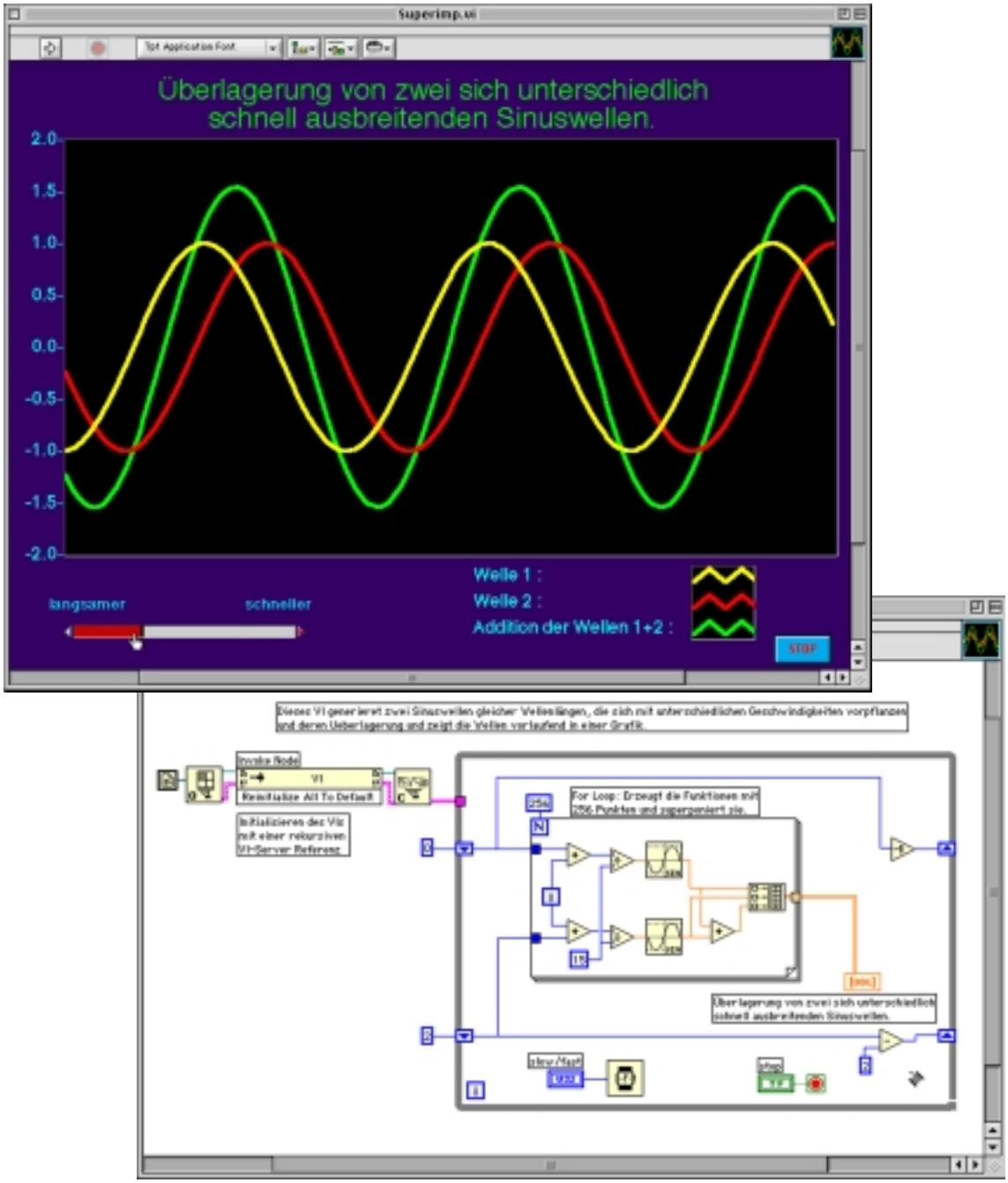
LabVIEW™

Einstieg in die grafische Programmiersprache zur
Erfassung und Analyse von Messdaten

Dezember 2001



Urs Lauterburg El. Ing. HTL Physikalisches Institut Universität Bern

Inhaltsverzeichnis

Kapitel 1

Willkommen in LabVIEW

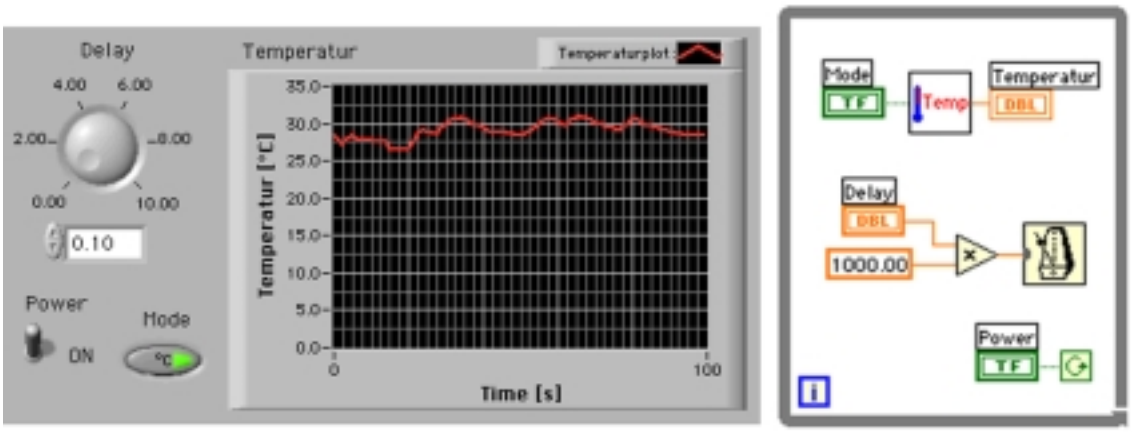
Dieses Kapitel gibt eine Übersicht von LabVIEW und seinen Möglichkeiten:

Themen im 1. Kapitel :

- LabVIEW
- Virtuelles Instrument (VI)
- em front panel
- *Block Diagram*
- *icon*
- *connector*
- Palette
- Hierarchie
- Modulares Programmieren

1.1 LabVIEW — die Kur für Textcode ?

LabVIEW (Abkürzung für **Lab**oratory **V**irtual **I**nstrument **E**ngineering **W**orkbench) ist ein mächtiges, vielseitiges Software-Messwerkzeug und Analysepaket für PCs und Workstations, (Apple Macintosh, Microsoft Windows, Sun SPARC und HP 9000/700 unter HP-UX).
LabVIEW ist eine Entwicklungsumgebung wie professionelle C- oder BASIC-Systeme. Mit einem grossen Unterschied allerdings: Herkömmliche Programmiersysteme verlangen die Anweisungen in Textform, LabVIEW aber besitzt seine eigene grafische Programmiersprache "G". Programme werden grafisch in Form von Flussdiagrammen erstellt, in LabVIEW spricht man von "Blockdiagrammen". Solche Diagramme eliminieren viele syntaktische Details konventioneller Hochsprachen.
Figur 1.1 auf Seite 4 zeigt eine einfache Benutzeroberfläche (*user interface*) und das grafische Programm (*LabVIEW-source code*) dahinter:



Figur 1.1: Benutzeroberfläche / grafisches Programm

LabVIEW benutzt Therme, Symbole und Konzepte die dem Ingenieur und Wissenschafter vertraut sind. LabVIEW definiert im Gegensatz zu den Textsprachen ein Programm mit grafischen Symbolen.
LabVIEW ist grundsätzlich ohne Programmiervorkenntnisse lernbar; Programmiererfahrung kann jedoch für das Studium von LabVIEW nützlich sein.

1.2 Was kann LabVIEW für uns tun?

LabVIEW kümmert sich für uns um viele Details. Es gibt umfangreiche Bibliotheken mit Funktionen und Subroutinen welche das Programmieren erleichtern. LabVIEW besitzt eine Menge applikationsspezifische *libraries* zur Messdatenerfassung, für GPIB, für Geräte welche über die serielle Schnittstelle kommunizieren, zur Datenanalyse, Datenpräsentation und Datenspeicherung. Die Analysebibliotheken stellen zahlreiche Funktionen zur Messdatenverarbeitung

wie Filter, Statistiken, Regressionen, Lineare Algebra und Matrizen-Arithmetik zur Verfügung.
LabVIEW bietet Programmentwicklungswerkzeuge zur Fehlerbeseitigung, wie das Setzen von *breakpoints*, das Schritt für Schritt Abarbeiten eines Programmes, sowie die Möglichkeit zur Datenflussanimation in den Blockdiagrammen.
Auf Grund der grafischen Struktur ist LabVIEW ein vielseitiges Präsentationspaket. Die Datenausgabe kann in gewünschter Form gestaltet werden. Datenschreiber, Grafiken und benutzerdefinierte Darstellungen sind nur ein kleiner Teil der Möglichkeiten. Die Messdatenerfassung, die Analyse- und Darstellungswerkzeuge machen LabVIEW zu einer umfassenden und mächtigen Programmierumgebung . Alle Problemlösungsmöglichkeiten einer konventionellen Programmiersprache sind in Form von LabVIEWs virtuellen Instrumenten möglich.

1.3 Wie funktioniert LabVIEW ?

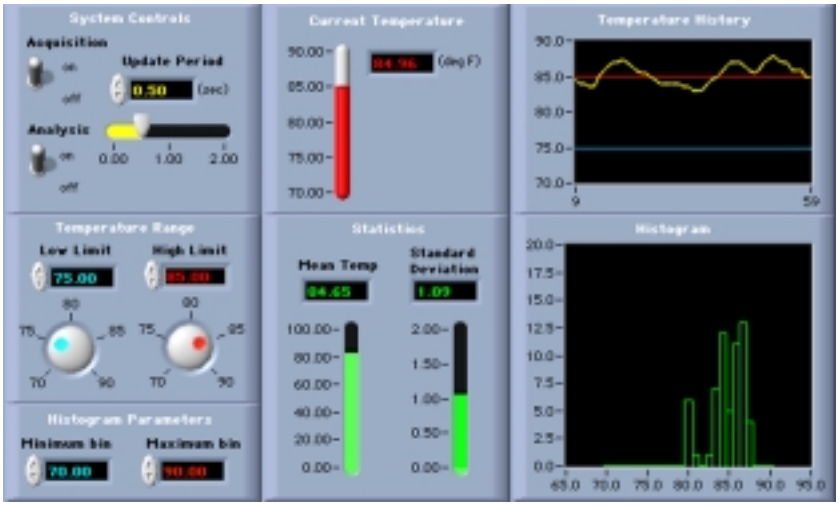
LabVIEW Programme werden virtuelle Instrumente *(Virtual Instruments)*, kurz VI genannt, da ihre Erscheinung und Funktion Hardwaregeräten ähnlich sind. Diese VIs funktionieren je nachdem wie die Hauptprogramme, Funktionen oder Subroutinen in konventionellen Programmiersprachen wie C, Pascal oder BASIC. VIs besitzen ein interaktives Benutzeroberflächen- und ein Quellcodeäquivalent welche Daten beliebig austauschen können. Ein VI besteht aus drei Hauptteilen:

- Das *front panel*: Dies ist die interaktive Benutzeroberfläche oder Variablenschnittstelle eines VIs. Es simuliert die Frontplatte eines Geräts mit Knöpfen, Schalter, Grafiken und vielen andern *controls* (Benutzereingaben) und *indicators* (Programmausgaben). Der Benutzer gibt die Daten über die Tastatur oder mit der Maus ein und sieht das Resultat nach der Verarbeitung durch das Programm auf dem Bildschirm. Ein Beispiel eines *front panel* wird in Figur 1.2 auf Seite 6 gezeigt.

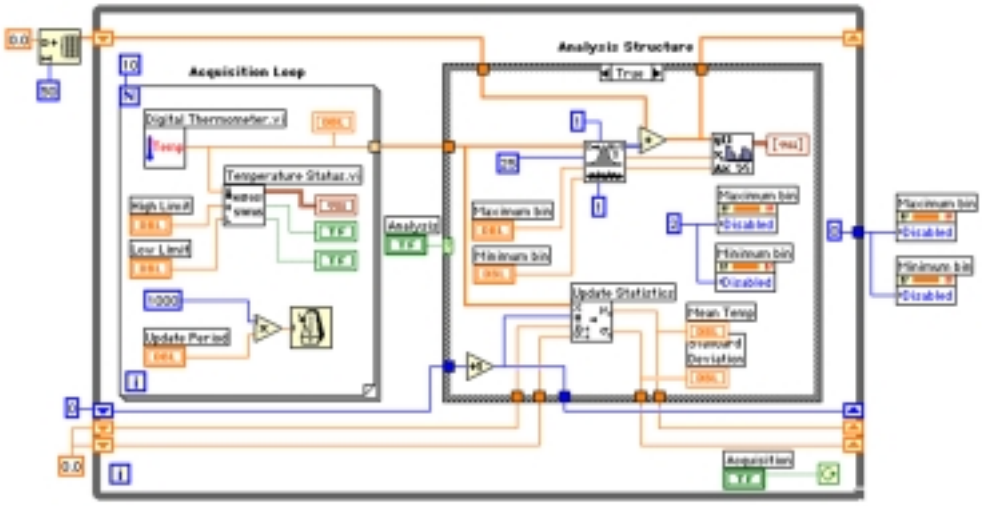
- Das Blockdiagramm, gezeigt in Figur 1.3 auf Seite 6, ist der Quellcode des VIs, konstruiert mit LabVIEWs grafischer Programmiersprache G. Dieses Blockdiagramm, obschon es sehr bildhaft aussieht, ist das lauffähige Programm. Die Teile des Blockdiagrammes, die Symbole (*icons*), stellen untergeordnete VIs dar, von LabVIEW zur Verfügung gestellte Funktionsmodule und Programmstrukturen. Die Blöcke werden mit Drähten *(wires)* zweckmässig verbunden, sie stellen die Variablen dar, welche den Datenfluss im Blockdiagramm und somit den Ablauf des Programmes bestimmen.

- Das Symbol (*icon*) und der Verbinder (*connector*) eines VIs erlauben den Transfer der Daten zu einem anderen VI. Das Symbol repräsentiert ein VI im Blockdiagramm eines übergeordneten VIs. Der Verbinder definiert die Ein- und Ausgänge des VIs. VIs können hierarchisch und modular in

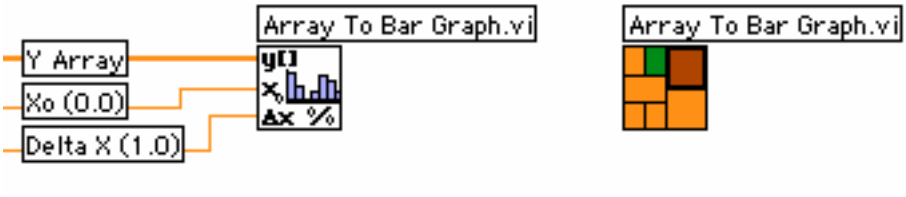
Baumstrukturen aufgebaut werden, sie können entweder ein *top-level*-Programm oder ein beliebiges Unterprogramm sein. Ein VI in einem höheren VI wird, in Analogie zur Subroutine, "subVI" genannt.



Figur 1.2: *front panel*



Figur 1.3: *block diagram*



Figur 1.4: *icon / connector*

Mit diesen Möglichkeiten unterstützt LabVIEW das modulare Programmieren. Zuerst wird eine Applikation in eine Reihe einfacher Module aufgeteilt. Danach baut man die funktionsfähigen Teilbausteine zusammen und kombiniert sie im *top-level*-Diagramm.
LabVIEW fördert eine modulare Programmierstruktur, da jedes subVI autonom lauffähig ist und somit leicht individuell getestet werden kann. Die *low level* subVIs können von Programmen an unterschiedlichen Stellen aufgerufen werden.

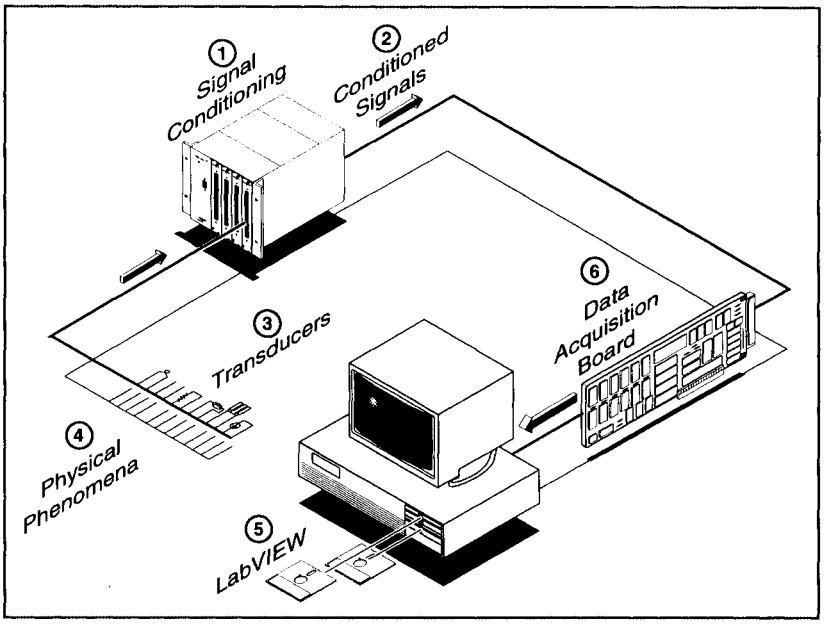
Kapitel 2

LabVIEW Messdatenerfassung

Dieses Kapitel zeigt die grundlegenden Möglichkeiten zur Messdatenerfassung mit LabVIEW:

Themen im 2. Kapitel :

- Messdatenerfassung mit Steckkarten (**D**ata **A**c**Q**uisition DAQ)
- GPIB
- IEEE 488
- Serielle Schnittstelle



Figur 2.1: System zur Messdatenerfassung

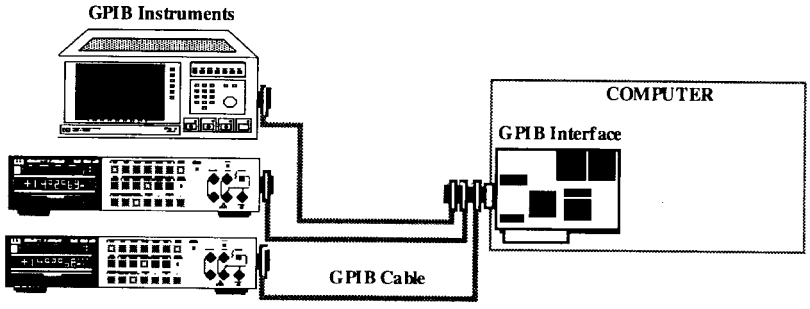
2.1 Was ist digitale Messdatenerfassung ?

Unter Datenerfassung verstehen wir den Transfer von physikalischen Signalen in einen Rechner zum Zweck der Verarbeitung, Analyse, Speicherung oder Manipulation. Die Figur 2.1 auf Seite 10 zeigt die Bausteine eines rechnergesteuerten Messsystems (engl. *DAQ system (***D***ata* **A**c**Q***uisition)*). Sensoren wandeln physikalische Parameter in elektrische Signale. In einem einfachen DAQ-System steuert LabVIEW eine DAQ-Karte im PC. Eine solche Karte kann z.B. Analogsignale mittels A/D-Wandler abtasten, Analogsignale mittels D/A-Wandler generieren, digitale Signale einlesen und ausgeben oder, durch Steuern von integrierten digitalen Zählerbausteinen, Frequenzen messen und Pulsfolgen erzeugen. Bei Messungen werden die Spannungen direkt der Steckkarte zugeführt, welche die Messwerte zur Verarbeitung in digitalisierter Form dem Speicher des Rechners übergibt. LabVIEW kommuniziert mit solchen Messkarten über den hochoptimierten NI-DAQ-Treiber, der sämtliche Funktionen der NI-Messkartenplatte bedienen kann.

Mit Signalaufbereitungsmodulen können die von Sensoren erzeugten Signale aufbereitet werden. Zum Beispiel wird man bei der Messung einer Hochspannung eine galvanische Trennung zum Rechner vorsehen. Die Module können dabei z.B. verstärken, linearisieren, filtern, isolieren usw. Nicht jede Anwendung verlangt ein speziell aufbereitetes Signal; vernünftige Pegel können direkt der Karte zugeführt werden.

