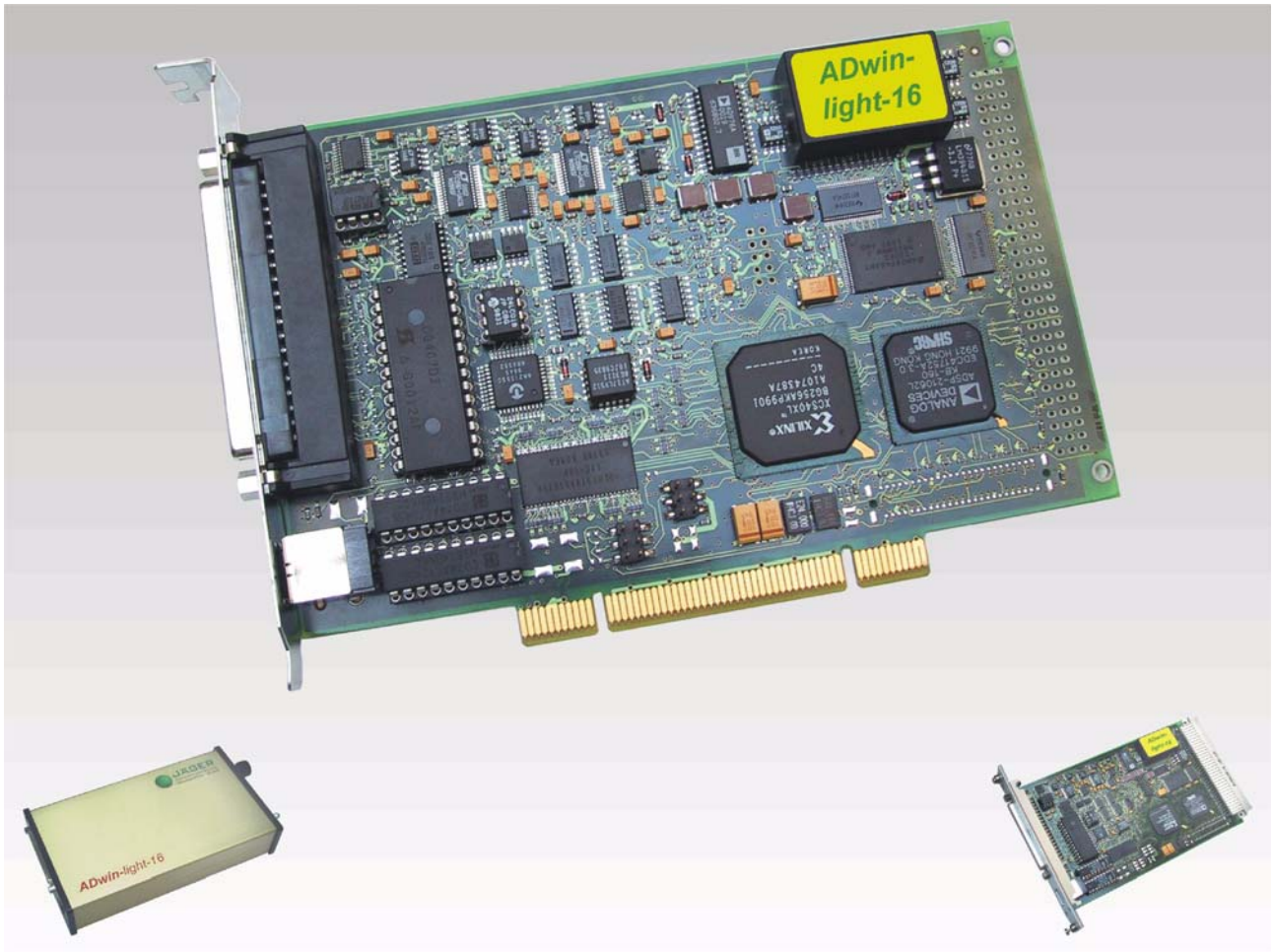
JÄGER
Computergesteuerte
Messtechnik GmbH

ADwin-light-16

Handbuch



ADwin-light-16 , Handbuch Version 3.5, November 2013

Hier finden Sie immer einen Ansprechpartner für Ihre Fragen:

Hotline:	(0 62 51) 9 63 20
Fax:	(0 62 51) 5 68 19
E-Mail:	info@ADwin.de
Internet	www.ADwin.de

JÄGER
Computergesteuerte
Messtechnik GmbH

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

Inhaltsverzeichnis

1 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.

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

<C:\ADwin\…>

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

Programmtext

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

```
Var_1
```

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 Betrieb Ihres *ADwin-light-16*-Systems. Es wird ergänzt durch

– das Handbuch „*ADwin* Installation“, das die Installation von Software und Hardware beschreibt.
  Beginnen Sie hier die Installation Ihres Systems!

– die Beschreibung des Konfigurationsprogramms *ADconfig*, mit dem Sie die Kommunikation von der jeweiligen Schnittstelle zu Ihrem *ADwin-light-16*-Gerät einrichten.

– das Handbuch *ADbasic*, das alle Befehle für den gleichnamigen Compiler enthält sowie das Funktionsprinzip von *ADwin*-Systemen erläutert.

  Die Online-Hilfe von *ADbasic* enthält die gleichen Informationen.

– die Installations- und Befehlsbeschreibungen für die Treiber der gängigen Entwicklungsumgebungen.

– das Handbuch „*ADwin* HSM-24V“ (ein Modul am LS-Bus).

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 Systembeschreibung

2.1 *ADwin* Systemkonzept

ADwin-Systeme garantieren den schnellen und zeitlich präzisen Ablauf von Messdatenerfassungs- und Automatisierungsaufgaben mit sehr schnellen Echtzeitanforderungen. Das bietet eine ideale Basis für Anwendungen wie:

- sehr schnelle digitale Regler
- sehr schnelle Steuerungen
- Datenerfassung mit sehr schneller Online-Analyse der Messdaten
- Überwachung komplexer Triggerbedingungen und vieles mehr

ADwin-Systeme sind optimiert für Abläufe mit **kurzen Prozesszykluszeiten** von einer Millisekunde bis zu wenigen Mikrosekunden.

Systemmerkmale

Das *ADwin*-System besitzt analoge und digitale Ein- und Ausgänge, einen schnellen Prozessor (32-Bit-Floating-Point Signalprozessor) und lokalen Speicher. Der Prozessor übernimmt die gesamte Echtzeitverarbeitung im System. Die Anwendungen **laufen eigenständig** und unabhängig vom PC und dessen Auslastung.

Prozessor

Der Prozessor des *ADwin*-Systems **verarbeitet** *jeden* **Messwert** *sofort*.

In einem Zyklus können die Zustände von Eingängen erfasst, diese mit beliebigen mathematischen Funktionen verarbeitet und auf dieses Ergebnis reagiert werden, und das sogar bei sehr kurzen Prozesszykluszeiten von wenigen Mikrosekunden. Es ergibt sich eine perfekte und logische Arbeitsteilung: auf dem PC läuft ein Programm zur Visualisierung von Daten, zur Eingabe und Bedienung der Abläufe mit Netzwerk- und Datenbankzugriffen, während gleichzeitig auf dem Prozessor des *ADwin*-Systems alle Aufgaben, die Echtzeit erfordern, abgearbeitet werden.

Echtzeitkern

Das Betriebssystem für den DSP des *ADwin*-Systems wurde auf das Erreichen kürzester Reaktionszeiten optimiert. Dieser Echtzeitkern verwaltet parallele Prozesse, die im **Multitasking-Verfahren** gleichzeitig ablaufen können. Prozesse mit niedriger Priorität werden in einem Zeitscheibenverfahren verwaltet. Prozesse mit hoher Priorität unterbrechen bei ihrer Anforderung alle niedrigpriorisierten Prozesse und werden sofort vollständig ausgeführt (präemptives Multitasking). Hochpriorisierte Prozesse werden zeitgesteuert oder von externen Events (Trigger) ausgelöst.

Zeitsteuerung

Für den präzisen Aufruf hochpriorisierter Prozesse sorgt der im System integrierte **Timer**. Er hat eine Auflösung von 25 Nanosekunden (3,3ns ab Prozessor T11). Zu beachten ist die extrem kurze Reaktionszeit von nur 300 Nanosekunden beim Wechsel von einem niedrig- zu einem hochpriorisierten Prozess. Ein ständig laufender Kommunikationsprozess ermöglicht einen kontinuierlichen Datenaustausch zwischen dem *ADwin*-System und dem PC auch während laufenden Anwendungen. Dabei hat die Kommunikation keinen Einfluss auf die Echtzeitfähigkeit des *ADwin*-Systems, trotzdem können jederzeit Daten ausgetauscht werden.

ADbasic

Das Echtzeit-Entwicklungstool *ADbasic* ermöglicht die einfache und schnelle Erstellung von zeitkritischen Programmen für *ADwin*-Systeme. *ADbasic* ist eine **integrierte Entwicklungsumgebung** unter Windows mit Möglichkeiten zum Online-Debugging. Die gewohnte, leicht erlernbare BASIC-Befehlssyntax wurde um Funktionen für den direkten Zugriff auf Ein- und Ausgänge sowie zur Prozesssteuerung und zur Kommunikation mit dem PC erweitert.

Die Kommunikation zwischen *ADwin*-System und PC

Schnittstellen

Das *ADwin*-System ist mit dem PC über eine **USB- oder Ethernet-Schnittstelle** verbunden. Über diese Schnittstelle kann das *ADwin*-System nach dem Einschalten vom PC gebootet werden. Nach dem Booten erwartet das *ADwin*-Betriebssystem Kommandos vom PC, die es abarbeitet.

Befehlsverarbeitung

Es gibt zwei Arten von Kommandos: Zum einen Kommandos, die nur Daten vom PC an das *ADwin*-System schicken, wie z.B. „Prozess laden“, „Prozess starten“ oder „Parameter setzen“, zum anderen Kommandos, die von dem *ADwin*-System eine Antwort erwarten, wie z.B. „Variablen lesen“ oder „Datensätze lesen“. Beide Arten von Kommandos werden vom *ADwin*-System sofort bearbeitet beziehungsweise sofort und

vollständig beantwortet. Das *ADwin*-System schickt nie unaufgefordert Daten an den PC. Die Datenübertragung an den PC ist immer nur die Antwort auf ein Kommando vom PC. Dadurch wird die Einbindung des *ADwin*-Systems in die unterschiedlichsten Programmiersprachen und messtechnischen Standardsoftwarepakete sehr erleichtert, denn diese müssen nur in der Lage sein, eine Funktion aufzurufen und den Rückgabewert zu verarbeiten.

Software-Schnittstellen

Unter den aktuellen Windows-Versionen stehen eine **DLL-** und eine **ActiveX- Schnittstelle** zur Verfügung. Darauf basierend gibt es Treiber für die folgenden **Entwicklungsumgebungen**:
.NET, Visual Basic, Visual-C, C/C++, C#, Delphi, VBA (Excel, Access, Word), TestPoint, LabVIEW / LabWINDOWS, Agilent VEE (HP-VEE), InTouch, DIA*dem*, DASYLab, SciLab, MATLAB.

Treiber für Linux, Mac OS und Java stehen ebenfalls zur Verfügung.

Die einfache, kommandoorientierte Kommunikation mit dem *ADwin*-System ermöglicht es, dass mehrere Windows Programme in Abstimmung miteinander gleichzeitig auf das gleiche *ADwin*-System zugreifen. Dies ist vor allem bei der Programmentwicklung und bei der Inbetriebnahme ein großer Vorteil.

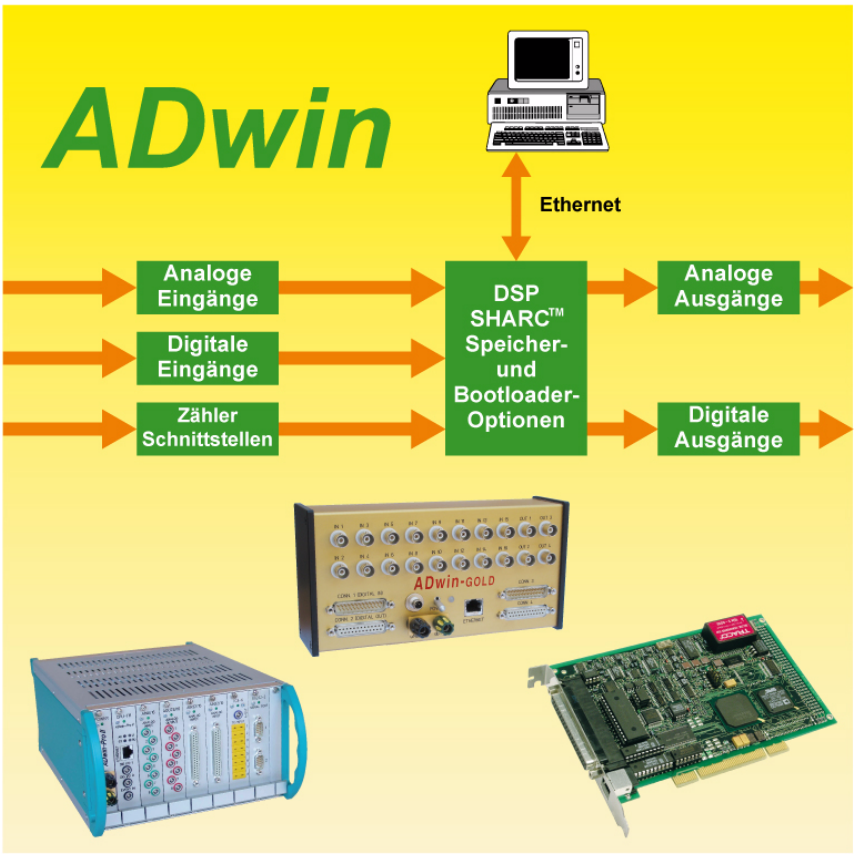


Abb. 1 – Konzept der ***ADwin***-Systeme

2.2 *ADwin-light-16*

Prozessor und Speicher

Das System *ADwin-light-16* besitzt den digitalen **32 Bit-Signalprozessor** ADSP 21062 (SHARC) von Analog Devices mit Floating-Point- und Integer-Verarbeitung. Er übernimmt die gesamte Messwerterfassung, Online-Verarbeitung und Signalausgabe und kann in Verbindung mit dem A/D-Wandler jeden Messwert mit Abtastraten bis zu 100 kHz sofort verarbeiten; ab Rev. B sind optional Abtastraten bis 500 kHz verfügbar.

Der **interne Speicher mit 256 KiB** hat eine sehr kurze Zugriffszeit von 25 ns und nimmt das *ADwin*-Betriebssystem, die *ADbasic*-Prozesse und alle Variablen auf.

Für maximale Zugriffsgeschwindigkeit liegen alle Ein- und Ausgänge direkt im Adressbereich des DSP. Zum Zwischenspeichern größerer Datenmengen nutzt der DSP einen **externen Speicher** (SDRAM) von **8 MiB** (Rev. B: **16 MiB**).

Analoge Eingänge

In einer 37-poligen Sub-D-Buchse sind **8 analoge Eingänge** bereit gestellt, die mit einem gemeinsamen Multiplexer verbunden sind. Das Eingangssignal wird mit einem 16 Bit Analog-Digitalwandler (ADC) konvertiert (siehe Abbildung unten). Ab Rev. B ist die automatische Wandlung mehrerer Eingänge mit einer Ablaufsteuerung verfügbar.

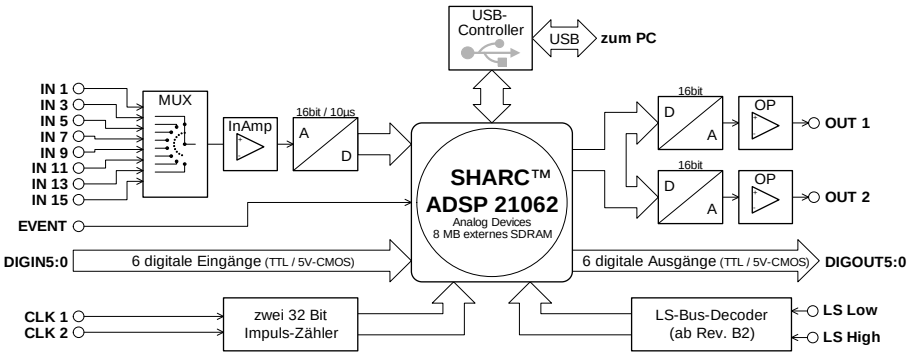


Abb. 2 – Funktionsschema (mit USB-Schnittstelle)

Analoge Ausgänge

ADwin-light-16 verfügt über **2 Analogausgänge** mit 16 Bit Auflösung und einem Ausgangsspannungsbereich von -10 V...+10 V. Per Software können Sie die Ausgabe der Spannung aller DAC synchronisieren sowie kalibrieren. Das Ausgangssignal durchläuft zur Glättung einen Tiefpassfilter mit einer Eckfrequenz von f_g = 700 kHz.

Digitale Ein- und Ausgänge

Auf der 37-poligen Sub-D-Buchse stehen **6 digitale Eingänge und 6 digitale Ausgänge** zur Verfügung. Die Ein- bzw. Ausgänge sind TTL-kompatibel. Daneben sind Eingänge für 2 Impuls-/Ereigniszähler mit 32 Bit vorhanden.

Trigger-Eingang

ADwin-light-16 besitzt einen Trigger-Eingang (EVENT, siehe auch Kapitel 5.2 “Digitale Ein- und Ausgänge“). Hiermit können Prozesse durch ein Signal (Trigger) ausgelöst und sofort vollständig abgearbeitet werden (siehe *ADbasic*-Handbuch oder Online-Hilfe: Kapitel „Prozesse im Betriebssystem“).

Eine serielle Schnittstelle (LS-Bus ab Rev. B2, siehe Seite 17) erlaubt den Anschluss von bis zu 15 zusätzlichen Modulen.

Standardlieferumfang

Der Standardlieferumfang eines *ADwin-light-16*-Systems beinhaltet

- *ADwin-light-16*-Gerät
- USB- oder Ethernet-Anschlusskabel, Länge 1,8 Meter
- *ADwin*-CD
- Handbuch „Treiber-Installation“
- das vorliegende Hardware-Handbuch.

Die Variante mit externem Gehäuse (*L16-EXT*) beinhaltet zusätzlich:

- Power-Adapter: Ein dreipoliges, verpolungssicheres Stromversorgungskabel (PC-intern) an einem Slotblech mit Steckbuchse.
- Stromversorgungskabel (zwischen Slot-Blech und *L16-EXT*)

Bauformen mit USB

Es gibt *ADwin-light-16* als Basisversion mit USB-Anschluss in verschiedenen Bauformen:

PC-Einsteckkarte (*L16-PCI*)

19“-Einschub (*L16-EURO*)

externes Gehäuse (*L16-EXT*)

Abb. 3 – Bauformen

Liefervarianten mit Ethernet

Die Bauformen *EURO* und *EXT* sind alternativ mit USB- oder mit 10/100 MBit Ethernet-Schnittstelle lieferbar. Die Liefervarianten der Basisversion sind in der folgenden Tabelle aufgeführt.

Bauform	**Schnittstelle**	
	USB	**Ethernet**
PC-Einsteckkarte	L16-PCI	–
19“-Einschub (Euro)	L16-EURO	L16-EURO-ENET
Externes Gehäuse	L16-EXT	L16-EXT-ENET

Abb. 4 – Liefervarianten der Basisversion *ADwin-light-16*

Beachten Sie bitte, dass die Spannungsversorgung für die Bauformen unterschiedlich ist:

– +5 Volt für *L16-PCI* und *L16-EURO*

– +10 ... +18 Volt für *L16-EXT*, ab Rev B +10 ... +36 Volt

2.2.1 Bestelloptionen (nicht nachrüstbar)

Das *ADwin-light-16*-System kann mit folgenden Optionen ausgerüstet sein:

– *L16-CO1*: Zähleroption mit einem 32 Bit Vor-/Rückwärtszähler mit Vier-Flanken-Auswertung für Inkrementalgeber (Beschreibung siehe Seite 25).

– *L16-DIO1*: Das Erweiterungsmodul (Beschreibung s. Seite 28) enthält
  - 32 digitale Ein-/Ausgänge (programmierbar in Gruppen zu 8)
  - Einen SSI-Decoder (erst ab Rev. B).
  - Eine CAN-Schnittstelle (High-Speed, alternativ Low-Speed)
  - Zwei 32 Bit Vor-/Rückwärtszähler zur Impuls-, Periodendauer- und Tastverhältnis-Messung sowie einer Vier-Flanken-Auswertung zum Anschluss von Inkrementalgebern.

– *L16-DIO2*: Das Erweiterungsmodul (Beschreibung s. Seite 46) enthält
  - 32 digitale Ein-/Ausgänge (programmierbar in Gruppen zu 8).
  - Einen SSI-Decoder.
  - Zwei 32 Bit Vor-/Rückwärtszähler zur Impuls-, Periodendauer- und Tastverhältnis-Messung sowie einer Vier-Flanken-Auswertung zum Anschluss von Inkrementalgebern.

– *L16-DIO3*: Das Erweiterungsmodul (Beschreibung s. Seite 46) enthält 32 digitale Ein-/Ausgänge (programmierbar in Gruppen zu 8).

  Die Option DIO3 ist mit der Option L16-CO1 kombinierbar.

– *L16-PWM1*: Das Erweiterungsmodul (Beschreibung s. Seite 57) enthält 1 PWM-Ausgang und 1 SPI-Schnittstelle (Master).

  Im Unterschied zu den vorigen Bestelloptionen ist *L16-PWM1* sowohl nachrüstbar als auch mit allen anderen Bestelloptionen kombinierbar.

– *L16-Boot*: Flash-EPROM-Bootloader zum eigenständigen Betrieb ohne PC (Beschreibung siehe Seite 61). Nur lieferbar in Verbindung mit Ethernet-Schnittstelle.

– *L16-Mount*: Gehäuseumbau für *L16-EXT* zur Hutschienen-Montage in einem Schaltschrank mit isolierten Befestigungshaltern.

Beachten Sie, dass die Zähler der Erweiterungskarten nicht zusätzlich zur Verfügung stehen, sondern jeweils die Zähler der Basisversion ersetzen. Die Zähler unterschiedlicher Erweiterungen können deswegen nicht gemeinsam genutzt werden.

2.2.2 Zubehör

Für das *ADwin-light-16*-System ist das folgende Zubehör (auch nachträglich) erhältlich; weitere Beschreibung siehe Seite 62:

ADbasic

– *ADbasic*, Echtzeit-Entwicklungstool für alle *ADwin*-Systeme

– Kabel-Stecker für eine externe Spannungsversorgung (nur für *L16-EXT*)

– externes Netzteil *ADwin-light-16*-pow (u.a. erforderlich für Notebook-Betrieb)

3 Betriebliche Umgebung

Die Platine des *ADwin-light-16*-Systems darf nur in einem geschlossenen Gehäuse betrieben werden (bei Bauform *L16-EXT* bereits erfüllt).

Je nach Variante und Ausrüstung kann das Gerät in 19“-Systemgehäusen, in Schaltschränken oder im mobilen Betrieb (z.B. im Kfz) eingesetzt werden (siehe Kapitel 2.2.1f: Bestelloptionen und Zubehör).

Das *ADwin-light-16*-Gerät **muss geerdet werden**, um

– einen Massebezugspunkt für die Elektronik herzustellen und

– Störungsenergie auf die Erde ableiten zu können.

Verbinden Sie dazu die GND-Buchse, die intern mit der Masse und dem Gehäuse verbunden ist, über ein kurzes impedanz-armes Masseband mit dem zentralen Erdungspunkt Ihrer Anlage.

Galvanische Kopplung

Die Liefervariante mit USB-Schnittstelle hat über diese eine galvanische Verbindung zum PC sowie ggf. über die Stromversorgung.

Bei der Liefervariante mit Ethernet-Schnittstelle sind die Datenleitungen galvanisch entkoppelt, die Massepotenziale sind jedoch gekoppelt, weil die Schirmung des Ethernet-Steckers (RJ-45) mit GND verbunden ist.

Ausgleichströme ausschließen

Ausgleichsströme, die über das Gehäuse oder die Schirmung abfließen, beeinflussen das Messsignal.
Wollen Sie Ausgleichströme vermindern, müssen Sie darauf achten, dass die Wirkung des Schirmes erhalten bleibt, indem Sie geeignete Maßnahmen zur Ableitung von Störungen treffen, wie z. B. das Auflegen des Schirms kurz vor dem Eintritt in den Schaltschrank. Je häufiger Sie die Schirmung auf dem Weg zur Maschine erden, desto besser ist die Schirmwirkung.

Verwenden Sie für die **Signalleitungen** Kabel mit beidseitig aufgelegtem Schirm. Auch hier sollte das Ableiten von Störungen über das Gehäuse mit der Verwendung von Schirmklemmen reduziert werden.

Schutzkleinspannung

Das *ADwin-light-16*-System wird intern mit einer Spannung von +5 V und ±15 V gegen GND betrieben und stellt von dieser Seite keine Gefahr für Leib und Leben dar. Für den Betrieb mit einem externem Netzteil gelten die Angaben des Herstellers.

Umgebungsklima

ADwin-light-16 ist für den Betrieb in trockenen Räumen konzipiert. Am Einbauort (im PC oder 19“-Gehäuse) sollen eine Umgebungstemperatur von +5 °C ... +50 °C und eine relative Luftfeuchte von 0 ... 80 % (nicht kondensierend, s.a. Anhang) vorhanden sein.

Gehäusetemperatur

Die Gehäusetemperatur (Oberflächentemperatur) der Liefervariante *L16-EXT* darf auch unter extremen betrieblichen Bedingungen, z.B. im Schaltschrank oder bei direkter Sonneneinstrahlung, +55 °C nicht überschreiten. Es besteht sonst die Gefahr, dass Schäden am Gerät entstehen oder nicht definierte Daten (Werte) ausgegeben werden, die unter ungünstigen Umständen zu Schäden in ihrer Anlage führen können.

4 Inbetriebnahme der Hardware

Schließen Sie bei der Inbetriebnahme keine Kabel an das *ADwin-light-16*-System an, bevor Sie nicht folgende Schritte durchgeführt haben:

1. Software-Installation / Einbau im PC oder 19“-Gehäuse:

   Folgen Sie der Anleitung im Handbuch: „*ADwin*-Installation“.
2. Richten Sie die betriebliche Umgebung ein wie in Kapitel 3 beschrieben.
3. Lesen Sie das Kapitel 5 “Ein- und Ausgänge“ in diesem Handbuch.
4. Schließen Sie erst jetzt Signalleitungen an die Ein- und Ausgänge an.

Hinweise

Sicherstellen der Spannungsversorgung

Achten Sie auf eine zuverlässige Spannungsversorgung.
Im Standardlieferumfang betrifft das den PC, ansonsten auch das externe Netzteil, bei Betrieb im Fahrzeug die Batteriespannung.

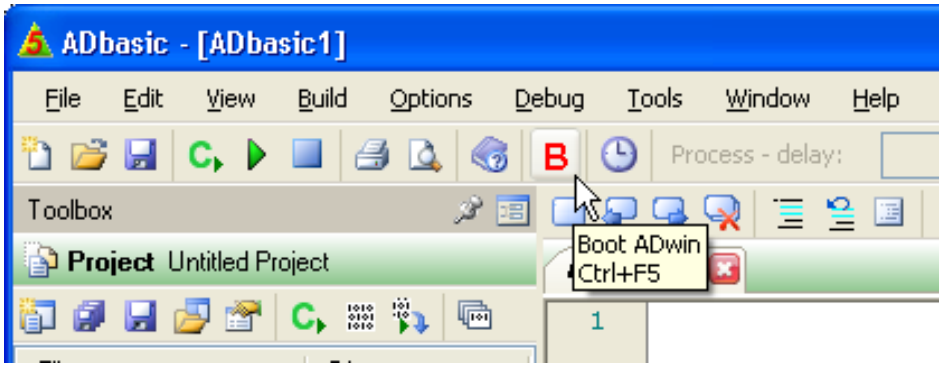
Achten Sie bei der Verwendung strombegrenzender Netzteile darauf, dass beim Einschalten der Strombedarf ein Mehrfaches des Betriebsstroms betragen kann. Genaue Angaben finden Sie bei den Technischen Daten (Anhang).

Bei Ausfall der Betriebsspannung gehen alle ungesicherten Daten verloren. Nicht definierte Daten (Werte) können unter ungünstigen Umständen zu Schäden in Ihrer Anlage führen.

Vermeiden Sie die direkte Berührung unisolierter Teile zum Schutz vor elektrostatischer Aufladung.

Datenverbindung testen

Starten Sie *ADbasic* und booten das *ADwin*-System durch Anklicken der Boot-Schaltfläche B.



Die Anzeige in der Statuszeile: „`ADwin is booted`“ zeigt an, dass das Betriebssystem richtig geladen ist und *ADbasic* eine Verbindung zum *ADwin*-System herstellen kann.

Programme mit *ADbasic*

Die Programmierung von *ADwin*-Systemen ist im *ADbasic*-Handbuch (oder der Online-Hilfe) ausführlich beschrieben.
Beginnen Sie mit Programmbeispielen im *ADbasic*-Tutorial.

5 Ein- und Ausgänge

Anschlüsse

Das System *ADwin-Light-16* besitzt folgende Steckverbindungen (Pinbelegung auf der nächsten Seite):

– Anschlussbuchse für USB oder Ethernet

– 37-polige Sub-D-Buchse `ADwin I/O CONNECTOR` für
  - 8 analoge Eingänge, 2 analoge Ausgänge
  - jeweils 6 digitale Ein- und Ausgänge
  - 1 digitaler Trigger-Eingang
  - 2 Impuls-/Ereigniszähler mit 32 Bit
  - Stromversorgungs-Ausgang +5V; bei *L16-PCI* auch ±12V

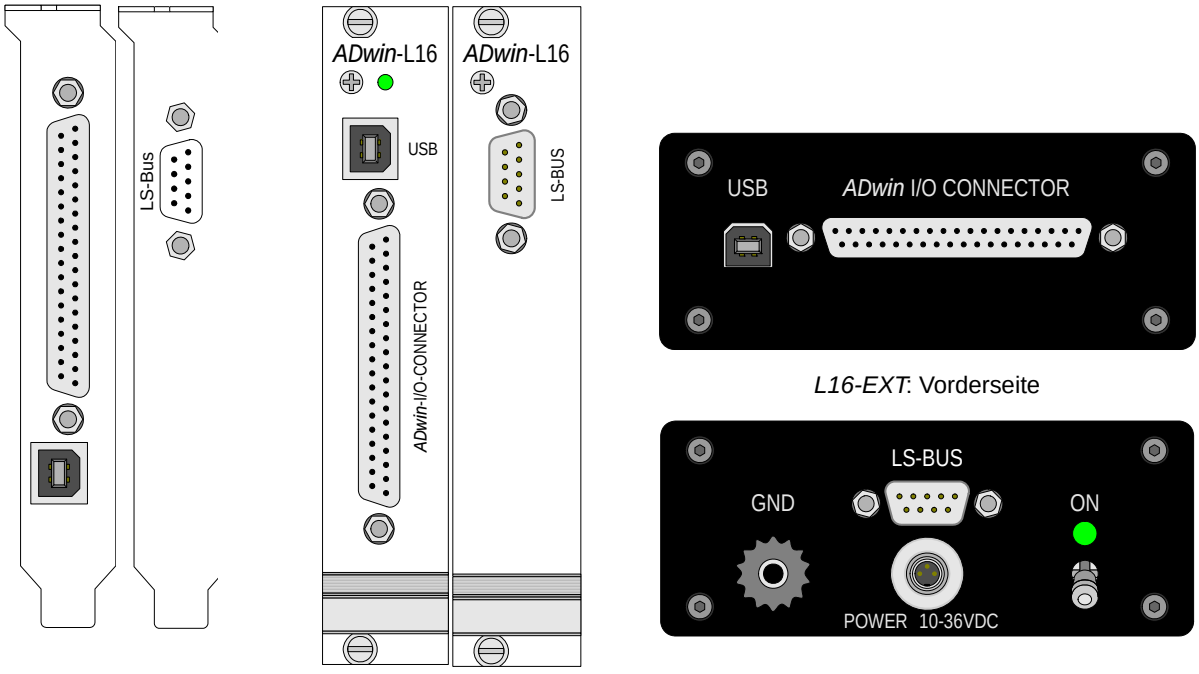
  Wenn die Hardware mit der Option PWM1 ausgerüstet ist, sind manche Pins der Sub-D-Buchse doppelt belegt. Die Pinbelegung zur Option PWM1 finden Sie in Kapitel 10 auf Seite 57.

– 9-polige Sub-D-Buchse `LS-BUS` für die LS-Bus-Schnittstelle, ab Rev. B2.

– Die Bauform *L16-EXT* besitzt zusätzlich an der Hinterseite eine GND-Buchse (siehe Seite 7, Erdung), eine Strom-Eingangsbuchse und einen manuellen Ein-/Ausschalter.

Alle Ein- und Ausgänge dürfen nur im Bereich der angegebenen Spezifikation betrieben werden (siehe Anhang A.1 Technische Daten). Im Zweifel wenden Sie sich bitte an den Hersteller des Gerätes, das Sie an das *ADwin-light-16*-System anschließen wollen.

Offene Eingänge können zu Fehlern führen – vor allem in einer nicht störungsfreien Umgebung. Zu Ihrer Sicherheit legen Sie nicht benutzte Eingänge möglichst nah an der Sub-D-Buchse auf einen definierten Pegel (z.B. GND). Trennen Sie diese Eingänge auch von offenen Leitungen.
Ausnahme hierzu ist der Event-Eingang, der bereits einen internen Pull-up-Widerstand (10 kΩ) besitzt.



L16-PCI *L16-EURO* *L16-EXT*: Vorderseite *L16-EXT*: Hinterseite

Abb. 5 – Steckverbindungen *ADwin-light-16*

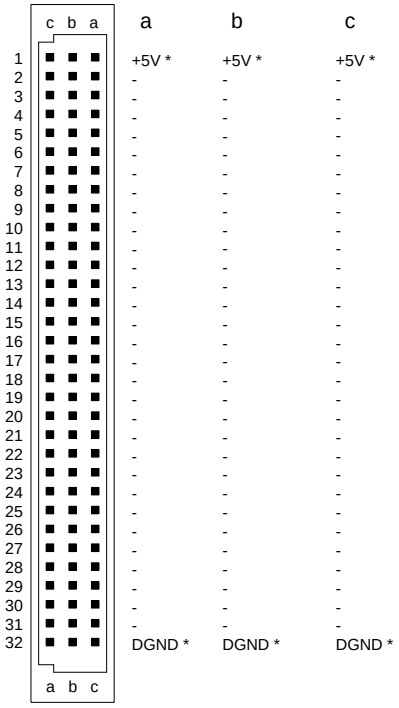


Abb. 6 – *L16-EURO* VG96-Federleiste zur Stromversorgung (Buchse)

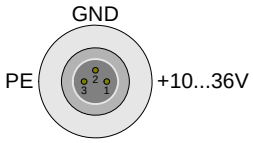


Abb. 7 – *L16-EXT* Strom-Eingangsstecker (Stecker)

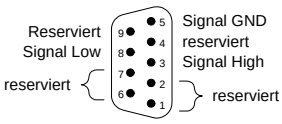
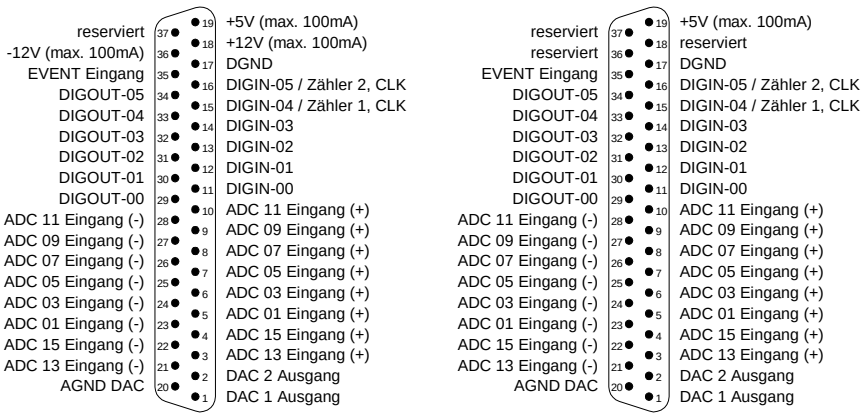


Abb. 8 – Pin-Belegung LS-BUS (Buchse)



L16-PCI

L16-EURO und *L16-EXT*

Abb. 9 – Pin-Belegung Ein-/Ausgänge (Buchse)

Für eine schnelle und einfache Programmierung gibt es im Compiler *ADbasic* Standardbefehle, die einfaches Messen bzw. Ausgeben von Daten ermöglichen (siehe ADC, Seite 67 und DAC, Seite 66). Verwenden Sie andere Befehle (z.B. direkter Registerzugriff) erst, wenn extrem zeitkritische oder besondere Aufgaben es erfordern.

Standardbefehle

Genauere Angaben zu den analogen sowie den digitalen Ein- und Ausgängen finden Sie in den folgenden Kapiteln.

5.1 Analoge Ein- und Ausgänge

Die Bauform *L16-EXT* muss geerdet werden, um Messungen störungsfrei durchführen zu können. Verbinden Sie dazu die GND-Buchse über ein impedanz-armes Masseband mit dem zentralen Erdungspunkt Ihrer Anlage. Bei der PCI- als auch der EURO-Variante obliegt die korrekte Erdung dem PC bzw. dem 19“-System.

Erdung

Beim *L16-EXT*-System ist das Gehäuse über die GND-Leitung des Stromversorgungskabels als auch über die GND-Leitung des USB-Kabels mit dem Schutzleiter des PC verbunden oder über die Schirmung des Ethernet-Steckers mit GND.

5.1.1 Analoge Eingänge

Das System hat 8 analoge Messeingänge, die über einen gemeinsamen Multiplexer zum 16 Bit Analog-Digitalwandler (ADC) geführt werden. Die Einschwingzeit des Multiplexers beträgt 6,5 µs beim maximalen Spannungssprung von 20V.

Multiplexer

Die Eingänge sind ausschliesslich mit ungeraden Zahlen (ADC 01, ADC 03, ... ADC 15) nummeriert, was bei der Programmierung zu berücksichtigen ist.

Die analogen Eingänge sind differentiell. Für jeden Messkanal sind jeweils ein Plus- und ein Minuseingang vorhanden, zwischen denen die Spannungsdifferenz gemessen wird (jedoch nicht potentialfrei).

Differentiell

Beachten Sie, dass zusätzlich zu Plus- und Minuseingang des Kanals immer eine Masseverbindung zwischen dem System (GND) und der Signalquelle bestehen muss.

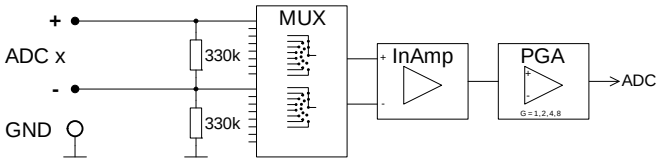


Abb. 10 – Eingangsbeschaltung eines analogen Eingangs

Das Signal am Multiplexer-Ausgang wird mit einem 16 Bit Analog-Digitalwandler (ADC) konvertiert, siehe auch Abb. 2 – Funktionsschema (mit USB-Schnittstelle). Die Wandlungszeit beträgt 10 µs (ab Rev. B per Software umschaltbar auf 2µs) bei einer Auflösung von 305 µV.

16 Bit-Messung

Der Befehl **ADC**() führt mit einem ADC eine komplette Messung auf einem analogen Eingang durch. So berücksichtigt dieser Befehl z.B. die Einschwingzeit des Multiplexers und stellt einwandfreie Messungen sicher (siehe auch ADC, Seite 67).

Komplette Messung

Ab Rev. B können die Signale ausgewählter analoger Eingänge mit einem einzigen Befehl (nacheinander) gewandelt werden. Die Eingänge werden mit **Seq_Init** ausgewählt, die Wandlung mit **Start_Conv** gestartet und die Ergebnisse mit **Seq_Read** gelesen.

Rev. B mit Ablaufsteuerung

5.1.2 Analoge Ausgänge

Das System hat 2 analoge Ausgänge (DAC1, DAC2), siehe Abb. 9 – Pin-Belegung Ein-/Ausgänge (Buchse). Den Ausgängen ist je ein eigener Digital-Analog-Wandler (DAC) zugeordnet.

Der Standardbefehl **DAC**(Nummer,Wert) prüft jeden Wert auf die Über- und Unterschreitung des 16-Bit Wertebereiches. Liegt der Wert innerhalb des 16-Bit Wertebereiches, wird der angegebene Wert auf dem Ausgang Nummer ausgegeben. Liegt er außerhalb, wird der Maximal- bzw. Minimalwert ausgegeben (siehe auch DAC, Seite 66).

DAC-Befehl

5.1.3 Berechnungsgrundlagen

Spannungsbereich

Das *ADwin-light-16*-System arbeitet bei den analogen Ein- und Ausgängen mit einem Spannungsbereich von –10 V bis +10 V (bipolar 10 V).

Zuordnung von Digit und Spannung

Die 65536 (2^{16}) Digits sind den jeweiligen Spannungsbereichen der ADCs und DACs so zugeordnet, dass

– 0 (Null) Digit der maximalen negativen Spannung und

– 65535 Digit der maximalen positiven Spannung

entspricht.

Der Wert für 65.536 Digit, genau 10 Volt, liegt gerade außerhalb des Mess-bereichs, womit sich für die 16 Bit-Wandlung ein maximaler Spannungswert von 9,999695 Volt ergibt.

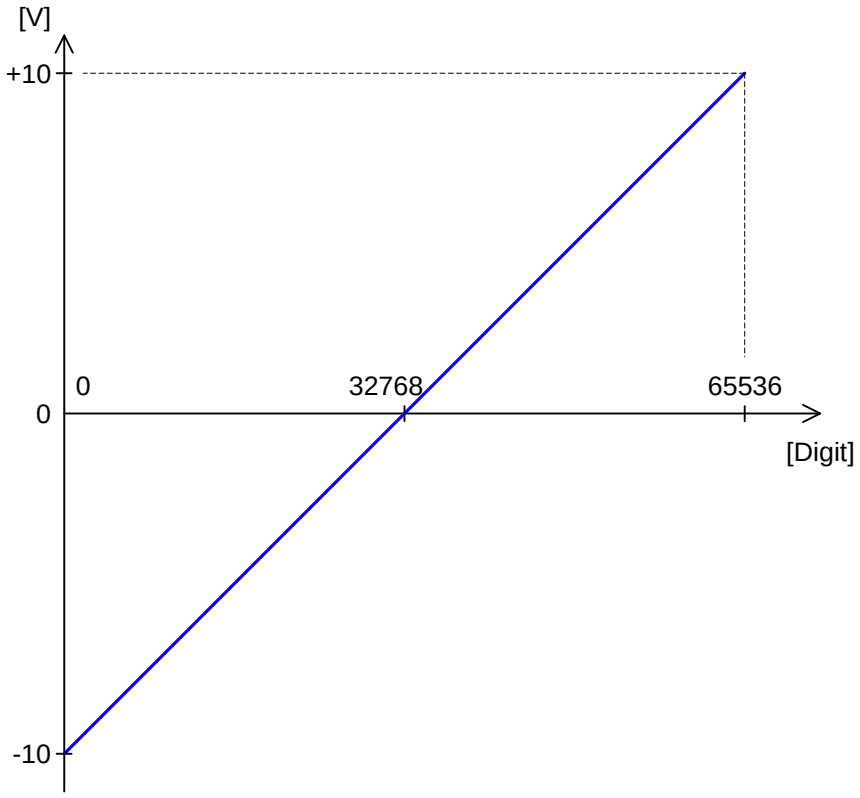


Abb. 11 – Nullpunktverschiebung bei Standardeinstellung bipolar 10 Volt

Die bipolare Einstellung führt zu einer Nullpunktverschiebung, die im folgenden auch als Offset bezeichnet wird.

