# Teststand plugin struktur til opsamling af data fra CompactDAQ chassis

Kasper Eis Kristiansen

Kongens Lyngby, 2011
IMM-B.Eng-2010-82

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk
IMM-PHD: ISSN 0909-3192

# Abstract

Dette projekt forsøger at lave et standard produkt til at lette opsætningen af nye test og målesystemer der benytter sig af National Instruments produkterne Labview, Testand og DAQmx moduler.

Der er udviklet i samarbejde med CIM Industrial Systems A/S – Hørsholm.

Fokus har på at udvikle et framework hvortil man kan udvikle simple plugins der kan foretage målinger fra forskellige slags moduler.
Der skal etableres en kommunikations vej imellem framework og plugins. Hovedfokus var i først omgang at få data indsamling på plads, altså indelse af data fra plugins til frameworket. Eventuelt output ville forsøges hvis der var tid.

Under implementering blev den oprindelige med et kø system opgivet, og i stedet benyttes der referencer til de enkelt plugins. Disse referencer kan benyttes til at manipulere med pluginsene. I den nuværende version er det kun muligt at læse data fra plugins.´Ved videre udvikling vil samme metode dog også sagtens kunne benyttes til at sende data fra frameworket til plugins.

Det endelige produkt er et fungerende framework der kan modtage data fra modul specifikke plugins. Der er dog rig mulighed for videre udvikling.

# Forord

Dette projekt er lavet som eksamens projekt for at afslutte uddannelsen Diplom ingeniør, IT. Projektet er lavet ved Instituttet for Matematik og Modellering tilhørende DTU.

Den fokusere på at højne brugervenligheden i test og målesystemer, hvor Labview og Teststand softwaren fra National Instruments benyttes.

Lyngby, November 2011

Kasper Eis Kristiansen

# Indledning

Dette projekt beskæftiger sig med at konsulenters tid i udviklings firmaer er dyrebar. Det er ofte en ligeså stor udgift som hardware, og i nogle tilfælde endda højere. Standard løsninger, der ikke kræver ekstra udviklings tid, er derfor et godt værktøj at have. Den udgift der vil være ved at udvikle produktet første gang kan nemlig derefter spredes ud på flere projekter. Man kan sælge samme produkt til flere kunder, til en langt lavere pris pr. kunde, end hvis man skal udvikle et nyt system hver gang en kunde henvender sig. At kunne tilbyde et billigere produkt ville betyde en bedre konkurrence dygtighed i branchen.

Ideen til dette projekt kom efter at have snakket med min erhvervsvejleder under mit praktik ophold i CIM Industrial Systems A/S. CIM har et tæt samarbejde med National Instruments og benytter mange af deres produkter i deres løsninger. Det er et af de førende firmaer indenfor Labview programmering i Norden, med 5 ud af de 27 certificerede Labview Architects i Europa. Labview ”Architect” er den højeste certificering man kan opnå fra National Instruments program. Under arkitekterne findes graden ”Developer”, hvilket mange CIM medarbejdere har opnået. Det laveste certificerings trin hedder ”Associated Developer”.

I forbindelse med mit praktik ophold skulle jeg lære det grafiske programmeringssprog Labview. Labview bliver udviklet af det amerikanske firma National Instruments. Deres hovedsæde er lokaliseret i Houston, Texas, USA, men samarbejdet med CIM foregår for det meste via det danske kontor. Det danske kontor ligger lige ved siden af CIMs Hørsholm afdelingen, hvilket hjælper i det daglige samarbejde.

National Instruments primære forretnings område er udviklingen og salget af måleinstrumenter til virksomheder verden over. De tilbyder et stort udvalg af moduler til at løse forskellige opgaver. Man kan sammensætte et test/måle opstilling med de moduler som National Instruments tilbyder, og programmere disse med Labview

Labview er lavet med henblik på muliggøre indsamling af data fra National Instruments måleinstrumenter, og lave efterfølgende databehandling. Der findes derfor mange værktøjer i Labview til at gøre disse opgaver nemmere. Det kræver dog stadig en del programmering at få projekter på plads, og en del erfaring at lave et program i Labview. Måden man programmere på er i nogle henseender noget anderledes end tekst programmering, men de mere basiske programmerings principper gælder stadig. Der benyttes stadig de gængse løkker som For og While, men IF løkken er erstattet af udtrykket Cases. Det er et programmerings sprog hvor man i nogle henseender nærmest skal tænke programmet som et elektrisk kredsløb. Det passer godt sammen med det som Labview for det meste bruges til, at programmere moduler i et elektrisk kredsløb.

Udover Labview tilbyder National Instruments også et sekvens program, Teststand. Dette bruges ofte hos CIM til at gennemgå et test forløb, og tjekke om måle resultaterne holdes sig inden for tolerancerne. En anden nyttig feature er at det automatisk kan generere en test rapport baseret på de resultater der opnås undervejs. Det kræver dog en ret stor ekspertise at programmere i, og er tidskrævende.

CIM bruger ofte National Instruments DAQmx moduler fra CIM, da de tilbyder stor fleksibiletet, og det er i den billigere ende af National Instruments måleinstrumenter. DAQmx moduler tilsluttes

på mange forskellige mådes. Det mest gængse er USB, eller Ethernet, alt efter om modulerne er langt fra computeren, eller er lige ved siden af.

Efter at have fået lært lidt Labview programmering, og fået et indblik i hvad Teststand var så foreslog min erhvervsvejleder at min eksamens projekt kunne være et produkt der gjorde det nemmere at foretage målinger via National Instruments DAQmx C-serie moduler. At det var begrænset til C-serien var for ikke at gøre opgaven for uoverskuelig. Det lød som et spændende projekt hvor jeg kunne få lov til at lære mere omkring Labview programmering, samt få et bedre indblik i hvad Teststand kunne. Det lød som kompetencer der ville kunne gavne mig senere hen. Udover det kunne jeg få lov til at programmere noget hvor man skulle tænke over at det skulle være så nemt for forbrugeren at benytte, altså kunne mine kurser inden for GUI design komme til gavn. Det var jo perfekt, og jeg sagde jatak til projektet, og vi begyndte at formulere det lidt mere konkret. Dette var for at kunne skrive en introduktion til projektet så der kunne findes en vejleder på DTU.

DTU fandt mig en vejleder og jeg kontaktede ham omkring projektet. Han synes også det lød spændende, omend han ikke var helt klar på hvad Labview og Testand var. Han hjalp mig med at skrive projekt beskrivelsen om til en mere akademisk form, og da den var godkendt hos DTU gik jeg i gang med projektet. Resultatet kan du læse omkring på de følgende sider.

# Indholdsfortegnelse


Abstract....3
Forord....4
Indledning....5
Indholdsfortegnelse....7
Problem Analyse og Kravspecifikation....9
Design og Analyse....10
  Interaktion med programmet....10
    Use case....10
    Sekvens Diagram....10
  Data flow....11
  Framework....11
  Datalag....11
  State Diagrams....12
    Framework State diagram....12
    Plug-in State diagram....13
  Begrænsning....13
Implementering....14
  GUI....14
  Datalag....14
  Waveforms....14
  Framework....14
    Main.vi....15
    Getdata.vi....16
    Merge Channels.vi....16
    Merge Waveform Arrays.vi....16
  Plug-ins....17
    9211....17
    9215....17
  Brugervejledning....18
    1. Load Plugin....19
    2. Ryd Plugin listen....19
    3. Start måling....19
    4. Stop Måling....19
    5. Stop programmeret....19
    6. Skift imellem data visning....20
Test....21
  Praktisk test....21
  Unit test....23
Fremtidige overvejelser....24
  Teststand integration....24
  Output....24
  Andre måleinstrumenter end DAQmx....24
Konklusion....25

Bilag....26
Labview Kildekode....26
Main.vi....26
Merge Channels....27
Merge Waveform Arrays. vi....28
Get Data.vi....29
9211.vi....30
9211-Userinput.vi....31
9215.vi....32

Det tager for lang tid at udvikle et unikt system hver gang man skal lave et nyt test og målesystem hvor der skal indsamles data fra flere moduler. Derfor skal der udvikles et standard system der gør det muligt hurtigt at sammensætte softwaren der er nødvendig for at indsamle data, og præsentere den for brugeren. Det skal gerne være muligt, selv for folk med et begrænset kendskab til Labview, at betjene systemet.

Det tager for meget tid at skulle skrive Labview kode til hver enkelt step i en test sekvens når en test opstilling skal kodes i Teststand. Dette gør at selv simple projekter bliver ret dyre, da konsulenterne skal bruge meget tid på kodning. Selve opgaven som modulet skal udføre er dog ofte den samme, som i andre test opstillinger, så kodningen er ofte ens. Selve princippet er det samme, men de forskellige moduler skal kalibreres forskelligt.
Denne ”dobbelt”-kodning vil gerne undgås, og man ønsker en struktur hvori man kan skrive kode til et enkelt modul. Dette moduls kode kan så benyttes forskellige test-opstillinger. Når opstillingen er på plads skal testsekvensen kodes, og dette skal kunne gøres uden et stort forudgående kendskab til Labview kodning. Dette vil gøre det både muligt for andre end Labview programmøre at ændre i test-sekvensen, hvilket gerne skulle resultere i at der behøves færre konsulent timer på et projekt. Det vil også gøre det muligt at senere ændringer til test sekvensen kan foretages af kunden.

Funktionelle krav

- Skal kunne integreres i Teststand
- Skal muliggøre nem udvikling af ekstra plugins
- Skal kunne håndtere input af data fra et plugin
- Skal kunne fremstille data fra plugin på en overskuelig måde

Ikke-funktionelle krav

- Skal kunne håndtere outputs
- Automatisk identificere hvilke moduler der er tilknyttet og loade de tilhørende plugins

Løsningsmål

- Forkorte tiden det tager at sammensætte en måling fra flere forskellige moduler
- Gøre det muligt for brugere der ikke har Labview/Teststand erfaring at foretage forskellige målinger.

Programmet skal altså kunne gøre det nemmere at kommunikere med forskellige moduler (over DAQmx), primært med fokus på input. Når dataene er modtaget skal de præsenteres for brugeren på en simpel måde. Det skal tage så lidt tid som muligt at kode et nyt plugin, så de fleste funktioner skal være i frameworket.

## Interaktion med programmet

### Use case

Et normalt scenarie for brugen af frameworket kunne være følgende:

1. Bruger starter frameworket
2. Bruger tilknytter et plugin til frameworket (kan gentages)
3. Bruger er færdig med at tilknytte plugins
4. Bruger starter frameworket
5. Målinger foretages
6. Resultater vises til brugeren

Dette scenarie skal gerne kunne opfyldes både af en erfaren Labview programmør, og en novice.

### Sekvens Diagram

Interaktionen imellem de forskellige moduler i systemet også brugeren kunne foregå på følgende måde.

