Tampereen ammattikorkeakoulu
Sähkötekniikan koulutusohjelma
Sähkövoimatekniikka
Henri Eskman

Opinnäytetyö

Digitaalisen pienoisrautatien ohjaus Labviewillä

Työn ohjaaja Lehtori Lauri Hietalahti
Työn tilaaja TAMK
Tampere 5/2010

Tampereen ammattikorkeakoulu
Sähkötekniikan koulutusohjelma, sähkövoimatekniikka

Tekijä	Henri Eskman
Työn nimi	Digitaalisen pienoisrautatien ohjaus Labviewillä
Sivumäärä	42
Valmistumisaika	5/2010
Työn ohjaaja	Lehtori Lauri Hietalahti
Työn tilaaja	TAMK

TIIVISTELMÄ

Tässä työssä on toteutettu tietokoneohjaus Märklinin valmistamaan digitaaliseen pienoisrautatiehen Labview-ohjelmoinnilla. Tämä ohjelmointitapa on National Instrumentsin kehittämä, ja se perustuu C-kieleen.

Työn tarkoituksena oli harjoitella Labview-ohjelmointia tekemällä ohjelma digitaalisen pienoisrautatien ohjaukseen. Märklinin digitaalisia pienoisrautateitä on ollut olemassa 1980-luvulta lähtien. Tavoitteena on alun perin ollut monen veturin ohjaaminen samalla ohjaimella. Lisäksi kullekin veturille pitäisi voida säätää nopeus ja suunta erikseen ja jättää se kulkemaan halutuilla säädöillä. Perinteisessä analogisessa järjestelmässä vetureiden suuntaa ja nopeutta ei voi muuttaa yksilöllisesti. Tekniikan kehityttyä samaan digitaalijärjestelmään voidaan nykyään liittää muitakin laitteita, esim. vaihteita, opastimia, nostureita. Märklinin digitaalijärjestelmä on Motorolan valmistama. Siinä kiskot muodostavat CAN-väylän. Tässä työssä on käytetty Central Station 2 -keskusyksikköä, josta on Ethernet-yhteys tietokoneeseen. Tämän vuoksi tietokoneelta voidaan lähettää CAN-väylän komentoja Ethernetin välityksellä CS2:lle, joka lähettää komennot edelleen CAN-väylään. Ongelmaksi tässä työssä tulikin CS2-keskusyksikkö. CS2 ei reagoinut tietokoneelta lähetettyihin komentoihin. Vian syy on ilmeisesti laitteen ohjelmistoversio.

Tuloksena syntyi kuitenkin ohjelma, joka muodostaa komentoja järjestelmään ja lukee sieltä vastaanotettuja vastauksia. Ohjelmaa on mahdollista kehittää myöhemmin, kun yhteys CS2:een saadaan toimimaan.

Avainsanat Labview, pienoisrautatie, CAN-väylä, Ethernet, UDP

TAMK University of Applied Sciences
Electrical Department, Power Electric

Writer	Henri Eskman
Thesis	Digital Model Railroad Control with Labview
Pages	42
Graduation Time	5/2010
Thesis Supervisor	Lauri Hietalahti
Co-operating Company	TAMK

ABSTRACT

This thesis is about controlling digital model railroad with Labview. The digital model railroad system used in this thesis is manufactured by Märklin. Labview is a graphical programming language developed by National Instruments and it is based in C-programming language.

Parts used in this system are a Märklin digital model railroad, a Märklin Central Station 2 control device and a PC with Labview software. There is a CAN-bus in the rails of the model railroad. Each locomotive and accessory connected to the rails of the digital model railroad has decoders to receive commands from the CAN-bus. The Märklin CS2 device is connected to the CAN-bus. The CS2 sends commands to the bus and receives answers from the devices of the railroad system. CS2 is connected to the PC via Ethernet connection. The commands between the CS2 and the PC are sent in UDP package format.

The Labview program to control the railroad was made so that there are two while-loops in the main program. The first while-loop sends commands to the railroad and the second receives answers from the CS2 device and interprets them to the indicators of the software's display.

As a result there is a program that can control the model railroad. However, for some reason the CS2 device did not react to any commands sent from the PC. The problem can be in the software version of the CS2. The program made with Labview can be improved later when the connection CS2 device starts to work.

Keywords Labview, model railroad, CAN-bus, Ethernet, UDP

Sisällysluettelo

LYHENTEIDEN SELITYS 5

1 JOHDANTO 6

1.1 Työn tarkoitus 6
1.2 Järjestelmän rakenne 6
1.3 Märklinin digitaalinen pienoisrautatiejärjestelmä 6
1.4 CAN-komennot CS2:n ja tietokoneen välillä 9
1.5 Labview-ohjelmointi 11

2 TYÖN VAIHEET 16

2.1 Labview-esimerkit 16
2.2 Yhteyden kokeilu 19
2.3 Komennon muodostaminen 20
2.4 Suunniteltava ohjelma 29
2.5 Komentojen lähetyssilmukka 29
2.5.1 Järjestelmän komennot 29
2.5.2 Veturin komennot 31
2.5.3 Vaihteiden komennot 36
2.6 Vastaanottosilmukka 37

3 YHTEENVETO JA JOHTOPÄÄTÖKSET 39

LÄHTEET 42

Lyhenteiden selitys

CAN	Control Area Network, yleisin ajoneuvoissa käytettävä väylä
CAN-ID	viestin tunnistuksessa käytettävä koodi
CS2	Central Station 2, Märklinin ohjauslaite
DHCP	Ethernetissä käytettävä palvelin
DLC	Data Lenght Code, datan pituuskoodi
Enum	Numeerinen kontrolleri Labviewissä, jotakin suuretta vastaava numero voidaan lähettää tällä tavalla
IP	Internet Protocoll
MFX	Uusin Märklin Digital -versio
MM1	Märklin Motorola 1, ensimmäinen Motorolan valmistama Märklinin digitaalijärjestelmä
MM2	Märklin Motorola 2
MM3	Märklin Motorola 3
TCP	Transmission Control Protocoll, tietoliikenneyhteysprotokolla Internetiin kytkettyjen tietokoneiden välillä
UDP	User Datagram Protocoll, yksi Ethernetissä käytettävistä protokollista

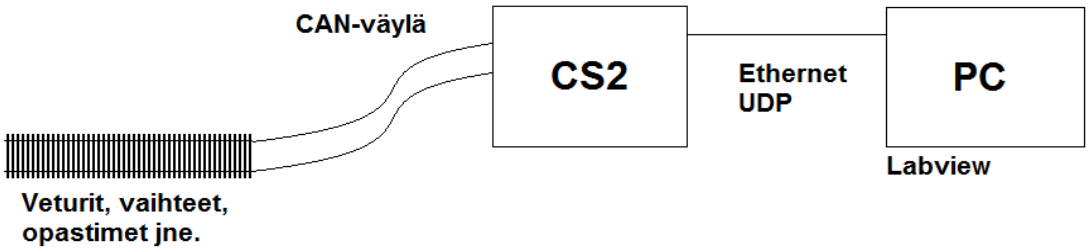
1 Johdanto

1.1 Työn tarkoitus

Työn tarkoituksena oli tehdä digitaaliseen pienoisrautatiehen ohjaus Labview-ohjelmoinnilla. Digitaalisen pienoisrautatien ideana on, että samalla ohjaimella on mahdollista ohjata useampaa veturia tai laitetta. Perinteisessä analogisessa pienoisrautatiessä yhdellä ohjaimella pystytään ohjaamaan käytännössä vain yhtä veturia. Labview-ohjelmointi on National Instrumentsin kehittämä graafinen ohjelmointikieli, joka perustuu C-kieleen.

1.2 Järjestelmän rakenne

Järjestelmä koostuu Märklinin valmistamasta digitaalisesta pienoisrautatiestä, Märklinin Central Station 2 (CS2) -ohjauslaitteesta ja tietokoneesta. Pienoisrautatiessä kiskoissa on CAN-väylä (CAN = control area network). CS2-ohjauslaite on kytketty tähän CAN-väylään. CS2-laite on kytketty Ethernet-yhteyden välityksellä tietokoneeseen. Tietokoneen ja CS2:n kommunikaatiossa käytetään UDP-paketteja (UDP = User Datagram Protocol) /2/. Järjestelmän rakenne on hahmoteltu kuvaan 1.



Kuva 1. Järjestelmän rakenne

1.3 Märklinin digitaalinen pienoisrautatiejärjestelmä

Märklinin digitaalisessa pienoisrautatiessä kiskoissa on CAN-väylä. Ohjauslaite lähettää väylään komentoja heksadesimaalimuodossa. Jokaisessa järjestelmän laitteessa (veturit, vaihteet, opastimet ym.) on dekooderi, joka tulkitsee komentoja ja lähettää vastauksen ohjauslaitteelle. Kiskoissa kuljetetaan myös laitteiden tarvitsema energia.

Märklinin CAN-viesti koostuu 29 bitin pituisesta viestin tunnistusosasta (Meldungskennung), 4 bitin pituisesta viestin pituuden kertovasta osasta (DLC = data lenght code) ja 8 datatavusta. Viestin tunnistusosassa on prioriteetti (Prio), komento (Command), vastaus (Resp.) ja Hash-osa (Hash on jonkinlainen viestin erottelukoodi). Viestin pituuden ilmoittava osa ilmoittaa, montako datatavua viestiin kuuluu. Datatavuihin sijoitetaan lähetettävä tieto. Viestissä tulee olla 13 tavua. Tyhjät paikat täytetään nollilla. Taulukossa 1 on Märklinin järjestelmässä lähetettävien CAN-pakettien rakenne /2/.

Taulukko 1. CAN-viestin rakenne /2/

Meldungskennung				DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Prio	Command	Resp.	Hash	DLC	D-Byte 0	D-Byte 1	D-Byte 2	D-Byte 3	D-Byte 4	D-Byte 5	D-Byte 6	D-Byte 7
2+2 Bit	8 Bit	1 Bit	16 Bit	4 Bit	8 Bit	8 Bit	8 Bit	8 Bit	8 Bit	8 Bit	8 Bit	8 Bit
Message Prio	Kommando Kenn-zeichnung	Cmd / Resp.	Kollisions-auflösung	Anz. Daten-bytes	Daten	....						

Prioriteettia käytetään Märklinin Kommunikationsprotokoll -oppaan mukaan versiossa 2 (V2.0) tai uudemmassa. Versiossa 1 (V1.0) prioriteettia ei käytetä vaan sen tilalle sijoitetaan nollia (0b0000). Tässä kohtaa jää hieman tulkinnan varaa siihen, mitä versio 1:llä ja versio 2:lla tarkoitetaan. Kyseessä voi olla jokin Märklinin CS2-tiedonsiirtoversio tai Märklinin digitaalijärjestelmän versio. Märklinin digitaalijärjestelmän ensimmäistä versiota nimitetään MM1:ksi (Märklin Motorola 1). Se on Märklinin alkuperäinen 80-luvulla julkaistu versio, jossa oli mahdollista käyttää 80 veturia. Sen ominaisuutena oli yksi funktio, jolla veturin valot voitiin sytyttää ja sammuttaa. Seuraava MM2-versio tuli 90-luvulla. Se mahdollisti vetureissa käytettäväksi viisi eri toimintoa: valot, äänimerkki, savu, kulkuäänet ja vaunujen irrottaminen. MM3 versioon on lisätty osoitteita aina 255 asti. Lisäksi siinä on mahdollisuus lisälaitteiden (vaihteet, opastimet ym.) digitaaliohjaukseen. Viimeisin ja kehittynein versio on nimeltään MFX. Sen ominaisuutena on automaattinen osoitteistaminen, ja siinä on mahdollista käyttää suurta määrää erilaisia toimintoja /4/. Prioriteetin käyttö on selvitettävä kokeilemalla myöhemmin, kun komentoja CS2:lle päästään lähettämään. Prioriteetit ovat seuraavia:

Prioriteetti 1: Stop, Go, oikosulkuilmoitus
Prioriteetti 2: takaisinkytkentä
Prioriteetti 3: veturin pysäyttäminen
Prioriteetti 4: veturin tai laitteen komento
Loput ovat vapaita eli näitä voi tulla tulevaisuudessa lisää. /2/

Prioriteetti 2 eli takaisinkytkentä täytyy myös kokeilla. Se voi tarkoittaa useampaakin eri asiaa tai toimintoa tässä järjestelmässä, esimerkiksi laitteen dekooderin lähettämää vastausta tai jotakin muuta. Märklinillä on myös erityinen takaisinkytkentälaite, johon viedään signaali radan pätkästä. Se kertoo veturin juuri ylittäneen kyseisen radan pätkän. Veturin voi myös pysäyttää useammalla eri tavalla: virran voi katkaista kokonaan, nopeuden voi asettaa arvoon 0 tai lähettää pysäyttämiskäskyn. Näin ollen prioriteetti 3 toimintaperiaatekin täytyy selvittää myöhemmin.

Komento-osa muodostuu 8 bitistä ja kertoo, mistä komennosta on kyse. Komentoja ovat esimerkiksi järjestelmän pysäyttäminen ja käynnistäminen, veturin nopeuden ja suunnan asettaminen sekä lisälaitteen ohjaus, esimerkiksi vaihteen kääntäminen. Taulukossa 2 on kaikki järjestelmässä käytetyt komennot. Komentojen nimet ovat samat saksankieliset nimitykset, joita on käytetty Märklinin saksankielisessä Kommunikationsprotokoll-oppaassa /2/. Taulukossa on myös komentoa vastaava heksadesimaaliluku ja CAN-ID-luku. Komentojen ja CAN-ID:n käyttö täytyy selvittää myöhemmin, kun yhteys CS2:teen saadaan toimimaan. Taulukon tarkoituksena on vain kertoa, minkälaisia komentoja järjestelmään lähetetään. Komentojen merkitystä ei tässä vaiheessa selitetä eikä suomenneta. Ne voidaan kääntää sitten, kun niitä tarvitaan. Tässä vaiheessa oleellisia ovat vain kaksi heksadesimaalilukua. Yksi heksadesimaaliluku vastaa 4 bittiä. Näin ollen kahdeksan bitin komento-osa muodostuu kahdesta heksadesimaaliluvusta.

