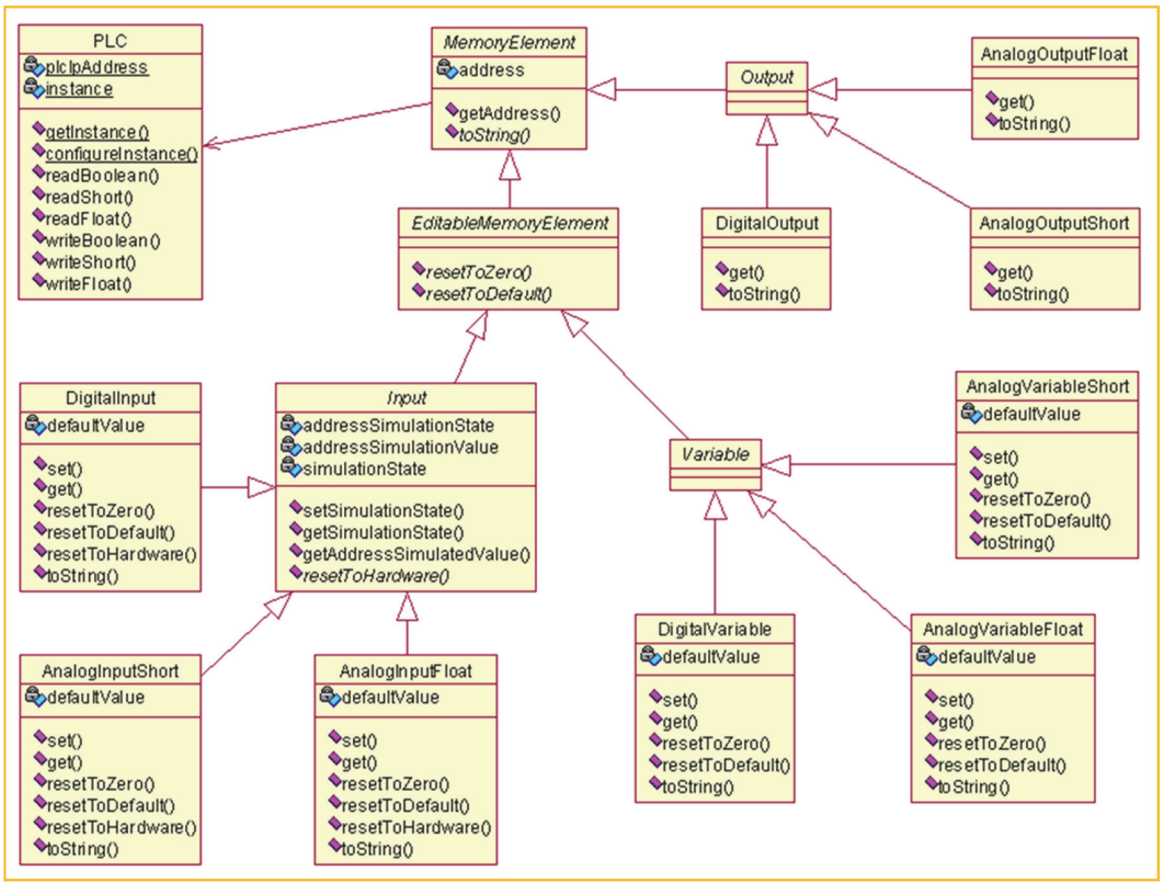
Näidisprojekti PLC loogika ettevalmistamine toimub eelpool kirjeldatud stiilis kõikidele PLC sisendite jaoks (vt jaotis 3.6.1.1). Igale sisendile defineeritakse vastav M(Sisend), M(Sisend SIM ON) ja M(Sisend SIM VAL) ning luuakse uus sektsioon loogikas, kus need muutujad lisatakse sisendi mõjutamise ahelasse. Ülejäänud PLC loogikas kasutatakse vastava DI(Sisend) või AI(Sisend) asemel M(Sisend) muutujat. PLC on PC-ga ühendatud läbi *Ethernet* liidese. PLC tarkvaras tehtavad ettevalmistused on seega võrdlemisi väikesed, eriti kui selle testimismeetodiga arvestada arenduse algusest peale.

3.6.2.2 Testimisprogrammi näidisrealisatsioon

Testimisprogrammi programmeerimiskeeleks on valitud Java, sest käesoleva töö autoril on kogemusi selle keele abil riistvaraga suhtlevate rakenduste loomisel. Java sõltumatus PC platvormist tuleb samuti kasuks. Testimisprogrammi lähtekood on avaldatud internetis. [28] Mõningaid näited lähtekoodist on toodud lisas 1.

Näidisprojekti testimisprogramm on töö autori poolt realiseeritud klasside kogumina, mida on võimalik kasutada kui teeki PLC mälupesade väärtuste jälgimiseks ja muutmiseks. Rõhutada tuleb, et need klassid ei määratle käesolevas peatükis kirjeldatud uut testimismeetodit, vaid on üheks võimalikuks selle meetodi realiseerimise viisiks.

Selleks, et aadresside lugemine ja kirjutamine oleks võimalikult lihtne, tuleb jälgida PLC mälus olevate andmete tüüpi (vaata jaotis 2.3.1). Omroni PLC mälus on võimalik hoida 9 erinevat tüüpi andmeid. [29] Näidisprojekti testimisprogrammi realiseerimiseks on valitud neist kolm, mida kasutatakse ka näidisprojekti PLC loogikas – Boolean (1 bit), Short (16bitine märgiga täisarv), ning Float (32bitine ujukomaga arv). Ülejäänud andmetüübid on realiseerimata näidise lihtsuse huvides. Loodud klassid võimaldavad mälust lugeda ja kirjutada vastavalt valitud andmetüübile. Klasside arhitektuuri on hea vaadata klassidiagrammilt (vt joonis 12).



Joonis 12. Testimisprogrammi realisatsiooni klassidiagramm

Kasutatud klasside lühikirjeldus:

- Klass PLC. Tegemist on *Adapteriga*, mis sisaldab endas Omroni PLC FIOS protokolli kommunikatsioonifunktsioone mälupesade lugemiseks ja kirjutamiseks vastavalt soovitud andmetüübile. Klass kasutab ka *Singleton* mustrit, mis võimaldab teiste klasside poolt selle klassi meetoditele ligi pääseda oluliselt lihtsamalt, sest neile pole vaja eraldi viidet edastada. Selle mustri kasutamine välistab ka mitme PLC objekti samaaegse loomise ning kasutamise. PLC klass ei sõltu teistest näidisrealisatsiooni klassidest.
- Klass MemoryElement. Tähistab ühte muutujat, sisendit või väljundit PLC mälus, mille sisu on kindlasti loetav. Tingimata ei tähista ühte mälupesa. Abstraktne klass.
- Klass EditableMemoryElement. Tähistab sellist muutujat, mille sisu võib lisaks jälgimisele ka muuta. Abstraktne klass.
- Klass Input. Tähistab sellist muutujat PLC mälus, mis on M(Sisend), ning mille jaoks on loodud ka vastavad M(Sisend SIM ON) ja M(Sisend SIM VAL). Sisaldab meetodeid simulatsiooni aktiveerimiseks. Abstraktne klass.

- Klassid `Output` ja `Variable`. Tähistavad vastavalt PLC väljundit või muud muutujat. Vajalikud eelkõige klassifitseerimise ning võimaliku laiendamise või täiendamise eesmärgil. Abstraktsed klassid.
- Klassid `DigitalInput`, `AnalogInputShort`, `AnalogOutputFloat`, jne. Testide läbiviimisel kasutamiseks mõeldud klassid, mis tähistavad konkreetse andmetüübiga sisendeid, väljundeid ning muutujaid. Sisaldavad meetodeid väärtuse lugemiseks ja kirjutamiseks ning realiseerivad ka teistelt klassidelt päritud abstraktseid meetodeid.

Selline võrdlemisi keeruline klasside struktuur ei ole käesoleva testimismeetodi realiseerimiseks hädavajalik, kuid võimaldab PLC muutujatega paindlikumalt ümber käia. Suuremate projektide puhul võib olla näiteks vajalik kõik sisendid ja muutujad kuskile massiivi salvestada, et nende kõigi väärtust nullida või algväärtustada.

3.6.2.3 Näidistesti läbiviimine

Ettevalmistatud raamistikku võib kasutada näidisprojektis testi T-5 läbiviimiseks. Seda testi polnud võimalik teiste käesolevas töös kirjeldatud testimismeetoditega läbi viia, sest testi nõuete täitmist pole võimalik inimsilmaga tuvastada.