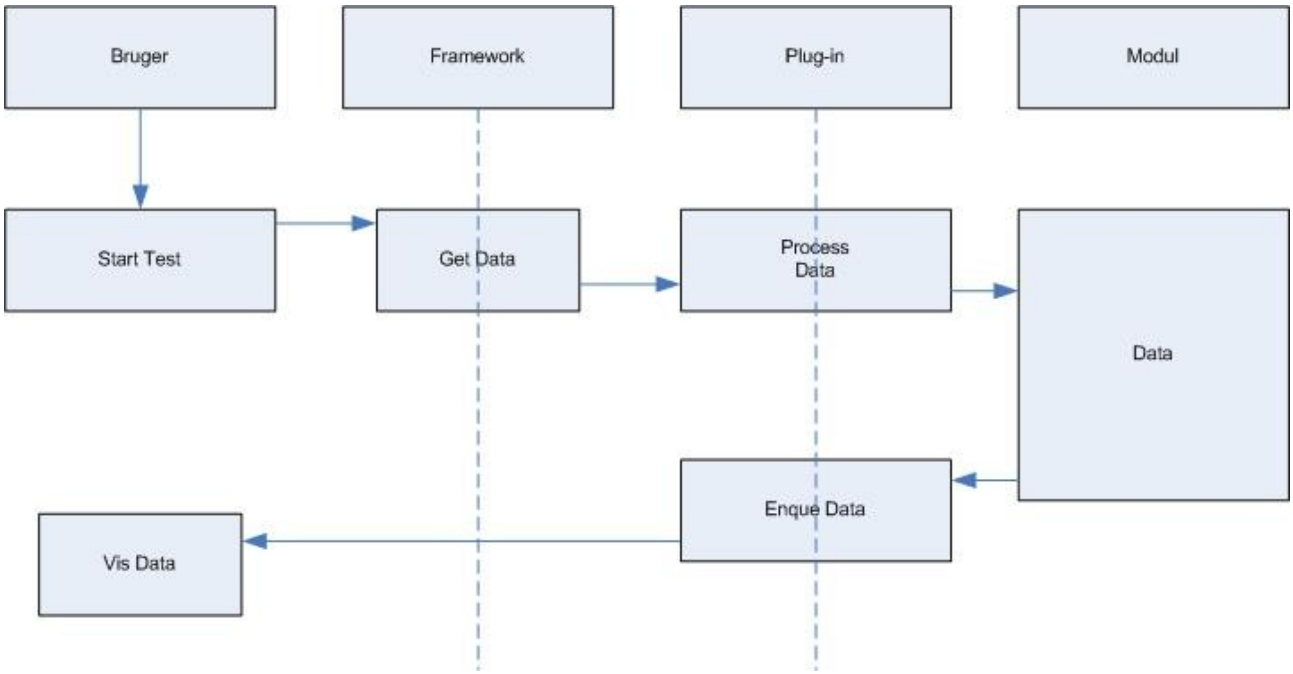



*Illustration 1:Sekvens diagram over interaktionen imellem de forskellige elementer*

## Data flow

Dataene skal ”flyttes” fra selve DAQmx modulet og præsenteres for brugeren i form som brugeren har besluttet sig for. I det enkelt plugin skal dataene helst ændres fra den rå form om til en protokol som er fastlagt for frameworket. Så kan det frameworket repræsentere dataene på samme måde, uanset hvilket plugin det kommer fra.

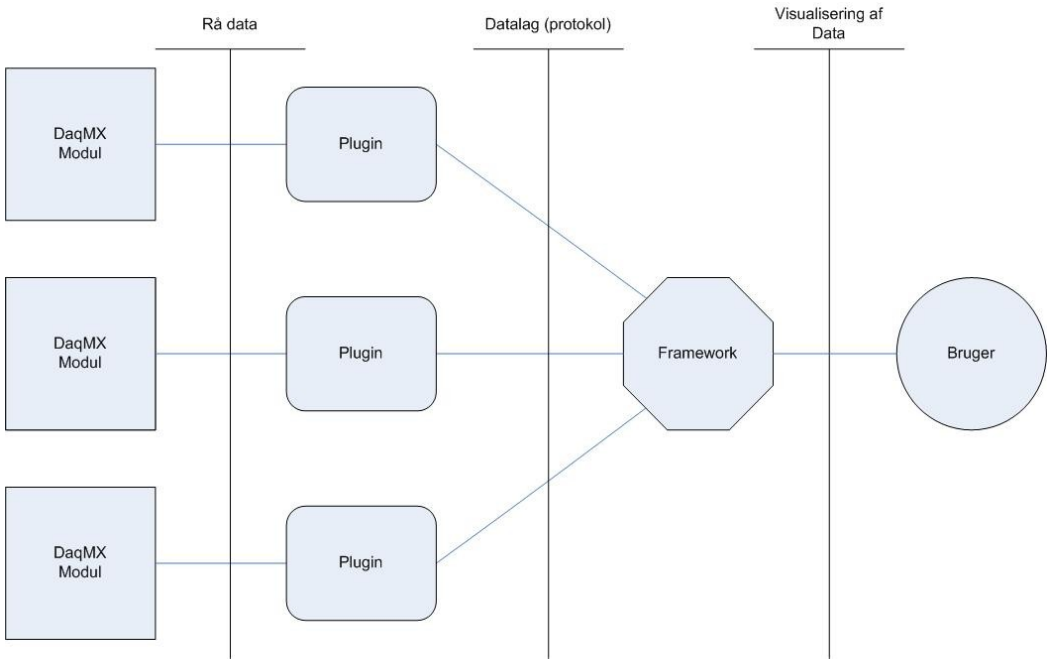



*Illustration 2: Data Flow Diagram*

## Framework

Den vigtigste funktion for frameworket er at kunne indsamle data og præsentere det for brugeren. Dataene skal kunne indhentes igennem de plugins der foretager målingen og fortolker dataene fra selve måle modulerne. Altså skal der bruges en kommunikations form imellem framework og plugins.
Der skulle gerne være mulighed for at kunne kommunikere begge veje, så det vil være muligt at indkorpere output fra frameworket.

## Datalag

Datalaget kan fungere som et kommunikations medie imellem framework og plugin. Datalaget ville opføre sig som en kø. Her kan pluginet den data som er konverteret til den aftalte protokol, og frameworket kan så efter behov læse data herfra. Med en kø til både input og output vil det være muligt for frameworket at kunne lægge outputs til plugins på en udadgående kø, og plugins kan lægge data til frameworket på en indadgående kø. Det vil dog være nødvendigt med ID system for at kunne at de forskellige plugins ikke tager outputs fra hinanden, eller frameworket læser noget data og tolker det som om det kommer fra et andet plugin end hvor det hører til.

## State Diagrams

Måden man programmere på i Labview er for det meste i state machines, da dette fungerer godt i Labview grundet måden man programmere som om det er et elektrisk kredsløb. Følgende er et udkast til hvordan programmering i frameworket kunne være bygget op.

### Framework State diagram

Frameworket skal håndtere interaktionen med brugeren. Det nedestående statediagram er et udkast til hvordan det kan sammensættes.

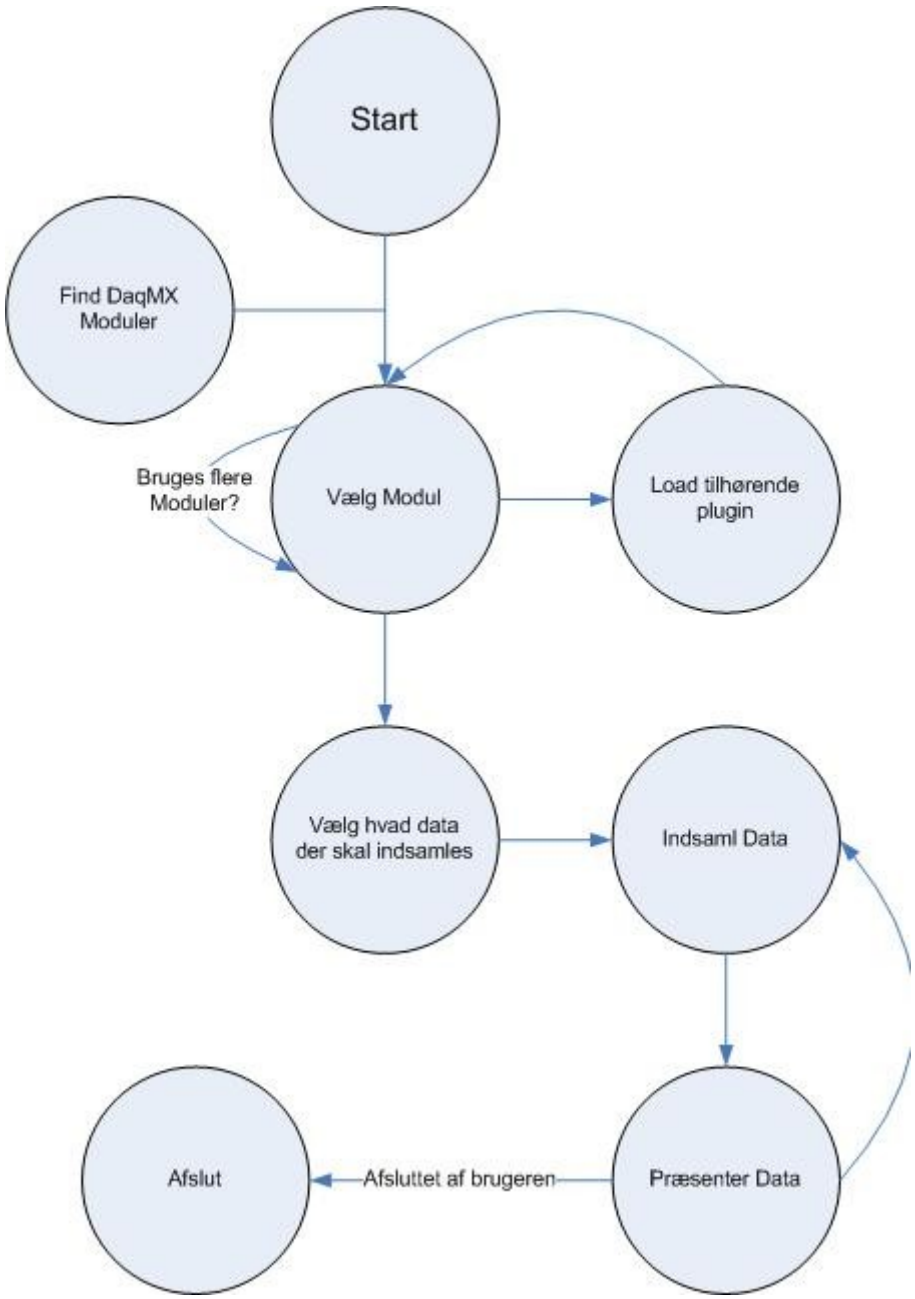



*Illustration 3:Statediagram for framework*

**Plug-in State diagram**

State diagrammet er mere simpelt, hvilket det jo også helst skal være. Målet er jo netop at udviklingen af plugins tager kort tid.