Taulukko 2. Komennot

Komennon nimi (saks.)	Hex	CAN-ID
System Stopp	0x00	0x00
System Go	0x00	0x00
System Halt	0x00	0x00
Lok Nothalt	0x00	0x00
Lok Zyklus Beenden	0x00	0x00
System MFX Neuanmeldezähler setzen	0x00	0x00
System Überlast	0x00	0x00
System Status	0x00	0x00
Lok Discovery	0x01	0x02
MFX Bind	0x02	0x04
MFX Verify	0x03	0x06
Lok Geschwindigkeit	0x04	0x08
Lok Richtung	0x05	0x0A
Lok Funktion	0x06	0x0C
Read Config	0x07	0x0E
Write Config	0x08	0x10
Zubehör Schalten	0x0B	0x16
Teilnehmer Ping	0x18	0x30
Statusdaten Konfiguration	0x1D	0x3A

Vastausosa Resp. ilmoittaa, onko kyseessä komento vai vastaus. Jos luku on 0, kyseessä on komento. Jos luku on 1, kyseessä on vastaus.

Hash muodostuu 16 bitistä eli se ilmoitetaan 4 heksadesimaaliluvulla. Hash on kiinteä osa jokaista viestiä. Sen tarkoitus on helpottaa viestien sotkeutumistilanteiden ratkaisua. Jokaiseen komentoon lisätään Hash-kohtaan 0x0300 /2/. Taulukossa 3 on Hashin muodostuminen. Ensimmäisellä rivillä on luku heksadesimaalimuodossa ja toisella binaarimuodossa.

Taulukko 3. Hashin muodostuminen

Hash			
0	3	0	0
0000	0011	0000	0000

Viestin pituuden ilmoittavaan DLC-osaan (data lenght code) tulee kyseisessä viestissä olevien datatavujen lukumäärä. Esimerkiksi jos viestiin sisältyy 5 datatavua, niin DLC on 5.

Datatavuihin sijoitetaan lähetettävä tieto. Esimerkiksi veturille, jonka osoite on 16 (0x10), lähetetään nopeuden arvo 800 (0x03 20):

000400300 6 00 00 00 10 03 20 00 00

Alussa 000400300 on viestin tunnistus osa. 6 on datatavujen määrän kertova bitti (6 kpl datatavuja). 00 00 00 10 on osoite (datatavut 0 - 3). 03 20 on nopeus 800 (datatavut 4 - 5). Järjestelmässä nopeus ilmoitetaan asteikolla 0 - 1024 ilman mittayksikköä. Kaksi viimeistä tavua täytetään nollilla, koska viestissä tulee olla 13 tavua /2/.

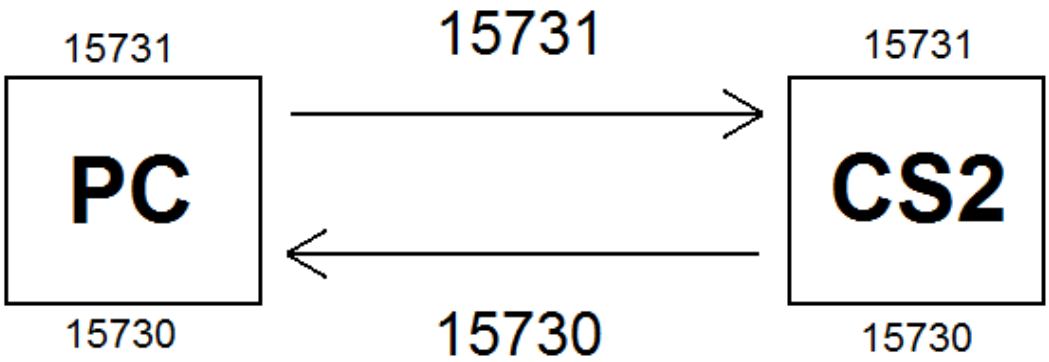
1.4 CAN-komennot CS2:n ja tietokoneen välillä

Tietokone ja CS2 kytketään toisiinsa Ethernet-yhteyden välityksellä. Tietokoneen ja CS2:n välillä tieto siirretään UDP-paketeissa. CAN-viesti välitetään Ethernetin välityksellä tietokoneelta CS2:lle ja siitä eteenpäin CAN-väylään. Paketti on 13 tavun mittainen.

Muun pituiset paketit jätetään huomioimatta. Kommunikaatiossa käytetään kahta kiinteää porttia:

Portti 15731: CS2 vastaanottaa, tietokone lähettää,
Portti 15730: tietokone lähettää, CS2 vastaanottaa. /2/

Kuvassa 2 on vielä hahmotettu yhteyden toiminta. Kummassakin laitteessa on siis portit 15731 ja 15730. Toinen laite käyttää toista näistä porteista lähetykseen ja toinen vastaanottoon.

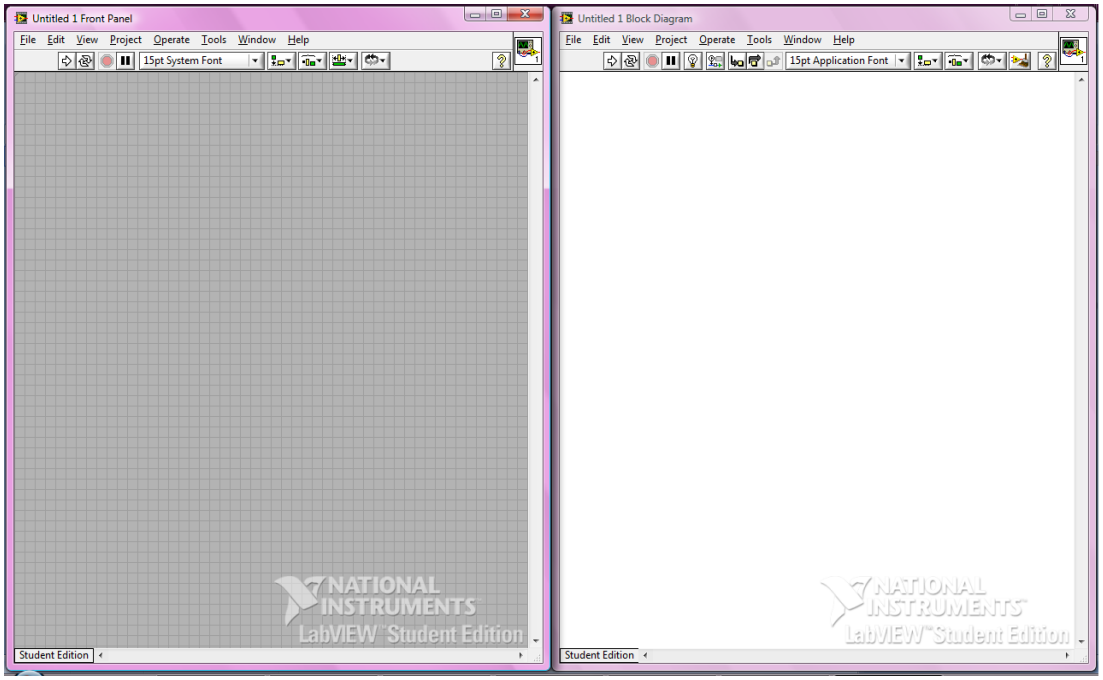


Kuva 2. Portit: PC:ltä CS2:lle lähetetään portista 15731, toisin päin käytetään porttia 15730.

Tässä työssä on toteutettu Labviewillä ohjelma, joka muodostaa komennon ja lähettää sen ja tulkitsee vastaanotettuja paketteja.

1.5 Labview-ohjelmointi

Labview-ohjelmointi on graafista ohjelmointia. Labviewissä on kaksi näkymää: etupaneeli (Front Panel) ja palikkakaavio (Block Diagram). Kuvassa 3 on Labviewin näkymät: vasemmalla on etupaneeli (harmaa ruudukollinen tausta) ja oikealla on palikkakaavio (valkoinen tausta). Etupaneeliin tulevat ohjelman ohjauspainikkeet tms. ja palikkakaavioon tulee ohjelman ”ohjelmointikieli”, tässä tapauksessa siis graafisessa muodossa.



Kuva 3. Labviewin etupaneeli ja palikkakaavio

Palikkakaavioon tulee ohjelmoinnissa yleisesti käytettyjä rakenteita. Käytetyimpiä rakenteita ovat while-silmukka (While Loop), for-silmukka (For Loop) ja case-rakenne (Case Structure).

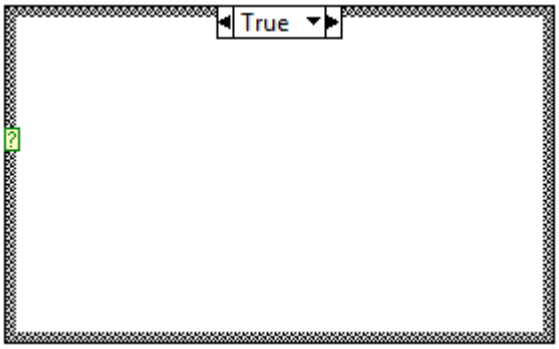
Kuvassa 4 on while-silmukka Labviewin palikkakaaviossa. Harmaa neliötä muistuttava laatikko on silmukka. Silmukan sisälle tehdään haluttu ohjelma. While-silmukkaa suoritetaan niin kauan, kunnes se pysäytetään. Silmukan vasemmassa alareunassa oleva neliö, jonka sisällä on i-kirjain, laskee, montako kertaa silmukka on suoritettu. Silmukan oikeassa alareunassa on stop-painike, joka on johdotettu silmukan pysäytysterminaaliin.

Kuva 4. While-silmukka

Kuvassa 5 on for-silmukka Labviewin palikkakaaviossa. For-silmukalle määritetään etukäteen, kuinka monta kertaa se suoritetaan. Vasemmassa yläkulmassa olevaan laatikkoon, jonka sisällä on N-kirjain, johdotetaan numero, joka määrää suorituskertojen lukumäärän.

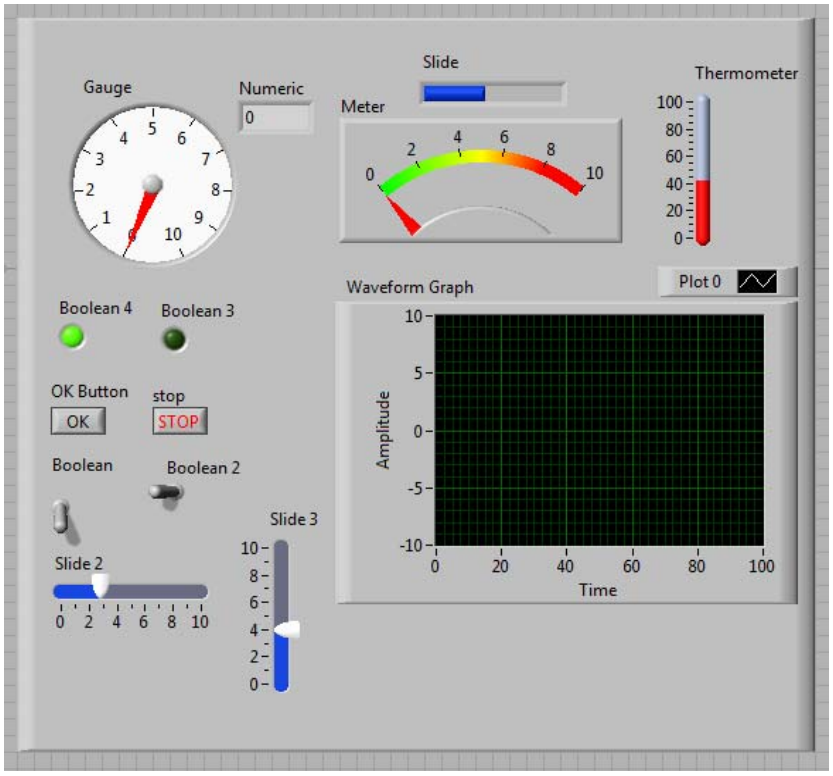
Kuva 5. For-silmukka

Kuvassa 6 on case-rakenne. Siihen voidaan sijoittaa useita vaihtoehtoisia ohjelman suoritusrakenteita. Esimerkiksi vaihtoehdot tosi ja epätosi voidaan suorittaa silloin, kun jokin ehto on joko tosi tai epätosi. Case-rakenteessa on ylhäällä keskellä nuolipainikkeita, joilla voidaan valita suoritettava rakenne. Vasemmalla keskellä on kysymysmerkki, johon johdotetaan esimerkiksi kytkin, joka määrittää suoritettavan vaihtoehdon.



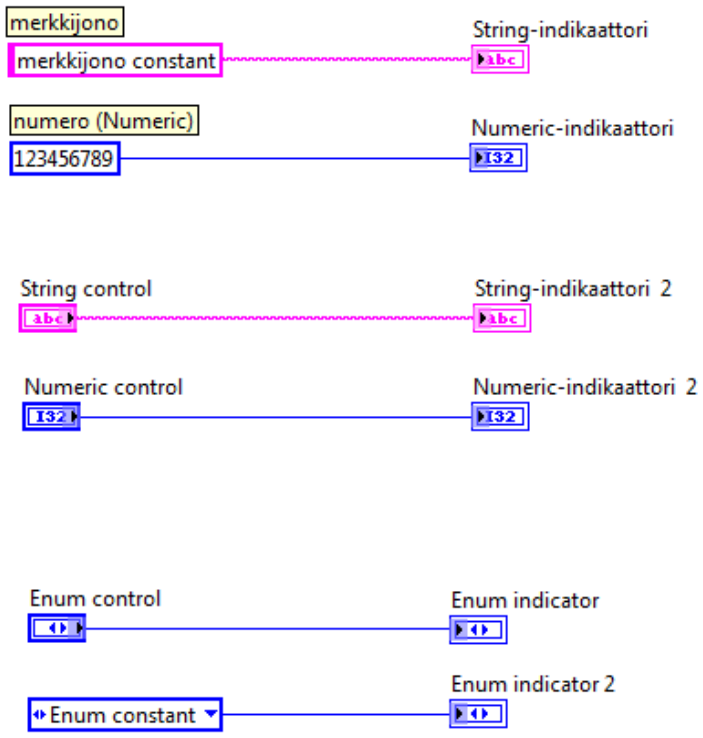
Kuva 6. Case-rakenne

Labviewin etupaneeliin voidaan sijoittaa ohjelman hallintapainikkeita ja jonkin suureen mittaustuloksen ilmoittavia mittareita. Kuvassa 7 on Labviewin etupaneeliin sijoitettu painikkeita, kytkimiä, liukukytkimiä, viisarimittareita, numeerinen mittari, pyöreitä led-indikaattoreita, lämpömittari ja aaltomuotonäyttö. Lisäksi etupaneeliin voidaan tehdä erilaisia koristeluja selkeyttämään, millä painikkeella ohjataan mitäkin suuretta. Tässä kuvassa on taustalle piirretty harmaa laatikko.