Testimisprogrammi olulisim ülesanne on selle testi läbiviimisel võimalikult täpselt ja kiirelt mõõta soojendi väljundi muutumist. Testimisprogrammi abil võib riistvarast lahti ühendada termilise kaitse, kuna seda ei saa muul moel mõjutada. Ülejäänud IO võib jätta riistvara külge ühendatuks. Soovi korral võib soojendi kui täituri füüsiliselt väljundi küljest lahti ühendada, et testimist oleks veel mugavam ning ohutum läbi viia. Testi läbimiseks sobilik minimaalne Java programm on järgmine:

```
PLC.configureInstance("192.168.1.90");

DigitalInput termilineKaitse = new DigitalInput("H7.1", "H8.1", "H9.1");
DigitalOutput soojendi = new DigitalOutput("H10.0");
AnalogVariableFloat soojendiVõimsus = new AnalogVariableFloat("D2010");

termilineKaitse.set(false);

Float ettenähtudVõimsus = soojendiVõimsus.get();

while (soojendi.get()) {}
long time1 = System.nanoTime();
while (!soojendi.get()) {}
long time2 = System.nanoTime();
while (soojendi.get()) {}
long time3 = System.nanoTime();
```

```
long kokkuKestvus = time3 - time1;
long seesKestvus = time3 - time2;
float arvutatudVõimsus = (float)seesKestvus / (float)kokkuKestvus * 100f;

System.out.println(ettenähtudVõimsus + ", " + arvutatudVõimsus + ", "
      + kokkuKestvus);
```

Toodud funktsioon on lihtsustatud ja ebatäiuslik, kuid piisav väljapakutud meetodi illustreerimiseks. Funktsiooni käivitamisel trükitakse konsoolile kolm numbrit, mille põhjal on võimalik hinnata, kas test on läbitud või mitte. Testi T-5 läbiviimise võimalik kirjeldus on toodud alljärgnevas tabelis (vt tabel 15).

Testi ID ja pealkiri	T-5. Soojendi võimsuse reguleerimise vastamine tingimustele
Testi tegevused	Kogu riistvara on PLC sisendite ja väljundite külge ühendatud. Kambri õhu temperatuur on piisavalt madal ning toote temperatuuriandur pannakse jäävee sisse. Suletakse kambri uks. Käivitatakse sulatusrežiim füüsilise nupuga. Käivitatakse arvutis testimisprogramm. Analüüsitakse programmi poolt väljastatud numbreid.
Testi võimalik järeldus	Testimisprogramm väljastab tulemuseks numbrid, mis vastavad oodatud väljunditele.

Tabel 15. Testi T-5 läbiviimise kirjeldus

Testi T-5 eduka läbimise üheks tingimuseks on PLC ja PC vahelise kommunikatsiooniliidese piisav kiirus. Ühe küsimus-vastus tsükli läbimise aeg peab olema väiksem kui nõudes N-5 seatud minimaalne Soojendi seesoleku aeg 0,1 sekundit. Omroni PLC puhul on testid näidanud, et see aeg on testi T-5 läbimiseks piisavalt väike (vt jaotis 3.6.6).

Sama meetodit võib näidisprojektis edukalt kasutada ka kõigi teiste testide läbiviimise hõlbustamiseks. Meetodi kasutamise täiendavaks illustreerimiseks on järgnevalt kirjeldatud mõningaid ülesandeid, mida võiks meetodi põhjal loodud testimisprogramm nende testide läbiviimisel täita:

- T-1 (Ventilaatori töötamine vastavalt režiimile). Režiimide vahetamise simuleerimine erinevas järjekorras lühikeste ja pikkade vahedega. Ventilaatori seisundi jälgimine. Ventilaatori töötamise või seismise minimaalse aja täpne arvutamine.

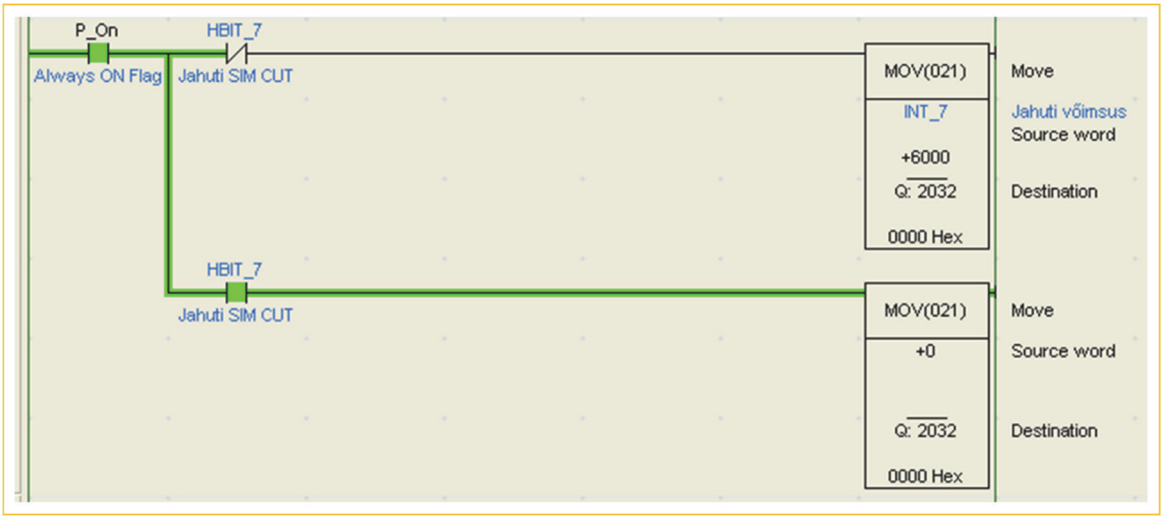
- T-2 (Ventilaatori ja soojendi peatamine ukse avamisel). Ukse avamise simuleerimine. Ventilaatori ja soojendi seisundi jälgimine mingi aja jooksul, kusjuures programm on võimeline tuvastama ka inimsilmale märkamatuks jäävaid soojendi lülitusi.
- T-3 (Sulatustemperatuuri hoidmine). Kambri õhu temperatuuri simuleerimine vastavalt soojendi võimsusele. Algoritm võib olla võrdlemisi lihtne, tõstes kambri temperatuuri proportsionaalselt soojendi võimsusega, simuleerides ka soojakadu. PID-protsesside testimiskeskkonna simulatsiooniks on välja pakutud ka mitmeid algoritme koos valmis tarkvara ja lähtekoodiga. [30]
- T-4 (Ventilaatori termilise kaitsme rakendumine). Ventilaatori termilise kaitsme rakendumise simuleerimine. Ventilaatori ja soojendi seisundi jälgimine pikema aja jooksul.
- T-6 (Automaatne sulatuse lõpetamine). Toote temperatuuri simuleerimine.

3.6.3 PLC loogika väljundite eraldamine riistvarast

Väljapakutud testimismeetodi abil võib sarnaselt sisenditele riistvarast eraldada ka PLC loogika väljundid. Väljundite eraldamine võimaldab jätta täiturid IO-moodulite küljest lahti ühendamata, kui nendele väljundsignaali saatmine välistatud peaks olema. Väljundite eraldamise tugi tuleb sarnaselt sisenditele lisada nii PLC loogikasse kui testimisprogrammi.

Väljundid tuleb (jällegi sarnaselt sisenditele) PLC loogikast eraldada vahemuutuja kaudu – `M(Väljund)`. Selle asemel, et protsessi juhtivas loogikas kasutada otse väljundile vastavat mälupesa, näiteks `DO(Väljund)`, tuleb kasutada sellele sisendile eraldatud vahemuutuja mälupesa `M(Väljund)`. Lisaks tuleb luua iga väljundi kohta üks *Boolean*-tüüpi muutuja `M(Väljund SIM CUT)`. Kui see muutuja on `true`, siis eraldataksegi loogika väljund riistvaralistest väljundist, ehk siis `M(Väljund)` muutuja väärtus jäetakse edasi kandmata `DO(Väljund)` või `AO(Väljund)` mälupesale. Võimalikud vastava loogika näited on toodud alljärgnevatel joonistel (vt joonis 13 ja 14).

Joonis 13. Näidisprojekti soojendi väljundi (DO) eraldamise loogika



Joonis 14. Näidisprojekti jahuti väljundi (AO) eraldamise loogika

Käesolevas töös varem kirjeldatud näidisrealisatsiooni testimisprogrammi disaini (vt jaotis 3.6.2.2) tuleks väljundi eraldamiseks täiendada `Output` klassi meetoditega, mis võimaldaksid iga väljundi `M(Väljund SIM CUT)` muutuja väärtust määrata. Kui näidisrealisatsiooni testimisprogrammis realiseerida selline väljundite PLC loogikast eraldamine, pole vaja näidisprojekti testi T-5 läbiviimisel isegi soojendit ohutuse mõttes IO-mooduli küljest füüsiliselt lahti ühendada (vt jaotis 3.6.2.3).

