JÄGER
Computergesteuerte
Messtechnik GmbH

ADwin-Gold II

Handbuch

ADwin-Gold II, Handbuch Nov. 2019

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-Gold II*-Systems. Es wird ergänzt durch

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

  Beginnen Sie hier die Installation Ihres Systems!

– die Beschreibung des Programms *ADconfig*. Das Programm dient zum Konfigurieren der Kommunikation von der jeweiligen Schnittstelle (Interface) zur Ihrem *ADwin*-Gerät.

– das Handbuch *ADbasic*. Das Handbuch enthält die Basisbefehle für den gleichnamigen Compiler und erläutert das Funktionsprinzip von *ADwin*-Systemen.

– das Handbuch *TiCoBasic*. Das Handbuch enthält die Basisbefehle für den gleichnamigen Compiler und erläutert das Funktionsprinzip des *TiCo*-Prozessors.

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

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

Fremdprodukte werden ohne Vermerk auf mögliche Patentrechte genannt, deren Existenz nicht auszuschließen ist.

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 Millisekunden bis weit unter eine Mikrosekunde.

Systemmerkmale

Das *ADwin*-System besitzt analoge und digitale Ein- und Ausgänge, einen schnellen Prozessor (32 Bit- oder 64 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 wenigen Nanosekunden. Zu beachten ist die extrem kurze Reaktionszeit von nur 100 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 **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 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.

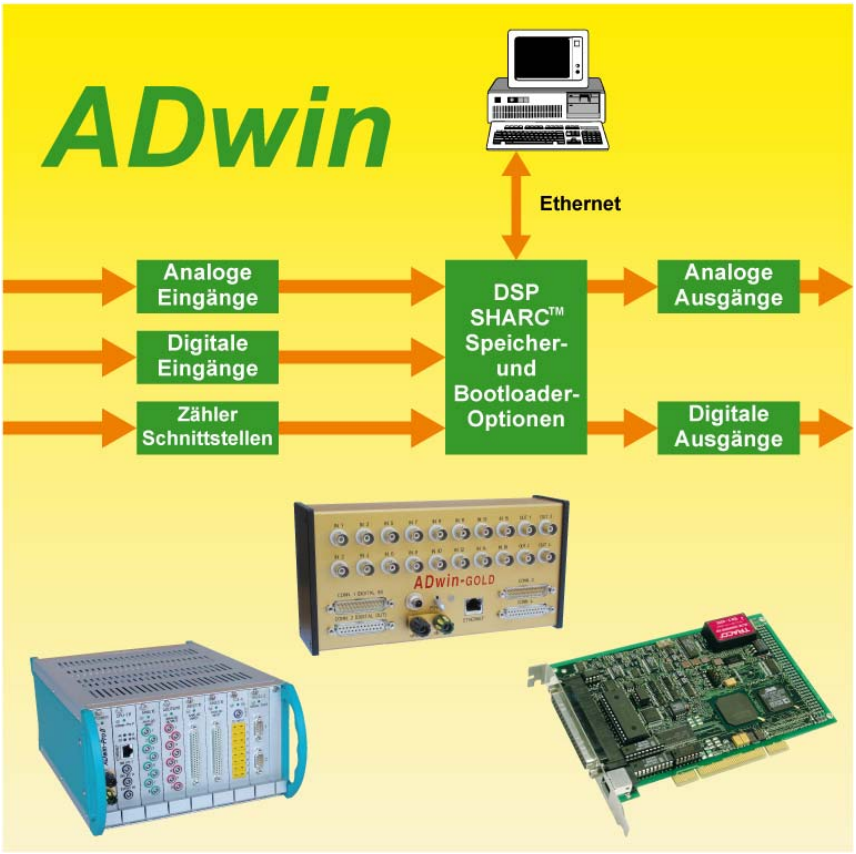


Abb. 1 – Konzept der *ADwin*-Systeme

2.2 Das *ADwin-Gold II*-System

Prozessor und Speicher

Das *ADwin-Gold II*-System besitzt den digitalen 32 Bit-Signalprozessor T11 (DSP TS101S TigerSharc) von Analog Devices mit Floating-Point- und Integer-Verarbeitung. Der Prozessor – auch als *ADwin* CPU bezeichnet – übernimmt die gesamte Messwerterfassung, Online-Verarbeitung und Signalausgabe und kann in Verbindung mit A/D-Wandlern jeden Messwert mit Abtastraten bis zu mehreren 100 Kilohertz sofort verarbeiten.

Der interne Speicher mit 3 × 256 KiB hat eine sehr kurze Zugriffszeit und nimmt das komplette *ADwin*-Betriebssystem, die *ADbasic*-Prozesse und alle Variablen auf. Der Prozessor arbeitet mit 300 MHz Taktgeschwindigkeit.

Für maximale Zugriffsgeschwindigkeiten liegen alle Ein- und Ausgänge direkt im Adressbereich des DSP. Zum Zwischenspeichern größerer Datenmengen benutzt der DSP einen externen Speicher von 256 MiB (DRAM).

***TiCo*-Zusatzprozessor**

ADwin-Gold II besitzt einen eigenständigen, frei programmierbaren Zusatzprozessor, den *TiCo*-Prozessor (*Ti*ming *Co*ntroller). Der *TiCo*-Prozessor kann auf alle Ein- und Ausgänge zugreifen und dadurch Sonderfunktionen wie Umrechnung, Kommunikationsprotokoll (SPI), Signalgenerator, Regelung etc. ausführen. Je nach Zielsetzung kann der *TiCo*-Prozessor dem Prozessor T11 zuarbeiten, indem er z. B. Daten vorverarbeitet, oder er übernimmt eine eigenständige Aufgabe.

Der *TiCo*-Prozessor ist für schnelle Reaktionszeiten und exaktes Timing optimiert. Der Prozessor arbeitet mit 50 MHz Taktgeschwindigkeit, einem Speicher von 28 KiB in PM und DM und er verarbeitet nur ganzzahlige Werte von 32 Bit Länge (Datentyp `LONG`).

Die Programmierung des *TiCo*-Prozessors mit *TiCoBasic* und die weiteren Details sind im Handbuch *TiCoBasic* beschrieben.

2 Prozessoren parallel

Tico-Prozessor und T11 (*ADwin* CPU) greifen über jeweils eigene Busse auf alle Ein- und Ausgänge sowie Schnittstellen zu und können dadurch unabhängig voneinander arbeiten. Im Funktionsschema ADwin-Gold II ist dargestellt, welche Ein- und Ausgänge zur Verfügung stehen. So kann z. B. der T11 Werte auf den DAC ausgeben, während der *Tico*-Prozessor die Zähler-Eingänge ansteuert.

Ein gleichzeitiger Zugriff von *Tico*-Prozessor und T11 auf die gleiche Peripherie ist nicht möglich.

In den folgenden Fällen sind mehrere Ein-/Ausgänge gemeinsam an T11-Bus und *Tico*-Bus angeschlossen:

– DAC-Ausgänge OUT1…OUT8

– CAN1, CAN2, COM1, COM2

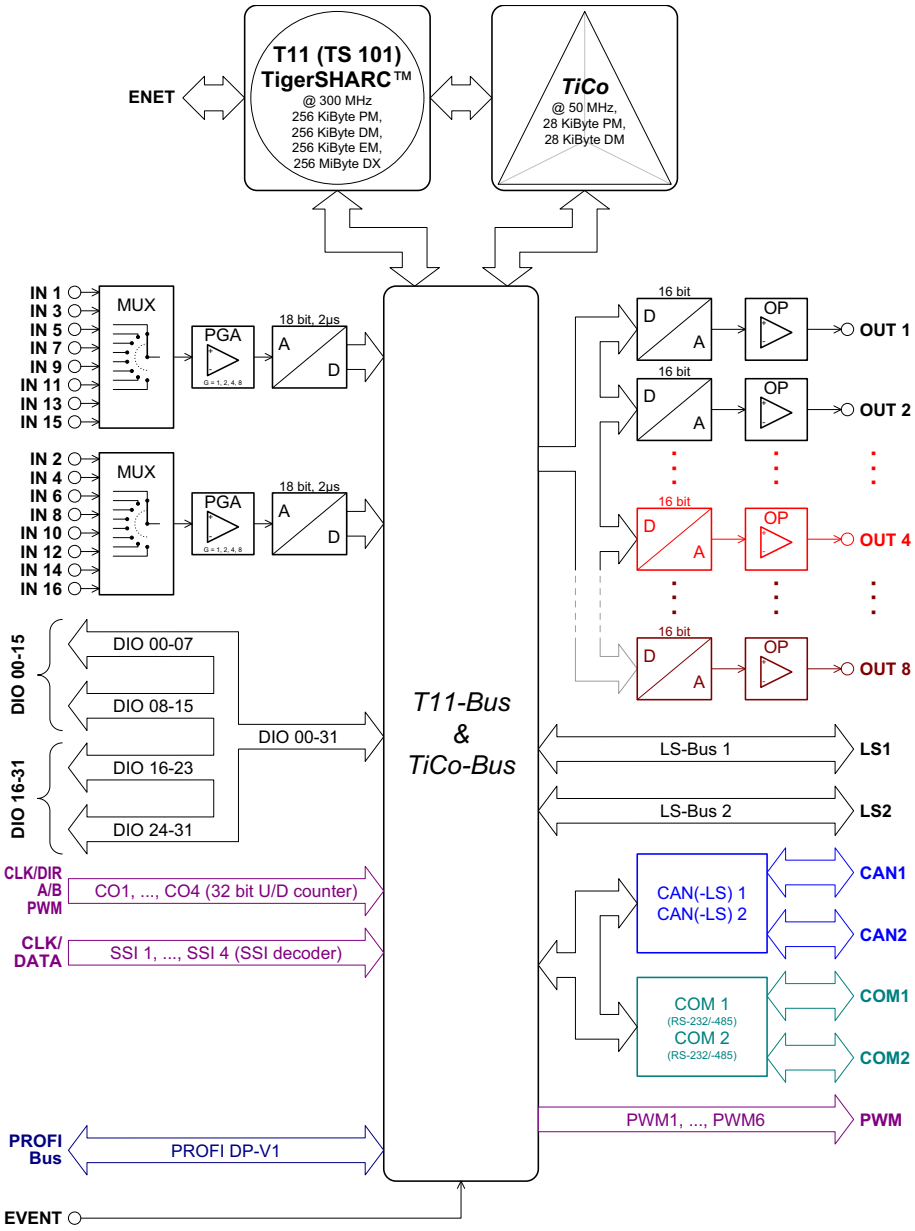


Abb. 2 – Funktionsschema *ADwin-Gold II*

Analoge Eingänge

Das System hat 2×8 analoge Eingänge mit BNC-Buchsen (alternativ: Sub-D-Buchsen), die in zwei Gruppen jeweils mit einem Multiplexer verbunden sind. Die Eingangssignale werden mit je einem 18 Bit Analog-Digital-Wandler (ADC) konvertiert (siehe Abb. 2 „Funktionsschema ADwin-Gold II“).

Analoge Ausgänge

In der Standardversion verfügt das *ADwin-Gold II* über 2 analoge Ausgänge (optional 4 oder 8) mit 16 Bit Auflösung und einem Ausgangs-Spannungsbereich von -10V…+10V. Per Software können Sie die Ausgabe der Spannung aller DAC synchronisieren.

Digitale Ein- und Ausgänge

Auf Sub-D-Anschlüssen stehen 32 digitale Ein- oder Ausgänge zur Verfügung. Sie sind in Gruppen zu jeweils 8 als Ein- oder Ausgang frei programmierbar. Die Ein- bzw. Ausgänge sind TTL-kompatibel.

Trigger-Eingang (EVENT)

Das *ADwin-Gold II*-System besitzt einen Trigger-Eingang (EVENT, siehe auch Kapitel 5.2 „Digitale Ein- und Ausgänge“). Durch ein Signal (Trigger) am Eingang können Prozesse ausgelöst werden, die dann sofort und vollständig abgearbeitet werden (siehe *ADbasic*-Handbuch, Kapitel „Struktur des ADbasic-Programms“).

Watchdog

Es kann eine Überwachung mit einem Watchdog-Zähler aktiviert werden. Der Watchdog erzeugt bei unvorhergesehenem Ausbleiben eines programmierten Signals einen Reset für den T11 und/oder *TiCo*. Das Reset-Signal kann auch auf einen Pin nach außen gegeben werden.

24 Volt-Signale

Über die beiden LS-Bus-Schnittstellen können bis zu 30 LS-Bus-Module angesteuert werden. Das LS-Bus-Modul HSM-24V ermöglicht den Anschluss von 24 V-Signalen auf 32 digitalen Kanälen.

Anbindung an den PC

Die Verbindung zwischen *ADwin-Gold II* und PC wird über die Ethernet-Schnittstelle hergestellt. So haben Sie direkten Zugriff auf Prozesse und globale Variablen des Prozessors T11.
Beachten Sie, dass nur der Prozessor T11 direkten Zugriff auf Daten des *TiCo*-Prozessors hat; ein direkter Zugriff vom PC aus ist nicht möglich.

Standard-Lieferumfang

Der Standard-Lieferumfang des *ADwin-Gold II*-Systems umfasst

- das *ADwin-Gold II*-Gerät mit Ethernet-Schnittstelle,
- ein „Cross-over“ Ethernet-Kabel vom PC zum Gold-Gerät, Länge 1,8 m,
- Dreipoliges Stromversorgungskabel mit einem Stromversorgungsstecker, Länge ca. 2 m.

  Das offene Kabelende dient zum Anschluss an die externe Stromversorgung (Selbstkonfektionierung); Kennwerte zur Stromversorgung siehe Anhang.
- *ADwin*-Software-Paket,
- Handbuch „Treiber-Installation“,
- das vorliegende Handbuch.

2.2.1 Bestelloptionen (nicht nachrüstbar)

Folgende Bestelloptionen stehen zur Verfügung:

- *Gold II-DA4 / -DA8* (Seite 21): Erweiterung auf 4 oder 8 analoge Ausgänge (single ended). Jeder Ausgang ist mit einem 16 Bit DAC ausgerüstet.
- *Gold II-CNT* (Seite 22): 4 Stück 32 Bit-Zähler, wahlweise zur Perioden-dauermessung, als Impulszähler, als Vorwärts-/Rückwärtszähler mit Takt-/Richtung, als Vier-Flanken-Auswertung für Inkremental-Enkoder. 4 Decoder zum Anschluss von Inkremental-Encodern mit SSI-Schnittstelle. 6 PWM-Ausgänge.
- *Gold II-CAN* (Seite 33): 2 CAN-Schnittstellen (beide high speed *oder* low speed) sowie 2 RSxxx-Schnittstellen (RS232, RS485).
- *Gold II-Profibus* (Seite 42): Profibus-Schnittstelle. Schließt die Optionen DeviceNet und EtherCAT aus.
- *Gold II-DeviceNet* (Seite 49): DeviceNet-Schnittstelle. Schließt die Optionen Profibus und EtherCAT aus.
- *Gold II-EtherCAT* (Seite 53): EtherCAT-Schnittstelle. Schließt die Optionen Profibus und DeviceNet aus.
- *Gold II-Storage-16* (Seite 57): Speicherkarte mit 16 GiB Speichervolumen sowie eine batteriegepufferte Echtzeituhr.
- *Gold II-Boot* (Seite 58): Flash-EPROM-Bootloader zum eigenständigen Betrieb ohne PC.

Alle Bestelloptionen sind miteinander kombiniert lieferbar; nur die Schnittstellen für Profibus, EtherCAT und DeviceNet können nicht miteinander kombiniert werden.

2.2.2 Zubehör

- *ADbasic*, Echtzeit-Entwicklungsumgebung für alle *ADwin*-Systeme.
- *TiCoBasic*, Echtzeit-Entwicklungsumgebung für *TiCo*-Prozessoren.
- *Gold II*-Pow: externes Netzteil (u.a. erforderlich für Notebook-Betrieb).
- *Gold II*-Pow-DIN: externes Netzteil für DIN-Hutschienen.
- *Gold II-Mount*: Gehäuseumbau zur Hutschienen-Montage in einem Schaltschrank mit isolierten Clipsen.
- Einzelner Stromversorgungs-Stecker für ein selbst-konfektioniertes Stromversorgungs-Kabel.
- *HSM-24V*: Hutschienenmodul für LS-Bus-Schnittstelle, 32 Digital-I/Os, 24V-Pegel, in 8er Gruppen konfigurierbar, Schraubklemmanschluss.

3 Betriebliche Umgebung

Die *ADwin-Gold II*-Elektronik ist in einem geschlossenen Aluminiumgehäuse untergebracht, und das System darf nur in diesem Zustand betrieben werden. Mit entsprechendem Zubehör ist die Unterbringung in Schaltschränken oder der mobile Betrieb (z.B. im Kfz) möglich (siehe Kapitel 2.2.2 „Zubehör“).

Erdung

Das *ADwin-Gold II*-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 über ein kurzes impedanzarmes Massband mit dem zentralen Massebezugspunkt (Erdungspunkt) Ihrer Anlage.

Galvanische Kopplung

Eine galvanische Kopplung verbindet das Massepotenzial des *ADwin-Gold II*-Geräts (und damit Ihrer Anlage) mit einem externen Massepotenzial. Spannungsunterschiede zwischen Massepotenzialen stören den Betrieb und können erhebliche Schäden verursachen. Vermeiden Sie galvanische Kopplung oder halten Sie zumindest die Spannungsunterschiede möglichst gering.

Folgende Bauteile stellen eine galvanische Verbindung her:

Bauteil	Verbindung zu
Manche Bauarten des Netzteils Gold II-Pow	Versorgungsnetz des Netzteils Gold II-Pow
Schirmung des Ethernet-Kabels	Gerät auf der Gegenseite des Ethernet-Kabels

Ob über das Netzteil Gold II-Pow tatsächlich eine galvanische Verbindung besteht, können Sie nur durch eine Messung feststellen.

Ausgleichströme ausschließen

Ausgleichsströme, die über das Gehäuse oder die Schirmung abfließen, beeinflussen das Messsignal.
Wenn Sie Ausgleichströme vermindern wollen, 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.

BNC-Kabel

Die Abschirmung von BNC-Kabeln wird üblicherweise als differentielle Masse verwendet und verliert dadurch an Schirmwirkung. Daher sind BNC-Kabel bei differentiellen Messungen Störeinflüssen ausgesetzt. Für die Signal- und Datenübertragung außerhalb des Schaltschranks ist eine Umsetzung auf Datenübertragungskabel erforderlich, die paarig verdrillt (twistet pair) und kanalweise geschirmt sind.

Schutzkleinspannung

Das *ADwin-Gold II*-System wird extern mit einer Schutz-Kleinspannung von 10V bis 28V versorgt; intern wird es mit einer Spannung von +5V und ±15V gegen GND betrieben. Es 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-Gold II ist für den Betrieb in trockenen Räumen konzipiert. Am Einbauort sollen eine Umgebungstemperatur von +5 °C … +50 °C und eine relative Luftfeuchte von 0 … 80 % (nicht kondensierend, siehe Anhang) vorhanden sein.

Gehäusetemperatur

Die Gehäusetemperatur (Oberflächentemperatur) darf auch unter extremen betrieblichen Bedingungen, z.B. im Schaltschrank oder bei direkter Sonneneinstrahlung, +60 °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-Gold II*-System an, bevor Sie nicht **folgende Schritte** durchgeführt haben:

– Führen Sie die Installation der Treiber am PC oder Notebook vollständig durch (siehe Handbuch „*ADwin*-Installation“).

  Stellen Sie die Stromversorgung für *ADwin-Gold II* her.

– Verbinden Sie das System nur mit dem PC oder Notebook (s.u.).

– Folgen Sie den Hinweisen im Kapitel 3 „Betriebliche Umgebung“.

– Lesen Sie das Kapitel 5 „Ein- und Ausgänge“ in diesem Handbuch.

– Beginnen Sie erst jetzt mit dem Anschluss von Ein- und Ausgängen.

Beachten Sie bitte, dass über die Ethernet-Leitung zwischen *ADwin-Gold II*-System und PC eine galvanische Verbindung besteht (siehe Kapitel 3, Abschnitt „Galvanische Kopplung“).

Sicherstellen der Spannungsversorgung

Achten Sie auf eine zuverlässige Spannungsversorgung, auch beim Betrieb in einem Fahrzeug (Batteriespannung).

Stromversorgung

Die Stromversorgung des ADwin-Gold II mit 12V (siehe Anhang, Technische Daten) erfolgt über den Einbaustecker links neben dem Power-Schalter (siehe Abb. 3). Schließen Sie dort einen 3-poligen Subminiatur-Rundsteckverbinder an; die Pin-Belegung entnehmen Sie bitte der folgenden Zeichnung.

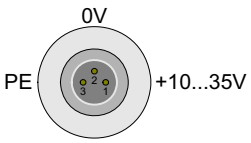


Abb. 3 – Stromversorgungsstecker (männlich)

Für die Benutzung eines separaten Netzteils benötigen Sie den o.g. Rundsteckverbinder. Sie erhalten diesen z.B. beim folgenden Hersteller unter der Bestell-Nummer 99-0406-00-03 (Serie 712):

Franz Binder GmbH + Co. elektrische Bauelemente KG
Rötelstrasse 27
D-74172 Neckarsulm
Tel.: 07132 / 325-0
www.binder-connector.de

Beim Betrieb mit einem Notebook muss die Stromversorgung durch ein separates Netzteil erfolgen (siehe Kapitel 2.2.2 auf Seite 7). Bitte beachten Sie, dass das Netzteil ausreichend dimensioniert ist.

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.

PC anschließen

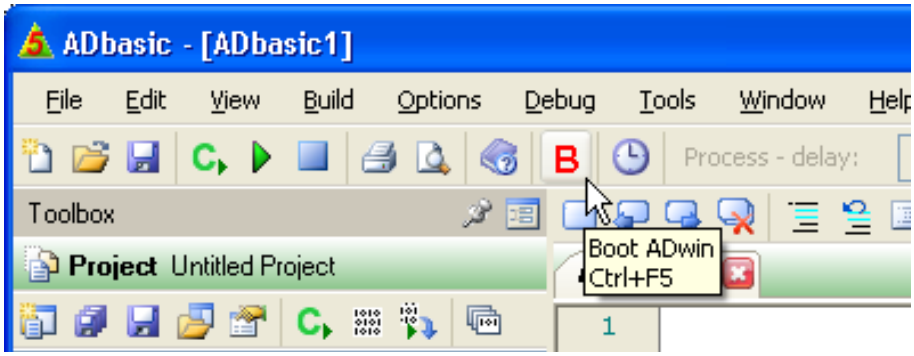
Wenn Sie die *ADwin*-Treiber-Installation und die Einstellungen im *ADbasic*-Menü „`Options\Compiler`“ abgeschlossen haben, schließen Sie jetzt das Ethernet-Datenübertragungs- und das Stromversorgungs-Kabel an. Starten Sie anschließend Ihren Rechner.

Einschalten

Um versehentliches Ausschalten zu verhindern, besitzt der Ein-/Ausschalt-Hebel eine Umschaltsperre. Ziehen Sie zum Einschalten den Hebel leicht heraus (ca. 1,5mm) und legen ihn in Richtung „`Power`“ um. Damit ist das Gerät eingeschaltet und die LED leuchtet kurz rot und dann grün auf.

Booten

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



Das Blinken der LED (jetzt grün) und die Anzeige in der Statuszeile: „`ADwin is booted`“ zeigen an, dass das Betriebssystem richtig geladen ist und *ADbasic* eine Verbindung zum *ADwin*-System herstellen kann (wenn nicht, überprüfen Sie zuerst die Anschlüsse).

Programme mit *ADbasic*

Die Programmierung von *ADwin*-Systemen ist im *ADbasic*-Handbuch ausführlich beschrieben.

Beginnen Sie mit Programmbeispielen aus dem *ADbasic*-Tutorial.

5 Ein- und Ausgänge

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 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 Stecker oder Buchse des *ADwin-Gold II* auf einen definierten Pegel (z.B. GND). Schließen Sie keine Kabel mit offenem Ende an die Eingänge an; dies kann Störimpulse an den Eingängen verursachen.

ADwin-Gold II stellt mehrere Pins mit einer Spannung von +5V zur Verfügung. Die maximale Stromstärke von 100mA gilt für alle Pins gemeinsam.