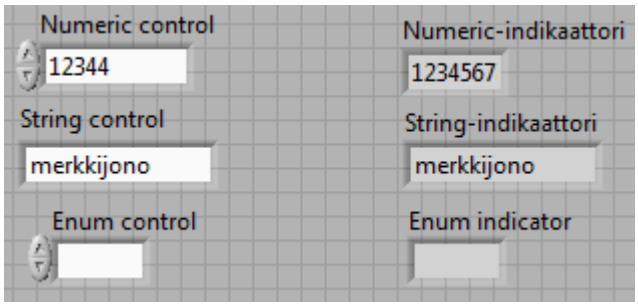


Kuva 7. Esimerkkejä etupaneeliin sijoitettavista kytkimistä ja mittareista

Numeerisia arvoja voidaan Labview-ohjelmaan syöttää etupaneelista numeerisilla kontrollereilla tai kiinteästi palikkakaaviosta constant-palikoilla. Merkkijonoja (String) käsiteltäessä voidaan toimia samalla tavalla. Lisäksi enum-toiminnolla on mahdollista lähettää jotakin suuretta tai asiaa vastaava numero. Jos merkkijonon tai numeron syöttö on tehty kontrollerina etupaneeliin, sitä on mahdollista muuttaa ennen ohjelman suorittamista tai mahdollisesti ohjelmaa suoritettaessa. Jos luku tai merkkijono lähetetään constant-palikalla palikkakaaviossa, siihen ei voida vaikuttaa ohjelman aikana lainkaan. Kuvassa 8 ovat em. palikat palikkakaaviossa ja kuvassa 9 ne ovat etupaneelissa.



Kuva 8. Merkkijonojen ja numeroiden syöttötavat palikkakaaviossa

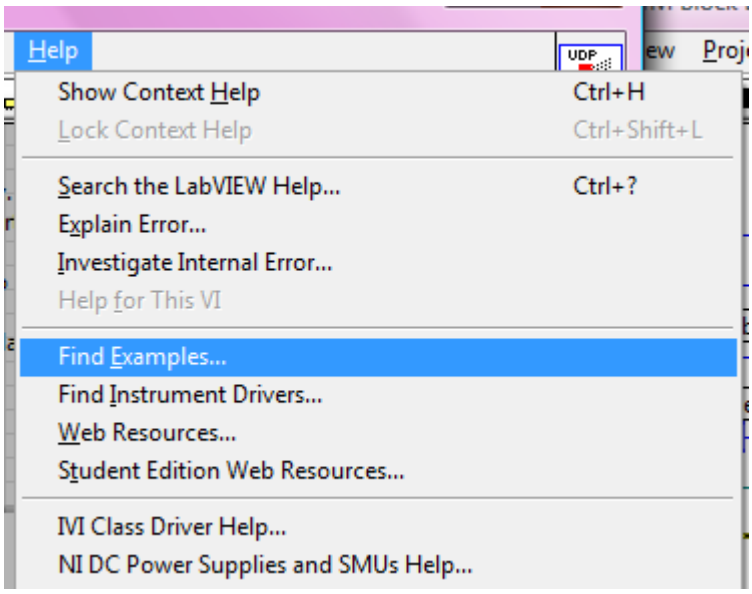


Kuva 9. Merkkijonojen ja numeroiden syöttö etupaneelissa

2 Työn vaiheet

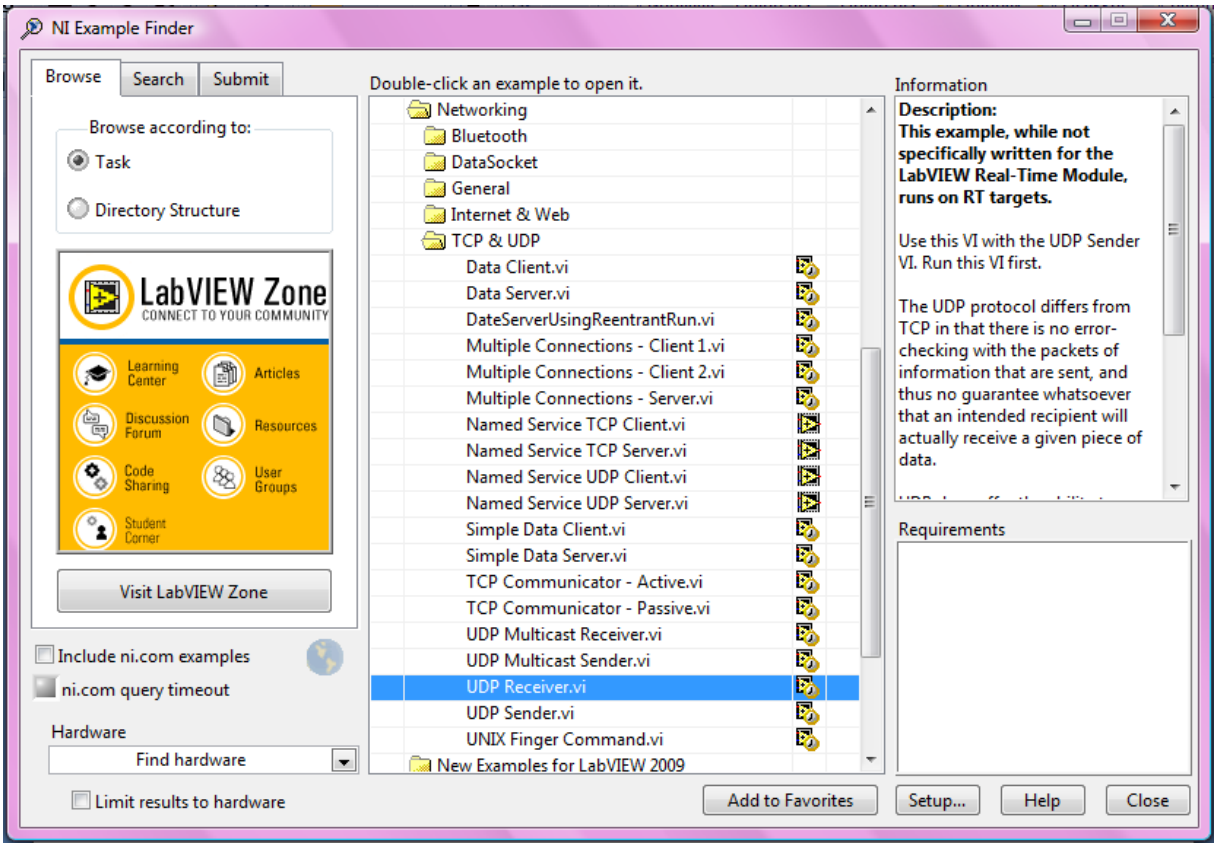
2.1 Labview-esimerkit

Labviewissä on mukana esimerkkejä yleisimmistä ohjelman palasista. Labviewin Help-valikosta löytyy kohta Find Examples. Kuvassa 10 on avattuna Help-valikko ja Find Examples. Aina kun Labviewillä alkaa tehdä jotakin, kannattaa esimerkeistä katsoa, löytyykö sieltä jotakin apua. Tähän työhön on tarkoituksena hakea esimerkkejä UDP-yhteyden käytöstä.



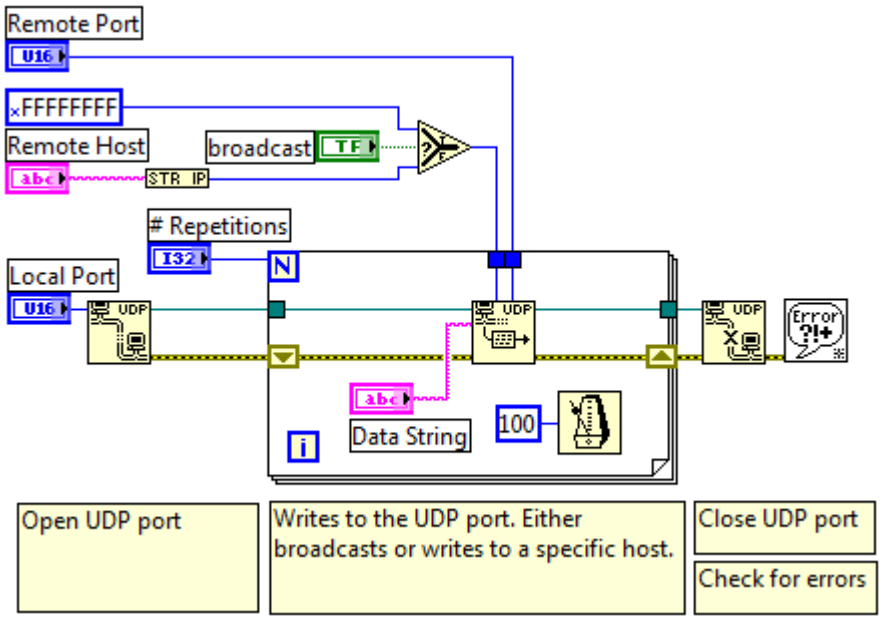
Kuva 10. Help-valikko ja Find Examples-kohta /1/

Esimerkeistä löytyi Networking-hakemistosta ja sen alta TCP & UDP-hakemistosta UDP Sender ja UDP Receiver esimerkit (TCP = Transmission Control Protocoll). Kuvassa 11 on Example Finder.



Kuva 11. Example Finder /1/

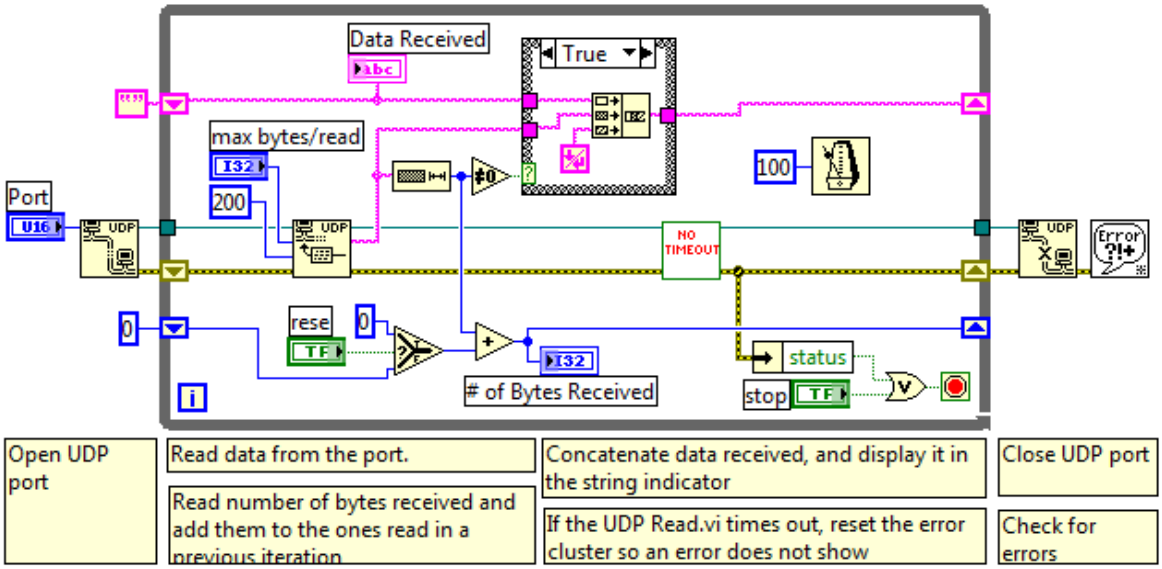
Aloitetaan tarkastelemalla UDP Sender -esimerkkiohjelmaa. Kuvassa 12 on UDP Senderin palikkakaavio (Block Diagram), jonka alareunassa on kommentteja englanniksi. Ohjelmassa avataan aluksi UDP-portti. Local Portilla avataan paikallinen portti, josta paketti lähetetään. Keskellä on for-silmukan (For Loop) sisässä UDP-write-palikka, jolla lähetetään UDP-viesti. UDP-write-palikkaan on kytketty kytkentä, jolla voidaan valita lähetys kaikkiin osoitteisiin tai johonkin tiettyyn osoitteeseen. Tätä voidaan yksinkertaistaa kytkemällä tähän pelkästään CS2:n IP-osoite. Data Stringistä lähetetään merkkijono. For-silmukan sisässä on 100 ms odotusaika silmukan pyörähtämisten välillä. #Repetitions -kohdalla määritetään for-silmukan suorituskertojen määrä. Silmukan jälkeen suljetaan UDP-portti ja sen jälkeen puretaan mahdolliset virheet ohjelman suorittamisesta ja tulostetaan ne näytölle. Tätä esimerkki ohjelmaa voidaan myöhemmin muokata ja hyödyntää UDP-pakettien lähettämisessä CS2:lle.



Kuva 12. UDP Sender /1/

Avataan seuraavaksi UDP Receiver -esimerkki. Kuvassa 13 on UDP Receiverin palikkakaavio. UDP Receiver -ohjelmassa aluksi avataan UDP-portti, joka vastaanottaa paketteja. Sen jälkeen tulee while-silmukka (While Loop), jota suoritetaan niin kauan, kunnes se pysäytetään. Silmukan sisällä on useita kytkentöjä. Oleellisin osa on UDP-read-palikka. Siihen kytketään luettavien tavujen määrä sekä aika, jonka palikka käyttää sopivan paketin odottamiseen (time out). Toiselta puolelta tulee ulos vastaanotettu merkkijono. Lyhyesti selitettynä ylhäällä oleva kytkentä kokoaa vastaanotetut paketit pinoon ja esittää ne listana ohjelman etupaneelissa (Front

Panel). Alapuolella on laskurikytkentä vastaanotettujen tavujen määrälle. Keskellä on mielenkiintoinen palikka ”No Time Out”. Se on aliohjelma, joka time out -virheilmoituksen tullessa kumoaa virhevirheilmoituksen ja sallii ohjelman jatkaa toimintaa normaalisti. Todennäköisesti lopullisessa ohjelmassa joudutaan käyttämään tämän kaltaista kytkentää, koska paketin odottamiseen voi mennä pitempi aika, kuin on määritetty. Silmukan sisällä on jälleen 100 ms:n odotusaika suorituskertojen välille. Silmukan jälkeen on yhteyden sulkemispalikka ja virheilmoitusten tulostuspalikka. Tätä esimerkkiohjelmaa voidaan muokata paremmin sopivaksi lopulliseen ohjelmaan.



