SAVONIA

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

PAALUTUSKONEEN SIMULAATIO-MALLIN JA TESTAUSYMPÄRISTÖN VÄLINEN RAJAPINTA

TEKIJÄ/T: Niko Heikkinen

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Sähkö- ja automaatiotekniikan tutkinto-ohjelma	
Työn tekijä(t) Niko Heikkinen	
Työn nimi Paalutuskoneen simulaatiomallin ja testausympäristön välinen rajapinta	
Päiväys 20 joulukuu 2021	Sivumäärä/Liitteet 23/0
Toimeksiantaja/Yhteistyökumppani(t) Junttan Oy	

Tiivistelmä

Opinnäytetyö keskittyi, kuinka paalutuskoneen simulaatiomallin ja testausympäristön välille toteutettiin kommunikaatiorajapinta, jonka avulla testauksessa olevaa laitteistoa voitiin ajaa simulaatiomallia vasten. Työhön vaadittavat komponentit oli jo toimeksiantajan puolesta hankittu ja päätetty.

Rajapinta tarvitsi vahvan pohjatyön, jotta ohjelman suoritusaika saatiin minimoitua ja työn toteutus oli sujuvaa. Pohjatyö sisälsi ennalta valittujen komponenttien dokumentaation etsimisen ja niiden tulkitsemisen. Suuri määrä tiedosta löytyi valmistajien sivuilta, mutta tietoa piti myös kysellä valmistajilta suoraan.

Toimeksiannon lopputuloksena oli valmis rajapinta, jolla pystyi testaamaan testauksessa olevaa ohjelmistoa simulaatiomallia vasten ja vasteaika oli hyvin minimaallinen.

Avainsanat

rajapinta, paalutuskoneen simulaatiomalli, testausympäristö

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Electrical and Automation Engineering	
Author(s) Niko Heikkinen	
Title of Thesis Interface between Piling Machine Simulation Model and Testing Environment	
Date 20 December 2021	Pages/Appendices 23/0
Client Organisation /Partners Junttan	

Abstract

The purpose of this thesis was to build communication interface between a simulation model of the piling rig and testing environment so that devices under testing could be run against the simulation model. The components needed for the work were decided and acquired by the commissioner before the work was started.

Firstly, to minimize the cycle time of the program and to produce the interface with ease, there needed to be a strong foundation to build on. The foundation included searching and interpreting the documentation found mostly from the manufacturer's sites, but some of them had to be asked from the manufacturers.

As a result of this thesis, an interface was made that could test software under test against the simulation model with minimal response time.

Keywords
interface, simulation model of piling rig, testing environment

SISÄLTÖ

1 JOHDANTO

Tässä opinnäytetyössä käydään läpi suunnittelu ja toteutus paalutuskonesimulaatiomallin ja testausympäristön välille. Paalutuskonesimulaatiomalli on tehty Mevean ohjelmistolla, joka soveltuu erityisesti reaaliaikaisten simulaatiomallien tekoon. Testausympäristössä käytettään ADAM-Ethernet IO moduuleita oikean koneen lähtöjä ja tuloja mallintamaan. Nämä lähdöt ja tulot on kytketty HIL (”Hardware In Loop”) järjestelmään.

HIL laitteisto on tuotekehityksessä oleva ohjausjärjestelmä, johon kuuluu ohjelmoitavia moduuleita. Järjestelmän tarkoitus on tehdä sellainen ympäristö ohjelmistolle, ettei ohjelmisto itse tiedä onko se oikeassa fyysisessä laitteessa vai simuloidussa ympäristössä.

Simulaatiomalli oli ennen työn aloitusta suurimmaksi osaksi tehty, joten sen tekemistä ei tässä opinnäytetyössä käydä läpi, muuten kuin kuinka saada yhteys tähän kyseiseen malliin.

2 JUNTTAN YRITYKSENÄ

Junttan toimii maailman johtavien paalutuskoneiden suunnittelijana ja valmistajana. Yritys on perustettu vuonna 1976 ja ensimmäinen hydraulinen paalutuskone on valmistunut vuonna 1979. Ensimmäinen tuotesarja, PM20, lanseerattiin vuonna 1983. Nämä koneet toimivat hydraulisesti esiohjattuna eli koneen ohjaus tapahtuu täysin hydraulisesti. Vuonna 2010 Junttan lanseerasi ensimmäisen sähköisesti ohjatun konesarjan PMx. Tämän opinnäytetyön rajapinta keskittyy vain näihin sähköisesti ohjattuihin koneisiin ja tarkemmin voimayksiköiden ohjausjärjestelmään. (Junttan Oy, 2021)

Voimayksikkö on yksinkertaistettuna erillinen laitteisto, jossa on vain hydraulisen voiman tuottoon vaadittava laitteisto eli moottori, pumppu ja muita tätä tukevaa laitteistoa.

3 SUUNNITTELU

Toimivan ohjelmiston teon ensimmäisenä vaiheena toimii suunnittelu. Tässä osiossa käydään läpi mitä ohjelmisto vaatii, mitä laitteistoa tarvitaan kyseiseen projektiin, millä ja miten ohjelmisto tehdään ja tässä tapauksessa, mistä saadaan dokumentaatiot yhdistettäviin ohjelmiin ja laitteistoon.

3.1 Vaatimukset

Ohjelmiston vaatimukset tulevat työn tilaajalta, ja ovat yleensä ylimalkaisia. Esimerkiksi, vaatimuksissa ei ole määritelty tarkasti mitä ohjelmointikieltä tulee käyttää, miten kommunikaatio tulee toteuttaa tai miten ohjelmointi tulee tehdä.