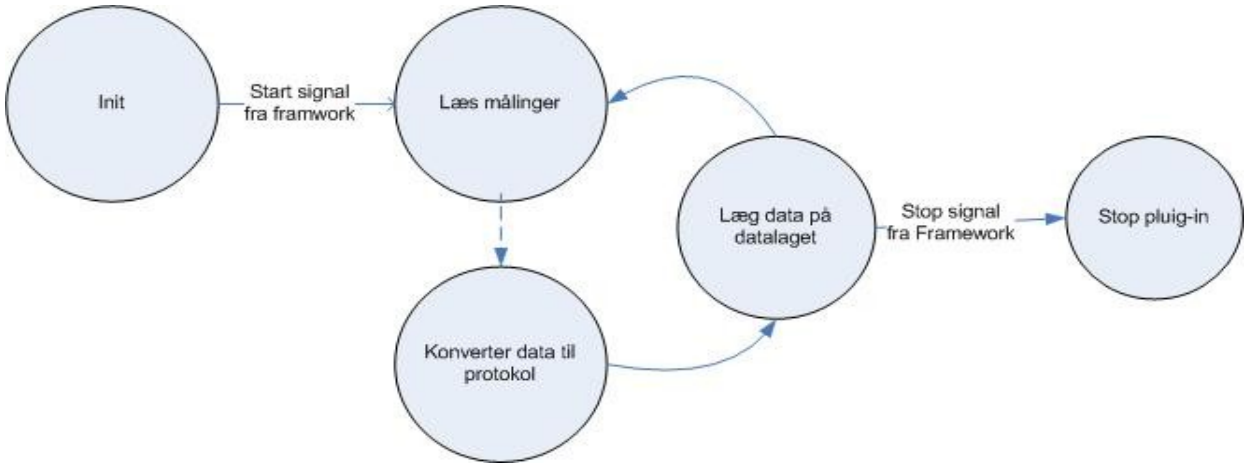



*Illustration 4:Statediagram for plugin*

## Begrænsning

Den primære fokus vil være at på kommunikationen imellem framework og plugin op at køre. Dernæst er det også kun input fra plugins der i første omgang bliver fokuseret på, det det netop er målinger som er i hovedsædet, og ikke udadgående signaler.
Første mål er at få 2 fungerende plugins til at foretage målinger, og kommunikere den data ind til frameworket, der så kan vise det på en tilfredsstillende måde. De 2 plugins og også moduler jeg vil gå efter at udvikle først er et til måling af strøm, også et andet analogt input modul.
Når dette mål er opnået kan der kigges på at lave output til plugins. Hvis dette også nås så er der rig mulighed for at udvikle flere forskellige plugins.

# Implementering

I det følgende vil jeg beskrive hvordan den endelige implementering blev. Jeg vil beskrive de funktioner der findes i frameworket, og hvordan det er bygget op. Derefter vil jeg forklare de 2 plugins.
**For at få et overblik over alle filerne i dette projekt kan man med fordel benytte sig af projekt filen "Teststand Pluginstruktur.lvproj".*

## GUI

Interfacet er designet med simplicitet for øje.
Det der dominerer grænsefladen er en Tab kontrol hvor man skifte imellem de forskellige måder at vise dataene på. Der i den nuværende version understøttelse af at kunne se dataene på en Waveform graph, eller et Waveform chart. Til højre i billet er kontrol knapperne til at styre programmet.
Ved tryk på load plugin vil der komme en fil dialog frem hvor man så kan vælge det relevante plugin man vil loade ind i frameworket.

## Datalag

I stedet for den planlagte kø funktion er valget i stedet faldet på en have en buffer på hvert plugin. Frameworket kan så læse denne data undervejs i data indsamlingen om kombinere den med data fra andre plugins. Grundet til dette skifte var at der opstod mange problemer med at have en kø funktion, også kunne adskille hvilket data der stammede fra hvilket plugin. I den løsning jeg har valgt bliver dataene hentet fra pluginet baseret på dets reference som er unik. Jeg er derfor sikker på at dataene jeg indhenter fra referencen til plugin X faktisk er målt på plugin X.
Det virkede som en mere elegant løsning i mine øjne.

## Waveforms

Måden jeg har valgt at repræsentere dataene i de 2 plugins jeg har lavet, og derved også i frameworket er Labview struktur waveform.
En waveform er en kombination af 3 variabler, T0, dt og Y, samt noget metadata.
T0 er det tidstemplet for den først måling foretaget. Variablen dt er tiden imellem de efterfølgende målinger. Variablen Y er et 1 dimensionalt array hvori de målte værdier er gemt, første måling på plads 0, også op til n antal målinger.
Metadaten der kan være med i waveform kan være navnet på kanalerne hvor målingen er foretaget.

## Framework

Her samles dataen og bliver vist på de 2 grafer det på nuværende tidspunkt findes i frameworket. De 2 plugins jeg har lavet kører med analogt input, og jeg har derfor ikke implementeret funktioner til at håndtere digitale input, eller nogen slag output. Dog vil det sagtens kunne lade sig gøre. Det er

blot at lave et ny faneblad til det, også lave den korrekte data opsamling.

**Main.vi**

Dette er statemachinen for frameworket. Det er også samme program som indeholder GUIen.
De 3 vigtigste states er ”Init”, ”Idle” og ”Run&DisplayData”. Disse 3 states er defineret i klassen ”MainStates.ctl”. Når programmet startes så starter det i staten ”Init”. Her gøres der ikke meget andet en at nulstille de forskellige elementer. Labview har det nemlig med at gemme værdier fra gang til gang. I dette tilfælde er det listen af plugin som skal køres der nulstilles. Grunden til den især er vigtig er at metoden jeg benytter til at tilføje ny plugins, blot ville føje den nye værdi oven i den eksisterende. Så hvis jeg havde loadet to plugins sidste gang, og listen ikke er blevet nulstillet, så ville det næste plugin der blev tilføjet til de to. Altså vil jeg have loadet tre plugins, i stedet for det ene jeg havde tænkt mig.
Efter at ”Init” statens kode er blevet kørt fortsættes der til ”Idle” staten.Transitionen imellem de forskellige states klares ved at tilskrive det shift register som holder styr forskellige states, det kan ses herunder.

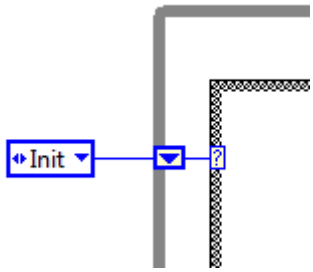



*Illustration 5:Statehandler*

Den yderste faste grå linje repræsentere den while løkke der omgiver hele programmet for at det ikke blot kører en gang. Den indre lettere brudte linje er den case struktur som sørger for at der bliver skiftet imellem de korrekte states. Der kan kun gives de input til case strukturen som er defineret i ”MainStates.ctl”. Hvis man ønsker at oprette en ny state skal der oprettes et ny element i ”MainStates.ctl”.
I ”Idle” staten er der en event struktur. Den sørger for at fange de forskellige bruger input, og foretage de korrekte ændringer i statesne. Den kan dog kun fange bruget inputtene så længe man befinder sig i ”Idle” staten. Der er i denne state at programmeringen til at tilføje et plugin til plugin listen forefindes. Når der trykkes på ”Start måling” gåes der videre til staten hvor alt det sjove sker, ”Run&DisplayData”.
I ”Run&DisplayData” gennemgås plugin listen først. Dette gøres med en for løkke, der tager placeringen for hver enkelt plugin og åbner en reference til det plugin. Referencen kan så bruges til at eksekvere koden der ligger i pluginet. Referencerne til plugins pakkes igen sammen i et nyt array som så benyttes til data indsamlingen.

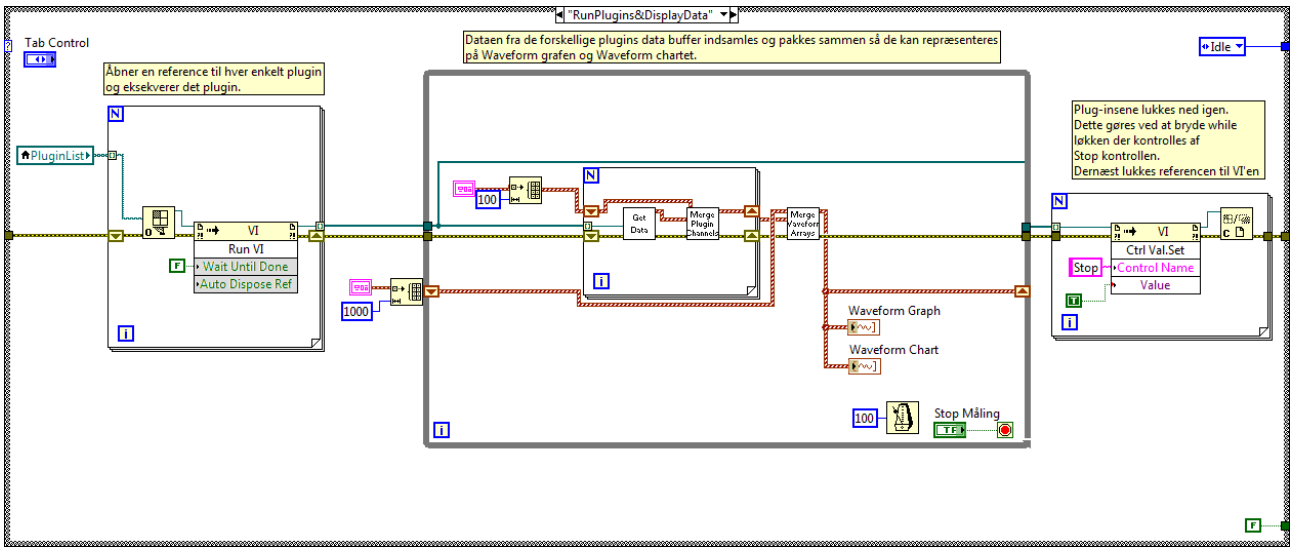



*Illustration 6:State - Run&DisplayData*

Ved hjælp af referencen til pluginet kan jeg tilgå den data buffer der er i pluginet. Selve indhentningen af dataene foretages i subvien ”Getdata.vi”. Den data fodres så ind i subvien ”Merge Channels”. Resultatet af dette er at de individuelle plugins kanaler nu alle optræder som kanaler i et stort modul. Dette ene modul kan nu vises på Waveform grafen og Waveform chartet.
While løkken sørger for at denne handling bliver ved indtil brugerne trykker på ”Stop måling” knappen. Så der skal tages hensyn til at sidste iterations resultater ikke bliver overskrevet af de nye. Dette klare med VI'en ”Merge Waveform Arrays.vi”, der også forklares nærmere senere i rapporten. Resultatet er dog at waveformsene fra de forskellige iterationer bliver sammenkoblet så de alle bliver vist på grafen.

### Getdata.vi

Denne vi modtager et array, samt en reference til det plugin hvorfra dataen skal hentes. Via værktøjer ”invoke node” kan jeg tilgå data bufferen der er programmeret i pluginet. Den data kommer som en ”variant”, og skal derfor typecastes ind til den korrekte form. Der klares ved hjælp at et tomt 1dimensionelt array med waveforms i. Dataene kan nu bruges som output fra subvien.

### Merge Channels.vi

Denne subvi modtager to array, data fra 2 moduler. Disse 2 array skal nu kobles sammen til 1 array, men på en sådan måde at det ligner outputtet fra 1 modul. Dette gøres ved at array B indsættes i array A. Array Bs størrelse findes først. Dernæste rykkes alle værdierne i Array A så langt længere ned i deres array. Der er nu plads til array B i array A, og B indsættes derfor i A. Nu har vi kun 1 array hvor værdierne fra begge arrays optræder.

### Merge Waveform Arrays.vi

Her har vi igen 2 arrays. Array A er arrayet med alt dataene fra forrige iterationer. Array B er de nye data. Vi ved at de 2 arrays pladser passer til hver sin kanal fra hvert sit plugin, men de vil have samme plads som før. Vi kan derfor via en for-løkke tage waveform på hver enkelt plads i Array A og kæde den sammen med waveformen på den tilsvarende plads i Array B.

## Plug-ins

Jeg valgte at fokusere på at have 2 fungerende plugins til at kunne teste frameworket. Det var ikke nok med blot et plugin, da jeg så ikke kunne være sikker på om mine funktioner til at kæde dataene sammen fungerede. Desværre har udviklingen af plugins lidt under dette da det meste af tiden er gået på at finde ud hvordan man kunne kombinere dataene fra de forskellige plugins.