Die Ein- und Ausgänge der Basisversion *ADwin-Gold II* sind auf den folgenden Seiten beschrieben:

- 16 analoge Eingänge über 2 Multiplexer (Seite 12)
- 2 analoge Ausgänge (Seite 14)
- 32 digitale Ein- oder Ausgänge (Seite 17)
- Watchdog-Ausgang (Seite 18)
- 2 LS-Bus-Schnittstellen (Seite 19); Anschluss von bis zu 30 LS-Bus-Modulen. Das Modul HSM-24V ermöglicht den Anschluss von 24V-Signalen auf 32 digitalen Kanälen.

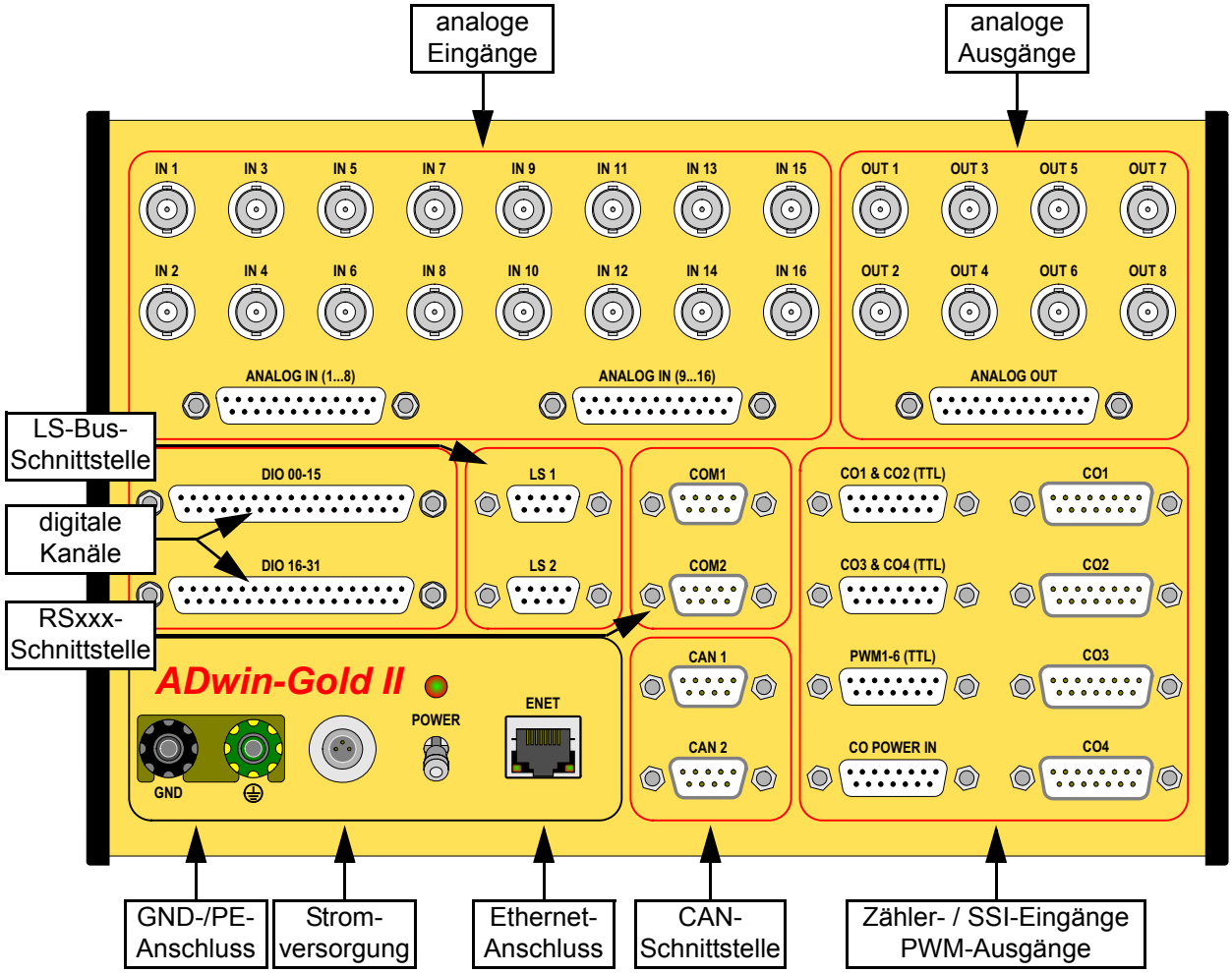


Abb. 4 – Übersichtsbild *ADwin-Gold II*

5.1 Analoge Ein- und Ausgänge

Für störungsfreien Betrieb der BNC-Anschlüsse sind isolierte BNC-Stecker erforderlich. Es besteht ansonsten die Gefahr von Schäden durch elektrostatische Entladungen und Kurzschlüssen an den Eingängen. Das gilt vor allem bei Verwendung von nicht isolierten BNC-T-Stücken.

Das *ADwin-Gold II*-Gerät muss geerdet werden, um Messungen störungsfrei durchführen zu können. Verbinden Sie dazu die GND-Buchse über ein impedanzarmes Masseband mit dem zentralen Erdungspunkt Ihrer Anlage. Über Ethernet oder das Netzteil Gold II-Pow besteht eine galvanische Verbindung zwischen *ADwin-Gold II*-System und PC, siehe Galvanische Kopplung (Seite 8).

Neben der Beschreibung der Ein- und Ausgänge finden Sie nachfolgend Hinweise zur Umrechnung zwischen Digits und analogen Spannungswerten und zur Eingangsbeschaltung der analogen Eingänge.

Standardbefehle

Für eine schnelle und einfache Programmierung gibt es im Compiler *ADbasic* Standardbefehle, die **einfaches Messen bzw. Ausgeben von Daten** ermöglichen (siehe `ADC` oder `DAC`, Seite 71ff). Verwenden Sie andere Befehle (z.B. für direkten Registerzugriff) erst, wenn extrem zeitkritische oder besondere Aufgaben es erfordern (siehe auch Handbuch *ADbasic*).

5.1.1 Analoge Eingänge

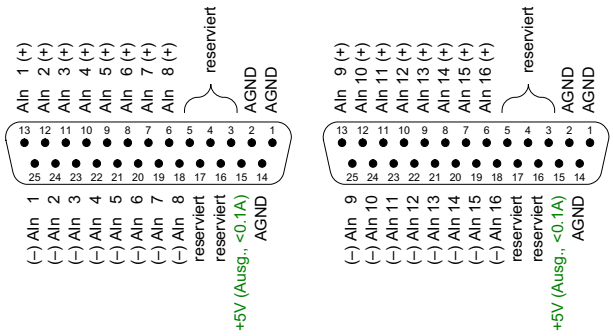
2 Multiplexer

ADwin-Gold II hat 16 analoge Eingänge IN1 … IN16. Die Eingänge mit ungeraden Zahlen (1, 3, … 15) sind dem Multiplexer 1, diejenigen mit geraden Zahlen (2, 4, … 16) sind dem Multiplexer 2 zugeordnet. Der Ausgang jedes Multiplexers ist mit einem 18 Bit-ADC verbunden (siehe auch Funktionsschema ADwin-Gold II, Seite 5).

Differentiell

Die analogen Eingänge sind differentiell. Für jeden Messkanal sind je ein Plus- und ein Minuseingang vorhanden, zwischen denen die Spannungsdifferenz gemessen wird (jedoch nicht potenzialfrei). Für jeden Kanal müssen Plus- und Minuseingang angeschlossen werden.

Die Eingänge sind mit BNC-Buchsen bestückt, die in zwei Reihen angeordnet sind; darunter sind die Eingänge auf die Sub-D-Buchse ANALOG IN gelegt. Bei den BNC-Buchsen ist der Innenleiter der Plus-Eingang, der Außenleiter der Minus-Eingang.



ANALOG IN 1…8 ANALOG IN 9…16

Abb. 5 – Pinbelegung der Analogeingänge auf Sub-D-Buchsen

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

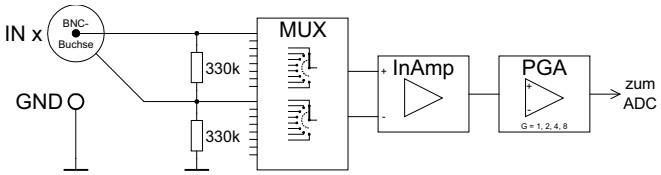


Abb. 6 – Eingangsbeschaltung eines analogen Eingangs

18 Bit-Messung

Signale an den Multiplexer-Ausgängen werden mit dem 18 Bit Analog-Digital-Wandler (ADC) konvertiert (siehe Abb. 2 „Funktionsschema ADwin-Gold II“). Der 18 Bit-ADC arbeitet schnell (max. 2 µs) und sehr genau (76 µV).

Es gibt zwei Methoden, Messwerte zu wandeln, die alternativ einsetzbar sind:

– Einzelwertmessung: Die Wandlung wird zu einem definierten Zeitpunkt gestartet und der Messwert nach der entsprechenden Zeit zurückgegeben. Der Wert kann 16 Bit oder 18 Bit Auflösung haben.

– Dauermessung: Eine Ablaufsteuerung wandelt kontinuierlich 16 Bit-Messwerte. Man kann den jeweils aktuellsten Wert ohne Wartezeit lesen, kennt aber den genauen Zeitpunkt der Messung nicht.

Einzelwertmessung

Der Befehl **ADC**() führt mit einem der ADCs eine **komplette Messung** auf einem analogen Eingang durch (siehe Seite 74) und liefert einen 16 Bit-Wert zurück. Der Befehl berücksichtigt z. B. die Einschwingzeit des Multiplexers und stellt einwandfreie Messungen sicher.

Messwerte mit der Auflösung 18 Bit sind mit dem Befehl **ADC24** möglich (siehe Seite 76); der Wert wird im Format 24 Bit zurückgegeben (siehe Seite 16).

Beide Befehle arbeiten mit dem 18 Bit-ADC, nur die Rückgabewerte haben unterschiedliche Formate.

Dauermessung mit der Ablaufsteuerung

ADwin-Gold II beinhaltet für jeden ADC eine Ablaufsteuerung, die Messwerte an ausgewählten Eingangskanälen des ADCs nacheinander einlesen kann. Dadurch kann der Prozessor stark entlastet werden, der die fertig gewandelten Werte nur noch aus dem Zwischenspeicher der Ablaufsteuerung ausliest. Die Ablaufsteuerungen der beiden ADCs arbeiten voneinander unabhängig.

Programmieren

Befehle zur Programmierung der analogen Eingänge sind ab Seite 74 beschrieben. Die Befehle sind in den Include-Dateien `<ADwinGoldII.inc>` für *ADbasic* und `<GoldIITiCo.inc>` für *TiCoBasic* enthalten und werden auch in der Online-Hilfe erläutert.

Bereich	Befehle
vollständige Messung durchführen	**ADC**, **ADC24**
Messung in Teilschritten durchführen (siehe Kapitel 5.5)	**Set_Mux1**, **Set_Mux2** **Start_Conv**, **Wait_EOC** **Read_ADC**
Ablaufsteuerung initialisieren und starten	**Seq_Mode** **Seq_Select** **Seq_Set_Delay** **Seq_Set_Gain** **Seq_Start**
Daten aus der Ablaufsteuerung abrufen	**Seq_Read** **Seq_Status**

Achten Sie auf einen möglichst geringen Ausgangswiderstand Ihrer Signalquelle (für die Eingangssignale), denn dieser kann die Messgenauigkeit beeinflussen. Falls dies nicht möglich ist:

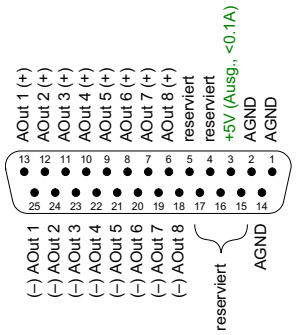
– Abhängig vom Ausgangswiderstand wird ein linearer Messfehler erzeugt (je 10 Ω etwa 1 Digit).

Sie können dies ausgleichen, indem Sie den Messwert mit einem entsprechenden Faktor multiplizieren und dadurch sozusagen „nachkalibrieren“.

– Ab etwa 3kΩ aufwärts verlängert sich zusätzlich die Einschwingzeit des Multiplexers.
Die im Standard-Befehl **ADC** definierte Einschwingzeit ist dann zu kurz, so dass zu früh ungenaue Werte abgerufen werden. Verwenden Sie für diesen Fall die in Kapitel 5.5 beschriebenen Befehle.

5.1.2 Analoge Ausgänge

ADwin-Gold II hat in der Basisversion 2 analoge Ausgänge (OUT1, OUT2) mit BNC-Buchsen; darunter liegen die Ausgänge auf der Sub-D-Buchse ANALOG OUT (siehe Abb. 7). Jedem Ausgang ist ein eigener Digital-Analog-Wandler (DAC) zugeordnet.



ANALOG OUT

Abb. 7 – Pinbelegung der Analogausgänge auf Sub-D-Buchsen

Zusätzliche Ausgänge siehe Kapitel 6 „DA-Erweiterung“.

Programmieren

Befehle zur Programmierung der analogen Ausgänge sind ab Seite 71 beschrieben. Die Befehle sind in den Include-Dateien `<ADwinGoldII.inc>` für *ADbasic* und `<GoldIITiCo.inc>` für *TiCoBasic* enthalten und werden auch in der Online-Hilfe erläutert.

Bereich	Befehle
vollständige Werteausgabe durchführen	**DAC**
Werteausgabe in Teilschritten durchführen	**Write_DAC** **Start_DAC**

Der Standardbefehl **DAC**`(Nummer,Wert)` prüft jeden Wert auf die Über- und Unterschreitung des Wertebereiches (0…65535). Liegt der Wert innerhalb dieses Bereiches, wird der angegebene Wert auf dem Ausgang `Nummer` ausgegeben. Liegt er außerhalb, wird der Maximal- bzw. Minimalwert ausgegeben.

5.1.3 Berechnungsgrundlagen

Spannungsbereich

ADwin-Gold II arbeitet bei den analogen Ein- und Ausgängen mit einem Spannungsbereich von –10V bis +10V ($\hat{=}$ bipolar 10V).

Zuordnung von Digits zu 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 65536 Digit, genau 10 Volt, liegt gerade *außerhalb* des Messbereichs, womit sich für einen 16 Bit-Messwert ein maximaler Spannungswert von 9,999695 Volt ergibt.

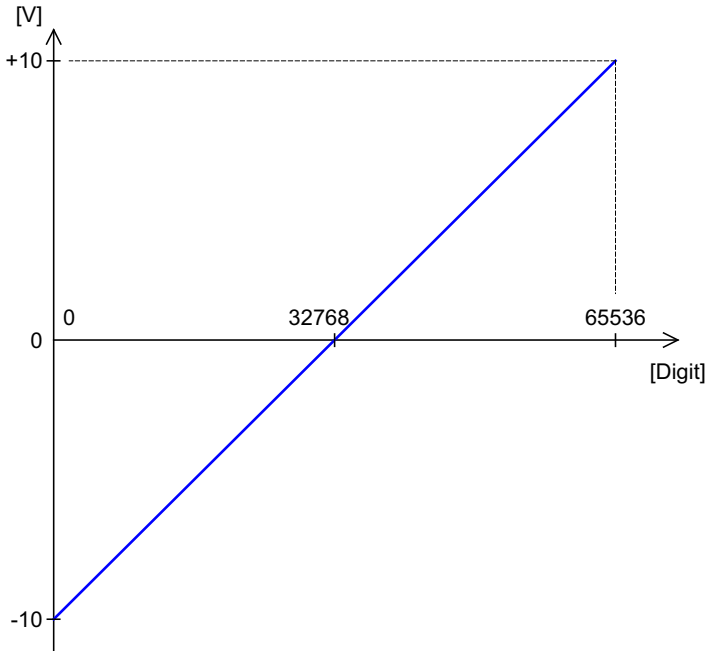


Abb. 8 – Nullpunktverschiebung bei Standardeinstellung bipolar 10 Volt

Nullpunktverschiebung U_{OFF}

Die bipolare Einstellung führt zu einer Nullpunktverschiebung, die im folgenden auch als Offset U_{OFF} bezeichnet wird.
Beim Spannungsbereich –10 V … +10 V gilt:

$U_{OFF} = -10\,V$

Verstärkungsfaktor k_V

ADwin-Gold II besitzt einen programmierbaren Verstärker (PGA), mit dem Sie die Eingangsspannung um die Faktoren 1, 2, 4 oder 8 verstärken können. Gleichzeitig verkleinert sich damit der Messbereich um den jeweiligen Verstärkungsfaktor k_V (siehe Anhang „Technische Daten“).

Beachten Sie bei Anwendungen mit $k_V > 1$, dass auch die Störsignale entsprechend mit verstärkt werden.

Quantisierungsstufe U_{LSB}

Die Quantisierungsstufe U_{LSB} ist die kleinste digital darstellbare Spannungsdifferenz und ist gleich der Spannung des niederwertigsten Bit (Least Significant Bit, LSB). Sie ist je nach Messwertauflösung unterschiedlich.

Der Messwert des 18 Bit-ADC kann im Format 16 Bit oder 24 Bit zurückgegeben werden. Der DAC verarbeitet Werte mit 16 Bit:

– 16 Bit-Format: Der Messwert steht in den unteren 16 Bits des Rückgabewerts, die oberen 16 Bits sind immer Null.
  $U_{LSB} = 20\,V / 2^{16} = 305{,}175\,\mu V$

  Das gleiche gilt für einen auszugebenden DAC-Wert.

– 24 Bit-Format: In dem 24 Bit-Wert ist der Messwert in den Bits 6…23 enthalten, d.h. der Messwert wird um 6 Bits nach links verschoben, die Bits 0…5 sind immer Null.
  $U_{LSB} = 20\,V / 2^{24} = 1{,}192\,\mu V$

Bit-Nr.	31…24	23…16	15…6	05…00
Inhalt	0	18-Bit Messwert in den Bits 6…23.		0
	0	0	16 Bit-Messwert in den Bits 0…15	

Abb. 9 – Ablage der ADC/DAC-Bits im Speicher

Umrechnung Digit in Spannung

DAC

Für einen DAC gilt:

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

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

ADC

Für einen ADC gilt:

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

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

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):

INL

– Die integrale Nicht-Linearität (INL) beschreibt die maximale Abweichung von der Geraden über den gesamten Eingangsspannungsbereich (siehe Abb. 8, Seite 15).

DNL

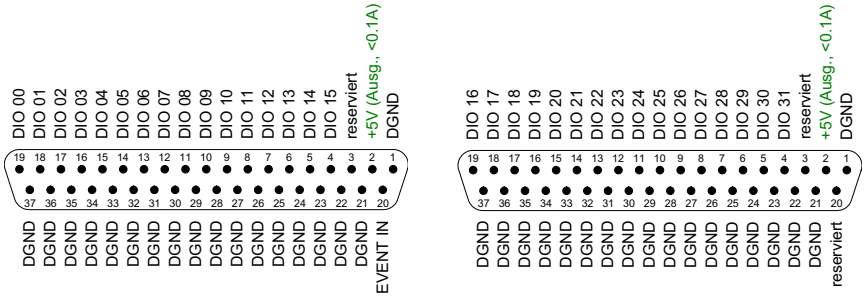
– Die differentielle Nicht-Linearität (DNL) beschreibt die maximale Abweichung von der Breite einer Quantisierungsstufe.

5.2 Digitale Ein- und Ausgänge

Digitale Ein- / Ausgänge

Auf zwei 37-poligen Sub-D-Buchsen stehen 32 digitale Ein- oder Ausgänge (DIO 00…DIO 31) zur Verfügung. Sie sind in Gruppen zu jeweils 8 als Ein- oder Ausgang programmierbar.

Nach dem Einschalten des Gerätes sind alle 4 Anschlussgruppen als Eingang konfiguriert.



DIO 00-15 DIO 16-31

Abb. 10 – Pinbelegung der Digitalkanäle

Die digitalen Eingänge sind TTL-kompatibel und gegen Überspannung nicht geschützt. Jeder Digitaleingang DIO 00…DIO 15 besitzt einen internen Pull-down-Widerstand (2,2 kΩ).
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-Gold II*-System besitzt einen externen Trigger-Eingang (EVENT) am Pin 20 der Sub-D-Buchse DIO 00-15 (siehe Abb. 10).

Ein externes Signal (Trigger) mit steigender Flanke an diesem Eingang kann Prozesse aufrufen, die sofort und vollständig abgearbeitet werden (siehe auch *ADbasic*-Handbuch, Kapitel: „Programmaufbau“). Der Event-Eingang besitzt einen internen Pull-down-Widerstand (4,7 kΩ).

Neben dem externen Trigger-Eingang gibt es weitere Quellen für Event-Signale (siehe Seite 63). Es darf nur eine der Quellen aktiv sein.

Flankenüberwachung

ADwin-Gold II kann automatisch die Flanken an ausgewählten Eingangskanälen überwachen. Hierzu stehen 2 Möglichkeiten zur Verfügung:

– Exaktes Protokoll aller Änderungen: Eine Flankenüberwachung prüft alle 10 ns, ob sich an den festgelegten Eingangskanälen ein Pegel geändert hat bzw. ob eine Flanke aufgetreten ist. Mit jeder Änderung wird ein Wertepaar in ein FIFO-Feld kopiert:
  - Wert 1 enthält den Pegelzustand aller Kanäle als Bitmuster.
  - Wert 2 enthält einen Zeitstempel, den aktuellen Stand eines 100 MHz-Zählers.

  Das FIFO-Feld kann maximal 511 Wertepaare (Pegelzustand und Zeitstempel) enthalten. Auf diese Weise wird ein zeitlich exaktes Bild der Pegeländerungen gespeichert. Die FIFO-Daten können ausgelesen und weiter verarbeitet werden.

– Flanken registrieren: Wenn eine positive oder eine negative Flanke an einem Eingang auftritt, wird in einem Zwischenspeicher das entsprechende Bit des Eingangskanals gesetzt. Die Anzahl und der Zeitpunkt der Flanken werden nicht festgehalten.

  Die Bits können abgefragt werden, um zu sehen, an welchen Eingängen eine positive oder negative Flanke aufgetreten ist. Eine Abfrage setzt die Bits auf Null zurück.

Programmieren

Befehle zur Programmierung der digitalen Ein-/Ausgänge sind ab Seite 96 beschrieben. Die Befehle sind in den Include-Dateien `<ADwinGoldII.inc>` für *ADbasic* und `<GoldIITiCo.inc>` für *TiCoBasic* enthalten und werden auch in der Online-Hilfe erläutert.

Funktion	Befehle
Konfigurieren	**Conf_DIO**
Eingangswerte lesen	**Digin_Long**, **Digin_Word1**, **Digin_Word2**
Ausgangswerte setzen	**Digout**, **Digout_Bits**, **Digout_Long**, **Digout_Word1**, **Digout_Word2**
Ausgangswerte rücklesen	**Get_Digout_Long**, **Get_Digout_Word1**, **Get_Digout_Word2**
Flanken an Digitaleingängen überwachen	**Digin_Fifo_Clear**, **Digin_Fifo_Enable** **Digin_Fifo_Full**, **Digin_Fifo_Read** **Digin_Fifo_Read_Timer**
Aufgetretene Flanken abfragen	**Digin_Edge**

Mit **Conf_DIO**(12) konfigurieren Sie DIO 15:00 als digitale Eingänge und DIO 31:16 als digitale Ausgänge (Pinbelegung siehe oben).

Nach dem Einschalten des Gerätes sind alle 4 Anschlussgruppen als Eingang konfiguriert; dies entspricht dem Befehl **Conf_DIO**(0). Die folgende Tabelle zeigt, wie Ein- und Ausgänge (IN, OUT) konfiguriert werden, wenn Sie den Wert der ersten Spalte als Befehlsargument verwenden.

Conf_DIO()	DIO31:24	DIO23:16	DIO15:08	DIO07:00
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
19	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

Abb. 11 – Übersicht der Konfigurationen mit **Conf_DIO**

5.3 Watchdog

Die Funktion von *ADwin-Gold II* kann mit einem Watchdog-Zähler überwacht werden. Wenn der Watchdog-Zähler aktiv ist, dekrementiert er seinen Zählerstand kontinuierlich. Sobald der Zählerstand 0 (Null) erreicht, nimmt das Sys-

tem eine Fehlfunktion an und löst eine Kombination vorher konfigurierter Notfunktionen aus:

– Prozessor T11 stoppen.
– *TiCo*-Prozessor stoppen.
– Ausgang `Watchdog Out` auf TTL-Pegel low setzen.