Tämän rajapinnan vaatimukset olivat, että simulaatiomallin pitää ohjata ADAM-moduulien lähtöjen tilaa ja lukea tulojen tilat, eli ohjelmiston pitää välittää tietoa simulaatiomallista fyysisiin lähtö/tulo moduuleihin ja takaisin, jotta simulaatiomallia vasten voidaan testata tuotekehityksessä olevia koneohjausmoduuleita.

3.2 Laitteisto

Laitteisto määräytyy joko jo työn tilaajalla olevista laitteista tai jos tilaajalla ei niitä ole niin laitteisto määritellään työn ja budjetin mukaan.

Tässä työssä tilaajalla oli laitteisto valmiina. Simulaatiomallia pyöritetään tehokkaalla Windows tietokoneella. Fyysisenä lähtö/tulo moduuleina käytetään erilaisia ADAM-Ethernet IO moduuleita, joita voi ohjata UDP/TCP protokollan avulla. Tietokoneen ja ADAM-moduulien välille on laitettu Ethernet kytkin, joka mahdollistaa yhden laitteen yhdistämisen useaan Ethernet protokollaa käyttävään laitteeseen.

3.3 Ohjelmisto

Ohjelmiston suunnittelun viimeisenä vaiheena oli määrittää mitä ohjelmointikieltä ja -ympäristöä ohjelmiston tekemiseen käytetään ja minkälainen käyttöliittymä tehdään.

Rajapinta ohjelmoidaan C#-ohjelmointikielellä. Tähän päätökseen vaikutti ohjelmointikielestä oleva valmis tietämys ja useiden valmiiden ohjelmointikirjastojen löytyminen esimerkiksi TCP/UDP kirjasto, jolla hoidetaan kommunikaatio kumpaakin suuntaan. Ohjelmointi alustana on Visual Studio 2019.

Ohjelma toteutuu konsoli applikaationa eli varsinaista käyttöliittymää ohjelmassa ei ole. Ohjelmiston tarpeissa ei ole nähdä mitään reaaliaikaista tietoa rajapinnan toiminnasta työn teko hetkellä. Konsoliin on kuitenkin tarkoitus näyttää mahdolliset vikatilanteet ja informoida, jos yhteyttä ei ole jompaankumpaan suuntaan, mutta lähtöjen ja tulojen tilaa ei tarvitse nähdä.

3.4 Dokumentaatio

Kummassakin päässä tulee olla ohjelmointirajapinta, että kahden asian välille voidaan tehdä rajapinta, tässä tapauksessa laitteiston ja ohjelmiston välille. Hyvän ja ajan tasaisen dokumentaation löytäminen yrityksille tarkoitettuun ohjelmistoon tai laitteistoon voi olla hyvin haastavaa ja tämänkin työn kohdalla tämä piti paikkansa.

ADAM-IO moduulien dokumentaatio löytyi netistä. Dokumentaatio oli laaja, mutta siitä löytyi paljon virheitä, kuten minkälaisia vastauksia voi odottaa moduulilta takaisin. Mevean dokumentaatiota löytyi joistakin ohjelmistoista suoraan ja toisten kohdalla tietoa pystyi pyytämään Mevean tuesta.

4 TYÖN ALOITUS

Ennen kuin ohjelmointi voitiin aloittaa, oli selvitettävä kuinka ADAM-moduulien ja Mevean ohjelmistorajapinnat toimivat. ADAM-moduulien dokumentaatiosta selvisi, että kommunikaatioon voidaan käyttää joko TCP tai UDP protokollaa ja tähän työhön valittiin UDP, koska datan vastaanottamisen varmistukselle ei ollut tarvetta ja UDP:n käyttö vähentää viivettä, jonka minimointi oli myös vaatimuksena rajapinnalle. Mevean ohjelmisto tukee molempia protokollia, joten myös siihen valittiin UDP.

4.1 ADAM-moduulien rajapinta

4.1.1 Käytössä olevat moduulit

Testipenkissä olevia ADAM-moduuleita on kahta sarjaa. 6000-sarja ja 6200-sarja. Näiden välillä on pieniä eroja lähetettävissä ja vastaanotetuissa viesteissä. Nämä pienet erot ovat lähinnä viestin ensimmäiset 1-2 merkkiä. (Advantech Co., Ltd., 2018) (Advantech Co., Ltd., 2018)

4.1.2 Viestin yleinen rakenne

ADAM-moduulit vastaanottaa ja lähettää dataa hyvin tiiviissä muodossa. Lähetetty viesti alkaa ”$”-merkillä ja viesti päätetään aina ”\r”-merkkiin, joka on vaunupalautus (Carriage return) merkki. Vastaanotettu viesti alkaa joko ”!”, ”?”, ”>” tai ”#” joka kertoo moduulista riippuen, oliko sille lähetetty viesti oikein alustettu. (Advantech Co., Ltd., 2018) (Advantech Co., Ltd., 2018)

4.1.3 Digitaaliset tulot

Kysellessä ADAM-moduulista digitaalisten tulojen tilat lähetetään moduulille viesti ”$016\r”. Jos tämä on lähetetty moduulille, jossa on digitaalisia tuloja, tulee vastaus seuraavanlaisessa muodossa: ”#0100XXXX”. ”X”-kirjain kuvastaa tässä yhtä tavua. Jos viesti ei ole oikein formatoitu esimerkiksi ”\r” puuttuu, niin vastaukseksi tulee ”?01”. Esimerkkinä jos tulot 0, 1, 4, 15 ovat päällä niin vastaukseksi moduulilta tulee ”#01008013”. Vastauksessa olevien tulojen järjestys on oikealta vasemmalle eli tulo 0 on oikeassa reunassa. (Advantech Co., Ltd., 2018)

4.1.4 Digitaaliset lähdöt

Digitaalisia lähtöjä voidaan ohjata joko yksitellen tai kaikki lähdöt kerrallaan. Työn määrittelyn perusteella ei ole tarvetta ohjata lähtöjä yksitellen, vaikka tällöin voisi tehdä ohjelmasta hieman joustavamman, joten ohjaus on tehty käskyttämällä kaikkia lähtöjä yhdellä viestillä.

