**Technische Universität Berlin**

Institut für Konstruktion, Mikro- und Medizintechnik

**Fachgebiet Mikrotechnik**

**Prof. Dr. rer. nat. Heinz Lehr**

# Integrierte Lehrveranstaltung Engineering Tools / Bachelor

Übungseinheit

# LabVIEW

Sommersemester 2016

Übungsleiter:

**Dipl.-Ing. Kilian Helfmeier**

Tel. 030 - 314 79886

helfmeier@fmt.tu-berlin.de

## 1. Aufbau von LabVIEW

Ein LabVIEW-Programm wird als virtuelles Instrument (VI) bezeichnet, da sein Aussehen und die Funktionalität sich an ein reales Instrument anlehnen.
Jedes VI wird in zwei Ebenen programmiert, die beim Start von LabVIEW erscheinen:

(a) **Frontpanel**: interaktive Benutzeroberfläche. Hier können beispielsweise Zahlen vorgegeben oder angezeigt werden (z. B. in Graphen und Diagrammen) sowie Knöpfe und Schieberegler betätigt werden.

(b) **Blockdiagramm**: enthält die Programmlogik. Das Blockdiagramm ist das tatsächlich ausführbare Programm. Hier können beispielsweise arithmetische Funktionen, Ablaufstrukturen, Schleifen (for, while) durchgeführt werden.

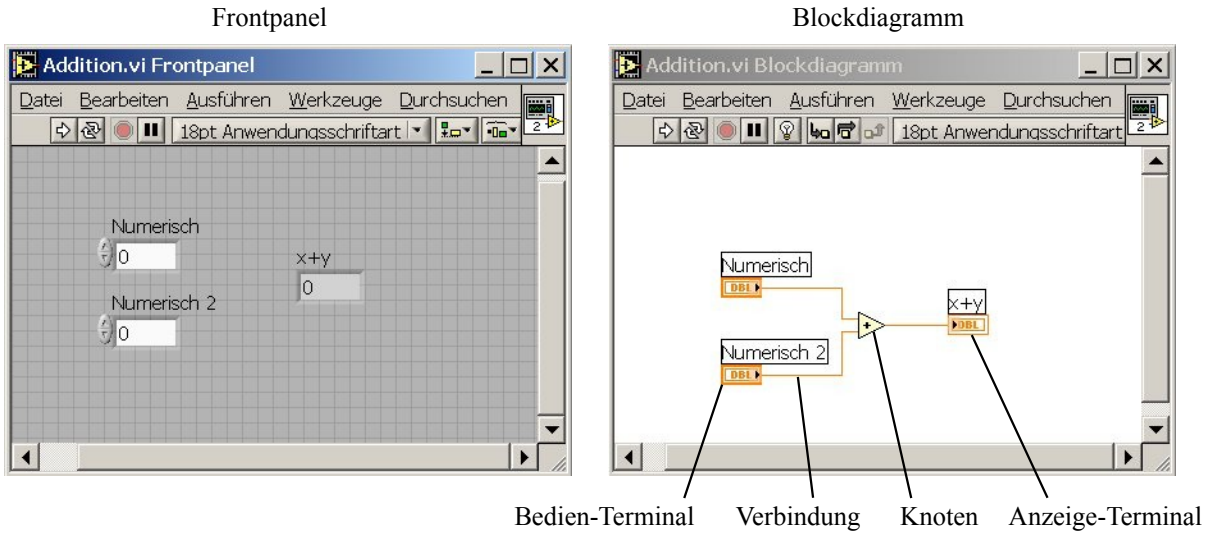



**Abb. 1-1 Beispiel eines einfachen LabVIEW VI's zur Addition zweier Zahlen**

Die wesentlichen Bestandteile des Programms sind die Terminals, die als Ein- oder Ausgabe dienen, sowie die Knoten als programmausführende Elemente. In klassischen Programmen entspricht dies einem Operator (wie in diesem Beispiel ein Plus-Operator) oder einer Subroutine. An die Knoten werden Werte/Parameter (über Verbindungsdrähte) übergeben und andere Werte von diesen zurückgegeben. Beliebige Teile des Programms können als Unterprogramm (SubVI) gespeichert werden und mit einem neu gestalteten Symbol in ein anderes Programm eingefügt werden.

Frontpanel und Blockdiagramm haben korrespondierende Anschlüsse, so dass Daten vom Anwender an das Programm übergeben werden können und umgekehrt. D. h. immer, wenn auf dem Frontpanel ein Ein- oder Ausgabefeld platziert wird, entsteht im Blockdiagramm ein dazu entsprechendes Terminal.

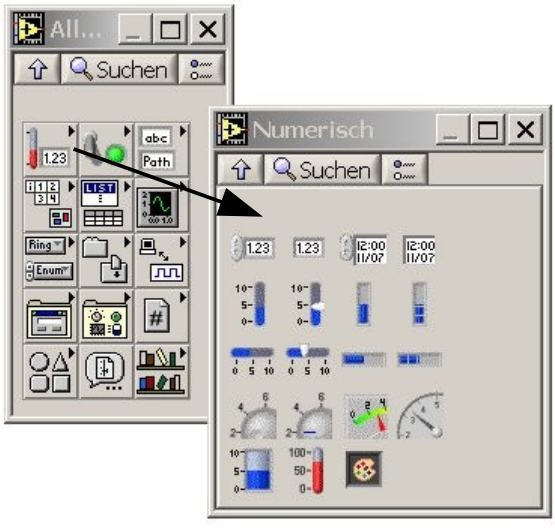
LabVIEW Ausdrücke

| **LabVIEW** | **konventionelle Sprache** |
|---|---|
| VI (virtual instrument) | (Haupt-) Programm |
| Funktion | Funktion |
| SubVI | Unterprogramm |
| Frontpanel | Benutzeroberfläche |
| Blockdiagramm | Sourcecode oder Quelltext |

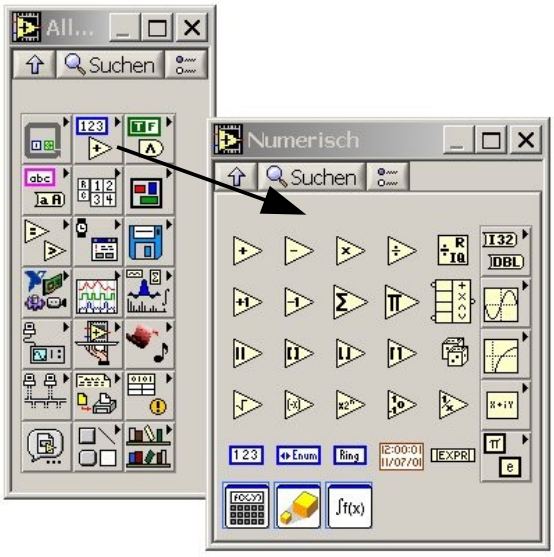
## 2. Paletten

Alle Elemente (Bedienterminal, Ausgabe-Terminal, arithmetischer Knoten, etc.), die im Frontpanel (FP) oder Blockdiagramm (BD) verwendet werden, können aus den entsprechenden Paletten (**Elementenpalette** für das FP und **Funktionenpalette** für das BD) entnommen werden. Diese befinden sich in der oberen Menüleiste unter *Ansicht >> Elementenpalette* oder *Ansicht >> Funktionenpalette*. Die Elementenpalette enthält sämtliche Elemente, die zur Bedienung und interaktiven Steuerung des Programms notwendig sind (Zahleneingaben, Schalter, Knöpfe, Graphen, etc.), während die Funktionenpalette alle Elemente zur Erstellung des Programms enthält (arithmetische und boolesche Funktionen, Schleifen, etc.).