Joonistelt 15 ja 16 on näha, et loogikast eraldamise korral on DO signaali väärtus pidevalt `false` ning AO signaali väärtus pidevalt `0`. Mõnes projektis võib sellise meetodi rakendamise puhul eksisteerida vajadus mingite muude vaikimisi väärtuste järele. Sellisel juhul tuleb ka PLC loogikat ning testimisprogrammi vastavate muutujate ja meetoditega täiendada.

3.6.4 Kasutamine simulaatoriga

Väljapakutav meetod on väga hästi kasutatav ka PLC simulaatoriga. Testimisprogrammil pole vahet, milline riistvara sisendite põhjal väljundeid mõjutab. Omroni PLC simulaator simuleerib ka kommunikatsiooniprotokolli, mistõttu pole testimisprogrammi kommunikatsiooniteeke vaja simulaatoril testimiseks üldse vahetada. Mõne teise tootja PLC puhul ei pruugi olla simulaatoriga väljapoolt ühendumine paraku võimalik.

Simulaatoriga testides on võimalik loobuda ka varem kirjeldatud (vt jaotis 3.6.1.1 ja 3.6.3) muutujate `M(Sisend)`, `M(Sisend SIM VAL)`, `M(Sisend SIM ON)`, `M(Väljund)`, `M(Väljund SIM CUT)` kasutamisest, kuna riistvaraseadmeid pole niikuinii taha ühendatud. Sellisel juhul pole PLC

ettevalmistamiseks tarvis pea midagi teha ning ka testimisprogrammi tarkvara on lihtsam. Samas ei ole see soovitatav, kuna siis on vaja hiljem PLC peal testide läbiviimiseks need muutujad sellegipoolest PLC loogikasse sisse kirjutada.

Simulaatoriga testides on võimalik kasu lõigata ka asjaolust, et simulaator võimaldab loogikat soovitud kohas pausile panna ning samm-sammult käivitada, võimaldades loogika toimimisse paremini süveneda ja sisendite mõjutamise testimisprogrammile jätta.

3.6.5 Kasutamine automaattestide tegemiseks

Käesolevas peatükis välja pakutud uut praktilist testimismeetodit, mille korral mõjutatakse riistvarast eraldatud loogikat välise testimisprogrammi abil, on lihtne kasutada ka automaattestide läbiviimiseks. Kui näitena toodud näidisprojekti testimisprogrammi (vt jaotis 3.6.2.2) lisada objektid kõigi sisend-seadmete jaoks, nende vajalikud väärtused programmi alguses ära määrata ning lõpus saadud väärtusi oodatavate väärtustega võrrelda, ongi sisuliselt tulemuseks automaattest. Sel viisil on võimalik automatiseerida kõik käesolevas töös kirjeldatud näidisprojekti testid.

Käesolevas töös käsitletud testimismeetoditest on tegu ainsa meetodiga, mis võrdlemisi lihtsalt ja loogiliselt automaatteste läbi viia võimaldab. Selliseid automaatteste on võimalik läbi viia nii füüsilise PLC peal kui PLC simulaatori peal. Simulaatori peal on neid teste võimalik käivitada ka kui regressiooniteste.

Väljapakutud testimismeetod annab arendajale võimaluse automaattestide korraldamisel kasutada tuntud testiraamistikke (näiteks Java puhul ühiktestiraamistik *JUnit*). Tuttavate tööriistade kasutamine tähendab väiksemat ajakulu. Testi T-5 baasil loodud näidistesti lähtekood on toodud lisas 1.

Väljapakutud meetod võimaldab põhimõtteliselt PLC tarkvara arendusel kasutada teatud mööndustega ka testidest juhitud tarkvaraarendust, mille puhul luuakse kõigepealt test ning alles seejärel vastav funktsionaalsus.

Meetodiga on võimalik läbi viia ka selliseid automaatteste, mille põhieesmärk on riistvaravigade avastamine, kui seda võimaldavad ka tootmis- või automaatikaseadmed. Näiteks võib olla tehases vajalik aegajalt üksvaaval täitureid diagnostika mõttes läbi käia ning mõõta anduritega nende töötamise parameetreid. PLC-ga on selliste katsete läbiviimise

programmeerimine väga keeruline ning aeganõudev, mistõttu tehakse selliseid teste pigem käsitsi. Väljapakutud meetodiga on see aga tunduvalt lihtsam.

Väljapakutud meetodit saab kasutada veel ühel kasulikul viisil. Nimelt on võimalik testimisprogrammis realiseerida funktsionaalsus, mille eesmärk on jälgida normaalselt töötavat tööstusprotsessi, salvestades kõigi sisendite ja väljundite väärtused. Seejärel on võimalik hiljem seda salvestist nii-öelda maha mängides anda PLC loogikale ette juba reaalsest elust pärit sisendite väärtused ning kontrollida, kas loogika poolt mõjutatavad väljundite väärtused vastavat salvestatud väljundite väärtusega. Sel viisil on võimalik läbi viia ka teste, mille sisendandmeid on raske või võimatu algoritmidega simuleerida. Mõnevõrra sarnaseid meetodeid on välja pakutud reaalajasüsteemide juures kasutamiseks mitmete autorite poolt (vt jaotis 2.8.2). [31]

3.6.6 PLC ja testimisprogrammi vahelise kommunikatsiooni kiirus

Käesolevas töös väljapakutud meetodi kasutamise võimalused sõltuvad suurel määral PLC ja testimisprogrammi vahelise kommunikatsiooni kiirusest. Kui testimisprogrammil kulub PLC-st muutuja väärtuse lugemiseks aega vähe, on meetodi abil võimalik jälgida ja analüüsida ka väga kiiresti toimuvaid tööstusprotsesse. Kui see ajakulu on võrdlemisi suur, võib selliste protsesside jälgimine olla problemaatiline.

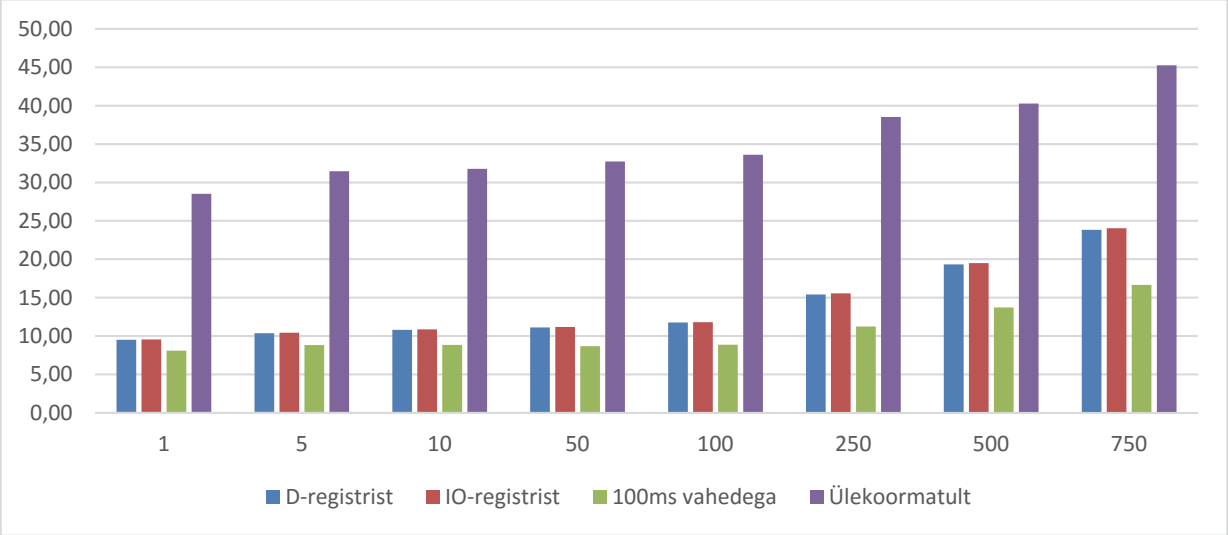
Käesoleva töö autor tegi PLC ja testimisprogrammi vahelise kommunikatsiooni kiiruse hindamiseks katsetusi reaalses tehases töötava Omroni PLC peal. PLC mudel oli *CJ2H CPU67-EIP*. Tegu on Omroni PLC valiku keskklassi kuuluva mudeliga. Läbiviidud katsetused ei võimalda küll teha põhjalikke järeldusi kõigi PLC tootjate ja mudelite kohta, kuid annavad siiski mingisuguse pildi olukorrast.