2.2 Was ist GPIB ?

In den späten 60er-Jahren entwickelte Hewlett Packard den **G***eneral* **P***urpose* **I***nterface* **B***us*, um die Kommunikation zwischen Computer und Messinstrumenten zu erleichtern. Das *Institute of Electrical and Electronic Engineers* (IEEE) hat 1975 den GPIB-Bus und das Protokoll als IEEE 488 standardisiert. Der ursprüngliche Zweck des GPIB-Bus war es, Test- und Messinstrumente mit einem Rechner zu steuern. GPIB expandierte aber auch in andere Gebiete wie die Computer--zu-Computer-Kommunikation und die Steuerung von Scanner und Filmaufnahmegeräten im Druckgewerbe.
GPIB ist ein digitaler 24-Leiter-Bus, zusammengesetzt aus 8 Datenleitungen, 5 Bussteuerleitungen (ATN, EOI, IFC, REN und SRQ), 3 *handshake*-Leitungen und 8 Erdleitungen. GPIB benutzt ein 8bit-paralleles, asynchron-serielles Byte-Übertragungsschema. Anders ausgedrückt: ganze Bytes werden sequentiell per *handshake* über den Bus übertragen. Die Geschwindigkeit wird vom langsamsten kommunizierenden Gerät bestimmt. Da GPIB einzelne Bytes parallel überträgt, werden sie oft als ASCII-Zeichenketten kodiert. Ein Computer ist erst GPIB-fähig, wenn eine GPIB-Karte installiert und die notwendige Treibersoftware installiert ist.
Mehrere Geräte und Computer können denselben GPIB-Bus verwenden. An jedem Gerät und jeder Computerkarte muss eine eigene GPIB-Adresse zwischen 0 und 30 eingestellt sein, um sie als Sender oder Empfänger von Daten zu identifizieren. GPIB besitzt einen *Controller*, gewöhnlich ist dies der Computer, welcher den Bus steuert. Seiner GPIB-Karte wird normalerweise die Adresse 0 zugewiesen. Um Instrumentenbefehle zu übermitteln, muss der *Controller* einen sog. *Talker* (Sprecher) und einen oder mehrere sog. *Listener* (Hörer) anwählen. Mit den LabVIEW GPIB-VIs können Geräte am GPIB-Bus adressiert und gesteuert werden. Figur 2.2 auf Seite 11 illustriert den Aufbau eines typischen GPIB-Systems.

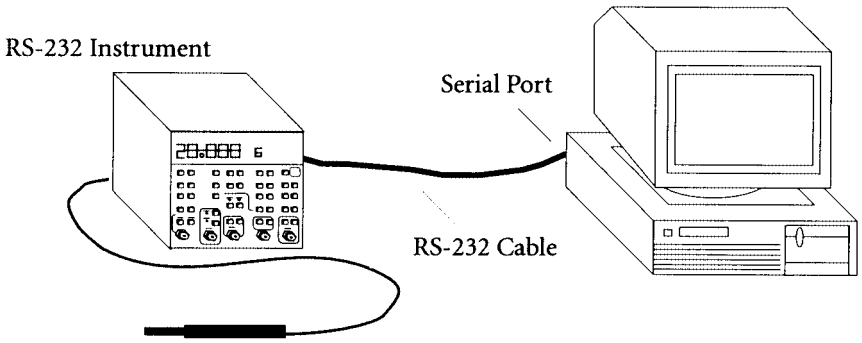


Figur 2.2: GPIB-System

GPIB-Messdatentransfer ist grundsätzlich verschieden vom Messen mit einer DAQ-Karte. Bei einer DAQ-Karte wird das zu messende Signal direkt auf der Karte digitalisiert, bei GPIB wird die Messung von einem externen Gerät ausgeführt, die Daten und Steuersignale werden kodiert übermittelt.

2.3 Die serielle Schnittstelle

Serielle Kommunikation ist eine andere populäre und kostengünstige Methode, um Daten zwischen Computer und Messgeräten oder Sensoren auszutauschen. Dazu wird die in vielen Computern fest eingebaute serielle Schnittstelle (z.B. RS-232 oder RS-422) benutzt. Bei der seriellen Kommunikation werden Datenbits vom Sender einzeln nacheinander über eine Leitung zum Empfänger übertragen. Die Methode wird benutzt, wenn die Datentransferraten relativ klein sind und längere Übermittlungsdistanzen vorliegen. Serielle Kommunikation ist langsamer und etwas weniger zuverlässig als GPIB, man braucht jedoch keine spezielle Steuerkarte und kein GPIB-fähiges Gerät. Figur 2.3 auf Seite 12 zeigt ein Messsystem mit serieller Datenübertragung .

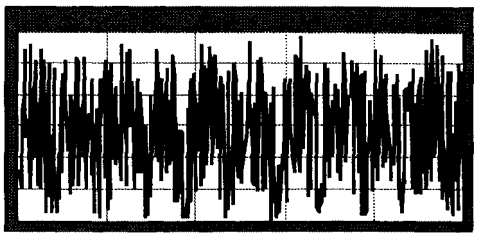


Figur 2.3: Serielle Kommunikation zur Gerätesteuerung

Die serielle Kommunikation ist denkbar einfach und praktisch, da Daten ohne zusätzliche Steckkarten gesendet und empfangen werden können. (Viele GPIB-Geräte besitzen neben dem GPIB-Anschluss zusätzlich einen seriellen Anschluss). Viele moderne Sensoren verfügen über einen seriellen Port und können direkt am PC angeschlossen werden. Heute sind auch Zusatzeinheiten zu kaufen, welche über mehrere serielle Anschlüsse verfügen. Grundsätzlich kann aber, im Gegensatz zu GPIB, nur mit einem Gerät gleichzeitig kommuniziert werden.

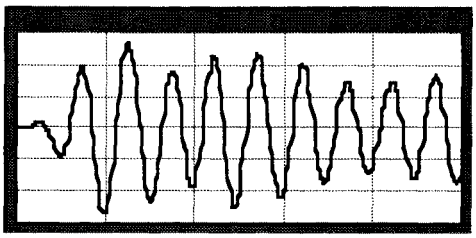
2.4 Wozu dient die Datenanalyse ?

Wenn die Messdaten einmal digital im Rechnerspeicher vorliegen, können sie analysiert und aufbereitet werden. Auf Messkarten integrierte Signalprozessoren können zur Vorverarbeitung von Messdaten in Echtzeit unterstützend eingesetzt werden. Weitere Analysen, Visualisierung und das Abspeichern auf geeignete Datenträger können mit LabVIEW erreicht werden. Einige mögliche Anwendungen sind biomedizinische Analysen, Spracherkennung oder digitale Audio- und Bildverarbeitung.

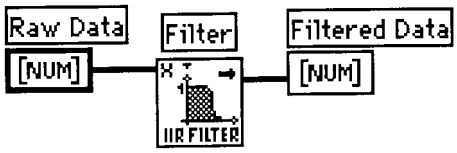


Figur 2.4: Verrauschtes Messignal

Der grosse Vorteil der LabVIEW-Programmierumgebung gegenüber einem vorgegebenen, alleinstehenden Analysegerät liegt in der individuellen Verarbeitung der Messdaten wie sie beispielsweise von einer DAQ-Karte oder einem GPIB-Gerät registriert wurden. Die gemessenen Signale zeigen nicht immer die brauchbare Information, wie in Figur 2.4 auf Seite 13 dargestellt. Oft muss das Signal transformiert, von Rauschanteilen befreit oder für Temperatureffekte kompensiert werden.
Die LabVIEW-Blockdiagramm-Methode und die vielseitigen Analysebibliotheken vereinfachen das Entwickeln von Analyseapplikationen. Z. B. kann aus dem verrauschten Signal in Figur 2.4 mittels eines einzigen Filterblocks (gezeigt in Figur 2.6) das ursprüngliche Signal (siehe Figur 2.5) zurückgewonnen werden.



Figur 2.5: Digital gefiltertes Signal



Figur 2.6: Filterblock im Blockdiagramm

Da LabVIEW viele gebräuchliche Analyseoperationen als VIs in Bibliotheken zur Verfügung stellt, können sie als Blöcke im Diagramm aneinandergereiht werden und dem Programmierer bleibt genug Zeit um sich dem spezifischen Problem des konkreten Messproblems zu widmen.

Bei den LabVIEW Analyse-VIs handelt es sich um C-Routinen, die sehr effizient arbeiten und auf ganze Datensätze angewendet werden können. Unter anderem werden folgende wichtige Bereiche durch LabVIEW Funktionen abgedeckt:

- Generieren von Mustern (*Pattern generation)*
- Digitale Signalaufbereitung (*Digital signal processing)*
- Digitale Filter (*Digital filtering)*
- Fensterfilter (*Smoothing windows)*
- Statistische Analysen (*Statistical analysis)*
- Regressionen (*Curve fitting)*
- Lineare Algebra (*Linear algebra)*
- Numerische Analysen (*Numerical analysis)*
- Messdatenanalysen (*Measurement-based analysis)*
- Datenübermittlung (*Data transmission)*

Kapitel 3

Die LabVIEW Umgebung

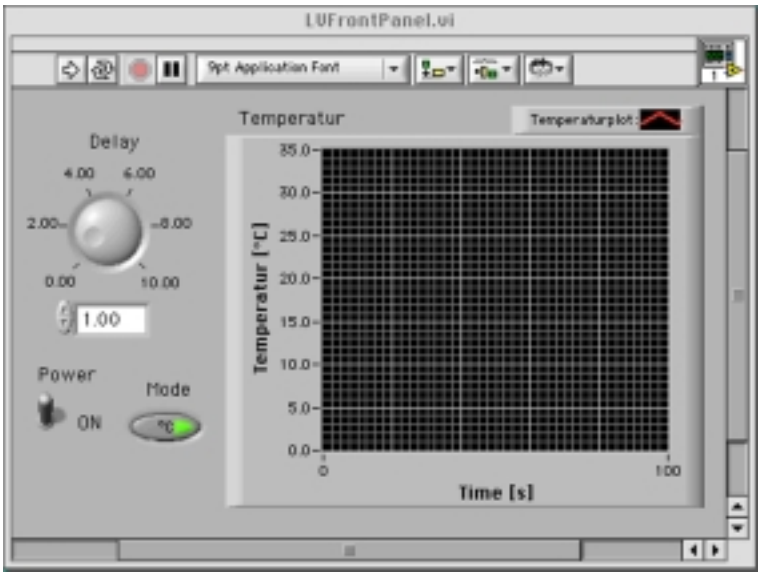
Dieses Kapitel beschreibt die LabVIEW-Umgebung, wie die drei Bereiche *front panel, block diagram* und *icon/connector* zusammenarbeiten, wie die Menüs aufgebaut sind und was die beiden Betriebsmodi *run mode* und *edit mode* bedeuten:

Themen im 3. Kapitel :

- *control*
- *indicator*
- *wire*
- *subVI*
- *terminal*
- *node*
- *dataflow*
- *icon*
- *connector*
- *pop-up Menüs*
- *run mode*
- *edit mode*
- *help window*

3.1 *Front Panel*

Das *front panel* ist das Fenster, durch welches der Benutzer mit dem Programm kommuniziert. Beim Ausführen eines Hauptprogramms sollte das *front panel* dem Benutzer zugänglich sein, so dass Eingangsparameter während dem Ablauf des Programms eingegeben und verändert werden können. Auch die Ausgabeparameter des Programms werden im *front panel* angezeigt. Die Figur 3.1 auf Seite 16 zeigt ein einfaches LabVIEW *front panel*.



Figur 3.1: LabVIEW *front panel*

Controls und *indicators*

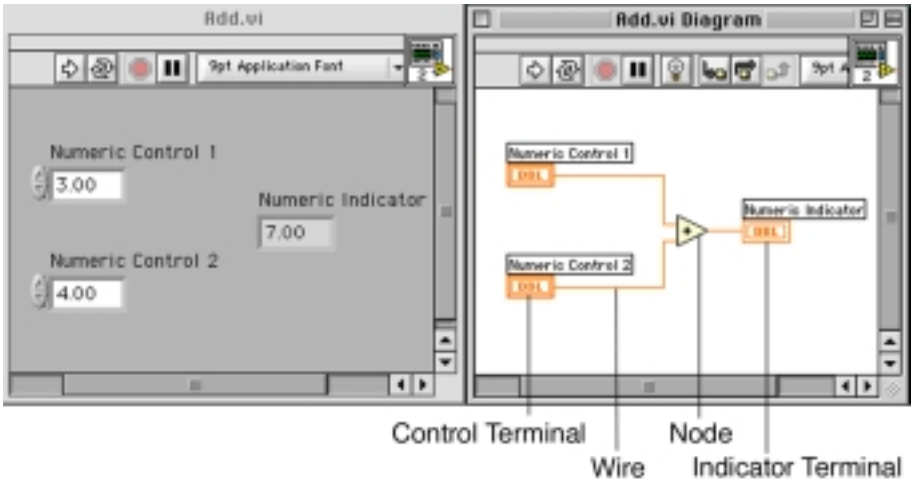
Das *front panel* kombiniert die Eingangsparameter (*controls*) und die Ausgabeparameter (*indicators*). *Controls* simulieren typische Bedienungselemente von Geräten wie Knöpfe und Schalter usw., sie übergeben die Eingaben dem Blockdiagramm. *Indicators* repräsentieren Ausgabeelemente, welche Messdaten anzeigen. Einfach gesagt:

Controls = Eingaben
Indicators = Ausgaben

Controls und *indicators* werden mit Hilfe der Maus aus der *Control*-Palette gezogen und auf dem *front panel* an beliebiger Stelle positioniert. Danach können die Erscheinungsformen wie Grösse, Farbe, Verhalten usw. der platzierten Objekte auf einfache Art festgelegt werden.

3.2 *Block diagram*

Die *block diagram*-Ebene umfasst das grafische Programm (*source code*) eines LabVIEW VIs. Das Blockdiagramm wird konstruiert durch Verbinden der Objektblöcke mit Drähten (*wires*). Die vom Programmierer gezeichnete bildliche Darstellung, die gewisse Ähnlichkeiten mit einem Flussdiagramm aufweist, ist das LabVIEW-Programm. Das einfache Blockdiagramm in Figur 3.2 auf Seite 17 rechnet die Summe zweier Eingaben.



Figur 3.2: *Front panel* Ein- und Ausgaben und dazugehörendes *block diagram*

Im Folgenden werden die Komponenten eines Blockdiagramms, nämlich die *terminals, nodes* und *wires* beschrieben.

Terminals

Nach Platzieren eines *controls* oder *indicators* auf dem *front panel* erscheint der entsprechende *terminal* auf dem Blockdiagramm. Der *terminal* kann auf dem Blockdiagramm nicht gelöscht werden, dies kann nur durch Löschen des entsprechenden *controls* auf dem *front panel* erreicht werden.

Control terminals zeigen dicke, *indicator terminals* dagegen dünne Umrandungen. Sie dürfen nicht verwechselt werden, da sie funktional NICHT identisch sind.

Terminals sind Ein- und Ausgabeschnittstellen. Die **Add**-Funktion auf dem Blockdiagramm in Figur 3.2 besitzt z. B. drei *terminals*. Die Variablen folgen den Drähten, sie werden der **Add**-Funktion über ihre *terminals* zugeführt. Nachdem die **Add**-Funktion ausgeführt wurde, erscheint das Resultat am Ausgangs*terminal*. Der Wert fliesst zum *numeric indicator terminal*, welcher danach im *front panel indicator* angezeigt wird.

Nodes

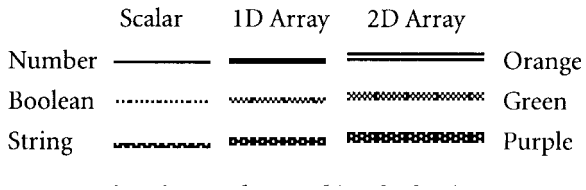
Ein *node* ist ein Programmelement, analog zu Befehlen, Operatoren, Funktionen und Subroutinen in anderen Programmiersprachen. Die LabVIEW **Add**- oder **Subtract**-Funktionen gehören zu einer Art von *nodes*. Strukturen (*structures*) sind eine weitere Art von *nodes*. Strukturen bearbeiten Programmteile repetierend mit Abbruchbedingungen, ähnlich wie es Programmschleifen traditioneller Programmiersprachen tun. LabVIEW besitzt spezielle *nodes* wie z. B. den Formel-*node*, welcher Formeln in Symbolschreibweise auswerten kann.

Wires

Wires sind Datenverbindungen zwischen Start- und Ziel*terminals*. Zwei Start*terminals* oder zwei Ziel*terminals* lassen sich nicht mit einem Draht (*wire*) verbinden, dagegen kann ein Start*terminal* mit mehreren Ziel*terminals* verbunden werden.

Obenstehendes Prinzip beschreibt den Unterschied zwischen *controls* und *indicators* : *Controls* sind Datenquellen, *indicators* sind Datensenken.

Ein *wire* besitzt je nach Datentyp unterschiedliche Farben und Formen. Das Blockdiagramm in Figur 3.2 zeigt die *wires* als numerische Skalarwerte. Die Figur 3.3 auf Seite 18 stellt einige *wires* und ihre entsprechenden Datentypen dar.



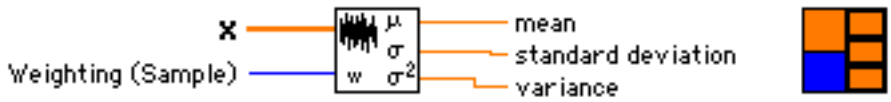
Figur 3.3: Einige *wires* und ihre Datentypen

Datenfluss-Programmierung

Im Gegensatz zu herkömmlichen Textsprachen ist der Ablauf eines LabVIEW-Programmes nicht durch die Befehlssequenz bestimmt. Der Ablauf richtet sich nach dem Datenflussprinzip. Ein *node* wird nur dann abgearbeitet, wenn an all seinen Eingängen Daten verfügbar sind. Der *node* kann seine Ausgangsvariablen erst nach Ausführen seiner Funktion weitergeben. Das Datenflussprinzip unterscheidet LabVIEW stark von anderen Hochsprachen mit ihrer zeilenorientierten, sequentiellen Natur. Textsprachen sind anweisungsorientiert, LabVIEW dagegen datenorientiert. Dieser Aspekt kann insbesondere erfahrenen Textsprachenprogrammierern Mühe bereiten und verlangt ein entsprechendes Umdenken.

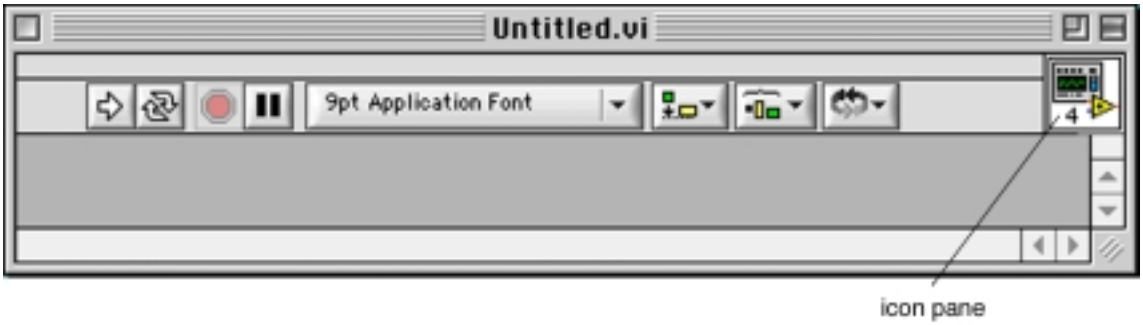
3.3 *Icon* und *connector*

Wenn ein VI als subVI eingesetzt wird, erhalten seine *controls* die Variablen vom übergeordneten VI. Nachdem das subVI seine Funktion ausgeführt hat, gibt es seine Ausgangsdaten wieder an das aufrufende VI zurück. Das *icon* des subVI symbolisiert die Funktion im übergeordneten Blockdiagramm des aufrufenden VIs. Die *terminals* sind die Schnittstellen zu den *controls* und *indicators* des subVIs. Der sogenannte VI-*connector* definiert die *terminals* der entsprechenden Ein- und Ausgänge. Der *connector* kann mit der Parameterliste einer Subroutine bei Textsprachen verglichen werden. Figur 3.4 auf Seite 19 illustriert die beiden Begriffe *icon* und *connector*.



Figur 3.4: Das *icon* und der dazugehörende *connector*

Jedes neue VI besitzt zunächst sein standard *icon* (*default icon*). Es befindet sich in der oberen, rechten Ecke des *front panel*, wie in Figur 3.5 gezeigt.



Figur 3.5: Das *default icon* oben rechts auf dem *front panel*

Ebenfalls besitzt jedes VI seinen eigenen *connector*, er wird durch Wählen der Option **Show Connector** im *icon pop-up* Menü zugänglich.

3.4 *Pull-down* Menüs

LabVIEW besitzt zwei Arten von Menüs: *pull-down* und *pop-up* Menüs. Sie werden beim Entwickeln einer LabVIEW-Applikation dauernd gebraucht. Dieses Unterkapitel erklärt die Menüumgebung. Die Menüleiste am oberen Rand des VI Fensters enthält die *pull-down*-HauptMenüs. Sie ermöglichen das Ausführen von allgemeinen und systemumfassenden Befehlen wie **Open, Save, Copy,** und **Paste** und von LabVIEW-spezifischen Funktionen. Für manche Befehle gibt es Tastenkombinationen als sogenannte *shortcuts*.

File Menü

Hier befinden sich die Befehle wie **Save** und **Print**, wie in anderen Anwenderprogrammen. Mit den Befehlen im **File** Menü werden neue VIs oder zuvor gespeicherte VIs geöffnet oder mit **Save with Options** kann eine ganze Hierarchie von VIs wahlweise in einen Ordner oder eine LabVIEW-Bibliothek abgespeichert werden.

