ADwin

Treiber für LabVIEW ab Version 2009

ADwin -Treiber LabVIEW, Handbuch Apr. 2023

Hier finden Sie immer einen Ansprechpartner für Ihre Fragen:

Hotline:	(0 62 51) 9 63 20
Fax:	(0 62 51) 9 63 299
E-Mail:	info@ADwin.de
Internet	www.ADwin.de

JÄGER
Computergesteuerte
Messtechnik GmbH

Jäger Computergesteuerte
Messtechnik GmbH
Rheinstraße 2-4
D-64653 Lorsch

Inhaltsverzeichnis

Typografische Konventionen

Das „Achtung“-Zeichen steht bei Informationen, die auf Folgeschäden durch Fehlbedienung an der Hard- oder Software, am Messaufbau oder an Personen hinweisen.

Einen „Hinweis“ finden Sie bei

– Informationen, die für einen fehlerfreien Betrieb unbedingt beachtet werden müssen.

– Tipps und Ratschlägen für einen effizienten Betrieb.

Das Zeichen „Information“ verweist auf weiterführende Informationen in dieser Dokumentation oder andere Quellen wie Handbücher, Datenblätter, Literatur etc.

Das Zeichen „Beispiel“ weist auf praktische Beispiele hin.

C:\ADwin\…

Dateinamen und -verzeichnisse sind in spitzen Klammern und im Schrifttyp Courier New angeben.

Programmtext

Programmanweisungen und Benutzer-Eingaben sind durch den Schrifttyp Courier New gekennzeichnet.

```
Var_1
```

Elemente eines Quelltextes wie Befehle, Variablen, Kommentar und sonstiger Text werden im Schrifttyp Courier New und farbig dargestellt.

In einem Datenwort (hier: 16 Bit) werden die Bits wie folgt nummeriert:

Bit-Nr.	15	14	13	…	1	0
Wert des Bits	2^{15}	2^{14}	2^{13}	…	2^1=2	2^0=1
Bezeichnung	MSB	-	-	-	-	LSB

1 Zu diesem Handbuch

Dieses Handbuch enthält umfassende Informationen für den Einsatz des *ADwin*-LabVIEW-Treibers für LabVIEW® ab Version 2009.

Der Treiber ist für den Einsatz unter früheren LabVIEW-Versionen (8 oder kleiner) nicht geeignet. Falls erforderlich, wenden Sie sich bitte an unsere Hotline.

Das Handbuch wird ergänzt durch

– das Handbuch „*ADwin* Installation“, das die Hardware-Schnittstellen-Installation zu allen *ADwin*-Systemen beschreibt.

  Beginnen Sie die Installation mit diesem Handbuch.

– Falls Sie unter Linux oder MacOS arbeiten: das Handbuch „ADwin für Linux/Mac“, das die Software-Installation und den *ADbasic*-Compiler unter Linux und MacOS beschreibt.

– die Handbücher „*ADbasic*“, „*ADsim*“ und „*ADwinC*“. Mit jedem der Programmpakete können Sie Ihre *ADwin*-Hardware programmieren und *ADwin*-Prozesse darauf laufen lassen.
  - *ADbasic* umfasst die komfortable Echtzeit-Entwicklungsumgebung und die Befehle des Compilers *ADbasic*.
  - Das Programmpaket *ADsim* macht Simulink®-Modelle auf *ADwin*-Systemen lauffähig.
  - Mit ADwinC für Visual Studio können Sie ADwin-Prozesse in der Sprache C entwickeln.

– die Hardware-Handbücher für die von Ihnen verwendeten *ADwin*-Systeme.

Es wird vorausgesetzt, dass Sie den Umgang mit der Entwicklungsumgebung LabVIEW® beherrschen.

Bitte beachten Sie folgende Hinweise

Damit Ihr *ADwin*-System sicher arbeitet, halten Sie sich an die Informationen dieser und weiterführender Dokumentationen, auf die hier verwiesen wird.

Einschränkung der Anwendergruppe

Der Hersteller des in dieser Dokumentation beschriebenen Systems geht davon aus, dass an dem Gerät nur qualifiziertes Personal arbeitet.

> *Qualifiziertes Personal sind Personen, die aufgrund ihrer Ausbildung, Erfahrung und Unterweisung sowie ihrer Kenntnisse über einschlägige Normen, Bestimmungen, Unfallverhütungsvorschriften und Betriebsverhältnisse von dem für die Sicherheit der Anlage Verantwortlichen berechtigt worden sind, die jeweils erforderlichen Tätigkeiten auszuführen und die dabei mögliche Gefahren erkennen und vermeiden können. (Definition für Fachkräfte nach VDE 105 und IEC 60364).*

Verfügbarkeit der Unterlagen

Diese Produktdokumentation und Unterlagen, auf die verwiesen wird, müssen stets verfügbar sein und konsequent beachtet werden. Für Schäden, die durch Missachtung der Informationen in dieser bzw. der weiterführenden Dokumentation entstehen, übernimmt die Firma *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch, keine Haftung.

Rechtliche Grundlagen

Änderungen vorbehalten.

Hotline-Adresse siehe vordere Umschlagseite, innen.

2 *ADwin*-LabVIEW-Treiber

ADwin erweitert die Möglichkeiten von LabVIEW

Das *ADwin*-System besteht aus einem eigenständigen Messrechner, der Mess- und Regelaufgaben extrem schnell und sicher erledigt, und einer Schnittstelle unter Windows, Linux oder Mac OS, über die Sie mit LabVIEW das *ADwin*-System steuern.

Sie verlagern also alle zeitkritischen Prozesse in das *ADwin*-System, haben aber die Steuerung der Prozesse und Verarbeitung der Daten weiterhin mit LabVIEW in der Hand.

Wie Sie das *ADwin*-System programmieren

ADwin-Systeme sind schnell, zuverlässig und flexibel. Um diese Vorteile optimal zu nutzen, verwenden Sie die einfache Programmiersprache *ADbasic*.

Bevor Sie die LabVIEW-Befehle anwenden können, empfehlen wir Ihnen eine Einarbeitung in *ADbasic*. Hierzu verwenden Sie bitte das *ADbasic*-Handbuch und die Programmieranleitung. Die Beschreibungen werden Ihnen auch das Verständnis des *ADwin*-Systems erleichtern.

Alternativ können Sie mit *ADsim* ein Simulink-Modell exportieren und auf der *ADwin*-Hardware als Prozess ausführen (siehe Handbuch *ADsim*).

ADwin-Systeme mit LabVIEW steuern

Jetzt ist der Moment gekommen, mit diesem Handbuch den Schritt in die Praxis zu tun.

Die Abschnitte 2.1 und 2.2 schildern, wie LabVIEW und *ADwin* kommunizieren und vertiefen Ihr Verständnis für das *ADwin*-Konzept.

In Kapitel 3 wird die Installation und Einbindung der neuen Befehle beschrieben.

Allgemeines zum LabVIEW-Treiber ist in Kapitel 4 erklärt, die einzelnen Funktionen in Form eines als Nachschlagewerks in Kapitel 5.

2.1 Schnittstelle zum *ADwin*-System

Der *ADwin*-LabVIEW-Treiber ist die Schnittstelle für die Entwicklungsumgebung LabVIEW® ab Version 2009 zur Kommunikation mit *ADwin*-Systemen.

Die Kombination von LabVIEW® mit einem *ADwin*-System bietet Ihnen völlig neue Möglichkeiten. Die Intelligenz und Rechenleistung des *ADwin*-Systems zum einen und die Funktionen zum Verwalten, Analysieren und Dokumentieren von Messdaten zum anderen bieten ein leistungsstarkes Konzept.

Sie legen das Verhalten der *ADwin*-Hardware selbst fest. Hierzu gibt es folgende Wege:

- *ADbasic*: Sie erstellen Echtzeit-Prozesse mit der Entwicklungsumgebung *ADbasic*, erzeugen daraus eine Binärdatei und übertragen sie auf das *ADwin*-System (siehe Handbuch oder Online-Hilfe *ADbasic*).
- *ADwinC*: Sie erstellen Echtzeit-Prozesse in Visual Studio in der Sprache C mit dem Programmpaket ADwinC für Visual Studio. Sie erzeugen eine Binärdatei und übertragen sie auf das *ADwin*-System (siehe Handbuch *ADwinC*).
- *ADsim*: Sie erstellen in Simulink ein Modell, exportieren es und kompilieren es mit *ADsim* für die *ADwin*-Hardware (siehe Handbuch *ADsim*).

Typische Anwendungen sind:

– Steuerung schneller Prüfstände
– Signale generieren
– Intelligent messen, Daten mit komplexen Triggerbedingungen erfassen
– Regeln und Steuern
– Online-Verarbeitung, Datenreduzierung
– Hardware-in-the-Loop, Simulation von Sensordaten

2.2 Kommunikation mit dem *ADwin*-System

Aus der Entwicklungsumgebung LabVIEW® können Sie Prozesse im *ADwin*-System steuern, sowie Daten von dort anfordern oder dorthin senden. Die Prozesse selbst programmieren Sie mit dem Echtzeit-Entwicklungstool *ADbasic*, erzeugen daraus eine Binärdatei und übertragen sie auf das *ADwin*-System (siehe Handbuch oder Online-Hilfe *ADbasic*).

Daten und Befehle zwischen LabVIEW® und dem *ADwin*-System durchlaufen den nachfolgend dargestellten Weg.