Kokku teostati 36 erinevat katset. Iga katse käigus loeti käesolevas töös kirjeldatud testimisprogrammiga PLC mälupesi 300 korda järjest ning arvutati keskmine aeg, mis kulus päringu väljasaatmisest vastuse saamiseni. Kuna ühe päringuga on võimalik lugeda ka rohkem kui ühe järjestikkuse mälupesa väärtust (maksimum on ligi 1000 [26]), teostati katseid erinevate mälupesade ehk sõnade arvuga. Andmeid loeti kahest erinevast registrist, et tuvastada võimalikud erisused registrite vahel. Lisaks jäeti mõningate katsete puhul iga järjestikkuse lugemise vahele 100 ms ooteaega, mõningate katsete puhul koormati aga PLC

kommunikatsioonimoodul üle mitme paralleelse lugemiskatsega. Katsetuste tulemused on toodud alljärgnevas tabelis (vt tabel 16) ning graafikul (vt joonis 15).

Loetud sõnu	1	5	10	50	100	250	500	750
D-registrist	9,52	10,38	10,80	11,13	11,76	15,41	19,33	23,84
IO-registrist	9,56	10,44	10,88	11,19	11,82	15,55	19,51	24,05
100ms vahedega	8,10	8,82	8,86	8,68	8,87	11,25	13,72	16,67
Ülekoormatult	28,53	31,45	31,76	32,74	33,60	38,53	40,27	45,25

Tabel 16. PLC-st lugemispäringule vastuse saamiseks kulunud keskmine aeg (ms)



Joonis 15. PLC-st lugemispäringule vastuse saamiseks kulunud keskmine aeg (ms) vastavalt loetud sõnade arvule

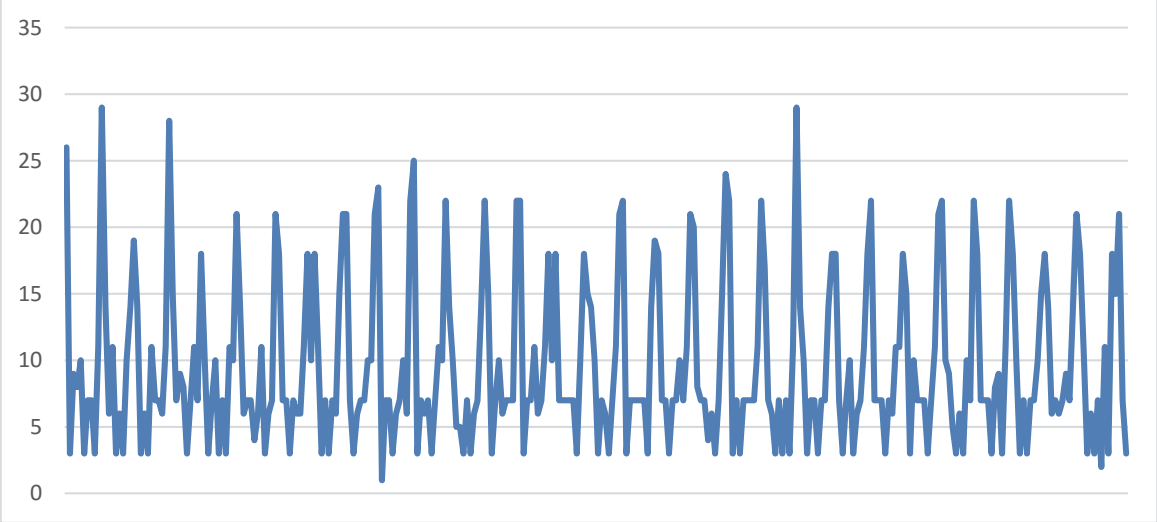
Tulemustest on näha, et lugemispäringu täitmisele kuluv aeg ei ole proportsionaalne loetud mälupesade arvuga. Mitme mälupesa korraga lugemine on oluliselt ajaefektiivsem kui mälupesade ükshaaval lugemine. Seetõttu on ajakriitiliste tööstusprotsesside testimisel mõistlik jälgitavad väärtused mälus järjestikku hoida. On näha ka seda, et pole vahet, millisest registrist andmeid lugeda.

PLC kommunikatsiooniressurss ei ole piiritu. Kui PLC-st andmete lugemisega tegelesid paralleelselt nii *CX-Programmer*i arenduskeskkond, *Omroni CX-Supervisor* platvormil põhinev SCADA kasutajaliides ning töö autori testimisprogramm, tekkisid kommunikatsioonis ilmselged lisaviivitused. Samas, päringute vahele lisatud puhkeaeg võimaldas PLC-l päringuid

mõnevõrra kiiremini täita. Seega sõltub kommunikatsiooni kiirus ka PLC kommunikatsiooni-mooduli hetkekoormusest.

Tulemustest on näha, et üksiku mälupesa lugemiseks kulub aega keskmiselt 9,52 ms, mis on üldiselt väga hea tulemus, kuid mitte ideaalne. Katsetuste läbiviimise ajal oli PLC tarkvara töötsükli kestvuseks 3,69 ms – seega ei pruugi olla võimalik testimisprogrammiga tuvastada olukorda, kus mingi muutuja väärtus muudetakse vaid üheks töötsükliks. Samas on seda võimalik arvesse võtta PLC tarkvara arendamisel ning kasutada pigem muutujaid ja loogikat, mille muutumist on võimalik kindlasti registreerida.

Üks võimalik murekoht PLC ja testimisprogrammi vahelises kommunikatsioonis on päringu täitmiseks kuluva aja suhteliselt suur kõikumine. D-registrist ühe mälupesa lugemisele kuluva aja standardhälve 300 järjestikkuse lugemise korral oli 5,9 ms ning 750 mälupesa lugemisel 9,3 ms. D-registrist ühe mälupesa lugemiseks kuluva aja graafik on toodud allpool (vt joonis 16). Graafikult on selgelt näha, et lugemiseks kuluva aja kõikumine on korduva iseloomuga, mis viitab sellele, et PLC poolt on nende lugemispäringute teenindamine piiratud mingite regulaarselt aset leidvate sisemiste protsessidega.



Joonis 16. PLC-st lugemispäringule vastuse saamiseks kulunud aja muutus läbi järjestikkuste päringute ühe mälupesa lugemisel D-registrist (ms)

Üldiselt võib katsetest järeldada, et töös välja pakutud meetod on Omroni PLC puhul PLC ja testimisprogrammi kommunikatsiooni kiiruse seisukohast hästi kasutatav meetod. Enamuste automaatikaprotsesside jälgimisel oleks isegi 100 ms viivitust aktsepteeritav, eriti kui ühe

päringuga lugeda mitme muutuja väärtus korraga. Mõõdetud 9,52-45,25 ms viivitused võimaldavad väga täpselt jälgida ka selliseid protsesse, mida inimsilm ei näe.

Väljapakutud testimismeetodi näidisrealisatsiooni testimisprogrammi disaini (vt jaotis 3.6.2.2) tuleks praktikas kasutamiseks kindlasti täiendada funktsioonidega, mis võimaldaksid PLC-st mitme muutuja väärtust korraga lugeda.

3.6.7 PLC mälu- ja ajakaotus meetodi kasutamisel

Käesolevas töös väljapakutud testimismeetod näeb ette PLC loogika täiendamist testimismeetodi jaoks vajalike muutujate ning loogikaga. Töö autor uuris, kui palju mõjutavad need meetmed PLC tarkvara normaalset tööd.

Meetodi kohaselt vajab iga sisend minimaalselt kolme uut muutujat (vt jaotis 3.6.1.1). Maksimaalselt tähendab see 8 sõna ja 1 bitti, kui sisendi teisendatud muutujatüüp vajab nelja sõna (64 bitti). Odavamatel PLC mudelitel on mälu 2x32768 sõna, kallimatel kuni 26x32768 sõna. Suurte andmevajaduste jaoks (näiteks andmete logimine) kasutatakse veel lisaks mälukaarte. Muutujate vabale mäluruumile selle testimismeetodi vajadused seega olulist mõju ei avalda.

Programmimälu suurust kirjeldatakse Omroni PLC puhul sammudes (1 samm on 32 bitti). Üks väljapakutud testimismeetodi jaoks vajalik DI eraldamise loogikasektsioon (vt joonis 11) vajab 8 sammu, AI eraldamise loogikasektsioon (vt joonis 12) vajab 10 sammu programmimälu. Kui eraldamise jaoks kasutada funktsiooniplokki, kulub DI puhul ploki definitsiooniks 19 sammu ning väljakutsumiseks iga sisendi kohta 27 sammu. Odavamatel PLC mudelitel on 2000 kuni 5000 sammu programmimälu, kallimatel 50000 kuni 400000 sammu. Odavamate PLC-mudelite puhul võib seega mäluruumi kitsikus olla reaalne probleem, kuid see on siiski ebatõenäoline, sest enamasti on nende kasutamise korral ka IO-seadmeid vähem ning tööstusprotsessid lihtsamad.