Edit Menü

Enthält die grundlegenden Befehle wie **Cut, Copy** und **Paste** zum Bearbeiten der *front panels* oder der *block diagrams*. Mit Hilfe von Editierkommandos können auch Objekte übereinander gestapelt oder falsche Verbindungen (*wires*) gelöscht werden. Auch die Werkzeugpaletten können von hier aus benutzerspezifisch angepasst und gruppiert werden. Unter **Preferences** sind globale Einstellungen der LabVIEW-Umgebung möglich.

Operate Menü

Von hier aus kann ein LabVIEW-Programm gestartet oder gestoppt werden, auch kann zwischen *run mode* und *edit mode* gewechselt werden. Optionen zum Programmablauf sind auch in diesem Menü auswählbar.

Project Menü

Hier befinden sich die Befehle, welche die Navigation durch grössere Projekte und ihre Hierarchien erleichtern. Es kann eine Liste aller VIs oder aller subVIs im Speicher angezeigt werden. Es gibt **Find**, um Textobjekte zu finden und **Show Profile Window**, um Programmabläufe zu optimieren.

Windows Menü

Im Windows-Menü sind die Befehle zum Manipulieren des *front panel* und des *block diagram* untergebracht. Es kann zwischen den *panel-, diagram-, error list-* und *clipboard*-Fenstern umgeschaltet werden. Von hier aus können auch die oft gebrauchten frei beweglichen Paletten wieder geöffnet werden, falls sie zuvor geschlossen wurden. Zusätzlich kann Information über ein VI und seine Entwicklungsgeschichte eingesehen oder editiert werden.

Help Menü

Mit **Show Help** wird ein frei bewegliches Hilfefenster geöffnet, welches eine kurze Beschreibung eines angewählten Objektes anzeigt. Durch Anklicken

des kleinen Fragezeichens am unteren Rand des Hilfefensters wird der entsprechende Auszug über das Objekt aus der LabVIEW-Dokumentation am Bildschirm gezeigt. Ferner ist es möglich, im *help* Menü die LabVIEW-Version und die Grösse des verfügbaren Speichers ausfindig zu machen.

3.5 Verschiebbare Paletten

Die drei frei verschiebbaren Paletten, die Werkzeugpalette (*tool palette,* Figur 3.6 auf Seite 21), die Ein- Ausgabenpalette (*controls palette,* Figur 3.7 auf Seite 23) und die Funktionspalette (*functions palette,* Figur 3.8 auf Seite 23) werden beim Entwickeln von LabVIEW-Programmen sehr oft gebraucht. Hier werden sie kurz beschrieben:

Tool Palette

Ein LabVIEW Werkzeug *(tool)* wird durch wählbare Cursorsymbole charakterisiert. Die Symbole werden in der *tool palette,* gezeigt in Figur 3.6 auf Seite 21, mit Hilfe der Maus ausgewählt. Die *tools* ermöglichen das Editieren und Bedienen insbesondere bei der Programmentwicklung. Es gibt insgesamt zehn Werkzeuge mit folgenden Eigenschaften:

Figur 3.6: Die LabVIEW-Werkzeuge in der *Tool*-Palette

- Der *operating tool* (Händchen mit ausgestrecktem Zeigefinger)

  Er dient im allgemeinen der Programmbedienung und dem interaktiven Einstellen von Werten im *front panel*. Es ist das einzige Werkzeug, das auch während dem Programmablauf benutzt werden kann.

- Der *positioning tool* (kleiner schwarzer, nach links oben zeigender Pfeil)

  In diesem Cursormodus können Objekte auf dem *front panel* und dem *block diagram* verschoben und positioniert werden. Dazu werden die Objekte vorerst aktiviert, erst dann sind sie verschiebbar oder können in ihrer Grösse verändert werden.

- Der *labeling tool* (schwarzer Buchstabe "A")

  Mit ihm können die Beschriftungsettiketten der Objekte editiert werden.

- Der *wiring tool* (Fadenspülchen)

  Mit Hilfe dieses Werkzeugs werden auf dem Blockdiagramm die Objekte miteinander verbunden. Mit dem Fadenspülchen werden sozusagen die Variablen verdrahtet.

- Der *pop-up tool* (kleines abstrahiertes Menü mit Pfeilchen oben links)

  Er aktiviert die unterschiedlichen *pop-up* Menüs von Objekten, was allerdings einfacher ebenfalls durch "Apfel klick" auf dem Mac oder "rechter Mausklick" in Windows gemacht werden kann. Die *pop-up* Menüs werden im nächsten Abschnitt näher beschrieben.

- Der *scroll tool* (Händchen mit ausgestreckten fünf Fingern)

  Mit ihm kann das aktive Fenster vertikal und horizontal verschoben werden.

- Der *breakpoint tool* (roter Knopf mit Pfeilchen)

  Mit diesem Werkzeug werden im Diagramm sogenannte *breakpoints* gesetzt um eine allfällige Fehlerbeseitigung zu erleichtern. Bei einem gesetzten *breakpoint* wird das Programm angehalten und kann so an einem kritischen Punkt mit neuen Eingangsvariablen getestet werden.

- Der *probe tool* (kleines eingekreistes "P" auf nach links zeigendem Pfeil)

  Mit ihm werden Messproben an gewünschter Stelle an Verbindungsdrähte geheftet um die in den Drähten "fliessenden" Werte anzuzeigen.

- Der *color copy tool* (kleiner Farbsauger)

  Mit dem Sauger werden bereits existierende Farben zum Pinselwerkzeug und dessen Farbpalette übertragen.

- Der *color tool* (Pinselchen)

  Mit ihm werden LabVIEW-Objekte eingefärbt. Wird mit dem Pinselchen auf ein Objekt geklickt, erscheint eine Farbpalette, aus der die gewünschte Farbe ausgewählt werden kann. Meistens kann ein Objekt sowohl mit einer Hintergrund- wie auch mit einer Vordergrundfarbe bemalt werden.

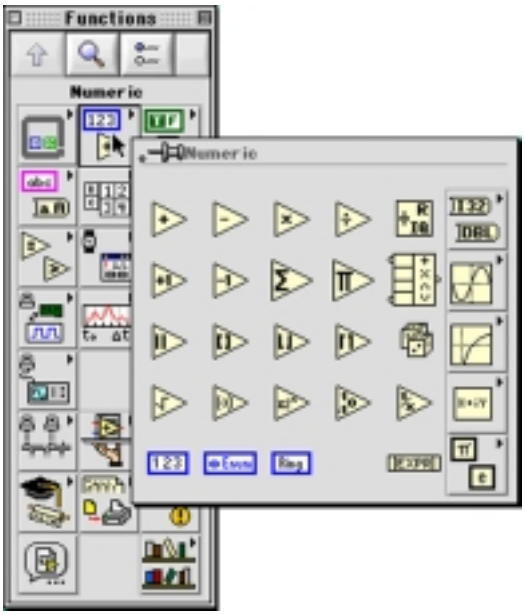
Mit dem Tabulator lassen sich die Werkzeuge der Reihe nach aktivieren. Mit der Leertaste kann im *front panel* zwischen *operating tool* und *positioning tool* und im *block diagram* zwischen *wiring tool* und *positioning tool* gewählt werden. Die Werkzeugpalette kann auch mit "Umschalt-Apfel-Klick" auf dem Mac oder "Umschalt-rechter-Mausklick" auf Windows oder Linux hervorgeholt werden.

Controls Palette

Auf diese Palette (gezeigt in Figur 3.7 auf Seite 23) ist ein Zugriff nur im *front panel* Modus möglich. Aus ihr werden die zahlreichen Objekte zum Platzieren auf dem *front panel* ausgewählt. Die Palette ist in Unterpaletten eingeteilt, welche die Objekte logisch gruppieren.

Figur 3.7: Die *Controls* Palette

Functions Palette



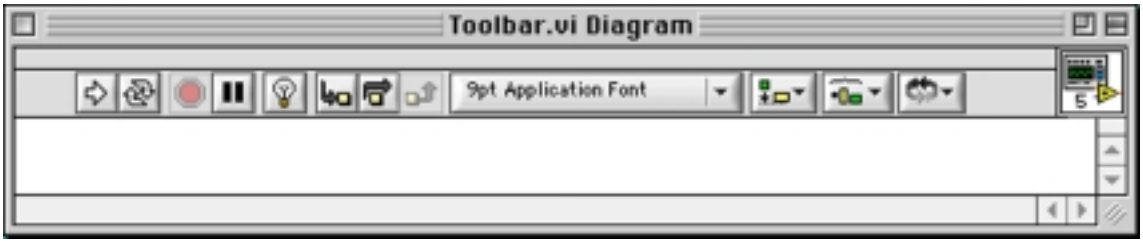
Figur 3.8: Die *Functions* Palette

Diese Palette (gezeigt in Figur 3.8 auf Seite 23) steht nur im *block diagram* Modus zur Verfügung. Mit ihrer Hilfe werden die umfangreichen LabVIEW-Funktions-

bibliotheken erreicht. Die Palette funktioniert analog zur *Controls* Palette und ist prinzipiell gleich strukturiert.

3.6 Werkzeugleiste (*toolbar)*

Die Werkzeugleiste *toolbar* befindet sich am oberen Rand jedes LabVIEW-Fensters. Sie ist in Figur 3.9 auf Seite 24 abgebildet. Hier findet man Parameter, die den Programmablauf beeinflussen, die Textdarstellung definieren und Objektausrichtungen ermöglichen. Die Werkzeugleiste zeigt im *block-diagram*-Fenster einige zusätzliche Einstellmöglichkeiten. Bei Unsicherheit kann der Cursor auf einen Werkzeugleistenknopf geführt werden, um eine Kurzbeschreibung zu aktivieren.



Figur 3.9: Die Werkzeugleiste *toolbar*

Die Bedeutung der Werkzeugleistenoptionen sind im Folgenden kurz umrissen:

- *Run Button* (nach rechts zeigender Pfeil)

  Dies ist der Startknopf der das LabVIEW Programm startet. Der Knopf ändert sein Symbol zu mehreren Pfeilen, wenn das Programm läuft. Ein zerbrochener Pfeil zeigt an, dass ein Programm nicht kompilierbar und deshalb nicht lauffähig ist.

- *Continuous Run Button* (zwei abgewinkelte Pfeile, die eine Schleife symbolisieren)

  Dieser Knopf startet das Programm nach Ablauf neu und lässt es wiederholt ablaufen bis es explizit gestoppt wird. Diese Option basiert auf einem GOTO ähnlichen Befehl und ist daher nur ausnahmsweise beim Testen zu verwenden.

- *Abort Button* (rotes Stop-Symbol)

  Dieser Knopf wird erst aktiv, nachdem ein Programm gestartet wurde. Mit ihm kann ein sofortiger "gewaltsamer" Programmunterbruch erzeugt werden. Auch diese Option ist nur beim Testen einzusetzen, zum Beispiel im Falle einer fälschlicherweise programmierten Endlosschleife.

Achtung: Der Stopknopf wirkt auf das laufende Programm wie eine *brake* Taste, das Programm wird sofort unterbrochen. Dies ist alles andere als ein schöner Anhalter und Daten können dadurch beschädigt werden. Es sollte stets eine saubere Stopfunktion ins Programm eingebaut werden. Der *Abort Button* ist nur im Notfall zu betätigen.

- *Pause Button* (zwei vertikale Balken)

  Diese Option unterbricht das Programm temporär, damit zur Fehlerbeseitigung in den Schritt-für-Schritt-Modus gewechselt werden kann. Ein erneutes Drücken dieses Knopfes hebt die Unterbrechung auf.

- *Step-Into Button, Step-Over Button, Step-Out Button* (drei Knöpfe mit abgewinkelten Pfeilen und kleinen Rechtecken)

  Bewirken ein bestimmtes Schritt-für-Schritt-Abarbeiten des Programms. (Mehr darüber im Kapitel "Fehlerbeseitigung".)

- *Execution Highlighting Button* (kleine Glühbirne)

  Das Aktivieren dieses Knopfes veranlasst die Darstellung des Programmablaufes in einer Animation des Blockdiagramms. Der Datenfluss wird damit bildlich angezeigt, was bei der Fehlerbeseitigung eine grosse Hilfe sein kann.

- *Warning Button* (Ausrufezeichen im Dreieck)

  Diese Option erscheint nur, wenn das VI zur Anzeige von Warnungen konfiguriert wurde. In diesem Falle wird durch Aktivieren dieses Knopfes eine Liste mit Warnungen aufgerufen. Eine Warnung entspricht nicht einem Fehler, sie zeigt lediglich eventuell unbeabsichtigte Situationen an, die möglicherweise problematisch sein könnten.

- *Font Ring* (Zeichensatzanzeige mit Wahlknopf)

  Hier werden alle Textdarstellungsmöglichkeiten eingestellt. Sie wirken je nachdem global oder nur in aktivierten Objekten, sowohl auf dem *front panel* wie auch im *block diagram*.

- *Alignment Ring, Distribution Ring* (Wahlknöpfe mit entsprechenden Rechtecksymbolen)

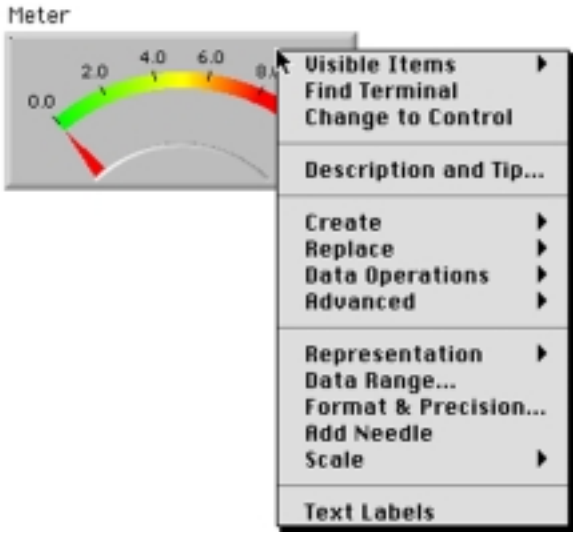
  Die beiden Auswahloptionen ermöglichen, den Symbolen entsprechend, eine gleichmässige Ausrichtung oder Verteilung von platzierten LabVIEW-Objekten auf dem *front panel* oder im *block diagram*.

- *Recorder Ring* (zwei geschwungene Pfeile mit Wahlknopf)

  Mit Hilfe dieser Option kann die Reihenfolge übereinanderliegender Objekte geändert werden.

3.7 Pop-up Menüs

Die meisten LabVIEW-Objekte lassen sich durch ihr eigenes *pop-up Menü* parametrisieren. Um zu diesen Menüs zu gelangen, muss der Cursor auf das gewünschte Objekt positioniert werden. Ein solches *pop-up* Menü ist in Figur 3.10 auf Seite 26 gezeigt. Durch "Apfel-Klick" auf Macintosh oder "rechte-Maustaste" auf Windows erscheint das jeweilige *pop-up* Menü.



Figur 3.10: Ein typisches *pop-up* Menü eines LabVIEW-Objektes

Sowohl *front-panel*- als auch *diagram*-Objekte können über diese *pop-ups* manipuliert, editiert und eingestellt werden. Manche *pop-ups* sind hierarchisch in Untermenüs gegliedert. Manche bieten eine Wahl, wobei die aktuelle mit einer Marke gekennzeichnet ist. Einige Menüpunkte verlangen bestimmte Einstellungen. Bei Optionen mit Punkten (...) erscheinen Einstellfenster, die Menüoptionen ohne ▸ werden sofort ausgeführt.

Oft weisen verschiedene Teile derselben Objekte unterschiedliche *pop-up* Menüs auf. So können ausgewählte Teilbereiche in ihren Funktionen verändert werden.

Change to Control / Change to Indicator

Beim Auswählen von **Change to Control / Indicator** kann ein Eingabe- in ein Ausgabeobjekt oder ein Ausgabe- in ein Eingabeobjekt konvertiert werden.

Show Terminals / Show Icons

Diese *pop-up* Option ist nur auf dem Blockdiagramm vorhanden, sie zeigt wahlweise das *icon* oder den *connector* eines subVIs im Blockdiagramm.

Show

Mit **Show** können bei manchen Objekten kosmetische Attribute gezeigt oder versteckt werden.

Data Operations

Mit dem **Data Operations** *pop-up* können einige praktische Optionen ausgewählt werden, welche die Daten und ihr Format in *controls* oder *indicators* beeinflussen.
Mit **Reinitialize to Default** wird ein Objekt auf einen vorher mit **Make Current Value Default** gesetzten Normwert zurückgesetzt. Mit **Cut Data, Copy Data** und **Paste Data** können Werte aus oder in einen *control* oder *indicator* kopiert werden. Letztendlich kann mit **Description** ein Objekt dokumentiert oder eine bereits existierende Beschreibung gelesen und editiert werden.

3.8 Help

Das LabVIEW *Help* Fenster bietet eine unentbehrliche Hilfe für Funktionen, Konstanten, *subVIs, controls* und *indicators*. Um das *Help* Fenster sichtbar zu machen, wird **Show Help** im **Help** Menü gewählt, oder man betätigt <**Cmd H**> (auf dem Macintosh) oder <**Ctrl H**> (auf WIN-PCs). Wenn nun ein *tool* über eine Funktion, ein *subVI* oder ein *VI icon* fährt, zeigt das *Help* Fenster, welche Datentypen die Ein- und Ausgänge benutzen. Eingänge kommen von links, Ausgänge zeigen nach rechts. Diejenigen Eingänge, die Daten benötigen, sind mit fetter Schrift angeschrieben, Eingänge mit *default* Werten können mit diesem Wert belassen werden, d. h. sie können ”offen” bleiben und müssen nicht verdrahtet werden.

3.9 SubVIs

Um von den Fähigkeiten von LabVIEW vollen Gebrauch zu machen, muss man mit der hierarchischen Struktur vertraut sein. Ein *subVI* ist ein unabhängiges *stand-alone*-Programm, welches von einem anderen Programm gebraucht wird. Nachdem ein VI kreiert wurde, kann es im Blockdiagramm eines höherliegenden VIs als *subVI* benutzt werden. Ein LabVIEW-*subVI* ist mit einer Subroutine in C zu vergleichen; d. h. insbesondere ist die Anzahl *subVIs*, die in einem LabVIEW Programm verwendet werden kann, nicht limitiert. Diese modulare Struktur macht die Programme verständlich, einfach veränderbar und hilft bei der Fehlersuche.

Kapitel 4

LabVIEWs einfache Datentypen

Dieses Kapitel beschreibt die grundlegenden LabVIEW-Prinzipien, die verschiedenen Datentypen und wie man ein LabVIEW Programm aufbaut. Es ist wichtig, dieses Kapitel gut zu verstehen, da hier die Grundbausteine von LabVIEW beschrieben werden.

Themen im 4. Kapitel :

- *numeric*
- *string*
- *Boolean*
- *path*
- *text ring*

4.1 Typen von *controls* und *indicators*

LabVIEW besitzt vier Grundtypen von *controls* und *indicators: numeric* (numerische), *boolean* (Boole'sche), *string* (Zeichenketten) und *path* (Pfad). Weiter gibt es die zusammengesetzten, komplexeren wie *array* (Matrizen), *clusters* (Verbünde), *charts* (Anzeigen) und *graphs* (Grafiken), die später erklärt werden.

Numerische *controls* und *indicators*

Numeric controls erlauben die Eingabe numerischer Werte, *numeric indicators* zeigen numerische Werte an. LabVIEW besitzt viele Typen von numerischen Objekten, einige sind in Figur 4.1 dargestellt: Knöpfe, Schieber, Tanks, Thermometer und natürlich die einfache Digitalanzeige. Alle Objekte können sowohl *indicator* als auch *control* sein. Numerische *terminals* erscheinen auf dem Blockdiagramm als kleine Kästchen unter dem angezeigte Datentyp z.B. **"DBL"** (32bit Fliesskomma) in entsprechender Farbe.

Figur 4.1: Numerische Objekte

Pop-up Optionen

Numerische *controls* und *indicators* besitzen ihre eigenen *Pop-up*-Menüs. Hier können das Format und die Genauigkeit der numerischen Darstellung eingestellt werden. Man kann auch die gewünschte Notation auswählen. Bei komplexeren Objekten wie Knöpfen und Thermometern kann mit **Show Digital Display** eine zusätzliche Digitalanzeige, die den genauen Wert anzeigt, erzeugt werden.

Text ring

Der *text ring* ist ein Spezialtyp eines numerischen Objekts, dabei wird eine Zahl mit einem Text assoziiert. Jede Texteingabe im Ring wird durch eine ganze Zahl von 0 bis n repräsentiert.

Boole'scher Datentyp *(Boolean)*

Boolean steht für einen digitalen Zustand wie "Ein und Aus". Boole'sche Daten können nur einen von zwei Zuständen annehmen: *true* und *false*. LabVIEW besitzt zahlreiche *Booleans* wie Schalter, LEDs und Knöpfe. Jedes boole'sche Objekt zeigt je nach wahrscheinlichem Gebrauch seinen *default*-Status, entweder als *control* oder als *indicator*. *Boolean terminals* auf dem Blockdiagramm sind grün und enthalten die Zeichen "**TF**". Die Figur 4.2 zeigt einige Boole'sche Objekte.