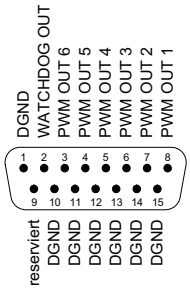
  Das Signal an `Watchdog Out` dient dazu, eine vom *ADwin*-System gesteuerte Anlage in einen sicheren Betriebszustand zu fahren. Beispielsweise kann das Watchdog-Signal die Freigabe für die Verstärkung (Endstufe) des Steuer- oder Regelsignals sperren.
– Ausgänge `OUT1`…`OUT8` und `DIO00`…`DIO32` auf TTL-Pegel low setzen.

Setzen Sie den Watchdog-Zähler während des Zählvorgangs mindestens einmal zurück auf den Startwert, um die Funktion des Systems zu gewährleisten.

Watchdog-Ausgang

Auf der 15-poligen Sub-D-Buchse `PWM1-6 (TTL)` steht der digitale Ausgang `Watchdog Out` zur Verfügung.

Nach dem Einschalten ist der Ausgang `Watchdog Out` auf TTL-Pegel Low gesetzt. Wenn der Watchdog-Zähler deaktiviert ist, kann der Ausgang `Watchdog Out` frei programmiert werden.



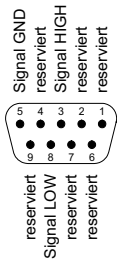
Programmieren

Befehle zur Programmierung des Watchdog-Zählers sind ab Seite 66 beschrieben. Die Befehle sind in den Include-Dateien `<ADwinGoldII.inc>` für *ADbasic* und `<GoldIITiCo.inc>` für *TiCoBasic* enthalten und werden auch in der Online-Hilfe erläutert.

Funktion	Befehle
Initialisieren	`Watchdog_Init`
Rücksetzen	`Watchdog_Reset`
Ausgangswert setzen	`Watchdog_Standby_Value`
Status lesen	`Watchdog_Status`

5.4 LS-Bus

ADwin-Gold II stellt zwei Anschlüsse für den LS-Bus auf 9-poligen Sub-D-Verbindern (Buchse) LS1 und LS2 zur Verfügung. Die Pinbelegung ist jeweils gleich.



Der LS-Bus (Low Speed) ist ein bidirektionaler, serieller Bus mit 5 MHz Taktrate. Der Bus ist eine Eigenentwicklung für den Anschluss externer Module. Es steht 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 für die Module am LS-Bus sind im Handbuch des LS-Bus-Moduls beschrieben und in der Online-Hilfe.

5.5 Zeitkritische Aufgaben

Für zeitkritische Aufgaben kann es sinnvoll sein, die Standardbefehle `ADC()` oder `ADC24()` durch mehrere Einzelbefehle zu ersetzen.

`ADC()` und `ADC24()`

Der Standardbefehl `ADC()` besteht aus einer Sequenz von mehreren Befehlen (im folgenden dargestellt, siehe auch Seite 74ff) und benötigt eine werkseitig festgelegte Zeit zur Ausführung. Die Ausführungszeit wird vor allem durch die Einschwingzeit des Multiplexers und die Wandlungszeit bestimmt.

```
Set_Mux1()
...                          'Einschwingzeit abwarten
Start_Conv()
Wait_EOC()                   'Auf Wandlungsende warten
Read_ADC()
```

Wartezeiten nutzen

Sie können die im Standardbefehl enthaltenen Wartezeiten durch Verwendung der Einzelbefehle für andere Zwecke nutzen (oder ggf. auch verlängern). Bei geschicktem Einsatz der Befehle können Sie dadurch schnellere Messvorgänge realisieren.

Es ist wichtig, den Befehl `Start_Conv` in ausreichendem Zeitabstand vom Befehl `Set_Mux1` zu setzen, um die Einschwingzeit des Multiplexers zu berücksichtigen.

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

- Die Einschwingzeit der Multiplexer beträgt beim maximalen Spannungssprung von 20 Volt höchstens 2 µs.
- Die Wandlungszeit der ADCs beträgt jeweils 2 µs.

6 DA-Erweiterung

Anschlüsse

Mit den DA-Erweiterungen erhalten Sie insgesamt 4 (Gold II-DA4) oder 8 (Gold II-DA8) analoge Ausgänge mit einer Auflösung von 16 Bit (und je einem DAC).

Die Ausgänge liegen sowohl auf den BNC-Buchsen OUT 1…OUT 8 als auch auf der 37-polige Sub-D-Buchse ANALOG OUT (siehe Abbildung).

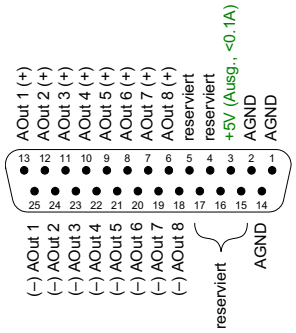


Abb. 12 – Pin-Belegung ANALOG OUT der DA-Erweiterung

Programmieren

Sie programmieren die zusätzlichen DAC wie bei den DAC 1 und DAC 2 (siehe Kapitel 5.1.2 auf Seite 14 und Befehlsreferenz in Kapitel 16 ab Seite 71).

7 CNT-Erweiterung

Die Erweiterung Gold II-CNT stellt zur Verfügung:

– 4 Zählerblöcke: Ein Zählerblock enthält zwei 32 Bit-Zähler: Zum einen einen Vor-/Rückwärtszähler mit Takt/Richtungs-Auswertung oder Vierflankenauswertung zum Anschluss von Encodern. Zum anderen einen Zähler zur Periodendauer- und Tastverhältnismessung.

– 4 SSI-Decoder (Seite 31)

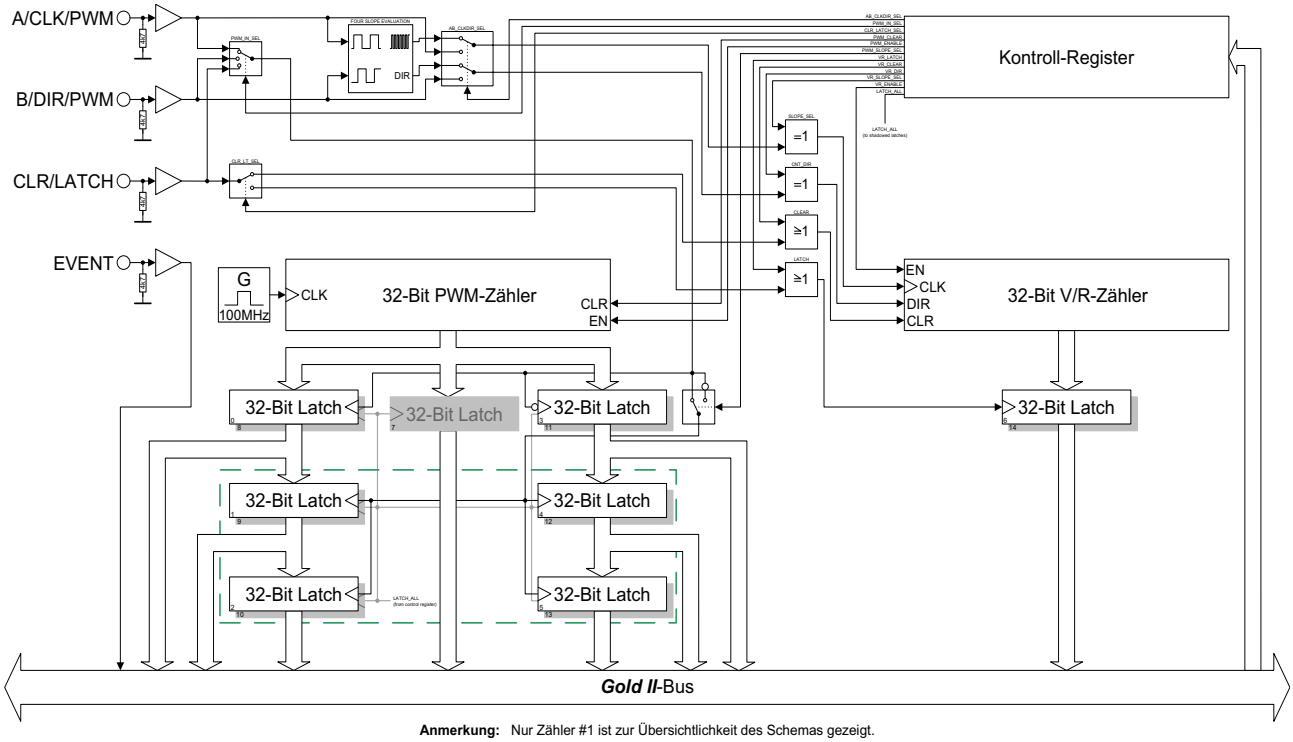
  Der SSI-Decoder eignet sich zum Anschluss von Inkremental-Encodern mit SSI-Schnittstelle. Die Eingänge liegen auf den Steckern CO1…CO4; die Signale sind differentiell und für RS422/485-Pegel ausgelegt.

– 6 PWM-Ausgänge (Seite 32): Ausgabe von pulsweitenmodulierten Signalen mit wählbarem Tastverhältnis.

7.1 Zähler-Hardware

Zählerblock

Die vier Zählerblocks der Zählererweiterung arbeiten mit jeweils 2 parallelen 32 Bit-Zählern, die voneinander unabhängig sind: Vor-/ Rückwärtszähler für externe Takte mit Takt/Richtung- oder Vierflankenauswertung und PWM-Zähler mit internem Takteingang zur Pulsweitenmessung. Sie können die Zähler per Software sowohl einzeln als auch gemeinsam konfigurieren. Die Daten der Zähler werden in Latches zum Auslesen zur Verfügung gestellt.



Anmerkung: Nur Zähler #1 ist zur Übersichtlichkeit des Schemas gezeigt.

Abb. 13 – Schema des Zählerblocks

V/R-Zähler (externer Takteingang)

Bei der Ereignismessung wird das In-/Dekrementieren des Zählers durch externe Rechtecksignale an den Eingängen A/CLK und B/DIR ausgelöst. Eine steigende Flanke an CLR/LATCH bewirkt, dass entweder der Zähler auf Null gesetzt (CLR) oder der Zählerstand ins Latch geschrieben wird (LATCH). Siehe auch Kapitel 7.3.

Es gibt die Modi:

1. **Takt und Richtung**: Eine steigende Flanke an A/CLK in- oder dekrementiert den Zählerstand um eins. Das Signal an B/DIR bestimmt die Zählrichtung (0 = Dekrementieren; 1 = Inkrementieren).
2. **Vierflankenauswertung (A/B)**: 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.

Sie können die Signale an den Eingängen A/CLK und B/DIR per Software (Befehl `Cnt_Mode`) invertieren und damit sowohl die auslösende Flanke als auch die Zählrichtung ändern.

PWM-Zähler (interner Takteingang)

Bei der Pulsweitenmessung wird das In-/Dekrementieren des PWM-Zählers von einem internen Referenztaktgeber mit einer Signalfrequenz von 100 MHz ausgelöst; Näheres siehe Kapitel 7.4 auf Seite 29.

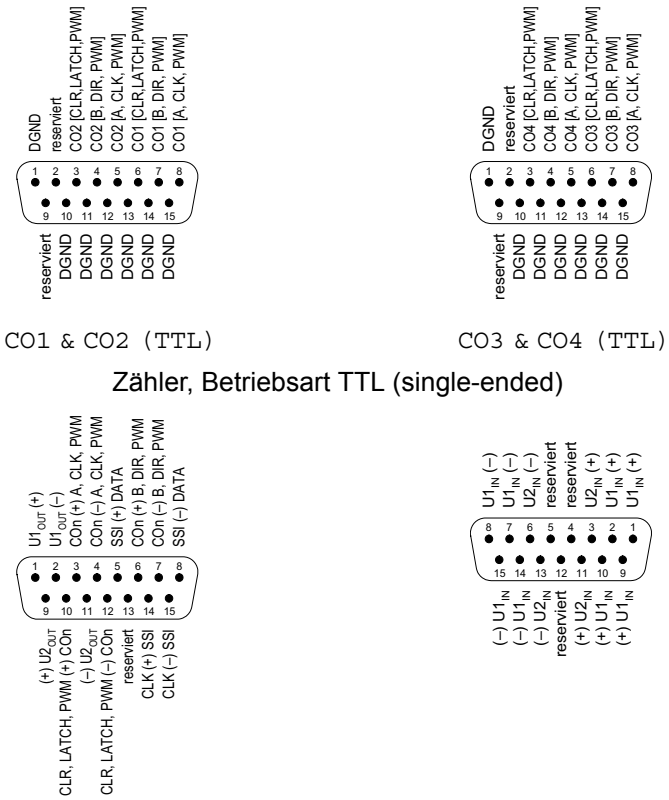
Der Zählerstand wird in einen Zwischenspeicher (Latch) geschrieben, wenn an dem gewählten Eingang (A/CLK, B/DIR oder CLR/LATCH) eine – nach Wahl positive oder negative – Flanke auftritt. Das Latchen kann auch per Software ausgelöst werden.

Aus dem Latch-Zwischenspeicher können Sie direkt Frequenz und Tastverhältnis oder Eintast- und Austastzeit abrufen.

Die Zähler werden mit *ADbasic*-Befehlen über Kontrollregister gesteuert (Befehlsübersicht siehe unten).

An den differentiellen Eingängen A/CLK, B/DIR und CLR/LATCH sind TTL-ähnliche Signale erforderlich.

Es ist in jedem Fall erforderlich, dass Sie die Zählereingänge mit dem Befehl `Cnt_SE_Diff` für die gewünschte Betriebsart (differentiell / single ended) einstellen.



Zähler, Betriebsart TTL (single-ended)

Zähler, Betriebsart differentiell / SSI

Zähler-Spannungsversorgung

Abb. 14 – Pinbelegungen der Zähler

Obwohl alle Eingänge der *CNT*-Erweiterung einen Pull-down-Widerstand von 4,7 kΩ besitzen, können offene Eingänge (bei dem zugehörigen Zähler) vor allem in einer nicht störungsfreien Umgebung zu Fehlern führen. Wenn Sie einen Eingang eines Zählers nicht benutzen, legen Sie sicherheitshalber beide Leitungen des (differentiellen) Eingangs auf ein definiertes Potenzial: Legen Sie den Pluseingang auf +5 V und den Minuseingang auf GND.

Externe Spannungsversorgung

Über den Anschluss `CO Power in` können Sie zwei Spannungen U_1 und U_2 einspeisen, die an jedem der Stecker CO1…CO4 an den Pins U_{1out} und U_{2out} zur Verfügung stehen, z.B. für externe Inkrementalgeber. An den Ausgangs-Pins kann jeweils ein maximaler Strom von 100 mA abgenommen werden; zwischen Ein- und Ausgangs-Pins ist eine entsprechende Sicherung eingebaut.

Beachten Sie: Die Minus-Eingänge U_{1in} (-) sind intern über eine gemeinsame Leitung galvanisch mit GND verbunden; auch die Minus-Eingänge U_{2in} (-) haben eine solche Verbindung.

7.2 Zähler-Software

Die für den Zugriff auf die Zähler benötigten Funktionen befinden sich in der Include-Datei `ADwinGoldII.inc` für *ADbasic* und in `GoldIITiCo.inc` für *TiCoBasic*.

Binden Sie die Include-Datei zum Beginn eines Programms ein, damit Sie die Befehle aus der nachfolgenden Tabelle benutzen können. Die Befehle sind in Kapitel 16 ab Seite 117 oder in der Online-Hilfe beschrieben.

Befehl	Funktion
Cnt_Clear	Zähler löschen.
Cnt_Enable	Zähler sperren oder freigeben (achten Sie auf bereits laufende Zähler).
Cnt_Get_Status	Statusregister auslesen.
Cnt_Latch	Zählerstand in Latch A kopieren.
Cnt_Sync_Latch	Zähler und PWM-Zähler gleichzeitig in Zwischenspeicher kopieren.
Cnt_Mode	Betriebsart eines Zählers festlegen.
Cnt_Read	Zählerstand in Latch A kopieren und auslesen.
Cnt_Read_Latch	Latch A (getriggert durch pos. Flanke) auslesen.
Cnt_Read_Int_Register	Inhalt eines Zählerregisters zurückgeben.
Cnt_SE_Diff	Differentielle oder TTL-Eingänge einstellen.
Cnt_PW_Latch	Zählerstand von PWM-Zählern in Latch kopieren.
Cnt_Get_PW	Frequenz und Tastverhältnis eines PWM-Zählers lesen.
Cnt_Get_PW_HL	Eintast- und Austastzeit eines PWM-Zählers lesen.

Abb. 15 – Befehle der *Gold II-CNT*-Zählererweiterung

Die Befehle beeinflussen oft **alle** Zähler. Achten Sie deshalb darauf, immer alle Bits korrekt zu setzen oder zu löschen. Sie können dadurch jeden Zähler einzeln oder beliebig viele Zähler gemeinsam beeinflussen.

Befehlsfolge

Konfigurieren Sie die Zähler bitte in dieser Reihenfolge:

1. Gewünschten Zähler sperren (Cnt_Enable)
2. Betriebsart einstellen (Cnt_Mode, Cnt_SE_Diff)
3. Zähler löschen (Cnt_Clear)
4. Zähler freigeben (Cnt_Enable)

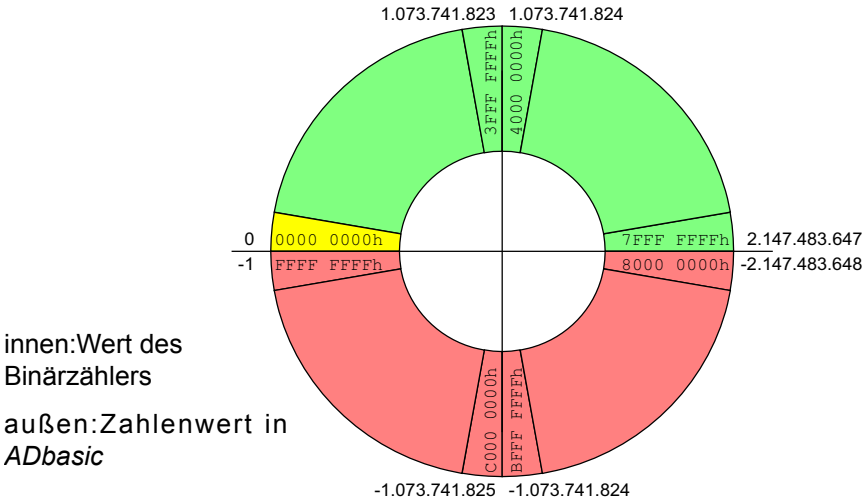
Für die Verarbeitung der Werte im *ADbasic*-Programm übertragen Sie die Werte ggf. ins Latch-Register und lesen sie dort aus.

Wenn Sie einen bestimmten Zähler sperren oder freigeben möchten, müssen Sie auch die schon laufenden Zähler freigeben (= Bits setzen). Wenn Sie (unbeabsichtigt) die Bits dieser Zähler nicht setzen, werden diese gesperrt.

7.2.1 Auswerten des Zählerinhalts

Die Binärzähler der *CNT*-Erweiterung erzeugen 32 Bit-Werte, die *ADbasic* nach dem Modell des unten stehenden Zahlenkreises als Zahlen mit Vorzeichen interpretiert: Das höchste Bit (MSB) stellt das Vorzeichen dar; die größte positive Zahl (2^{31}-1) schließt an die höchste negative Zahl (-2^{31}) an und die kleinste positive (0) an die kleinste negative Zahl (-1).

Zahlenkreis



innen:Wert des Binärzählers

außen:Zahlenwert in *ADbasic*

Abb. 16 – Zahlenkreis als Interpretation von Zählerwerten

Beachten Sie deswegen bei der Programmierung die nachstehenden Regeln:

a) Verarbeiten Sie den gelesenen 32 Bit-Werts nur mit Variablen vom Typ LONG. *ADbasic* behält dann intern das gelesene Bitmuster unverändert bei und berücksichtigt automatisch den Übergang zwischen positivem und negativem Zahlenbereich. Damit gilt:

Zählrichtung

b) 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.

„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 Eingangsfrequenz von 100 MHz nach etwas mehr als 42 Sekunden ein.

7.3 Ereigniszähler einsetzen

Externe Rechtecksignale an den Eingängen A/CLK und B/DIR takten in dieser Betriebsart den jeweiligen Zähler.

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

– den Zähler zu löschen (CLR)

– den Zählerstand ins Latch-Register A zu übernehmen (LATCH).

7.3.1 Takt und Richtung

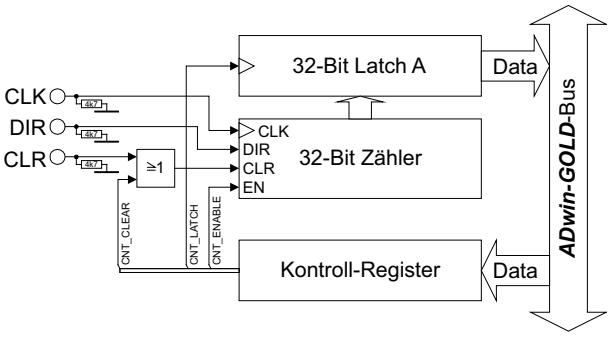


Abb. 17 – Schema *CNT*-Erweiterung im Modus „Takt und Richtung“

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 sowohl statisch sein, für eine feste Zählrichtung, oder auch dynamisch, für wechselnde Zählrichtungen.

Die Signale an den Eingängen A/CLK und B/DIR können mit **Cnt_Mode** (unabhängig voneinander) invertiert werden.

Programmbeispiel

```
#Include ADwinGoldII.inc
Init:
  …
  Cnt_Enable(0)           'alle Zähler anhalten
  Cnt_Clear(0001b)        'Zähler 1 löschen
  Rem Betriebsmodus Zähler 1 einstellen:
  Rem Bit 0: Modus Takt-Richtung
  Rem Bit 1: Löschmodus mit Clr-Eingang
  Rem Bit 2: Eingang A/CLK nicht invertieren
  Rem Bit 3: Eingang B/DIR nicht invertieren
  Rem Bit 4: Eingang CLR/LATCH als CLR-Eingang
  Rem Bit 5: Eingang CLR/LATCH freigeben
  Cnt_Mode(1,100000b)
  Cnt_SE_Diff(0000b)      'Alle Eingänge single-ended
  Cnt_Enable(0001b)       'Zähler 1 starten
  …
Event:
  …
  Cnt_Latch(0001b)        'Zähler 1 latchen
  val = Cnt_Read_Latch(0001b) 'Latch-Wert lesen
```

7.3.2 Vier-Flanken-Auswertung

Dieser Modus ermittelt Takt und Zählrichtung aus zwei Rechteck-Signalen, die an den Eingängen A und B um 90 Grad versetzt anliegen. Die Zählrichtung ergibt sich aus der zeitlichen Abfolge mit der die steigenden und fallenden Flanken der beiden Signale eintreffen.

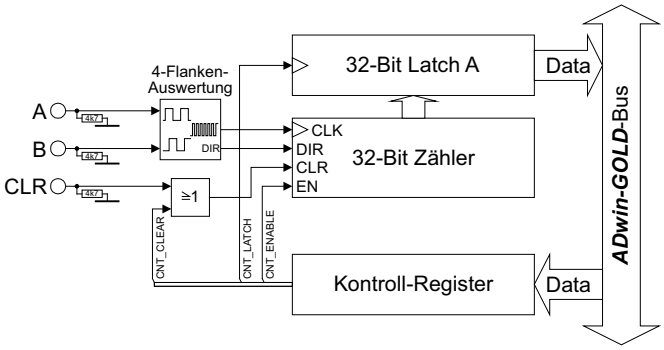


Abb. 18 – Schema *CNT*-Erweiterung im Modus „4-Flanken-Auswertung“

Berücksichtigen Sie bitte:

– Der Zähler registriert bei einem Zyklus des A/B-Signals 4 Flanken.

– Die maximale Zählfrequenz beträgt 20 MHz. Gemeinsam mit den 4 Flanken je Zyklus ergibt sich daraus eine maximale Eingangsfrequenz 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 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.