Beim Spannungsbereich –10 V ... +10 V gilt U_{OFF} = –10 V

Least Significant Bit

Die Quantisierungsstufe U_{LSB} ist die kleinste digital darstellbare Spannungsdifferenz und ist gleich der Spannung des niederwertigsten Bit (Least Significant Bit). Das U_{LSB} entspricht dem 2^{16}-ten Teil von 20 V gleich 305,175 µV.

Der gemessene 16 Bit-Wert des ADC wird im unteren Wort der Speicherzelle zurückgeliefert. Dort muss sich auch ein auszugebender DAC-Wert befinden.

Bit-Nr.	31...16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
32 Bit-Speicher	oberes Wort	16 Bit-Wert des ADC / DAC im unteren Wort															

Umrechnung Digit in Spannung

DAC

Für einen DAC gilt:

$$U_{OUT} = Digits \cdot U_{LSB} + U_{OFF}$$

$$Digits = \frac{U_{OUT} - U_{OFF}}{U_{LSB}}$$

Für einen ADC gilt:

ADC

$$Digits = \frac{U_{IN} - U_{OFF}}{U_{LSB}}$$

$$U_{IN} = Digits \cdot U_{LSB} + U_{OFF}$$

Toleranzbereiche

Geringe Abweichungen zu den rechnerischen Werten können innerhalb der Toleranzbereiche einzelner Bauteile liegen. Es gibt zwei charakteristische Abweichungsarten, die in diesem Handbuch angegeben sind (in LSB):

– Die integrale Nicht-Linearität (INL) beschreibt die maximale Abweichung von der Geraden über den gesamten Eingangsspannungsbereich. **INL**

– Die differentielle Nicht-Linearität (DNL) beschreibt die maximale Abweichung von der Breite einer Quantisierungsstufe. **DNL**

5.2 Digitale Ein- und Ausgänge

Digitale Ein- / Ausgänge

Auf der 37-poligen Sub-D-Buchse stehen 6 digitale Eingänge (Digin 00 … Digin 05) und 6 digitale Ausgänge (DIGOUT 00…DIGOUT 05) zur Verfügung. Die Pin-Belegung der Sub-D-Buchse entnehmen Sie Abbildung 9 auf Seite 10.

Die Eingänge Digin 04 und Digin 05 sind gleichzeitig als Zählereingang geschaltet und können alternativ für einen der beiden Zwecke (Digital- oder Zählereingang) genutzt werden. Dies gilt nicht für die DIO1-Erweiterung (siehe Seite 28) oder die DIO2-Erweiterung (siehe Seite 46).

Die digitalen Eingänge sind TTL-kompatibel und gegen Überspannung nicht geschützt. Beschalten Sie keine freien Anschlüsse, die als „reserviert“ gekennzeichnet sind. Diese sind Änderungen oder Erweiterungen vorbehalten; Nichtbeachten kann das System beschädigen.

Trigger-Eingang (EVENT)

Das *ADwin-light-16*-System besitzt einen externen Trigger-Eingang (EVENT). Ein externes Signal (Trigger) mit steigender Flanke an diesem Eingang kann Prozesse auslösen, die sofort und vollständig abgearbeitet werden (siehe *ADbasic*-Handbuch oder -Online-Hilfe, Kapitel „Prozesse im Betriebssystem“).

Programmierung

Die Funktionalität der Digitalkanäle wird mit *ADbasic*-Befehlen komfortabel programmiert:

Bereich	Befehle
Einzelnen Digitaleingang lesen.	`Digin`
Alle Digitaleingänge lesen.	`Digin_Word`
Alle Digitalausgänge setzen.	`Digout_Word`
Einen Digitalausgang auf High setzen.	`Set_Digout`
Einen Digitalausgang auf Low setzen.	`Clear_Digout`

Die Befehle sind ab Seite 78 oder in der Online-Hilfe beschrieben.

5.3 Impuls-/Ereigniszähler

Das *ADwin-light-16*-System stellt 2 Impuls-/Ereigniszähler mit 32 Bit zur Verfügung, die Sie per Software einzeln oder gemeinsam konfigurieren und auslesen können.

Bei den Lieferoptionen *L16-CO1*, *L16-DIO1* und *L16-DIO2* werden die hier beschriebenen durch andere Zähler ersetzt. Sie finden die entsprechende Beschreibung in Kapitel 7 “CO1-Zählererweiterung“, Kapitel 8 “DIO1-Erweiterung“ oder Kapitel 9 “DIO2- / DIO3-Erweiterung“.

5.3.1 Hardware

Die Grafik zeigt den Aufbau eines einzelnen Zählers.

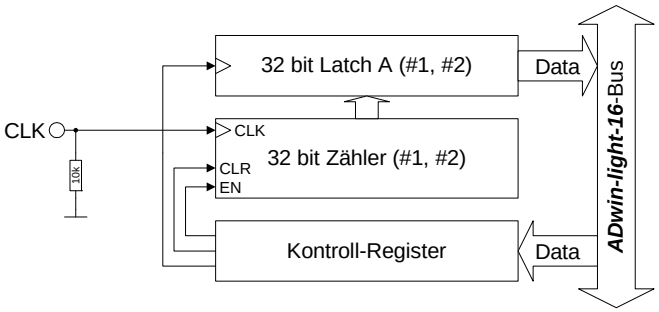


Abb. 12 – Schema des Impuls-/Ereigniszählers

Die Inkrementalzähler werden extern getaktet, d.h. sie erhöhen ihren Zählerstand um eins bei jeder positiven Flanke am Takteingang (CLK). Beide Zähler haben ein Latch A, das den Zählerstand programmgesteuert zum Auslesen übernehmen kann.
Die Zähler werden mit besonderen *ADbasic*-Befehlen über Kontrollregister gesteuert, siehe Kurzreferenz auf Seite 15.

Eingänge beschalten

Die Takteingänge liegen auf den Pins 15 und 16 (siehe Abb. 9 – Pin-Belegung Ein-/Ausgänge (Buchse), Seite 10); für die korrekte Funktion sind TTL-kompatible Signale erforderlich. Einzelheiten und Grenzwerte finden Sie bei den Technischen Daten im Anhang. Beide Eingänge können alternativ als digitaler Signaleingang eingesetzt werden (siehe auch Kapitel 5.2).

Die Eingänge der Impuls-/Ereigniszähler besitzen einen Pull-down-Widerstand. Dennoch können offene Eingänge vor allem in einer nicht störungsfreien Umgebung zu Fehlern führen. Legen Sie deshalb sicherheitshalber die nicht benutzten Eingänge auf einen definierten Pegel (z.B. GND).

5.3.2 Software

Include-Datei

Die Zähler-Funktionen werden mit *ADbasic*-Befehlen komfortabel programmiert. Die Befehle sind in einer Include-Datei enthalten; binden Sie diese Datei am Beginn des Programms ein mit der Befehlszeile

```
#Include ADWL16.INC
```

Die Zähler-Befehle der folgenden Tabelle ab Seite 95 und in der Online-Hilfe vollständig beschrieben:

Zähler-Nr. Bit	2 1	1 0	Kommentar
Cnt_Clear( )	0	0	ohne Einfluss
	1	1	Zähler löschen *
Cnt_Enable( )	0	0	Zähler sperren
	1	1	Zähler freigeben (laufende Zähler beachten)
Cnt_Latch( )	0	0	ohne Einfluss
	1	1	Zählerstand in Latch A übernehmen *
*Nach der Befehlsausführung werden gesetzte Bits automatisch wieder gelöscht. Bei anderen Befehlen müssen gesetzte Bits durch einen neuen Befehlsaufruf gelöscht werden.			

Abb. 13 – Zähler-Befehle Kurzreferenz

Zähler-Nr. Bit	2 1	1 0	Kommentar
Cnt_ReadLatch(#)			Latch A auslesen (# = Zähler-Nr. 1, 2)
Cnt_Read(#)			Zählerstand in Latch A übernehmen und auslesen (# = Zähler-Nr. 1, 2)
*Nach der Befehlsausführung werden gesetzte Bits automatisch wieder gelöscht. Bei anderen Befehlen müssen gesetzte Bits durch einen neuen Befehlsaufruf gelöscht werden.			

Abb. 13 – Zähler-Befehle Kurzreferenz

Sie können jeden Zähler einzeln oder beide Zähler gemeinsam konfigurieren.

Initialisierung

Initialisieren Sie die Zähler bitte in dieser Reihenfolge:

1. Gewünschten Zähler sperren (**Cnt_Enable**)

   Der Befehl **Cnt_Enable** spricht alle Zähler gemeinsam an. Auch wenn Sie nur einen Zähler sperren (Bit = 0) möchten, müssen Sie alle anderen Zähler konfigurieren, deren Zustand unverändert bleiben soll (Bit = 1).
   Beim Freigeben des Zählers gilt dies entsprechend.

2. Zähler löschen (**Cnt_Clear**)
3. Zähler freigeben (**Cnt_Enable**)

Für die Verarbeitung der Werte im *ADbasic*-Programm übertragen Sie die Werte in Latch A und lesen sie dort aus.

5.3.3 Auswerten des Zählerinhalts

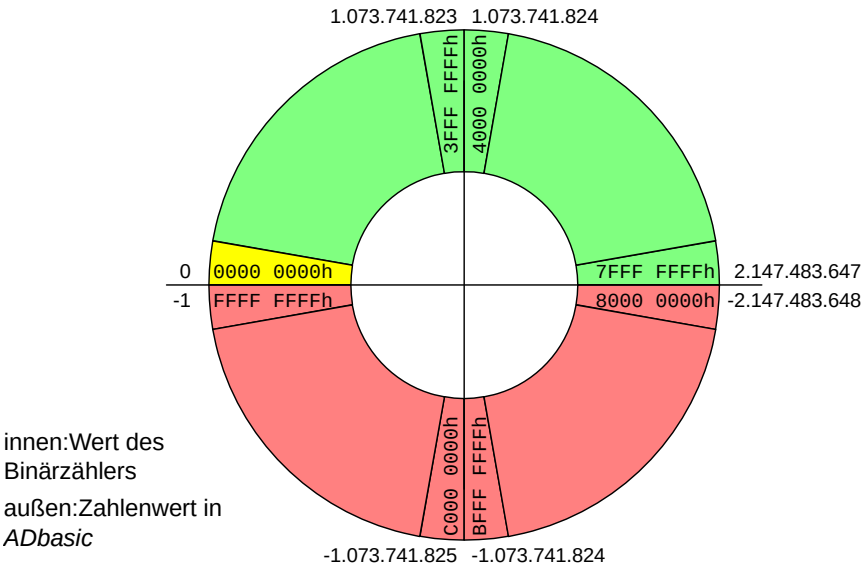
Die Binärzähler erzeugen 32 Bit-Werte. Unterscheiden Sie die Auswertung dieser Binärwerte (z.B. Differenzen) klar von der Darstellung der Werte als Dezimalzahl.

unveränderte Bitmuster

Für die Auswertung des Zählerinhalts benötigen Sie dessen 32 Bit-Werte ohne Veränderung, insbesondere bei der Bildung von Differenzen. Dies gewährleisten unter *ADbasic* nur Variablen vom Typ **Long**.

Die Anzeige von 32-Bit-Binärwerten in *ADbasic* sorgt oft für Verwirrung, weil hierbei der vorzeichenlose Zählerwert als vorzeichenbehafteter Dezimalwert dargestellt wird (siehe Zahlenkreis unten). Als Folge entsteht in der Darstellung ein Übergang zwischen positivem und negativem Zahlenbereich, der jedoch für die Auswertung des Binärwerts ohne Bedeutung ist.

Zahlenkreis



innen:Wert des Binärzählers

außen:Zahlenwert in *ADbasic*

Abb. 14 – Zahlenkreis als Interpretation von Zählerwerten

Zur Vollständigkeit wird nachfolgend die Umwandlung zur Dezimalzahl beschrieben: Das höchste Bit (MSB) des Binärwerts stellt das Vorzeichen dar. Bei positivem Vorzeichen werden die weiteren 31 Bits direkt als Zahl interpretiert, d.h. die Absolutwerte von

Dezimalzahl und Binärwert sind gleich. Bei negativem Vorzeichen werden die 31 Bits invertiert, eine Eins addiert und als Zahl interpretiert (2er-Komplement); negative Dezimalzahlen haben deshalb einen anderen Absolutwert als der zugehörige Binärwert.

Bilden Sie Differenzen nur mit Integer-Zahlen (**LONG**).

„Überlauf“

Berücksichtigen Sie bei der Programmierung, dass ein „Überlauf“ zwischen dem Auslesen von zwei Zählerständen - d.h. der aktuelle Zählerstand „überrundet“ den zuletzt gelesenen - nicht erfasst wird.

Ein solcher Überlauf tritt bei einer Frequenz von 20 MHz nach etwas mehr als 3½ Minuten ein, bei 5 MHz nach über 14 Minuten.

5.4 LS-Bus

ADwin-light-16 stellt einen Anschluss für den LS-Bus auf einem 9-poligen Sub-D-Verbinder (Buchse LS-BUS) zur Verfügung; die Pin-Belegung ist auf Seite 10 dargestellt.

Der LS-Bus (Low Speed) ist ein bidirektionaler, serieller Bus mit 5 MHz Taktrate. Der Bus ist eine Eigenentwicklung für den Anschluss externer Module. Als erstes steht der der Modultyp HSM-24V zur Verfügung, mit dem 24 V-Signale auf 32 digitalen Kanälen verarbeitet werden können.

Der Bus ist als Linienverbindung aufgebaut, d.h. die *ADwin*-Schnittstelle und bis zu 15 LS-Bus-Module sind jeweils über Zweipunktverbindungen miteinander verbunden. Am letzten LS-Bus-Modul muss der Busabschluss aktiviert sein. Die maximale Buslänge beträgt 5 m.

Die Module am LS-Bus werden mit *ADbasic*-Befehlen programmiert, die über die LS-Bus-Schnittstelle am *ADwin*-System geschickt werden. Die Befehle sind meistens modulspezifisch und sind im Handbuch des LS-Bus-Moduls beschrieben (oder in der Online-Hilfe).

Bei den Bauformen L16-PCI und L16-Euro liegt der Sub-D-Verbinder LS-BUS auf einer separaten Blende. Falls der LS-Bus nicht genutzt wird, kann das Verbindungskabel von der Platine abgezogen und die Blende entfernt werden.
Achten Sie beim erneuten Anschließen darauf, dass die Verbindungsbuchse alle 10 Stifte des Platinensteckers aufnimmt und das Kabel nicht verdreht ist.

5.5 Zeitkritische Aufgaben

Zeitkritische Aufgaben

Für extrem zeitkritische Aufgaben können Sie Befehle einsetzen, mit denen Sie direkt auf die Steuer- und Datenregister der Hardware zugreifen (**Peek** oder **Poke**, siehe *ADbasic*-Handbuch oder -Online-Hilfe). Diese Register liegen im Speicheradressbereich des Prozessors (memory mapped). Die Befehle ermöglichen auch eine Optimierung der Programmstruktur.

Im Gegensatz zu den Standardbefehlen **ADC**( ) und **DAC**( ) verzichten die Befehle für den Direktzugriff auf jegliche Prüfroutinen. Vor deren Benutzung sollten Sie deshalb über genaue Kenntnisse der Programmierung und der Zeit- und Funktionsabläufe in einem Analog-Digitalwandler verfügen, da Sie nun hardware-nah programmieren.

Analoge Ein- und Ausgänge

ADC()

Führen Sie anstelle des Standardbefehls **ADC()** die folgenden *ADbasic*-Befehle in dieser Reihenfolge aus:

Programmstruktur

```
Set_Mux()
...                          'Einschwingzeit abwarten
Start_Conv()
Wait_EOC()                   'das Wandlungsende abwarten
ReadADC()
```

Legen Sie zwischen die Ausführung der Befehle **Start_Conv**( ) und **Set_Mux**( ) durch weitere Programmierschritte einen ausreichenden Zeitabstand, um die Einschwingzeit des Multiplexers zu berücksichtigen (siehe Seite 74 oder Online-Hilfe).

Nutzen Sie die unten stehenden Wartezeiten, z.B. für Rechenoperationen, und sparen Sie somit Rechenzeit ein:

- Die Einschwingzeit des Multiplexers beträgt 6,5µs beim maximalen Spannungssprung von 20V.
- Wandlungszeit des 16 Bit-ADC: 10µs; optional ab Rev. B: 2µs.

Hardware-Adressen der Steuer- und Datenregister

Mit den Befehlen **Peek** und **Poke** (siehe *ADbasic*-Handbuch oder -Online-Hilfe) können Sie direkt auf Steuer- und Datenregister zugreifen. Damit können Sie den Prozessablauf beschleunigen, beispielsweise:

ADC

- eine Messung sehr schnell ausführen.

DAC

- sehr schnell ein oder mehrere DAC-Register beschreiben und synchron die Ausgabe aktivieren.

Stellen Sie sicher, dass die Werte der Befehlsparameter innerhalb der zulässigen Bereichsgrenzen liegen.

Die Hardware-Adressen der Register finden Sie nachfolgend, gruppiert nach analogen Eingängen, analogen Ausgängen, digitalen Ein-/Ausgängen und Zählern.

Beachten Sie, dass manche Register mehrere Vorgänge gleichzeitig beeinflussen können.

Adresse [HEX]	Funktion	Bit Nr. 31-16	15-10	9	8	7	6	5	4	3	2	1	0	Kommentar
20 40 00 00	Multiplexer auf Eingangskanal setzen (ADC 01... ADC 15)	-	-	-	-	-	-	0	0	0	n	n	n	„nnn“ binär = 0...7 dezimal; gewählter Kanal = nnn*2 + 1
20 40 00 10	Konvertierung starten: ADC#1	-	-	-	-	-	-	-	1	1	1	1	s	s = 0 : Konvertierung starten s = 1 : kein Einfluss
20 40 00 20	Konvertierungs-Status (EOC): ADC#1	-	-	-	-	-	-	-	-	-	-	-	e	e = 0 : Konvertierung beendet e = 1 : Konvertierung läuft
20 40 00 30	Register auslesen: ADC#1	-	x	x	x	x	x	x	x	x	x	x	x	x : Ergebnis der Konvertierung
20 40 01 00	Register auslesen und Konvertierung starten: ADC#1	-	x	x	x	x	x	x	x	x	x	x	x	

Abb. 15 – ADC Hardware-Adressen der Steuer- und Datenregister

Adresse [HEX]	Funktion	Bit Nr. 31-16	15-10	9	8	7	6	5	4	3	2	1	0	Kommentar
20 40 00 10	Konvertierung starten: alle DAC synchron	-	-	-	-	-	-	-	1	1	s	1	1	s = 0 : Konvertierung starten s = 1 : kein Einfluss
20 40 00 50	Register nur beschreiben: DAC #1	-	x	x	x	x	x	x	x	x	x	x	x	x : zu konvertierender Digitalwert
20 40 00 60	Register nur beschreiben: DAC #2	-	x	x	x	x	x	x	x	x	x	x	x	
20 40 02 00	Register beschreiben und sofort Konvertierung starten: DAC #1	-	x	x	x	x	x	x	x	x	x	x	x	
20 40 02 10	Register beschreiben und sofort Konvertierung starten: DAC #2	-	x	x	x	x	x	x	x	x	x	x	x	

Abb. 16 – DAC Hardware-Adressen der Steuer- und Datenregister

Adresse [HEX]	Funktion	Bit Nr. 31:16	15:6	5	4	3	2	1	0	Kommentar
20 40 00 B0	Eingangs-Register Digin-05...Digin-00	-	-	x	x	x	x	x	x	x : eingelesener Digitalwert
20 40 00 C0	Ausgangs-Register DIGOUT-05 ... DIGOUT-00	-	-	x	x	x	x	x	x	x : auszugebender Digitalwert
20 40 00 C4	DIGOUT Bit-SET-Register	-	-	0	0	0	0	0	0	kein Einfluss
		-	-	1	1	1	1	1	1	Bit setzen
20 40 00 C8	DIGOUT Bit-CLEAR-Register	-	-	0	0	0	0	0	0	kein Einfluss
		-	-	1	1	1	1	1	1	Bit löschen

Abb. 17 – DIO Hardware-Adressen der Steuer- und Datenregister

Adresse [HEX]	Funktion	Bit Nr. 31:16	15:6	5	4	3	2	1	0	Kommentar
20 40 02 04	Inhalt Latch A, Zähler #1	x	x	x	x	x	x	x	x	x : gelatchter Zählerstand
20 40 02 14	Inhalt Latch A, Zähler #2	x	x	x	x	x	x	x	x	x : gelatchter Zählerstand
20 40 03 00	Zähler freigeben / sperren (= `Cnt_Enable()`)	-	-	-	-	-	-	0	0	Zähler gesperrt
		-	-	-	-	-	-	1	1	Zähler freigeben
20 40 03 10	Zähler löschen (= `Cnt_Clear()`)	-	-	-	-	-	-	0	0	kein Einfluss
		-	-	-	-	-	-	1	1	Zähler löschen*
20 40 03 20	Zähler latchen (= `Cnt_Latch()`)	-	-	-	-	-	-	0	0	kein Einfluss
		-	-	-	-	-	-	1	1	Zählerstand in Latch A übertragen*

*Nach der Befehlsausführung werden gesetzte Bits automatisch wieder gelöscht. Bei anderen Befehlen müssen gesetzte Bits durch einen neuen Befehlsaufruf gelöscht werden.

Adresse [HEX]	Funktion	Bit Nr. 31:16	15:6	5	4	3	2	1	0	Kommentar

Abb. 18 – Zähler Hardware-Adressen der Steuer- und Datenregister

6 Kalibrierung

Die beiden Digital/Analog- (DAC) und der Analog/Digital-Wandler (ADC) Ihres *ADwin*-Systems sind bei der Auslieferung werkseitig kalibriert. Entsprechend den Vorschriften zur Einhaltung der Messgenauigkeit für Ihr Anwendungsgebiet sind die Geräte in regelmäßigen Abständen zu kalibrieren.

Sie führen die Kalibrierung mit dem Programm <L16Calib.exe> durch; der Pfad bei Standardinstallation ist <C:\ADwin\Tools\ADwin-L16>.

Hilfsmittel

Zur Kalibrierung benötigen Sie folgende Hilfsmittel:

– Ein Digital-Multimeter (DMM) mit einer Messgenauigkeit von 30µV.

– Eine Referenz-Gleichspannungsquelle mit stabiler Einstellgenauigkeit von 30µV. Ersatzweise verbinden Sie DAC 1 mit ADC 01(+), DAC 2 mit ADC 03 (+) und AGND DAC mit ADC 01(-) und ADC 03(-), z.B. in Form eines Teststeckers. Diese Verbindungen benötigen Sie auch für das Kalibrier-Diagramm.

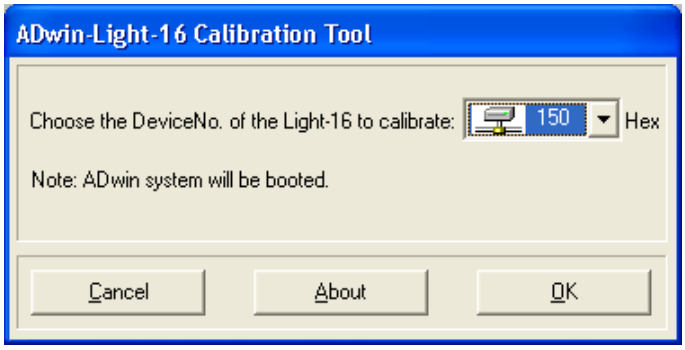
– Verbindungskabel von den Ein/Ausgängen zur Referenzspannungsquelle und zum Messgerät.

Schritt 1

Verbinden Sie Ihr *ADwin-light-16*-Gerät mit dem PC und konfigurieren Sie es mit dem Programm <ADconfig.exe>.

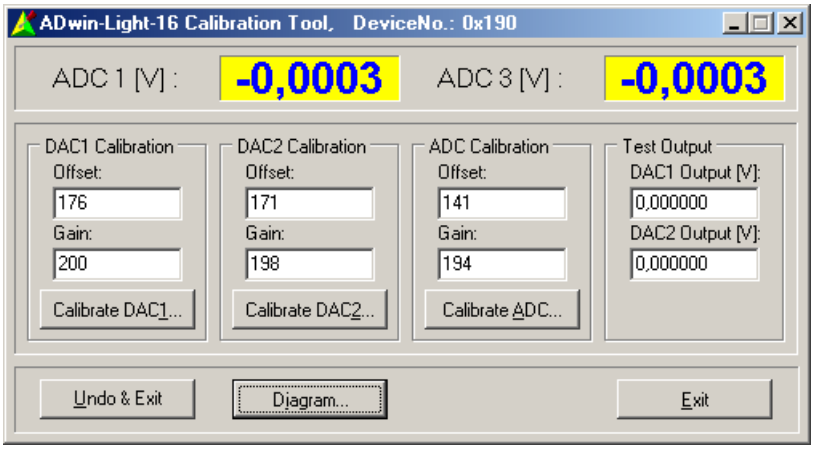
Schritt 2

Starten Sie das **Kalibrierprogramm** <L16Calib.exe>. Es erscheint das Fenster „ADwin-light-16 Calibration Tool“.



Wählen Sie die Device-Nummer des zu kalibrierenden Geräts und bestätigen Sie durch Drücken von „OK“.
Sie erhalten eine Warnung, wenn Sie kein *ADwin-light-16* Gerät gewählt haben oder eines mit einer älteren Firmware-Version. Sie können die Warnung mit „YES“ übergehen oder mit „NO“ zum vorigen Fenster zurückkehren.

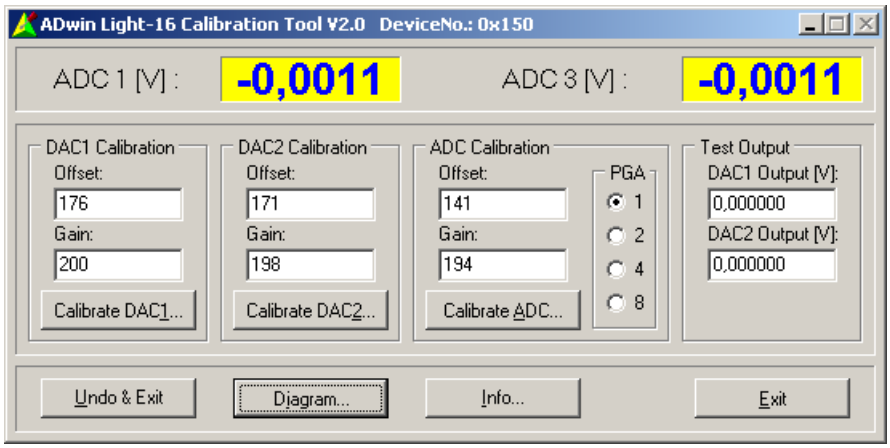
Es erscheint das **Übersichtsfenster**. In der Kopfzeile wird die von Ihnen gewählte Device-Nummer angezeigt.



Das obere Feld zeigt die aktuellen Messwerte an den Eingängen ADC 01 und ADC 03. Darunter sehen Sie die gültigen Kalibriereinstellungen für Offset und Gain der beiden DAC und des ADC; Sie können dort direkt Werte eingeben. Mit der Taste „Calibrate …“ starten Sie die Kalibrierung des jeweiligen Wandlers.

Am rechten Rand können Sie in die Felder DAC1 und DAC2 Spannungswerte eingeben, die automatisch auf die entsprechenden Ausgänge gelegt werden.

Bei Rev. B enthält das Übersichtsfenster zusätzlich das Auswahlfeld PGA für den Verstärkungsfaktor und die Schaltfläche `Info...`, über die Versionsinformationen des Geräts angezeigt werden können.



Alle von Ihnen eingegebenen Einstellungen werden automatisch gespeichert. In der unteren Zeile können Sie mit „`Undo&Exit`“ alle Eingaben rückgängig machen und das Kalibrierprogramm verlassen. „`Diagram`“ zeigt Ihnen in einer Grafik die Genauigkeit der aktuellen Kalibriereinstellung. Wenn Sie das Programm mit „`Exit`“ verlassen, bleiben die neuen Einstellungen erhalten.

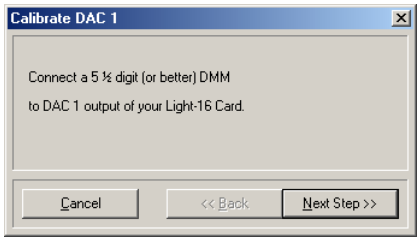
Kalibrieren Sie die Wandler in beliebiger Reihenfolge (nur mit Referenz-Spannungsquelle). Die Kalibrierung eines Wandlers wird jeweils in 3 Stufen ausgeführt; Sie können zwischen den Fenstern der Stufen durch Vorwärts-/ Rückwärts-Schaltflächen hin- und herschalten.

Ohne Referenz-Spannungsquelle ist die Kalibrierung möglich, aber ungenauer. Kalibrieren Sie dabei zuerst DAC1 und DAC2, dann erst den ADC.

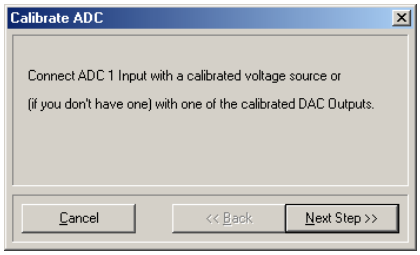
Schritt 3

Die 3 Stufen zum **Kalibrieren eines Wandlers** sind nachfolgend beschrieben, jeweils in der linken Spalte für einen DAC, in der rechten für den ADC.

1. Externes Hilfsgerät (DMM / Spannungsquelle) anschließen:
   Wählen Sie zur Kalibrierung eines Wandlers die entsprechende Taste „`Calibrate ...`“, es erscheint das erste Fenster:



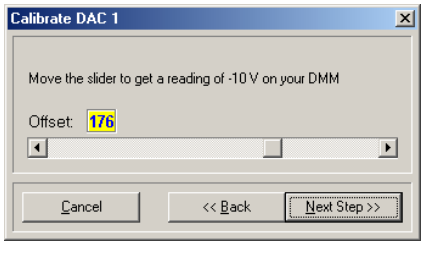
Schließen Sie ein DMM an den Pins AGND DAC und DAC1 oder DAC2 an.

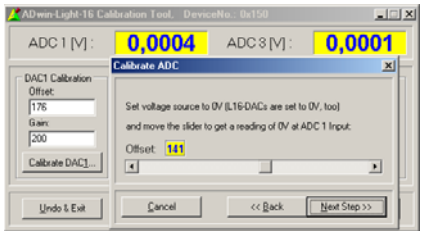


Schließen Sie die Spannungsquelle (oder einen DAC-Ausgang) an den Eingang ADC 01 oder ADC 03 an.

Beachten Sie bitte Abb. 9 – Pin-Belegung Ein-/Ausgänge (Buchse).
Wählen Sie „`Next Step` >>“.

2. Offset einstellen:





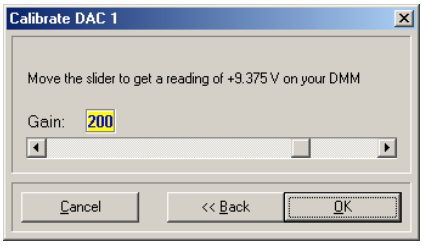
Verstellen Sie am Rollbalken den Offset-Wert so, dass Ihr Digital-Multimeter -10 V anzeigt.

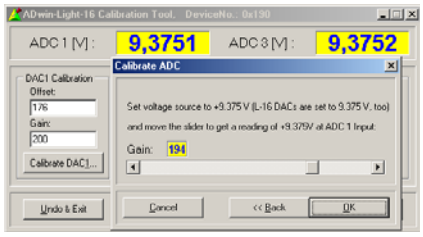
Stellen Sie an der Spannungsquelle den Sollwert 0 V ein. Die Einstellung des ADC auf diesen Wert erfolgt automatisch.
Stellen Sie am Rollbalken den Offset-Wert so ein, dass der Sollwert am ADC 01 im Übersichtsfenster angezeigt wird.

Bei Rev. B erfolgen die Einstellungen dieses Schritts automatisch.

Wählen Sie „`Next Step` >>“.

3. Gain einstellen:





Verstellen Sie am Rollbalken den Offset-Wert so, dass Ihr Digital-Multimeter -10 V anzeigt.

Stellen Sie an der Spannungsquelle den Sollwert 9,375 V ein. Die Einstellung des ADC auf diesen Wert erfolgt automatisch.
Stellen Sie am Rollbalken den Offset-Wert so ein, dass der Sollwert am ADC 01 im Übersichtsfenster angezeigt wird.

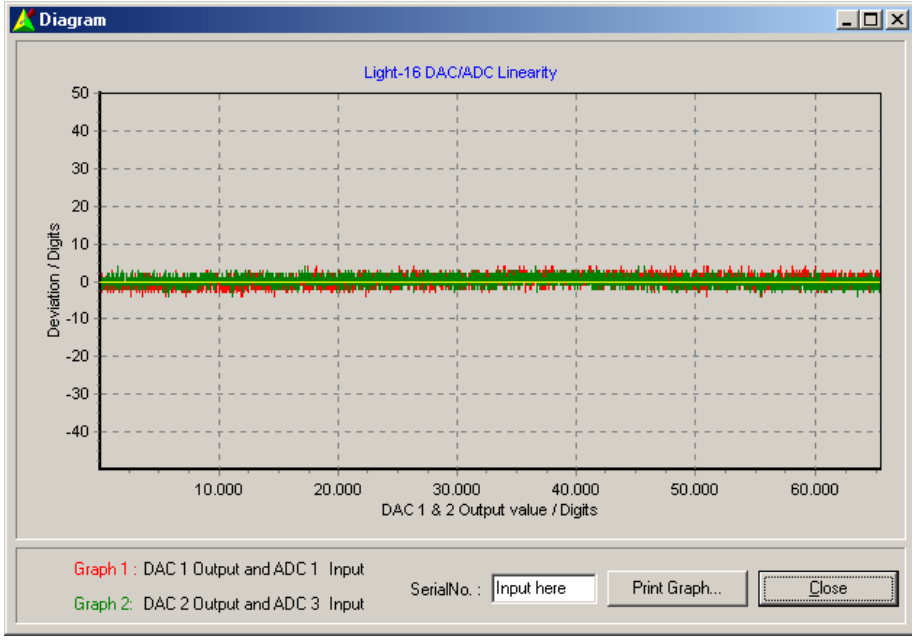
Bei Rev. B erfolgen die Einstellungen dieses Schritts automatisch.

Die Kalibrierung für diesen Wandler ist beendet. Wählen Sie „`OK`“.

Wiederholen Sie Schritt 3 ggf. für die anderen Wandler.

Schritt 4

Die **Genauigkeit** der eingestellten Kalibrierung können Sie anhand eines Diagramms prüfen (Schaltfläche `Diagram` im Übersichtsfenster).
Verbinden Sie zunächst den Ausgang DAC 1 mit dem Eingang ADC 01 sowie den Ausgang DAC 2 mit dem Eingang ADC 03.



Das Programm gibt die Werte 0…65.535 Digits auf beide DAC aus, vergleicht sie mit den gemessenen Eingangswerten und stellt die Abweichung in Kurven dar: Kurve 1 (rot) für DAC 1 / ADC 01 und Kurve 2 (grün) für DAC 2 / ADC 03.
Die Abweichung sollte kleiner als 5 Digits sein.

Sie können die Grafik mit `Print Graph` ausdrucken (Farbdrucker empfohlen). Geben Sie hierfür die Seriennummer Ihres *ADwin*-Systems ein, damit Sie den Ausdruck später zuordnen können. Auf dem Ausdruck sind außerdem die Kalibriereinstellungen und das Druckdatum enthalten.

Mit `Close` kehren Sie zum Übersichtsfenster zurück.

Schritt 5

Die Kalibrierung ist beendet.

7 CO1-Zählererweiterung

Die Zählererweiterung CO1 (Bestelloption *L16-CO1*) stellt einen 32 Bit Vor-/ Rückwärtszähler mit Vierflankenauswertung zur Verfügung und ersetzt die Inkrementalzähler der Grundversion (Abb. 19 – Schema der L16-CO1 Zählererweiterung zeigt den Zähleraufbau).

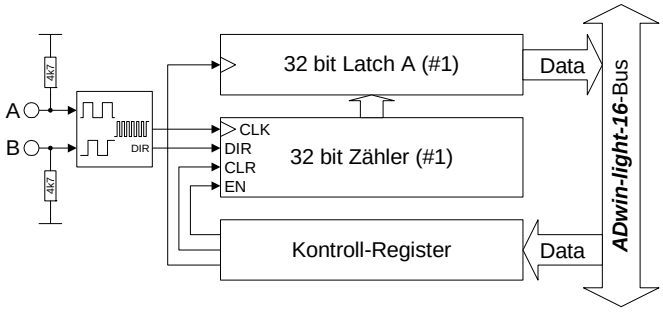


Abb. 19 – Schema der *L16-CO1* Zählererweiterung

In der Bestelloption *L16-CO1* stehen insgesamt folgende Funktionen zur Verfügung:

– 8 analoge Eingänge (Seite 11)
– 2 analoge Ausgänge (Seite 11)
– 6 digitale Eingänge, 6 digitale Ausgänge (Seite 14)
– 1 Vor-/ Rückwärtszähler 32 Bit mit Vierflankenauswertung (Seite 27)
– 1 serielle Schnittstelle LS-Bus, ab Rev. B2 (Seite 17)

7.1 Hardware

Der Zähler wird **extern getaktet** und besitzt eine Vierflanken-Auswertung zum Anschluss eines Encoders. Ausgelesen wird der Zähler über das Latch A, die Steuerung erfolgt mit *ADbasic*-Befehlen über ein Kontrollregister (siehe “CO1-Befehle, Kurzübersicht” auf Seite 27).

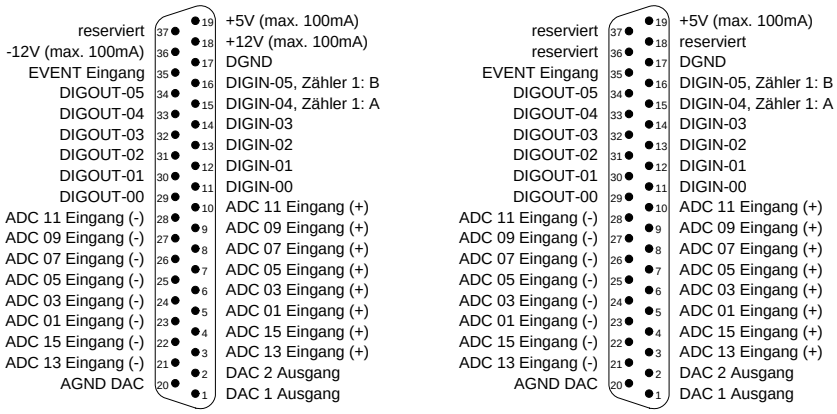
Die Vierflankenauswertung setzt die um 90 Grad versetzten digitalen Eingangssignale A und B in ein Takt- (CLK) und Richtungssignal (DIR) für den Zähler um. Dabei wird aus jeder Flanke des A- und B-Signals ein Taktsignal erzeugt. Die Zählrichtung (DIR) ergibt sich aus der Reihenfolge der steigenden bzw. fallenden Flanken dieser Signale.

Eingänge beschalten

Die Takteingänge A und B liegen auf den Pins 15 und 16 der Sub-D-Buchse `ADwin I/O CONNECTOR` (siehe Pin-Belegung unten); für die korrekte Funktion sind TTL-kompatible Signale erforderlich. Einzelheiten und Grenzwerte finden Sie bei den Technischen Daten im Anhang.

Beide Eingänge können alternativ als digitaler Signaleingang eingesetzt werden (siehe auch Kapitel 5.2).

Obwohl alle Eingänge der CO1-Erweiterung einen Pull-down-Widerstand besitzen, können offene Eingänge vor allem in einer nicht störungsfreien Umgebung zu Fehlern führen. Legen Sie deshalb sicherheitshalber die nicht benutzten Eingänge auf einen definierten Pegel (z.B. GND).



L16-PCI *L16-EURO* und *L16-EXT*

Abb. 20 – Pin-Belegung der *L16-CO1*

7.2 Programmierung

Die Zähler-Funktionen der CO1-Erweiterung werden mit *ADbasic*-Befehlen komfortabel programmiert. Die Befehle sind in einer Include-Datei enthalten; binden Sie diese Datei am Beginn des Programms ein mit der Befehlszeile

```
#Include ADWL16.INC
```

Die Zähler-Befehle der folgenden Tabelle sind ab Seite 95 oder in der Online-Hilfe vollständig beschrieben:

Zähler-Nr. Bit	1 0	Kommentar
Cnt_Clear()	0	ohne Einfluss
	1	Zähler löschen *
Cnt_Enable()	0	Zähler sperren
	1	Zähler freigeben
Cnt_Latch()	0	ohne Einfluss
	1	Zählerstand in Latch A übernehmen *
Cnt_ReadLatch(#)		Latch A auslesen (# = Zähler-Nr. 1)
Cnt_Read(#)		Zählerstand in Latch A übernehmen und auslesen (# = Zähler-Nr. 1)
*Nach der Befehlsausführung werden gesetzte Bits automatisch wieder gelöscht. Bei anderen Befehlen müssen gesetzte Bits durch einen neuen Befehlsaufruf gelöscht werden.		

Abb. 21 – CO1-Befehle, Kurzübersicht

Werten Sie bitte den Zählerinhalt nur mit Variablen vom Typ **Long** aus, vor allem für die Ermittlung von Differenzen oder der Zählrichtung (siehe auch Seite 16).

Die Zählrichtung (vor- oder rückwärts) ergibt sich zuverlässig nur aus dem

Vorzeichen der Differenz: [**neuer** Zählerstand] **minus** [**alter** Zählerstand]

und nicht aus dem *Vergleich* der Zählerstände.

Für extrem zeitkritische Aufgaben kann es sinnvoll sein, wenn Sie mit den Befehlen **Peek** und **Poke** direkt auf die Steuer- und Datenregister des Zählers zugreifen. In der Tabelle sind die entsprechenden Hardware-Adressen dargestellt.

Die Hardware-Adressen des CO1-Zählers ersetzen diejenigen der Grundversions-Zähler und sind deswegen identisch.

Adresse [HEX]	Funktion	Bit-Nummer 31:24	23:16	15:08	07:01	00	Kommentar
20 40 02 04	Inhalt Latch A, Zähler #1	x	x	x	x	x	x : gelatchter Zählerstand
20 40 03 00	Zähler freigeben **Cnt_Enable()**	-	-	-	-	0	Zähler sperren
		-	-	-	-	1	Zähler freigeben
20 40 03 10	Zähler löschen **Cnt_Clear()**	-	-	-	-	0	kein Einfluss
		-	-	-	-	1	Zähler löschen*
20 40 03 20	Zähler latchen **Cnt_Latch()**	-	-	-	-	0	kein Einfluss
		-	-	-	-	1	Zähler latchen*
*Nach der Befehlsausführung werden gesetzte Bits automatisch wieder gelöscht. Bei anderen Befehlen müssen gesetzte Bits durch einen neuen Befehlsaufruf gelöscht werden.							

Abb. 22 – CO1-Hardware-Adressen der Steuer- und Datenregister

Include-Datei

8 DIO1-Erweiterung

Die DIO1-Erweiterung stellt Ihnen zusätzlich zur Verfügung:

– 32 digitale Ein-/Ausgänge (programmierbar in Gruppen zu 8), Seite 32.

– Zwei 32 Bit Vor-/Rückwärtszähler (Seite 34) zur Impuls-, Periodendauer- und Tastverhältnismessung sowie einer Vierflankenauswertung zum Anschluss von Encodern.
Eingänge per DIP-Schalter umschaltbar zwischen „single ended“ und „differentiell“.