Kuva 13. UDP Receiver /1/

2.2 Yhteyden kokeilu

Kun kommunikaatioesimerkkiin kuuluvat ohjelmat on löydetty, voidaan kokeilla, minkälaisia paketteja CS2 lähettää vai lähettääkö se mitään. UDP Receiver asetettiin kuuntelemaan CS2:lta tulevia paketteja. Mitään ei tullut. CS2 kytkettiin suoraan kiinni tietokoneeseen tavallisella ja ristiin kytketyllä verkkokaapelilla. Sittenkään ei tullut mitään. Tämän jälkeen CS2 kytkettiin koulun DHCP-palvelimen välityksellä tietokoneeseen. Nytkään ei kuulunut mitään. Kolmas tietokone asetettiin Etherpeek-ohjelman (tarkoitettu Ethernet-yhteyden liikenteen selvittämiseen) avulla tarkkailemaan liikennettä CS2:n ja tietokoneen välillä. Nyt huomattiin, että CS2 ei lähetä mitään. UDP Senderillä lähetettiin yksinkertainen pelkästään nollia sisältävä stop-komento laitteelle. Etherpeek-ohjelma havaitsi tietokoneelta lähetetyn paketin,

mutta CS2 ei reagoinut siihen mitenkään. Yhteys toiminut, mutta CS2 ei vastaa saamiinsa viesteihin.

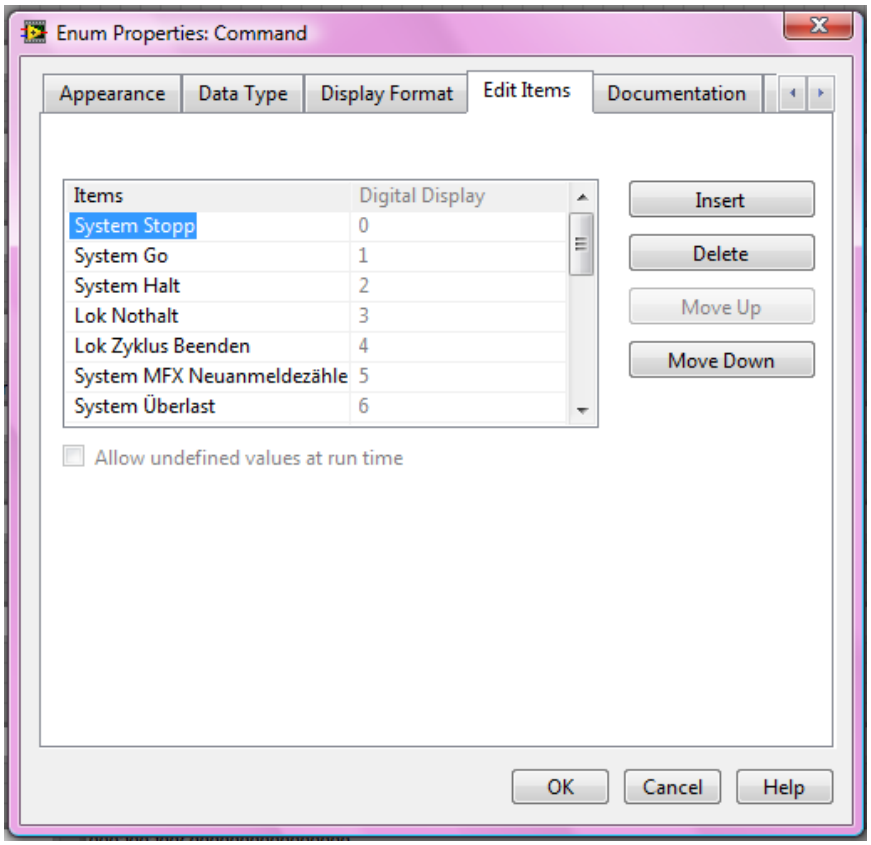
CS2 ohjelmisto päivitettiin Märklinin kotisivuilta löytyvällä päivityksellä, joka ladattiin muistitikulle. Se kytkettiin CS2:n USB-liitäntään ja CS2 latasi päivityksen tikulta. Tälläkään ohjelmalla yhteys ei toiminut.

Seuraavaksi lähetettiin sähköpostia Märklinille. Märkliniltä vastattiin: ” *Die Central Station 60213 kann nach dem Aufspielen des neuen Updates, welches noch im Mai erscheinen soll, mit dem PC gesteuert werden. Das CS2-CAN-Protokoll haben wir dieser Email beigefügt.*”/3/ Siis toukokuussa tulevalla päivityksellä sen pitäisi toimia. Liitteenä oli sama Kommunikations Protokoll PDF -tiedosto, joka oli saatavilla heidän nettisivuiltaan. Nettisivuilla oli vielä lisäys, että päivityksen saa vain kytkemällä CS2:n suoraan Internetiin. CS2 oli kytkettynä koulun verkkoon ja siis myös Internetiin. Päivitystä yritettiin ladata, mutta se ei onnistunut. CS2 ilmoitti, että viimeisin päivitys ei onnistunut. Varmaan jokin koulun verkon palomuuri estää päivityksen lataamisen. Ohjelman tekemistä oli nyt vain pakko jatkaa ja asiaan on palattava myöhemmin.

2.3 Komennon muodostaminen

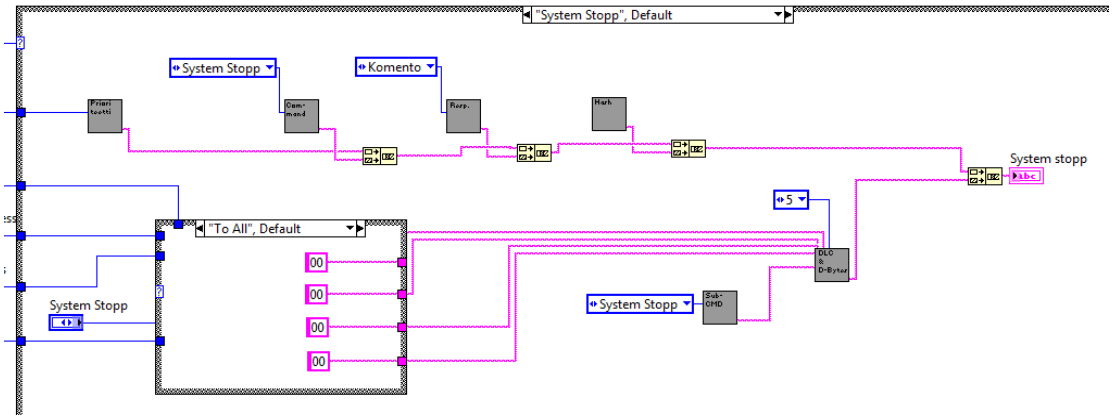
Seuraavaksi tehtiin Labviewillä ohjelma, joka muodostaa tarvittaessa minkä tahansa järjestelmässä käytettävän komennon. Tämä ohjelma on enemmänkin työkalu, kuin jokin osa tulevaa ohjelmaa. Kaikkia komentoja tuskin tarvitsee käyttää, mutta samalla vaivalla tekee sellaisen työkaluohjelman, jolla voi pikaisesti kokeilla kaikkia komentoja. Tästä ohjelmasta voidaan tarvittavat osat kopioida myöhemmin varsinaiseen ohjelmaan. Ohjelmoinnissa on hyvänä tapana käyttää osien nimeämisessä kielenä englantia. Tässä ohjelmassa on hyvien tapojen vastaisesti käytetty saksaa, englantia ja suomea sekaisin. Saksankieliset termit tulevat sen takia, että valmistajan lähdemateriaali on saksan kielellä. Niiden kääntäminen muulle kielelle voi myöhemmin haitata termien etsimistä lähdemateriaalista. Englannin kieli tulee ohjelman kielen sekä sen takia, että tekijä joutui työtä tehdessään työskentelemään jonkin aikaa englanninkielisessä ympäristössä. Muiden mielipiteen kysyminen ohjelmasta oli näin helpompaa. Suomenkieliset kohdat tulevat tekijän äidinkielen takia. Kaikille termeille ei ole keksitty englanninkielisiä nimiä. Pahoittelut sekamelskasta.

Komennon muodostamista varten Labviewissä luotiin case-rakenne (Case Structure), joka voidaan kääntää suorittamaan valittua osaa ohjelmasta. Lisäksi etupaneelissa luotiin enum-kontrolleri (Control), jolla voidaan ohjata esim. case-rakennetta. Sen valikoista valittiin kohta ”Edit Items”, minkä jälkeen näytölle tuli näkymä (Enum Properties) johon voidaan luoda halutut ohjaimen osat. Sen kummemmin mitään saksankielisen komennon nimeä kääntämättä lisättiin kaikki komennot enum-kontrolleriin. Kuvassa 14 on Enum Properties -valikko.



Kuva 14. Enum Properties -valikko

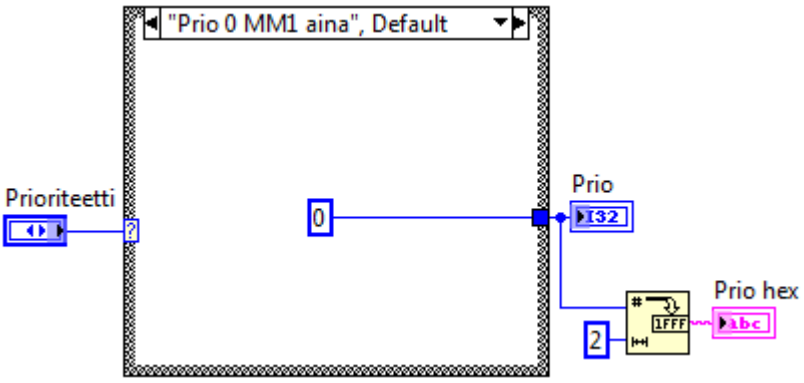
Tämän jälkeen enum-kontrolleri kytkettiin kiinni case-rakenteen kysymysmerkkikohtaan, ja case-rakenteen valikoista valittiin ”Add Case For Every Value”, eli lisää case jokaiselle arvolle. Nyt voitiin ryhtyä muodostamaan komentoa. Kuvassa 15 on kuva yhdestä valmiin komennon muodostamisohjelman casesta. Harmaat palikat ovat aliohjelmia, jotka muodostavat kunkin tavun 13 tavuisesta paketista. Palataan näiden luomiseen myöhemmin.



Kuva 15. Valmis komennonmuodostusohjelma

Komennon muodostamista varten luotiin aliohjelmat (subVI:t) kaikille tavuille, jotka tulevat osaksi 13 tavun pakettia. Näitä ovat siis prioriteetti, komento, vastaus, hash, datatavujen määrän ilmoittava DLC ja datatavut.

Ensimmäisenä tehtiin aliohjelma prioriteetille samaan tapaan kuin pääohjelmassa, eli luotiin case-rakenne ja enum-kontrolleri. Näihin tehtiin osiot prioriteeteille 0, 1, 2, 3 ja 4. Kuvassa 16 on prioriteetin aliohjelman palikkakaavio. Ohjelma toimii siten, että alussa valitaan enum-kontrollerilla prioriteetti. Tämän jälkeen case-rakenne kääntyy suorittamaan valittua prioriteettia, joka lähetetään lukuna ulos case-rakenteesta ja muutetaan heksadesimaalioperaattorilla heksadesimaalimuotoon. Tämä on turha operaatio luvuille 0, 1, 2, 3 ja 4, koska ne ovat samoja heksadesimaalimuodossa. Labviewin erilaisten toimintojen harjoitteluksi tämä oli kuitenkin hyödyllinen vaihe.

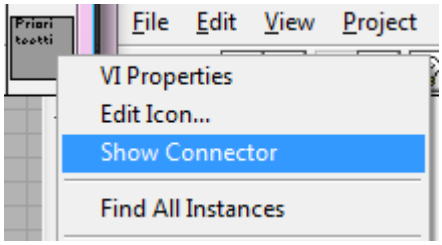


Kuva 16. Prioriteetin palikkakaavio

Tämän jälkeen prioriteetista tehtiin aliohjelma Labviewissä etupaneelin oikeassa yläreunassa sijaitsevasta ikonista. Ikoni on kuvassa 17.

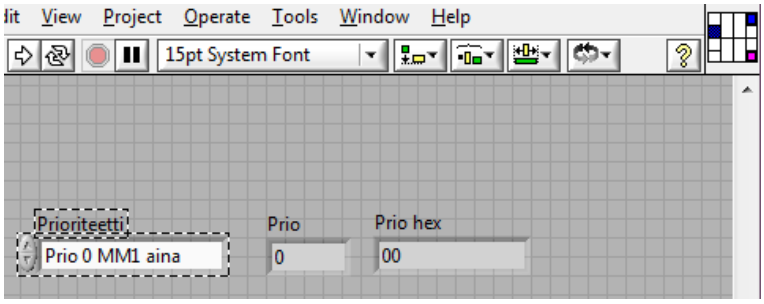
Kuva 17. Etupaneelin yläreunassa sijaitseva ikoni

Kun hiiren osoitin viedään ikonin päälle ja painetaan hiiren oikeaa painiketta, tulee näkyviin valikko, josta valitaan ”Show Connector”. Kuvassa 18 on valikko avattuna.



Kuva 18. Show Connector

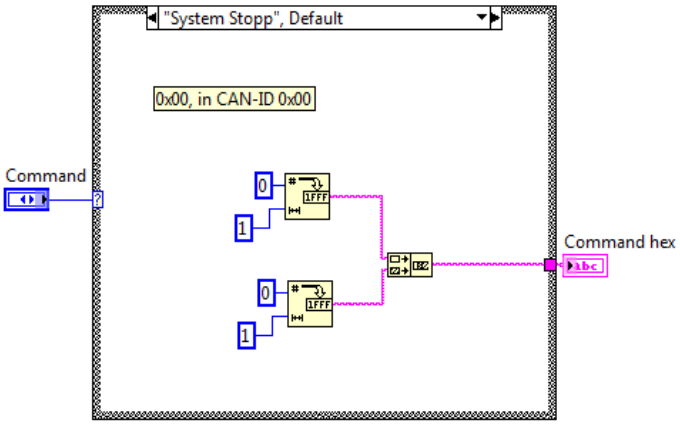
Ikonin paikalle aukeaa ruudukko. Painamalla ruutua ja sen jälkeen haluttua sisääntuloa tai ulosmenoa saadaan ne kytkettyä. Kuvassa 19 on ruudukko ja aktiivinen sisääntulo.