Digitaalisten lähtöjen ohjaus tapahtuu komennolla ”#0100XXXX\r”. ”X”-kirjain kuvastaa yhtä tavua. Jos viesti on kirjoitettu oikein niin vastaukseksi tulee ”>01” ja lähdöt ovat komennon mukaisessa tilassa. Esimerkki: Laitetaan moduulin lähdöt 0, 6, 7, 8 ja 12 lähdöt päälle viestillä ”#010011C1” ja saame vastaukseksi ”>01”. (Advantech Co., Ltd., 2018)

4.1.5 Analogiset tulot

Analogisten tulojen tilat voi lukea yksitellen tai kaikki kerrallaan ja tässä työssä ne luetaan kerrallaan.

Analogiset tulot voi kysellä komennolla ”#01\r”. Tällöin vastauksena tulee hex numeroiden sijaan ASCII muodossa kaikki lähtöjen tilat ”+” tai ”-”-merkillä erottaen. Esimerkki: kirjoitetaan moduulille ”#01\r”, jolloin moduuli vastaa ”>+0010.000+0010.000+0010.000+0010.000+0010.000+0010.000+0010.000+0010.000”. Arvot menevät vasemmalta oikealle alkaen ensimmäisestä analogisesta inputista. (Advantech Co., Ltd., 2018)

4.1.6 Analogiset lähdöt

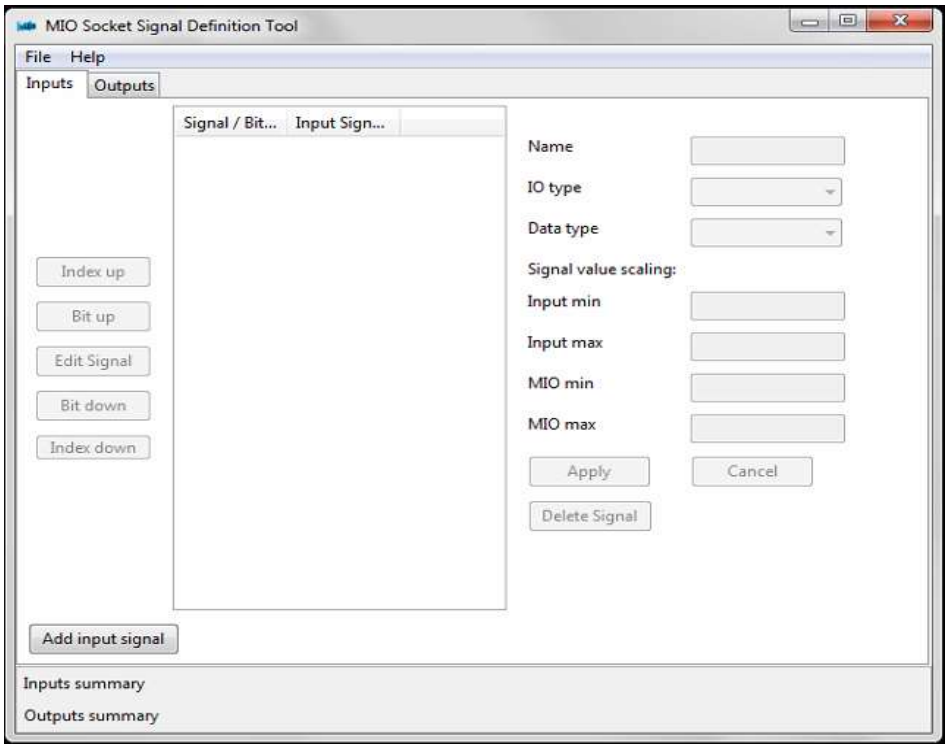
Analogiset lähdöt tulee ohjata yksi kerrallaan. Analogisia lähtöjä käyttäessä tulee huomioida, etteivät ne ohjaudu niin nopeasti kuin digitaaliset. Tämä viive kantautuu myös moduulilta saatavaan vastaukseen. Moduuli antaa vastauksen vasta sitten, kun lähtö on asetetussa arvossa.

Analogisten lähtöjen ohjaus tapahtuu komennolla ”#01BCccXXXX”. ”c”-kirjain kuvastaa lähdön numeroa ja kirjain ”X” kuvastaa haluttua arvoa heksadesimaalina 0-0FFF väliltä. Koska lähtöjen konfiguraatio voi olla esimerkiksi 0-10 volttia tai 0-20mA, niin ohjattavan lähdön maksimi arvo on tiedettävä ennen ohjaamista. (Advantech Co., Ltd., 2018)

4.2 Mevea rajapinta

Mevean rajapintana toimii Mevea ”Socket MIO Client”-sovellus, joka vastaanottaa bittitaulukon. Tämän bitti taulukon sisältö määrätään joko Mevean tekemällä työkalulla nimeltä ”MIO Socket Signal defenition tool”, jolla luodaan xml-tiedosto, joka sisältää bittien järjestyksen tai tekemällä itse tämä xml-tiedosto. Tämä xml-tiedosto viedään vielä ohjelmaan nimeltä ”MIO Socket Signal Defenition Tool”, joka yhdistää nämä bittitaulukon arvot johonkin mallin tuloihin tai lähtöihin.