Die Zähler der Basisversion werden von den DIO1-Zählern ersetzt.

– CAN-Schnittstelle (High-Speed), Seite 42.

– 1 SSI-Decoder (Seite 45), erst ab Rev. B.

Der SSI-Decoder eignet sich zum Anschluss von Inkremental-Encodern mit SSI-Schnittstelle. Die Eingänge liegen auf dem Stecker COUNTER; die Signale sind differentiell und für RS422/485-Pegel (5V) ausgelegt.

Informationen zu den Funktionen der Basisversion finden Sie auf folgenden Seiten:

– 8 analoge Eingänge: Seite 11

– 2 analoge Ausgänge: Seite 11

– 6 digitale Eingänge, 6 digitale Ausgänge: Seite 14

– 1 serielle Schnittstelle LS-Bus, ab Rev. B2: Seite 17

Das Schema zeigt Ihnen gemeinsam die Grundfunktionen eines L16-Geräts mit den Zusatzfunktionen der DIO1-Erweiterung (als USB-Variante).

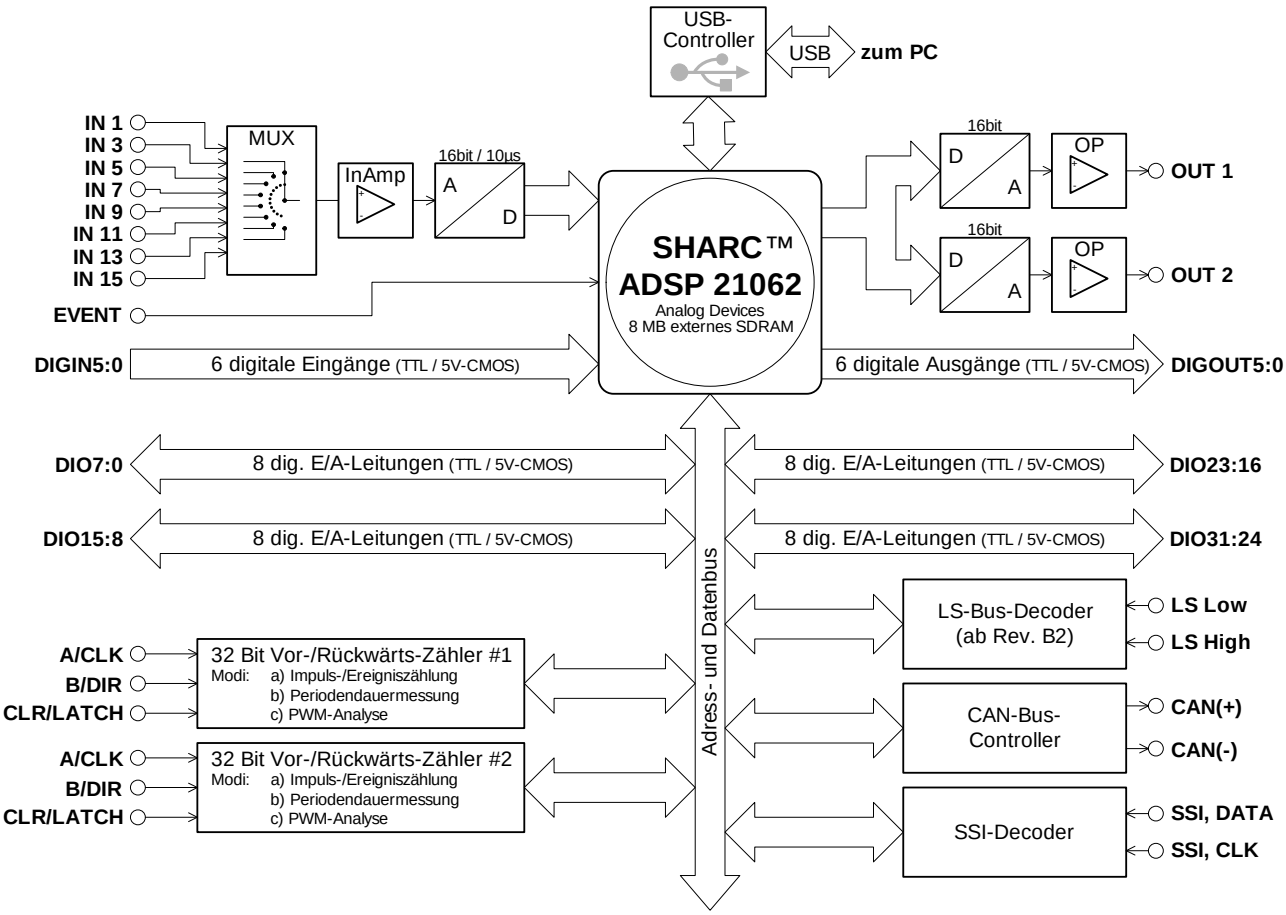


Abb. 23 – Schema *L16-DIO1* (mit USB-Schnittstelle)

Die Pin-Belegung am Anschluss „ADwin I/O-CONNECTOR“ ist mit der Basisversion identisch, bis auf einen Unterschied: Die in der Basisversion doppelt beschalteten Pins 15/16 stehen nun ausschließlich als Digin-04 und Digin-05 zur Verfügung.

Die Pin-Belegung der LS-Bus-Schnittstelle ist auf Seite 10 dargestellt.

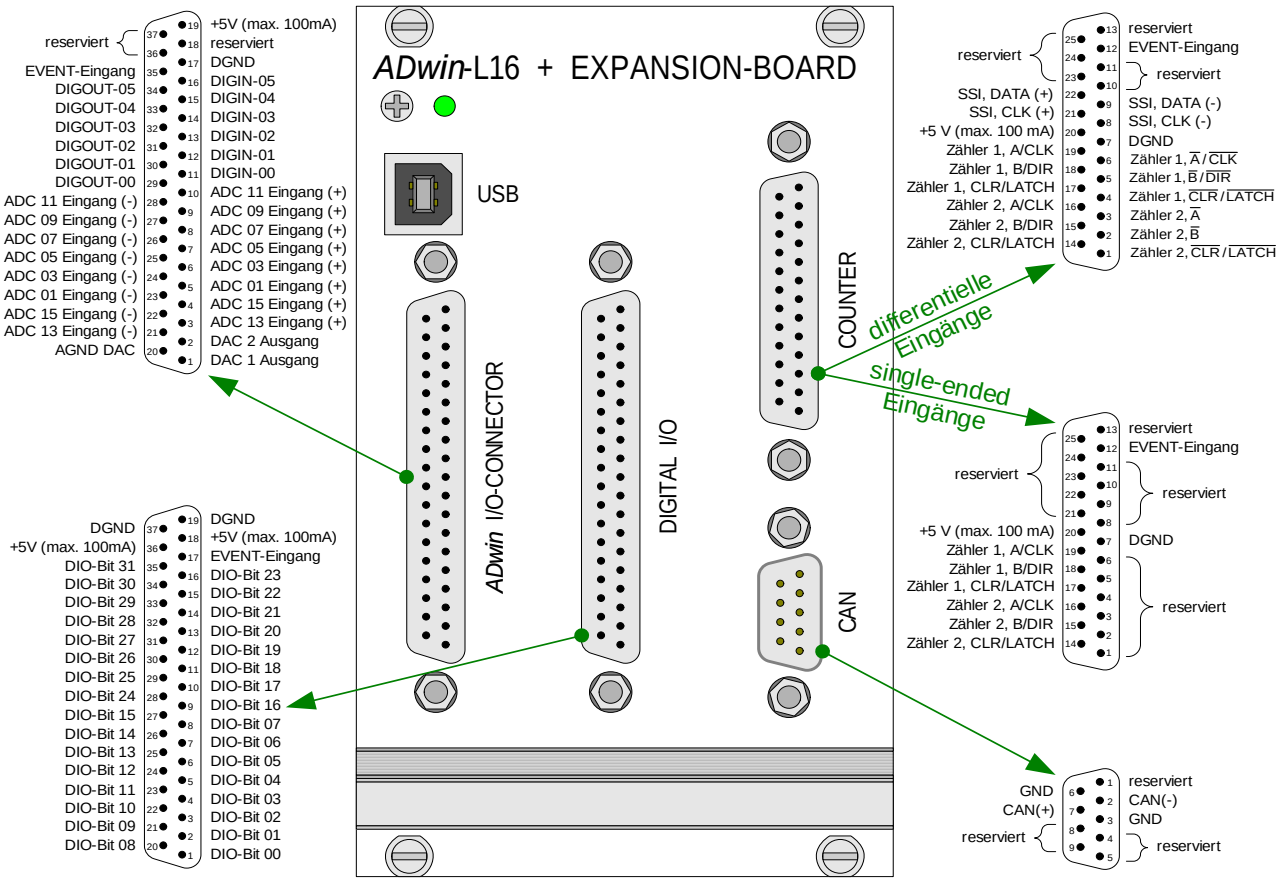


Abb. 24 – Übersichtsbild *L16-EURO-DIO1* mit Pinbelegungen
Für die anderen L16-Varianten sind die Steckerbezeichnungen identisch.

Auf der DIO1-Platine sind mehrere DIP-Schalter vorhanden, mit denen Sie Einstellungen für die Zähler und die CAN-Schnittstelle verändern können.

Die Einstellung ist nur bei ausgebautem bzw. geöffnetem Gerät möglich. Beachten Sie den Sicherheitshinweis am Anfang dieser Dokumentation.

Einstellungen an der Hardware (DIP-Schalter)

Bauform	Vorgehen
L16-EURO, L16-PCI	**Ausbau der Platine** Der Ausbau der Platine erfolgt umgekehrt zum Einbau, der im Handbuch „ADwin-Treiber-Installation beschrieben ist.
L16-EXT	**Öffnen des Gehäuses** Entfernen Sie an den schwarzen Seitenplatten die oberen Inbus-Schrauben (Innensechskant) mit einem 2 mm Inbus-Schlüssel und lösen die unteren Schrauben. Stellen Sie die Seitenplatten etwas schräg und ziehen das Oberteil ab. Merken Sie sich die Orientierung des Gehäuses für das spätere Schließen des Geräts. Sie blicken nun direkt auf DIO1-Platine.

Der **Zusammenbau** erfolgt in umgekehrter Reihenfolge zum Ausbau / Öffnen.

Lage der DIP-Schalter

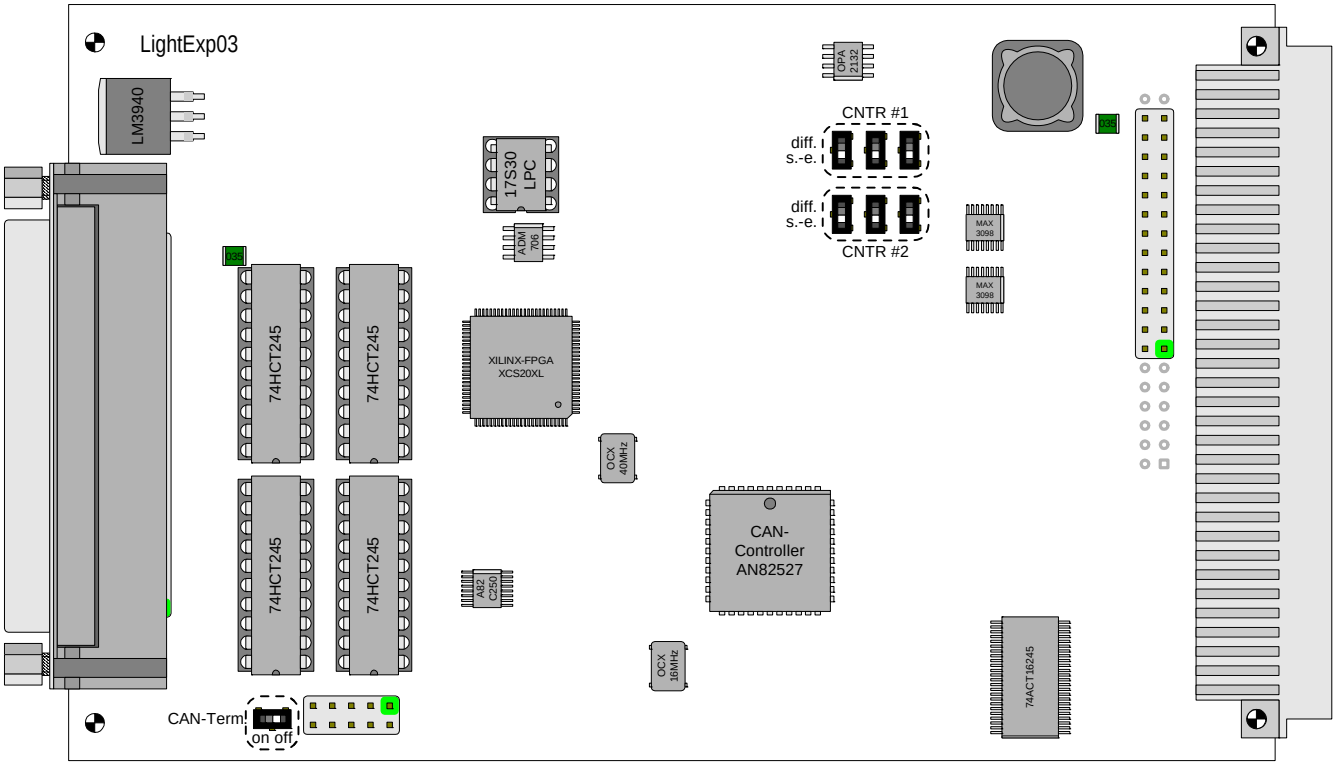
Entnehmen Sie bitte die Lage der DIP-Schalter der nachfolgenden Abbildung (achten Sie auf die Revision). Die gestrichelten Rahmen zeigen die Zuordnung der DIP-Schalter zu den Zählern 1 und 2 (obere Hälfte, halbrechts) und zur CAN-Schnittstelle (unten links).

Zähler 1 und 2: Für differentiellen Betrieb legen Sie alle 3 DIP-Schalter des entsprechenden Zählers nach oben um; für single ended-Betrieb legen Sie die Schalter nach unten um.

CAN-Schnittstelle: Wenn die Terminierung erforderlich ist, legen Sie den DIP-Schalter nach oben um (L16 Rev. B) bzw. in Richtung zum CAN-Stecker (L16 Rev. A).

Weitere Informationen zur Einstellung der DIP-Schalter siehe Kapitel 8.2 “Zähler“ und Kapitel 8.3 “CAN-Bus“.

L16-DIO1 Rev. A



L16-DIO1 Rev. B

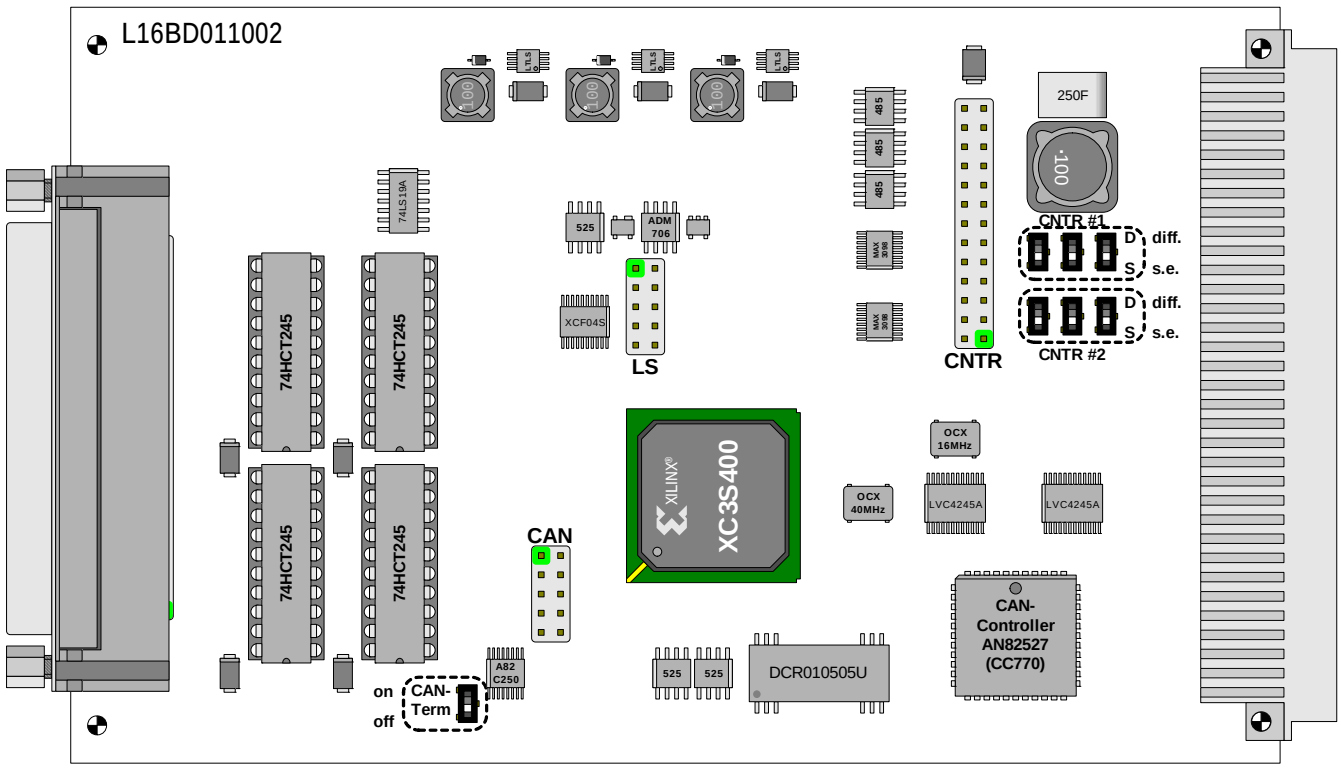


Abb. 25 – Lage der DIP-Schalter auf der DIO1-Platine

Die technischen Daten der DIO1-Erweiterung sind im Anhang enthalten.

8.1 Digitale Ein- und Ausgänge

Zusätzlich zu den digitalen Ein-/Ausgängen der Basisversion (Digin, DIGOUT, EVENT) stehen Ihnen auf der 37-poligen Sub-D-Buchse `Digital I/O` 32 digitale Ein- oder Ausgänge (Abkürzung: DIO) zur Verfügung. Sie sind in Gruppen zu jeweils 8 als Ein- oder Ausgang programmierbar.

Trigger-Eingang

Es gibt je einen externen Trigger-Eingang (EVENT) an folgenden Sub-D-Buchsen: `ADwin I/O Connector`, `Counter` und `Digital I/O`. Die Eingänge `Counter` und `Digital I/O` sind galvanisch verbunden und haben einen gemeinsamen Pull-up-Widerstand von 4,7kΩ, so dass Sie nach Bedarf einen der Eingänge auswählen können. Verwenden Sie nur einen der drei Event-Eingänge.

Mit einem externen Signal (Trigger) am Event-Eingang kann ein Prozesse ausgelöst werden, der sofort und vollständig abgearbeitet wird (siehe *ADbasic*-Handbuch oder -Online-Hilfe, Kapitel: Prozesse im Betriebssystem“).

Die digitalen Eingänge sind TTL-kompatibel und gegen Überspannung nicht geschützt.

Einschaltkonfiguration

Nach dem Einschalten des Gerätes sind alle Anschlüsse als Eingang konfiguriert; dies entspricht dem Befehl **`Conf_DIO_E(0)`**. Mit dem Befehl
`Conf_DIO_E(n)`
programmieren Sie die 32 DIO-Leitungen in 4 Gruppen zu jeweils 8 Leitungen als Ein- oder Ausgang (siehe Kapitel 13 ab Seite 63).

Die folgende Tabelle zeigt Ihnen die 16 möglichen Konfigurationen dieses Befehls. In der untersten Zeile stehen die Befehle, die für bestimmte DIO-Gruppen anwendbar sind.

`Conf_DIO_E(n)`	DIO 31 : DIO 24	DIO 23 : DIO 16	DIO 15 : DIO 08	DIO 07 : DIO 00
0	**IN**	**IN**	**IN**	**IN**
1	IN	IN	IN	OUT
2	IN	IN	OUT	IN
3	IN	IN	OUT	OUT
4	IN	OUT	IN	IN
5	IN	OUT	IN	OUT
6	IN	OUT	OUT	IN
7	IN	OUT	OUT	OUT
8	OUT	IN	IN	IN
9	OUT	IN	IN	OUT
10	OUT	IN	OUT	IN
11	OUT	IN	OUT	OUT
12	OUT	OUT	IN	IN
13	OUT	OUT	IN	OUT
14	OUT	OUT	OUT	IN
15	OUT	OUT	OUT	OUT
Anwendbare *ADbasic*-Befehle:	**`Digin_Word2_E`** **`Digout_Word2_E`** **`Digout_Set2_E`** **`Digout_Reset2_E`**		**`Digin_Word1_E`** **`Digout_Word1_E`** **`Digout_Set1_E`** **`Digout_Reset1_E`**	

Abb. 26 – Konfigurationen mit **`Conf_DIO_E`**

Programmierung

Die digitalen Kanäle werden mit *ADbasic*-Befehlen komfortabel programmiert:

Bereich	Befehle
Ein-/Ausgänge konfigurieren	**`Conf_DIO_E`**
Digitaleingänge lesen	**`Digin_Word1_E`**, **`Digin_Word2_E`**, **`Digin_Long_E`**

Bereich	Befehle
Ausgewählte Digitalausgänge setzen	**Digout_Reset1_E**, **Digout_Reset2_E**, **Digout_Set1_E**, **Digout_Set2_E**
Alle Digitalausgänge setzen	**Digout_Long_E**, **Digout_Word1_E**, **Digout_Word2_E**

Die Befehle sind in der Include-Datei <ADWL16.INC> enthalten und werden ab Seite 83 und in der Online-Hilfe erläutert. Näheres zur Programmierung zeitkritischer Aufgaben finden Sie im Kapitel 5.5 auf Seite 18.

8.2 Zähler

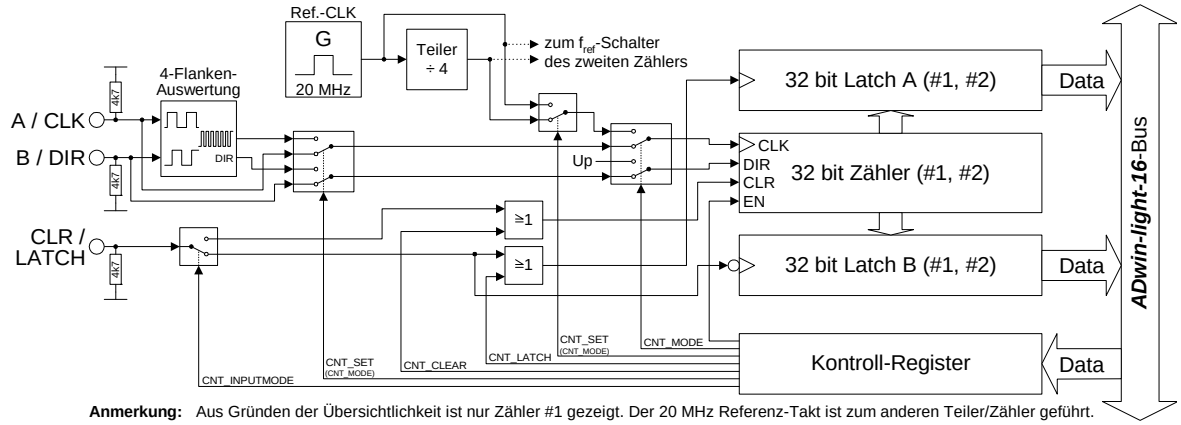
Zähler

Die Erweiterung DIO1 stellt **zwei 32 Bit Zähler** zur Verfügung, die Sie per Software sowohl einzeln als auch gemeinsam konfigurieren und auslesen können. Sie können an den Eingängen differentielle oder „single ended“-Signale anlegen; dafür stellen Sie die Betriebsart per DIP-Schalter um (siehe unten).
Die Zähler ersetzen die Inkrementalzähler der Grundversion.

Latch

Die Zähler können **intern oder extern getaktet** werden und werden über zugeordnete Latches ausgelesen. Alle Zähler haben je ein Latch A sowie ein Latch B (die Grafik zeigt den Aufbau eines einzelnen Zählers).

Der Zählerstand kann mit Programmierbefehlen oder (bei entsprechender Einstellung) bei einem externen Signal an CLR/LATCH gelöscht oder in ein Latch übertragen werden.



Anmerkung: Aus Gründen der Übersichtlichkeit ist nur Zähler #1 gezeigt. Der 20 MHz Referenz-Takt ist zum anderen Teiler/Zähler geführt.

Abb. 27 – Schema DIO1-Zähler

Es gibt die Betriebsarten Ereigniszählung (externer Takt) und Pulsbreitenmessung (interner Takt); siehe auch Kapitel 8.2.2 / 8.2.3:

Externer Takteingang

1. **Ereigniszählung**: Das In-/Dekrementieren des Zählers wird durch externe Rechtecksignale an den Eingängen A/CLK und B/DIR ausgelöst. Ein Signal an CLR/LATCH bewirkt, dass entweder der Zähler auf Null gesetzt (CLR) oder der Zählerstand ins Latch geschrieben wird (LATCH).

   Es gibt die Modi:
   - **Takt und Richtung**: Jede positive Flanke an CLK in- oder dekrementiert den Zählerstand um eins. Das Signal an B/DIR bestimmt die Zählrichtung (0 = Dekrement; 1 = Inkrement).
   - **Vierflankenauswertung**: Jede Flanke der (um 90 Grad) versetzten Signale an A/CLK und an B/DIR löst ein In-/ Dekrementieren des Zählers aus. Die Zählrichtung ergibt sich aus der Reihenfolge der steigenden/fallenden Flanken dieser Signale. Dieser Modus wird besonders für Inkrementalgeber (Winkel-Encoder) eingesetzt.

Interner Takteingang

2. **Pulsbreitenmessung**: Die Signale zum In-/Dekrementieren erhält der Zähler von einem internen Referenztaktgeber mit einer Signalfrequenz von 20 MHz (alternativ 5 MHz nach einem Teiler). Ausgewertet wird das an CLR/LATCH anliegende Rechtecksignal: Mit jeder positiven Flanke wird der Zählerstand in Latch A geschrieben, mit jeder negativen in Latch B.

   Sie können berechnen:
   - die Periodendauer des Eingangssignals an CLR/LATCH aus den Werten in Latch A.
   - die Pulsbreite und Pausenzeit aus den Werten in Latch A und Latch B.

Betriebsart der Zählereingänge umstellen

Die Eingänge der Zähler können single ended oder differentiell betrieben werden. Die Einstellung bei Auslieferung ist nicht festgelegt. Stellen Sie daher die von Ihnen gewünschte Betriebsart an den DIP-Schaltern der DIO1-Platine ein (siehe Abb. 25 – Lage der DIP-Schalter auf der DIO1-Platine).

Für differentiellen Betrieb legen Sie alle 3 DIP-Schalter des entsprechenden Zählers nach oben um; für single ended-Betrieb legen Sie die Schalter nach unten um.

8.2.1 Programmierung

Die Zähler-Funktionen der DIO1-Erweiterung werden mit *ADbasic*-Befehlen komfortabel programmiert. Die Befehle sind in einer Include-Datei enthalten; binden Sie diese Datei am Beginn des Programms ein mit der Befehlszeile

```
#Include ADWL16.INC
```

Die Zähler-Befehle der folgenden Tabelle sind ab Seite 95 oder in der Online-Hilfe vollständig beschrieben:

Zähler-Nr.	2	1	Kommentar
Bit	1	0	
Cnt_Clear( )	0	0	ohne Einfluss
	1	1	Zähler löschen *
Cnt_Enable( )	0	0	Zähler sperren
	1	1	Zähler freigeben (auf laufende Zähler achten)
Cnt_InputMode( )	0	0	CLR/LATCH-Eingang auf CLR-Modus stellen
	1	1	CLR/LATCH-Eingang auf LATCH-Modus stellen
Cnt_Latch( )	0	0	ohne Einfluss
	1	1	Zählerstand in Latch-Register A übernehmen*
Cnt_Mode( )	0	0	externer Takteingang
	1	1	interner Referenztakt (20MHz / 5MHz)
Cnt_Set( )	0	0	**Cnt_Mode**-Bit = 0 : 4-Flankenauswertung
			Cnt_Mode-Bit = 1 : interner Referenztakt von 20MHz
	1	1	**Cnt_Mode**-Bit = 0 : Takt- und Richtungseingang (CLK & DIR)
			Cnt_Mode-Bit = 1 : interner Referenztakt von 5MHz
Cnt_ClearEnable()	0	0	CLR-Eingang sperren
	1	1	CLR-Eingang freigeben
Cnt_GetStatus(#)**			Status-Register auslesen (Bedeutung der Bits siehe Seite 99 oder Online-Hilfe)
Cnt_Read(#)**			Zählerstand in Latch A übernehmen und auslesen
Cnt_ReadLatch(#)**			Latch A (getriggert durch positive Flanke) auslesen
Cnt_ReadFLatch(#)**			Latch B (getriggert durch negative Flanke) auslesen

* Nach der Befehlsausführung werden gesetzte Bits automatisch wieder gelöscht. Bei anderen Befehlen müssen gesetzte Bits durch einen neuen Befehlsaufruf gelöscht werden.
** # = Zähler-Nummer 1 oder 2

Abb. 28 – DIO1-Zähler Befehle Kurzreferenz

Sie können mit den Befehlen der Tabelle jeden Zähler einzeln oder beide Zähler gemeinsam konfigurieren.

Initialisierung

Initialisieren Sie die Zähler bitte in dieser Reihenfolge:

1. Gewünschten Zähler sperren (**Cnt_Enable**)

   Der Befehl **Cnt_Enable** spricht alle Zähler gemeinsam an. Auch wenn Sie nur einen Zähler sperren (Bit = 0) möchten, müssen Sie alle anderen Zähler konfigurieren, deren Zustand unverändert bleiben soll (Bit = 1).
   Beim Freigeben des Zählers gilt dies entsprechend.

2. Betriebsart / Modus einstellen
   (**Cnt_Mode**, **Cnt_Set**, **Cnt_InputMode**)

   Beachten Sie die Abhängigkeit des Befehls **Cnt_Set** vom Befehl **Cnt_Mode**.

3. Zähler löschen (**Cnt_Clear**)

4. Zähler freigeben (**Cnt_Enable**)

Werten Sie bitte den Zählerinhalt nur mit Variablen vom Typ **LONG** aus. *ADbasic* behält dann intern das gelesene Bitmuster unverändert bei und berücksichtigt automatisch den Übergang zwischen positivem und negativem Zahlenbereich (siehe auch Seite 16). Damit gilt:

Die Zählrichtung (vor- oder rückwärts) ergibt sich zuverlässig nur aus dem

Vorzeichen der Differenz: [neuer Zählerstand] minus [alter Zählerstand]

und nicht aus dem *Vergleich* (< >) der Zählerstände.

Ein Prozess kann sehr schnell abgearbeitet werden, wenn Sie mit den Befehlen **Peek** und **Poke** direkt auf die Steuer- und Datenregister zugreifen (siehe auch *ADbasic*-Handbuch / Online-Hilfe).

Die Hardware-Adressen der DIO1-Erweiterung können Sie folgender Tabelle entnehmen (vgl. Abb. 28 – DIO1-Zähler Befehle Kurzreferenz).

Adresse [hex]	Funktion	Bit 31:16	15:10	9	8	7	6	5	4	3	2	1	0	Kommentar
20 40 02 04	Inhalt Latch A, Zähler-#1	x	x	x	x	x	x	x	x	x	x	x	x	x : Inhalt des Latch
20 40 02 08	Inhalt Latch B, Zähler-#1	x	x	x	x	x	x	x	x	x	x	x	x	x : Inhalt des Latch
20 40 02 14	Inhalt Latch A, Zähler-#2	x	x	x	x	x	x	x	x	x	x	x	x	x : Inhalt des Latch
20 40 02 18	Inhalt Latch B, Zähler-#2	x	x	x	x	x	x	x	x	x	x	x	x	x : Inhalt des Latch
20 40 03 00	Zähler sperren / freigeben **Cnt_Enable**()	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : Zähler sperren x = 1 : Zähler freigeben
20 40 03 10	Zähler löschen **Cnt_Clear**()	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : kein Einfluss x = 1 : Zähler löschen
20 40 03 20	Zähler latchen **Cnt_Latch**()	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : kein Einfluss x = 1 : Zähler latchen
20 40 03 30	Eingang: CLR oder LATCH	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : CLR-Eingang x = 1 : LATCH-Eingang
20 40 03 40	Impuls-/Ereigniszähler- oder Puls-breiten-/Periodendauermessung	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : externer Takteingang x =1: interner Referenztakt (20MHz/5MHz)
20 40 03 50	4-Flankenauswertung / CLK+DIR oder 20MHz / 5MHz Referenztakt	-	-	-	-	-	-	-	-	x	x	x	x	**Cnt_Mode**=0: x=0: 4-Fl.; x=1:CLK+DIR **Cnt_Mode**=1: x=0: 20MHz; x=1: 5MHz
20 40 04 54	DIO_Erweiterung Bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x: setzt/löscht Ausgang <Bit-Nr.>
20 40 04 64	DIO_Erweiterung Bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x: setzt/löscht Ausgang <Bit-Nr.+16>
20 40 04 74	DIO_Erweiterung Set_Bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang <Bit-Nr.> setzen *
20 40 04 84	DIO_Erweiterung Set_Bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang <Bit-Nr.+16> setzen*
20 40 04 94	DIO_Erweiterung Reset_Bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang <Bit-Nr.> löschen *
20 40 04 A4	DIO_Erweiterung Reset_Bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang <Bit-Nr.+16> löschen *
* Funktion bei Eingängen ohne Einfluss														

Abb. 29 – DIO1 Hardware-Adressen der Steuer- und Datenregister

Adresse [hex]	Funktion	Bit 31:16	15:10	9	8	7	6	5	4	3	2	1	0	Kommentar
20 40 04 6C	Ein-/Ausgänge konfigurieren **Conf_DIO**() Bit 0: DIO 00...07; Bit 1: DIO 08...15 Bit 2: DIO 16...23; Bit 3: DIO 24...31	-	-	-	-	-	-	-	-	x	x	x	x	x = 0: Kanäle als Eingang definieren x = 1: Kanäle als Ausgang definieren
* Funktion bei Eingängen ohne Einfluss														

Abb. 29 – DIO1 Hardware-Adressen der Steuer- und Datenregister

8.2.2 Betriebsart Impuls-/Ereigniszähler

Externe Rechtecksignale an den Eingängen A/CLK und B/DIR takten in dieser Betriebsart den jeweiligen Zähler. Mit **Cnt_Set** aktivieren Sie entweder den Modus zur Ermittlung von Taktfrequenz und Richtung oder die Vierflankenauswertung.

Der Eingang CLR/LATCH kann benutzt werden, um (jeweils bei einem dort anliegenden High-Signal)

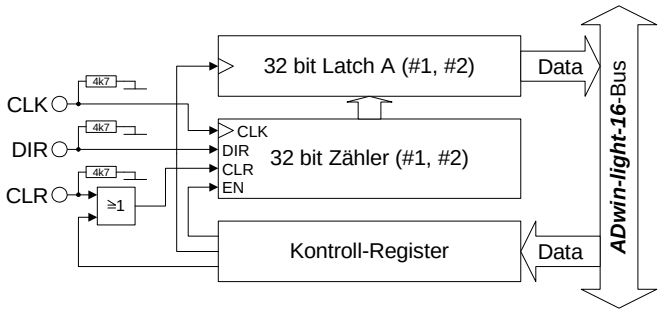
Löschen

– den Zähler zu löschen (CLR)

Latchen

– den Zählerstand in Latch A zu übernehmen (LATCH).

Takt und Richtung



Jede positive Flanke eines Rechtecksignals auf dem CLK-Eingang (Clock) wird bis zu einer maximalen Frequenz von 20 MHz gezählt. Die Richtung ergibt sich aus einem High- (vorwärts) bzw. Low-Signal (rückwärts) auf dem DIR-Eingang (Direction); dieses Signal kann als konstante Spannung oder als Rechtecksignal (z.B. vorgegeben durch eine externe Logikschaltung) angelegt sein.

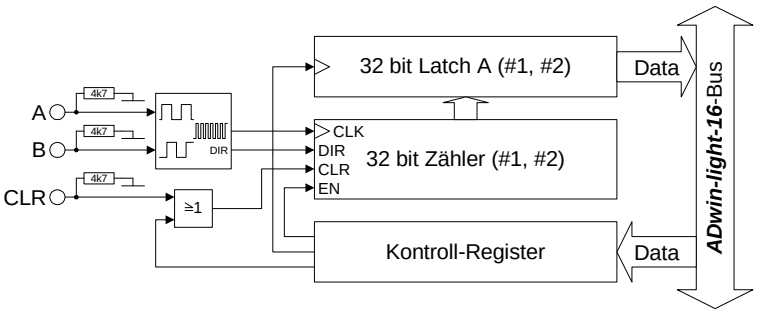
Programmierbeispiel

```
INIT:                          Initialisieren
. . .                          Alle Zähler sperren
CNT_ENABLE(0)                  Inhalt des Zählers 1 löschen
CNT_CLEAR(1)                   externer Takteingang am CLK-Eingang
CNT_MODE(0)                    Modus Takt und Richtung aktivieren
CNT_SET(1)                     Eingang CLR/LATCH auf CLR-Modus stellen
CNT_INPUTMODE(0)               Zähler 1 freigeben
CNT_ENABLE(1)                  Aktuellen Zählerstand in Latch A übernehmen
. . .                          Latch A auslesen
                               Auswerten im Programm

EVENT:
CNT_LATCH(1)
CNT_READLATCH(1)
. . .
```

Vier-Flanken-Auswertung

Dieser Modus ermittelt Takt und Zählrichtung aus zwei Rechteck-Signalen, die an den Eingängen A und B um 90 Grad (ideal) versetzt anliegen. Die Zählrichtung ergibt sich logisch aus der zeitlichen Reihenfolge der steigenden / fallenden Flanken zueinander.



Berücksichtigen Sie bitte:

– Der Zähler registriert bei einem Zyklus 4 Flanken.

– Die maximale Zählfrequenz beträgt 20 MHz. Gemeinsam mit den 4 Flanken je Zyklus ergibt sich daraus eine maximale Eingangsfrequenz (an A oder B) von 5 MHz.

– Der Abstand zwischen einer Flanke an A und einer Flanke an B darf 50 ns nicht unterschreiten.
Impulsbreiten oder Pausenzeiten kürzer als 100 ns werden nicht gezählt.

– Eine Änderung der Phasenverschiebung (d.h. ≠90 Grad) hat Einfluss auf die maximale Eingangsfrequenz wegen der Mindestabstände der Flanken. Bei einem Abweichen von 90 Grad sinkt die maximale Eingangsfrequenz von 5 MHz beispielsweise bei 45 Grad auf 2,5 MHz.

Programmierbeispiel

```
INIT:                          Initialisieren
. . .                          Alle Zähler sperren
CNT_ENABLE(0)                  Inhalt des Zählers 1 löschen
CNT_CLEAR(1)                   externer Takteingang am CLK-Eingang
CNT_MODE(0)                    Modus Vier-Flankenauswertung aktivieren
CNT_SET(0)                     Eingang CLR/LATCH auf CLR-Modus stellen
CNT_INPUTMODE(0)               Zähler 1 freigeben
CNT_ENABLE(1)                  Aktuellen Zählerstand in Latch A übernehmen
. . .                          Latch A auslesen
                               Auswerten im Programm

EVENT:
CNT_LATCH(1)
CNT_READLATCH(1)
. . .
```

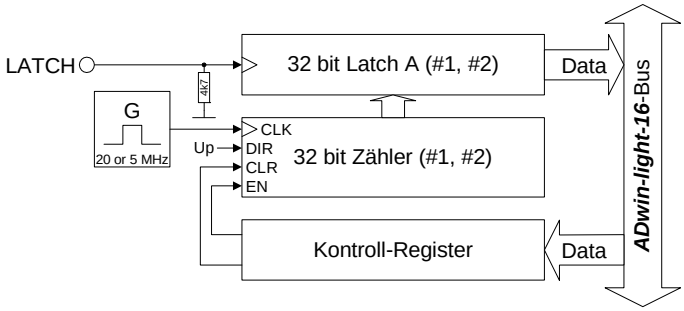
8.2.3 Betriebsart Pulsbreiten- und Periodendauer-Messung

In dieser Betriebsart taktet ein interner Referenztaktgeber den Zähler mit einer Signalfrequenz von 20 MHz oder (nach einem Teiler) 5 MHz. Alle Zähler besitzen einen Umschalter für die Signalfrequenz. Es können die Periodendauer oder die Pulsbreite eines Rechtecksignals am Eingang CLR/LATCH gemessen werden.

In diesem Modus müssen Sie bei hohen Frequenzen berücksichtigen, dass Ihr **`ProcessDelay`** kleiner bleibt als eine Signalperiode, um jeden Zyklus zu erfassen.

Periodendauer-Messung

In diesem Modus wird bei jeder positiven Flanke am Eingang LATCH der Zählerstand ins Latch A geschrieben, wobei der jeweils vorherige Stand überschrieben wird. Die Pulsbreite ergibt sich aus dem Produkt von Periodendauer des Referenztaktes und Zählerstandsdifferenz.



Programmierbeispiel

```
INIT:
. . .
CNT_ENABLE(0)
CNT_CLEAR(1)
CNT_MODE(1)
CNT_SET(0)
CNT_SET(1)
CNT_INPUTMODE(1)
CNT_ENABLE(1)
. . .

EVENT:
CNT_READLATCH(1)
. . .
```

Initialisieren

Alle Zähler sperren

Inhalt des Zählers 1 löschen

Modus interner Referenztakt am internen CLK-Eingang des Zählers ...

... mit 20 MHz oder

... mit 5 MHz

Eingang CLR/LATCH auf LATCH-Modus stellen

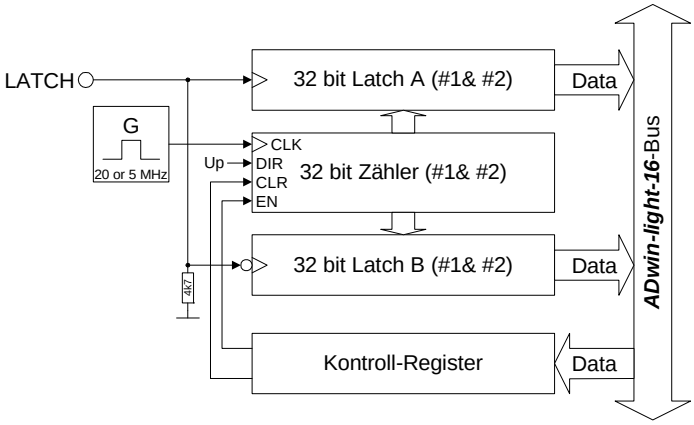
Zähler 1 freigeben

Latch A auslesen

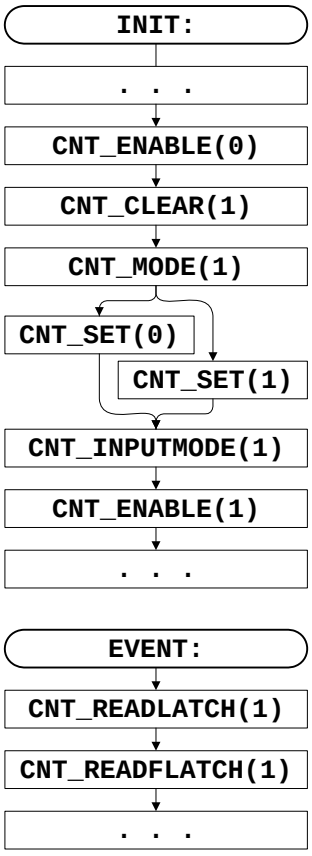
Auswerten im Programm

Messung von Pulsbreite und Pausenzeit

Die Zähler besitzen je ein Latch A für positive Flanken und ein Latch B für negative Flanken. Aus der Zählerstandsdifferenz der Latches können Pulsbreite und Pausenzeit unabhängig voneinander ermittelt werden.



