

# Programmieren in LabVIEW<sup>1</sup>

## Inhaltsverzeichnis

<b>WAS IST LABVIEW?</b> .....	<b>3</b>
<b>KURZE HISTORIE VON LABVIEW</b> .....	<b>4</b>
<b>BENUTZEROBERFLÄCHE UND BLOCKDIAGRAMM</b> .....	<b>5</b>
<b>TERMINALS, KNOTEN UND DRÄHTE</b> .....	<b>6</b>
<b>ABKÜRZUNGEN - SHORTCUTS</b> .....	<b>10</b>
<b>LABVIEW VERSUCHT MITZUDENKEN</b> .....	<b>10</b>
<b>PROGRAMMIEREN IST EINE KUNST – PROGRAMMIEREN IST EIN HANDWERK</b> .....	<b>11</b>
<b>ABSPEICHERN UND EINLESEN VON PROGRAMMEN</b> .....	<b>13</b>
<b>ERSTELLUNG VON ANWENDUNGSPROGRAMMEN</b> .....	<b>13</b>
<b>FEHLERSUCHE, DEBUGGING</b> .....	<b>14</b>
<b>ERSTELLEN VON EIGENEN SUBVPS</b> .....	<b>16</b>
<b>DIE VERSCHIEDENEN GRUNDDATENTYPEN IN LABVIEW</b> .....	<b>17</b>
<b>PROGRAMMSTRUKTUREN</b> .....	<b>20</b>
DIE FOR-SCHLEIFE UND DIE WHILE SCHLEIFE.....	20
SHIFT-REGISTER / SCHIEBEREGISTER.....	22
LOKALE VARIABLE IN LABVIEW.....	24
WAS SIND RACE-CONDITIONS?.....	26
GLOBALE VARIABLE.....	27
ENTSCHEIDUNGEN (ALTERNATIVEN): IF ... THEN ... ELSE ... ODER CASE-STRUKTUR.....	28
ABLAUFSTRUKTUR ODER AUCH: SEQUENZ.....	30
FORMELKNOTEN.....	32
AUSDRUCKSKNOTEN.....	33
MATHSCRIPT-KNOTEN.....	33
DIALOG-BOXEN.....	34
DIE EVENT-STRUKTUR.....	35
FELDER.....	37
FUNKTIONEN ZUR BEARBEITUNG VON FELDERN.....	39
POLYMORPHISMUS.....	44
EIGENE POLYMORPHE VIS PROGRAMMIEREN.....	44

<sup>1</sup>Skript für die LV „Programmieren in LabVIEW“ (WS 07/08)

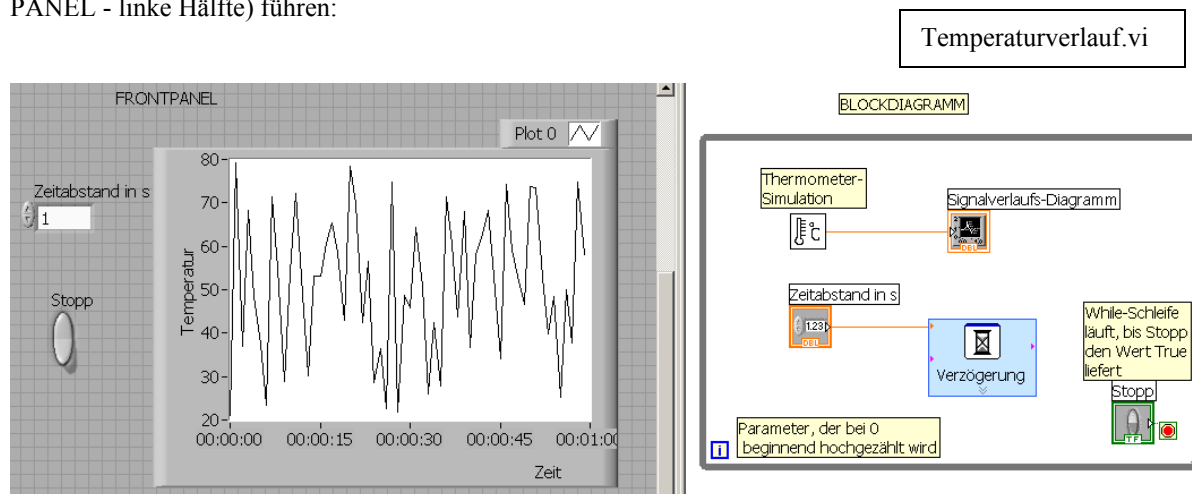
Dr. Blersch, TU Berlin, Fakultät II, Physikalische Institute

CLUSTER.....	46
<b>SIGNALVERLAUFS-DIAGRAMME UND SIGNALVERLAUFS-GRAPHEN.....</b>	<b>49</b>
SKALIERUNG VON CHARTS UND GRAPHEN.....	51
DIE LEGENDE.....	51
DIE PALETTE.....	51
<b>EINFACH- UND MEHRFACH-PLOT-GRAPHEN.....</b>	<b>52</b>
<b>EINFACH- UND MEHRFACH-PLOT-XY-GRAPHEN.....</b>	<b>53</b>
<b>ATTRIBUTKNOTEN ZUM STEuern VON BEDIENELEMENTEN UND DIAGRAMMEN.....</b>	<b>54</b>
<b>ERZEUGUNG UND VERÄNDERUNG VON ZEICHENKETTEN (STRINGS).....</b>	<b>56</b>
<b>EIN- AUSGABE IN DATEIEN.....</b>	<b>59</b>
SCHREIBEN VON MESSDATEN IN EINE DATEI.....	60
DAS WAR NUR DER ANFANG VOM ANFANG.....	61
<b>EINIGE WORTE ÜBER WAVEFORMS (ZU DEUTSCH: SIGNALVERLÄUFE).....</b>	<b>62</b>
<b>EXPRESS-VIS.....</b>	<b>65</b>
<b>MESSEN MIT LABVIEW.....</b>	<b>67</b>
DER GPIB-BUS.....	67
DAQMX.....	69
EINFACHE MESSPROGRAMME ERSTELLEN MIT DAQ-ASSISTENT.....	72
A) SIGNALE ERFASSEN: SPANNUNG EINLESEN.....	73
B) SIGNALE ERFASSEN: DIGITALE SIGNALE ERFASSEN.....	77
C) SIGNALE ERFASSEN: ZÄHLER AUSLESEN.....	78
D) SIGNALE ERZEUGEN: ANALOGE AUSGABE EINER SPANNUNG.....	80
E) SIGNALE ERZEUGEN: DIGITALE AUSGABE AN PORT0.....	81
E) SIGNALE ERZEUGEN: AUSGABEIMPULS EINES ZÄHLERS.....	83
WAS KOMMT NACH DEM DAQ-ASSISTENTEN?.....	84
ERFASSUNG VON N MESSWERTEN.....	86
AUSGABE VON N SPANNUNGSWERTEN.....	87
GETRIGGERTER MESSDATENERFASSUNG.....	88
<b>EINBINDEN VON FREMDEM PROGRAMMCODE.....</b>	<b>89</b>
1. ANFORDERUNG: AUFRUF EINES EXTERNEN PROGRAMM.....	89
2. ANFORDERUNG: EINBINDEN EINER DLL-DATEI.....	91
3. ANFORDERUNG: VERWENDUNG VON SELBST GESCHRIEBENEM C-PROGRAMMCODE.....	93
<b>LABVIEW UND DAS INTERNET.....</b>	<b>95</b>
VERÖFFENTLICHUNG VON VIS IM INTERNET.....	95
AUSTAUSCH VON LABVIEW-DATEN IM INTERNET.....	97
DIODENKENNLINIENMESSUNG IM INTERNET.....	100

## Was ist LabVIEW?

LabVIEW ist eine Programmierumgebung, mit der man effizient und ökonomisch Programme zur Steuerung von Messgeräten und zur Erfassung und Verarbeitung von Messdaten erstellen kann. Eine große Zahl vorgefertigter Funktionen erspart Ihnen die Mühe, diejenigen Räder, die andere schon lange vor Ihnen erfunden haben, noch einmal erfinden zu müssen. Das, was am meisten an LabVIEW verblüfft, ist die graphische Programmiersprache namens „G“, die es erlaubt, Programme zu schreiben, die besser strukturier- und dokumentierbar sind, als textbasierende (z.B. in C oder Pascal) geschriebene Programme. Die Grundidee dieser graphischen Programmierung wurde von einer Kursteilnehmerin einmal so zusammengefasst: „LabVIEW ist wie Lego!“. Ganz so einfach ist es sicher nicht, dennoch gilt, dass LabVIEW-Programme leichter erstellt, verbessert, erweitert und verstanden werden können, als textorientierte Programme. LabVIEW ist übrigens die Abkürzung für „**L**aboratory **V**irtual **I**nstrument **E**ngineering **W**orkbench.

Angenommen, ein Meteorologe will mit LabVIEW ein Programm schreiben, das die Aussentemperatur einlesen und in einem Fenster graphisch darstellen soll. Der Zeitabstand zwischen zwei aufeinanderfolgenden Messungen soll einstellbar sein. Dann könnte dies mit Hilfe von LabVIEW zu folgender Benutzeroberfläche (FRONT-PANEL - linke Hälfte) führen:



Das zugehörige Programm (BLOCKDIAGRAM) – in „graphischer Schreibweise“ – finden Sie oben (rechte Hälfte). Das darin enthaltene VI Thermometersimulation, simuliert die Temperaturerfassung. Sobald alles nach Wunsch läuft, wird es ersetzt durch die echte Temperaturerfassung per A/D-Wandler, die vielleicht ein anderer Programmierer erstellt hat.

LabVIEW enthält eine große Anzahl von Funktionen, die speziell im Bereich der Meßdatenerfassung und Meßdatenverarbeitung benötigt werden, in Bereichen also, die in der experimentellen Physik von großer Bedeutung sind. Natürlich können Sie mit diesem Programm auch Daten weiterverarbeiten, die nicht mit LabVIEW erfaßt wurden. LabVIEW kann ebenfalls zur Simulation physikalischer und vieler anderer Phänomene eingesetzt werden. Last but not least kann man sich mit LabVIEW viele Aspekte und Möglichkeiten der Messdatenverarbeitung sehr gut veranschaulichen. Und : LabVIEW macht mehr Spass, als jede andere Programmiersprache! Vorbemerkung: weil das Skript sonst zu langweilig und mühselig geworden wäre, gibt es keine genaue Erläuterung aller Menüpunkte des Programms – die meisten darunter sind intuitiv erfassbar. Wenn sich Fragen ergeben: verwenden Sie die LabVIEW-Hilfe, probieren Sie, fragen Sie Ihre Kollegen, fragen Sie in der Vorlesung oder in den Übungen oder lesen Sie in den verschiedenen Handbüchern nach! Dieses Skript soll die wichtigsten Lab-

VIEW-typischen Probleme und Denkweisen erläutern und darstellen und Ihnen damit den eigenen Einstieg erleichtern. Bearbeiten Sie unbedingt auch die Übungsblätter zu dieser LV und sehen Sie sich das LabVIEW Tutorial an, das beim Aufruf von LabVIEW angeboten wird. Etliche der hier aufgeführten Beispiele stammen von dort. Wenn Sie ernsthaft vorhaben, ein Projekt mit LabVIEW zu bearbeiten, oder wenn Sie vermuten, daß es für Ihr Problem eigentlich schon eine (Teil-)Lösung geben müsste, sollten Sie unbedingt im Internet nachsehen. Eine wichtige Adressen ist: <http://ni.com/devzone>. Dort gibt es unter den Stichpunkten „Development Library“ und „Example Code“ tausende von kostenlosen Quellcode-Beispielen. Des weiteren gibt es ein deutschsprachiges LabVIEW-Forum, wo Sie ebenfalls hilfsbereite LabVIEW-begeisterte Menschen finden können: <http://www.labview-forum.com/>.

**Literatur zu LabVIEW:** Ausser den Manuals zu LabVIEW von National Instruments und der Hilfe in LabVIEW gibt es inzwischen sehr viele Bücher zu allen möglichen Anwendungsbereichen von LabVIEW. Für diese Lehrveranstaltung habe ich hauptsächlich das folgende Buch verwendet:

Jeffrey Travis, LabVIEW FOR EVERYONE, Prentice Hall

In deutscher Sprache gibt es u.a.: Wolfgang Georgi, Ergun Metin, Einführung in LabVIEW, Hanser Verlag

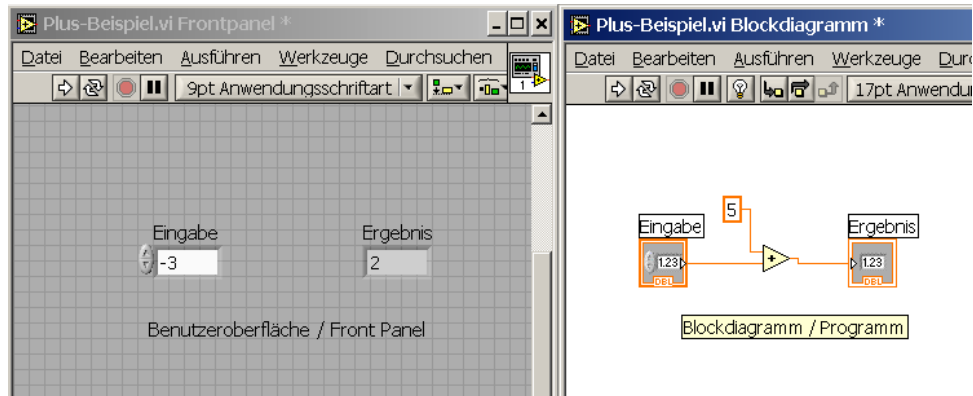
Wenn Sie sich LabVIEW auf Ihrem privaten Rechner installieren wollen, können Sie die Studentenversion von LabVIEW erwerben. Am besten, Sie kaufen für 49,50 € das folgende Buch, das im April 2007 herausgekommen ist: Bernward Mütterlein: *Handbuch für die Programmierung mit LabVIEW. Mit Studentenversion LabVIEW 8 (Gebundene Ausgabe)*, 1. Auflage April 2007

## Kurze Historie von LabVIEW

Die erste Version von LabVIEW gab es im Jahre 1987. D.h. LabVIEW ist heute grade erst 20 Jahre alt. Als Entwickler des LabVIEW-Konzepts werden Jeff Kodofsky und Jim Truchard genannt, aber natürlich ist das LabVIEW von heute (grade eben kam die Version 8.50 raus) das Ergebnis der Arbeit vieler Leute. Da LabVIEW auf einer graphischen Oberfläche beruht, waren die ersten Versionen für Mac-Rechner bestimmt. Erst seit 1995 gibt es auch LabVIEW für Windows. Seither hat sich LabVIEW zu einem Standard im Bereich von Messdatenerfassung und Messdatenverarbeitung entwickelt. Auch in der Entwicklung, Produktion und Qualitätskontrolle in der Industrie werden mehr und mehr qualifizierte LabVIEW-Programmierer eingesetzt und gesucht, weil mit dieser Software in kürzester Zeit sehr stabile Anwendungsprogramme geschrieben werden können.

## Benutzeroberfläche und Blockdiagramm

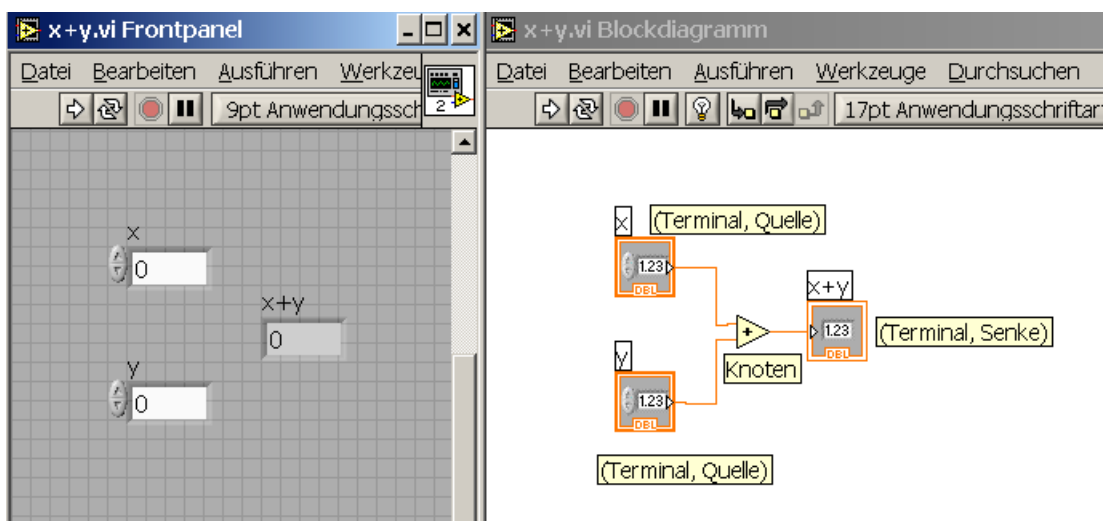
Nach dem Start von LabVIEW erhält man zwei leere Fenster: Benutzeroberfläche (Front Panel) und Blockdiagramm. Die Benutzeroberfläche ist die dem Anwender zugewandte Seite von LabVIEW, d.h. dies ist der Ort, an dem der Benutzer Parameter eingibt, die an das Programm und eventuell an die Geräte weiter gegeben werden und wo auch die Meßdaten (in Form von Tabellen oder Kurven) dargestellt werden können. Hinter dem „Front-



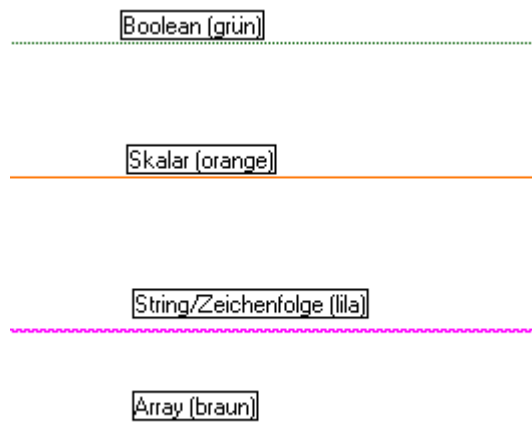
Panel“ versteckt sich das Programm, das „Blockdiagramm“ genannt wird. Dieses Blockdiagramm – obwohl es wie eine Grafik aussieht – **ist** das Programm. Die Zusammenfassung von Panel und Blockdiagramm heißt „Virtuelles Instrument“, kurz: „VI“. Im Normalfall ist das Blockdiagramm selbst wieder aus einer Reihe von Sub-Programmen oder Sub-VIs aufgebaut, die auch wieder Unter-VIs enthalten können, usw. Im Unterschied zu herkömmlichen Programmiersprachen erfolgt der Datenfluß in LabVIEW über „Drähte“, die die Ein- und Ausgänge der SubVI's auf der Diagrammseite miteinander verbinden. (Bei klassischen Programmiersprachen erfolgt der Datenfluß über die Parameterübergabe.)

## Terminals, Knoten und Drähte

Immer wenn auf der Panelseite ein Ein- oder Ausgabeobjekt plaziert wird, entsteht im Blockdiagramm eine Entsprechung dazu – ein sogenanntes „Terminal“. Ein Eingabe-Terminal (ein „digital control“) kann als „Datenquelle“ verstanden werden – aus ihm kommen Daten raus, während man sich ein Ausgabeterminal (ein „digital indicator“) als „Datensenke“ vorstellen kann – in ihm verschwinden Daten. Worin unterscheiden sich im folgenden Programm die Datenquellen und die Datensenken? Wodurch wird in einem textbasierenden Programm die Reihenfolge der Abarbeitung der Befehle festgelegt? Wie, stellen Sie sich vor, könnte das bei LabVIEW gehen? Es gibt einen Modus (Highlight-Funktion/Glühbirne) bei der Programmausführung, bei dem man sehen kann, wie kleine Kügelchen (in deren Innern sich die Daten befinden ;-)) aus den Datenquellen herauskommend, die Drähte entlang laufen, um schließlich in einer Datensenke am Drahtende zu verschwinden. Und was passiert dann?



Ein „Knoten“ ist ein „Programmausführungselement“ – er entspricht in klassischen Programmen einem Operator oder einer Subroutine. Wie in jeder anderen Programmiersprache auch müssen Werte/Parameter übergeben und (daraus berechnete) andere Werte zurückgegeben werden. Simple Beispiel: der Plus-Operator im obigen Beispiel ist ein Knoten mit zwei Eingabeparametern und einem Rückgabeparameter.


















Ein „Draht“ ist ein Datenpfad zwischen einem Eingabe-Terminal und einem Ausgabe-Terminal. Drähte haben verschiedene Farben und Dicken, je nach dem Datentyp, der darüber geschickt wird. Links sehen Sie einige Beispiele (im Skript allerdings ohne die Farben). Eine Boolesche Variable kann nur zwei Werte annehmen: TRUE oder FALSE. Ganze Zahlen sind blau, Skalare („Zahl mit Komma“) sind orange. Ein String ist eine Zeichenkette aus Buchstaben, Ziffern und Sonderzeichen, wie %, \$, >, §. Arrays bestehen aus lauter gleichartigen Elementen. Ein Array-Draht ist dicker, als ein normaler Draht, er hat die Farbe seiner Elemente

Im folgenden Bild finden Sie Angaben zur Darstellung der verschiedenen Zahlentypen in LabVIEW:

### Numeric Data Types Table

The following table displays the [numeric data types](#) available in LabVIEW. LabVIEW [stores](#) each data type in a different way.

Terminal	Numeric Data Type	Bits of Storage on Disk	Approximate Number of Decimal Digits	Approximate Range on Disk
	Single-precision, floating-point	32	6	Minimum positive number: 1.40e-45 Maximum positive number: 3.40e+38 Minimum negative number: -1.40e-45 Maximum negative number: -3.40e+38
	Double-precision, floating-point	64	15	Minimum positive number: 4.94e-324 Maximum positive number: 1.79e+308 Minimum negative number: -4.94e-324 Maximum negative number: -1.79e+308
	Extended-precision, floating-point	128	varies from 15 to 20 by platform	Minimum positive number: 6.48e-4966 Maximum positive number: 1.19e+4932 Minimum negative number: -6.48e-4966 Maximum negative number: -1.19e+4932
	Complex single-precision, floating-point	64	6	Same as single-precision, floating-point for each (real and imaginary) part
	Complex double-precision, floating-point	128	15	Same as double-precision, floating-point for each (real and imaginary) part
	Complex extended-precision, floating-point	256	varies from 15 to 20 by platform	Same as extended-precision, floating-point for each (real and imaginary) part
	Byte signed integer	8	2	-128 to 127
	Word signed integer	16	4	-32,768 to 32,767
	Long signed integer	32	9	-2,147,483,648 to 2,147,483,647
	Quad signed integer	64	18	-1e19 to 1e19
	Byte unsigned integer	8	2	0 to 255
	Word unsigned integer	16	4	0 to 65,535
	Long unsigned integer	32	9	0 to 4,294,967,295
	Quad unsigned integer	64	19	0 to 2e19
	128-bit time stamp	<64.64>	15	Minimum time (in seconds): 5,4210108624275221700372640043497e-20 Maximum time (in seconds): 9,223,372,036,854,775,808

Sie können Zahlen in verschiedenen Basissystemen (rechter Mausklick!) darstellen:



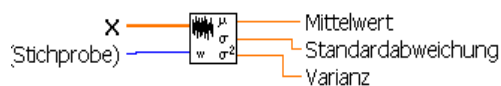
Im obigen Beispiel wird die Zahl binär eingegeben und dezimal dargestellt. Warum ist die Zahlausgabe eine negative Zahl? Experimentieren Sie selbst mit verschiedenen Darstellungen. Dass es sich um eine binäre bzw. dezimale Darstellung handeln kann man sehen. Versuchen Sie mal in der binären Eingabe eine 7 einzugeben.





Klassische Programmiersprachen (Fortran, Pascal, C) bestehen i. a. aus Unterprogrammen, Prozeduren genannt. Prozeduren sind selbst wiederum Programmstücke, an die Parameter übergeben werden können, um aus diesen Parametern „etwas“ machen und dieses „etwas“ schliesslich an die aufrufende Instanz zurückgeben. Beispiel: einer Sortierprozedur wird ein unsortiertes Feld von Zahlen als Parameter übergeben, das dann innerhalb dieser Prozedur der Grösse nach geordnet und anschliessend in geordnetem Zustand an die aufrufende Prozedur zurück gegeben wird.

Wie jedes anständige (d.h. gut überlegte und gut strukturierte) textbasierende Programm aus textbasierenden Prozeduren aufgebaut wird, wird auch ein LabVIEW-Programm aus SubVI's aufgebaut. Einem SubVI entspricht ein mit einem treffenden Symbol gekennzeichnetes Kästchen, auch als Icon bezeichnet. Das Kästchen hat Ein- und Ausgänge, sogenannte „Connectors“ („Anschlüsse“), mit deren Hilfe die Ein- und Ausgabeparameter übergeben werden können. Üblicherweise liegen die Eingänge auf der linken Seite des Kästchens und die Ausgänge auf der rechten Seite. LabVIEW stellt dem Benutzer eine große Anzahl SubVI's zur Verfügung. So muß man zur Berechnung der Standardabweichung selbstverständlich kein eigenes VI mehr programmieren, sondern kann auf das entsprechende VI von LabVIEW zurückgreifen:



**Standardabweichung und Varianz**  
[Std Deviation and Variance.vi]

Das VI links hat fünf Anschlüsse (Connectors): zwei Eingabe- und drei Ausgabeanchlüsse, jeweils für den Array X der Meßwerte, die Stichprobe (als Eingabe) und den Mittelwert, die Standardabweichung und die Varianz (als Ausgabe).



die Varianz (als Ausgabe).

Ein SubVI kann bis zu 28 Anschlüsse haben: darüber freut sich jeder Augenarzt... Aber was macht man, wenn man trotz Augenarzt und neuer Brille ein SubVI mit 56 Anschlüssen basteln muss? Sobald Sie den Datentyp Cluster kennengelernt haben wissen Sie die Lösung und ersparen sich gleichzeitig auch den Gang zum Arzt.

**Aufgabe:** Versuchen Sie, sich einen Überblick über die von LabVIEW angebotenen SubVIs zu verschaffen. Sie werden es nicht schaffen, fangen Sie trotzdem an damit und überlegen Sie sich beim Surfen durch die einzelnen Menüs, warum es wohl in das jeweilige Untermenü gehört. Schalten Sie dabei die Kontext-Hilfe ein

Obwohl in LabVIEW schon eine sehr grosse Zahl von vorgefertigten SubVIs zu allen möglichen Aufgabengebieten vorhanden ist, ist es dennoch unumgänglich auch eigene SubVIs zu programmieren – dieser Schritt wird Ihnen demnächst gezeigt werden.

## Abkürzungen - Shortcuts

Nach kurzer Zeit stört es, wenn man bei sich häufig wiederholenden Aufgaben immer zuerst eine ganze Reihe von Menüpunkten durchklicken muß. Deshalb gibt es auch in LabVIEW sogenannte „Shortcuts“, Tastenkombinationen, mit denen Sie schneller ihr Ziel erreichen:

Strg+r - **R**un a VI

Strg+z – UnDo: letzten Schritt rückgängig machen

Strg+h - Kontexthilfe-Fenster an/aus

Strg+w - Aktives **W**indow schließen

Strg+b - **B**roken Wires entfernen

Strg+e - Zwischen Panel-Fenster und Diagram-Fenster hin und her springen.

Während des Verdrahtens kann man einen Fehler rückgängig machen, indem man die rechte Maustaste drückt. Wenn der Draht um Ecken geführt werden soll kann man ihn durch einen Mausklick an der betreffenden Stelle „festkleben“.

Objekte können dupliziert werden, wenn man nach dem Markieren die „Strg-Taste“ drückt und die Kopie mit der Maus dann an die gewünschte Stelle verschiebt.

**Aufgabe:** Üben Sie diese Shortcuts und Verdrahtungstricks in einem kleinen von Ihnen selbst erdachten Testprogramm.

## LabVIEW versucht mitzudenken



Die Entwickler von LabVIEW haben sich viele Gedanken darüber gemacht, wie der Anwender möglichst schnell und ohne viele Mausklicks das erreicht, was er erreichen will. Dazu zwei Beispiele. Je nachdem was der Anwender grade mit dem Cursor macht, denkt sich LabVIEW: jetzt will er einen Draht anschliessen, jetzt will er was verschieben, jetzt will er einen Wert eingeben und bietet ihm die entsprechende Funktion an. Damit diese Angebote kommen können muss im Tools-Fenster in der obersten Zeile die grüne Lampe leuchten. Wenn der Anwender sagt: ich will aber gar keinen Wert eingeben, ich will die Farbe ändern, dann muss er diese Mitdenk-Option ausschalten und anschliessend das entsprechende Werkzeug im Tools-Fenster von Hand auswählen.

Ein weiteres wichtiges Beispiel: der Programmierer will an einen VI-Anschluss ein Steuerungs- oder Anzeigeelement anschliessen. Er kann dann mit einem rechten Mausklick auf diesen Anschluss über den Menüpunkt „Erzeuge“ eine geeignete Konstante oder das passende Control- oder Indicatorelement erzeugen. Man erspart sich so viel Mühe beim Suchen und Klicken durch die Menüs.

## Programmieren ist eine Kunst – Programmieren ist ein Handwerk

Programmieren ist zuallererst sehr viel Arbeit. Der grösste Teil dieser Arbeit besteht meist darin, die Fehler auszumergen, die man selbst in das Programm eingebaut hat. Man könnte sagen, dass diese Fehler alle auf mangelnden Vorüberlegungen beruhen und darauf, dass man sich als Programmierer nicht richtig im Klaren darüber ist, welche Anforderungen man selbst (oder der Auftraggeber) eigentlich an die zu bearbeitende Aufgabe stellt.

Eine gute Regel beim Programmieren (zumindest bei etwas grösseren Aufgabenfeldern) lautet, zunächst ohne Rechner nur mit Bleistift und Papier, das Problem zu analysieren, das mit dem zu erstellenden Programm gelöst werden soll. Danach geht es darum, die grosse Aufgabe in kleinere Aufgabenblöcke usw. zu zerlegen (im Programmiererjargon heisst dieser Vorgang: Modularisierung). Kurz gesagt: erst meditieren, dann programmieren. Allerdings ist es illusionär zu glauben, dass mit einer gründlichen Vorbereitung alle Fehler vermieden werden können. Die gute Nachricht: der Rechner wird seinen Programmierer beharrlich und stur auf Denk- und Strukturfehler hinweisen, so lange, bis auch der letzte Fehler gefunden wurde .... was nie eintritt, zumindest nicht bei Programmen, die länger als 100 Zeilen sind. Es gibt einen alten Witz unter Programmierern: 1. Erstens: jedes Programm enthält mindestens 1 Fehler und zweitens: jedes Programm kann um eine Zeile gekürzt werden. Daraus ergibt sich direkt durch vollständige Induktion, dass selbst ein leeres Programm immer noch mindestens 1 Fehler enthält, das hat was Beunruhigendes ...

Programmieren verlangt weiterhin eine sorgfältige Lektüre der Manuals und der Beschreibungen, die zu einem Programmsystem gehören.

Wenn Sie vermeiden wollen, Ihre eigenen Programme nach drei Tagen nicht mehr zu verstehen verlangt dies insbesondere auch eine sorgfältige Dokumentation des eigenen Programms. Programme genau zu beschreiben, hat auch den Hintergrund, dass auch Kollegen mit diesem Programm arbeiten können und es entsprechend neuer Anforderungen erweitern und modifizieren können müssen.

Alles was oben gesagt wurde, gilt für jede Programmiersprache, also auch für LabVIEW. Für LabVIEW kommt noch eine weitere Möglichkeit des schlechten Programmierens hinzu: wenn Sie Ihre Sub-VIs so anordnen, dass die Verdrahtung um 100 Ecken führt kommen Sie sehr schnell zum berüchtigten „Spagetti-Code“.

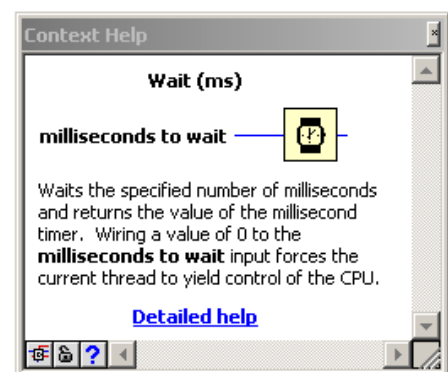
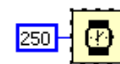
In LabVIEW gibt es zunächst zwei Arten von Hilfen:

### a) Kontexthilfe/Direkthilfe

In einem kleinen Fenster wird das jeweilige Objekt erläutert, auf das der Mauszeiger gerade zeigt:

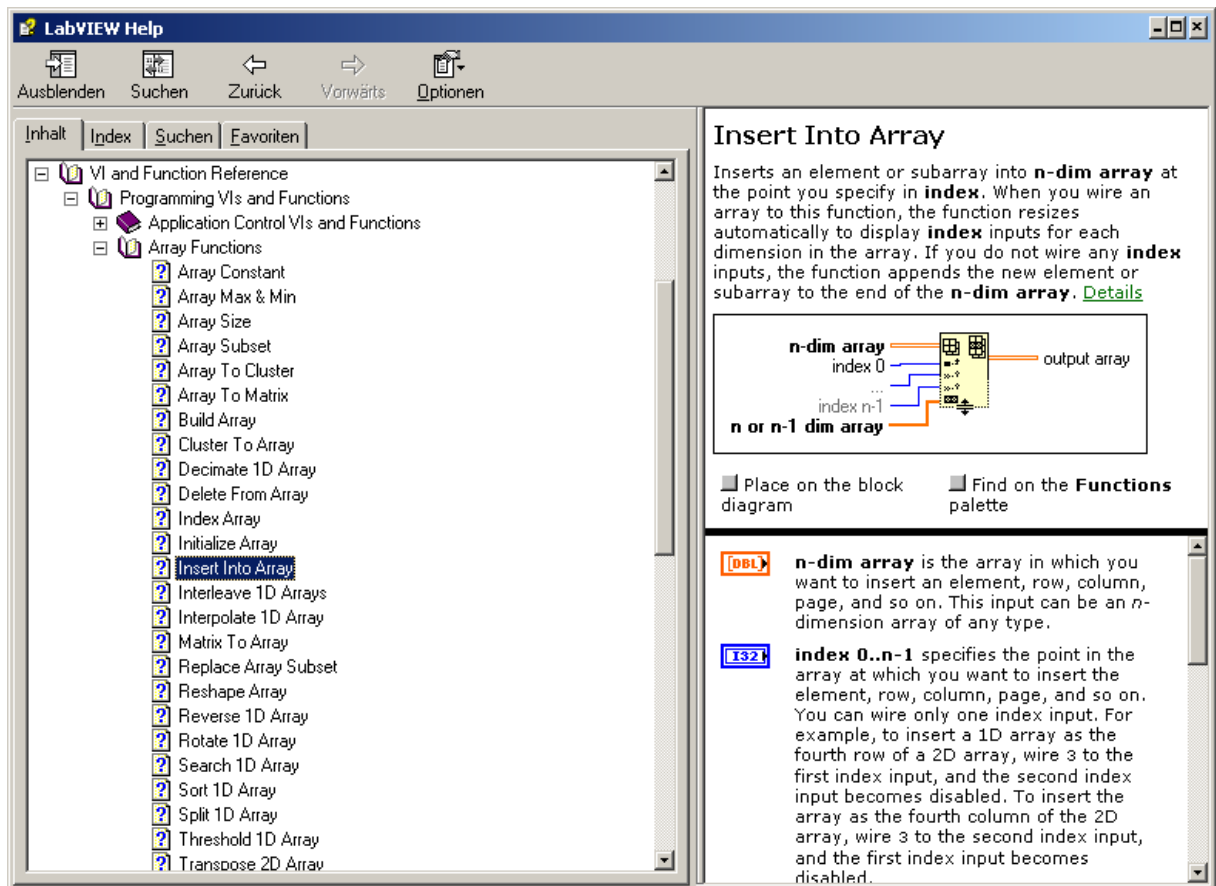
Diese Kontexthilfe-Fenster muss aktiviert werden z.B. durch Strg+h. Damit lassen sich sehr gut fremde Programme verstehen, wenn man sich über die Bedeutung der einzelnen Element noch nicht im Klaren ist.

Wie man auch die selbst programmierten SubVIs mit einer Kontexthilfe versieht wird demnächst erläutert.



## b) Die ausführliche Hilfe erreicht man über Strg+?

Hier erhält man Hilfe für alles – man muss nur Geduld haben und lange genug suchen und ausführlich und ausgiebig lesen. Ein Beispiel:



## c) Programmbeispiele (unter dem Hilfe-Menüpunkt)

Ganz sicher am meisten lernen werden Sie mit den **LabVIEW-Beispielen**. Dort werden Ihnen Hunderte von Beispielen aus den verschiedensten Bereichen bereitgestellt, die Sie zunächst untersuchen können nach dem Motto: „wie machen es eigentlich die anderen?“. Sie können diese Beispiele aber auch nehmen und, nachdem sie nach Ihren Bedürfnissen modifiziert wurden, in Ihre eigenen Programme einfügen.

## d) Hilfe im Internet

Schliesslich und endlich finden Sie noch die **Web-Ressourcen** zu LabVIEW. Dort findet man Produkte, Technische Unterstützung und die NI Developer Zone mit Diskussionsforen und Programmierbeispielen.

## Abspeichern und Einlesen von Programmen

Wie bei jedem anderen Anwendungsprogramm können die mit LabVIEW erzeugten Programm-Dateien abgespeichert werden. LabVIEW bietet dazu zwei Möglichkeiten. Einmal können die Dateien mit „Save“ in irgendeinem Verzeichnis abgelegt werden. Zum anderen kann man VI's in einer sogenannten VI-Library abspeichern. Dabei werden die Dateien komprimiert. VI-Librarys stellen sich gegenüber dem Betriebssystem als eine einzige Datei dar. Auf die darin enthaltenen einzelnen VI's kann man nur per LabVIEW zugreifen. Wenn eine solche Datei beschädigt wird, werden alle darin enthaltenen VI's unbrauchbar. Andererseits kann man diese Dateien leichter von einer Rechnerplattform auf eine andere Plattform transferieren. In Dialogboxen sehen Libraries wie Ordner aus. Die Namen dieser Bibliotheken enden mit „.lib“.

## Erstellung von Anwendungsprogrammen

Falls Sie mit LabVIEW-Programmierung Ihren Lebensunterhalt verdienen müssen, können Sie für Ihre Kunden reine Exe-Dateien erzeugen, die anschliessend auf dem Kundenrechner installiert werden. Dadurch bleibt der eigentliche Programmcode im Besitz des Programmierers und der Kunde benötigt für seine konkrete Anwendung keine LabVIEW-Entwicklungsumgebung (die in der Tat sehr teuer ist). Sie können darüber hinaus auch Vorgaben für den Windows-Installer geben, damit das Programm auf dem Zielrechner ordnungsgemäss installiert wird. Diese Anwendungsprogramme für den Kunden werden mit unserer LabVIEW-Umgebung im PC-Pool Physik mit dem Menüpunkt Tools/Build Executable erzeugt.

Das oben gesagte gilt insbesondere auch für den Fall, dass Sie auf einem Rechner mit LabVIEW-Programmierungsumgebung im Büro Ihr Programm entwickeln und testen und dieses schliesslich als exe-Datei auf dem Rechner im Labor einsetzen.

## Fehlersuche, Debugging

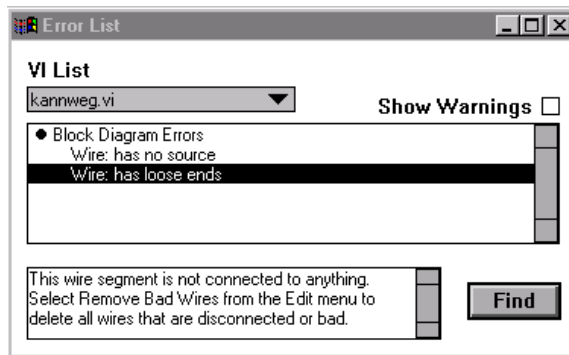
Programme können aus zwei Gründen nicht funktionieren: erstens kann die Syntax falsch sein, dann versteht der Rechner gar nicht, was er machen soll und weigert sich schlicht zu arbeiten: das Programm ist nicht lauffähig. Immerhin sagt LabVIEW noch: „an dr und der Stelle ist whrscheinlich ein Fehler“. Zweitens aber kann die Syntax richtig sein, der Rechner weigert sich dann also nicht mehr zu arbeiten, aber was er macht ist nicht das, was er machen soll: das ist leider der Normalzustand.



Wenn das Programm nicht lauffähig ist, wenn es also Syntaxfehler enthält, kann man das daran erkennen, daß der „Run-Pfeil“ zerbrochen (links) ist: Ein VI hat normalerweise so-



lange einen zerbrochenen Pfeil, bis es vollständig verdrahtet ist. Ein nicht zerbrochener Pfeil (rechts) deutet also nur darauf hin, daß das Programm ablaufen kann, daß keine Syntaxfehler vorliegen. Er bedeutet auf gar keinen Fall, daß das Programm macht, was es machen soll! Daß der Pfeil zerbrochen ist, kann

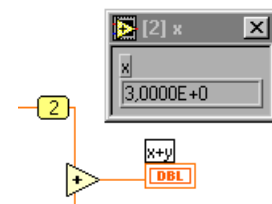


auch daran liegen, daß von einem mißlungenen Verdrahtungsversuch noch kleine Drahtstücke im Diagramm herum liegen, die entfernt werden müssen. Dies kann man mit dem Kommando „Remove Broken Wires“ aus dem Edit-Menü (oder Strg+b) erledigten. Des weiteren kann man sich mit „Show Error List“ die von LabVIEW erzeugte Fehlerliste ansehen und eventuell mit dem „FIND“-Button anzeigen lassen, welches Objekt den Fehler verursacht.

Wenn der Run-Button nicht zerbrochen, das Programm also lauffähig ist, aber nicht das macht, was es machen soll, muss man als Programmierer das Programm Schritt für Schritt untersuchen, um die Stelle zu finden, die für den Fehler verantwortlich ist. Dazu gibt es in der LabVIEW-Programmierungsumgebung eine sehr gute Unterstützung:

### a) Entnahme einer Probe

Wenn Sie den Verdacht haben, daß über ein Drahtstück nicht die richtigen Werte laufen, können Sie dies nachprüfen, indem Sie (im Run-Modus!) den Draht mit der rechten Maustaste anklicken und eine „Probe“ (Tools-Palette) entnehmen: Sie erhalten ein Fenster mit einer Nummer in der Nähe des Drahtes, in dem die aktuellen Werte angezeigt werden!