Käesoleva töö autor viis läbi katsed reaalses tehases töötava Omroni PLC peal (vt jaotis 3.6.6), et hinnata väljapakutud testimismeetodi mõju PLC tarkvara töötsükli kestvusele (vt jaotis 2.3.3). Töötsükli kestvust on võimalik jälgida Omroni PLC puhul *CX-Programmer* arenduskeskkonnast, kus kuvatakse kord sekundis eelmise sekundi keskmist töötsükli kestvust. Vaadata saab ka töötsükli kestvuse miinimum- ja maksimumväärtust.

Katsete läbiviimine kinnitas eeldust, et töötsükli kestvus sõltub otseselt PLC poolt läbikäidavate loogikasammude arvust. Seega, kui testimismeetodi tõttu juurde lisatud sammude arv on ülejäänud loogika sammude arvuga võrreldes märkimisväärne, on ka mõju töötsükli läbimise ajale tuntav. Samas on see mõju võrreldav või väiksem loomulikust töötsükli läbimise aja kõikumisest, mis tuleneb parasjagu täitmist vajavate loogikasammude arvu muutumisest vastavalt loogikale endale. Tööstusprotsessides, kus on vaja mingil põhjusel konstantset töötsükli kestvust, fikseeritakse see nagunii konkreetse ja piisavalt suure väärtuse peale.

Katsete läbiviimine näitas veel seda, et töötsükli läbimise aeg ei sõltu üldse PLC-le saadetud kommunikatsioonipäringute täitmisest. Töötsükli kestvus jäi muutumatuks ka siis, kui ühe päringu täitmisele kulus ülekoormuse tõttu tavaolukorrast kordades rohkem aega. Muutumatuks jäi nii kestvuse keskmine kui miinimum- ja maksimumväärtus.

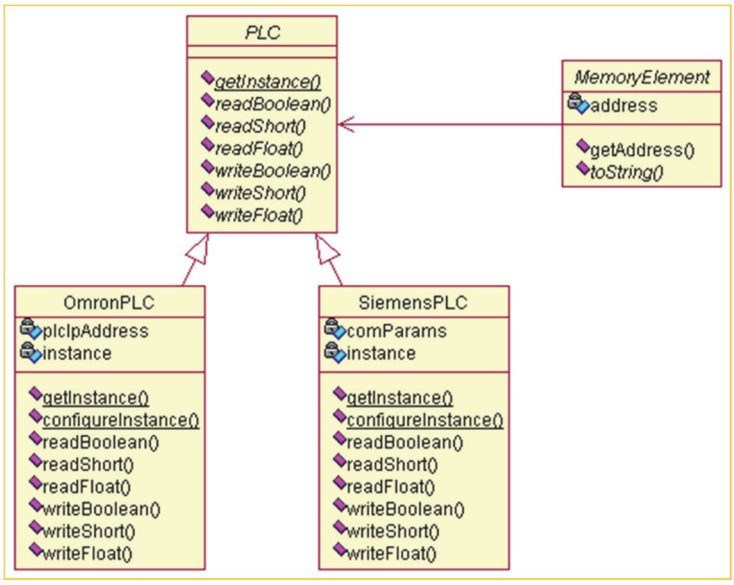
3.6.8 Meetodi korduvkasutamine ja laiendamine

Meetodi, eriti testimisprogrammi realisatsioon võib olla mitmesugune. See programm võib olla universaalne, aga võib olla ka väga spetsiifiline. Näiteks võib testimisprogrammi arhitektuur olla otseselt seotud PLC tarkvara arhitektuuriga või projekti probleemivaldkonna arhitektuuriga. Loodud tarkvaraklassid võivad sisendite ja väljundite asemel tähistada mootoreid, klappe ja teisi probleemivaldkonna objekte. PLC loogika põhjalikuks läbitestimiseks võib see olla igati mõistlik ja vajalik. Samas võib olla valminud spetsiifilise testimisprogrammi raamistik hõlpsasti taaskasutatav ainult väga sarnase valdkonna projektide juures, kui sealgi. Teisest küljest vaadatuna võivad sellised keerulised klassid moodustada vastavate PLC moodulitega ühise terviku, mida on hea ka koos uuesti kasutusele võtta.

Eelpool kirjeldatud näidisrealisatsiooni teed minnes (vt jaotis 3.6.2.2) on tulemus aga oluliselt universaalsem. See raamistik kirjeldab kasutamiseks lihtsad klassid ja meetodid, mis pole üldse konkreetsest projektist sõltuvad. Lisaks on neid klasse või kirjeldatud abstraktseid alusklasse võimalik laiendada uute klassidega, et keerulisemate projektide puhul testimist hõlbustada. Seega on seda raamistikku võrdlemisi lihtne kasutusele võtta iga projekti puhul.

Näidisrealisatsiooni piirab praegusel kujul asjaolu, et see on loodud Omroni PLC jaoks. Samas on väljapakutud lahenduse disaini juures silmas peetud, et seda oleks lihtne laiendada ka teiste tootjate PLC-de jaoks. PLC klassi puhul on selleks kasutatud *Adapter* mustrit, mille eesmärk on luua testimisprogrammi jaoks ühtne PLC kommunikatsiooni liides PLC

platvormist sõltumata. Näidisrealisatsiooni klasse võiks seega täiendada PLC alamklassidega iga PLC tootja platvormi jaoks eraldi. Praeguse PLC klassi sisu tuleks üle viia uude Omroni PLC klassi. Võimalik PLC kommunikatsiooniklasside näidisarhitektuur on toodud allpool klassidiagrammil (vt joonis 17).



Joonis 17. Näidisrealisatsiooni kommunikatsiooniklasside klassidiagramm mitme erineva PLC platvormiga ühendamise võimaldamiseks

Üks võimalus väljapakutud testimismeetodi laiendamiseks on kasutada seda ka SCADA loomisel. PLC tootjate poolt pakutavad SCADA loomise vahendid on tihti piiratud ega sobi hästi kõigi probleemivaldkondade puhul. PLC testimiseks loodud testimisprogrammi raamistiku abil on võimalik luua kasutajaliidest arendajale sobivas keskkonnas, näiteks *Microsoft Visual Studio*. Testimisfunktsionaalsuse saaks siis otse SCADA sisse integreerida. Selline kaks-ühes lahendus võimaldab potentsiaalselt aega säästa ning ka testide kvaliteeti tõsta. Lisaks on võimalik sellise lahenduse puhul hõlpsalt kasutajaliidest ennast ning kasutajaliidese ja PLC ühistööd testida.

3.6.9 Meetodi sobivus

Väljapakutud testimismeetod täidab kõiki tingimusi, mis on käesoleva töö poolt otsitavale testimismeetodile seatud. Meetodi olulisemad tugevused:

- Võimaldab keerukaid tööstusprotsesse mõjutada ning jälgida palju täpsemalt kui inimene. Kiirete tööstusprotsesside tarkvara testimine pole selle meetodiga probleem.

- Võimaldab teste läbi viia nii täiesti riistvarast sõltumatult kui osalise riistvaraga, lubades tarkvara testida olukorras, kus PLC on puudu või on osa riistvaraseadmeid paigaldamata. Ühtlasi võimaldab sellega kõrvaldada testimisvigade esinemisel eksisteeriva ohu seadmetele, keskkonnale ning inimestele.
- Ainus meetod, millega on võimalik kõigis olukordades ka automaatteste läbi viia, mis avab täiesti uue ja olulise testimisvõimaluse PLC tarkvaraarendajate jaoks.
- Ühtemoodi sobilik nii väikeste ja lihtsate kui suurte ja keeruliste automaatika-süsteemide tarkvara testimiseks.
- Väga paindlik, võimaldades igal arendajal kasutada programmeerimiskeelt ning tööriistu, mis talle tuttavad on. Sellega on võimalik olulisel määral kompenseerida neid piiranguid, mille seab testimisele PLC arenduskeskkond (vt jaotis 2.6.1).
- Kergesti korduvkasutatav ja laiendatav.
- Sobib hästi kasutamiseks koos kõigi käesolevas töös käsitletud teiste testimismeetoditega. Seda illustreeris ka testprojekti põhjal toodud näide.
- Sobib teiste meetoditega testimata jäänud testi T-5 läbimiseks.

Väljapakutud meetod omab ka puudusi:

- Vajab arendaja poolt ühe täiendava programmeerimiskeele kasutamise oskust.
- PLC tarkvarasse selle meetodi puhul tehtavad täiendused võivad pikendada tarkvara töötsükli läbimise aega (vt jaotis 3.6.7). Kui see on probleemiks, võib neist loobuda ning meetodit vaid simulaatoriga koos kasutada.
- PLC poolt testimisprogrammile andmete saatmiseks kuluv aeg kõigub üsna palju (vt jaotis 3.6.6).
- Arenduskeskkonna ülesseadmine ning testide programmeerimine võib olla aega-nõudev ja kulukas.