Kuva 19. Sisääntulojen ja ulosmenojen kytkeminen

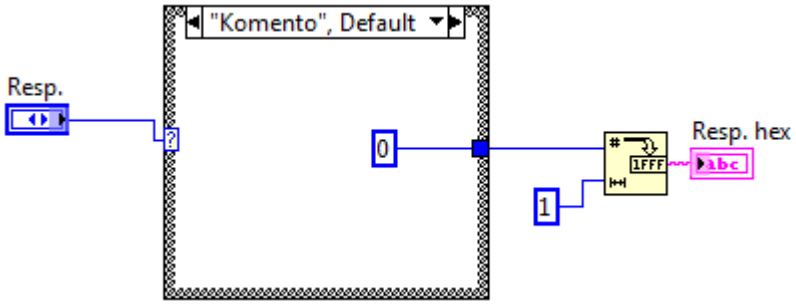
Tämän jälkeen ikoni palautetaan näkyviin (oikeaa hiiren painiketta painetaan ruudukon päällä ja valitaan ”Show Icon”) ja ohjelma tallennetaan. Tämän jälkeen ikoni vain siirretään hiiren vasenta painiketta pohjassa pitämällä pääohjelman palikkakaavioon. Kun prioriteetti-aliohjelma oli siirretty pääohjelmaan, sen sisääntuloon kytkettiin kontrolleri, jolla voidaan valita prioriteetti. Prioriteetti-aliohjelman ulostulo kytkettiin kaksiosaiseen merkkijonot yhdistävään operaattoriin, jonka toiseen sisääntuloon kytketään seuraavaksi komento-aliohjelman ulostulo.

Seuraavaksi tehtiin komento-aliohjelma. Jälleen kerran tehtiin case-rakenne, johon sijoitettiin kaikki taulukon 2 komennot. Ideana oli se, että alkupäästä valitaan komento ja loppupäästä tulee komento ulos heksadesimaaleina. Kuvassa 20 on komento-aliohjelman palikkakaavio. Case-rakenteen sisässä kaksi numeroa muutetaan heksadesimaalimuotoon ja yhdistetään merkkijonoksi. Tämäkin ohjelma liitettiin edellä kuvatulla tavalla osaksi pääohjelmaa.



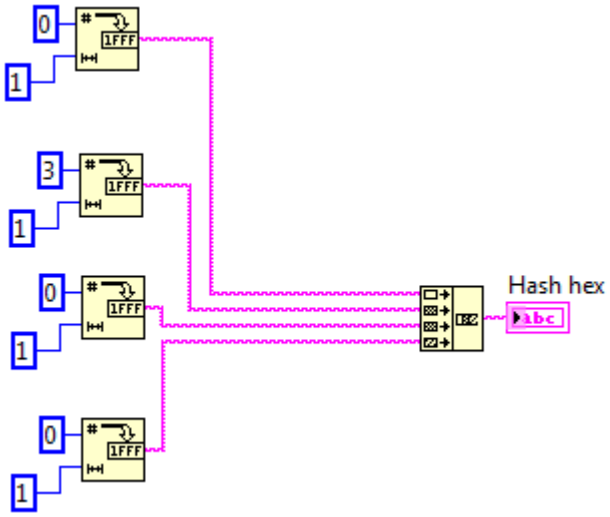
Kuva 20. Komento-aliohjelma

Seuraavaksi oli vuorossa ohjelman tekeminen vastaukselle. Komentoa lähetettäessä vastaus-kohtaan tulee 0 ja vastaukseen tulee 1. Tämä tehtiin samalla tavalla kuten aikaisemmatkin. Ohjelma on turhan monimutkaisesti toteutettu, sillä komentoa lähetettäessä riittäisi vain nollan sijoittaminen viestiin. Ainakin ohjelman rakenne on jonkin verran selkeämpi, kun se koostuu aliohjelmista eikä komentoa tehdä vain lähettämällä yksittäisiä numeroita pääohjelmassa sinne ja tänne. Kuvassa 21 on vastaus-aliohjelma. Jälleen kerran aliohjelma liitettiin pääohjelmaan ja ulostulo yhdistettiin merkkijonojen yhdistämistyökalulla muuhun merkkijonoon.



Kuva 21. Vastaus-aliohjelma

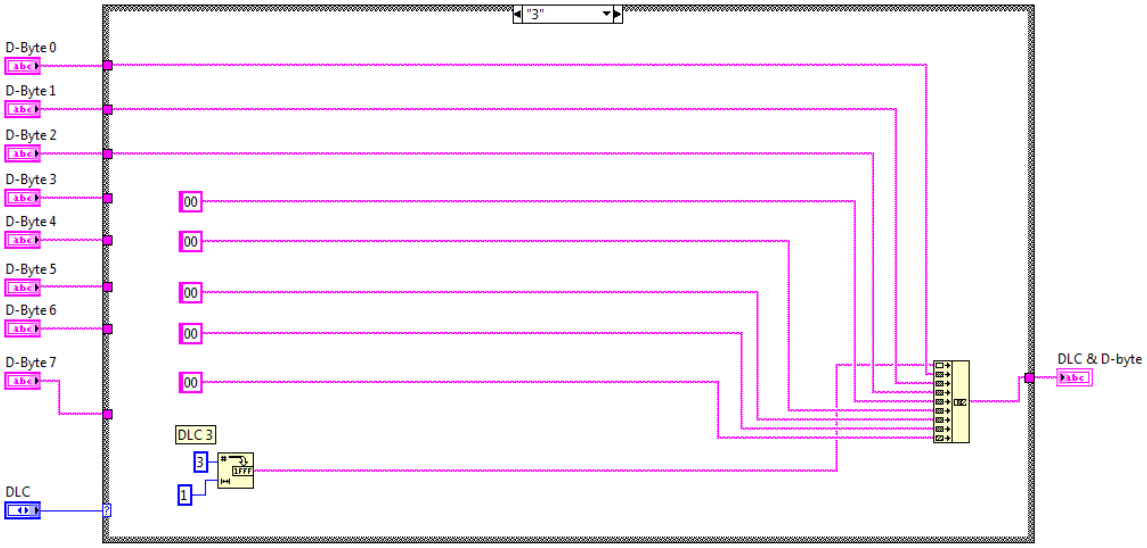
Seuraavaksi merkkijonoon liitetään Hash, jolle tehtiin oma aliohjelmansa. Kuvassa 22 on Hash. Siinä luku 0300 lähetetään heksadesimaalimuodossa eteenpäin.



Kuva 22. Hash-komennon aliohjelman palikkakaavio

Seuraavaksi tehtiin viestin datatavujen määrän ilmoittava DLC-aliohjelma. Kuvassa 23 on DLC-aliohjelman palikkakaavio. DLC-aliohjelmaan tuodaan kaikki datatavut ja DLC:n lukuarvo. Kuvassa 23 on DLC- aliohjelma, kun lukuarvo on 3. Ohjelma

lähettää DLC-lukuarvon 3 ja kolme datatavua, jotka on tarkoitus lähettää eteenpäin. Loput datatavut täytetään nollilla.

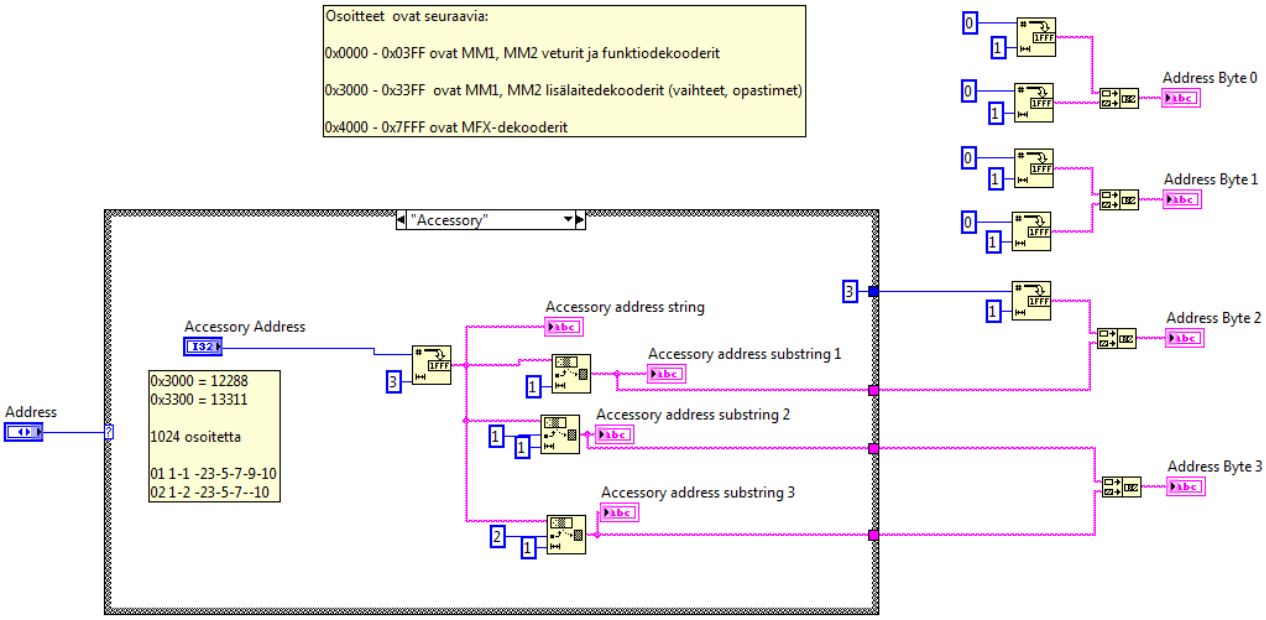


Kuva 23. DLC-aliohjelman palikkakaavio

Datatavuihin 0 - 3 sijoitetaan yleensä osoite. Järjestelmässä on myös mahdollisuus lähettää komento kaikille sen laitteille. Kun komento lähetetään kaikille, osoitteeksi laitetaan 00 00 00 00. Osoitteen muodostukselle tehtiin seuraavaksi aliohjelma. Märklinin järjestelmässä osoitteet ovat järjestetty seuraavalla tavalla:

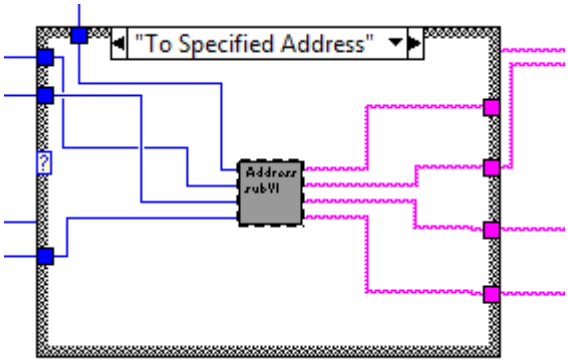
Osoitteet välillä 0x0000 - 0x03FF ovat MM1 ja MM2 veturien dekooderit sekä funktiodekooderit. Osoitteet välillä 0x3000 - 0x33FF ovat lisälaitteet (vaihteet, opastimet ym.). MFX-dekooderien osoitteet ovat välillä 0x4000 - 0x7FFF /2/. Osoite-aliohjelmassa on case-rakenne, jolla valitaan muodostettavan osoitteen tyyppi (veturi, lisälaite tai MFX). Datatavuihin 0 ja 1 sijoitetaan pelkkiä nollia. Esimerkiksi vaihteen 1 osoite on 00 00 30 01 ja veturin 03 osoite on 00 00 00 03.

Kuvassa 24 on osoitteen muodostuksen aliohjelma.



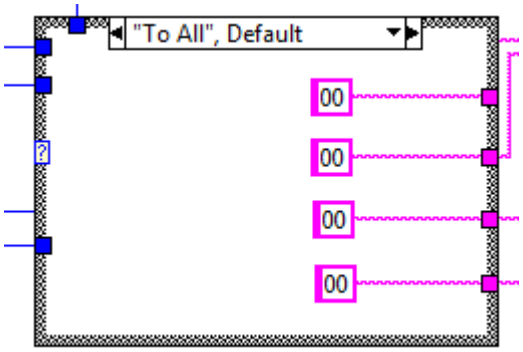
Kuva 24. Osoitteen muodostus

Osoite-aliohjelma sijoitettiin pääohjelmaan case-rakenteen sisään. Case-rakenteella voidaan valita, meneekö komento tiettyyn osoitteeseen tai kaikille. Kuvassa 25 on case-rakenne siitä, kun lähetetään komento tiettyyn osoitteeseen.



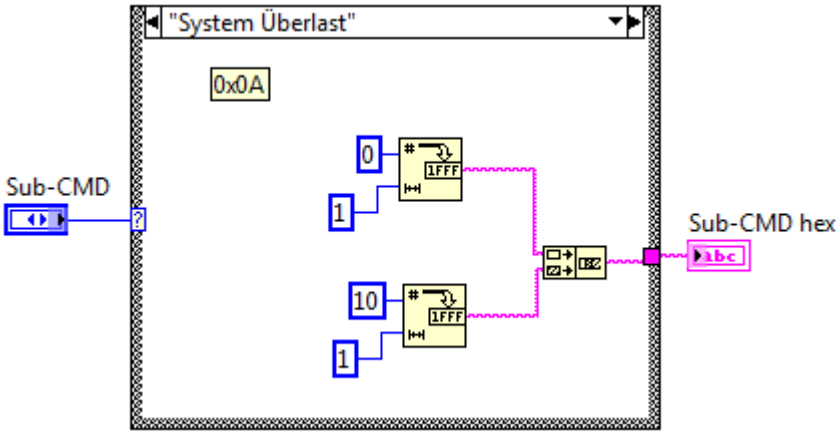
Kuva 25. Kun komento lähetetään tiettyyn osoitteeseen, case-rakenne käännetään tähän asentoon.

Kuvassa 26 lähetetään komento kaikille. Osoitetavuihin sijoitetaan nollia.



Kuva 26. Case-rakenne näyttää tältä, kun komento lähetetään kaikille laitteille.

Komennoissa ”System Stopp”, ”System Go”, “System Halt”, “Lok Nothalt”, “Lok Zyklus Beenden”, ”System MFX Neuanmeldezähler setzen”, ”System Überlast” ja ”System Status” datatavu 4:ään sijoitetaan sub-komento (Sub-CMD) /2/. Sub-komentoa varten tehtiin myös oma aliohjelmansa. Kuvassa 27 on sub-komento -aliohjelman palikkakaavio.



Kuva 27. Sub-komento

Loppujen komentojen jäljelle jäävät datatavut käsitellään tapauskohtaisesti myöhemmin.

2.4 Suunniteltava ohjelma

Yhteys ei edelleenkään toimi, joten varsinainen ohjelma on suunniteltava ilman kokeilumahdollisuutta. Pääohjelman palikkakaavioon on tarkoituksena sijoittaa kaksi while-silmukkaa. Toinen lähettää komentoja ja toinen vastaanottaa CS2 tulevia paketteja.

Komentoja lähettävään silmukkaan sijoitetaan UDP-write palikoita toinen toisensa perään. Jokainen UDP-write palikka lähettää yhden komennon. Komennoista tullaan lähettämään ainakin järjestelmäkomennot (Stopp, Go, Halt), veturien komennot kahdelle veturille ja lisälaitteiden komentoja kahdelle vaihteelle.