Beim Umschalten zwischen FP und BD ist nur die zugehörige Palette eingeblendet. Die Palette des inaktiven Fensters wird ausgeblendet. Wird ein Element ausgewählt, kann es nur in dem entsprechenden Fenster eingefügt werden.




Elementenpalette und die Unterpalette „Numerisch“




Funktionenpalette und die Unterpalette „Numerisch“

**Abb. 2-1 Elementenpalette/Funktionenpalette**

In der **Werkzeugpalette** kann der Maus ihre Funktion zugeteilt werden. Die enthaltenen Werkzeuge sind vergleichbar mit den Werkzeugen eines Zeichenprogramms. Die Werkzeuge gelten sowohl im FP als auch im BD. So kann man beispielsweise auswählen, ob man Eingaben und Anzeigen bedienen (Hand), Objekte positionieren (Pfeil) oder beschriften (Buchstabe „A“) oder Knoten und Anschlüsse miteinander verbinden möchte (Rolle).

## 3. Erstellen von Formelknoten

In Abbildung 1-1 haben wir gesehen, wie man einfache arithmetische Operationen graphisch programmieren kann. Wird die Formel jedoch komplexer, so wird sie in der graphischen Programmierweise auch schnell unübersichtlich. Die nachfolgende Abbildung zeigt eine Gleichung und deren graphische Umsetzung in LabVIEW.

$$\text{Ergebnis} = \sqrt{\frac{x^2 - y}{2}}$$

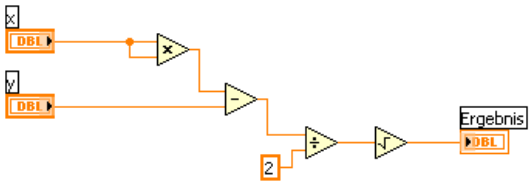



**Abb. 3-1 Graphische Umsetzung einer Gleichung**

Zur besseren Übersichtlichkeit können mathematische Funktionen auch in sogenannten Formelknoten programmiert werden. Der entsprechende Knoten kann im Blockdiagramm unter *Funktionenpalette >> Programmierung >> Strukturen >> Formelknoten* gefunden werden.
Sobald der Formelknoten im BD eingefügt ist, können durch Klicken mit der rechten Maustaste auf den Rand der Formelbox Ein- oder Ausgänge erzeugt werden. Zur Formeleingabe muss in der Werkzeugpalette der Schreibmodus (Buchstabe „A“) aktiviert werden. Wichtig: im Formelknoten muss jede Anweisung mit einem Semikolon abgeschlossen werden!

Die oben genannte Formel sieht dann folgendermaßen aus:

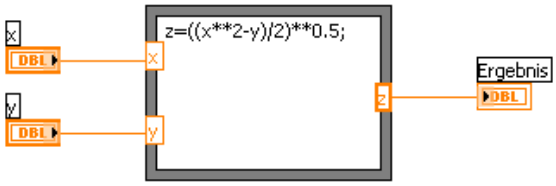



**Abb. 3-2 Formelknoten**

## 4. Cluster

Cluster werden eingesetzt, um eine Vielzahl von Daten zusammenzufassen. Hierbei können sämtliche Datentypen Integer, Float, String, Arrays kombiniert werden. Damit kann die Programmierung deutlich übersichtlicher gestaltet werden.

Cluster werden in zwei Schritten im Frontpanel (FP) erstellt:

(a) In der Elementenpalette das Element Cluster  in *z.B. Modern >> Array, Matrix & Cluster >> Cluster* auswählen und das Symbol in das FP ziehen.

(b) Die entstandene Box nach Bedarf mit Ein- bzw. Ausgabe-Terminals füllen. Wichtig: in einem Cluster können zwar verschiedene Datentypen zusammengefasst werden, jedoch können innerhalb eines Clusters nur Eingaben oder Ausgaben getätigt werden.

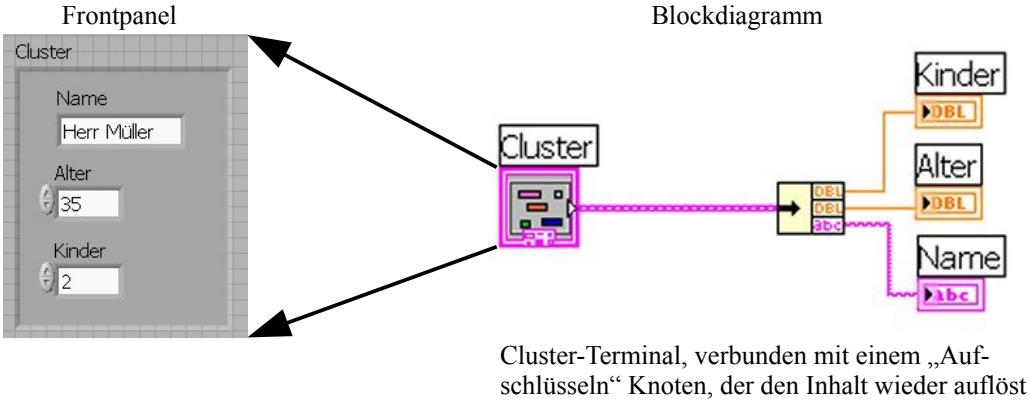



**Abb. 4-1 Cluster-Eingabe-Terminal im FP und BD bestehend aus einem String und zwei numerischen Eingaben. Im BD wird der Cluster zusätzlich wieder aufgelöst und an Anzeige-Terminals weitergegeben.**

### Daten Bündeln und Aufschlüsseln

In Abbildung 4-1 ist ein Beispiel des Aufschlüsselns (Entbündelung) gezeigt. Umgekehrt können Daten auch gebündelt werden, um das VI übersichtlicher zu gestalten. Der Knoten zum Bündeln ist ebenso wie der zum Aufschlüsseln in *Funktionenpalette >> Programmierung >> Cluster* zu finden.

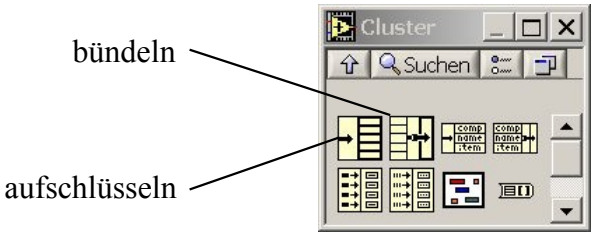



**Abb. 4-2 Daten bündeln**

Abbildung 4-3 zeigt ein Beispiel wie verschiedene Daten zu einem Cluster gebündelt werden.

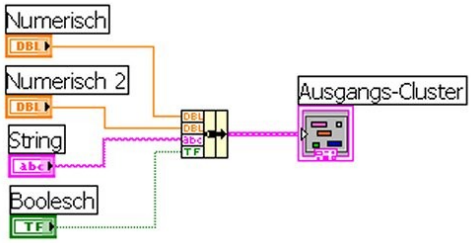



**Abb. 4-3 Verschiedene Daten zu Cluster bündeln**

## 5. Erstellen von SubVI's

Im Sinne einer übersichtlichen Programmierung können beliebige Programmabschnitte als Unterprogramm (SubVI) zusammengefasst werden. Dieses VI wird dann durch ein Symbol oder Anschlussblock gekennzeichnet. Auf diese Weise kann das Programm übersichtlicher gestaltet werden und wiederkehrende Programmabschnitte durch wiederholtes Einfügen des SubVI's schnell erstellt werden.