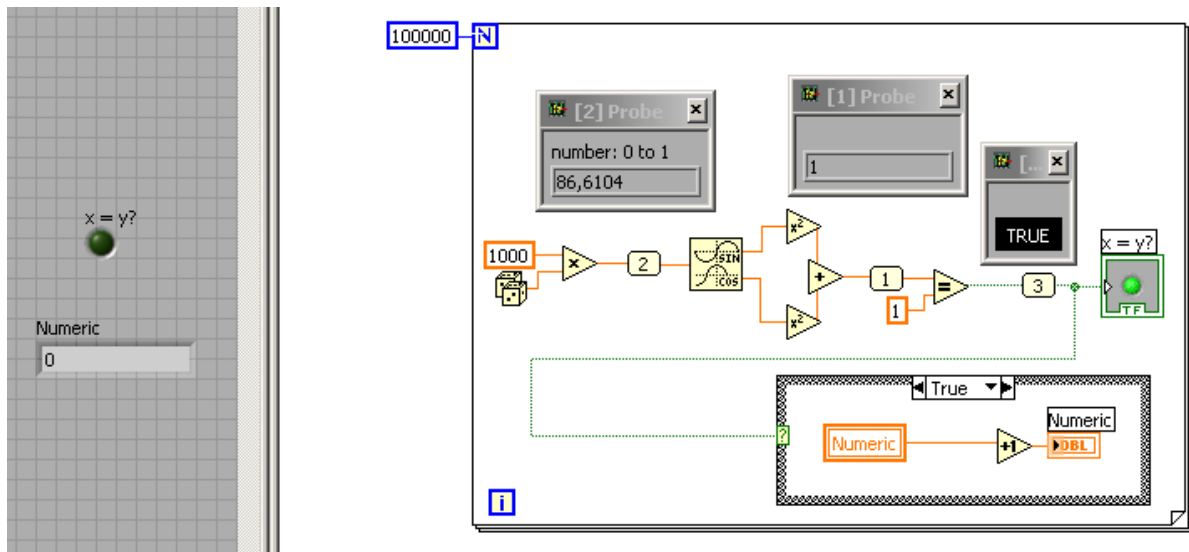


### b) Schritt-für-Schritt



Mit den drei links dargestellten Buttons können Sie sich in Einzelschritten durch das Programm bewegen und so „Schritt-für-Schritt“ untersuchen, welche Zustände und Werte sich ergeben und ob diese Zustände so sind, wie Sie es sich beim Programmentwurf gedacht haben.

**Kleine Aufgabe dazu:** „Beweisen“ Sie mit LabVIEW, dass die Formel  $\sin^2x + \cos^2x == 1$  für alle reellen Zahlen  $x$  gültig ist. (Warum 2 Gleichheitszeichen?) Die Bedingung „für alle  $x$ “ schafft LabVIEW zwar nicht, aber „sehr viele  $x$ “ reicht vielleicht auch???



Wenn Sie das obige Programm ablaufen lassen, werden Sie feststellen, dass die grüne Lampe immer leuchtet. Wirklich? Wirklich misstrauische Leute zählen mit und erwarten, dass in der Numeric-Ausgabe die Zahl 99999 steht. Leider liefert unser Experiment das Ergebnis, dass die Gleichheit nur in ca 75% der Fälle eintritt. Hat die Mathematik unrecht oder LabVIEW? Wie müsste man das Programm abändern, um die gewünschte Zahl zu erreichen? Durchlaufen Sie das Programm „Step by Step“. Bemerkung: das VI enthält einige Dinge, die noch nicht behandelt wurden: die while-Schleife, eine if-else-Alternative und eine lokale Variable Numeric, die eine Kopie des Anzeigeelements Numeric ist ... all das wird demnächst hier behandelt, machen Sie sich also keine Sorgen, wenn Sie das obige Programm noch nicht 100%ig verstanden haben.

### c) Verzögerung

Wenn alles zu schnell abläuft, können Sie Verzögerungsglieder („Timing“ im Funktionsmenü) einbauen. Wenn Sie die kleine Glühlampe (Highlight Execution) anklicken, sehen Sie in Zeitlupe wie die Daten als kleine Kugeln von den Quellen zu den jeweiligen Senken laufen – spätestens jetzt müßte ihnen allmählich eine Erleuchtung kommen zu den möglichen Fehlern in Ihrem Programm.



### d) Breakpoints

Weiterhin haben Sie noch die Möglichkeit Breakpoints zu setzen. Breakpoints sind Punkte, an denen das Programm anhält, so daß man in Ruhe die zu diesem Zeitpunkt berechneten Werte untersuchen kann. Einzelheiten dazu finden Sie im Handbuch!



Breakpoints finden Sie im Tools-Menü. Sie können damit Breakpoints setzen und bestehende Breakpoints wieder löschen.

## Erstellen von eigenen SubVI's

Erstellen Sie ein VI, das den Mittelwert von drei Werten  $x$ ,  $y$  und  $z$  berechnet. Angenommen, Sie benötigen



diese Funktion sehr oft, dann ist es wünschenswert, dafür ein eigenes SubVI zu schaffen, das immer wieder eingesetzt werden kann. Wie geht das? Die eigentliche Arbeit, das Programm zu erstellen und zu testen, haben Sie schon gemacht. Die Frage bleibt nur noch: wie macht man daraus ein kleines Kästchen (einen speziellen Legobaustein eben) mit drei Eingängen und einem Ausgang, das im Laufe der weiteren Arbeit immer wieder eingesetzt werden kann? Die Antwort auf diese Frage umfasst drei Schritte:

**Erster Schritt:** *Entwurf eines aussagekräftigen Icons.* Dazu klickt man paneelseitig mit der rechten Maustaste in



das rechts oben platzierte Standard-Icon. Sie finden dort eine einfache Palette von Werkzeugen um ein neues Icon zu zeichnen. Denken Sie daran aussagekräftige Icons zu entwerfen! Nicht umsonst heisst es: ein gutes Bild ist mehr wert als 1000 Worte – das bestätigt Ihnen jeder Comic-Leser!

**Zweiter Schritt:** *Zuordnung der Anschlüsse.* Zuerst im Panel auf die Drahtspule klicken. Durch einen Klick mit der rechten Maustaste in das VI-Symbol in der rechten oberen Panelecke kann man sich eine Tabelle verschiedener Anschlüsse/Connectors anzeigen lassen und darin den geeigneten Typ auswählen. Die Zuordnung der Ein-Ausgabeobjekte aus dem Panel zu den jeweiligen Anschlüssen erfolgt in einem Dreisprung: zuerst wird der Anschluß angeklickt, dann das diesem Anschluß zuzuordnende Objekt und schließlich irgendwo daneben. Die Anschlußfarbe wechselt dabei von schwarz zur gedämpften Farbe des Datentyps und endet mit der korrekten Farbe. Wenn Sie dann die Help-Ebene einschalten (<Ctrl><h>) erhalten Sie ein Bild, in dem auch schon die Namen der Terminals eingetragen sind (s.u.). Wenn Sie noch weitere Anschlüsse benötigen kann man mit „Show Connectors/Patterns“ weitere Anschlüsse hinzufügen.



**3-MW.vi**

Dieses VI berechnet den Mittelwert der 3 Eingänge  $x_1$ ,  $x_2$  und  $x_3$

**Dritter Schritt:** *Dokumentation.* In Datei/VI-Einstellungen gibt es eine Kategorie „Dokumentation“ und einen Bereich VI Beschreibung. Dort sollten Sie unbedingt in kurzer und prägnanter Form den Dokumentationstext eintragen.

Nun sollten Sie bei eingeschalteter Direkthilfe etwas der Abbildung links Entsprechendes sehen.

**Vierter Schritt:** *Einfügen selbst erstellter Vis:* Gehen Sie im Funktionsmenü im Blockdiagramm zum Unterpunkt „Select a VI“ und klicken Sie sich dann zu Ihrem VI durch.

**Anmerkung:** Was passiert, wenn das selbe SubVI an mehreren Stellen im Oberprogramm aufgerufen wird? Kann ein VI sich selbst aufrufen? Denken Sie daran, wie elegant man in C z.B. die Fakultätsfunktion programmieren kann. Die Antwort für LabVIEW ist nicht so einfach und soll erst später behandelt werden.

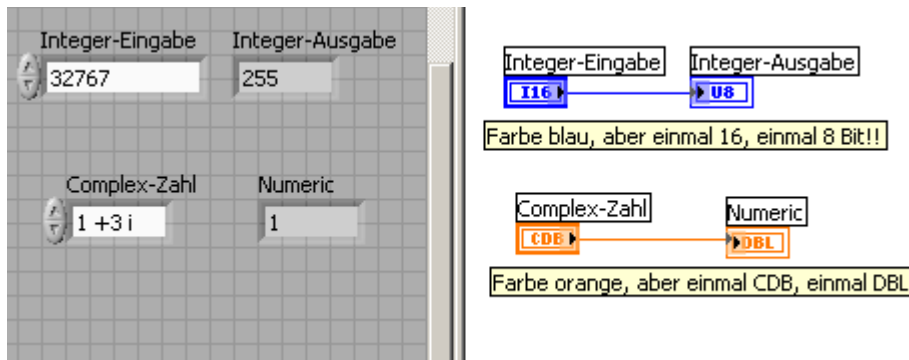


# Die verschiedenen Grunddatentypen in LabVIEW

## a) Numerische Steuer- und Anzeigeelemente

Damit lassen sich numerische Werte eingeben, bzw. anzeigen.

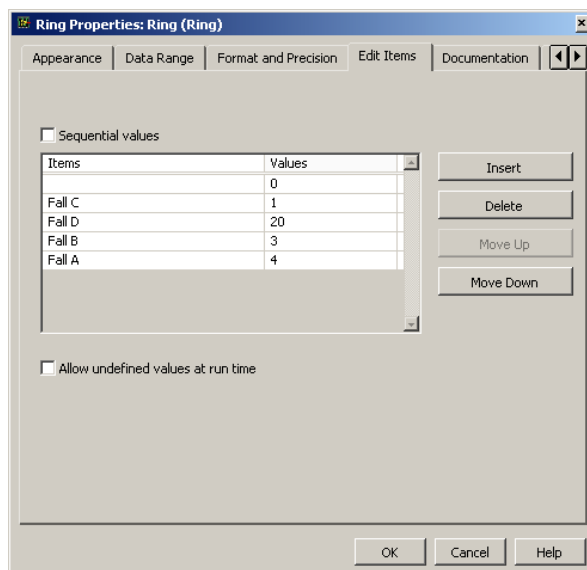
In der Tabelle der numerischen Datentypen von LabVIEW auf Seite 7 können Sie die verschiedenen Datentypen erkennen. Sie reicht vom einfachen Byte (I8 / U8), das zur Darstellung 8 Bits benötigt, bis zu 256 Bit für complex extended Datentypen (CXT), darunter soll man sich erstmal sehr grosse und sehr genau komplexe Zahlen vorstellen.



Mit einem rechten Mausklick lassen sich verschiedene Dinge ändern: Datentyp, Format und Genauigkeit, Minimal- und Maximalwert, usw.

## b) Ringe

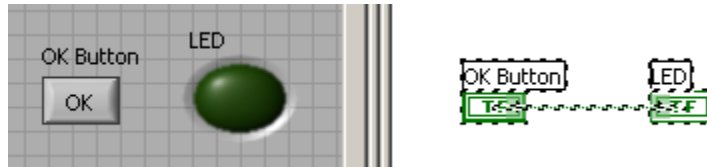
Ringe sind spezielle numerische Objekt, die verschiedene Auswahlmöglichkeiten mit einem Integer-Wert verknüpfen.



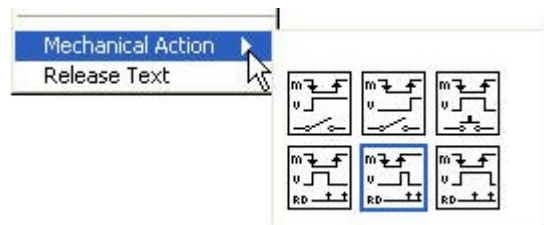
Die Zuordnung der Werte zu den einzelnen Fällen wird erreicht durch Rechter Mausklick auf den Ring/Edit Items (s. Bild links).

### c) Boole'sche Steuer und Anzeigeelemente

Boolesche Variable können nur zwei Werte annehmen: T oder F, TRUE oder FALSE, AN oder AUS. Am besten stellen sie sich unter einem Booleschen Steuerelement 1 Druckschalter vor und unter einem Booleschen Anzeigeelement eine Lampe/LED.



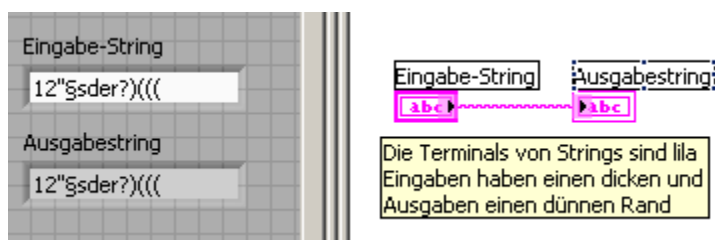
Wann soll der Schalter schalten? Beim Draufdrücken, oder beim Loslassen? Und wie lange soll er anbleiben? Immer oder soll er wieder zurückspringen? All dies lässt sich unter LabVIEW einstellen.



Stellen Sie im obigen Programmbeispiel per rechtem Mausklick die verschiedenen „Mechanical Actions“ ein und lassen Sie das Programm im Dauerlauf laufen. Sie bekommen dann eine Vorstellung davon, was diese verschiedenen Actions bedeuten!

### d) Zeichenfolgen / Strings

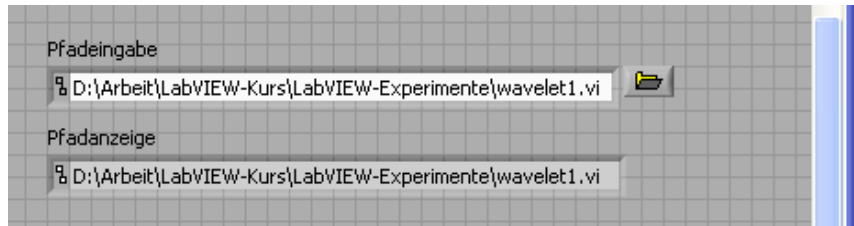
Zeichenfolgen oder Strings sind Ausdrücke wie „12qw@?&%“. Wie numerische Werte kann man sie eingeben oder auslesen. Zeichenfolgen haben die Farbe lila:



Mit einem rechten Mausklick lässt sich die Darstellungsart eines Strings einstellen: Normal Display, ‚\‘ Codes Display, Password Display und Hex Display. Probieren Sie die verschiedenen Möglichkeiten einfach aus.

### e) Dateipfade

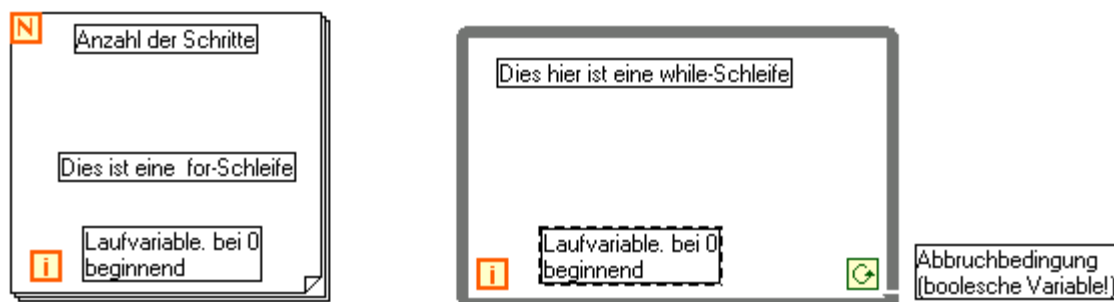
Mit Dateipfaden gibt man den Weg zu einer Datei in einem Dateisystem an:



## Programmstrukturen

Programmabläufe im Rechner, so könnte man sagen, bestehen aus Wiederholungen mit kleinen Variationen, daher benötigen wir - wie bei allen anderen Sprachen - auch in LabVIEW Kontrollstrukturen, die den Ablauf dieser Wiederholungen steuern. Es handelt sich dabei im wesentlichen um for- und while-Schleifen, Alternativen (if ...else, case) und einige LabVIEW-spezifische Strukturen. Alle Strukturen, die im folgenden beschrieben werden, finden Sie diagrammseitig im Menüpunkt „Funktionen/Strukturen“.

### Die for-Schleife und die while Schleife



Die **for-Schleife** führt die Programmteile, die im Inneren enthalten sind, genau N-mal aus. Die Variable N *muß* gesetzt werden, indem eine numerische (ganzzahlige) Konstante mit dem Anschluß „N“ verdrahtet wird. Die Laufvariable „i“ enthält die laufende Nummer der gerade anstehenden Iteration, wobei die Zählung von „i“ mit „0“ beginnt! Bei jedem Schritt wird „i“ um den Wert 1 erhöht. Die LabVIEW-„for-Schleife“ ist äquivalent zu folgendem C-Code:

```
for (i=0; i<N;i++) {statement1;statement2; ....}
```

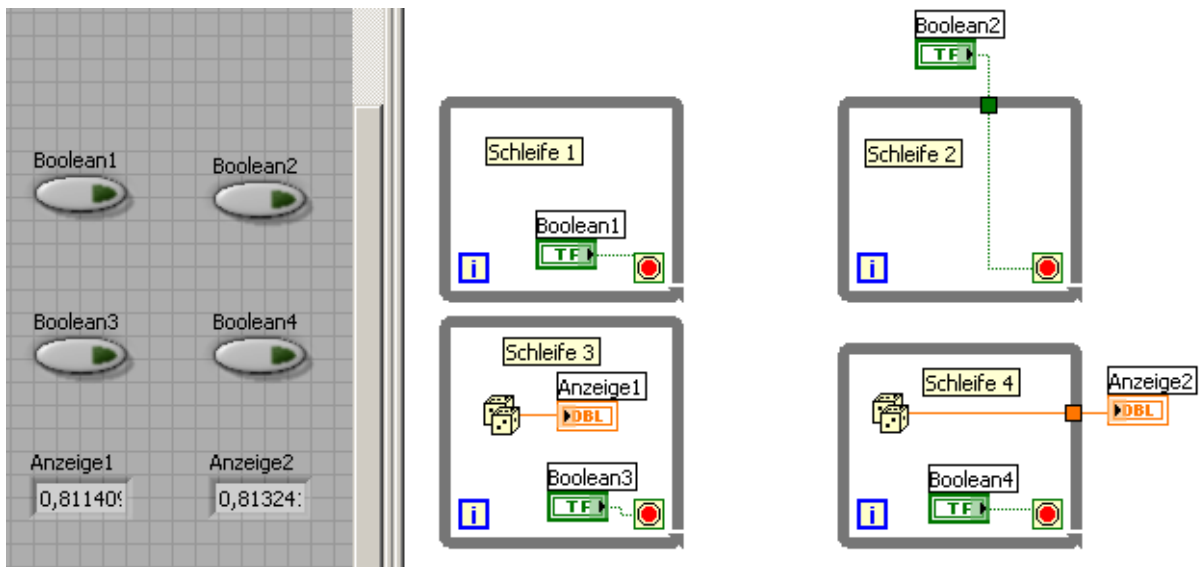
Die **while-Schleife** führt die in ihrem Inneren enthaltenen Programmteile so lange aus, wie die Abbruchbedingung wahr ist. An die Abbruchbedingung muß ein logischer Ausdruck angeschlossen werden. Die while-Schleife ist äquivalent zu folgendem C-Code:

```
do {statement1;statement2; ....} while (condition == TRUE )
```

Deshalb wird jede while-Schleife mindestens einmal durchlaufen, selbst wenn condition==FALSE von vornherein erfüllt sein sollte. Wenn Sie einen rechten Mausklick auf die Abbruchbedingung machen, sehen Sie, dass es da noch gewisse Gestaltungsmöglichkeiten gibt.

**Aufgabe:** Schreiben Sie ein Programm, das bei 10 beginnend, in Abständen von einer Sekunde und in Dreier-schritten die natürlichen Zahlen 10,13,16,... unter 100 ausgibt. Hinweis:  $3*i+10$  ergibt was?

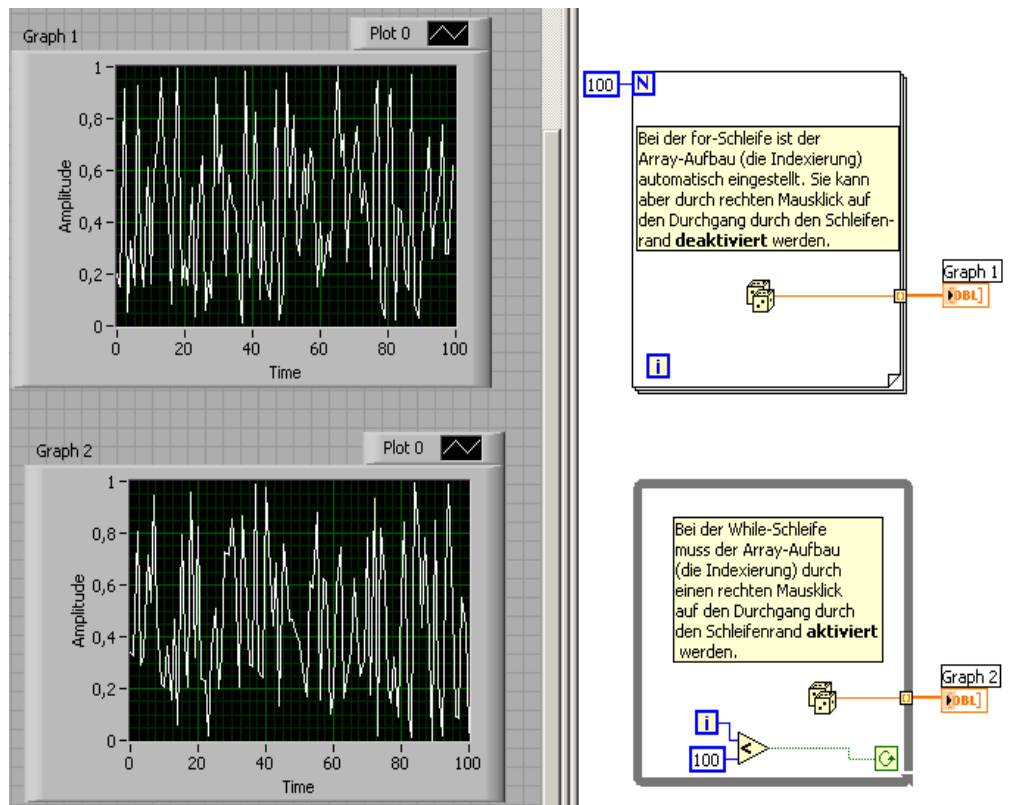
Betrachten Sie das folgende LabVIEW-Programm, das 4 while-Schleifen enthält:



Wie verhalten sich die verschiedenen unabhängig voneinander laufenden Schleifen?

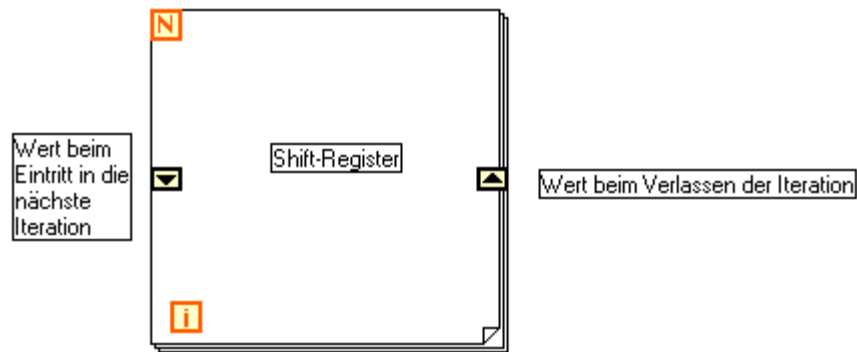
Schleife 1 stoppt, wenn Boolean 1 gedrückt wird. Schleife 2 ist durch Boolean 2 nicht abbrechbar! Schleife 3 zeigt nacheinander solange Zufallswerte an, bis Boolean 3 gedrückt wird. Und Schleife 4 zeigt nur einen einzigen Wert an und zwar erst nachdem Boolean 4 betätigt wurde. Haben Sie diesen Verhaltensweisen erwartet und können Sie sie erklären?

Vorgriff: Wenn Sie das nebenstehende Programm ausführen, dann können Sie feststellen, dass die sie mit der for-Schleife wie auch mit der while-Schleife auf den Rändern Arrays aufbauen können, die man sich mit dem Waveform-Graphen leicht ansehen kann. Wir werden diesen Aspekt später noch ausführlich vertiefen, aber sie sollten jetzt schon in der Lage sein, einfache Arrays aufzubauen und anzusehen.

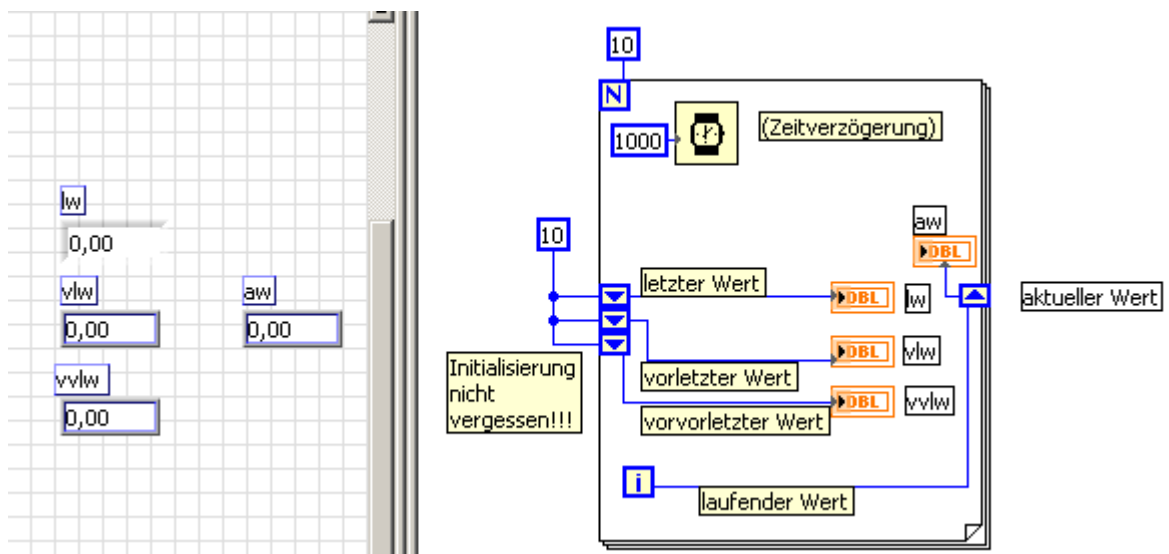


## Shift-Register / Schieberegister

Shift-Register - verfügbar für while-Schleifen und for-Schleifen - sind lokale Variable, die Werte von einer Iteration der Schleife in die nächste Iteration mitnehmen. Das Konzept dieser Register ist LabVIEW-spezifisch. In anderen Programmiersprachen wird es nicht benötigt.



Nun kommt ein Beispiel in dem auf der linken Seite der Schleife drei Shiftregister enthalten sind. In ihnen ist der Reihe nach der letzte (lw), der vorletzte (vl) und der vorvorletzte (vvl) Wert enthalten. (Die Zeitverzögerung wurde nur gemacht um den Programmablauf zu bremsen).



Ein Shift-Register besteht aus einem Paar von Terminals, das man erhält, wenn man mit der rechten Maustaste auf den rechten oder linken Rand der Schleife klickt. Erstellen Sie das obenstehende Programm und studieren Sie sein Verhalten. Die eingebaute Verzögerung von 1000 ms in der for-Schleife hat nur den Sinn, daß Sie den Ablauf Schritt für Schritt verfolgen können. An der linken Seite, ausserhalb der for-Schleife können Sie die Initialisierung der Shift-Register erkennen: bevor die for-Schleife anfängt zu laufen werden alle drei Register anfangs auf den Wert 10 gesetzt.

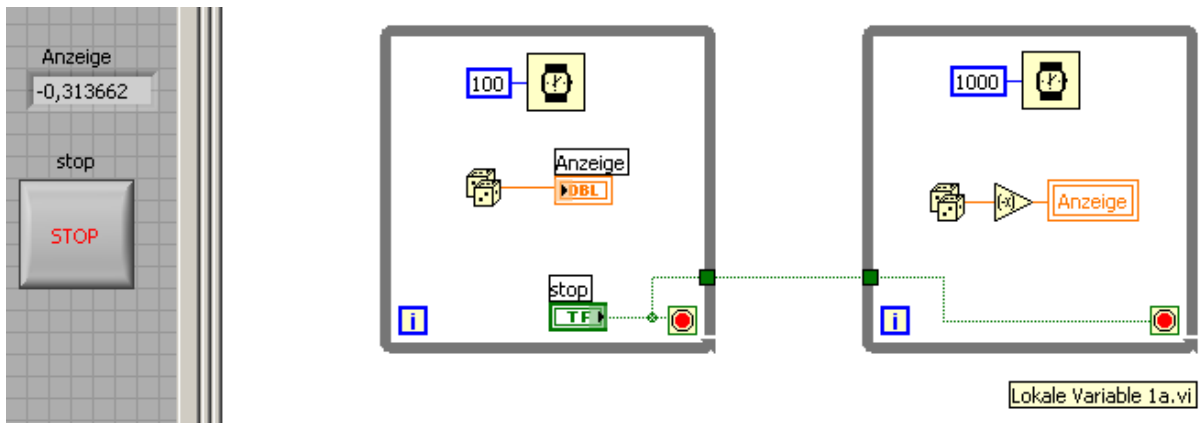
Die Register sind nicht immer paarweise vorhanden! Die Anzahl der Register auf der linken Seite kann durch einen rechten Mausklick erhöht oder erniedrigt werden. Ein typisches Beispiel für die Verwendung ist das sogenannte „averaging“, d.h. die Mittelwertbildung etwa über die 5 letzten Meßwerte.

**Aufgabe:** Schreiben Sie zunächst ein Programm, in dem so lange Zufallswerte ausgegeben werden, bis Sie auf eine Stop-Taste drücken. Ändern Sie das Programm dann so ab, daß immer der Mittelwert über die letzten 5 Zufallswerte ausgegeben wird. Was muss man machen, damit der Mittelwert über alle bis dato erzeugten Zufallswerte berechnet wird?

**Aufgabe:** Was passiert, wenn man die Initialisierung vergisst?

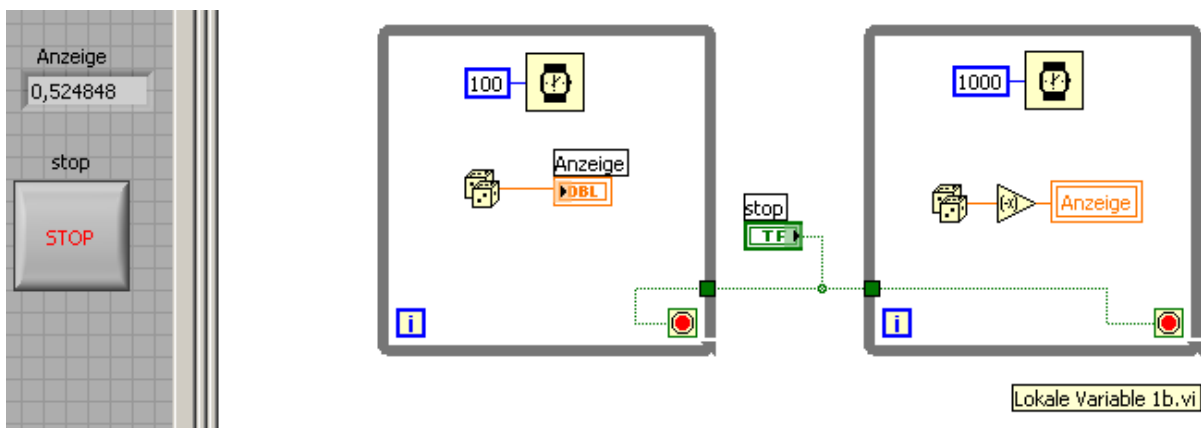
## Lokale Variable in LabVIEW

Da wir schon bei lokalen Variablen sind, führen wir hier gleich noch die andere Art lokaler Variablen ein, die dem LabVIEW-Programmierer zur Verfügung stehen. Stellen Sie sich vor, daß Sie in einem LabVIEW-Programm den Zugriff auf eine Variable an vielen verschiedenen Stellen benötigen. Mit vielen Drahtmetern kann man dieses Problem vielleicht *manchmal* lösen. Sie erhalten bei dieser Vorgehensweise jedoch auf jeden Fall ein im wahrsten Sinne des Wortes verwirrendes Programm! Tatsächlich aber gibt es sogar Fälle, wo man grundsätzlich, selbst mit noch so viel Draht nicht weiter kommt. Stellen Sie sich vor, Ihr Programm enthält zwei while-Schleifen, die beide mit dem gleichen Stop-Button abgebrochen werden sollen. Dann bekommen Sie Schwierigkeiten, die Sie am besten selbst am folgenden Beispiel ausprobieren.



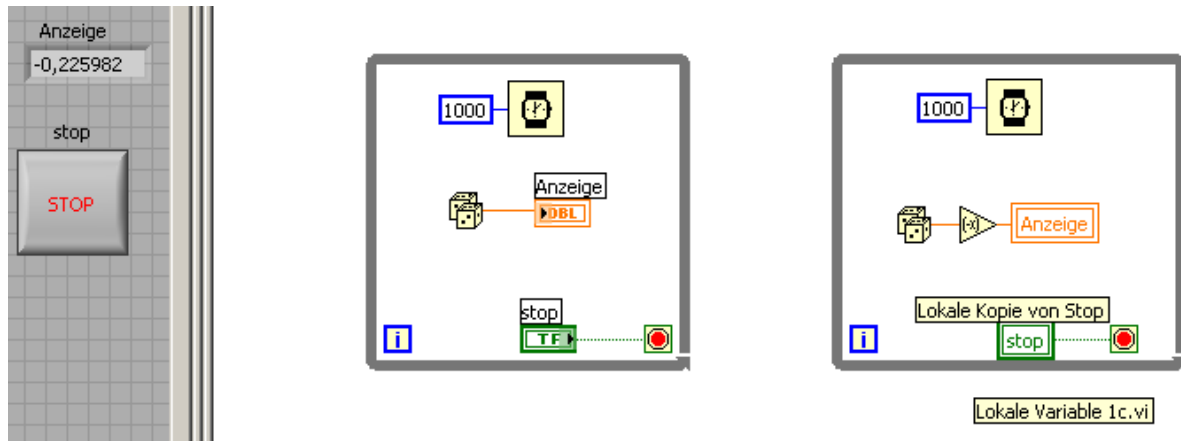
Egal wohin Sie die Programmabbruch-Variable schieben: Sie werden es nicht schaffen, das Programm vernünftig zum Laufen zu bringen – genauer gesagt: vernünftig abzubrechen. (Schalten Sie in den „High-Lighting-Modus“ um und gehen Sie Step by Step durch das Programm, um zu verstehen warum es **so nicht** gehen kann).

Das folgende Programm mischt zwar die negativen und die positiven Zufallszahlen, aber mit einer einzigen Stop-Taste abbrechen lässt es sich immer noch nicht:



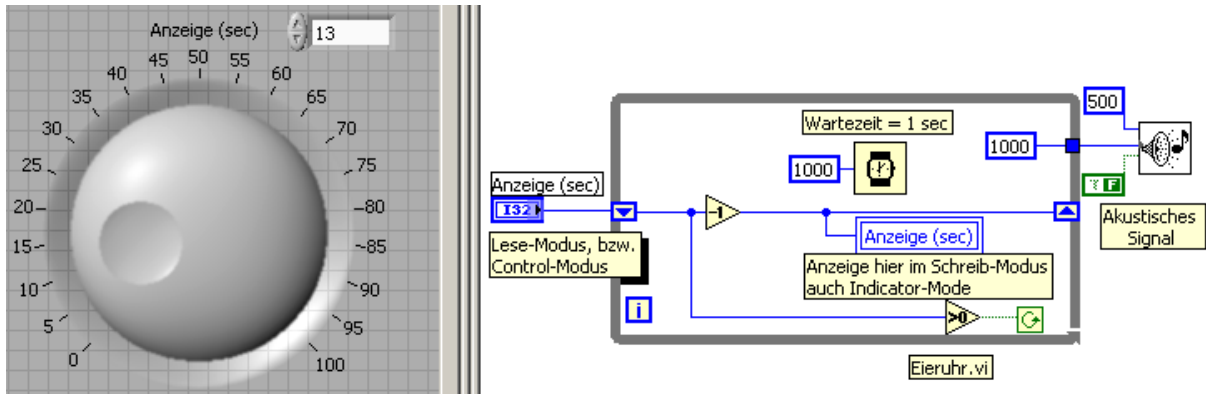


Dieses Problem des Abbruchs von 2 Schleifen mit einer einzigen Taste kann man lösen, wenn in der rechten Schleife eine Kopie der Programmabbruchvariablen „Stop“ eingebaut wird.



Lokale Variable finden Sie unter dem Programmpunkt „Structures“, dort wo auch die „while-Schleife“ zu finden ist. Nach dem Anklicken müssen Sie diese lokale Variable mit der Programmabbruchvariablen verbinden (rechter Mausklick/Objekt wählen) – damit erhalten Sie eine Kopie dieser Variablen, die Sie in der zweiten Schleife plazieren können, um so den Wert von „Programmabbruch“ in die zweite Schleife hineinzutransferieren. Lokale Kopien zu einer bestimmten Variablen erhält man einfacher, wenn man diese per rechter Mausklick/Erzeuge/Lokale Variable herstellt (damit wird auch gleich die Zuordnung zu der gewünschten Variablen hergestellt). Lokale Variable haben noch eine wichtige Eigenschaft: Sie können je nach programmtechnischem Zusammenhang als Datenquelle oder als Datensenke definiert werden – beachten Sie diesen Punkt, wenn Sie Verdrahtungsprobleme bekommen! Wenn es von einer Variablen mehrere lokale Kopien gibt, kann die eine Kopie an der einen Programmstelle im Lesemodus (Quelle) sein und eine andere Kopie an anderer Stelle im Schreibmodus, also Senke! Dazu das folgende Beispiel mit der Eieruhr.

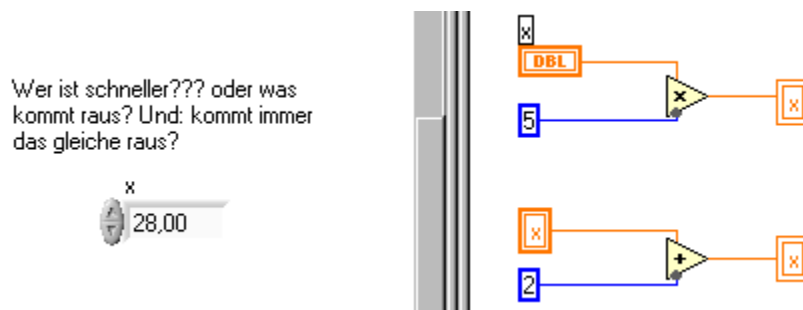
Mit lokalen Variablen können Sie zum Beispiel eine virtuelle Eieruhr programmieren: eine Uhr also, die auf einen gewissen Wert gestellt (control-Modus, dicke Umrandung) wird und dann bis auf Null zurückläuft (indicator-Modus, dünne Umrandung)



Das akustische Signal können Sie mit dem Beep.vi aus Graphics & Sound erzeugen

### Was sind Race-Conditions?

Im Zusammenhang mit lokalen Variablen kann man sich gefährliche Probleme einhandeln. Betrachten Sie das folgende einfache Programm, das drei lokale Variable enthält. Diese lokalen Variablen sind alle insgesamt Kopien einer einzigen Variablen: „x“, eine davon im Lesemodus und zwei im Schreibmodus.



Im LabVIEW-Handbuch steht in diesem Zusammenhang: „To avoid race conditions, do not write to the same variable you read from,,

## **Globale Variable**

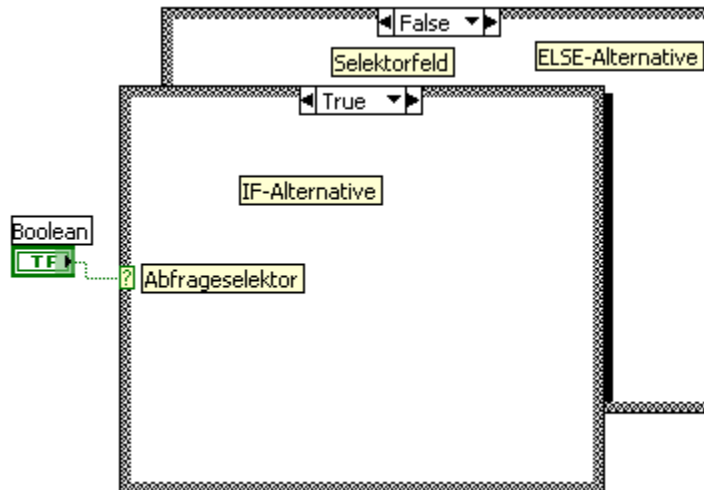
Wenn Sie schon mit klassischen Programmiersprachen gearbeitet haben, ist Ihnen der Begriff „globale Variable“ sicher bekannt. Lokale Variable sind nur in einem begrenzten Programmbereich gültig, während auf globale Variable aus dem gesamten Programm (lesend und schreibend) zugegriffen werden kann. Man kann ganz schnell zu total verwirrenden Situationen und gut versteckten Denk- und Programmfehlern kommen, wenn man den Überblick darüber verliert, wer, wann, von weiß nicht wo, eine globale Variable lesen oder ändern darf.

Die lokalen Variable in LabVIEW haben wir oben kennengelernt. Auf sie kann **innerhalb eines VIs** an verschiedenen Stellen lesend und schreibend zugegriffen werden.

Weiterhin gibt es auch in LabVIEW globale Variable, auf die von mehreren ganz getrennten, quasiparallel arbeitenden VIs lesend und schreibend zugegriffen werden kann. Und auch in LabVIEW gibt es, wen wundert es noch, die eben beschriebenen Fehlermöglichkeiten. Globale Variable werden wir später behandeln (wenn noch Zeit übrig ist). Falls wir nicht dazu kommen gibt es einen Trost: manche (kompetente) LabVIEW-Entwickler behaupten: „Use of globals usually indicates failures of design“.