Tabelist 17 on kokkuvõtlikult näha väljapakutud uue meetodi võrdlus kõigi teiste käesolevas töös käsitletud ning praktikas kasutatavate testimismeetoditega.

Olukord	Lihtne	Asendus	Diagnost.	Sim.	Uus
Lihtne loogika, ohtu pole, seadmed koos	Jah	Jah	Jah	Jah	Jah
Seadmetega koos	Jah	Jah	Jah	Ei	Jah
Osaliste seadmetega	Ei	Jah	Jah	Ei	Jah
Ilma seadmeteta koos PLC-ga	Ei	Raske	Ei	Ei	Jah
Ilma seadmeteta ja ilma PLC-ta	Ei	Ei	Ei	Jah	Jah
Kiirust või täpsust nõudvad protsessid	Ei	Ei	Jah	Ei	Jah
Puudub toode või tooraine	Ei	Jah	Ei	Jah	Jah
Oht seadmetele	Ei	Jah	Ei	Jah	Jah
Oht keskkonnale	Ei	Jah	Ei	Jah	Jah
Oht inimestele	Ei	Jah	Ei	Jah	Jah
Keeruline loogika, mahukas projekt	Jah	Ei	Ei	Raske	Jah
Automaattestid	Ei	Ei	Ei	Ei	Jah

Tabel 17. Töös käsitletud testimismeetodite rakendatavus testimisolukorrast lähtuvalt

Väljapakutud meetod on mõningatele puudustele vaatamata väga universaalne ning kõigis käesoleva töö poolt kirjeldatud olukordades oluliselt paremini rakendatav kui eelnevalt käsitletud ning antud valdkonna praktikas kasutatavad teised testimismeetodid. Parima tulemuse saab siis, kui väljapakutud meetodit rakendada koos teiste testimismeetoditega, vastavalt vajadusele.

3.7 Näpunäited tarkvara kvaliteedi tõstmiseks ning hilisema testimise hõlbustamiseks

Selles peatükis on välja toodud mõned näpunäited, mida on soovituslik jälgida PLC tarkvara arenduse juures, et eelpool kirjeldatud testimismeetodite rakendamine tõhusam ning lihtsam oleks.

3.7.1 Modulaarne tarkvara ülesehitus

PLC tarkvara mõistlik tükeldamine on kasulik mitmel põhjusel. Moodulitesse jaotatud tarkvara on oluliselt lihtsam osade kaupa testida. Kui üks moodul on põhjalikult läbi testitud, võib sellele järgmist moodulit testides juba kindlamini toetuda. Tarkvara moodulite põhjal on mugav luua ka automaatseid ühikteste.

Moodulid peaksid olema võimalikult iseseisvad. Näitena võib tuua mingi üksiku seadme (pump, klapp) juhtimise või iseseisva tööstusliku protsessi. Korduvkasutatava mooduli peaks looma funktsiooniplokina, millel on oma sisendid ja väljundid. Üksikute riistvaraseadmete juhtimiseks kasutatavaid mooduleid nimetatakse ka ***draiveriteks***. Protsessijuhtimise moodul võib olla ka lihtsalt eraldi sektsioon või loogikaplokk, millel omaette sisendeid ja väljundeid ei ole. [32]

Käesolevas töös väljapakutud testimistarkvara realiseerimiseks vajalikke PLC mooduleid (vt jaotis 3.6.1.1) on võimalik realiseerida kahel viisil. Esimene võimalus on IO-sisendite sidumine vahemuutujatega teostada ühe suure moodulina kõikide sisendite jaoks. Sellisel juhul peab kohe PLC tarkvara alguses olema üks sektsioon, kus on palju ridu, mis sarnanevad joonisele 11. Teine võimalus on integreerida need vahemuutujad ülalmainitud ***draiveritega*** – funktsiooniplokkidega, mis tegelevad ühe füüsilise seadme sisendite ja väljunditega.

Näiteks võib olla vajalik temperatuurianduri jaoks niikuinii luua funktsiooniplokk, mille ülesanne on arusaamatust AI signaali väärtusest teisendada arusaadav temperatuur. Kui sellele plokile lisada ka sisendsignaali simuleerimisega seotud vahemuutujad, võib tulemuseks olla järgmiste sisendite ja väljunditega funktsiooniplokk:

- Sisendid:
    - `AI(Sisend)` – analoogsisendi signaali väärtus
    - `M(Sisend SIM ON)` – sisendsignaali simuleerimise sisse lülitamine
    - `M(Sisend SIM VAL)` – simuleeritud sisendsignaali väärtus välise testimis-programmi poolt muutmiseks
    - `M(Sisend MIN)` – sisendi teisendamise miinimum
    - `M(Sisend MAX)` – sisendi teisendamise maksimum
- Väljundid:

- M(Sisend) – loetavale kujule teisendatud temperatuur, mis võib olla teisendatud AI(Sisend) või M(Sisend SIM ON) väärtusest, sõltuvalt M(Sisend SIM ON) määratlusest.
- M(Sisend VIGA) – võimaldab edasi anda teisendus- või sisendiviga.

Sellise lähenemise puhul on ühe füüsilise seadme haldamisega seotud loogika ja mälupesad/muutujad kõik ühes kohas koos. Ainus muutuja, mida mujal moodulites kasutatakse, on M(Sisend) ning võib-olla ka M(Sisend VIGA). Samas võib eksisteerida ka selliseid IO-seadmeid, mille jaoks pole mõistlik eraldi funktsiooniplokki luua. Selliste seadmete sisendite eraldamine loogikast peab siis toimuma ikkagi kuskil mujal, mis võib olla halb tarkvara järjepidevuse suhtes. Lisaks ei pruugi iga arendaja või tellija soovida seda, et olulises loogikamoodulis on sisse ehitatud testimiseks mõeldud funktsionaalsust.

3.7.2 Mälu korrastatus

Kuna PLC tööpõhimõte on seotud mälust lugemise ja kirjutamisega, on väga oluline mäluruumiga korrektselt ümber käia. Tähelepanu tuleb osutada nii muutujate tüübile, nimele kui valitud aadressipiirkonnale.

PLC mälu iseärasusi on käesolevas töös juba kirjeldatud (vt jaotis 2.3.2). Üks kirjeldatud põhimõte on, et muutujad on vaid kasutaja jaoks ning programmi jaoks on muutujatüübid määratletud kasutatud käskudega. See tähendab, et loogikat arendades tuleb olla väga tähelepanelik muutujatüüpide suhtes. Näiteks temperatuurianduri analoogisendi teisendamisel reaalarvuks tuleb meeles pidada, et reaalarvu pikkuseks Omroni PLCs on kaks mälupesa ehk neli baiti. See tähendab, et reaalarvu kopeerimiseks tuleb MOV käsu asemel kasutada MOVF käsku, vastasel juhul kopeeritakse vaid esimene pool muutujast. Ehk siis, kui mälupessa D50 salvestada reaalarv, salvestatakse pool sellest ka pessa D51 ning järgmise reaalarvu saab salvestada alles pesasse D52. Pesa D50 jaoks võib olla tekitatud küll õige tüübiga muutuja, kuid seda infot loogika ei kasuta. Selliseid konflikte ei otsi ka arenduskeskkond ise, mistõttu tekivad vead kergesti. Siemensi PLC-de lähenemine on mõnevõrra erinev, kuid sarnased probleemid eksisteerivad sealgi. [3]

Muutujate nimede puhul tuleb meeles pidada seda, et LD ning FBD programmeerimiskeelte puhul roheliste joontega värvitud loogikas hea ülevaate saamiseks ei tohi muutujate nimed olla väga pikad. Joonisel 10 (vt jaotis 3.5) on hästi näha, kuidas muutujanimede lõpud ära kaovad. Nimedele saab küll ruumi juurde anda, kuid mitte lõputult. Testimise käigus hea

ülevaate saamiseks on oluline, et näha oleks võimalikult palju korraga ning kõik oleks võimalikult hästi arusaadav.

Muutujate aadressipiirkonna valimisel tuleb jälgida kolme aspekti:

- Püsiv või mittepüsiv mälu. Olemas on mõlemat tüüpi mälupiirkondi ning nende vahel tuleb valida lähtuvalt konkreetsest probleemist ja selle lahendusest. Eksisteerib muutujaid, mille väärtus peab alles jääma pärast voolukatkestust. Eksisteerib ka muutujaid, mille väärtus ei tohi kindlasti alles jääda pärast voolukatkestust.
- Vaba või mittevaba mälu. Osa mälupiirkondi on reserveeritud IO-moodulite signaalide või muude süsteemimuutujate jaoks. Teine osa mälupiirkondi võib olla kasutuses funktsiooniplokkide muutujate jaoks. Kolmas osa võib olla arendaja enda poolt varasema loogikaga hõivatud. Kasutatud mälupesade suhtes tuleb olla väga ettevaatlik, sest PLC loogika ise selles osas kaasa ei aita. Arenduskeskkonnas on siiski tööriistu, mis aitavad mälu kasutusest ülevaadet saada.
- Muutujad, mille väärtust on vaja testimisprogrammil lugeda, peaksid paiknema mälus lähestikku, et neid oleks võimalik ühe päringuga lugeda (vt jaotis 3.6.6).

