Sissejuhatus: LabView programmeerimine

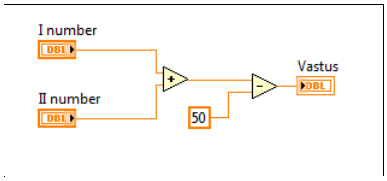
Kokkuvõte peamistest võtetest LabView programmide koostamisel.

Õpikeskkond:	e-Õppe Arenduskeskuse Moodle
Kursus:	Mehhatroonikasüsteemide komponendid
Koosta raamat:	Sissejuhatus: LabView programmeerimine
Printja:	ahti põlder
Kuupäev:	Tuesday, 29 November 2011, 06:35 AM

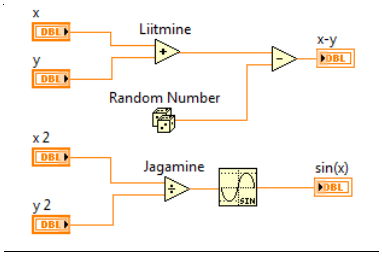
Sisukord

1 Andmete voog LabViews

Enamus tekstil põhinevaid programmeerimiskeeli kasutavad nn käsuvoo mudelit (ingl. k. *Control Flow Model*, vt Wikipediast), *LabView* seevastu kasutab programmi täitmisel andmevoo mudelit (ingl. k. *Data Flow Model*, vt Wikipediast). Lihtsustatult, käsuvoo mudeli järgi, määrab programmi täitmise järjekorra programmi elementide järjestus. *LabView*s kasutatava andmevoo mudel töötab nii, et blokkdiagrammil asuv element (nt funktsioon) käivitub alles siis, kui selle **sisenditesse** on **kõik nõutud andmed**, pärast seda tekitab element oma väljundile tulemuse. Edasi liigub tulemus mööda andmevoogu juba järgmise elemendi sisendisse. See liikumine määrab LabView programmi täitmise järjekorra (vt video näidet).



Alloleval pildil on näha LabView programmeerimise üks eripärasid. Antud näites pole võimalik kindlaks määrata milline funktsioon, liitmine, jagamine või *Random Number*, esimesena täidetakse. Liitmine ja jagamine saavad oma andmed koheselt pärast programmi käivitumist ja *Random Number* funktsioonil puuduvad sisendid, mis annab ka sellele õiguse koheselt käivituda. Olukorras, kus üks koodi osa sõltub teise koodi osa tulemustest tuleb tähelepanelikult jälgida, mis järjestuses need täidetakse.