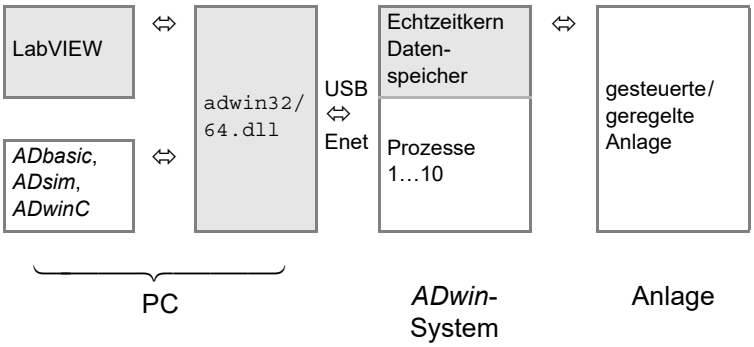


Abb. 1 – *ADwin*-LabVIEW Schnittstelle

adwin32/64.dll

Die `adwin32.dll` (64 Bit: `adwin64.dll`) ist die einzige Schnittstelle für Anwendungen zum *ADwin*-System und wird daher auch vom *ADwin*-LabVIEW-Treiber genutzt. Diese Schnittstelle ermöglicht, dass mehrere Windows-Programme gleichzeitig mit dem *ADwin*-System kommunizieren: So können Entwicklungsumgebungen, *ADbasic* und *ADtools* parallel mit dem *ADwin*-System arbeiten.

Echtzeitkern

Die Schnittstelle `adwin32/64.dll` kommuniziert mit dem Echtzeitkern des *ADwin*-Systems, dem Betriebssystem. Deshalb müssen Sie nach jedem Einschalten des Systems zunächst das Betriebssystem (z. B. in Form einer Datei wie `<adwin12.btl>`) dorthin laden. Nach erfolgreicher Übertragung kann das System Prozesse empfangen und ausführen, Befehle vom PC entgegen nehmen und Daten mit ihm austauschen. Die in *ADbasic* programmierten Prozesse enthalten den Programmcode zur Messung, Steuerung oder Regelung Ihrer Applikation.

Die Aufgaben des Betriebssystems sind:

10 Prozesse

– Verwaltung von bis zu 10 Echtzeit-Prozessen mit niedriger oder hoher Priorität (frei wählbar). Niedrig priorisierte Prozesse können von hoch priorisierten Prozessen unterbrochen werden, letztere können nicht von anderen Prozessen unterbrochen werden.

Datenspeicher

– Bereitstellung von globalen Variablen:
  - 80 ganzzahlige Variablen (`Par_1` ... `Par_80`), bereits vordefiniert
  - 80 Fließkomma-Variable (`FPar_1` ... `FPar_80`), bereits vordefiniert
  - 200 Datenfelder (`Data_1` ... `Data_200`), Länge und Datentyp frei definierbar

  Sie können die Werte dieser Variablen bzw. Datenfelder aus der Entwicklungsumgebung jederzeit lesen und ändern.

Kommunikation

– Kommunikation zwischen *ADwin*-System und PC (`adwin32/64.dll`).

  Der Kommunikationsprozess läuft mit mittlerer Priorität auf dem *ADwin*-System und kann niedrig priorisierte Prozesse für kurze Zeit unterbrechen. Er interpretiert oder bearbeitet alle Befehle, die Sie aus LabVIEW (oder aus anderen Programmen) an das *ADwin*-System richten: Steuerbefehle und Befehle für den Datenaustausch.

  Der Kommunikationsprozess sendet niemals unaufgefordert Daten an den PC. Das stellt sicher, dass nur dann Daten zum PC übertragen werden, wenn Sie diese ausdrücklich angefordert haben.

2.3 *ADsim*-Treiber für LabVIEW

Ergänzend zum *ADwin*-Treiber für LabVIEW gibt es für Simulink-Anwender zusätzlich den *ADsim*-Treiber für LabVIEW. Die Befehle entsprechen denen des Treibers für MATLAB (siehe Handbuch *ADsim*-Treiber für Matlab).

Der *ADsim*-Treiber arbeitet nur unter Windows.

Während Sie mit dem *ADwin*-Treiber für LabVIEW auf *ADwin*-Variablen des Simulink-Modells zugreifen können, ermöglicht der *ADsim*-Treiber den Zugriff auf die übrigen Variablen und Felder des Simulink-Modells, also auf Parameter, Signale und Blockzustände.
Die beiden Treiber können unabhängig voneinander verwendet werden. Beachten Sie, dass sich die Treiber bei der Fehlerverarbeitung sehr unterschiedlich verhalten.

Wir empfehlen, vorwiegend den *ADwin*-Treiber für LabVIEW zu verwenden. Im Vergleich bietet er folgende Vorteile:

– Der Treiber unterstützt die kontinuierliche Datenübertragung über Fifo-Felder.
– Kompilierte Simulink-Modelle (mit *ADsim*) können mit dem VI `Boot` auf die *ADwin*-Hardware übertragen werden.
– Der Treiber arbeitet unter Linux und Mac OS.

Anwender von *ADsim* können alle Funktionen des *ADwin*-Treibers für LabVIEW nutzen bis auf folgende:

– Free_Mem.VI
– String_Length.VI, SetData_String.VI, GetData_String.VI

Anwender von *ADsim* T11 (!) müssen außerdem auf folgende Funktionen verzichten:

– Load_Process.VI, Start_Process.VI, Stop_Process.VI, Clear_Process.VI, Process_Status.VI

3 LabVIEW-Treiber installieren

Bitte befolgen Sie die unten beschriebenen Installationsschritte, um auf einfache Weise Zugang aus LabVIEW® zum *ADwin*-System zu bekommen.

3.1 „*ADwin* Installation“ ausführen

Für die Installation benötigen Sie das aktuelle *ADwin*-Software-Paket.

3.1.1 Installation unter Linux oder Mac

Folgen Sie der Installationsanleitung im Handbuch „ADwin für Linux / Mac“. Achten Sie darauf, auch das Archiv `adwin-labview-x.y.zip` zum Schluss zu installieren.

Nach erfolgreicher Installation finden Sie die Dateien in den folgenden Verzeichnissen unterhalb von `</opt/adwin/share>` (Standardinstallation):

VI-Sammlung und Beispiele für LabVIEW	`./ADwin_v3.2.lib`
VI-Sammlung für ältere LabVIEW-Versionen	`./ADwin_v2.2.lib`
Beispiele für *ADbasic*	`./samples_ADwin`

Fahren Sie fort mit Kapitel 3.2 „ADwin VIs einbinden“.

3.1.2 Installation unter Windows

Falls *ADwin* installiert ist

Wenn Sie bereits *ADwin*-Hardware und -Software aus dem *ADwin*-Software-Paket installiert haben, können Sie diesen Abschnitt überspringen und mit Kapitel 3.2 weiter arbeiten.

Sonst: Neue Installation

Falls *ADwin*-Hardware neu installiert werden soll, beginnen Sie bitte die Installation mit dem Handbuch „*ADwin* Installation“, das mit der *ADwin*-Hardware ausgeliefert wird. Es beschreibt, wie Sie

- das *ADwin*-Software-Paket installieren.
- die Kommunikations-Treiber unter Windows installieren.
- die Hardware im PC einbauen (falls erforderlich) und die Hardware-Verbindung zwischen PC und *ADwin*-System aufbauen.

Nach erfolgreicher Installation finden Sie die Dateien in den folgenden Verzeichnissen unterhalb von `<C:\ADwin\>` (Standardinstallation):

VI-Sammlung und Beispiele für LabVIEW	`.\Developer\LabVIEW\ADwin_v3.2.lib`
VI-Sammlung für ältere LabVIEW-Versionen	`.\Developer\LabVIEW\ADwin_v2.2.lib`
Beispiele für *ADbasic*	`.\ADbasic\samples_ADwin`
Testprogramm für *ADwin-Gold*, *ADwin-light-16* und Einsteckkarten	`.\Tools\Test\ADtest`
Testprogramm für *ADwin-Pro*	`.\Tools\Test\ADpro`

Fahren Sie fort mit dem nächsten Abschnitt: „ADwin VIs einbinden“.

3.2 *ADwin* VIs einbinden

In LabVIEW können Sie mit Hilfe vorgefertigter VIs Befehle und Daten an das *ADwin*-System schicken und Daten vom System abrufen. Die VIs nutzen hierzu Funktionen der *ADwin*-LabVIEW-Schnittstelle (siehe Kapitel 2.2).

So binden Sie die VIs in LabVIEW ein:

– Kopieren Sie den Ordner `<ADwin_v3.2.lib>` aus dem Installationsverzeichnis in das LabVIEW-Verzeichnis, unter Windows z.B. nach `C:\Program Files\National Instruments\LabVIEW 2021\user.lib`.

  Damit ist die EInbindung bereits abgeschlossen.

– Wenn Sie in LabVIEW in der Gruppe `User Libraries` die Maus über das Icon `ADwin_v3.2.lib` bewegen, erscheinen alle VIs in einem neuen Fenster (siehe rechts).

  Sie können die VIs aus der Gruppe direkt in ein LabVIEW-Diagramm ziehen. Die einzelnen VIs sind in Kapitel 5 beschrieben.