Programmierbeispiel



Initialisieren

Alle Zähler sperren

Inhalt des Zählers 1 löschen

Modus interner Referenztakt am internen CLK-Eingang des Zählers ...

... mit 20 MHz oder

... mit 5 MHz

Eingang CLR/LATCH auf LATCH-Modus stellen

Zähler 1 freigeben

Latch A auslesen

Latch B auslesen

Auswerten im Programm

8.3 CAN-Bus

Die Erweiterungskarte DIO1 besitzt eine CAN-Bus Schnittstelle für das „high speed“-CAN-Protokoll. Die Anschlüsse stehen auf einem 9-poligen Sub-D-Verbinder (Stecker / männlich) zur Verfügung; die Pin-Belegung ist auf Seite 29 dargestellt.

Bus-Terminierung

Der CAN-Bus muss an seinem physikalischen Ende (und nur dort) mit einem Abschlusswiderstand terminiert werden, also nur am ersten und am letzten CAN-Knoten. Dies geschieht, indem Sie einen DIP-Schalter auf der DIO1-Platine entsprechend einstellen (siehe Abb. 25 – Lage der DIP-Schalter auf der DIO1-Platine).

Wenn die Terminierung erforderlich ist, legen Sie den DIP-Schalter nach oben um (L16 Rev. B) bzw. in Richtung zum CAN-Stecker (L16 Rev. A).

8.3.1 Funktionsbeschreibung des CAN-Controllers

Die CAN-Schnittstelle ist mit dem CAN-Controller AN82527 von Intel® bestückt und arbeitet nach der Spezifikation „CAN 2.0 part A+B“ sowie ISO 11898. Sie programmieren die Schnittstelle mit *ADbasic*-Befehlen, die direkt auf die Register des Controllers zugreifen.

Über den CAN-Bus verschickte Nachrichten sind Datentelegramme mit bis zu 8 Bytes, die durch sogenannte „Identifier“ gekennzeichnet sind. Der CAN-Controller unterstützt Identifier mit 11 Bit und 29 Bit Länge. Die eigentliche Kommunikation, d.h. die Verwaltung der Bus-Nachrichten, erfolgt über 15 „Message-Objekte“.

Zur Konfiguration und Statusanzeige des CAN-Controllers dienen die in ihm enthaltenen Register. Hier werden Busgeschwindigkeit, Interrupt handling usw. eingestellt (siehe separate Dokumentation „82527 - Serial Communications Controller, Architectural Overview“ von Intel®).

Der CAN-Bus (high speed) ist auf Frequenzen bis 1 MHz einstellbar und wird standardmäßig mit 1 MHz betrieben; bei CAN low speed beträgt die max. Frequenz 125 kHz. Der CAN-Bus ist durch Optokoppler vom *ADwin*-System galvanisch getrennt.

Der Eingang einer Nachricht kann einen Interrupt auslösen, der sofort einen Event am Prozessor erzeugt. Dadurch kann eine sofortige Bearbeitung der Nachrichten gewährleistet werden.

Nachrichten verwalten

Identifier

Der CAN-Controller unterscheidet über den Bus verschickte Nachrichten durch „Identifier“, das sind Kennzahlen mit einer definierten Bitlänge. Aus der Bitlänge ergeben sich hier die möglichen Kennzahlen $0...2^{11}-1$ bzw. $0...2^{29}-1$.

Message-Objekte

Jede Nachricht (zu sendende oder zu empfangende) speichert der Controller in einem von 15 „Message-Objekten“. Die Message-Objekte können jeweils entweder zum Senden oder zum Empfangen konfiguriert werden. Als Ausnahme kann das Message-Objekt 15 nur zum Empfangen genutzt werden.
Nach der Initialisierung des CAN-Controllers sind sämtliche Message-Objekte nicht konfiguriert und beteiligen sich nicht am Busverkehr.

Jedes Message-Objekt erhält einen Identifier, der die Zuordnung einer Nachricht zu einem Message-Objekt ermöglicht.

Nachricht übergeben

In *ADbasic* übergeben Sie eine Nachricht an ein Message-Objekt über das Feld **`can_msg`**, das 8 Datenbytes plus die Anzahl der Datenbytes aufnehmen kann (9 Elemente). Ebenso wird eine Nachricht beim Auslesen aus einem Message Objekt in das Feld **`can_msg`** übertragen.

Nachricht senden

Das Versenden einer Nachricht läuft in folgenden Schritten ab:

– Sie konfigurieren ein Message-Objekt zum Senden und definieren den Identifier des Objekts (Befehl **`En_Transmit`**).

– Sie speichern die Nachricht im Feld **`can_msg`**.

– Sie senden die Nachricht (Befehl **`Transmit`**). Die Nachricht im Feld **`can_msg`** wird an das Message-Objekt übergeben. Sobald der Bus frei ist, wird die Nachricht gesendet (mit dem Identifier des Message-Objekts).

Nachricht empfangen

Das Empfangen einer Nachricht läuft in folgenden Schritten ab:

– Sie konfigurieren ein Message-Objekt für Empfang und definieren den Identifier des Objekts (Befehl **En_Receive**).

– Der Controller überwacht den CAN-Bus auf eingehende Nachrichten und speichert Nachrichten mit dem richtigen Identifier in dem Message-Objekt.

– Sie übertragen die Nachricht aus dem Message-Objekt in das Feld **can_msg** (Befehl **Read_Msg**) und lesen den zugehörigen Identifier aus.

Eine eingehende Nachricht überschreibt die alten Daten in dem Message-Objekt, die dadurch unwiderruflich verloren sind. Achten Sie daher beim Programmieren darauf, dass die Daten schneller ausgelesen als empfangen werden. Ein Datenverlust wird durch ein Flag angezeigt.

Bei dem Message Objekt 15 existiert ein zusätzlicher interner Zwischenspeicher, so dass dort 2 Nachrichten gespeichert werden können.

Nachricht zuordnen

Die Zuordnung einer eingehenden Nachricht zu einem Message-Objekt wird automatisch durch einen Vergleich ihrer Identifier gesteuert. Die globale Maske (CAN-Register 6...7 bzw. 6...9) steuert diesen Vergleich:

– Der Identifier der Nachricht wird bitweise mit dem Identifier des Message-Objekts verglichen. Wenn die relevanten Bits gleich sind, wird die Nachricht in das Message-Objekt übernommen. Nicht relevante Bits werden nicht verglichen, d.h. die Nachricht wird (sofern es von diesem Bit abhängt) in das Objekt übernommen.

– Relevante Bits werden in der globalen Maske festgelegt, indem sie dort gesetzt werden.

Globale Maske

Durch die globale Maske kann ein Message-Objekt für den Empfang von Nachrichten mit **verschiedenen Identifiern** (ID) genutzt werden. Das folgende Beispiel zeigt die Zuordnung der Nachrichten-ID 1...4 zu den Message-Objekt-ID 1...4, wenn alle Bits der globalen Maske gesetzt sind bis auf die beiden niederwertigsten (bei einem 11-Bit-Identifier also **11111111100b**).

Nachrichten-ID	ID des Message-Objekts			
	1 **…001b**	2 **…010b**	3 **…011b**	4 **…100b**
1 (**…001b**)	x	x	x	0
2 (**…010b**)	x	x	x	0
3 (**…011b**)	x	x	x	0
4 (**…100b**)	0	0	0	x

x: Nachricht wird übernommen
0: Nachricht wird nicht übernommen

In diesem Beispiel entscheidet nur der Vergleich des Bits 2 über die Zuordnung, denn die Bits 3...10 der hier verglichenen Identifier sind identisch (= 0) und die Bits 0 und 1 werden nicht verglichen, weil sie in der globalen Maske auf Null gesetzt sind (= nicht relevant).

Busfrequenz einstellen

Die **CAN-Bus-Frequenz** hängt von der Konfiguration des Controllers ab.

Bei der Initialisierung mit **Init_CAN** wird der Controller automatisch so konfiguriert, dass die CAN-Bus-Frequenz 1 MHz beträgt. Soll der CAN-Bus mit einer anderen Frequenz betrieben werden, geschieht dies am einfachsten mit dem Befehl **Set_CAN_Baudrate**.

Bei CAN low speed muss die Busfrequenz auf Werte ≤125 kBit/s eingestellt werden.

Busfrequenz für Sonderfälle

In Sonderfällen kann es vorteilhaft sein, die Einstellungen anders zu wählen, als es mit **Set_CAN_Baudrate** möglich ist. Zu diesem Zweck müssen bestimmte Register mit dem Befehl **Poke** gesetzt werden. Der Registeraufbau ist in der Dokumentation des Controllers beschrieben.

Interrupt freigeben / Event auslösen

Sie können bei einem Message-Objekt freigeben, ob es beim Eingang einer Nachricht einen Interrupt auslöst. Der Interrupt-Ausgang des CAN-Controllers ist intern mit dem Event-Eingang des Prozessors verbunden. Dadurch kann der Prozessor sofort auf eingehende Nachrichten reagieren, ohne den Nachrichteneingang kontrollieren zu müssen (Polling).

Sie können die Interrupts mehrerer Message-Objekte freigeben. Welches Objekt den Interrupt ausgelöst hat, kann aus dem Interrupt-Register (**5Fh**) ersehen werden: Es enthält die Nummer des auslösenden Message-Objekts. Wird das Interrupt-Flag (new message flag) im Message-Objekt zurückgesetzt, wird das Interrupt-Register aktualisiert. Wenn kein Interrupt mehr ansteht, wird das Register auf „0“ gesetzt. Ist während der Bearbeitung des ersten Interrupts ein weiterer aufgetreten, so wird dessen Quelle nun im Interrupt-Register angezeigt. Ein weiterer Hardware-Interrupt erfolgt in diesem Fall nicht.

Wenn Sie Message-Objekte für das Auslösen von Interrupts freigeben, dürfen die Event-Eingänge (siehe Seite 32) nicht gleichzeitig beschaltet sein.

Programmierung

Die CAN-Funktionen der DIO1-Erweiterung werden mit *ADbasic*-Befehlen komfortabel programmiert. Die Befehle sind in einer Include-Datei enthalten; binden Sie diese Datei am Beginn des Programms ein mit der Befehlszeile

```
#Include ADWL16.INC
```

Die unten stehenden CAN-Befehle sind ab Seite 110 und in der Online-Hilfe vollständig beschrieben:

Bereich	Befehle
Initialisierung des CAN-Controllers	**Init_CAN**
Setzen und Lesen von Registern	**Set_CAN_Reg**, **Get_CAN_Reg**
Initialisieren von Message Objekten	**En_Receive**, **En_Transmit**
Senden und Empfangen von Datensätzen	**Transmit** **Read_Msg**, **Read_Msg_Con**
Freigeben von Interrupts	**En_Interrupt**
Baudrate einstellen	**Set_CAN_Baudrate**

Abb. 30 – DIO1 Übersicht CAN-Befehle

8.4 SSI-Decoder

An den SSI-Decoder kann ein Inkremental-Encoder mit SSI-Schnittstelle angeschlossen werden. Die Signale sind differentiell und haben RS422/485-Pegel.

Der Decoder kann entweder (auf Anforderung) einen einzelnen Wert auslesen oder aber kontinuierlich den aktuellen Wert bereit stellen.

Die Anschlüsse des Decoders stehen auf dem Stecker COUNTER (25-polig, Sub-D) zur Verfügung, und zwar auf den Pins 8, 9, 21 und 22 (siehe Abb. 24 – Übersichtsbild L16-EURO-DIO1 mit Pinbelegungen Für die anderen L16-Varianten sind die Steckerbezeichnungen identisch.). Auf den Pins 7 und 20 stehen DGND und +5V zur Verfügung.

Eigenschaften einstellen

Folgende Eigenschaften des SSI-Decoders sind per Software einstellbar:

– Taktrate: Über einen Vor-Teiler sind Taktraten von ca. 40kHz bis 1MHz möglich mit **SSI_Set_Clock**.

– Auflösung: Einstellbar bis 32 Bit mit **SSI_Set_Bits**.

Beispiel: Umsetzung von Gray-Code

Eine Umsetzung von Gray- in Binär-Code erfolgt durch eine zu programmierende Routine im *ADbasic*-Prozess (siehe unten).

```
REM PAR_1 = zu wandelnder Gray-Wert
REM PAR_2 = Flag für einen neuen Gray-Wert
REM PAR_9 = Ergebnis der Gray-zu-Binär-Wandlung

DIM m, n AS LONG

EVENT:
  IF(PAR_2=1) THEN      'Start der Wandlung
    m=0                 'Variable initialisieren
    PAR_9=0             ' -"-
    FOR n=1 TO 32       'Alle 32 Bits durchgehen
     m=(SHIFT_RIGHT(PAR_1,(32-n)) AND 1) XOR m
     PAR_9=(SHIFT_LEFT(m,(32-n))) OR PAR_9
    NEXT n
    PAR_2=0             'Nächste Wandlung ermöglichen
  ENDIF
```

Abb. 31 – Listing: Konvertierung von Gray- in Binär-Code

Programmierung

Die Funktionalität des SSI-Decoders wird mit *ADbasic*-Befehlen komfortabel programmiert:

Bereich	Befehle
Decoder-Modus einstellen	**SSI_Mode**
Daten empfangen	**SSI_Read**
Encoder-Auflösung einstellen	**SSI_Set_Bits**
Encoder-Taktrate einstellen	**SSI_Set_Clock**
Auslesen des Encoders starten	**SSI_Start**
Lesestatus	**SSI_Status**

Die Befehle sind in der Include-Datei <ADWL16.INC> enthalten und werden ab Seite 125 und in der Online-Hilfe erläutert.

9 DIO2- / DIO3-Erweiterung

Die DIO2-Erweiterung stellt Ihnen zusätzlich zur Verfügung:

– 32 digitale Ein-/Ausgänge (programmierbar in Gruppen zu 8), Seite 48.

  Die digitalen Ein- und Ausgänge der Basisversion stehen weiterhin zur Verfügung.

– 2 Zähler, Seite 49: 32 Bit Vor-/Rückwärtszähler zur Impuls-, Periodendauer- und Tastverhältnismessung sowie einer Vierflankenauswertung zum Anschluss von Encodern.
  Zähler 1 hat TTL-Eingänge (single ended), Zähler 2 hat differentielle Eingänge.

  Die Zähler der Basisversion werden von den DIO2-Zählern ersetzt.

– 1 SSI-Decoder (Seite 55)

  Der SSI-Decoder eignet sich zum Anschluss von Inkremental-Encodern mit SSI-Schnittstelle. Die Eingänge liegen auf dem Stecker COUNTER; die Signale sind differentiell und für RS422/485-Pegel (5V) ausgelegt.

Informationen zu den Funktionen der Basisversion finden Sie auf folgenden Seiten:

– 8 analoge Eingänge: Seite 11

– 2 analoge Ausgänge: Seite 11

– 6 digitale Eingänge, 6 digitale Ausgänge: Seite 14

– 1 serielle Schnittstelle LS-Bus, ab Rev. B2: Seite 17

Das Schema zeigt gemeinsam die Grundfunktionen eines L16-Geräts mit den Zusatzfunktionen der DIO2-Erweiterung (als USB-Variante).

DIO3-Erweiterung

Die DIO3-Erweiterung enthält nur die 32 digitalen Ein-/Ausgänge (siehe Seite 48) zusätzlich. Eigenschaften und Funktion dieser Ein-/ Ausgänge sind identisch zu den Ein-/Ausgängen der DIO2-Erweiterung.

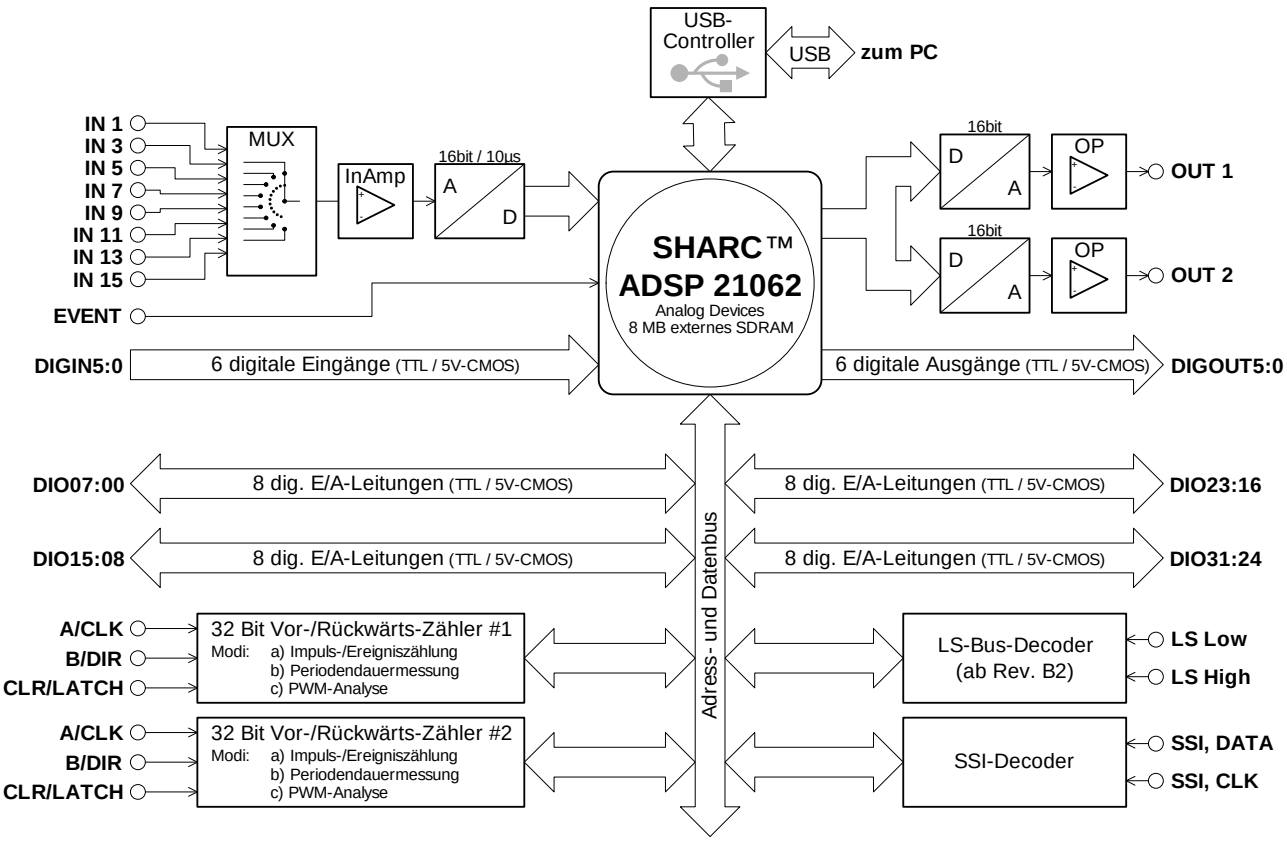


Abb. 32 – Schema *L16-DIO2* (mit USB-Schnittstelle)

Die Pin-Belegung am Anschluss „ADwin I/O-CONNECTOR“ ist identisch mit der Basisversion. Abweichend dazu dienen bei der Erweiterung DIO2 die Pins 14…16 nun alternativ auch als Eingänge für den Zähler 1.

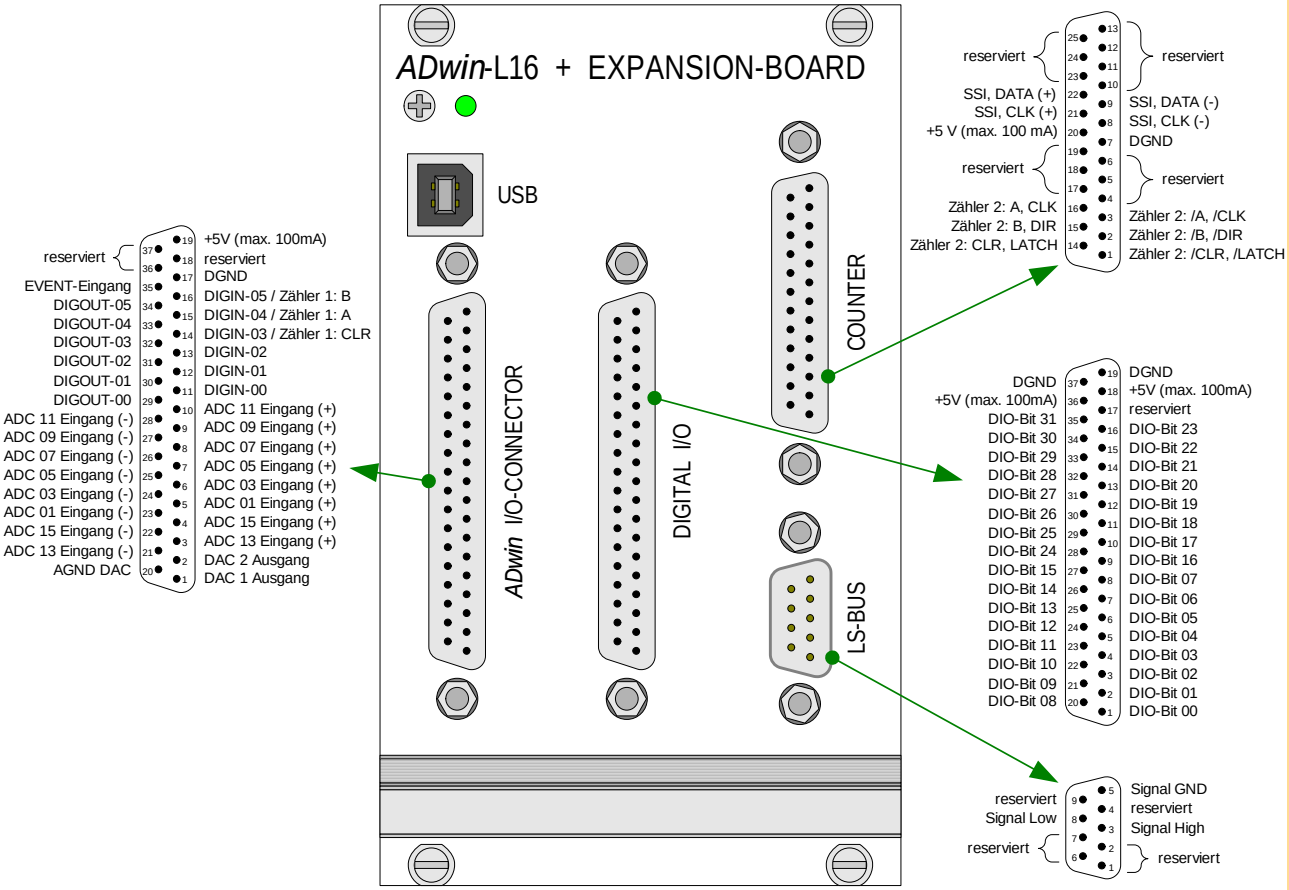


Abb. 33 – Übersichtsbild *L16-EURO-DIO2* mit Pinbelegungen

Die technischen Daten der DIO2- und DIO3-Erweiterung sind im Anhang enthalten.

9.1 Digitale Ein- und Ausgänge

Zusätzlich zu den digitalen Ein-/Ausgängen der Basisversion (Digin, Digout, EVENT) stehen Ihnen auf der 37-poligen Sub-D-Buchse „Digital I/O“ 32 digitale Ein- oder Ausgänge (Abkürzung: DIO) zur Verfügung. Sie sind in Gruppen zu jeweils 8 als Ein- oder Ausgang programmierbar.

Die digitalen Eingänge sind TTL-kompatibel und gegen Überspannung nicht geschützt.

Einschaltkonfiguration

Nach dem Einschalten des Gerätes sind alle 32 DIO-Leitungen als Eingang konfiguriert; dies entspricht dem Befehl **Conf_DIO_E(0)**.

Mit dem Befehl **Conf_DIO_E(n)** programmieren Sie die 32 DIO-Leitungen in 4 Gruppen zu jeweils 8 Leitungen als Ein- oder Ausgang (siehe Kapitel 13 oder *ADbasic* Online-Hilfe).

Die folgende Tabelle zeigt Ihnen die 16 möglichen Konfigurationen dieses Befehls. In der untersten Zeile stehen die Befehle, die für die DIO-Gruppen anwendbar sind.

Conf_DIO_E(n)	DIO 31 : DIO 24	DIO 23 : DIO 16	DIO 15 : DIO 08	DIO 07 : DIO 00
0	**IN**	**IN**	**IN**	**IN**
1	IN	IN	IN	OUT
2	IN	IN	OUT	IN
3	IN	IN	OUT	OUT
4	IN	OUT	IN	IN
5	IN	OUT	IN	OUT
6	IN	OUT	OUT	IN
7	IN	OUT	OUT	OUT
8	OUT	IN	IN	IN
9	OUT	IN	IN	OUT
10	OUT	IN	OUT	IN
11	OUT	IN	OUT	OUT
12	OUT	OUT	IN	IN
13	OUT	OUT	IN	OUT
14	OUT	OUT	OUT	IN
15	OUT	OUT	OUT	OUT
Anwendbare *ADbasic*-Befehle:	**Digin_Word2_E** **Digout_Word2_E** **Digout_Set2_E** **Digout_Reset2_E**		**Digin_Word1_E** **Digout_Word1_E** **Digout_Set1_E** **Digout_Reset1_E**	
	Digin_Long_E, **Digout_Long_E**			

Abb. 34 – Konfigurationen mit **Conf_DIO_E**

Näheres zur Programmierung zeitkritischer Aufgaben finden Sie im Kapitel auf Seite 17.

9.2 Zähler

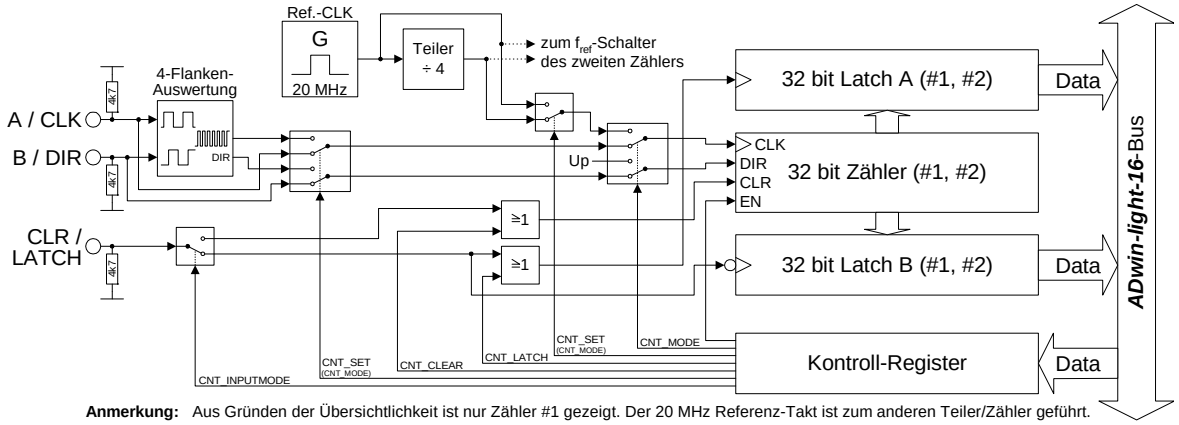
Zähler

Die Erweiterung DIO2 stellt **zwei 32 Bit Zähler** zur Verfügung, die Sie per Software sowohl einzeln als auch gemeinsam konfigurieren und auslesen können. Sie können an Zähler 1 TTL-Signale (single ended) anlegen, an Zähler 2 differentielle Signale.
Die Zähler ersetzen die Inkrementalzähler der Grundversion.

Latch

Die Zähler können **intern oder extern getaktet** werden und werden über zugeordnete Latches ausgelesen. Alle Zähler haben je ein Latch A sowie ein Latch B (die Grafik zeigt den Aufbau eines einzelnen Zählers).

Der Zählerstand kann mit Programmierbefehlen oder (bei entsprechender Einstellung) bei einem externen Signal an CLR/LATCH gelöscht oder in ein Latch übertragen werden.



Anmerkung: Aus Gründen der Übersichtlichkeit ist nur Zähler #1 gezeigt. Der 20 MHz Referenz-Takt ist zum anderen Teiler/Zähler geführt.

Abb. 35 – Schema DIO2-Zähler

Es gibt die Betriebsarten Ereigniszählung (externer Takt) und Pulsbreitenmessung (interner Takt); siehe auch Kapitel 9.2.2 / 9.2.3:

Externer Takteingang

1. **Ereigniszählung**: Das In-/Dekrementieren des Zählers wird durch externe Rechtecksignale an den Eingängen A/CLK und B/DIR ausgelöst. Ein Signal an CLR/LATCH bewirkt, dass entweder der Zähler auf Null gesetzt (CLR) oder der Zählerstand ins Latch geschrieben wird (LATCH).

   Es gibt die Modi:
   - **Takt und Richtung**: Jede positive Flanke an CLK in- oder dekrementiert den Zählerstand um eins. Das Signal an B/DIR bestimmt die Zählrichtung (0 = Dekrement; 1 = Inkrement).
   - **Vierflankenauswertung**: Jede Flanke der (um 90 Grad) versetzten Signale an A/CLK und an B/DIR löst ein In-/ Dekrementieren des Zählers aus. Die Zählrichtung ergibt sich aus der Reihenfolge der steigenden/fallenden Flanken dieser Signale. Dieser Modus wird besonders für Inkrementalgeber (Winkel-Encoder) eingesetzt.

Interner Takteingang

2. **Pulsbreitenmessung**: Die Signale zum In-/Dekrementieren erhält der Zähler von einem internen Referenztaktgeber mit einer Signalfrequenz von 20 MHz (alternativ 5 MHz nach einem Teiler). Ausgewertet wird das an CLR/LATCH anliegende Rechtecksignal: Mit jeder positiven Flanke wird der Zählerstand in Latch A geschrieben, mit jeder negativen in Latch B.

   Sie können berechnen:
   - die Periodendauer des Eingangssignals an CLR/LATCH aus den Werten in Latch A.
   - die Pulsbreite und Pausenzeit aus den Werten in Latch A und Latch B.

9.2.1 Programmierung

Die Zähler-Funktionen der DIO2-Erweiterung werden mit *ADbasic*-Befehlen komfortabel programmiert. Die Befehle sind in einer Include-Datei enthalten; binden Sie diese Datei am Beginn des Programms ein mit der Befehlszeile

```
#INCLUDE ADWL16.INC
```

Die Zähler-Befehle der folgenden Tabelle sind im *ADbasic*-Handbuch oder der Online-Hilfe vollständig beschrieben:

Zähler-Nr.	2	1	Kommentar
Bit	1	0	
Cnt_Clear( )	0	0	ohne Einfluss
	1	1	Zähler löschen *
Cnt_Enable( )	0	0	Zähler sperren
	1	1	Zähler freigeben (auf laufende Zähler achten)
Cnt_InputMode( )	0	0	CLR/LATCH-Eingang auf CLR-Modus stellen
	1	1	CLR/LATCH-Eingang auf LATCH-Modus stellen
Cnt_Latch( )	0	0	ohne Einfluss
	1	1	Zählerstand in Latch-Register A übernehmen*
Cnt_Mode( )	0	0	externer Takteingang
	1	1	interner Referenztakt (20MHz / 5MHz)
Cnt_Set( )	0	0	**Cnt_Mode**-Bit = 0 : 4-Flankenauswertung
			Cnt_Mode-Bit = 1 : interner Referenztakt von 20MHz
	1	1	**Cnt_Mode**-Bit = 0 : Takt- und Richtungseingang (CLK & DIR)
			Cnt_Mode-Bit = 1 : interner Referenztakt von 5MHz
Cnt_ClearEnable()	0	0	CLR-Eingang sperren
	1	1	CLR-Eingang freigeben
Cnt_GetStatus(#)**			Status-Register auslesen (Bedeutung der Bits siehe *ADbasic*-Handbuch / Online-Hilfe)
Cnt_Read(#)**			Zählerstand in Latch A übernehmen und auslesen
Cnt_ReadLatch(#)**			Latch A (getriggert durch positive Flanke) auslesen
Cnt_ReadFLatch(#)**			Latch B (getriggert durch negative Flanke) auslesen

* Nach der Befehlsausführung werden gesetzte Bits automatisch wieder gelöscht. Bei anderen Befehlen müssen gesetzte Bits durch einen neuen Befehlsaufruf gelöscht werden.
** # = Zähler-Nummer 1 oder 2

Abb. 36 – DIO2-Zähler Befehle Kurzreferenz

Initialisierung

Sie können mit den Befehlen der Tabelle jeden Zähler einzeln oder beide Zähler gemeinsam konfigurieren.

Initialisieren Sie die Zähler bitte in dieser Reihenfolge:

1. Gewünschten Zähler sperren (**Cnt_Enable**)

   Der Befehl **Cnt_Enable** spricht alle Zähler gemeinsam an. Auch wenn Sie nur einen Zähler sperren (Bit = 0) möchten, müssen Sie alle anderen Zähler konfigurieren, deren Zustand unverändert bleiben soll (Bit = 1).
   Beim Freigeben des Zählers gilt dies entsprechend.

2. Betriebsart / Modus einstellen
   (**Cnt_Mode**, **Cnt_Set**, **Cnt_InputMode**)

   Beachten Sie die Abhängigkeit des Befehls **Cnt_Set** vom Befehl **Cnt_Mode**.

3. Zähler löschen (**Cnt_Clear**)

4. Zähler freigeben (**Cnt_Enable**)

Werten Sie bitte den Zählerinhalt nur mit Variablen vom Typ LONG aus. *ADbasic* behält dann intern das gelesene Bitmuster unverändert bei und berücksichtigt automatisch den Übergang zwischen positivem und negativem Zahlenbereich (siehe auch Seite 16). Damit gilt:

Die Zählrichtung (vor- oder rückwärts) ergibt sich zuverlässig nur aus dem

Vorzeichen der Differenz: [neuer Zählerstand] – [alter Zählerstand]

und nicht aus dem *Vergleich* (< >) der Zählerstände.

Bestimmen Sie die Differenz nur mit Variablen vom Typ LONG.

Ein Prozess kann sehr schnell abgearbeitet werden, wenn Sie mit den Befehlen **Peek** und **Poke** direkt auf die Steuer- und Datenregister zugreifen (siehe auch Kapitel oder *ADbasic*-Handbuch / Online-Hilfe).
Die Hardware-Adressen der DIO2-Erweiterung können Sie folgender Tabelle entnehmen (vgl. Abb. 36 – DIO2-Zähler Befehle Kurzreferenz).

Adresse [hex]	Funktion	Bit 31:16	15:10	9	8	7	6	5	4	3	2	1	0	Kommentar
20 40 02 04	Inhalt Latch A, Zähler-#1	x	x	x	x	x	x	x	x	x	x	x	x	x : Inhalt des Latch
20 40 02 08	Inhalt Latch B, Zähler-#1	x	x	x	x	x	x	x	x	x	x	x	x	x : Inhalt des Latch
20 40 02 14	Inhalt Latch A, Zähler-#2	x	x	x	x	x	x	x	x	x	x	x	x	x : Inhalt des Latch
20 40 02 18	Inhalt Latch B, Zähler-#2	x	x	x	x	x	x	x	x	x	x	x	x	x : Inhalt des Latch
20 40 03 00	Zähler sperren / freigeben **Cnt_Enable**()	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : Zähler sperren x = 1 : Zähler freigeben
20 40 03 10	Zähler löschen **Cnt_Clear**()	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : kein Einfluss x = 1 : Zähler löschen
20 40 03 20	Zähler latchen **Cnt_Latch**()	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : kein Einfluss x = 1 : Zähler latchen
20 40 03 30	Eingang: CLR oder LATCH	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : CLR-Eingang x = 1 : LATCH-Eingang
20 40 03 40	Impuls-/Ereigniszähler- oder Puls-breiten-/Periodendauermessung	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : externer Takteingang x =1: interner Referenztakt (20MHz/5MHz)
20 40 03 50	4-Flankenauswertung / CLK+DIR oder 20MHz / 5MHz Referenztakt	-	-	-	-	-	-	-	-	x	x	x	x	**Cnt_Mode**=0: x=0: 4-Fl.; x=1:CLK+DIR **Cnt_Mode**=1: x=0: 20MHz; x=1: 5MHz
20 40 04 54	DIO_Erweiterung Bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x: setzt/löscht Ausgang <Bit-Nr.>
20 40 04 64	DIO_Erweiterung Bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x: setzt/löscht Ausgang <Bit-Nr.+16>
20 40 04 74	DIO_Erweiterung Set_Bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang <Bit-Nr.> setzen *
20 40 04 84	DIO_Erweiterung Set_Bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang <Bit-Nr.+16> setzen*
20 40 04 94	DIO_Erweiterung Reset_Bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang <Bit-Nr.> löschen *
20 40 04 A4	DIO_Erweiterung Reset_Bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang <Bit-Nr.+16> löschen *
20 40 04 6C	Ein-/Ausgänge konfigurieren **Conf_DIO**() Bit 0: DIO 00...07; Bit 1: DIO 08...15 Bit 2: DIO 16...23; Bit 3: DIO 24...31	-	-	-	-	-	-	-	-	x	x	x	x	x = 0: Kanäle als Eingang definieren x = 1: Kanäle als Ausgang definieren
* Funktion bei Eingängen ohne Einfluss														

Abb. 37 – DIO2 Hardware-Adressen der Steuer- und Datenregister

9.2.2 Betriebsart Impuls-/Ereigniszähler

Externe Rechtecksignale an den Eingängen A/CLK und B/DIR takten in dieser Betriebsart den jeweiligen Zähler. Mit **Cnt_Set** aktivieren Sie entweder den Modus zur Ermittlung von Taktfrequenz und Richtung oder die Vierflankenauswertung.

Der Eingang CLR/LATCH kann benutzt werden, um (jeweils bei einem dort anliegenden High-Signal)

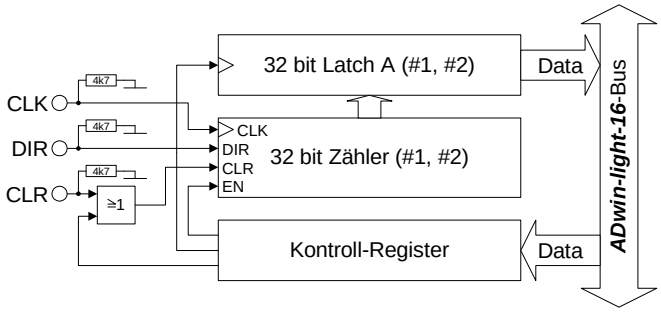
Löschen

– den Zähler zu löschen (CLR)

Latchen

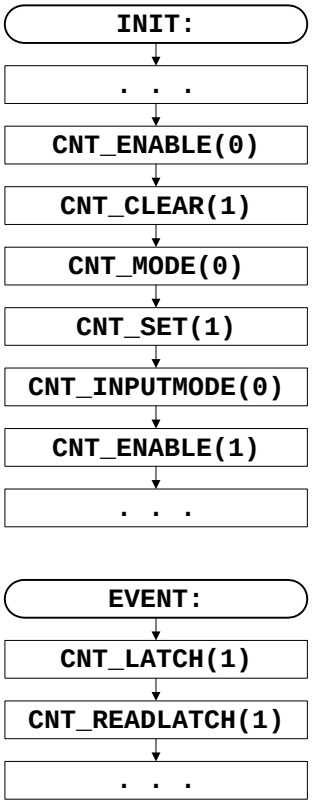
– den Zählerstand in Latch A zu übernehmen (LATCH).

Takt und Richtung



Jede positive Flanke eines Rechtecksignals auf dem CLK-Eingang (Clock) wird bis zu einer maximalen Frequenz von 20 MHz gezählt. Die Richtung ergibt sich aus einem High- (vorwärts) bzw. Low-Signal (rückwärts) auf dem DIR-Eingang (Direction); dieses Signal kann als konstante Spannung oder als Rechtecksignal (z.B. vorgegeben durch eine externe Logikschaltung) angelegt sein.

Programmierbeispiel



Initialisieren

Alle Zähler sperren

Inhalt des Zählers 1 löschen

externer Takteingang am CLK-Eingang

Modus Takt und Richtung aktivieren

Eingang CLR/LATCH auf CLR-Modus stellen

Zähler 1 freigeben

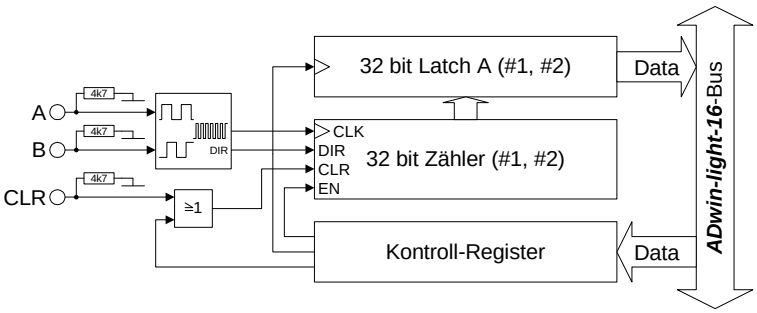
Aktuellen Zählerstand in Latch A übernehmen

Latch A auslesen

Auswerten im Programm

Vier-Flanken-Auswertung

Dieser Modus ermittelt Takt und Zählrichtung aus zwei Rechteck-Signalen, die an den Eingängen A und B um 90 Grad (ideal) versetzt anliegen. Die Zählrichtung ergibt sich logisch aus der zeitlichen Reihenfolge der steigenden / fallenden Flanken zueinander.



Berücksichtigen Sie bitte:

– Der Zähler registriert bei einem Zyklus 4 Flanken.

– Die maximale Zählfrequenz beträgt 20 MHz. Gemeinsam mit den 4 Flanken je Zyklus ergibt sich daraus eine maximale Eingangsfrequenz (an A oder B) von 5 MHz.

– Der Abstand zwischen einer Flanke an A und einer Flanke an B darf 50 ns nicht unterschreiten.
Impulsbreiten oder Pausenzeiten kürzer als 100 ns werden nicht gezählt.

– Eine Änderung der Phasenverschiebung (d.h. ≠90 Grad) hat Einfluss auf die maximale Eingangsfrequenz wegen der Mindestabstände der Flanken. Bei einem Abweichen von 90 Grad sinkt die maximale Eingangsfrequenz von 5 MHz beispielsweise bei 45 Grad auf 2,5 MHz.

Programmierbeispiel

```
INIT:                          Initialisieren
. . .                          Alle Zähler sperren
CNT_ENABLE(0)                  Inhalt des Zählers 1 löschen
CNT_CLEAR(1)                   externer Takteingang am CLK-Eingang
CNT_MODE(0)                    Modus Vier-Flankenauswertung aktivieren
CNT_SET(0)                     Eingang CLR/LATCH auf CLR-Modus stellen
CNT_INPUTMODE(0)               Zähler 1 freigeben
CNT_ENABLE(1)                  Aktuellen Zählerstand in Latch A übernehmen
. . .                          Latch A auslesen
                               Auswerten im Programm

EVENT:
CNT_LATCH(1)
CNT_READLATCH(1)
. . .
```

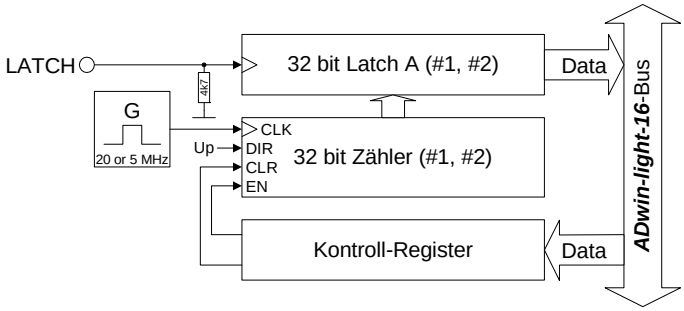
9.2.3 Betriebsart Pulsbreiten- und Periodendauer-Messung

In dieser Betriebsart taktet ein interner Referenztaktgeber den Zähler mit einer Signalfrequenz von 20 MHz oder (nach einem Teiler) 5 MHz. Alle Zähler besitzen einen Umschalter für die Signalfrequenz. Es können die Periodendauer oder die Pulsbreite eines Rechtecksignals am Eingang CLR/LATCH gemessen werden.