Wenn Sie das geöffnete VI als SubVI definieren wollen, müssen Sie alle gewünschten Eingaben- und Ausgaben-Terminals mit den Anschlüssen des Anschlussblocks mit Hilfe des Drahtrollenwerkzeugs zuordnend verbinden. Hierzu klicken Sie zunächst mit der rechten


Maustaste auf das VI-Symbol in der rechten oberen Ecke des *Frontpanels* neben dem Hilfemenü und lassen sich die Anschlüsse anzeigen. Der dargestellte Anschlussblock besitzt beispielsweise drei Anschlüsse. In der Regel werden die Eingänge links und die Ausgänge rechts angeordnet. Die Anzahl der Anschlüsse kann über ein Klicken mit der rechten Maustaste und den dort aufgelisteten Befehlen auf maximal 24 Anschlüsse verändert werden.

Anschließend werden die Eingänge des Anschlussblocks mit der Drahtrolle mit den Eingabe-Terminals verbunden. Hierzu klickt man mit der linken Maustaste auf den gewünschten Anschluss im VI-Anschlussblock und klickt anschließend mit der linken Maustaste auf den dazuzuordnenden Eingabe-Terminal im Eingabefeld des Frontpanels. Mit der rechten Maustaste kann der Vorgang abgeschlossen werden. Auf diese Weise werden nacheinander alle Anschlüsse zugeordnet. Die Farbe der zugeordneten Anschlüsse ändert sich bei diesem Vorgang.

Zur Erstellung eines aussagekräftigen Icons betätigen Sie die rechte Maustaste auf dem Anschlussblock und gehen auf *Symbol bearbeiten*. Der Rest erklärt sich von selbst.

Wichtig: ein SubVI kann nicht in sich selber aufgerufen werden!

Dem VI kann ein beschreibender Kommentar beigefügt werden (*Datei >> VI-Einstellungen >> Dokumentation*), der dann bei Aufruf der Kontext-Hilfe unterhalb des VI-Symbols erscheint.
SubVI's können auch auf sehr einfache Weise über den Befehl *Bearbeiten >> SubVI erstellen* automatisch erzeugt werden. Hierzu muss der Bereich des Programms, der zu einem SubVI zusammengefasst werden soll markiert werden. LabVIEW erstellt dann automatisch ein Icon und ein Anschlussblock mit der entsprechenden Anzahl an Ein- und Ausgängen. Beim Beenden des Programms muss man darauf achten, das SubVI zu speichern (achten Sie auf die Abfrage nach Beendigung des Programms!). Um das Symbol nach Wunsch zu bearbeiten kann das SubVI geöffnet werden und das Symbol nach der oben beschriebenen Vorgehensweise verändert werden.

## 6. Arrays

Arrays sind ein- oder mehrdimensionale Felder. Ein Array ist eine Sammlung von Daten, die beispielsweise bei jeder Messreihe entsteht. Im Gegensatz zu Clustern ist ein Array eine Sammlung von Daten gleichen Typs (entweder numerische Zahlen, Strings oder boolesche Werte).

Erstellung von Arrays:
Zum Erstellen eines Arrays sind, vergleichbar zur Clustererstellung, zwei Schritte nötig. Zunächst muss ein Array-Fenster aus der Bedienelementen-Palette *z.B. Modern >> Array, Matrix & Cluster >> Array* entnommen und auf dem Frontpanel platziert werden (Abbildung 6-1). Anschließend muss ein Bedien- oder Anzeige-Terminal des gewünschten Datentyps (numerisch, boolesch oder String) in das Array-Fenster gezogen werden.

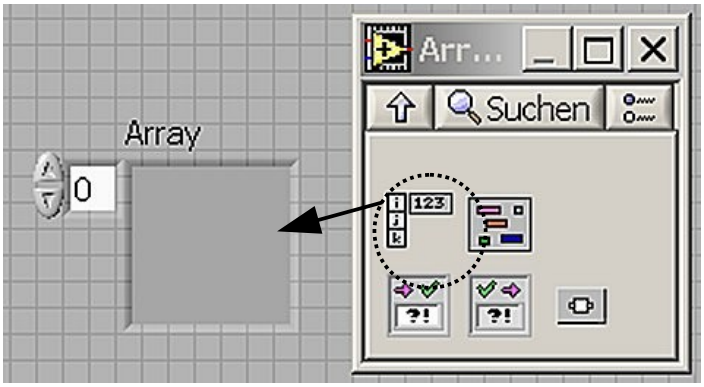



**Abb. 6-1 Erstellen eines leeren Array-Fensters**

In der Funktionen-Palette des Blockdiagramms sind unter *Programmierung >> Arrays* viele Möglichkeiten gegeben ein Array zu verändern (Zeilen und Spalten vertauschen, Arraygröße ermitteln, Elemente einfügen oder entfernen, etc.).

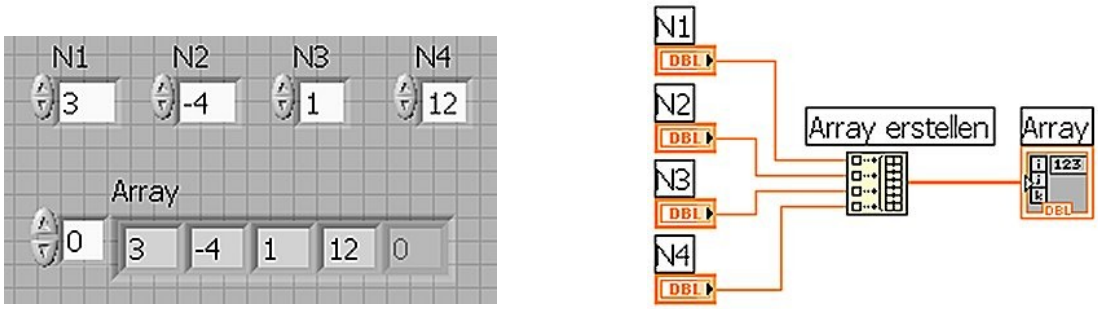



**Abb. 6-2 Erstellung eines Arrays aus Skalaren**

## 7. Ablaufstrukturen

Alle Ablaufstrukturen werden in LabVIEW durch einen Rahmen dargestellt, der sämtlichen Variablen und Befehle, die innerhalb der Ablaufstruktur bearbeitet werden sollen, umgibt. Die vier verschiedenen Ablaufstrukturen (While-Schleife, For-Schleife, die Case-Anweisung und die Sequenz-Struktur) sind in der Funktionen-Palette unter *alle Funktionen >> Strukturen* zu finden und sollen im Folgenden erläutert werden

***While-Schleife***

Die While-Schleife stellt neben der For-Schleife eine von zwei Möglichkeiten dar, sich wiederholende Vorgänge zu realisieren. Abbildung 7-1 zeigt die Schleifenfunktion, die als Rahmen alle Variablen und Befehle, die innerhalb der Schleife bearbeitet werden sollen, umgibt. Die Schleifenzählvariable gibt die bisherige Anzahl der Iterationen i wieder. Der Iterationszähler fängt bei Null an zu zählen wird nach jedem Schleifendurchlauf um eins inkrementiert. Das Iterationssymbol kann an seinem Ausgang „verdrahtet“ werden, um beispielsweise über ein Ausgabe-Terminal angezeigt zu werden. Das Schleifenterminal kann nach Wahl (rechte Maustaste auf das Symbol) ein Wiederholungsterminal oder ein Abbruchterminal sein. An den Eingang des Terminals kann beispielsweise ein boolescher Schalter angeschlossen werden. Solange der Schalter auf TRUE steht wiederholt sich die While-Schleife mit Wiederholungsterminal, während die andere gestoppt wird und nur bei einer FALSE-Eingabe läuft.