Nendest tingimustest lähtuvalt on soovituslik iga PLC tarkvaraprojekti mälu hoolikalt planeerida, kasutades selleks vajadusel väliseid abivahendeid (näiteks Exceli tabel). See aitab tarkvaravigade arvu oluliselt vähendada ning lihtsustada testimist. Mäluga seotud vigade avastamine on väga keeruline, mistõttu on nende ennetus vajalik.

3.7.3 Tarkvara genereerimine programmiliselt

Üks võimalus lihtvigade vähendamiseks ning testimise hõlbustamiseks on osa PLC tarkvara loogikast genereerida programmiliselt. See tähendab, et loogika genereerib mingi lihtne tarkvara ning see kopeeritakse PLC arenduskeskkonda.

Selline meetod on kasutatav eelkõige palju rutiinset tööd sisaldava töö asendamisel. Käesoleva töö kontekstis võiks seda kasutada moodulite loomise ning mälu korrastatuse tagamise juures (vt jaotised 3.7.1 ja 3.7.2). Kui seadmete funktsiooniplokke ning seotud muutujaid on vaja tekitada suures koguses, näiteks on riistvaraseadmeid väga palju, võib kergesti tekkida tarkvarasse lihtsaid näpuvigasid. Neid vigu on võimalik vähendada, kui võimalikult palju tarkvara genereerida programmiliselt.

Üks võimalik meetod selliseks tarkvara programmiliseks genereerimiseks on kasutada näiteks Exceli tabelit ning selle juurde loodud lihtsat tarkvara, mis tabelis kirjeldatu põhjal tekitab kõik vajalikud PLC mälu muutujad, funktsiooniplokkide väljakutsed ning ka käesolevas töös kirjeldatud välise testimisprogrammi (vt jaotis 3.6) jaoks vajalikud objektid. Kui mäluaadresside jaotamise, muutujate tekitamise ning nende muutujate kasutuselevõtu teeb inimese asemel tarkvaraprogramm, on palju vähem ohtu rutiinsusest tulenevate vigade tekkimiseks ning arendaja saab keskenduda juba rakendusloogika ning testide kirjutamisele. Välise testimisprogrammi kasutamine koos tarkvara programmilise genereerimisega tõstab selle testimismeetodi usaldusväärsust oluliselt, vähendades testimistarkvara enda vigade arvu.

Tarkvara genereerimine programmiliselt on Omroni PLC arenduskeskkonna puhul võimalik, kuna nii muutujaid kui loogikat on võimalik projekti väljastpoolt lisada. Kõigi teiste PLC platvormide puhul ei pruugi see võimalik olla.

Alternatiiv tarkvara genereerimisele programmiliselt on tarkvara programmiline ülevaatus. [9] Selle eesmärgiga programm võiks jälgida, kas testimise jaoks loodud funktsiooniplokkide väljakutsetes kasutatud mälupesad järgivad teatud mustrit. Lisaks võiks see programm tuvastada võimalikku mälupiirkondade ristkasutust. Samas oleks sellise tarkvara arendamine üsna keerukas ning tulemus mõnevõrra küsitava väärtusega.

4. Kokkuvõte

Programmeeritava loogikakontrolleri tarkvara testimisel selle käivitamise teel peavad arendajad silmitsi seisma probleemidega, mis paljudes teistes arendusvaldkondades ei esine. Testimiseks vajalikud riistvaraseadmed ei pruugi kättesaadavad olla, samas riistvaraga koos testimine võib vigade ilmnemisel ohustada seadmeid endid, keskkonda või inimest. Tööstusprotsessid võivad olla liiga kiired või mahukad inimesele testimiseks, vajadus on automaattestide tegemise järele. Käesoleva töö eesmärk oli identifitseerida need tekkida võivad probleemid ja anda vastus küsimusele, kas eksisteerib ja milline näeks välja PLC tarkvara praktiline testimismeetod, mida oleks võimalik rakendada nendes olukordades. Töö tulemus pidi andma küllaldaselt praktilisi näpunäiteid PLC testimise läbiviimiseks. (vt jaotis 1.2 ja 2.7)

Käesolev töö andis ülevaate probleemi taustast ja testimise temaatikast üldiselt ning kirjeldas lähemalt PLC testimisega seotud probleeme. Senise praktika tuvastamiseks küsitleti PLC tarkvara arendajaid ning analüüsiti kirjandust.

Näidisprojekti näitena kasutades kirjeldati ning analüüsiti valdkonna senises praktikas kasutatud testimismeetodeid – lihtsat riistvaraga testimist, riistvara asendusmeetodeid, diagnostikafunktsioonide lisamist PLC tarkvarasse ning PLC simuleerimist. Selgus, et kõigil meetoditel on oma tugevad ja nõrgad küljed, kuid ühiselt katavad nad enamuse testimisel esilekerkivatest probleemsetest olukordadest. Siiski ei kata nad päris kõiki olukordi.

Töö autor pakkus omaltpoolt välja testimismeetodi, millega on võimalik katta kõik kirjeldatud probleemsed olukorrad, ka need, mis teiste meetoditega katmata jäid. Meetodi puhul eraldatakse loogika osaliselt või tervenisti riistvaraseadmete küljest ning mõjutatakse loogika sisendeid välise testimistarkvara abil. Meetod annab testimise läbiviimise osaliselt või tervenisti inimeselt üle arvutiprogrammile, võimaldades korraldada oluliselt mahukamaid ja täpsemaid teste. See meetod on ühtlasi ka ainus töö autori poolt leitud viis praktiliste PLC tarkvara automaattestide läbiviimiseks, võimaldades mõnda tuntud testimisraamistikku kasutades automaatteste kasutada kõigis töö poolt kirjeldatud probleemsetes olukordades.

Parima tulemuse saavutamiseks PLC tarkvara testimisel on mõistlik rakendada võimalikult palju erinevaid testimismeetodeid. Alustada tuleks töö autori poolt välja pakutud uue meetodiga ja simulaatoriga testides ning tasapisi haarata testimisse kaasa järjest enam riistvaraseadmeid, jõudes lõpuks täies riistvarakoosseisus vastuvõtutestideni välja. Käesolevas töös antud soovituste ja näpunäidete abil on seda võimalik teha teadlikumalt ning süstemaatilisemalt, võimaldades paremat testimise tulemust – rohkemate vigade leidmist.

Käesoleva töö alguses seatud eesmärgid saavutati, püstitatud küsimustele anti vastused. Väljapakutud uus praktiline testimismeetod katab kõiki töös kirjeldatud probleemseid olukordi. Töö tulemused on praktilist laadi ning tarkvaraarendajate poolt kergesti kasutusele võetavad.

Töö edasiarendusena võiks disainida ning realiseerida töö lõpus näitena välja toodud tarkvararaamistiku universaalselt kasutataval kujul (vt jaotis 3.7.3).

Summary

When testing the software of Programmable Logic Controllers, developers must face problems that do not exist in many other development domains. Hardware necessary for testing could be absent. Testing with hardware could be hazardous for equipment, environment or people, if software faults emerge. Industrial processes could be too quick or big to be tested by humans, implying a need for automatic testing. The goal of this thesis was to identify these potential problems and give an answer to question, is it possible to have a PLC testing method that would be usable in all these problematic situations.

This thesis gave an overview of the background of the problem and of testing in general, also describing the problems associated with PLC testing more thoroughly. Identifying the testing methods used practically was done by interviewing software developers and analysing respective literature.

These practically used testing methods were then described and analysed – simple testing with hardware; methods that manipulate hardware; adding diagnostic functions to PLC code; using PLC simulators. The analysis was illustrated with a sample project. It turned out that all of the methods have their strengths and weaknesses, but together they cover most of the problematic situations that can occur while testing. But not all of the situations.

This thesis proposed a new testing method that covers all of the problems concerned, including the ones not covered by other methods. This new method involves separating the PLC logic partially or entirely from the hardware and affecting the inputs with an external testing software program. The method hands conducting the tests partially or entirely over to a computer program. This allows running more complex tests. This method is also the only method found by the author of this thesis to allow conducting practical automatic tests, by allowing the use of any common automatic testing framework in all of the problematic testing situations described by the thesis.