Programmbeispiel

```
#Include ADwinGoldII.inc
Init:
  ...
  Cnt_Enable(0)           'alle Zähler anhalten
  Cnt_Clear(0001b)        'Zähler 1 löschen
  Rem Betriebsmodus Zähler 1 einstellen:
  Rem Bit 0: Modus A/B = Vierflankenauswertung
  Rem Bit 1: Löschmodus mit Clr-Eingang
  Rem Bit 2: Eingang A/CLK nicht invertieren
  Rem Bit 3: Eingang B/DIR nicht invertieren
  Rem Bit 4: Eingang CLR/LATCH als CLR-Eingang
  Rem Bit 5: Eingang CLR/LATCH freigeben
  Cnt_Mode(1,100001b)
  Cnt_SE_Diff(1111b)      'Alle Eingänge differentiell
  Cnt_Enable(0001b)       'Zähler 1 starten
  ...
Event:
  ...
  Cnt_Latch(0001b)        'Zähler 1 latchen
  val = Cnt_Read_Latch(0001b) 'Latch-Wert lesen
```

7.4 PWM-Zähler einsetzen

Referenztaktgeber

In dieser Betriebsart taktet ein interner Referenztaktgeber den PWM-Zähler mit einer Signalfrequenz von 100 MHz. Es können Frequenz und Tastverhältnis oder Eintast- und Austastzeit gemessen werden.

Programmbeispiel

```
#Include ADwinGoldII.inc
#Define frequency FPAR_1
#Define dutycycle FPAR_2
#Define hightime PAR_1
#Define lowtime PAR_2
Init:
  …
  Cnt_Enable(0)           'alle Zähler anhalten
  Rem Betriebsmodus PWM-Zähler 1 einstellen:
  Rem Bits 0..5: ohne Bedeutung
  Rem Bit 6: steigende Flanke als PWM-Signal
  Rem Bit 7,8: Eingang B/DIR als PWM-Eingang
  Cnt_Mode(1,010000000b)
  Cnt_SE_Diff(1111b)      'Alle Eingänge differentiell
  Cnt_Enable(100000000b) 'nur PWM-Zähler 1 starten
  …
Event:
  …
  Rem Zähler 1 latchen
  Cnt_PW_Latch(0001b)
  Rem Frequenz und Tastverhältnis lesen
  Cnt_Get_PW(1,frequency,dutycycle)
  Rem Impuls- und Pausendauer lesen
  Cnt_Get_PW_HL(1,hightime,lowtime)
```

Ausnahme: PWM-Register selbst auswerten

Zu jedem PWM-Zähler gehören mehrere Register, die im folgenden beschrieben werden. Wenn Sie die PWM-Zähler wie im Beispiel mit den Standard-Befehlen `Cnt_Get_PW` und `Cnt_Get_PW_HL` auswerten, benötigen Sie keine Kenntnisse über die PWM-Register. Nur für spezielle Lösungen ist es sinnvoll, wenn Sie die PWM-Register selbst auswerten.

Um PWM-Signale auszuwerten zu können, werden in Latch-Registern die Zählerstände für das aktuelle und zwei vorhergehende PWM-Signale gespeichert, sowohl für steigende als auch für fallende Flanken. Für die Auswertung gibt es für jedes der 6 Register ein sogenanntes „Schattenregister“.

Register	Latch	Schatten-register
Latch 1 für positive Flanken (aktuell)	L1+	SL1+
Latch 2 für positive Flanken	L2+	SL2+
Latch 3 für positive Flanken	L3+	SL3+
Latch 1 für negative Flanken (aktuell)	L1–	SL1–
Latch 2 für negative Flanken	L2–	SL2–
Latch 3 für negative Flanken	L3–	SL3–

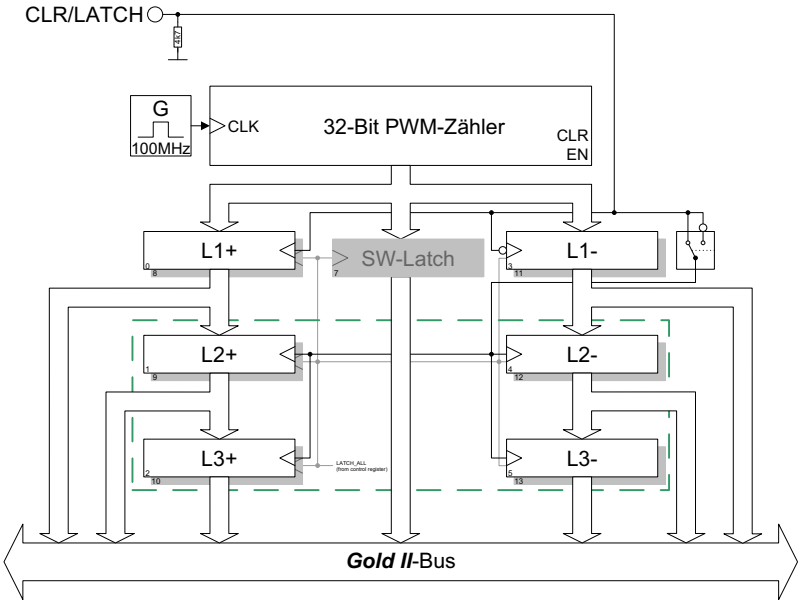
Die Registerwerte werden bei einer Flanke wie folgt geändert:

– Steigende Flanke:
  - Zählerstand nach L1+ kopieren
  - Wenn die steigende Flanke als Referenzflanke eingestellt ist:
    Register L2+ nach L3+ kopieren
    Register L1+ nach L2+ kopieren

Register L2– nach L3– kopieren
Register L1– nach L2– kopieren

– Fallende Flanke:
  • Zählerstand nach L1– kopieren
  • Wenn die fallende Flanke als Referenzflanke eingestellt ist:
    Register L2– nach L3– kopieren
    Register L1– nach L2– kopieren
    Register L2+ nach L3+ kopieren
    Register L1+ nach L2+ kopieren

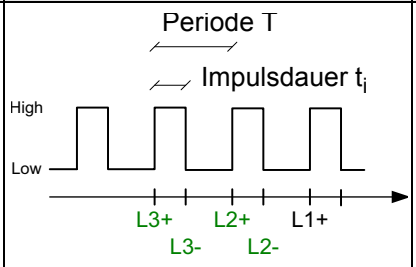
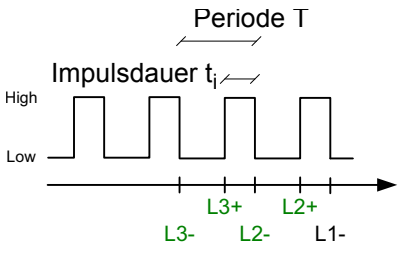
Zusätzlich gibt es ein einzelnes Latch-Register, in das der Zählerstand per Software (Befehl `Cnt_PW_Latch`) kopiert wird.



Beispiel: Auswertung der PWM-Register

Bei der Auswertung werden immer die PWM-Register der Ebenen 2 und 3 verwendet. Zunächst werden alle Registerwerte mit `Cnt_Sync_Latch` in die Schattenregister kopiert und anschließend ausgewertet.

Die Berechnung ist abhängig von der eingestellten Referenzflanke:

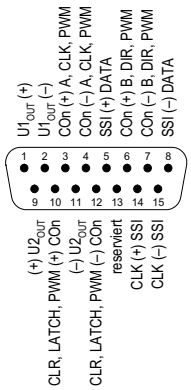
Parameter	steigende Flanke	fallende Flanke
Schema		
Periode	T = L2+ – L3+	T = L2- – L3-
Impulsdauer	t_H = L3- – L3+	t_H = L2- – L3+
Pausendauer	$t_L = T - t_H$ = L2+ – L3-	$t_L = T - t_H$ = L3+ – L3-
Frequenz	f = 1 / T = 1 / (L2+ – L3+)	f = 1 / T = 1 / (L2- – L3-)
Tastverhältnis	g = t_H / T = (L3- – L3+) / (L2+ – L3+)	g = t_H / T = (L2- – L3+) / (L2- – L3-)

7.5 SSI-Decoder

An die 4 SSI-Decoder kann jeweils ein Inkremental-Encoder mit SSI-Schnittstelle angeschlossen werden. Die Signale sind differentiell und haben RS422/485-Pegel.

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

Die Anschlüsse der Decoder stehen auf den Steckern CO1…CO4 (15-polig, Sub-D, siehe Seite 11) zur Verfügung und zwar auf den Pins 5, 8, 14 und 15.



Eigenschaften einstellen

Folgende Eigenschaften des SSI-Decoders sind per Software einstellbar:

– Taktrate: Über einen Vor-Teiler sind Taktraten von ca. 100 kHz bis 2,5 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. 19 – Listing: Konvertierung von Gray- in Binär-Code

Programmierung

Die SSI-Decoder werden mit Befehlen aus <ADwinGoldII.inc> für *ADbasic* und aus <GoldIITiCo.inc> für *TiCoBasic* komfortabel programmiert; Beschreibung ab Seite 134 oder in der Online-Hilfe.

Bereich	Befehle
Decoder initialisieren	**SSI_Mode** **SSI_Set_Bits** **SSI_Set_Clock**
Encoder-Daten auslesen	**SSI_Read** **SSI_Start** **SSI_Status**

7.6 PWM-Ausgänge

Die PWM-Ausgabe ermöglicht, auf den 6 Ausgängen jeweils ein pulsweitenmoduliertes Signal mit wählbarem Tastverhältnis auszugeben. Die Ausgabe wird mit 50 MHz getaktet.

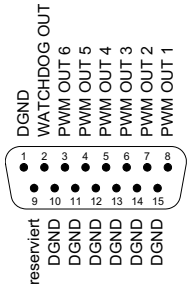


Abb. 20 – Pinbelegungen der PWM-Ausgänge, PWM 1-6 (TTL)

Programmierung

Die PWM-Ausgänge werden mit Befehlen aus `<ADwinGoldII.inc>` für *ADbasic* und aus `<GoldIITiCo.inc>` für *TiCoBasic* komfortabel programmiert; Beschreibung in Kapitel 16 ab Seite 141:

Bereich	Befehle

PWM-Ausgänge 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`

8 CAN-Erweiterung

Die Erweiterung *Gold II-CAN* beinhaltet mehrere zusätzliche Schnittstellen, die unabhängig voneinander konfiguriert und betrieben werden können:

– 2 CAN-Schnittstellen (Seite 34)Die beiden Schnittstellen eignen sich je nach Bestelloption entweder für „High-speed“ oder für „Low-speed“. Ein Umschalten im Betrieb ist nicht möglich.

  Die Eingänge der Schnittstellen liegen auf den Steckern CAN 1 und CAN 2.

– 2 RSxxx-Schnittstellen (Seite 38)

  Beide Schnittstellen können unabhängig voneinander per Software auf RS232 oder auf RS485 eingestellt und betrieben werden.

  Die Schnittstellen-Eingänge liegen auf den Steckern COM1 und COM2.

Beachten Sie: Sie können entweder mit *ADbasic*-Befehlen auf die CAN- und RSxxx-Schnittstellen zugreifen oder aber mit *TiCoBasic*-Befehlen, aber nicht mit beiden gleichzeitig.

8.1 CAN-Schnittstelle

Die CAN-Schnittstellen 1 und 2 können voneinander unabhängig betrieben werden. Sie eignen sich je nach Bestelloption entweder für „High-speed“ oder für „Low-speed“. Eine Umschaltung im Betrieb ist nicht möglich.

8.1.1 Hardware-Beschreibung

Die Anschlüsse der Schnittstellen stehen auf 9-poligen Sub-D-Verbindern CAN 1 und CAN 2 zur Verfügung, die Belegung ist gleich.

Die Pinbelegung für CAN „High speed“ und „Low speed“ ist unterschiedlich.

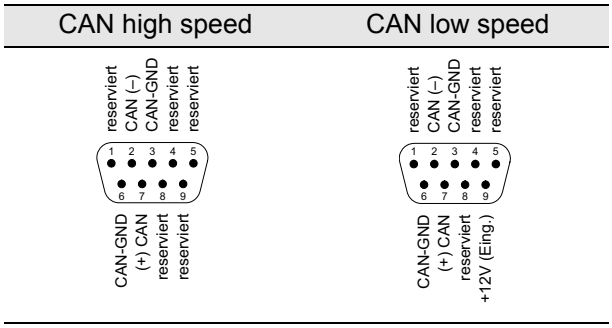


Abb. 21 – CAN: Pinbelegungen

Beide Schnittstellen haben ein eigenes Potential CAN-GND, die sowohl voneinander galvanisch getrennt sind als auch vom Masse-Potential (GND) des Gehäuses.

Spannungsversorgung (nur Low speed)

Bei der „low speed“-Variante ist eine externe Spannungsversorgung mit 12V Gleichstrom erforderlich. Die Spannung muss für jede Schnittstelle separat eingespeist werden.

Bus-Terminierung (nur High speed)

Wenn die CAN-Schnittstelle das physikalische Ende eines CAN-Bus vom Typ „High speed“ bildet, muss es mit einem Abschlusswiderstand 120Ω terminiert werden (also nur am ersten oder letzten CAN-Knoten). An CAN-Knoten, die sich nicht an einem physikalischen Ende der Kette befinden, darf nicht terminiert werden.

Wenn für eine der beiden (oder beide) Schnittstellen die Terminierung erforderlich ist, müssen Sie die Pins CAN(+) und CAN(-) durch einen Widerstand von 120Ω verbinden.

8.1.2 Beschreibung der CAN-Schnittstelle

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 1MHz einstellbar und wird standardmäßig mit 1MHz betrieben; bei CAN low speed beträgt die max. Fre-

quenz 125kHz. 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.

Programmierung

Die CAN-Schnittstellen werden mit Befehlen aus `<ADwinGoldII.inc>` für *ADbasic* und aus `<GoldIITiCo.inc>` für *TiCoBasic* komfortabel programmiert; Beschreibung in Kapitel 16 ab Seite 150 oder in der Online-Hilfe:

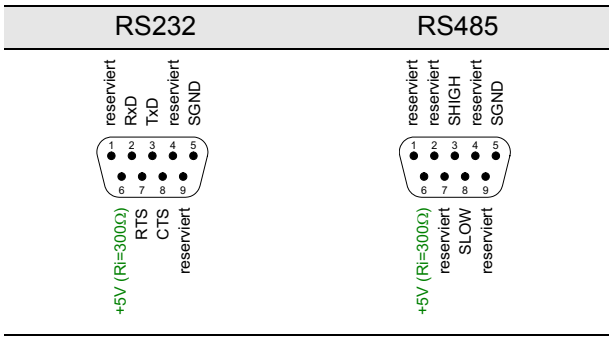
Bereich	Befehle
Initialisierung	**Init_CAN** **En_CAN_Interrupt** **Set_CAN_Baudrate**
Empfangen und Senden von Daten	CAN_Msg **En_Receive**, **En_Transmit** **Read_Msg**, **Read_Msg_Con**, **Transmit**
Schreib- / Lesezugriff auf Controller-Register	**Set_CAN_Reg** **Get_CAN_Reg**

8.2 RSxxx-Schnittstellen

Jede der 2 RSxxx-Schnittstellen ist mit dem Controller „Quad Universal Asynchronous Receiver/Transmitter“ (UART) vom Typ TL16C754 der Firma Texas Intruments® bestückt. Die Funktionalität und Programmierung der Schnittstellen beruhen auf diesem Controller.

Beide Schnittstellen können unabhängig voneinander mit dem Protokoll RS232 oder RS485 betrieben werden. Der physikalische Unterschied der Protokolle liegt in den Pegeln der Signale, die auf dem „Bus“ durch entsprechende Treiber-Bausteine bereit gestellt werden.

Pinbelegung



Bus-Terminierung (nur RS485)

Wenn eine RS485-Schnittstelle das physikalische Bus-Ende bildet, muss es mit einem Abschlusswiderstand terminiert werden (also nur der erste und der letzte RS485-Teilnehmer). Bei RS485-Teilnehmern, die sich nicht an einem physikalischen Ende der Kette befinden, darf nicht terminiert werden.

Für die Terminierung steht – falls die gewählte Schaltungsvariante dies erfordert – am Pin 6 eine Spannung von +5 V zur Verfügung. In dieser Leitung ist ein Widerstand von 300 Ω vorhanden (Strombegrenzung).

8.2.1 Einstellung der Schnittstellen-Parameter

Jede Schnittstelle verfügt über einen Eingangs- und einen Ausgangs-FIFO mit einer Länge von jeweils 64 Byte.

Die Schnittstellen-Parameter werden mit Hilfe der Controller-Register eingestellt, und zwar getrennt für jede Schnittstelle. Im folgenden werden die Einstellmöglichkeiten dargestellt.

Handshake

– Handshake: Die Schnittstelle kann in 4 Modi betrieben werden:

1. RS232 ohne Handshake
2. RS232 mit Software-Handshake (Xon/Xoff)
3. RS232 Hardware-Handshake (RTS/CTS). Hierbei müssen die Signale RTS und CTS angeschlossen sein.
4. RS485

Parität

– Parität: Um einen Fehler bei der Übertragung und damit fehlerhafte Daten erkennen zu können, kann ein Paritätsbit mit übertragen werden. Die Parität kann gerade oder ungerade sein, oder es kann auf das Paritätsbit verzichtet werden.

Daten-Bits

– Datenbits: Die Nutzdaten, die übertragen werden sollen, können aus 5…8 Bits bestehen.

Stopp-Bits

– Stopp-Bits: Die Anzahl der Stopp-Bits kann auf 1, 1½ oder 2 eingestellt werden. Dabei ist die Anzahl der Stoppbits von der Anzahl der Datenbits abhängig:
  - 5 Datenbits: 1 oder 1½ Stoppbits.
  - 6…8 Datenbits: 1 oder 2 Stoppbits.

Baudrate

– Baudrate: Die physikalisch erreichbaren Werte liegen zwischen 35 Baud und 2,304 MBaud; bei einer RS-232 Schnittstelle liegt die max. Baudrate laut Spezifikation bei 115,2 kBaud.

  Die einstellbaren Baudraten werden vom moduleigenen Taktgeber abgeleitet; der Grundtakt hat eine Frequenz von 2,304 MHz. Davon ausgehend ist jede Baudrate möglich, die sich durch ganzzahlige Division dieses Grundtakts ergibt. Der Teiler kann Werte im Bereich von 1…0FFFFh annehmen. Die nachfolgende Tabelle zeigt einige gängige Baudraten und die zugehörigen Teiler.

Baudrate	Teiler	
	dez.	hex.
2304000	1	0001h
1152000	2	0002h
460800	5	0005h
230400	10	000Ah
115200	20	0014h
57600	40	0028h
38400	60	003Ch

Baudrate	Teiler	
	dez.	hex.
19200	120	0078h
9600	240	00F0h
4800	480	01E0h
2400	960	03C0h
1200	1920	0780h
600	3840	0F00h
300	7680	1E00h

Abb. 22 – RS-xxx: Gängige Baudraten

Besonderheiten RS485

Über eine RS485-Schnittstelle können – im Gegensatz zu RS232 – mehr als 2 Teilnehmer miteinander kommunizieren. So kann mit Hilfe von RS485-Schnittstellen ein Bus aufgebaut werden.

Daraus ergeben sich folgende Hinweise:

– Es gibt keinen Handshake, da ein Handshake immer nur zwischen 2 Teilnehmern funktioniert.

– Jeder Schnittstelle muss vor dem Betrieb mitgeteilt werden, ob sie auf den Bus schreiben soll oder nur Daten vom Bus übernehmen darf (**RS485_SEND**).

8.2.2 Programmierung

Die Funktionalität und Programmierung der Schnittstelle beruhen auf dem eingebauten Schnittstellen-Controller. Der Controller wird mit Befehlen aus `<ADwinGoldII.inc>` für *ADbasic* und aus `<GoldIITiCo.inc>` für *TiCoBasic* komfortabel programmiert; Beschreibung in Kapitel 16 ab Seite 165 oder in der Online-Hilfe:

Bereich	Befehle
Initialisierung	**RS_Init**, **RS_Reset**
Empfangen und Senden von Daten	**Check_Shift_Reg**, **RS485_Send**, **Read_Fifo**, **Write_Fifo**, **Write_Fifo_Full**
Schreib- / Lesezugriff auf Controller-Register	**Get_RS**, **Set_RS**

RS232

Beispielprogramme

Das nachfolgende Programm zeigt die Initialisierung der seriellen RS232-Schnittstelle im Abschnitt **Init:** und das zyklische Lesen und Schreiben von Daten im Abschnitt **Event:**. Der Prozess ist zeitgesteuert.

```
REM Das Programm initialisiert die seriellen
REM Schnittstellen im Abschnitt Init:
REM Im Abschnitt Event: werden Daten zwischen den
REM Schnittstellen 1 & 2 des RS-Moduls ausgetauscht.
REM Mit Hilfe dieses Programms können die Schnittstellen
REM untereinander getestet werden. Dazu müssen Sie die
REM Schnittstellen vor dem Programmstart miteinander
REM verbinden.

#Include ADwinGoldII.inc
DIM DATA_1[1000] AS LONG'Sendedaten
DIM DATA_2[1000] AS LONG'Empfangsdaten
DIM lauf AS LONG         'Laufvariable

INIT:
 FOR lauf = 1 TO 1000   'Initialisierung der Sendedaten
  DATA_1[lauf] = lauf AND 0FFh
 NEXT lauf
 REM Initialisierung der Schnittstellen:
 REM 9600 Baud, Kein Paritätsbit, 8 Datenbits,
 REM 2 Stoppbits, RS232 ohne Handshake
 RS_INIT(1,9600,0,8,1,0)
 RS_INIT(2,9600,0,8,1,0)
 PAR_1 = 1
 PAR_4 = 1

EVENT:
 REM Einen Datensatz lesen und schreiben
 IF (PAR_1 <= 1000) THEN'Daten senden
  PAR_2 = WRITE_FIFO(1,DATA_1[PAR_1])
  IF (PAR_2 = 0) THEN INC PAR_1
 ENDIF

 PAR_3 = READ_FIFO(2)   'Daten lesen
 If (PAR_3 <> -1) THEN
  DATA_2[PAR_4] = PAR_3
  INC PAR_4
 ENDIF
 IF (PAR_4 > 1000) THEN END 'Alle Daten sind übertragen
```

RS485

In diesem Beispiel wird eine RS485-Schnittstelle als passiver Teilnehmer verwendet, der alle Daten liest, die an seinem Eingang anliegen. Wenn ein bestimmter Wert (55) empfangen wird, wird die Schnittstelle aktiv und sendet dann ihrerseits fortlaufend den Wert 44.

```
REM Schnittstelle 2 liest so lange alle Daten vom Bus, bis
REM sie den Wert 55 empfängt. Danach wird die Schnittstelle
REM aktiv und sendet den Wert 44.

#Include ADwinGoldII.inc
dim ret_val, val as Long

init:
 rs_reset()
 REM Initialisierung der Schnittstellen:
 REM 38400 Baud, Kein Paritätsbit, 8 Datenbits,
 REM 1 Stoppbit, RS485 Software-Handshake
 rs_init(1,38400,0,8,0,3)
 rs_init(2,38400,0,8,0,3)
 rs485_send(1,1)          'Schnittstelle 1 senden
 rs485_send(2,0)          'Schnittstelle 2 empfangen

event:
  val = read_fifo(2)     'Daten aus Schnittstelle 2 lesen

  if (val = 55) then
   rs485_send(2,1)       'Schnittstelle 2 senden
   ret_val = write_fifo(2,44) 'Daten schreiben
  endif
```

9 Profibus-Erweiterung

Die Erweiterung *Gold II-Profibus* stellt einen Feldbusknoten mit der Funktionalität eines Profibus-Slave zur Verfügung. Alle Einstellungen werden per Software vorgenommen.

Funktionsbeschreibung

Nach dem Einschalten muss der Feldbusknoten initialisiert werden. Mit der Initialisierung werden die Stationsadresse (Slave-Knotenadresse) auf dem Profibus und die Größe der Ein- und Ausgangsbereiche festgelegt.

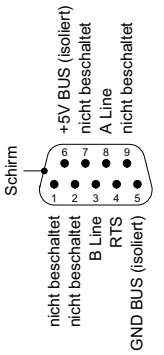
Es gibt je einen Bereich für eingehende und für ausgehende Daten; jeder Bereich hat eine maximale Größe von 76 Byte. Die Begriffe „Eingang“ und „Ausgang“ sind aus Sicht des Feldbus-Controllers (Slave) zu sehen.