3.3 *ADwin* VIs konvertieren

Die *ADwin* VIs wurden mit LabVIEW 2009 gespeichert. Wenn Sie eine spätere LabVIEW-Version verwenden, empfiehlt National Instruments®, die VIs zu konvertieren und neu zu speichern, damit LabVIEW mit weniger Arbeitsspeicher auskommt und die VIs merklich schneller ausführt.

Weitere Hinweise finden Sie im Handbuch LabVIEW von National Instruments® unter „Konvertieren von VIs“; um VIs verzeichnisweise zu konvertieren, suchen Sie nach dem Stichwort „Massenkompilierung“.

4 Allgemeines zu *ADwin*-VIs

4.1 Grundlagen

Die Farben der VI-Symbole (Icons) zeigen die Funktion der VIs an; die Farben orientieren sich teilweise am LabVIEW-Standard:

Grün:	Systeminformationen und Steuerung von Prozessen
Blau:	Übertragung von globalen Long-Variablen
Orange:	Übertragung von globalen Float-Variablen
Rot:	Fehlerfunktion
Weiß:	Übertragung von Data-Feldern
Weiß mit Ring:	Übertragung von Fifo-Feldern (Ringspeicher)
Weiß mit Rahmen:	Übertragung von Zeichenfolgen
Hellblau:	Hilfsfunktionen
Pink:	Beispielprogramme

Beachten Sie bitte für die Verwendung der *ADwin*-VIs folgende Hinweise:

Fehlerverdrahtung

– Verbinden Sie alle *ADwin*-VIs mit „`error in`“ und „`error out`“ durch eine Fehlerverdrahtung (siehe Kapitel 4.3).

Device No.

– Übergeben Sie eine „Device No.“ (siehe Kapitel 4.4) als Eingangswert an die VIs. Nur wenige VIs kommen ohne „Device No.“ aus.

Kontext-Hilfe

– Zu allen VIs gibt es eine Kontext-Hilfe, die Sie durch `[Strg]`+`[H]` starten.

  Wenn Sie die Maus bei geöffneter Hilfe auf ein VI-Symbol bewegen, werden ein (englischer) Hilfetext und eine Funktionsskizze des VI angezeigt. Dieses Handbuch enthält nur den Hilfetext, nicht aber die Funktionsskizze.

4.2 Treiber für LabVIEW 4…8 verwenden

Wenn Sie einen älteren Treiber für LabVIEW 4…8 besitzen, können Sie VIs des älteren Treibers unter 32 Bit-Programmversionen weiterhin verwenden, wir empfehlen es jedoch nicht. Für 64 Bit-Programmversionen sind die alten VIs nicht einsetzbar.

Wenn Sie mit einer LabVIEW-Version 6…8 arbeiten, verwenden Sie die VI-Sammlung 2.2. Das zugehörige Handbuch finden Sie im entsprechenden Verzeichnis, siehe Kapitel 3 auf Seite 5.
Für Treiber zu den LabVIEW-Versionen 4…5 wenden Sie sich bitte an unseren Support; die Adresse finden Sie auf der vorderen Umschlagseite des Handbuchs, innen.

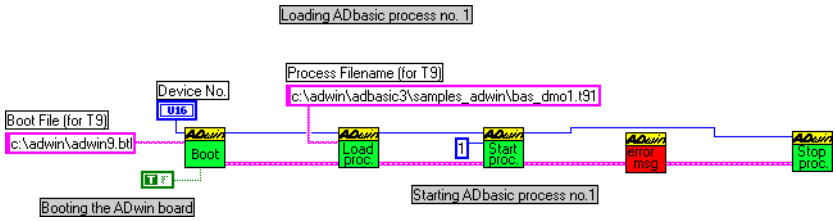
Im Vergleich zu früheren Versionen gibt es im aktuellen Treiber folgende Änderungen:

– unterstützt LabVIEW mit 32 Bit und mit 64 Bit
– verbesserte Hilfe
– zahlreiche neue VIs
– Vorgabewert für Device No. ist 1 (siehe Kapitel 4.4)
– veraltete VIs sind entfallen

4.3 Fehlerbehandlung

Der LabVIEW-Treiber stellt Ihnen mit der „Fehlerverdrahtung“ eine Möglichkeit der Fehlerbehandlung zur Verfügung. Die Fehlerverdrahtung legt außerdem eindeutig fest, in welcher Reihenfolge LabVIEW die VIs bearbeitet.

In dem Beispiel unten verbindet eine gestrichelte Linie (pink) alle VIs: Dies ist die Fehlerverdrahtung. Der Fehlereingang des ersten VI bleibt offen.



Sobald bei einem *ADwin*-VI ein Fehler auftritt, wird über die Verdahtung ein Fehlersignal an alle folgenden VI weiter geleitet, die daraufhin das *ADwin*-System nicht mehr ansprechen. Mit dem VI `Error_msg` kann eine Fehlermeldung generiert werden.

Wenden Sie die Fehlerverdrahtung konsequent an:

– Verbinden Sie alle VIs miteinander über die Anschlüsse „`error in`“ und „`error out`“.
– Benutzen Sie die Fehlermeldungen zur Fehlerbehandlung, indem Sie die Meldungen mit dem VI `error_msg` abfragen (beispielsweise einmal pro Schleifendurchlauf).

Beachten Sie: Wegen der Kompatibilität mit früheren Versionen geben die meisten VI beim Auftreten eines Fehlers auf einem Datenausgang einen bestimmten Wert zurück. Bei manchen VI (Free_Mem.VI, Get_Processdelay.VI, Get_Par.VI, Get_FPar.VI, Get_FPar_Double.VI, Fifo_Empty.VI, Fifo_Full.VI) kann dieser Wert aber auch ein gültiger Datenwert sein. Der Rückgabewert ist dann nicht eindeutig.

4.4 Die „DeviceNo.“

Eine „Device No.“ kennzeichnet ein *ADwin*-System eindeutig und wird für die korrekte Kommunikation mit dem *ADwin*-System benötigt.

Sie stellen die „Device No.“ eines *ADwin*-Systems mit dem Programm ADconfig ein.

In LabVIEW benötigt fast jedes *ADwin*-VI die „Device No.“. Schließen Sie an das erste VI die „Device No.“ an. Verbinden Sie den Ausgang „Device No. out“ mit dem nächsten VI und erstellen so eine durchgehende Verdrahtung (im Beispiel im Abschnitt „Fehlerbehandlung“ die obere Linie in blau).

4.5 Ungültige Fließkommawerte der ADwin-Hardware

Bei 32 Bit-Fließkommazahlen bis Prozessor T11 werden Bitmuster von ungültigen oder grenzwertigen Werten in der *ADwin*-Hardware bei der Übertragung auf den PC in andere Werte gewandelt, siehe folgende Tabelle. Zahlen im gültigen Wertebereich (normalized numbers) bleiben unverändert.
Beim Prozessor T12 werden dagegen die IEEE-Bezeichnungen wie `#INF` verwendet.

Bezeichnung gemäß IEEE	Bitmuster-Bereich	32 Bit-Wert auf dem PC
+0	00000000h	0
Positive denormalized numbers	00000001h 007FFFFFh	0
Positive normalized numbers	00800000h 7F7FFFFFh	+1,175494 · 10-38 +3,402823 · 10+38
+∞ (Infinity, #INF)	7F800000h	3.402823E+38
Signalling Not a number (SNaN)	7F800001h 7FBFFFFFh	3.402823E+38
Quiet Not a number (QNaN)	7FC00000h 7FFFFFFFh	3.402823E+38
-0	80000000h	0
Negative denormalized numbers	80000001h 807FFFFFh	0
Negative normalized numbers	80800000h FF7FFFFFh	-1,175494 · 10-38 -3,402823 · 10+38
-∞ (Infinity, #INF)	FF800000h	-3.402823E+38
Signalling Not a number (SNAN)	FF800001h FFBFFFFFh	3.402823E+38
Indeterminate	FFC00000h	3.402823E+38
Quiet Not a number (QNAN)	FFC00001h FFFFFFFFh	3.402823E+38

4.6 Beispielprogramme

Die Treiberinstallation enthält einige Beispielprogramme, die das Zusammenspiel von LabVIEW mit dem *ADwin*-System verdeutlichen. Jedes Beispiel besteht jeweils aus einem LabVIEW-Diagramm und einem *ADbasic*-Programm:

– Das LabVIEW-Diagramm steuert den *ADwin*-Prozess und zeigt die übertragenen Daten an.
– Das *ADbasic*-Programm definiert den Ablauf des Prozesses auf dem *ADwin*-System.

Die LabVIEW-Diagramme sind im Installationsverzeichnis abgelegt (siehe Kapitel 3.1), die *ADbasic*-Programme in folgendem Verzeichnis:

– Windows: `<C:\ADwin\ADbasic\samples_ADwin\>`
– Linux: `</opt/adwin/share/samples_ADwin/>`

Zum besseren Verständnis der Beispiele betrachten Sie bitte die *ADbasic*-Quelltexte und die zugehörigen LabVIEW-Diagramme.

5 Beschreibung der *ADwin*-VIs

Die Beschreibung der VIs ist in folgende Abschnitte unterteilt:

Beachten Sie auf jeden Fall das Kapitel 4, in dem allgemeine Hinweise zur Verwendung der *ADwin*-VIs beschrieben sind.

5.1 Systemsteuerung und -information

Initialisierung des *ADwin*-Systems und Informationen über den Betriebszustand.