### 9211

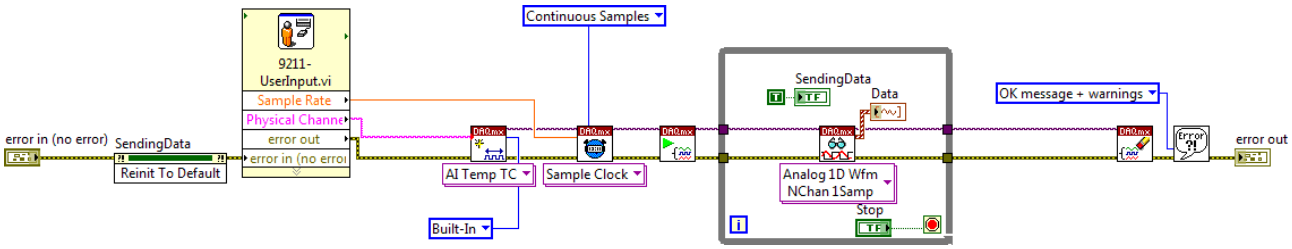



*Illustration 7:Blockdiagram for pluginet til modul 9211*

Modul 9211 bruges til at måle temperature. Det er et analogt input, hvilket skal huskes når man definere måle opgaven i Labview.
Det første pluginet gør er at bede brugeren omkring nogle parametre for målingen. Dette gøres med subvien ”9211-UserInput”, som er en modificeret ekspres vi.
Parametrene der spørges om er samplerate, og hvilke fysiske kanaler som modulet man ønsker data fra er lokaliseret. Med disse parametre på plads kan man opsætte en måle opgave i labview. Først definere man opgaven, i dette tilfælde en temperatur måling. Her sættes der hvilke fysiske kanaler der skal måles på. Dernæst sættes man sample raten via ”Sample Clock” funktionen. Med parametrene på plads sættes opgaven i gang. Plug-inet går nu ind i en while løkke, hvor der hele tiden læses værdier fra modulet, og resultaterne bliver lagt i bufferen ”Data”. Dette fortsættes indtil frameworket ændrer på værdien af ”Stop”. Til sidst frigøres måleopgaven og derved også målemodulet.

### 9215

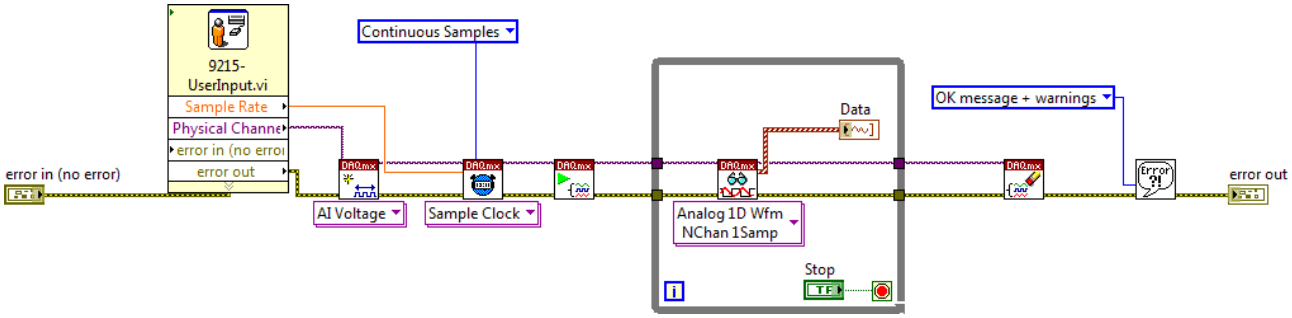


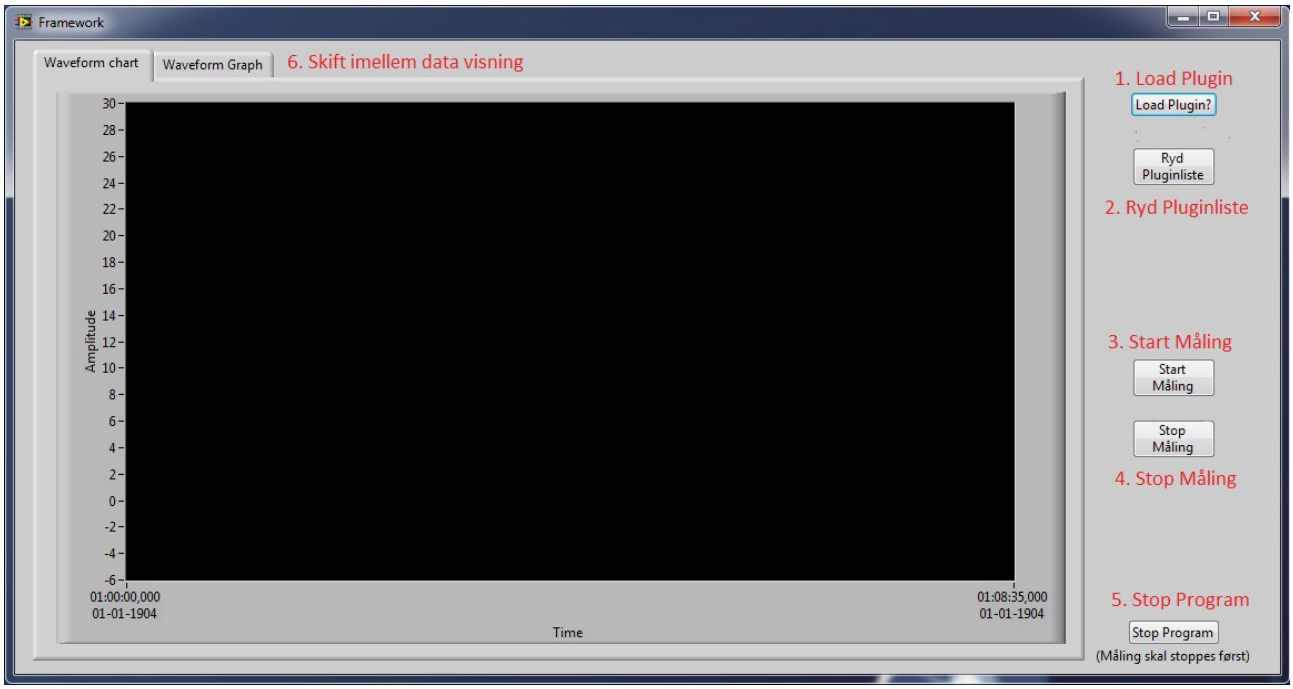

*Illustration 8:Blockdiagram for pluginet  til modul 9215*

Modul 9215 bruges til at måle strøm via analogt input.
Først bedes brugeren omkring parametre for målingen. For dette plugin er det sample rate og hvilke fysiske kanaler at måle modulet befinder sig. Dernæst defineres måle opgaven, og sample raten sættes ved hjælp af ”Sample Clock”. Måleopgaven sættes i gang, og inde i while løkken blive der foretaget målinger og resultaterne skrevet til data bufferen. Når frameworket beder pluginet om at stoppe så frigøres måleopgaven og måle modulet.

## Brugervejledning

Frameworket kan både benyttes som et selvstændig program, eller bruges som et action step i en teststand sekvens. Filen ”TeststandSekvens.seq” er et eksempel på dette. Ved at køre sekvensen kan men tilknytte en ID nummer for kørslen, og få generet en rapport på om der var nogle fejl under eksekveringen. Forskellen er blot hvordan programmeret bliver eksekveret, og ikke så meget i brugen af selve frameworket.

Herunder er et billede af interfacet, og en nærmere forklaring om hvordan det benyttes.




### 1. Load Plugin

For at loade et nyt plugin trykkes der på knappen ”Load Plugin” i højre side. Dernæst kommer der en fil dialog frem hvor man findes frem til sit plugin.

Hvis man ønsker at loade et plugin trykker man blot på knappen igen.

### 2. Ryd Plugin listen

Hvis man loader et forkert plugin, eller blot ønsker at sammensætte andre plugins til sin måling kan man benytte denne knap. Den fjerne samtlige plugins der er i plugin listen.

### 3. Start måling

Når du har loadet alle de plugins som de ønsker, så kan du begynde målingen med denne knap.

Så eksekveres de plugins du har valgt at loade. Der vil højst sandsynligt være ekstra parametre som skal sættes, det vil der komme pop-ups med. Udfyld parametrene efter ønske, og klik på OK-knappen. Så snart du har udfyldt et plugins parametre og trykket OK vil målingen gå i gang i baggrunden. Så snart ud har udfyldt alle parametrene for de forskellige plugins så vil du kunne se de kører på grafen.

**4. Stop Måling**

Når du ønsker at stoppe måling benytter du denne knap. Den stopper alle plugins, og derved også målingerne. Du kan vælge at starte måling igen hvis du lyster, men vil så skulle udfylde parametrene igen.

**5. Stop programmeret**

Hvis der ikke er en måling igen kan du afslutte programmet med denne knap. Det er dog nødvendigt at der ikke er en måling i gang.

**6. Skift imellem data visning.**

Hvis du gerne vil se dataene repræsenteret på en anden måde, så kan du skifte imellem dem ved hjælp af fanebladene.

## Praktisk test

I testen vil jeg se om jeg kan loade 2 forskellige plugins, og få vist hver deres data på den samme graf i frameworket.
Det ene plugin vil være til et 9211 modul der måler temperature. 9211 modulet vil være tilsluttet via et DAQmx chassi, et 9272, der er tilsluttet computeren via USB.
Det andet plugin vil være til et 9215 modul der måler strøm via analogt input. Det vil være simuleret via National Instruments ”Measurement & Automation” software. Det er det nemmeste at lave i det tilfælde med et simuleret modul, da man i ”Measurement & Automation” kan sætte det simulerede modul til at simulere forskellige output signaler. Altså kræver det ikke en signal generator for at test inputtet på et fysisk modul. I denne test vil det simulerede 9215 modul genere en sinus bølge på alle 4 kanaler.
Jeg vil eksekvere mit program igennem Teststand da dette jo netop var det vi ville gøre det nemmere at bruge.
For at kunne eksekvere mit program indsætter jeg et aktion step i Teststand. Dette kan ses ved den øvre røde prik. De røde prikker er nogle jeg har sat for at kunne henlede opmærksomheden på de mest relevante områder.