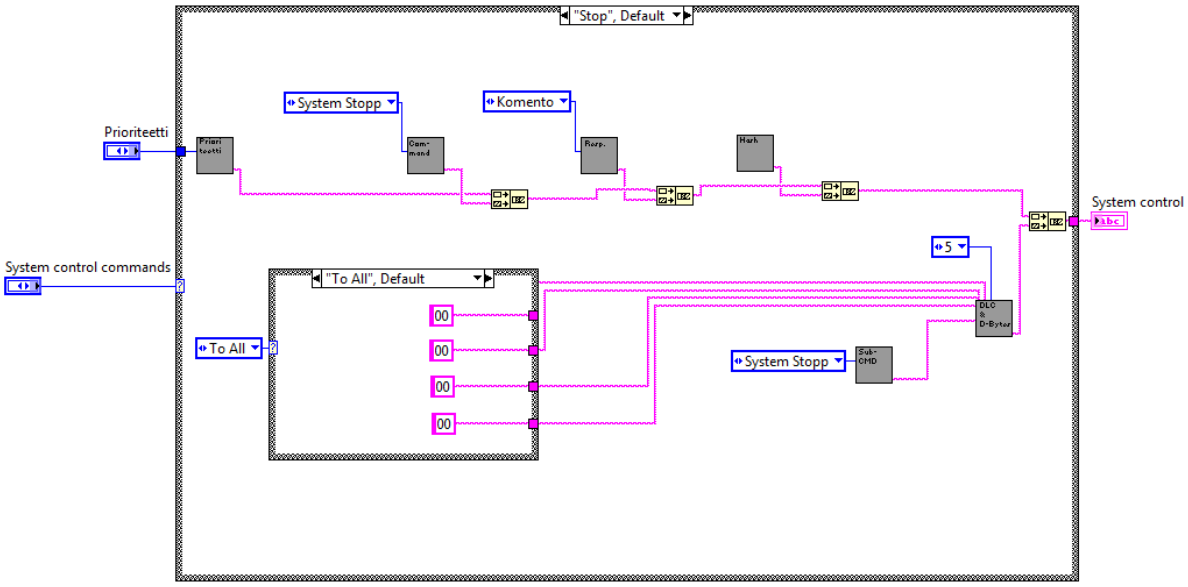
Vastaanottavaan silmukkaan sijoitetaan yksi UDP-read-palikka, jonka perään tehdään paketin purku. Purettujen osien perusteella paketit ohjataan oikeille indikaattoreille esim. veturin nopeus nopeusmittareille ja järjestelmäkomentojen vastaukset LED-indikaattoreille.

2.5 Komentojen lähetyssilmukka

2.5.1 Järjestelmän komennot

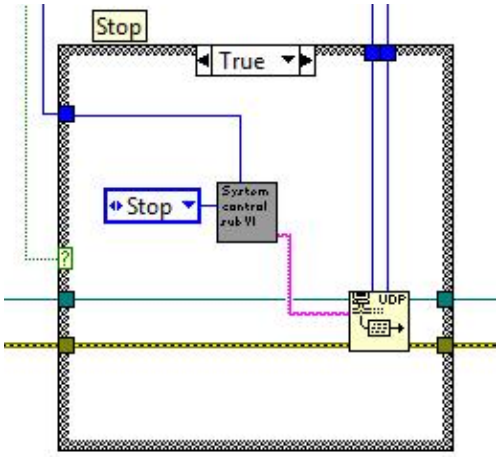
Komentojen lähetyssilmukan ensimmäisessä aliohjelmassa lähetetään järjestelmäkomennot Stop, Go ja Halt. Stop-komennolla pysäytetään koko järjestelmä katkaisemalla virta kokonaan pois. Go-komennolla virta kytketään takaisin päälle. Halt-komennon tarkka toimintaperiaate täytyy selvittää kokeilemalla. Halt-komennolla laite ilmeisesti pysäytetään jollain tavalla.

Ensimmäisenä luotiin aliohjelma, jossa muodostetaan nämä komennot. Kuvassa 28 on tämän aliohjelman palikkakaavio. Siinä on keskellä iso kolmiosainen case-rakenne, josta voidaan valita Stop, Go tai Halt. Komennon muodostusosa on kopioitu aikaisemmin tehdystä työkaluohjelmasta.



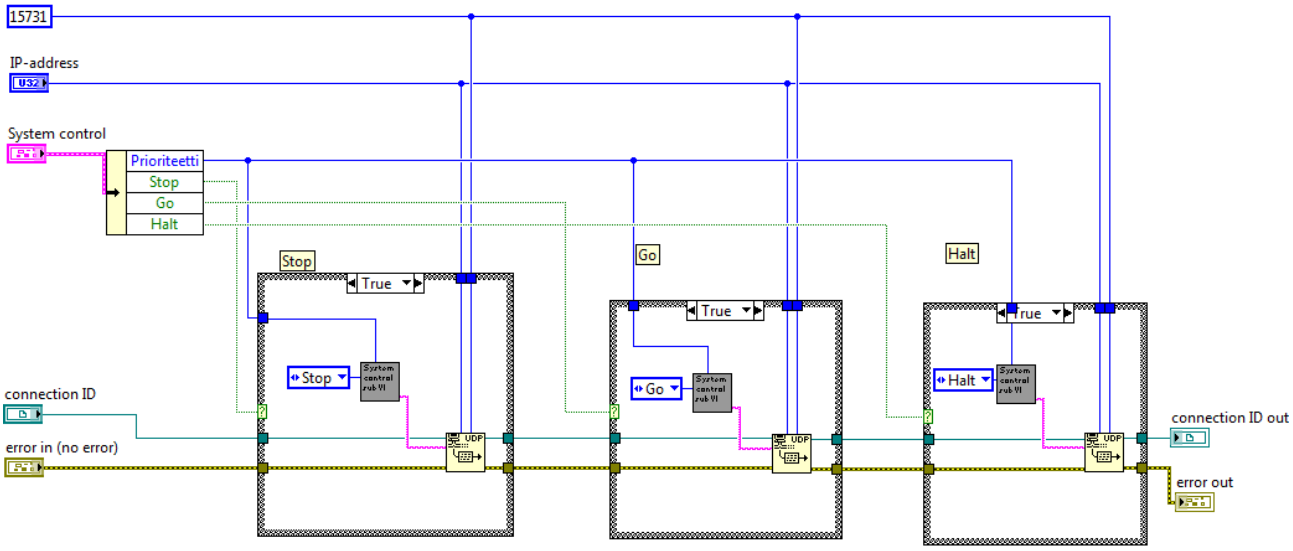
Kuva 28. Järjestelmäkomennon muodostava aliohjelma

Komennon muodostava aliohjelma liitettiin järjestelmäkomennon lähettävään aliohjelmaan, joka on kuvassa 29. Komennon muodostavan osan ulostulo on kytketty UDP-write-palikan sisään menoon. Nämä molemmat ovat case-rakenteen sisällä. Case-rakenteella on tilat tosi ja epätosi, joita ohjataan painikkeella. Kun painike on painettuna, tila on tosi. Tällöin molemmat palikat ovat näkyvillä ja komento lähetetään. Kun painiketta ei ole painettu, tila on epätosi. Tällöin case-rakenteen läpi menee suorat johdot eikä mitään komentoa lähetetä.



Kuva 29. Järjestelmäkomennon lähetyskytkentä

Kaikki kolme komentoa liitettiin lähetysohjelmaan, jonka palikkakaavio on kuvassa 30. Alhaalla on UDP-yhteyden johdotus ja sen virheilmoitusjohdotus. Ylhäältä menee portin numero 15731 ja IP-osoite UDP-write-palikoille. Painikkeet ja prioriteetin ohjain ovat liitetty samaan klusteriin, eli jonkinlaiseen ryppääseen (Cluster). Aliohjelmaa liitettäessä on helpompi johdottaa yksi klusteriohjain, kuin kaikkia painikkeita erikseen.



Kuva 30. Järjestelmäkomentojen lähetys

2.5.2 Veturin komennot

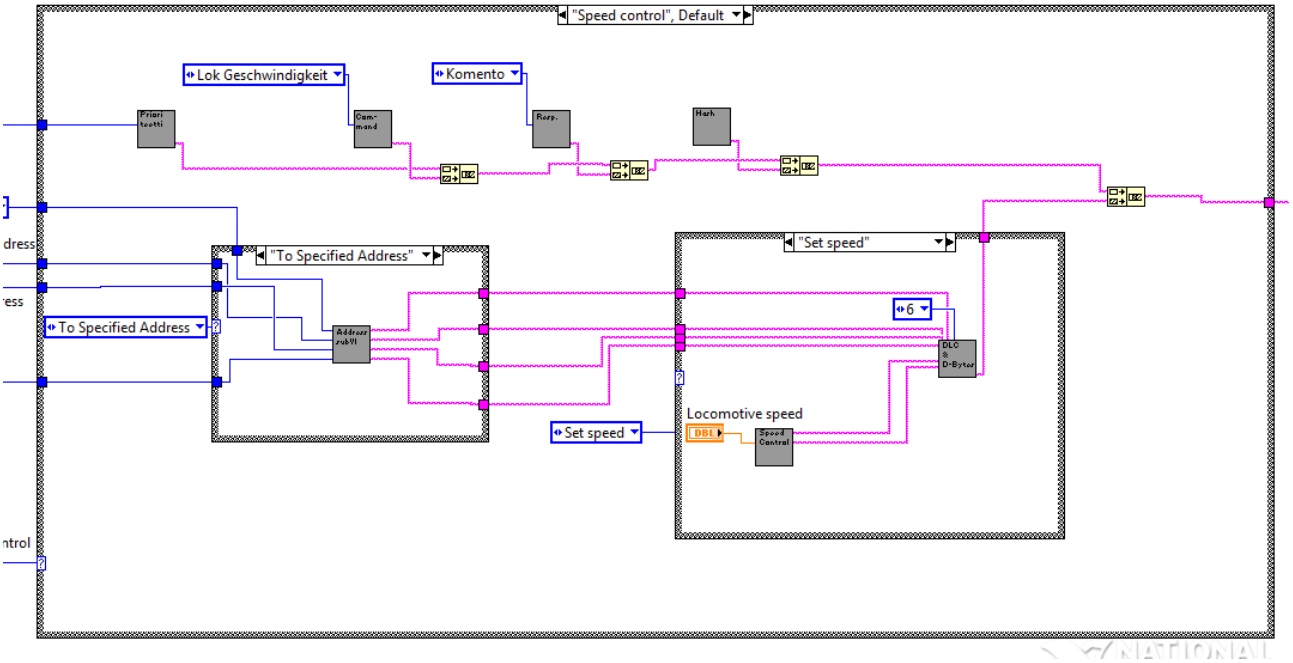
Veturin komentojen lähetystä varten tehtiin myös oma aliohjelmansa, jolla lähetetään seuraavat komennot:

Järjestelmäkomentojen kaltaiset Stop-, Go-, Halt- ja Nothalt-komentojen lähetys hoidetaan samalla tavalla painonappikytkennällä, kuin järjestelmäkomennotkin. Ainoa ero on, että nämä lähetetään tiettyyn osoitteeseen.

Veturin nopeuden, suunnan ja funktioiden kysymisen ja asettamisen komennot lähetetään silmukan jokaisella suorituskerralla. Kysymisessä viimeiset datatavut jätetään tyhjiksi. Asettaessa haluttu arvo asetetaan viimeisiin datatavuihin /2/.

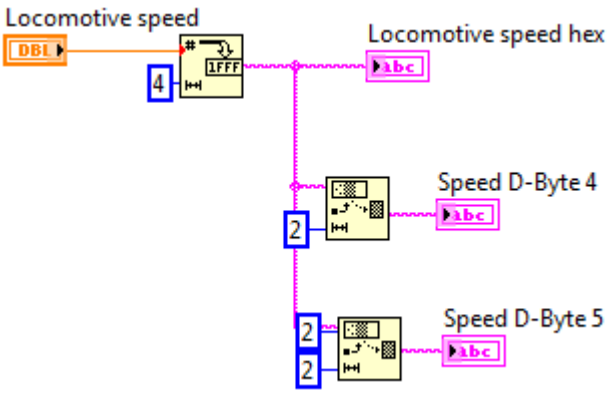
Veturin dekooderin päivityssyklin lopettaminen suoritetaan silloin, kun veturin ohjaus vaihdetaan toiseen veturiin. Edellisen veturin dekooderi lopettaa päivityksen ja jatkaa kulkuaan viimeisimpänä lähetetyillä arvoilla /2/.

Veturin komentojen muodostamiseen tehtiin oma aliohjelmansa, johon tehtiin case-rakenne, jossa muodostetaan kaikki edellä mainitut komennot. Aikaisemmin tehty työkaluohjelma helpotti tässäkin kohdassa. Siitä oli helppo kopioida tarvittavat komennot. Kuvassa 31 on veturin ohjauskomentojen muodostusaliohjelman palikkakaavio. Kuvassa näkyy nopeuden asettamisen muodostaminen.



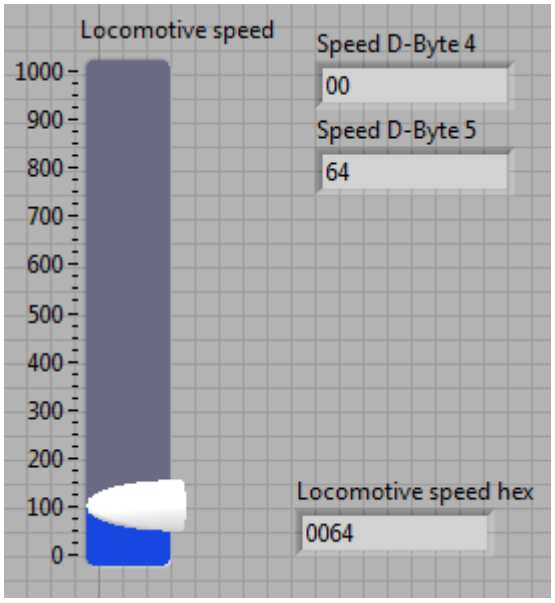
Kuva 31. Veturin komennon muodostaminen

Nopeuden ja suunnan asettamista varten tehtiin vielä aliohjelmat, joilla pystytään valitsemaan haluttu nopeus ja suunta ja sen jälkeen lähettää se datatavuihin 4 ja 5. Kuvassa 32 muodostetaan ensin 4 heksadesimaaliluvun pituinen merkkijono ja sen jälkeen se pilkotaan datatavuihin 4 ja 5.