In diesem Modus müssen Sie bei hohen Frequenzen berücksichtigen, dass Ihr **PROCESSDELAY** kleiner bleibt als eine Signalperiode, um jeden Zyklus zu erfassen.

Periodendauer-Messung

In diesem Modus wird bei jeder positiven Flanke am Eingang LATCH der Zählerstand ins Latch A geschrieben, wobei der jeweils vorherige Stand überschrieben wird. Die Pulsbreite ergibt sich aus dem Produkt von Periodendauer des Referenztaktes und Zählerstandsdifferenz.



Programmierbeispiel

```
INIT:
. . .
CNT_ENABLE(0)
CNT_CLEAR(1)
CNT_MODE(1)
CNT_SET(0)
CNT_SET(1)
CNT_INPUTMODE(1)
CNT_ENABLE(1)
. . .

EVENT:
CNT_READLATCH(1)
. . .
```

Initialisieren

Alle Zähler sperren

Inhalt des Zählers 1 löschen

Modus interner Referenztakt am internen CLK-Eingang des Zählers ...

... mit 20 MHz oder

... mit 5 MHz

Eingang CLR/LATCH auf LATCH-Modus stellen

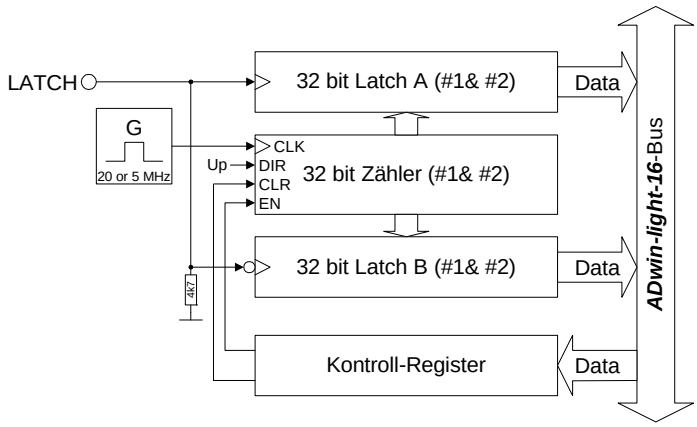
Zähler 1 freigeben

Latch A auslesen

Auswerten im Programm

Messung von Pulsbreite und Pausenzeit

Die Zähler besitzen je ein Latch A für positive Flanken und ein Latch B für negative Flanken. Aus der Zählerstandsdifferenz der Latches können Pulsbreite und Pausenzeit unabhängig voneinander ermittelt werden.



Programmierbeispiel

```
INIT:
. . .
CNT_ENABLE(0)
CNT_CLEAR(1)
CNT_MODE(1)
CNT_SET(0)
CNT_SET(1)
CNT_INPUTMODE(1)
CNT_ENABLE(1)
. . .

EVENT:
CNT_READLATCH(1)
CNT_READFLATCH(1)
. . .
```

Initialisieren

Alle Zähler sperren

Inhalt des Zählers 1 löschen

Modus interner Referenztakt am internen CLK-Eingang des Zählers ...

... mit 20 MHz oder

... mit 5 MHz

Eingang CLR/LATCH auf LATCH-Modus stellen

Zähler 1 freigeben

Latch A auslesen

Latch B auslesen

Auswerten im Programm

9.3 SSI-Decoder

An den SSI-Decoder kann ein Inkremental-Encoder mit SSI-Schnittstelle angeschlossen werden. Die Signale sind differentiell und haben RS422/485-Pegel.

Der Decoder kann entweder (auf Anforderung) einen einzelnen Wert auslesen oder aber kontinuierlich den aktuellen Wert bereit stellen.

Die Anschlüsse des Decoders stehen auf dem Stecker COUNTER (25-polig, Sub-D) zur Verfügung, und zwar auf den Pins 8, 9, 21 und 22 (siehe Abb. 33 – Übersichtsbild L16-EURO-DIO2 mit Pinbelegungen). Auf den Pins 7 und 20 stehen DGND und +5V zur Verfügung.

Eigenschaften einstellen

Folgende Eigenschaften des SSI-Decoders sind per Software einstellbar:

– Taktrate: Über einen Vor-Teiler sind Taktraten von ca. 40 kHz bis 1 MHz möglich mit **SSI_Set_Clock**.

– Auflösung: Einstellbar bis 32 Bit mit **SSI_Set_Bits**.

Beispiel: Umsetzung von Gray-Code

Eine Umsetzung von Gray- in Binär-Code erfolgt durch eine zu programmierende Routine im *ADbasic*-Prozess (siehe unten).

```
REM PAR_1 = zu wandelnder Gray-Wert
REM PAR_2 = Flag für einen neuen Gray-Wert
REM PAR_9 = Ergebnis der Gray-zu-Binär-Wandlung

DIM m, n AS LONG

EVENT:
  IF(PAR_2=1) THEN       'Start der Wandlung
    m=0                  'Variable initialisieren
    PAR_9=0              ' -"-
    FOR n=1 TO 32        'Alle 32 Bits durchgehen
     m=(SHIFT_RIGHT(PAR_1,(32-n)) AND 1) XOR m
     PAR_9=(SHIFT_LEFT(m,(32-n))) OR PAR_9
    NEXT n
    PAR_2=0              'Nächste Wandlung ermöglichen
  ENDIF
```

Abb. 38 – Listing: Konvertierung von Gray- in Binär-Code

Programmierung

Die Funktionalität des SSI-Decoders wird mit *ADbasic*-Befehlen komfortabel programmiert:

Bereich	Befehle
Decoder-Modus einstellen	**SSI_MODE**
Daten empfangen	**SSI_Read**
Encoder-Auflösung einstellen	**SSI_Set_BITS**
Encoder-Taktrate einstellen	**SSI_Set_CLOCK**
Auslesen des Encoders starten	**SSI_START**
Lesestatus	**SSI_STATUS**

Die Befehle sind in der Include-Datei <ADWL16.INC> enthalten und werden im Handbuch *ADbasic* und der Online-Hilfe erläutert.

10 PWM1-Erweiterung

Die PWM1-Erweiterung stellt Ihnen zusätzlich zur Verfügung:

– 1 PWM-Ausgang (Seite 58).

  Ein digitaler Ausgang der Basisversion wird von dem PWM1-Ausgang ersetzt.

– 1 SPI-Schnittstelle mit der Funktion eines SPI-Master (Seite 59).

Die PWM1-Erweiterung kann mit der Light-16-Basisvariante und allen Erweiterungen (CO1, DIO1, DIO2, DIO3) kombiniert werden.

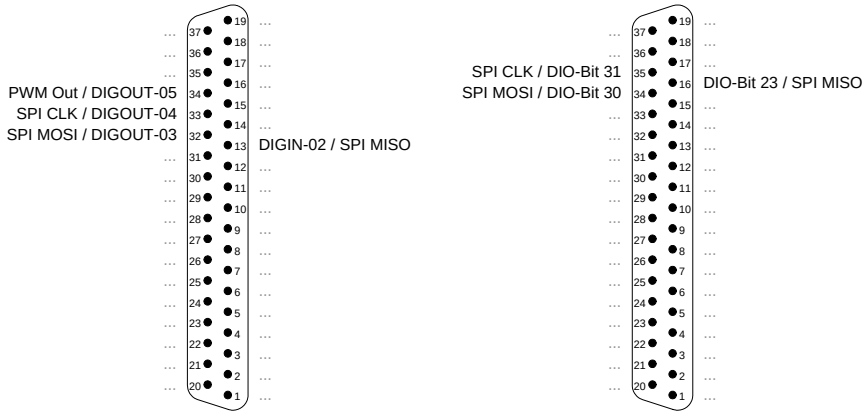
Pin-Belegungen

Die PWM1-Erweiterung ist eine nachrüstbare Erweiterung und verwendet bereits vorhandene Pins auf den Sub-D-Buchsen `ADwin I/O-Connector` und `Digital I/O`. Die nun doppelt belegten Pins werden per Software zwischen der Originalfunktion und der neuen Funktion umgeschaltet.

Bei der Light-16-Basisvariante und den Erweiterungen CO1 und DIO1 können nur Pins auf der Sub-D-Buchse `ADwin I/O-Connector` umgeschaltet werden.

Bei den Erweiterungen DIO2 und DIO3 können alternativ auch Pins für die SPI-Schnittstelle auf der Sub-D-Buchse `Digital I/O` umgeschaltet werden.

Die umschaltbaren Pins haben in den Pin-Belegungen (unten) die folgenden Bezeichnungen: `PWM Out`, `SPI CLK`, `SPI MOSI` und `SPI MISO`.



Sub-D-Buchse `ADwin I/O-Connector`

Sub-D-Buchse `Digital I/O` (nur bei DIO2 und DIO3)

Abb. 39 – Pinbelegungen

10.1 PWM-Ausgang

Die Erweiterung PWM1 stellt einen PWM-Ausgang zur Verfügung. Die PWM-Ausgabe ermöglicht, auf dem PWM-Ausgang ein pulsweitenmoduliertes Signal mit wählbarem Tastverhältnis auszugeben. Die Ausgabe wird mit 40 MHz getaktet.

Der PWM-Ausgang steht auf Pin 34 der 37-poligen Sub-D-Buchse `ADwin I/O-Connector` zur Verfügung (siehe oben). Pin 34 ist doppelt belegt und kann entweder als digitaler Ausgang `DIGOUT-05` betrieben werden oder als PWM-Ausgang. Sie schalten die Funktion des Pins per Software mit dem Befehl **`PWM_Activate`** um.

Nach dem Einschalten ist Pin 34 als digitaler Ausgang `DIGOUT-05` konfiguriert.

Programmierung

Die Funktionalität des PWM-Ausgangs wird mit *ADbasic*-Befehlen komfortabel programmiert:

Bereich	Befehle
Pin 34 auf PWM-Ausgabe umschalten	**`PWM_Activate`**
PWM-Ausgang initialisieren Betriebsmodus setzen	**`PWM_Init`** **`PWM_Reset`** **`PWM_Standby_Value`**
PWM-Ausgabe starten	**`PWM_Enable`**
PWM-Modus einstellen	**`PWM_Write_Latch`**
PWM-Modus und -Status lesen	**`PWM_Get_Status`** **`PWM_Latch`**

Die Befehle sind in der Include-Datei `<ADwL16.inc>` enthalten und werden in Kapitel 13.7 ab Seite 131 oder in der Online-Hilfe von *ADbasic* erläutert.

10.2 SPI-Schnittstelle

Die Erweiterung PWM1 stellt eine SPI-Schnittstelle mit der Funktion eines SPI-Master zur Verfügung. Der SPI Master arbeitet mit einer Busfrequenz von max. 5 MHz.

Die SPI-Schnittstelle verwendet für die SPI-Signale 3 Pins der Sub-D-Buchse ADwin I/O-Connector oder der Sub-D-Buchse Digital I/O (siehe Tabelle). Diese Pins sind doppelt belegt und können entweder als digitaler Ausgang betrieben werden oder für ein SPI-Signal. Sie schalten die Funktion per Software mit dem Befehl **SPI_Enable** um. Die Pinbelegung ist auf Seite 57 dargestellt.

SPI-Signal	ADwin I/O-Connector		Digital I/O	
	Pin	ersetzt	Pin	ersetzt
SPI CLK: Takt	33	DIGOUT-04	35	DIO-31
SPI MOSI: Master out, slave in	32	DIGOUT-03	34	DIO-30
SPI MISO: Master in, slave out	13	DIGIN-02	16	DIO-23
Slave Select (SS)	z.B. digitaler Ausgang			

Beachten Sie: Wenn Sie Pins auf der Sub-D-Buchse Digital I/O für SPI-Signale umschalten, werden automatisch die Pins DIO-24…DIO-31 als Ausgänge und die Pins DIO16…DIO-23 als Eingänge konfiguriert. Wenn Sie die Pins später wieder als Digitalkanäle schalten, gilt wieder die vorherige Konfiguration.

Zusätzlich benötigen Sie zum Ansprechen jedes SPI-Slave eine separate Slave Select-Leitung. Wenn Sie hierzu die übrigen digitalen Ausgänge verwenden, stellen Sie das gewünschte TTL-Signal mit den entsprechenden Befehlen für digitale Ausgänge ein, z.B. mit **Digout_Clear** oder **Digout_Set**.

SPI-Protokoll[1]

Es können theoretisch beliebig viele Teilnehmer an den SPI-Bus angeschlossen werden, wobei es immer exakt einen Master geben muss. Der Master erzeugt das Taktsignal (Clock) an SPI CLK und legt über eine Leitung Slave Select (SS) fest, mit welchem Slave er kommunizieren will. Wird SS auf den passenden Pegel gezogen, wird der jeweilige Slave aktiv, "lauscht" an SPI MOSI und legt seine Daten im Takt von SPI CLK an SPI MISO. Es wird somit ein Datenwort vom Master zum Slave und ein anderes Datenwort vom Slave zum Master transportiert.

Ein Protokoll für die Datenübertragung wurde nicht festgelegt, doch haben sich in der Praxis vier Modi durchgesetzt. Die Modi werden durch die Parameter Clock Polarität (CPOL) und Clock Phase (CPHA) festgelegt:

Modus	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

- Clock Polarität: Bei CPOL=0 ist der Clock Idle Low, bei CPOL=1 ist er Idle High.
- Clock Phase: CPHA gibt an, bei der wievielten Flanke die Daten übernommen werden. Bei CPHA=0 werden sie bei der ersten Flanke übernommen, nachdem SS aktiv gezogen wurde, bei CPHA=1 bei der zweiten.
- Modi 0…3: Somit werden die Daten bei CPOL=0 und CPHA=0 mit der ersten Flanke übernommen, die nur eine High-Flanke sein kann. Bei CPHA=1 wäre es die zweite, also eine Low-Flanke. Bei CPOL=1 ist das ganze folglich genau andersherum, bei CPHA=0 Low-Flanke und bei CPHA=1 High-Flanke.

Zu beachten ist noch, dass der Slave bei CPHA=0 seine Daten schon beim Aktivieren von SS an SPI MISO anlegt, damit der Master sie beim ersten Flankenwechsel übernehmen kann. Bei CPHA=1 werden die Daten vom Slave erst beim ersten Flankenwechsel an SPI MISO gelegt, damit sie beim zweiten Flankenwechsel vom Master übernommen werden können.

Mit jeder Taktperiode wird ein Bit übertragen. Beim üblichen Bytetransfer sind also 8 Taktperioden für eine vollständige Übertragung nötig. Es können auch mehrere Bytes

1. Quelle: Wikipedia

Programmierung

hintereinander übertragen werden, wobei nicht festgelegt ist, ob zwischen jedem Byte das Signal SS kurz wieder auf High gezogen werden muss. Eine Übertragung ist beendet, wenn das Signal SS endgültig auf High gesetzt wird.

Die Funktionalität der SPI-Schnittstelle wird mit *ADbasic*-Befehlen komfortabel programmiert:

Bereich	Befehle
SPI-Slave für Datenübertragung aktivieren (slave select)	beispielsweise **Digout_Clear**, **Digout_Set**
Pins auf SPI-Signale umschalten	**SPI_Enable**
SPI-Schnittstelle konfigurieren	**SPI_Config**
Daten schreiben und lesen	**SPI_Set_MOSI**, **SPI_Get_MISO**
Datenübertragung starten	**SPI_Start**
Ende der Datenübertragung abwarten	**SPI_Wait**
Status der Datenübertragung lesen	**SPI_Status**
TTL-Pegel auf der Datenleitung lesen	**SPI_Static_MISO**

Die Befehle sind in der Include-Datei <ADwL16.inc> enthalten und werden in Kapitel 13.8 ab Seite 141 oder in der Online-Hilfe von *ADbasic* erläutert.

11 *ADwin-light-16*-Boot

Diese Option ist nur in Verbindung mit dem Ethernet-Interface möglich, d.h. nur bei *L16-EXT-ENET* oder *L16-EURO-ENET*.

ADwin-light-16-Boot startet eine zuvor programmierte Anwendung automatisch nach dem Einschalten. Damit ist nach dem Einrichten der Anwendung ein Betrieb ohne PC möglich.

Folgende Schritte führt *ADwin-light-16*-Boot nach dem Einschalten aus:

– Laden des Betriebssystems
– Laden der mit dem *ADbasic* kompilierten Prozesse (max. 10)
– Automatisches Starten des Prozesses Nr. 10. Hier müssen Sie auch das Starten weiterer Prozesse programmieren.

Wenn Sie nicht mit der Bootloader-Option arbeiten wollen:

– Booten Sie das System nach dem Einschalten, und die gespeicherten Prozesse werden deaktiviert.
– Nach dem Ausschalten und erneutem Einschalten ist die Bootloader-Option wieder aktiv.

Durch Beschreiben des Flash-EEPROMs ohne Prozesse und nur mit der <ADwin9.btl>-Datei wird das System nach dem erneuten Einschalten nur noch gebootet aber ein Prozess kann nicht ausgeführt werden.

Wenn Sie *ADwin*-Software für Entwicklungsumgebungen von der beigefügten *ADwin*-CD installieren, wird das Programm für die Bootloaderoption (ADethflash) automatisch kopiert. Die Version der CD sollte 3.00.2735 oder höher sein.

Benutzen Sie für ein *ADwin-light-16*-System mit einem Ethernet-Interface das Programm ADethflash.

Bei Standardinstallation finden Sie das Programm in dem Verzeichnis <C:\ADwin\Tools\Ethernet Interface\...>.

Hinweise zum Bootloader mit Ethernet-Interface finden Sie in der *ADwin* Treiber-Installation.

In Verbindung mit dem Ethernet-Interface mit Bootloader können Sie bis zu 2.000 Long- oder Float-Werte zu 32 Bit mittels *ADbasic*-Prozess im Flash-EEPROM-Speicher ablegen und auslesen. Eine nähere Beschreibung hierzu können Sie im Programm <ADethflash.exe> aufrufen mit dem Schaltknopf „Info about eeprom support“.

12 Zubehör

Für das *ADwin-light-16*-System ist folgendes Zubehör lieferbar:

– *ADbasic*, das Echtzeit-Entwicklungstool für die Programmierung aller *ADwin*-Systeme.

  ADbasic wird benötigt, um Prozesse für *ADwin*-Systeme zu entwickeln, die aus einer Entwicklungsumgebung (z. B. C#, Visual Basic oder Matlab) geladen und gesteuert werden.

– *ADwin-light-16*-pow: externes 12V-Netzteil

  Das Netzteil stellt auf der Sekundärseite 12 Volt bei einer maximalen Dauerbelastung von 2 Ampere zur Verfügung. Es ist für maximale Erweiterung und Auslastung ausgelegt.

– Kabel in verschiedenen Längen zur Spannungsversorgung, für USB und für Ethernet.

  Achten Sie bei USB- und Ethernet-Kabeln auf ausreichende Schirmung, um Störungen auf den Datenleitungen zu vermeiden. Störungen müssen vor dem Gehäuse über die Masse abgeleitet werden (siehe auch Kapitel 3).

– Kabel-Stecker für eine externe Spannungsversorgung

  Jedem *ADwin-light-16*-Gerät ist ein Kabelstecker zur Montage an ein externes Spannungsversorgungs-Kabel beigelegt, falls Sie eine alternative Spannungsquelle nutzen möchten.

13 Software

Sie programmieren *ADwin-light-16* inklusive aller Erweiterungen mit einfachen *ADbasic*-Befehlen. Die Basis-Befehle sind im *ADbasic*-Handbuch beschrieben.

Befehle für den Zugriff auf Ein- und Ausgänge und Schnittstellen befinden sich auf folgenden Seiten:

– Analoge Ein- und Ausgänge: Seite 65
– Digitale Ein- und Ausgänge: Seite 77
– Zähler: Seite 94
– CAN-Schnittstelle: Seite 109
– SSI-Schnittstelle: Seite 124
– PWM-Ausgang: Seite 131
– SPI-Schnittstelle: Seite 141

13.1 Beispiele

13.1.1 Beispiel CAN: Zyklisches Lesen und Senden

Dieses Programm zeigt die Initialisierung des CAN-Controllers im Abschnitt **Init** und das zyklische Auslesen und Senden im Abschnitt **Event**.

```
REM Das Programm initialisiert den CAN-Controller,
REM konfiguriert je ein Message-Objekt als Sender
REM und als Empfänger. Das Programm tauscht alle 10 ms
REM Daten zwischen CAN-Controller und Transputer aus.

#INCLUDE ADWL16.inc
DIM ergebnis AS LONG

Init:
  REM Initialisierung des CAN-Controllers
  Init_CAN()
  REM Baudrate auf 125 kBit/s setzen
  Set_CAN_Baudrate(125000)
  REM Message Objekt 2 zum Lesen konfigurieren
  REM mit 11 Bit und Identifier 385
  En_Receive(2,385,0)
  REM Message Objekt 3 zum Lesen konfigurieren
  REM mit 11 Bit und Identifier 1
  En_Receive(3,1,0)

Event:
  REM 1 Datensatz lesen und 1 Datensatz schreiben
  ergebnis = Read_Msg(2)      'Daten einlesen
  'Wenn es neue Daten gibt, werden sie in das Feld
  'CAN_Msg geschrieben

  CAN_Msg[1]=1                'Sende-Daten
  CAN_Msg[2]=2                'werden in das Feld CAN_Msg
  CAN_Msg[3]=3                'geschrieben. Aus diesem werden
  CAN_Msg[4]=4                'beim späteren Senden die Daten
  CAN_Msg[5]=5                'entnommen.
  CAN_Msg[6]=6
  CAN_Msg[7]=7
  CAN_Msg[8]=8
  CAN_Msg[9]=8                '8 Datenbytes

  Transmit(3)                 'Nachricht in Objekt 3 senden
```

13.1.2 Beispiel CAN: Interruptgesteuertes Lesen

Das nachfolgende Beispiel zeigt die Initialisierung des CAN-Controllers und das nachfolgende interruptgesteuerte Auslesen von neuen Nachrichten:

```
REM Programm initialisiert den CAN-Controller und
REM konfiguriert ein Message Objekt als Empfänger.
REM Das Programm liest interruptgesteuert Nachrichten
REM ein, sobald eine neue eintrifft.

#INCLUDE ADWL16.inc

DIM ergebnis,status,objekt AS LONG

INIT:
  REM Initialisierung des CAN-Controllers
  Init_CAN()

  REM Baudrate auf 125 kBit/s setzen
  Set_CAN_Baudrate(125000)

  REM Message Objekt zum Lesen konfigurieren
  REM mit 11 Bit und Identifier 385
  En_Receive(2,385,0)

  status = Get_CAN_Reg(1)      'Status einlesen
  En_Interrupt(1)              'Bei Eingang einer Nachricht in
                               'Message Objekt 1 wird ein
                               'Interrupt ausgelöst

EVENT:
  objekt = Get_CAN_Reg(5Fh)    'Interruptregister lesen

  IF (objekt = 2) THEN         'Daraus die Nummer des Message
    objekt = 15                'Objekts ermitteln, in dem die
  ELSE                         'neue Nachricht steht
    objekt = objekt - 2
  ENDIF

  ergebnis = Read_Msg(objekt) 'Neue Daten auslesen

  REM Die Daten stehen im Feld CAN_Msg bereit
```

13.2 Analoge Ein- und Ausgänge

Dieser Abschnitt beschreibt folgende Befehle:

DAC

DAC gibt eine definierte Spannung auf einem bestimmten analogen Ausgang aus.

Syntax

```
DAC (channel,value)
```

Parameter

dac_no	Nummer des analogen Ausgangs: 1…2	LONG
value	Wert in Digits, der die auszugebende Spannung definiert: 0…65535	LONG

Beschreibung

Wenn der Digit-Wert `value` außerhalb des zulässigen Wertebereichs liegt, wird er automatisch auf den systemspezifischen Minimal- oder Maximalwert korrigiert.

Siehe auch

ADC

Gültig für

L16

Beispiel

```
Rem Digitaler P-Regler
Dim set_to, gain, diff, out As Long'Deklaration
Init:
  Processdelay = 10000

Event:
  set_to = Par_1              'Sollwert
  gain = Par_2                'Dimensionieren
  diff = set_to - ADC(1)      'Regelabweichung berechnen
  out = diff * gain           'Stellgröße berechnen
  DAC(1, out)                 'Ausgabe der Stellgröße
```

ADC

ADC misst die Spannung an einem analogen Eingang und gibt eine (dem Messergebnis entsprechende) ganze Zahl zurück.

Syntax

```
ret_val = ADC(input_ch)
```

Parameter

input_ch	Nummer (1, 3, …15) des analogen Eingangskanals.	LONG
ret_val	Messergebnis in Digits (0…65535).	LONG

Bemerkungen

ADC ist eine Zusammenstellung aufeinander folgender Funktionen:

- **Set_Mux**: Multiplexer auf den angegebenen Eingangskanal stellen.
- Einschwingen des Multiplexers abwarten.
- **Start_Conv**: Messung starten: Das angelegte Analogsignal in einen Digitalwert konvertieren.
- **Wait_EOC**: Das Ende der Konvertierung abwarten.
- **ReadADC**: Den Digitalwert aus einem Register lesen und zurückgeben.

Multiplexer-Einschwingzeit und Wandlungszeit sind auf Seite 18 angegeben.

Wenn Sie einen nicht vorhandenen Eingangskanal angeben, ist das Messergebnis undefiniert.

Wenn Sie die Zykluszeit des Prozesses (**Processdelay**) auf einen Wert unter 20µs einstellen, beträgt die Ausführungszeit des Befehls nur noch etwa die Hälfte. Dies ist möglich, indem der Compiler für diesen Fall die Wartezeit für das Einschwingen des Multiplexers überspringt. Es wird also angenommen, dass Sie eine Messung ohne Umstellung des Multiplexers beabsichtigen.
Wenn (bei solch kurzen Zykluszeiten) auch die erste Messung korrekt sein soll, müssen Sie eine bestimmte Zeit vor der ersten Benutzung der Anweisung **ADC** mit **Set_Mux** den Multiplexer auf den gewünschten Eingangskanal setzen. Diese Zeit muss mindestens so groß sein wie die Multiplexer-Einschwingzeit.

Messung mit Multiplexer	14,7µs
Kürzere Ausführungszeit bei Zykluszeit kleiner als	20µs
Messung ohne Multiplexer	7,9µs
Einschwingzeit des Multiplexers	6,5µs

In den folgenden Fällen sollte der Befehl **ADC** ersetzt werden durch die Befehle **Set_Mux**, **Start_Conv**, **Wait_EOC** und **ReadADC**:

- Sehr kurze Zykluszeiten: **Processdelay** < 240 (s.o.).
- Hoher Innenwiderstand (>3kΩ) der Spannungsquelle des Messsignals: Dies verlängert die Einschwingzeit des Multiplexers.
- Sie möchten unvermeidliche Wartezeiten für zusätzliche Programmschritte nutzen.

Der Messbereich ist abhängig vom Verstärkungsfaktor:

Verstärkung	Eingangs-Spannungsbereich	Messbereich
1	-10V … 10V	20V
2	-5V … 5V	10V
4	-2,5V … 2,5V	5V
8	-1,25V … 1,25V	2,5V

Mit der folgenden Formel berechnen Sie aus dem zurückgegebenen Digitalwert die gemessene Spannung:

$$\text{Spannung} = (\text{Digits} - 32768_{\text{bipolar}}) \cdot \frac{\text{Messbereich}}{65536}$$

Für den Fall, dass die Verstärkung gleich 1 gewählt wurde (Messbereich von 20 Volt), gelten die in der Tabelle angegebenen Werte:

Messbereich	Rückgabewert von **ADC**			
	0	32768	65535	1 Digit
20V	-10V	0V	+9,999695V	305,175µV

Siehe auch

ReadADC, Set_Mux, Start_Conv, Wait_EOC, L16_Mode

Gültig für

L16

Beispiel

```
Dim iw As Long                'Deklaration

Event:
  Rem Analogen Eingang 1 messen
  iw = ADC(1) * 10900
  Rem Messwert in globale Variable schreiben, damit er
  Rem vom PC gelesen werden kann
  Par_1 = iw
```

L16_Mode

L16_Mode stellt den Betriebsmodus für *ADwin-light-16* Rev. B ein.

Syntax

```
#Include ADWL16.Inc
L16_Mode(mode)
```

Parameter

`mode` Bitmuster zur Einstellung des Betriebsmodus. LONG

Bits in mode	Bedeutung
Bit 0:	Bit = 0: Standardbetrieb (Default). Bit = 1: Schnellbetrieb.
Bits 1…31:	reserviert

Bemerkungen

Im Modus „Standardbetrieb“ arbeitet das Gerät vollständig kompatibel zu der Revision A. Nach dem Einschalten ist das Gerät immer im Modus „Standardbetrieb“.

Im Modus „Schnellbetrieb“ arbeitet der A/D-Wandler mit einer maximalen Abtastrate von 500 kHz.

Siehe auch

ADC, ReadADC, Set_Mux, Start_Conv, Wait_EOC

Gültig für

L16 Rev. B

Beispiel

```
#Include ADWL16.Inc

Init:
  Rem Modus „Schnellbetrieb“ aktivieren
  L16_Mode(1)
```

ReadADC

ReadADC gibt einen gewandelten Wert von einem A/D-Wandler mit 16 Bit Auflösung zurück.

Syntax

```
ret_val = ReadADC(1)
```

Parameter

1	Nummer des zu lesenden Wandlers .	LONG
ret_val	Messwert in Digits (0…65535), der der anliegenden Spannung am Wandler entspricht.	LONG

Bemerkungen

- / -

Siehe auch

ADC, Set_Mux, Start_Conv, Wait_EOC, L16_Mode

Gültig für

L16

Beispiel

```
Event:
  Rem Multiplexer auf Kanal 3 setzen
  Set_Mux(001b)
  Rem MUX-Einschwingzeit überbrücken
  Rem ...
  Start_Conv(1)              'ADC-Wandlung starten
  Wait_EOC(1)                'Ende der Wandlung abwarten
  Par_1 = ReadADC(1)         'Wert einlesen
```

Seq_Init

Seq_Init initialisiert die Ablaufsteuerung.

Eingestellt werden der Arbeitsmodus, der Verstärkungsfaktor, die Kanäle und die Einschwingzeit des Multiplexers.

Syntax

```
#Include ADWL16.Inc
Seq_Init(mode, gain, channel_pattern, mux_time)
```

Parameter

mode — LONG

Arbeitsmodus der Ablaufsteuerung:
0: Normaler Modus (Default), Einzelmessung.
1: Modus „single shot“, einfacher Messzyklus.
2: Modus „continuous“, regelmäßiger Messzyklus.
3: Modus „continuous max“ mit max. Geschwindigkeit.

gain — LONG

Verstärkungsfaktor (nur für die Modi 1 … 3):
0 Faktor = 1, Spannungsbereich -10V…+10V.
1 Faktor = 2, Spannungsbereich -5V…+5V.
2 Faktor = 4, Spannungsbereich -2,5V…+2,5V.
3 Faktor = 8, Spannungsbereich -1,25V…+1,25V.

channel_pattern — LONG

Bitmuster, das die zu wandelnden Kanäle bestimmt.
Bit = 0: Kanal nicht wandeln.
Bit = 1: Kanal wandeln.

Bitnr.	31:15	14	13	12	…	3	2	1	0
Kanalnr.	–	15	–	11	…	–	3	–	1

mux_time — LONG

Anzahl der Zeiteinheiten, aus der sich die Einschwingzeit der Ablaufsteuerung ergibt:
0: Werkseinstellung (200 ≙ 5µs) für die Wartezeit.
200…2^31: Einschwingzeit in Einheiten von 25ns.

Beschreibung

Nach dem Einschalten des Geräts ist Modus 0 aktiv.

Die Modi 1 … 3 aktivieren die Ablaufsteuerung, die an mehreren Kanälen nacheinander eine Wandlung durchführt, je nach Modus einmal oder in zyklischen Abständen. Die Steuerung bezieht sich immer nur auf die mit **channel_pattern** definierte Auswahl an Kanälen.

Die Modi unterscheiden sich wie folgt:

Modus		Art der Messung
0	Normal:	Standard: Einzelne Messung an einem Kanal, siehe **ADC**.
1	single shot:	Die Ablaufsteuerung wird mit **Start_Conv** gestartet; die Ablaufsteuerung endet, sobald die gewählten Kanäle je einmal gewandelt sind. Das Ende der Ablaufsteuerung wird mit **Wait_EOC** abgefragt und die Messwerte mit **Seq_Read** eingelesen.
2	continuous:	Die Ablaufsteuerung wandelt für jeden Prozesszyklus einen Satz neuer Messwerte. Die Wandlung wird im Abschnitt **Init:** gestartet, mit **Start_Conv** als letztem Befehl. Das Wandlungsende (für alle Kanäle) wird automatisch mit dem Beginn des Prozesszyklus synchonisiert. Daher können – und sollten auch – alle Messwerte zu Beginn des Prozesszyklus mit **Seq_Read** gelesen werden.

3	continuous max:	Die Ablaufsteuerung wandelt ununterbrochen und mit maximaler Wandlungsrate neue Messwerte an den gewählten Kanälen, d.h. Wandlung und Prozesszyklus laufen asynchron. Die Wandlung wird mit **Start_Conv** im Abschnitt **Init:** gestartet. Im Prozesszyklus wird mit **SEQ_Read** der jeweils neueste Messwert gelesen.

Beachten Sie beim Modus 2 (continuous): Die Synchronisierung geschieht nur einmal und gilt nur für die gerade eingestellte Zykluszeit (**Processdelay**). Wenn sich das Zeitverhalten ändert, z.B. durch Ändern der Zykluszeit, geht die Synchronisation verloren. Als Folge werden entweder Messwerte zu früh und damit mehrfach gelesen, oder es gehen Messwerte verloren, weil sie beim Auslesen bereits von neueren Werten überschrieben sind.

Die Einschwingzeit (Parameter mux_time) legt den Zeitabstand zwischen 2 Wandlungen der Ablaufsteuerung fest. Wir empfehlen, den angegebenen Wertebereich nicht zu unterschreiten, weil eine zu kurze Einschwingzeit zu ungenaueren bis falschen Messwerten führt.

Wenn der Innenwiderstand der Spannungsquelle des Messsignals zu groß ist, ist die Werkseinstellung des Multiplexers zu kurz für eine genaue Messung. Stellen Sie dann mit dem Parameter mux_time eine längere Einschwingzeit für den Multiplexer ein.

Siehe auch

ADC, Seq_Read, Start_Conv, Wait_EOC

Gültig für

L16 Rev. B

Beispiel

```
#Include ADWL16.Inc

Dim Data_1[8] As Long At DM_Local
Dim i As Long

Init:
  Rem Ablaufsteuerung: Continuous Mode, Verstärkung 2
  Rem Kanäle 1, 3, ..., 15, Standard-Einschwingzeit
  Seq_Init(3,1,5555h,0)
  Start_Conv(1)             'Messzyklus starten

Event:
  Rem Die Wandlung ist für alle gewählten Kanäle gerade
  Rem beendet, die Messwerte werden abgeholt.
  FOR i = 1 TO 8
    Data_1[i] = SEQ_Read(i*2-1) 'Messwerte holen
  NEXT i
  Rem Messwerte verarbeiten
```

Seq_Read

Seq_Read gibt den zuletzt gespeicherten Messwert des angegebenen Kanals zurück.

Syntax

```
#Include ADWL16.Inc
ret_val = Seq_Read(channel)
```

Parameter

channel	Kanalnummer (1, 3, …, 15).	LONG
ret_val	Messwert (0…65535) des angegebenen Kanals.	LONG

Beschreibung

Der Rückgabewert ist nur sinnvoll, wenn vorher mit **Seq_Init** die Ablaufsteuerung aktiviert und dabei der angegebene Kanal mit ausgewählt wurde.

Im Modus „single shot“ muss zuerst das Ende der Ablaufsteuerung mit **Wait_EOC** abgefragt werden, bevor die Messwerte gelesen werden.

Siehe auch

Seq_Init, Start_Conv, Wait_EOC

Gültig für

L16 Rev. B

Beispiel

```
#Include ADWL16.Inc

Dim Data_1[400] As Long At DM_Local

Init:
  Rem Ablaufsteuerung: Single shot, Verstärkung 1
  Rem Kanäle 5, 7, 13, 15, Standard-Einschwingzeit
  Seq_Init(1,0,1010000010010000b,0)
  Start_Conv(1)             'Messzyklus starten

Event:
  Wait_EOC(1)               'Wandlungsende abwarten
  Rem Kanäle 5, 7, 13, 15 lesen
  Data_1[1] = Seq_Read(5)
  Data_1[2] = Seq_Read(7)
  Data_1[3] = Seq_Read(13)
  Data_1[4] = Seq_Read(15)
  Start_Conv(1)             'nächsten Messzyklus starten
```

Set_Mux

Set_Mux setzt den Multiplexer auf den gewählten Messkanal.

Syntax

```
Set_Mux(pattern)
```

Parameter

pattern — Bitmuster zur Zuordnung von Messkanälen: LONG

Bitnr.	31…3	2	1	0
	–	MUX		

MUX — Die Bits 0…2 bestimmen den Kanal, auf den der Multiplexer eingestellt wird:
000: Kanal 1
001: Kanal 3
010: Kanal 5
011: Kanal 7
100: Kanal 9
101: Kanal 11
110: Kanal 13
111: Kanal 15

Beschreibung

Die Umstellung des Multiplexers auf einen anderen Kanal benötigt eine definierte Einschwingzeit. Erst danach darf die Wandlung der anliegenden Spannung gestartet werden.

Die Einschwingzeit des Multiplexers beträgt 6,5µs beim maximalen Spannungssprung von 20V.

Wandlungszeit des 16 Bit-ADC: 10µs; optional ab Rev. B: 2µs.

Wir empfehlen für die Angabe des Bitmusters die binäre Schreibweise (Suffix „b“). Sie können damit die erforderlichen Bitmuster besser darstellen als mit der (gleichfalls zulässigen) dezimalen oder hexadezimalen Schreibweise.

Siehe auch

ADC, ReadADC, Start_Conv, Wait_EOC, L16_Mode

Gültig für

L16

Beispiel

```
Dim val1 As Long

Event:
  Set_Mux(0)                'Multiplexer auf Kanal 1 setzen
  Rem Überbrücken Sie hier die Einschwingzeit des
  Rem Multiplexers mit Befehlszeilen
  Start_Conv(1)             'Start ADC1 A/D-Wandlung
  Wait_EOC(1)               'Ende der Wandlung abwarten
  val1 = ReadADC(1)         'Wert auslesen
```

Start_Conv

Start_Conv kann die Wandlung am A/D-Wandler und an allen D/A-Wandler starten.

Syntax

```
Start_Conv(adc_pattern)
```

Parameter

pattermn	Bitmuster, das die zu startenden Wandler festlegt (nur Bits 0 und 2 verwendbar): 1: Wandler starten. 0: Wandler nicht starten.	CONST LONG

Bit-Nr.	31…3	2	1	0
ADC1, 16 Bit	–	–	–	x
alle DAC	–	x	–	–

Beschreibung

Sie können als Parameter nur Konstanten einsetzen, keine Variablen.

Wir empfehlen für die Angabe des Bitmusters die binäre Schreibweise (Suffix „b“). Sie können damit die erforderlichen Bitmuster besser darstellen als mit der (gleichfalls zulässigen) dezimalen oder hexadezimalen Schreibweise.

Siehe auch

ADC, ReadADC, Set_Mux, Wait_EOC, L16_Mode

Gültig für

L16

Beispiel

```
Dim val1 As Long

Event:
  Set_Mux(0)                'Multiplexer auf Kanal 1 setzen
  Rem Überbrücken Sie hier die Einschwingzeit des
  Rem Multiplexers mit Befehlszeilen
  Start_Conv(1)             'Start ADC1 A/D-Wandlung
  Wait_EOC(1)               'Ende der Wandlung abwarten
  val1 = ReadADC(1)         'Wert auslesen
```

Die Multiplexer-Einschwingzeit ist auf Seite 18 angegeben.

Wait_EOC

Wait_EOC wartet auf das Ende der A/D-Wandlung.

Syntax

```
Wait_EOC(1)
```

Parameter

Als Übergabeparameter ist nur die Konstante 1 zulässig. Der Übergabeparameter wird als Bitmuster zur Auswahl des Wandlers interpretiert.

CONST
LONG

Beschreibung

Setzen Sie immer nur die Bits vorhandener ADC. Anderenfalls führt dies in einem hochprioren Prozess zur Unterbrechung der Kommunikation zwischen *ADwin*-System und PC.

Siehe auch

ADC, ReadADC, Set_Mux, Start_Conv, L16_Mode

Gültig für

L16

Beispiel

```
Dim val1 As Long

Event:
  Set_Mux(0)                'Multiplexer auf Kanal 1 setzen
  Rem Überbrücken Sie hier die Einschwingzeit des
  Rem Multiplexers mit Befehlszeilen
  Start_Conv(1)             'Start ADC1 A/D-Wandlung
  Wait_EOC(1)               'Ende der Wandlung abwarten
  val1 = ReadADC(1)         'Wert auslesen
```

Die Multiplexer-Einschwingzeit ist auf Seite 18 angegeben.

13.3 Digitale Ein- und Ausgänge

Dieser Abschnitt beschreibt folgende Befehle:

Clear_Digout

Clear_Digout setzt einen der digitalen Ausgänge auf 0 (TTL-Pegel low).

Syntax

```
Clear_Digout(bit_no)
```

Parameter

bit_no — Nummer (0…5) des Bits, das den Ausgang festlegt (siehe Tabelle). CONST, LONG

bit_no	0	1	…	5
Ausgang	0	1	…	5

Beschreibung

Clear_Digout akzeptiert als Parameter nur eine Konstante. Wenn Sie den Ausgang durch eine Variable festlegen wollen, verwenden Sie den Befehl **Digout_Word**.