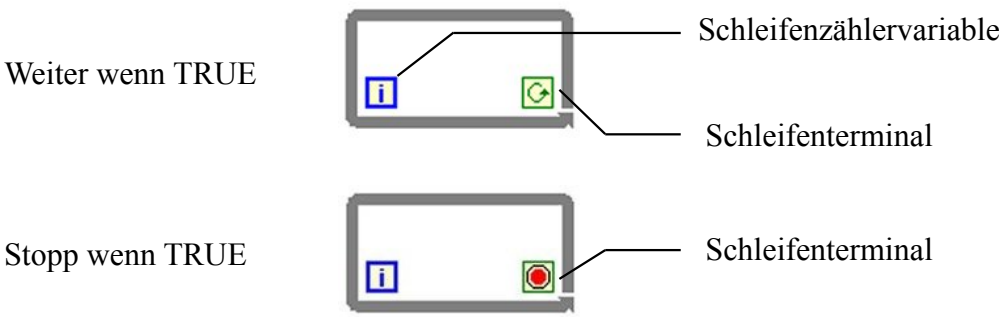



**Abb. 7-1 While-Schleife**

Die While-Schleife wird häufig dazu verwendet, einen „sauberen“ Start oder Stopp in das Programm einzubauen. Zum Stoppen wird das gesamte Programm beispielsweise in eine While-Schleife geschrieben, die solange läuft, bis auf dem Frontpanel ein boolescher Schalter betätigt wird, der das Schleifenterminal zum Abbruch bringt.

***For-Schleife***

Eine For-Schleife führt alle Anweisungen, die innerhalb ihrer Grenzen enthalten sind, so oft aus, wie das Zählerterminal N bestimmt, das von außen mit einer Zahl versorgt wird.

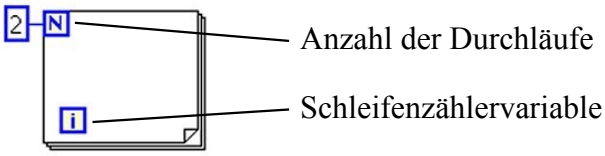



**Abb. 7-2 For-Schleife, die zwei mal wiederholt wird**

Wie bei der While-Schleife zeigt die Schleifenzählvariable die momentane Anzahl der durchlaufenen Schleifen an, angefangen bei Null. Wird eine Null an das Zählerterminal angelegt, so wird die Schleife nicht ausgeführt.

***Wichtig****:* Die For- und While-Schleife nehmen nur zu Beginn der Schleifeniteration die Werte, die an die Schleifen übergeben werden auf. Änderungen von Parametern außerhalb der Schleife werden während der Schleifenbearbeitung nicht aufgenommen! Alle Terminals, die sich in der Schleife befinden können dagegen während der Schleifenbearbeitung geändert werden.

***Automatische Indizierung***

Fähigkeit von Schleifenstrukturen, ein- oder ausgegebene Arrays an ihrem Rahmen aufzuteilen oder zusammenzusetzen. Abbildung 7-3 zeigt wie eine Zufallszahl dreimal in einer For-Schleife generiert wird. Einmal wird das Array mit den drei Zufallszahlen nach Beendigung der Schleife als Array ausgegeben (Indizierung aktiv) und zweimal als Skalar ausgegeben (Indizierung inaktiv). An „Skalar 1“ wird nach Beendigung der Schleife der letzte Wert übergeben, während an „Skalar 2“ stets der aktuell generierte Wert steht.

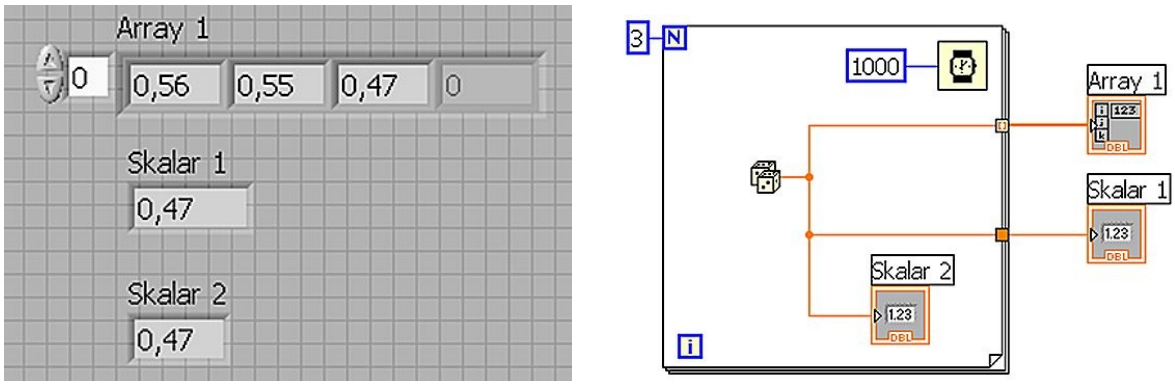



**Abb. 7-3 Generierung einer Zufallszahl in einer For-Schleife mit 1000 ms Wartzeit vor jeder Inkrementierung**

***Shift-Register***

Das Shift-Register ist eine lokale Variable, die sowohl in der For-Schleife als auch in der While-Schleife zur Verfügung steht. Diese Variable ermöglicht die Übergabe eines Wertes von einer Iterationsschleife zur nächsten. Ein Shift-Register wird eingefügt, durch Klicken mit der rechten Maustaste auf den Rahmen der For-Schleife oder While-Schleife. Ein Beispiel zeigt Abbildung 7-4. Die Register sind nicht immer paarweise vorhanden! Die Anzahl auf der linken kann größer sein als auf der rechten Seite. Eine beliebte Anwendung ist das *averaging*, bei dem z. B. die letzten fünf Messwerte fortlaufend gemittelt werden können.

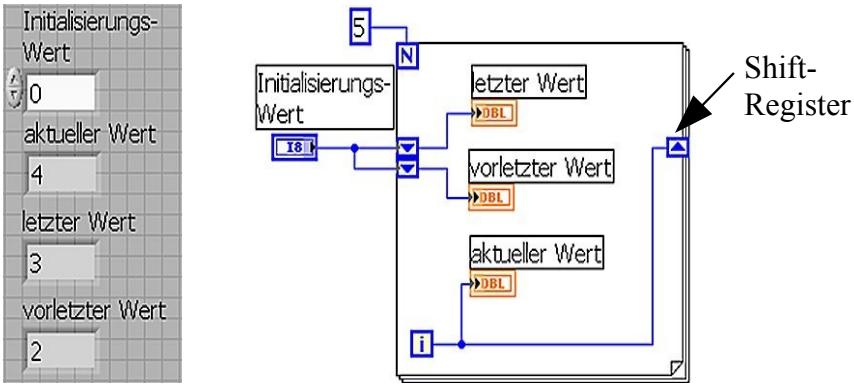
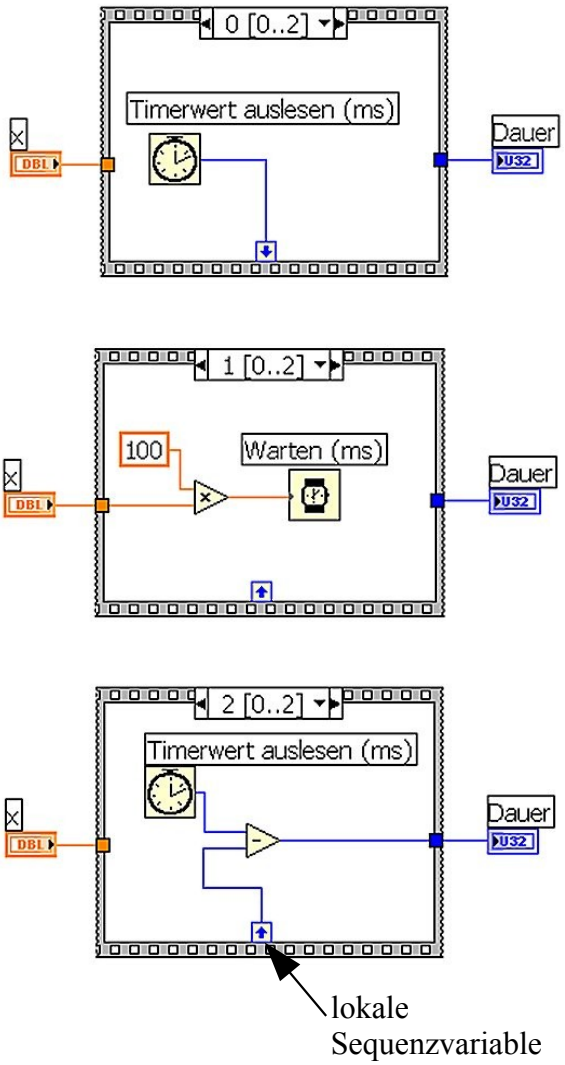