## Entscheidungen (Alternativen): if ... then ... else ... oder Case-Struktur

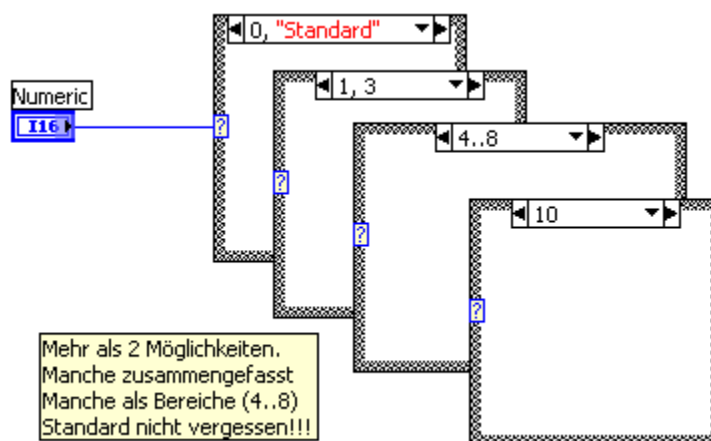
Ein Programmablauf muß sich verzweigen können. Solche Verzweigungen werden in herkömmlichen Programmen mit den Sprachelementen „if {Bedingung == TRUE} then {Anweisung1} else {Anweisung2}“



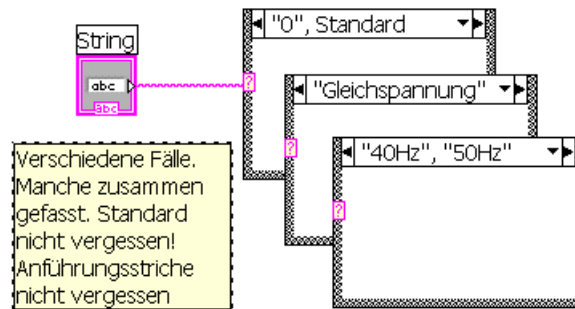
erzeugt.

Wenn an den Selektor (wie oben) eine Boolesche Variable angeschlossen wurde, besteht die Alternativ-Struktur aus 2 Möglichkeiten (realisiert in 2 hintereinanderliegenden Subdiagrammen, siehe oben).

Es besteht aber auch die Möglichkeit an den Selektor eine numerische (ganzzahlige!) Variable oder eine Stringvariable anzuschließen. (Wenn man eine Kommazahl anschliesst, wird sie vor der Abfrage im Abfrageselektor gerundet. Einer der insgesamt möglichen Fälle **muß** auf jeden Fall einen **Default-, bzw. Standard-Fall** enthalten, damit es weiter gehen kann im Programm. Es ist möglich Fälle, die für ein Diagramm zutreffen durch Kommas getrennt zu kennzeichnen. Es ist auch möglich Bereiche anzugeben: 11..17 z.Bsp. In diesem Fall landet man immer dann, wenn der Selektor auf Zahlen 11,12,13,14,15,16,17 trifft. Einen neuen Fall (Case) erzielt man durch rechten Mausklick auf den Rand. Bei mehr als zwei verschiedenen Fällen spricht man üblicherweise von einer Case-Struktur. Wahrscheinlich gibt es auch noch die Möglichkeit, die Sache richtig verwirrend zu machen: am besten Sie bleiben bei booleschen oder Integer- oder Ringvariablen. (Was passiert z.B. wenn Sie einen Fall „1..7“ anlegen und noch einen Extrafall „5“?)



Wenn statt der numerischen Variablen eine String-Variable an den Selektor angeschlossen wird ergibt sich Folgendes: (0 ist hier keine Zahl mehr, sondern ein Zeichen!)



**Aufgabe:** Schreiben Sie kleine Test-Programme, mit denen Sie lernen, mit diesen Alternativstrukturen umzugehen, insbesondere, wie man Fälle hinzufügt, verdoppelt, entfernt und verschiebt, und wie man von Fall zu Fall kommt.

**Aufgabe:** Bauen Sie ein VI, das einen Würfel simuliert und schreiben Sie dann ein Programm zum Testen des Würfels. Es soll bei diesem Test also die Frage geklärt werden, ob bei vielen Würfeln alle Zahlen mit der gleichen Häufigkeit vorkommen.

**Aufgabe:** Angenommen Sie schießen auf eine Zielscheibe mit Durchmesser 2, die in ein Quadrat der Kantenlänge 2 eingebettet ist. Angenommen alle Schüsse treffen das Quadrat – das schaffen Sie mit einem Zufallsgenerator, der Punkte (x,y) mit x und y zwischen -1 und 1 würfelt. Wenn  $x^2 + y^2 \leq 1$  ist haben Sie ausserdem auch noch die Scheibe getroffen. Das Verhältnis von Treffern zur Anzahl der Schüsse insgesamt konvergiert (ziemlich langsam – aber wozu gibt's Computer?) gegen  $\pi/4$ . Bauen Sie ein VI um dieses Verhalten nachzuprüfen.

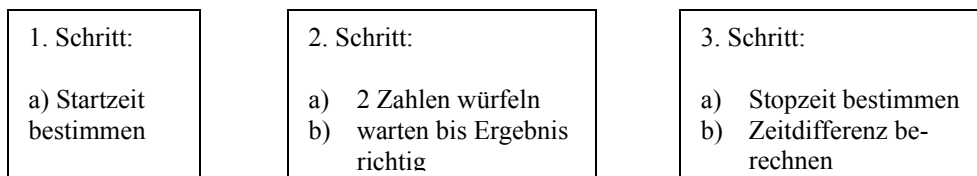
Frage: warum eigentlich konvergiert dieses Verhältnis gegen  $\pi/4$ ?

## Ablaufstruktur oder auch: Sequenz

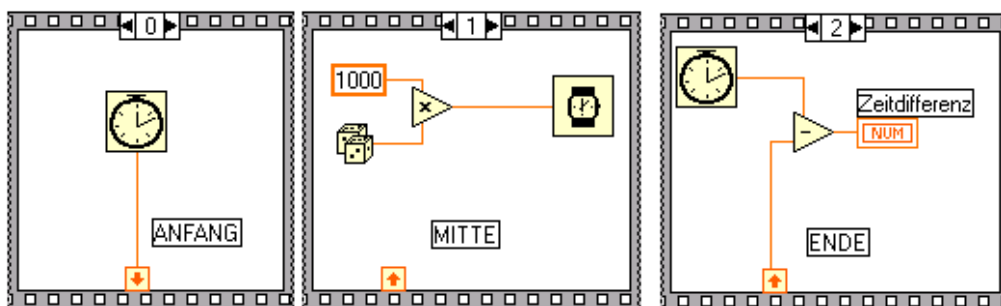
Alle klassischen Programmiersprachen enthalten einen inhärent durch die zeilenweise Abfolge der Programm-anweisungen festgelegten Ablauf. Bei LabVIEW ist das anders, denn LabVIEW selbst legt diese Abfolge fest. Wenn Sie dies verhindern wollen, weil Sie auf einen fest (von Ihnen, vom Problem, vom Lauf der Dinge) be-stimmten Ablauf angewiesen sind, müssen Sie die Sequenzstruktur einsetzen.

**Aufgabe:** Schreiben Sie ein Programm, das zwei Eingaben addiert und das Ergebnis ausgibt und mit zwei anderen (von den ersten beiden unabhängigen) Eingaben einige Rechnungen mehr ausführt. Lassen Sie das Pro-gramm mit dem Lämpchen laufen (Highlight-Funktion) und beobachten Sie die Reihenfolge der Abarbeitung der einzelnen Programmteile. Verstehen Sie jetzt, wie LabVIEW den Ablauf bestimmt?

**Aufgabe:** Bauen Sie einen „Kleines-Einmaleins-Trainer“, der nicht nur auf richtige Ergebnisse, sondern auch auf die Schnelligkeit der Antworten achtet. Die Programmstruktur sollte wie folgt aussehen:



Die Sequenzstruktur sieht aus wie die aufeinanderfolgenden (in LabVIEW übereinander liegenden!!!) Bilder eines Films, wobei das mittlere Bild durch andere Programmteile ersetzt werden muß, um die oben gestellte Auf-gabe zu lösen!

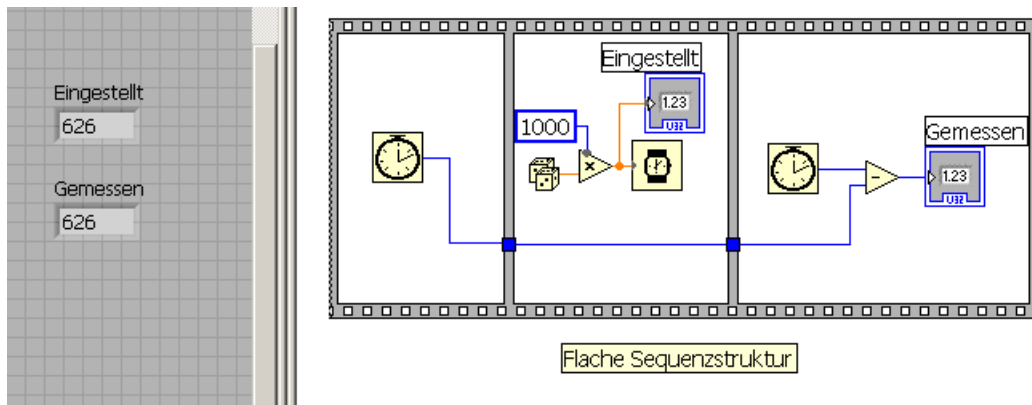


Beachten Sie am unteren Filmrand die sogenannten „lokalen Sequenzen“. Sie dienen der Übergabe eines Wertes aus einem Bild in ein anderes Bild. Man erhält sie durch einen rechten Mausklick auf den Filmrand. Am Anfang sind sie ohne Pfeil, erst wenn sie verdrahtet werden, erkennt LabVIEW, ob hier was raus oder reingehen soll und trägt die entsprechende Pfeilrichtung ein. Achtung: einer lokalen Sequenz kann nur einmal ein Wert zugewiesen werden, in allen anderen Frames kann der Wert nur ausgelesen, nicht aber verändert werden!

Das obige Beispiel mag als Inspiration für die vorstehende Programmieraufgabe genommen werden.

Tatsächlich liegen die drei Bilder des Films hintereinander. Nur hier im Ausdruck sind sie nebeneinander gelegt worden.

In neueren LabVIEW-Versionen gibt es neben der gestapelten Sequenzstruktur eine sogenannte „Flat Sequence Structure“ im Gegensatz zur früher benutzten „Stacked Sequence Structure“. Sehen Sie sich das Teil mal an.



**Aufgabe:** Erweitern Sie den „Kleines-Einmaleins-Trainer“, indem Sie ein Bewertungssystem einbauen: zuerst eine Note für richtige und falsche Antworten, dann noch eine Note für die Geschwindigkeit.

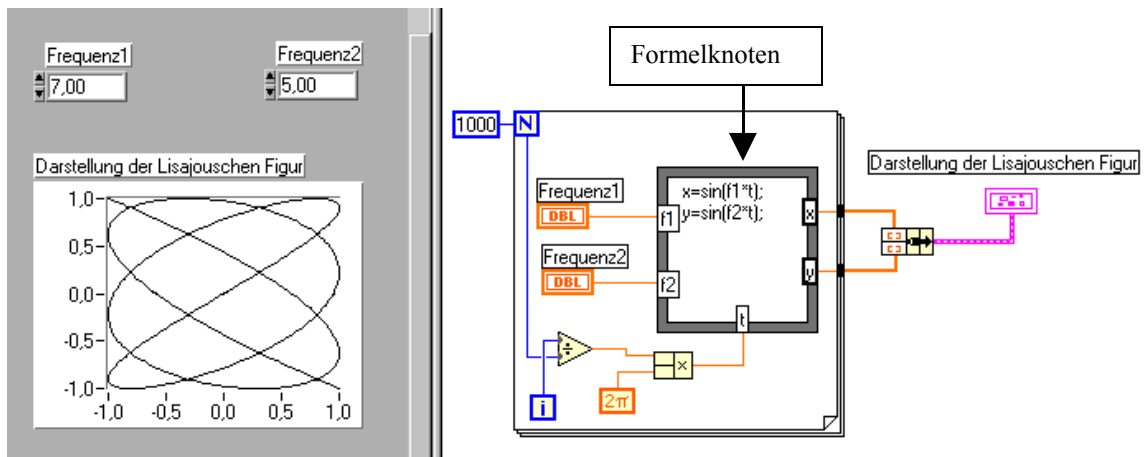
**Aufgabe:** Wie lange braucht LabVIEW, um den Wert „1+1“ zu berechnen? Wie lange für  $\sqrt{2}$  ?

(Wahrscheinlich werden Sie bei Ihrem ersten Versuch den Wert 0 erhalten. Warum? Ein winzig kleiner Trick überwindet diese Schwierigkeit!)

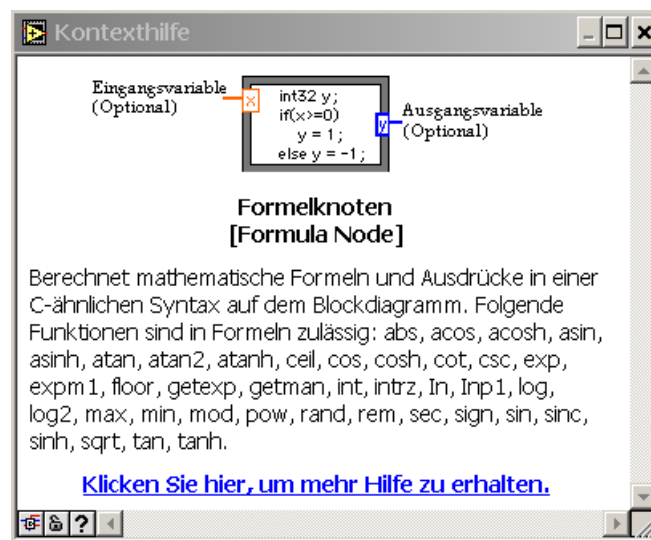
**Beachten Sie:** Um bei der gestapelten Sequenzstruktur Variable aus einem Bild in ein anderes Bild zu überführen benötigen Sie auf dem Rand eine sogenannte „Lokale Sequenz-Variable“, die bei der flachen Sequenzstruktur nicht erforderlich ist!

## Formelknoten

Der Formelknoten ist eine Box, in die mathematische Formeln eingetragen werden können zur Herstellung von Beziehungen zwischen Ein- bzw. Ausgängen. Unten finden Sie eine Übersicht, der Operatoren und Funktionen die innerhalb des Formelknotens verwendet werden können. Sie können diese Liste jederzeit bekommen, wenn Sie mit dem Cursor auf den Formelknoten zeigen und das Hilfenfenster einschalten. Durch Klicks mit der rechten Maustaste auf den Rand der Box kann man entweder Eingänge oder Ausgänge erzeugen, die einen Namen be-



kommen müssen. Dann werden in die Box die jeweiligen Anweisungen eingetragen. Jede Anweisung muß mit einem **Semikolon** abgeschlossen werden. Das obenstehende Programm könnte man dazu verwenden um die Lissajouschen Figuren zu untersuchen.

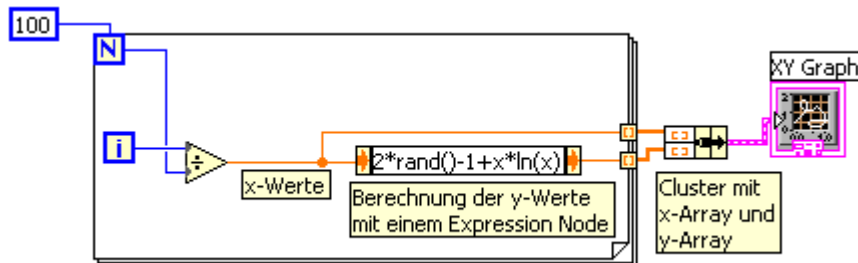


**Liste der Operatoren und Funktionen, die innerhalb eines Formelknotens verwendet werden dürfen**



## Ausdrucksknoten

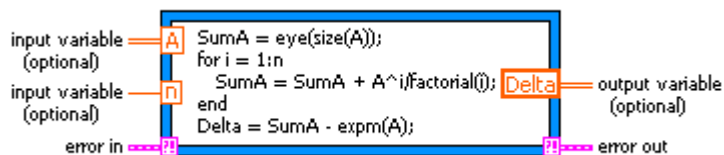
Eine vereinfachte Fassung des Formelknoten ist der Ausdrucksknoten (Expression Node). Er wird verwendet um Ausdrücke und Gleichungen zu berechnen, die nur eine einzige Variable enthalten. Sie finden ihn unter Funktionen/Numerisch.



Es werden 100 x-Werte zwischen 0 und 0,99 berechnet, die nacheinander dem Expression-Node übergeben werden und der dann die gewünschten y-Werte berechnet. Jeder x- und jeder y-Wert wird ausserdem in einem Array auf dem Rand der for-Schleife abgelegt. Aus diesen beiden Arrays wird ein Cluster gebildet, das dem XY-Graph übergeben wird.

## MathScript-Knoten

MATLAB ist eine sehr umfangreiche kommerzielle mathematische Software mit umfangreichen Bibliotheken für Statistik, Signal- und Bildverarbeitung. In früheren LabVIEW-Versionen war es möglich, MATLAB -Programme in LabVIEW einzubinden, allerdings benötigte man dafür eine MATLAB -Lizenz. Um dieses Problem zu umgehen wurde von National Instruments MathScript entwickelt. Es handelt sich dabei um eine textbasierte Programmiersprache, die MATLAB sehr nahe kommt. Im allgemeinen - so steht es im LabVIEW-Handbuch – kann man mit MathScript auch MATLAB -Programme ausführen, aber eben nur im allgemeinen, das sollte man im Hinterkopf haben. Der MathScript-Knoten ist nun ein LabVIEW-Element, das MathScript-Programme ausführt. Sie finden ihn unter Funktionen/Programmstrukturen



Im Prinzip gleicht der MathScript-Knoten dem Funktionsknoten, nur dass im Inneren des Knotens MathScript-Programme stehen. Weitere Informationen zu MathScript müssen Sie sich aus dem Handbuch holen

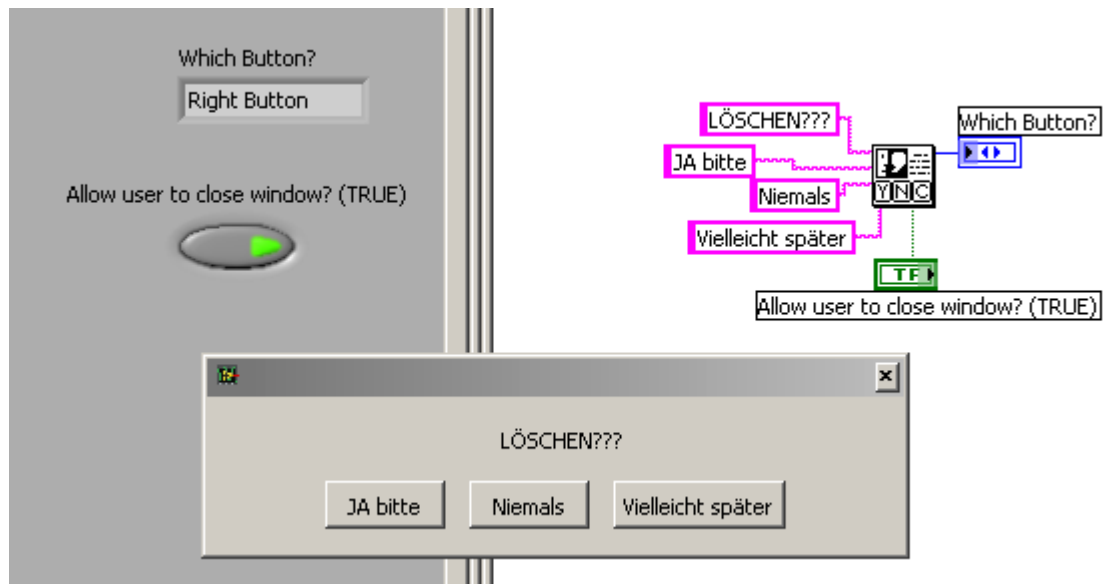
## Dialog-Boxen

Mit Dialog-Boxen können aus dem Programm heraus Meldungen/Fragen an den Nutzer geschickt werden, um anschliessend entsprechend zu reagieren.

In LabVIEW gibt es drei Typen von Dialogen: den „One Button Dialog“, den „Two Button Dialog“ und den „Three Button Dialog“, die Sie alle unter dem Menüpunkt „Dialog&User Interface“ finden können



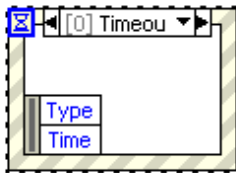
Hier ein Beispiel für einen „3 Knöpfe Dialog“



## Die Event-Struktur

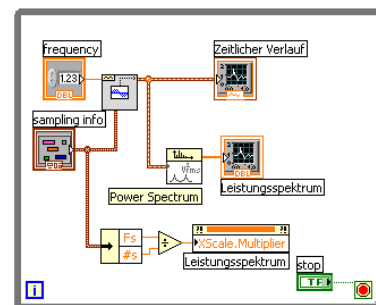
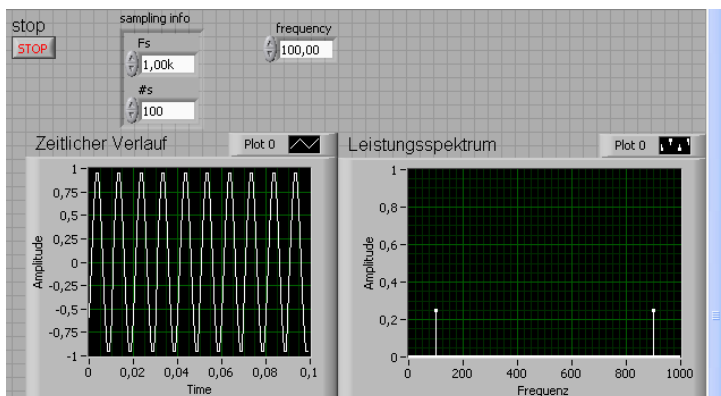
Angenommen Sie stehen vor der Aufgabe, ein Programm zu erstellen, in dem es darauf ankommt, schnell auf Frontpanel-Aktivitäten (beispielsweise ein Mausklick auf einen Schalter oder Werteänderung eines Steuerelements) zu reagieren. Dann kennen wir bisher in LabVIEW nur die Möglichkeit in einer While-Schleife immer wieder abzufragen, ob diese Aktivität/Änderung eingetreten ist oder nicht. Wenn es darüber hinaus so ist, dass dieser Aktivität relativ selten auftritt, dass aber trotzdem schnell reagiert werden muss, dann sieht man, dass ein solches Programm hauptsächlich mit Abfragen beschäftigt ist und immer nur kurzzeitig zwischen zwei Abfragen wirklich arbeiten kann. Dieser Zustand kann vermieden werden, wenn man „ereignisgesteuert“ programmiert. Unter einem Event – einem Ereignis – soll man sich in LabVIEW den Mausklick des Nutzers auf einen Button des Frontpanels, die Werteänderung eines Eingabeelements, eine Mausbewegung und anderes mehr vorstellen. Eventgesteuerte Programme werden nicht linear durchlaufen, sondern es werden, wenn ein gewisses Ereignis eintritt, für den LabVIEW-Programmierer transparent zugehörige Ereignisbehandlungsroutinen (event handler) aufgerufen. (Ein zu Events verwandtes Konzept ist übrigens das Interrupt-Konzept).

**Aufgabe:** Bringen Sie das “Mouse Rollover Demo.vi” aus den Beispielen zum Laufen, dann haben Sie eine erste Vorstellung davon, was mit Eventprogrammierung gemeint ist.

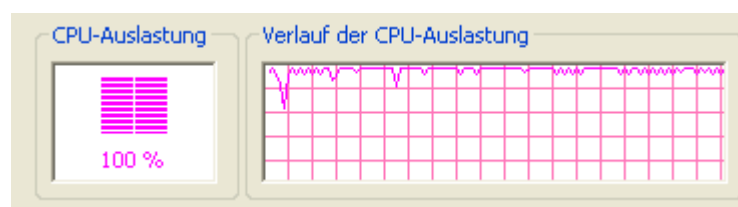


Die LabVIEW-Event-Struktur besteht aus mehreren Unterdiagrammen, die einzelnen Ereignissen zugeordnet sind und immer dann ausgeführt werden, wenn das jeweilige Ereignis eintritt. Zunächst muss überlegt werden, wie lange maximal auf Ereignisse gewartet werden soll. Links oben wird diese Timeoutzeit in Millisekunden an der Sanduhr eingegeben. Der Standardwert ist -1, was kein Timeout bedeutet, eine solche Eventstruktur wartet ewig lange auf die möglichen Ereignisse. Per rechtem Mausklick können neue Ereignisfälle hinzugefügt werden, und mit “Edit Events Handled by This Case...” die zugehörigen Ereignisse definiert werden.

Wir betrachten nun ein Beispiel. Zunächst wird in einer klassischen While-Schleife ein sinusförmiges Signal erzeugt und dessen Leistungsspektrum berechnet. Sampling-Parameter und Signalfrequenz können dabei variiert werden.

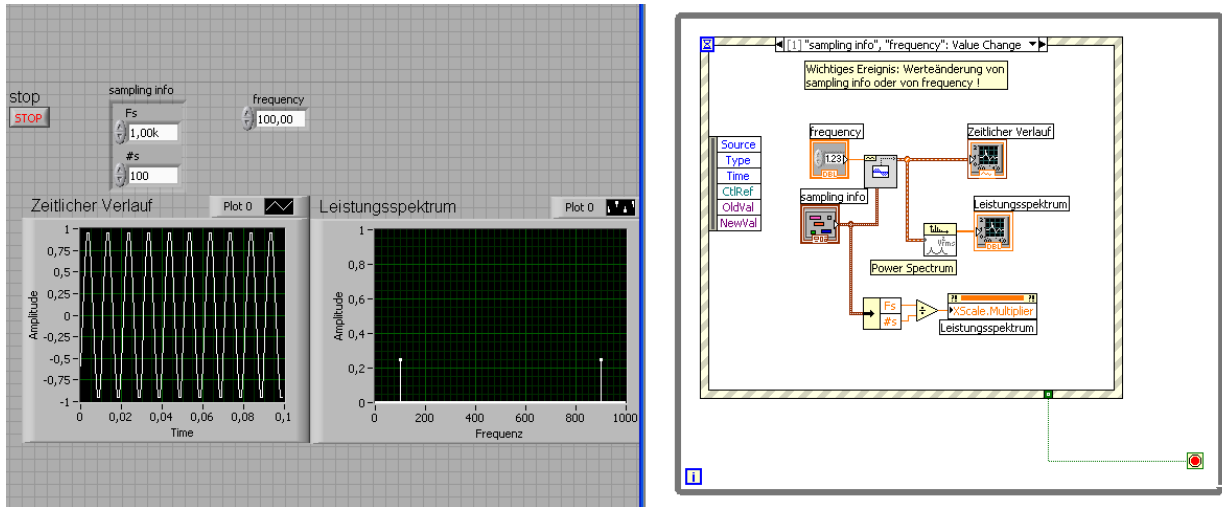


Wenn man sich nun die CPU-Auslastung ansieht, erkennt man, dass sie 100% beträgt:

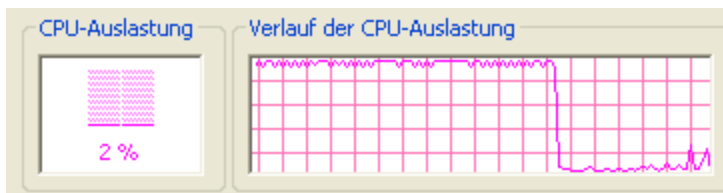


Beim vorigen Beispiel wurde bei jedem Schleifendurchlauf das Leistungsspektrum neu berechnet: das ist die Ursache für die hohe Prozessorauslastung.

Im folgenden Beispiel, das mit Events arbeitet, wird das Spektrum nur dann neu berechnet, wenn sich einer der Werte Sampling-Parameter oder Signalfrequenz verändert:



Sie können nun feststellen, dass die CPU-Auslastung fast bis auf Null sinkt:



**Aufgabe:** Bestimmen Sie die Koordinaten der Mausposition zum Zeitpunkt des Klicks auf ein Control-Element auf dem Frontpanel. (Hinweis: Beispiele).

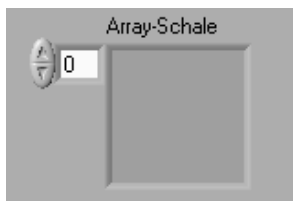
## Arrays und Cluster - alias: Datenfelder und Datenstrukturen

Datenfelder kurz: Felder oder Arrays, sind Listen von Daten vom gleichen Datentyp. Beispiel: eine Tabelle von Meßwerten – alle Elemente der Tabelle sind vom gleichen Typ: Zahlen z.B., nichts als Zahlen, meistens: floating-point Zahlen. Oder ausschliesslich Zeichen/Characters/Bytes: je nachdem, wie sie sie sehen wollen.

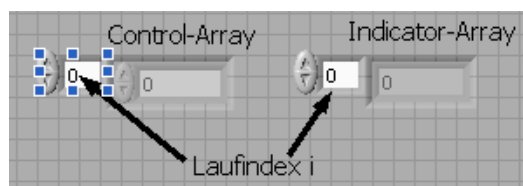
Datenstrukturen (oder auch Cluster) sind Zusammenfassungen von Variablen mit verschiedenen Datentypen in eine neue Datenstruktur. Denken Sie etwa an Personaldaten. Sie bestehen etwa aus: Name, Vorname, Geburtsdatum, Alter, Monatsgehalt, Anzahl der Kinder, Familienstand. Diese Datenstruktur besteht also zunächst aus einigen Zeichenketten, auch Strings genannt (Name, Vorname etwa), Zahlen (Gehalt, Kinderzahl), usw. Für eine effektive Programmierung sind solche Konstrukte sehr brauchbar. Natürlich können Sie auch daraus wieder Arrays aufbauen um darin z.B. den Personalbestand aus einer Personaldatei abzulegen, zu sortieren. Man kommt so zu einem Array von (gleichartigen) Personaldaten-Clustern.

### Felder

Zurück zu den Feldern. Felder erzeugt man z.B. bei der Aufnahme einer Meßreihe: wenn Sie 20 verschiedene Temperaturwerte messen, tragen Sie diese Werte in eine Liste ein. Und eine solche Liste heisst dann eben: Feld. Ein Array können Sie sich auch wie ein 1- oder 2-dimensionales (oder noch höher-dimensionales) Gebilde vorstellen:  $a[i]$ , oder  $a[i,j]$ . Auf die Einzelkomponenten können Sie per Index zugreifen. Beachten Sie immer, daß genau wie in C auch in LabVIEW die Zählung der Indizes bei Null beginnt!



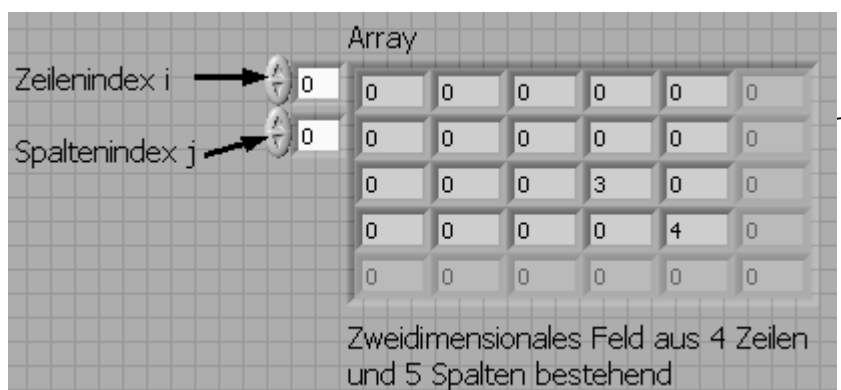
In LabVIEW erzeugt man einen Array in zwei Schritten auf der Panelseite. Dort gibt es zunächst einen Menüpunkt Array & Cluster/Array mit dem ein leeres Array, eine Array-Schale, erzeugt werden kann. Im zweiten Schritt muß gesagt werden, mit welchem Datentyp das Feld gefüllt wird. Angenommen, wir wollen ein Feld mit Meßdaten haben, dann muß man sich überlegen, ob die einzelnen Werte des Feldes dargestellt oder verändert werden können sollen. Je nachdem wird man ein „numeric control“-Objekt oder ein „numeric-indicator“-Objekt in das bisher leere Feld schieben.



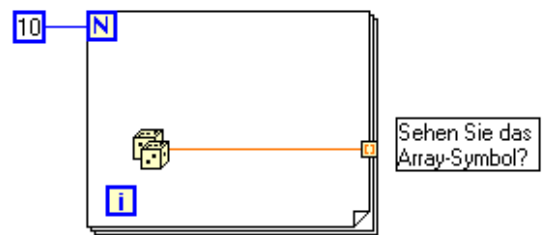
Das linke Beispiel stellt ein Feld von “numeric-controls” dar, während das rechte Beispiel ein Feld von numerischen Anzeigen ist. Der Unterschied ist erst auf den zweiten Blick ersichtlich. Wenn Sie den Laufindex durchklicken, können

Sie die Array-Werte ablesen, beim linken Beispiel können Sie diese Werte auch noch abändern.

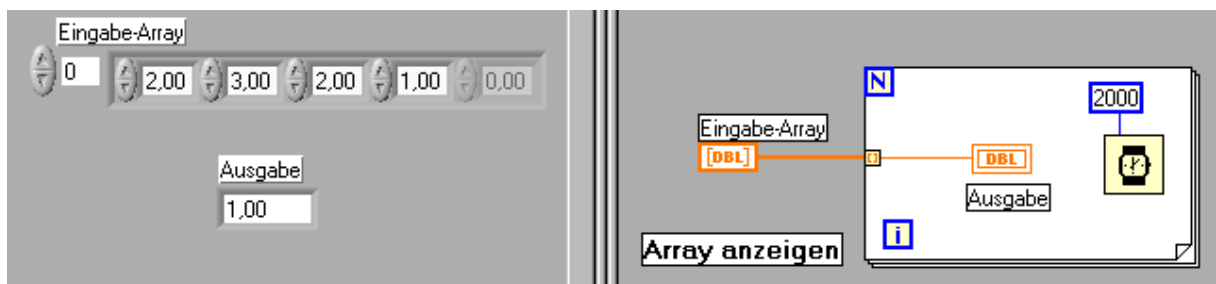
Ein zwei- oder höherdimensionales Feld erhält man mit einem rechten Mausklick auf den Indexbereich.



Die for- und die while-Schleife können Arrays auf ihren Rändern aufbauen und indizieren. Das nebenstehende Programm etwa füllt das Array auf dem Schleifenrand mit 10 Zufallszahlen. Erzeugen Sie einen Indicator an dieser Stelle. Dann können Sie sich die erzeugten Zufallswerte einzeln ansehen.

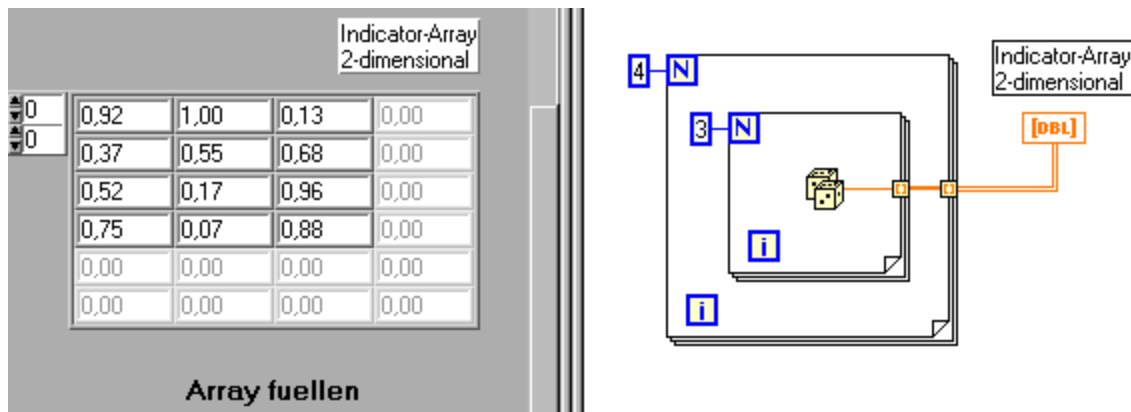


LabVIEW erkennt selbständig, wieviele Werte in einem Array abgelegt sind. Im nächsten Beispiel können Sie das Feld von Hand z. B. mit 4 Werten füllen. Beim Programmablauf muß die Endbedingung der for-Schleife (N) nicht mit der Zahl 4 belegt werden. LabView erkennt selbst, daß das angeschlossenen Eingabe-Array mit 4 Werten belegt worden ist.



For-Schleifen indexieren standardmäßig, während while-Schleifen standardmäßig **nicht** indexieren. Die Möglichkeit des Indexierens kann jedoch per rechtem Mausklick verändert werden. (Was könnte Ihrer Meinung nach der Grund für diese unterschiedliche Behandlung von for- und while-Schleife sein?)

Mit dem folgenden Programm können Sie ein zweidimensionales Feld füllen. Die innere Schleife füllt nacheinander die 0-te, erste, zweite und schließlich die dritte (und letzte!) Zeile des 2-dimensionalen Arrays.



Das vorstehende Programm läßt sich übrigens nur dann problemlos verdrahten, wenn der Array panelseitig auch tatsächlich 2-dimensional ist! Wenn die Felder nicht zu groß sind, lassen sie sich aufziehen, so daß man jedes einzelne Feldelement sehen kann. Oben sehen Sie ein zweidimensionales Array, das aus vier Zeilen und drei Spalten besteht.

**Aufgabe:** Schreiben Sie ein Programm, das bestimmt, wieviele negative und nichtnegative Zahlen in einem ein-dimensionalen Array reeller Zahlen enthalten sind.

**Aufgabe:** Erzeugen Sie ein 2-dimensionales (12x7)-Array, das zunächst mit  $-1$  gefüllt wird und das dann (zum Zusehen) zeilenweise mit dem Wert 1000 gefüllt wird. (12x7-Array.vi)

**Aufgabe:** Erzeugen Sie ein 2-dimensionales 10x10-Array, in dem die einzelnen Elemente die Werte von 0 bis 99 enthalten.

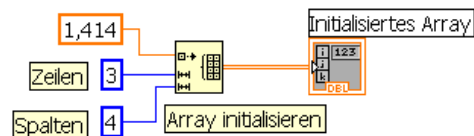
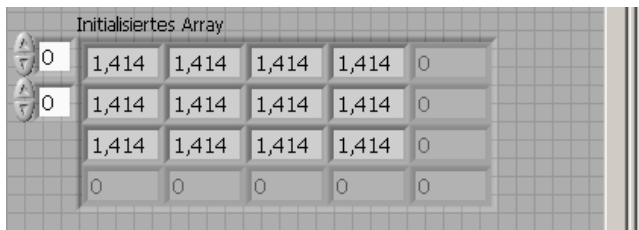
**Aufgabe:** Erzeugen Sie ein Array von Booleschen Anzeigeelementen. Was kann man machen, damit alle Anzeigen leuchten? Wie kann man Elemente zufällig aus- oder anschalten. Kennen Sie das „Game of life“? Haben Sie genügend Zeit um es in LabVIEW zu programmieren? Ein weiteres nicht ganz einfaches Problem: Programmieren Sie ein Schriftband, bei dem die Zeichen nach links oder rechts wandern.

**Aufgabe:** Können Sie sich einen dreidimensionalen Array vorstellen? Bauen Sie sich einen mit LabVIEW! Können Sie ihm auch eine physikalische Interpretation geben?

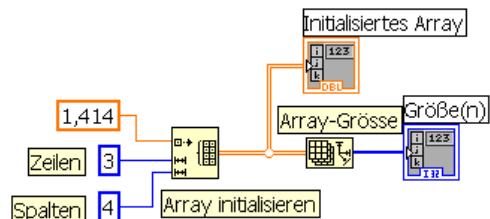
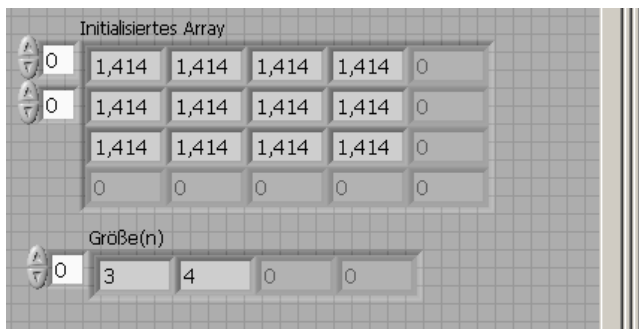
## Funktionen zur Bearbeitung von Feldern

LabVIEW bietet Ihnen unter dem Menüpunkt „Array“ eine ganze Reihe von Funktionen zur Bearbeitung von Feldern. Dazu gehören:

**Initialisierung eines Arrays** – im folgenden Beispiel wird mit dem VI „Array initialisieren“ eine 3x4-Matrix (3 Zeilen, 4 Spalten) erzeugt, deren Elemente alle den Wert 1,414 haben.

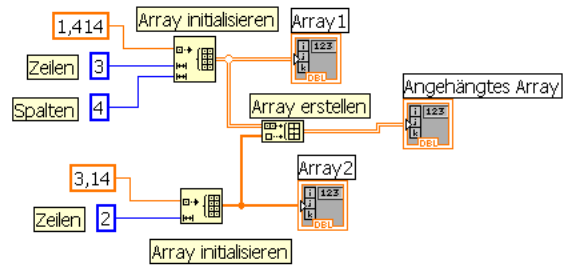
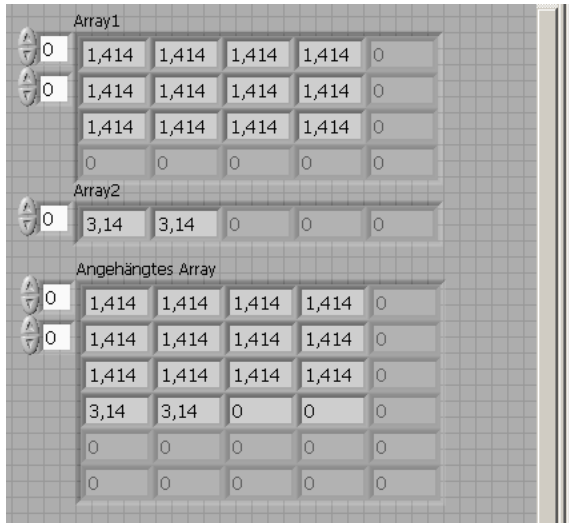


Die **Größe eines Arrays**, d.h. seine Ausdehnung in jeder Dimension, wird mit dem VI „Array-Größe“ bestimmt. Es gibt einen 1-dimensionalen Array zurück, bei dem das 0., 1., 2., ... Element jeweils die Größe in der 1., 2., 3., ... Dimension darstellt.