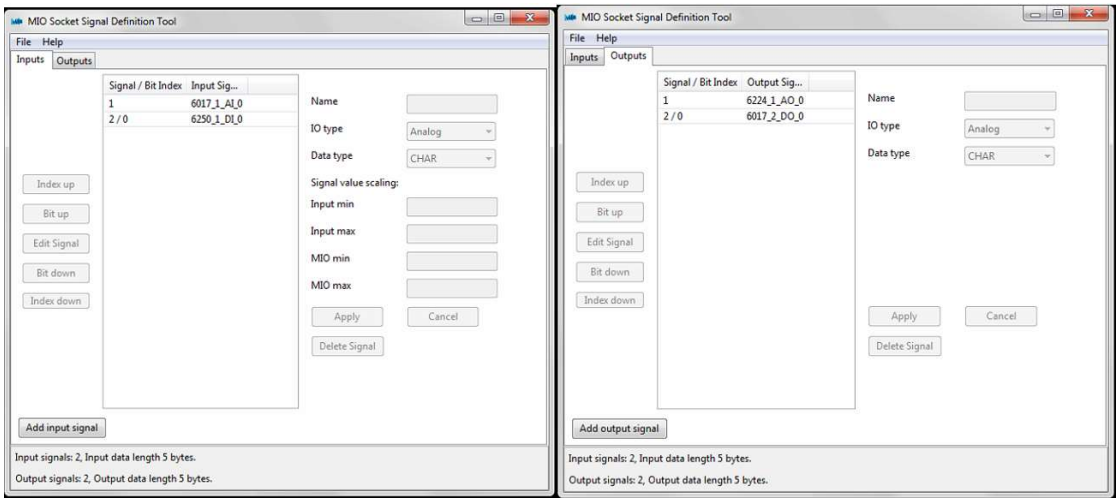
4.2.1 MIO Socket Signal Definition Tool



Kuva 1. MIO Socket Signal Definition Tool

Tällä työkalulla määritellään mitä bittejä on missäkin kohdassa bittitaulukkoa.

Painamalla nappia ”Add input signal” ohjelma lisää yhden signaalin listan, jolle voi määrittää nimen, IO tyypin, Data tyypin, minimi ja maksimi arvot. Signaalin indeksiä ja bittiä voi myös muuttaa.



Kuva 2. Ohjelmaan lisätty muutama lähtö ja tulo

Kun signaalit on tallennettu, näyttää siitä tuleva XML-tiedosto tältä:

```
<?xml version="1.0" encoding="UTF-8"?>
<mevea-mio-socket-signals>
  <socket-signals>
    <socket-signal io-type="AO" name="6224_1_AO_0" data-type="FLOAT" index="1" bit-index="0" mio-value-min="0" mio-value-max="0" />

    <socket-signal io-type="DO" name="6017_2_DO_0" data-type="BIT" index="13" bit-index="0" />

    <socket-signal io-type="AI" name="6017_1_AI_0" data-type="FLOAT" index="1" bit-index="0" mio-value-min="0" mio-value-max="1500" input-value-min="0" input-value-max="0.15" />

    <socket-signal io-type="DI" name="6250_1_DI_0" data-type="BIT" index="17" bit-index="0" />
  </socket-signals>
  <version-info version="0.1.1" />
</mevea-mio-socket-signals>
```

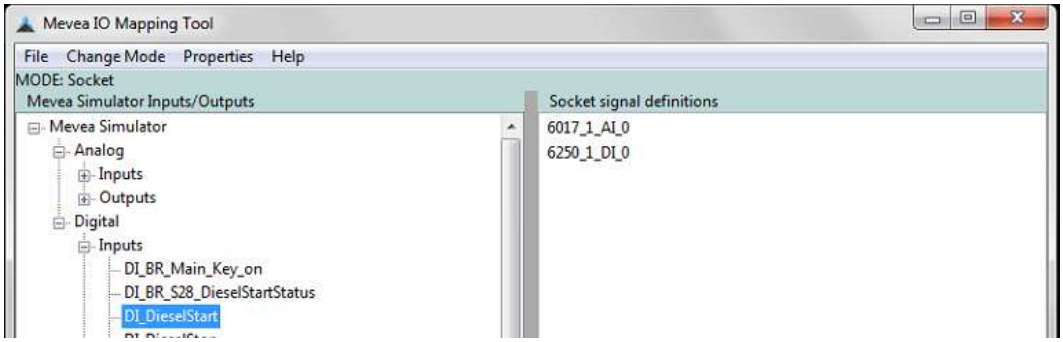
Kuva 3. Socket Signal Defenition Tool:n xml tiedosto

Tiedostossa on kolme ”tasoa” viimeisenä näistä on ”socket-signal”-tagi. Tämä tagi sisältää kaikki nämä samat kohdat, jotka näkyvät sovelluksen kenttinä. AO-tyyppisissä lähdöissä lopussa on ”mio-value-min” ja ”mio-value-max”-attribuutit, jolla voidaan määrittää minimi ja maksimi arvot, jota mevean mallilta voidaan odottaa. Arvojen ollessa nollia, ei näitä rajoja käytetä. DO ja DI tyyppisissä lähdöissä/tuloissa ei ole muuta kuin indeksi ja bitti indeksi. AI-tyyppisessä tulossa on skaalaus mahdollisuus. Skaalain skaalaa tulevan luvun annettujen arvojen mukaan.