**Abb. 7-4 Shift-Register**

**Sequenz-Struktur**




Normalerweise entscheidet LabVIEW in welcher Reihenfolge die einzelnen Bestandteile des Programms abgearbeitet werden. Mit der Sequenz-Struktur steht dem Programmierer ein Hilfsmittel zur Verfügung die Reihenfolge des Programmablaufs - falls nötig - an den entscheidenden Stellen vorzugeben.

Zum Einfügen zusätzlicher Rahmen wird mit der rechten Maustaste auf den Sequenzrahmen geklickt und der entsprechende Befehl aus der Liste ausgewählt wird. In derselben Liste befindet sich auch ein Befehl zum einfügen von **lokalen Sequenzvariablen**. Wie Abbildung 7-5 zeigt können damit Variablen von einem in andere Rahmen übergeben werden.

**Abb. 7-5 Sequenzstruktur mit zwei aufeinanderfolgenden Rahmen. Im ersten Rahmen werden x und y miteinander multipliziert und das Ergebnis über eine „lokale Sequenzvariable“ an den zweiten Rahmen übergeben, in dem mit 100 multipliziert wird.**

Für den Einstieg bietet es sich an, sogenannte flache Sequenzstrukturen zu benutzen. Bei diesen ist der Datenfluss, die prinzipielle Funktionsweise, sowie die Ein-, Aus- und Übergabe von Variablen in die und aus der Sequenz sowie von einem Frame in den nächsten wesentlich einfacher zu verstehen.

*Case-Struktur*

Die Case-Struktur ermöglicht die Ausführung verschiedener Datenflüsse in Abhängigkeit vom Ergebnis einer Berechnung oder vom Zustand eines Eingangswerts. Als Eingangswert kann entweder ein boolesches Bedienelement oder eine numerische Zahl dienen.

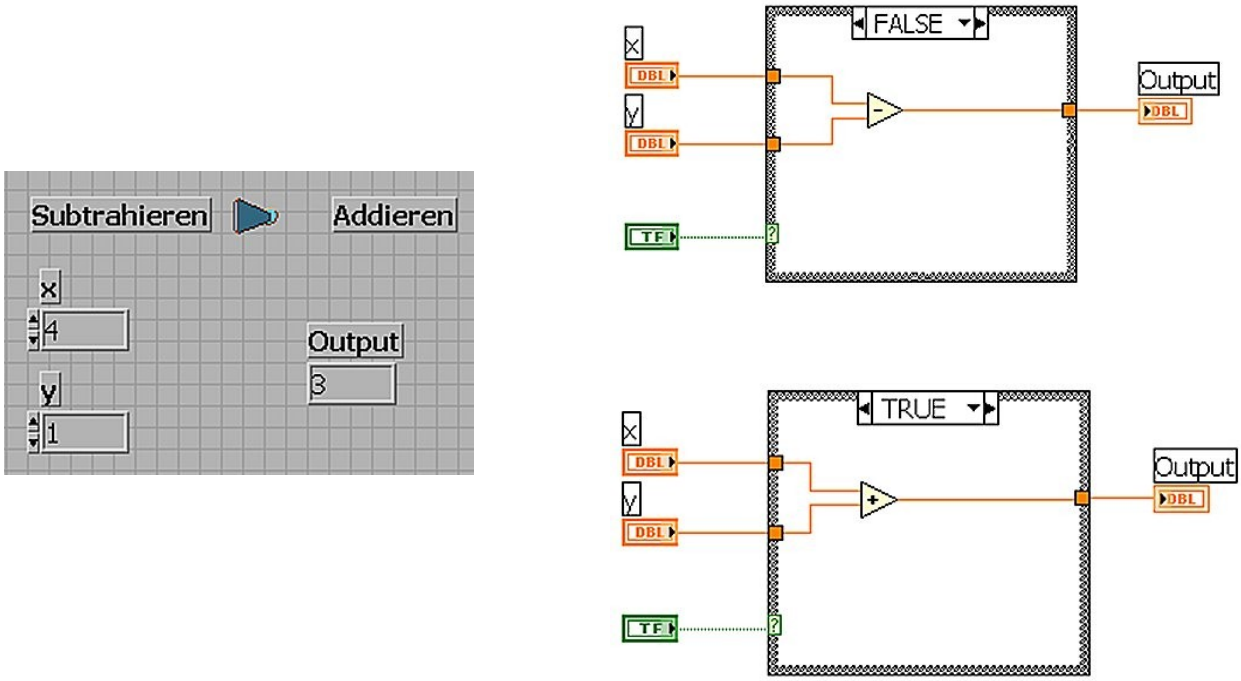



**Abb. 7-6 Case-Struktur, die je nach Schalterstellung eine Addition oder Subtraktion durchführt**

## 8. Graphen und Diagramme

Mit LabVIEW können die bei einer Messung erfassten oder bei einer Simulation berechneten Werte bequem in Form von Diagrammen und Graphen dargestellt werden. Diagramme stellen die aktuellen Werte sofort da, während bei Graphen zunächst die Gesamtheit aller Werte erfasst oder berechnet und dann erst dargestellt wird.

Um mehrere Kurven in einem gemeinsamen Diagramm darzustellen, müssen die Daten zunächst über den Bündel-Operator (aus dem Cluster-Menü) zusammengefasst werden (Abbildung 8-1).