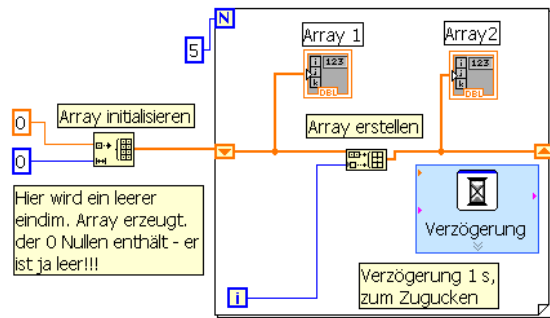
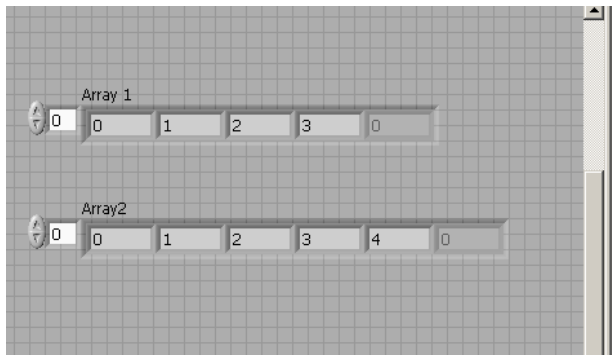
Das **Zusammensetzen von Arrays** aus Teil-Arrays oder durch Anfügen von weiteren Elementen erfolgt mit dem VI „Array erstellen“. Dabei gibt es je nach Konstellation der angeschlossener Elemente verschiedene Möglichkeiten:

Ein 2-dimensionaler und ein 1-dimensionaler Array werden wie folgt aneinander angehängt:



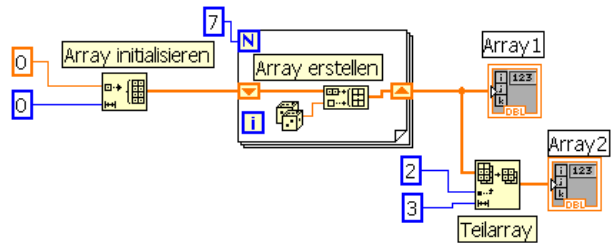
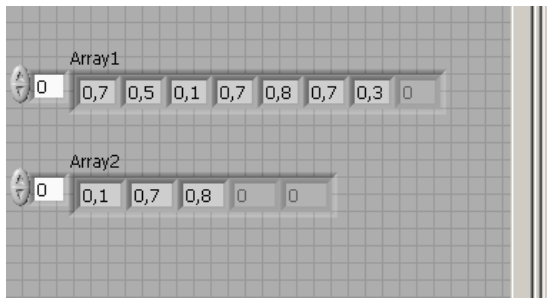
Im obigen Beispiel ist Array1 ein zweidimensionaler 3x4-Array und Array2 ein eindimensionaler 1x2-Array. Beim Anhängen wird der eindimensionale Array noch mit Nullen aufgefüllt, sodass man einen 1x4-Array erhält, der dann erst an Array1 angehängt wird.

Im unten abgebildeten Beispiel wird ein zunächst leerer 1-dimensionaler Array sukzessive mit den Werten der Laufvariablen  $i$  gefüllt. Woher kommen die verschiedenen Ergebnisse für Array1 und Array2.



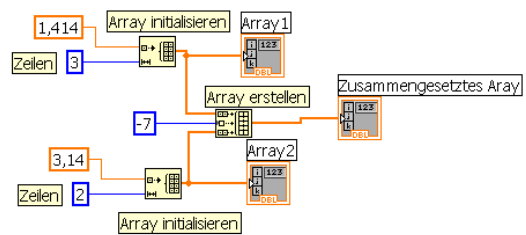
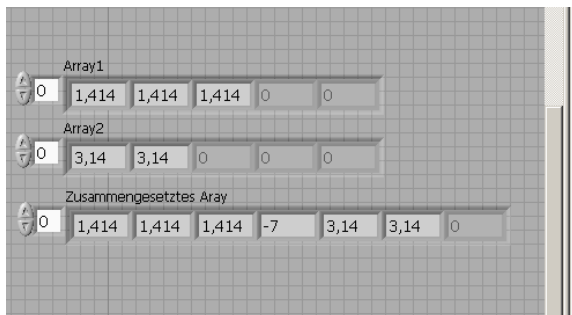


Mit dem SubVI „Teilarray“ kann man einem Array ab einem Index (2) eine gewisse Anzahl (3) von Elementen entnehmen?

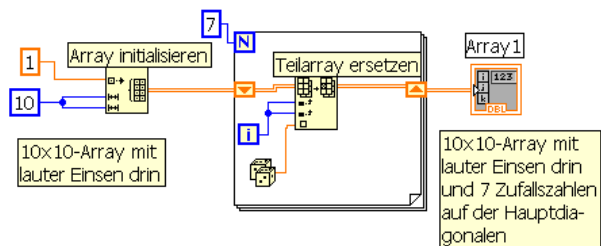
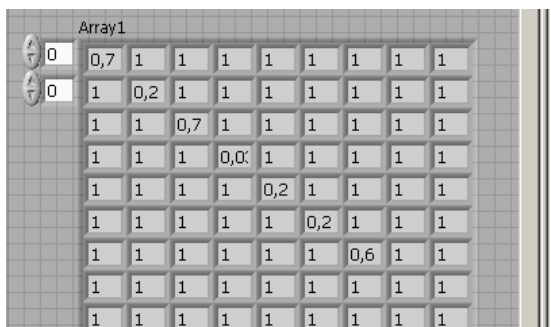


Will man auf ein ganz konkretes Array-Element zugreifen verwendet man das SubVI „Teilarray“, mit der man auch Teilbereiche eines Arrays auslesen kann:

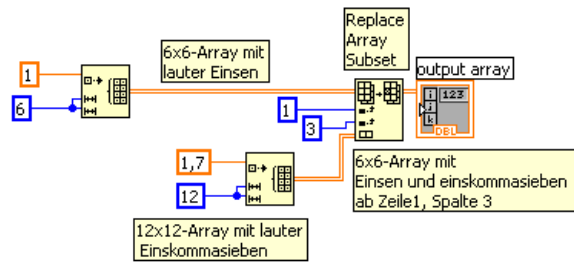
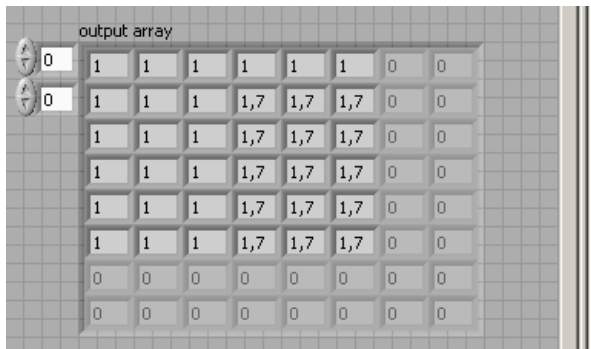
Die Eingänge des VIs „Array erstellen“ können entweder Arrays oder Skalare sein. Was passiert im folgenden Beispiel? Hier hat das VI „Array erstellen“ zwei eindimensionale Array-Eingänge und einen skalaren Eingang – beachten Sie die Symbolik! Es entsteht so ein neuer eindimensionaler Array durch direkte Aneinanderhängung (Concatenation).



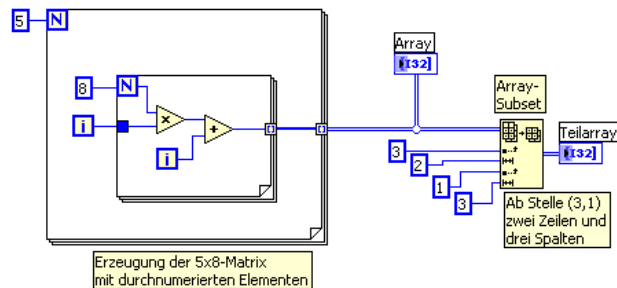
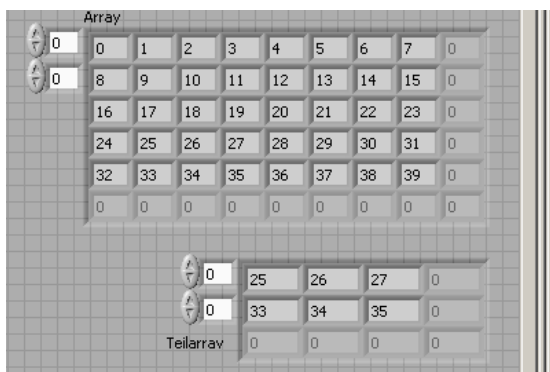
Mit dem VI „Teilarray ersetzen“ kann man einzelne Arrayelemente ersetzen durch andere Werte.



Mit dem VI „Teilarray“ ersetzen lassen sich aber auch ganze (rechteckige) Bereich eines Array ersetzen

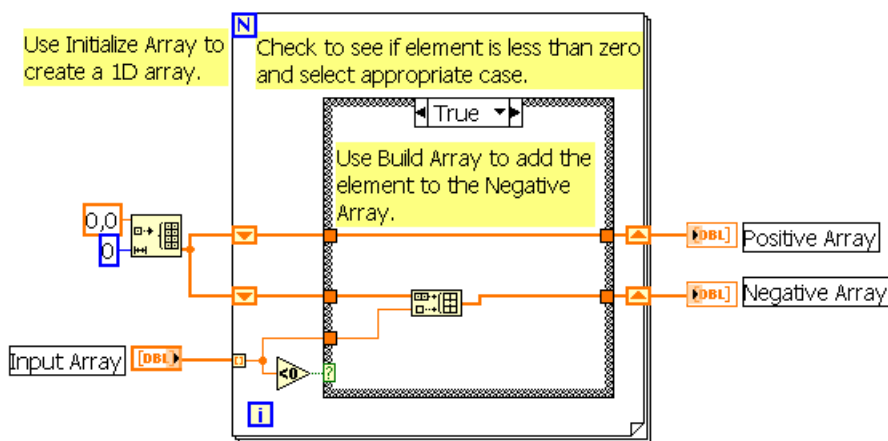


Wie kann man einem Array eine Spalte oder eine Zeile entnehmen? Hier eine verallgemeinerte Lösung:



**Aufgabe:** Experimentieren Sie mit den Funktionen „Build Array“, „Array Subset“, „Index Array“. Bei Anfangsschwierigkeiten hilft oft die Hilfefunktion!

Hier ein kleines Programm, das die negativen und nicht negativen Werte in einem Input-Array findet und in getrennte Arrays (Negative Array, Positive Array) schreibt:



**Aufgabe:** Gegeben sei die gewöhnliche Differentialgleichung  $dy/dx = y$ , deren allgemeine Lösung Sie ja sicher kennen. Bestimmen Sie im Intervall  $[0,1]$  eine numerische Lösung, die durch den Punkt  $(0,1)$  verläuft. (Hinweis: Euler-Cauchy) Variieren Sie die Schrittweite. Vergleichen Sie Ihre Werte mit der exakten Lösung. Schreiben Sie nun ein VI mit einem Formelknoten im Innern der Iterationsschleife zur Lösung beliebiger gewöhnlicher DGLen  $y' = g(x,y)$

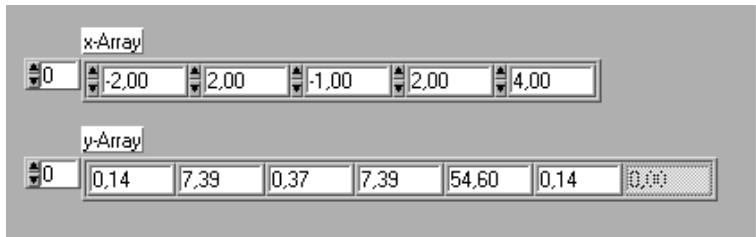
**Aufgabe:** Schreiben Sie ein Programm, das einen eindimensionalen Array umfüllt nach dem Prinzip: „Die letzten werden die ersten sein“.

**Aufgabe:** Schreiben Sie ein Programm, das 100 Würfe mit dem Würfel simuliert und dabei mitzählt, wie häufig jeder der 6 Werte gewürfelt wurde. Hinweis: Verwenden Sie ein Shift-Register für jeden der 6 Werte.

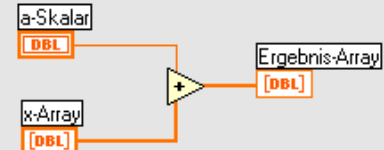
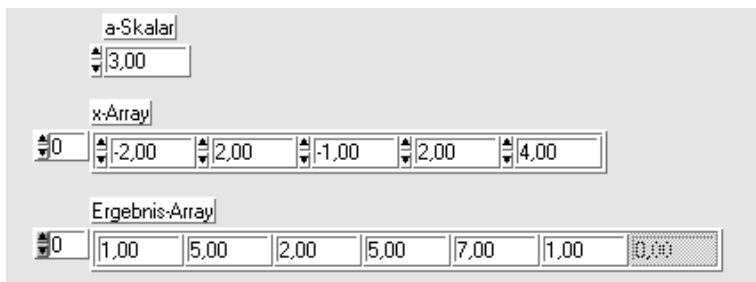
**Aufgabe:** Lösen Sie die vorige Aufgabe, indem sie in einer For-Schleife in einem sechselementigen Array an der nullten Stelle die Anzahl der gewürfelten Einsen, an der ersten Stelle, die Anzahl der gewürfelten Zweien, usw. eintragen und immer mehr Durchläufe machen. Wenn Sie statt der Anzahl im Array die relative Häufigkeit eintragen, dann müssten diese Häufigkeiten alle gegen die Zahl  $1/6$  konvergieren? Oder nein?

## Polymorphismus

Viele LabVIEW-Grundfunktionen haben eine weitere angenehme Eigenschaft, die mit „Polymorphismus“ bezeichnet wird. Polymorphismus bedeutet: „Vielgestaltigkeit“. In Zusammenhang mit der LabVIEW-Programmierung ist damit gemeint, daß die Eingänge der arithmetischen Funktionen ADD, MULTIPLY, DIVIDE, SUBTRACT und auch alle anderen Funktionen aus dem „Arithmetic“-Menüpunkt unterschiedliche Datentypen wie „numerics“ oder „arrays“ akzeptieren, und daß sie (die Funktionen) sich dann trotzdem „vernünftig“ verhalten, wie folgende Beispiele zeigen:



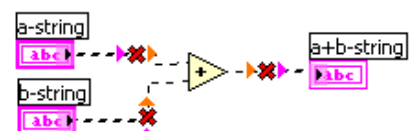
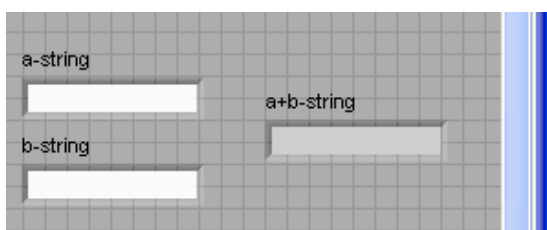
Beim folgenden Beispiel wird der eine Eingang der ADD-Funktion mit einem Skalar belegt, während am anderen Eingang ein Array liegt:



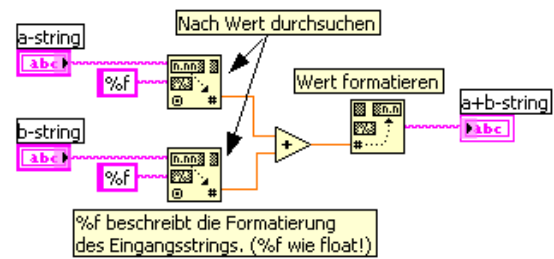
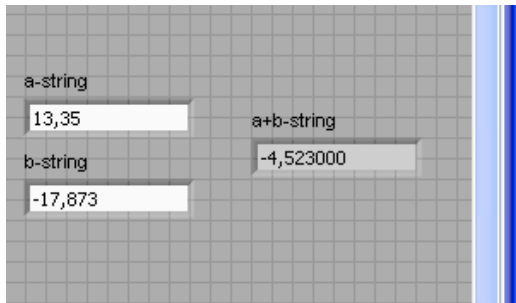
Ob gewisse Funktionen, die man verwenden will, sich polymorph im gewünschten Sinne verhalten, probiert man am besten aus.

## Eigene polymorphe Vis programmieren

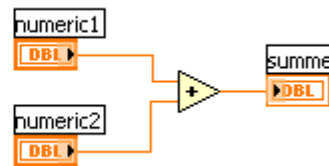
Sie können selbst auch selbstgebaute VIs zu polymorphen VIs zusammenfügen. Wie man das macht, soll hier an einem einfachen Beispiel erläutert werden. Wie Sie oben gesehen haben, kann der Plus-Operator auf alle möglichen Zahlentypen und Arrays von Zahlen angewendet werden. Wenn ich aber zwei Strings (selbst wenn sie eine Zahl darstellen) anlege, weigert er (der Plus-Operator) sich, diese zu addieren und das Ergebnis als String auszugeben.



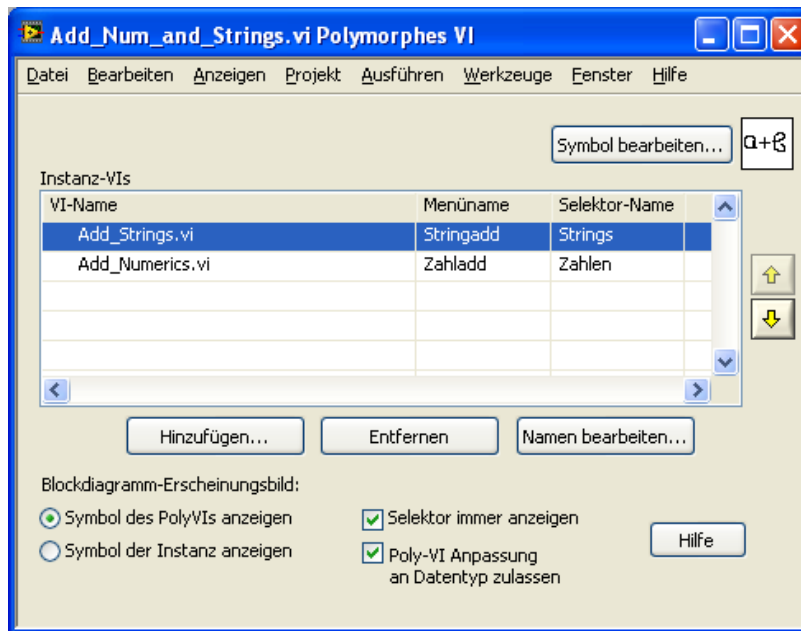
Wir programmieren nun zunächst ein SubVI, namens Add\_Strings, das als Eingabe Ziffernstrings erwartet, diese in Zahlen konvertiert, addiert und dann das in einen String zurückgewandelte Ergebnis ausgibt.



Dann erstellt man ein banales Add\_Numerics – Subvi:



Mit diesen beiden VIs baut man sich nun sein polymorphes VI zusammen und das geht folgendermassen. Über Datei/Neu... kommt man in ein Fenster, in dem man „Polymorphes VI“ auswählt. In der Tabelle, die man nun erhält, werden diese beiden VIs (AddStrings und AddNumerics) eingetragen. Polymorphe VIs kann man also wie Container betrachten, die andere VIs enthalten.



Die Bedeutung der einzelnen Auswahlpunkte werden Ihnen beim Anklicken wie bei der Kontexthilfe erläutert. Es gibt noch einige Details, die Sie sich selbst erarbeiten müssen, die aber keine grundsätzliche Schwierigkeit darstellen. Der Punkt „Poly-VI Anpassung an Datentyp zulassen“ ermöglicht Ihnen z.B. den unkomplizierten Anschluss der verschiedenen Datentypen, die dieses polymorphe VI verarbeiten kann. Natürlich müssen die ent-

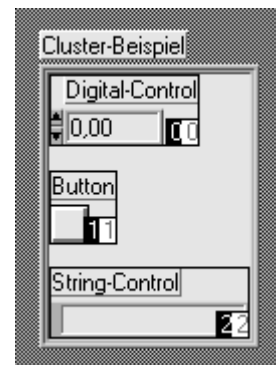
sprechenden Anschlüsse der Komponenten-VIs an entsprechenden Stellen liegen: Sie können z.B. nicht an einem Teil-VI an einer Stelle einen Ausgang haben, wo beim anderen Teil-VI ein Eingang liegt.

## Cluster

In Clustern werden unterschiedliche Datentypen zu einem neuen Datentyp zusammen gefaßt. Wie Arrays, wo es immer nur um denselben Datentyp geht, werden Cluster dadurch erzeugt, daß verschiedenartige Objekte innerhalb des Clusters abgelegt werden. Dabei gilt die Regel, daß es sich dabei ausschließlich um „Controls“ oder „Indicators“ handeln darf, daß also in einem Cluster nicht „Controls“ und „Indicators“ nebeneinander platziert werden können! Ein Cluster nimmt die Datenflußrichtung des ersten in ihm platzierten Objektes an.



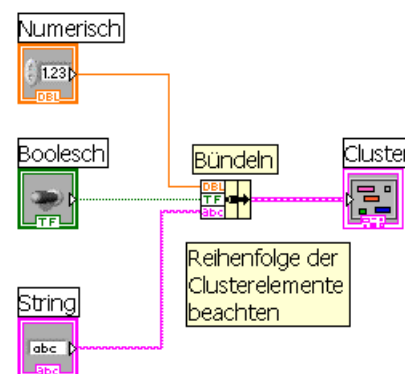
Clusterelemente haben eine logische Ordnung, die bei der Erzeugung des Clusters definiert wird: das erste eingefügte Element ist Element 0, das zweite eingefügte Element ist Element 1, usw. Elemente des Clusters können gelöscht werden, die Ordnung der Elemente kann geändert werden, wenn man mit der rechten Maustaste auf den Rand des Clusters klickt (s. rechts!). Auf die Elemente des Clusters wird über ihre Ordnung,



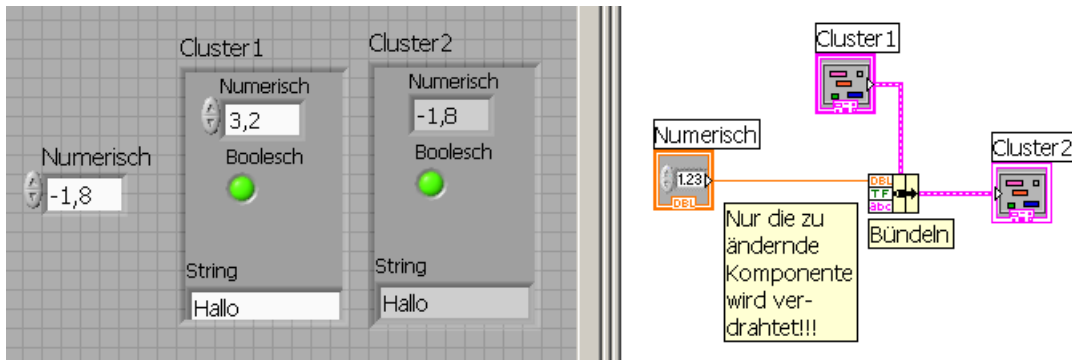
nicht über ihren Namen zugegriffen.

Cluster werden u.a. dann eingesetzt, wenn es darum geht, eine große Zahl von Parametern an ein VI zu übergeben. Es ist zwar möglich ein VI mit bis zu 28 Ein-/Ausgängen zu versehen, es ist aber ganz offensichtlich, daß dann sehr schnell die falschen Drähte an den falschen Kontakten angeschlossen werden.

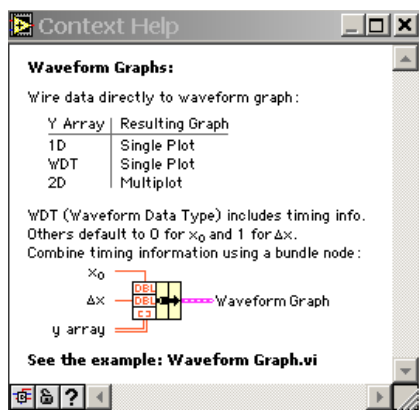
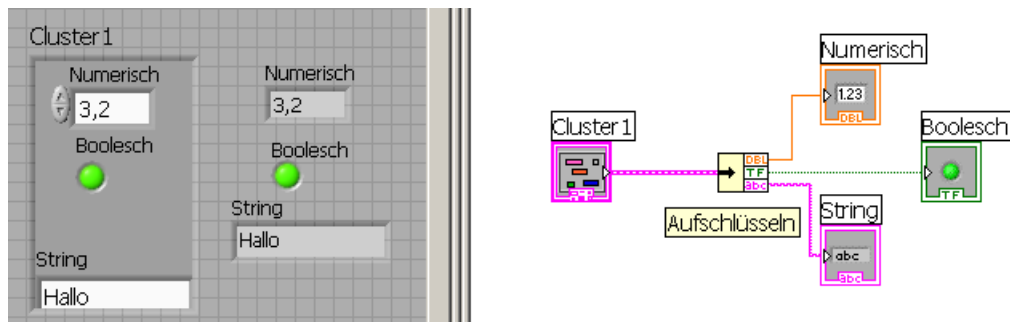
Die Bündelung der Daten erfolgt diagrammseitig unter Verwendung des „Bündeln-Operators“ (im Menü „Cluster“). Mit diesem Operator können entweder Daten zu einem neuen Cluster zusammengefaßt oder die Komponenten eines existierenden Bündels können mit neuen Werten belegt werden, wie im folgenden Beispiel vorgeführt wird.



Wenn man den Wert einer Komponente eines Clusters verändern will kann dies ebenfalls mit Hilfe des „Bündeln“-Operators erfolgen:



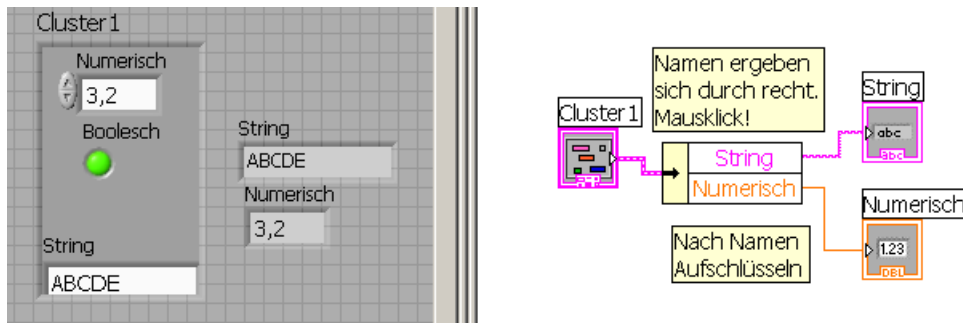
Die Zerlegung, die „Entbündelung“ eines Clusters erfolgt mit dem Operator „Aufschlüsseln“. Er spaltet den Cluster wieder in seine einzelnen Komponenten auf. Die Ausgangskomponenten werden entsprechend der Ordnung der Komponenten aufgeführt:



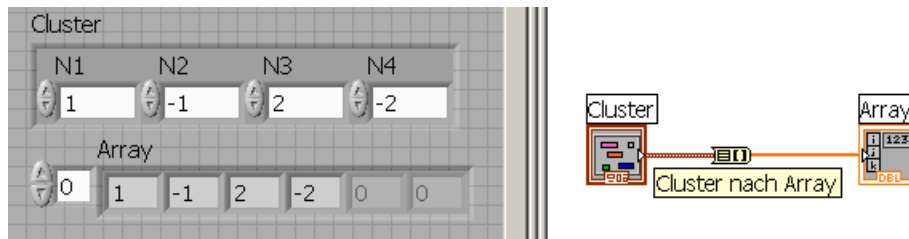
Der LabVIEW-Anfänger stolpert zum allerersten Mal über Cluster, wenn er versucht eine Messung in einem Graphen korrekt darzustellen. Meist werden ja Messwerte  $y_n$  bei einem Startzeitpunkt  $t_0$  beginnend in gleichen Zeitabständen  $\Delta t$  aufgenommen. Es gilt dann  $y_n = f(t_0 + n \cdot \Delta t)$ , (mit  $0 \leq n \leq N$ ), wobei unter  $f$  der jeweils vorliegende funktionale Zusammenhang zu verstehen ist. Wenn man dann die Zeitachse der Darstellung richtig beschriften will, muss man also dem entsprechenden Graphen drei verschiedene Dinge mitteilen:  $t_0$ ,  $\Delta t$  und den Array der Messwerte  $[y_n]$ . Diese drei Dinge werden in einem Cluster  $\{t_0, \Delta t, [y_n]\}$  zusammengefasst und dann an den LabVIEW-Graphen übergeben, so

wie es links dargestellt wird.

Die Bündelung und Aufschlüsselung kann auch „nach Namen“ erfolgen. Im Cluster-Menü finden Sie zwei Operatoren „Nach Namen aufschlüsseln“ und „Nach Namen bündeln“, die zu diesem Zweck verwendet werden können und sich eigentlich selbst erklären. Die Namen, nach denen aufgeschlüsselt werden kann bzw. soll, ergeben sich durch einen rechten Mausklick.



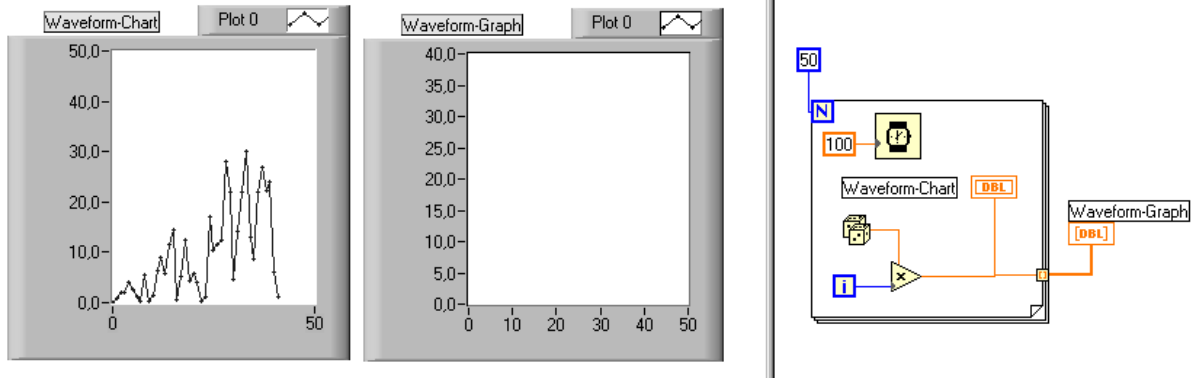
Im Menüpunkt Cluster gibt es noch zwei weitere Operatoren: „Cluster nach Array“ und „Array nach Cluster“. „Cluster nach Array“ konvertiert ein Cluster, das aus n gleichen Datentypen besteht in ein Array dieser Datentypen. Was „Array nach Cluster“ macht ist wohl ebenfalls intuitiv klar. Wozu das Ganze? Angenommen Sie haben z.B. ein Cluster von Booleschen Anzeigen, deren Reihenfolge geändert werden soll. Bei den Array-Operatoren gibt es eine ganze Menge von Operatoren, die alle möglichen Aufgaben lösen. So könnte man etwa aus dem vorliegenden Cluster eine Array erzeugen, das mit der geeigneten Array-Funktion bearbeitet wird und dann anschließend den neu erzeugten Array zurück in ein Cluster verwandeln.



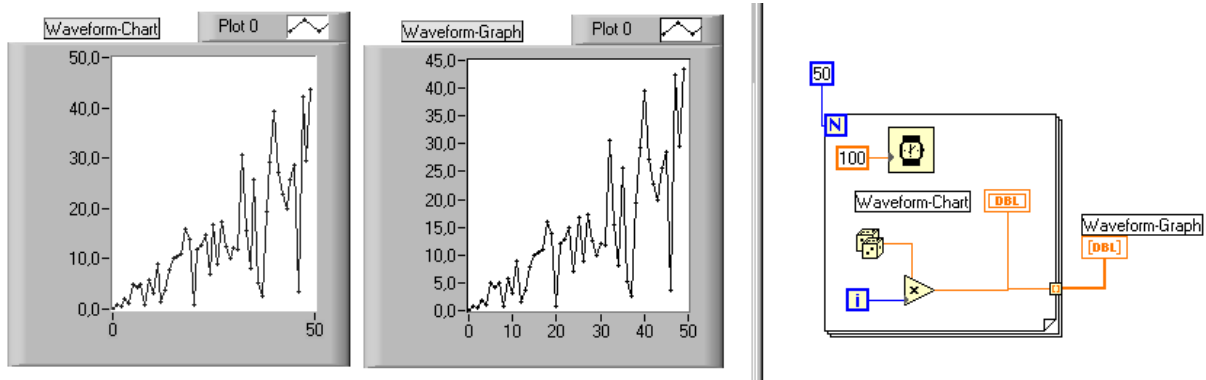


## Signalverlaufs-Diagramme und Signalverlaufs-Graphen

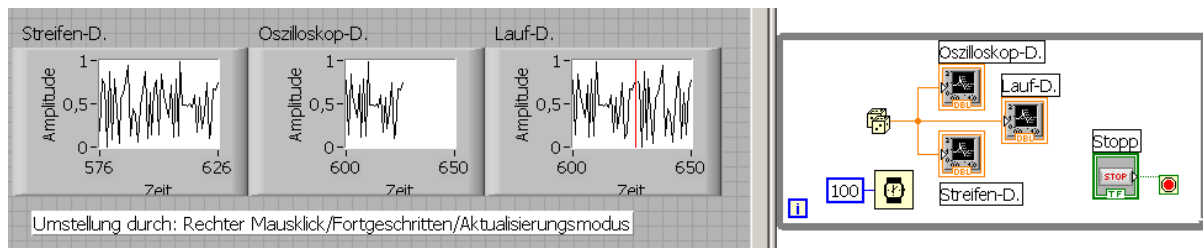
Mit LabVIEW können Sie die bei einer Messung erfaßten oder die bei einer Simulation berechneten Werte in sehr bequemer Weise in graphischer Form in Form von Diagrammen oder Graphen darstellen. Der Unterschied dabei ist, daß Diagramme (Charts) jeden neu erfaßten oder berechneten Meßwert sofort darstellen, während bei Graphen zunächst die Gesamtheit aller Werte erfaßt oder berechnet werden und dann erst dargestellt werden.



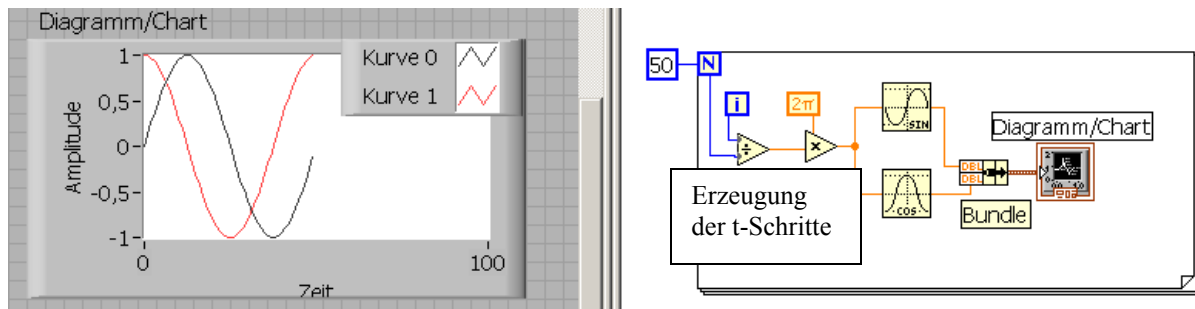
Oben wurde das Programm abgebrochen, so daß nur im Chart Punkte eingezeichnet sind und der Graph leer bleibt. Unten wurde abgewartet, bis das Programm zuende gelaufen ist: sowohl Chart als auch Graph zeigen den gleichen Kurvenverlauf an.



Wenn LabVIEW beginnt, Kurvenpunkte in einen Chart einzutragen, ist noch unbekannt, wieviele Messpunkte bis zum Ende der Messung insgesamt erfasst sein werden und wie gross und wie klein diese Messwerte im Laufe dieser Messung werden können. D.h.: manchmal überschreitet die Kurve den oberen oder unteren Rand, dann muss neu skaliert werden. Wenn man den rechten Rand überschreitet, gibt es drei verschiedene Möglichkeiten, weiterzumachen: man lässt die alten Punkte nach links wegrutschen, wie bei einem EKG-Schreiber (Streifendiagramm) oder der Darstellungsbereich wird vollständig gelöscht, bevor vom linken Rand ausgehend die nächste Folge der Messpunkte aufgezeichnet werden (Oszilloskopdiagramm) oder aber es werden immer etwa die letzten 100 Messpunkte dargestellt, die durch einen senkrechten Strich in alte und neue Werte getrennt werden. (Laufdiagramm). Auf dem folgenden Bild sind diese verschiedenen Darstellungsarten zu sehen.



In Charts, wie in Graphen können mehrere verschiedene Kurven parallel gezeichnet werden. Bei Charts müssen dazu die Werte mit dem Bundle-Operator gebündelt werden.



**Aufgabe:** Schreiben Sie ein Programm, das die beiden Funktionen  $f(x) = 1 - x^2$  und  $g(x) = \exp(-x^2)$  in einem Graphen darstellt. Für  $x$  soll gelten:  $-1 \leq x \leq 2$

## **Skalierung von Charts und Graphen**

Um ein möglichst großes Bild zu erhalten skaliert LabVIEW die x-Achse und die y-Achse automatisch. Dies ist eine Option, die ausgeschaltet werden kann, indem Sie mit der rechten Maustaste auf den Graphen klicken.

Mit der „Loose Fit“-Option können Sie die Achsen mit „schöneren“ Zahlen beschriften.

Mit der „Formatting...“-Option können Sie Gitter einzeichnen, den Skalenstil verändern, logarithmisch skalieren, u.v.a.m.

## **Die Legende**

Hier können Sie den Punktstil des Graphen (Pünktchen, Kreuze, u.s.w.), Linienstil (gestrichelt, u.s.w.), Farbe, und den Interpolationsstil für jede darzustellende Kurve einzelnen festlegen. Eventuell müssen Sie zunächst erst mal wieder mit einem rechten Mausklick auf den Graphen klicken um diese überhaupt anzuzeigen.

## **Die Palette**

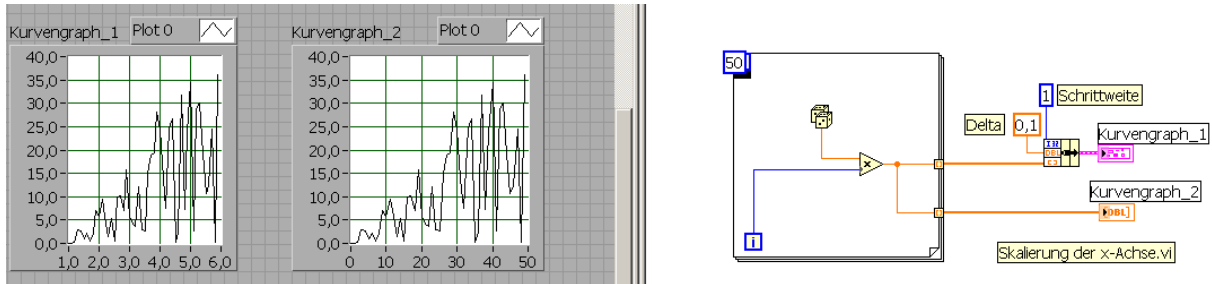
Damit können Sie Bildteile vergrößern („zoomen“).

**Aufgabe:** Erzeugen Sie mit einem Programm einen Funktionsverlauf und üben Sie dann damit die verschiedenen „features“ von LabVIEW, die eben beschrieben wurden.

## Einfach- und Mehrfach-Plot-Graphen

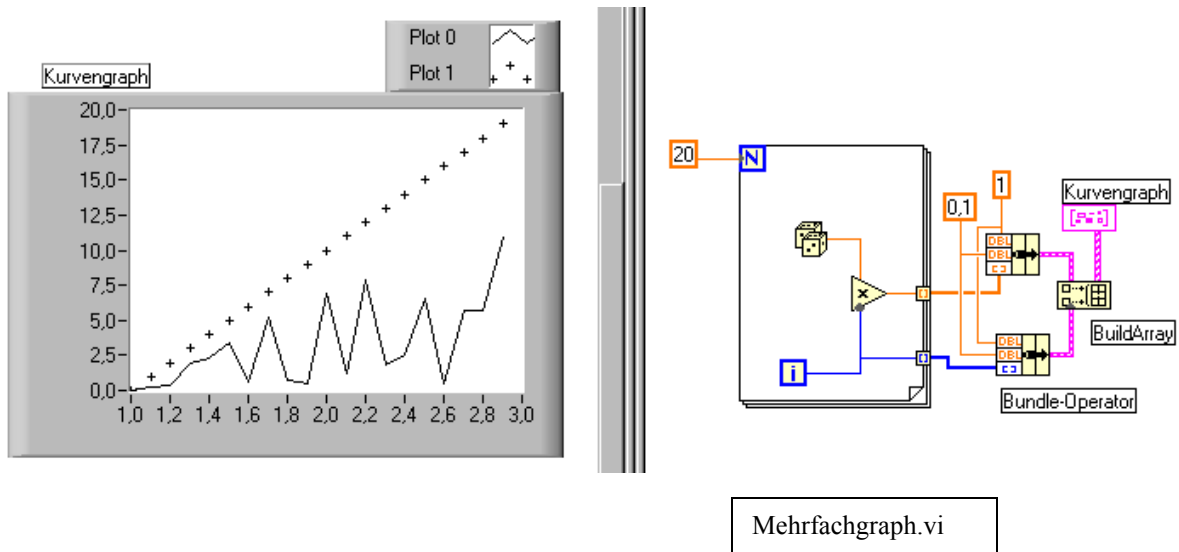
Wie schon gesagt: bei Graphen wird der Plot erst erstellt, wenn alle Meßpunkte erfasst, bzw. berechnet sind. Im folgenden sehen Sie zwei Beispiele. Das zweite Beispiel stellt einfach eine Folge von 50 y-Werten dar:  $[y_n]$ . Bei der Erstellung des Graphen wird dabei angenommen, daß zwei aufeinanderfolgende Abszissen immer den Abstand 1 haben und daß die erste Abszisse Wert 0 hat.

Im Beispiel unten wird zweimal dieselbe Folge von Meßpunkten erzeugt. Bei der Erstellung des Graphen wird aber bei Kurvengraph1 berücksichtigt, daß die Abszissen alle den Abstand „Delta“ haben und daß die erste Abszisse den Wert „Anfangswert“ besitzt.



Der obere Graph läuft in 0,1-er-Schritten von 1 bis 5,9, weil als Anfangswert 1 und als Schrittweite Delta = 0.1 übergeben wurden, während im unteren Graphen, der x-Bereich von 0 an in 1-er-Schritten bis 49 läuft.