Nimeämis-logiikkana toimii Adam moduulien nimet ja lähdöt/tulot. Esimerkiksi ”6017_1_DO_0” viittaa ensimmäiseen ADAM-6017 moduulin DO 0 lähtöön. Tällä tavalla on rakennettu jokainen rajapinnan ohjaama moduuli tähän tiedostoon.

4.2.2 Mevea IO Mapping tool

Mevea IO Mapping Tool yhdistää edellisessä osiossa tehdyn XML-tiedoston signaalit malliin ja luo siitä ”mio”-päätteisen tiedoston.



Kuva 4. Mevea IO Mapping tool signaalin yhdistys

Ohjelmaan ladataan malli ja XML-tiedosto, jonka jälkeen näkyviin tulee kaksi listaa, jossa voidaan yhdistää XML-tiedostossa olevat signaalit mallissa oleviin lähtöihin tai tuloihin. Valinta tapahtuu painamalla vasemman puolen listalta haluamaansa mallin lähtöä/tuloa ja klikkaamalla oikean puolen listalta haluttuun signaaliin. Tässä vaiheessa valitaan myös kommunikaatio protokolla. Tästä sovelluksesta tuleva tiedostoa käytetään seuraavassa sovelluksessa, jonka kanssa kommunikoidaan nimeltä ”MIO Socket Client”.

4.2.3 MIO Socket Client

MIO Socket Client ylläpitää kommunikaation mallin ja ulkopuolisen clientin kanssa. Kommunikaatio tapahtuu UDP tai TCP yhteydellä. Clientiin ladataan edellisessä ohjelmassa luotu ”mio”-päätteinen

tiedosto. Sovellus antaa määritellyllä nopeudella lähtöjen tilat mallilta ja lähettää ne yhdistettyyn rajapintaan. Tämän kysely välin voi määritellä clientistä. Vakiona kysely väli on 10ms.

5 RAJAPINTA

Alustava suunnitelma työn osioiden tekemiseen oli: XML-tiedoston parsinta, lähtöjen tilojen vastaanotto Mevealta ja datan parsiminen, Mevean datan lähetys Adam-moduuleille, tulojen tilojen lukeminen Adam-moduuleilta ja parsiminen ja Adam-moduuleiden datan lähetys Mevealle. Rajapintaa varten myös tarvitsi jonkinlaisen tavan määrittää Adam-moduulien puoli.

5.1 Adam-moduulien konfiguraatio tiedosto

Adam-moduuleitten konfiguraatio toteutettiin XML-tiedostolla. Tiedosto sisältää jokaisen moduulin ja moduulin lähdöt ja tulot ja myös Mevean MIO clientin IP-osoitteen ja portin:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
<MIO-
client name="Junttan" ip="192.168.0.3" port="2348" signalXMLPath="signal.xml"
/>
<IO-Modules>
    <Module model="Adam-6017" module-type ="Analog" description="Adam-
6017 #1 AI-module" ip="192.168.0.10">
        <Signal io-pin="0" socket-signal-name="6017_1_DO_0"/>
        <Signal io-pin="1" socket-signal-name="6017_1_DO_1"/>
        <Signal io-pin="0" socket-signal-name="6017_1_AI_0"/>
        <Signal io-pin="1" socket-signal-name="6017_1_AI_1"/>
        <Signal io-pin="2" socket-signal-name="6017_1_AI_2"/>
        <Signal io-pin="3" socket-signal-name="6017_1_AI_3"/>
        <Signal io-pin="4" socket-signal-name="6017_1_AI_4"/>
        <Signal io-pin="5" socket-signal-name="6017_1_AI_5"/>
        <Signal io-pin="6" socket-signal-name="6017_1_AI_6"/>
        <Signal io-pin="7" socket-signal-name="6017_1_AI_7"/>
    </Module>
</IO-Modules>
</Configuration>
```

Kuva 5. Adam moduulien konfiguraatio tiedosto

Tagilla ”MIO-client” on kuvattu mikä on vastaanottavan clientin nimi, osoite ja portti. Seuraavana tagina on ”IO-Modules”, jonka sisällä on kaikki Adam-moduulit. Yksi moduuli on kuvattu tagilla ”Module”. Tämän moduulin attribuuteiksi laitetaan moduulin malli, tyyppi, kuvaus ja IP-osoite. Tyyppi määräytyy sen mukaan, onko moduulissa analogisia/digitallisia lähtöjä/tuloja. Moduulin sisällä on ”Signal”-tagi, johon on laitettu IO pinni ja nimi, joka korreloi Mevean ohjelmalla tehtyyn XML-tiedoston signaaleihin.

5.2 XML-tiedoston parsinta

Ensimmäisenä ohjelmoinnissa oli ”Signal Defenition Toolista” saadun XML-tiedoston ja itse tehdyn konfiguraatio tiedoston parsiminen. Parsiminen tehdään vain kerran aina ohjelman käynnistyksessä, joten tämän suorituskyky ei ole kriittinen.

Parsimiseen käytettään .Net Framework:sta löytyvää ”XMLReader” kirjastoa. Kirjasto toimii antamalla XMLReader-luokalle XmlReaderSetting-luokan ja halutun tiedosto sijainnin(Kuva 6)