Figur 4.2: Boole'sche Objekte

Achtung: *Control terminals* haben DICKE Umrandungen im Gegensatz zu den *indicator terminals* welche eine DÜNNE Umrandung besitzen. Der Unterschied ist wichtig: *Control* = Eingang, *Indicator* = Ausgabe.

String

String controls und *indicators* behandeln Daten als Zeichen im ASCII Format. Sie erlauben Textanzeigen und Texteingaben. *String terminals* und Drähte die Zeichenketten transportieren, sind rosarot. Die *terminals* enthalten die Buchstaben "**abc**".

Achtung: *string controls* und *indicators* können auch Zahlen enthalten. Es handelt sich dabei NICHT um numerische Daten, sondern um ASCII-Zeichen.

String controls und *indicators* sind recht elementar. Ihre *pop-up* Menüs enthalten einige spezielle Optionen. Mit **Show Scrollbar** kann ein Teil eines längeren Texts als Ausschnitt in einem kleineren *String control* oder *indicator* angezeigt werden.

Mit der Option '\' **Code Display** werden Spezialzeichen in einen *string control* bzw. *string indicator* geschrieben, die nicht angezeigt werden können. Diese Zeichen müssen sofort nach dem '\' eingefügt werden (siehe folgende Tabelle).

LabVIEW "\" Codes:

Code	LabVIEW Interpretation
\ 00 - \ FF	Hexadezimaler Wert eines 8-bit Zeichens
\ b	Backspace (ASCII BS, äquivalent zu \ 08)
\ f	Formfeed (ASCII FF, äquivalent zu \ OC)
\ n	New Line (ASCII LF, äquivalent zu \ OA)
\ r	Return (ASCII CR, äquivalent zu \ 0D)
\ s	Space (äquivalent zu \ 20)
\ \	Backslash (ASCII \, äquivalent zu \ 5C)

für hexadezimale Zeichen müssen grosse Buchstaben gebraucht werden. Die kleinen Buchstaben sind den Spezialzeichen wie *formfeed* und *backspace* vorenthalten.

Path

Die *path controls* und *indicators* werden gebraucht, um Pfade zu Dateien, Ordner oder Verzeichnisse anzuzeigen. Wenn in einer Funktion, die einen Pfad ausgeben soll, ein Fehler auftritt, wird im *path indicator* die Meldung <**Not a Path**> erscheinen. *Path* ist ein spezieller Datentyp, seine *terminals* und *wires* werden auf dem Blockdiagramm blaugrün dargestellt.

Kapitel 5

LabVIEW Entwicklungsumgebung

Dieses Kapitel beschreibt Grundlagen der LabVIEW Entwicklungsumgebung, das LabVIEW-eigene Bibliothekenformat, Fehlersuchhilfen, Dokumentationsmöglichkeiten, subVI Implementationen und hilfreiche Verfahren, welche die Entwicklungszeit verkürzen.

Themen im 5. Kapitel :

- *VI library* (die LabVIEW VI Bibliothek)
- ”Zerbrochene” VIs
- Programmablauf in Einzelschritten
- Fehlersuchhilfe durch Animation auf dem Blockdiagramm
- Setzen von *probes*
- *breakpoints*
- *subVI*
- *Icon editor*

5.1 Laden und Speichern von VIs

Ein VI wird mit **Open** im **File** Menü geladen. Während dem Ladevorgang erscheint ein Statusfenster, das anzeigt, welche subVIs gerade geladen werden. Gespeichert wird ein VI mit **Save** im **File** Menü. LabVIEW kann VIs in einem LabVIEW-eigenen komprimierten Format in sogenannten *VI libraries* abspeichern. LabVIEW adressiert die VIs mit Namen, d.h. es darf sich nicht mehr als ein VI unter demselben Namen im Arbeitsspeicher befinden. Wenn mehrere VIs mit gleichen Namen gespeichert vorliegen, lädt LabVIEW das erste, das es findet.

Abspeicheroptionen

Ein VI kann unter verschiedenen **File** Menü-Optionen abgespeichert werden:

- Unter der **Save** Option für normales Speichern eines VIs als Einzeldatei auf Disk.
- Unter der **Save As...** Option, die einem VI im Arbeitsspeicher einen neuen Namen gibt und gleichzeitig eine Kopie unter diesem Namen auf Disk speichert. Wenn kein neuer Name eingegeben wird, überschreibt LabVIEW nach einer Warnung die alte Kopie.
- Unter der **Save a Copy As...** Option, die eine Kopie des VIs im Arbeitsspeicher auf Disk kopiert. Diese Option ändert den Namen des VIs im Arbeitsspeicher nicht.
- Unter der **Save with Options...** Option, welche eine Dialogbox mit weiteren Möglichkeiten aufruft. So kann mit **Save Entire Hierarchy** eine ganze VIHierarchie mit allen subVIs entweder in einem Ordner oder einer LabVIEW *library* gespeichert werden. Ein VI oder eine ganze Hierarchie kann hier auch ohne dazugehörende Blockdiagramme gespeichert werden. Bevor man das tut, muss man allerdings sicher sein, dass eine entsprechende Kopie mit Diagramm irgendwo vorhanden bleibt, da sonst keine Änderungen an sämtlichen Blockdiagrammen der gesamten Hierarchie von VIs mehr gemacht werden können.

Achtung: LabVIEW VIs nie ohne Blockdiagramm abspeichern, wenn nicht zuvor eine Sicherheitskopie mit Diagramm gemacht wurde.

5.2 VI Bibliotheken (.llb Dateien)

VI libraries (VI Bibliotheken) lassen sich wie Verzeichnise speichern, laden und öffnen. Das Betriebssystem behandelt Bibliotheken wie einzelne Dateien. Eine

VI library kann allerdings nur von LabVIEW geöffnet werden. LabVIEW *libraries* enthalten die einzelnen VIs in komprimierter Form, womit Speicherplatz gespart werden kann.

> **Achtung:** Wenn eine *VI library* beschädigt wird, so sind alle Dateien darin beschädigt, daher sollen öfters Sicherheitskopien gemacht werden.

In der Dialogbox zum Auswählen einer Datei zeigen sich *VI libraries* wie Ordner mit der Endung ".llb". In LabVIEW kann eine neue VI Bibliothek mit dem **New...** Knopf sowie bei **Save, Save As...** oder **Save a Copy As...** erstellt werden. Ist eine *library* einmal kreiert worden, können darin viele VIs abgespeichert werden. Um eine bestehende *VI library* zu editieren, braucht man die Option **Edit VI Library...** im **File** Menü.
Alle LabVIEW Funktionen sind in *VI libraries* gespeichert. Der Zugriff auf diese Bibliotheken erfolgt über die *Functions* Palette.

> **Achtung:** Eigene VIs sollen nie im vi.lib Ordner (Verzeichnis) abgespeichert werden, da dieser Ordner für LabVIEW-eigene Bibliotheken reserviert ist. Eigene VIs sollten sich in eigenen Bibliotheken oder Ordnern befinden.

Wenn LabVIEW gestartet wird, baut das Programm mit den im LabVIEW Ordner gefundenen Ordnern mit .llb Endungen seine Funktionspaletten auf. Falls eigene VIs ebenfalls in diesen Paletten erscheinen sollen, so müssen sich die Ordner oder *VI libraries* in einem Ordner mit der Endung .llb befinden.

5.3 Techniken zur Fehlersuche

LabVIEW besitzt viele Werkzeuge zur Fehlerbeseitigung *(debugging tools).* Dieses Unterkapitel erklärt, wie man sie nutzen kann.

Reparieren eines "zerbrochenen" VIs

Ein zerbrochenes VI kann nicht kompiliert und gestartet werden. Der *run* Knopf zeigt einen zerbrochenen Pfeil. Ein VI bleibt während des Programmierens zerbrochen, bis alle *icons* auf dem Blockdiagramm verdrahtet sind. Wenn nach dem richtigen Verdrahten immer noch ein zerbrochener Pfeil vorliegt, ist es ratsam, mit **Remove Bad Wires** im **Edit** Menü das VI fit zu machen.
Um herauszufinden, warum ein VI zerbrochen ist, kann mit der Maus auf den zerbrochenen Pfeil geklickt oder **Show Error List** im **Windows** Menü gewählt werden. Es erscheint eine Liste mit Hinweisen auf die gefundenen Fehler. In dieser Liste können mittels Dialogring auch Fehler anderer offener VIs angezeigt werden. Um mehr über einen bestimmten Fehler zu erfahren, kann die entsprechende Fehlermeldung angeklickt werden. Um den Fehler zu lokalisieren, kann man ihn doppelt anklicken; LabVIEW bringt dann das entsprechende Fenster in den Vordergrund und markiert die fehlerhafte Zone.

Warnungen

Bei Aktivieren des **Show Warnings** Feldes in der *Error List* (Fehlerliste) zeigt LabVIEW Warnungen mit einem Warnzeichen in der Editierpalette an. Warnungen unterscheiden sich von Fehlern, die zu einem zerbrochenen VI führen. LabVIEW warnt bei ungewöhnlichen Situationen, z.B. wenn ein *control terminal* auf dem *block diagram* nicht verdrahtet wurde.

Einzelschrittmodus

Zum Suchen von Programmfehlern kann ein Blockdiagramm Schritt für Schritt ausgeführt werden. Dazu muss der *single-step mode* (Einzelschrittmodus) aktiviert werden, was durch Drücken der *pause* Taste gemacht wird. Die *pause* Taste kann auch während des normalen Programmablaufs aktiviert werden. Jeder Schritt wird nun manuell durch Drücken einer Schritttaste ausgelöst. Es gibt drei Arten von Einzelschritten, repräsentiert durch drei entsprechende Tasten in der Werkzeugleiste:

- **Step Over** um eine LabVIEW Struktur (Schleife, subVI etc.) auszuführen und danach anzuhalten.
- **Step Into** um Einzelschritte in einer ganzen LabVIEW Struktur auszuführen. Sind in einem subVI Einzelschritte ebenfalls auszuführen, muss sich das subVI auch im Einzelschrittmodus befinden. SubVIs, die sich nicht in diesem Modus befinden, werden normal ausgeführt.
- **Step Out** um aus der momentanen Struktur auszutreten und anzuhalten.

Blockdiagramm Animation

Um mit einer Animation den Ablauf des Programms auf dem Blockschaltbild zu verfolgen, kann die *execution highlighting* Option (der Knopf mit der Glühbirne in der Werkzeugleiste) aktiviert werden.
Meist wird *execution highlighting* zusammen mit Einzelschritten aktiviert, um zu sehen, wie sich die Daten in Form bewegter Blasen von Knoten zu Knoten bewegen.

Benutzen von *probes*

Proben dienen dazu, Zwischenergebnisse in einem VI, welches fragwürdige oder unerwartete Resultate erzeugt, im Diagramm anzuzeigen. Eine Probe wird entweder mit dem *probe tool* aus der Werkzeugpalette an einen Draht geheftet oder mit Hilfe des *pop-up* Menüs eines Drahtes gesetzt. Proben sind besonders effektiv, wenn sie zusammen mit *execution highlighting* und *single-step mode* eingesetzt werden.

Setzen von *breakpoints*

Breakpoints unterbrechen den Ablauf eines Programms am Ort des gesetzten *breakpoints*. Sie werden mit dem *breakpoint tool* dort gesetzt, wo das Programm unterbrochen werden soll. Mit dem *run*-Pfeil kann danach zum nächsten *breakpoint* gesprungen werden.

Praktische Tips

Die folgende Liste enthält hilfreiche Tips zum Entwickeln von LabVIEW Applikationen:

- Oft benutzte Menü Optionen besitzen Kurzbefehle. Sie stehen bei jeder Menü Option. Einige der meist gebrauchten *shortcuts* sind:
  <Cmd> <r> oder **<Ctrl> <r>** - Startet ein VI
  <Cmd> <f> oder **<Ctrl> <f>** - Hin- und Herschalten von *front panel* zu *diagram*
  <Cmd> <h> oder **<Ctrl> <h>** - Ein- und Ausschalten des *help* Fensters
  <Cmd> <b> oder **<Ctrl> <b>** - Löscht die *bad wires*
  <Cmd> <w> oder **<Ctrl> <w>** - Schliesst das aktive Fenster
- Um zum nächstfolgenden Werkzeuge der *tool* Palette zu gelangen, drücke **<Tab>**.
- Um die Richtung eines *wires* während des Verdrahtens zu ändern, drücke die Leertaste.
- Um kleine Bewegungen von angewählten Objekten auf dem *front panel* oder dem *diagram* zu machen, brauche die Pfeiltasten.
- Um in numerischen Anzeigen schneller hinauf- oder hinunterzuzählen, drücke **<Shift>** während des Klickens auf die Zählknöpfe der Anzeige.
- Um schneller Einträge in *ring controls* zu machen, drücke **<Shift-Enter>** nach dem Schreiben. **<Shift-Enter>** akzeptiert den Eintrag und positioniert für den nächsten Eintrag.
- Um ein vorerst ausgewähltes Objekt zu duplizieren, drücke **<Ctrl>** auf WIN-PCs und **<option>** auf einem Macintosh unter gleichzeitigem Ziehen des Objekts mit der Maus. Das sich bewegende Objekt ist eine Kopie.
- Um die Bewegung eines Objekts nur in einer der beiden Koordinaten zu erreichen, drücke **<Shift>** während des Ziehens mit der Maus.
- Um eine Farbe eines Objekts auszuwählen, wähle zuerst das Pinselwerkzeug aus der Palette. Positioniere den Pinsel auf dem Objekt und drücke **<Ctrl>** auf WIN-PCs und **<option>** auf einem Macintosh, dadurch wird der

Pinsel zum Sauger. Wähle die Farbe durch Klicken auf das Objekt. Lasse die **<Ctrl>** oder **<option>** Taste los, um andere Objekte durch Klicken mit dem Pinsel mit der gewählten Farbe einzufärben.

- Ein häufig auftretender Fehler tritt beim Verdrahten zweier *controls* oder zweier *controls* mit einem *indicator* auf. Es erscheint die Fehlermeldung: *Signal has multiple sources.* Wähle **Change to Indicator** mit dem *pop-up* Menü des *controls.*

- Um einen *wire* während dem Verdrahten zu löschen, klicke die rechte Maustaste (auf WIN-PCs) oder verdrahte über das Fenster hinaus und klicke dann (auf dem Macintosh).

5.4 Kreieren eines *subVIs*

LabVIEW verdankt viele seiner Bequemlichkeiten seiner Modularität. Die Teilfunktionen grösserer Programme können als vollständige Module eines nach dem anderen unabhängig programmiert und getestet werden. Um ein VI als *subVI* zu benutzen, muss ein *icon* gestaltet und der *connector* (Verbinder) definiert werden. Der *connector* funktioniert dabei wie eine Parameterliste bei konventionellen Programmiersprachen, durch ihn laufen die Variablen vom und zum aufrufenden VI.

Gestalten eines *icons*

Jedes *subVI* besitzt ein *icon*, welches seine Funktion im Blockdiagramm des übergeordneten VIs identifiziert. Mit **Edit Icon** vom *pop-up* Menü des kleinen *icon* Fensters oben rechts im *front panel* gelangt man in den sog. *Icon Editor,* dem Editierfenster eines *icons.* Mit den vorhandenen Werkzeugen kann ein Symbol, ein Text oder ein Bild in das *icon* gemalt werden. Die Werkzeuge haben folgende Funktionen:

Bleistift Zeichnet und löscht einzelne Pixel.

Linie Zeichnet gerade Linien. Drücke <Shift> um horizontale, vertikale oder diagonale Linien zu zeichnen.

Tupfer Kopiert die Vordergrundfarbe eines Elementes ins *icon.*

Füllkessel Füllt einen markierten Bereich mit der Vordergrundfarbe.

Rechteck Zeichnet einen Rahmen in der Vordergrundfarbe um ein aufgespanntes Rechteck. Bei einem Doppelklick auf dieses Werkzeug wird ein Rahmen in der Vordergrundfarbe um das ganze *icon* gezeichnet.

gefülltes Rechteck Zeichnet ein gerahmtes Rechteck mit dem Rahmen der Vordergrundfarbe, gefüllt mit der Hintergrundfarbe. Durch Doppelklick wird das ganze *icon* in neu gewählten Farben eingefärbt.

gestricheltes Rechteck Wählt einen Bereich des *icons* für Bewegungen, Kopien und Änderungen aus.

Buchstabe A Schreibt Text ins *icon*.

Vordergrund / Hintergrund Zeigt die momentane Vorder- oder Hintergrundfarbe. Klicke darauf, um eine neue Farbe aus der Palette zu wählen.

Definieren des *Connectors*

Erst ein definierter *connector* erlaubt den Variablentransport in und aus dem VI. Dazu wird im *icon pop-up* Menü **Show Connector** gewählt. LabVIEW wählt automatisch einen geeigneten *connector*-Raster aufgrund der vorhandenen *controls* und *indicators*, die sich bereits auf dem *front panel* befinden. Die *connector*-Felder werden *terminals* genannt. Falls ein neuer *terminal* gebraucht wird, kann er mittels *terminal pop-up* erzeugt werden. Ein *connector*-Raster kann wahlweise auch gedreht oder gespiegelt werden. Um einen *connector terminal* einem *control* oder einem *indicator* zuzuordnen, sind folgende Schritte nötig:

- Mit dem Drahtspulwerkzeug wird auf das Feld desjenigen *connector* Rasters geklickt, welcher einem entsprechenden *control* oder *indicator* auf dem *front panel* zugeordnet werden soll. Dieses Feld wird dabei schwarz gefärbt.

- Danach wird mit der Drahtspule auf den zuzuweisenden *control* oder *indicator* geklickt.

- Abschliessend klicke man ebenfalls mit der Drahtspule auf einen leeren Bereich im *front panels*, der definierte *terminal* wechselt zu grauer Farbe, um die Zuweisung zu quittieren. Falls der *terminal* weiss bleibt, konnte die Verbindung nicht hergestellt werden und der Vorgang muss wiederholt werden.

Benutzen eines VIs als *subVI*

Jedes VI mit einem definierten *icon* und *connector* kann wie eine LabVIEW-eigene Funktion in einem beliebigen Blockdiagramm als *subVI* plaziert werden. Ein abgespeichertes VI wird mit Hilfe der Funktionspalette durch Anwählen des Feldes mit dem VI-Symbol auf das Blockdiagramm geholt.

5.5 Dokumentation

Es ist wichtig, ein VI so zu beschreiben, damit die Funktion von einer anderen Person verstanden werden kann. Dies ist auch für den Programmierer hilfreich, wenn er nach längerer Zeit Änderungen vornehmen muss.

Beschreibungen einzelner Objekte

Um einen *control* oder *indicator* zu beschreiben, wähle man **Description** (Beschreibung) aus dem **Data Operation** Untermenü des *pop-up* Menüs des Objekts. Die Beschreibung kann nun in die Dialogbox eingetragen werden. Mit **Description** kann die Information später gelesen werden.

Dokumentation eines VIs mit Get Info... Option

Bei Auswahl von **Get Info** aus dem **File** Menü wird die Informationsbox für das VI gezeigt. Sie kann gebraucht werden um

- eine Beschreibung des VIs zu machen.
- das VI zu sperren oder wieder zu entsperren *lock / unlock.* (Ein gesperrtes VI kann nicht editiert werden)
- eine Liste der Änderungen des VIs anzuschauen.
- den Pfad eines VIs festzustellen.
- den Speicherplatz des VIs abzufragen.

Kapitel 6

Strukturen

Strukturen sind eine sehr wichtige Art von *nodes*. Sie bestimmen den Ablauf eines VI's, wie Schleifen in einer zeilenorientierten Programmiersprache. Dieses Kapitel behandelt die vier Strukturtypen *While loop, For loop, Case structure* und *Sequence structure.* Auch der *Formula Node* (Formelknoten) und wie man in LabVIEW eine *pop-up* Meldedialogbox erzeugen kann, wird beschrieben. Ferner wird die Steuerung des zeitlichen Ablaufs der *loops* beschrieben.

Themen im 6. Kapitel :

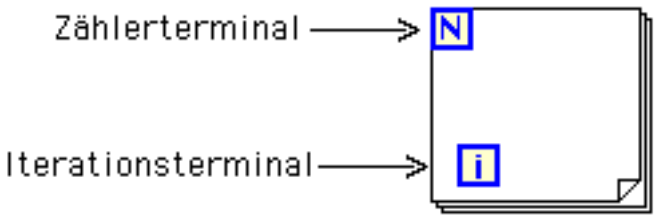
- *For loop* ("For"-Schleife)
- *While loop* ("While"-Schleife)
- *iteration terminal* (Iterationsabfrage)
- *conditional terminal* (Bedingungsabfrage)
- *count terminal* (Schleifenzähler)
- *shift register* (Schieberegister)
- *case structure* ("Case"-Struktur)
- *selector terminal* (Wahlabfrage)
- *dialog box* (Meldebox)
- *sequence structure* (Sequenzstruktur)
- *sequence local* (lokale Sequenzvariable)
- *formula node* (Formelknoten)

6.1 Die Schleifen *(loops)*

Die beiden Schleifen *For Loop* und *While Loop* steuern Wiederholungen in einem VI. Die "For" Schleife führt ihren Inhalt in einer bestimmten Anzahl Durchgängen aus, die "While" Schleife führt ihren Inhalt aus, bis die Abbruchbedingung erfüllt ist. Die *loops* findet man in der *Functions* Palette ganz oben links.