Siehe auch

Digout_Word, Set_Digout

Gültig für

L16, L16-CO1, L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
Dim val As Long                 'Deklaration

Init:
  Set_Digout(0)                 'Dig. Ausgang 0 auf Pegel high
                                'setzen

Event:
  val = ADC(1)                  'Messwerterfassung
  If (val > 3000) Then
    Clear_Digout(0)             'Dig. Ausgang 0 auf Pegel low
                                'zurücksetzen
  EndIf
```

Digin

Digin gibt den Wert eines der digitalen Eingänge 0…5 zurück.

Syntax

```
ret_val = Digin(channel_no)
```

Parameter

channel_no	Nummer, die den abzufragenden Eingang festlegt (siehe Tabelle unten).	CONST LONG
ret_val	1: TTL-Pegel high liegt an 0: TTL-Pegel low liegt an	LONG

channel_no	0	…	5
Eingang Nr.	0	…	5

Bemerkungen

Digin akzeptiert als Parameter nur eine Konstante.

Diese Anweisung ist für das Auslesen weniger Bits geeignet. Wenn mehrere Bits (z.B. auch in Schleifen) auszulesen sind, ist die Verwendung der Anweisung **Digin_Word** deutlich schneller. Beachten Sie dies insbesondere bei zeitkritischen Anwendungen.

Siehe auch

Digin_Word, Digout_Word

Gültig für

L16

Beispiel

```
Dim Data_1[10000] As Long As Fifo

Event:
  Rem Ist der digitale Eingang 0 gesetzt?
  If (Digin(0) = 1) Then
    Data_1 = ADC(1)            'Messwerterfassung
  EndIf
```

Digin_Word

Digin_Word gibt die Werte aller digitalen Eingänge auf einmal zurück.

Syntax

```
ret_val = Digin_Word()
```

Parameter

ret_val	Bitmuster, das den TTL-Pegeln an den digitalen Eingängen entspricht (Zuordnung s.u.). 1: TTL-Pegel high liegt an 0: TTL-Pegel low liegt an	LONG

Bitnummer in ret_val	31 … 6	5	…	0
Eingang Nr.	–	5	…	0

Bemerkungen

- / -

Siehe auch

Digin, Digout_Word

Gültig für

L16

Beispiel

```
Dim Data_1[10000] As Long As Fifo

Event:
  Rem Abfrage, ob die Eingänge 0 und 1 gesetzt sind
  If ((Digin_Word() And 11b) = 11b) Then
    Data_1 = ADC(1)             'Messwerterfassung
  EndIf
```

Digout_Word

Digout_Word setzt die digitalen Ausgänge gleichzeitig auf definierte TTL-Pegel.

Syntax

```
Digout_Word(pattern)
```

Parameter

pattern — Bitmuster, das den TTL-Pegeln an den digitalen Ausgängen entspricht (s. Tabelle). LONG
1: Setzen auf TTL-Pegel high
0: Setzen auf TTL-Pegel low

Bit-Nr. in pattern	31 … 6	5	…	0
Ausgang Nr.	–	5	…	0

Bemerkungen

- / -

Siehe auch

Clear_Digout, Digin_Word, Set_Digout

Gültig für

L16

Beispiel

```
Dim value As Long

Event:
  value = ADC(1)                'Messwerterfassung
  If (value > 3000) Then'Grenzwert überschritten?
    Digout_Word(101b)           'Ausgänge 0 und 2 setzen, alle
                                'anderen
                                'Ausgänge werden gelöscht!
  EndIf
```

Set_Digout

Set_Digout setzt einen der digitalen Ausgänge auf 1 (TTL-Pegel high).

Syntax

```
Set_Digout(bit_no)
```

Parameter

bit_no — Nummer (0…5) des Bits, das den Ausgang festlegt (siehe Tabelle). CONST, LONG

bit_no	0	1	…	5
Ausgang	0	1	…	5

Beschreibung

Set_Digout ist für das Setzen weniger Bits geeignet. Wenn mehrere Bits (z.B. auch in Schleifen) zu setzen sind, ist die Verwendung der Anweisung **Digout_Word** deutlich schneller. Beachten Sie dies insbesondere bei zeitkritischen Anwendungen.

Wenn Sie den zu setzenden Ausgang durch den Wert einer Variablen bestimmen wollen, verwenden Sie den Befehl **Digout_Word**.

Siehe auch

Clear_Digout, Digout_Word

Gültig für

L16

Beispiel

```
Dim val As Long

Event:
  val = ADC(1)                  'Messwerterfassung
  If (val > 3000) Then
    Set_Digout(0)               'Dig. Ausgang 0 auf TTL-Pegel
                                'high setzen
  EndIf
```

Conf_DIO_E

Conf_DIO_E konfiguriert die 32 digitalen Kanäle in Gruppen zu je 8 als Ein- oder Ausgänge.

Syntax

```
#Include ADWL16.Inc
Conf_DIO_E(pattern)
```

Parameter

pattern	Bitmuster, das die digitalen Kanäle als Ein- oder Ausgang konfiguriert: Bit=0: Kanäle als Eingänge. Bit=1: Kanäle als Ausgänge.	LONG

Bitnr. in pattern	15…4	3	2	1	0
Kanäle	–	DIO31 … DIO24	DIO23 … DIO16	DIO15 … DIO08	DIO07 … DIO00

Bemerkungen

Die digitalen Kanäle sind nach dem Einschalten zunächst als Eingänge konfiguriert (und können also auch noch nicht als Ausgänge angesprochen werden). Sie können nur in Gruppen zu je 8 als Ein- oder Ausgänge konfiguriert werden.

Wir empfehlen für die Angabe des Bitmusters die binäre Schreibweise (Suffix „**b**“). Sie können damit die erforderlichen Bitmuster besser darstellen als mit der (gleichfalls zulässigen) dezimalen oder hexadezimalen Schreibweise.

Siehe auch

Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(1100b)          'Konfiguriert DIOs 15:00 als
                             'Eingänge
                             'und DIOs 31:16 als Ausgänge
```

Digin_Word1_E

Digin_Word1_E gibt die Werte der digitalen Eingänge 0…15 auf einmal zurück.

Syntax

```
#Include ADWL16.Inc
ret_val = Digin_Word1_E()
```

Parameter

ret_val	Bitmuster, das den TTL-Pegeln an den digitalen Eingängen entspricht (Zuordnung s.u.). 1: TTL-Pegel high liegt an 0: TTL-Pegel low liegt an	LONG

Bitnummer in ret_val	31 … 16	15	14	…	1	0
Eingang Nr.	–	DIO15	DIO14	…	DIO01	DIO00

Bemerkungen

Wenn Sie Kanäle als Ausgang konfiguriert haben, so wird der Inhalt des Ausgangsregisters an diesen Bits zurückgelesen.

Siehe auch

Conf_DIO_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(1100b)             'Konfiguriert DIOs 15:00 als
                                'Eingänge
                                'und DIOs 31:16 als Ausgänge

Event:
  Par_1 = Digin_Word1_E()'Unteres Wort (Bits 15:00) einlesen
```

Digin_Word2_E

Digin_Word2_E gibt die Werte der digitalen Eingänge 16…31 auf einmal zurück.

Syntax

```
#Include ADWL16.Inc
ret_val = Digin_Word2_E()
```

Parameter

ret_val	Bitmuster, das den TTL-Pegeln an den digitalen Eingängen entspricht (Zuordnung s.u.). 1: TTL-Pegel high liegt an 0: TTL-Pegel low liegt an	LONG

Bitnummer in ret_val	31 … 16	15	14	…	1	0
Eingang Nr.	–	DIO31	DIO30	…	DIO17	DIO16

Bemerkungen

Wenn Sie Kanäle als Ausgang konfiguriert haben, so wird der Inhalt des Ausgangsregisters an diesen Bits zurückgelesen.

Siehe auch

Conf_DIO_E, Digin_Word1_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(0)                 'Konfiguriert DIOs 31:00 als
                                'Eingänge
                                '(auch power-up-Zustand!)

Event:
  Par_1 = Digin_Word1_E()'Unteres Wort (Bits 15:00) einlesen
  Par_2 = Digin_Word2_E()'Oberes Wort (Bits 31:16) einlesen
```

Digin_Long_E

Digin_Long_E gibt die Werte der digitalen Eingänge 0…31 auf einmal zurück.

Syntax

```
#Include ADWL16.Inc
ret_val = Digin_Long_E()
```

Parameter

ret_val	Bitmuster, das den TTL-Pegeln an den digitalen Eingängen entspricht (Zuordnung s.u.). 1: TTL-Pegel high liegt an 0: TTL-Pegel low liegt an	LONG

Bitnummer in ret_val	31	30	…	1	0
Eingang Nr.	DIO31	DIO30	…	DIO01	DIO00

Bemerkungen

Wenn Sie Kanäle als Ausgang konfiguriert haben, so wird der Inhalt des Ausgangsregisters an diesen Bits zurückgelesen.

Siehe auch

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E, Digout_Long_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(0)                 'Konfiguriert DIOs 31:00 als
                                'Eingänge
                                '(auch power-up-Zustand!)

Event:
  Par_1 = Digin_Long_E()'Bits 31:00 einlesen
```

Digout_Reset1_E

Digout_Reset1_E setzt bestimmte der digitalen Ausgänge 0…15 auf TTL-Pegel low.

Syntax

```
#Include ADWL16.Inc
Digout_Reset1_E(clear)
```

Parameter

clear	Bitmuster, um bestimmte Ausgänge zu setzen: Bit = 1: Setzen auf TTL-Pegel low Bit = 0: Kein Einfluss	LONG

Bitnummer in clear	31 … 16	15	14	…	1	0
Ausgang Nr.	–	DIO15	DIO14	…	DIO01	DIO00

Bemerkungen

- / -

Siehe auch

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(11b)               'Konfiguriert DIOs 15:00 als
                                'Ausgänge
                                'und DIOs 31:16 als Eingänge

Init:
  Par_1 = 5555h                 'Alle ungeraden Bits des unteren
                                'Worts
                                'bei der Ausgabe löschen
  Digout_Word1_E(0FFFFh)'DIO-Bits 15:00 ausgeben

Event:
  Digout_Reset1_E(Par_1)'DIO-Bits entsprechend Par_1 löschen
  Par_1 = Par_1 XOr 0FFFFh'Output-Word invertieren
  Digout_Word1_E(Par_1)'DIO-Bits 15:00 ausgeben
```

Digout_Reset2_E

Digout_Reset2_E setzt bestimmte der digitalen Ausgänge 16…31 auf TTL-Pegel low.

Syntax

```
#Include ADWL16.Inc
Digout_Reset2_E(clear)
```

Parameter

clear	Bitmuster, um bestimmte Ausgänge zu setzen: Bit = 1: Setzen auf TTL-Pegel low Bit = 0: Kein Einfluss	LONG

Bitnummer in clear	31 … 16	15	14	…	1	0
Ausgang Nr.	–	DIO31	DIO30	…	DIO17	DIO16

Bemerkungen

- / -

Siehe auch

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(1100b)             'Konfiguriert DIOs 15:00 als
                                'Eingänge
                                'und DIOs 31:16 als Ausgänge

Init:
  Par_2 = 5555h                 'Alle ungeraden Bits des
                                'High-Words
                                'bei der Ausgabe löschen
  Digout_Word2_E(0FFFFh)'DIO-Bits 31:16 ausgeben

Event:
  Digout_Reset2_E(Par_2)'DIO-Bits entsprechend Par_2 löschen
  Par_2 = Par_2 XOr 0FFFFh'Output-Word invertieren
  Digout_Word2_E(Par_2)         'DIO-Bits 31:16 ausgeben
```

Digout_Set1_E

Digout_Set1_E setzt bestimmte der digitalen Ausgänge 0…15 auf TTL-Pegel high.

Syntax

```
#Include ADWL16.Inc
Digout_Set1_E(set)
```

Parameter

set	Bitmuster, um bestimmte Ausgänge zu setzen: Bit = 1: Setzen auf TTL-Pegel high Bit = 0: Kein Einfluss	LONG

Bitnummer in set	31 … 16	15	14	…	1	0
Ausgang Nr.	–	DIO15	DIO14	…	DIO01	DIO00

Bemerkungen

- / -

Siehe auch

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(1100b)           'Konfiguriert DIOs 15:00 als
                              'Ausgänge
                              'und DIOs 31:16 als Eingänge
  Par_1 = 0AAAAh              'Alle geraden Bits des unteren
                              'Worts
                              'bei der Ausgabe setzen
  Digout_Word1_E(0)           'DIO-Bits 15:00 ausgeben

Event:
  Digout_Set1_E(Par_1)        'DIO-Bits entsprechend Par_1
                              'setzen
  Par_1 = Par_1 XOr 0FFFFh'Output-Word invertieren
  Digout_Word1_E(Par_1)'DIO-Bits 15:00 ausgeben
```

Digout_Set2_E

Digout_Set2_E setzt bestimmte der digitalen Ausgänge 16…31 auf TTL-Pegel high.

Syntax

```
#Include ADWL16.Inc
Digout_Set2_E(set)
```

Parameter

set	Bitmuster, um bestimmte Ausgänge zu setzen: Bit = 1: Setzen auf TTL-Pegel high Bit = 0: Kein Einfluss	LONG

Bitnummer in set	31 … 16	15	14	…	1	0
Ausgang Nr.	–	DIO31	DIO30	…	DIO17	DIO16

Bemerkungen

- / -

Siehe auch

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Word1_E, Digout_Word2_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(0011b)           'Konfiguriert DIOs 15:00 als
                              'Ausgänge
                              'und DIOs 31:16 als Eingänge
  Par_1 = 0AAAAh              'Alle geraden Bits des unteren
                              'Worts
                              'bei der Ausgabe setzen
  Digout_Word2_E(0)           'DIO-Bits 31:16 ausgeben

Event:
  Digout_Set2_E(Par_2)        'DIO-Bits entsprechend Par_1
                              'setzen
  Par_2 = Par_2 XOr 0FFFFh'Output-Word invertieren
  Digout_Word2_E(Par_2)       'DIO-Bits 31:16 ausgeben
```

Digout_Word1_E

Digout_Word1_E setzt mit einem Bitmuster gleichzeitig die digitalen Ausgänge 0…15 auf definierte TTL-Pegel.

Syntax

```
#Include ADWL16.Inc
Digout_Word1_E(pattern)
```

Parameter

pattern	Bitmuster, das den TTL-Pegeln an den digitalen Ausgängen entspricht (Zuordnung s.u.). Bit = 1: Setzen auf TTL-Pegel high Bit = 0: Setzen auf TTL-Pegel low	LONG

Bitnummer in pattern	31 … 16	15	14	…	1	0
Ausgang Nr.	–	DIO15	DIO14	…	DIO01	DIO00

Bemerkungen

- / -

Siehe auch

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word2_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(0011b)          'Konfiguriert DIOs15:00 als
                             'Ausgänge
                             'und DIOs 31:16 als Eingänge
  Par_1 = 5555h              'Alle ungeraden Bits des unteren
                             'Worts
                             'setzen

Event:
  Digout_Word1_E(Par_1)      'DIO-Bits 15:00 ausgeben
```

Digout_Word2_E

Digout_Word2_E setzt mit einem Bitmuster gleichzeitig die digitalen Ausgänge 16…31 auf definierte TTL-Pegel.

Syntax

```
#Include ADWL16.Inc
Digout_Word2_E(pattern)
```

Parameter

pattern	Bitmuster, das den TTL-Pegeln an den digitalen Ausgängen entspricht (Zuordnung s.u.). Bit = 1: Setzen auf TTL-Pegel high Bit = 0: Setzen auf TTL-Pegel low	LONG

Bitnummer in pattern	31 … 16	15	14	…	1	0
Ausgang Nr.	–	DIO31	DIO30	…	DIO17	DIO16

Bemerkungen

- / -

Siehe auch

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E

Gültig für

L16-DIO1, L16-DIO2

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(12)              'Konfiguriert DIOs 15:00 als
                              'Eingänge
                              'und DIOs 31:16 als Ausgänge
  Par_2 = 0AAAAh              'Alle geraden Bits des unteren
                              'Worts
                              'setzen

Event:
  Digout_Word2_E(Par_2)       'DIO-Bits 31:16 ausgeben
```

Digout_Long_E

Digout_Long_E setzt mit einem Bitmuster gleichzeitig die digitalen Ausgänge 0…31 auf definierte TTL-Pegel.

Syntax

```
#Include ADWL16.Inc
Digout_Long_E(pattern)
```

Parameter

pattern	Bitmuster, das den TTL-Pegeln an den digitalen Ausgängen entspricht: Bit = 1: Setzen auf TTL-Pegel high Bit = 0: Setzen auf TTL-Pegel low	LONG

Bitnummer in pattern	31	30	…	1	0
Ausgang Nr.	DIO31	DIO30	…	DIO01	DIO00

Bemerkungen

- / -

Siehe auch

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digin_Long_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Gültig für

L16-DIO1, L16-DIO2, L16-DIO3

Beispiel

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(1111b)           'Konfiguriert DIOs 31:00 als
                              'Ausgänge
  Par_2 = 0AAAAAAAAh          'Alle geraden Bits setzen

Event:
  Digout_Long_E(Par_2)        'DIO-Bits 31:00 ausgeben
```

13.4 Zähler

Dieser Abschnitt beschreibt folgende Befehle:

Cnt_Clear

Cnt_Clear setzt einen oder mehrere Zähler auf Null, gemäß dem Bitmuster in pattern.

Syntax

```
#Include ADWL16.Inc
Cnt_Clear(pattern)
```

Parameter

pattern	Bitmuster Bit = 0: Kein Einfluss Bit = 1: Zähler auf Null setzen	LONG

Bit-Nr.	31…2	1	0
Zähler-Nr.	–	2[a]	1

a. nicht verfügbar bei L16-CO1

Bemerkungen

Nach Ausführung von **Cnt_Clear** wird das Bitmuster automatisch auf 0 (Null) zurückgesetzt, d.h. die zurückgesetzten Zähler beginnen zu zählen.

Siehe auch

Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16, L16-CO1, L16-DIO1, L16-DIO2

Beispiel

```
#INCLUDE ADWL16.Inc

Dim old_1, new_1 AS LONG'Variablen…
Dim old_2, new_2 AS LONG'Dimensionieren

INIT:
  old_1 = 0                    'Variablen…
  old_2 = 0                    'initialisieren
  Cnt_Mode(0)                  'Alle Zähler auf externen
Takteingang
  Cnt_Set(11b)                 'Zähler 1+2 mit Takt(CLK)- und
                               'Richtungs(DIR)-Eingang
  Cnt_InputMode(0)             'Funktionalität CLR/LATCH
festlegen:
                               'Alle als CLR-Eingang
  Cnt_ClearEnable(11b)         'CLR-Funktion beider Zähler
                               'freischalten
  Cnt_Clear(11b)               'Zähler 1+2 auf 0 zurücksetzen
  Cnt_Enable(11b)              'Zähler 1+2 starten

EVENT:
  Cnt_Latch(11b)               'Zähler 1+2 gleichzeitig latchen
  new_1 = Cnt_ReadLatch(1)'Latch A Zähler 1 und…
  new_2 = Cnt_ReadLatch(2)'Latch A Zähler 2 auslesen.
  PAR_1 = new_1 - old_1'Differenz bilden (f = Impulse / Zeit)
  PAR_2 = new_2 - old_2'-"-
  old_1 = new_1                'Neuen Zählerstand als alten
speichern
  old_2 = new_2                '-"-
```

Cnt_ClearEnable

Cnt_ClearEnable sperrt den CLR-Eingang eines oder mehrerer Zähler oder gibt ihn frei, gemäß dem Bitmuster in pattern.

Syntax

```
#Include ADWL16.Inc
Cnt_ClearEnable(pattern)
```

Parameter

pattern	Bitmuster Bit = 0: CLR-Eingang am Zähler sperren Bit = 1: CLR-Eingang am Zähler freigeben	LONG

Bit-Nr.	31…2	1	0
Zähler-Nr.	–	2[a]	1

a. nicht verfügbar bei L16-CO1

Bemerkungen

Der Befehl beeinflusst alle Zähler gleichzeitig. Er ist nur sinnvoll einsetzbar, wenn mit **Cnt_InputMode** der CLR-Modus eingestellt ist.

Verwenden Sie diesen Befehl möglichst nur bei gesperrtem Zähler.

Siehe auch

Cnt_Clear, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16-DIO1, L16-DIO2

Beispiel

siehe Cnt_Clear

Cnt_Enable

Cnt_Enable hält die mittels pattern gewählten Zähler an oder gibt sie frei, um eingehende Impulse zu zählen.

Syntax

```
#Include ADWL16.Inc
Cnt_Enable(pattern)
```

Parameter

pattern	Bitmuster Bit = 0: Zähler anhalten Bit = 1: Zähler freigeben	LONG

Bit-Nr.	31…2	1	0
Zähler-Nr.	_	2[a]	1

a. nicht verfügbar bei L16-CO1

Siehe auch

Cnt_Clear, Cnt_ClearEnable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16, L16-CO1, L16-DIO1, L16-DIO2

Beispiel

siehe Cnt_Clear

Cnt_GetStatus

Cnt_GetStatus gibt den Inhalt des Zähler-Statusregisters zurück.

Syntax

```
#Include ADWL16.Inc
ret_val = Cnt_GetStatus()
```

Parameter

ret_val	Inhalt des Statusregisters: Hinweise auf mögliche Fehlerquellen; Bedeutung der Bits siehe Tabelle.	LONG

Bit Nr.	31… 28	27	26	25	24	23… 20	19	18	17	16	15… 06	05	04	03… 02	01	00
Signal	-	L2	C2	L1	C1	-	B2	A2	B1	A1	-	N2	N1	-	R2	R1

- :don't care (Signalzustände sind nicht definiert (mit **0F0F0033h** ausmaskieren)
Ax:Signal A (statisch)
Bx: Signal B (statisch)
Cx:Korrelationsfehler* (Signal A und B sind identisch, d.h. nicht um ca. 90° phasenverschoben)
Lx: Leitungsfehler* (Kabel abgezogen oder Leitung unterbrochen)
Nx:CLR-/LATCH-Eingang (statisch)
Rx:Reset-Enable (Wert, der durch **Cnt_ClearEnable** gesetzt wurde)
x:Zählernummer (1…2)

* Auto-Reset (wird beim Auslesen zurückgesetzt)

Bemerkungen

- / -

Siehe auch

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16-DIO1, L16-DIO2

Beispiel

```
#INCLUDE ADWL16.Inc
Dim error AS LONG

INIT:
  Cnt_Mode(0)                   'Alle Zähler auf externen
Takteingang
  Cnt_Set(0)                    'Alle Zähler mit A/B-Eingang
(z.B.
                                'für Inkremental-Encoder)
  Cnt_InputMode(0)              'Funktionalität CLR/LATCH
festlegen:
                                'Alle als CLR-Eingang
  Cnt_ClearEnable(11b)          'Schaltet die CLR-Funktion
beider
                                'Zähler frei
  Cnt_Clear(11b)                'Zähler 1+2 auf 0 zurücksetzen
  Cnt_Enable(1)                 'Zähler 1 starten
  error = 0                     'Fehlerindikator zurücksetzen

EVENT:
  PAR_1 = Cnt_Read(1)           'Zähler 1 auslesen
  PAR_2 = Cnt_GetStatus() AND 0F0F0033h'Statusregister
                                'Zähler auslesen
  If (PAR_2 AND 2000000h = 2000000h) THEN'Leitungs- bzw.
                                'Kabelfehler Zähler 1?
    Inc PAR_3                   'Anzahl Leitungs- bzw.
Kabelfehler bis
                                'jetzt…
    error = 1                   'Fehlerindikator setzen
  ENDIF
  If (PAR_2 AND 1000000h = 1000000h) THEN'Korrelationsfehler
                                'Zähler 1?
    Inc PAR_4                   'Anzahl Korrelationsfehler bis
jetzt…
    error = 1                   'Fehlerindikator setzen
  ENDIF
  PAR_5 = Shift_Right(PAR_2 AND 10h,4)
                                'akt. Zustand CLR-Eingang
  PAR_6 = Shift_Right(PAR_2 AND 10000h,16)
                                'akt. Zustand A-Eingang
  PAR_7 = Shift_Right(PAR_2 AND 20000h,17)
                                'akt. Zustand B-Eingang
```

Cnt_InputMode

Cnt_InputMode stellt die Funktion des CLR/LATCH-Eingangs eines oder mehrerer Zähler ein.

Syntax

```
#Include ADWL16.Inc
Cnt_InputMode(pattern)
```

Parameter

pattern	Bitmuster Bit = 0: CLR-Modus einstellen Bit = 1: LATCH-Modus einstellen	LONG

Bit-Nr.	31…2	1	0
Zähler-Nr.	–	2	1

Bemerkungen

Verwenden Sie diesen Befehl möglichst nur bei gesperrtem Zähler.

Siehe auch

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16-DIO1, L16-DIO2

Beispiel

siehe Cnt_Clear

Cnt_Latch

Cnt_Latch überträgt den aktuellen Zählerstand eines oder mehrerer Zähler in das zugehörige Latch A, je nach Bitmuster in pattern.

Syntax

```
#Include ADWL16.Inc
Cnt_Latch(pattern)
```

Parameter

pattern — Bitmuster — LONG

Bit = 0: keine Funktion
Bit = 1: Zählerstand in Latch A übertragen

Bit-Nr.	31…2	1	0
Zähler-Nr.	–	2[a]	1

a. nicht verfügbar bei L16-CO1

Bemerkungen

Nach Ausführung des Befehls wird das Bitmuster automatisch auf 0 (Null) zurückgesetzt.

Das Latch A wird mit **Cnt_ReadLatch** in eine Variable ausgelesen.

Siehe auch

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16, L16-CO1, L16-DIO1, L16-DIO2

Beispiel

siehe Cnt_Clear

Cnt_Mode

Cnt_Mode definiert die Betriebsart aller Zähler, d.h. welchen Takteingang diese nutzen, gemäß dem Bitmuster in pattern.

Syntax

```
#Include ADWL16.Inc
Cnt_Mode(pattern)
```

Parameter

pattern	Bitmuster Bit = 0: externer Takteingang für Ereigniszählung (CLK/DIR oder A/B) Bit = 1: interner Takteingang für Pulsbreitenmessung (5MHz oder 20MHz)	LONG

Bit-Nr.	31…2	1	0
Zähler-Nr.	–	2	1

Bemerkungen

Cnt_Set legt den Modus des gewählten Takteingangs fest.

Verwenden Sie **Cnt_Mode** möglichst nur bei gesperrtem Zähler.

Siehe auch

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16-DIO1, L16-DIO2

Beispiel

siehe Cnt_Clear

Cnt_Read

Cnt_Read überträgt einen aktuellen Zählerstand in das zugehörige Latch A und gibt ihn als Rückgabewert zurück.

Syntax

```
#Include ADWL16.Inc
ret_val = Cnt_Read(counter_no)
```

Parameter

channel	Zählernummer: 1…2; L16-CO1: 1.	LONG
ret_val	Zählerstand	LONG

Bemerkungen

Verwenden Sie den Rückgabewert in Berechnungen (z.B. Differenzen oder Zählrichtung) nur mit Variablen vom Typ Long.

Siehe auch

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16, L16-CO1, L16-DIO1, L16-DIO2

Beispiel

```
#INCLUDE ADWL16.Inc

Dim old, new As Long          'Dimension

Init:
  old = 0                     'Initialize
  Cnt_Mode(0)                 'All counters on external clock
input
  Cnt_Set(1b)                 'counter 1 with clock (CLK) and
                              'direction (DIR) input
  Cnt_InputMode(0)            'Determine functionality
CLR/LATCH:
                              'All as CLR
  Cnt_ClearEnable(1b)         'Enables the CLR function of
counter 1
  Cnt_Clear(1b)               'Reset counter 1 to 0
  Cnt_Enable(1b)              'Start counter 1

Event:
  new = Cnt_Read(1)           'read out Latch A counter 1
  Par_1 = new - old           'Calculate the difference (f =
                              'impulses / time)
  old = new                   'Save new counter values as old
```

Cnt_ReadLatch

Cnt_Read_Latch gibt den Wert aus dem Latch A eines Zählers als Rückgabewert zurück.

Syntax

```
#Include ADWL16.Inc
ret_val = Cnt_ReadLatch(channel)
```

Parameter

channel	Zählernummer: 1…2.	LONG
ret_val	Inhalt des Latch A des Zählers	LONG

Bemerkungen

Verwenden Sie den Rückgabewert in Berechnungen (z.B. Differenzen oder Zählrichtung) nur mit Variablen vom Typ Long.

Der Zeitpunkt, zu dem der Zählerstand ins Latch übertragen wurde, hängt von der Einstellung mit **Cnt_Mode** ab:

- Externer Takteingang (**Cnt_Mode**-Bit = 0): Nur der Befehl **Cnt_Latch** überträgt den Zählerstand (in Latch A).
- Interner Takteingang (**Cnt_Mode**-Bit = 1): Jede Flanke des extern angelegten Messsignals löst einen Latch-Vorgang aus.
  In Latch A wird der Zählerstand bei der positiven Flanke des Eingangssignals zwischengespeichert, während in Latch B (siehe **Cnt_ReadFLatch**) der Zählerstand bei der negativen Flanke des Eingangssignals gespeichert wird.

Siehe auch

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16, L16-CO1, L16-DIO1, L16-DIO2

Beispiel

```
#INCLUDE ADWL16.Inc
Dim rise, rise_old, fall, fall_old AS LONG
#Define high PAR_1
#Define low PAR_2
#Define T PAR_9
#Define f PAR_10

INIT:
  rise_old = 0                  'Variablen…
  fall_old = 0                  'initialisieren
  Cnt_Mode(11b)                 'Zähler 1+2 auf internen
Takteingang
  Cnt_Set(0)                    'Alle Zähler mit 20 MHz internem
                                'Referenztakt
  Cnt_InputMode(11b)            'Funktionalität CLR/LATCH
festlegen:
                                'Zähler 1+2 als LATCH-Eingang
  Cnt_ClearEnable(0)            'Sperrt die CLR-Funktion beider
Zähler
  Cnt_Clear(11b)                'Zähler 1+2 auf 0 zurücksetzen
  Cnt_Enable(1)                 'Zähler 1 starten

EVENT:
  rise = Cnt_ReadLatch(1)'Latch A Zähler 1 auslesen
  fall = Cnt_ReadFLatch(1)'Latch B Zähler 1 auslesen
  If (rise <> rise_old) THEN'Steigende Flanke detektiert?
    T = (rise - rise_old) * 50'Periodendauer in Nanosekunden
    f = 1E9 / T               'Frequenz in Hertz
    If (fall <> fall_old) THEN'Fallende Flanke detektiert?
      high = (fall - rise) * 50'Impulsdauer in Nanosekunden
      low = (rise - fall_old) * 50'Pausendauer in Nanosekunden
    ELSE                      'Keine fallende Flanke
detektiert
      high = (fall - rise_old) * 50'Impulsdauer in Nanosekunden
      low = (rise - fall) * 50  'Pausendauer in Nanosekunden
    ENDIF
  ENDIF
  rise_old = rise               'Latch-Inhalt speichern
  fall_old = fall               'Latch-Inhalt speichern
```

Cnt_ReadFLatch

Cnt_ReadLatch gibt den Wert aus dem Latch B eines Zählers als Rückgabewert zurück (nur im Modus Pulsbreitenmessung einsetzbar).

Syntax

```
#Include ADWL16.Inc
ret_val = Cnt_ReadFLatch(CounterNo)
```

Parameter

CounterNo	Zählernummer: 1…2.	LONG
ret_val	Inhalt des Latch B des Zählers	LONG

Bemerkungen

Verwenden Sie den Rückgabewert in Berechnungen (z.B. Differenzen oder Zählrichtung) nur mit Variablen vom Typ Long.

Der Zeitpunkt, zu dem der Zählerstand ins Latch übertragen wurde, hängt von der Einstellung mit **Cnt_Mode** ab:

- Externer Takteingang (**Cnt_Mode**-Bit = 0): Nur der Befehl **Cnt_Latch** überträgt den Zählerstand (in Latch A).
- Interner Takteingang (**Cnt_Mode**-Bit = 1): Jede Flanke des extern angelegten Messsignals löst einen Latch-Vorgang aus.
  In Latch A wird der Zählerstand bei der positiven Flanke des Eingangssignals zwischengespeichert, während in Latch B (siehe **Cnt_ReadFLatch**) der Zählerstand bei der negativen Flanke des Eingangssignals gespeichert wird.

Siehe auch

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadFLatch, Cnt_Set

Gültig für

L16-DIO1, L16-DIO2

Beispiel

siehe Cnt_ReadLatch

Cnt_Set

Cnt_Set definiert den Betriebsmodus für alle Zähler (in Abhängigkeit von **Cnt_Mode**) gemäß dem Bitmuster in pattern.

Syntax

```
#Include ADWL16.Inc
Cnt_Set(pattern)
```

Parameter

pattern Bitmuster, Bit-Bedeutung siehe Tabelle. LONG

Bit-Wert in pattern	externer Takteingang Bit = 0 in **Cnt_Mode**	interner Takteingang Bit = 1 in **Cnt_Mode**
Bit = 0	4-Flankenauswertung	Referenztakt 20 MHz
Bit = 1	Takt- und Richtungseingang	Referenztakt 5 MHz

Bit-Nr.	31…2	1	0
Zähler-Nr.	–	2	1

Bemerkungen

Verwenden Sie diesen Befehl möglichst nur bei gesperrtem Zähler.

Siehe auch

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch

Gültig für

L16-DIO1, L16-DIO2

Beispiel

```
#INCLUDE ADWL16.Inc
INIT:
  Cnt_Mode(0)                 'Alle Zähler auf externen
Takteingang
  Cnt_Set(00b)                'Zähler 1+2 mit
Vierflankenauswertung
  Cnt_Enable(11b)             'Zähler 1+2 aktivieren
```

13.5 CAN-Schnittstelle

Dieser Abschnitt beschreibt folgende Befehle:

CAN_Msg

CAN_Msg ist ein eindimensionales Feld mit 9 Elementen, in dem die Message-Objekte gespeichert sind oder werden.

Syntax

```
#Include ADWL16.Inc
CAN_Msg[n] = value
```

oder

```
value = CAN_Msg[n]
```

Parameter

n	Elementnummer im Feld **CAN_Msg** (1…9)	LONG
value	Wert (8 Bit), der in das Message-Objekt geschrieben oder daraus gelesen wird.	LONG

Bemerkungen

Die Elemente des Felds **CAN_Msg**[] haben folgende Funktion:

Elementnr. in **CAN_Msg**	1…8	9
Inhalt	Message-Objekt(e) = Datenbyte(s)	Anzahl (0…8) belegter Datenbytes

Tragen Sie die zu übertragenden Werte in das Feld **CAN_Msg**[] ein, *bevor* Sie diese mit **Transmit** übertragen.

Siehe auch

Init_CAN, Read_Msg, Read_Msg_Con, Transmit

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc
Rem Sendet eine 32 Bit-FLOAT-Zahl (hier: Pi) als
 Folge von
Rem 4 Bytes in einem Message-Objekt

#Define pi 3.14159265
Dim i As Long

Init:
  Init_CAN()                    'CAN-Controller
                                'initialisieren

  Rem Message-Objekt 6 der Schnittstelle
   initialisieren
  Rem zum Senden von CAN-Nachrichten mit dem
   Identifier 40
  En_Transmit(6,40,0)

  Rem Bitmuster von Pi mit Datenformat Long
   erzeugen
  Par_1 = Cast_FloatToLong(pi)

  Rem Bitmuster (32 Bit) in 4 Bytes aufteilen
  CAN_Msg[4] = Par_1 And 0FFh 'LSB zuweisen
  For i = 1 To 3
    CAN_Msg[4-i] = Shift_Right(Par_1,8*i) And 0FFh
  Next i
  CAN_Msg[9] = 4                'Länge der Nachricht in
                                'Bytes

Event:
  Transmit(6)                   'Message-Objekt 6 senden
```

Empfangen einer Fließkomma-Zahl siehe Bsp. bei Read_Msg.

En_Interrupt

En_Interrupt konfiguriert ein bestimmtes Message-Objekt so, dass bei Eintreffen einer Nachricht ein externer Event erzeugt wird.

Syntax

```
#Include ADWL16.Inc
En_Interrupt(msg_no)
```

Parameter

msg_no	Nummer (1...15) des Message-Objektes im CAN-Controller	Long

Bemerkungen

- / -

Siehe auch

CAN_Msg, En_Receive, Get_CAN_Reg, Set_CAN_Reg

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc

Init:
  Init_CAN()                    'CAN-Controller initialisieren
  En_Receive(1,200,0)           'Initialisiere das Message-
Objekt 1
                                'zum Empfangen von CAN-
Nachrichten mit
                                'dem Identifier 200
  En_Interrupt(1)               'Gibt das Auslösen von
Interrupts
                                '(ext. EVENT) beim Empfang des
                                'Message-Objektes 1 frei
```

En_Receive

En_Receive gibt ein bestimmtes Message-Objekt zum Nachrichten-Empfang frei.

Syntax

```
#Include ADWL16.Inc
En_Receive(msg_no, id, id_extend)
```

Parameter

msg_no	Nummer (1…15) des Message-Objekts.	LONG
id	Identifier (0…2^{11} oder 0…2^{29}) der Nachrichten, die in diesem Message-Objekt empfangen werden können.	LONG
id_extend	Länge des Identifiers: 0: 11 Bit 1: 29 Bit	LONG

Bemerkungen

Ein Message-Objekt kann nur Nachrichten vom CAN-Bus empfangen, wenn Sie es zuvor mit **En_Receive** zum Empfang freigegeben haben.

Das Message-Objekt empfängt nur Nachrichten mit dem von Ihnen angegebenen Identifier.

Siehe auch

CAN_Msg, En_Transmit, Read_Msg, Read_Msg_Con

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc
Init:
  Init_CAN()                    'CAN-Controller initialisieren
  En_Receive(1,200,0)           'Initialisiere Message-Objekt 1
der
                                'zum Empfangen von Nachrichten
mit
                                'dem Identifier 200
```

En_Transmit

En_Transmit gibt ein bestimmtes Message-Objekt zum Nachrichten-Senden frei.

Syntax

```
#Include ADWL16.Inc
En_Transmit(msg_no, id, id_extend)
```

Parameter

msg_no	Nummer (1…14) des Message-Objektes	LONG
id	Identifier, der mit den Nachrichten dieses Message-Objekts gesendet wird.	LONG
id_extend	Länge des Identifiers: 0: 11 Bit 1: 29 Bit	LONG

Bemerkungen

Erst wenn ein Message-Objekt mit **En_Transmit** zum Senden freigegeben ist, kann das Objekt Nachrichten auf dem CAN-Bus senden.

Siehe auch

CAN_Msg, En_Receive, Transmit

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc
Init:
  Init_CAN()                    'CAN-Controller initialisieren
  En_Transmit(6,40,0)           'Initialisiere das Message-
Objekt 6
                                'zum Senden von CAN-Nachrichten
mit
                                'dem Identifier 40
```

Get_CAN_Reg

Get_CAN_Reg liest den Wert eines bestimmten Registers im CAN-Controller.

Syntax

```
#Include ADWL16.Inc
ret_val = Get_CAN_Reg(regno)
```

Parameter

regno	Register-Nummer (0…255) im CAN-Controller	LONG
ret_val	Inhalt des Registers (übergeben in den unteren 8 Bit)	LONG

Bemerkungen

Sie finden die Registernummern des CAN-Controllers AN82527 im Intel®-Datenblatt. Beispiele sind:

- Adresse **00h**: Kontroll-Register
- Adresse **01h**: Status-Register
- Adresse **5fh**: Interrupt-Register

Siehe auch

Init_CAN, Set_CAN_Baudrate, Set_CAN_Reg

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc
Init:
  Init_CAN()                    'CAN-Controller initialisieren
  Par_1 = Get_CAN_Reg(0)'Control-Register auslesen
```

Init_CAN

Init_CAN initialisiert den CAN-Controller.

Syntax

```
#Include ADWL16.Inc
Init_CAN()
```

Parameter

- / -

Bemerkungen

Die Anweisung führt folgende Aktionen aus:

- Reset (Hardware-Reset des CAN-Controllers)
- Alle Filter auf „must match“ setzen.
- Clockout-Register auf 0 setzen (= externe Frequenz wird nicht geteilt).
- Register „Bus-Configuration“ auf 0 setzen.
- Übertragungsrate für den CAN-Bus auf 1 MBit/s setzen.
- Alle Message-Objekte sperren.

Sie müssen diese Anweisung ausführen, bevor Sie mit anderen Befehlen auf den CAN-Controller zugreifen. Wir empfehlen die Angabe im Prozessabschnitt **LowInit:** oder **Init:**.

Siehe auch

CAN_Msg, En_Interrupt, En_Receive, En_Transmit, Get_CAN_Reg, Set_CAN_Baudrate, Set_CAN_Reg

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc
Init:
  Init_CAN()                    'Initialisiere den CAN-
Controller
```

Read_Msg

Read_Msg prüft, ob neue Nachrichten in einem bestimmten Message-Objekt empfangen wurden.

Falls ja, wird die Nachricht in **CAN_Msg** gespeichert und der Identifier der Nachricht zurückgegeben.

Syntax

```
#Include ADWL16.Inc
ret_val = Read_Msg(msg_no)
```

Parameter

msg_no	Nummer (1…15) des Message-Objektes	LONG
ret_val	-1: keine neue Nachricht >0:Neue Nachricht; Wert = Identifier der Nachricht	LONG

Bemerkungen

Um eine Nachricht zu empfangen, müssen Sie folgende Reihenfolge einhalten:

- Einmal: Geben Sie das Message-Objekt mit **En_Receive** zum Empfangen frei.
- Sooft erforderlich: Prüfen Sie auf eine neue Nachricht und – falls vorhanden – speichern die Nachricht in **CAN_MSG** mit **Read_Msg**.

Sie können eine empfangene Nachricht nur einmal auslesen.

Siehe auch

CAN_Msg, En_Interrupt, En_Receive, En_Transmit, Read_Msg

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.inc
Rem Wenn eine neue Nachricht mit dem passenden Identifier
Rem empfangen wurde, werden die Daten gelesen. Die
Rem ersten 4 Bytes der Nachricht werden zu einer Fließkomma-
Rem Zahl mit 32 Bit Länge zusammengesetzt.
Dim n As Long

Init:
  Par_1 = 0
  Init_CAN()                    'CAN-Controller initialisieren
  En_Receive(1,40,0)            'Message-Objekt 1 initialisieren
                                'zum Empfangen von CAN-
Nachrichten mit
                                'dem Identifier 40

Event:
  Rem Wenn das Message-Objekt geändert wurde, werden die
  Rem empfangenen Daten aus Objekt 1 gelesen und der
  Rem Identifier an Par_9 übergeben.
  Rem Die Daten stehen im Feld CAN_Msg[] bereit.
  Par_9 = Read_Msg(1)

  If (Par_9 = 40) Then
    Rem Für das Message-Objekt ist eine neue Nachricht mit dem
    Rem Identifier 40 eingetroffen
    Par_1 = CAN_Msg[1]          'High-Byte auslesen
    For n = 2 To 4              'Mit restlichen 3 Bytes zu 32
Bit-Zahl
      Par_1 = Shift_Left(Par_1,8) + CAN_Msg[n] 'zusammenfügen
    Next n
    Rem Das Bitmuster in Par_1 in den Datentyp FLOAT wandeln und
    Rem der Variablen FPar_1 zuweisen.
    FPar_1 = Cast_LongToFloat(Par_1)
  EndIf
```

Senden einer Fließkomma-Zahl siehe Bsp. bei Transmit.

Read_Msg_Con

Read_Msg_Con prüft, ob eine vollständige neue Nachricht in einem bestimmten Message-Objekt empfangen wurde.

Falls ja, wird die Nachricht in **CAN_Msg** gespeichert und der Identifier der Nachricht zurückgegeben.

Syntax

```
#Include ADWL16.Inc
ret_val = Read_Msg_Con(msg_no)
```

Parameter

msg_no	Nummer (1…15) des Message-Objekts.	LONG
ret_val	-1: keine neue Nachricht >0:Neue Nachricht; ret_val = Identifier der Nachricht	LONG

Bemerkungen

Im Unterschied zu **Read_Msg** stellt **Read_Msg_Con** sicher, dass die Nachricht konsistent ist: Wenn während des Auslesens eine neue Nachricht eintrifft, kann es nicht zu einer Mischung der alten und der neuen Nachricht kommen.

Um eine Nachricht zu empfangen, müssen Sie folgende Reihenfolge einhalten:

- Einmal: Geben Sie das Message-Objekt mit **En_Receive** zum Empfangen frei.
- Sooft erforderlich: Prüfen Sie auf eine neue Nachricht und – falls vorhanden – speichern die Nachricht in **CAN_MSG** mit **Read_Msg**.

Sie können eine empfangene Nachricht nur einmal auslesen.

Siehe auch

CAN_Msg, En_Interrupt, En_Receive, En_Transmit, Read_Msg

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc
Rem Wenn eine neue Nachricht mit dem passenden Identifier
Rem empfangen wurde, werden die Daten gelesen. Die
Rem ersten 4 Bytes der Nachricht werden zu einer Fließkomma-
Rem Zahl mit 32 Bit Länge zusammengesetzt.
Dim n As Long

Init:
  Par_1 = 0
  Init_CAN()                    'CAN-Controller initialisieren
  En_Receive(1,40,0)            'Message-Objekt 1 initialisieren
                                'zum Empfangen von CAN-
Nachrichten mit
                                'dem Identifier 40

Event:
  Rem Wenn das Message-Objekt geändert wurde, werden die
  Rem empfangenen Daten aus Objekt 1 gelesen und der
  Rem Identifier an Par_9 übergeben.
  Rem Die Daten stehen im Feld CAN_Msg[] bereit.
  Par_9 = Read_Msg_Con(1)

  If (Par_9 = 40) Then
    Rem Für das Message-Objekt ist eine neue Nachricht mit dem
    Rem Identifier 40 eingetroffen
    Par_1 = CAN_Msg[1]          'High-Byte auslesen
    For n = 2 To 4              'Mit restlichen 3 Bytes zu 32
Bit-Zahl
      Par_1 = Shift_Left(Par_1,8) + CAN_Msg[n] 'zusammenfügen
    Next n
    Rem Das Bitmuster in Par_1 in den Datentyp FLOAT wandeln und
    Rem der Variablen FPar_1 zuweisen.
    FPar_1 = Cast_LongToFloat(Par_1)
  EndIf
```

Senden einer Fließkomma-Zahl siehe Bsp. bei Transmit.