```
XmlReaderSettings settings = new XmlReaderSettings();
List<AdamModule> modules = new List<AdamModule>();
string signalsLocation = "XMLFiles/Signals.xml";
string configLocation = "XMLFiles/Configuration.xml";

/*    Read Configuration.xml file    */
using (XmlReader reader = XmlReader.Create(configLocation, settings))
{
```

Kuva 6. XML Reader

”Configuration.xml”-tiedoston luku tapahtuu kolmessa osassa: MIO-Clientin asetusten(”MIO-client”), moduulin(”Module”) ja pinnin(”Signal”) tietojen luku. Kyseisistä osioista luetaan yllä mainitut tiedot ja laitetaan ”AdamModule” nimiseen luokkaan (ks. kuva 7).

```
public class AdamModule
{
    public IPAddress ipAddress { get; set; }
    public string moduleType { get; set; }
    public int port { get; set; }
    public IPEndPoint endPoint { get; set; }
    public UdpClient socket { get; set; }
    public string value { get; set; }
    public bool responsePending { get; set; }
    public int responsePendingOnPin { get; set; }

    public List<Signal> signals = new List<Signal>();
    public AdamModule()
    {
        socket = new UdpClient();
        socket.Client.ReceiveTimeout = 10;
        port = 1025;
        responsePending = false;
    }
}
```

Kuva 7. Adam moduuli objekti

”AdamModule”-luokka sisältää kaikki tarvittavat tiedot, jotka ovat myös xml tiedostossa ja lisäksi tarkentavia tietoja. ”signals”-lista sisältää aina moduulista riippuen 4-16 ”Signal”-luokkaa

```
public class Signal
{
    public string name { get; set; }
    public string type { get; set; }
    public string dataType { get; set; }
    public int ioPin { get; set; }
    public int byteIndex { get; set; }
    public int bitIndex { get; set; }
    public double value { get; set; }
}
```

Kuva 8. Signaali objekti

Tämä sisältää myös samat tiedot kuin mitä xml-tiedostossa.

```
if (reader.Name == "Signal")
{
    if (tempModule != null) // Check if there is tempModule
    {
// This is to prevent if there is some empty tags so we don't make null object
        if (reader.HasAttributes)
        {
            Signal tempSignal = new Signal();
            while (reader.MoveToNextAttribute())
            {
                if (reader.Name == "io-pin")
                {
                    tempSignal.ioPin = int.Parse(reader.Value);
                }
                else if (reader.Name == "socket-signal-name")
                {
                    tempSignal.name = reader.Value;
                }
            }
            tempSignal.byteIndex = int.MinValue;
            tempModule.signals.Add(tempSignal);
        }
    }
    else
    {
   Console.WriteLine("tempModule was not created before getting signal tag!");
    }
}
```

Kuva 9. Signaalin luku konfiguraatio tiedostosta

Funktiolla ”reader.Name” saa lukijan tämän hetkisen kohdan tagin nimen. Jos ohjelma on ”Signal”-tagin kohdalla, luetaan kyseisestä objekteista pinnin numero ja nimi käyttämällä funktiota ”reader.Value” ja kääntämällä se oikeaan muuttuja muotoon. ”reader.HasAttributes” tarkistaa, että kyseisellä tagilla on attribuutteja, jotta ohjelma ei kaadu tyhjän tagin tullessa vastaan. Kaikki muut tagit on luettu samalla tavalla vastaaviin luokan muuttujiin.

Seuraavana piti parsia ”Signals.xml”-tiedosto, joka luotiin kohdassa 4.2.1. Erona edellä parsittuun tiedostoon on, että lopuksi aikaisemmasta tiedostosta parsitut datat yhdistetään näihin. Tämä tapahtuu lukemalla yksi signaali kerrallaan xml-tiedostosta ja lukemalla olemassa olevat moduulit ja niiden signaalit, jotta signaalille löytyy pari. Parin löytämiseen käytetään hyödyksi pinnien nimeämistä, jotka löytyvät kummastakin xml-tiedostosta (”name” konfiguraatio tiedostossa ja ”socket-signal-name” signaali tiedostossa). Koodi yhdistyksestä kuvassa (ks. kuva 10).