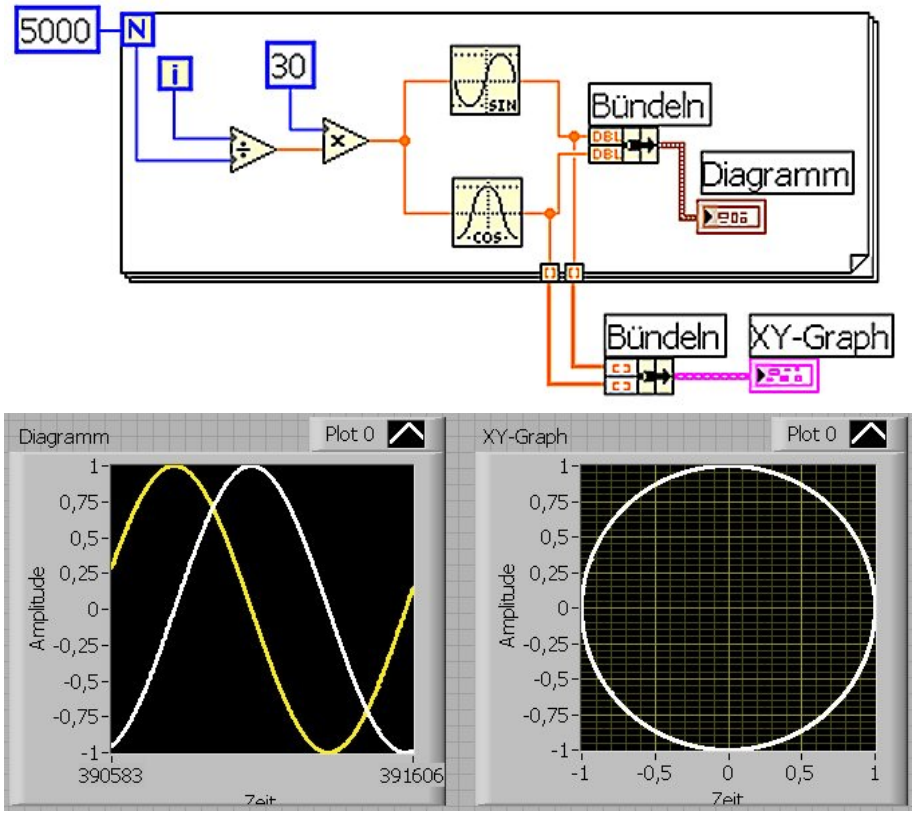



**Abb. 8-1 Gebündelte Darstellung von zwei Kurven in einem Diagramm und in einem X-Y-Graphen**

Um zwei oder mehrere Kurven in einem Graphen darzustellen wird der Operator Array erstellen verwendet (Abbildung 8-2).

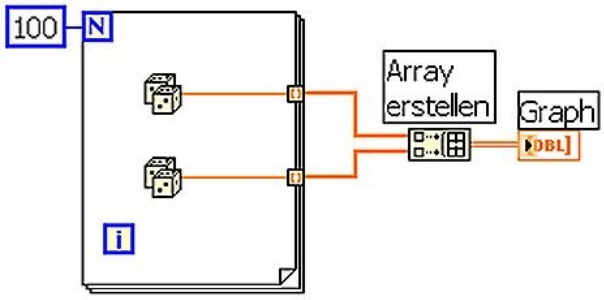



**Abb. 8-2 Operator Array erstellen**

Zur individuellen Gestaltung der Zeitachse können einem Graphen der Startwert der Zeitachse $X_0$ und der Abstand einer Zeiteinheit $\Delta X$ zusammen mit dem Datenarray zu einem Cluster gebündelt an den Graphen übergeben werden.

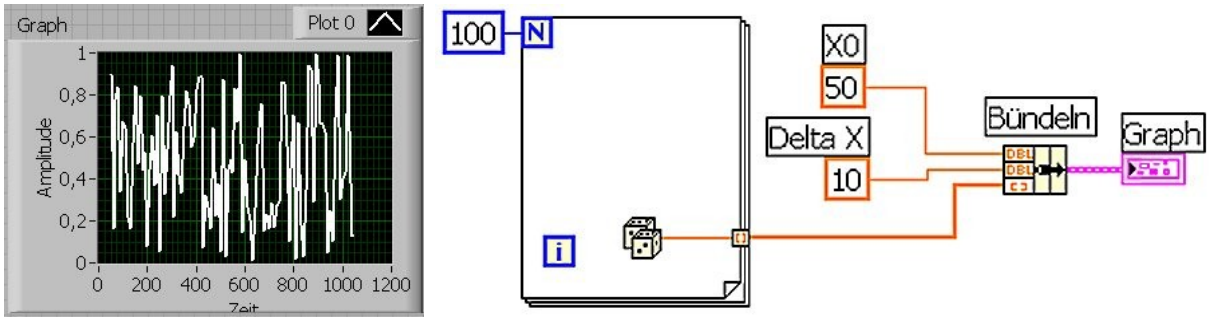



**Abb. 8-3 Graphen mit individuell vorgegebener Zeitachse. Zeitachse beginnt bei 50 und jeder Wert wird in einem Abstand von 10 in den Graphen eingetragen**

## 9. Arbeiten mit Strings (Zeichenketten)

Eine Zeichenkette ist eine Ansammlung von ASCII-Zeichen. Oft werden Strings für einfache Textbotschaften mit dem Ziel einer verständlicheren Kommunikation zwischen Anwender und Programm verwendet. Bei der Eingabe von Daten in ein Instrument kann es sinnvoll oder erforderlich sein, die Daten als Zeichenketten zu übertragen. Sie konvertieren dann diese Zeichenkette in Zahlen, um die Daten dann auf dem Instrument verarbeiten zu können. Beim Speichern von Daten auf einen Datenträger können auch Zeichenketten verwendet werden.
In vielen VI's mit Datei I/O konvertiert LabVIEW zuerst numerische Daten in Zeichenketten, bevor sie in einer Datei gespeichert werden.

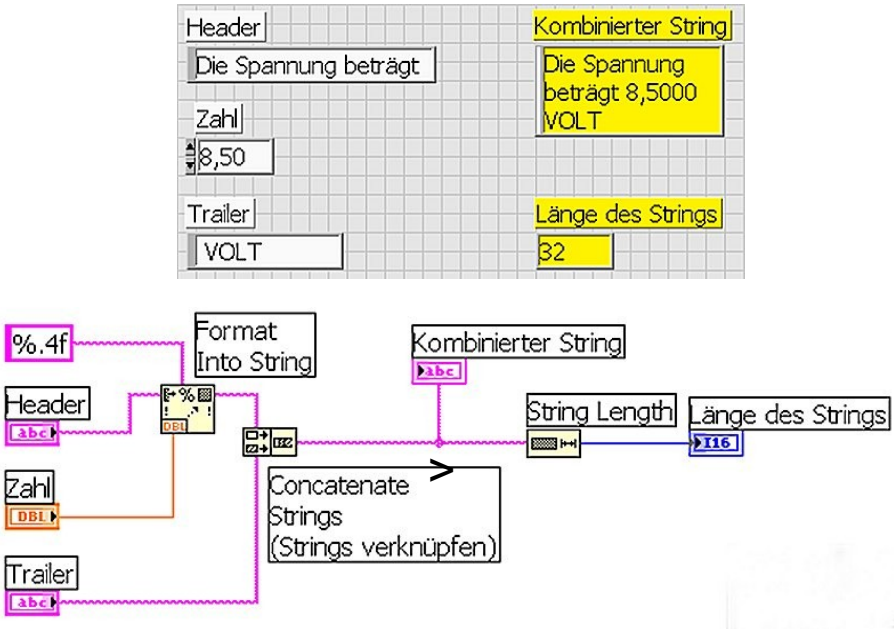



**Abb. 9-1 Beispiel einer Zeichenkettenbearbeitung mit Integration einer (beispielsweise gemessenen) Zahl.**

Das VI „in String formatieren“ (Abbildung 9-1) formt die eingegebene Zahl zu einem String um und hängt diesen an den Eingabe-String („Header“) an.
Durch die Eingabe von „%.4f“-Eingabe wird die Zahl mit vier Nachkommastellen angegeben. Im „String verknüpfen“ wird ein weiteres String (Trailer) angehängt.

**Formatierung**

Mit dem „%“-Zeichen beginnt die Festlegung der Formatierung. Die Zahl vor dem Punkt gibt die Feldbreite an, die Zahl nach dem Punkt gibt die Genauigkeit der Zahl, d. h. die Anzahl der Ziffern nach dem Komma, an. Ein *„f“* formatiert die Eingangszahl als Fließkommazahl in Dezimalbruchschreibweise, ein *„d“* als dezimale Ganzzahl und *„e“* als Fließkommazahl in wissenschaftlicher Notation. Die Formatierung wird nicht nur bei dem VI „in String formatieren“ benötigt. Auch bei der Daten-I/O kann mit der Formatierung festgelegt werden, in welchem Format die Daten abgespeichert oder ausgelesen werden sollen.