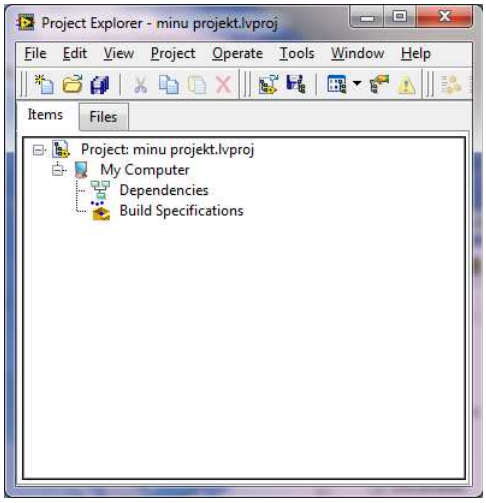


2 Programmeerimiseks ettevalmistumine

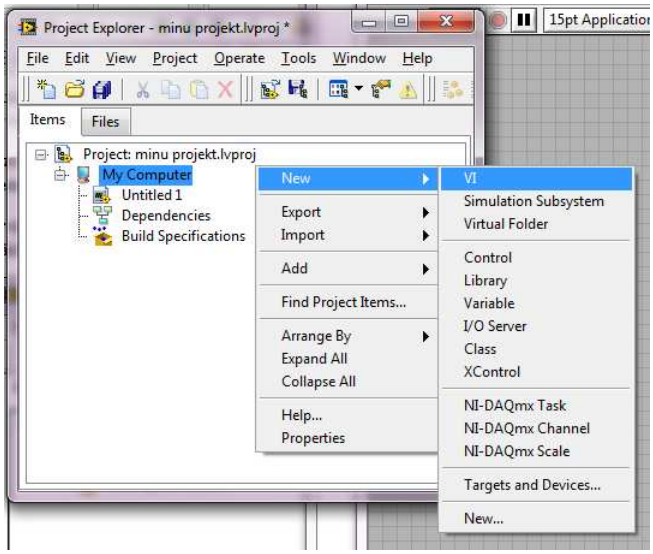
Kuigi *LabView*s on võimalik luua iseseisvaid VI-sid, on soovituslik oma programmi koostamiseks kasutada **projekte**. Projekti eeliseks on see, et kõik VI-d mis te loote antud ülesande raames on ühes kohas ja säästab teid failide arvutist otsimisest. Projekti on võimalik luua valides *Getting Started* aknast *Empty Project*.

LabView projektid salvestuvad .lvproj laiendiga. Projekti kuuluvad kasutaja loodud VI-d, viited failidele mida kasutatakse (ingl. k. *Dependencies*), lõpprakenduste infor ja konfiguratsiooni infot. Projektile on võimalik lisada ka eemalseisvaid süsteeme nagu võrgukaamerad, kontrollerid ja taskuarvuti moodulid.

Projekti aknal on kaks sakki, *Items* ja *Files*. *Items* saki all on näidatud projekti ülesseade struktuuriga *Project: nimi.lvproj >> My Computer*. Kõik VI-d mis asuvad *My Computer* jaotises käivituvad kasutaja arvutis. *Files* saki all näidatakse kus kohas asuvad erinevad projekti failid arvuti kõvakettal.

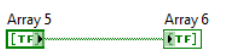
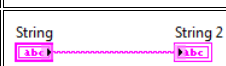
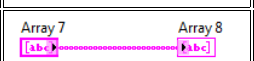
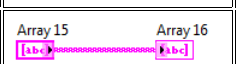


Uue VI tekitamiseks tuleb valida parema hiire nupuga ***My Computer >> New >> VI*** (vt video näidet). Siis avanevad uue VI esipaneel ja blokkdiagramm ning VI viide lisatakse *My Computer* jaotisesse.



3 Andmetüübid

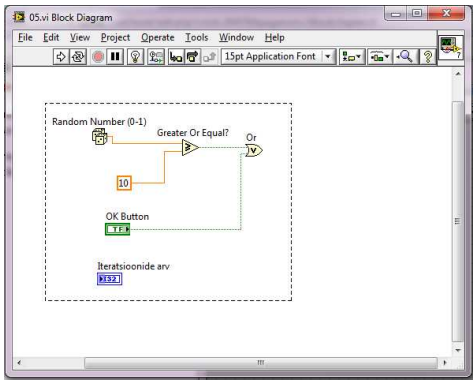
LabViews liiguvad andmed mööda juhtmeid. Igal juhtmel võib olla üks andmete allikas, kuid juhet saab ühendada mitmete VI-de ja funktsioonidega. Juhtmetel võib olla vastavalt nende **andmete tüübile** mitmeid erinevaid stiile, värve ja paksusi. Järgnevas tabelis on toodud peamised andmetüübid blokkdiagrammilt nähtuna:

Juhtme tüüp	Skalaar (wiki)	1D jada (wiki)	2D jada (wiki)	Märkused
Reaalarv				Värvus oranž.
Täisarv				Värvus sinine
Kahendväärtus				Värvus roheline
String				Värvus roosa

4 Tsüklite kasutamine

Tüklid võimaldavada programmeerijal korrata oma programmiosa tööd mitmeid kordi (tsükliliselt) ja määrata tingimusi, millal see tsükkel kordusest väljub. Igat korda, mil tükli sees olev programm läbitakse nimetatakse iteratsiooniks. *LabView*s on programmeerijal võimalik kasutada *for*- ja *while*-tsüklit.

While*-tsükkel** täidab programmi osa nii kaua, kuni peatumise tingimus on täidetud. *While*-tsükli asetamiseks tuleb valida blokkdiagrammi funktsioonide palletilt ***Programming* >> *Structures* >> *While Loop. Järgmiseks tuleb hiire kursor asetada blokkdiarammile nii, et korratav kood jääks tekkiva nelinurga sisse (vt ka video näidet). Hiire nupust lahti lastes piiratakse valitud ala *while*-tsükliga.