```
// Loop trough every adam and match names to link pins
bool found = false;
foreach (AdamModule module in modules)
{
    foreach (Signal signal in module.signals)
    {
        if (signal.name.Equals(tempSignal.name))
        {
            signal.type = tempSignal.type;
            signal.byteIndex = tempSignal.byteIndex;
            signal.bitIndex = tempSignal.bitIndex;
            signal.dataType = tempSignal.dataType;
            linkedSignalCount++;
            found = true;
            break;
        }
    }
    if (found) { break; }
}
```

Kuva 10. Signaalien yhdistäminen

Oikean signaalin löydyttäessä kummankin silmukan suorittaminen katkaistaan ja asetetaan datat oikeisiin paikoihin. Tämä vähentää läpi käytävien signaalien määrää huomattavasti, mutta ei käytännössä vaikuta ohjelman käynnistysaikaan huomattavasti pienissä moduuli/signaali määrissä.

5.3 Objektien korjaaminen ja siivoaminen

Koska Mevean xml tiedostossa ”index” kasvaa aina yhdellä riippumatta siitä onko kyseinen arvo byte tai float-tyyppinen, niin moduuli listaan on korjattava kyseisten datojen oikeat tavu/bitti-indeksit. Signaali luokassa on tämän takia string-muuttuja, joka sisältää muuttujan tyypin. Tyypin avulla pystyi määrittelemään kuinka monta tavua signaali vie.

Signaalien indexit ovat erilliset lähdöille ja tuloille, koska toiset ovat vastaanotettavia ja toiset lähetettäviä, joten signaalit pitää lajitella omiin listoihinsa. Listoihin jakaminen tehdään katsomalla moduuli ja signaali kerrallaan, että onko kyseinen signaali tulo vai lähtö ja lisäämällä kyseinen signaali sille kuuluvaan listaan.

5.3.1 Listojen siivoaminen

Ennen kuin signaalien indeksit voidaan korjata, on listoilta poistettava ei käytössä olevat signaalit eli signaalit, joilla ei ole vastaparia. Tämä tehdään, jotta seuraavassa osiossa tehtävä indeksin korjaus on helpompi tehdä.

5.3.2 Indexien korjaus

Vastaanotettavien ja lähetettävien viestien listat järjestellään tavuindeksin ja bitti indeksin mukaan, jonka jälkeen listat luetaan signaali kerrallaan ja asetellaan indeksi oikein seuraamalla muuttujan kokoa (ks. Kuva 11).

```
if (signal.byteIndex != int.MinValue)
{
    signal.byteIndex = index; // set the signals byteIndex to index
    if (signal.bitIndex == 7 || (signal.dataType != "BIT" && signal.dataType != null) ||
signal.bitIndex < oldBitIndex)
    {
        index += SizeOfDataType(signal.dataType);