Einen „Mehrfach-Waveform-Plot“ erhalten Sie indem Sie ein Array der verschiedenen Plots aufbauen, dabei müssen wieder die beiden Fälle von oben unterschieden werden:

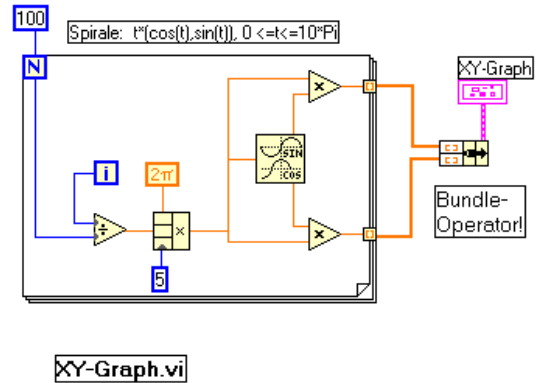
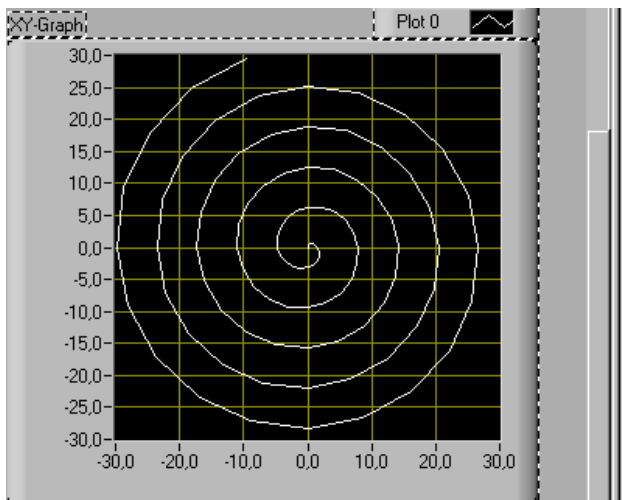


Im obigen Beispiel müssen zuerst die beiden Strukturen (Cluster) bestehend aus  $\{ t_0, \Delta t, [y_n] \}$  gebildet werden. In einem zweiten Schritt wird daraus ein Array (aus 2 Elementen) gebaut, das dann an den Graphen übergeben wird. Analog geht man dann vor, wenn drei und mehr Graphen dargestellt werden sollen.

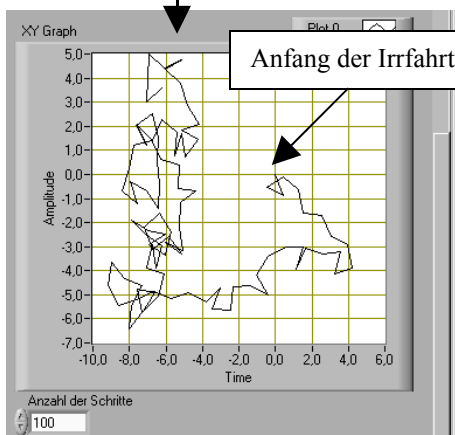
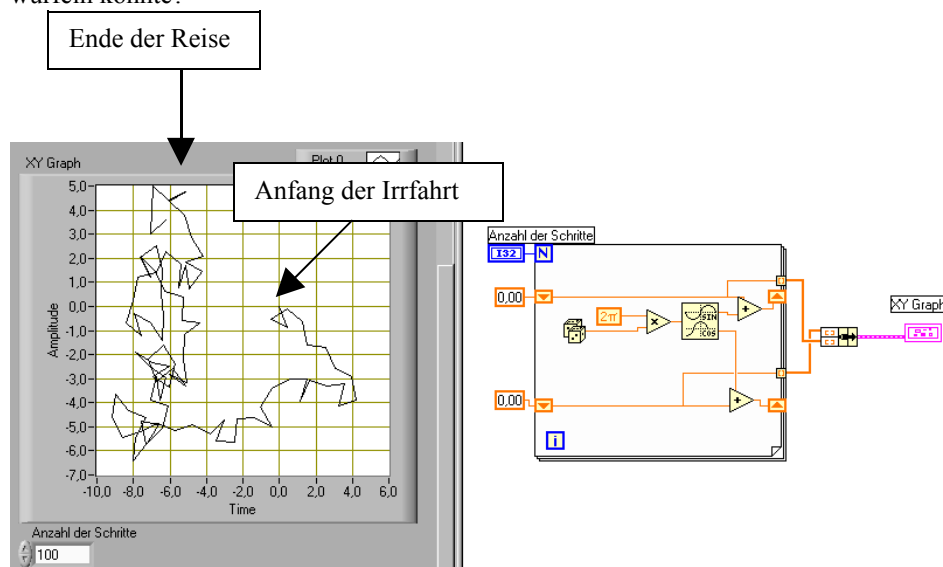
**Aufgabe:** Was passiert, wenn die beiden Cluster verschiedene Anfangspunkte und Schrittweiten enthalten?

## Einfach- und Mehrfach-Plot-XY-Graphen

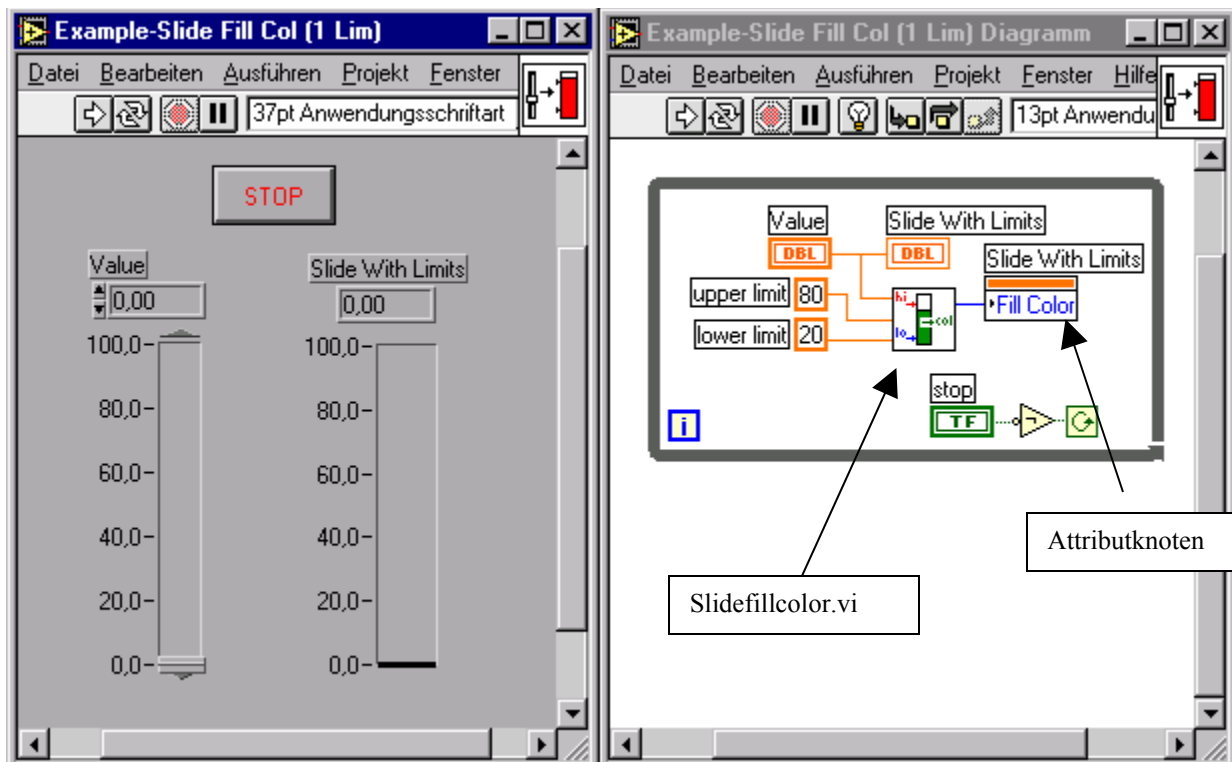
Bei XY-Graphen müssen die x-Werte nicht äquidistant sein. Die x-Werte und die y-Werte müssen in diesem Fall zu einer Struktur gebündelt und dann an den Graphen übergeben werden: Bei Mehrfachplots muß für jeden Plot (mit „bundle“ die Struktur erzeugt werden und dann (mit Build Array) zu einem Array zusammengefasst werden, bevor sie zur Ausgabe übergeben werden. XY-Graphen eignen sich insbesondere zur Erzeugung von Parameterdarstellungen von Kurven in der Ebene.



**Aufgabe:** Programmieren Sie eine Irrfahrt: Sie würfeln jede Sekunde eine Richtung (oben, unten, links, rechts) ,dann gehen Sie einen Schritt der Länge 1 in diese Richtung und würfeln dann eine neue Richtung, usw. Unten sehen Sie ein Beispiel für eine Irrfahrt, in der die gewürfelte Richtung ein beliebiger Wert zwischen 0 und  $2*\pi$  ist. Wie wäre das Programm, wenn der Irrfahrer nur die Möglichkeiten „Norden, Süden, Osten, Westen“ würfeln könnte?



## Attributknoten zum Steuern von Bedienelementen und Diagrammen



Die in LabVIEW eingesetzten Anzeige- und Bedienelemente haben noch spezielle Attribute (Farbe, Position, Blinkmodus, Sichtbarkeit, usw.), die vom Programm selbst aus gesteuert werden können. Im obigen Beispiel sehen Sie zwei Schieberegler, einer im Eingabe- der andere im Anzeigemodus („Value“ und „Slide with limits“). Je nach Wert von „Value“, soll sich die Anzeigefarbe des rechten Reglers von blau nach grün nach rot ändern (denken Sie an die Kühlwassertemperaturanzeige in einem Auto). Auf der Diagrammseite sehen Sie den Attributknoten „Slide With Limits“ bei dem das Attribut „Fill Color“ ausgewählt wurde. Der Wert von „Fill Color“ wird vom VI Slidefillcolor.vi in Abhängigkeit vom aktuellen Wert und der oberen und unteren Grenze bestimmt und an das Attribut übergeben. Einen Attributknoten zu einem bestimmten Objekt erhält man über Rechten Mausklick/Erstelle/Attributknoten. Auf den im Diagramm erschienen Attributknoten geht man dann wieder über Rechten Mausklick/Objekt wählen zu dem konkreten Attribut, das beeinflusst werden soll.

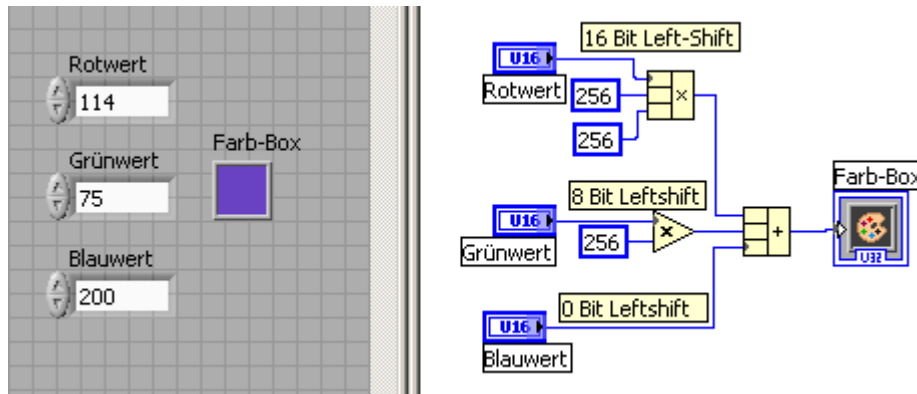
**Aufgabe:** Ändern Sie das obige Programm so ab, daß der rechte Regler blinkend oder nicht blinkend dargestellt werden kann.

**Aufgabe:** Farben (für das obige VI Slidefillcolor) werden oft als Mischung von drei Grundfarben Rot, Grün, Blau dargestellt. Ein Farbwert einer Farbe setzt sich wie folgt aus drei Bytes zusammen. Das heisst man hat für jede Grundfarbe 256 Werte. (Wieviel Farbwerte insgesamt ergibt das?)

R R R R R R R R G G G G G G G G B B B B B B B B

Wie kommt nun das Byte mit dem roten Farbwert an die richtige Stelle? Man muss es um 16 Bit nach links verschieben! Wie verschiebt man um 1 Bit nach links? Man multipliziert mit 2! Wie verschiebt man um 2 Bit? Wie verschiebt man also um 16 Bit? Wir finden also Farbwert = Rotwert\* $2^8$ \* $2^8$  + Grünwert\* $2^8$  + Blauwert

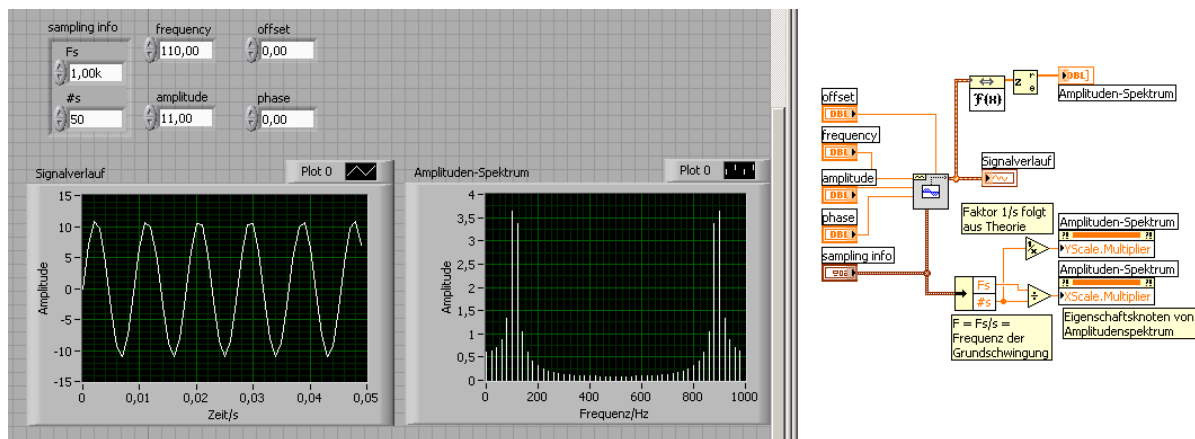
Probieren Sie es aus mit LabVIEW. Sie benötigen Frontpanelseitig dazu eine Farb-Box. Die Übertragung in ein LabVIEW-Programm sehen Sie hier:



Eine weiteres Einsatzfeld von Eigenschaftsknoten haben wir schon früher kennengelernt. Es ging dabei darum die Frequenzachse eines Leistungsspektrums richtig zu beschriften. Der Zusammenhang geht wie folgt: aus Abtastrate  $S_f$  (Sampling-Frequenz) und aus Anzahl der erfassten Messpunkte  $s$  (samples) ergibt sich die Zeitspanne

$T = s/S_f$  in der unser Signal betrachtet wird, denn  $1/S_f$  ist ja gerade der Zeitunterschied zwischen zwei direkt aufeinanderfolgenden Messpunkten.  $1/T = S_f / s$  ist dann die Frequenz der Grundschwingung bei der Fourieranalyse. Ausserdem ergibt sich in der Fouriertheorie noch ein Faktor von  $1/s$  für die y-Achse.

Mit den entsprechenden Multiplikationsfaktoren ergibt sich im folgenden Programm eine genaue Übertragung der Fouriertheorie in die Darstellung des Spektrums. Ohne diese Berücksichtigung würde man im Amplitudenspektrum immer nur  $s$  Spektrallinien sehen, ohne zu wissen, welcher Frequenz sie entsprechen.



## Erzeugung und Veränderung von Zeichenketten (Strings)

Viele computergesteuerte Geräte in der Messtechnik erwarten Steueranweisungen vom Rechner in Form von Zeichenfolgen („Set Range 10 mV“) und ebenso geben sie ihre Messergebnisse in Form von Zeichenfolgen an den Computer zurück. Daher steht man oft vor der Aufgabe Zahlen in Zeichenketten und umgekehrt Zeichenketten in Zahlen umzuwandeln.

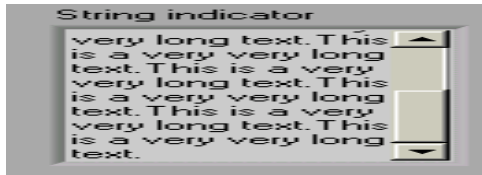
Die Formatanweisung bei der Konvertierung von Zahlen in Strings ist denjenigen, die C kennen, sicher geläufig. Für diejenigen unter ihnen, die sie noch nicht kennen, hier das Wesentliche, das Sie besser verstehen, wenn Sie gleichzeitig, das nachfolgende Programm („formatundsoweiter.vi“) erzeugen um die jeweiligen Angaben gleich nachzuprüfen. Eine Formatanweisung in LabVIEW hat folgende Syntax:

[String]%-[0][Stringbreite][.Stringgenauigkeit]Umwandlungskennung[String]

Ausdrücke in eckigen Klammern [ ] sind dabei optional, d.h. Sie können, müssen aber nicht, Bestandteil einer Formatanweisung sein. Da fast alles (bis auf „%“ und „Umwandlungskennung“ optional ist, heißt das zunächst: eine Formatanweisung muß mindesten ein %-Zeichen und eine Umwandlungskennung enthalten! Die folgende Tabelle erklärt Ihnen die einzelnen Elemente der obigen Syntaxregel:

Syntaxelement	Beschreibung
String	Zeichenfolge, die gewisse der unten beschriebenen Zeichen enthalten kann
%	Zeichen, das die Formatspezifizierung einleitet
- (Bindestrich)(optional!)	Erzwingt Ausrichtung an der linken Kante
0 (zero)(optional)	Zeichen, das freien Raum links von der Zahl mit Nullen, statt mit Abständen („spaces“) auffüllt
Stringbreite (optional)	Mindestbreite des Felds, das die konvertierte Zahl enthalten soll. Mehr Platz wird verwendet, falls nötig. Freier Platz wird von LabVIEW mit Spaces aufgefüllt und zwar links oder rechts der Zahl, je nach Ausrichtung. Bei fehlender Stringbreite wird so viel Platz wie nötig bereit gestellt.
.(Dezimalpunkt – Komma-)	Zeichen, das die Stringbreite von der Stringgenauigkeit trennt
Stringgenauigkeit	Zahl, die die Anzahl der Stellen rechts vom Dezimalpunkt festlegt, wenn die zu konvertierende Zahl eine Floating-Point-Zahl ist. Wenn keine Angabe zur Genauigkeit erfolgt, werden 6 Stellen ausgegeben.
Umwandlungskennung	Einzelnes Zeichen, das festlegt, wie die Zahl zu konvertieren ist: d            Dezimal Integer x            Hexadezimal Integer o            Oktal Integer f            Floating-Point-Zahl normal e,g        Floating-Point-Zahl in wiss. Schreibweise  Bemerkung: Diese Umwandlungskennungen können groß oder klein geschrieben werden.





Hexdarstellung, \Code und Passwortmodus)

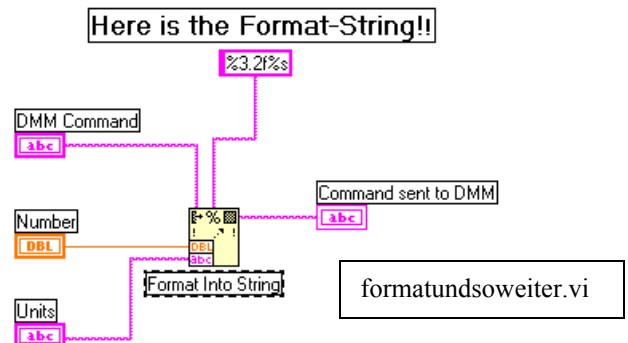
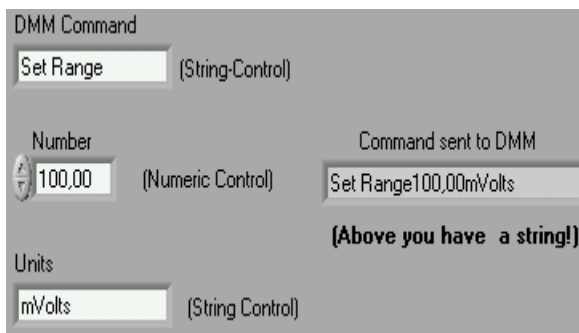
Ein String-Indicator oder ein String-Control kann übrigens auf die übliche Art und Weise vergrößert oder durch Klick auf den Rand auch mit einem Scroll-Balken versehen werden. Auch die Darstellungsform der Zeichen kann verändert werden (normal,

**Aufgabe:** Schreiben Sie ein kleines Programm, das in einem ersten Schritt die Eingabe von Username und Passwort erfordert.

Im Zusammenhang mit Zeichenfolgen gibt es drei wichtige Aufgaben:

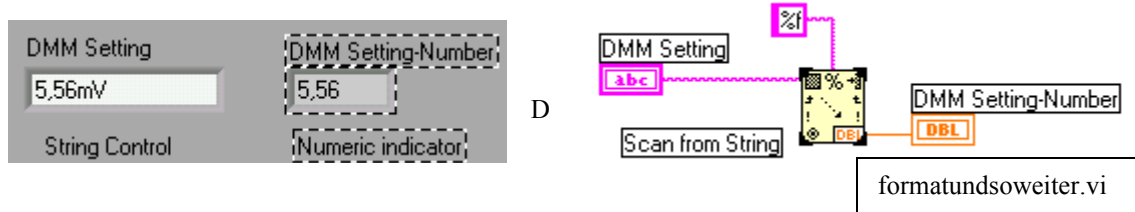
- **Format Into String:** Aneinandersetzen und Formatierung von Strings und Zahlen zu einem Gesamtstring.
- **Scan From String:** Durchsuchen einer Zeichenfolge nach gültigen Ziffernfolgen (0...9, A...F, a...f und, nicht vergessen: das dezimale Trennzeichen Komma oder Punkt, je nach dem, in welcher Welt (USA vs. Old Europe) Sie leben. Und diese Zeichenfolgen müssen dann noch in Zahlen (natürliche oder Floating-pointzahlen) konvertiert werden.
- **Match Pattern:** Darunter versteht man das Durchsuchen eines Strings, beginnend bei einem bestimmten Offset und die Aufteilung dieses Strings in drei Sub-Strings: den Substring vor dem gesuchten Muster, das Muster selbst und den Reststring danach.

Unten sehen Sie ein Programmfragment, das zur Kommunikation mit einem digitalen Multimeter (DMM) dient. Um etwa den Messbereich auf 100 mV einzustellen erwartet das DMM einen String der Form: „SetRange100mVolts“. In LabVIEW bastelt man sich so einen String wie folgt zusammen:

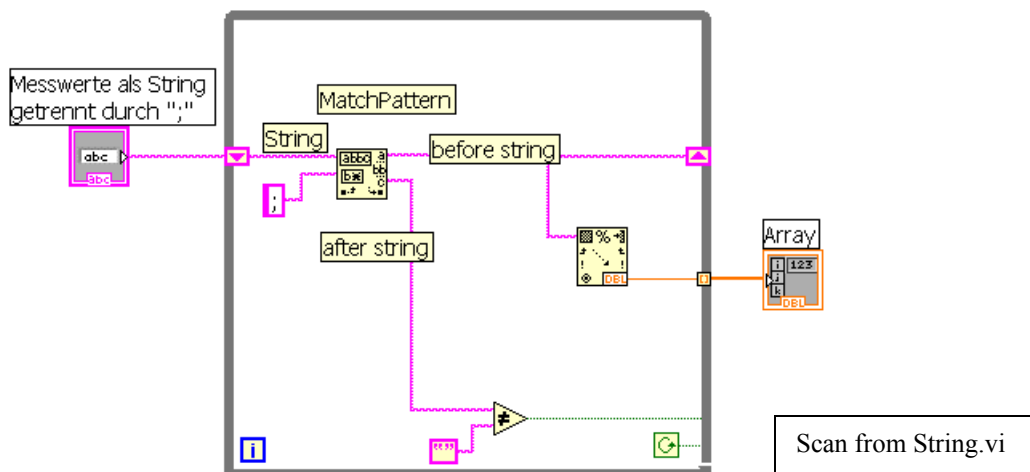


Die wesentliche Routine im Diagramm ist dabei das Format Into String.vi, das Sie im String Sub-Menü finden. Mit einem Rechtsklick auf das VI kommen Sie in einen Edit-Modus, der Ihnen hilft den richtigen Format-String zu finden – damit können Sie dann die vorige Seite über die Syntax von Format-Anweisungen vergessen.

Nun geht es in die entgegengesetzte Richtung. Ihr DMM liefert Ihnen, nachdem die Messung durchgeführt worden ist, den Messwert in Form einer Zeichenfolge und Sie müssen daraus wieder eine Zahl machen, mit der man rechnen kann. Das für diesen Zweck zu verwendende VI heisst: Scan from String.vi. Experimentieren Sie damit!



**Aufgabe:** Angenommen Sie erhalten eine Serie von Messwerten als String, in dem die einzelnen Messwerte durch Semikolon getrennt sind. Schreiben Sie ein Programm, das diese Messwerte in einem Array von reellen Zahlen ablegt. Überlegen Sie sich zunächst Ihr eigenes Programm, versuchen Sie das untenstehende Programm zu verstehen und überlegen Sie anschliessend, ob die Aufgabe noch einfacher gelöst werden könnte. Aber zuerst müssen Sie den darin enthaltenen Fehler beheben!!!



## Ein- Ausgabe in Dateien

Die Funktionen, die benötigt werden, um Dateien auf die Festplatte zu schreiben oder von der Festplatte zu lesen finden Sie unter dem Menüpunkt „File I/O“. Im wesentlichen gibt es vier verschiedene Dateitypen, in die man mit LabVIEW schreiben, bzw. aus denen man lesen kann:

- Tabellenkalkulationsdateien
- Textdateien
- Binärdateien
- TDM-Dateien (Das TDM-Dateiformat (Technical Data Management) ist ein strukturiertes, für die Ablage technischer Daten und für die anschließende Datenauswertung optimiertes Format. Es wurde entwickelt, um alle zu einer Messung oder Simulation gehörenden Daten und Zusatzinformationen erfassen und verwalten zu können. TDM-Dateien werden etwa von DIAdem (der Messdatenverarbeitungssoftware von National Instruments benutzt).

Wenn die Funktionen zum Schreiben solcher Dateien keinen Pfadnamen zur betreffenden Datei erhalten, wird dieser Pfad abgefragt.

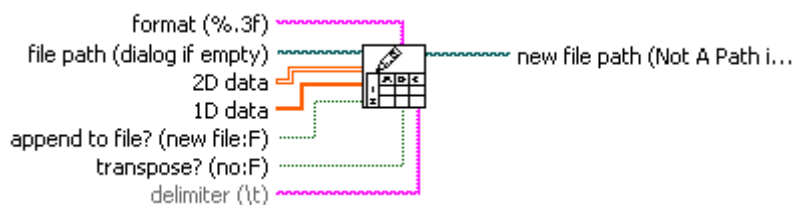
Eine weit verbreitete Anwendung besteht darin, daß Dateien so abgespeichert werden, daß sie von einem Tabellenkalkulationsprogramm geöffnet werden können. Meistens werden in solchen Programmen Spalten durch Tabulatoren getrennt und Zeilen durch EOL's („End of Lines“). Die beiden Funktionen „Write To Spreadsheet File“ und „Read from Spreadsheet File“ behandeln diesen Fall.

Die erzeugten Dateien sind aber auch oft Textdateien, die mit jedem Editor einsehbar und bearbeitbar sind.

In Binärdateien werden die Dateien in binärer Form abgespeichert. Das erlaubt eine kompaktere Speicherung, was besonders bei einem hohen Aufkommen an Messpunkten von Bedeutung sein kann.

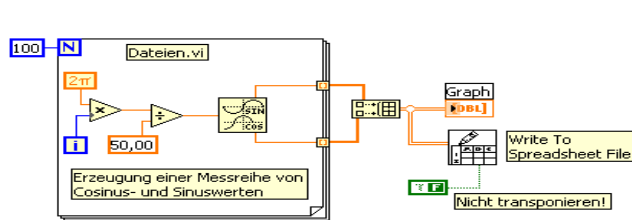
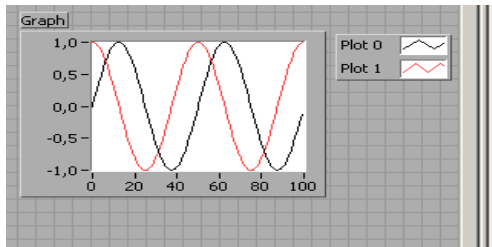
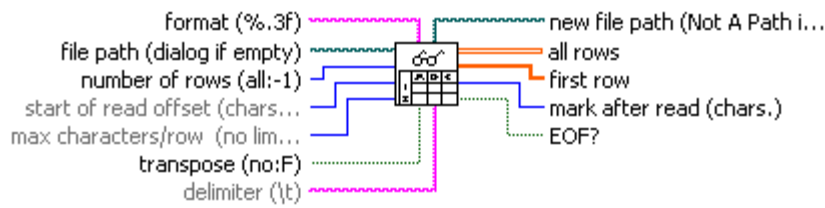
**Aufgabe:** Experimentieren Sie mit den im Folgenden beschriebenen VI's. Vielleicht haben Sie Meßwerte aus dem AP oder dem PL, die Sie mit Hilfe eines Editors eingeben könnten, um Sie anschließend mit LabVIEW weiterzuverarbeiten?

Die „Write To Spreadsheet File“-Funktion (Stift!) hat folgende Parameter:



Dies Funktion konvertiert einen ein- oder zweidimensionalen Array von Floatingpointzahlen in Zeichen, die in die Datei geschrieben werden. Dabei wird entweder eine neue Datei erzeugt oder an die bestehende Datei angehängt. Wenn file path nicht belegt wird, erscheint ein Dialog, der nach dem Dateinamen und -pfad fragt. Die so entstandene Datei kann von fast allen Tabellenkalkulationen gelesen werden.

Die „Read From Spreadsheet File“-Funktion (Brille) hat folgende Parameter:



**Aufgabe:** Untersuchen Sie das und experimentieren Sie mit dem obige(n) Programm

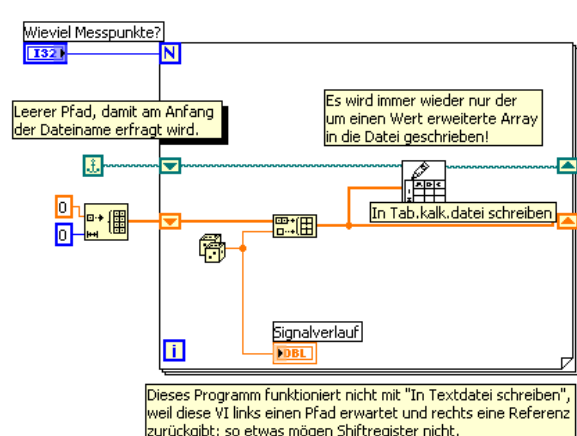
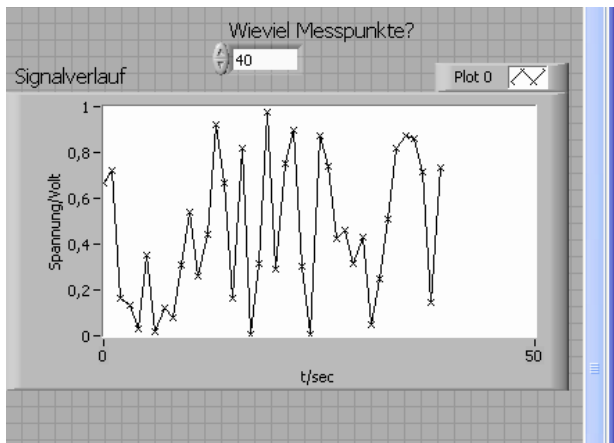
Öffnen Sie die dabei erzeugte Datei mit einem Tabellenkalkulationsprogramm und sehen Sie nach, was Sie erhalten haben. Was ändert sich, wenn die logische Konstante („Transponieren“) auf „false“ gesetzt wird?

Schreiben Sie ein neues Programm, in dem Sie diese eben erzeugten Werte aus der Datei zurück in das Programm einlesen und graphisch darstellen. Ändern Sie das Programm so ab, dass zu jedem Messwert die laufende Nummer und die Einheit (z.B. „MegaWatt“) mit abgespeichert wird.

Schreiben Sie ein neues Programm, in dem Sie diese eben erzeugten Werte aus der Datei zurück in das Programm einlesen und graphisch darstellen. Ändern Sie das Programm so ab, dass zu jedem Messwert die laufende Nummer und die Einheit (z.B. „MegaWatt“) mit abgespeichert wird.

### Schreiben von Messdaten in eine Datei

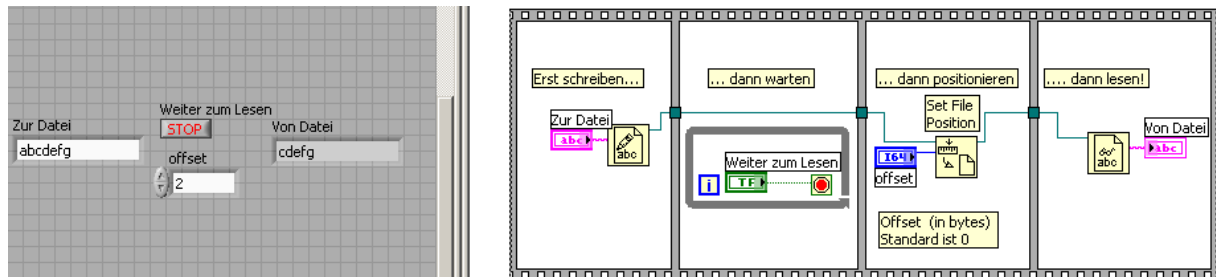
Angenommen Sie machen Temperaturmessungen und wollen eine Protokolldatei erzeugen, in der alle Ihre Messwerte enthalten sind. Das folgende einfache LabVIEW-Programm erledigt diese Aufgabe (nur die Temperaturwerterfassung darin wird mit dem Zufallsgenerator simuliert).



Verstehen Sie das Programm? Wozu der Aufwand mit den Shiftregistern im Zusammenhang mit dem Pfadnamen? Stellen Sie sich vor, die Messwerte würden sehr oft gescannt (z. B. 10000 mal pro Sekunde), verschluckt sich dann der Rechner, bei der Aufgabe, die Messwerte auf die Platte zu schreiben?

## Das war nur der Anfang vom Anfang

Wenn Sie den Menüpunkt File I/O anwählen finden Sie noch eine ganze Reihe weiterer Funktionen um mit Dateien zu arbeiten. Insbesondere gibt es den Punkt Adv File Funcs. Hier gibt es u.a. eine Funktion um den Filepointer, das ist die Stelle in der man sich aktuell in der Datei grade befinden zu ermitteln und neu zu positionieren:



Unter den erweiterten Funktionen finden sie z.B. auch eine Funktion, mit der sie Dateien komprimieren können. Desgleichen gibt es einen Untermenüpunkt „Dateikonstanten“, wo Sie z.B. den leeren Pfad aus dem Beispiel auf der vorigen Seite finden können. Man kann so Dateipfade erstellen und zerlegen, Pfadkonstanten definieren und noch einige andere Aufgabe im Zusammenhang mit Dateien erledigen.

Alle weiteren Funktionen aus den vielen angebotenen Möglichkeiten, die man sonst noch im Zusammenhang mit Dateien und Dateisystemen benötigen könnte, sollten Sie in Ruhe studieren, indem Sie sich selbst einfache Testprogramme zusammenbauen.

## Einige Worte über Waveforms (zu deutsch: Signalverläufe)

Wie Sie schon bei der Darstellung von irgendwelchen Messkurven mit dem Signalverlaufsgraphen gesehen haben, muss ein gewisser Aufwand betrieben werden, um die x-Achse eines solchen Graphen richtig zu beschriften. Bis jetzt wissen wir nur, dass für eine korrekte Darstellung nicht nur die jeweiligen Messwerte benötigt werden, sondern auch der Startwert und die Schrittweite zwischen zwei aufeinanderfolgenden Messpunkten. Man packt diese drei Angaben zusammen in einen Cluster und der Signalverlaufsgraph weiss dann, was zu tun ist.

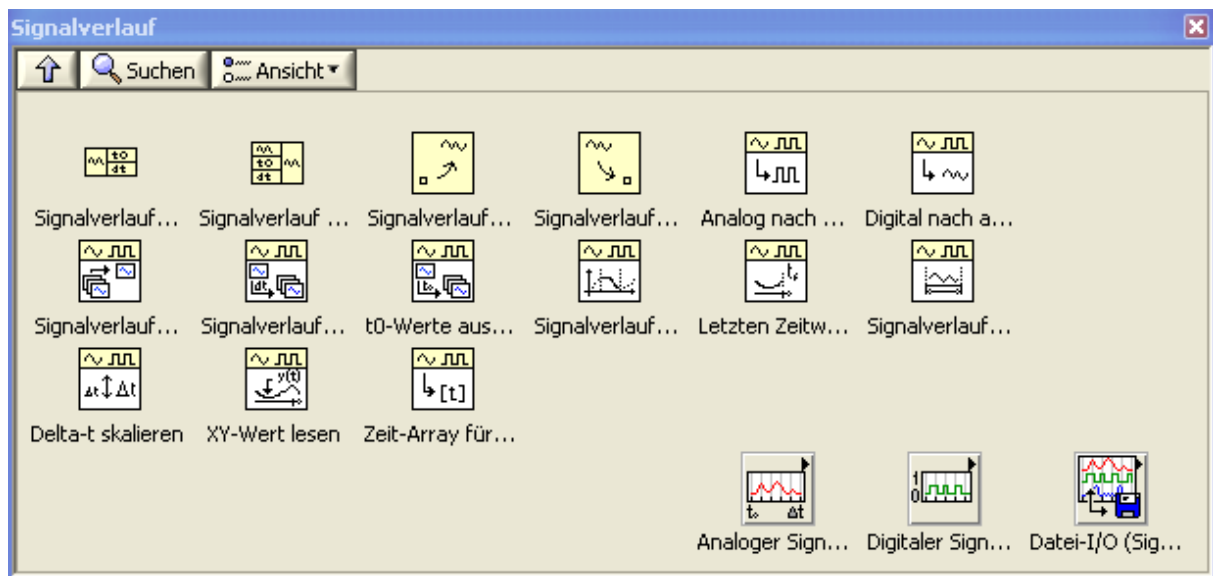
Signalverläufe (Waveforms) sind eine spezielle Art von Cluster, die aus eben diesen Komponenten bestehen:

$t_0$ , als Startwert

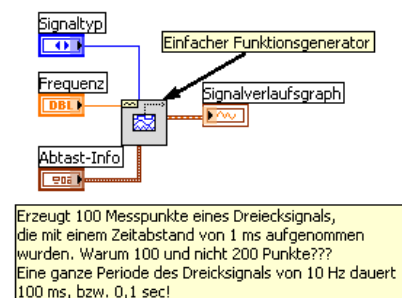
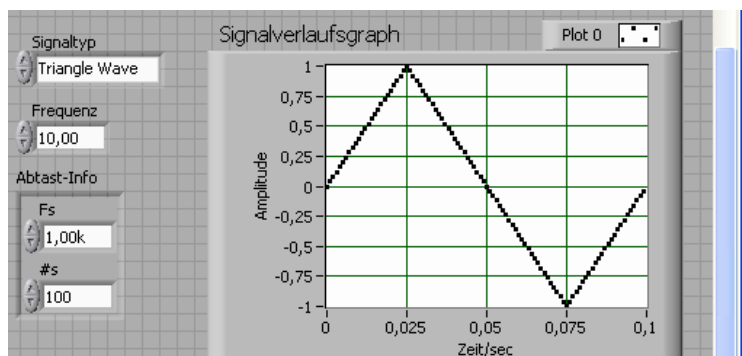
$\Delta t$ , als Schrittweite

Y, als Array der aufgenommenen Messwerte

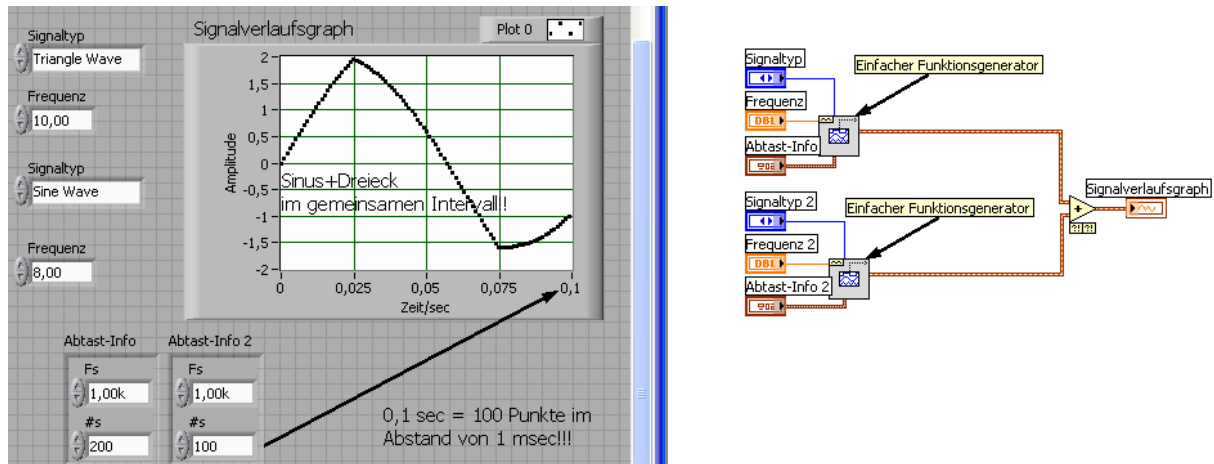
Seltsamerweise kann man zum Bearbeiten dieses Clusters nicht die schon bekannten Clusterfunktionen (Bundle, Unbundle) verwenden sondern benötigt eigene von LabVIEW dafür bereitgestellte Funktionen, die Sie unter dem Menüpunkt „Funktionen/Signalverlauf“ finden können:



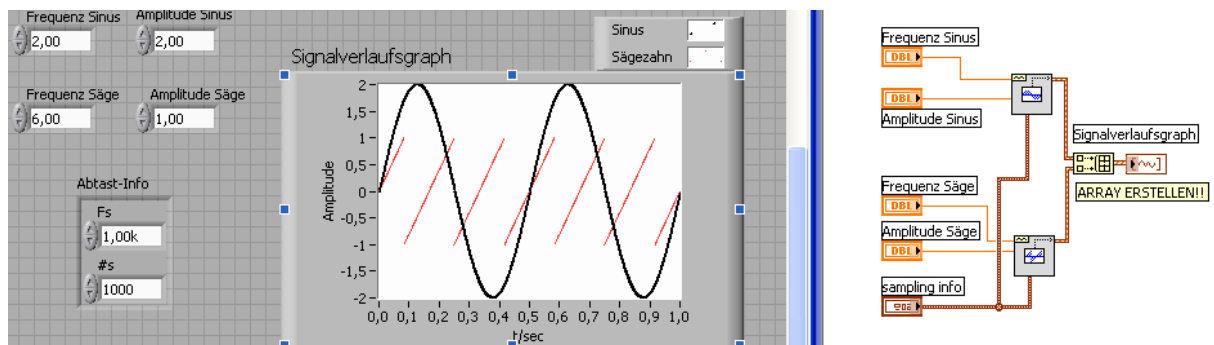
Signalverläufe und der Signalverlaufsgraph gehören zusammen! Wenn man vom obigen Menü in das Untermenü Analoges Signalverlauf/Signalverlaufserzeugung hinabsteigt kommt man zu einer Reihe von VIs, mit deren Hilfe man häufig benötigte Signalverläufe, wie Sinussignal, Rechtecksignal, Rauschen usw. erzeugen kann.