Die "For"-Schleife *(For Loop)*

Die Figur 6.1 zeigt eine "For"-Schleife *(For Loop)*, sie führt den Code innerhalb des Rahmens n-mal aus. Das kleine **"i"** ist der Schleifenzähler *(iteration terminal)*, er zählt wieviele Male die Schleife ausgeführt ist. Die Anzahl Schleifendurchgänge wird von aussen an den **"N"** *(count terminal)* verdrahtet.

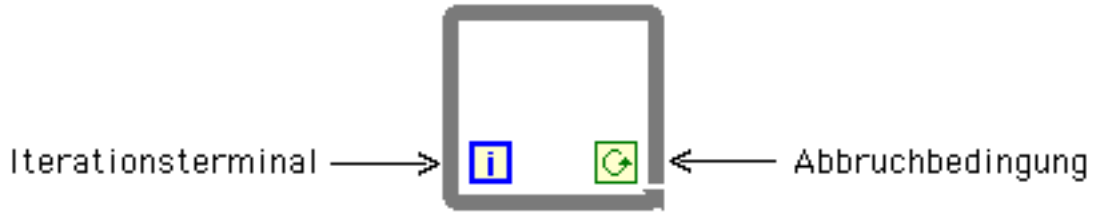


Figur 6.1: Eine "For"-Schleife

Der *iteration terminal* enthält die laufende Anzahl der bereits abgearbeiteten Iterationen: 0 während der ersten, 1 in der zweiten usw. bis N-1. Wenn der *count terminal* eine Null erhält, wird die Schleife nicht ausgeführt.

Die "While"-Schleife *(While Loop)*

Die "While"-Schleife wie sie in Figur 6.2 gezeigt ist, führt das Diagramm im Rahmen wiederholt aus, bis der boole'sche Wert am *conditional terminals* den Wert FALSE hat. LabVIEW kontrolliert den Wert am Ende jedes Schleifendurchgangs, im Falle eines TRUE-Wertes wird ein neuer Durchgang gestartet. Der *default* Wert des *conditional terminals* ist *false*, d.h. wenn der *conditional terminal* nicht angeschlossen ist, macht die Schleife nur einen einzigen Durchgang.



Figur 6.2: Eine "While"-Schleife

Platzieren von Strukturen

Die Strukturen werden mit Hilfe der Maus in der **Functions** Palette ausgewählt und dann auf das Blockdiagramm gezogen. Sie können auf dem Diagramm um bestehende Elemente herum platziert werden. Mit der Maus können auch angewählte Objekte von ausserhalb in den Rahmen der Struktur hineingezogen werden.

Terminals in Schleifen

Da LabVIEW nach dem Datenflussprinzip arbeitet, müssen die Eingangsvariablen in sog. *tunnels* vor Ablauf der Schleife in diese hineingelangen. Die Ausgangsvariablen verlassen die Schleifenstruktur erst nach Beenden der Iterationen. Wenn *terminals* bei jedem Schleifendurchgang abgefragt werden sollen, müssen sie sich in der Schleife befinden.

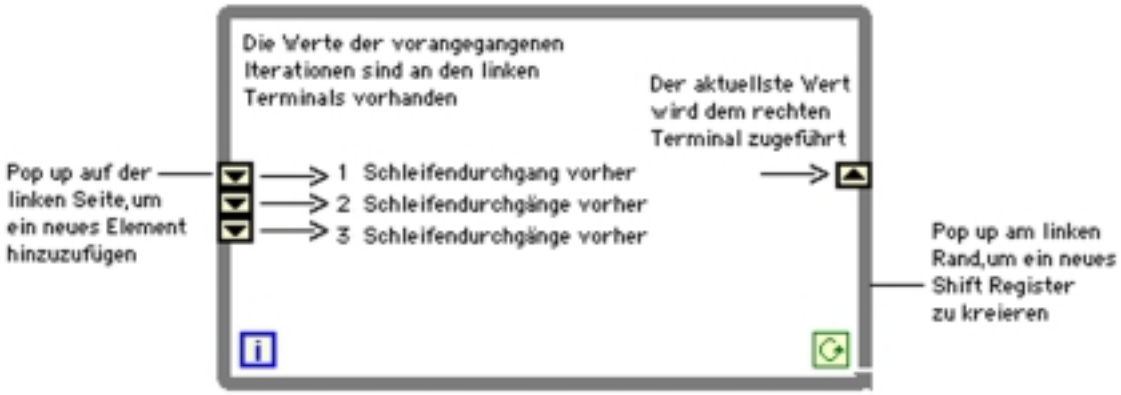
Merke: Während des ersten Durchgangs einer "For"- oder "While"-Schleife ist der *iteration counter* (Iterationszähler) 0. Wenn die Iterationen gezählt werden sollen, muss eine 1 zum Zählerstand addiert werden.

6.2 Schieberegister *(shift register)*

Schieberegister gibt es für "For"- und "While"-Schleifen. Es sind lokale Variable, die ihren Wert an die nächste Iteration weitergeben. Ihre Form ist eine LabVIEW Spezialität, ein Produkt der grafischen Programmierung, bildlich sehr leicht zu verstehen.
Schieberegister werden mit dem *pop-up* Menü der betreffenden Struktur mit **Add Shift Register** erzeugt. Sie manifestieren sich als Rechtecke mit Dreieckpfeilen *(shift register terminals)* am rechten und linken Rand der Schleife. Der rechte *terminal* speichert den erhaltenen Wert nach dem Schleifendurchgang. Dieser Wert wird danach in den linken *terminal* geschoben, bereit für die Nutzung in der nächsten Iteration. Das Prinzip wird in Figur 6.3 bildlich erläutert.

Figur 6.3: Schieberegister *shift registers*

Schieberegister können sämtliche Datentypen aufnehmen; sie passen sich automatisch den ankommenden Datentypen an.



Figur 6.4: Schieberegister als Speicher für frühere Iterationen

Schieberegister können vergrössert werden so, dass sie die Werte mehrerer vorhergehender Iterationen speichern, wie dies in Figur 6.4 dargestellt ist. Diese Möglichkeit kann man z.B. dazu gebrauchen, den Mittelwert von Datenpunkten, die fortlaufend bei jeder Iteration aufgenommen werden, zu bilden.

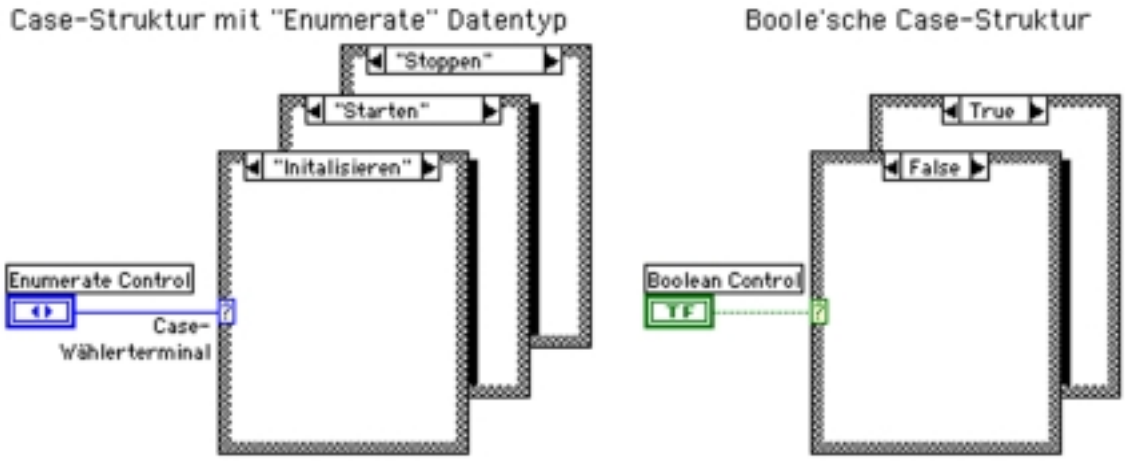
Initialisieren von Schieberegistern

Schieberegister sollten in der Regel mit einem Startwert versehen werden. Um ein Schieberegister mit einem bestimmten Wert zu initialisieren, wird der Initalwert von ausserhalb der Schleife mit dem linken Schieberegister-*terminal* verbunden. Im Falle eines nicht initialisierten Schieberegisters setzt LabVIEW je nach Datentyp einen spezifischen Initialwert, der nur beim ersten Start des Programms nach dem Laden definiert ist. Beim Schieberegisterdatentyp *Boolean* ist z.B. der Initialwert FALSE, bei numerischen Datentypen ist er 0. Wird das Programm dann nochmals gestartet, entspricht der Initialwert dem letzten Wert des vorangegangenen Programmablaufes.

Achtung: LabVIEW behält die Werte in Schieberegister bis zum Schliessen der VI's. Um unerwarteten Resultaten vorzubeugen, sollten Schieberegister immer initialisiert werden.

6.3 Die "Case"-Struktur

Die *case structure* ("Case"-Struktur) ist die LabVIEW-Methode zur Steuerung des Programmablaufes durch Bedingungen, ähnlich wie bei einem "if-then-else" Befehl. Die "Case"-Struktur ist wie "For" und "While" in der **Functions** Palette zu finden. Sie besteht, wie in Figur 6.5 auf Seite 45 gezeichnet, aus zwei oder mehreren Subdiagrammen. Je nach Wert, der dem *selector terminal* zugeführt wurde, wird ausschliesslich eines der Subdiagramme ausgeführt. Der *selector terminal* kann entweder ein boole'scher oder ein numerischer Datentyp sein. Im Falle eines *Boolean* gibt es nur zwei *cases*: TRUE und FALSE. Bei einem *numeric*



Figur 6.5: Die "Case"-Struktur

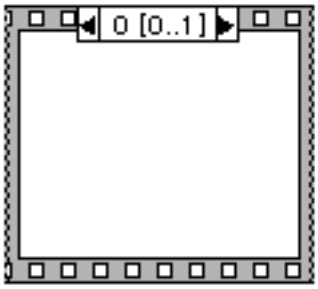
kann eine "Case"-Struktur 0 bis $2^{15} - 1$ *cases* haben. Die "Case"-Struktur zeigt nach dem Platzieren zwei boole'sche *cases*. Der Struktur können aber mittels *pop-up* leicht weitere *cases* hinzugefügt werden.

Die "Case"-Struktur besitzt mehrere Subdiagramme, sichtbar ist jedoch nur ein *case*, ähnlich wie bei einem Kartenstapel. Durch Klicken auf die Rechts/Links-Pfeilchen wird der nächste "Case" nach oben geholt.

Verdrahtet man eine Fliesskommazahl an den *selector*, wird sie zur nächsten Integerzahl gerundet, negative Zahlen erhalten den Wert 0. Zahlen höher als die maximale Anzahl *cases* werden auf $2^{15} - 1$ gesetzt. Der *selector* kann am linken Rand der Struktur frei plaziert werden, muss aber zwingend einen Wert erhalten. Der *selector* passt sich dem verdrahteten Datentyp an. Wenn ein *numeric* in einen *Boolean* umgewandelt wird, wechseln die *cases* 0 und 1 zu FALSE und TRUE. Die anderen *cases* aber bleiben bestehen bis sie explizit gelöscht werden. Praktisch im Zusammenhang mit der "Case"-Struktur ist der Datentyp *enumerate*, welcher einer Textauswahl eine Integerzahl von 0 bis n zuordnet. Wird ein *enumerate* Typ an den *selector* verdrahtet, erscheint der entsprechend zugeordnete Text zwischen den Wahlpfeilchen am oberen Rand der *case structure*.

Verdrahten von Ein- und Ausgängen

Die Daten aller Eingänge *(terminals)* gelangen in jeden *case*. Die einzelnen *cases* müssen nicht zwingend Daten von ausserhalb der Struktur erhalten oder abgeben, wenn jedoch einer der *cases* Daten nach aussen abgibt, müssen alle *cases* dies auch tun, sonst zeigt LabVIEW den zerbrochenen Pfeil.



Figur 6.6: Die Sequenzstruktur

Anfügen von *cases*

Im *pop-up* Menü der "Case"-Struktur gibt es die Optionen **Add Case After** bzw. **Add Case Before** zum Anfügen eines neuen *case* vor oder nach dem gegenwärtigen *case*. Falls der gegenwärtige dubliziert werden soll, geschieht dies mit **Duplicate Case**, mit **Remove Case** wird der oberste gelöscht. **Remove Case Structure** löscht die ganze "Case"-Struktur.

Dialogboxen

Einfache Dialogboxen zum Darstellen von Meldungen befinden sich in der **Functions** Palette unter **Time & Dialog**. Der **One Button Dialog** erscheint und bleibt offen bis "OK" gedrückt wird, der **Two Button Dialog** bleibt offen bis entweder "OK" oder "Cancel" gewählt wird.

6.4 Die Sequenzstruktur

Bei zeilenorientierten Programmiersprachen wird der Ablauf eines Programmes durch die Textsequenz bestimmt. Da LabVIEW eine datenflussorientierte Programmiersprache ist, wo Aktivitäten quasi parallel laufen können, gibt es die Sequenzstruktur *sequence structure,* die LabVIEW zu einem streng sequenziellen Ablauf zwingt, was in gewissen Fällen notwendig sein kann. Wie in Figur 6.6 abgebildet ist, hat die Sequenzstruktur die Form eines Filmbildes. Wie die anderen Strukturen, befindet sich auch die *sequence structure* in der **Functions** Palette unter **Struct & Constants**. Analog zur "Case"-Struktur ist nur ein Rahmen sichtbar. Die einzelnen Rahmen sind nummeriert; es ist diese Nummerierung, welche die Abfolge bestimmt. (erster Rahmen 0, zweiter 1, dritter 2 usw.) Wie bei der "Case"-Struktur können über das *pop-up* Menü Rahmen vor oder nach dem aktuellen angefügt werden.
Im Unterschied zur "Case"-Struktur, wo von allen *cases* eine Ausgangsvariable des selben Types verlangt wird, können Daten bei der Sequenzstruktur eines beliebigen Rahmens die Struktur durch *tunnels* verlassen. Man soll sich aber bewusst sein, dass die Daten erst nach Beenden der Sequenzstruktur nach aussen

gelangen. Eingangsdaten von ausserhalb werden für jeden Rahmen durch *tunnels* zugeführt.

Lokale Variablen in Sequenzen

Um Daten von einem Rahmen in andere folgende zu bringen, wird der *terminal* "lokale Sequenzvariable" *(sequence local)* benutzt. Eine solche *sequence local* wird mit der **Add Sequence Local** Option vom *pop-up* Menü des Sequenzstrukturrandes erzeugt. Mit **Remove** kann ein *sequence local terminal* wieder gelöscht werden.

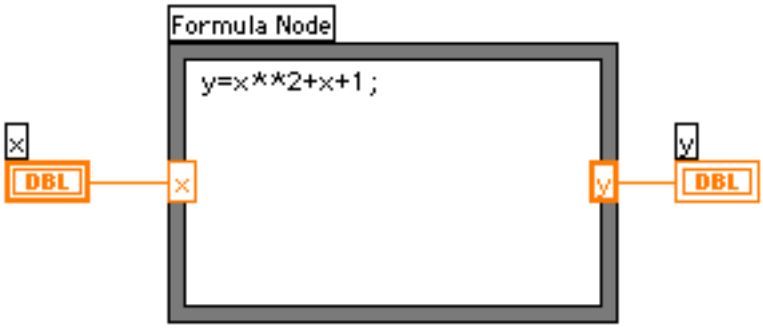
Die Sequenzstruktur in LabVIEW ist umstritten, es wird sogar diskutiert, ob sie ganz weggelassen werden soll. In der Tat ist es nicht sehr übersichtlich, wenn zuviele Teildiagramme in einer Sequenzstruktur versteckt werden. Besser ist es, eine Programmstruktur durch verbundene *subVIs* oder das Aneinanderreihen von "While"-Schleifen ohne Abbruchbedingungen zu erreichen.

Timing

Manchmal ist es nützlich, den zeitlichen Ablauf eines VI's zu steuern oder zu kontrollieren. **Wait (ms)** und **Tick Count (ms)**, die sich in der **Functions** Palette unter **Time & Dialog** befinden, dienen diesem Zweck. **Wait (ms)** veranlasst ein VI für die angegebene Anzahl Millisekunden zu warten, bevor es mit dem Programm weiterfährt. Dies kann praktisch sein, wenn es darum geht, einen *loop* z.B. einmal pro Sekunde auszuführen. **Tick Count (ms)** gibt den Stand der internen Uhr in Millisekunden an. Die interne Uhr besitzt aber keine sehr gute Zeitauflösung. Ein "Tick" dieser Uhr bedeutet auf dem Macintosh ca. 17 ms, bei WIN-PCs ca. 55 ms unabhängig von der Taktrate des PCs. Die Uhr kann unter anderem dazu gebraucht werden, Programmlaufzeiten zu messen.

6.5 Der Formelknoten *(Formula Node)*

Der Formelknoten *Formula Node* ist eine Box, mit deren Hilfe komplizierte algebraische Formeln in ein Diagramm integriert werden können. Eine Abbildung ist in Figur 6.7 gezeigt. Die Struktur ist sehr bequem, wenn komplizierte Formeln berechnet werden sollen. Z.B. würde schon der Ausdruck $y = x^2 + x + 1$ sehr unübersichtlich, wenn dazu die arithmetischen LabVIEW Funktionen eingesetzt würden. Die Formel wird innerhalb des Rahmens des *Formula Nodes* als Text geschrieben. Die Ein- und Ausgangs *terminals* werden mit dem *pop-up* Menü mittels **Add Input** oder **Add Output** definiert. Die Namen der Variablen sind auf zwei Buchstaben limitiert. Jede Formel muss mit einem Strichpunkt abgeschlossen werden.



Figur 6.7: Der *Formula Node*

Der *Formula Node* befindet sich in der *Functions* Palette zusammen mit den bereits erwähnten Strukturen. Die Operatoren und Funktionen, welche innerhalb des *Formula Nodes* gültig sind, findet man im *Function Reference* Band der LabVIEW-Anleitung.

Kapitel 7

Matrix und Verbund

In diesem Kapitel werden die beiden wichtigen Datentypen die Matrix *(array)* und der Verbund *(cluster)* behandelt. Diese zusammengesetzten Datentypen erlauben komplexe Manipulationen von Daten und deren Speicherung. Auch die LabVIEW Funktionen zur Manipulation dieser Datentypen werden vorgestellt.

Themen im 7. Kapitel :

- *array* (Matrix)
- *cluster* (LabVIEW-eigener Verbundtyp)
- *auto-indexing* (automatisches Indizieren)
- *polymorphism* (Datenanpassung)
- *to bundle* (bündeln)
- *to unbundle* (auffächern)

7.1 Was ist eine Matrix ?

Eine Matrix *(array)* ist eine Sammlung von Daten des selben Datentyps. Ein *array* kann ein- oder mehrdimensional sein und bis 2^{31} Elemente pro Dimension enthalten, falls das der Arbeitsspeicher des Rechners zulässt. *Arrays* in LabVIEW können alle Datentypen, ausgenommen *array, chart* oder *graph*, aufnehmen. Jedes Element wird mit seinem Index adressiert. Der Index nimmt die Werte von 0 bis n-1 an, wobei n die Anzahl Elemente im *array* ist. Das erste Element besitzt den Index 0, das zweite den Index 1, usw.

Messdaten werden meistens in *arrays* gespeichert, jedes Element entspricht dann einer Messung. Bei Mehrkanalmessungen mit einer DAQ-Karte werden die Messungen in zweidimensionalen *arrays* abgelegt. Jede Spalte einer Matrix enthält die Messpunkte eines der abgetasteten Kanäle.

7.2 Erzeugen einer Matrix *(array)*

Um auf dem *front panel* komplexe Datentypen wie *arrays* oder *clusters* zu erzeugen, sind zwei Schritte nötig. Ein *array control* oder *indicator* wird durch die Kombination einer *array shell* (Matrixschale) mit dem gewünschten Datenobjekt (*numeric, Boolean* oder *string*) erreicht. Die *array shell* befindet sich in der **Array & Graph** Palette des **Controls** Menü.

Erzeugt wird ein *array* durch Schieben oder Einfügen eines Datenobjektes *(control* oder *indicator)* in die Elementanzeige *(element display)* einer *array shell.* Die Elementanzeige kann vergrössert werden um mehr Elemente sichtbar zu machen, dazu wird mit dem Positionierwerkzeug *(positioning tool)* die Elementanzeige vergrössert. Das kleine Fenster links oben ist die Indexanzeige *(index display)*, dort können die Elemente angewählt werden. Das Element neben dem *index display* entspricht dem angezeigten Index.