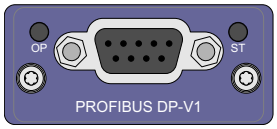
Sie legen für beide Bereiche separat fest, wieviele Datenbereiche in jedem Bereich vorhanden sind und wie groß die Datenbereiche sind.

Hardware

Die Pinbelegung der 9-poligen D-Sub-Buchse entspricht der DIN E 19245, Teil 3.

Der Profibus muss am physikalischen Anfang und Ende der Busleitung mit einem Abschlusswiderstand abgeschlossen werden. Falls erforderlich, müssen Sie den Abschlusswiderstand selbst an den entsprechenden Datenleitungen des Feldbusknotens anbringen oder einen entsprechenden Steckverbinder verwenden.





Neben der D-Sub-Buchse befinden sich zwei LEDs, die den Betriebszustand des Knotens im Profibus anzeigen, nämlich Betriebsmodus (OP) und Status (ST).

LED	Status	Bedeutung
OP	Aus	Nicht online oder keine Stromversorgung.
	Grün	Feldbusknoten online, Datenaustausch.
	Grün blinkend	Feldbusknoten online, Status Clear.
	Rot, einfach blinkend	Fehler: Ein/Ausgangskonfiguration stimmt nicht mit der Masterkonfiguration überein.
	Rot, doppelt blinkend	Fehler bei der Profibus-Konfiguration.
ST	Aus	Nicht online oder keine Stromversorgung.
	Grün	Initialisiert.
	Grün blinkend	Initialisiert, Diagnosemeldung liegt an.
	Rot	Ausnahmefehler.

Abb. 23 – Profibus: Bedeutung der LED

Profibus projektieren

Sie projektieren den Profibus mit einem – zum Bus-Master passenden – Konfigurations-Tool. Für das folgende Beispiel wurden ein Profibus-Master der Firma Hilscher und das zugehörige Programm `SyCon` verwendet.

Für andere Konfigurations-Tools gilt die folgende Ablaufbeschreibung entsprechend. Entnehmen Sie die genaue Vorgehensweise bei der Busprojektierung der Dokumentation Ihres Konfigurations-Tools.

GSD-Datei kopieren

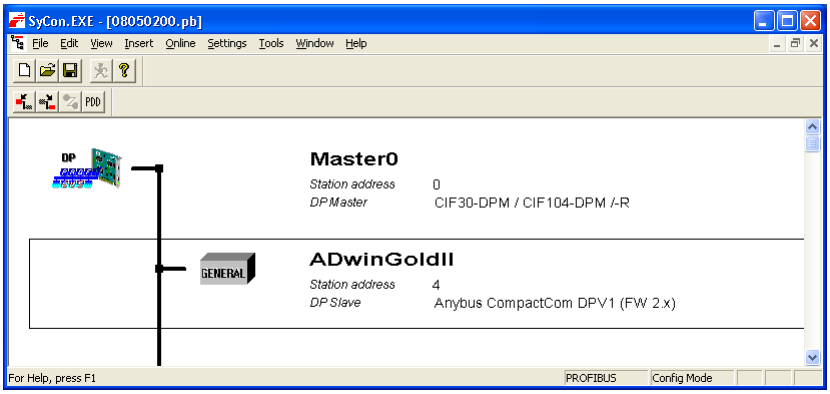
– Kopieren oder importieren Sie die GSD-Datei `hmsb1811.gsd` (Gerätestammdaten-Datei) des Feldbusknotens von `C:\ADwin\Fieldbus\Profibus` in das Quellverzeichnis des Konfigurations-Tools.

  Das Konfigurations-Tool lädt die benötigten Informationen über einen neuen Slave aus der zugehörigen GSD-Datei; der Dateiinhalt ist durch die EN 50170 festgelegt. Anschließend kann der Slave von jedem Master angesprochen werden.

Slave einbinden

– Fügen Sie im Konfigurations-Tool den Slave, also den Feldbusknoten zum Profibus hinzu, in dem Sie die GSD-Datei `hmsb1811.gsd` auswählen. Die Stationsadresse muss die gleiche sein wie bei der Initialisierung in *ADbasic* mit **`Init_Profibus`**.

  Danach könnte das Profibus-Layout wie folgt aussehen:



Slave konfigurieren

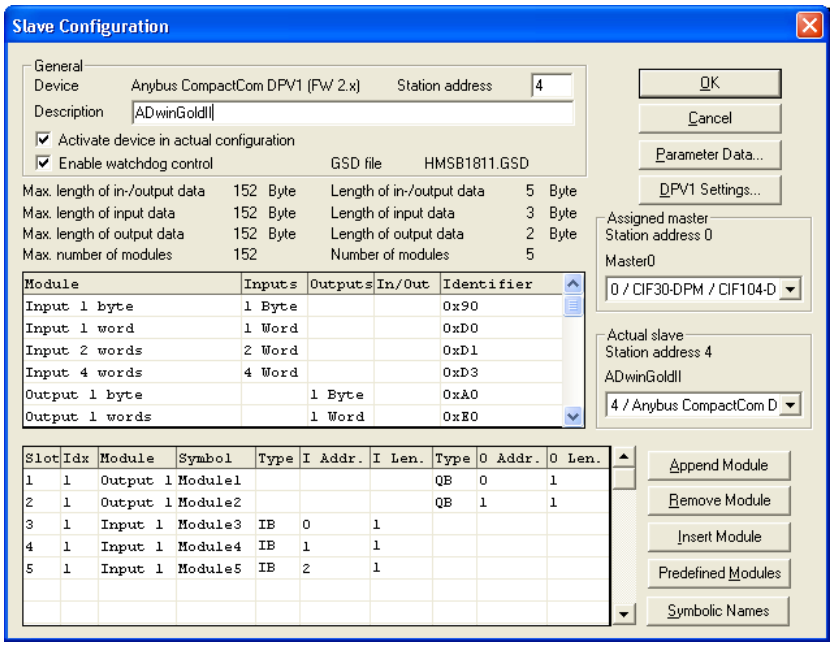
– Konfigurieren Sie die Anzahl und Größe der Datenbereiche für eingehende und für ausgehende Daten jeweils einzeln.
  Eventuelle Standard-Konfigurationen sind in aller Regel nicht sinnvoll verwendbar.

  Beachten Sie dabei folgende Regeln:
  - Die Begriffe „Eingang“ und „Ausgang“ in *ADbasic* (Slave) und im Konfigurations-Tool (Master) sind vertauscht.
  - Konfigurieren Sie (aus Sicht des Masters) zuerst die Ausgänge und dann die Eingänge.
    Wenn also in *ADbasic* Eingänge initialisiert wurden, müssen dafür im Master Ausgänge konfiguriert werden.
  - Anzahl und Größe der Datenbereiche müssen die gleichen sein wie bei der Initialisierung in *ADbasic* mit **`Init_Profibus`**.
  - Verwenden Sie für Eingang und Ausgang im Speicher des Feldbusknotens die gleiche Datenbereichsgröße. Datenbereiche können in Größen von 1, 2, 4 oder 8 Byte angelegt werden (2 byte = 1 word).

  Mit der folgenden Beispielzeile wird der Slave in *ADbasic* mit 2 Eingängen mit 1 Byte und 3 Ausgängen mit 1 Byte initialisiert.

```
PAR_31 = Init_Profibus(2, 2, 1, 3, 1, conf_Arr, DATA_1)
```

  Um den Slave im Konfigurations-Tool richtig einzurichten, müssen nun zuerst 2 Ausgänge und dann 3 Eingänge angelegt werden (jeweils einzeln mit 1 Byte). Die unten stehende Grafik zeigt die Konfiguration beispielhaft.



Programmieren in *ADbasic*

Der Feldbusknoten wird mit den folgenden *ADbasic*-Befehlen (nicht aber mit *TiCoBasic*) komfortabel programmiert:

Bereich	Befehle
Stationsadresse und Datenbereiche initialisieren	`Init_Profibus`
Daten schreiben und lesen	`Run_Profibus`

Die Befehle sind ab Seite 177 oder in der Online-Hilfe erläutert.

Die Initialisierung muss mit niedriger Priorität ablaufen, da sie einige Sekunden in Anspruch nimmt; bei hoher Priorität würde der PC nach einer bestimmten Zeit (time-out) die Kommunikation abbrechen. Aus dem gleichen Grund sollte auch das Schreiben und Lesen von Daten mit niedriger Priorität ablaufen.

Spezifikationen

Der Feldbusknoten entspricht dem europäischen Standard EN 50170 Volume 2. Dieser kann von der Profibus-Nutzerorganisation bezogen werden:

Profibus Nutzerorganisation e.V.
Haid-und-Neu-Str.7
76131 Karlsruhe
Tel.: +497219658590
Fax : +497219658589
Bestellnummer: 0.042

Betriebszustände des Feldbusknotens

Die nachfolgende Tabelle zeigt die Betriebszustände, die der Feldbusknoten unterstützt und welches Verhalten er in den verschiedenen Zuständen zeigt.

Betriebs-Zustand	Verhalten
Operate	Der Profibusslave nimmt am zyklischen Datenverkehr teil. Eingangsdaten werden von einem Master über den Bus übernommen und Ausgangsdaten werden für den Master zum Abholen bereitgestellt.
Clear	Die Eingänge werden weiterhin aktualisiert und die Ausgänge werden auf Null gesetzt.
Stop	Der Slave nimmt nicht an der Buskommunikation teil.

Abb. 24 – Profibus: Betriebszustände

10 Profinet-IO-Erweiterung

Die Erweiterung *Gold II-Profinet-IO* stellt einen Feldbusknoten mit der Funktionalität eines Profinet-Slave zur Verfügung. Alle Einstellungen werden per Software vorgenommen.

Funktionsbeschreibung

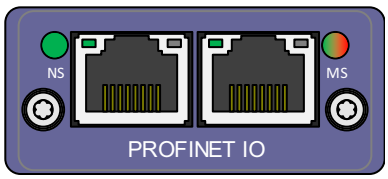
Nach dem Einschalten muss der Feldbusknoten initialisiert werden. Mit der Initialisierung wird die Größe der Ein- und Ausgangsbereiche festgelegt.

Es gibt je einen Bereich für eingehende und für ausgehende Daten; jeder Bereich hat eine maximale Größe von 76 Byte. Die Begriffe „Eingang“ und „Ausgang“ sind aus Sicht des Feldbus-Controllers (Slave) zu sehen.

Sie legen für beide Bereiche separat fest, wieviele Datenbereiche in jedem Bereich vorhanden sind und wie groß die Datenbereiche sind.

Hardware

An die RJ-45-Buchsen werden handelsübliche Ethernet-Stecker angeschlossen.



Neben den beiden RJ-45-Buchsen befinden sich zwei LEDs, die den Betriebszustand des Knotens im Profinet anzeigen: Netzwerk-Status (NS) und Modul-Status (MS).

LED	Status	Bedeutung
NS	Aus	Nicht online oder keine Stromversorgung.
	Grün	Feldbusknoten online, IO-Controller arbeitet.
	Grün, blinkt	Feldbusknoten online, IO-Controller ruht.
MS	Aus	Nicht online oder keine Stromversorgung.
	Grün	Normaler Betrieb.
	Grün blinkt 1-fach	Initialisiert, Diagnosemeldung liegt an.
	Grün blinkt 2-fach	Sonderfunktion zur Identifizierung.
	Rot	Ausnahmefehler.
	Rot, blinkt 1-fach	Fehler bei der Konfiguration.
	Rot, blinkt 2-fach	IP-Adresse ist nicht gesetzt.
	Rot, blinkt 3-fach	Stationsname fehlt.
	Rot, blinkt 4-fach	Schwerer interner Fehler.

Abb. 25 – Profinet: Bedeutung der LED

Profinet projektieren

Sie projektieren den Profinet-Bus mit einem – zum Bus-Master passenden – Konfigurations-Tool. Für das folgende Beispiel wurden ein Profinet-Master der Firma Siemens und das zugehörige Programm `SIMATIC-Manager` verwendet.

Für andere Konfigurations-Tools gilt die folgende Ablaufbeschreibung entsprechend. Entnehmen Sie die genaue Vorgehensweise bei der Busprojektierung der Dokumentation Ihres Konfigurations-Tools.

GSD-Datei kopieren

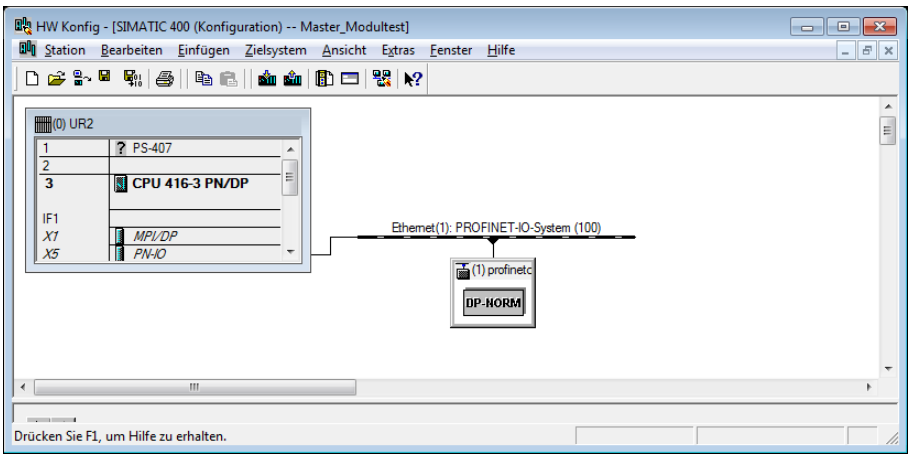
– Kopieren oder importieren Sie die GSD-Datei `GSDML-V2.25-HMS-ABCC-PRT2P-20110223.xml` (Gerätestammdaten-Datei) des Feldbusknotens von `C:\ADwin\Fieldbus\Profinet` in das Quellverzeichnis des Konfigurations-Tools.

  Das Konfigurations-Tool lädt die benötigten Informationen über einen neuen Slave aus der zugehörigen XML-Datei. Anschließend kann der Slave von jedem Master angesprochen werden.

Slave einbinden

– Fügen Sie im Konfigurations-Tool den Slave, also den Feldbusknoten zum Profinet hinzu, in dem Sie die oben genannte XML-Datei auswählen.

  Danach könnte das Profinet-Layout wie folgt aussehen:



Slave konfigurieren

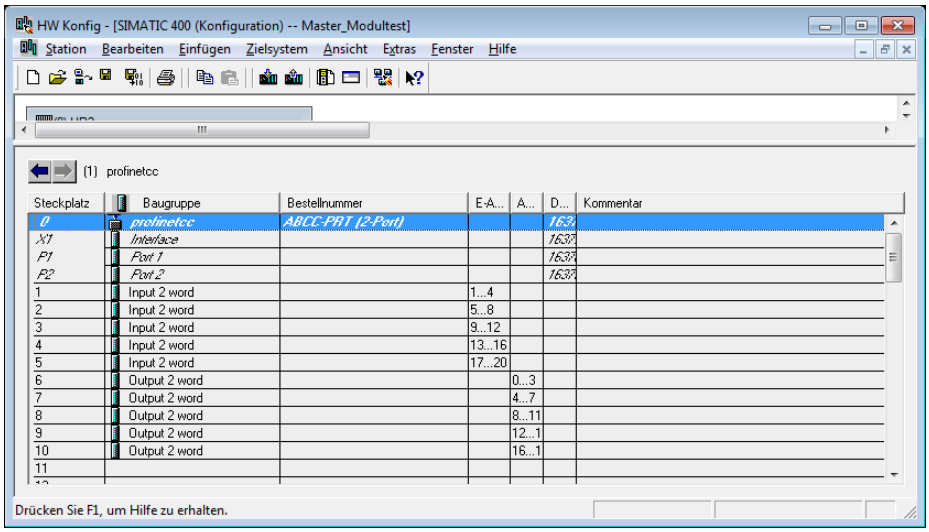
– Konfigurieren Sie die Anzahl und Größe der Datenbereiche für eingehende und für ausgehende Daten jeweils einzeln.
  Eventuelle Standard-Konfigurationen sind in aller Regel nicht sinnvoll verwendbar.

  Beachten Sie dabei folgende Regeln:
  - Die Begriffe „Eingang“ und „Ausgang“ in *ADbasic* (Slave) und im Konfigurations-Tool (Master) sind vertauscht.
    Wenn also in *ADbasic* Eingänge initialisiert wurden, müssen dafür im Master Ausgänge konfiguriert werden.
  - Anzahl und Größe der Datenbereiche müssen die gleichen sein wie bei der Initialisierung in *ADbasic* mit **`Init_Profinet`**.
  - Verwenden Sie für Eingang und Ausgang im Speicher des Feldbusknotens die gleiche Datenbereichsgröße. Datenbereiche können in Größen von 1, 2, 4 oder 8 Byte angelegt werden (2 byte = 1 word).

  Mit der folgenden Beispielzeile wird der Slave in *ADbasic* für 5 Eingänge mit 4 Byte und 5 Ausgänge mit 4 Byte initialisiert.

```
Init_Profinet(5, 3, 5, 3, work_arr)
```

  Um den Slave im Konfigurations-Tool richtig einzurichten, müssen 5 Ausgänge und 5 Eingänge angelegt werden (jeweils einzeln mit 4 Bytes). Die unten stehende Grafik zeigt die Konfiguration beispielhaft.



Programmieren in *ADbasic*

Der Feldbusknoten wird mit den folgenden *ADbasic*-Befehlen (nicht aber mit *TiCoBasic*) komfortabel programmiert:

Bereich	Befehle
Reset, Datenbereiche initialisieren	`Init_ProfinetIO`
Daten schreiben und lesen, Nachrichtenverwaltung	`Run_ProfinetIO`

Die Befehle sind ab Seite 181 oder in der Online-Hilfe erläutert.

Die Initialisierung muss mit niedriger Priorität ablaufen, da sie einige Sekunden in Anspruch nimmt; bei hoher Priorität würde der PC nach einer bestimmten Zeit (time-out) die Kommunikation abbrechen. Aus dem gleichen Grund sollte auch das Schreiben und Lesen von Daten mit niedriger Priorität ablaufen.

Spezifikationen

Der Feldbusknoten entspricht dem Standard IEC 61158 (Feldbus). Dieser kann von der Profibus-Nutzerorganisation bezogen werden:

Profibus Nutzerorganisation e.V.
Haid-und-Neu-Str.7
76131 Karlsruhe
Tel.: +497219658590
Fax : +497219658589
www.profibus.com

Betriebszustände des Feldbusknotens

Die nachfolgende Tabelle zeigt die Betriebszustände, die der Feldbusknoten unterstützt und welches Verhalten er in den verschiedenen Zuständen zeigt.

Betriebs-Zustand	Verhalten
Setup	Initialisierung der Schnittstelle.
Wait	Slave wartet auf Busstart durch den Master.
Active	Der Profinet-Slave nimmt am zyklischen Datenverkehr teil.
Error	Fehler. Slave nimmt nicht am Busverkehr teil.
Exception	Schwerer interner Fehler. Slave nimmt nicht am Busverkehr teil.

Abb. 26 – Profinet: Betriebszustände

11 DeviceNet-Erweiterung

Die Erweiterung *Gold II-DeviceNet* stellt einen Feldbusknoten mit der Funktionalität eines DeviceNet-Slave zur Verfügung. Alle Einstellungen werden per Software vorgenommen.

Funktionsbeschreibung

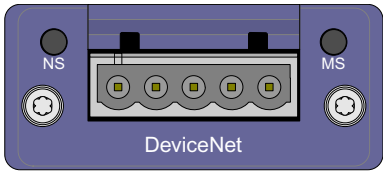
Nach dem Einschalten muss der Feldbusknoten initialisiert werden. Mit der Initialisierung werden die Stationsadresse (Slave-Knotenadresse) auf dem DeviceNet, die Baudrate und die Größe der Ein- und Ausgangsbereiche festgelegt.

Es gibt je einen Bereich für eingehende und für ausgehende Daten; jeder Bereich hat eine maximale Größe von 255 Byte. Die Begriffe „Eingang“ und „Ausgang“ sind aus Sicht des Feldbus-Controllers (Slave) zu sehen.

Sie legen für beide Bereiche separat fest, wieviele Datenbereiche in jedem Bereich vorhanden sind und wie groß die Datenbereiche sind.

Hardware

Die Pinbelegung der DeviceNet-Buchse entspricht der Spezifikation der Open DeviceNet Vendor Association (Dachorganisation ODVA).



Der DeviceNet-Bus muss am physikalischen Anfang und Ende der Busleitung mit einem Abschlusswiderstand abgeschlossen werden. Falls erforderlich, müssen Sie den Abschlusswiderstand selbst an den entsprechenden Datenleitungen des Feldbusknotens anbringen oder einen entsprechenden Steckverbinder verwenden.

Rechts und links neben der Buchse befinden sich zwei LEDs, die den Betriebszustand des Knotens im DeviceNet anzeigen, nämlich Netzwerkstatus (NS) und Modulstatus (MS; Modul steht hier für den Knoten).

LED	Status	Bedeutung
NS	Aus	Nicht online oder keine Stromversorgung.
	Grün	Knoten online, eine oder mehrere Verbindungen bestehen.
	Grün blinkend	Knoten online, keine Verbindung.
	Rot	Fehler.
	Rot blinkend	Eine oder mehrere Verbindungen haben eine Zeitüberschreitung (time-out).
	Rot-Grün wechselnd	Selbsttest.
MS	Aus	Keine Stromversorgung.
	Grün	Knoten arbeitet normal.
	Grün blinkend	Fehlende oder unvollständige Initialisierung, Eingriff erforderlich.
	Rot	Nicht zu behebender Fehler.
	Rot blinkend	Zu behebender Fehler.
	Rot-Grün wechselnd	Selbsttest.

Abb. 27 – DeviceNet: Bedeutung der LED

DeviceNet projektieren

Sie projektieren den DeviceNet mit einem – zum Bus-Master passenden – Konfigurations-Tool. Entnehmen Sie die genaue Vorgehensweise bei der Busprojektierung der Dokumentation Ihres Konfigurations-Tools.

Für die Projektierung steht das Geräteprofil des Feldbusknotens in der Datei `324-8172-EDS_ABCC_DEV_V_2_3.eds` im Ordner `C:\ADwin\Fieldbus\DeviceNet` zur Verfügung.

Stellen Sie sicher, bei der Projektierung die gleichen Einstellungen (Adresse, Baudrate, Anzahl und Größe der Datenbereiche) zu verwenden, die Sie bei der Initialisierung in *ADbasic* mit dem Befehl **`Init_DeviceNet`** eingestellt haben.

Beachten Sie, dass die Begriffe „Eingang“ und „Ausgang“ in *ADbasic* (Slave) und im Konfigurations-Tool (Master) vertauscht sein können. Ebenso kann beim Konfigurations-Tool die Reihenfolge beim Anlegen der Bereiche eine Rolle spielen.

Programmieren in *ADbasic*

Der Feldbusknoten wird mit den folgenden *ADbasic*-Befehlen (nicht aber mit *TiCoBasic*) komfortabel programmiert:

Bereich	Befehle
Stationsadresse, Baudrate und Datenbereiche initialisieren	**`Init_DeviceNet`**
Daten schreiben und lesen	**`Run_DeviceNet`**

Die Befehle sind ab Seite 187 oder in der Online-Hilfe erläutert.

Die Initialisierung muss mit niedriger Priorität ablaufen, da sie einige Sekunden in Anspruch nimmt; bei hoher Priorität würde der PC nach einer bestimmten

Zeit (time-out) die Kommunikation abbrechen. Aus dem gleichen Grund sollte auch das Schreiben und Lesen von Daten mit niedriger Priorität ablaufen.

12 EtherCAT-Erweiterung

Die Erweiterung *Gold II-EtherCAT* stellt einen Feldbusknoten mit der Funktionalität eines EtherCAT-Slave zur Verfügung. Alle Einstellungen werden per Software vorgenommen.

Funktionsbeschreibung

Nach dem Einschalten müssen Sie den Feldbusknoten in *ADbasic* initialisieren. Mit der Initialisierung wird die Größe der Ein- und Ausgangsbereiche festgelegt.