Set_CAN_Baudrate

Set_CAN_Baudrate stellt die Baudrate des CAN-Controllers ein.

Syntax

```
#Include ADWL16.Inc
ret_val = Set_CAN_Baudrate(rate)
```

Parameter

rate	Baudrate des CAN-Controllers in Bit/Sekunde.	LONG
ret_val	Status der Befehlsausführung: 0: Baudrate wurde eingestellt 1: Baudrate unzulässig	LONG

Bemerkungen

Die möglichen Baudraten (= Busfrequenzen) entnehmen Sie bitte der Tabelle „Einstellbare Baudraten“ im Anhang. Übernehmen Sie bitte die genaue Schreibweise, d.h. nicht ganzzahlige Baudraten mit 4 Nachkommastellen; Werte mit abweichender Schreibweise werden als nicht zulässig zurückgewiesen.

Die Anweisung führt folgende Aktionen aus:

- Prüfen, ob die übergebene Baudrate zulässig ist. Falls nicht, dann den Rückgabewert auf 1 setzen und die Bearbeitung beenden.
- Die Register des CAN-Controllers für die Baudrate setzen.
- Sampling Mode auf 0 setzen: Ein Sample pro Bit.
- Die Einstellungen so wählen, dass der Sample-Punkt immer zwischen 60% und 72% der Gesamt-Bitlänge liegt.
- Die Sprungweite zur Synchronisation auf 1 setzen.

In Sonderfällen kann es vorteilhaft sein, die Einstellungen anders zu wählen als oben beschrieben. Sie finden hierzu eine Erläuterung im Hardware-Handbuch.

Die Anweisung sollte in den Programm-Abschnitten **LowInit:** oder **Init:** aufgerufen werden, und zwar erst nach der Anweisung Initialisierung, weil sonst die eingestellte Baudrate wieder mit der Standardeinstellung (1MBit/s) überschrieben wird.

Siehe auch

Get_CAN_Reg, Init_CAN, Set_CAN_Reg

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc
Dim status As Long

Init:
  Init_CAN()                    'CAN-Controller initialisieren
  status = Set_CAN_Baudrate(125000)'Baudrate 125 kBit/s
                                'setzen
```

Set_CAN_Reg

Set_CAN_Reg schreibt einen Wert in ein bestimmtes Register des CAN-Controllers.

Syntax

```
#Include ADWL16.Inc
Set_CAN_Reg(regno, value)
```

Parameter

regno	Register-Nummer (0…255) im CAN-Controller	LONG
value	Wert (8 Bit), der ins Register geschrieben wird.	LONG

Bemerkungen

Sie finden die Registernummern des CAN-Controllers AN82527 im Intel®-Datenblatt.

Siehe auch

Get_CAN_Reg, Init_CAN, Set_CAN_Baudrate

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc

Init:
  Init_CAN()                  'CAN-Controller initialisieren
  Set_CAN_Reg(0,1)            'Control-Register auf den Wert 1
                              'setzen
```

Transmit

Transmit sendet die Nachricht in CAN_Msg über ein bestimmtes Message-Objekt.

Syntax

```
#Include ADWL16.Inc
Transmit(msg_no)
```

Parameter

msg_no	Nummer (1…14) des Message-Objektes	LONG

Bemerkungen

Um eine Nachricht zu senden, müssen Sie folgende Reihenfolge einhalten:

- Einmal: Geben Sie das Message-Objekt mit **En_Transmit** zum Senden frei.
- Sooft erforderlich: Geben Sie die Nachricht in das Feld **CAN_MSG** ein: Die Datenbytes und die Anzahl der Datenbytes.
- Senden Sie die Nachricht mit **Transmit**.

Die CAN-Schnittstelle sendet die Nachricht, sobald das Message-Objekt Zugriffsrecht auf den CAN-Bus hat.

Siehe auch

CAN_Msg, En_Transmit, Init_CAN, Set_CAN_Baudrate

Gültig für

L16-DIO1

Beispiel

```
#Include ADWL16.Inc
#Define pi 3.14159265
Dim i As Long

Init:
  Init_CAN()                'CAN-Controller initialisieren

  Rem Message-Objekt 6 der Schnittstelle initialisieren
  Rem zum Senden von CAN-Nachrichten mit dem Identifier 40
  En_Transmit(6,40,0)

  Rem Bitmuster von Pi mit Datenformat Long erzeugen
  Par_1 = Cast_FloatToLong(pi)

  Rem Bitmuster (32 Bit) in 4 Bytes aufteilen
  CAN_Msg[4] = Par_1 And 0FFh 'LSB zuweisen
  For i = 1 To 3
    CAN_Msg[4-i] = Shift_Right(Par_1,8*i) And 0FFh
  Next i
  CAN_Msg[9] = 4            'Länge der Nachricht in Bytes

Event:
  Transmit(6)               'Message-Objekt 6 senden
```

Empfangen einer Fließkomma-Zahl siehe Bsp. bei Read_Msg.

13.6 SSI-Schnittstelle

Dieser Abschnitt beschreibt folgende Befehle:

SSI_Mode

SSI_Mode stellt den Modus aller SSI-Decoder ein, entweder „single shot“ (einzeln lesen) und „continuous“ (kontinuierlich lesen).

Syntax

```
#Include ADWL16.Inc
SSI_Mode(pattern)
```

Parameter

pattern	Betriebsmodus der SSI-Decoder, angegeben als Bitmuster. Jedem Decoder ist ein Bit zugeordnet (siehe Tabelle). Bit = 0: Modus „Single shot“, der Encoder wird einmal ausgelesen. Bit = 1: Modus „Continuous“, der Encoder wird kontinuierlich ausgelesen.	LONG

Bitnr.	31:1	0
SSI-Decoder	–	1

Bemerkungen

Wenn Sie den Modus „continuous“ wählen, startet das Auslesen des entsprechenden Encoders sofort. **SSI_Start** ist hierzu nicht erforderlich.

Manche Encoder-Typen liefern im Modus „continuous“ gelegentlich den falschen Messwert 0 (Null) anstelle des korrekten Messwerts zurück. Im Modus „single shot“ tritt dieser Fehler nicht auf.

Siehe auch

SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Start, SSI_Status

Gültig für

L16-DIO1, L16-DIO2

Beispiel

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,10)          'Taktrate  1.0 MHz einstellen
  SSI_Mode(1b)                 'Continuous-Mode setzen
  SSI_Set_Bits(1,10)           '10 Encoder-Bits

Event:
  Par_1 = SSI_Read(1)          'Pos.wert auslesen
```

SSI_Read

SSI_Read gibt den zuletzt gespeicherten Zählerstand eines bestimmten SSI-Zählers zurück.

Syntax

```
#Include ADWL16.Inc
ret_val = SSI_Read(dcdr_no)
```

Parameter

dcdr_no	Nummer (1) des SSI-Decoders, dessen Zählerstand auszulesen ist.	LONG
ret_val	Letzter Zählerstand des SSI-Zählers (= Absolutwert-Position des Encoders)	LONG

Bemerkungen

Ein Encoder-Wert wird dann gespeichert, wenn die durch **SSI_Set_Bits** angegebene Anzahl von Bits eingelesen wurde.

Siehe auch

SSI_Mode, SSI_Set_Bits, SSI_Set_Clock, SSI_Start, SSI_Status

Gültig für

L16-DIO1, L16-DIO2

Beispiel

```
#Include ADWL16.Inc
Rem Decoder läuft mit 200 kHz
Dim m, n, y AS LONG

Init:
  Rem Einstellungen für Decoder 1
  SSI_Set_Clock(1,50)          'Taktrate
  SSI_Mode(1)                  'Continuous-Mode
  SSI_Set_Bits(1,23)           '23 Encoder-Bits

Event:
  PAR_1 = SSI_Read(1)          'Pos.wert auslesen

  Rem Wert von Gray-Code in Binärwert wandeln:
  m = 0                        'vorigen Wert löschen
  y = 0                        ' -"-
  FOR n = 1 TO 32              'Alle 32 mögl. Bits durchgehen
    m = (Shift_Right(PAR_1,(32 - n)) AND 1) XOR m
    y = (Shift_Left(m,(32 - n))) OR y
  NEXT n
  Rem Das Ergebnis der Gray-/Binär-Wandlung in PAR_9
  PAR_9 = y
```

SSI_Set_Bits

SSI_Set_Bits stellt für einen bestimmten SSI-Zähler die Anzahl der zu Bits ein, die einen vollständigen Encoder-Wert bilden.

Die Zahl der Bits sollte mit der Auflösung des Encoders identisch sein.

Syntax

```
#Include ADWL16.Inc
SSI_Set_Bits(dcdr_no,bit_count)
```

Parameter

dcdr_no	Nummer (1) des SSI-Decoders, dessen Zählerstand auszulesen ist.	LONG
bit_count	Anzahl der Bits (1…32) der zu lesenden Bits für einen Encoder-Wert (entspricht der Encoder-Auflösung).	LONG

Bemerkungen

Die Auflösung (Anzahl der Bits) des SSI-Encoders sollte mit der Anzahl der zu übertragenden Bits übereinstimmen.

Achten Sie darauf, dass die zu lesenden Bits mit der Encoder-Auflösung genau übereinstimmen.

Siehe auch

SSI_Mode, SSI_Read, SSI_Set_Clock, SSI_Start, SSI_Status

Gültig für

L16-DIO1, L16-DIO2

Beispiel

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,10)          'Taktrate 1,0 MHz einstellen
  SSI_Mode(11b)                'Continuous-Mode setzen
  SSI_Set_Bits(1,10)           '10 Encoder-Bits

Event:
  Par_1 = SSI_Read(1)          'Pos.wert auslesen
```

SSI_Set_Clock

SSI_Set_Clock stellt die Taktrate (ca. 40 kHz bis 1 MHz) ein, mit der der Encoder getaktet wird.

Syntax

```
#Include ADWL16.Inc
SSI_Set_Clock(dcdr_no,prescale)
```

Parameter

dcdr_no	Nummer (1) des SSI-Decoders, dessen Zählerstand auszulesen ist.	LONG
prescale	Teilerfaktor (10…255) zur Einstellung der Taktrate nach der Formel: Taktrate = 10 MHz / prescale	LONG

Bemerkungen

Teilerfaktoren kleiner 10 werden automatisch auf den Wert 10 korrigiert; bei Werten über 255 werden die niederwertigsten 8 Bits als Teilerfaktor verwendet.

Die mögliche Taktfrequenz ist abhängig von Kabellänge, Kabeltyp und den jeweils verwendeten Sende- und Empfangsbausteinen des Encoders und des Decoders. Grundsätzlich gilt als Regel: Je höher die Taktfrequenz, desto kürzer die mögliche Kabellänge.

Siehe auch

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Start, SSI_Status

Gültig für

L16-DIO1, L16-DIO2

Beispiel

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,10)          'Taktrate 1,0 MHz einstellen
  SSI_Mode(11b)                'Continuous-Mode setzen
  SSI_Set_Bits(1,10)           '10 Encoder-Bits

Event:
  Par_1 = SSI_Read(1)          'Pos.wert auslesen
```

SSI_Start

SSI_Start startet das Auslesen der gewählten SSI-Encoder (nur im Modus „single shot“).

Syntax

```
#Include ADWL16.Inc
SSI_Start(pattern)
```

Parameter

pattern	Bitmuster zur Auswahl der SSI-Decoder, die gestartet werden sollen: Bit = 0: keine Funktion Bit = 1: Auslesen des SSI-Decoders starten	LONG

Bemerkungen

Im Modus „continuous“ ist diese Anweisung ohne Funktion, weil dann die Encoder-Werte ohnehin kontinuierlich ausgelesen werden.

Ein Encoder-Wert wird dann gespeichert, wenn die durch **SSI_Set_Bits** angegebene Anzahl von Bits eingelesen wurde.

Es wird immer ein vollständiger Encoder-Wert übertragen, auch wenn währenddessen der Modus umgestellt wird.

Siehe auch

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Status

Gültig für

L16-DIO1, L16-DIO2

Beispiel

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,250)          'Taktrate 40 kHz einstellen
  SSI_Mode(0)                   'Single shot-Mode einstellen
  SSI_Set_Bits(1,23)            '23 Encoder-Bits

Event:
  SSI_Start(1b)                 'Positionswert lesen
  Do
  Until (SSI_Status(1) = 0)
  Rem Wenn Positionswert komplett gelesen ist …

  Par_1 = SSI_Read(1)           'Positionswert auslesen
```

SSI_Status

SSI_Status liefert für einen bestimmten Decoder den aktuellen Lese-Status zurück.

Syntax

```
#Include ADWL16.Inc
ret_val = SSI_Status(dcdr_no)
```

Parameter

dcdr_no	Nummer (1) des SSI-Decoders, dessen Zählerstand auszulesen ist.	LONG
ret_val	Lese-Status des Decoders: 0: Decoder ist bereit, d.h. ein vollständiger Wert wurde gelesen. 1: Decoder liest einen Encoder-Wert ein.	LONG

Bemerkungen

Verwenden Sie die Status-Abfrage nur im SSI-Modus „single shot“. Im Modus „continuous“ ist eine Status-Abfrage nicht sinnvoll.

Siehe auch

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Start

Gültig für

L16-DIO1, L16-DIO2

Beispiel

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,250)          'Taktrate 40 kHz einstellen
  SSI_Mode(0)                   'Single shot-Mode einstellen
  SSI_Set_Bits(1,23)            '23 Encoder-Bits

Event:
  SSI_Start(1b)                 'Positionswert lesen
  Do
  Until (SSI_Status(1) = 0)
  Rem Wenn Positionswert komplett gelesen ist …

  Par_1 = SSI_Read(1)           'Positionswert auslesen
```

13.7 PWM-Ausgang

Dieser Abschnitt beschreibt Befehle zum Ansprechen des PWM-Ausgangs auf *ADwin-light-16* mit PWM1-Erweiterung:

PWM_Activate

PWM_Activate schaltet Pin 34 auf der Sub-D-Buchse ADwin I/O-Connector um als PWM-Ausgang oder als digitalen Ausgang.

Syntax

```
#Include ADWL16.inc
PWM_Activate(enable)
```

Parameter

enable	Status des Pins 34 auf der Sub-D-Buchse ADwin I/O-Connector: 0: digitaler Ausgang DIGOUT-05. 1: PWM-Ausgang.	LONG

Bemerkungen

Nach dem Einschalten der Hardware ist Pin 34 als digitaler Ausgang geschaltet.

Siehe auch

PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Gültig für

L16-PWM1

Beispiel

siehe PWM_Init (Seite 135)

PWM_Enable

PWM_Enable gibt den PWM-Ausgang zur Ausgabe frei oder sperrt die Ausgabe.

Syntax

```
#Include ADWL16.inc
PWM_Enable(enable)
```

Parameter

enable	Status des PWM-Ausgangs: 0: PWM-Ausgabe sperren. 1: PWM-Ausgabe freigeben.	LONG

Bemerkungen

Wann der PWM-Ausgang gesperrt wird – sofort oder nach dem nächsten Periodenende – hängt von der Einstellung ab, die mit **PWM_Init** gemacht wurde (Parameter mode).

Siehe auch

PWM_Activate, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Gültig für

L16-PWM1

Beispiel

siehe PWM_Init (Seite 135)

PWM_Get_Status

PWM_Get_Status liest den aktuellen Betriebsstatus des PWM-Ausgangs.

Syntax

```
#Include ADWL16.inc
ret_val = PWM_Get_Status()
```

Parameter

ret_val	Status des PWM-Ausgangs. 0: PWM-Ausgabe ist abgeschlossen. 1: PWM-Ausgabe läuft.	LONG

Bemerkungen

- / -

Siehe auch

PWM_Activate, PWM_Enable, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Gültig für

L16-PWM1

Beispiel

- / -

PWM_Init

PWM_Init setzt die Voreinstellungen für den PWM-Ausgang.

Syntax

```
#Include ADWL16.inc
PWM_Init(pwm_output, startdelay, startvalue, mode,
    count)
```

Parameter

pwm_output	Nummer (1) des PWM-Ausgabekanals.	LONG
startdelay	Startverzögerung in Einheiten von 25ns.	LONG
startvalue	Startpegel für die PWM-Ausgabe: 0: TTL-Pegel low. 1: TTL-Pegel high.	LONG
mode	Betriebsmodus des PWM-Ausgangs als Bitmuster; nur die Bits 0…2 sind relevant, andere Bits werden ignoriert. Bit 0: Übernahme einer neuen PW-Frequenz: • Bit = 0: Übernahme bei Periodenende • Bit = 1: Übernahme sofort. Bit 1: Anzahl der Pulse: • Bit = 0: unendlich viele Perioden. • Bit = 1: Anzahl der Perioden ist count. Bit 2: Anhalten bei Stopp-Befehl: • Bit = 0: Anhalten bei Periodenende • Bit = 1: Anhalten sofort.	LONG
count	Anzahl der Perioden (1…32768), die ausgegeben werden. Nur relevant, wenn mode, bit 1 = 1.	LONG

Bemerkungen

- / -

Siehe auch

PWM_Activate, PWM_Enable, PWM_Get_Status, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Gültig für

L16-PWM1

Beispiel

```
#Include ADWL16.inc
Rem You can set frequency and duty cycle online with the
Rem global variables FPar_1 and FPar_2:
#Define freq1 FPar_1       'frequency
#Define pw1 FPar_2         'duty cycle

Init:
  PWM_Activate(1)          'enable pin 34 as PWM output
  freq1 = 1000             '1000 Hz
  pw1 = 50                 '50 %
  PWM_Reset(01b)           'stop PWM channel
  PWM_Init(1,0,0,0,0)      'initialize PWM settings

  PWM_Write_Latch(1,pw1,freq1)'set frequency and duty cycle
  PWM_Latch(1)             'enable output of PWM signal
  PWM_Enable(1)            'start output

Event:
  PWM_Write_Latch(1,pw2,freq2)'set new frequency and duty cycle
  PWM_Latch(1)             'set frequency and duty cycle

Finish:
  PWM_Activate(0)          'reset pin 34 as digital output
```

PWM_Latch

PWM_Latch gibt Frequenz und Tastverhältnis des PWM-Ausgangs für die Ausgabe frei.

Syntax

```
#Include ADWL16.inc
PWM_Latch(pattern)
```

Parameter

pattern	Ausgabestatus des PWM-Ausgangs: 0: Kein Einfluss. 1: Latchen = für Ausgabe freigeben.	LONG

Bemerkungen

Frequenz und Tastverhältnis werden mit **PWM_Write_Latch** in das Latch-Register geschrieben. Erst mit **PWM_Latch** werden die Werte aus dem Latch-Register ausgegeben.

Wann die Ausgabe mit den neuen Werten beginnt – sofort oder nach dem nächsten Periodenende – hängt von der Einstellung ab, die mit **PWM_Init** gemacht wurde (Parameter mode).

Siehe auch

PWM_Activate, PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Gültig für

L16-PWM1

Beispiel

siehe PWM_Init (Seite 135)

PWM_Reset

PWM_Reset stoppt die Ausgabe auf dem PWM-Ausgang sofort.

Syntax

```
#Include ADWL16.inc
PWM_Reset(pattern)
```

Parameter

pattern	Status des PWM-Ausgangs: 0: Kein Einfluss 1: Ausgabe sofort stoppen	LONG

Bemerkungen

Die Ausgabe wird auch dann sofort gestoppt, wenn mit dem Parameter mode bei **PWM_Init** ein anderer Modus eingestellt ist.

Siehe auch

PWM_Activate, PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Standby_Value, PWM_Write_Latch

Gültig für

L16-PWM1

Beispiel

siehe PWM_Init (Seite 135)

PWM_Standby_Value

PWM_Standby_Value setzt den Vorgabewert (TTL-Pegel) für den PWM-Ausgang.

Syntax

```
#Include ADWL16.inc
PWM_Standby_Value(level)
```

Parameter

level	Vorgabewert für PWM-Ausgang: 0: TTL-Pegel low 1: TTL-Pegel high	LONG

Bemerkungen

Wenn der PWM-Ausgang nicht mit **PWM_Enable** freigegeben ist, wird der Ausgang auf den Vorgabepegel aus pattern gesetzt. Der Vorgabepegel wird auch gesetzt, wenn der PWM-Ausgang stoppt.
Nach dem Einschalten ist der PWM-Ausgang zunächst auf TTL-Pegel low gesetzt.

Siehe auch

PWM_Activate, PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Write_Latch

Gültig für

L16-PWM1

Beispiel

- / -

PWM_Write_Latch

PWM_Write_Latch schreibt Frequenz und Tastverhältnis in das Latch-Register.

Syntax

```
#Include ADWL16.inc
PWM_Write_Latch(pwm_output, dutycycle, frequency)
```

Parameter

pwm_output	Nummer des PWM-Ausgabekanals (1).	LONG
dutycycle	Tastverhältnis in Prozent zwischen 0.0 und 100.0 (die Werte 0.0 und 100.0 sind nicht zulässig).	FLOAT
frequency	Frequenz in Hertz: 0,02 Hz …20 MHz.	FLOAT

Bemerkungen

Frequenz und Tastverhältnis werden mit **PWM_Write_Latch** nur in das Latch-Register geschrieben. Erst mit **PWM_Latch** werden die Werte für die PWM-Ausgabe aktiviert.

Der Wert für dutycycle ist abhängig von der Einstellung des Parameters startvalue bei dem Befehl **PWM_Init:**

- startvalue = 1: Geben Sie für dutycycle das Tastverhältnis an.
- startvalue = 0: Geben Sie für dutycycle das „inverse Tastverhältnis“ an: dutycycle = 100% - Tastverhältnis

Die höchste Ausgangsfrequenz, bei der das Tastverhältnis noch in 1%-Schritten einstellbar ist, beträgt etwa 400 kHz.

Siehe auch

PWM_Activate, PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value

Gültig für

L16-PWM1

Beispiel

siehe PWM_Init (Seite 135)

13.8 SPI-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen der SPI-Schnittstelle auf *ADwin-light-16* mit PWM1-Erweiterung:

SPI_Config

SPI_Config konfiguriert den SPI-Master.

Syntax

```
#Include ADWL16.inc
SPI_Config(data_order, mode, bits, clock)
```

Parameter

data_order	Reihenfolge, in der Daten gesendet werden: 0: höchstwertiges Bit (MSB) zuerst senden 1: kleinstwertiges Bit (LSB) zuerst senden	LONG
mode	Modus des SPI-Masters: 0: CPOL=0, CPHA=0 1: CPOL=0, CPHA=1 2: CPOL=1, CPHA=0 3: CPOL=1, CPHA=1	LONG
bits	Anzahl (1…32) der übertragenen Bits in einer SPI-Nachricht.	LONG
clock	Einstellung (1…7) der Taktrate: 1: 5000kHz 2: 2500kHz 3: 1250kHz 4: 620kHz 5: 312,5kHz 6: 156,25kHz 7: 78,125kHz	LONG

Bemerkungen

Wir empfehlen, zunächst alle Slaves über die Slave Select-Leitungen zu deaktivieren, bevor Sie den SPI-Master konfigurieren. Beim Konfigurieren des SPI Masters können Spikes entstehen, die von angeschlossenen Slaves falsch interpretiert werden. Dieser Fehler bringt die Datenübertragung durcheinander.

Weitere Hinweise zum SPI-Protokoll finden Sie in Kapitel 10.2 „SPI-Schnittstelle“ auf Seite 59.

Siehe auch

SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Status, SPI_Wait

Gültig für

L16-PWM1

Beispiel

```
#Include ADwL16.inc

Rem SPI settings
#Define data_order 0          'MSB first
#Define mode 3                'CPOL = 1, CPHA = 1
#Define bits 8                'number of data bits
#Define clock 7               'clock rate: 78.125 kHz

Init:
  Rem Disable slave select via TTL high on pin DIGOUT-1
  Set_Digout(0)
  SPI_Enable(1)               'enable ADwin I/O Connector pins
  SPI_Config(data_order, mode, bits, clock)
  Par_1 = 10                  'first data value

Event:
  SPI_Set_MOSI(Par_1)         'set data value to be sent

  Rem To select a slave you have to send the signal „Slave
  Rem select“ to the SPI slave. Connect a free DIO output to
  Rem the slave SPI input and set the required TTL output level
  Rem with the appropriate standard DIO instruction.

  Rem Start slave select via TTL low on pin DIGOUT-1
  Clear_Digout(0)
  SPI_Start()                 'start data transfer
  SPI_Wait()                  'wait until end of data transfer
  Set_Digout(0)               'end slave select

  Par_2 = SPI_Get_MISO()      'read received data value
```

SPI_Enable

SPI_Enable schaltet Pins als SPI-Signale CLK, MISO und MOSI oder als digitale Kanäle.

Syntax

```
#Include ADWL16.inc
SPI_Enable(enable)
```

Parameter

enable	Status der Pins auf den Sub-D-Buchsen: 0: alle Pins als digitale Ausgänge (Default). 1: Pins 13, 32 und 33 auf der Buchse ADwin I/O-Connector für SPI-Signale. 3: Pins 16, 34 und 35 auf der Buchse Digital I/O für SPI-Signale.	LONG

Bemerkungen

Die Pinbelegung finden Sie auf Seite 59, Kapitel 10.2 „SPI-Schnittstelle“.

Beachten Sie: Wenn Sie Pins auf der Sub-D-Buchse Digital I/O für SPI-Signale umschalten, werden automatisch die Pins DIO-24…DIO-31 als Ausgänge und die Pins DIO16…DIO-23 als Eingänge konfiguriert. Wenn Sie die Pins später wieder als Digitalkanäle schalten, gilt wieder die vorherige, mit **Conf_DIO_E** vorgenommene Konfiguration.

Zusätzlich zu den hier eingestellten Pins benötigen Sie zum Ansprechen jedes SPI-Slave eine separate Slave Select-Leitung. Wenn Sie hierzu die übrigen digitalen Ausgänge verwenden, stellen Sie das gewünschte TTL-Signal mit den entsprechenden Befehlen für digitale Ausgänge ein, z.B. mit **Clear_Digout** oder **Set_Digout** (siehe Kapitel 13.3 auf Seite 77).

Siehe auch

SPI_Config, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Status, SPI_Wait, Conf_DIO_E, Clear_Digout, Set_Digout

Gültig für

L16-PWM1

Beispiel

siehe SPI_Config (Seite 142)

SPI_Get_MISO

SPI_Get_MISO liest eine (bereits empfangene) SPI-Nachricht aus dem Eingangsregister.

Syntax

```
#Include ADWL16.inc
ret_val = SPI_Get_MISO()
```

Parameter

ret_val	Datenwort, das vom angesprochenen SPI-Slave geschickt wurde.	LONG

Bemerkungen

Die Anzahl der übertragenen Bits im Datenwort legen Sie mit dem Parameter bits des Befehls **SPI_Config** fest.

Sie können die SPI-Nachricht erst lesen, wenn die Datenübertragung beendet ist; fragen Sie dazu den Status mit **SPI_Wait** oder **SPI_Status** ab.

Siehe auch

SPI_Config, SPI_Enable, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Status, SPI_Wait

Gültig für

L16-PWM1

Beispiel

siehe SPI_Config (Seite 142)

SPI_Set_MOSI

SPI_Set_MOSI stellt Daten zum Senden an einen ausgewählten SPI-Slave bereit.

Syntax

```
#Include ADWL16.inc
SPI_Set_MOSI(value)
```

Parameter

value	Datenwort, das an einen ausgewählten SPI-Slave geschickt werden soll.	LONG

Bemerkungen

Das Datenwort wird mit **SPI_Set_MOSI** nur bereit gestellt. Sie starten die Datenübertragung mit dem Befehl **SPI_Start**.

Die Anzahl der übertragenen Bits im Datenwort legen Sie mit dem Parameter bits des Befehls **SPI_Config** fest.

Siehe auch

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Start, SPI_Static_MISO, SPI_Status, SPI_Wait

Gültig für

L16-PWM1

Beispiel

siehe SPI_Config (Seite 142)

SPI_Start

SPI_Start startet die Datenübertragung über den SPI-Bus.

Syntax

```
#Include ADWL16.inc
SPI_Start()
```

Parameter

- / -

Bemerkungen

Die Datenübertragung arbeitet in beide Richtungen: **SPI_Start** sendet die SPI-Nachricht, die zuletzt mit **SPI_Set_MOSI** bereit gestellt wurde. Der SPI-Slave antwortet in der Regel noch während der gleichen Datenübertragung; die Antwort lesen Sie mit **SPI_Get_MOSI** aus.

Sie können das Ende der Datenübertragung mit **SPI_Wait** oder mit **SPI_Status** abfragen.

Siehe auch

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Static_MISO, SPI_Status, SPI_Wait

Gültig für

L16-PWM1

Beispiel

siehe SPI_Config (Seite 142)

SPI_Static_MISO

SPI_Static_MISO liest den aktuellen TTL-Pegel auf der Datenleitung des SPI-Bus.

Syntax

```
#Include ADWL16.inc
ret_val = SPI_Static_MISO()
```

Parameter

ret_val	TTL-Pegel auf der Datenleitung: 0: TTL-Pegel low. 1: TTL-Pegel high.	LONG

Bemerkungen

Manche SPI-Slaves verwenden die Datenleitung nicht nur zur Datenübertragung, sondern übermitteln darüber auch Signale an den SPI-Master. Für diesen Fall können Sie mit **SPI_Static_MISO** den TTL-Pegel der Datenleitung lesen und je nach SPI-Slave darauf entsprechend reagieren.

Siehe auch

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Status, SPI_Wait

Gültig für

L16-PWM1W

Beispiel

```
#Include ADwL16.inc
Rem The program communicates with an SPI slave to make it
Rem convert an analog value and put the converted value on
Rem the SPI bus.

Rem SPI settings
#Define data_order 0          'MSB first
#Define mode 0                'CPOL = 0, CPHA = 0
#Define bits 8                'number of data bits
#Define clock 3               'clock divider: 1250 kHz

Dim state As Long
Dim value As Long

Init:
  Rem Disable slave select via TTL high on pin DIGOUT-1
  Set_Digout(0)
  SPI_Enable(1)               'enable ADwin I/O Connector pins
  SPI_Config(data_order, mode, bits, clock)
  state = 0                   'state: idle

Event:
  If (state = 0) then         'send command „start conversion“
    Rem Set output value 11 to make the SPI slave start a
    Rem conversion with the slave’s ADC.
    SPI_Set_MOSI(11)
    Clear_Digout(0)           'start Slave Select
    SPI_Start(1)              'send command
    state = 1                 'state: start conversion
  EndIf

  If (state = 1) Then         'check if command is transferred
    value = SPI_Status()
    If (value = 0) Then state = 2 'state: slave runs conversion
  EndIf

  If (state = 2) Then         'check if slave is ready
    Rem The slave sets the data line TTL high if the conversion
    Rem is completed.
    value = SPI_Static_MISO()
    If (value = 1) Then state = 3 'state: conversion completed
  EndIf

  If (state = 3) Then'send command „transfer ADC value“
    Rem Set output value 12 to make the SPI slave put the
    Rem converted value on the bus.
    SPI_Set_MOSI(12)
    SPI_Start(1)              'send command and receive value
    state = 4                 'state: transfer ADC value
  EndIf

  If (state = 4) Then         'check if command is transferred
    value = SPI_Status()
    If (value = 0) Then state = 5 'state: ADC value is ready
  EndIf

  If (state = 5) Then
    Set_Digout(0)             'end Slave Select
    Par_2 = SPI_Get_MISO()    'read ADC value
    End
  EndIf
```

SPI_Status

SPI_Status gibt den Status der aktuellen SPI-Datenübertragung zurück.

Syntax

```
#Include ADWL16.inc
ret_val = SPI_Status()
```

Parameter

ret_val	Status der SPI-Datenübertragung: 0: Datenübertragung ist beendet. 1: Datenübertragung dauert noch an.	LONG

Bemerkungen

Der Befehl **SPI_Status** kann als Alternative zu **SPI_Wait** verwendet werden, um das Ende einer Datenübertragung zu erkennen.

Siehe auch

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Wait

Gültig für

L16-PWM1

Beispiel

siehe SPI_Static_MISO (Seite 148)

SPI_Wait

SPI_Wait wartet bis die SPI-Datenübertragung beendet ist.

Syntax