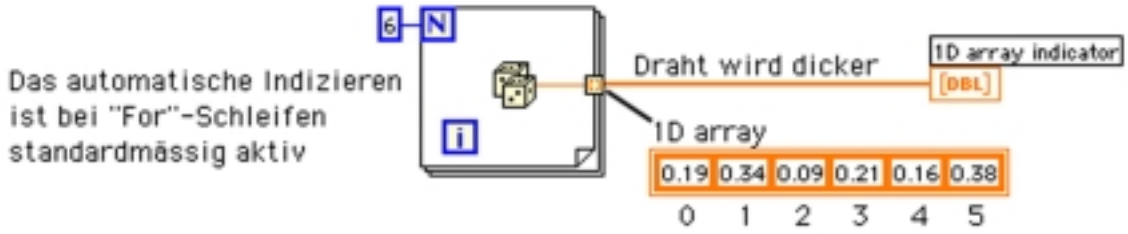
Eine leere *array shell* erzeugt auf dem Blockdiagramm einen schwarzen *terminal*, ein kleines Rechteck das zwei eckige Klammern umschliesst. Schwarz bedeutet, dass die *array shell* kein Datenobjekt enthält, d.h. es wurde noch kein Datentyp zugeordnet. Nachdem ein *control* oder *indicator* in der Elementanzeige platziert wurde, erscheint zwischen den eckigen Klammern das Symbol des Datentyps, und der *array terminal* nimmt die Farbe des Datentyps an.

Zweidimensionale Matrix

Ein zweidimensionaler *array* braucht zwei Indizes, um seine Elemente, die in Spalten und Zeilen angeordnet sind, zu lokalisieren. Eine weitere Dimension kann dem *array* mit dem *pop-up* Menü **Add Dimension** des *array controls* oder *indicator* hinzugefügt werden.

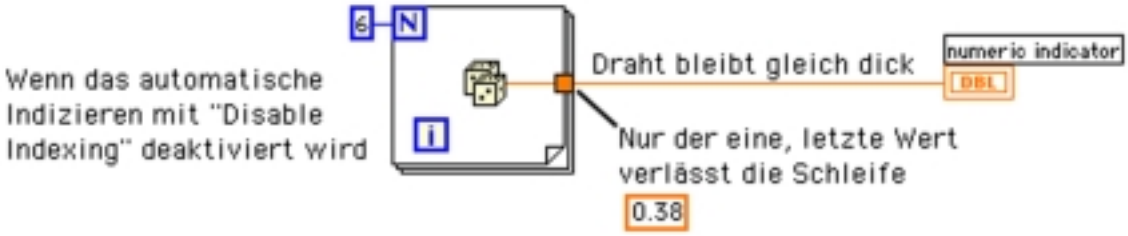
Erzeugen einer Matrix durch automatisches Indizieren

Eine mächtige Eigenschaft von LabVIEW ist die Fähigkeit, beim Eintreten in oder Austreten aus einer Schleife Matrizen automatisch zu indizieren oder zu erzeugen. Die "For"- und die "While"-Schleife besitzen die Möglichkeit, an ihren Grenzen durch automatisches Indizieren *arrays* aufzubauen. Dieser Prozess wird *auto-indexing* genannt. Die Figur 7.1 zeigt das Prinzip: Jede Iteration indiziert das nächste *array* Element, bis die Schleife abgearbeitet ist und das ganze *array* die Schleife verlässt.



Figur 7.1: Automatisches Indizieren aktiv

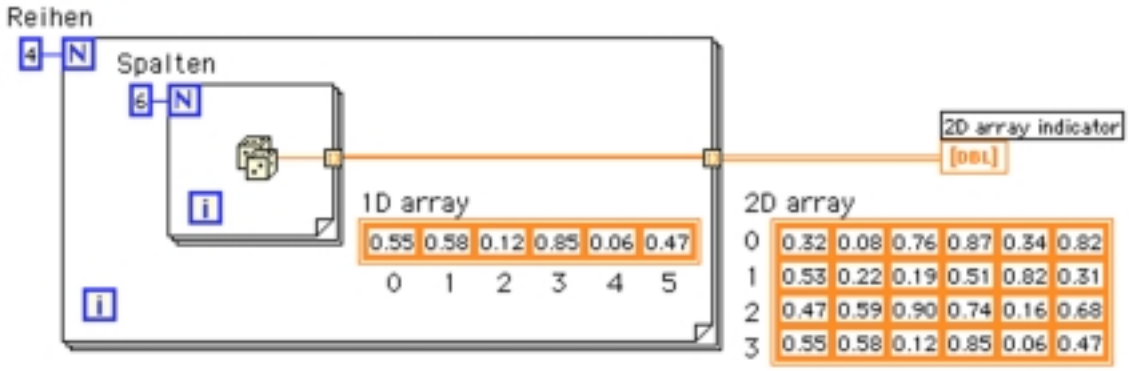
Wenn ein Wert aus der Schleife geführt werden soll, ohne ein *array* zu erzeugen, muss man, wie dies in der Figur 7.2 gezeigt ist, mit **Disable Indexing** im *pop-up* Menü die automatische Indizierung desaktivieren.



Figur 7.2: Automatisches Indizieren deaktiviert

Auto-indexing ist ebenfalls aktiv, wenn ein *array* einer Schleife zugeführt wird. Dann wird das eintreffende *array* elementweise abgearbeitet, der Schleifenzähler entspricht in diesem Fall automatisch der Anzahl Elemente in diesem *array*. Soll ein *array* als Ganzes in die Schleife gelangen, muss **Disable Indexing** gewählt werden.

Achtung: Da "For"-Schleifen oft zum Indizieren von *arrays* benutzt werden, aktiviert LabVIEW *auto-indexing*, wenn ein array an die Schleife "angeschlossen" wird. Hingegen ist bei "While"-Schleifen der *auto-indexing* Modus standardmässig nicht aktiv.



Figur 7.3: Aufbauen einer 2D Matrix

Erzeugen einer 2D Matrix

Zum Aufbauen von zweidimensionalen *arrays* werden zwei, ineinandergeschachtelte "For"-Schleifen gebraucht. Dies ist in Figur 7.3 illustriert. Die innere "For"-Schleife erzeugt die Reihen, die äussere stapelt die Reihen zu Spalten.

7.3 Funktionen zur Matrixmanipulation

LabVIEW kennt viele Funktionen zum Arbeiten mit *arrays.* Hier seien einige grundlegende erklärt:

Initialize Array: Erzeugt und initialisiert ein n-dimensionales *array.* Die Funktion ist bequem zum Zuweisen von Speicherplatz für *arrays* bekannter Grössen.

Array Size: Gibt die Anzahl Elemente eines *arrays* an. Im Fall eines zweidimensionalen *arrays* als Eingang der Funktion, gibt **Array Size** ein eindimensionales *array* mit n-Elementen zurück, jedes Element entspricht der Anzahl Elemente einer Dimension.

Build Array: Kombiniert zwei *arrays* oder fügt einem bestehenden *array* neue Elemente hinzu. Wenn die Funktion auf das Blockdiagramm gebracht wird, besitzt sie einen Ein- und einen Ausgang. Der Block kann aber mit dem Positionierwerkzeug vergrössert werden, um mehrere Eingänge zur Verfügung zu haben. **Build Array** besitzt zwei Typen Eingänge, solche für *arrays* und solche für einzelne Elemente.

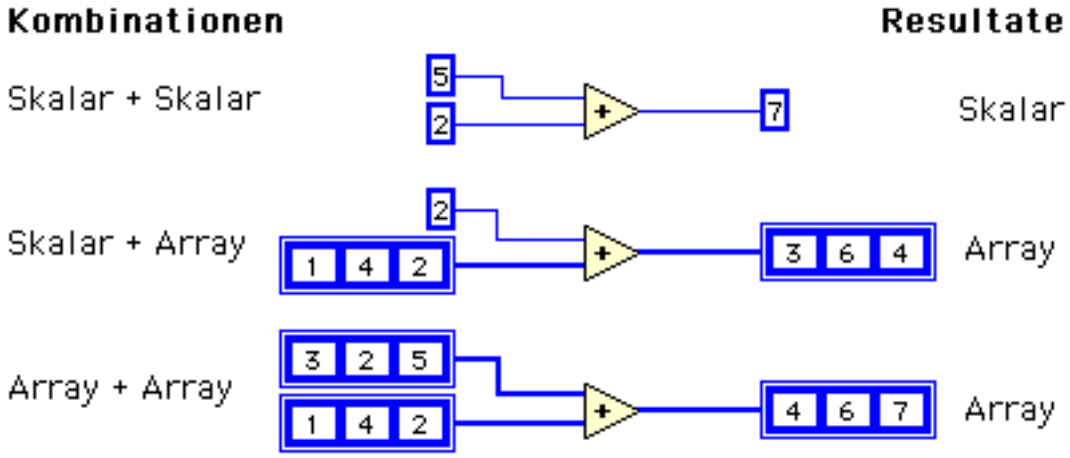
> **Achtung:** Beachte die Eingänge einer **Build Array** Funktion. *Array* Eingänge zeigen kleine eckige Klammern, Eingänge für Elemente aber nicht. Sie dürfen nicht zu verwechselt werden!

Array Subset: Extrahiert eine Teilmatrix ab dem Element mit Index *index* und der Länge *length.*

Index Array: Nimmt ein bestimmtes Element aus einem *array* heraus. Diese Funktion kann auch dazu gebraucht werden, ein zweidimensionales *array* in zwei eindimensionale aufzuspalten. Dazu muss die Funktion auf zwei Eingänge vergrössert werden, wobei mittels *pop-up* Menü einer der beiden auf **Disable Indexing** geschaltet wird um zu verhindern. dass nur das Element in der Spalte n und der Reihe m ausgewählt wird. Beachte die Veränderung des *terminal* Symbols von einem gefüllten zu einem leeren Rechtecklein.

7.4 Datenanpassung

Die arithmetischen Funktionen von LabVIEW, wie **Add** (Addition), **Multiply** (Multiplikation), **Divide** (Division) usw. passen sich automatisch dem angeschlossenen Datentyp an. Diese Eigenschaft heisst *polymorphism*. Z.B. kann ein Skalar zu einem *array* oder zwei *arrays* zueinander addiert werden. Figur 7.4 zeigt drei solche Kombinationen und ihre Resultate.



Figur 7.4: *Polymorphism:* Beispiel Addition

Beachte: Bei arithmetischen Operationen von zwei *arrays* verschiedener Länge erhält das resultierende *array* die Länge des kürzeren *arrays* am Eingang, d.h. die Operation wird nur solange ausgeführt, wie korrespondierende Elemente vorhanden sind, die übrigbleibenden werden ignoriert.

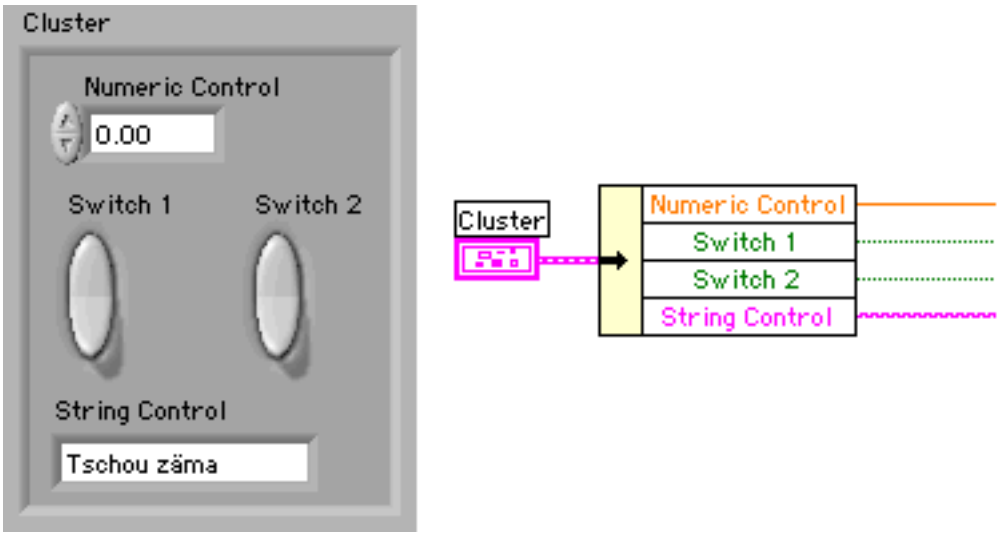
7.5 Der LabVIEW Verbund-Datentyp

Ein Verbund *(cluster)* ist wie das *array* eine Struktur, welche Daten gruppiert. Im Gegensatz zum *array* kann aber ein *cluster* unterschiedliche Datentypen aufnehmen, analog zu einem *record* in Pascal oder *struct* in C. Ein *cluster* kann in LabVIEW als Drahtbündel ähnlich einem Telefonkabel betrachtet werden; jeder

Draht im Kabel repräsentiert ein Element eines Datentyps. Da mit *cluster* unterschiedliche Drähte zusammengefasst werden können, ist dieser Datentyp sehr nützlich zum Reduzieren des Drahtgewirrs auf einem komplizierteren Blockdiagramm. Der *cluster* erscheint oft im Zusammenhang mit LabView-Grafiken. Sowohl *cluster* wie auch *array* Elemente liegen geordnet vor. *Cluster*-Elemente können jedoch im Gegensatz zur sequenziellen Indizierung bei *arrays.* durch sog. Auffächern *(unbundle)* gleichzeitig freigelegt werden. Der *unbundle* Vorgang kann analog zum Aufspleissen eines Telefonkabels betrachtet werden.
Cluster terminals lassen sich nur mit einem andern *cluster* verbinden, wenn sie in der Anzahl Elemente und deren Datentypen übereinstimmen. Allerdings gelten auch hier die Regeln des *polymorphism,* solange es sich um kompatible Datentypen handelt.

7.6 Erzeugen eines Verbunds (Clusters)

Ähnlich dem Erzeugen von *arrays*, werden *cluster controls* und *cluster indicators* durch Auswählen einer *cluster shell* aus der **Array & Graph** Palette des **Controls** Menüs und dem Hineinbringen der gewünschten Objekte konstruiert. Alle auf dem *front panel* platzierbaren Objekte können als *cluster*-Elemente zusammengefasst werden, allerdings müssen alle entweder *controls* oder *indicators* sein, eine Kombination der beiden ist ungültig. Figur 7.5 zeigt einen *cluster* auf dem *front panel* von vier *controls.*



Figur 7.5: *Cluster* mit vier *control* Objekten und dem Blockdiagramm-Terminal

Cluster Hierarchie

Cluster Elemente besitzen eine logische Ordnung, entsprechend der Reihenfolge ihrer Platzierung in der *cluster shell.* Das erstplatzierte Element ist das Element

0, das Zweitplatzierte 1 usw. Wenn ein Element gelöscht wird, passt sich die Reihenfolge automatisch an. Mit dem *pop-up* Menü am Rand des *clusters* kann durch Anwählen von **Cluster Order** die Ordnungszahl eines *cluster* Elements manuell mit dem *cluster order cursor* (Händchen mit Zahlsymbol) eingestellt werden.

Datentransfer mit *clusters* zu einem *subVI*

Der *connector* eines VIs kann höchstens 20 *terminals* aufnehmen. Es sollten aber nie alle gebraucht werden, da sonst ein schrecklicher Drahtknäuel entsteht. Durch Bündeln kann eine ganze Anzahl *controls* durch einen *terminal* dem *subVI* als Eingänge übergeben werden. Das gleiche gilt auch für das Zusammenfassen der Ausgänge. Diese Technik führt zu übersichtlicheren Diagrammen.

Bündeln von Daten

Die **Bundle** Funktion im **Array & Cluster** Menü fügt Komponenten zu einem *cluster* zusammen. Die Funktion kann vergrössert werden, um mehr Elemente zu bündeln. Die Reihenfolge der Elemente ist durch ihre Anordnung von oben nach unten gegeben.

Ersetzen von *cluster* Elementen

Bundle erlaubt auch ein Auswechseln einzelner *cluster*-Elemente. Dazu muss die **Bundle** Funktion soweit vergrössert werden, dass sie der Anzahl Elemente des betreffenden *clusters* entspricht. Danach wird der *cluster wire* am *terminal* im Zentrum der Funktion angeschlossen. Neue Elementwerte können nun in die **Bundle**-Eingänge gespiesen werden.

Auffächern eines *clusters*

Die **Unbundle** Funktion im **Array & Cluster** Menü spaltet einen *cluster* in seine Elemente. Auch hier wird die Reihenfolge durch die Anordnung von oben nach unten angezeigt. Die Grösse der **Unbundle** Funktion muss auch hier mit der Anzahl Elemente im *cluster* übereinstimmen.

Kapitel 8

Grafische Anzeigen

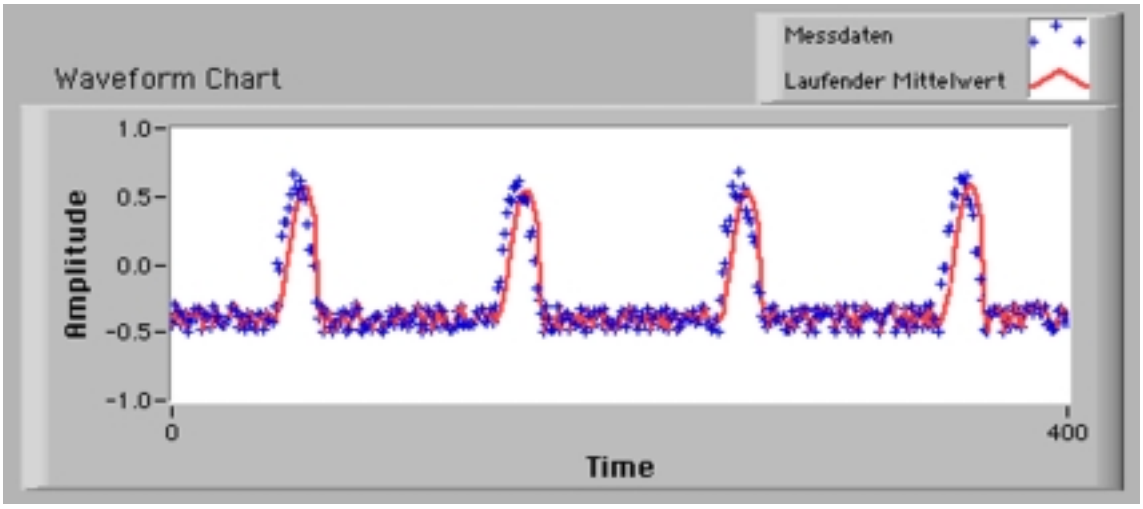
LabVIEW bietet sehr mächtige und individuell konfigurierbare Grafikanzeigen. Vom Benutzer interaktiv bedienbare Grafikanzeigen tragen Kurvenverläufe auf, neu ankommende Daten werden dabei laufend an die alten angefügt, so dass die neuen Datenwerte mit früheren verglichen werden können. Grafiken können eine grosse Anzahl Datenpunkte darstellen und geeignet skaliert aufzeichnen. Dieses Kapitel behandelt die Grafikfähigkeiten von LabVIEW.

Themen im 8. Kapitel :

- *waveform chart* (Datenschreiber)
- *waveform graph* (Grafik)
- *XY graph* (XY-Grafiken)
- *strip mode* (Linienschreiber-Modus)
- *sweep mode* (Sweep-Modus)
- *scope mode* (Oszilloskop-Modus)
- *legend* (Legende)
- *palette* (Grafikpalette)
- *mechanical action* (Mechanische Funktion von Schaltern)
- *latch action*
- *switch action* (Schaltfunktion)

8.1 Grafische Anzeigen *(waveform chart)*

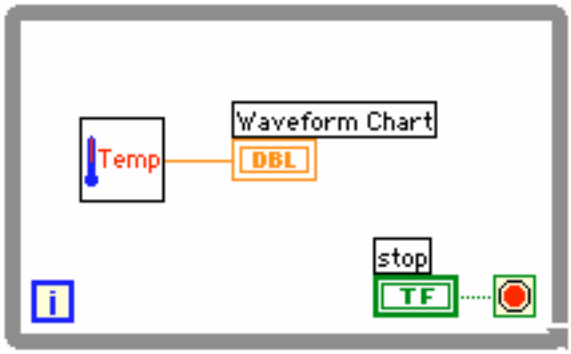
LabVIEW besitzt drei grafische Anzeigen, die auf unterschiedliche Arten Daten darstellen können. Eine davon ist die *waveform chart*, welche wie die anderen Graphikanzeigen unter **Graph** in der **Controls** Palette gefunden werden. Eine *waveform chart* ist ein spezieller *numeric indicator*, der eine oder mehrere Kurven anzeigen kann. Figur 8.1 zeigt ein *waveform chart* mit zwei Kurven.



Figur 8.1: *Waveform chart*, Anzeige mit zwei Kurven

Die *waveform chart* Anzeigearten

Die *waveform chart* verfügt über drei Anzeigearten: *strip chart* (Linienschreibermodus), *scope chart* (Oszilloskopmodus) und *sweep chart* ("Sweep"-Modus). Die drei Arten können mit dem *pop-up* Menü der *waveform chart* unter **Data Operations ▸ Update Mode** ausgewählt werden.