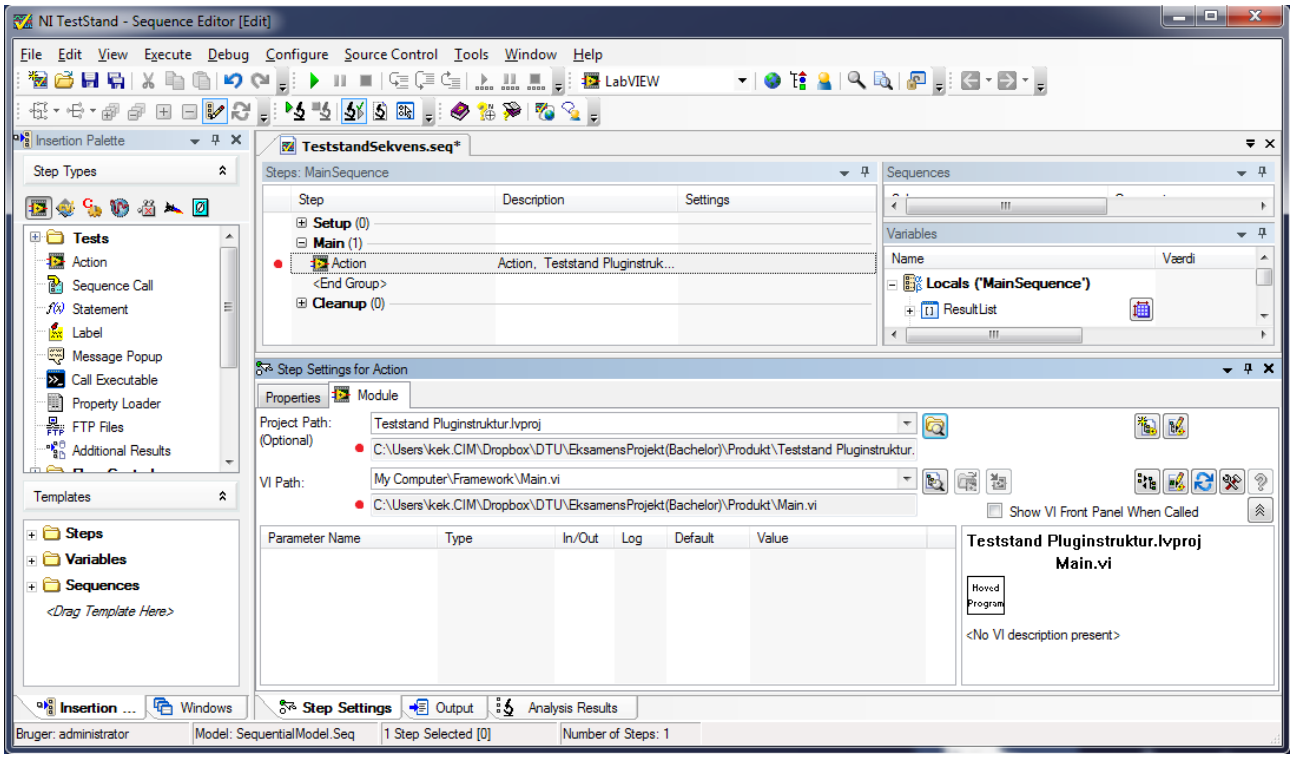


*Illustration 9:Teststand sekvens fil*

For at eksekvere hovedprogrammet hvori plugin kan loades så lader man ”VI Path” (markeret ved den tredje røde prik, talt fra oven) pege på Main.vi filen. Man kan også bruge give aktion steppet placeringen for helt projektet(markeret ved den anden røde prik, talt fra oven).

Da dette er på plads så starter jeg test sekvensen, jeg vælger UUT Serie nummer 4242 og trykker OK. Mit framework kommer nu frem på skærmen og jeg kan loade plugins. Jeg vælger plugin 9211, og plugin 9215. Da det er på plads sætter jeg gang i målingen med knappen ”Start Måling”. Jeg skal nu sætte parametrene for hvert plugin. Jeg sætter sample raten til 10, da 9211 ikke kan sample så hurtigt, og formålet med denne test ikke at teste hvor hurtigt det kan køre, det skal hellere stemme overens med sample rate. De fysiske kanaler sætter jeg til henholdsvis, a0 på ”cDAQ1Mod1” som er mit USB tilkoblede 9211 modul for 9211 pluginnet. For 9215 pluginnet vælger jeg mit simulerede 9215 modul, cDAQ9215, med alle 4 kanaler a0, a1, a2, og a3.
Da det er på plads lader jeg målingen køre et stykke tid, samtidig med at jeg stimulere temperatur sensoren på mit 9211 modul. Dette er resultatet.

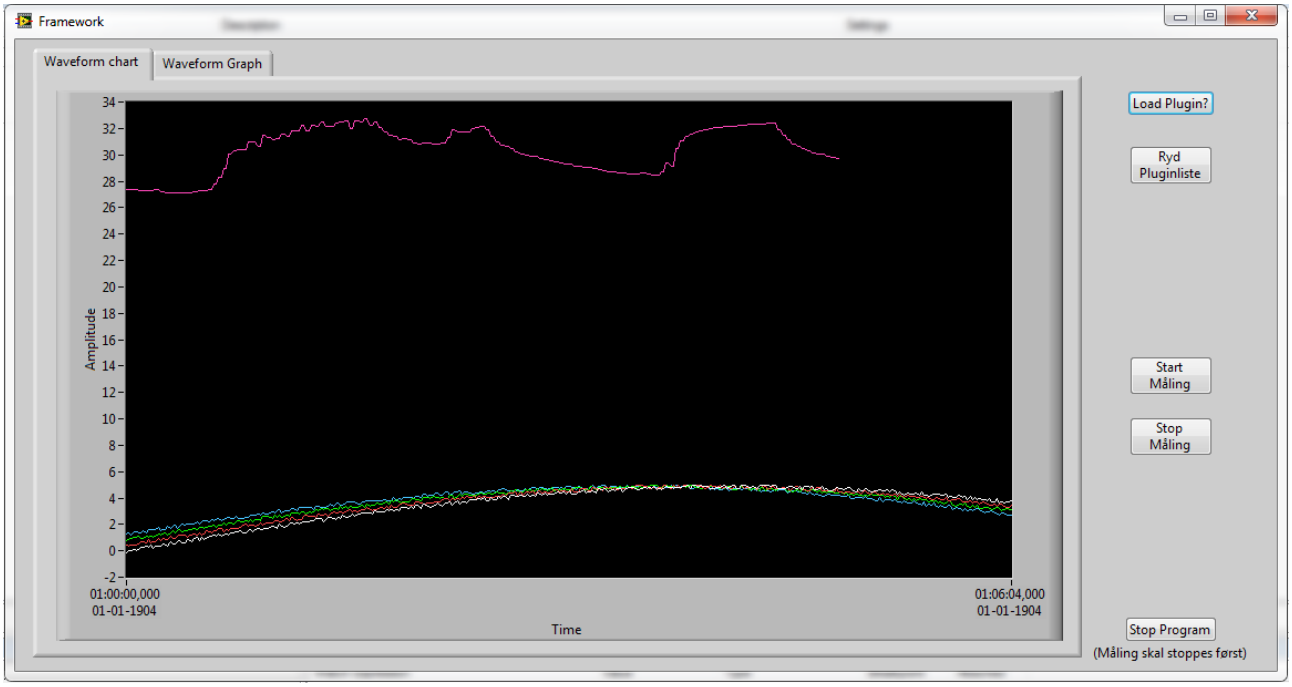
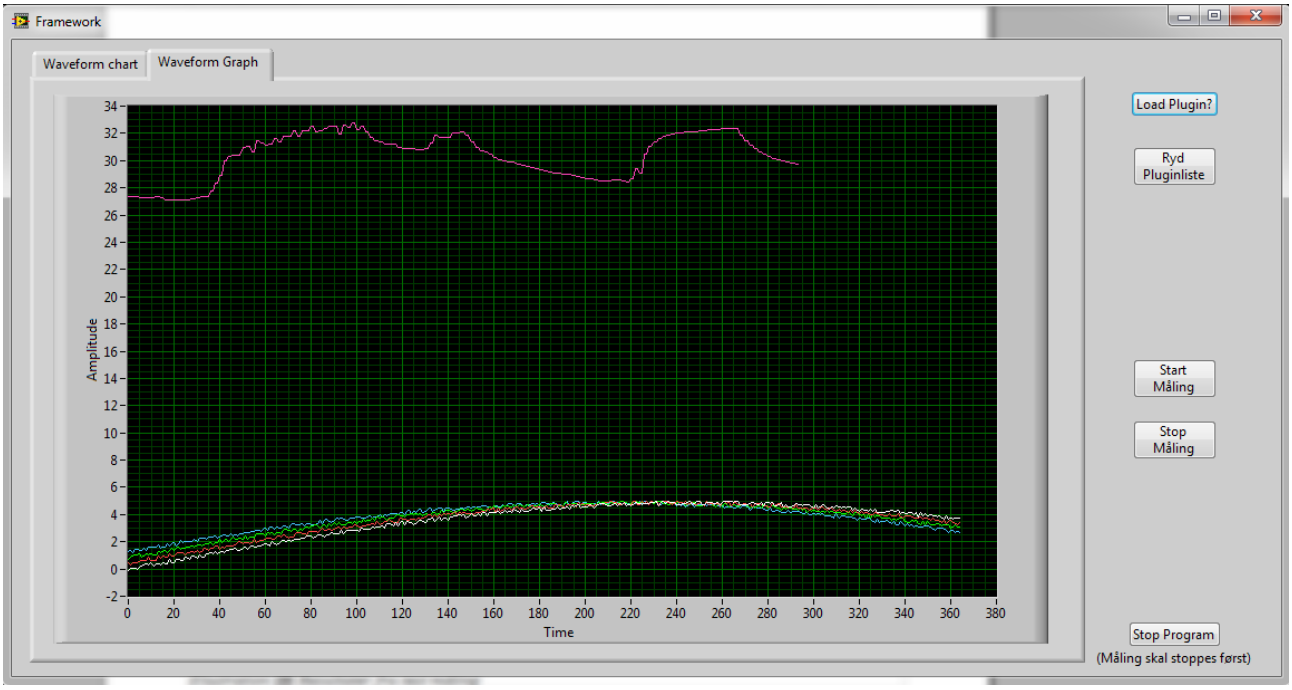



*Illustration 10:Resultater fra test måling*

Øverst er den enlige måling fra temparatur sensore tilkoblet mit 9211 modul, og nederst er de 4 kanaler fra det simulerede 9215 modul.

Og her er de samme resultater vist på det andet faneblad.




Samme måleresultater, blot en lidt anden repræsentation.
Det må man jo sige opfylder mine kriterier. Jeg kan foretage målinger fra flere forskellige plugins, og de kan blive repræsenteret i mit framework.
Testen er en succes.

**Unit test**

Unit test er ikke noget der som sådan benyttes indenfor Labview måle systemer. Det tager meget lang tid at kode, og er meget omstændigt. Det er kun hvis softwaren skal godkendes til medico virksomheder at man giver sig i kast med det. Det software jeg har udviklet er slet ikke sigtet imod så store og præcise systemer, da disse systemer altid vil være special udviklet til virksomheden.
Jeg har derfor forholdt mig til den praktiske test, da dette var normalen indenfor software af min kaliber.

# Fremtidige overvejelser

## Teststand integration

Ved videre udviklings ville en fokus på at få en bedre integration med Teststand klart været et plus. Hvis man kunne udnytte den automatisk rapport generering bedre ville det kunne give et rigtig kraftfuldt værktøj. Det har bare ikke været muligt for mig at udforske Teststand i en sådan grad undervejs, der var rigelig udfordring i at få de forskellige Labview elementer på plads.

## Output

Udover så ville output via andre moduler og plugins være noget der kan udforskes. Det vil være fuldt ud muligt via en smule videre udvikling af frameworket. Det gøres allerede en smule på måden jeg stopper de individuelle plugins når målingen stoppes. Jeg overskrive en kontrol i pluginnet, og sendes altså data fra frameworket til pluginnet. Det ville dog kræve en mere udviklet metode at kunne sende forskellige data til forskellige plugins, men metoden er på plads.