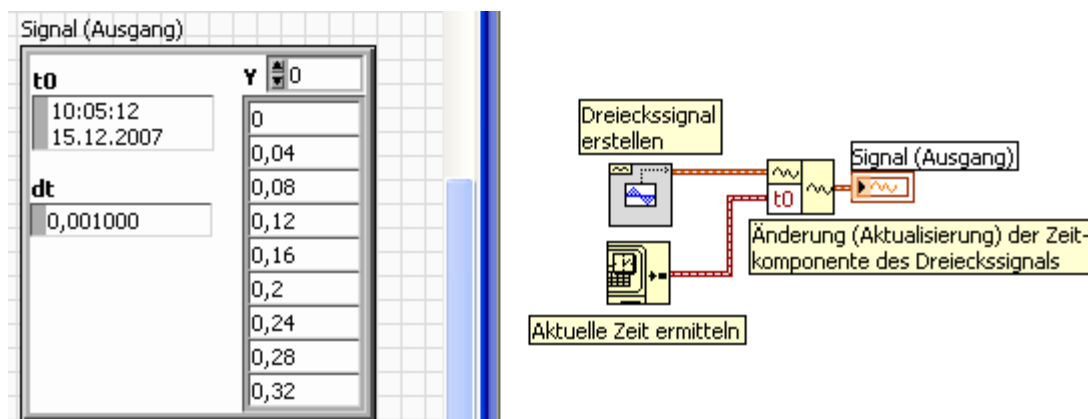
Signalverläufe können addiert werden. Aber nur, wenn es sinnvoll ist!!! Z.Bsp. gibt es eine Fehlermeldung, wenn die Abtastfrequenzen unterschiedlich sind. Wenn die Anzahl der Messpunkte verschieden ist, wird nur das Minimum beider Messpunkte dargestellt (was anderes wär ja auch Unsinn!):



Mehrere Signalverläufe können in einem einzigen Signalverlaufsgraphen dargestellt werden, indem man einen Array dieser Signalverläufe bildet, der dann an den Signalverlaufsgraphen übergeben wird:



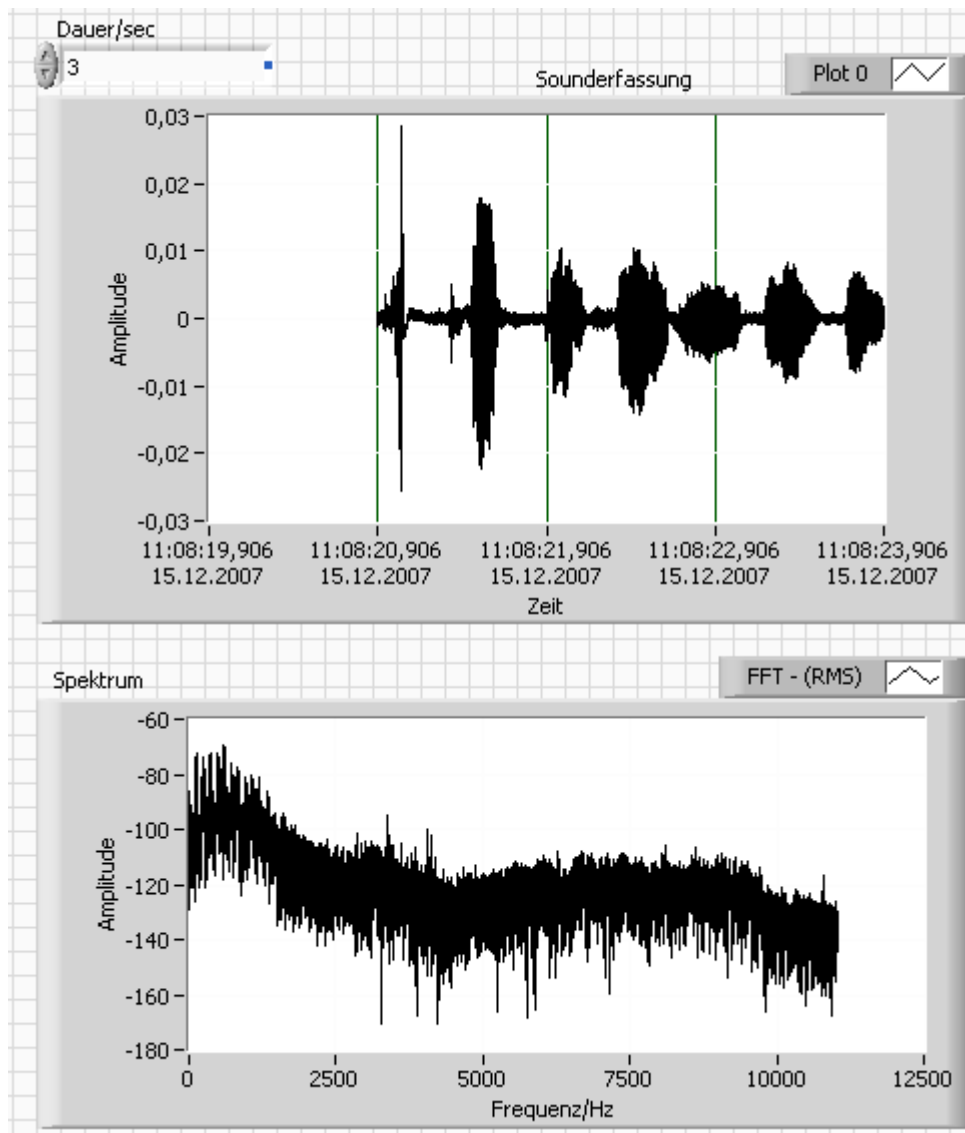
Ausserdem kann auch nun die Zeitachse mit der „echten Zeit“, d.h. mit der zum Zeitpunkt des Programmablaufs geltenden Zeit versehen werden. (Natürlich nur, wenn die Uhr des Rechners richtig gestellt war). Wir benötigen dazu das VI „Datum/Zeit in Sekunden ermitteln“



## Express-VIs

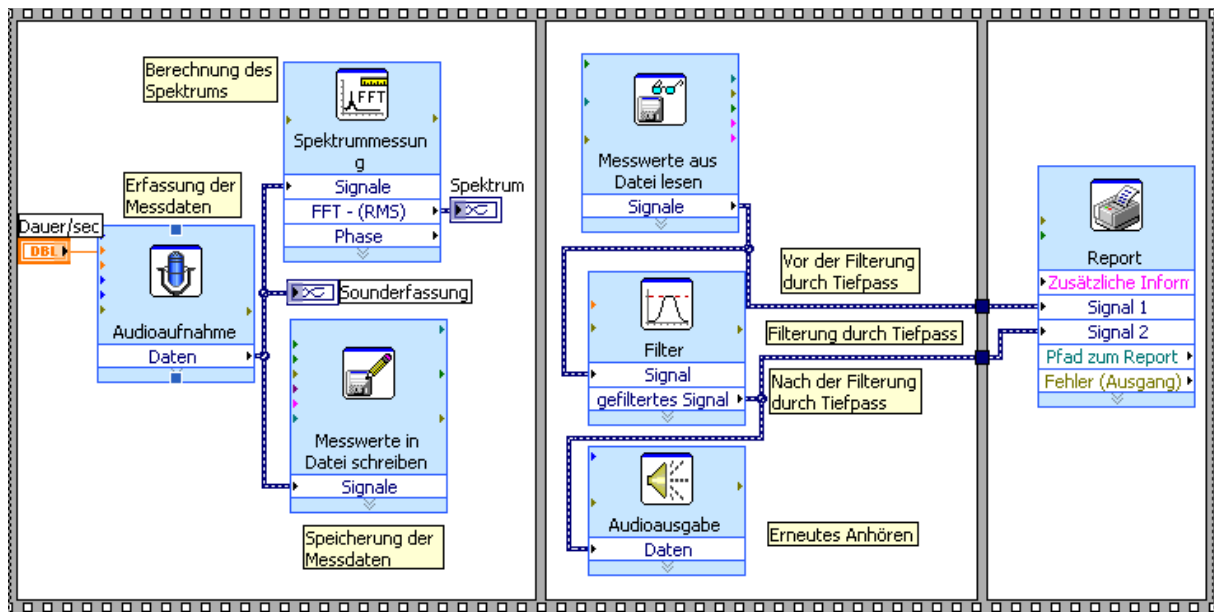
Für Leute, die es ganz besonders eilig haben gibt es seit einigen Versionen von LabVIEW noch die ExpressVIs. Der Anwender kann sich damit recht einfach und bequem Programme zur einfachen Messdatenerfassung und Verarbeitung „zusammenklicken“.

Zunächst sehen Sie das Frontpanel (mit bekannten Elementen):



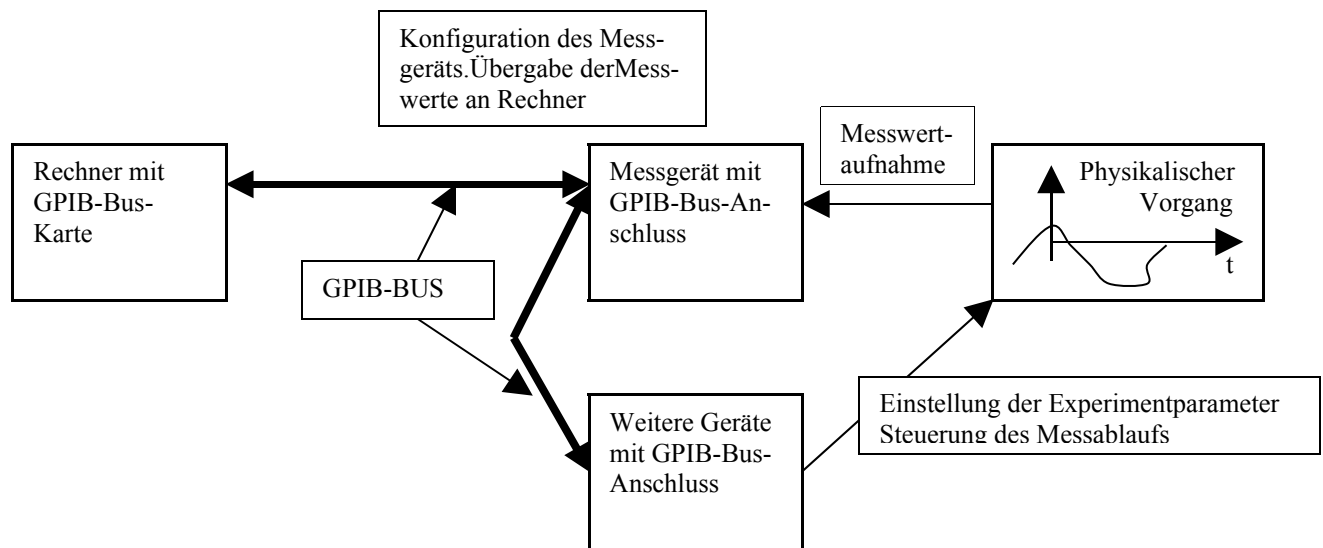
Unten finden Sie das LabVIEW-Express-Programm das zum auf der vorigen Seite dargestellten Frontpanel gehört. Es besteht aus den grossen Blöcken Audioaufnahme, Berechnung des Spektrums, Messwerte in Datei schreiben, Messwerte aus Datei lesen, Filterung, Kontrolle durch erneutes Anhören und schliesslich die Erzeugung eines Berichts zur Veröffentlichung im Internet. Express-VIs erkennt man übrigens am hellblauen Hintergrund...





# Messen mit LabVIEW

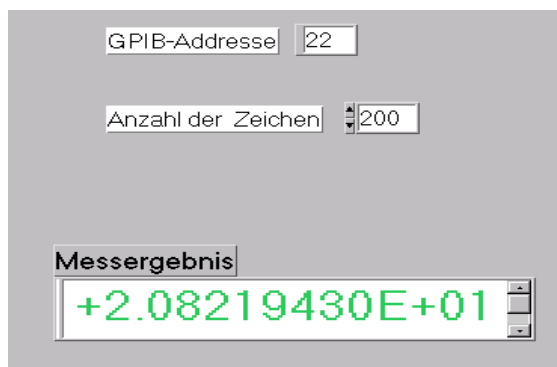
## Der GPIB-Bus



Zu Beginn jeder Messung müssen Messgeräte konfiguriert werden. Die Experimentparameter müssen eingestellt und der Ablauf des Experiments muss kontrolliert werden. Die Messwerte des (zeitlich veränderlichen) physikalische Vorgangs, der erfasst werden soll, werden durch das Messgerät aufgenommen und über den GPIB-Bus an den Rechner übergeben. Jedes der angeschlossenen Geräte hat eine eindeutige GPIB-Adresse. GPIB (**General Purpose Interface Bus**) ist ein von der Firma HP entwickelter Kommunikationsstandard, der für viele Messgeräte realisiert ist. Je nach Gerät müssen gewisse Kommandos zur Steuerung des betreffenden Geräts vom Rechner übergeben werden. Wie diese Kommandos heißen und was genau sie bewirken, muss dem zugehörigen Handbuch des Geräts entnommen werden. Dort gibt es auch oft Programmbeispiele für erste eigene Tests, aus denen die Bedeutung der Kommandos ebenfalls erschlossen werden kann.

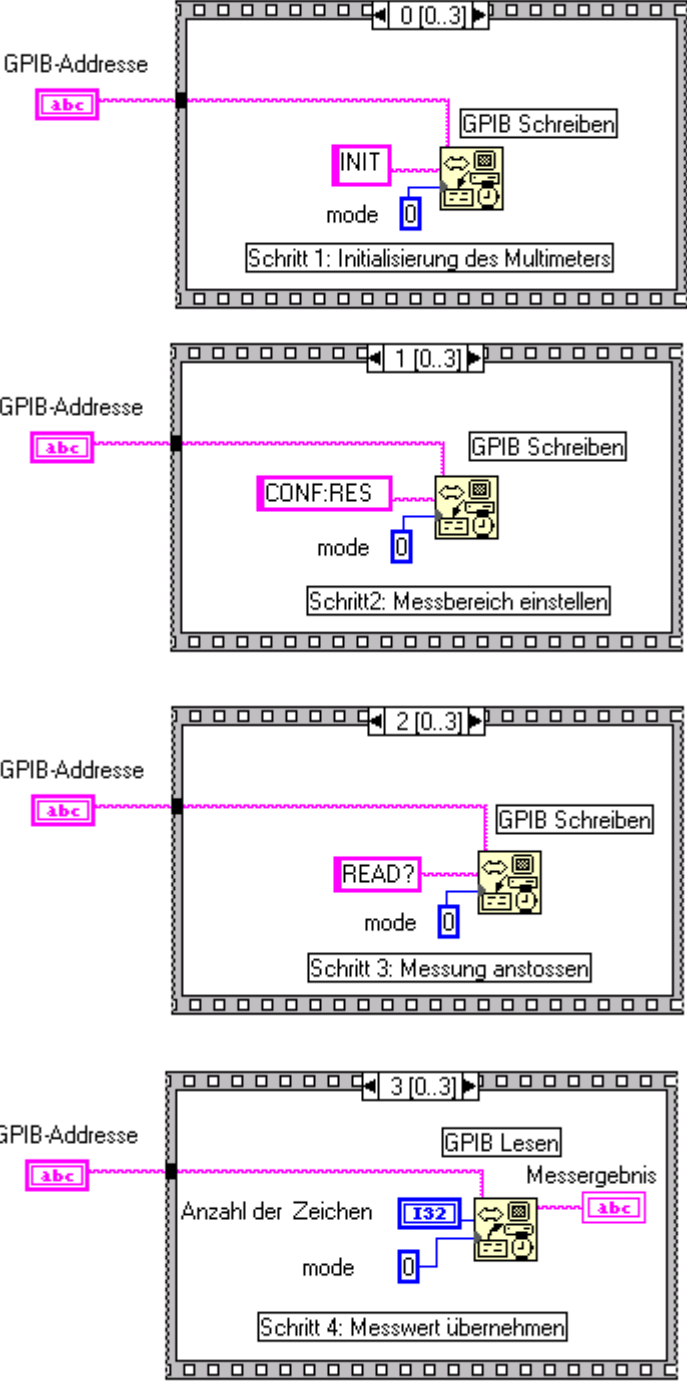
Hier in diesem Skript soll nun in einem sehr einfachen Beispiel die Ansteuerung eines Multimeters (34401A) zur Widerstandsmessung beschrieben werden. Dem Kapitel „Externe Programmierung“ des Handbuchs kann entnommen werden, daß das Meßgerät dazu folgende Kommandos erwartet:

- (1) INIT (Bedeutung: das Gerät wird in einen eindeutigen Grundzustand versetzt)
- (2) CONF:RES (Bedeutung: die Betriebsart Widerstandsmessung wird ausgewählt)
- (3) READ? (Bedeutung: die Messung wird angestossen)
- (4) In einem letzten Schritt muss der Messwert über den GPIB-Bus an den Rechner übergeben werden.



Dieser Ablauf wird in LabVIEW in einer vierstufigen Sequenz realisiert. Links sehen Sie das Frontpanel des Programms. Es erlaubt, die GPIB-Adresse des zu bedienenden Geräts einzugeben. Durch die „Anzahl der Zeichen“ wird festgelegt, wieviele Zeichen maximal vom Gerät an den Rechner übergeben werden sollen. In der String-Control-Anzeige unten wird das Messergebnis dargestellt.

Das Programm besteht, wie schon gesagt, aus vier Schritten, die als Sequenz realisiert werden. Es ist sehr einfach und es lohnt sich nicht darüber Worte zu verlieren, denn noch klarer kann ein Programm nicht beschrieben werden.

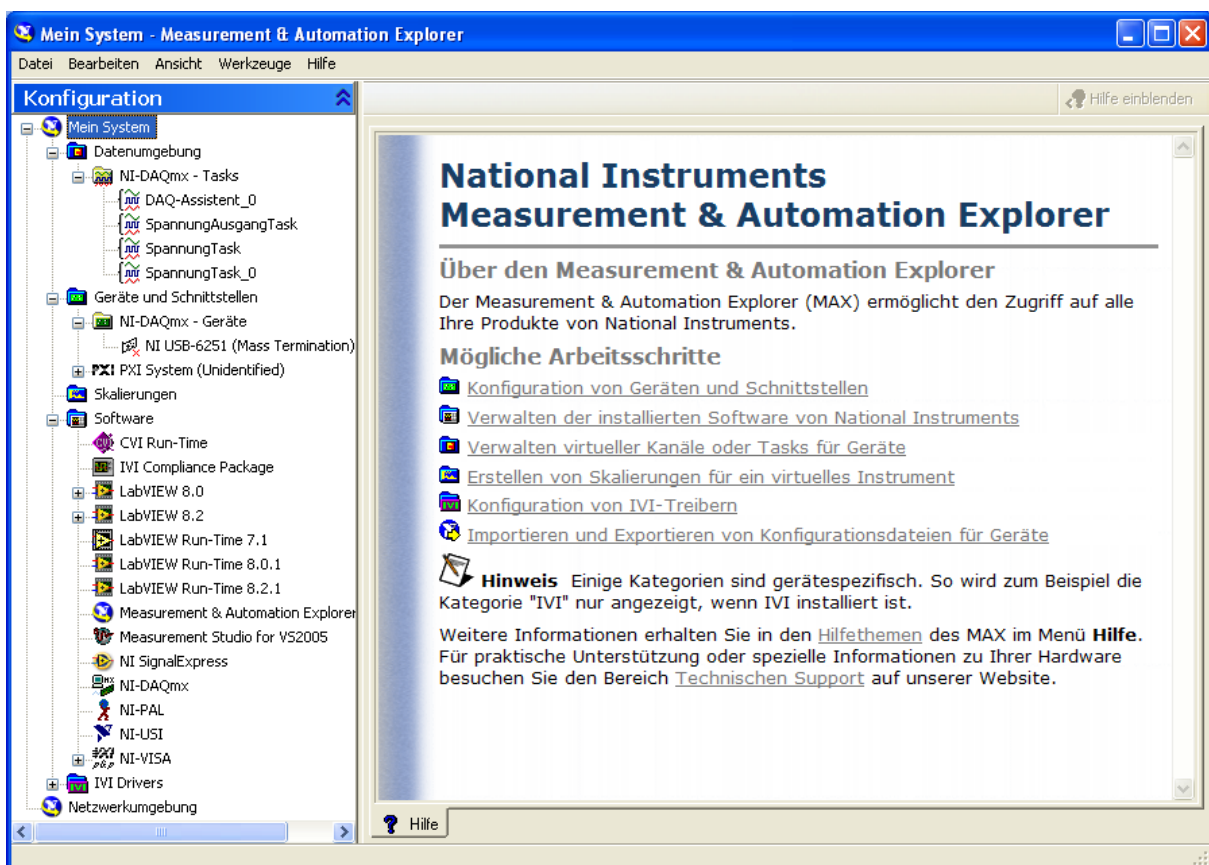


## DAQmx

Wofür DAQmx genau die Abkürzung ist, steht nirgendwo, nicht mal bei Google und Wikipedia. Die Abkürzung DAQmx enthält aber eine allgemein übliche Abkürzung: „DAQ“ für **DataAcquisition**. In unserem Zusammenhang wollen wir nun unter DAQmx alle Softwarebausteine von National Instruments (NI) verstehen, mit deren Hilfe Messdaten erfasst oder ausgegeben werden können. Die NI-DAQmx vorausgehende Software hiess übrigens NI-DAQ.

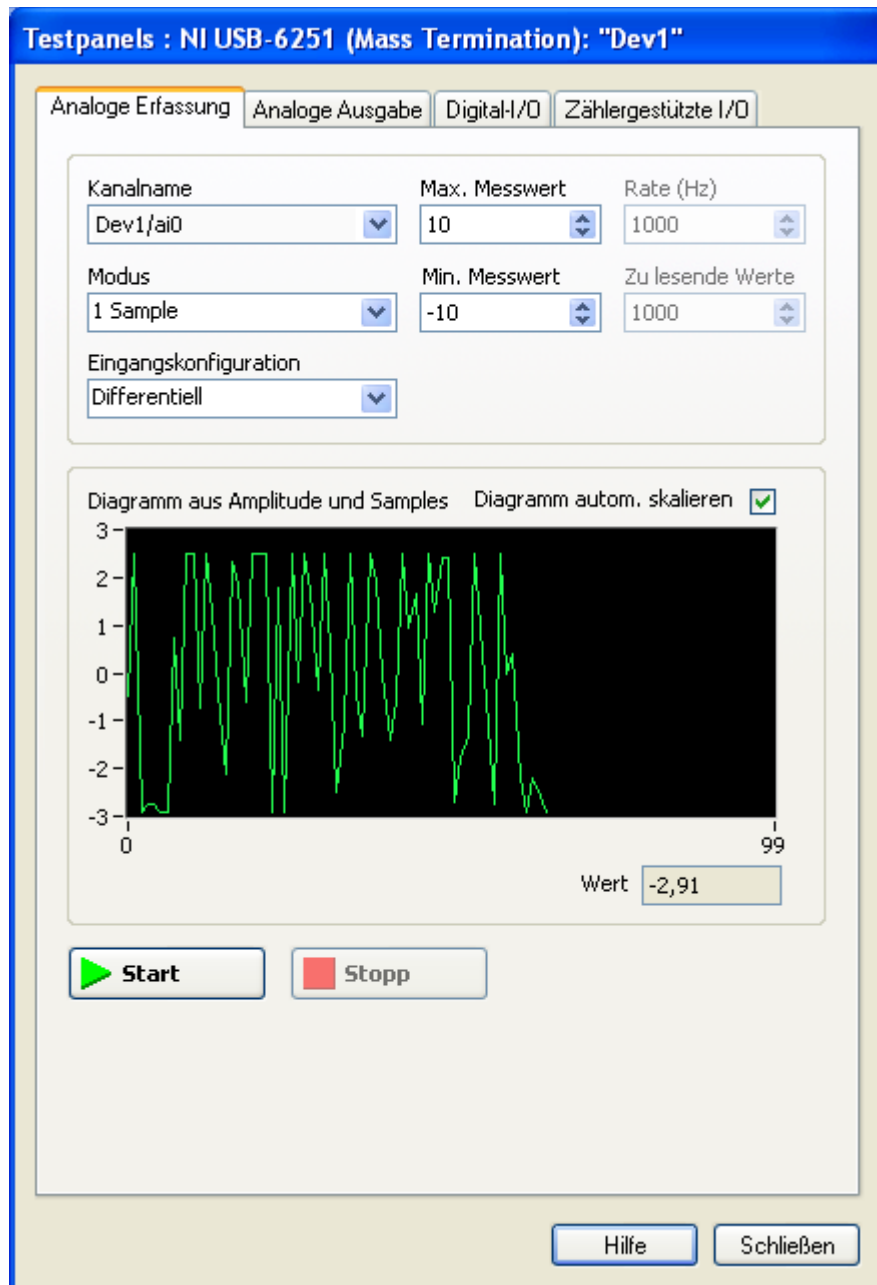
NI-DAQmx umfasst insbesondere auch Software, die nicht nur in LabVIEW, sondern auch in NI LabWindows/CVI-, Visual Basic-, Visual Studio .NET- und C/C++ - Umgebungen eingesetzt werden kann. Zu NI-DAQmx gehören aber auch der NI **M**asurement & **A**utomation **E**Xplorer kurz: „MAX“, der DAQ-Assistent und die Software LabVIEW SignalExpress LE, mit deren Hilfe zum einen die in einem Rechner eingebauten Messhardwarekomponenten konfiguriert und getestet werden können (MAX) und mit deren Hilfe ziemlich schnell einfachere Messprogramme zusammenglickt werden können: ein erstes Beispiel dazu haben wir oben schon gesehen, als wir die Soundkarte als Datenerfassungshardware benutzt haben.

Messdatenerfassung ist eine komplexe Angelegenheit, die man nicht von heute auf morgen erlernen kann. Wir werden uns hier einzelnen Aspekten dieses riesigen Gebiets in mehreren Schritten langsam annähern. Zunächst soll nun MAX, der „Measurement & Automation Explorer“ beschrieben werden. Sie finden ihn, als reguläres Programm unter „National Instruments“:



In der rechten Hälfte können Sie erkennen, welche Aufgaben mit MAX erledigt werden können, auf der linken Seite sehen Sie die in Ihrem Rechner vorhandene Hardware (Geräte und Schnittstellen) und die installierten Software-Komponenten.

Unter NI-DAQmx-Geräte sehen Sie den Eintrag: „NI USB-6251 (Mass Termination)“. Es handelt sich dabei um die für diese LV im PC-Pool Physik zur Verfügung gestellte Hardware, die per USB mit den dortigen Rechnern verbunden ist. Ein rechter Mausklick auf diesen Eintrag öffnet das folgende Fenster:



Man sieht vier Karteikarten – die den vier Grundfunktionen, die mit dem Gerät NI USB-6251 ausgeführt werden können, entsprechen.

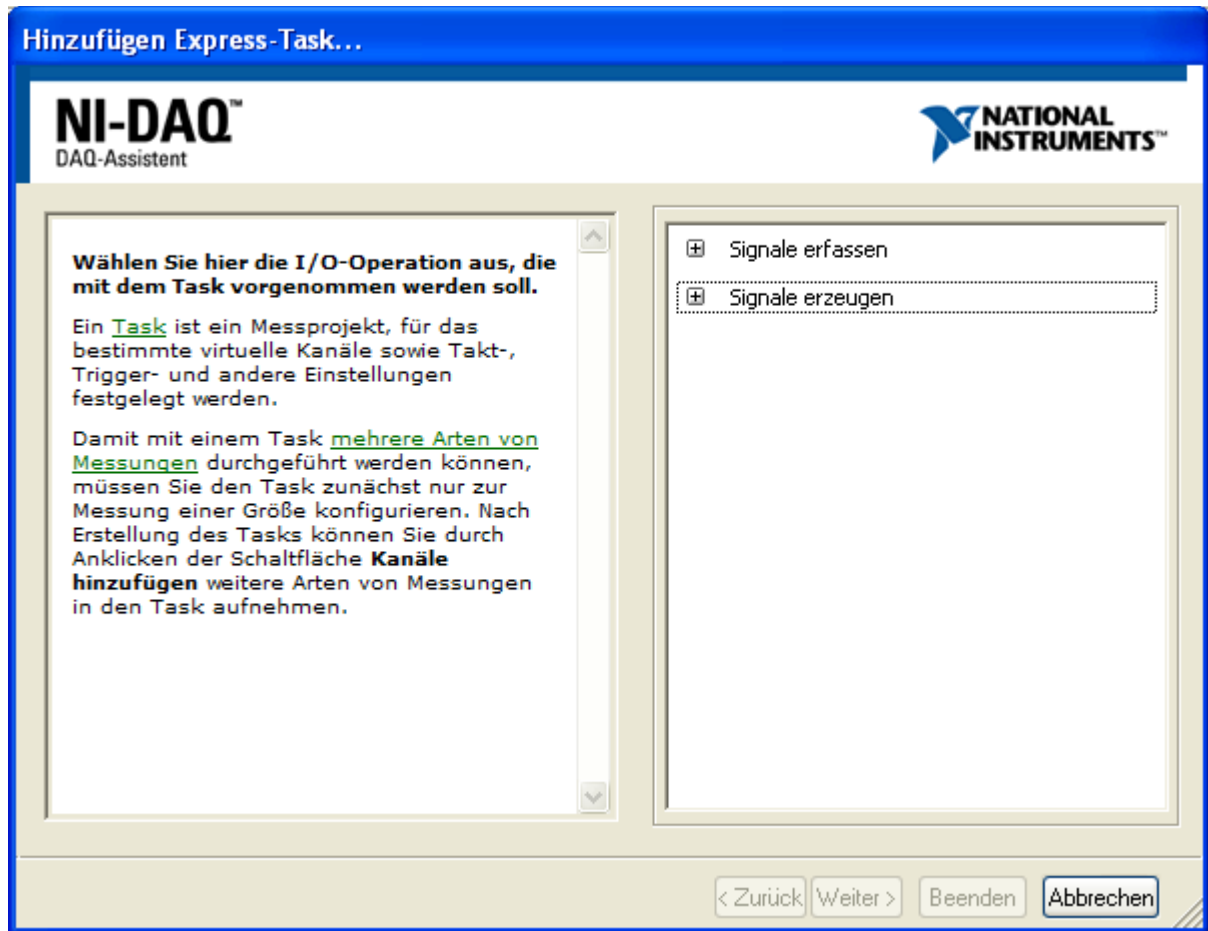
- Analoge Erfassung
- Analoge Ausgabe
- Digital-I/O
- Zählergestützte I/O

Die im Diagramm dargestellte Kurve ist entstanden nach einem Klick auf Start/Stop und zeigt eine in diesem Zeitintervall am Anschluss mit dem Kanalnamen „Dev1/AI0“ erfasste elektrische Spannung. Da mehrere Datenerfassungs-Sets an den Rechner angeschlossen werden können, unterscheidet man diese durch: „Dev1“, „Dev2“, .... AI0 bedeutet Anschluss „Analog Input 0“. „Dev1/AI0“ bedeutet dann logischerweise: analoger Input 0 von Datenerfassungsset 1. Man kann aus dem Ergebnis schliessen, dass unsere Hardware funktionsfähig ist, zumindest was den Aufgabenbereich analoge Erfassung betrifft.

Genauso können Sie nun die drei anderen Grundfunktionen testen. Wenn Sie diese Tests durchgeführt haben, haben Sie einen ersten Eindruck über die Möglichkeiten der an Ihren Rechner angeschlossenen Geräte gewonnen.

## Einfache Messprogramme erstellen mit DAQ-Assistent

Unter Funktionen / Mess-/IO / DAQmx-Dateerfassung finden Sie den DAQ-Assistent(en). In ihm haben Anfänger ein Werkzeug, mit dem leicht einfache Messaufgaben programmiert werden können. Nach dem Platzieren des DAQ-Assistenten im Blockdiagramm öffnet sich das folgende Fenster:



Der DAQ-Assistent führt Sie nun durch die verschiedenen Aufgabenbereiche, die mit dem Assistenten programmiert werden können. Zuerst muss man wissen, ob Signale erfasst oder erzeugt werden sollen. Angenommen, Sie entscheiden sich für Signalerfassung, dann müssen Sie sich weiter entscheiden, ob Sie eine Spannung, Temperatur, Dehnung usw. erfassen wollen.



### a) Signale erfassen: Spannung einlesen

Beginnen wir mit einem VI zur Spannungserfassung. Dann erscheint im DAQ-Assistenten das folgende Fenster. Sie können zunächst die physikalischen Anschlüsse auswählen, an denen die Spannungsverläufe erfasst werden sollen. (Auf dem Anschluss-Board sehen Sie nur die Anschlüsse AI 0, AI 1, ..., AI 7.). Es besteht die Möglichkeit mehrere dieser Anschlüsse auszuwählen.



**Wählen Sie aus, welche physikalischen Kanäle dem Task hinzugefügt werden sollen.**

Wenn Sie bereits [globale virtuelle Kanäle](#) für die gleiche Art der Messung wie im aktuellen Task konfiguriert haben, klicken Sie auf die Registerkarte **Virtuell**, um die Angaben von einem des virtuellen Kanals zu kopieren.

Wenn Sie mit TEDS-Sensoren arbeiten, klicken Sie auf die Registerkarte **TEDS**, um dem Task TEDS-Kanäle hinzuzufügen.

Bei Geräten, bei denen [mehrere Kanäle](#) in einem Task verwendet werden können, lassen sich auch mehrere Kanäle gleichzeitig zum Task hinzufügen.

**Physikalisch**

## Unterstützte physikalische Kanäle

- Dev1 (USB-6251 (Mass Termination))
- ai0
  - ai1
  - ai2
  - ai3
  - ai4
  - ai5
  - ai6
  - ai7
  - ai8
  - ai9
  - ai10
  - ai11
  - ai12
  - ai13

Halten Sie zum Markieren mehrerer Kanäle <Strg> oder <Shift> gedrückt.

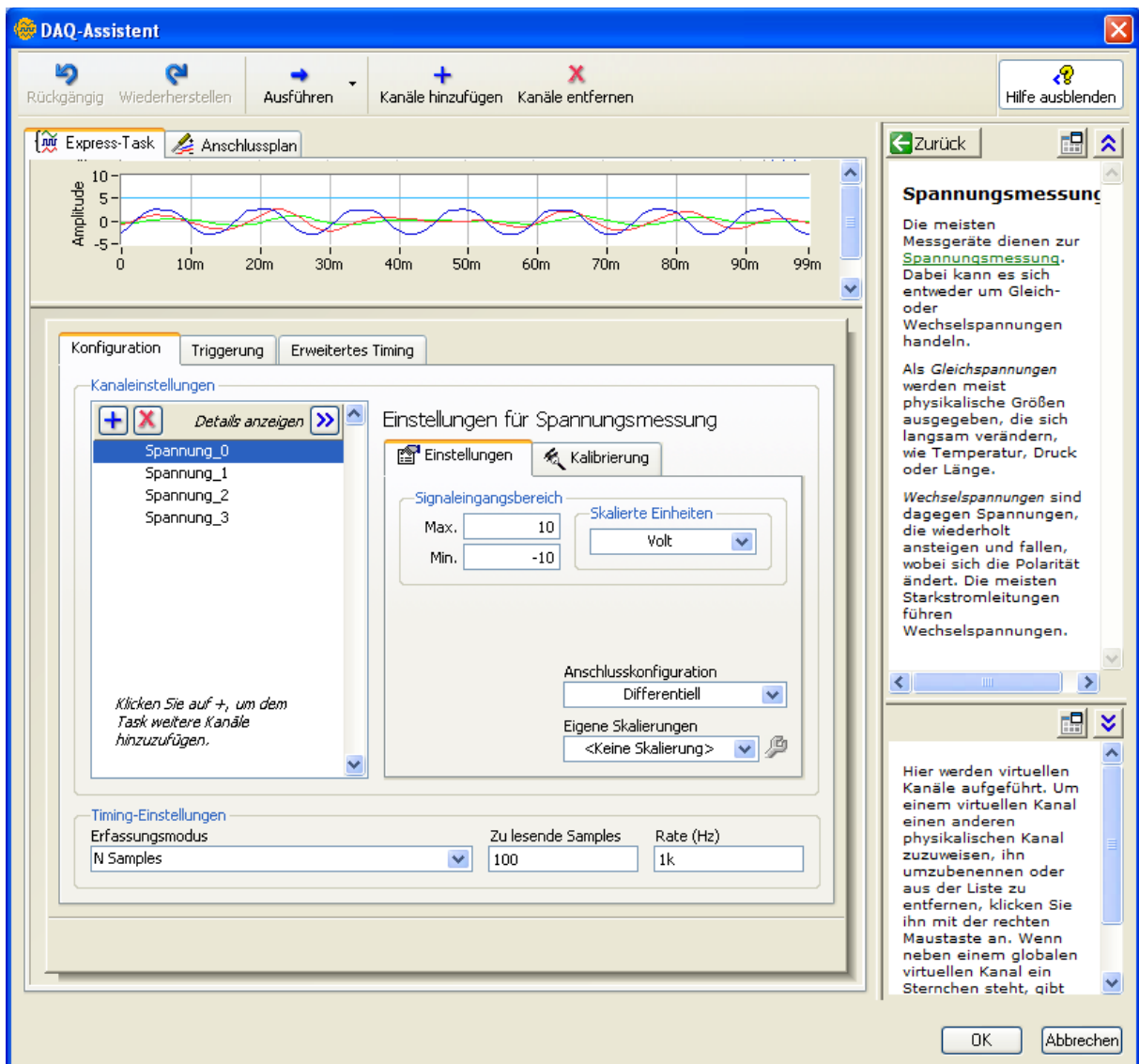
&lt; Zurück

Weiter &gt;

Beenden

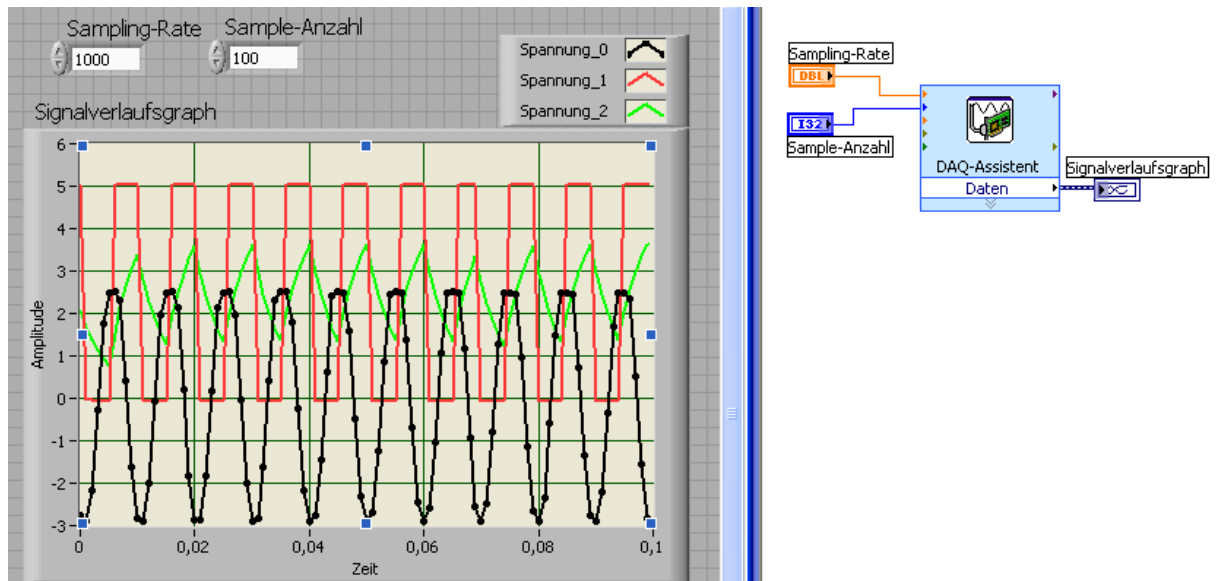
Abbrechen

Mit dem Klick auf Beenden kommt man in einen letzten Bereich von DAQ-Assistent, in dem noch verschiedene Parameter festgelegt werden können und indem ein kleiner Test (Ausführen) den aktuellen Kurvenverlauf anzeigt. In den Seitenfenstern erscheinen Hilfetexte zur jeweiligen Aufgabe, bzw. Einstellung



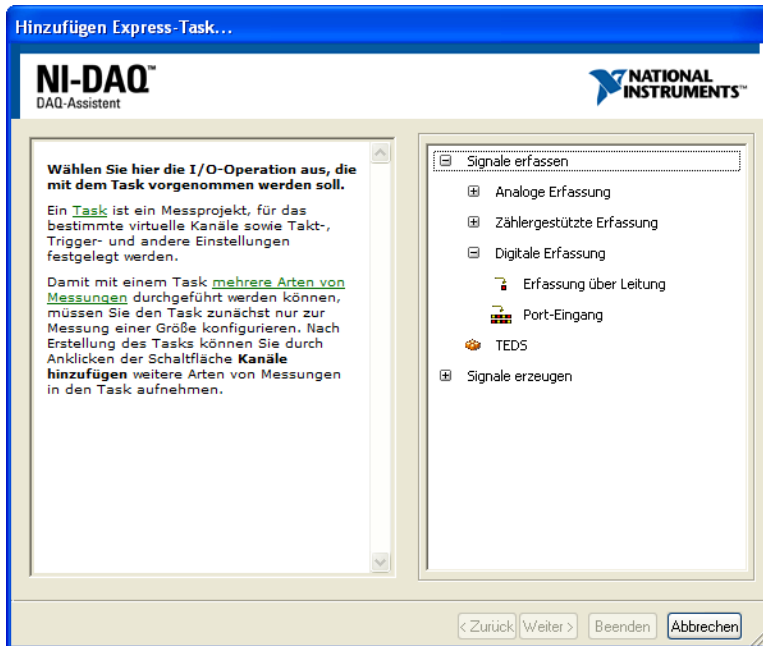
Mit OK wird der DAQ-Assistent abgeschlossen.

Man erhält dann im Blockdiagramm zunächst ein hellblaues Express-VI, an das dann noch einige der schon bekannten Ein- und Ausgabeelemente angeschlossen werden können.