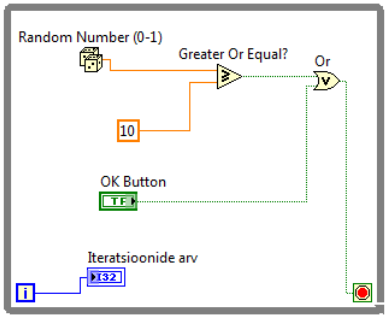


While-tsükkel kordab selle sees olevat koodi niikaua kuni tsükli tingimuse terminal (ikoon tsükli all paremas nurgas) saab teatud kahendväärtuse tüüpi väärtuse. Tingimuse terminale on kaks tükki:

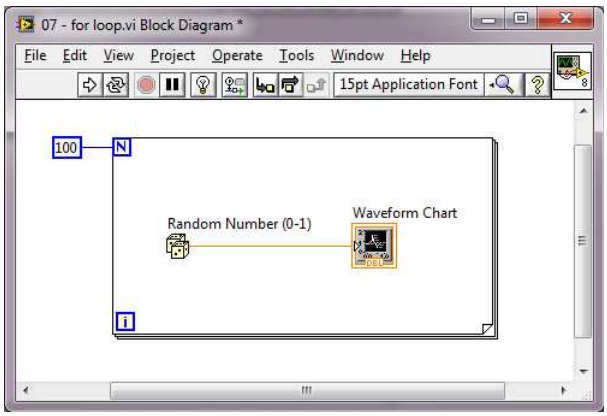
Kui tingimuse terminal on ***Continue if True*** režiimis siis tsüklis olev programm käivitub uuesti nii kaua kuni terminali jõuab väär (*False*) signaal. Režiimi saab muuta klõpsates parema nupuga teminali äärel ja valides *Stop if True* režiimi.

Kui tingimuse terminal on ***Stop if True*** režiimis (vaikimisi režiim) siis tsüklis olev programm käivitub uuesti nii kaua kuni terminali jõuab tõene (*True*) signaal. Režiimi saab muuta klõpsates parema nupuga teminali äärel ja valides *Continue if True* režiimi.

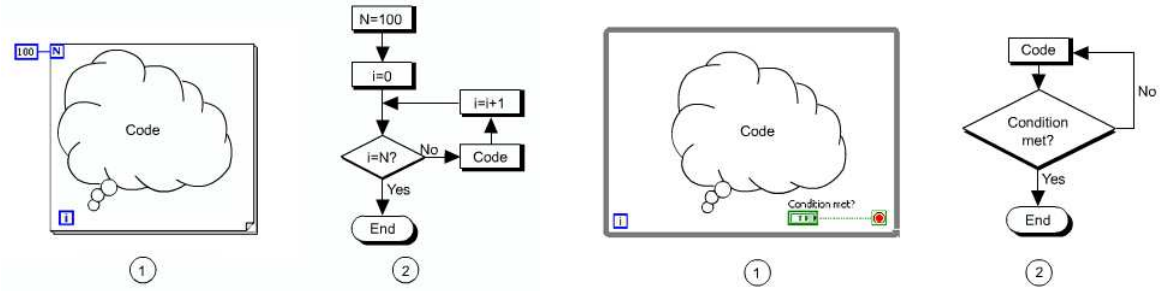
Näitena *while*-tsükli kasutamisest on allpool toodud pildil tsükli sisse piiratud blokkdiagramm. Antud juhul kordub tsükkel nii kaua kuni *Random Number* funktsiooni väljund on suurem või võrdne konstandiga 10 **või** vajutatakse esipaneelil OK nuppu, mille korral tekib nupu teminalile tõene (*True*) väärtus. Vastasel juhul annavad funktsioon *Greater or Equal?* ja OK nupu terminal väära (*False*) tulemuse ja seejärel ka *Or* funktsioon väära (*False*) tulemuse ning programm kordub otsast peale.