## Andre måleinstrumenter end DAQmx

Metoden jeg benytter til at hente data fra de forskellige plugins gør faktisk at det ikke er begrænset til kun at være DAQmx moduler. Det vil være muligt at kunne benytte andre moduler, blot man vil kunne skrive selve pluginnet i labview. Det ville altså være muligt at kunne udvide frameworket til måske at omhandle andre moduler i National Instruments store udbud af måleinstrumenter. Så længe pluginnet er lavet i Labview, og der en databuffer i pluginnet så vil frameworket kunne læse den data og repræsentere den.

# Konklusion

Da jeg begyndte projektet ville jeg finde en metode hvor jeg kunne programmere plugins og samle den data de målte i et samlet framework. Det er lykkedes, og de enkelt plugins er meget simple og hurtige at udvikle. Der er lagt en god grobund for en videre udvikling af dette produkt. Kommunikationen imellem framework og plugins er på plads, og det er blot at udvikle videre på denne teknik.

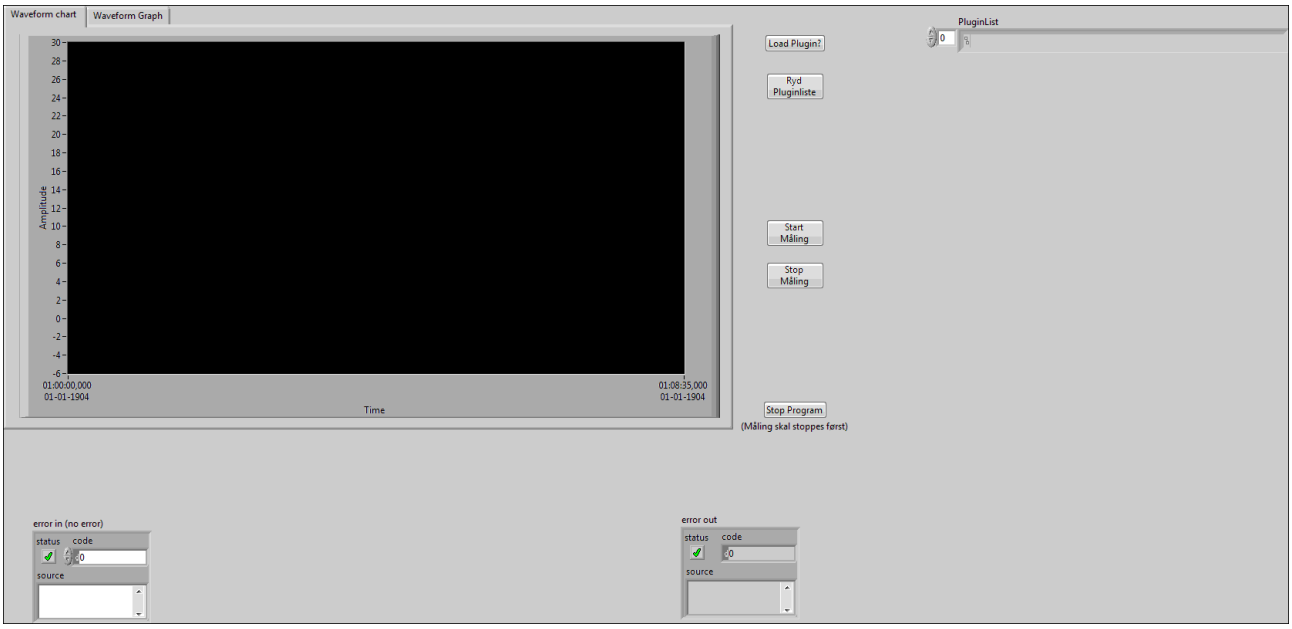
Jeg har undervejs i dette projekt fået et endnu bedre indblik i hvor stor den verden Labview er, og ikke mindst Teststand. Jeg kan forstå hvorfor de gerne vil forkorte udviklings tiden for Teststand sekvenser, og gøre det mere simpelt. Teststand er meget avanceret og det kræver meget erfaring for at man kan udnytte alle tricksene. Integrationen ind i Teststand er det som mangler mest i dette projekt, da den del simpelthen har været for stor en mundfuld.

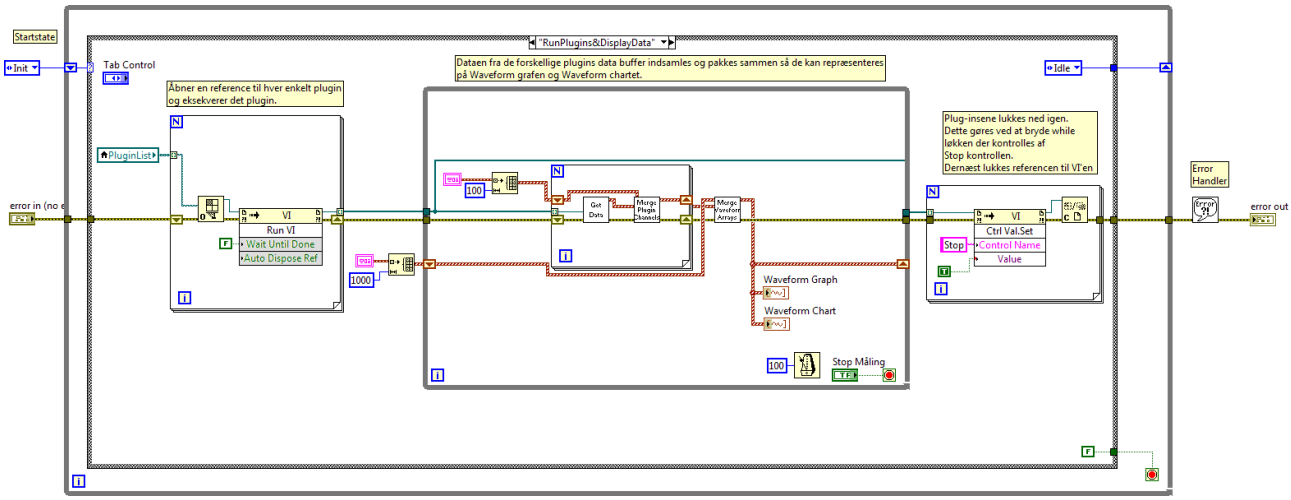
Alt i alt vil jeg betegne projektet som en succes. Jeg fandt ud af de vigtigste punkter, og har fundet ud hvor der skal udvikles videre.

## Labview Kildekode

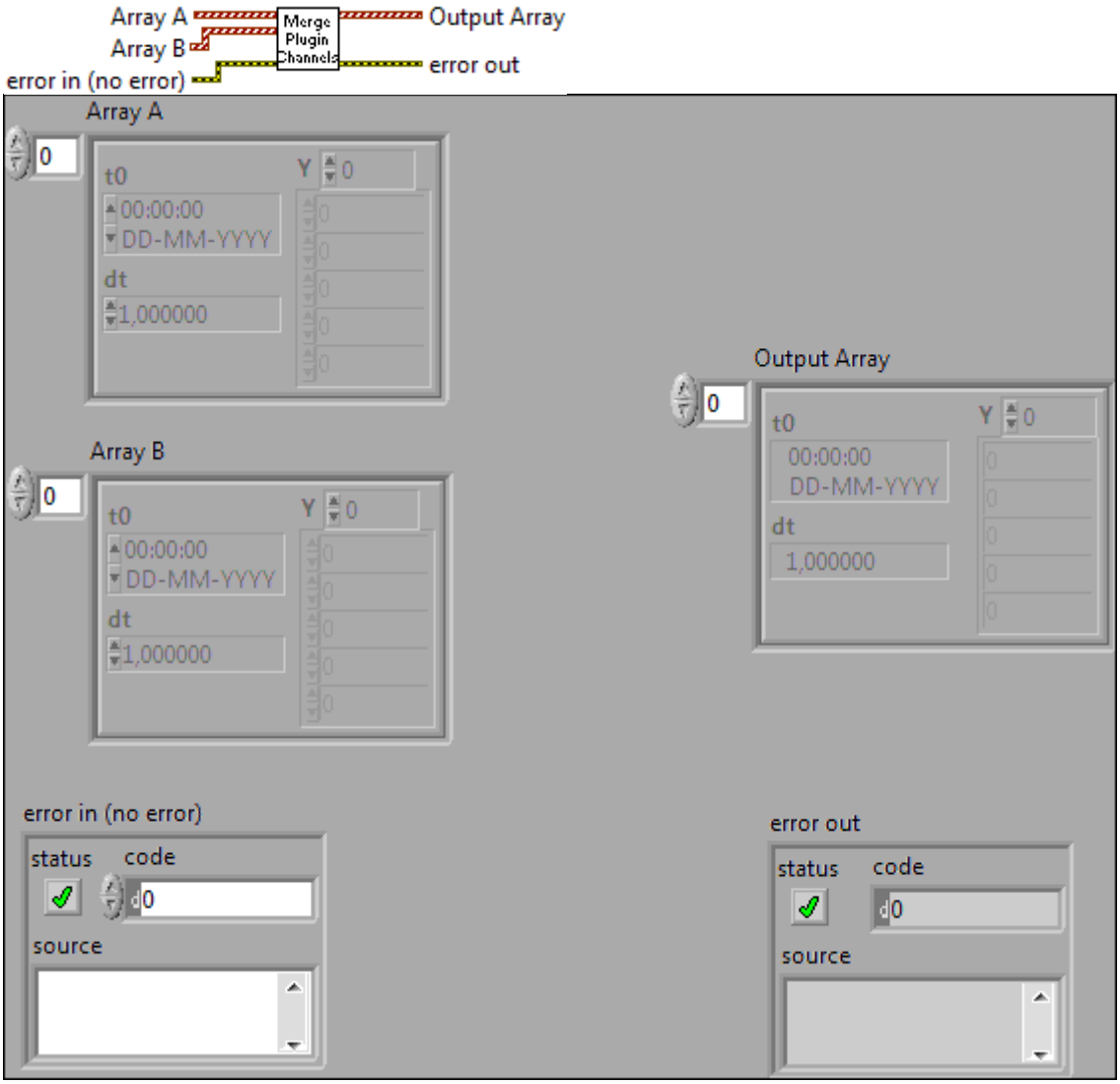
Da Labview er et grafisk kodesprog er det lidt besværligt at printe koden. Jeg har dog gjort et forsøg. Der er billeder af ikonet for den enkelt VI med input og Output, Frontpaneler (GUI, kontrols), samt Blokpanelerne (selve koden).