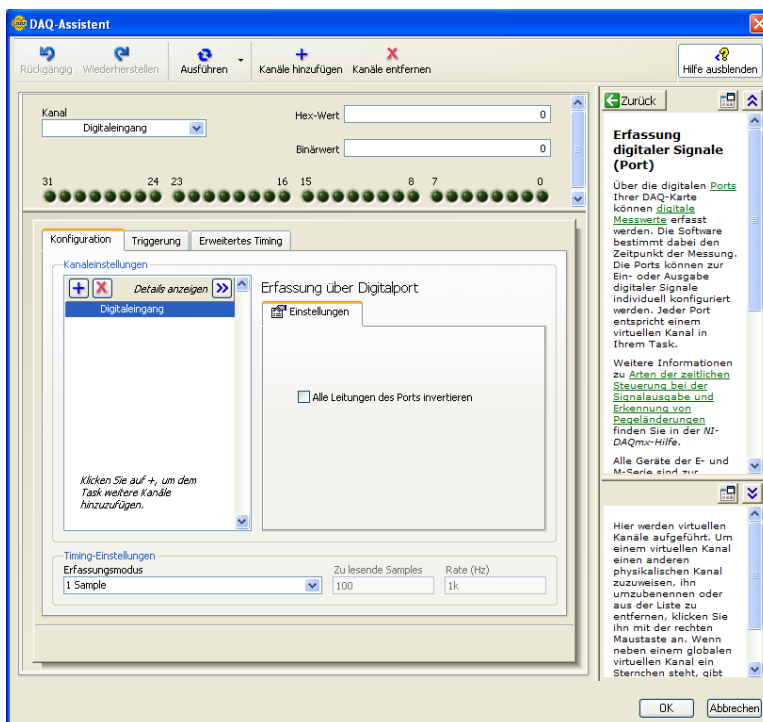
Somit hat man schliesslich ein kleines Messprogramm erstellt, das die an den Eingangskanälen AI 0, AI 1, AI 2 und AI3 anliegenden Spannungen mit einer Samplingrate von 1000 Messpunkte/Sekunde messen. Bei jedem Aufruf des Programms werden mit den oben verwandten Messparametern 100 Messpunkte erfasst, d.h. jede Messung erfasst einen physikalischen Vorgang, der insgesamt  $100 \cdot 10^{-3} \text{ sec} = 0,1 \text{ sec}$  dauert.

## b) Signale erfassen: Digitale Signale erfassen



Bei der Digitalen Erfassung können entweder einzelne Leitungen oder Ports, in denen üblicherweise 8 Leitungen zusammengefasst sind, abgefragt werden.

Auf Port 0 werden 8 Leitungen P0.0, P0.1, ..., P0.7 auf Schraubklemmen zur Abfrage bereitgestellt. Je nachdem, ob man dort Masse (D GND) oder +5V anlegt zeigt das Abfrageprogramm dann TRUE oder FALSE an (ob Masse dabei TRUE oder FALSE entspricht sollte durch 1 Versuch geklärt werden.

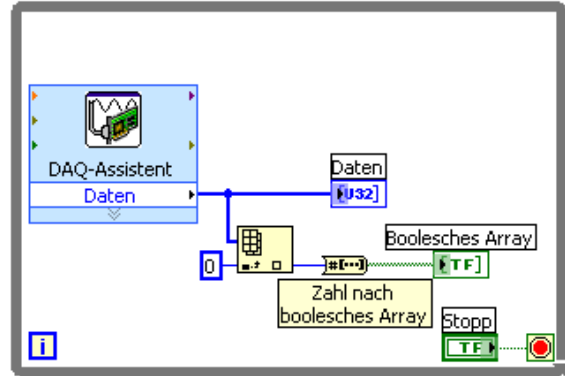
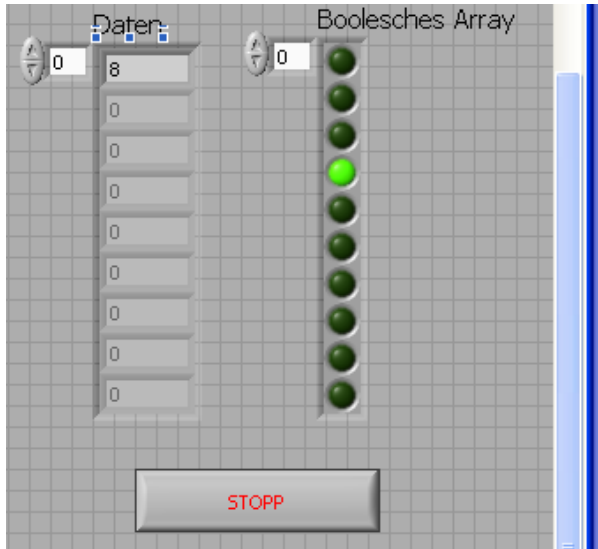


Diesen Versuch können Sie schon innerhalb des DAQ-Assistenten machen (Ausführen, oberste Zeile).

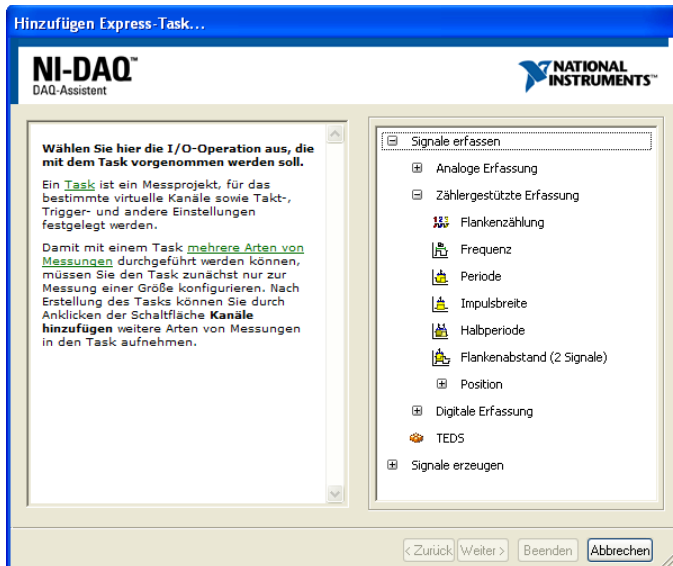
Wenn Sie mit einem Stück Draht +5V an einen der Kontakte P0.1,...,P0.7 anlegen, dann leuchtet im nebenstehenden Fenster jeweils eine der entsprechend von 0..7 benannten LEDs auf.

Nach der Bestätigung durch OK erhält man schliesslich im Blockdiagramm ein Express-VI, mit dem dann ganz normal weiter gearbeitet werden. Beachten Sie dabei. Der Ausgang Daten ist ein Array von Integerwerten vom Typ U32 (32 Bit, unsigned, vorzeichenlos).

Auf der folgenden Seite sehen Sie eine simple Anwendung, in der in einer Schleife die 8 Digitalen Input-Leitungen von DAQ 6251 abgefragt werden.

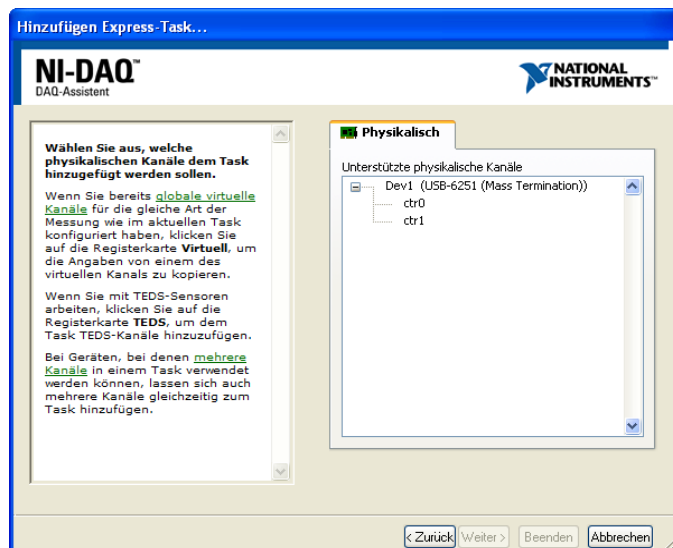


### c) Signale erfassen: Zähler auslesen

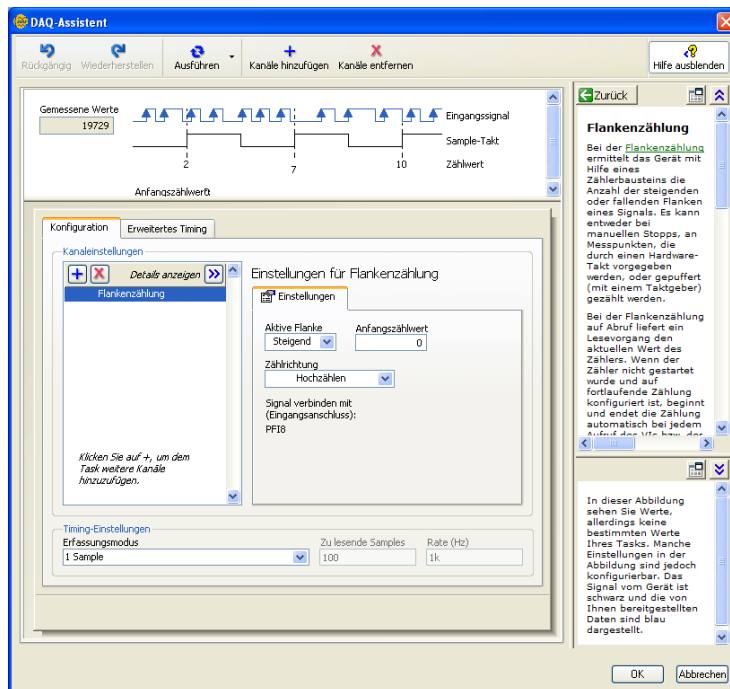


Die dritte Möglichkeit der Messdatenaufnahme ist das Auslesen eines Zählers.

In der einfachsten Form geht es dabei zunächst darum, zu zählen, wieviele positive Flanken eines Rechtecksignals aufgetreten sind.

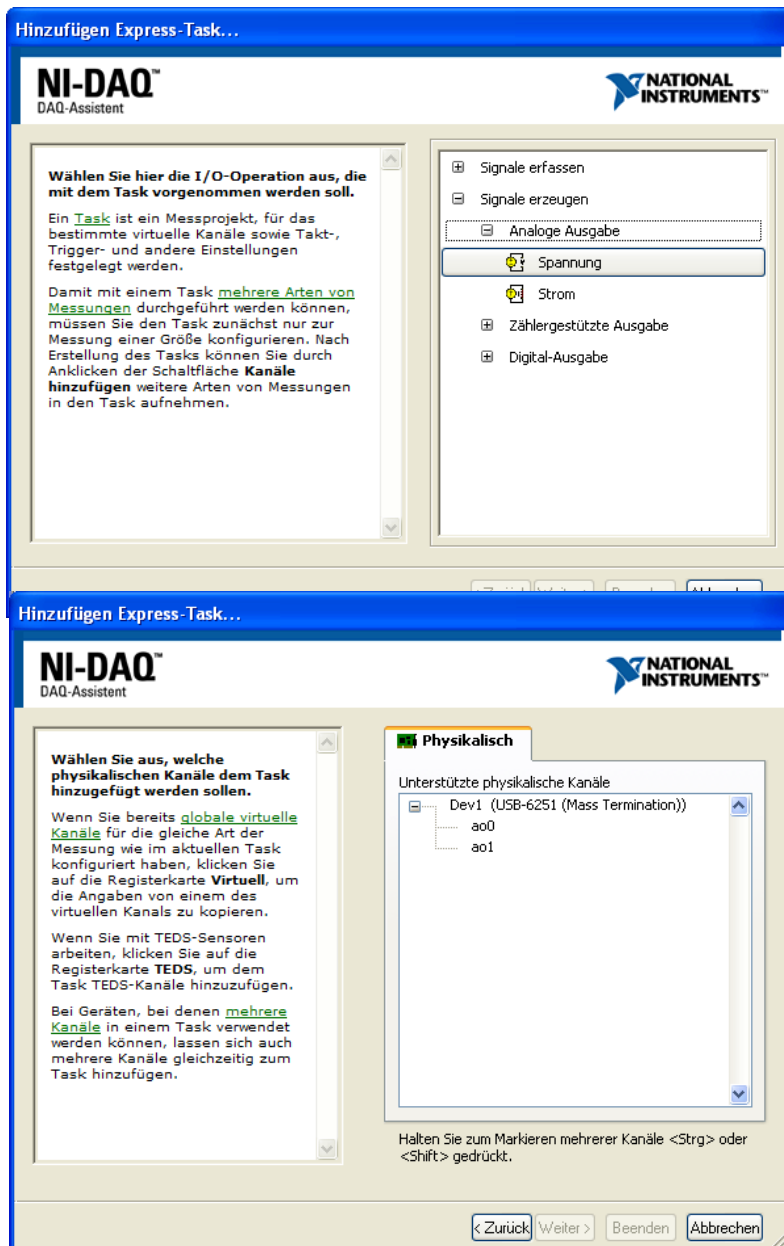


USB-6251 hat zwei Zähleranschlüsse: ctr0 und ctr1. Wir nehmen mal an, dass wir uns für ctr0 entschieden und dann auf Weiter geklickt hätten.



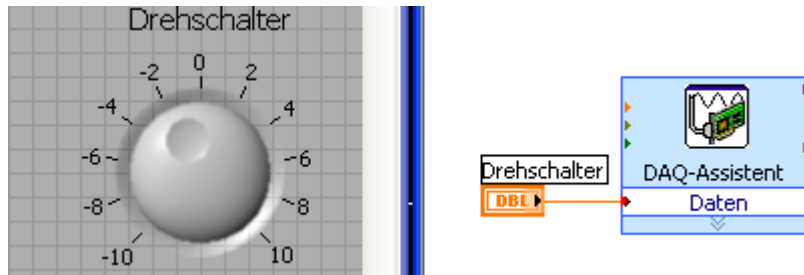
Wenn man den Anschluss PFI 8 (bzw. CTR 0 Source) mit dem TTL Square Wave Anschluss des Function Generators verbindet und im DAQ-Assistenten „Ausführen“ anklickt, kann man erkennen, wie der Zähler „Gemessene Werte“ hoch gezählt wird. Dies geht jeweils schneller oder langsamer, je nachdem welche Frequenz am Funktionsgenerator eingestellt wird.

## d) Signale erzeugen: Analoge Ausgabe einer Spannung



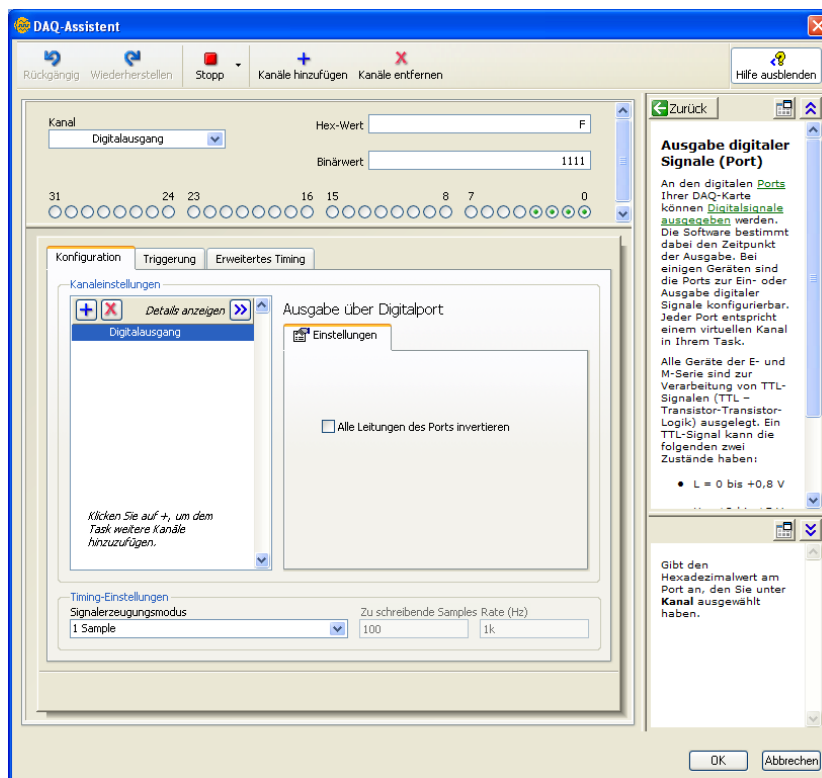
USB-6251 enthält einen Digital-Analog-Wandler, mit dessen Hilfe elektrische Spannungen zwischen  $-10\text{V}$  und  $+10\text{V}$  an den Anschlüssen AO 0 und AO 1 (im Feld Analog Outputs) ausgegeben werden können

Nach dem Abschluss des DAQ-Assistenten kann man sich dann ein erstes einfaches VI zur Ausgabe von Spannungen am Ausgang AO 0 erstellen:



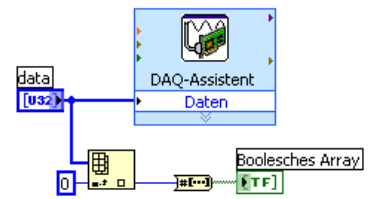
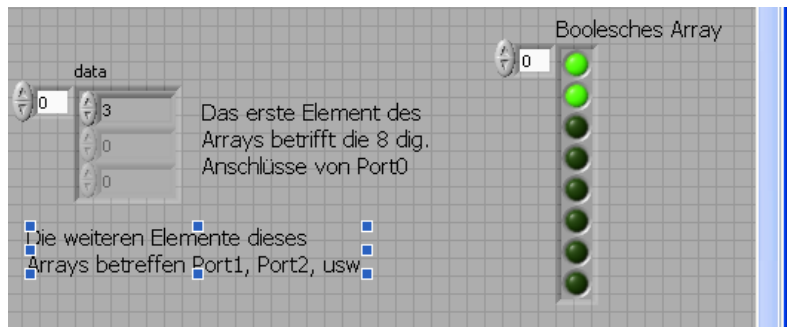
### e) Signale erzeugen: Digitale Ausgabe an Port0

Um die Digitalen Ausgänge PO.0, ... , PO.7 in der Port-Sprechweise anzusprechen muss dem vom DAQ-Assistenten erzeugten Express-VI ein entsprechendes Byte übergeben werden. (8 Bit  $\cong$  8 LED's  $\cong$  128 verschiedene Möglichkeiten diese 8 Dioden/Relais/was auch immer, die an PO.0, ... , PO.7 angeschlossen wurden, ein- bzw. auszuschalten).

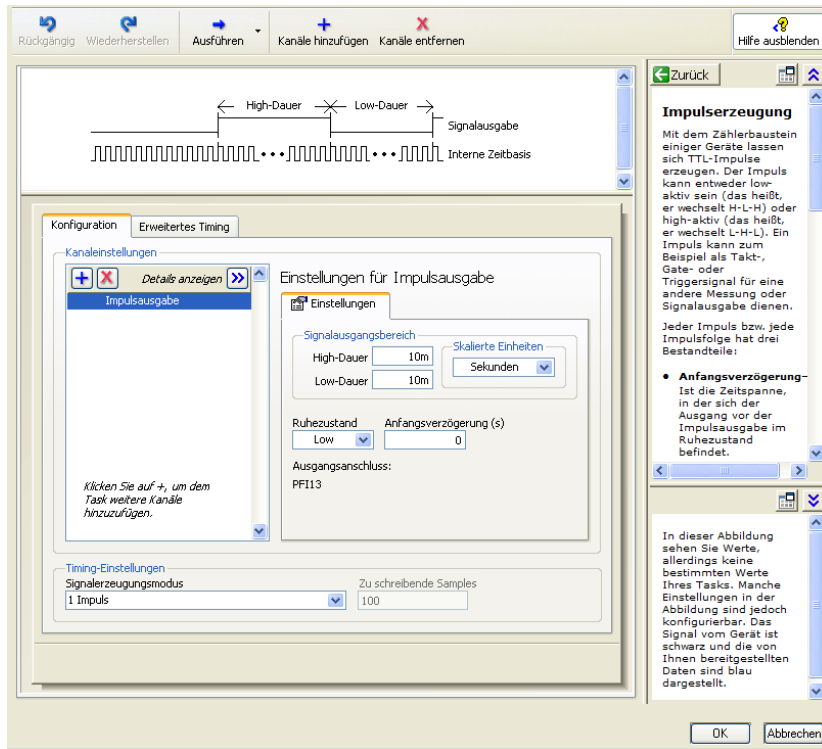


Im LabVIEW-Programm unten sehen Sie eine Anwendung des oben erzeugten Express-VIs.

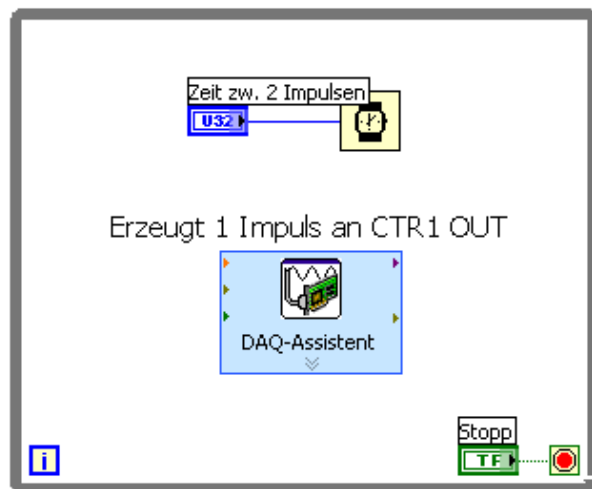




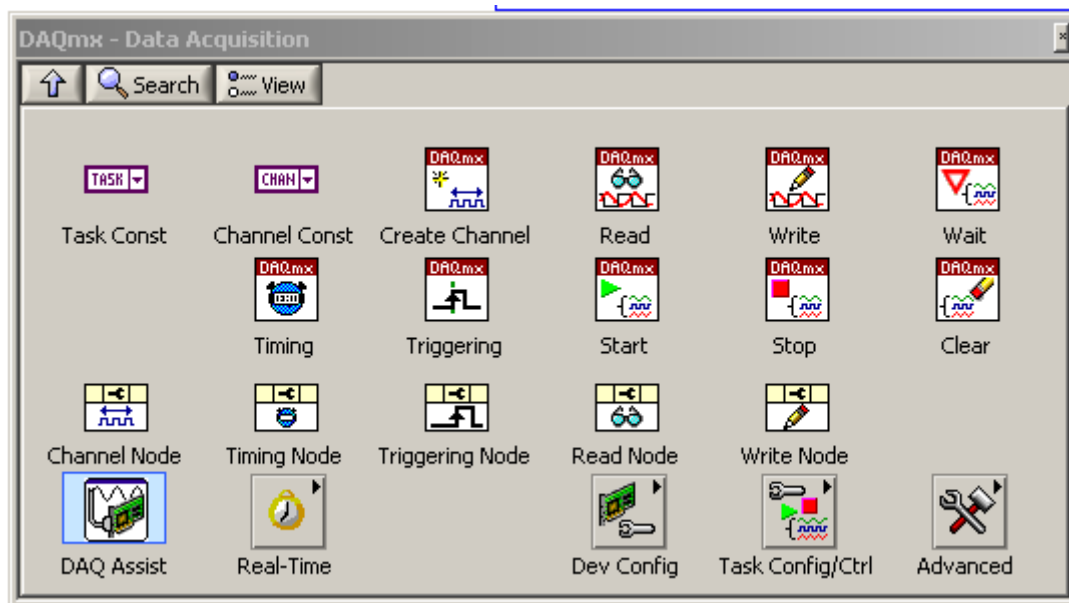
### e) Signale erzeugen: Ausgabeimpuls eines Zählers



Das vom obigen DAQ-Assistenten erzeugte VI erzeugt bei jedem Aufruf einen Ausgabeimpuls an CTR 1 OUT  
 Das unten aufgeführte LabVIEW-Programm erzeugt dann Impulse mit dem jeweils eingestellten Zeitabstand so lange die While-Schleife läuft.



## Was kommt nach dem DAQ-Assistenten?



Der DAQ-Assistent ist in einer ersten Phase sehr hilfreich, um ganz einfache Messprogramme zu erstellen. Mit dem DAQ-Assistenten erstellte Programme können übrigens auch per rechtem Mausklick in ganz normale LabVIEW-Programme übersetzt werden. Diese Übersetzung bringt allerdings manchmal Überraschungen mit sich und muss dann nachgearbeitet werden und schon anspruchsvolle Messprobleme, wie die Erfassung des Einschaltvorganges eines Kondensators können m.E. nicht zufriedenstellend mit dem DAQ-Assistenten gelöst werden. Wir befassen uns deshalb hier auch noch mit einigen der anderen Sub-Vis, die unter Funktionen / Measurement-I/O / NI-DAQmx – Data Acquisition angeboten werden.



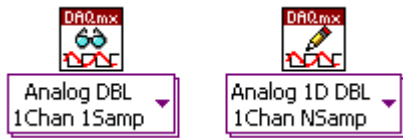
Ein Messprojekt heisst in LabVIEW **Task**. In einem Task werden alle Parameter (Kanäle, Takt-, Trigger- und andere Einstellungen) einer Messung oder Signalerzeugung zusammengefasst. Einen Task erzeugen kann man, indem man das „DAQmx Create Virtual Channel VI“ aufruft. Dadurch wird gleichzeitig ein Task eröffnet *und* ein virtueller Messkanal geschaffen, dem physikalische Ein-/Ausgangskanäle und andere Parameter per Control-Element zugewiesen werden können.



Ein eröffneter Task muss auch regulär beendet werden. (Wenn das nicht gemacht wird, kann es passieren, dass eine Menge unbeendeter Tasks unnötigerweise die Ressourcen des Rechner belegen).

Die *allereinfachste Messaufgabe*, in der nichts gemessen oder sonst verändert wird, besteht also zumindest aus diesen beiden SubVIs: „Task eröffnen“ und „Task schliessen“. Beachten Sie bitte, dass bei der Eröffnung eines Tasks noch genauer spezifiziert werden muss, was die eigentliche Aufgabe des Tasks ist. Diese Spezifizierung erfolgt über eine spezielle Auswahl (Polymorphie-Eigenschaft!), die man nach dem Platzieren des VIs im Blockdiagramm noch vornehmen muss. Das polymorphe VI links zeigt z.B. an, dass dieser Task einen oder mehrere analoge Spannungswert(e) (AI Voltage) erfassen soll.

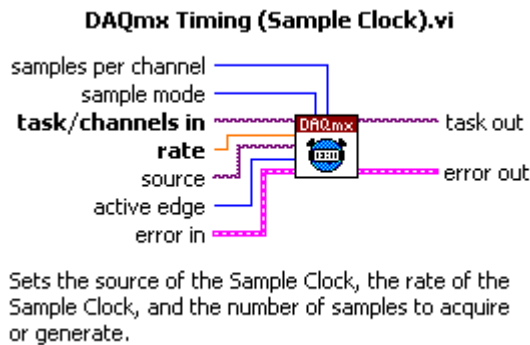




Mit den beiden links aufgeführten polymorphen SubVIs (beachten Sie die Symbolik!) lassen sich nun (Mess-)daten einlesen, bzw. (Mess-)daten ausgeben. Das Lese-VI (1Chan 1Samp) dient etwa dazu, aus einem einzelnen Kanal einen einzelnen Messwert einzu-

lesen, wohingegen durch das rechts oben dargestellte Schreib-VI auf einem einzelnen Kanal ein eindimensionaler Array von Messdaten ausgegeben wird.

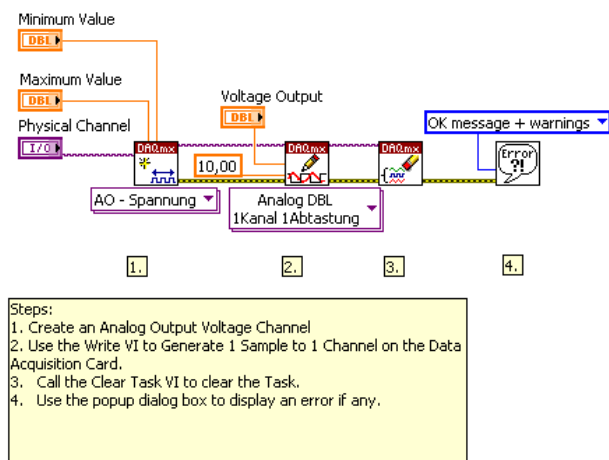
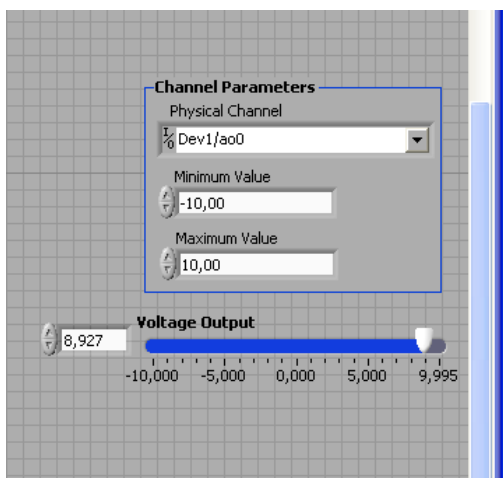
Mit diesen vier VIs ist man nun in der Lage die beiden Grundaufgaben der Messtechnik zu erledigen: „1 Wert lesen“ bzw. „1 Wert schreiben“.



Schon das oben aufgeführte Schreib-VI führt einen schnell zu der naheliegende Frage, in welchem zeitlichen Abstand (rate) die N (samples per channel) Messwerte des Arrays ausgegeben werden sollen. Diese Informationen werden dem Messtask durch das nebenstehende SubVI DAQmx Timing (Sample Clock).vi zugeordnet. Die beiden wesentlichen Anschlüsse darin sind: „rate“ und „samples per channel“, mit dem die oben erwähnten Parameter

festgelegt werden können.

Zunächst betrachten wir das allereinfachste Beispiel aus den LabVIEW-Beispielen:

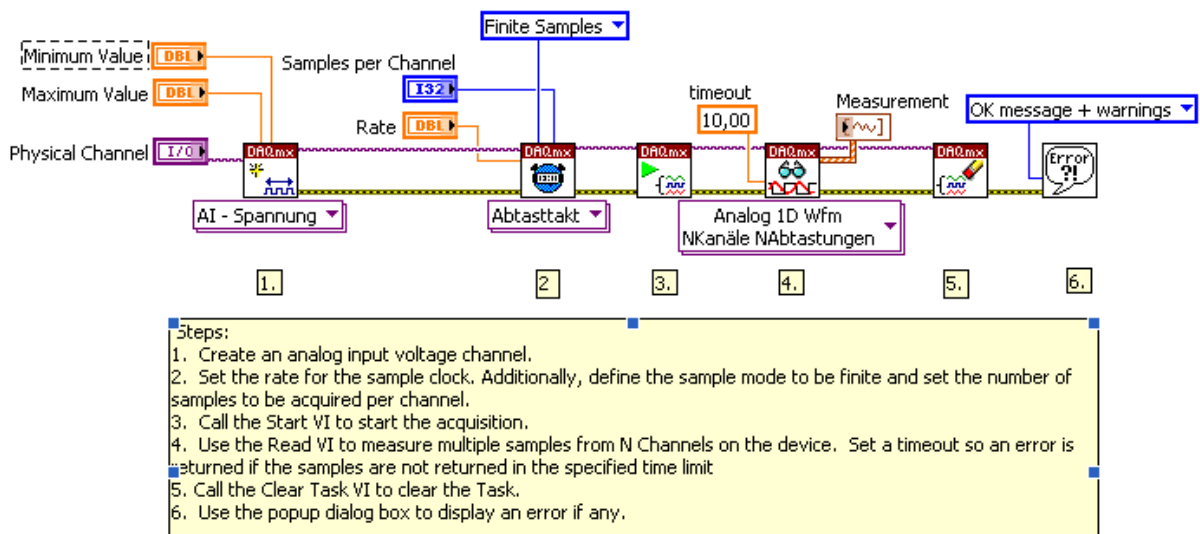
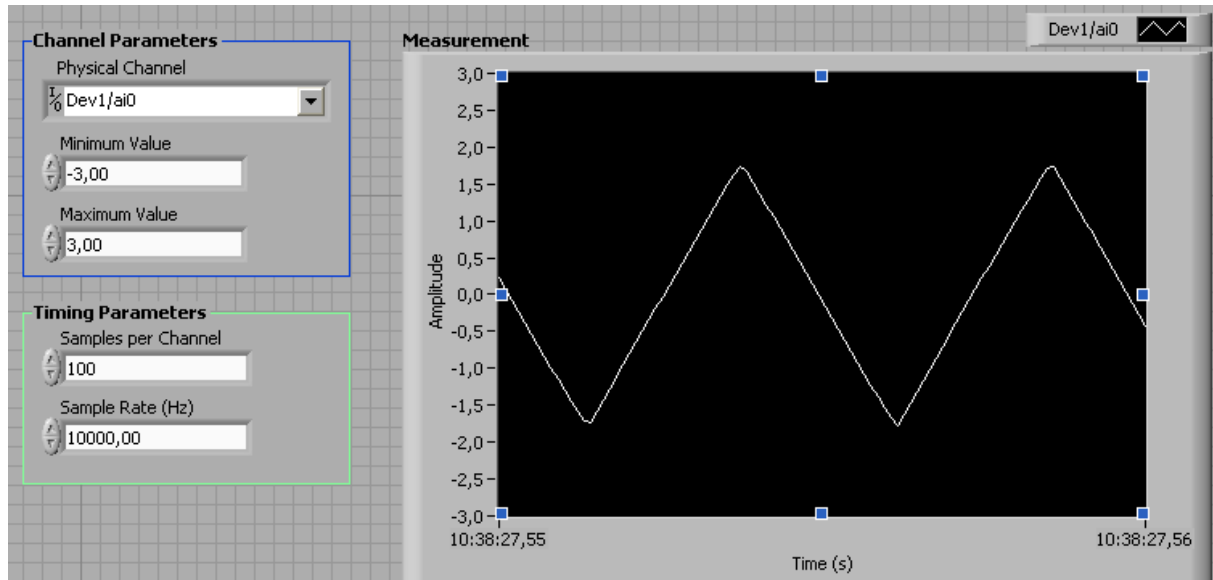


Mit diesem Programm (Gen Voltage Update.vi) kann am physikalischen Anschluss eine Spannung zwischen –10V und +10V ausgegeben werden.

Genauso (nur andersrum...) erstellen Sie ein simples VI zum Einlesen eines einzigen Spannungswertes.

## Erfassung von N Messwerten

Das einfachste Beispiel zur Erfassung von N Messwerten mit einem wohldefinierten Zeitabstand  $dt$  zwischen zwei Messpunkten ist das folgende (Acq & Graph Voltage-Int Clk.vi):

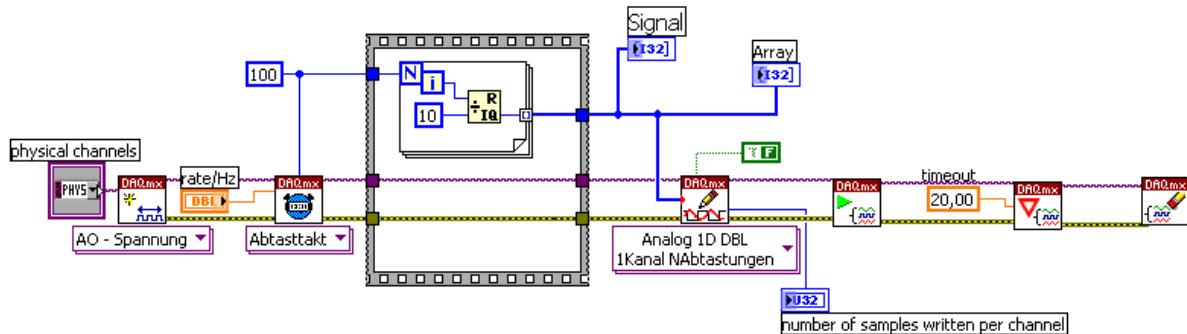


Das wichtige VI im obigen Programm ist das Abtasttakt-VI, mit dem die Abtastrate und die Anzahl N der zu erfassenden Messpunkte festgelegt wird.

Bei jedem Aufruf des VIs beginnt die Erfassung der Daten an einer anderen Stelle des Kurvenverlaufs. Das im obigen Programm enthaltene Start VI (Punkt 3.) ist nicht erforderlich. Es können gleichzeitig mehrere physikalische Kanäle abgetastet werden, indem man für Physical Channel z.B. Dev1/ai0:7 eingibt. Das Lese-VI gibt immer einen Array von (in diesem Fall 8) Waveforms zur graphischen Darstellung aus.

## Ausgabe von N Spannungswerten

Will man einen ganzen Signalverlauf bestehend aus N Werten im zeitlichen Abstand von dt ausgeben kann man das mit folgendem Beispiel-VI erreichen:

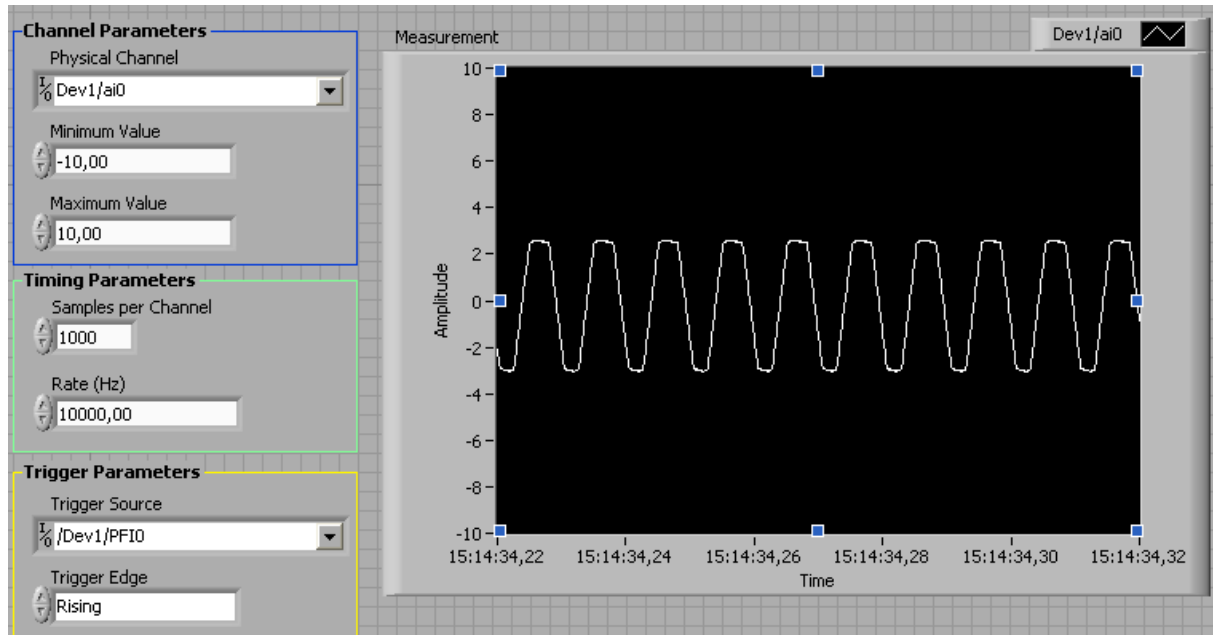


Die (100) auszugebenden Werte (Treppenfunktion) werden in der Sequenzstruktur erzeugt. Rate/Hz legt den Zeitabstand (dt) zwischen zwei aufeinanderfolgenden Werten fest. Nachdem die Messwerte erzeugt wurden werden sie an das Schreib-VI übergeben, das aber diese 100 Werte in Wirklichkeit diese Werte nur „zur Ausgabe“ ablegt. Die eigentliche Ausgabe beginnt erst mit dem Aufruf des Start-VIs. Das vorletzte VI heisst „Warten bis beendet –VI“. Es ist notwendig, damit nicht versucht wird, den Task zu beenden, bevor dieser mit der Ausgabe der Werte fertig ist.

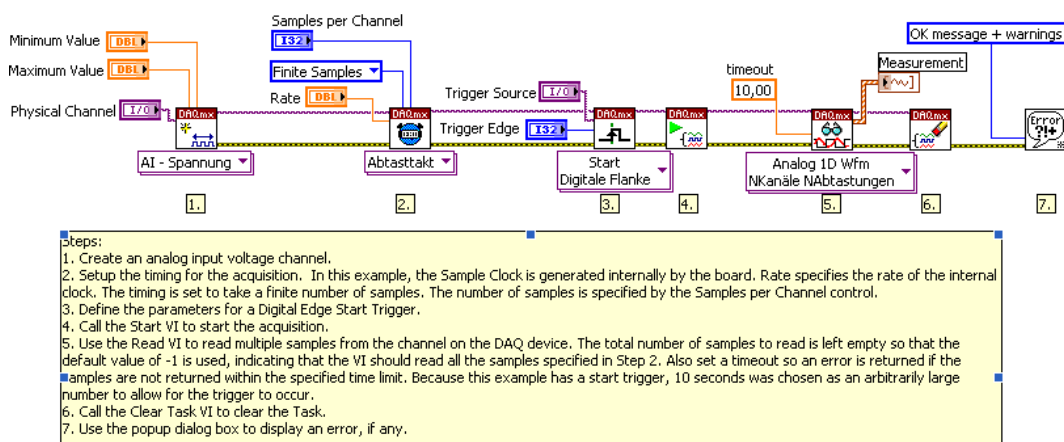
Wenn Sie das obige VI (Bufferausgabe) und das davor beschriebene VI (Acq & Graph Voltage-Int Clk.vi) gleichzeitig aufrufen und mit niedrigen Abtastfrequenzen (ca 10 Hz) arbeiten können Sie die Mechanismen von gebufferter Ein- und Ausgabe sehr anschaulich studieren.

## Getriggerte Messdatenerfassung

Wenn man erreichen will, dass die Erfassung der Messdaten nicht zu einem beliebigen Zeitpunkt erfolgt sondern dass die Erfassung durch ein ganz bestimmtes Ereignis ausgelöst werden soll, muss man getriggerte Messdatenerfassung betreiben. Hier das einfachste Beispiel dazu aus der Reihe der von LabVIEW angebotenen Anwendungsbeispielen (Acq & Graph Voltage-Int Clk-Dig Start.vi):



Sobald am Trigger Eingang (Trigger Source) eine positive Flanke eines digitalen Signals angelegt wird, wird der Messvorgang ausgelöst. Eine Möglichkeit eine solche positive Flanke zu erzeugen bekommt man, indem man den TTL Square Wave Ausgang verwendet. (Wenn man dabei den Sine/Triangle Ausgang abtastet beginnt die Messung immer im gleichen Punkt einer Periode. Legt man einfach +5 V an den Triggereingang, so erhält man wieder zeitlich unterschiedliche Beginnpunkte für die Messung. Das Blockprogramm sieht folgendermassen aus:



In der Mitte sehen Sie das Trigger-VI. Eingangsparameter dabei sind Trigger Source und Triggerkante (Rising oder Falling, d.h. dig. Übergang von 0 V nach 5 V (Rising), bzw. von 5V nach 0V (Falling)).