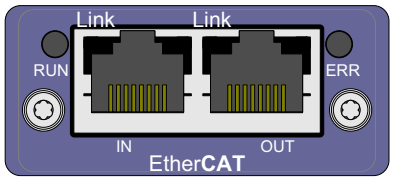
Es gibt je einen Bereich für eingehende und für ausgehende Daten; jeder Bereich hat eine maximale Größe von 254 Byte. Die Begriffe „Eingang“ und „Ausgang“ sind aus Sicht des Feldbus-Controllers zu sehen.

Sie legen für beide Bereiche separat fest, wie viele Datenbereiche in jedem Bereich vorhanden sind und wie groß die Datenbereiche sind.

Hardware

Die Schnittstelle hat je eine Buchse vom Typ RJ45 für den Dateneingang (`IN`) und den Datenausgang (`OUT`). An jeder Buchse ist oben links eine LED „`Link / Activity`“, die den Betriebszustand des Knotens im EtherCAT-Bus anzeigt. Die beiden weiteren LEDs sind ohne Funktion.

Rechts und links neben den Buchsen befinden sich LEDs, die den den Status der EtherCAT-Zustandsmaschine (`RUN`) und das Auftreten von Kommunkationsfehlern (`ERR`) anzeigen.



LED	Status	Bedeutung
`Link / Activity`	Aus	Nicht online (oder keine Stromversorgung).
	An	Feldbusknoten online, kein Datenaustausch.
	flackernd	Feldbusknoten online, mit Datenaustausch.
`RUN`	Aus	Status INIT: Die Schnittstelle wird initialisiert (oder keine Stromversorgung).
	blinkt grün	Status PRE-OP: Schnittstelle hat Kontakt zum Bus-Master.
	leuchtet einmal grün	Status SAFE-OP: Schnittstelle kann Daten vom Bus lesen, aber nicht senden.
	leuchtet grün	Status OP: Schnittstelle ist vollständig eingerichtet, Ein- und Ausgänge sind aktiv.
	leuchtet rot	Status EXCEPTION: Ausnahmesituation.
`ERR`	Aus	Kommunikation arbeitet ohne Fehler (oder keine Stromversorgung).
	blinkt rot	Fehler bei der Konfiguration.
	leuchtet einmal rot	Lokaler Fehler in der Schnittstelle; der EtherCAT-Status wurde geändert.
	leuchtet doppelt rot	Fehler durch Zeitüberschreitung (timeout).
	leuchtet rot	Kritischer Kommunikationsfehler.

Abb. 28 – EtherCAT: Bedeutung der LED

Wenn beide LEDs RUN und ERR rot leuchten, ist ein gravierender Fehler in der Schnittstelle aufgetreten. Melden Sie sich dann bitte beim Support von Jäger Messtechnik; die Adresse finden Sie auf der vorderen Umschlagseite des Handbuchs, innen.

EtherCAT projektieren

Sie projektieren den EtherCAT mit einem – zum Bus-Master passenden – Konfigurations-Tool. Für das folgende Beispiel wurde das Programm „TwinCAT System Manager“ der Firma Beckhoff als EtherCAT-Master verwendet.

Für andere Konfigurations-Tools gilt die folgende Ablaufbeschreibung entsprechend. Entnehmen Sie die genaue Vorgehensweise bei der Busprojektierung der Dokumentation Ihres Konfigurations-Tools.

– Kopieren Sie die Beschreibungsdateien ESI_ABCC*.XML und DDF_ABCC*.XML des Feldbusknotens von C:\ADwin\Fieldbus\EtherCAT in das Quellverzeichnis des Konfigurations-Tools.

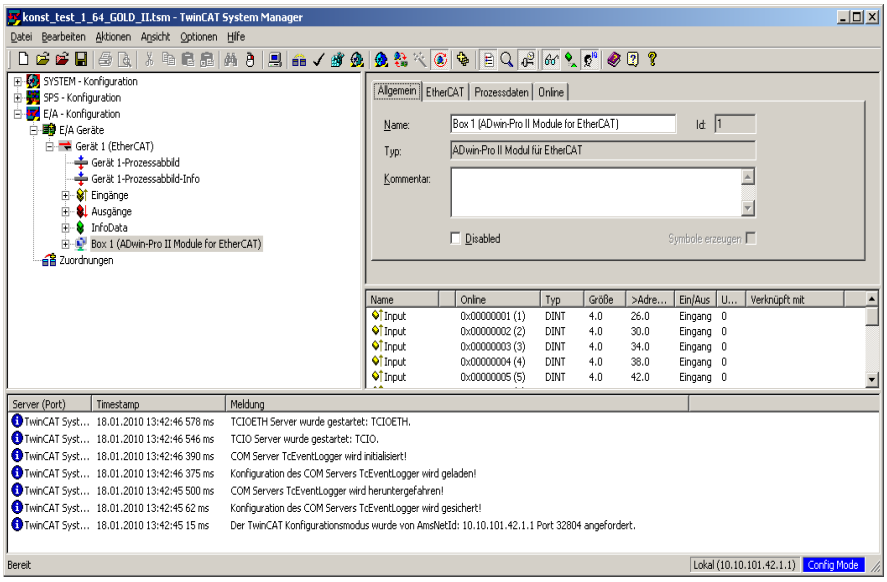
  Beim Starten lädt das Konfigurations-Tool die benötigten Informationen über einen neuen Slave aus der zugehörigen Beschreibungsdatei.

– Fügen Sie die *ADwin*-EtherCAT-Slave als Busteilnehmer zum EtherCAT-Bus hinzu.

  Im TwinCAT System Manager markieren Sie dazu den EtherCAT-Master und wählen im Kontextmenü (rechte Maustaste) den Menüpunkt Boxen scannen.
  Ihnen wird eine Liste aller Busteilnehmer angezeigt.

– Wählen Sie aus der Liste den *ADwin*-EtherCAT-Slave; damit ist der Slave als Busteilnehmer bestätigt.

  Danach könnte das EtherCAT-Layout wie folgt aussehen:



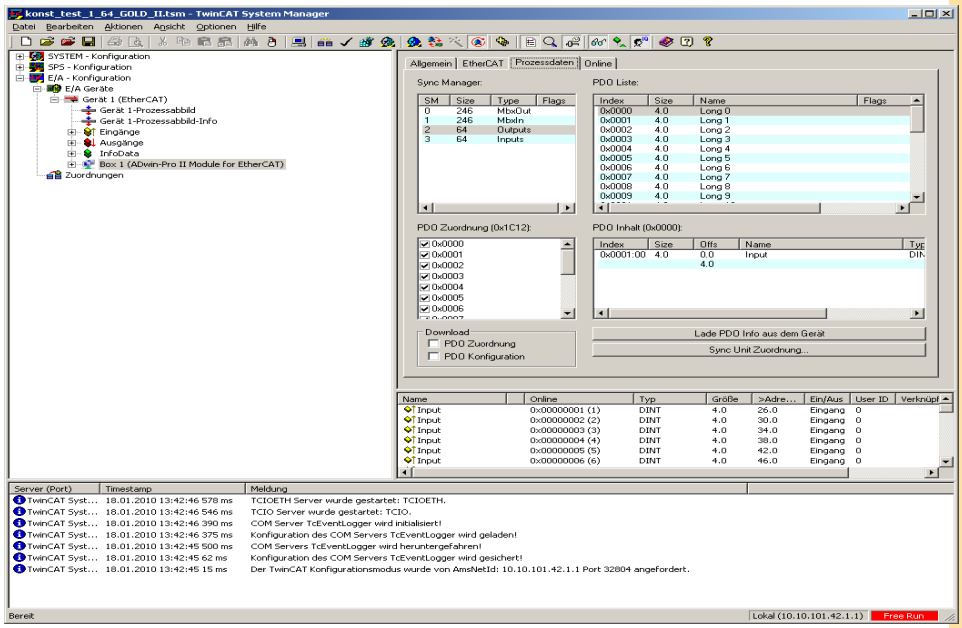
– Konfigurieren Sie den *ADwin*-EtherCAT-Slave in einem *ADbasic*-Programm mit dem Befehl **ECAT_Init**.

  Zwar können Sie den Slave auch im Konfigurations-Tool konfigurieren. Die Konfiguration in *ADbasic* muss dann aber trotzdem – und zwar mit den gleichen Einstellungen – ausgeführt werden.

– Lesen Sie die Konfiguration im Konfigurations-Tool aus.

  Im TwinCAT System Manager markieren Sie dazu den *ADwin*-EtherCAT-Slave und klicken auf die Schaltfläche `Lade PDO Info aus dem Gerät`.

  Danach könnte die Konfiguration des Slave wie folgt aussehen: 2 Bereiche zu 4 Byte als Eingänge und 2 Bereiche zu 4 Byte als Ausgänge.



Programmieren in *ADbasic*

Der Feldbusknoten wird mit den folgenden *ADbasic*-Befehlen (nicht aber mit *TiCoBasic*) programmiert:

Bereich	Befehle
Stationsadresse und Datenbereiche initialisieren	`ECAT_Init`
Daten schreiben und lesen	`ECAT_Run`

Die Befehle sind ab Seite 191 oder in der Online-Hilfe erläutert.

Die Initialisierung muss mit niedriger Priorität ablaufen, da sie einige Sekunden in Anspruch nimmt; bei hoher Priorität würde der PC nach einer bestimmten Zeit (time-out) die Kommunikation abbrechen. Aus dem gleichen Grund sollte auch das Schreiben und Lesen von Daten mit niedriger Priorität ablaufen.

Spezifikationen

Der Feldbusknoten entspricht den internationalen Standards IEC 61158 (Protokolle und Dienste) und IEC 61784-2 (Kommunikationsprofile für die spezifischen Geräteklassen). Nähere Informationen erhalten Sie von der EtherCAT-Nutzerorganisation:

EtherCAT Technology Group
Ostendstraße 196
90482 Nürnberg
Tel.: +499115405620
Fax : +499115405629
http://www.ethercat.org/

Betriebszustände der EtherCAT-Schnittstelle

Die nachfolgende Tabelle zeigt die Betriebszustände, die die EtherCAT-Schnittstelle unterstützt.

Betriebs-Zustand	Verhalten
Init	Der EtherCAT-Slave wird vom Bus-Master initialisiert.
PreOp	Die Schnittstelle nimmt am Datenverkehr teil, Ein- und Ausgänge sind noch inaktiv.
SafeOp	Die Schnittstelle kann Daten empfangen, die Ausgänge sind noch inaktiv.
Op	Die Schnittstelle ist vollständig betriebsbereit; Ein- und Ausgänge sind aktiv.

Abb. 29 – EtherCAT: Betriebszustände

13 Erweiterung Storage-16

Mit der Erweiterung *Gold II-Storage-16* enthält *ADwin-Gold II* eine Speicherkarte mit 16 GiB Speichervolumen sowie eine batteriegepufferte Echtzeituhr.

Speicherkarte

Sie können aus *ADbasic* oder *TiCoBasic* auf die Speicherkarte zugreifen. Ein gleichzeitiger Zugriff aus *ADbasic* und *TiCoBasic* ist nicht erlaubt.

Die Speicherkarte muss in jedem Fall in *ADbasic* initialisiert werden, auch wenn alle weiteren Zugriffe in *TiCoBasic* stattfinden.

Die Initialisierung muss mit niedriger Priorität ablaufen, da sie einige Sekunden in Anspruch nimmt; bei hoher Priorität würde der PC nach einer bestimmten Zeit (time-out) die Kommunikation abbrechen.

Es können Daten gelesen oder geschrieben werden. Daten werden immer in Blöcken von 128 Werten auf der Speicherkarte abgelegt. Wie bei der Initialisierung sollte auch das Schreiben und Lesen von Daten mit niedriger Priorität ablaufen.

Die Übertragungsrate bei einem Schreib-/Lesevorgang steigt mit der Anzahl der übertragenen Blöcke. Die optimale Datenrate wird in jedem Fall bei vollständig gelöschter Speicherkarte (nach `Media_Erase`) erreicht.

Echtzeituhr

Die Erweiterung enthält eine Echtzeituhr, um bestimmte Daten mit einem „Datumstempel“ versehen zu können.

Sie können Datum und Uhrzeit mit einfachen Befehlen setzen und auslesen. Die Uhr arbeitet unabhängig von der Speicherkarte.

Die Zeitangabe muss mit einem gültigen Datum und einer gültigen Uhrzeit gestellt werden; sie hat eine Auflösung von einer Sekunde. Bei den Datumsangaben werden Schaltjahre berücksichtigt.

Die Uhr ist batteriegepuffert (Typ CR1632) und kann bis zu 2 Jahre ohne externe Spannungsversorgung – also bei ausgeschaltetem Gerät – auskommen. Zum Erneuern der Batterie schicken Sie das *ADwin-Gold II* bitte an die Adresse auf der vorderen Umschlagseite, innen.

Programmieren in *ADbasic* und *TiCoBasic*

Die Speicherkarte wird mit den folgenden Befehlen komfortabel programmiert. Beachten Sie, dass manche Befehle nur in *ADbasic*, nicht aber in *TiCoBasic* verfügbar sind:

Bereich	Befehle	Verfügbar in
Speicherkarte initialisieren	`Media_Init`	*ADbasic*
Alle Daten auf der Speicherkarte löschen	`Media_Erase`	*ADbasic*
Daten lesen	`Media_Read`	*ADbasic* *TiCoBasic*
Daten schreiben	`Media_Write`	*ADbasic* *TiCoBasic*
Echtzeituhr lesen	`RTC_Get`	*ADbasic* *TiCoBasic*
Echtzeituhr setzen	`RTC_Set`	*ADbasic*

Die Befehle sind ab Seite 196 oder in der Online-Hilfe erläutert.

14 *ADwin-Gold II*-Boot

ADwin-Gold II-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-Gold II*-Boot nach dem Einschalten aus:

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

Bootloader deaktivieren

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 erneuten Einschalten ist die Bootloader-Option wieder aktiv.

Durch Beschreiben des Flash-EEPROM ohne Prozesse und nur mit der Datei `<ADwin11.btl>` wird das System nach dem erneuten Einschalten nur noch gebootet, aber ein Prozess kann nicht ausgeführt werden.

Wenn Sie *ADwin*-Software für Entwicklungsumgebungen aus dem *ADwin*-Software-Paket installieren, wird das Programm für die Bootloader-Option (ADethflash) automatisch kopiert.

Bei Standardinstallation finden Sie das Programm in dem Verzeichnis

`<C:\ADwin\Tools\Ethernet Interface\…>`.

Hilfe zur Ethernet-Schnittstelle

Hinweise zum Bootloader mit Ethernet-Interface finden Sie im Handbuch „*ADwin*-Installation“.

16000 Werte zur freien Verfügung

In Verbindung mit dem Ethernet-Interface mit Bootloader können Sie bis zu 16000 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 der Schaltfläche „`Info about eeprom support`“.

Beachten Sie: Der Prozessor T11 arbeitet bei Berechnungen mit dem Datentyp Float mit einer Rechengenauigkeit von 40 Bit. Beim Datenaustausch in / von dem Flash-EEPROM-Speicher werden auch bei Float-Werten nur 32 Bit übertragen.

15 Zubehör

Für das *ADwin-Gold II*-System ist folgendes Zubehör lieferbar:

– *Gold II*-Pow: externes 12 V-Netzteil (u.a. erforderlich für Notebook-Betrieb).

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

  Achten Sie auf ausreichende Schirmung des Ethernet-Kabels, um Störungen auf den Datenleitungen zu vermeiden. Störungen müssen vor dem Gehäuse über die Masse abgeleitet werden (siehe auch Kapitel 3 „Betriebliche Umgebung“).

– *Gold II*-Pow-DIN: externes Netzteil wie *Gold II*-Pow, jedoch für die Montage auf für DIN-Hutschienen.

– diverse Längen der Spannungsversorgungs- und Ethernet-Kabel

– *Gold II-Mount*: Gehäuseumbau zur Hutschienen-Montage in einem Schaltschrank mit isolierten Clipsen.

– Einzelner Stromversorgungs-Stecker für ein selbst-konfektioniertes Stromversorgungs-Kabel.

16 Software

Sie programmieren das *ADwin-Gold II*-System 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 sind auf folgenden Seiten beschrieben:

– Seite 61: Systemfunktionen
– Seite 70: Analoge Ein- und Ausgänge
– Seite 95: Digitale Ein- und Ausgänge
– Seite 117: Zähler
– Seite 133: SSI-Schnittstelle
– Seite 140: PWM-Ausgänge
– Seite 149: CAN-Schnittstelle
– Seite 164: RSxxx-Schnittstelle
– Seite 177: Profibus-Schnittstelle
– Seite 186: DeviceNet-Schnittstelle
– Seite 191: EtherCAT-Schnittstelle
– Seite 195: Echtzeituhr
– Seite 198: Storage-Erweiterung (ADbasic)

– Befehle für LS-Bus-Module (wie HSM24V) sind in einem separaten Handbuch und in der Online-Hilfe beschrieben.

Der *TiCo*-Prozessor im *ADwin-Gold II*-System kann ebenfalls auf die Ein- und Ausgänge und Schnittstellen zugreifen. Die Befehlsbeschreibungen für T11 in *ADbasic* gelten daher auch für *TiCoBasic*. Beachten Sie, dass zur gleichen Zeit immer nur einer der beiden Prozessoren *TiCo* oder T11 auf Ein-/Ausgänge zugreifen kann.

In jeder Befehlsbeschreibung ist mit Symbolen (siehe rechts) markiert, für welchen Prozessor (T11, *TiCo*) der Befehl gültig ist.

T11 TiCo

Beachten Sie bitte:

– Es gibt 2 verschiedene Include-Dateien:

  ADbasic:`ADwinGoldII.inc`; *TiCoBasic*: `GoldIITiCo.inc`.

– Der *TiCo*-Prozessor arbeitet mit 50 MHz Taktgeschwindigkeit (Einheit Processdelay: 20 ns), der T11 dagegen mit 300 MHz (Einheit Processdelay: 3,3 ns).

Online-Hilfe

Sie finden alle Befehlsbeschreibungen auch in den Online-Hilfen der Entwicklungsumgebungen *ADbasic* und *TiCoBasic*. Zum Aufruf bewegen Sie den Cursor auf das Befehlswort und drücken die Taste [F1].

16.1 Systemfunktionen

Dieser Abschnitt beschreibt Befehle für Systemfunktionen von *ADwin-Gold II*:

Event_Config

T11 TiCo

Event_Config konfiguriert den externen Event-Eingang.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Event_Config(min_hold,edge,prescale)
```

Parameter

min_hold	Mindestzeit, die eine Flanke anliegen muss, damit sie akzeptiert wird: 0: 15 ns (Default). 1: 50 ns.	LONG
edge	Flankentyp, der akzeptiert wird: 1: positive Flanke (Default). 2: negative Flanke. 3: positive und negative Flanke. 4: externes Event-Signal sperren.	LONG
prescale	Anzahl (1…255) an Flanken, nach der ein Event-Signal erzeugt wird (Default: 1).	LONG

Bemerkungen

- / -

Siehe auch

Event_Enable, Reset_Event

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

Init:
  Rem Event-Eingang konfigurieren für
  Rem Mindestzeit 15 ns, neg. Flanken, Event-Signal nach 4 Flanken
  Event_Config(0,2,4)
```

Event_Enable

Event_Enable legt die Event-Quelle fest.

Syntax

```
#Include ADwinGoldII.inc
Event_Enable(pattern)
```

Parameter

pattern	Bitmuster, in dem das gesetzte Bit die Event-Quelle festlegt: Bit 0: Schnittstelle CAN 1. Bit 1: Schnittstelle CAN 2. Bit 3: TiCo-Prozessor mit dem Befehl **Trigger_Event**. Bit 4: Event-Eingang an der Sub-D-Buchse DIO 00-15 (IN, siehe Seite 17); Voreinstellung. Alle weiteren Bits sind reserviert.	LONG

Bemerkungen

Es soll nur eine Event-Quelle (eines der Bits) aktiviert werden. Zum Deaktivieren aller Event-Quellen müssen alle Bits auf 0 gesetzt werden.

Zur erstmaligen Konfiguration einer CAN-Schnittstelle als Event-Quelle muss **En_CAN_Interrupt** verwendet werden; der Befehl setzt unter anderem die Event-Quelle auf CAN 1 oder CAN 2.

Siehe auch

Event_Config, En_CAN_Interrupt, Reset_Event

Gültig für

Gold II

Beispiel

```
#Include ADwinGoldII.inc

Init:
  Rem Schnittstelle CAN 2 als Event-Quelle festlegen
  Event_Enable(00010b)
```

Set_LED

T11 TiCo

`Set_LED` schaltet die LED (neben dem Power-Schalter) ein oder aus.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Set_LED(value)
```

Parameter

value	Gewünschter Schaltzustand der LED. 0: aus. 1: ein, grün leuchtend. 2: ein, rot leuchtend.	LONG

Bemerkungen

Die LED wird normalerweise durch den Prozess 15 ein- und ausgeschaltet. Um `Set_LED` zu verwenden, ist es daher sinnvoll, diesen Prozess zu stoppen.

Siehe auch

- / -

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Stop_Process(15)
  Set_LED(1)                  'LED einschalten, grün

Event:
  Rem …

Finish:
  Set_LED(0)                  'LED ausschalten
  Start_Process(15)
```

Calc_Processdelay

T11

Calc_Processdelay gibt die Anzahl der Prozesszyklen zu einer Prozessfrequenz zurück.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Calc_Processdelay(frequency)
```

Parameter

frequency	Prozessfrequenz in Hertz.	LONG
ret_val	Anzahl Prozesszyklen (= **Processdelay**).	LONG

Bemerkungen

- / -

Siehe auch

Processdelay

Gültig für

Gold II

Beispiel

```
#Include ADwinGoldII.Inc

Init:
  Rem Processdelay für 150kHz einstellen
  Processdelay = Calc_Processdelay(150000)
```

Watchdog_Init

T11 TiCo

`Watchdog_Init` konfiguriert und aktiviert den Watchdog-Zähler.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Watchdog_Init(enable, time, mode)
```

Parameter

`enable`	Betriebszustand des Watchdog-Zählers: 0: aus (Default). 1: ein.	LONG
`time`	Startwert (1…0FFFEh) des Watchdog-Zählers in Einheiten von 10µs.	LONG
`mode`	Bitmuster, das die Funktion des Watchdog-Zählers festlegt. Wenn die Bits 3:0 gesetzt sind, haben sie folgende Auswirkung: Bit 0: *ADwin* CPU (T11) stoppen. Bit 1: *TiCo*-Prozessor stoppen. Bit 2: Ausgang `Watchdog Out` auf TTL-Pegel low setzen. Bit 3: Ausgänge `OUT1…OUT8` und `DIO00…DIO32` auf TTL-Pegel low setzen.	LONG

Bemerkungen

Solange der Watchdog-Zähler aktiv ist, dekrementiert er seinen Zählerstand kontinuierlich. Wenn der Zählerstand 0 (Null) erreicht, nimmt das System eine Fehlfunktion an und der Watchdog löst die mit `mode` eingestellten Funktionen aus.

Setzen Sie den Watchdog-Zähler während des Zählvorgangs mindestens einmal zurück auf den Startwert, um die Funktion des Systems zu gewährleisten (Befehl `Watchdog_Reset`).

Wenn Bit 2 gesetzt (und der Zähler eingeschaltet) ist, gibt der Ausgang `Watchdog Out` den Status des Watchdog-Zählers an: TTL-Pegel High = Zähler ist aktiv, TTL-Pegel Low = Zählerstand 0 ist erreicht.
Wenn Bit 2 nicht gesetzt oder der Watchdog-Zähler deaktiviert ist, kann der Ausgang `Watchdog Out` mit `Watchdog_Standby_Value` eingestellt werden.

Siehe auch

Watchdog_Reset, Watchdog_Standby_Value, Watchdog_Status

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Watchdog_Init(1,0FFFEh,1111b)'enable and configure watchdog

Event:
  Watchdog_Reset()          'reset watchdog regularly
  Rem …
```

Watchdog_Reset

T11 TiCo

`Watchdog_Reset` setzt den Watchdog-Zähler zurück auf den Startwert. Der Zähler bleibt aktiv.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Watchdog_Reset()
```

Parameter

- / -

Bemerkungen

Solange der Watchdog-Zähler aktiv ist, dekrementiert er seinen Zählerstand kontinuierlich. Wenn der Zählerstand 0 (Null) erreicht, nimmt das System eine Fehlfunktion an und löst die mit `Watchdog_Init` eingestellten Funktionen aus.