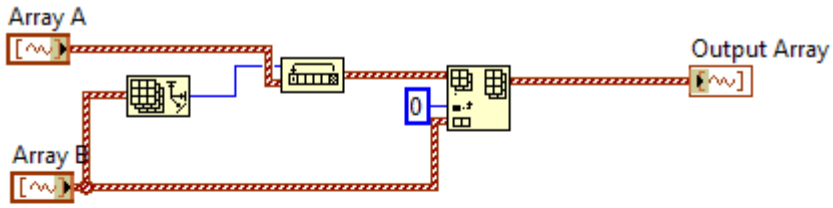
## Main.vi

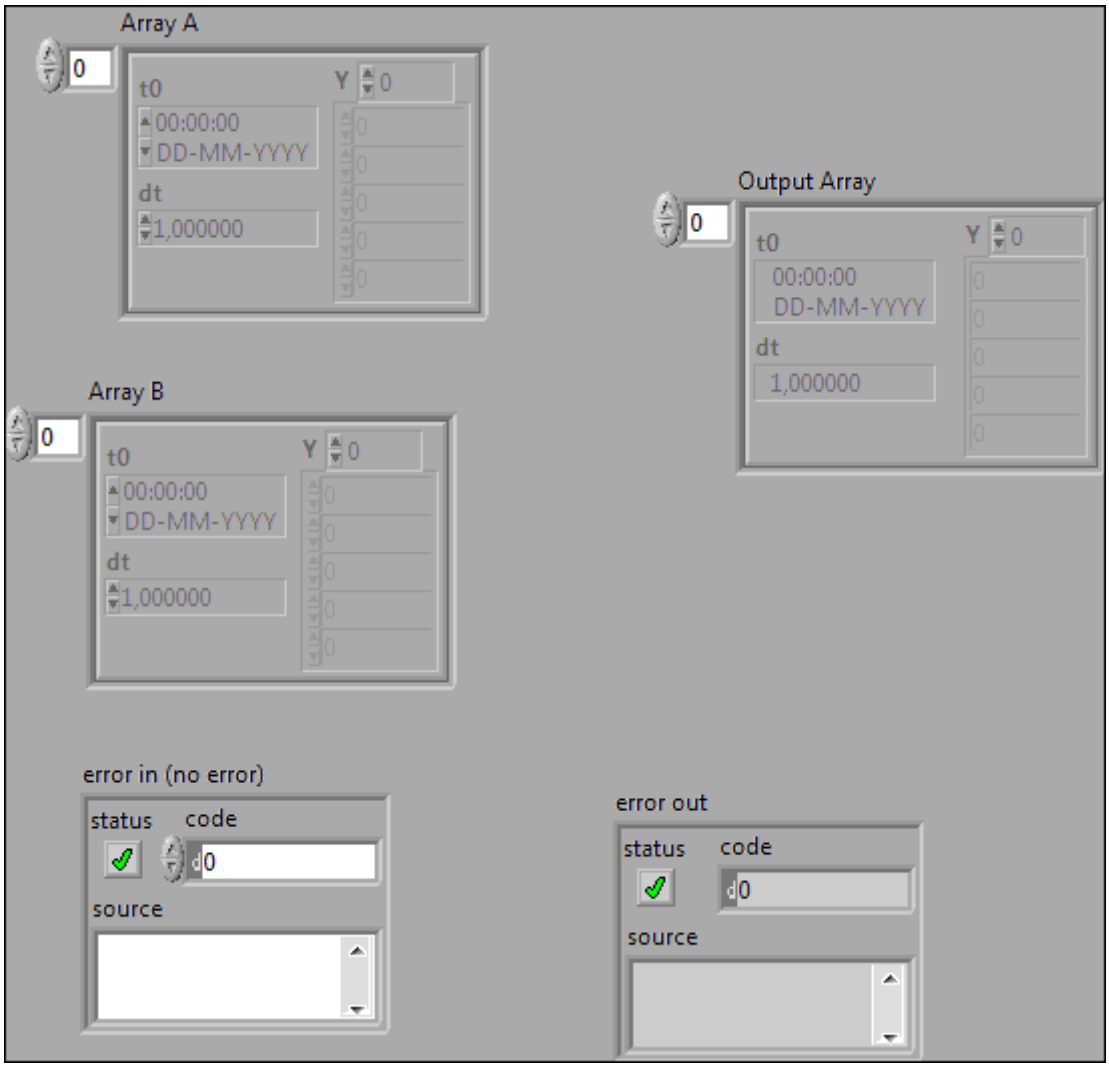



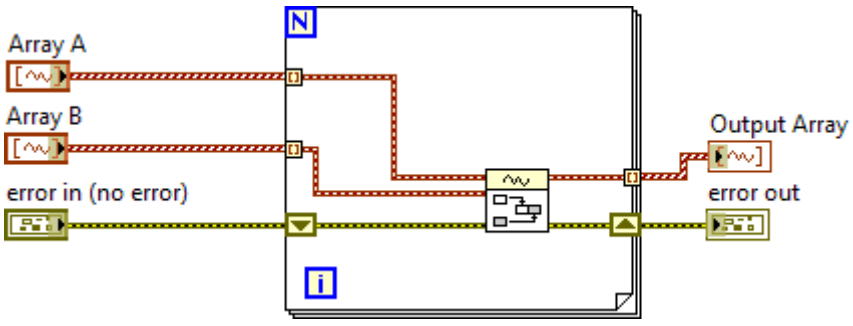
## Merge Channels



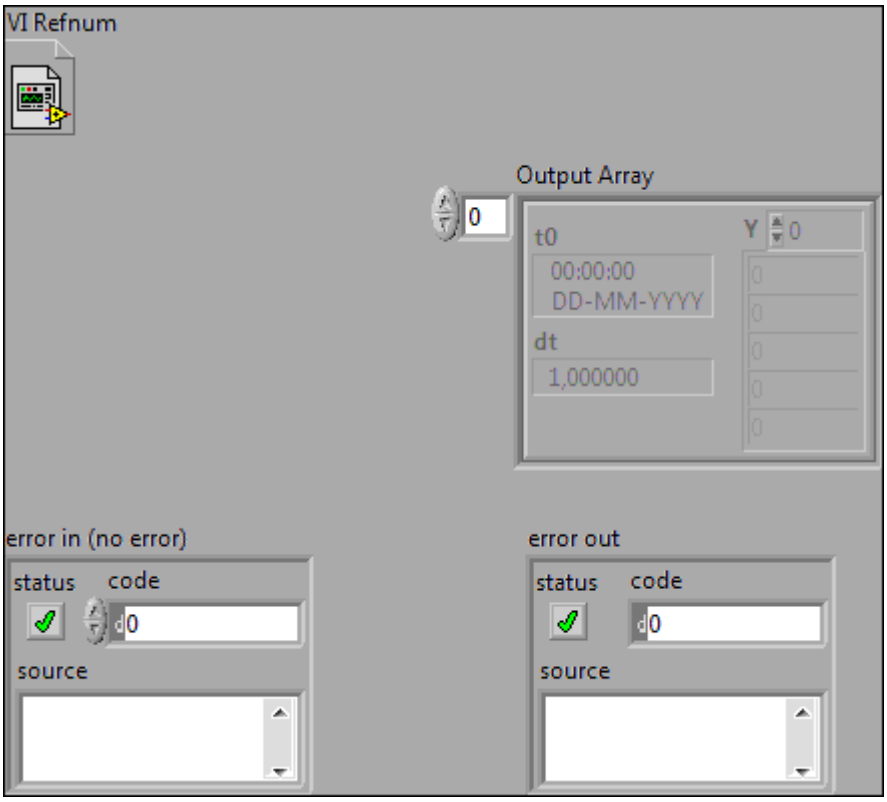

**Merge Waveform Arrays. vi**

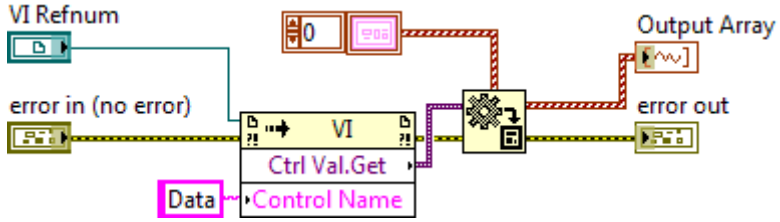



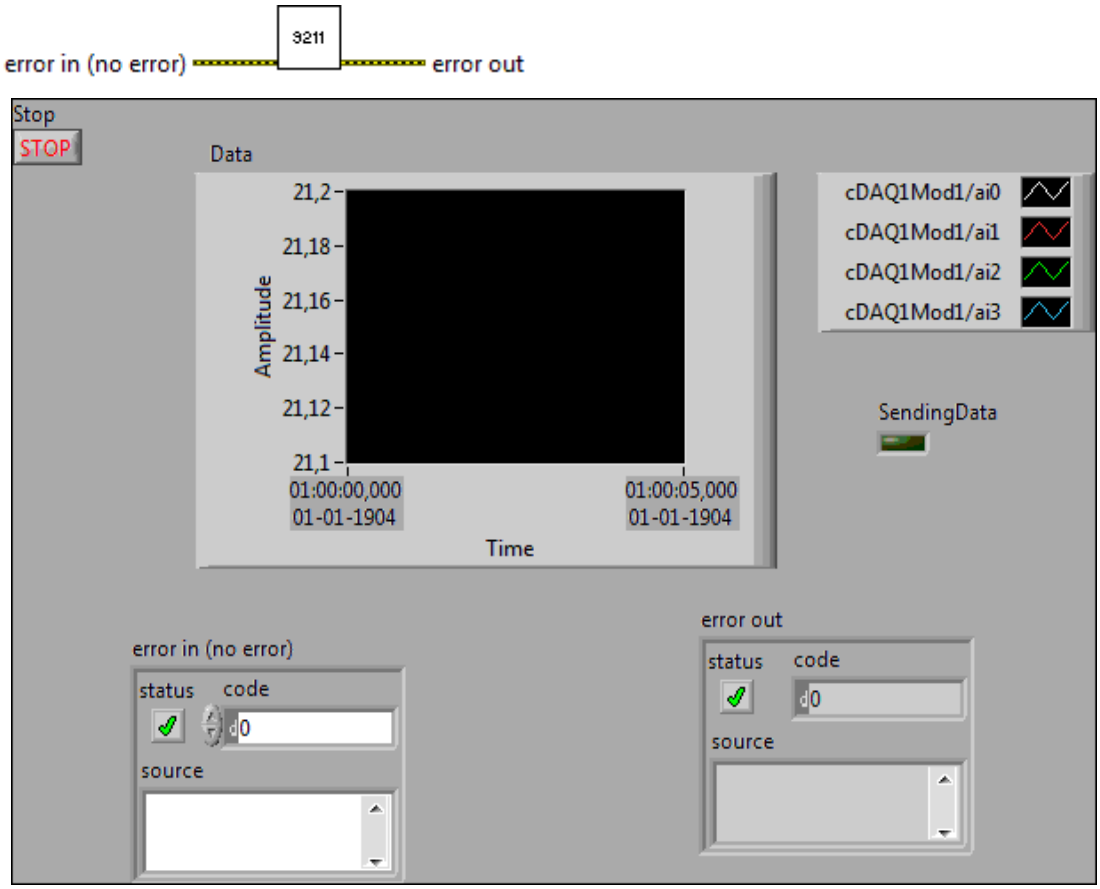
**Get Data.vi**

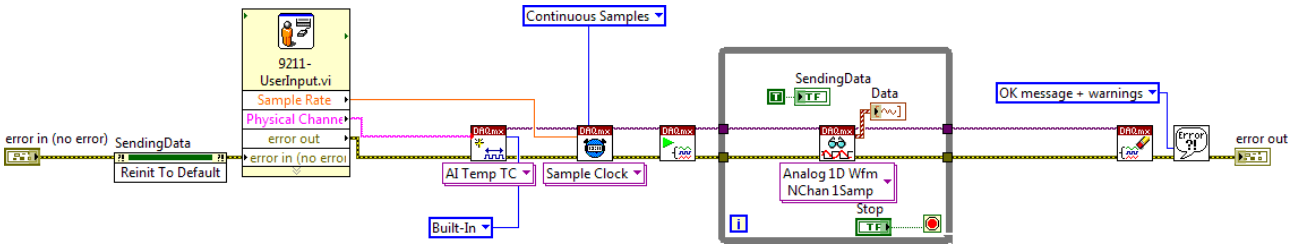

VI Refnum
error in (no error)
Get Data
Output Array
error out


VI Refnum
Output Array
0
t0
00:00:00
DD-MM-YYYY
dt
1,000000
Y
0
error in (no error)
status
code
0
source
error out
status
code
0
source