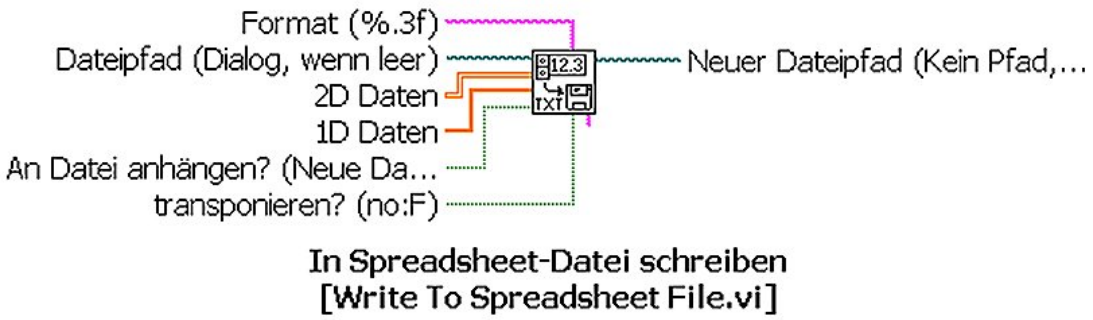
Das VI „Extract Numbers“ kann ebenfalls sehr nützlich sein. Hiermit können Werte aus einem String herausgefiltert und in eine Zahl zurückgewandelt werden.

## 10. Datei I/O (Input/Output)

Die Funktionen, die benötigt werden, um Dateien auf die Festplatte zu schreiben oder von dort zu lesen finden Sie in dem Menü *Daten I/O*. Am Eingang dieser Funktionen können Sie den gewünschten Pfad angeben. Wird dieser nicht angegeben, so fragt LabVIEW den aktuellen Pfad des bearbeiteten VIs ab.
LabVIEW kann die Daten in drei verschiedenen Formaten ausgeben:

- *Textdatei (ASCII)*: Textdatei-Daten haben den Vorteil, dass sie leicht portierbar sind. Praktisch jeder Rechner mit jedem Betriebssystem kann dieses Format lesen und eine Importierung und Bearbeitung ist mit vielen Programmen (Editor, Excel, Origin, etc.) möglich. Nachteil: Sie sind am speicherintensivsten und es erfordert bei entsprechend vielen Datenpunkten viel Prozessorzeit, die numerischen Daten in ein Textformat umzuwandeln.
- *Binärdateien* werden auch als Byte-Stream-Dateien bezeichnet. Sie enthalten normalerweise eine Abbildung der Daten, die Byte für Byte mit der Darstellung im Speicher übereinstimmt. Nachteil: Eine Binärdatei kann nicht ohne weiteres mit einem Editor oder irgendeinem anderen Programm (Excel, Origin, etc.) gelesen werden. Bei umsichtiger Anwendung kann dieser Datentyp auch von anderen Programmen gelesen werden. Beim Lesen von Binärdaten muss das Speicherformat bekannt sein. Binärdaten erfordern also eine genaue Einhaltung von Programmprotokollen. Vorteil: Diese Dateien bieten die effizienteste Speicherausnutzung und benötigt die wenigste Prozessorzeit. Schnell in der Verarbeitung (*kontinuierliche Plattenspeicherung* bzw. *Streaming*). Eignet sich für Anwendungen, in denen große Datenmengen in Echtzeit aufgezeichnet werden sollen.
- *Protokolldateien* sind spezielle Binärdateien, die nur von LabVIEW verwendet werden, in der Regel zur Speicherung der Daten vom Frontpanel oder allen anderen LabVIEW-Datentypen.

Vergleich: Um ein Array, bestehend aus 100 Ganzzahlen zu je acht Bit in einer Binärdatei zu speichern, werden etwa 100 Byte benötigt, für Textdateien können es über 400 Byte sein. Dies liegt daran, dass eine Ganzzahl mit acht Bit genau ein Byte belegt. Für dieselbe Zahl als Text dargestellt werden dazu bis zu drei oder vier Zeichen benötigt (ein Byte für jedes ASCII-Zeichen und ein Begrenzungszeichen zur Unterscheidung der einzelnen Zahlen).



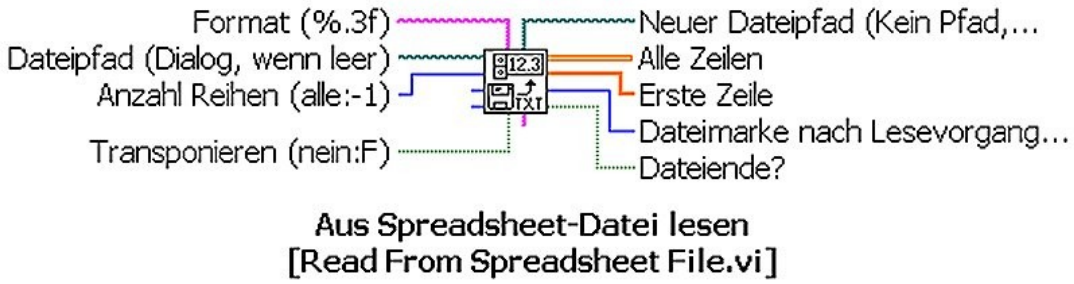




**Abb. 10-1 „In Spreadsheet-Datei schreiben“ konvertiert ein 2-D- oder 1-D-Array mit Zahlen einfacher Genauigkeit in eine Textzeichenkette und schreibt sie in eine neue Datei oder hängt sie an eine bestehende Datei an.**

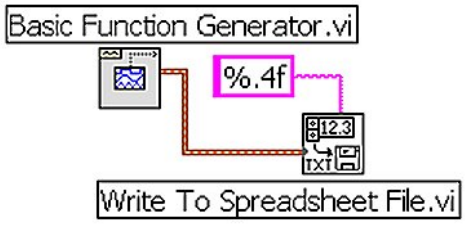



**Abb. 10-2 Die 1-D-Array-Daten des Funktionsgenerators werden als Fließkommazahlen mit einer Genauigkeit von vier Kommastellen in ein Spreadsheet geschrieben**

# Anhang

## A. Hilfe

Eine nützliche Hilfe ist die **Kontext-Hilfe**. Über *Hilfe >> Kontext-Hilfe anzeigen* (oder Strg + H) kann ein Fenster geöffnet werden, dass nützliche Information des Knotens, der Verbindungslinie oder des VI's anzeigt, über der sich gerade die Maus befindet.

Über *Hilfe >> Beispiele suchen...* können anschauliche Beispiele zu VI's, SubVI's oder Knoten gesucht werden.

## B. Fehlersuche und Debugging

### Fehlerliste

Wenn das Programm aufgrund von Syntaxfehlern nicht lauffähig ist, erscheint auf dem Startknopf ein gebrochener Pfeil.

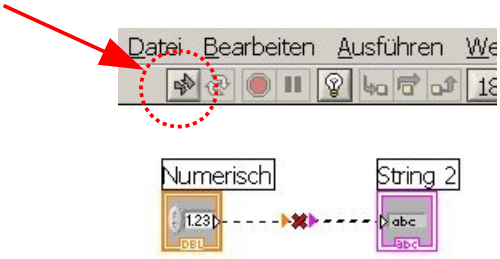



**Abb. 10-3 Fehlerhafte Syntax: hier wurde eine numerische Eingabe mit einer String-Anzeige verbunden. Die gebrochene Verbindung und der gebrochene Pfeil deuten einen Fehler an**

Ist der Fehler nicht sofort ersichtlich, kann der Fehler im Menü *Fenster >> Fehlerliste* (oder über die Tastenkombination Strg + L) angezeigt werden.