Boot.VI

`Boot.VI` intialisiert das *ADwin*-System und lädt die Betriebssystem-Datei dorthin.

Eingänge

Device No.	Device No. des Systems
Filename	Name der Betriebssystem-Datei mit vollständigem Pfad, abhängig vom Prozessortyp: T2: `ADwin2.btl` T4: `ADwin4.btl` T5: `ADwin5.btl` T8: `ADwin8.btl` T9: `ADwin9.btl` T10: `ADwin10.btl` T11: `ADwin11.btl` T12: `ADwin12.btl` T12.1: `ADwin121.btl`
Memsize	Nur für Prozessoren T2…T8: Eingebaute Speichergröße (64 KiB…32 MiB)
error in	Anschluss des Fehlersignals.
showError	true: bei anstehendem Fehlersignal die Fehlermeldung in einem Meldungsfenster ausgeben. false: keine Fehlermeldung ausgeben.

Ausgänge

Device No. out — Die am Eingang „Device No.“ angelegte Nummer.

Memory — Statusmeldung:
<1000: Fehler beim Booten
8000: Booten o.k.; ab T9
>8000: Booten o.k. und MemSize ist korrekt; nur für T2…T8. Werte für Speichergröße MemSize:

10000:	64 KiB
100000:	1 MiB
200000:	2 MiB
400000:	4 MiB
800000:	8 MiB
1000000:	16 MiB
2000000:	32 MiB

error out — Fehlersignal

Bemerkungen

Die Initialisierung löscht alle Prozesse auf dem System und setzt alle globalen Variablen auf den Wert 0. Der Parameter `Processdelay` wird auf 1000 voreingestellt.

Der PC kann erst mit dem System kommunizieren, nachdem Sie das Betriebssystem geladen haben. Laden Sie das Betriebssystem nach jedem Aus- und Einschalten des Systems neu.

Für Nutzer von *ADsim* mit Prozessor T11:

- Sie geben unter `Filename` das mit *ADsimDesk* kompilierte Simulink-Modell an, in das auch das Betriebssystem für den Prozessor integriert ist. Die Datei ist im Modellordner abgelegt, im Unterordner `<model>_ert_rtw/ADwin/` unter `<model>11c.btl`.
- `<model>` steht für den Namen des Simulink-Modells. Die Bezeichnung `11c` steht für den Prozessortyp T11 der *ADwin*-Hardware.
- Beachten Sie, dass Sie *ADbasic*-Prozesse und mit *ADsim* T11 kompilierte Simulink-Modelle nicht gleichzeitig auf *ADwin*-Hardware ausführen können.

Das erfolgreiche Laden des Betriebssystems nimmt bis zu 1 Sekunde in Anspruch.

Stellen Sie die Speichergröße `Memsize` ein, indem Sie mit der rechten Maustaste auf den Eingang klicken und „`Create` ▸ `Constant`“ auswählen. In dem erscheinenden Auswahlfenster wählen Sie die passende Speichergröße.

Test_Version.VI

`Test_Version.VI` prüft, ob Treiberversion und Betriebssystem des Prozessors zueinander passen, und ob der Prozessor ansprechbar ist.

Eingänge

Device No.	Device No. des Systems.
error in	Anschluss des Fehlersignals.
showError	true: bei anstehendem Fehlersignal die Fehlermeldung in einem Meldungsfenster ausgeben. false: keine Fehlermeldung ausgeben.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
Version	0: OK ≠0: Betriebssystem hat falsche Version oder antwortet nicht.
error out	Fehlersignal

Sanitize_Floating_Point_Value.VI

`Sanitize_Floating_Point_Value.VI` stellt den Bereinigungs-Modus für NaN-Werte in Fließkomma-Feldern ein.

Inputs

flag	Der Flag-Wert stellt den Bereinigungs-Modus ein: 0: NaN-Werte werden weitergegeben wie sie sind, sie bleiben unverändert. 1: Die *ADwin*-DLL konvertiert NaN-Werte in Fließkomma-Feldern in reguläre Werte, in der Regel in MAX_FLOAT (Voreinstellung).
error in	Anschluss des Fehlersignals.
showError	true: bei anstehendem Fehlersignal die Fehlermeldung in einem Meldungsfenster ausgeben. false: keine Fehlermeldung ausgeben.

Outputs

error out	Fehlersignal

Notes

Die Voreinstellung ist, NaN-Werte in reguläre Werte zu konvertieren (flag=1).

Processor_Type.VI

`Processor_Type.VI` gibt den Prozessortyp des Systems zurück.

Eingänge

Device No.	Device No. des Systems.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
Processor type	Prozessortyp: 0: Fehler 2: T2 4: T4 5: T5 8: T8 9: T9 1010: T10 1011: T11 1012: T12 10121: T12.1
error out	Fehlersignal

Workload.VI

`Workload.VI` gibt die Prozessor-Auslastung zurück.

Eingänge

Device No.	Device No. des Systems.
priority	0: Gesamtauslastung des Prozessors ≠0: keine Funktion
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
Workload	≠255: Prozessorauslastung in % 255: Fehler
error out	Fehlersignal

Free_Mem.VI

`Free_Mem.VI` ermittelt den auf dem System verfügbaren freien Speicher für verschiedene Speicherarten.

Eingänge

Device No.	Device No. des Systems.
MemType	Speicherart: 0 : alle Speicherarten gemeinsam; nur für T2, T4, T5, T8 1 : interner Programmspeicher (PM_LOCAL); T9…T11 2: interner Datenspeicher (DM_LOCAL); T9…T11 3: externer DRAM-Speicher (DRAM_EXTERN); T9…T11 4: interner Datenspeicher (EM_LOCAL); nur T11 5: Cacheable: Speicher, der Daten an den Cache liefern kann; nur T12/T12.1. 6: Uncacheable: Speicher, der keine Daten an den Cache liefern kann; nur T12/T12.1.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
Memory	≠255: Zusammenhängender freier Speicher in Byte. Bei `MemType` = 5/6 ist der Wert in Einheiten von kBytes angegeben. 255: Fehler oder Speichergröße.
error out	Fehlersignal

Bemerkungen

Beachten Sie die Hinweise zur Fehlerbehandlung in Kapitel 4.3.

5.2 Prozess-Steuerung

Die Steuerung von *ADbasic*- und *TiCoBasic*-Prozessen ist grundsätzlich verschieden:

– ADbasic-Prozesse
– TiCoBasic-Prozesse

5.2.1 ADbasic-Prozesse

Befehle zur Steuerung einzelner *ADbasic*-Prozesse auf dem *ADwin*-System.

Load_Process.VI

`Load_Process.VI` lädt eine Binärdatei als Prozess ins *ADwin*-System.

Eingänge

Device No.	Device No. des Systems.
Filename	Name der Binärdatei mit vollständigem Pfad.
error in	Anschluss des Fehlersignals.
showError	true: bei anstehendem Fehlersignal die Fehlermeldung in einem Meldungsfenster ausgeben. false: keine Fehlermeldung ausgeben.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	1: OK ≠1: Fehler
error out	Fehlersignal

Bemerkungen

Eine Binärdatei wird mit dem Programm *ADbasic* erzeugt.

Der letzte Buchstabe im Dateinamen der Binärdatei (eine Ziffer) gibt an, mit welcher Nummer der Prozess im *ADwin*-System angesprochen wird. Die „0“ steht für Prozess 10.

Bevor Sie den Prozess ins *ADwin*-System laden, müssen Sie sicherstellen, dass dort nicht bereits ein Prozess mit der gleichen Prozessnummer läuft. Wenn das doch der Fall ist, müssen Sie den laufenden Prozess zuerst mit `Stop_Process` stoppen.

Wenn Sie mehrmals Prozesse laden, kann es zu einer Speicherfragmentierung kommen. Beachten Sie die entsprechenden Hinweise im Handbuch *ADbasic*.

Start_Process.VI

`Start_Process.VI` startet einen Prozess.

Eingänge

Device No.	Device No. des Systems.
ProcessNo	Nummer (1…10) des Prozesses.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Der Prozess startet mit der Priorität, mit der er in *ADbasic* kompiliert wurde.

Stop_Process.VI

`Stop_Process.VI` stoppt einen Prozess.

Eingänge

Device No.	Device No. des Systems.
ProcessNo	Nummer (1…10) des Prozesses.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

`Clear_Process.VI` löscht einen Prozess aus dem Speicher.

Eingänge

Device No.	Device No. des Systems.
ProcessNo	Nummer (1…10, 15) des Prozesses.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠1:OK 1: Fehler
error out	Fehlersignal

Bemerkungen

Diese Funktion gibt es nur bei Systemen mit einem Prozessor T9 oder T10 und ist unter Linux nicht verfügbar.

Das Löschen eines Prozesses gibt den zuvor belegten Programmspeicher frei. Um einen Prozess zu löschen, führen Sie die folgenden Schritte aus:

1. Beenden Sie den Prozess mit `Stop_Process.VI`.
2. Überprüfen Sie mit `Process_Status.VI`, ob der Prozess tatsächlich beendet ist.
3. Löschen Sie den Prozess mit `Clear_Process.VI`.

Der Prozess 15 erzeugt das Blinken der LED beim *ADwin-Gold*- und beim *ADwin-Pro*-System und belegt nur wenig Programmspeicher.