Figur 8.2: Einfache *waveform chart* mit einer Kurve

Die *strip chart* besitzt eine "rollende" Anzeige, ähnlich einem Papierstreifen eines Linienschreibers. Die *scope chart* und *sweep chart*-Arten zeichnen die

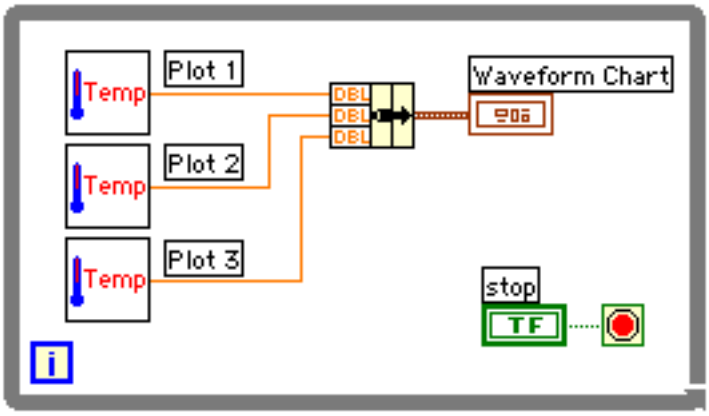
Kurven repetitiv, ähnlich einem Oszilloskop. Da die beiden letzten nicht sehr rechenintensiv arbeiten, sind sie schneller als die *strip chart.*

waveform chart mit einer Kurve

Ein skalarer Ausgang kann direkt mit einer *waveform chart* verbunden werden, wie dies in Figur 8.2 dargestellt ist.

Verdrahten einer *waveform chart* für mehrere Kurven

Um Anzeigen für mehrere Kanäle zu erhalten, müssen die Daten vor Eintritt in die *waveform chart* gebündelt werden, wie das in Figur 8.3 ersichtlich ist. Wenn ein *cluster* an eine *waveform chart* angeschlossen wird, verändert sich die Form und Farbe des *chart terminals* auf dem Diagramm entsprechend.



Figur 8.3: *Waveform chart* mit drei Kurven

Anfügen einer Digitalanzeige

Wie andere numerische Anzeigen, besitzt die *waveform chart* die Option, zusätzlich eine Digitalanzeige zu setzen, die den Momentanwert der Kurve anzeigt. Man aktiviert sie mit der **Show** Option im *pop-up* Menü der *waveform chart.*

Der Rollbalken

Waveform charts besitzen einen Rollbalken, der im *pop-up* Menü aktiviert werden kann. Der Rollbalken holt die Datenpunkte, die bereits den Rand der Anzeige überschritten haben, wieder ins sichtbare Fenster.

Löschen einer Grafik

Um eine Grafikanzeige zu löschen, kann dies mit **Clear Chart** im **Data Operations** *pop-up* Menü geschehen. Es kann aber auch der **Clear** Knopf der *chart* Palette benutzt werden.

Gestapelte oder übereinanderliegende Kurven

Wenn mehrere Kurven gezeigt werden, besteht die Wahl zwischen gestapelten oder übereinanderliegenden Kurven. Bei der gestapelten Variante können die Kurven unabhängig voneinander skaliert werden. Die Wahl geschieht mit **Stack Plots** (gestapelte Kurven) oder **Overlay Plots** (übereinanderliegende Kurven) im *pop-up* Menü der *chart.*

Grösse des *waveform chart buffers*

Der Standardwert (*default*-Wert) der Anzahl darstellbarer Punkte ist 1024. Falls mehr oder weniger Punkte aufgezeichnet werden sollen, soll dieser Wert mit **Chart History Length...** angepasst werden, die Punktezahl ist auf 100'000 beschränkt.

8.2 Mechanische Funktionen von Schaltern

Booleans als Schalter oder Knöpfe können auf ihr Ansprechen modifiziert werden. LabVIEW stellt dazu sechs Arten mechanischer Funktionen zur Verfügung. Die Modifikationen werden unter **Mechanical Action** im *pop-up* Menü der Schalter gefunden. Nach der Wahl einer gewünschten Option empfiehlt es sich, diese mit **Data Operations ▸ Make Current Value Default** zu sichern. Es sind folgende Schaltoptionen vorhanden:

Switch When Pressed ändert den *control* Wert jedesmal, wenn mit dem *Operating tool* darauf geklickt wird. Der Schalter simuliert einen Lichtschalter. Er wird durch das Lesen seines Status nicht beeinflusst.

Switch When Released ändert den *control* Wert erst, nachdem der Mausknopf im Bereich des Schalters losgelassen wird. Der Modus ist ähnlich einem *check button* eines Dialogfensters. Auch diese Option wird durch das Lesen des Status nicht beeinflusst.

Switch Until Released ändert seinen *control* Wert, wenn auf den Schalter geklickt wird und hält diesen Wert, bis der Mausknopf wieder losgelassen wird, dann kehrt er zurück zum Originalwert. Der Schalter simuliert einen Klingelknopf. Er wird nicht beeinflusst durch das Lesen des Status.

Latch When Pressed ändert seinen Wert beim Klicken auf den Schaltbereich. Der Schalter hält seinen Wert, bis er vom VI gelesen wird, in diesem Moment kehrt er zum Originalwert zurück. (Dieser Vorgang ist unabhängig davon, ob weiter auf den Schalter gedrückt wird oder nicht.) Dieser Modus simuliert einen Schutzschalter und kann zum Anhalten einer "While"-Schleife oder zum Auslösen einer Funktion auf Knopfdruck gebraucht werden.

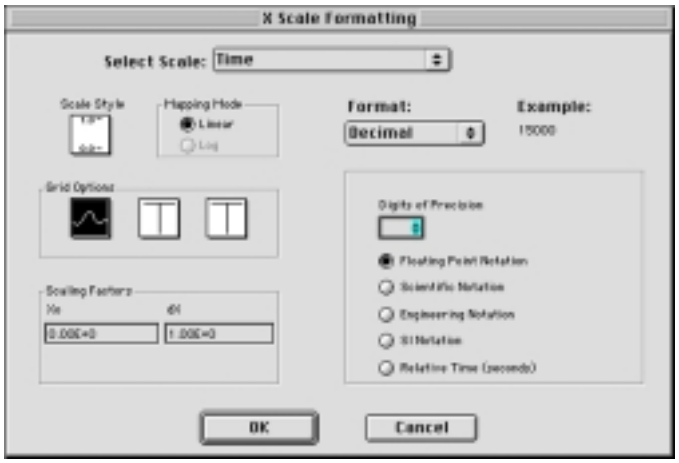
Latch When Released ändert seinen Wert erst, nachdem der Mausknopf losgelassen wird. Nachdem das VI einmal den Wert gelesen hat, nimmt der Schalter seinen Originalzustand wieder ein. Diese Option garantiert mindestens einen neuen Wert. Wie **Switch When Released** ist dieser Modus einem *check buttons* einer Dialogbox ähnlich.

Latch Until Released ändert den *control* Wert beim Klicken auf den Schalter. Er behält diesen Wert, bis er einmal vom VI gelesen oder bis der Mausknopf losgelassen wird.

8.3 Komponenten grafischer Anzeigen

Charts und *graphs* besitzen eine Reihe praktischer Zusatzkomponenten, um die Ausgabe von Daten zu gestalten.

Skalieren von grafischen Anzeigen



Figur 8.4: Die **Formatting...** Dialogbox

Charts und *graphs* skalieren standardmässig ihre Vertikal- und Horizontalachsen automatisch. Dieser automatische Modus *(autoscaling)* kann deaktiviert werden, indem die Optionen **Autoscale X** und **Autoscale Y** im **Data Operations** Menü ausgeschaltet werden. Dasselbe kann auch in den **X Scale** oder **Y Scale** Untermenüs des *pop-up* Menüs getan werden. Diese *autoscaling* Möglichkeiten

können auch über die Palette der Anzeigen vom Benutzer gesteuert werden. *Autoscaling* kann je nach Rechnertyp oder Videosystem die grafischen Anzeigen verlangsamen.
Wenn der *autoscaling* Modus nicht gefragt ist, können die Vertikal- und Horizontalachsen direkt mit dem *Operating* oder *Labeling tool* skaliert werden, wie das mit jedem anderen LabVIEW *control* oder *indicator* möglich ist. LabVIEW passt die Dichte der darzustellenden Punkte automatisch der Anzeige an. Die X- und die Y-Achsen haben ihre eigenen SubMenü Optionen:
Mit **AutoScale** wird das *autoscaling* ein- und ausgeschaltet. Normalerweise werden im *autoscaling* Modus die Skalen genau der Anzahl Datenpunkte angepasst. Mit der **Loose Fit** Option wird die Skala durch Runden auf einen vernünftigen Skalenwert gebracht.
Formatting... bringt eine Dialogbox zum Vorschein, die in Figur 8.4 abgebildet ist. Mit ihr lassen sich folgende Eigenschaften einstellen:

Scale Style lässt das Verändern der Skalenmarken zu, auch können mit dieser Option die Skalenanzeigen zum Verschwinden gebracht werden.

Mapping Mode ändert die Skala von linear zu logarithmisch.

Grid Options wählt verschiedene Rastermuster und deren Farbe.

Xo Hier kann der Anfangswert der gewünschten Skala gesetzt werden.

dX definiert den Abstand der Inkremente.

Format & Precision lässt eine Wahl der Anzahl darzustellender Dezimalwerte der Skalenbeschriftung zu; ferner kann die Notation von **Floating Point** (Dezimal) auf **Scientific** (Potenzdarstellung) umgeschaltet werden.

Die Legende

Mit dem **Show** Untermenü der *chart* wird eine Legende für die dargestellten Kurven angezeigt. Die Legenden-Anzeige lässt sich für mehrere Kurven nach unten vergrössern. Jede Kurve kann mit einem Namen versehen werden.
Im *pop-up* Menü der Legende lassen sich Parameter wie **Point Style** (Punkteform) und **Line Style** (Darstellung der Kurven), **Interpolation** (Interpolation) und **Color** (Farbe) der Kurven verändern.

Benutzen der *graph* Palette

Auch die Palette kann mit **Show** vom *chart* oder *graph pop-up* Menü gezeigt werden. Mit ihrer Hilfe lassen sich Grafiken während der Laufzeit des Programms auf vielfältige Art und Weise verändern und manipulieren. Am Besten findet man die Funktionen durch Experimentieren selber heraus. Mit der Palette

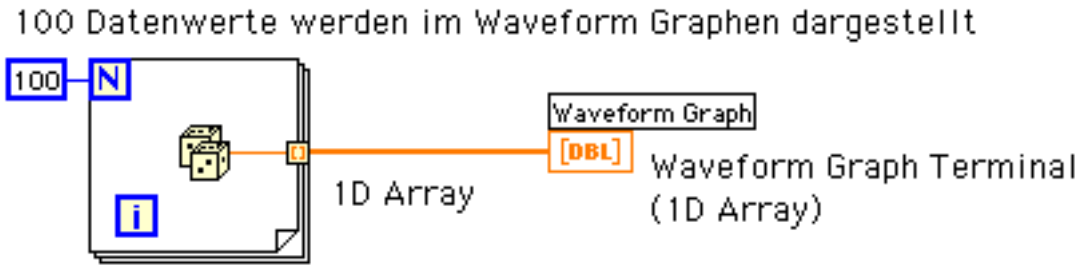
kann durch Manipulationen mit der Maus das *autoscaling* für beide Achsen unabhängig ein- und ausgeschaltet oder blockiert werden. Ebenfalls können das Format und die Präzision der Achsenbeschriftung eingestellt, verändert, auf verschiedene Arten "gezoomt", und mit dem Händchen die Grafik verschoben werden.

8.4 Grafiken

Im Unterschied zur *waveform chart* stellen die *waveform graphs* (Grafiken) ganze Datensets, d.h. viele Punkte, auf einmal dar. LabVIEW stellt neben den beiden Spezialtypen *intensity chart* und *intensity graph* für 3D-Datensätze zwei weitere grundlegende Arten von Grafikanzeigen zur Verfügung, den *waveform graph* und den *XY graph.* Beide Typen sehen auf dem *front panel* gleich aus, funktionieren aber sehr unterschiedlich. Beide Grafiktypen findet man unter **Graph** in der *Controls* Palette. Der *waveform graph* stellt Funktionen variabler Daten mit äquidistanter Punkteverteilung auf der X-Achse dar. Der *waveform graph* ist ideal zur Darstellung von Messkurven, die in gleichmässigen Zeitabständen abgetastet wurden. Der *XY graph* eignet sich zur Darstellung ganzer *arrays* kartesischer Punkte mit zwei Koordinaten. Die beiden Grafiktypen verlangen unterschiedliche Datentypen. Diesen ist deshalb spezielle Aufmerksamkeit zu schenken.

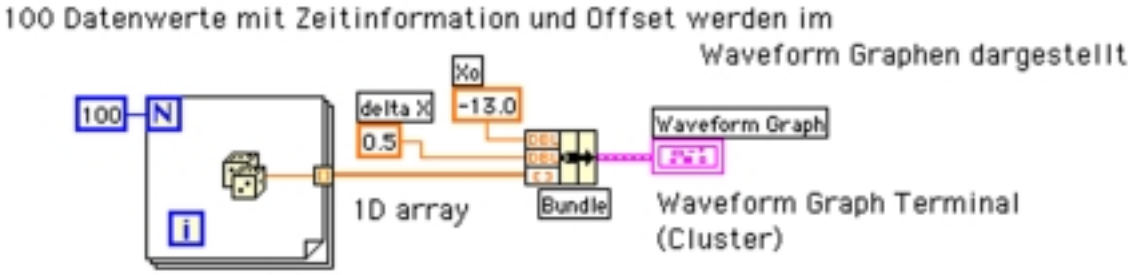
Waveform graph mit einer Kurve

Ein *array* von Y-Werten kann, wie dies in der Figur 8.5 gezeigt ist, direkt in einen *waveform graph* gebracht werden. Diese Methode geht davon aus, dass der Initialwert in $X_0 = 0$ und das $\Delta X = 1$. Einmal verdrahtet, erscheint das *graph terminal* Kästchen als *array indicator.*



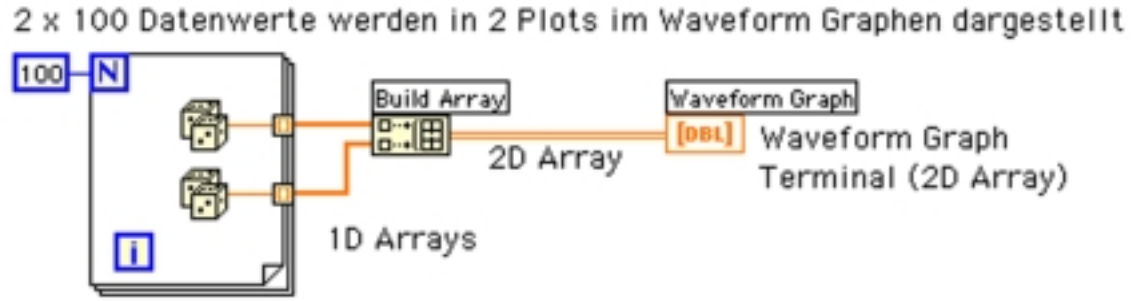
Figur 8.5: *Waveform graph* aus einem *array* von Y-Werten

Soll nun die Zeitskala geändert werden oder soll $X_0 \neq 0$ sein, kann der *waveform graph* mit einem *cluster*, der zusätzlich das ΔX und den X_0-Wert enthält, angesteuert werden. Dies wird durch die in Figur 8.6 gezeigte Konstruktion erreicht. In diesem Fall wird der *graph terminal* zu einem *cluster terminal.*



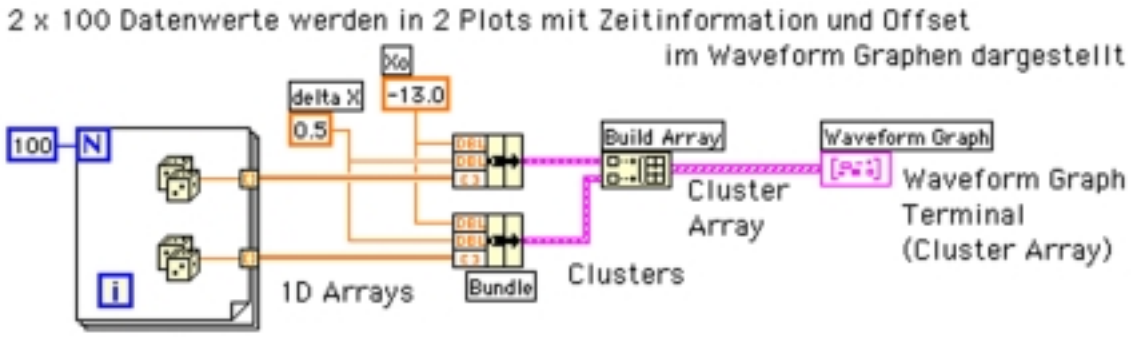
Figur 8.6: *Waveform graph* mit *bundle* für X_0 und ΔX

Waveform graph mit mehreren Kurven



Figur 8.7: *Waveform graph* aus zwei *arrays* von Y-Werten

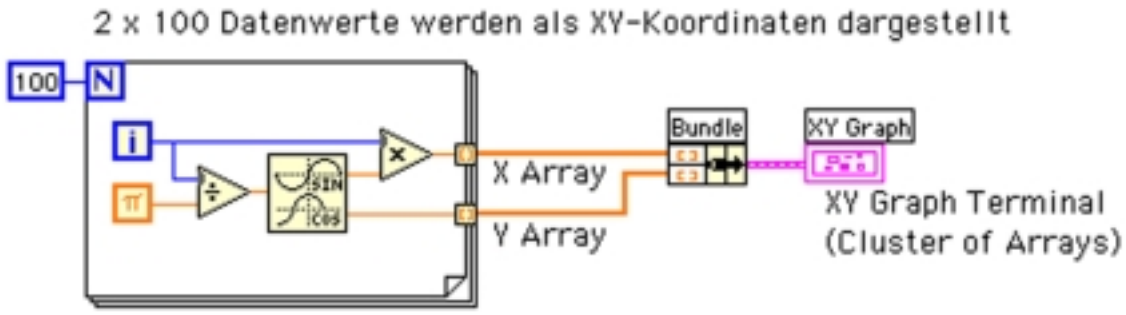
Wie weitere Kurven im gleichen *waveform graph* dargestellt werden, zeigt Figur 8.7. Zwei *arrays* lassen sich mit einer **build array** Funktion zu einem zweidimensionalen *array* zusammenführen. Auch hier passt sich der *graph terminal* dem erhaltenen Datentyp an. Wenn bei einem *waveform graph* mehrere



Figur 8.8: *Waveform graph* für zwei Kurven mit *bundle* für X_0 und ΔX

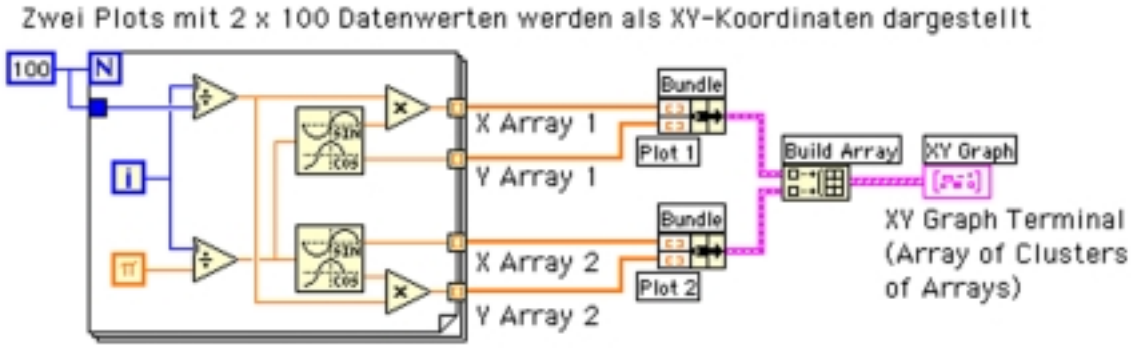
Kurven mit $\Delta X \neq 1$ und $X_0 \neq 0$ vorliegen, werden diese Werte mit *bundle* in die Daten-*clusters* gebracht und mit *build array* zu einem *cluster array* zusammengeführt. In dieser Form zeigt der *waveform graph* die Kurven entsprechend skaliert an. Dieses Konstrukt ist in Figur 8.8 zu sehen.

XY graph für eine und mehrere Kurven



Figur 8.9: *XY graph* für eine XY-Kurve

XY graphs können Kurven mit Wertepaaren in (X, Y) darstellen. Soll nur eine solche XY Kurve gezeichnet werden, wird das gemacht wie in Figur 8.9 gezeigt. Zwei 1D *arrays* mit je den Werten X und Y werden durch die Funktion *bundle* in einen *cluster* gebracht und auf diese Art dem *XY graph* zugeführt.



Figur 8.10: *XY graph* mit zwei XY-Kurven

Sollen mehrere XY Kurven gezeichnet werden, geschieht das, wie in Figur 8.10 gezeigt, durch Zusammenfassen der beiden XY *clusters* in ein *cluster array.*

Kapitel 9

Zeichenketten und Speichern von Daten

Dieses Kapitel stellt einige Möglichkeiten zur Manipulation von Zeichenketten vor. Zeichenketten sind die Basis zum Programmieren von GPIB- und RS-232-Geräten. LabVIEW besitzt zahlreiche Zeichenkettenfunktionen ähnlich denjenigen für *arrays.* Es wird auch beschrieben, wie Daten gespeichert und wieder eingelesen werden können.