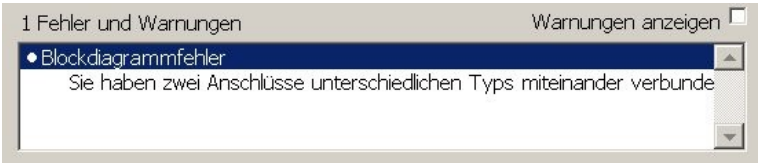
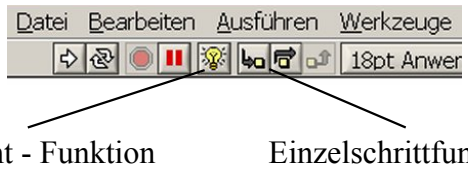



**Abb. 10-4 Fehlerliste für Fehler aus Abbildung 10-3**

### Einzelschrittmodus

Durch Klicken des Knopfes „Einzelschrittausführung starten" wird das Programm im Einzelschrittmodus gestartet. Durch wiederholtes Klicken kann das Programm schrittweise durchlaufen werden.




### Programmablauf Visualisieren „Highlight-Funktion"

Der Programmablauf lässt sich im Blockdiagramm mit Hilfe der Highlight-Funktion (aktiviert über die Glühbirne, die sich rechts von der Start-, Stopp- und Pausentaste befindet)

visualisieren. Während Daten von einem Knoten zum anderen laufen, wird die Bewegung durch wandernde Blasen angezeigt. Die berechneten Werte an den Knoten werden automatisch angezeigt. Bei aktiver Highlight-Funktion ist der Programmablauf erheblich verlangsamt.

**Breakpoints**

Ist das Programm umfangreich, kann es in der Highlight-Funktion oder im Einzelschrittmodus lange dauern bis die eigentlich zu untersuchende Stelle des Programms an der Reihe ist. Daher kann das Programm alternativ dazu in normaler Geschwindigkeit ablaufen, bis es an einem zuvor eingefügten Breakpoint angelangt ist und in den Pausenmodus wechselt. Anschliessend kann das Programm im Einzelschrittmodus oder in der Highlight-Funktion zur Untersuchung weitergeführt werden. Breakpoints lassen sich einfügen, indem in der Werkzeugpalette das Werkzeug „Breakpoints“ aktiviert ist. Der Kursor nimmt dann die Form eines Stoppschilds an. Ein Breakpoint kann durch Klicken an eine beliebige Stelle auf einem Verbindungsdraht oder einem Knoten platziert werden und wird durch einen roten Punkt auf der Linie oder eine rote Umrandung des Knotens gekennzeichnet.

**Zwischenergebnisse Visualisieren „Probe-Werkzeug“**

Mit der „Probe-Funktion“, die in der Werkzeugpalette aktiviert werden kann, können die Werte an beliebigen Stellen im Programm abgefragt werden. Dazu wird mit dem aktiven Werkzeug einfach auf die Verbindung geklickt, die abgefragt werden soll. Die Ablaufgeschwindigkeit des Programms wird nicht (merklich) beeinflusst.

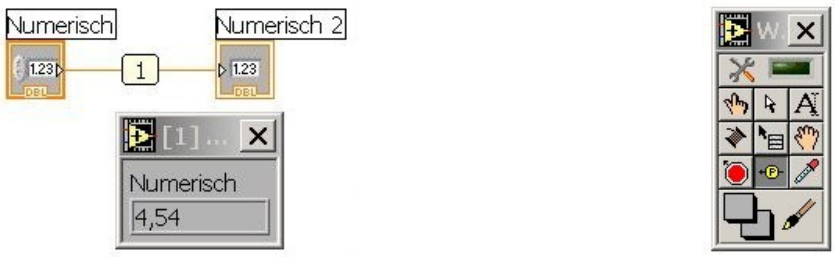


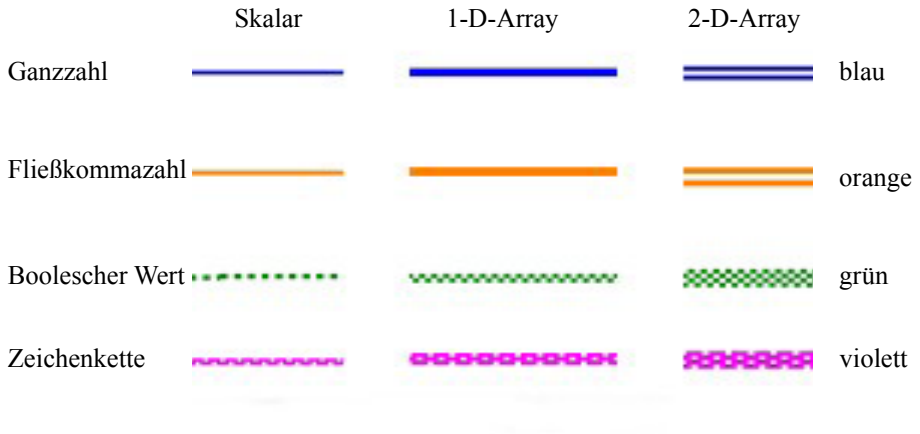

**Abb. 10-5 Links: Entnahme der Probe auf der Verbindung zwischen zwei Terminals. Rechts: Aktiviertes „Probe-Werkzeug“ in der Werkzeugpalette.**

## C. Datentypen und ihre Speicherbelegung

| Darstellung | Abkürzung | Größe (in Bytes) | Wertebereich |
|---|---|---|---|
| Byte | I8 | 1 | -128...127 |
| Vorzeichenloses Byte | U8 | 1 | 0...255 |
| Word | I16 | 2 | -32768...32767 |
| Vorzeichenloses Word | U16 | 2 | 0...65535 |
| Long | I32 | 4 | -2147483648…2147483647 |
| Vorzeichenloses Long | U32 | 4 | 0…4294967295 |
| Single Precision | SGL | 4 | |
| Double Precision | DBL | 8 | |
| Extended Precision | EXT | 10(a) / 12(b) / 16(c) | |
| Complex single | CSG | 8 | |
| Complex double | CDB | 16 | |
| Complex extended | CXT | 12(a) / 24(b) / 32(c) | |

(a) Windows (b) Mac (c) Sun und HP-UX

Die Verbindungen im Blockdiagramm haben entsprechend des vorliegenden Datentyps folgende graphische Darstellung:




## D. Nützliche Tastenkombinationen

| Tastenkombination | Befehl |
|---|---|
| Strg + B | Alle fehlerhaften Verbindungen aus dem Diagramm entfernen (**B**roken wire) |
| Strg + E | Umschalten zwischen Frontpanel und Blockdiagramm |
| Strg + F | Ein LabVIEW Objekt oder einen Text finden (**F**ind) |
| Strg + H | Das Hilfefenster anzeigen/verbergen (**H**elp) |
| Strg + N | Neues VI (**N**ew) |
| Strg + Q | Aktuelle Sitzung beenden (**Q**uit) |
| Strg + R | Aktuelles VI ausführen (**R**un) |
| Strg + T | Gleichmäßige Bildschirmaufteilung von Frontpanel und Blockdiagramm erzeugen (**T**oggle) |
| Strg + W | Aktuelles VI schließen |
| Strg + . | Aktuelles VI anhalten |
| Strg + ? | Hilfe öffnen |
| Strg + / | Gesamter Bildschirm |