Löschen Sie Prozesse in umgekehrter Reihenfolge zum Laden der Prozesse, den zuletzt geladenen Prozess also zuerst. Sie vermeiden damit eine Fragmentierung des Programmspeichers.

Beachten Sie bitte, dass das Löschen eines Prozesses die im Prozess deklarierten Felder NICHT löscht.

Process_Status.VI

`Process_Status.VI` gibt den Status eines Prozesses zurück.

Eingänge

Device No.	Device No. des Systems.
ProcessNo	Nummer (1…10, 15) des Prozesses.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	0: Prozess ist gestoppt. ≠0:Prozess läuft. 255: Fehler
error out	Fehlersignal

Set_Processdelay.VI

`Set_Processdelay.VI` stellt den Parameter Processdelay für einen Prozess ein.

Eingänge

Device No.	Device No. des Systems.
ProcessNo	Nummer (1…10) des Prozesses.
Processdelay	Einzustellender Processdelay-Wert.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Mit dem Parameter `Processdelay` legen Sie die Zeitspanne zwischen zwei Event-Aufrufen eines zeitgesteuerten Prozesses fest (siehe *ADbasic*-Handbuch).

Die Zeitspanne wird in einer Zeiteinheit angegeben, die vom Prozessortyp und von der Priorität des Prozesses abhängt:

Prozessortyp	Prozesspriorität	
	hoch	niedrig
T2, T4, T5, T8	1000 ns	64 µs
T9	25 ns	100 µs
T10	25 ns	50 µs
T11	3,3 ns	0,003 µs = 3,3 ns
T12	1 ns	1 ns
T12.1	1,5 ns	1,5 ns

`Get_Processdelay.VI` gibt den Wert des Parameter Processdelay für einen Prozess zurück.

Eingänge

Device No.	Device No. des Systems.
ProcessNo	Nummer (1…10) des Prozesses.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: Wert des Parameters `Processdelay`. 255: Fehler oder ein gültiger Wert für `Processdelay`.
error out	Fehler

Bemerkungen

Weitere Informationen unter `Set_Processdelay.VI` oder im Handbuch *ADbasic*.

Beachten Sie die Hinweise zur Fehlerbehandlung in Kapitel 4.3.

Für *ADsim*-Nutzer: Der Parameter `Processdelay` entspricht der Basis-Abtastzeit (fixed-step size) in Simulink. Während die Basis-Abtastzeit in Sekunden angegeben wird, ist `Processdelay` ein Vielfaches der Prozessor-Taktzyklus.

5.2.2 TiCoBasic-Prozesse

Bei einer *ADwin*-Hardware mit *TiCo*-Prozessor können Sie eine *TiCoBasic*-Binärdatei als Prozess auf die *ADwin*-Hardware übertragen und starten. In LabVIEW sind dafür folgende Schritte notwendig:

– Erzeugen Sie die Binärdatei mit *TiCoBasic*.

  Die Binärdatei muss zunächst in ein globales Feld `Data_x` des *ADwin*-Prozessors übertragen werden und gelangt dann mit Hilfe eines *ADbasic*-Prozesses weiter zum *TiCo*-Prozessor.

– Erzeugen Sie mit *ADbasic* einen Prozess, der folgende Aufgaben erfüllt:
  - Dimensionierung eines globalen Felds `Data_x` vom Datentyp `Long`. Achten Sie darauf, dass das Feld größer ist als die Binärdatei.
  - Übertragen der Daten aus `Data_x` zum *TiCo*-Prozessor mit dem Befehl **`TiCo_Load`** / **`P2_TiCo_Load`**. Der Prozess startet automatisch.
  - Speichern des Befehls-Rückgabewerts in einer globalen Variablen `Par_x`.
  - Beenden des *ADbasic*-Prozesses mit `Exit`.

  Sie finden Beispiele für solche *ADbasic*-Prozesse im Installationsverzeichnis (siehe Kapitel 3.1) unter
  `<.\ADbasic\samples_ADwin_GoldII>`
  `<.\ADbasic\samples_ADwin_ProII>`

– Erzeugen Sie in *ADbasic* die Binärdatei des Prozesses.

– In LabVIEW sind folgende Schritte notwendig:
  - Laden Sie die *ADbasic*-Binärdatei mit `Load_Process.VI` als Prozess auf die *ADwin*-Hardware, starten den Prozess aber noch nicht.
  - Übertragen Sie die *TiCoBasic*-Binärdatei mit `File2Data.VI` in das richtige Feld `Data_x` des *ADwin*-Prozessors.
  - Starten Sie den *ADbasic*-Prozess mit `Start_Process.VI`.
  - Lesen Sie die globale Variable `Par_x` und prüfen, ob die Übertragung erfolgreich war.

5.3 Übertragung von globalen Variablen

Befehle zur Datenübertragung zwischen PC und *ADwin*-System mit den vordefinierten globalen Variablen `Par_1` ... `Par_80` und `FPar_1` ... `FPar_80`.

5.3.1 Globale Variablen `Par_1` ... `Par_80`

Die globalen Variablen `Par_1` ... `Par_80` auf dem *ADwin*-System haben folgenden Wertebereich:

`Par_1` ... `Par_80`: -2147483648 ... +2147483647
= -2^{31} ... $+2^{31}-1$

Die VIs übertragen ganzzahlige Werte mit 32 Bit Breite.

Set_Par.VI

`Set_Par.VI` setzt eine globale Variable `Par_1` ... `Par_80` auf den gewünschten Wert.

Eingänge

Device No.	Device No. des Systems.
Index	Nummer (1...80) der globalen Variablen `Par_1` ... `Par_80`.
Value	Neuer Wert für die Variable.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Get_Par.VI

`Get_Par.VI` gibt den Wert einer globalen Variablen `Par_1` ... `Par_80` zurück.

Eingänge

Device No.	Device No. des Systems.
Index	Nummer (1...80) der globalen Variablen `Par_1` ... `Par_80`.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
Parameter value	≠255: Wert der gewählten Variablen. 255: Fehler oder ein gültiger Wert der Variablen.
error out	Fehlersignal

Bemerkungen

Beachten Sie die Hinweise zur Fehlerbehandlung in Kapitel 4.3.

Get_Par_Block.VI

`Get_Par_Block.VI` liest eine anzugebende Anzahl an globalen Variablen `Par_1` ... `Par_80`.

Eingänge

Device No.	Device No. des Systems.
Startindex	Nummer (1…80) der ersten gelesenen globalen Variablen `Par_1` ... `Par_80`.
Count	Anzahl der Variablen, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	0: OK ≠0: Fehler
Data	Feld mit den ausgelesenen Werten.
error out	Fehlersignal

Bemerkungen

Die gelesenen Werte werden im Feld Data ab dem Element 0 abgelegt. Wenn beispielsweise Startindex = 5 ist, ist der Wert von `Par_5` im Element 0 des Felds Data abgelegt.

5.3.2 Globale Variablen FPar_1 ... FPar_80

Die globalen Fließkomma-Variablen FPar_1 ... FPar_80 auf dem *ADwin*-System haben folgenden Wertebereich, je nach Prozessortyp:

- FLOAT (bis T11)
  negativ: $-3{,}402823 \cdot 10^{+38} \ldots -1{,}175494 \cdot 10^{-38}$
  positiv: $+1{,}175494 \cdot 10^{-38} \ldots +3{,}402823 \cdot 10^{+38}$
- FLOAT64 (T12/T12.1)
  negativ: $-1{,}7976931348623 15 \cdot 10^{+308} \ldots -2{,}2250738585072014 \cdot 10^{-308}$
  positiv: $+2{,}2250738585072014 \cdot 10^{-308} \ldots +1{,}7976931348623 15 \cdot 10^{+308}$

Die VIs arbeiten unabhängig vom Datentyp auf dem *ADwin*-System. Der Name des VIs verweist auf den Datentyp, der in LabView erwartet bzw. zurückgegeben wird.

Set_FPar.VI

Set_FPar.VI setzt eine globale Variable FPar_1 ... FPar_80 auf den gewünschten Wert vom Datentyp Single.

Eingänge

Device No.	Device No. des Systems.
Index	Nummer (1...80) der globalen Variablen FPar_1 ... FPar_80.
Value	Neuer Wert für die Variable vom Datentyp Single.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Set_FPar überträgt immer einen Fließkommawert mit 32 Bit, auch wenn FPar eine Genauigkeit von 64 Bit haben kann.

Set_FPar_Double.VI

`Set_FPar_Double.VI` setzt eine globale Variable `FPar_1` ... `FPar_80` auf den gewünschten Wert vom Datentyp `Double`.

Eingänge

Device No.	Device No. des Systems.
Index	Nummer (1...80) der globalen Variablen `FPar_1` ... `FPar_80`.
Value	Neuer Wert für die Variable vom Datentyp `Double`.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Bei Prozessoren bis T11 hat die Zielvariable `FPar` auf dem *ADwin*-System nur eine Genauigkeit von 32 Bit.

Get_FPar.VI

`Get_FPar.VI` gibt den `Single`-Wert einer globalen Variablen `FPar` zurück.

Eingänge

Device No.	Device No. des Systems
Index	Nummer (1...80) der globalen Variablen `FPar_1` ... `FPar_80`.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
Parameter value	≠255: Wert der Variablen vom Datentyp `Single`. 255: Fehler oder ein gültiger Wert der Variablen.
error out	Fehlersignal

Bemerkungen

Beim Prozessor T12/T12.1 haben `FPAR`-Variablen im *ADwin*-System doppelte Genauigkeit (64 Bit). `Get_FPar` liefert auch dann einen Wert vom Datentyp `Single` zurück.