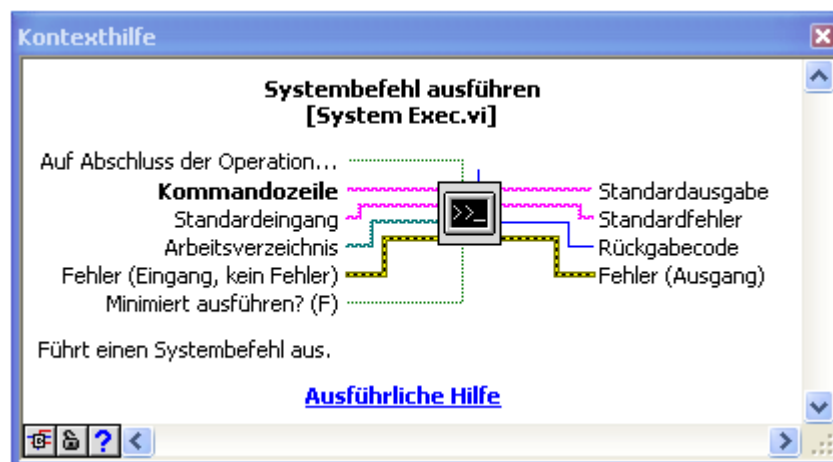
## Einbinden von fremdem Programmcode

In LabVIEW ist es möglich Programmteile einzubinden, die in einer anderen Programmiersprache (C, Pascal, Fortran, Basic) erstellt wurden. Das ist besonders dann wichtig, wenn schon grosse Teile eines Programms in einer anderen Sprache vorliegen und man sich die Mühe ersparen will, diesen Programmcode in LabVIEW zu übertragen. Ein weiterer wichtiger Grund ist die Verwendung von Hardware von Drittherstellern. Dabei wird nämlich häufig Programmcode mitgeliefert, der entweder wieder umgeschrieben werden müsste oder in Form einer „Dynamic Link Library“ vorliegt. Drei grundlegend verschiedene Möglichkeiten zum Einbinden fremden Programmcodes sollen hier kurz beschrieben werden. Sie finden diese Möglichkeiten unter Funktionen/Konnektivität/Bibliotheken & Programme:



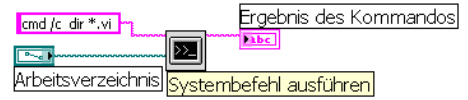
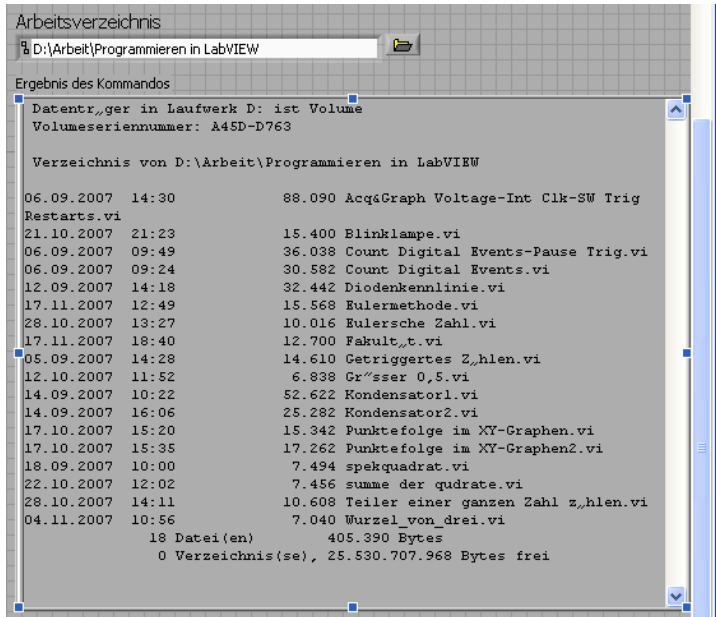
### 1. Anforderung: Aufruf eines externen Programm

Man will aus einem LabVIEW-Programm heraus ein getrenntes ausführbares Programm aufrufen, das auf dem Arbeitsrechner installiert ist. Ein solcher Aufruf erfolgt über das VI „Systembefehl ausführen“ [System Exec.vi]. Die Kontexthilfe dazu sehen Sie im folgenden Bild:



Im Beispiel auf der nächsten Seite wurde zuerst der DOS-Kommandointerpreter aufgerufen und dann alle Dateien im Arbeitsverzeichnis angezeigt, die mit „vi“ enden.





## 2. Anforderung: Einbinden einer DLL-Datei.

Mit DLL-Dateien kann man den von Anwendungen benötigten Platz im Arbeitsspeicher und auf der Festplatte reduzieren, indem Programmcode, der von mehr als einer Anwendung benötigt wird, in einer einzigen Datei auf der Festplatte gespeichert wird. Ebenso wird dieser Programmcode auch nur einmal in den Arbeitsspeicher geladen, auch wenn mehrere verschiedene im Arbeitsspeicher befindliche Programme diesen Programmcode benötigen.

Um aus einem LabVIEW-Programm heraus eine Funktion zu verwenden, die sich in einer DLL-Bibliothek befindet, muss man vier Dinge wissen:

1. Bibliotheks-Name und Pfad zu dieser Datei
2. Funktionsname
3. Datentypen der Eingangsparameter und des Rückgabewerts der Funktion
4. Art des Funktionsaufrufes (Calling-Convention)

Wie kommt man an diese Angaben? Entweder hat man auch die DLL-Bibliothek selbst programmiert, dann ist man fein heraus, oder man findet sie in der Dokumentation zu der betreffenden Bibliothek.



Im folgenden Beispiel werden vom Hersteller einer USB-Messkarte Funktionen per DLL-Datei bereitgestellt. Nach der Installation der Software gibt es ein Verzeichnis auf der Festplatte, in der man diese DLL-Datei finden kann. Der Aufruf einer DLL-Prozedur erfolgt mit dem VI „Call Library Function Node“. Links sehen sie die Kontext-Hilfe dazu. Auf der linken Seite sehen Sie die Eingabeparameter und auf der rechten Seite die Rückgabewerte. Um diese richtig

zuzuordnen benötigt man wie gesagt eine präzise Dokumentation der verwendeten DLL und der darin bereitgestellten Funktionen und Prozeduren.

Aus dem Handbuch zur verwendeten DLL entnimmt man etwa folgende Informationen über die Prozedur „OutputAnalogChannel“:

## OutputAnalogChannel

### Syntax

```
PROCEDURE OutputAnalogChannel(Channel: Longint; Data: Longint);
```

### Parameter

**Channel:** Wert zwischen 1 und 2, der mit der 8-Bit DA Kanal-Nummer, deren Daten eingestellt werden sollen, übereinstimmt.

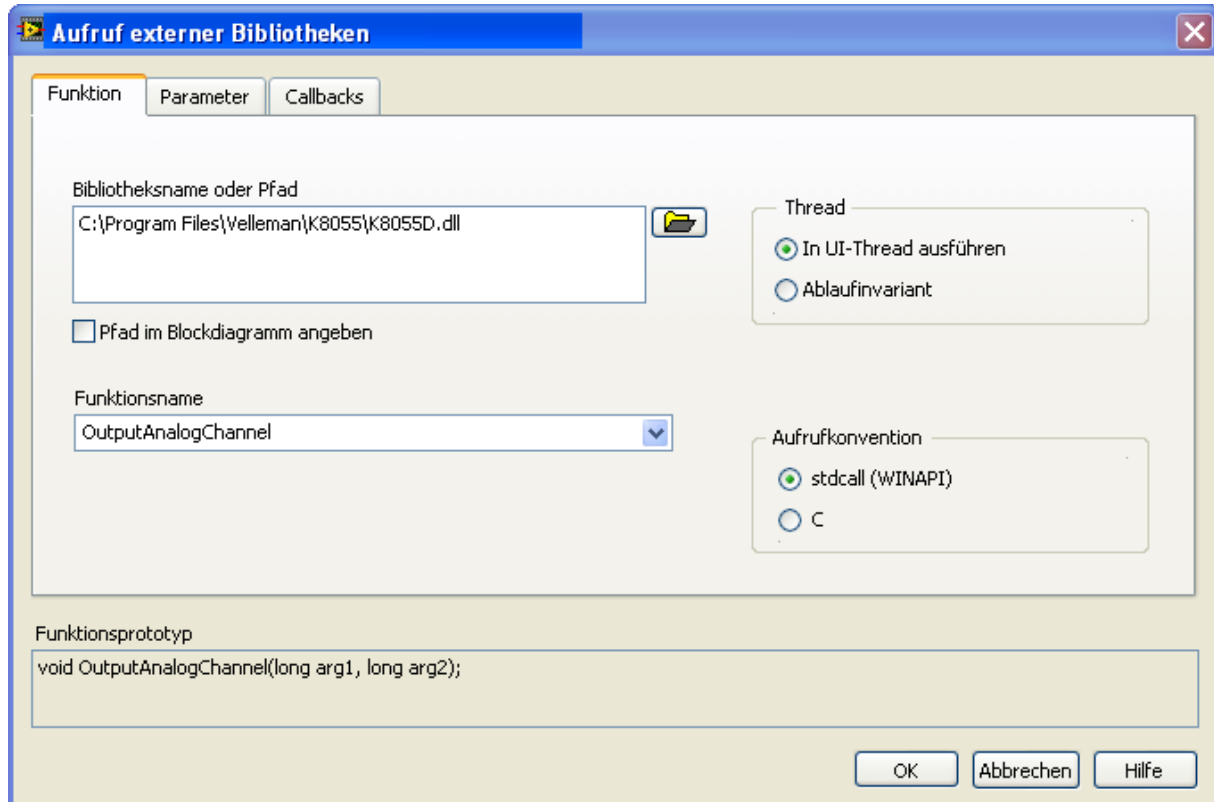
**Data:** Wert zwischen 0 und 255, der zu dem 8-Bit 'Digital to Analogue Converter' geschickt werden soll.

### Beschreibung

Der angegebene 8-Bit 'Digital to Analogue Converter'-Kanal ist nach den neuen Daten geändert worden. Das bedeutet, dass die Daten einer spezifischen Spannung entsprechen. Der Wert 0 stimmt mit der minimalen Ausgangsspannung überein (0 Volt) und der Wert 255 ist die maximale Ausgangsspannung (+5V).

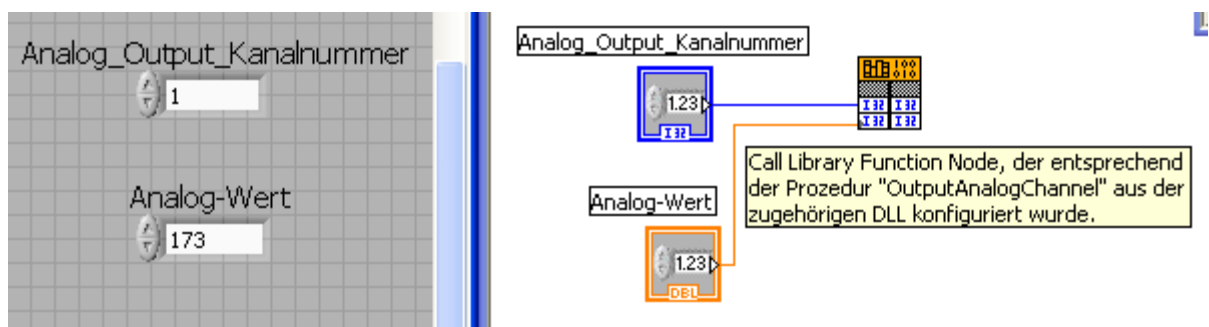
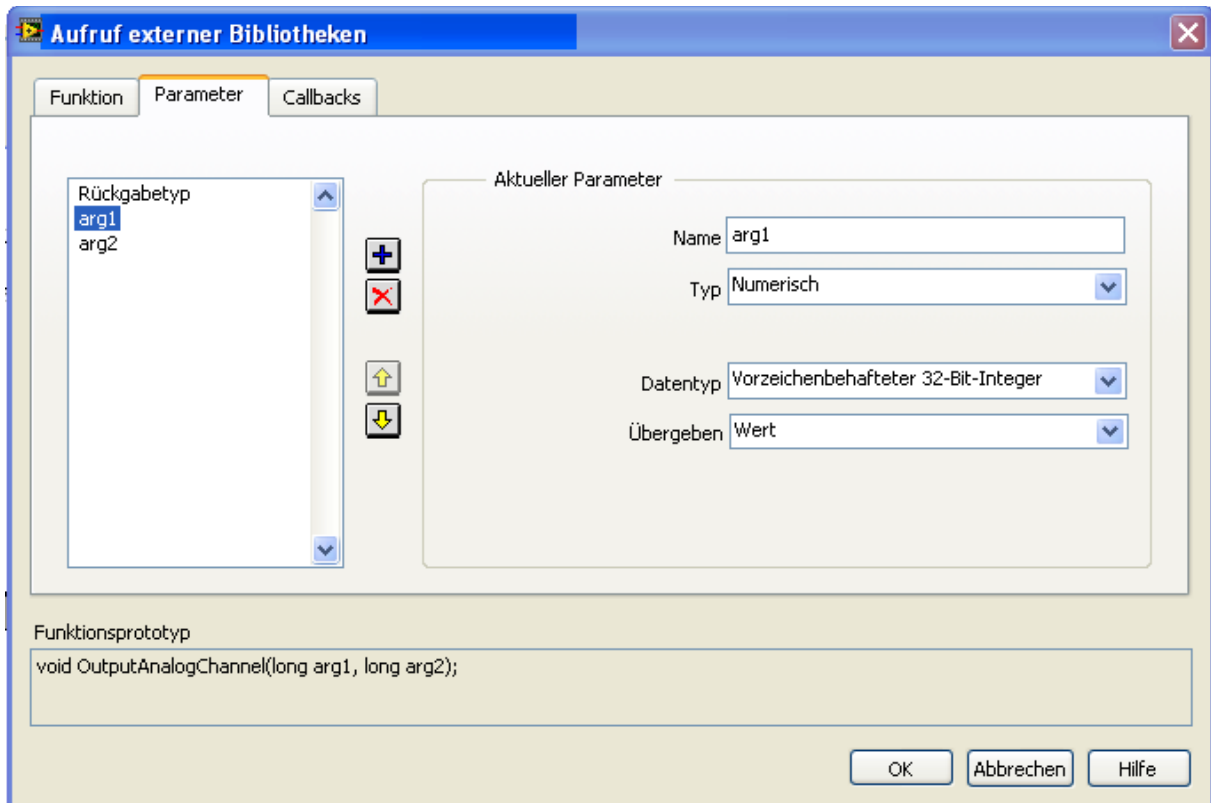
Man ersieht aus dieser Beschreibung, dass OutputAnalogChannel also eine Prozedur mit zwei Eingabeparametern ist, von denen der erste Parameter („Channel“) die Kanalnummer betrifft, er ist vom Datentyp Longint und kann im vorliegenden Beispiel vernünftigerweise nur zwei Werte annehmen, nämlich 1 oder 2, da nur 2 Ausgabekanäle auf der Karte vorhanden sind. Der zweite Parameter, ebenfalls mit Datentyp longint, betrifft den Wert der auszugebenden Spannung („Data“). Dieser Wert kann sich zwischen 0 und 255 bewegen, wobei 0 die Spannung 0 Volt und 255 die Spannung 5 Volt erzeugt.

Das VI „Call Library Function Node“ wird nun per rechtem Mausklick und Auswahl von Configure an diese Prozedur angepasst.



Sie sehen in der oben durchgeführten Konfiguration im ersten Feld den Pfad zur verwendeten DLL (K8055D.dll). Im Feld Funktionsname wird der Prozedurnamen ausgewählt. Das Feld Aufrufkonvention betrifft die Aufrufvereinbarung - in Windows wählt man dazu „WINAPI“. (Die andere Möglichkeit führt gnadenlos zum Absturz Ihres Programms).

Die Registerkarte Parameter betrifft die Datentypen der Parameter der Prozedur:



Anschliessend stellt man die Ein- und Ausgabeparameter ein. Im untersten Feld wird dargestellt, wie der Prototyp der Prozedur jeweils aussieht. Das ist alles! Im Blockdiagramm ist von all diesen Einstellungen nichts zu sehen:

### 3. Anforderung: Verwendung von selbst geschriebenem C-Programmcode

Man will ein selbst geschriebenes und selbst kompiliertes Programm innerhalb von LabVIEW verwenden. Dieser Fall tritt dann ein, wenn man sich zu einem früheren Zeitpunkt schon mal viel Mühe mit der Programmierung einer Aufgabe gegeben hat und diese Mühe nicht umsonst gewesen sein soll. Manchmal gibt es in Pro-

grammen auch sogenannte Flaschenhalse. Man versteht darunter Programmteile, die besonders zeitintensiv sind und meist häufig aufgerufen werden müssen, und die dann das Gesamtprogramm erheblich verlangsamen. Dann ist es manchmal von Vorteil ein C-Programm zu schreiben, das die zu bearbeitende Aufgabe ganz besonders schnell löst. Solche Programme werden per Code Interface Node (CIN) ähnlich wie im obigen Beispiel die DLL-Prozeduren in das darüber liegende LabVIEW-Programm eingebettet. Es gibt hier so viele Gesichtspunkte die zu betrachten wären, dass hier in diesem Einführungskurs nicht darüber gesprochen werden kann. Ein Punkt ist offensichtlich: LabVIEW-Programme, die CINs enthalten, können nicht zum Beispiel auf Mac-Rechnern entwickelt werden und später auf Windows-Rechnern ablaufen.

## LabVIEW und das Internet

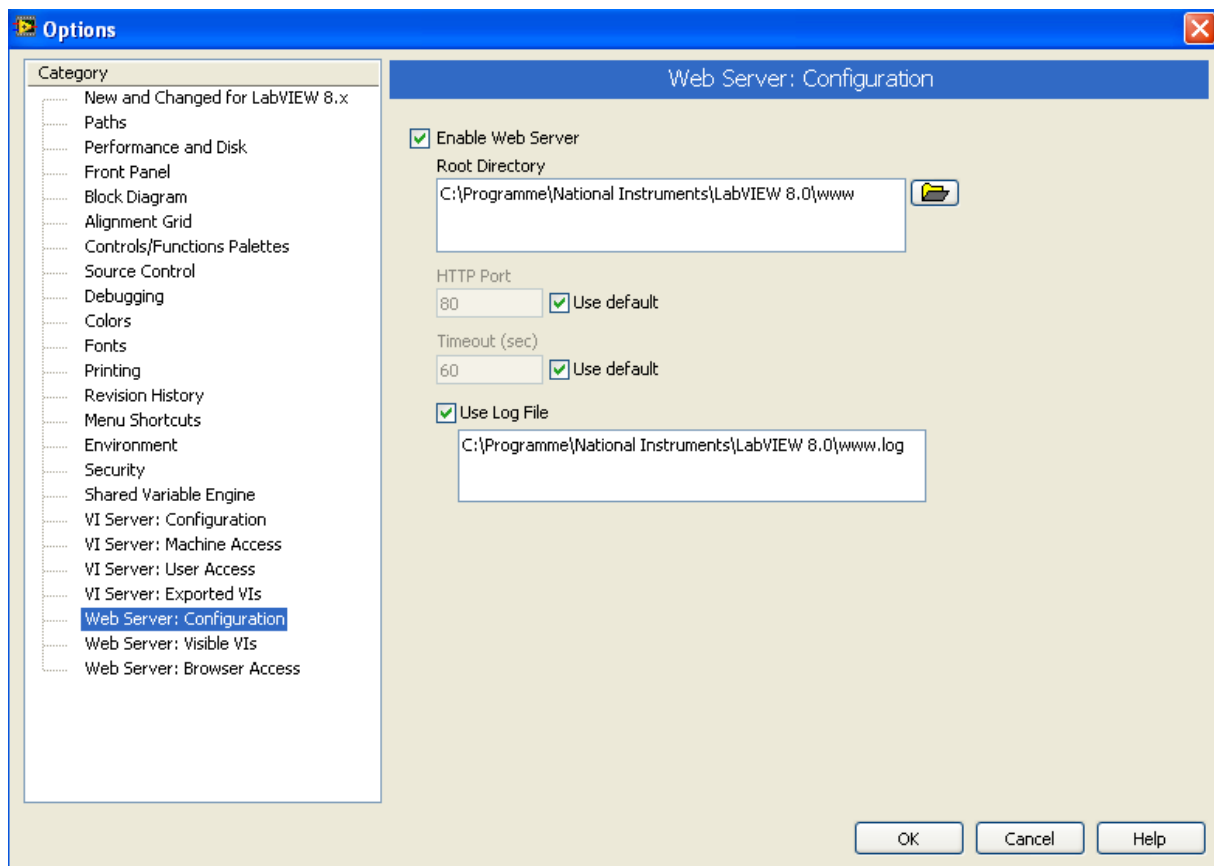
Grundsätzlich können zunächst zwei Aufgaben gelöst werden. Die erste Aufgabe besteht in der simplen Veröffentlichung der Bilder von VI-Frontpanels im Internet. Das ist so zu verstehen, dass Messergebnisse oder Prozesszustände, die an einer Stelle erfasst werden, an ganz anderer Stelle beobachtet werden können, ohne dass der Betrachter die Möglichkeit hat, die eigentliche Messung/den eigentlichen Prozess irgendwie zu beeinflussen.

Die zweite Aufgabe behandelt aber auch die Möglichkeit über das Internet Messungen zu beeinflussen, indem die Steuerelemente eines Vis sozusagen über das Internet fernbedient werden können und so der ganze Messprozess oder Teile davon ausgelagert werden können.

Betrachten wir zunächst die erste Möglichkeit.

### **Veröffentlichung von Vis im Internet**

Mit dem in LabVIEW eingebauten Webserver können dynamisch Webseiten erstellt werden, die Bilder des Frontpanels des zu veröffentlichenden Vis enthalten. Zunächst muss dazu der in LabVIEW eingebaute Webserver aktiviert werden. Dies erfolgt im Menüpunkt Tools/Options/Web Server: Configuration. Stellen Sie dabei die folgenden Werte ein:



Nachdem die Konfiguration bestätigt wurde, muss LabVIEW beendet und neu gestartet werden. Erst dann ist der WebServer aktiv. Dieser Webserver macht nichts anderes, als dynamisch Bilder eines VI-Frontpanels zu erstellen, wenn eine entsprechende Web-Browser-Anforderung an den Webserver gestellt wurde. Um dies vorzuführen starten wir nun das Beispiel Temperature System Demo.vi aus der Library examples\apps\tempsys.llb in C:\Programme\National Instruments\LabVIEW 8.0.

Um ein statisches Bild des Frontpanels von Temperature System Demo.vi muss nun in einem Web-Browser entweder die Adresse: `http://127.0.0.1/.snap?Temperature+System+Demo.vi` oder `http://127.0.0.1/.monitor?Temperature+System+Demo.vi` eingegeben werden. Die erste Adresse liefert ein einmaliges Bild, das mit dem Refresh-Button des Browsers immer wieder aktualisiert werden kann, während die zweite Adresse (mit monitor?) immer wieder neue Bilder anfordert (ohne, dass der Refresh-Button benötigt wird). Die Komponenten der Web-Adresse haben folgende Bedeutung:

127.0.0.1 ist die Adresse des lokalen WebServers. Würde man das auf einem anderen Rechner laufende VI-Frontpanel betrachten wollen, so müsste hier die IP-Adresse des betreffenden Rechners eingegeben werden.

.snap? Ist ein spezielles Kommando, das den Web-Server auffordert ein einzelnes Bild des im Anschluss an „.snap?“ folgenden VI-Frontpanels anzufertigen und auszuliefern.

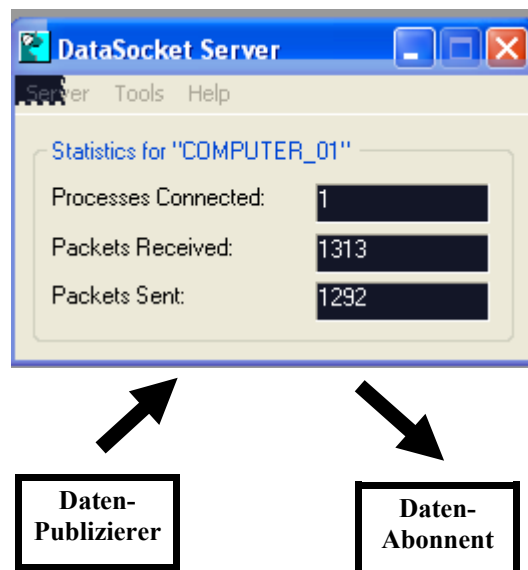
.monitor? Entspricht .snap? bis auf den Unterschied, dass eine Animation von Bildern des aufgeführten VI-Frontpanels erzeugt wird.

Temperature+System+Demo.vi ist das zu beobachtende VI. Das Pluszeichen muss anstelle der im VI-Namen enthaltenen Wortabstände eingefügt werden. Statt des +-Zeichens könnte auch: `Temperature%20System%20Demo.vi` geschrieben werden.

## Austausch von LabVIEW-Daten im Internet

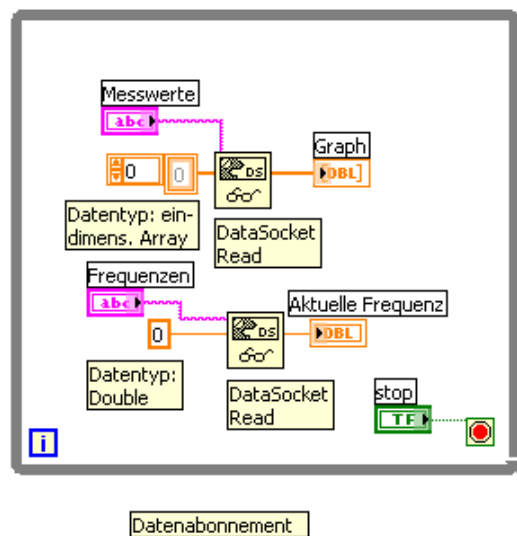
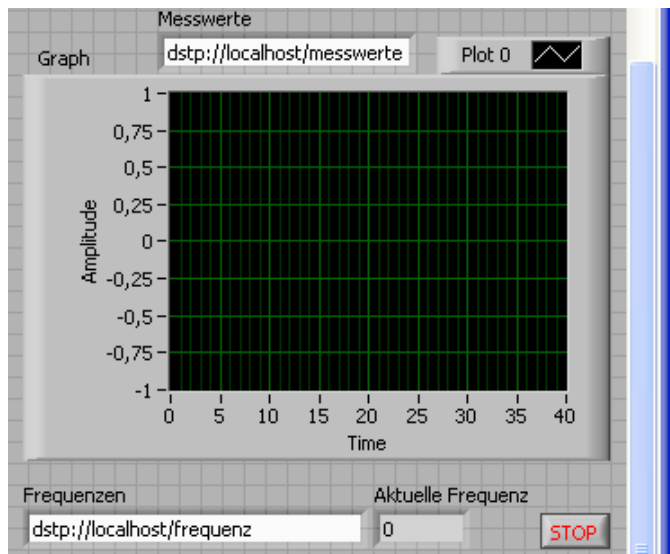
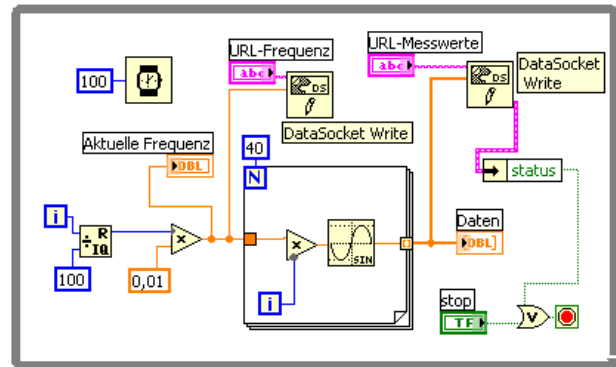
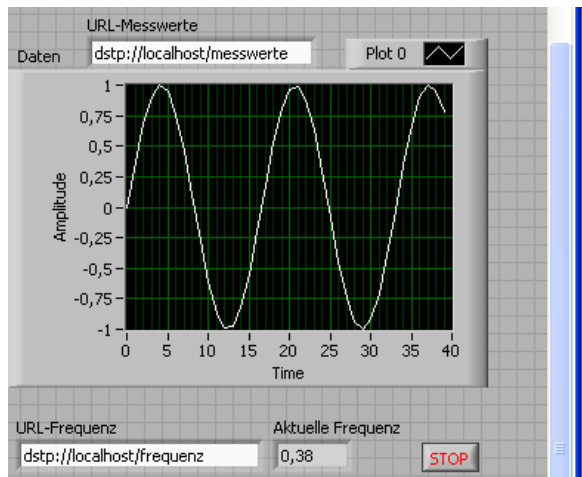
Mit dem in LabVIEW eingebauten Webserver können, wie wir oben gesehen haben, Bilder der Frontpanels von VIs im Internet übertragen werden. Das ist schön, aber bei weitem nicht ausreichend, denn normalerweise sollen die an einer Stelle gemessenen Daten entweder vor Ort, oder an anderer Stelle direkt weiterverarbeitet werden können. Zu diesem Zweck gibt es die LabVIEW-spezifische Schnittstelle DataSocket. Weitergehende Informationen dazu finden Sie unter: <http://ni.com/datasocket>. Die Übersetzung von Socket ist: Anschlussdose, Steckdose, Buchse. Über ein „DataSocket Write-VI“ übermittelt ein VI LabVIEW-Daten an den DataSocket Server (man spricht dann von Publikation bzw. Veröffentlichung). Analog kann ein VI per „DataSocket Read-VI“ LabVIEW-Daten vom DataSocket Server abholen (in diesem Fall wird von Subskription bzw. Abonnement gesprochen). Der DataSocket Server ist ein Programm, das Sie in Windows unter Start/Programme/National Instruments/DataSocket/DataSocket Server finden und aufrufen können. Das der DataSocket-Schnittstelle zugeordnete Protokoll heisst: „DSTP“ (DataSocket Transfer Protokoll). Die Adressen der Messdaten werden dann ähnlich wie Webdokumente im Internet etwa durch „dstp://localhost/messdaten“ gekennzeichnet. (In diesem Fall befinden sich die Messdaten auf dem lokalen Rechner). Ansonsten muss eine Internetadresse oder ein Rechnername angegeben werden. DataSocket ist also im wesentlichen ein Stück Software, das erlaubt, in einem Rechnernetz Daten zu versenden und zu empfangen. Dieses Konzept erspart dem Programmierer die Mühe, sich über netzwerkspezifische Probleme den Kopf zerbrechen zu müssen, da diese Aufgabe vom DataSocket Server zusammen mit den oben erwähnten VIs „DataSocket Read“ und „DataSocket Write“ übernommen wird. Das DataSocket Protokoll ist auch in anderer Umgebung einsetzbar. Dazu gehören: C++, Visual Basic und Java. Die zu übertragenden Daten können Integer-, Floating Point-, String-, Boolean-Daten sein, sowie Arrays, die daraus gebildet wurden.

Im nächsten Bild wird das eben beschriebene DataSocket Konzept noch einmal graphisch dargestellt:

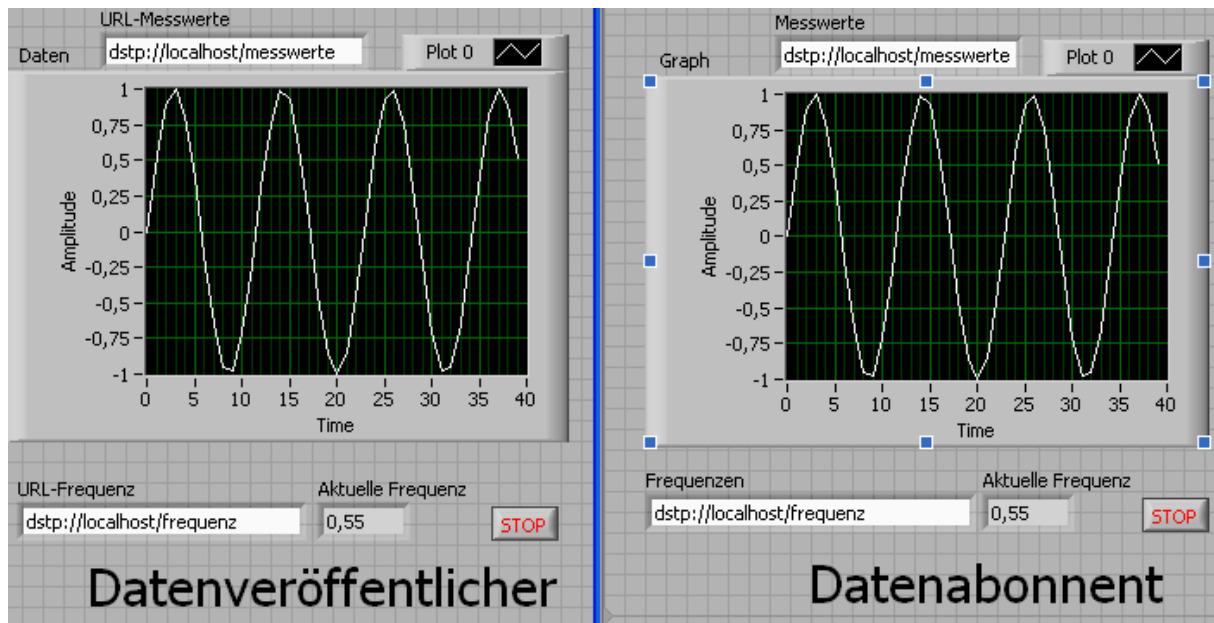




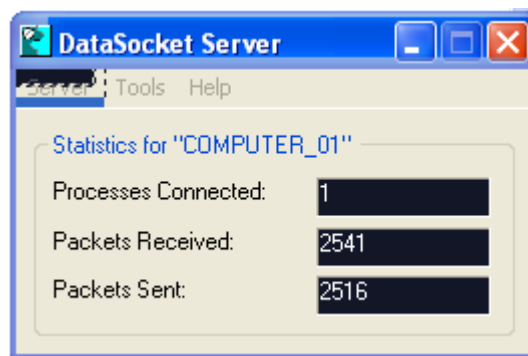
Im folgenden sehen Sie die Programme für einen Daten-Veröffentlicher, der eine Reihe von Sinuskurven mit sich verändernder Frequenz erzeugt und sowohl diese Sinusdaten, als auch die zugeordnete aktuelle Frequenz zur Veröffentlichung über den DataSocket Server freigibt. Das zweite Programm, der Daten-Abonnent holt sich diese Daten vom DataSocket Server



Man lässt nun den Veröffentlicher und den Abonnenten gemeinsam laufen und erhält den auf der nächsten Seite dargestellten Sachverhalt.



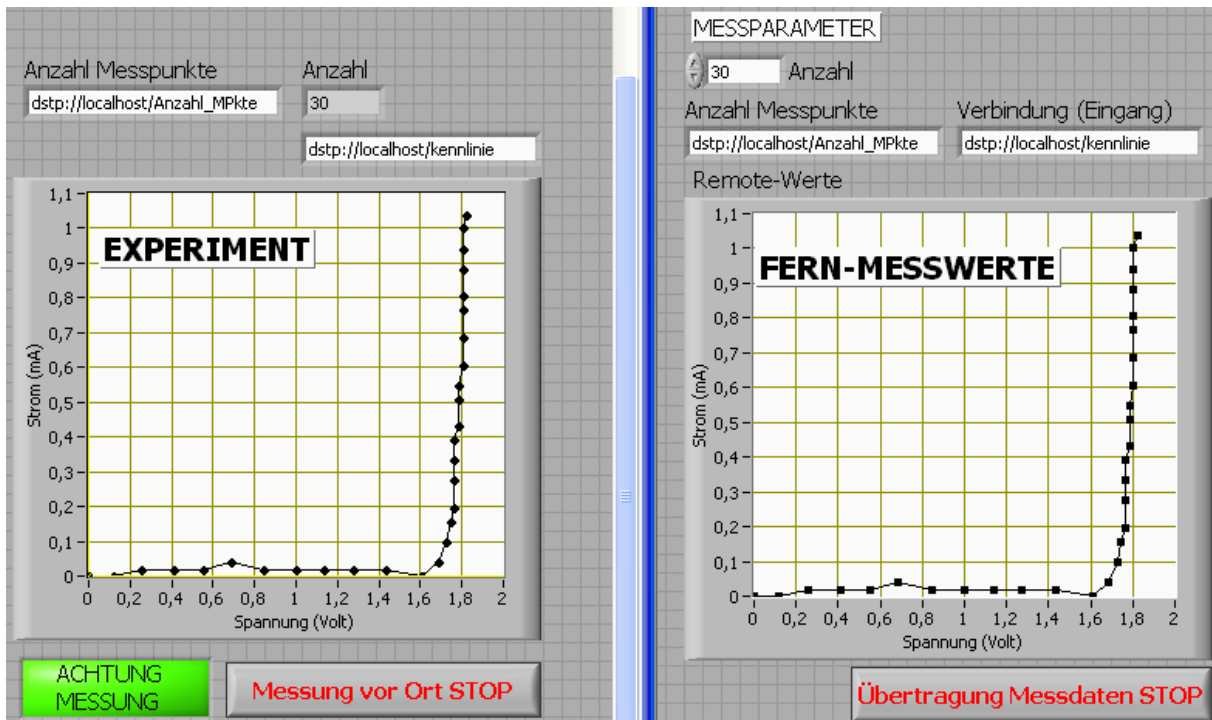
Während die beiden Programme laufen kann man sich gleichzeitig noch Informationen des DataSocket Servers ansehen:



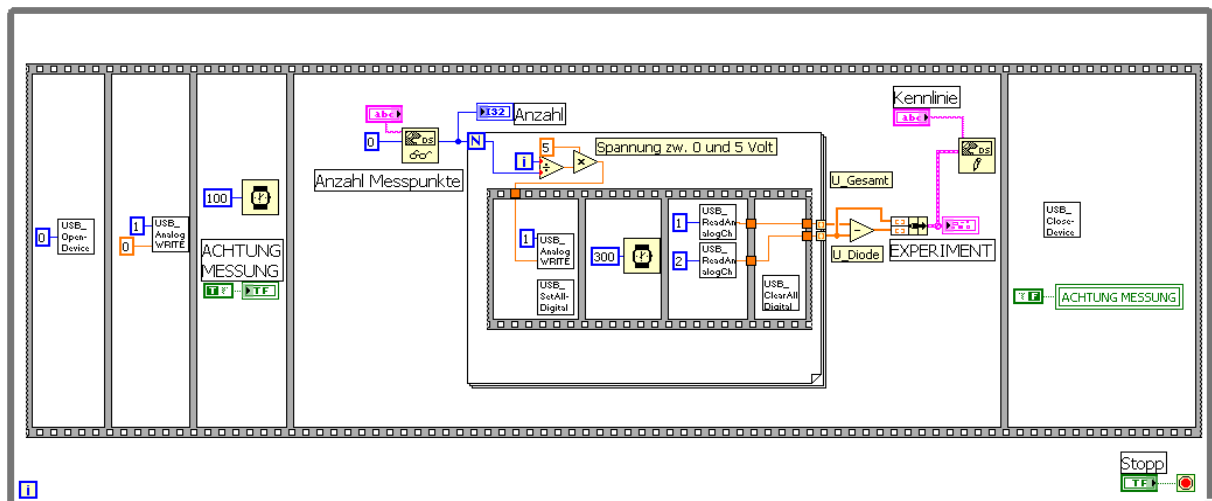
Die Zahl „Processes Connected“ bezieht sich offensichtlich auf die Zahl der Datenabonnenten, die beim DataSocket Server Datenpakete abholen.. In der zweiten und dritten Zeile werden die Zahlen über die aktuell verarbeiteten Pakete aufgeführt.

## Diodenkennlinienmessung im Internet

Mit der aus den Übungen bekannten Messdatenerfassungskarte von Velleman würde eine einfache Erfassung einer Diodenkennlinie über das Internet an den einzelnen Standorten Frontpanelmässig wie unten aussehen. Man muss dabei zunächst rechts die Messparameter (Anzahl der Messpunkte) eingeben. Diese werden dem Experiment über das Netz zugeschickt, das dann die Messung damit durchführt und anschliessend das Ergebnis der Messung dem Kennlinienleser zurück übermittelt.

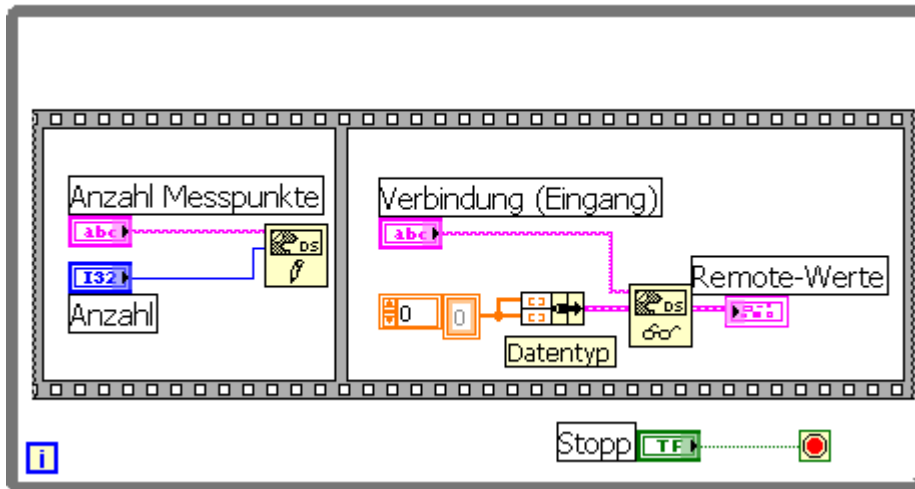


Das Programm des Kennlinienmessers (**EXPERIMENT**) sieht so aus:



Im grossen Rahmen der verwendeten Sequenz wird in der For-Schleife die eigentliche Messreihe erfasst. Vor jeder einzelnen Messung werden alle 8 digitalen Outputdioden angeschaltet und danach wieder gelöscht. Wenn der verwendete Widerstand für die Stromberechnung genau 1 K $\Omega$  gross ist, dann ist  $(U_{\text{Gesamt}} - U_{\text{Diode}})$  der Diodenstrom in Milliampere.

Das Programm des Kennlinienlesers (**FERN-MESSWERTE**) ist relativ einfach:



Oben wird im ersten Rahmen zunächst die Anzahl der zu erfassenden Messpunkte an das Remote-Experiment geschickt.

Im zweiten Rahmen wird die Grafik, die entfernt erstellt wird, geholt und dargestellt. Dazu muss dem Data-Socket-Leser-VI der Datentyp der Grafik erklärt werden. Man erzeugt sich dazu einen leeren eindimensionalen Floatingpoint-Array, der dann jeweils im Cluster den Array der X-Werte und den Array der Y-Werte darstellt.

Das war „Programmieren in LabVIEW“ im WS 2007/08. Viel Erfolg beim weiteren Einsatz von LabVIEW!

Falls Sie noch Anregungen und Beispiele suchen: unter <http://cnx.org/content/m15510/latest/> finden Sie eine spannende multimediale Umgebung um einerseits LabVIEW zu erlernen und andererseits mit LabVIEW die musikalische Signalverarbeitung zu erforschen.