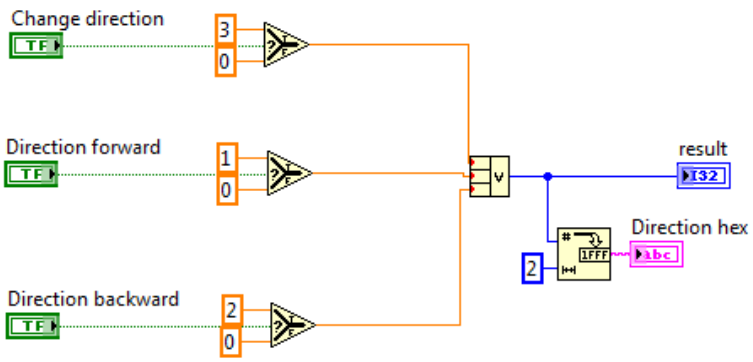
Kuva 32. Veturin nopeuden asettamisen palikkakaavio

Kuvassa 33 on nopeuden asettamisen etupaneeli. Vasemmalla on liukukytkin, jolla asetetaan nopeuden suuruus. Oikealla on indikaattoreita, joista näkyy nopeuden suuruus heksadesimaalilukuina.



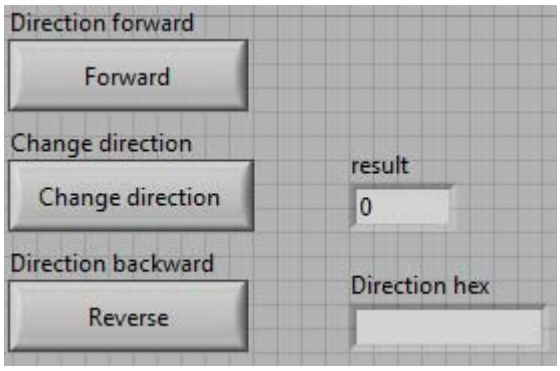
Kuva 33. Nopeuden asettamisen etupaneeli

Veturin suunnan asettamisen palikkakaavio on kuvassa 34. Kun halutaan, että suunta pysyy samana, lähetetään 0. Kun halutaan muuttaa suuntaa, lähetetään 3. Suunta eteen on luku 1 ja taakse luku 2 /2/. Painiketta painamalla saadaan suunta muutettua. Kun mitään painiketta ei paineta, lähetetään 0 eli suuntaa ei muuteta.



Kuva 34. Suunnan asettamisen palikkakaavio

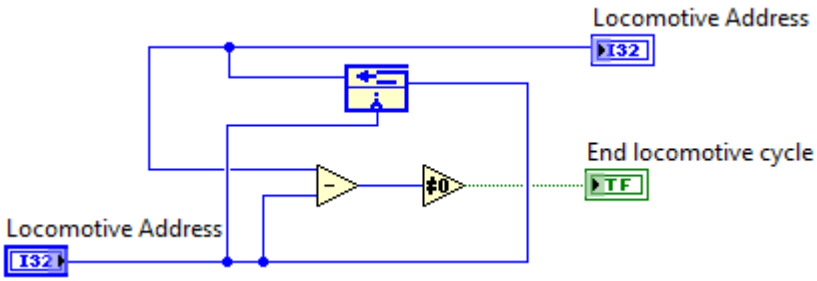
Kuvassa 35 on suunnan asettamisen etupaneeli. Painiketta painamalla saadaan veturi kulkemaan haluttuun suuntaan.



Kuva 35. Suunnan asettamisen etupaneeli

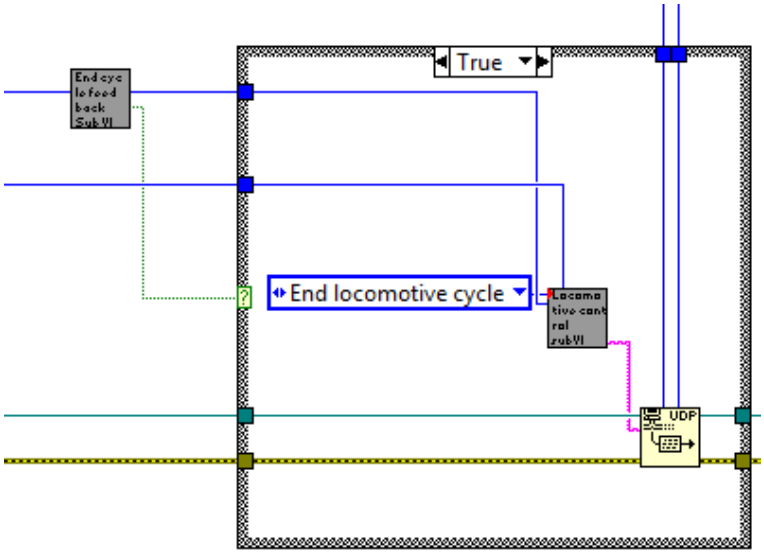
Veturin funktioita ohjataan numerolla, joka on funktiosta riippuen 0…31. Funktion numero sijoitetaan datatavuun 4. Funktion lisäksi lähetetään myös lukuarvo (Wert) datatavussa 5. Lukuarvo on välillä 0…31 /2/. Se miten funktiota ohjataan sen numerolla ja lukuarvolla, on selvitettävä kokeilemalla. Sen takia tässä vaiheessa funktion ohjaamiseen on kytketty kaksi numeron syöttökontrolleria. Kun toiminta on selvitetty, voidaan sinne tehdä halutunlainen painikekytkentä.

Veturin dekooderin päivityssyklin lopettamiseen tehtiin myös apuohjelma. Se vertaa, onko veturin osoite muuttunut ohjelman edellisestä suorituskerrasta. Jos se on muuttunut, lähettää se päivityssyklin pysäyttämiskomennon edelliseen osoitteeseen. Kuvassa 36 on päivityssyklikomennon apuohjelma.



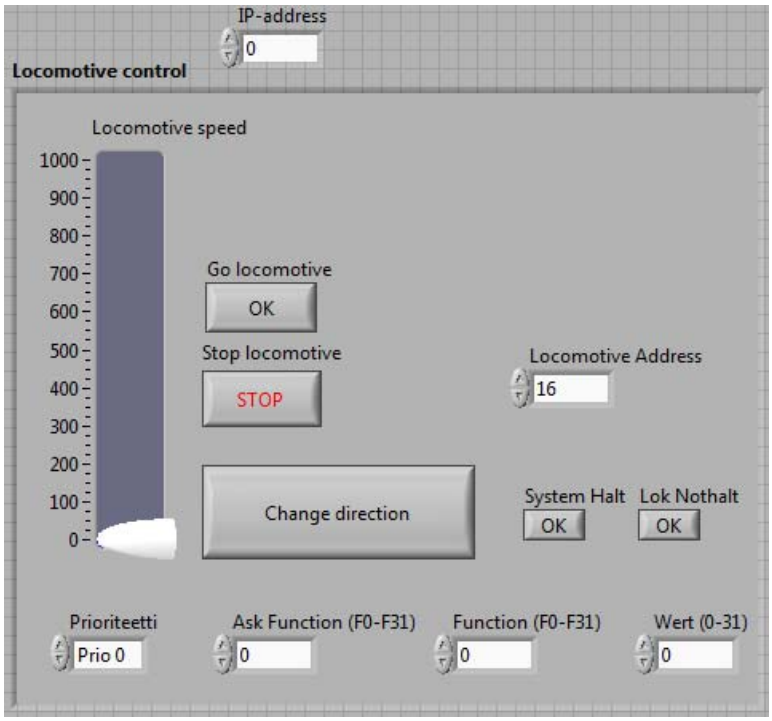
Kuva 36. Dekooderin päivityssyklin lopettamisapuohjelma

Tämä apuohjelma sijoitettiin kuvassa 37 olevalla tavalla komentojen lähetysohjelman alkuun. Kun se on tunnistanut osoitteen muuttuneen, lähettää se tosi-arvon case-rakenteelle. Case-rakenne lähettää syklin lopettamiskomennon.



Kuva 37. Veturin päivityssyklin lopettamisen lähetyskytkentä

Veturin ohjauksen etupaneeli on kuvassa 38. Kaikkien painikkeiden ja kytkimien tarpeellisuus ja toiminta täytyy selvittää kokeilemalla, sitten kun yhteys toimii.



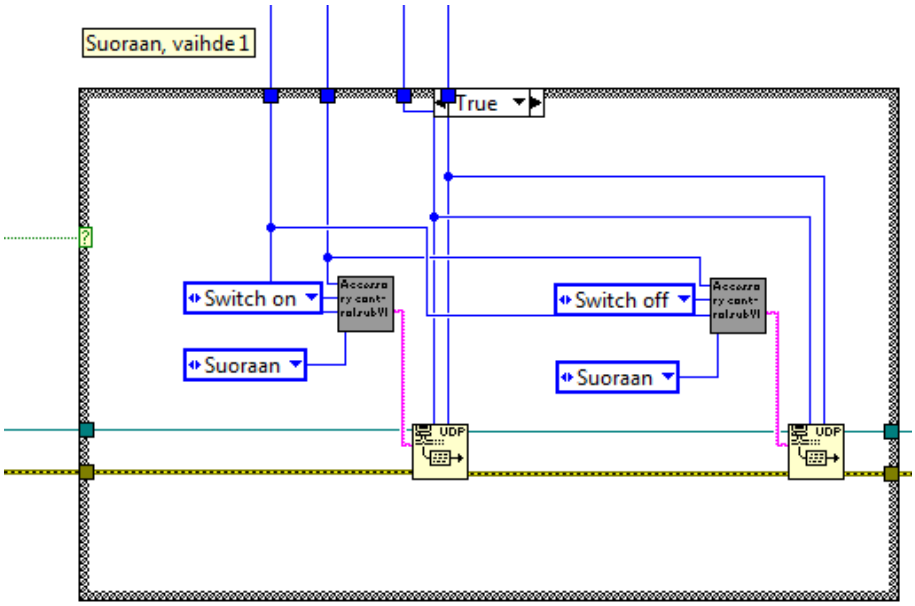
Kuva 38. Veturin ohjauksen etupaneeli

2.5.3 Vaihteiden komennot

Vaihteille lähetetään järjestelmäkomennot (Stop, Go, Halt) samalla tavalla, kuten vetureillekin.

Vaihteiden ohjauskomennot ovat 00 = oikealle, 01 = suoraan, 10 = vasemmalle. Lisäksi vaihdetta ohjattaessa on kytkettävä virta ensin päälle ja sen jälkeen pois /2/. Lopullinen toimintaperiaate selviää kokeilemalla varovasti, ettei mikään pala. Jos vaihdemoottori jää päälle vaihteen vaihtamisen jälkeen, se voi palaa.

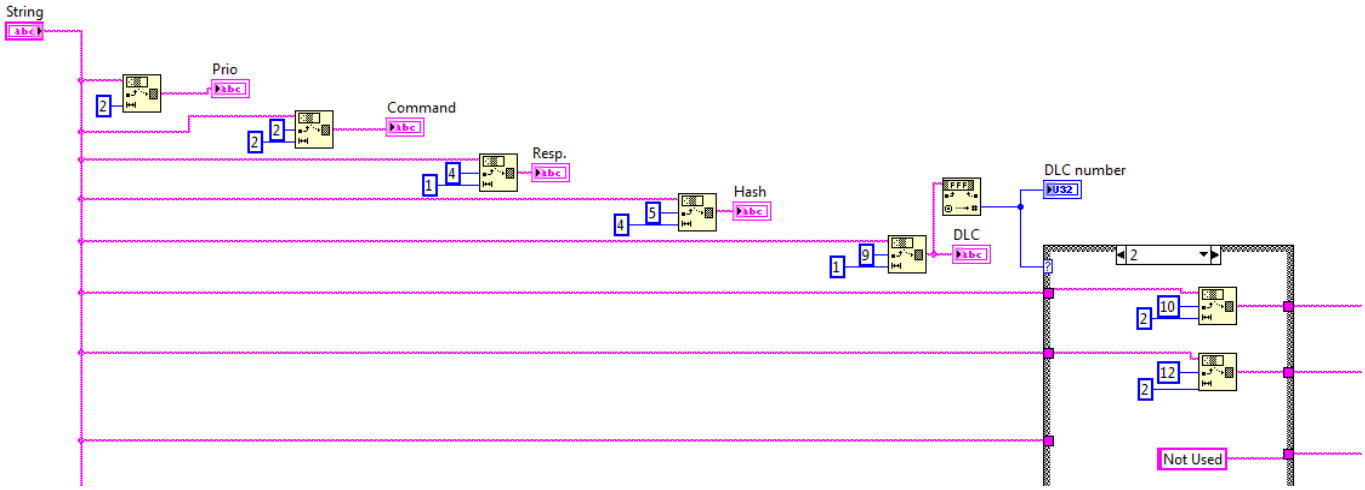
Kuvassa 39 on kytkentä ohjauskomennon lähettämisestä, jolla vaihde käännetään suoraan. Ensin kytketään virta päälle ja annetaan suoraan-komento. Lopuksi kytketään virta pois päältä.



Kuva 39. Vaihteen ohjauskomennon lähettäminen

2.6 Vastaanottosilmukka

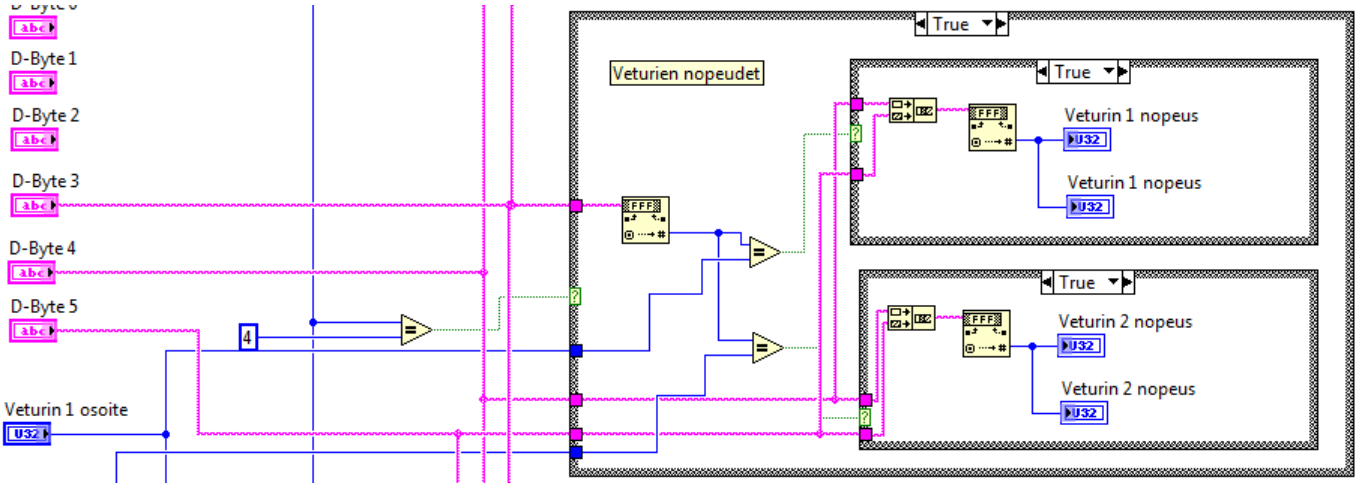
Esimerkkien vastaanotto-ohjelmasta karsittiin aika paljon ylimääräistä pois. Paikalleen jätettiin yhteyden muodostamis- ja sulkemisosio ja UDP-read-palikka. UDP-read -palikan ulostuloon tehtiin aliohjelma, joka hajottaa vastaanotetun paketin osiin. Kuvassa 40 on paketin hajotusohjelman palikkakaavio. Siinä käytetään merkkijonon hajotuspalikoita. Merkkijono katkaistaan aina tietyn luvun kohdalta ja palaset erotaan toisistaan.