Beachten Sie die Hinweise zur Fehlerbehandlung in Kapitel 4.3.

Get_FPar_Double.VI

`Get_FPar_Double.VI` gibt den `Double`-Wert einer globalen Variablen `FPar` zurück.

Eingänge

Device No.	Device No. des Systems
Index	Nummer (1...80) der globalen Variablen `FPar_1` ... `FPar_80`.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
Parameter value	≠255: Wert der gewählten Variablen vom Datentyp `Double`. 255: Fehler oder ein gültiger Wert der Variablen.
error out	Fehlersignal

Bemerkungen

Bis T11 gilt: Beachten Sie, dass Fließkommawerte im *ADwin*-System einfache Genauigkeit (32 Bit) haben. `Get_FPar_Double` liefert auch dann einen Wert vom Datentyp `Double` zurück.

Beachten Sie die Hinweise zur Fehlerbehandlung in Kapitel 4.3.

Get_FPar_Block.VI

`Get_FPar_Block.VI` liest die angegebene Anzahl an globalen Variablen als Feld mit `Single`-Werten.

Eingänge

Device No.	Device No. des Systems.
Startindex	Nummer (1…80) der ersten gelesenen globalen Variablen `FPar_1` ... `FPar_80`.
Count	Anzahl der Variablen, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	0: OK ≠0: Fehler
Data	Feld mit den ausgelesenen Werten vom Datentyp `Single`.
error out	Fehlersignal

Bemerkungen

Die gelesenen Werte werden im Feld Data ab dem Element 0 abgelegt. Wenn beispielsweise Startindex = 9 ist, ist der Wert von `FPar_9` im Element 0 von Data abgelegt.

Get_FPar_Block_Double.VI

`Get_FPar_Block_Double.VI` liest die angegebene Anzahl an globalen Variablen als Feld mit `Double`-Werten.

Eingänge

Device No.	Device No. des Systems.
Startindex	Nummer (1…80) der ersten gelesenen globalen Variablen `FPar_1` ... `FPar_80`.
Count	Anzahl der Variablen, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	0: OK ≠0: Fehler
Data	Feld mit den ausgelesenen Werten vom Datentyp `Double`.
error out	Fehlersignal

Bemerkungen

Die gelesenen Werte werden im Feld Data ab dem Element 0 abgelegt.

Bis T11 gilt: Beachten Sie, dass Fließkommawerte im *ADwin*-System einfache Genauigkeit (32 Bit) haben.

5.4 Übertragung von Datenfeldern (Arrays)

Befehle zur Datenübertragung zwischen PC und *ADwin*-System mit globalen `Data`-Feldern (`Data_1...Data_200`):

- Einfache Data-Felder
- Fifo-Felder
- Data-Felder mit Zeichenfolgen

Achten Sie darauf, dass Sie jedes Feld vor seiner Verwendung im *ADbasic*-Programm mit `DIM` deklarieren müssen (vgl. Handbuch „*ADbasic*“).

5.4.1 Einfache Data-Felder

Sie müssen jedes Data-Feld vor seiner Verwendung unter *ADbasic* deklarieren (vgl. Handbuch „*ADbasic*“):

```
DIM Data_x[n] AS LONG/FLOAT/FLOAT32/FLOAT64
```

Ein Feldelement hat je nach Datentyp folgenden Wertebereich:

– `LONG`	-2147483648 ... +2147483647
– `FLOAT` (bis T11), `FLOAT32`	negativ: $-3{,}402823 \cdot 10^{+38}$ ... $-1{,}175494 \cdot 10^{-38}$ positiv: $+1{,}175494 \cdot 10^{-38}$ ... $+3{,}402823 \cdot 10^{+38}$
– `FLOAT64`	negativ: $-1{,}7976931348623 15 \cdot 10^{+308}$ ... $-2{,}2250738585072014 \cdot 10^{-308}$ positiv: $+2{,}2250738585072014 \cdot 10^{-308}$ ... $+1{,}7976931348623 15 \cdot 10^{+308}$

Die VIs arbeiten unabhängig vom Datentyp auf dem *ADwin*-System. Der Name des VIs verweist auf den Datentyp, der in LabView erwartet bzw. zurückgegeben wird.

Data_Length.VI

`Data_Length.VI` gibt die in *ADbasic* deklarierte Länge eines Felds vom Typ `LONG` oder `FLOAT`/`FLOAT64` zurück, d.h. die Anzahl der Elemente.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1...200) des Felds `Data_1...Data_200`.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠0: Deklarierte Länge des Data-Felds. 0: Fehler oder Data-Feld ist nicht deklariert.
error out	Fehlersignal

Bemerkungen

Bei einem `DATA`-Feld vom Typ `STRING` stellen Sie die Länge der Zeichenfolge mit `String_Length` fest.

SetData_Long.VI

`SetData_Long.VI` überträgt `Long`-Werte von LabVIEW in ein `Data`-Feld im *ADwin*-System.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1…200) des Felds `Data_1` … `Data_200`, in das die Werte übertragen werden. Das Feld `Data` kann den Datentyp `Long`, `Float`, `Float32` oder `Float64` haben.
Data	Feld mit den zu übertragenden Werten vom Datentyp `Long`.
Startindex	Index des ersten übertragenen Feldelements im Feld Data.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Es werden alle Werte aus dem LabVIEW-Feld übertragen (ab dem Startelement). Das `Data`-Feld auf dem *ADwin*-System muss größer sein als die Anzahl der übertragenen Werte.

Die `Long`-Werte des Quellfelds werden bei der Übertragung automatisch in den Datentyp des Zielfelds auf dem *ADwin*-System gewandelt.

GetData_Long.VI

`GetData_Long.VI` überträgt Daten aus einem `Data`-Feld vom *ADwin*-System als `Long`-Daten nach LabVIEW.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1…200) des Data-Felds `Data_1`…`Data_200`. Das Feld `Data` kann den Datentyp `Long`, `Float`, `Float32` oder `Float64` haben.
Startindex	Index des ersten Feldelements im gewählten Feld.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
Data	Feld mit den übertragenen Werten vom Datentyp `Long`.
error out	Fehlersignal

Bemerkungen

Die Werte des Quellfelds werden bei der Übertragung nach LabView automatisch in den Datentyp `Long` gewandelt.

SetData_Float.VI

`SetData_Float.VI` überträgt `Single`-Werte von LabVIEW in ein `Data`-Feld im *ADwin*-System.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1…200) des Data-Felds `Data_1`...`Data_200`, in das die Werte übertragen werden. Das Feld `Data` kann den Datentyp `Long`, `Float`, `Float32` oder `Float64` haben.
Data	Feld mit den zu übertragenden Werten vom Datentyp `Single`.
Startindex	Index des ersten übertragenen Feldelements im Feld Data.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal.

Bemerkungen

Die `Single`-Werte des Quellfelds werden bei der Übertragung automatisch in den Datentyp des Zielfelds auf dem *ADwin*-System gewandelt.

GetData_Float.VI

`GetData_Float.VI` überträgt Daten aus einem `Data`-Feld vom *ADwin*-System als `Single`-Daten nach LabVIEW.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1…200) des Data-Felds `Data_1`...`Data_200`. Das Feld `Data` kann den Datentyp `Long`, `Float`, `Float32` oder `Float64` haben.
Startindex	Index des ersten Feldelements im gewählten Feld.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
Data	Feld mit den übertragenen Werten vom Datentyp `Single`.
error out	Fehlersignal.

Bemerkungen

Die Werte des Quellfelds werden bei der Übertragung nach LabView automatisch in den Datentyp `Single` gewandelt.

SetData_Double.VI

`SetData_Double.VI` überträgt `Double`-Werte von LabVIEW in ein `Data`-Feld im *ADwin*-System.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1…200) des Data-Felds `Data_1`...`Data_200`, in das die Werte übertragen werden.
Data	Feld mit den zu übertragenden Werten vom Datentyp `Double`.
Startindex	Index des ersten übertragenen Feldelements im Feld Data.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Die `Double`-Werte des Quellfelds werden bei der Übertragung automatisch in den Datentyp des Zielfelds auf dem *ADwin*-System gewandelt.

GetData_Double.VI

`GetData_Double.VI` überträgt Daten aus einem `Data`-Feld vom *ADwin*-System als `Double`-Daten nach LabVIEW.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1…200) des Data-Felds `Data_1`...`Data_200`.
Startindex	Index des ersten Feldelements im gewählten Feld.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
Data	Feld mit den übertragenen Werten.
error out	Fehlersignal

Bemerkungen

Die Werte des Quellfelds werden bei der Übertragung nach LabView automatisch in den Datentyp `Double` gewandelt.

Data2File.VI

`Data2File.VI` speichert Daten vom Typ `Long`, `Float`/`Float32` oder `Float64` aus einem `Data`-Feld des *ADwin*-Systems in einer Datei (auf der Festplatte).

Eingänge

Device No.	Device No. des Systems.
Mode	Schreibmodus: 0: Datei überschreiben. 1: Daten an die Datei anhängen.
Filename	Name der Speicherdatei mit vollständigem Pfad.
DataNo	Nummer (1…200) des Felds `Data_1`...`Data_200`, dessen Werte übertragen werden.
Startindex	Index des ersten übertragenen Feldelements.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	0: OK ≠0:Fehler
error out	Fehlersignal

Bemerkungen

Die Funktion kann nicht mit `Data`-Feldern benutzt werden, die als `Fifo`-Feld definiert sind.