Setzen Sie den aktiven Watchdog-Zähler während des Zählvorgangs mindestens einmal zurück auf den Startwert (siehe Watchdog_Init), um die Funktion Ihres Systems zu gewährleisten. Bei gesperrtem Zähler hat Watchdog_Reset keine Funktion.

Siehe auch

Watchdog_Init, Watchdog_Standby_Value, Watchdog_Status

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

Init:
  Rem
  Watchdog_Init(1,0FFFEh,1111b)'enable and configure watchdog

Event:
  Watchdog_Reset()          'reset watchdog regularly
  Rem …

Finish:
  Watchdog_Init(0,0,0)      'disable watchdog
```

Watchdog_ Standby_Value

T11 TiCo

Watchdog_Standby_Value setzt den Ausgang Watchdog Out auf einen definierten TTL-Pegel.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Watchdog_Standby_Value(level)
```

Parameter

level	TTL-Pegel des Ausgangs Watchdog Out: 0: TTL-Pegel low. 1: TTL-Pegel high.	LONG

Bemerkungen

Der Ausgang kann nur mit Watchdog_Standby_Value gesetzt werden, wenn der Watchdog-Zähler den Ausgang nicht selbst nutzt oder ausgeschaltet ist (siehe Watchdog_Init).

Nach dem Einschalten ist der Ausgang auf TTL-Pegel Low gesetzt.

Siehe auch

Watchdog_Init, Watchdog_Reset, Watchdog_Status

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc   'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic


Init:
  Watchdog_Init(1,0FFFEh,0001b)'enable and configure watchdog
  Watchdog_Standby_Value(1)'set watchdog value to level high

Event:
  Watchdog_Reset()          'reset watchdog regularly
  Rem …
```

Watchdog_Status

T11 TiCo

Watchdog_Status gibt den Status des Watchdog-Zählers zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Watchdog_Status()
```

Parameter

ret_val	Bitmuster mit dem Status des Watchdog-Zählers. Bit 0: Betriebszustand. Bit 0 = 0: Watchdog-Zähler ausgeschaltet. Bit 0 = 1: Watchdog-Zähler eingeschaltet. Bit 1: Fehlfunktion ausgelöst. Bit 1 = 0: Watchdog-Zähler ist aktiv. Bit 1 = 1: Zählerstand 0 ist erreicht, Watchdog-Zähler hat ausgelöst. Die übrigen Bits haben keine Funktion.	LONG

Bemerkungen

Solange der Watchdog-Zähler aktiv ist, dekrementiert er seinen Zählerstand kontinuierlich. Wenn der Zählerstand 0 (Null) erreicht, nimmt das System eine Fehlfunktion an und löst die mit **Watchdog_Init** eingestellten Funktionen aus.

Wenn der Watchdog-Zähler ausgeschaltet ist (Bit 0 = 0), dann ist Bit 1 ohne Funktion.

Siehe auch

Watchdog_Init, Watchdog_Reset, Watchdog_Standby_Value

Gültig für

Gold II

Beispiel

- / -

16.2 Analoge Ein- und Ausgänge

Dieser Abschnitt beschreibt Befehle zum Ansprechen der analogen Eingänge und Ausgänge auf ADwin-Gold II:

DAC

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
DAC (channel,value)
```

Parameter

dac_no	Nummer des analogen Ausgangs: Gold II: 1…2 Gold II-DA4: 1…4 Gold II-DA8: 1…8	LONG
value	Wert in Digits, der die auszugebende Spannung definiert: 0…65535	LONG

Bemerkungen

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

Siehe auch

Start_DAC, Write_DAC

Gültig für

Gold II, Gold II-DA4, Gold II-DA8

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc'für TiCoBasic
Rem Digitaler P-Regler
#Define set_to Par_1        'Sollwert
#Define gain FPar_2         'Verstärkungsfaktor
Rem #Define gain Par_2      'TiCoBasic: Verstärkung ganzzahlig
#Define diff Par_3          'Regelabweichung
#Define out_val Par_4       'Stellgröße

Init:
  Processdelay = 10000

Event:
  diff = set_to - ADC(1)     'Regelabweichung berechnen
  out_val = diff * gain      'Stellgröße berechnen
  DAC(1, out_val)            'Ausgabe der Stellgröße
```

Start_DAC

T11 TiCo

`Start_DAC` startet die Wandlung bzw. Ausgabe aller DAC.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Start_DAC()
```

Parameter

- / -

Bemerkungen

Der Wert im Ausgaberegister eines DAC wird mit `Write_DAC` gesetzt.

Siehe auch

DAC, Write_DAC

Gültig für

Gold II, Gold II-DA4, Gold II-DA8

Beispiel

```
REM Simultane Ausgabe von zwei verschiedenen Signalverläufen
REM auf den Ausgängen DAC 1 und 2.
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim i As Long

Init:
  Processdelay = 10000
  i=0
  Write_DAC(1,i)              'Ausgaberegister DAC1 setzen
  Write_DAC(2,65535-i)        'Ausgaberegister DAC2 setzen

Event:
  Start_DAC()                 'Ausgabe auf allen DAC starten
  Write_DAC(1,i)              'Ausgaberegister DAC1 setzen
  Write_DAC(2,65535-i)        'Ausgaberegister DAC2 setzen
  Inc(i)
  If (i=65535) Then i=0
```

Write_DAC

T11 TiCo

Write_DAC schreibt einen Digitalwert in das Ausgaberegister eines bestimmten DAC.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Write_DAC(dac_no,value)
```

Parameter

dac_no	Nummer des analogen Ausgangs: Gold II: 1…2 Gold II-DA4: 1…4 Gold II-DA8: 1…8	LONG
value	Wert (0…65535) in Digits, der die auszugebende Spannung definiert.	LONG

Bemerkungen

Die Wandlung des Registerwerts in eine Ausgangsspannung wird mit **Start_DAC** gestartet.

Siehe auch

DAC, Start_DAC

Gültig für

Gold II, Gold II-DA4, Gold II-DA8

Beispiel

```
REM Simultane Ausgabe von vier verschiedenen Signalverläufen
REM auf den DAC 1, 2, 3 und 4 (Gold II-DA4).
REM Die Signalverläufe sind in vier DATA-Feldern abgelegt und
REM können vor dem Programmstart vom PC übergeben werden.
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim i As Long                'Deklaration
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_4[1000] As Long

Init:
  Processdelay = 10000
  i=1
  Write_DAC(1,Data_1[i])     'Ausgaberegister DAC1 setzen
  Write_DAC(2,Data_2[i])     'Ausgaberegister DAC2 setzen
  Write_DAC(3,Data_3[i])     'Ausgaberegister DAC3 setzen
  Write_DAC(4,Data_4[i])     'Ausgaberegister DAC4 setzen

Event:
  Start_DAC()                'Ausgabe auf allen DAC starten
  Write_DAC(1,Data_1[i])     'Ausgaberegister DAC1 setzen
  Write_DAC(2,Data_2[i])     'Ausgaberegister DAC2 setzen
  Write_DAC(3,Data_3[i])     'Ausgaberegister DAC3 setzen
  Write_DAC(4,Data_4[i])     'Ausgaberegister DAC4 setzen
  INC(i)
  IF (i>1000) Then i=1
```

ADC

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = ADC(channel)
```

Parameter

channel	Nummer (1…16) des analogen Eingangskanals.	LONG
ret_val	Messergebnis in Digits (0…65535).	LONG

Bemerkungen

Der Befehl **ADC24** gibt Messwerte mit 24 Bit Auflösung zurück.

ADC ist eine Zusammenstellung aufeinander folgender Funktionen:

- **Set_MUX1**, **Set_MUX2**: 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.
- **Read_ADC**: Den Digitalwert aus einem Register lesen und zurückgeben.

Die Einschwingzeit der Multiplexer beträgt beim maximalen Spannungssprung von 20 Volt höchstens 2µs. Die Wandlungszeit der ADC beträgt jeweils 2µs.

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

In folgenden Fällen sollten Sie anstelle der Anweisung **ADC** die Anweisungen **Set_MUX1/2**, **Start_Conv**, **Wait_EOC** und **Read_ADC** verwenden:

- 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.
  Beispielsweise können mehrere Wandlungen schneller als mit **ADC** durchgeführt werden, wenn Sie die Einzelfunktionen geschickt einsetzen, siehe Wartezeiten nutzen (Seite 129).

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

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

Der Messbereich ist hier 20V (Eingangsspannung: -10V … 10V).

Siehe auch

ADC24, Read_ADC, Set_Mux1, Set_Mux2, Start_Conv, Wait_EOC

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim iw As Long              'Deklaration
Init:
  Processdelay = 10000

Event:
  REM Spannung am analogen Eingang 1 messen
  iw = ADC(1)
  REM Messwert in globale Variable schreiben, damit er
  REM vom PC gelesen werden kann.
  Par_1 = iw
```

ADC24

T11 TiCo

ADC24 misst die Spannung an einem analogen Eingang und gibt eine (dem Messergebnis entsprechende) ganze Zahl zurück. Der Rückgabewert ist auf 24 Bit normiert.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = ADC24(channel)
```

Parameter

channel	Nummer (1…16) des analogen Eingangskanals.	LONG
ret_val	Messergebnis in Digits (0…16777215 = 2^{24}-1).	LONG

Bemerkungen

Messwerte mit 16 Bit Auflösung gibt **ADC** zurück.

Der Rückgabewert von **ADC24** enthält in den Bits 23:6 den 18 Bit-Messwert, die Bits 5:0 sind immer Null (siehe auch Seite 14).

Bitnr.	31:24	23:16	15:6	5:0
Inhalt	0	18-Bit Messwert in den Bits 23:6		0

ADC24 ist eine Zusammenstellung aufeinander folgender Funktionen:

- **Set_MUX1**, **Set_MUX2**: 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.
- **Read_ADC24**: Den Digitalwert aus einem Register lesen und zurückgeben.

Die Einschwingzeit der Multiplexer beträgt beim maximalen Spannungssprung von 20 Volt höchstens 2µs. Die Wandlungszeit der ADC beträgt jeweils 2µs.

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

In folgenden Fällen sollten Sie anstelle der Anweisung **ADC24** die Anweisungen **Set_MUX1/2**, **Start_Conv**, **Wait_EOC** und **Read_ADC24** verwenden:

- 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.
  Beispielsweise können mehrere Wandlungen schneller als mit **ADC** durchgeführt werden, wenn Sie die Einzelfunktionen geschickt einsetzen, siehe Wartezeiten nutzen (Seite 129).

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

$$\text{Spannung} = \frac{\text{Messbereich}}{16777216} \cdot \left(\text{Digits} - 8388608_{\text{bipolar}}\right)$$

Der Messbereich ist hier 20V (Eingangsspannung: -10V … 10V).

Siehe auch

ADC, Read_ADC24, Set_Mux1, Set_Mux1, Start_Conv, Wait_EOC

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim iw As Long              'Deklaration
Init:
  Processdelay = 10000

Event:
  REM Spannung am analogen Eingang 1 messen
  iw = ADC24(1)
  REM Messwert in globale Variable schreiben, damit er
  REM vom PC gelesen werden kann.
  Par_1 = iw
```

Read_ADC

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Read_ADC(adc_no)
```

Parameter

adc_no	Nummer des zu lesenden Wandlers (1, 2)	LONG
ret_val	Messwert in Digits (0…65535), der der anliegenden Spannung am Wandler entspricht.	LONG

Bemerkungen

Read_ADC24 liest einen gewandelten Wert mit 24 Bit Auflösung.

Siehe auch

ADC, Read_ADC24, Set_Mux1, Set_Mux2, Start_Conv, Wait_EOC

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Processdelay = 10000

Event:
  REM Multiplexer setzen: ADC1 auf Kanal 3, ADC2 auf Kanal 4
  Set_Mux1(001b)
  Set_Mux2(001b)
  Rem IO-Zugriff für 2 µs (MUX-Einschwingzeit) unterbrechen
  IO_Sleep(200)
  Rem Wartezeit nutzen (nicht für Zugriffe auf IOs oder
  Rem ext. Speicher)
  Rem …
  Start_Conv(11b)           'Wandlung für beide ADC starten
  Wait_EOC(11b)             'Ende der Wandlungen abwarten
  Par_1 = Read_ADC(1)       'Wert von ADC1 einlesen
  Par_2 = Read_ADC(2)       'Wert von ADC2 einlesen
```

Read_ADC24

T11 TiCo

Read_ADC24 gibt einen gewandelten Wert von einem ADC mit 24 Bit Auflösung zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Read_ADC24(adc_no)
```

Parameter

adc_no	Nummer des zu lesenden Wandlers (1, 2).	LONG
ret_val	Messwert in Digits (0…16777215 = 2^{24}-1), der der anliegenden Spannung am Wandler entspricht.	LONG

Bemerkungen

Read_ADC liest einen gewandelten Wert mit 16 Bit Auflösung.

Siehe auch

ADC24, Read_ADC, Set_Mux1, Set_Mux2, Start_Conv, Wait_EOC

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Processdelay = 10000

Event:
  REM Multiplexer setzen: ADC1 auf Kanal 3, ADC2
  REM auf Kanal 4 (ohne Verstärkung)
  SET_MUX1(001b)
  SET_MUX2(001b)
  Rem IO-Zugriff für 2 µs (MUX-Einschwingzeit) unterbrechen
  IO_Sleep(200)
  Rem Wartezeit nutzen (nicht für Zugriffe auf I/Os oder
  Rem ext. Speicher)
  Rem …
  Start_Conv(11b)           'Wandlung für beide ADC starten
  Wait_EOC(11b)             'Ende der Wandlungen abwarten
  Par_1 = Read_ADC24(1)     'Wert von ADC1 einlesen
  Par_2 = Read_ADC24(2)     'Wert von ADC2 einlesen
```

Set_Mux1

T11 TiCo

`Set_Mux1` setzt den Multiplexer am ADC1 auf den gewählten Messkanal und stellt die Verstärkung ein.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Set_Mux1(pattern)
```

Parameter

`pattern` Bitmuster zur Zuordnung von Messkanal und Verstärkung — `LONG`

Bitnr.	31:5	4 3	2 1 0
	–	PGA 1	MUX 1

PGA 1 — Die Bits 4:3 bestimmen den Verstärkungsfaktor:

`00`: Faktor 1
`01`: Faktor 2
`10`: Faktor 4
`11`: Faktor 8

MUX 1 — Die Bits 2:0 bestimmen den Messkanal:

`000`: Kanal 1
`001`: Kanal 3
`010`: Kanal 5
`011`: Kanal 7
`100`: Kanal 9
`101`: Kanal 11
`110`: Kanal 13
`111`: Kanal 15

Bemerkungen

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

Die Einschwingzeit der Multiplexer beträgt beim maximalen Spannungssprung von 20 Volt höchstens 2µs. Die Wandlungszeit der ADC beträgt jeweils 2µs.

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

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, ADC24, Read_ADC, Read_ADC24, Set_Mux2, Start_Conv, Wait_EOC

Gültig für

Gold II

Beispiel

Sie möchten den Multiplexer des ADC1 auf den Kanal 5 und die Verstärkung 8 einstellen. Hierzu benötigen Sie das Bitmuster: 11010b (dezimal: 26).

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim val As Long

Event:
  Set_Mux1(11010b)          'Mux 1: Kanal und Verstärkung setzen
  Rem IO-Zugriff für 2 µs (MUX-Einschwingzeit) unterbrechen
  IO_Sleep(200)
  Rem Wartezeit nutzen (nicht für Zugriffe auf IOs oder
  Rem ext. Speicher)
  Rem …
  Start_Conv(1)             'Start AD-Wandlung ADC1
  Wait_EOC(1)               'Wandlungsende des ADC1 abwarten
  val = Read_ADC(1)         'Wert von ADC1 einlesen
```

Set_Mux2

T11 TiCo

`Set_Mux2` setzt den Multiplexer am ADC2 auf den gewählten Messkanal und stellt die Verstärkung ein.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Set_Mux2(pattern)
```

Parameter

`pattern` — Bitmuster zur Zuordnung von Messkanal und Verstärkung | LONG

Bitnr.	31:5	4 3	2 1 0
	–	PGA 2	MUX 2

PGA 2 — Die Bits 4:3 bestimmen den Verstärkungsfaktor:

`00`: Faktor 1
`01`: Faktor 2
`10`: Faktor 4
`11`: Faktor 8

MUX 2 — Die Bits 2:0 bestimmen den Messkanal:

`000`: Kanal 2
`001`: Kanal 4
`010`: Kanal 6
`011`: Kanal 8
`100`: Kanal 10
`101`: Kanal 12
`110`: Kanal 14
`111`: Kanal 16

Bemerkungen

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

Die Einschwingzeit der Multiplexer beträgt beim maximalen Spannungssprung von 20 Volt höchstens 2 µs. Die Wandlungszeit der ADC beträgt jeweils 2 µs.

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, ADC24, Read_ADC, Read_ADC24, Set_Mux1, Start_Conv, Wait_EOC

Gültig für

Gold II

Beispiel

Sie möchten den Multiplexer des ADC2 auf den Kanal 10 und die Verstärkung 2 einstellen. Hierzu benötigen Sie das Bitmuster: `01100b` (dezimal: 12).

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim val As Long

Event:
  Set_Mux2(01100b)          'Mux 2: Kanal und Verstärkung setzen
  Rem IO-Zugriff für 2 µs (MUX-Einschwingzeit) unterbrechen
  IO_Sleep(200)
  Rem Wartezeit nutzen (nicht für Zugriffe auf IOs oder
  Rem ext. Speicher)
  Rem …
  Start_Conv(2)             'Start AD-Wandlung ADC2
  Wait_EOC(2)               'Wandlungsende des ADC2 abwarten
  val = Read_ADC(2)         'Wert von ADC2 einlesen
```

Start_Conv

T11 TiCo

Start_Conv kann die Wandlung an einem oder mehreren A/D-Wandlern starten.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Start_Conv(pattern)
```

Parameter

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

Bitnr.	1	0
ADC1	–	x
ADC2	x	–

Bemerkungen

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, ADC24, Read_ADC, Read_ADC24, Set_Mux1, Set_Mux2, Wait_EOC

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim val1 As Long
Init:
  Processdelay = 10000

Event:
  SET_MUX1(0)                'Multiplexer 1 auf Kanal 1 setzen
  Rem IO-Zugriff für 2 µs (MUX-Einschwingzeit) unterbrechen
  IO_Sleep(200)
  Rem Wartezeit nutzen (nicht für Zugriffe auf IOs oder
  Rem ext. Speicher)
  Rem …
  Start_Conv(1)              'Start ADC1 A/D-Wandlung
  Wait_EOC(1)                'Ende der Wandlung abwarten
  val1 = Read_ADC(1)         'Wert auslesen
```

Die Einschwingzeit der Multiplexer beträgt beim maximalen Spannungssprung von 20 Volt höchstens 2 µs. Die Wandlungszeit der ADC beträgt jeweils 2 µs.

Wait_EOC

T11 TiCo

`Wait_EOC` wartet auf das Ende der Wandlung an den bestimmten A/D-Wandlern.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Wait_EOC(pattern)
```

Parameter

pattern	Bitmuster, für die Wandler, auf die gewartet wird (nur Bits 1:0 verwendbar).	LONG

Bitnr.	1	0
ADC1	–	x
ADC2	x	–

Bemerkungen

- / -

Siehe auch

ADC, ADC24, Read_ADC, Read_ADC24, Set_Mux1, Set_Mux2, Start_Conv

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim val1 As Long
Init:
  Processdelay = 10000

Event:
  SET_MUX2(001b)            'Multiplexer 2 auf Kanal 4 setzen
  Rem IO-Zugriff für 2 µs (MUX-Einschwingzeit) unterbrechen
  IO_Sleep(200)
  Rem Wartezeit nutzen (nicht für Zugriffe auf IOs oder
  Rem ext. Speicher)
  Rem …
  Start_Conv(10b)           'Start ADC1 A/D-Wandlung
  Wait_EOC(10b)             'Ende der Wandlung abwarten
  val1 = Read_ADC(2)        'Wert auslesen
```

Die Einschwingzeit der Multiplexer beträgt beim maximalen Spannungssprung von 20 Volt höchstens 2µs. Die Wandlungszeit der ADC beträgt jeweils 2µs.

Seq_Mode

T11 TiCo

Seq_Mode stellt den Arbeitsmodus der Ablaufsteuerung für einen ADC ein.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Mode(adc_no, mode)
```

Parameter

adc_no	Nummer (1, 2) des ADC.	LONG
mode	Arbeitsmodus der Ablaufsteuerung: 0: Einzelmessung (Default), ohne Ablaufsteuerung. 1: Modus „single shot“, einfacher Messzyklus. 2: Modus „continuous“, regelmäßiger Messzyklus.	LONG

Bemerkungen

Nach dem Einschalten ist der Modus 0 aktiv.

Die Modi 1 und 2 aktivieren die Ablaufsteuerung des angegebenen ADC, Einzelmessungen mit **ADC** sind dann (auf diesem ADC) nicht möglich. Die Ablaufsteuerung führt an mehreren Kanälen nacheinander eine Wandlung durch. Die Kanäle werden mit **Seq_Select** ausgewählt.

Die Modi unterscheiden sich wie folgt:

Modus	Art der Messung
0 Normal:	Standard: Einzelne Messung an einem Kanal ohne Ablaufsteuerung, siehe **ADC**.
1 single shot:	Die Ablaufsteuerung wird mit **Seq_Start** gestartet; die Ablaufsteuerung endet, sobald die gewählten Kanäle je einmal gewandelt sind. Die Messwerte der Ablaufsteuerung werden mit **Seq_Read** eingelesen.
2 continuous:	Die Ablaufsteuerung wandelt ununterbrochen neue Messwerte an den gewählten Kanälen, d.h. Wandlung und Prozesszyklus laufen asynchron. Die Wandlung wird mit **Seq_Start** gestartet. Im Prozesszyklus wird mit **Seq_Read** der jeweils neueste Messwert gelesen.

Siehe auch

ADC, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)              'continuous mode
  Seq_Set_Delay(2,125)       'waiting time 2.5 µs (125*20ns)
  Seq_Set_Gain(2,0)          'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)              'reset to standard mode
```

Seq_Read

T11 TiCo

Seq_Read gibt den aktuellen Messwert (16 Bit) eines Eingangskanals aus der Ablaufsteuerung zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Seq_Read(channel)
```

Parameter

channel	Eingangskanal (1…16).	LONG
ret_val	Messwert (16 Bit).	LONG

Bemerkungen

Diese Anweisung gibt nur einen sinnvollen Wert zurück, wenn vorher die Ablaufsteuerung mit **Seq_Mode** und **Seq_Start** sowie der Eingangskanal mit **Seq_Select** aktiviert wurde.

Siehe auch

Seq_Mode, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)              'continuous mode
  Seq_Set_Delay(2,125)       'waiting time 2.5 µs (125*20ns)
  Seq_Set_Gain(2,0)          'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)              'reset to standard mode
```

Seq_Read8

T11

Seq_Read8 gibt die Messwerte (16 Bit) der Eingangskanäle 1…8 aus der Ablaufsteuerung zurück.

Syntax

```
#Include ADwinGoldII.inc
Seq_Read8(array[], array_idx)
```

Parameter

array[]	Feld, das die Messwerte der Eingangskanäle 1…8 aufnimmt.	ARRAY LONG
array_idx	Feldelement, das den ersten der Messwerte aufnimmt.	LONG

Bemerkungen

Seq_Read8 gibt nur sinnvolle Werte zurück, wenn vorher die Ablaufsteuerung mit **Seq_Mode** und **Seq_Start** aktiviert wurde, und auch nur für die Eingangskanäle, die mit **Seq_Select** aktiviert wurden.

Wenn weniger als 4 Eingangskanäle gelesen werden sollen, ist eine Schleife mit dem Befehl **Seq_Read** schneller als **Seq_Read8**.