Themen im 9. Kapitel :

- Funktionen für *strings*
- *file I/O* Datenspeicherung
- *spreadsheet file*

9.1 Die Zeichenkettenfunktionen

Zeichenketten *(strings)* wurden bereits im Kapitel 4 kurz erwähnt. Ein *string* ist eine Sammlung von ASCII-Zeichen. Oft werden *strings* für weit mehr als für einfache Textmeldungen gebraucht. Zum Beispiel werden bei der Kommunikation mit GPIB auch numerische Daten als *strings* übermittelt; dabei müssen die *strings* in numerische Werte umgewandelt werden, bevor man sie auswerten kann. Um numerische Daten sicher so abzuspeichern, dass sie von andern Programmen gelesen werden können, sollten sie vor dem Speichern in *strings* umgewandelt werden.
Die folgende Liste der wichtigsten *string* Funktionen, zu finden unter **String** der *Functions* Palette, deckt einen Grossteil der Zeichenkettenoperationen ab:

String Length gibt die Anzahl Zeichen *(length)* einer Zeichenkette *(string)* aus.

Concatenate Strings reiht die Zeichenketten *(string0, string1..)* an den Eingängen zu einer langen Zeichenkette *(concatenation of string0, string1,...)* am Ausgang zusammen.

Format & Append formatiert eine numerische Zahl *(number)* in eine Zeichenkette *(output string)* nach Spezifikation des Format-Eingangs *(format string)* und hängt diese Zeichen hinten an eine bereits bestehende Zeichenkette *(string" ").*

Get Date/Time String gibt das Datum *(date string)* und die Zeit *(time string)* in Form zweier Zeichenketten aus. Die Eingänge können, wenn nichts Spezielles erforderlich ist, offen gelassen werden. Diese Funktion wird u.a. gebraucht, um Messdaten mit Zeitmarken zu versehen.

Zeichenkettenanalyse *(parsing functions)*

String Subset gibt eine Teilzeichenkette *(substring)* aus, die bei einem Offset *(offset)* beginnt und die Länge *length* hat.

Format & Strip sucht am Anfang einer Zeichenkette *(string)* ein gegebenes Muster *(format string)* und konvertiert in diesem Teilbereich vorhandene Zahlenzeichen nach einem vorgegebenen Format *(format string)* in ein numerisches Format. Die Funktion gibt die numerische Zahl *(number)* und die übrigbleibende Zeichenkette *(output string)* aus. Falls kein zutreffendes Muster gefunden wurde, gibt die Funktion einen vorgegebenen Wert *(default)* aus.

9.2 Datenspeicherung auf Disk

Unter Datenspeicherung *(file I/O)* versteht sich das Speichern und Laden von Daten auf oder von einem Speichermedium wie z.B. einer Harddisk. LabVIEW

bietet transparente Funktionen, die einfach zu gebrauchen sind und dem Programmierer die schwierigen Detailaspekte einer sicheren Datenspeicherung abnehmen. Diese Funktionen befinden sich unter **File** in der *Functions* Palette.

Wie arbeiten die *file I/O* Funktionen?

Die *file* Funktionen erwarten einen Pfad *(path)* am Eingang. Wenn kein Pfad angegeben wird, erzeugen die Funktionen eine Dialogbox, welche den Benutzer zur Angabe eines Dateinamens auffordert. Danach erzeugen oder öffnen die Funktionen eine Datei, lesen oder schreiben Daten und schliessen die Datei am Ende des Vorgangs. Im Fall einer einfachen Textdatei können die Daten nach dem Speichern mit praktisch jedem Textprogramm geöffnet und betrachtet werden.
Es ist auch sehr populär, Daten so zu speichern, dass sie von einem "Spreadsheet"-Programm wieder gelesen und analysiert werden können. Die meisten "Spreadsheet"-Programme verlangen die Daten geordnet in Spalten mit Tabulatoren getrennt und EOLs *(End of Line)* zum Separieren der Zeilen. **Write To Spreadsheet File** und **Read from Spreadsheet File** Funktionen erzeugen oder lesen solche Dateien.

Write Characters To File schreibt eine Zeichenkette *(character string)* in eine neue Datei oder fügt sie einer bestehenden an.

Read Characters From String liest die vorgeschriebene Anzahl Zeichen vom Anfang der Datei oder ab einem angegebenen Offset.

Read Lines From File liest die vorgeschriebene Anzahl Zeilen vom Anfang der Datei oder ab einer angegebenen Anzahl ASCII-Zeichen *(bytes)*.

Write To Spreadsheet File konvertiert 2D oder 1D *single-precision* Zahlen in eine Textzeichenkette und schreibt die Zeichenkette in ein neue oder eine existierende Datei. Wahlweise können die zu speichernden Datenarrays auch transponiert werden. Die beiden Eingänge für die 2D und 1D *arrays* können nicht gleichzeitig benutzt werden. (Einer der beiden wird ignoriert.) Die mit dieser Funktion generierten Textdateien können mit den meisten Spreadsheet-Programmen gelesen werden.

Read From Spreadsheet File liest eine spezifizierte Anzahl Zeilen einer numerischen Textdatei ab einem angegebenen Offset und konvertiert die Daten in ein *single-precision* 2D *array.* Wahlweise kann das *array* transponiert werden. Dieses VI liest Spreadsheet-Textdateien.

Kapitel 10

Verschiedene Konzepte

Das letzte Kapitel diskutiert einige wichtige LabVIEW-Aspekte, die nicht recht in die vorangehenden Kapitel passten. Einige der wichtigsten Punkte zur Manipulation der Erscheinung und des Ablaufs von VIs und *subVIs* werden kurz erklärt. Am Schluss wird beschrieben wie man von LabVIEW aus drucken kann.

Themen im 10. Kapitel :

- *VI setup*
- *subVI node setup*
- *reentrant*

10.1 Konfigurieren des VI Setups

Manchmal möchte man gewisse Eigenschaften eines VIs oder dessen Erscheinungsbild im Monitorfenster bestimmen. Solche VI Einstellungen können entweder mit den *VI setup options* oder den *subVI node setup options.* vorgenommen werden. Obschon sich einige dieser Optionen überlappen, gibt es Unterschiede in den Auswirkungen der Optionen. Die Einstellungen im *VI setup* beeinflussen den Ablauf eines VIs immer, während Optionen im *subVI node setup* nur den Aufruf eines *subVIs* betreffen.

Optionen im *subVI node setup*

Die *subVI node setup* Optionen werden durch das *pop-up* Menü des *subVI icons* erreicht. Die Optionen hier beeinflussen nur den *subVI node.* Einige der Optionen sind:

Open Front Panel when loaded: Öffnet das VI *front panel* beim Laden in den Speicher, ob es nur als VI oder als *subVI* geöffnet wird.

Show Front Panel when called: Öffnet das VI *front panel* bei einem Aufruf als *subVI* sofort.

Close afterwards if originally closed: Wenn "Show Front Panel when called" ausgewählt wurde, wird sich das VI bei einem Aufruf öffnen, "Close afterwards if originally closed" führt zum sofortigen Schliessen, nachdem das *subVI* ausgeführt wurde. Es resultiert ein *pop-up*-Fenster-Effekt.

Suspend when called: Stoppt das VI bei einem Aufruf und ergibt denselben Effekt wie das Setzen eines *breakpoints.*

Optionen im *VI setup*

Die Optionen im *VI setup* sind im *icon* des VIs in der rechten oberen Ecke des *front panels* einzustellen. Im *pop-up* Menü dieses *icons* kann **VI Setup** gewählt werden, es erscheint ein *VI setup* Fenster, das mögliche Optionen zeigt. Die hier gesetzten Optionen beeinflussen jede Art von Aufruf des VIs.

Show Front Panel When Loaded: Das VI *front panel* wird sich beim Laden in den Speicher öffnen. Dies geschieht auch wenn das VI ein *subVI* ist.

Show Front Panel When Called: Das VI *front panel* wird sich öffnen, wenn das VI als *subVI* aufgerufen wird.

Close Afterwards if Originally Closed: Wenn "Show Front Panel When Called" ausgewählt wurde, wird sich das VI bei einem Aufruf als *subVI* öffnen. "Close Afterwards if Originally Closed" führt zum sofortigen Schliessen,

nachdem das VI ausgeführt wurde. Dies ergibt wie im *subVI Node Setup* einen *pop-up*-Fenster-Effekt.

Run When Opened: Das VI wird beim Öffnen gestartet.

Suspend When Called: Das Auswählen dieser Option entspricht dem Setzen eines *breakpoints* in der *execution palette.* Ein *subVI* wird bei einem Aufruf angehalten.

Reentrant Execution: Wenn es geplant ist, das VI als *subVI* durch mehrmaliges Aufrufen zu nutzen, muss man vorsichtig sein und gleichzeitige Aufrufe vermeiden, da sonst Daten zerstört werden könnten. Wenn jedoch das VI *reentrant* ist, reserviert LabVIEW für jeden Aufruf separaten Speicherplatz. Auf diese Art kann das VI von beliebigen Orten im Programm gleichzeitig aufgerufen werden.

Print Panel When VI Completes Execution: Wenn diese Option gesetzt ist, wird das *front panel* des VIs nach Beenden auf dem Standarddrucker ausgedruckt. Es gibt auch weitere Druckoptionen wie: **print header** (Drucken eines Titels auf jedem Blatt), **scale to fit** (passt die Druckgrösse dem Blatt an) und **surround panel with border** (Drucken eines Rahmens) .

10.2 Drucken

In LabVIEW kann manuell oder automatisch gedruckt werden. Manuell kann entweder alles oder nur bestimmte Teile gedruckt werden, während das automatische Drucken nur den Ausdruck des *front panels* gestattet.
Das aktive Fenster wird mit **Print Window** aus dem **File** Menü gedruckt. Bei **Print Documentation** erscheint eine Dialogbox. In dieser Box kann spezifiziert werden, welche Teile gedruckt werden und wie die Seiten aussehen sollen. Mit **Preview** können die Seiten vor dem Drucken angeschaut werden. Unter **Preferences** werden weitere, zum Teil plattformspezifische Optionen gesetzt.

Automatisches Drucken

Im *run* Modus kann durch Wahl des **Print...**-Menüs eine Dokumentation sofort gedruckt werden. Durch Aktivieren der Option **Print Panel When VI Completes Execution** kann zur Laufzeit des Programms ein Ausdruck verlangt werden. Das VI wird in diesem Fall sein *front panel* inklusive aller momentanen Informationen und Datenanzeigen nach Ende des Programms ausdrucken. Sollen nur Teile eines *front panels* gedruckt werden, müssen diese Teile einem *subVI*, für welches die automatische Druckoption gesetzt ist, zugeführt werden. Diese Methode erlaubt das Drucken während des Ablaufs des Programms.

10.3 Abschliessende Bemerkungen

Zum Schluss sei anzumerken, dass dieser Kurs lediglich eine Einführung in die wesentlichen Aspekte der LabVIEW-Entwicklungsumgebung ist mit dem Ziel, jemanden, der wenig bis keine LabVIEW Erfahrung hat, möglichst schnell mit den grundlegenden Prinzipien des Programmierens in LabVIEW vertraut zu machen. Der Kurs orientiert sich stark am Aufbau der LabVIEW Student Edition und kann als deren geraffte Übersetzung ins Deutsche betrachtet werden.
Die Umgebung der LabVIEW-Vollversion umfasst ein Vielfaches des behandelten Stoffes und viele kleine Details, die in der Beschreibung der LabVIEW-Vollversion sehr gut dokumentiert sind. Auch die *context sensitive online help* Funktion umfasst beinahe die ganze gedruckte LabVIEW-Dokumentation, mit dem Vorteil des gezielten, raschen Zugriffs auf spezifische Themen.
Die ursprüngliche LabVIEW-Idee war jedoch, das Programmieren von Messhardware so einfach und transparent zu machen, dass auch nicht-professionelle Programmierer Mess- und Auswertapplikationen realisieren können. Diese Idee ist heute Realität. Besonders im schnell mutierenden Umfeld von Forschung und Entwicklung, wo im Labor mit möglichst kleinem Zeitaufwand Messsysteme aufgebaut werden, hat sich LabVIEW gut etabliert. Auch die produzierende Industrie hat in den letzten Jahren immer mehr und immer umfangreichere Aufgaben unter Anwendung von LabVIEW und dem darauf basierenden BridgeVIEW gelöst. Besonders die Einsparungen im Vergleich mit den langen Entwicklungszeiten bei Verwendung konventioneller Programmiersprachen hat LabVIEW in der Industrie populär gemacht. LabVIEW kann insbesondere im Messbereich effektiv und vielseitig eingesetzt werden.
Das einfache Programmieren in LabVIEW kann aber auch zu schlechtem Programmierstil verführen, da rasch etwas entwickelt werden kann, das zwar funktioniert, aber schlecht konzipiert ist. Nach wie vor ist der wichtigste Teil des Programmierens die Planung. Dieser Teil der Arbeit wird auch von LabVIEW selbstverständlich nicht geleistet.
Anderseits führen, bedingt durch das grafische Programmieren, elegante Lösungen zu visuell attraktiven Diagrammen und Benutzeroberflächen. Dieser Aspekt ist ein grosses Potential, da auch visuell begabten und künstlerisch veranlagten Menschen der Zugang zur Programmierkunst geöffnet wird. Nicht zufällig sind es junge, künstlerisch und geisteswissenschaftlich orientierte Menschen und viele Frauen, die sowohl bei der Entwicklung von LabVIEW selbst wie auch beim Programmieren von professionellen LabVIEW-Applikationen mitwirken. Auch in der Lehre kann LabVIEW sehr motivierend sein, da nebst der rationalen auch die spielerische Komponente der Softwareentwicklung unterstützt wird.

Appendix A

Bibliografie

A.1 LabVIEW Bücherliste

Eine aktualisierte Bücherliste findet man gegenwärtig unter:

`http://www.ni.com/devzone/reference/books/`

6i LabVIEW Student Edition Der Erfolg und die Nachfrage für LabVIEW in Mittelschulen und Universitäten führte zur Entwicklung der aktuellsten LabVIEW 6i Student Edition. Diese neueste Version bietet sämtliche Programmiermöglichkeiten der professionellen LabVIEW 6i Vollversion in zusammengefasster Form. Mit der LabVIEW 6i Student Edition können Studenten im Unterricht auf dem eigenen Rechner ihre Mess- und Auswertprogramme für Labor- und Praktikumsexperimente selber programmieren. Die grafische Benutzeroberfläche erleichtert die Darstellung der Messdaten und die umfangreichen Analysefunktionen bieten die Möglichkeit zur Anwendung raffinierter Analyseverfahren. Die 6i LabVIEW Student Edition mit dem didaktisch gut strukturierten Begleitbuch ist ideal zum Lernen der LabVIEW Grundlagen. Die Hardware-Treiber unterstützen im Unterschied zu früheren Versionen der LabVIEW Student Edition alle LabVIEW Datentypen. Die gesamte Analysebibliothek der Vollversion ist vorhanden. Die 6i Student Edition besitzt gegenüber der Vollversion folgende Einschränkungen:

- Keine ActiveX Komponenten.
- Kein Einbinden externer Programmiermodule in Form von DLLs und CINs möglich.
- Keine Möglichkeit unabhäng lauffähige Programme zu generieren

Systemanforderungen:

Windows 2000/NT/Me/9x (Für Windows NT ist Windows NT 4.0 Service Pack 3 oder später erforderlich)
32 MB RAM (64 MB empfohlen)

65 MB Speicherplatz auf Disk für minimale LabVIEW Installation, 200 MB für die ganze LabVIEW Installation (Treibersoftware benötigt zusätzlichen Speicherplatz auf Disk)
Pentium Prozessor oder äquivalent empfohlen

Mac OS
Mac OS 7.6.1 oder später
32 MB RAM (64 MB empfohlen)
100 MB Speicherplatz auf Disk für minimale LabVIEW Installation, 225 MB für die ganze LabVIEW Installation (Treibersoftware benötigt zusätzlichen Speicherplatz auf Disk)
PowerPC Prozessor

LVSE ISBN: 0-13-032550-3

Bei Freihofer Buchhandlung in Zürich. Tel (01) 363 42 82 (ca. Fr.100.-)

LabVIEW Graphical Programming, Practical Applications in Instrumentation and Control : Zweite Edition. Sehr gutes Buch, unterhaltsam geschrieben von einem LabVIEW Spezialist der ersten Stunde. Sehr geeignet für Programmierer welche LabVIEW professionell einsetzen wollen aber auch interessant für engagierte Anfänger die LabVIEW über die Grundlagen hinaus zu praktizieren gedenken. Das Buch beinhaltet zahlreiche Tips und Tricks der Messdatenerfassung mit LabVIEW, welche bei praktischen Anwendungen sehr wertvoll sein können. Kommt mit einer CD voll intelligenter Beispiele und vielen brauchbaren VIs. (Englisch)

Autor: Gary W Johnson (Labviewer seit V 1.2)
Verlag: McGraw-Hill, Inc.
ISBN 0-07-032915-X
Kann im Buchhandel oder bei National Instruments Schweiz bestellt werden. Tel (056) 200 51 51 (ca. Fr.120.-)

LabVIEW Lernhandbuch : Ist das auf Deutsch übersetzte "Tutorial Manual" der Vollversion 3.1.1. Ein kompletter Einführungskurs, didaktisch nicht ganz so umfassend wie die Student Version. Wenn die Englische Sprache ein Problem ist, kann dieses Handbuch als Grundkurs gute Dienste leisten.

Bei National Instruments Schweiz (siehe oben). (ca. Fr.95.-)

LabVIEW for Everyone, Graphical Programming Made Even Easier : Gutes Einsteigerbuch mit zahlreichen interessanten Übungsbeispielen auch geeignet für das autodidaktische Erlernen von LabVIEW. Inklusive CD mit Übungsbeispielen basierend auf LabVIEW 4.0. (Englisch)

Autoren: Lisa K. Wells & Jeffrey Travis
Verlag: Prentice Hall PTR
ISBN 0-13-268194-3
Bezug im Buchhandel. (ca. Fr.100.-)

Das LabVIEW-Buch : Deutsche Übersetzung von Lisa Wells und Jeffrey Travis "LabVIEW for Everyone" inklusive CD basierend auf LabVIEW 4.0.

Autoren: Lisa K. Wells & Jeffrey Travis
Verlag: Prentice Hall PTR
ISBN 3-8272-9540-8
Bezug im Buchhandel. (ca. Fr.100.-)

Joint Time Frequency Analysis, Methods and Applications : Umfassende Beschreibung moderner Signalanalyseverfahren im Zeit- und Frequenzbereich. Hat keinen direkten Bezug zum LabVIEW Programmentwicklungssystem, beschreibt aber die Grundlagen zum JTFA Toolkit, einer LabVIEW Zusatzbibliothek für Signalanalysen wie STFT (Short-Time Fourier Transform), Wavelets, Wigner-Ville Distribution, Cohen's Class usw.

Autoren: Shie Qian & Dapang Chen
Verlag: Prentice Hall PTR
ISBN 0-13-254384-2
Bezug im Buchhandel. (ca. Fr.100.-)

LabVIEW Beispiele, Die Kunst der grafischen Programmierung : Eine Sammlung von ausgewählten Beispielen für praktische Anwendungen eher für Fortgeschrittene. (Deutsch)

Autoren: Ralph Griemert & Wolfgang Erhart
Verlag: Eigenverlag
Bezug bei National Instruments Schweiz (siehe oben). (ca. Fr.50.-)

Praxisbuch LabVIEW 3 : Deutsches Lernbuch für LabVIEW auch für Anfänger. Kommt mit einer Diskette zum Buch.

Autor: Hans-Günter Dahn
Verlag: IWT Verlag GmbH
ISBN 3-88322-445-6 (ca. Fr.95.-)

A.2 LabVIEW Internet Adressen

National Instruments Austin Texas :

- Web-site: **http://www.ni.com** National Instruments *home page*.
- ftp-site: **ftp.ni.com** (für drivers, Info usw.)

LabVIEW Mailgroup :

- Web-site: **http://www.info-labview.org/** Eine Internetseite der LabVIEW *mailgroup*, wo zur Zeit ca. 2000 LabView-Programmierer registriert sind und international über E-mail kommunizieren.

Brian Renken LabVIEW-pages :

- Web-site: **http://LabVIEW.BrianRenken.com** Brian Renken ist ein engagierter LabVIEW-Programmierer; er unterhält eine Internetseite mit vielen *links* zu anderen LabVIEW *sites* in Form eines *Web Ring*. Er offeriert dort einen eine gute Suchmaschine für die info-labview Archive.

Lego "Robolab" :

- Web-site: **http://www.lego.com/dacta/** Für Information zum Ausbildungsprojekt "Robolab" von Lego, National Instruments und Tufts University Engineering. Einfache Programmierung autonomer intelligenter Lego Mindstorm Komponenten im Unterricht.

- Web-site: **http://ldaps.arc.nasa.gov/LEGOEngineer/** Beispiele von Lego-Robolab-Applikationen.