In order to achieve the best results while testing PLC software, it is wise to use as many testing methods as possible, starting with the new method proposed by the thesis, using it with the simulator, then gradually adding hardware until the final configuration is reached and full integration tests conducted. The suggestions and tips provided by this thesis help

allow for a more conscious and systematic approach that leads to better results – more found software bugs.

The goals set in the beginning of this thesis were met, the raised questions were answered. The proposed new testing method covers all of the described problematic situations. The results of this thesis are practical and can be easily implemented by software developers.

A further development of this thesis could implement the software example brought out in the end of the thesis as a universal software framework (see section 3.7.3).

Kasutatud kirjandus

[1] D. K. T. Erickson, „Programmable logic controllers: Hardware, software architecture," *InTech Magazine,* nr October, 2010.

[2] E. Csanyi, „Three generations of SCADA system architectures," Electrical Engineering Portal, 2015.

[3] N. Matic, „Introduction to PLC controllers," 2003. [Võrgumaterjal]. [Kasutatud 24 9 2015].

[4] *IEC 61131-3:2013: Programmable controllers - Part 3: Programming languages,* 2013.

[5] *IEC 61499: Function Blocks for Embedded and Distributed Control Systems Design, Second Edition,* 2012.

[6] D. K. T. Erickson, „Programmable logic controller hardware," *InTech Magazine,* nr December, 2010.

[7] *CX-Programmer User Manual Version 3.0,* 2012.

[8] M. Jackson, Software Requirements & Specification, 1995.

[9] C. Kaner, J. Bach ja B. Pettichord, Lessons Learned in Software Testing: A Context-Driven Approach, 2011.

[10] U. Kramer, „Continuous Testing as a Strategy of Improving the PLC software development cycles," %1 *2001 IEEUASME International Conference on Advanced Intelligent Mechatronics Proceedings*, 2001.

[11] L. Pearson, „The Four Levels of Software Testing," 2015. [Võrgumaterjal]. [Kasutatud 3 10 2015].

[12] C. Kaner, „What Is a Good Test Case?," %1 *Florida Institute of Technology*, 2003.

[13] R. Hametner, B. Kormann, B. Vogel-Heuser, D. Winkler ja A. Zoitl, „Test Case Generation Approach for Industrial Automation Systems," %1 *5th International Conference on Automation, Robotics and Applications*, 2011.

[14] B. Marick, When Should a Test Be Automated?, 2000.

[15] L. Wang ja K. C. Tan, „Modern Industrial Automation Software Design," 2006. [Võrgumaterjal]. [Kasutatud 18 10 2015].

[16] S. V. Styp ja L. Yu, „Symbolic Model-Based Testing of Industrial Automation Software," %1 *Hardware and Software: Verification and Testing: 9th International Haifa Verification Conference*, 2013.

[17] J. W. Valvano, Embedded Systems: Real-Time Interfacing to Arm® Cortex(TM)-M Microcontrollers, 2011.

[18] K. Karvinen ja T. Karvinen, Make: Arduino Bots and Gadgets, 2011.

[19] M. Krichen ja S. Tripakis, „Black-Box Conformance Testing for Real-Time Systems," *Verimag,* 2004.

[20] J. Wegener, H. Sthamer, B. F. Jones ja D. E. Eyres, „Testing real-time systems using genetic algorithms," *Software Quality Journal,* 6 1997.

[21] R. Cardell-Oliver ja T. Glover, „A practical and complete algorithm for testing real-time systems,“ %1 *Lecture Notes in Computer Science*, 2006.

[22] H. Thane, Monitoring, Testing and Debugging of Distributed Real-Time Systems, 2000.

[23] C. Iman, „I&C, Unit Testing and Coding of PLC for Plant System,“ [Võrgumaterjal]. [Kasutatud 28 9 2015].

[24] C. Garvey, „Unit testing for Industrial Automation software development. What is it?,“ HAL Software, 2014.

[25] „Emulate3D Controls Testing,“ [Võrgumaterjal]. Available: http://www.demo3d.com/show/controls_testing. [Kasutatud 14 11 2015].

[26] „FINS Command Technical Guide,“ Omron, 2012.

[27] „Optomux Protocol Guide,“ 6 2014. [Võrgumaterjal]. Available: http://documents.opto22.com/1572_Optomux_Protocol_Guide.pdf. [Kasutatud 12 10 2015].

[28] I. Ploompuu, „Väljapakutud uue testimismeetodi näidisprogrammi lähtekood,“ 16 11 2015. [Võrgumaterjal]. Available: https://dl.dropboxusercontent.com/u/83446495/PloompuuPLCTestLib.zip.

[29] *Omron FINS Ethernet Driver Help,* 2015.

[30] J. A. Bruinsma, „PID simulator,“ 2013. [Võrgumaterjal]. Available: http://sourceforge.net/projects/pid-simulator/. [Kasutatud 23 11 2015].

[31] G. Biggs, „Applying regression testing to software for robot hardware interaction,“ %1 *2010 IEEE International Conference on Robotics and Automation*, 2010.

[32] C. Lesher, „Modular Software Simplifies Complex Programming,“ Automation Direct, 2013.

[33] M. Lotz, „Waterfall vs. Agile: Which is the Right Development Methodology for Your Project?,“ Segue Technologies, 2013.

Lisa 1. Testimisprogrammi Java klasside näidised

Klassi DigitalInput implementatsiooni näidis

```
public class DigitalInput extends Input {

	private boolean defaultValue = false;

	public DigitalInput(String address, String addressSimulationState,
		String addressSimulatedValue) {
		super(address, addressSimulationState, addressSimulatedValue);
	}

	public DigitalInput(String address, String addressSimulationState,
		String addressSimulatedValue, boolean defaultValue) {
		this(address, addressSimulationState, addressSimulatedValue);
		this.defaultValue = defaultValue;
	}

	private void setValueWithSimulationState(boolean newValue) {
		if (!getSimulationState()) setSimulationState(true);
		PLC.getInstance().writeBoolean(getAddressSimulatedValue(),
			newValue);
	}

	public void set(boolean newValue) {
		setValueWithSimulationState(newValue);
	}

	public boolean get() {
		return PLC.getInstance().readBoolean(getAddress());
	}

	@Override
	public void resetToZero() {
		setValueWithSimulationState(false);
	}

	@Override
	public void resetToDefault() {
		setValueWithSimulationState(defaultValue);
	}

	@Override
	public String toString() {
		return formatToString("DI", get() ? "ON" : "OFF");
	}

}
```

Testi T-5 põhjal implementeeritud automaattesti näidis JUnit raamistikus

```
public class SulatuskamberTests {

	@Test
	public void TestT5() throws InterruptedException {

		PLC.configureInstance("192.168.1.90");

		DigitalInput ukseandur =
			new DigitalInput("H7.0", "H8.0", "H9.0");
		DigitalInput termilineKaitse =
			new DigitalInput("H7.1", "H8.1", "H9.1");
		DigitalInput sulatusreziimiNupp =
			new DigitalInput("H7.2", "H8.2", "H9.2");
		AnalogInputFloat kambriTemperatuur =
			new AnalogInputFloat("D5000", "H8.7", "D5100");
		AnalogInputFloat tooteTemperatuur =
			new AnalogInputFloat("D5002", "H8.8", "D5102");
		DigitalOutput soojendi = new DigitalOutput("H10.0");
		AnalogVariableFloat soojendiVõimsus =
			new AnalogVariableFloat("D2010");

		ukseandur.set(false);
		termilineKaitse.set(false);
		kambriTemperatuur.set(25.0f);
		tooteTemperatuur.set(-5.0f);
		sulatusreziimiNupp.set(true);
		Thread.sleep(200);
		sulatusreziimiNupp.set(false);
		Thread.sleep(500);

		Float ettenähtudVõimsus = soojendiVõimsus.get();

		while (soojendi.get()) {}
		long time1 = System.nanoTime();
		while (!soojendi.get()) {}
		long time2 = System.nanoTime();
		while (soojendi.get()) {}
		long time3 = System.nanoTime();

		long kokkuKestvus = time3 - time1;
		long seesKestvus = time3 - time2;
		float arvutatudVõimsus = (float)seesKestvus /
			(float)kokkuKestvus * 100f;
		float erinevus = Math.abs(arvutatudVõimsus - ettenähtudVõimsus);

		assertTrue("Kogukestvus on pikem kui 0,1 sek",
			kokkuKestvus > 100000000l);
		assertTrue("Kogukestvus on lühem kui 2 sek",
			kokkuKestvus < 2000000000l);
		assertTrue("Ettenähtud võimsuse ja tegeliku võimsuse erinevus on
			väiksem kui 1 protsendipunkt", erinevus < 1f);
	}

}
```