Kuva 40. Paketin hajotusaliohjelman palikkakaavio

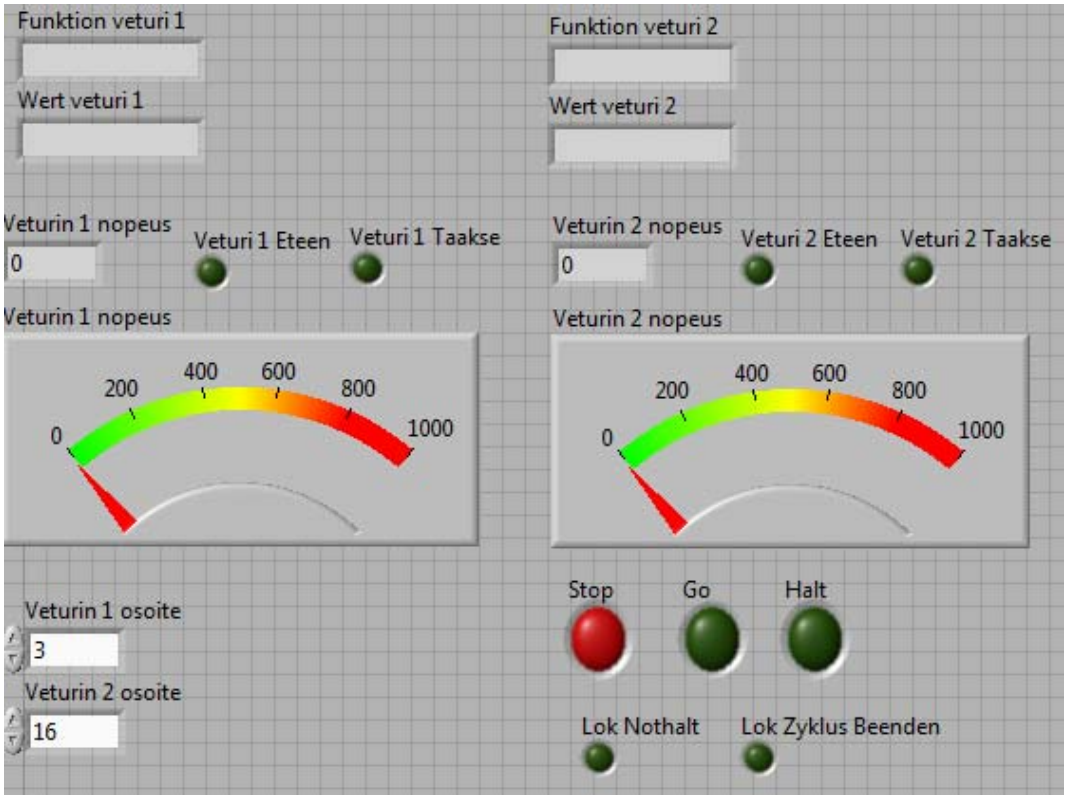
Vastaanottosilmukkaan tehtiin myös toinen aliohjelma, joka tulkitsee pilkotun vastauksen. Se järjestää komento-osan perusteella vastauksen oikeaan osaan ohjelmaa. Esimerkiksi nopeuden komento 0x04 järjestää viestin nopeuden käsittelyosioon. Tämän jälkeen viestin osoiteosiota verrataan valittuun osoitteeseen. Sen perusteella nopeus menee joko veturin 1 tai 2 mittarille. Nopeuden suuruustieto määrittää nopeusmittarin asennon. Kaikkien tavujen tietoja ei oteta huomioon.

Kuvassa 41 on vastauksen tulkintaohjelman palikkakaavio.



Kuva 41. Vastauksen tulkinta

Kuvassa 42 on vastauksen tulkinnan etupaneeli. Siinä on indikaattorit veturin funktioille, suunnalle, nopeudelle ja järjestelmäkomennoille. Tässä vaiheessa tehty on vain esimerkki siitä, mitä kaikkea on mahdollista tehdä. Yhteys pitäisi saada toimimaan, että kaikkea voitaisiin kokeilla. Esimerkiksi nopeusmittarin skaalausta varten täytyisi kokeilla, kuinka suuri arvo riittää.

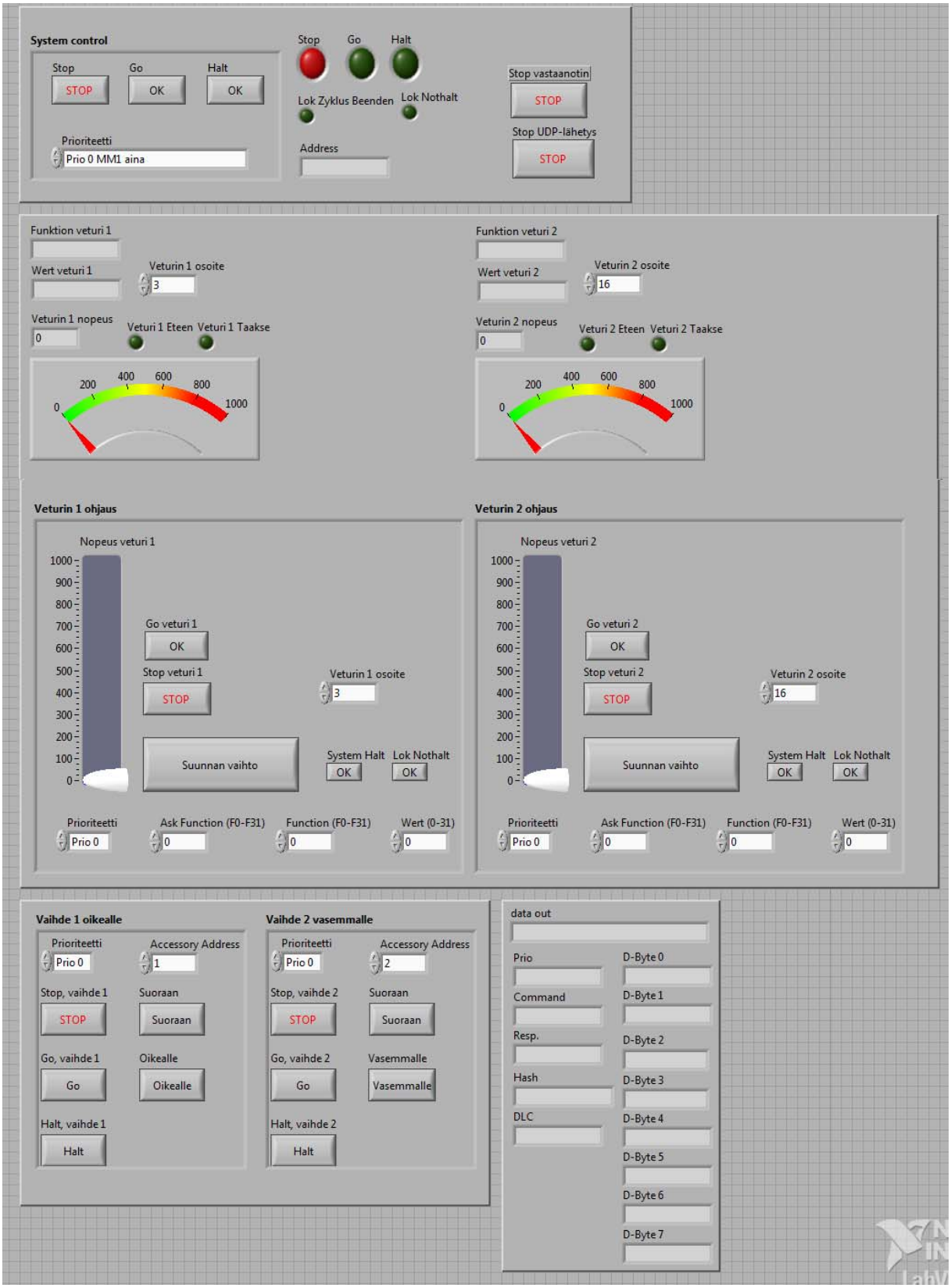


Kuva 42. Vastauksen tulkinnan etupaneeli

3 Yhteenveto ja johtopäätökset

Työn tarkoituksena oli siis tehdä digitaaliseen pienoisrautatiehen ohjaus Labviewillä. Lopputuloksena saatiin ohjelma, jolla voidaan lähettää ja vastaanottaa paketteja tietokoneen ja CS2:n välillä. Yhteys ei kuitenkaan jostain syystä toimi tai jos oikein tarkkoja ollaan, niin yhteys toimii, mutta CS2 reagoi komentoihin. Vian syy voi olla mikä tahansa, mutta erittäin todennäköisesti se on kiinni siitä ohjelmistopäivityksestä, jota ei CS2:een saatu ladattua. Tätä ohjelmaa voidaan kuitenkin myöhemmin käyttää CS2:n kanssa ja parantaa sitä, kun toimintojen toimintaperiaate selviää. Tämän ohjelman kokemusten perusteella voidaan myöhemmin tehdä jopa parempi ohjelma.

Kuvassa 43 on tämän ohjelman lopullinen etupaneeli. Turhia painikkeita voi myöhemmin karsia pois. Kun yhteys alkaa toimia, funktioiden ja muiden painikkeiden toimintaperiaate täytyy selvittää ja muokata paremmiksi.



Kuva 43. Pääohjelman etupaneeli

Tämä työ päätettiin toteuttaa CS2-laitteen kanssa. Jollakin muulla tavalla lopulliseen tavoitteeseen eli toimivaan ohjaukseen olisi voitu päästä. Jokin toinen toteutustapa, esimerkiksi National Instrumentsin valmistamaa CAN-väylän ohjaukseen tarkoitettua laitetta käyttämällä, olisi voinut olla parempi vaihtoehto. Laitteella olisi ehkä voinut lähettää komennot suoraan CAN-väylään ilman Märklinin ohjauslaitetta. CS2-laitteen kanssa päästiin siis tähän asti. CS2-laite on vielä kuitenkin varsin uusi tuote. Siinä on valmiudet monenlaiselle liitännälle ja toiminnolle, mutta näyttää siltä, että vielä siihen ei ole saatavilla ohjelmistoa, jolla niitä kaikkia pystyttäisiin hyödyntämään. Tulevaisuudessa luultavasti CS2-laitteeseen tulee ohjelmaversio, jolla niitä pystytään käyttämään ja ehkä myös tätä ohjelmaa voidaan myöhemmin käyttää.

Tämän työn päätarkoituksena oli ehkä enemmänkin toimia harjoituksena Labviewin käyttöön. Sitä voidaan käyttää erilaisten prosessien ja järjestelmien ohjaamiseen ja valvontaan. Tässä tapauksessa sillä oli tarkoitus ohjata digitaalista pienoisrautatietä. Tämä työ teknisistä ongelmista huolimatta antaa jonkinlaisen kuvan siitä, mitä kaikkea Labviewillä on mahdollista tehdä. Labview on siis ohjelmointia graafisessa muodossa. Se on ehkä jonkin verran havainnollisempaa, kuin perinteisellä rivikoodi-ohjelmoinnilla suoritettu ohjelmointi. Kun ohjelmassa on silmukka, näytölle piirretään silmukka. Perinteisessä ohjelmoinnissa kirjoitettaisiin silmukkaa vastaavaa ohjelmointikoodia. Samaten ohjelman suoritusjärjestystä on ehkä helpompi seurata graafisessa muodossa, kuin kirjoitetussa muodossa. Labviewissä on toiminto, jolla ohjelman saa suoritettua hidastettuna ja siinä saa myös ohjelman välivaiheiden suoritustulokset näkyviin. Tällä tavalla voidaan tarkistaa, meneekö jokin toimitus oikein. On varmasti myös tapauksia, joissa kirjoitettu koodikieli on helpompi toteuttaa kuin graafinen kuvio. Ohjelmointia vähemmän tuntevalle graafinen muoto on ehkä selkeämpää.

Tätä työtä aloittaessani en tiennyt oikeastaan mitään ohjelmoinnista. Olen sähkövoimatekniikan opiskelija enkä tietotekniikan. En osannut myöskään tehdä mitään Labviewillä. TAMKissa järjestettiin Labview-kurssi, mutta sen perusteella ei vielä näin laajaa ohjelmaa olisi pystynyt tekemään. Märklinin ohjekirja (Kommunikationsprotokoll) oli saksan kielellä. Minulle se ei ollut ongelma. Labview-kurssin opettajasta sen sijaan ei ollut apua tähän työhön, koska hän ei osannut saksaa. Materiaali olisi kuulemma pitänyt olla englanniksi. Ilman ohjausta jouduin tämän työn siis tekemään. Aluksi tämä työ vaikutti täysin mahdottomalta. 5 kk:n työharjoittelujakso Belgiassa auttoi erittäin paljon tämän työn tekemistä. Työskentelin

siellä Labviewin kanssa muissa projekteissa. Siellä hankitun kokemuksen avulla sain tämän työn tehtyä. Labview-ohjelmointi on ”tekemällä oppii”-tyyppistä. Mitä enemmän tekee, sitä paremmin osaa.

Lähteet

/1/ Labview-esimerkit, Labview 2009 Student Edition

/2/ Märklin. Kommunikationsprotokoll. PDF-tiedosto. Saksa 2008. http://www.maerklin.de/de/produkte/tools_downloads.html, 13.2.2008

/3/ Sähköpostitiedustelu Märkliniltä. 7.5.2009

/4/ Märklin Magazin, Sonderausgabe digital-special, Märklin Magazin -lehden Märklin-digitaalia käsittelevä erikoisnumero