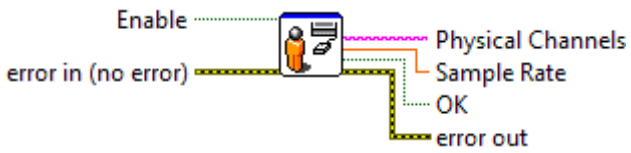

VI Refnum
error in (no error)
0
VI
Ctrl Val.Get
Data
Control Name
Output Array
error out

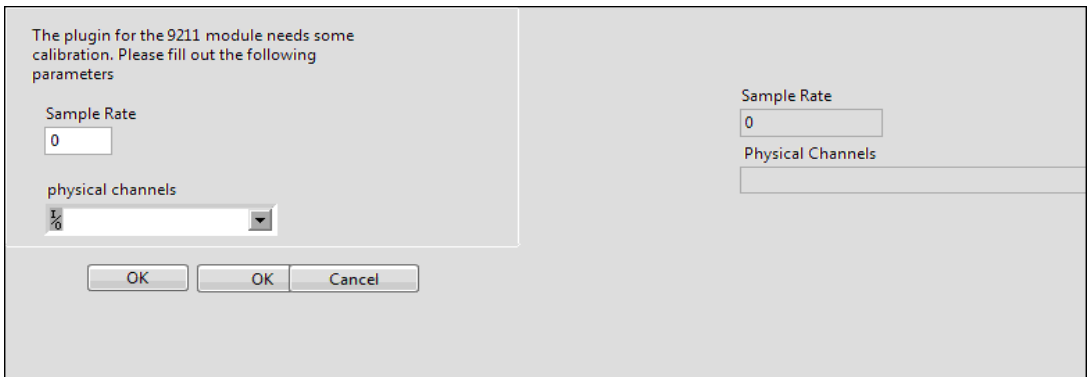
**9211.vi**

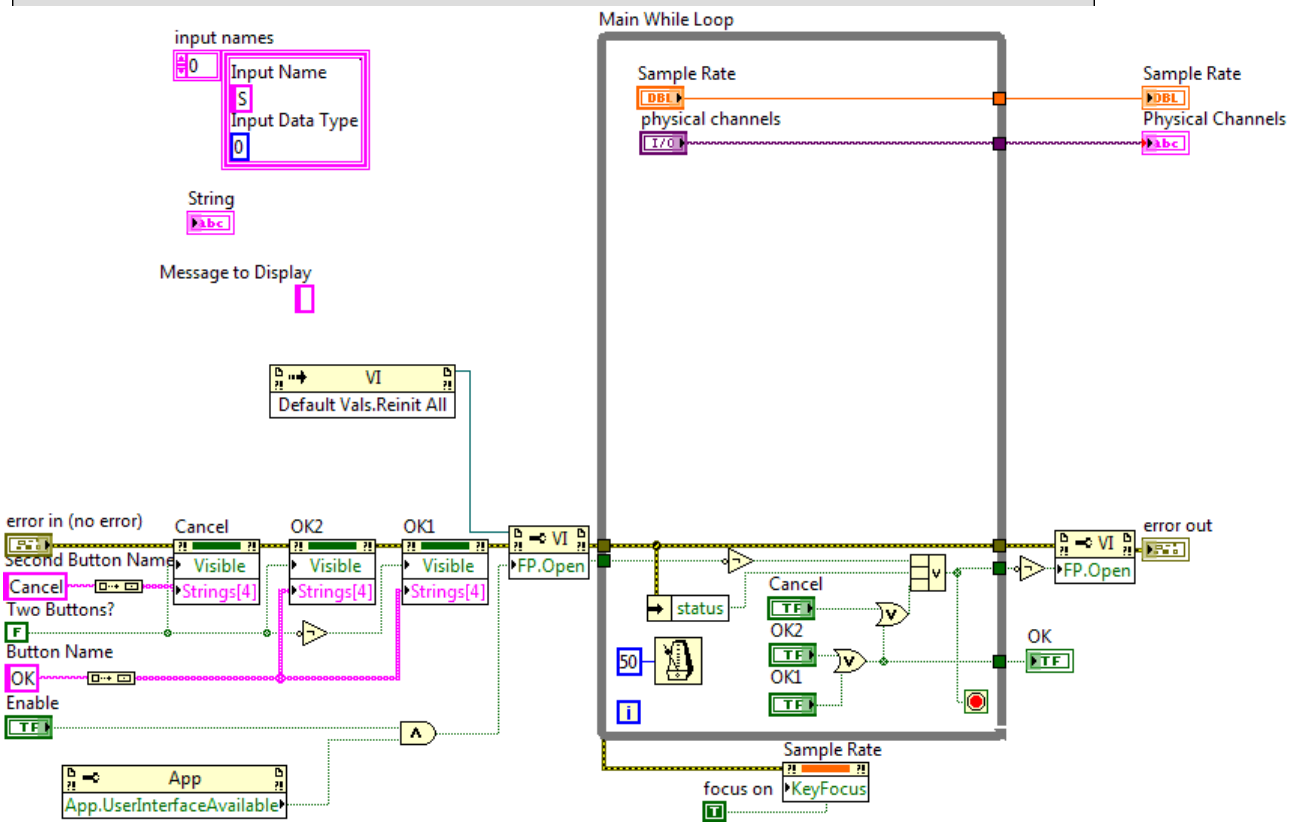

9211
error in (no error)
error out
Stop
STOP
Data
21,2
21,18
21,16
21,14
21,12
21,1
Amplitude
01:00:00,000
01-01-1904
01:00:05,000
01-01-1904
Time
cDAQ1Mod1/ai0
cDAQ1Mod1/ai1
cDAQ1Mod1/ai2
cDAQ1Mod1/ai3
SendingData
error in (no error)
status
code
0
source
error out
status
code
0
source


error in (no error)
SendingData
Reinit To Default
9211-
UserInput.vi
Sample Rate
Physical Channe
error out
error in (no erro
Continuous Samples
AI Temp TC
Sample Clock
Built-In
SendingData
Data
Analog 1D Wfm
NChan 1Samp
Stop
OK message + warnings
error out

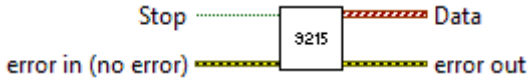
## 9211-Userinput.vi

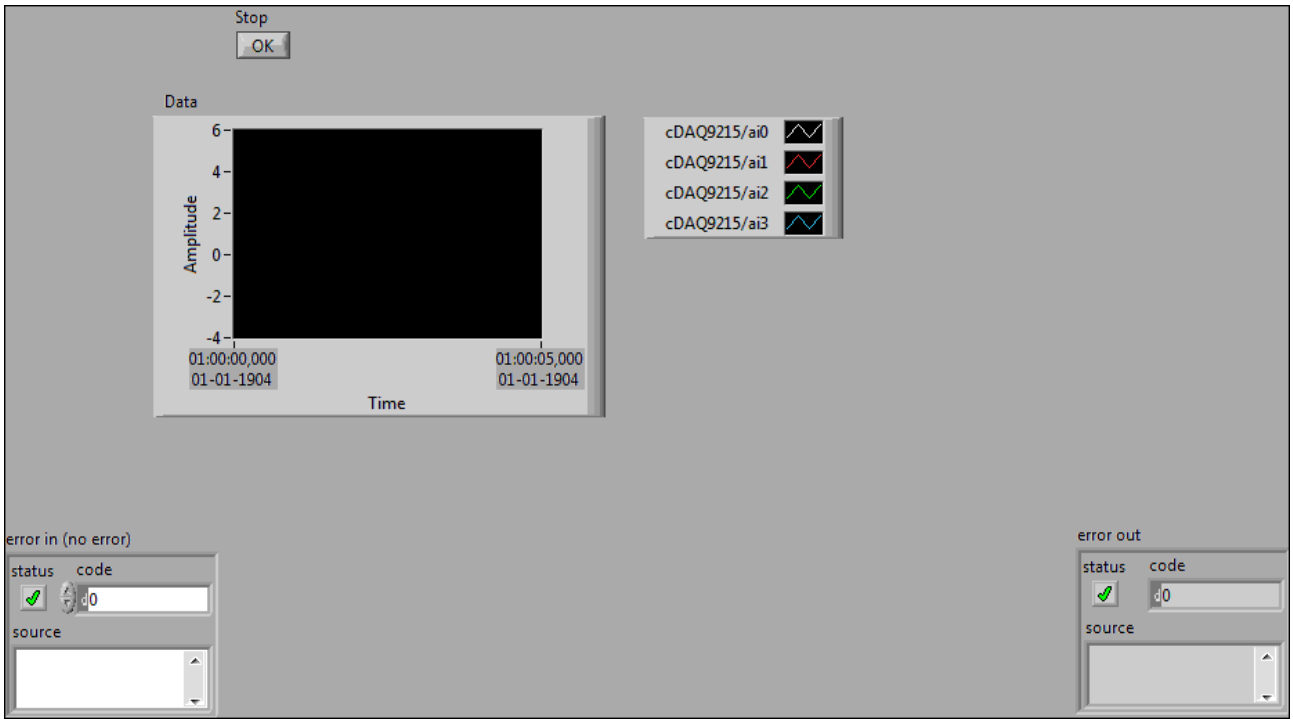


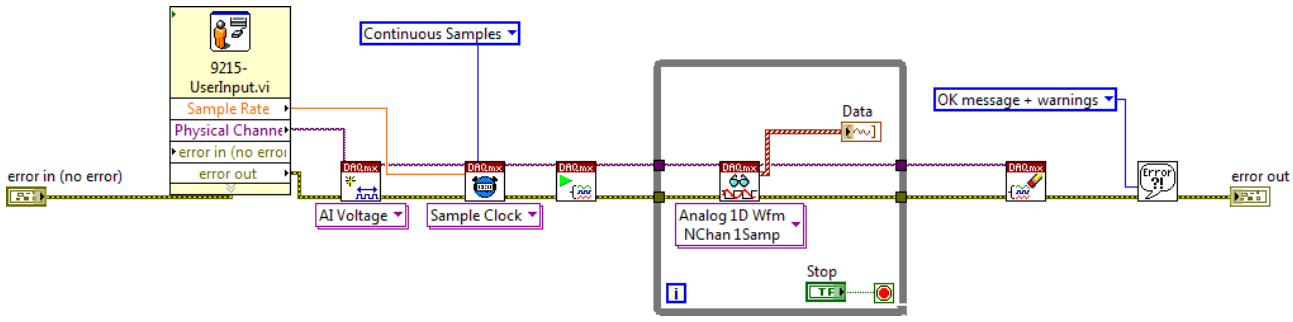



**9215.vi**


Stop
9215
Data
error in (no error)
error out


Stop
OK
Data
Amplitude
6
4
2
0
-2
-4
01:00:00,000
01-01-1904
01:00:05,000
01-01-1904
Time
cDAQ9215/ai0
cDAQ9215/ai1
cDAQ9215/ai2
cDAQ9215/ai3
error in (no error)
status
code
0
source
error out
status
code
0
source


9215-
UserInput.vi
Sample Rate
Physical Channe
error in (no erro
error out
Continuous Samples
AI Voltage
Sample Clock
Data
Analog 1D Wfm
NChan 1Samp
Stop
OK message + warnings
error in (no error)
error out

9215-UserInput.vi