For*-tsükkel** on mõeldud blokkskeemi tsükliliseks kordamiseks teatud arv korda. *For*-tsükkel asub funktsioonide palletil ***Programming* >> *Stuctures* >> *For Loop. *For*-tsükli asetamine blokkskeemile käib sarnaselt *while*-tsükli asetamisega. Alloleval pildil genereerib *for*-tsükkel iga iteratsiooni ajal suvalise arvu ja kuvab selle graafikul. Tsükkel käivitub 100 korda (graafikule tekib 100 andmepunkti). *For*-tsüklil on *while*-tsükliga sarnane iteratsioonide loenduri terminal "**i**" mis loendab iteratsioone nullist üles. Teise terminali "**N**" abil määratakse mitu korda *for*-tsükkel käivitub.

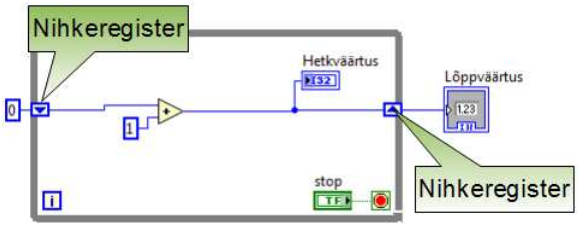


Võrdluseks *for*- ja *while*-tsükli toimimise erinevustest on allpool toodud tsüklite blokkskeemid (1) koos vastavate vooskeemidega (2).



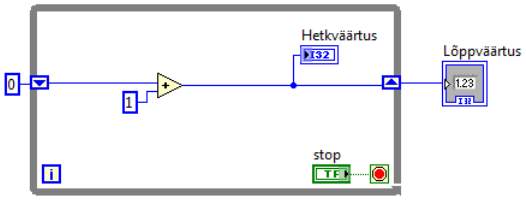
5 Nihkeregistrite kasutamine

Tihti tuleb tsükliliste programmide puhul **saata informatsiooni ühest iteratsioonist teise**. Selleks on LabView-s võimalik luua **nihkeregistreid** (ingl.k. *Shift register*). Nihkeregistrid tekivad tsükli piirjoonele kui vastastikune paar noolekesega terminale. Parempoolsel teminalil on nooleke üles suunatud ja on mõeldud andmete salvestamiseks iteratsiooni lõpus. Iteratsiooni lõppedes saadetakse salvestatud info vasakul pool asuvale terminalile millel näitab nooleke alla suunas. Järgneva iteratsiooni alguses on sealt terminalilt võimalik saada eelmises iteratsioonis salvestatud informatsiooni.

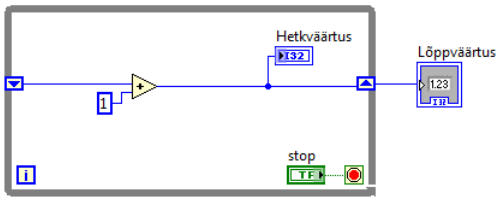


Nihkeregistri lisamiseks tuleb klõpsata parema hiire pupuga tsükli äärel ja valida ***Add Shift Register*** (vt ka video näidet). Olemasolevate tunnelite puhul tuleb klõpsata parema hiire nupuga tunneli terminalil ja valida ***Replace With Shift Register*** (vt ka video näidet). Nihkeregister võtab automaatselt selle andme tüübi, mis selle terminali sisendisse ühendatakse. Andmetüübid, mis nihkeregistri terminalidele ühendatakse, **peavad olema sama tüüpi**.

Nihkeregistrtl on võimalik kasutada kahel viisil - algväärtustatult ja mitte algväärtustatult. **Algväärtustatud** nihkeregistrile tuleb anda vasakpoolsesse terminali mingi soovitud väärtus (vt allpool olevat pilti), mis kindlustab selle, et iga kord kui programm käivitub on nihkeregistri väärtuseks kindel arv (antud juhul null).