```
#Include ADWL16.inc
SPI_Wait()
```

Parameter

- / -

Bemerkungen

- / -

Siehe auch

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Status

Gültig für

L16-PWM1

Beispiel

siehe SPI_Config (Seite 142)

Anhang

A.1 Technische Daten

Allgemeine Daten / Grenzwerte						
	Symbol	Konditionen	min.	typ.	max.	Einheit
Versorgungs-Spannnung						
Spannung	U_b	*L16-PCI, -EURO*	4,75	5	5,25	V
		L16-EXT, Rev. A	10	12	16	
		L16-EXT, Rev. B	10	12	36	
Versorgungs-Strom mit USB-Schnittstelle						
Betriebsstrom						
L16, *L16-CO1*	I_{idle} bei $U_{b, typ.}$	*L16-PCI, -EURO*	0,75	0,9	1,5	A
		L16-EXT	0,35	0,5	0,7	
L16-DIO1, -DIO2, -DIO3		*L16-PCI, -EURO*	0,85	1,0	1,6	
		L16-EXT	0,55	0,7	1,0	
Einschaltstrombedarf						
L16, *L16-CO1*	$I_{power-on}$ bei $U_{b, typ.}$	*L16-PCI, -EURO*		6		A
		L16-EXT		3		
L16-DIO1, -DIO2, -DIO3		*L16-PCI, -EURO*		7		
		L16-EXT		3		
Versorgungs-Strom mit Ethernet-Schnittstelle						
Betriebsstrom						
L16, *L16-CO1*	I_{idle} bei $U_{b, typ.}$	*L16-PCI, -EURO*	1,0	1,2	1,8	A
		L16-EXT	0,55	0,7	0,9	
L16-DIO1, -DIO2, -DIO3		*L16-PCI, -EURO*	1,15	1,3	1,9	
		L16-EXT	0,55	0,7	1,0	
Einschaltstrombedarf						
L16, *L16-CO1*	$I_{power-on}$ bei $U_{b, typ.}$	*L16-PCI, -EURO*		6,4		A
		L16-EXT		3,3		
L16-DIO1, -DIO2, -DIO3		*L16-PCI, -EURO*		7,5		
		L16-EXT		3,3		
Betrieb						
Temperatur	$T_{Umgebung}$	*L16-PCI,* -EURO	+5		+50	°C
	$T_{Gehäuse}$	L16-EXT	+5		+55	
rel. Feuchte	F_{rel}	nicht kondensierend	0		80	%
Lagerung						
Temperatur	T		-20		+70	°C

Allgemeine Daten / Grenzwerte						
	Symbol	Konditionen	min.	typ.	max.	Einheit
Abmessungen						
Breite x Höhe x Tiefe	B x H x T	L16-PCI-USB		21,5 x 121,0 x 172,5		mm
		L16-EURO-USB [a]		25,4 x 133,35 x 187		
		L16-EURO-ENET [a]		50,8 x 133,35 x 187		
		L16-EXT-USB		226 x 109 x 44		
		L16-EXT-ENET		226 x 109 x 74		
		+ CO1-Erweiterung		wie Basisversion		
		+ DIO1/DIO2-Erweiterung		*L16-EURO*[a]: Breite +50,8 *L16-EXT*: Höhe +30		
		+ DIO3-Erweiterung		*L16-EURO*[a]: Breite +25,4 *L16-EXT*: Höhe +16		
Nettogewicht						
Gewicht	m_{Netto}	*L16-PCI*-USB		135		g
		L16-EURO-USB		165		
		L16-EURO-ENET		275		
		L16-EXT-USB		1.100		
		L16-EXT-ENET		1.400		
		+ CO1-Erweiterung		wie Basisversion		
		+ DIO1 / DIO2-Erweiterung		+140		
		+ DIO3-Erweiterung		+80		
Steckverbinder						
Sub-D-Verbinder	Metrisches ISO-Gewinde; UNC-Gewinde als Bestelloption erhältlich					
Montage						
Standard	*L16-PCI*: Einbau im PC *L16-EURO*: Einbau im 19"-Gehäuse *L16-EXT*: Tischgehäuse					
optional	Hutschienen- und Wandmontage für *L16-EXT*					

[a] Umrechnung für L16-EURO: 25,4mm = 5 TE; 50,8mm = 10 TE; 133,35mm = 3 HE

Digitale Ein- / Ausgänge						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
I/O-Leitungen						
Anzahl	Digin05:00 DIGOUT05:00	6 Eingänge und 6 Ausgänge (TTL- / 5V-CMOS-Pegel) 2 Eingänge alternativ als Zähler-Eingang verwendbar				
	EVENT	1 ext. Trigger-Eingang (positive TTL-Logik)				
Beschaltung siehe „Beschaltung digitaler Ein-/Ausgänge“, TTL-Eingänge / TTL-Ausgänge, Seite A-6						
Zähler						
Anzahl	2 Inkrementalzähler zur Ereigniszählung.					
Zähler- und Latch-Breite				32		Bit
max. Zählfrequenz	f_{CNT}			10		MHz
EVENT-Eingang						
Flankenerkennung, pos.	V_{T+} (Low)	V_{CC} = 5V	1,65	1,9	2,15	V
Flankenerkennung, neg.	V_{T-} (High)	V_{CC} = 5V	0,75	1,0	1,25	
Schalthysterese	V_{T+} - V_{T-}		0,4	0,9		
Eingangsstrom	I_{IH}	V_I = 2,7V			20	µA
	I_{IL}	V_I = 0,4V			-50	

Analoge Ein- / Ausgänge						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Eingänge						
Anzahl	8, differentiell über Multiplexer					
Eingangswiderstand	R_i		323,4	330	336,6	kΩ
Spannungsfestigkeit	$U_{in\ max.}$	ON & OFF			±35	V
Multiplexer-Einschwing-zeit	t_{MUX}	2 LSB 16Bit		6,5 *		µs
* Bei einem Innenwiderstand der Spannungsquelle von <10Ω						
ADC 16Bit						
Konvertierungszeit	t_{conv}	Rev. A			10	µs
		Rev. B [a]			2	
Messbereich	U_{in}		-10		+9,999695	V
Differentielle Gleichtaktspannung					±2,0	V
Integrale Nichtlinearität	INL			±1	±3	LSB
Differentielle Nichtlinearität	DNL			±0,25	±0,5	
Offset	Drift[b]			±2		ppm/°C
	Fehler	abgleichbar				
Gain	Drift[b]			±20		ppm/°C
	Fehler	abgleichbar				
DAC 16 Bit						
Anzahl	2					
Ausgangsspannung	U_{out}		-10		+9,999695	V
Einschwingzeit	t_{settle}	2V-Sprung			3	µs
		FSR* (20V)			10	

Analoge Ein- / Ausgänge						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Zulässiger Strom					±5	mA
Integrale Nichtlinearität	INL				±2	LSB
Differentielle Nichtlinearität	DNL				±1	
Offset	Fehler	abgleichbar				
Gain	Fehler	abgleichbar				
* FSR = Full Scale Range						

[a] Software-Befehle verwenden die Wandlungszeit 10µs bis die schnellere Wandlung freigegeben wird.

[b] bezogen auf den gesamten Spannungsbereich (FSR)

Prozessor						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Typ	ADSP21062 (SHARC™)					
Hersteller	Analog Devices					
Taktfrequenz	f_{CLK}			40		MHz
Register-Breite				32		Bit
Interner Speicher	SRAM	für Programm		128		kByte
		für Daten		128		
Externer Speicher	SDRAM	Rev. A		8		MByte
		Rev. B		16		

CO1-Erweiterung						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Zähler						
Anzahl und Funktion	1 Vor-/Rückwärtszähler zur Ereigniszählung mit Takt / Richtung oder Vierflanken-Auswertung. Die Inkremental-Zähler der Grundversion werden ersetzt.					
Zählereingänge	2 Eingänge (A, B), alternativ als Digital-Eingänge verwendbar					
Zähler- und Latch-Breite				32		Bit

DIO1-Erweiterung						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Referenz-Quarzoszillator						
Referenzfrequenz	f_{ref}			20		MHz
Vorteiler durch 4	f_{ref} / 4			5		
Genauigkeit und Drift					100	ppm
Zähler						
Anzahl und Funktion	2 Vor-/Rückwärtszähler zur Messung von Tastverhältnis, Impuls- und Periodendauer sowie zur Ereigniszählung mit Takt / Richtung oder Vierflanken-Auswertung. Die Inkremental-Zähler der Grundversion werden ersetzt.					
Zählereingänge	Je Zähler 3 Eingänge (A/CLK, B/DIR, CLR/LATCH); Zähler programmierbar mit differentiellen oder single-ended Eingängen. Eingangsbeschaltung siehe „Beschaltung digitaler Ein-/Ausgänge“, Seite A-6.					

DIO1-Erweiterung						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Zähler- und Latch-Breite				32		Bit
Zählfrequenz	f_{CLK}	Eingang CLK		20		MHz
		Eingang A/B		5		
Digitale Ein- / Ausgänge						
Anzahl	DIO31:00	32 (in Gruppen zu 8 als Ein- oder Ausgang programmierbar)				
	EVENT	ext. Trigger-Eingang (pos. TTL-Logik)				
Beschaltung siehe „Beschaltung digitaler Ein-/Ausgänge“, TTL-Eingänge / TTL-Ausgänge, Seite A-6						
Schnittstellen						
CAN	CAN High speed, 1 Schnittstelle					
SSI	SSI-Decoder (ab Rev. B), 1 Schnittstelle					

DIO2-Erweiterung						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Referenz-Quarzoszillator						
Referenzfrequenz	f_{ref}			20		MHz
Vorteiler durch 4	f_{ref} / 4			5		
Genauigkeit und Drift					100	ppm
Zähler						
Anzahl und Funktion	2 Vor-/Rückwärtszähler zur Messung von Tastverhältnis, Impuls- und Periodendauer sowie zur Ereigniszählung mit Takt / Richtung oder Vierflanken-Auswertung. Die Inkremental-Zähler der Grundversion werden ersetzt.					
Zählereingänge	Je Zähler 3 Eingänge (A/CLK, B/DIR, CLR/LATCH), bei Zähler 1 TTL-Eingänge, bei Zähler 2 differentielle Eingänge. Eingangsbeschaltungen siehe unten: „Beschaltung digitaler Ein-/Ausgänge“.					
Zähler- und Latch-Breite				32		Bit
Zählfrequenz	f_{CLK}	Eingang CLK		20		MHz
		Eingang A/B		5		
Digitale Ein- / Ausgänge						
Anzahl	DIO31:00	32 (in Gruppen zu 8 als Ein- oder Ausgang programmierbar)				
	EVENT	ext. Trigger-Eingang (pos. TTL-Logik)				
Beschaltungsdaten siehe „Beschaltung digitaler Ein-/Ausgänge“, TTL-Eingänge / TTL-Ausgänge.						
Schnittstellen						
SSI	SSI-Decoder (ab Rev. B), 1 Schnittstelle					

DIO3-Erweiterung						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Digitale Ein- / Ausgänge						
Anzahl	DIO31:00	32 (in Gruppen zu 8 als Ein- oder Ausgang programmierbar)				
	EVENT	ext. Trigger-Eingang (pos. TTL-Logik)				
Beschaltungsdaten siehe „Beschaltung digitaler Ein-/Ausgänge“, TTL-Eingänge / TTL-Ausgänge.						

Beschaltung digitaler Ein-/Ausgänge						
TTL-Eingänge						
max. Eingangsspannung		TTL-Pegel	-0,5		+5,5	V
Logik-Eingangsspannung	V_{IH} (High)	V_{CC} = 5V	2			
	V_{IL} (Low)	V_{CC} = 5V			0,8	
Logik-Eingangsstrom	I_I	V_{CC} = 5V		±0,1	±1000	nA
differentielle Eingänge						
Differentielle Eingangs-Schwellenspannung	V_{TH}	-10V ≤ V_{CM} ≤ 13,2V	-200		+200	mV
Schalthysterese	ΔV_{TH}	-10V ≤ V_{CM} ≤ 13,2V		40		mV
Bereich der Gleichtaktspannung	V_{CM}		-10		+13,2	V
Anstieg-/Abfallgeschw. der differentiellen Spg.			0,33			V/µs
Zulässige differentielle Eingangsspannung		für jeden Eingang			±3,9	V
TTL-Ausgänge						
Logik-Ausgangsspannung	V_{OH} (High)	I_{OH}= -6mA	3,84	4,3		V
	V_{OL} (Low)	I_{OL}= +6mA		0,17	0,33	
Logik-Ausgangsstrom	I_O	je DIO-Leitung			±35	mA
	I_{TOTAL}	je DIO-Gruppe (8) über V_{CC} / GND			±70	

A.2 Hardware-Adressen - Gesamtübersicht

Adresse [HEX]	Funktion	Bit Nr.								Kommentar	Register verfügbar im Modul		
		31...16	15...6	5	4	3	2	1	0		L16	L16+ CO1	L16+ DIO1
20 40 00 00	Multiplexer auf Eingangskanal setzen (ADC 01...ADC 15)	-	-	-	-	-	n	n	n	„nnn“ binär = 0...7 dezimal, gewählter Kanal = nnn*2 + 1	x	x	x
20 40 00 10	Konvertierung starten: ADC #1	-	-	-	-	-	-	-	s	s = 0 : Konvertierung starten s = 1 : kein Einfluss	x	x	x
	Konvertierung starten: alle DAC synchron	-	-	-	-	-	s	-	-				
20 40 00 20	Konvertierungs-Status (EOC) ADC #1	-	-	-	-	-	-	-	e	e = 0 : Konvertierung beendet e = 1 : Konvertierung läuft	x	x	x
20 40 00 30	Register auslesen: ADC #1	-	x	x	x	x	x	x	x	x : Ergebnis der Konvertierung	x	x	x
20 40 00 50	Register nur beschreiben: DAC #1	-	x	x	x	x	x	x	x	x : zu wandelnder Digitalwert	x	x	x
20 40 00 60	Register nur beschreiben: DAC #2	-	x	x	x	x	x	x	x				
20 40 00 B0	Eingangs-Register Digin-05:00	-	-	x	x	x	x	x	x	x : eingelesener Digitalwert	x	x	x
20 40 00 C0	Ausgangs-Register DIGOUT-05:00	-	-	x	x	x	x	x	x	x : auszugebender Digitalwert	x	x	x
20 40 00 C4	DIGOUT Bits setzen	-	-	x	x	x	x	x	x	x = 0 : kein Einfluss x = 1 : Bit setzen	x	x	x
20 40 00 C8	DIGOUT Bits löschen	-	-	x	x	x	x	x	x	x = 0 : kein Einfluss x = 1 : Bit löschen	x	x	x
20 40 01 00	Register auslesen und Konvertierung starten: ADC #1	-	x	x	x	x	x	x	x	x : zu wandelnder Digitalwert	x	x	x
20 40 02 00	Register beschreiben und sofort Konvertierung starten: DAC #1	-	x	x	x	x	x	x	x	x : zu wandelnder Digitalwert	x	x	x
20 40 02 04	Inhalt Latch A, Zähler #1	x	x	x	x	x	x	x	x	x : Inhalt des Latch-Registers	x	x	x
20 40 02 08	Inhalt Latch B, Zähler #1 (nur DIO1)	x	x	x	x	x	x	x	x	x : Inhalt des Latch-Registers	-	-	x
20 40 02 10	Register beschreiben und sofort Konvertierung starten: DAC #2	-	x	x	x	x	x	x	x	x : zu wandelnder Digitalwert	x	x	x
20 40 02 14	Inhalt Latch A, Zähler #2	x	x	x	x	x	x	x	x	x : Inhalt des Latch-Registers	x	-	x
20 40 02 18	Inhalt Latch B, Zähler #2	x	x	x	x	x	x	x	x	x : Inhalt des Latch-Registers	-	-	x
20 40 03 00	Zähler freigeben / sperren: **Cnt_Enable**()	-	-	-	-	-	-	x	x	x = 0 : Zähler sperren x = 1 : Zähler freigeben	x	nur Bit 0	x
20 40 03 10	Zähler löschen: Cnt_Clear() *	-	-	-	-	-	-	x	x	x = 0 : kein Einfluss x = 1 : Zähler löschen	x	nur Bit 0	x
20 40 03 20	Zähler latchen: Cnt_Latch() *	-	-	-	-	-	-	x	x	x = 0 : keine Einfluss x = 1 : Zähler latchen	x	nur Bit 0	x
20 40 03 30	Zählereingang CLR oder LATCH	-	-	-	-	-	-	x	x	x = 0 : CLR-Eingang x = 1 : LATCH-Eingang	-	-	x
20 40 03 40	Impuls-/Ereigniszähler- oder Pulsbreiten-/Periodendauermessung	-	-	-	-	-	-	x	x	x = 0 :externer Takteingang x = 1 : interner Referenztakt (20MHz / 5MHz)	-	-	x
20 40 03 50	4-Flankenauswertung / CLK+DIR oder 20MHz / 5MHz Referenztakt	-	-	-	-	-	-	x	x	Cnt_Mode=0: x=0: 4-Flanken-Auswertung; x=1: CLK+DIR Cnt_Mode=1: x=0: 20MHz; x=1: 5MHz	-	-	x
20 40 04 54	Bits DIO-15:00	-	x	x	x	x	x	x	x	x = 0: Ausgang löschen x = 1: Ausgang setzen	-	-	x
20 40 04 64	Bits DIO-31:16	-	x	x	x	x	x	x	x	x = 0: Ausgang löschen x = 1: Ausgang setzen	-	-	x
20 40 04 74	Bits setzen DIO-15:00 **	-	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang setzen	-	-	x
20 40 04 84	Bits setzen DIO-31:16 **	-	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang setzen	-	-	x
20 40 04 94	Bits löschen DIO-15:00 **	-	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang löschen	-	-	x
20 40 04 A4	Bits löschen DIO-31:16 **	-	x	x	x	x	x	x	x	x = 0: kein Einfluss x = 1: Ausgang löschen	-	-	x
20 40 04 6C	DIOs konfigurieren CONF_DIO() Bit 0: DIO 07:00; Bit 1: DIO 15:08 Bit 2: DIO 23:16; Bit 3: DIO 31:24	-	-	-	-	x	x	x	x	x = 0: Gruppe als Eingang x = 1: Gruppe als Ausgang	-	-	x

* Das Register wird nach der Durchführung automatisch wieder zurückgesetzt.
** Funktion bei Eingängen ohne Einfluss

A.3 Hardware-Revisionen

Auf dem Gerät befindet sich ein Aufkleber mit der Revisionsbezeichnung. Die Unterschiede der Revisionsstände sind nachfolgend dargestellt:

Revision	Erst-ausgabe	Änderung zur Vorgänger-Version
A	1998	Erst-Version.
B1	März 2006	Überarbeitetes Layout, dennoch kompatibel zu Rev. A. Externer Speicher auf 16 MiB erweitert. Schnellere A/D-Wandlung über den Befehl **L16_MODE** verfügbar. Zusätzliche Ablaufsteuerung zur Wandlung analoger Eingänge. Zusätzlicher SSI-Schnittstelle für DIO1-Erweiterung, neue Erweiterungen DIO2 und DIO3.
B2	Aug. 2006	Neue Schnittstelle für LS-Bus.

A.4 RoHS Konformitätserklärung

Die Richtlinie 2002/95/EG der Europäischen Union zur Beschränkung und Verwendung gefährlicher Stoffe in elektrischen und elektronischen Geräten (RoHS-Richtlinie) ist am 1. Juli 2006 in Kraft getreten.

Dabei handelt es sich um folgende Substanzen:

– Blei (Pb)
– Cadmium (Cd)
– Hexavalentes Chrom (Cr VI)
– Polybromierte Biphenyle (PBB)
– Polybromierte Diphenylether (PBDE)
– Quecksilber (Hg)

Die Produktlinie *ADwin-light-16* erfüllt seit der Revision B1 die Voraussetzungen der RoHS-Richtlinie in allen gelieferten Varianten.

A.5 Baudraten für den CAN-Bus

ADwin-light 16 DIO1 besitzt Schnittstellen für den CAN-Bus. Dort können folgende Baudraten eingestellt werden:

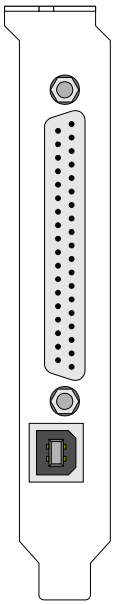
Einstellbare Baudraten [Bit/s]				
1000000.0000	888888.8889	800000.0000	727272.7273	666666.6667
615384.6154	571428.5714	533333.3333	500000.0000	470588.2353
444444.4444	421052.6316	400000.0000	380952.3810	363636.3636
347826.0870	333333.3333	320000.0000	307692.3077	296296.2963
285714.2857	266666.6667	250000.0000	242424.2424	235294.1176
222222.2222	210526.3158	205128.2051	200000.0000	190476.1905
181818.1818	177777.7778	173913.0435	166666.6667	160000.0000
156862.7451	153846.1538	148148.1481	145454.5455	142857.1429
140350.8772	133333.3333	126984.1270	125000.0000	123076.9231
121212.1212	117647.0588	115942.0290	114285.7143	111111.1111
106666.6667	105263.1579	103896.1039	102564.1026	100000.0000
98765.4321	95238.0952	94117.6471	90909.0909	88888.8889
87912.0879	86956.5217	84210.5263	83333.3333	81632.6531
80808.0808	80000.0000	78431.3725	76923.0769	76190.4762
74074.0741	72727.2727	71428.5714	70175.4386	69565.2174
68376.0684	67226.8908	66666.6667	66115.7025	64000.0000
63492.0635	62500.0000	61538.4615	60606.0606	60150.3759
59259.2593	58823.5294	57971.0145	57142.8571	55944.0559
55555.5556	54421.7687	53333.3333	52631.5789	52287.5817
51948.0519	51282.0513	50000.0000	49689.4410	49382.7160
48484.8485	47619.0476	47337.2781	47058.8235	46783.6257
45714.2857	45454.5455	44444.4444	43956.0440	43478.2609
42780.7487	42328.0423	42105.2632	41666.6667	41025.6410
40816.3265	40404.0404	40000.0000	39215.6863	38647.3430
38461.5385	38277.5120	38095.2381	37037.0370	36363.6364
36199.0950	35714.2857	35555.5556	35087.7193	34782.6087
34632.0346	34482.7586	34188.0342	33613.4454	33333.3333
33057.8512	32921.8107	32388.6640	32258.0645	32000.0000
31746.0317	31620.5534	31372.5490	31250.0000	30769.2308
30651.3410	30303.0303	30075.1880	29629.6296	29411.7647
29304.0293	29090.9091	28985.5072	28673.8351	28571.4286
28070.1754	27972.0280	27777.7778	27681.6609	27586.2069
27210.8844	27027.0270	26936.0269	26755.8528	26666.6667
26315.7895	26143.7908	25974.0260	25806.4516	25641.0256
25396.8254	25078.3699	25000.0000	24844.7205	24767.8019
24691.3580	24615.3846	24390.2439	24242.4242	24024.0240
23809.5238	23668.6391	23529.4118	23460.4106	23391.8129
23255.8140	23188.4058	22988.5057	22857.1429	22792.0228
22727.2727	22408.9636	22222.2222	22160.6648	22038.5675
21978.0220	21739.1304	21680.2168	21621.6216	21505.3763
21390.3743	21333.3333	21276.5957	21220.1592	21164.0212
21052.6316	20833.3333	20779.2208	20671.8346	20512.8205
20460.3581	20408.1633	20202.0202	20050.1253	20000.0000
19851.1166	19753.0864	19704.4335	19656.0197	19607.8431
19512.1951	19323.6715	19230.7692	19138.7560	19047.6190
18912.5296	18867.9245	18823.5294	18648.0186	18604.6512

Einstellbare Baudraten [Bit/s]				
18518.5185	18433.1797	18390.8046	18306.6362	18181.8182
18140.5896	18099.5475	18018.0180	17857.1429	17777.7778
17738.3592	17582.4176	17543.8596	17429.1939	17391.3043
17316.0173	17241.3793	17204.3011	17094.0171	17021.2766
16949.1525	16913.3192	16842.1053	16806.7227	16771.4885
16666.6667	16632.0166	16563.1470	16528.9256	16460.9053
16393.4426	16326.5306	16260.1626	16227.1805	16194.3320
16161.6162	16129.0323	16000.0000	15873.0159	15810.2767
15779.0927	15686.2745	15625.0000	15594.5419	15503.8760
15473.8878	15444.0154	15384.6154	15325.6705	15238.0952
15180.2657	15151.5152	15122.8733	15094.3396	15065.9134
15037.5940	15009.3809	14842.3006	14814.8148	14705.8824
14652.0147	14571.9490	14545.4545	14519.0563	14492.7536
14414.4144	14336.9176	14311.2701	14285.7143	14260.2496
14184.3972	14109.3474	14035.0877	13986.0140	13937.2822
13913.0435	13888.8889	13840.8304	13793.1034	13722.1269
13675.2137	13605.4422	13582.3430	13559.3220	13513.5135
13468.0135	13445.3782	13377.9264	13333.3333	13289.0365
13223.1405	13157.8947	13136.2890	13114.7541	13093.2897
13071.8954	13008.1301	12987.0130	12903.2258	12882.4477
12820.5128	12800.0000	12759.1707	12718.6010	12698.4127
12578.6164	12558.8697	12539.1850	12500.0000	12422.3602
12403.1008	12383.9009	12345.6790	12326.6564	12307.6923
12288.7865	12195.1220	12158.0547	12121.2121	12066.3650
12030.0752	12012.0120	11994.0030	11922.5037	11904.7619
11851.8519	11834.3195	11764.7059	11730.2053	11695.9064
11661.8076	11627.9070	11611.0305	11594.2029	11544.0115
11494.2529	11477.7618	11428.5714	11396.0114	11379.8009
11363.6364	11347.5177	11299.4350	11220.1964	11204.4818
11188.8112	11111.1111	11080.3324	11034.4828	11019.2837
10989.0110	10943.9124	10928.9617	10884.3537	10869.5652
10840.1084	10810.8108	10796.2213	10781.6712	10752.6882
10695.1872	10666.6667	10638.2979	10610.0796	10582.0106
10540.1845	10526.3158	10457.5163	10430.2477	10416.6667
10389.6104	10335.9173	10322.5806	10296.0103	10269.5764
10256.4103	10230.1790	10204.0816	10101.0101	10088.2724
10062.8931	10025.0627	10012.5156	10000.0000	9937.8882
9925.5583	9876.5432	9852.2167	9828.0098	9803.9216
9791.9217	9768.0098	9756.0976	9696.9697	9685.2300
9661.8357	9615.3846	9603.8415	9569.3780	9523.8095
9456.2648	9433.9623	9411.7647	9400.7051	9367.6815
9356.7251	9324.0093	9302.3256	9291.5215	9259.2593
9227.2203	9216.5899	9195.4023	9153.3181	9142.8571
9090.9091	9070.2948	9049.7738	9039.5480	9009.0090
8958.5666	8928.5714	8918.6176	8888.8889	8879.0233
8869.1796	8859.3577	8771.9298	8743.1694	8714.5969
8695.6522	8658.0087	8648.6486	8620.6897	8602.1505
8592.9108	8556.1497	8547.0085	8510.6383	8483.5631
8474.5763	8465.6085	8456.6596	8421.0526	8403.3613

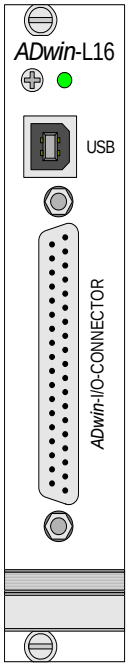
Einstellbare Baudraten [Bit/s]				
8385.7442	8333.3333	8281.5735	8264.4628	8255.9340
8230.4527	8205.1282	8196.7213	8163.2653	8130.0813
8113.5903	8105.3698	8097.1660	8088.9788	8080.8081
8064.5161	8000.0000	7976.0718	7944.3893	7936.5079
7905.1383	7843.1373	7812.5000	7804.8780	7797.2710
7774.5384	7751.9380	7736.9439	7729.4686	7714.5612
7692.3077	7662.8352	7655.5024	7619.0476	7590.1328
7575.7576	7561.4367	7547.1698	7532.9567	7518.7970
7469.6545	7441.8605	7421.1503	7407.4074	7400.5550
7386.8883	7352.9412	7326.0073	7285.9745	7272.7273
7259.5281	7246.3768	7187.7808	7168.4588	7142.8571
7136.4853	7130.1248	7111.1111	7098.4916	7092.1986
7054.6737	7017.5439	6993.0070	6956.5217	6944.4444
6926.4069	6902.5022	6896.5517	6861.0635	6820.1194
6808.5106	6802.7211	6791.1715	6779.6610	6734.0067
6688.9632	6683.3751	6666.6667	6611.5702	6578.9474
6568.1445	6562.7564	6557.3770	6535.9477	6530.6122
6493.5065	6456.8200	6451.6129	6441.2238	6410.2564
6400.0000	6379.5853	6349.2063	6324.1107	6289.3082
6274.5098	6269.5925	6250.0000	6245.1210	6211.1801
6172.8395	6163.3282	6153.8462	6144.3932	6102.2121
6060.6061	6046.8632	6037.7358	5997.0015	5961.2519
5952.3810	5925.9259	5895.3574	5865.1026	5847.9532
5818.1818	5797.1014	5772.0058	5747.1264	5714.2857
5702.0670	5681.8182	5649.7175	5614.0351	5610.0982
5555.5556	5521.0490	5517.2414	5464.4809	5434.7826
5423.7288	5376.3441	5333.3333	5291.0053	5245.9016
5208.3333	5161.2903	5079.3651	5000.0000	

A.6 Übersicht Steckverbindungen / Gehäuse

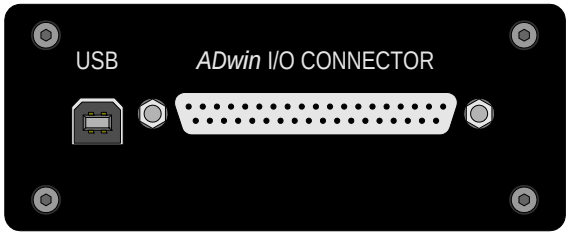
Basisversion und Erweiterung CO1 bis Rev. B1



L16-PCI



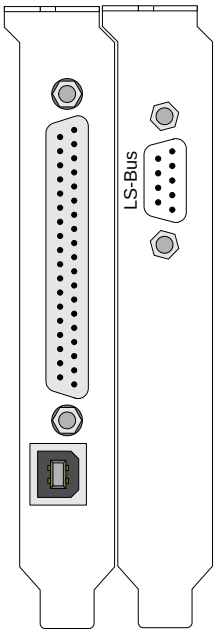
L16-EURO



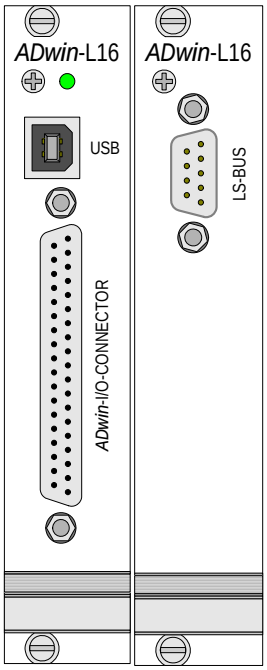
L16-EXT: Vorderseite

L16-EXT: Rückseite

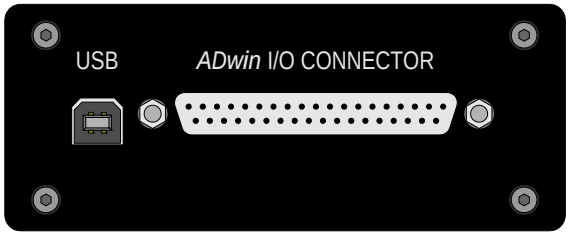
Basisversion und Erweiterung CO1 ab Rev. B2



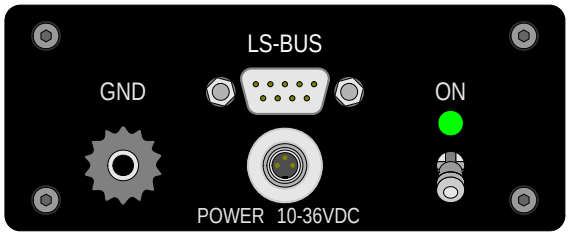
L16-PCI



L16-EURO

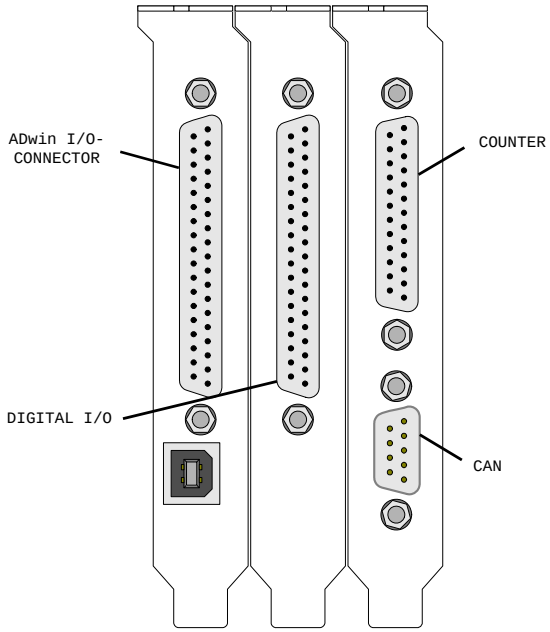


L16-EXT: Vorderseite

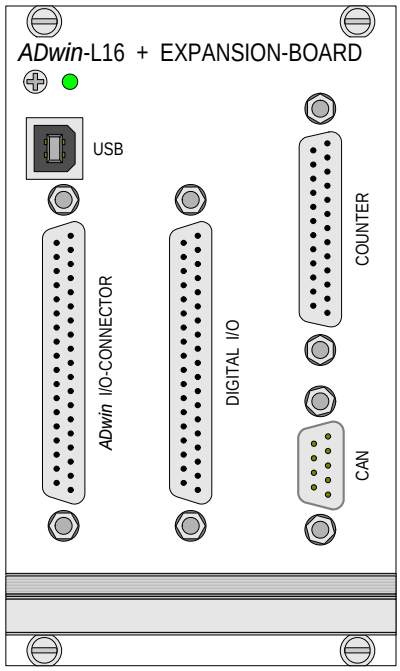


L16-EXT: Rückseite

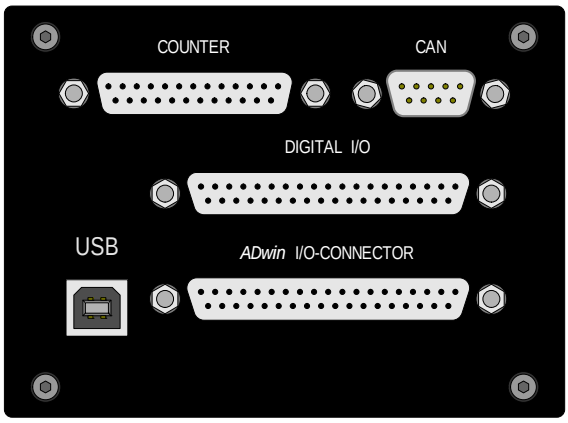
ADwin-light-16 mit DIO1-Erweiterung bis Rev. B1



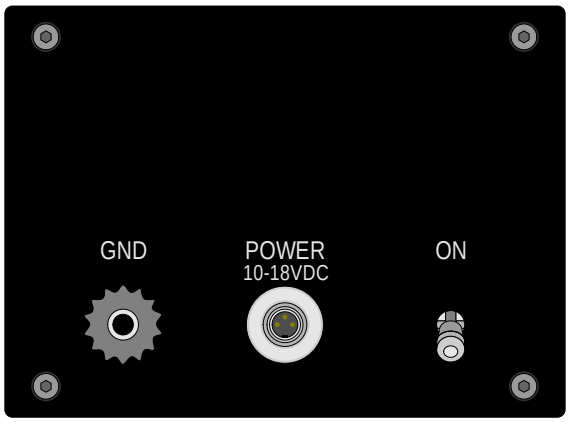
L16-DIO1-PCI



L16-DIO1-EURO

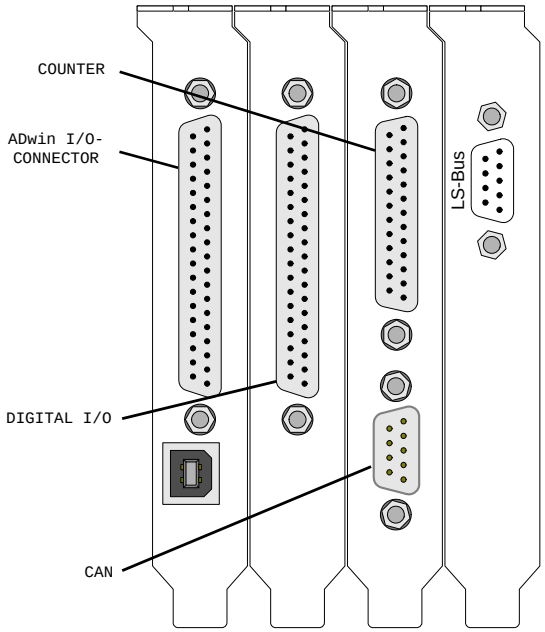


L16-DIO1-EXT: Vorderseite

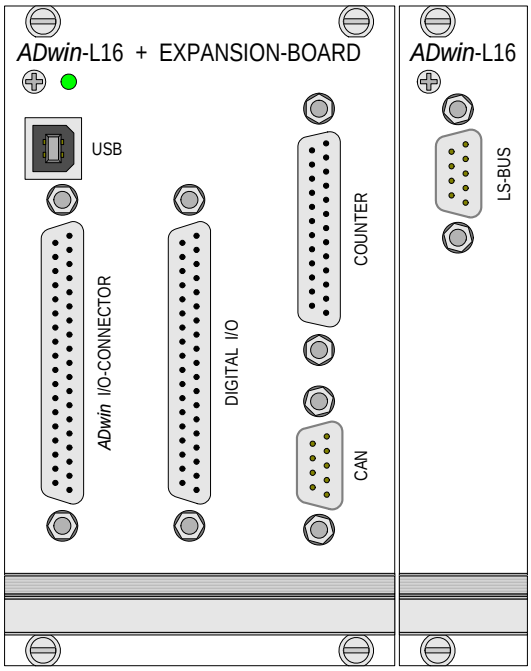


L16-DIO1-EXT: Rückseite

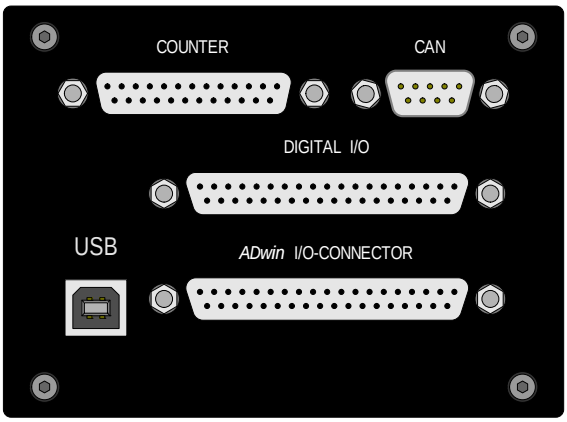
ADwin-light-16 mit DIO1-Erweiterung ab Rev. B2



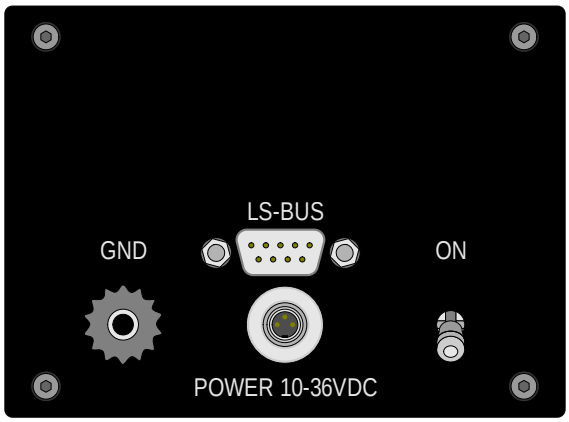
L16-DIO1-PCI



L16-DIO1-EURO

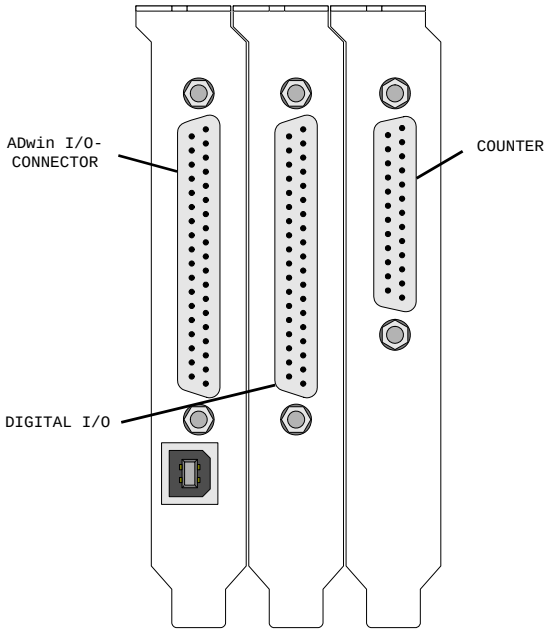


L16-DIO1-EXT: Vorderseite

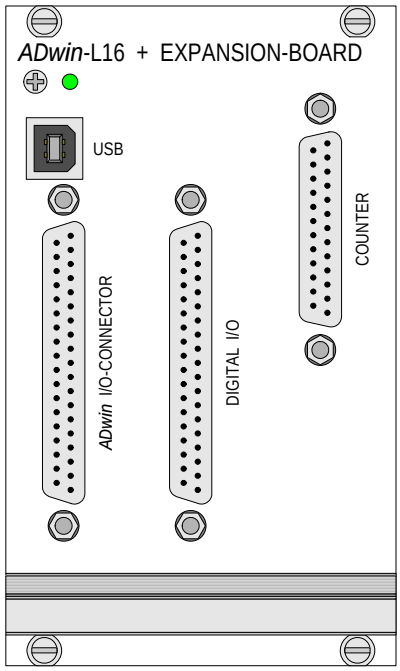


L16-DIO1-EXT: Rückseite

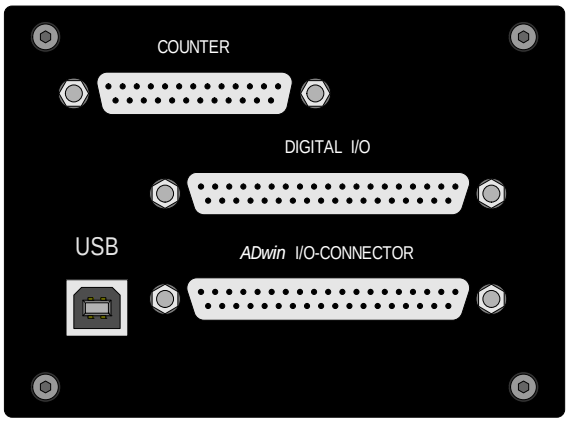
ADwin-light-16 mit DIO2-Erweiterung, Rev. B1



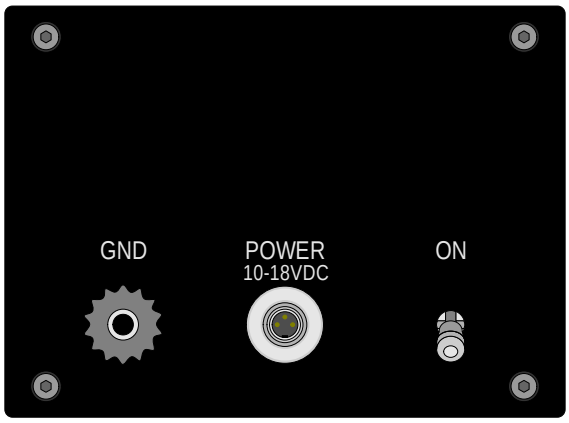
L16-DIO2-PCI



L16-DIO2-EURO

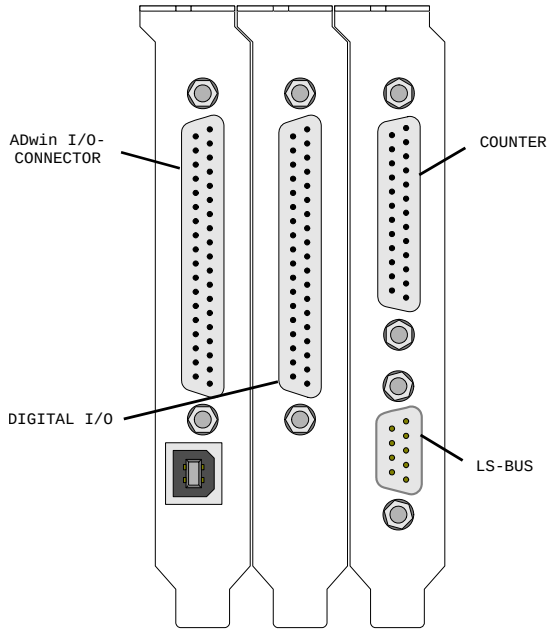


L16-DIO2-EXT: Vorderseite

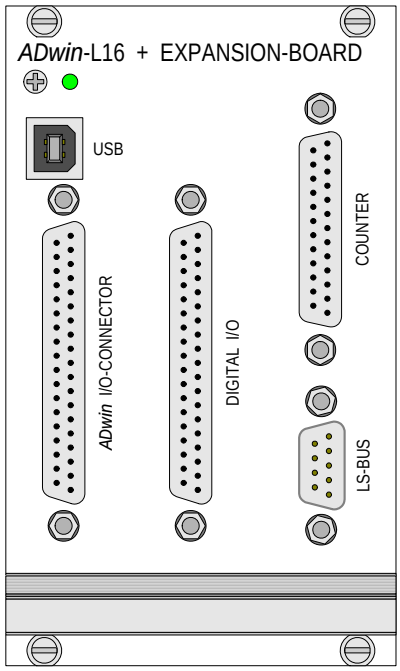


L16-DIO2-EXT: Rückseite

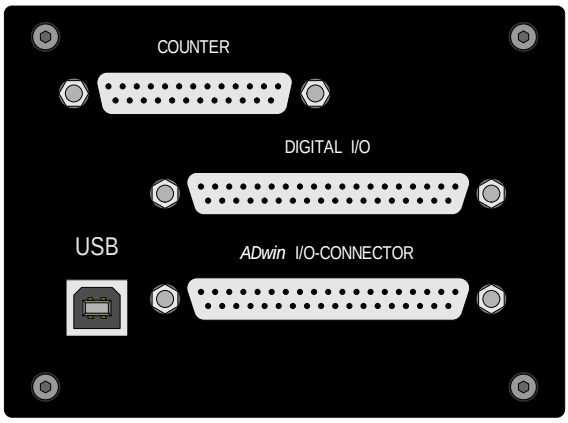
ADwin-light-16 mit DIO2-Erweiterung ab Rev. B2



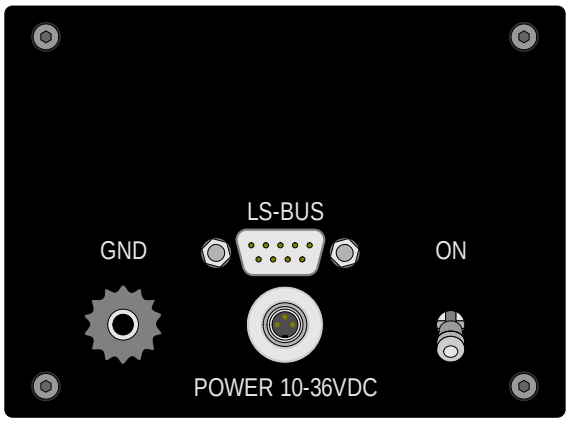
L16-DIO2-PCI



L16-DIO2-EURO

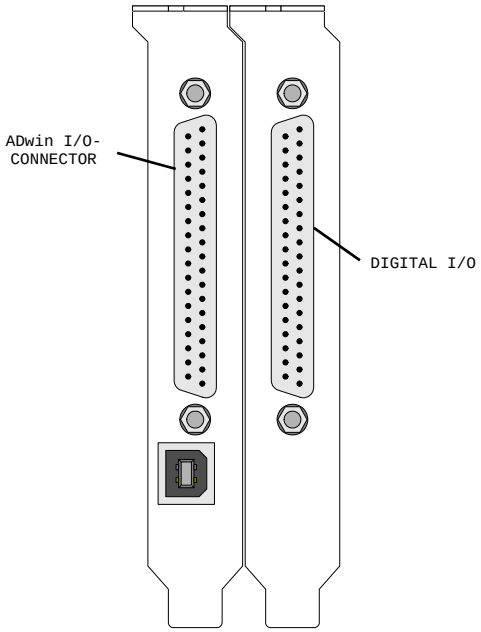


L16-DIO2-EXT: Vorderseite

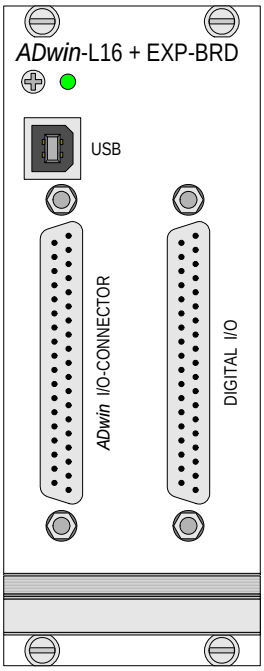


L16-DIO2-EXT: Rückseite

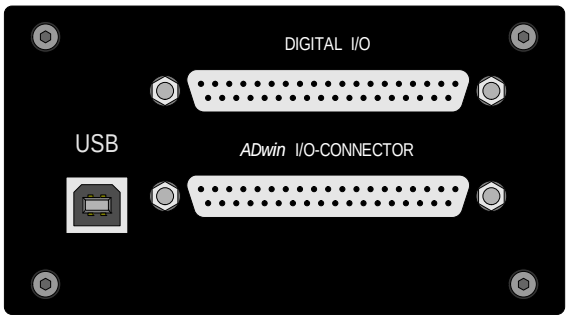
ADwin-light-16 mit DIO3-Erweiterung, Rev. B1



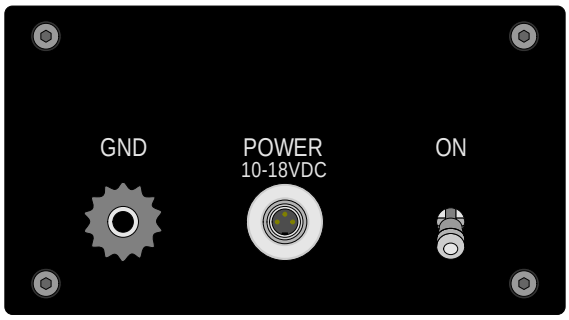
L16-DIO3-PCI



L16-DIO3-EURO

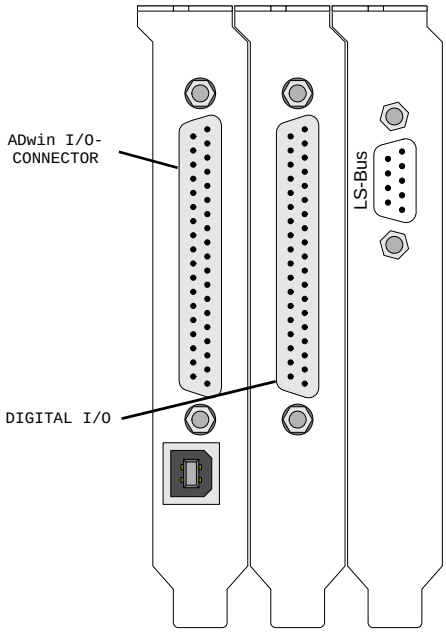


L16-DIO3-EXT: Vorderseite

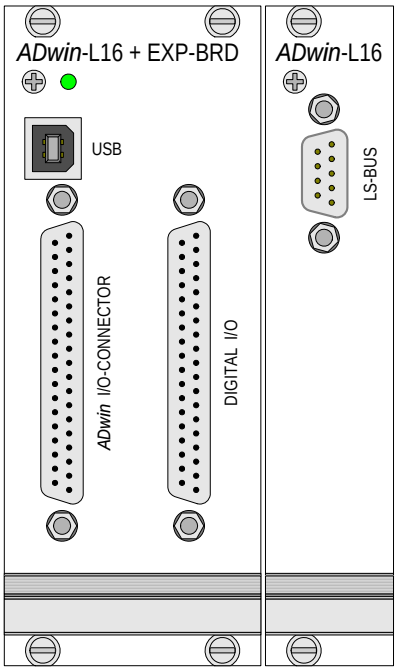


L16-DIO3-EXT: Rückseite

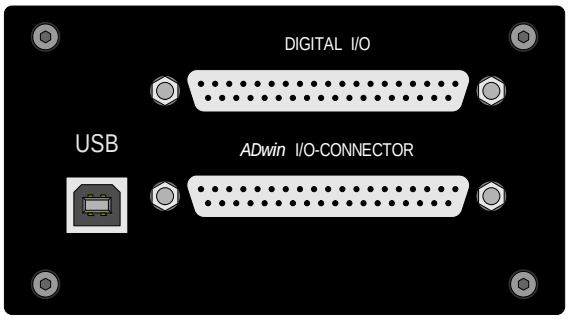
ADwin-light-16 mit DIO3-Erweiterung ab Rev. B2



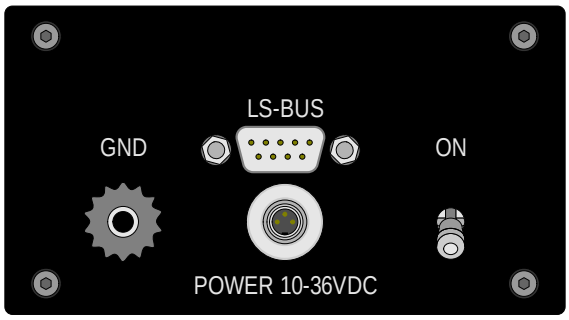
L16-DIO3-PCI



L16-DIO3-EURO



L16-DIO3-EXT: Vorderseite



L16-DIO3-EXT: Rückseite

A.7 Abbildungsverzeichnis

A.8 Index

Z