Die Daten werden binär gespeichert und zwar entsprechend dem Datentyp, mit dem das `DATA`-Feld auf dem ADwin-System angelegt ist (siehe Tabelle). Wenn die Datei nicht vorhanden ist, wird sie neu angelegt.

Datentyp des `DATA`-Felds	gespeicherter Datentyp
Long	`Long`
Float (bis Prozessor T11) Float32 (Prozessor T12/T12.1)	`Float` (32 Bit)
Float64 (Prozessor T12/T12.1)	`Float64`

File2Data.VI

File2Data.VI kopiert Daten aus einer Datei (aus dem Dateisystem) in ein Data-Feld des *ADwin*-Systems.

Das VI arbeitet nicht unter Linux und MacOS.

Eingänge

Device No.	Device No. des Systems.
Filename	Name der Quelldatei mit vollständigem Pfad.
DataType	Datentyp der Werte in der Quelldatei Wählen Sie eine der folgenden Konstanten: TYPE_LONG: Ganzzahlige Werte (32 Bit). TYPE_FLOAT: Fließkommawerte (32 Bit). TYPE_FLOAT64: Fließkommawerte (64 Bit).
DataNo	Nummer (1…200) des Data-Felds Data_1...Data_200, in das die Werte übertragen werden.
Startindex	Index (≥1) des ersten Feldelements im Zielfeld, das beschrieben wird.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	0: OK ≠0:Fehler
error out	Fehlersignal

Bemerkungen

Alle Werte in der Datei Filename müssen binär in einem der Formate Long, Float/Float32 oder Float64 vorliegen.

Das Data-Feld darf nicht als FIFO definiert sein. Das Data-Feld muss so groß dimensioniert sein, dass alle Werte aus der Datei aufgenommen werden können.

Wenn das Zielfeld einen anderen Datentyp hat als DataType, werden die Werte aus der Quelldatei in das Zielformat gewandelt. Es gibt die Zielformate Long, Float/Float32 und Float64.

5.4.2 Fifo-Felder

Befehle zur Datenübertragung zwischen PC und *ADwin*-System mit globalen `Data`-Feldern (`Data_1`…`Data_200`), die als Fifo deklariert sind.

Sie müssen jedes Fifo-Feld vor seiner Verwendung unter *ADbasic* deklarieren (vgl. Handbuch „*ADbasic*“): `DIM Data_x[n] as TYPE as Fifo`.

Ein FIFO-Feldelement hat je nach Datentyp folgenden Wertebereich:

– `LONG`	-2147483648 … +2147483647
– `FLOAT` (bis T11), `FLOAT32`	negativ: $-3{,}402823 \cdot 10^{+38}$ … $-1{,}175494 \cdot 10^{-38}$ positiv: $+1{,}175494 \cdot 10^{-38}$ … $+3{,}402823 \cdot 10^{+38}$
– `FLOAT64`	negativ: $-1{,}797693134862315 \cdot 10^{+308}$ … $-2{,}2250738585072014 \cdot 10^{-308}$ positiv: $+2{,}2250738585072014 \cdot 10^{-308}$ … $+1{,}797693134862315 \cdot 10^{+308}$

Die VIs arbeiten unabhängig vom Datentyp auf dem *ADwin*-System. Der Name des VIs verweist auf den Datentyp, der in LabView erwartet bzw. zurückgegeben wird.

Beim Arbeiten mit Fifo-Feldern sollten Sie als Faustregel nicht mehr Elemente beschreiben als deklariert sind. Auf keinen Fall sollten Sie mehr Elemente auslesen als vorher beschrieben wurden. Vermeiden Sie dies wie folgt:

– Prüfen Sie mit der Funktion `Fifo_Full`, ob ein Fifo-Feld mindestens so viele Elemente enthält, wie Sie mit `GetFifo_X` auslesen möchten.
– Prüfen Sie mit der Funktion `Fifo_Empty`, ob ein Fifo-Feld mindestens so viele freie Elemente enthält, wie mit `SetFifo_X` beschreiben möchten.

Fifo_Empty.VI

`Fifo_Empty.VI` gibt die Anzahl der freien Elemente eines Fifo-Felds zurück.

Eingänge

Device No.	Device No. des Systems.
FifoNo	Nummer (1…200) des Fifo-Felds `Data_1`…`Data_200`.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
Count	≠255: Anzahl der freien Elemente im Fifo-Feld. 255: Fehler oder eine gültige Anzahl freier Elemente.
error out	Fehlersignal

Bemerkungen

Beachten Sie die Hinweise zur Fehlerbehandlung in Kapitel 4.3.

Fifo_Full.VI

`Fifo_Full.VI` gibt die Anzahl der belegten Elemente eines Fifo-Felds zurück.

Eingänge

Device No.	Device No. des Systems.
FifoNo	Nummer (1…200) des Fifo-Felds `Data_1`...`Data_200`.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
Count	≠255: Anzahl der belegten Elemente im Fifo-Feld. 255: Fehler oder eine gültige Anzahl belegter Elemente.
error out	Fehlersignal

Bemerkungen

Beachten Sie die Hinweise zur Fehlerbehandlung in Kapitel 4.3.

Fifo_Clear.VI

`Fifo_Clear.VI` initialisiert den Schreib- und den Lesezeiger eines Fifo-Felds.

Eingänge

Device No.	Device No. des Systems.
FifoNo	Nummer (1…200) des Fifo-Felds `Data_1`...`Data_200`.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Beim Start eines *ADwin*-Prozesses werden die FIFO-Zeiger eines Felds nicht automatisch initialisiert. Wir empfehlen deshalb für *ADbasic*, `Fifo_Clear` gleich zu Beginn Ihres Programms aufzurufen.

Beachten Sie bei einem *ADsim*-Prozess: Innerhalb des *ADsim*-Prozesses kann keine Initialisierung stattfinden, daher kann eine Initialisierung mit `Fifo_Clear` sinnvoll sein.

Das Initialisieren der FIFO-Zeiger im Programmablauf ist sinnvoll, wenn alle beschriebenen Elemente verworfen werden sollen, z.B. wegen eines Fehlers.

SetFifo_Long.VI

`SetFifo_Long.VI` überträgt `Long`-Werte aus einem Feld in LabVIEW in ein Fifo-Feld im *ADwin*-System.

Eingänge

Device No.	Device No. des Systems.
FifoNo	Nummer (1…200) des Fifo-Felds `Data_1`…`Data_200`.
Data	Feld mit den zu übertragenden Werten vom Datentyp `Long`.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Prüfen Sie zuerst mit der Funktion `Fifo_Empty.VI`, ob noch genügend Platz im Fifo ist.
Übertragen Sie erst dann Werte mit `SetFifo_Long.VI`.

Die `Long`-Werte des Quellfelds werden bei der Übertragung automatisch in den Datentyp des Zielfelds auf dem *ADwin*-System gewandelt.

GetFifo_Long.VI

`GetFifo_Long.VI` überträgt Daten aus einem Fifo-Feld im *ADwin*-System als `Long`-Werte nach LabVIEW.

Eingänge

Device No.	Device No. des Systems.
FifoNo	Nummer (1…200) des Fifo-Felds `Data_1`…`Data_200`.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
Data	Feld mit den übertragenen Werten vom Datentyp `Long`.
error out	Fehlersignal

Bemerkungen

Prüfen Sie zuerst mit der Funktion `Fifo_Full.VI`, ob noch genügend Elemente im Fifo sind.
Lesen Sie erst dann Werte mit `GetFifo_Long.VI` aus.

Die Werte des Quellfelds werden bei der Übertragung nach LabView automatisch in den Datentyp `Long` gewandelt.

SetFifo_Float.VI

`SetFifo_Float.VI` überträgt `Single`-Werte aus einem Feld in LabVIEW in ein Fifo-Feld im *ADwin*-System.

Eingänge

Device No.	Device No. des Systems.
FifoNo	Nummer (1…200) des Fifo-Felds `Data_1`...`Data_200`.
Data	Feld mit den zu übertragenden Werten.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Prüfen Sie zuerst mit der Funktion `Fifo_Empty.VI`, ob noch genügend Platz im Fifo ist.
Übertragen Sie erst dann Werte mit `SetFifo_Float.VI`.

Die `Single`-Werte des Quellfelds werden bei der Übertragung automatisch in den Datentyp des Zielfelds auf dem *ADwin*-System gewandelt.

GetFifo_Float.VI

`GetFifo_Float.VI` überträgt Daten aus einem Fifo-Feld im *ADwin*-System als `Single`-Werte nach LabVIEW.

Eingänge

Device No.	Device No. des Systems.
FifoNo	Nummer (1…200) des Fifo-Felds `Data_1`...`Data_200`.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
Data	Feld mit den übertragenen Werten.
error out	Fehlersignal

Bemerkungen

Prüfen Sie zuerst mit der Funktion `Fifo_Full.VI`, ob noch genügend Elemente im Fifo sind.
Lesen Sie erst dann Werte mit `GetFifo_Float.VI` aus.

Die Werte des Quellfelds werden bei der Übertragung nach LabView automatisch in den Datentyp `Single` gewandelt.