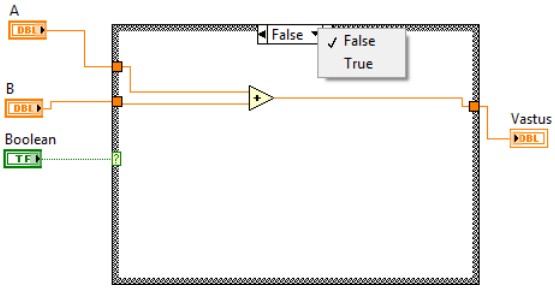
Algväärtustamata nihkeregistri puhul vasakul asuvale terminalile väärtust ei anta (vt allpool olevat pilti). Sellisel juhul säilitab nihkeregister oma viimase väärtuse. Kui VI uuesti käivitada siis antakse algväärtuseks eelmise korra viimane registri väärtus. See väärtus säilib nii kaua kuni VI suletakse või lõplik programm töötab.



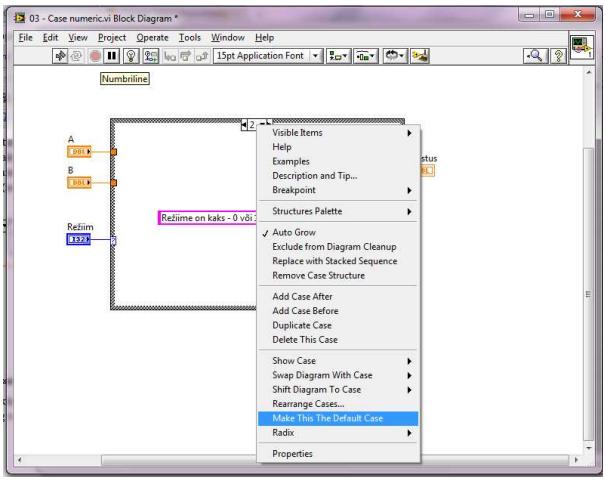
Nihkeregistrit saab lisada nii *while*- kui ka *for*-tsüklile.

6 Programmiliste otsuste tegemine

Programmiliste otsuste tegemiseks kasutatakse LabViews **juhustruktuuri** (ingl.k. *Case Structure*). Juhustruktuuri kasutatakse, kui VI-s on vaja otsustada teatud tingimuste eeldusel kuidas VI töö kulgeb. Juhustruktuuril võib olla kaks või enam alamdiagrammi, mida valitakse vastavalt sisendi tingimustele. Korraga on näha ainult üks diagramm ja vastavalt sisendile käivitub ka ainult üks olemasolevatest diagrammidest. Allpool oleval pildil on näha juhustruktuur, millel on kaks alamdiagrammi. Kui Boolean nupult tuleb rohelise küsimärgiga terminali sisendisse *False* siis lülitub sisse vastav alamdiagramm, kui *True,* siis sellele vastav alamdiagramm (vt videot). Juhustruktuuri sisend võib olla ükskõik mis tüüpi skalaar (number, kahendväärtus, sõna).

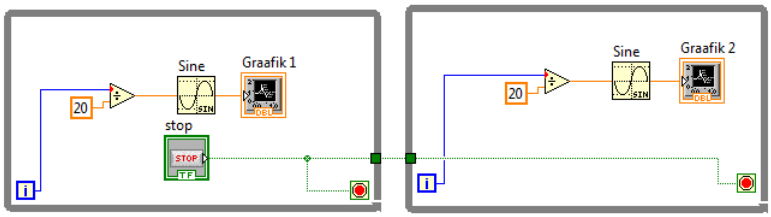


Kui kasutaja ei soovi defineerida juhustruktuuri alamdiagrammi igale võimalikule sisendile (nt kui sisendiks on arv), siis tuleb määrata üks **vaikejuht** (ingl.k. *Default Case*). Selleks tuleb navigeerida vaikejuhtumi alamdiagrammile ja vajutada parema hiire nupuga päisele ning valida *Make This The Default Case* (vt videot). Vaikejuht aktiveeritakse juhul, kui struktuuri sisendisse tuleb väärtus, mis pole juhuna defineeritud.