        if (signal.bitIndex < oldBitIndex)
        {
            signal.byteIndex = index;
        }
        oldBitIndex = 0;
        // For edge case
        goto SkipIndexing;
    }
    oldBitIndex = signal.bitIndex;
SkipIndexing:;
}
```

Kuva 11. Indexin korjaus

Ensimmäisenä katsotaan, että korjauksessa oleva signaalin tavu indeksi ei ole kokonaisluvun minimi arvossa. Tämä erottaa käytössä olevat signaalit listasta. Jos signaalilla on joku tavu indeksi, on kyseinen signaali käytössä ja tälle korjataan indeksi. Ensimmäisenä asetetaan signaalille tämän hetken indeksi ja lisätään signaalille tarvittava määrä tilaa. Reuna ehtojen takia ennen tilan varausta tarkistetaan kolme ehtoa: Bitti indeksi, onko signaali tyyppi bitti tai null vai onko uusi indeksi vähemmän kuin vanha indeksi. Näillä tarkistellaan, tarvitseeko indeksiä kasvattaa. Jos bitti indeksi on ei ole 7 ja signaalin data tyyppi on bitti niin indeksiä ei tarvitse kasvattaa. ”OldBitIndex”-muuttujalla katsotaan, onko tavu muuttunut, jossa kyseinen bitti löytyy. ”goto” funktiota käytetään, jotta bitti indeksi muuttujaa ei muuteta siinä tapauksessa, kun tavua vaihdetaan.

”SizeOfDataType”-funktio palauttaa annetun signaalin tyypin koon perustuen mitä xml-tiedostossa lukee kyseisen signaalin kohdalla. Tästä saadun koon mukaan lisätään kyseiselle signaalille riittävä tila ja siirrytään seuraavaan, jos ei yllä olevia poikkeus tilanteita täytetä.

5.4 Pääohjelmakierto

Pieni sykli aika oli prioriteetti ohjelman pääkiertosilmukkaa toteuttaessa. Tämän takia ohjelma tekee asioita myös silloin kun se odottaa Adam moduuleilta kysyttyä tietoa.

TAULUKKO 1. Pääohjelmakierron päärakenne

Description	Receive outputs	Ask for inputs	Ask for inputs	Parse Mevea outputs while waiting answers	Receive inputs	Receive inputs
Time step	1	2	3	4	5	6
Mevea	Send					
Interface	Receive	Send	Send	Parse(Outputs)	Receive	Receive
Adam #1		Receive			Send	
Adam #2			Receive			Send

Send parsed Outputs	Send parsed Outputs	Parse Adam inputs while waiting	Reveive ack	Reveive ack	Send inputs to Mevea
7	8	9	10	11	12
					Receive
Send	Send	Parse(Inputs)	Receive	Receive	Send
Receive			Send		
	Receive			Send	

Taulukosta (ks. taulukko 1) näkee, kuinka ohjelma kierto kulkee. Ensimmäinen vaihe on vastaanottaa Mevean simulaatio mallista bitti taulukko, jonka järjestys tiedetään aikaisemmin parsitusta XML-tiedostosta, mutta ennen kuin ohjelma parsii tiedot kyseisestä taulukosta, lähetetään Adam moduuleille järjestyksessä kyselyt tulojen tiloista. Kun ohjelma on lähettänyt tarvittaville Adam moduuleille pyynnöt, niin parsii ohjelma odottaessa Mevealta saadun bitti taulukon. Tiedot parsitaan aikaisemmin luotuihin Adam moduuli objekteihin, jotta on helpompi myöhemmin lähettää data oikeisiin paikkoihin.

Parsimisen jälkeen vastaanotetaan jokaiselta moduulilta tulojen tilat ja lähetetään saman tien takaisin parsitut lähtöjen tilat. Jälleen kerran, kun ohjelma on odottamassa vastauksia Adam moduuleita, niin parsii ohjelma Adameilta saadut tulojen tilat samoihin objekteihin kuin aikaisemminkin. Toiseksi viimeisin vaihe on vastaanottaa jokaiselta Adam moduulilta ”Acknowledgement”-bitti, jotta tietää lähtöjen tila vaihtuneen.

Viimeisimpänä parsittu data Adameilta lähetetään Mevean simulaatio mallille takaisin ja näin ohjelma välittää tarvittavan tiedon kahden ohjelmiston välillä.

6 ONGELMAT

Analogisien lähtöjen ohjaus ostautui ongelmaksi hyvin aikaisessa vaiheessa. Adam moduulit palauttavat ”Acknowledgement”-viestin vasta, kun moduuli on saanut asetettua moduulin lähdön pyydettyyn arvoon. Tämä aiheutti ohjelman kierros ajan nousun jopa useaan sataan millisekuntiin ja toimeksiannon tavoitteena oli saada se alle 10 millisekuntiin.

Ongelma saatiin kierrettyä lähettämällä Adam moduuleille pyyntö analogi lähdön arvosta ja tarkistamalla jokaisella ohjelmakierroksella, onko vastaanotettu vastausta. Jos vastaus ei ole tullut, niin ohittaa ohjelma tämän vaiheen ja yrittää lukea uudestaan seuraavalla ohjelma kierrolla. Jos vastaus on tullut, niin lähetetään moduuleille uudet arvot.

Ongelman kiertäminen ei korjaa juuri syytä, analogi Adam moduulien hitautta, mutta tällein moduulien hitaus ei vaikuta muiden tulojen/lähtöjen luku/kirjoitus nopeuteen. Analogi lähtöjen hitaus sovellutuksessa ei vaikuta suuresti simulaation toimintaan, joten oli hyväksyttävää, että lähtöjen ohjaus tapahtuu hitaammin kuin muilla lähdöillä/tuloilla.

7 JATKOKEHITYS

Kuten ”ONGELMAT”-osiossa kerrotaan, on analogi lähtöjen asettamisessa hitautta. Tämä ei vaikuttanut tämänhetkiseen sovellutukseen, mutta voi tulevaisuudessa tulla vastaan rajoitteena. Vaihtoehtoisena ratkaisuna kiertämisen sijasta pystyisi katsoa löytyykö markkinoilla vastaavanlaista, mutta nopeampaa moduulia.

Jatkossa myös uusien ja erilaisten moduulien tai simulaatioympäristöjen tukeminen voi olla yksi niistä asioista, joita sovelletukseen voisi lisätä. Jotta muutosten tekeminen olisi järkevää, on kuitenkin mietittävä kuinka iso työ olisi tehdä edellä mainitut muutokset olemassa olevaan sovellutukseen vai pitäisikö tehdä uusi alusta lähtien.

Kun puhutaan sovellus kehittämisestä, voi aina sovellusta kehittää/optimoida enemmän ja paremmaksi, mutta toimeksiannon tuote on riittävä tämänhetkisen yrityksen tuotekehityksen tarpeisiin.

8 YHTEENVETO

Työn lähtökohtana oli saada mahdollisuus testata olemassa olevaa paalutuskoneen simulaatio mallia vasten testauksessa olevaa ohjelmistoa. Simulaatio mallia vasten testaus mahdollistaa yritykselle nopeamman ja turvallisemman tavan tutkia ja kehittää tuotteita varsinkin, kun tuote on hyvin massiivinen ja vaarallinen kone kuten paalutuskone, joka kokemattoman käsissä voi aiheuttaa aineellisia tai henkilövahinkoja. Myös koneen oleminen testaus kunnossa aiheuttaa yritykselle taloudellisen taakan, kun kallis kone seisoo testaus käytössä ja mahdollisesti myös vikaantuu tämän aikana, joka vaatii enemmän resursseja.

Rajapinnan ansiosta ohjelma pystytään testaamaan pienellä sijoituksella, kun testauksen pyörittämiseen menee vain murto-osa kustannuksista ja ei aiheuta lisä resurssointia työntekijöiden puolelta. Kustannukset koostuvat lähinnä mallin teosta ja testausympäristöön vaativista moduuleista, joista suurin osa voidaan uudelleen käyttää valmiissa tuotteessa tarpeen vaatiessa.

Sovellutuksen suunnittelu ja toteutus auttoi minua ammattilaisesti kasvamaan huomattavasti. Suurin asia minkä huomasin, on suunnittelu osion tärkeyden ja sen vaikutuksen toteutuksen nopeuteen. Ilman vahvaa pohjatyötä työn toteutus olisi vienyt enemmän aikaa. Työ on tätä myötä myös antanut kyvyn arvioida mahdollisten saatavien työtehtävien ajankulun arvioineissa.

9 LÄHTEET

Advantech Co., Ltd. 2018. advantech 6000. *advantech.* [Online] Elokuu 2018. http://advdownload.advantech.com/productfile/Downloadfile4/1-1M99LTH/ADAM-6000_User_Manaul_Ed_9.pdf.

—. 2018. advantech 6200. *advantech.* [Online] Elokuu 2018. http://advdownload.advantech.com/productfile/Downloadfile1/1-1MPZBF2/ADAM-6200_User_Manaul_Ed_3.pdf.

Junttan Oy. 2021. Junttan. *Junttan.* [Online] 2021. https://junttan.com/.

—. 2020. *Sisäinen materiaali.* Kuopio : s.n., 2020.

Mevea Ltd. 2020. *Sisäinen materiaali.* Lappeenranta : s.n., 2020.