SetFifo_Double.VI

`SetFifo_Double.VI` überträgt `Double`-Werte aus einem Feld in LabVIEW in ein Fifo-Feld im *ADwin*-System.

Eingänge

Device No.	Device No. des Systems.
FifoNo	Nummer (1…200) des Fifo-Felds `Data_1`…`Data_200`.
Data	Feld mit den zu übertragenden Werten.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
error out	Fehlersignal

Bemerkungen

Prüfen Sie zuerst mit der Funktion `Fifo_Empty.VI`, ob noch genügend Platz im Fifo ist.
Übertragen Sie erst dann Werte mit `SetFifo_Double.VI`.

Die `Double`-Werte des Quellfelds werden bei der Übertragung automatisch in den Datentyp des Zielfelds auf dem *ADwin*-System gewandelt.

GetFifo_Double.VI

`GetFifo_Double.VI` überträgt Daten aus einem Fifo-Feld im *ADwin*-System als `Double`-Werte nach LabVIEW.

Eingänge

Device No.	Device No. des Systems.
FifoNo	Nummer (1…200) des Fifo-Felds `Data_1`…`Data_200`.
Count	Anzahl der Feldelemente, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠255: OK 255: Fehler
Data	Feld mit den übertragenen Werten.
error out	Fehlersignal

Bemerkungen

Prüfen Sie zuerst mit der Funktion `Fifo_Full.VI`, ob noch genügend Elemente im Fifo sind.
Lesen Sie erst dann Werte mit `GetFifo_Double.VI` aus.

Die Werte des Quellfelds werden bei der Übertragung nach LabView automatisch in den Datentyp `Double` gewandelt.

5.4.3 Data-Felder mit Zeichenfolgen

Befehle zur Übertragung von Zeichenfolgen mit globalen `Data`-Feldern (`Data_1...Data_200`).

Sie müssen jedes Data-Feld vor seiner Verwendung unter *ADbasic* deklarieren (vgl. Handbuch „*ADbasic*“): `DIM Data_x[n] as STRING`.

SetData_String.VI

`SetData_String.VI` überträgt eine Zeichenfolge von LabVIEW in ein `Data`-Feld im *ADwin*-System.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1…200) des Felds `Data_1...Data_200`.
Data	Feld mit der zu übertragenden Zeichenfolge.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠-1: Anzahl der übertragenen Zeichen. -1: Fehler
error out	Fehlersignal

GetData_String.VI

`GetData_String.VI` überträgt eine Zeichenfolge von einem `Data`-Feld im *ADwin*-System nach LabVIEW.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1…200) des Felds `Data_1...Data_200`.
MaxCount	Max. Anzahl der Zeichen, die übertragen werden.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠-1: Anzahl der übertragenen Zeichen. -1: Fehler
Data	Feld mit der übertragenen Zeichenfolge.
error out	Fehlersignal

Bemerkungen

Eine Zeichenfolge in einem `Data`-Feld wird mit einer Ende-Kennung beendet (ASCII-Wert 0). Das Auslesen der Zeichenfolge endet genau an dieser Stelle. Der Rückgabewert ist die Anzahl der gelesenen Zeichen ohne Ende-Kennung.

Wenn MaxCount größer ist als die in *ADbasic* definierte Zeichenzahl des Strings, erhälten Sie über das `Error_msg.VI` den Fehler "`Data too small`".

Wenn Sie einen großen Wert für MaxCount angeben, hat die Funktion eine entsprechend lange Ausführungszeit, selbst wenn der übertragene

String nur kurz ist.
Bei zeitkritischen Anwendungen mit großen Strings kann es günstiger sein, wie folgt vorzugehen:

- Sie stellen die tatsächliche Anzahl der Zeichen im String mit dem `String_Length.VI` fest.
- Sie lesen den String mit `Getdata_String.VI` und übergeben die tatsächliche Zeichnanzahl als MaxCount.

String_Length.VI

`String_Length.VI` gibt die Länge einer Zeichenfolge in einem `Data`-Feld im *ADwin*-System zurück.

Eingänge

Device No.	Device No. des Systems.
DataNo	Nummer (1…200) des Felds `Data_1`…`Data_200`.
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
return value	≠-1: Länge der Zeichenfolge im Data-Feld. -1: Fehler
error out	Fehlersignal

Bemerkungen

`String_Length` zählt die Zeichen in einerm `Data`-Feld bis zum Auftreten der ersten Endekennung (ASCII-Wert 0). Die Endekennung wird nicht als Zeichen gezählt.

5.5 Fehlerbehandlung

Error_msg.VI

`Error_msg.VI` gibt eine Fehlermeldung zu einem Fehlersignal aus.

Eingänge

show error dialog?	`true`: Fehlermeldung ausgeben `false`: Keine Funktion
error in	Anschluss des Fehlersignals.

Ausgänge

error ?	true: Fehler liegt an false: kein Fehler
error out	Fehlersignal

Bemerkungen

Der Ausgang „error ?“ ist geeignet, um im Falle eines Fehlers das Programm zu stoppen, z. B. in einer Schleife.

Sie finden eine Liste aller Fehlermeldungen im Abschnitt A.2 im Anhang.

Set_Language.VI

`Set_Language.VI` stellt ein, in welcher Sprache Fehlermeldungen angezeigt werden.

Eingänge

Device No.	Device No. des Systems.
Language	gewählte Sprache: 0: Windows-Spracheinstellung übernehmen. 1: englisch 2: deutsch
error in	Anschluss des Fehlersignals.

Ausgänge

Device No. out	Die am Eingang „Device No.“ angelegte Nummer.
error out	Fehlersignal

Ausgänge

Wenn die Windows-Spracheinstellung übernommen wird, wird für jede Sprache außer deutsch die Einstellung „englisch“ verwendet.

5.6 Hilfsberechnungen

Volt2Value.VI

`Volt2Value.VI` berechnet aus einem Volt-Wert den entsprechenden Digit-Wert für den DAC.

Eingänge

Input voltage	Einzelner Spannungswert in Volt.
Input voltage array	Feld mit Spannungswerten in Volt.
Input voltage array 2dim.	2-dimensionales Feld mit Spannungswerten in Volt.
Converter resolution	Auflösung des DAC.
Voltage range	Spannungsbereich des DAC.

Ausgänge

Output value	Digit-Wert zum Eingang „Input voltage“.
Output value array	Digit-Werte zum Eingang „Input voltage array“.
Output value array 2dim.	Digit-Werte zum Eingang „Input voltage array 2dim.“

Bemerkungen

Das VI berechnet aus einem Spannungswert in Volt den passenden Digit-Wert. Wenn der Digit-Wert an den DAC übergeben wird, erzeugt der DAC den Spannungswert am Analogausgang. Converter resolution und Voltage range stellen Sie je nach verwendetem DAC ein.

Die Eingänge „Input voltage“ können einzeln oder parallel verwendet werden.

Value2Volt.VI

`Value2Volt.VI` berechnet aus einem Digit-Wert eines ADC den entsprechenden Volt-Wert.

Eingänge

Input value	Einzelner Digit-Wert.
Input value array	Feld mit Digit-Werten.
Input value array 2dim.	2-dimensionales Feld mit Digit-Werten.
Converter resolution	Auflösung des ADC.
Voltage range	Spannungsbereich des ADC.

Ausgänge

Output voltage	Volt-Wert zum Eingang „Input value“.
Output voltage array	Volt-Werte zum Eingang „Input value array“.
Output voltage array 2dim.	Volt-Werte zum Eingang „Input value array 2dim.“.

Bemerkungen

Das VI berechnet aus einem vom ADC gelesenen Digit-Wert die Spannung in Volt, die am ADC anliegt. Converter resolution und Voltage range stellen Sie je nach verwendetem ADC ein.

Die Eingänge „Input voltage“ können einzeln oder parallel verwendet werden.

Anhang

A.1 Index der Funktionen

A.2 Fehlermeldungen

Fehler-Nr.	Fehlermeldung
0	Kein Fehler
1	Timeout Fehler beim Schreiben zum ADwin-System.
2	Timeout Fehler beim Lesen vom ADwin-System.
10	Die Device-Nummer ist nicht erlaubt.
11	Die Device-Nummer ist nicht konfiguriert.
15	Funktion für dieses Device nicht erlaubt.
20	Inkompatible Versionen von ADwin-Betriebssystem, Treiberdatei adwin32.dll und / oder ADbasic-Binärdatei.
100	Das Data-Feld ist zu klein.
101	Das Fifo-Feld ist zu klein oder nicht genug Daten.
102	Das Fifo-Feld hat nicht genug Daten.
103	Das Data-Feld ist nicht deklariert.
150	Nicht genug Speicher oder Speicherzugriffsfehler.
200	Datei konnte nicht gefunden werden.
201	Temporäre Datei konnte nicht erstellt werden.
202	Die Datei ist keine ADbasic Binärdatei.
203	Die Datei ist ungültig.[1]
204	Die Datei ist keine BTL.
205	ADbasic Binärdatei ist für den falschen Prozessor oder beschädigt.
2000	Netzwerk Fehler (TCP/IP).
2001	Netzwerk timeout.
2002	Falsches Passwort.
3000	USB Gerät nicht gefunden.
3001	Gerät nicht gefunden.

1. Möglicherweise fehlt bei `<ADwin5.btl>` die „memory table“, eine andere Datei wurde zu `<ADwin5.btl>` umbenannt oder diese ist beschädigt