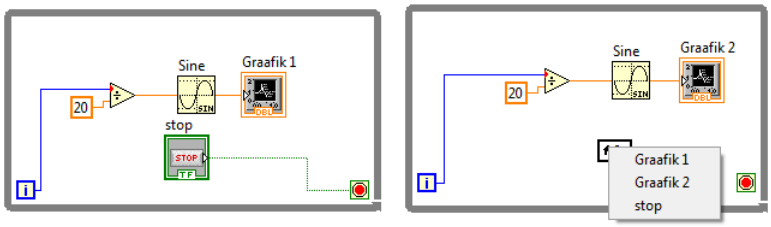


7 Mitme tsükli vaheline kommunikatsioon

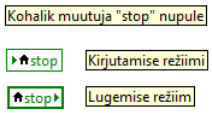
Nagu varem mainitud, määrab *LabView*-s programmi käigu pigem andmete voog kui funktsioonide ja VI-de järjestus. Seega, on võimalik luua programme, milles toimuvad operatsioonid samaaegselt. Näiteks saab kasutada kahte (või enamat) paralleelselt töötavat tsüklit korraga. Tänu *LabView* andmevoogude põhimõttele ei ole võimalik kahe tsükli vahel juhtmete kaudu infot saata. Juhtmed tekitavad kahe tsükli vahel andmete sõltuvuse. Näiteks allpool oleval diagrammil on näha programm, kus parempoolne tsükkel ei käivitu enne, kui vasakpoolses tsüklis on vajutatud Stop nuppu ja andmed on jõudnud läbi tunneli parempoolse tsüklini. Et neid kahte tsüklit korrga tööle saada ja peatada tuleb eemaldada nende vaheline juhe ja kasutada **kohalikku muutujat** (ingl.k. *Local Variable*).



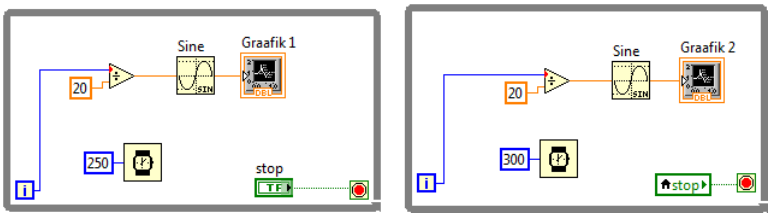
Kohalikud muutujad salvestavad andmeid, mis on esitatud esipaneelil olevatel nuppudel ja näidikutel. Kohaliku muutuja tekitamiseks peab vajutama parema hiire nupuga esipaneeli objektil (nupul või näidikul) või blokkdiagrammil olevale terminalile ja valima ***Create >> Local Variable***. Blokkdiagrammile tekib selle objelti nimeline muutuja ikoon. Teine viis kohaliku muutuja loomiseks on valida funktsioonide palletilt ***Programming >>Structures >> Local Variable*** ja asetada see blokkdiagrammile. Kui hiirega klõpsata muutuja keskkohal avaneb menüü, kus on loetletud esipaneelil asuvad objektid (vt videot).



Vaikimisi on kohalik muutuja kirjutamise režiimis (terminali ühenduskoht on vasakul pool). See tähendab, et VI käivitamisel loetakse muutuja sisend ja kirjutatakse väärtus esipaneelili olevale objektile. Režiimi muutmiseks tuleb valida parema hiire nupuga ***Change to Read*** (vt videot).



Kohaliku muutuja kasutamise näitena on allpool toodud järgmine VI. Vi-s on kaks erineva kiirusega käivituvat tsüklit (üks 250 ms tagant, teine 300 ms tagant) ja kahe tsükli vahel saadetakse "stop" nupu informatsioon. Parempoolses tsüklis olev "stop" kohalik muutuja loeb iga iteratsiooni ajal esipaneelil oleva "stop" nupu väärtust. Kui esipaneelil nuppu vajutada annab "stop" nupu terminal väärtuse vasakpoolsesse tsüklisse ja see jääb seisma. Samal ajal loeb kohalik muutuja samuti nupu väärtuse ja seiskab ka parempoolse tsükli. Seega töötavad tsüklid teine-teisest sõltumatult (paralleelselt) ja peatatkse samas sama-aegselt esipaneelil olevat nuppu vajutades.



8 Allikad

Kasutatud allikad

1. **National Instruments**, *Introduction to LabView - Basics 1*, NI 2010
2. **National Instruments**, *Introduction to LabView - Basics 2*, NI 2010

Lisalugemist

1. **National Instruments**, *Introduction to LabView web site (viit), NI 2011*