Siehe auch

Seq_Mode, Seq_Read, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Gültig für

Gold II

Beispiel

```
#Include ADwinGoldII.inc
Dim Data_1[16] As Long At DM_Local

Init:
  Rem settings for sequential controls
  Seq_Mode(1,2) : Seq_Mode(2,2)'continuous mode
  Rem waiting time 2.5 µs (125*20ns)
  Seq_Set_Delay(1,125) : Seq_Set_Delay(2,125)
  Seq_Set_Gain(1,0) : Seq_Set_Gain(2,0)'gain factor 1
  Seq_Select(0FFh)          'select channels 1…8
  Rem start sequential controls of ADC1 and ADC2
  Seq_Start(11b)

Event:
  Rem read values of channels 1…8
  Seq_Read8(Data_1,1)

Finish:
  Seq_Mode(1,0)             'reset to standard mode
  Seq_Mode(2,0)             'reset to standard mode
```

Seq_Read16

T11

Seq_Read16 gibt die Messwerte (16 Bit) der Eingangskanäle 1…16 aus der Ablaufsteuerung zurück.

Syntax

```
#Include ADwinGoldII.inc
Seq_Read16(array[], array_idx)
```

Parameter

array[]	Feld, das die Messwerte der Eingangskanäle 1…16 aufnimmt.	ARRAY LONG
array_idx	Feldelement, das den ersten der Messwerte aufnimmt.	LONG

Bemerkungen

Seq_Read16 gibt nur sinnvolle Werte zurück, wenn vorher die Ablaufsteuerung mit **Seq_Mode** und **Seq_Start** aktiviert wurde, und auch nur für die Eingangskanäle, die mit **Seq_Select** aktiviert wurden.

Siehe auch

Seq_Mode, Seq_Read, Seq_Read8, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Gültig für

Gold II

Beispiel

```
#Include ADwinGoldII.inc
Dim Data_1[16] As Long At DM_Local

Init:
  Rem settings for sequential controls
  Seq_Mode(1,2) : Seq_Mode(2,2)'continuous mode
  Rem waiting time 2.5 µs (125*20ns)
  Seq_Set_Delay(1,125) : Seq_Set_Delay(2,125)
  Seq_Set_Gain(1,0) : Seq_Set_Gain(2,0)'gain factor 1
  Seq_Select(0FFFFh)        'select all channels
  Rem start sequential controls of ADC1 and ADC2
  Seq_Start(11b)

Event:
  Rem read values of channels 1…16
  Seq_Read16(Data_1,1)

Finish:
  Seq_Mode(1,0)             'reset to standard mode
  Seq_Mode(2,0)             'reset to standard mode
```

Seq_Set_Delay

T11 TiCo

Seq_Set_Delay stellt die Einschwingzeit der Ablaufsteuerung für einen ADC ein.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Set_Delay(adc_no, mux_time)
```

Parameter

adc_no	Nummer (1, 2) des ADC.	LONG
mux_time	Anzahl der Zeiteinheiten für die Einschwingzeit der Ablaufsteuerung: 100…2^31: Zeit in Einheiten von 20ns.	LONG

Bemerkungen

Der Befehl ist nur sinnvoll, wenn die Ablaufsteuerung mit **Seq_Mode** aktiviert ist.

Der Zeitabstand zwischen 2 Wandlungen der Ablaufsteuerung ist gleich der Einschwingzeit.

Nach dem Einschalten ist die Multiplexer-Einschwingzeit auf 2µs eingestellt (entspricht mux_time = 100). Tendenziell ergeben kürzere Einschwingzeiten ungenauere und längere Einschwingzeiten genauere Messergebnisse. Der angegebene Wertebereich soll daher nicht unterschritten werden.

Wenn der Innenwiderstand der Spannungsquelle des Messsignals zu groß ist, genügt die voreingestellte Einschwingzeit des Multiplexers nicht mehr aus für eine genaue Messung. Sie können die Einschwingzeit des Multiplexers mit dem Parameter mux_time verlängern.

Siehe auch

Seq_Mode, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)              'continuous mode
  Seq_Set_Delay(2,125)       'waiting time 2.5 µs (125*20ns)
  Seq_Set_Gain(2,0)          'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)              'reset to standard mode
```

Seq_Set_Gain

T11 TiCo

Seq_Set_Gain stellt den Verstärkungsfaktor der Ablaufsteuerung für einen ADC ein.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Set_Gain(adc_no, gain)
```

Parameter

adc_no	Nummer (1, 2) des ADC.	LONG
gain	Verstärkungsfaktor (nur bei aktiver Ablaufsteuerung): 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.	LONG

Bemerkungen

Der Befehl ist nur sinnvoll, wenn die Ablaufsteuerung mit **Seq_Mode** aktiviert ist.

Siehe auch

Seq_Mode, Seq_Set_Delay, Seq_Select, Seq_Read, Seq_Start, Seq_Status

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)              'continuous mode
  Seq_Set_Delay(2,125)       'waiting time 2.5 µs (125*20ns)
  Seq_Set_Gain(2,0)          'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)              'reset to standard mode
```

Seq_Select

T11 TiCo

Seq_Select wählt die Eingangskanäle aus, die mit den Ablaufsteuerungen der beiden ADC gewandelt werden.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Select(pattern)
```

Parameter

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

Bitnr.	31:16	15	…	2	1	0
Kanal-Nr.	–	16	…	3	2	1

Bemerkungen

Die Auswahl von Eingangskanälen hat nur eine Bedeutung, wenn die Ablaufsteuerung des entsprechenden ADC mit **Seq_Mode** aktiviert ist.

Auch wenn die Eingangskanäle mit **Seq_Select** gemeinsam ausgewählt werden, arbeiten die Ablaufsteuerungen der beiden ADC unabhängig voneinander. ADC 1 wandelt nur Eingänge mit ungerader Nummer, ADC 2 nur Eingänge mit gerader Nummer.

Die Eingangskanäle werden automatisch in aufsteigender Reihenfolge der Kanalnummern gewandelt, d. h. die Ablaufsteuerungen wandeln den Kanal mit der jeweils niedrigsten Nummer zuerst.

Siehe auch

Seq_Mode, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Start, Seq_Status

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)              'continuous mode
  Seq_Set_Delay(2,125)       'waiting time 2.5 µs (125*20ns)
  Seq_Set_Gain(2,0)          'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)              'reset to standard mode
```

Seq_Start

T11 TiCo

Seq_Start startet oder stoppt die Ablaufsteuerungen für beide ADC.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Start(adc_pattern)
```

Parameter

adc_pattern	Bitmuster zum Auswählen der ADC: Bit = 0: Ablaufsteuerung für den ADC deaktivieren. Bit = 1: Ablaufsteuerung für den ADC aktivieren.	LONG

Bit	31:2	1	0
ADC-Nr.	–	2	1

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher die Ablaufsteuerung mit **Seq_Mode** aktiviert wurde.

Wenn beide Ablaufsteuerungen genutzt werden, sollten sie mit den gleichen Einstellungen (**Seq_Set_Delay**) arbeiten und gleichzeitig gestartet werden. Damit stellen Sie sicher, dass die Messungen der beiden Ablaufsteuerungen synchron stattfinden und die Messwerte zur gleichen Zeit gelesen werden können.

Siehe auch

Seq_Mode, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Status

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim Data_1[1600] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential controls
  Seq_Mode(1,2) : Seq_Mode(2,2)'continuous mode
  Rem waiting time 2.5 µs (125*20ns)
  Seq_Set_Delay(1,125) : Seq_Set_Delay(2,125)
  Seq_Set_Gain(1,0) : Seq_Set_Gain(1,0) 'gain factor 1
  Rem select channels 1..7, i.e. channels 1,3,5,7 for sequential
  Rem control 1 and channels 2,4,6 for seq. control 2
  Seq_Select(07Fh)
  Rem start both sequential controls
  Seq_Start(11b)
  i = 1

Event:
  Rem read current values of selected channels.
  Rem Channel 8 is not read since it was not selected before.
  Seq_Read8(Data_1,i) : i = i + 8
  If (i > 1600) Then i = 1

Finish:
  Seq_Mode(1,0)              'reset to standard mode
  Seq_Mode(2,0)              'reset to standard mode
```

Seq_Status

T11 TiCo

`Seq_Status` gibt zurück, ob der einfache Messzyklus der Ablaufsteuerung eines ADC beendet ist.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Seq_Status(adc_no)
```

Parameter

adc_no	Nummer (1, 2) des ADC.	LONG
ret_val	Status der Ablaufsteuerung für den Modus „single shot“: 0: Einfacher Messzyklus ist beendet. 1: Einfacher Messzyklus ist noch nicht beendet.	LONG

Bemerkungen

Der Rückgabewert ist nur sinnvoll, wenn die Ablaufsteuerung mit `Seq_Mode` im Modus „single shot“ aktiviert ist.

Siehe auch

Seq_Mode, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 1
  Seq_Mode(1,1)              'single shot mode
  Seq_Set_Delay(1,125)       'waiting time 2.5 µs (125*20ns)
  Seq_Set_Gain(1,1)          'gain factor 2
  Rem select all channels. selection is valid only for active
  Rem sequential control 1 = odd numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC1
  Seq_Start(01b)

Event:
  Do
  Until (Seq_Status(1)=0)
  Rem read values of odd channels
  For i = 1 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i
  Rem start sequential control of ADC1
  Seq_Start(01b)

Finish:
  Seq_Mode(2,0)              'reset to standard mode
```

16.3 Digitale Ein- und Ausgänge

Dieser Abschnitt beschreibt Befehle zum Ansprechen der digitalen Eingänge und Ausgänge auf *ADwin-Gold II*:

Conf_DIO

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Conf_DIO(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 Ihnen die Konfiguration `Conf_DIO(1100b)`, d.h. DIO15:DIO00 sind Eingänge und DIO31:DIO16 sind Ausgänge (siehe auch Seite 17).

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, Digin_Long, Digin_Word1, Digin_Word2, Digout, Digout_Bits, Digout_Long, Digout_Word1, Digout_Word2, Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic

Init:
    Rem Konfiguriere DIO00…DIO15 als Eingänge
    Rem und DIO16…DIO31 als Ausgänge
    Conf_DIO(1100b)
```

Digin

T11 TiCo

Digin gibt den TTL-Pegel eines Digitaleingangs zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin(channel)
```

Parameter

channel	Nummer (0…31) des Digitaleingangs.	LONG
ret_val	Der TTL-Pegel des gewählten Digitaleingangs: 1: TTL-Pegel high liegt an. 0: TTL-Pegel low liegt an.	LONG

Bemerkungen

Für digitale Kanäle, die als Ausgang konfiguriert sind, hat **Digin** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Siehe auch

Conf_DIO, Digin_Edge, Digin_Long, Digin_Word1, Digin_Word2, Digout

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic

Init:
  Conf_DIO(1100b)           'channels 15:0 as inputs

Event:
  Rem Eingang DIO00 = high?
  If (Digin(0) = 1) Then
    Rem Eingänge DIO02 und DIO05 auf DIO18 und DIO20 ausgeben
    Digout(18, Digin(2))
    Digout(20, Digin(5))
  EndIf
```

Digin_Edge

T11 TiCo

Digin_Edge gibt zurück, ob an den Digitaleingängen eine positive oder negative Flanke aufgetreten ist.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Edge(edge)
```

Parameter

edge	Art der zu prüfenden Flanke: 1: Auf positive Flanke prüfen. 0: Auf negative Flanke prüfen.	LONG
ret_val	Bitmuster, das angibt, an welchen Eingängen eine Flanke aufgetreten ist. Die Zuordnung der Bits zu den Eingängen ist unten dargestellt. Bit = 1: Flanke ist aufgetreten. Bit = 0: Keine Flanke aufgetreten.	LONG

Bitnr.	31	30	…	2	1	0
Eingang	31	30	…	2	1	0

Bemerkungen

Ein gesetztes Bit in ret_val bedeutet, dass die gesuchte Flanke seit dem vorigen Abfragen mindestens einmal am Digitaleingang aufgetreten ist. Für Ausgangskanäle sind die Bits immer Null.

Der Aufruf von **Digin_Edge** setzt alle Bits zurück auf 0.

Siehe auch

Digin_Fifo_Clear, Digin_Fifo_Enable, Digin_Fifo_Full, Digin_Fifo_Read, Digin_Fifo_Read_Timer

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic

Init:
  Conf_DIO(1100b)            'channels 15:0 as inputs

Event:
  Rem check rising and falling edges, mask out outputs
  Par_1 = Digin_Edge(1) And 0Fh
  Par_2 = Digin_Edge(0) And 0Fh

  Rem output edge changes to outputs
  If (Par_1 + Par_2 > 0)Then
   Digout_Bits(Shift_Left(Par_1,16),Shift_Left(Par_2,16))
  EndIf
```

Digin_Fifo_Clear

T11 TiCo

`Digin_Fifo_Clear` löscht den FIFO der Flankenüberwachung.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digin_Fifo_Clear()
```

Parameter

- / -

Bemerkungen

- / -

Siehe auch

Digin_Fifo_Enable, Digin_Fifo_Full, Digin_Fifo_Read, Digin_Fifo_Read_Timer, Digin_Edge

Gültig für

Gold II

Beispiel

siehe Digin_Fifo_Enable

Digin_Fifo_Enable

T11 TiCo

Digin_Fifo_Enable legt fest, an welchen Eingangskanälen die Flanken überwacht werden.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digin_Fifo_Enable(channels)
```

Parameter

channels	Bitmuster, das die zu überwachenden Eingangskanäle festlegt.	LONG

Bitnr.	31	30	…	2	1	0
Eingang	31	30	…	2	1	0

Bemerkungen

Es können nur Eingangskanäle überwacht werden. Die Kanäle werden mit **DigProg** als Eingänge oder Ausgänge programmiert.

Die Flankenüberwachung prüft alle 10 ns, ob an den festgelegten Eingangskanälen eine Flanke aufgetreten ist bzw. ob sich ein Pegel geändert hat. Sobald eine Flanke aufgetreten ist, wird ein Wertepaar in ein FIFO-Feld kopiert:

- Wert 1 enthält den Pegelzustand aller Kanäle als Bitmuster.
- Wert 2 enthält einen Zeitstempel, den aktuellen Stand eines 100 MHz-Zählers.

Das FIFO-Feld kann maximal 511 Wertepaare (Pegelzustand und Zeitstempel) enthalten. Wenn das FIFO-Feld voll ist, können keine weiteren Wertepaare gespeichert werden und gehen damit verloren.

Siehe auch

Digin_Fifo_Clear, Digin_Fifo_Full, Digin_Fifo_Read, Digin_Fifo_Read_Timer, Digin_Edge, Conf_DIO

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic

Dim Data_1[10000], Data_2[10000] As Long
Dim i, num, index As Long

Init:
  Conf_DIO(1100b)           'channels 15:0 as inputs
  Digin_Fifo_Enable(0)      'edge control off
  Digin_Fifo_Clear()        'clear FIFO
  Digin_Fifo_Enable(101010b)'control channels 1,3,5
  index = 1


Event:
  num = Digin_Fifo_Full()   'get number of value pairs
  If (num > 0) Then
    If (index + num > 10000) Then index = 1
    Rem read value pairs
    For i = 1 To num
      Digin_Fifo_Read(Data_1[index], Data_2[index])
      index = index+1
    Next i
  EndIf
```

Digin_Fifo_Full

T11 TiCo

`Digin_Fifo_Full` gibt die Anzahl der gespeicherten Wertepaare im FIFO der Flankenüberwachung zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Fifo_Full()
```

Parameter

ret_val	Anzahl (0…511) der belegten Wertepaare im FIFO.	LONG

Bemerkungen

Das FIFO-Feld kann maximal 511 Wertepaare (Pegelzustand und Zeitstempel) enthalten. Wenn das FIFO-Feld voll ist, können keine weiteren Wertepaare gespeichert werden und gehen damit verloren.

Siehe auch

Digin_Fifo_Clear, Digin_Fifo_Enable, Digin_Fifo_Read, Digin_Fifo_Read_Timer, Digin_Edge

Gültig für

Gold II

Beispiel

siehe Digin_Fifo_Enable

Digin_Fifo_Read

T11 TiCo

Digin_Fifo_Read liest ein Wertepaar aus dem FIFO der Flankenüberwachung.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digin_Fifo_Read(value_by_ref, timestamp_by_ref)
```

Parameter

value_by_ref	Variable, in die ein Bitmuster der Pegelzustände geschrieben wird. Die Zuordnung der Bits zu den Eingängen ist unten dargestellt.	LONG | CONST
timestamp_by_ref	Variable, in die der zugehörige Zeitstempel geschrieben wird.	LONG | CONST

Bitnr.	31	30	…	2	1	0
Eingang	31	30	…	2	1	0

Bemerkungen

Es dürfen nicht mehr Wertepaare gelesen werden als im FIFO gespeichert sind. Dazu muss vor dem Auslesen mit **Digin_Fifo_Full** geprüft werden, ob mindestens ein Wertepaar im FIFO gespeichert ist.

Der Zeitabstand zwischen 2 Pegelzuständen ist die Differenz der zugehörigen Zeitstempel, gemessen in Einheiten von 10 ns:

$$\Delta t = 10\ \text{ns} \cdot (\text{stamp}_1 - \text{stamp}_2)$$

Siehe auch

Digin_Fifo_Clear, Digin_Fifo_Enable, Digin_Fifo_Full, Digin_Fifo_Read_Timer, Digin_Edge

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic

Dim Data_1[10000], Data_2[10000] As Long
Dim index As Long

Init:
  Conf_DIO(1100b)           'channels 15:0 as inputs
  Digin_Fifo_Enable(0)      'edge control off
  Digin_Fifo_Clear()        'clear FIFO
  Digin_Fifo_Enable(10011b) 'control channels 0,1,4
  index = 1

Event:
  If (Digin_Fifo_Full() > 0) Then
    Rem read one value pair
    Digin_Fifo_Read(Data_1[index], Data_2[index])
    index = index + 1
    If (index>10000) Then index = 1
  EndIf
```

Digin_Fifo_Read_Timer

T11 TiCo

Digin_Fifo_Read_Timer gibt den aktuellen Stand des 100 MHz-Zählers zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Fifo_Read_Timer()
```

Parameter

ret_val	Aktueller Stand (-2^{31}-1 … 2^{31}) des 100 MHz-Zählers.	LONG

Bemerkungen

Der Zähler wird für das Erzeugen der Zeitstempel bei der Flankenüberwachung benutzt, siehe **Digin_Fifo_Enable**.

Der Zähler wird alle 10 ns um 1 erhöht, so dass der Zähler nach jeweils etwa 43 Sekunden (= 10 ns × 2^{32}) seinen ursprünglichen Wert erneut erreicht. Bei Zeitvergleichen muss dieser „Überlauf“ berücksichtigt werden, der Zählerstand muss daher im Programm regelmäßig vor dem Überlauf abgefragt werden.

Siehe auch

Digin_Fifo_Enable, Digin_Fifo_Read

Gültig für

Gold II

Beispiel

Wählen Sie das passende Include für ADbasic / TiCoBasic

```
#Include ADwinGoldII.inc   'for ADbasic
Rem #Include GoldIITiCo.inc 'for TiCoBasic
Rem provide number of counter overflows
#Define count_overflow Par_1
Dim t_start, diff_new, diff_old As Long

Init:
  count_overflow = 0        'overflow occurs every 43 seconds
  t_start = Digin_Fifo_Read_Timer()
  diff_old = 0

Event:
  Rem Event section must be run at least once every 20 seconds.
  Rem Else you will miss counter overflows.

  Rem get timer difference
  diff_new = Digin_Fifo_Read_Timer() - t_start
  If ((diff_new > 0) And (diff_old < 0)) Then
    Inc(count_overflow)     'increase number of counter overflows
  EndIf
  diff_old = diff_new
```

ähnliche Beispiele siehe

- *ADbasic*-Beispiel im Ordner
  `C:\ADwin\ADbasic\samples_ADwin`: `seconds_timer.bas`
- *TiCoBasic*-Beispiel `seconds_timer_TiCo.bas` im Ordner
  `C:\ADwin\TiCoBasic\samples_ADwin`

Digin_Long

T11 TiCo

Digin_Long gibt den Wert der digitalen Kanäle DIO31:DIO00 zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Long()
```

Parameter

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

Bitnr. in ret_val	31	30	…	1	0
Kanal	DIO31	DIO30	…	DIO01	DIO00

Bemerkungen

Für digitale Kanäle, die als Ausgang geschaltet sind, gibt **Digin_Long** undefinierte Werte zurück.

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.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge. Eine Standard-Konfiguration ist **Conf_DIO**(1100b), d.h. DIO00…DIO15 sind Eingänge und DIO16…DIO31 sind Ausgänge.

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

Conf_DIO, Digin, Digin_Word1, Digin_Word2, Digout_Long

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim Data_1[10000] As Long As FIFO
Init:
  REM Alle Kanäle als Eingänge konfigurieren
  CONF_DIO(0000b)
  Processdelay = 10000


Event:
  REM Ist der digitale Eingang DIO17 gesetzt?
  IF ((Shift_Right(DIGIN_Long(),17) And 1) = 1) Then
    Data_1 = ADC(1)          'Messwerterfassung
  EndIf
```

Digin_Word1

T11 TiCo

Digin_Word1 gibt die Werte der digitalen Kanäle DIO15:DIO00 zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Word1()
```

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

Bitnr. in ret_val	31:16	15	14	…	1	0
Kanal	–	DIO15	DIO14	…	DIO01	DIO00

Bemerkungen

Für digitale Kanäle, die als Ausgang geschaltet sind, gibt **Digin_Word1** undefinierte Werte zurück.

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.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge. Eine Standard-Konfiguration ist **Conf_DIO**(1100b), d.h. DIO00…DIO15 sind Eingänge und DIO16…DIO31 sind Ausgänge.

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

Conf_DIO, Digin, Digin_Long, Digin_Word2, Digout_Word1

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim Data_1[10000] As Long As FIFO
Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(1100b)
  Processdelay = 10000

Event:
  REM Abfrage, ob die Eingänge DIO01 und DIO02 gesetzt sind
  IF ((Digin_Word1() And 110b) = 110b) Then
    Data_1 = ADC(1)          'Messwerterfassung
  EndIf
```

Digin_Word2

T11 TiCo

`Digin_Word2` gibt die Werte der digitalen Kanäle DIO31:DIO16 zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Word2()
```

Parameter

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

Bitnr. in ret_val	31:16	15	14	…	1	0
Kanal	–	DIO31	DIO30	…	DIO17	DIO16

Bemerkungen

Für digitale Kanäle, die als Ausgang geschaltet sind, gibt `Digin_Word2` undefinierte Werte zurück.

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.

Der Befehl `Conf_DIO` konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge. Eine Standard-Konfiguration ist `Conf_DIO(1100b)`, d.h. DIO00…DIO15 sind Eingänge und DIO16…DIO31 sind Ausgänge.

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

Conf_DIO, Digin, Digin_Long, Digin_Word1, Digout_Word2

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim Data_1[10000] As Long As FIFO
Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(0011b)
  Processdelay = 10000

Event:
  REM Abfrage, ob die Eingänge DIO16 und DIO17 gesetzt sind
  IF ((Digin_Word2() And 11b) = 11b) Then
    Data_1 = ADC(1)          'Messwerterfassung
  EndIf
```

Digout

T11 TiCo

Digout setzt einen einzelnen Kanal auf einen definierten TTL-Pegel.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout(channel,value)
```

Parameter

channel	Nummer (0…31) des Digitalausgangs.	LONG
value	TTL-Pegel, der am Ausgang gesetzt wird: 1: Setzen auf TTL-Pegel high 0: Setzen auf TTL-Pegel low	LONG

Bemerkungen

Für digitale Kanäle, die als Eingang geschaltet sind, hat **Digout** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Siehe auch

Conf_DIO, Digin, Digout_Bits, Digout_Long, Digout_Word1, Digout_Word2

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim value As Long

Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(1100b)
  Processdelay = 10000

Event:
  value = ADC(1)              'Messwerterfassung
  IF (value > 1600) Then'Grenzwert überschritten?
    Digout(19,1)              'Ausgang DIO19 auf Pegel high setzen
    Digout(23,0)              'Ausgang DIO23 auf Pegel low setzen
  EndIf
```

Digout_Bits

T11 TiCo

Digout_Bits setzt ausgewählte digitale Kanäle auf definierte TTL-Pegel.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Bits(set,clear)
```

Parameter

set	Bitmuster, das die Ausgänge festlegt, die auf den TTL-Pegeln High gesetzt werden (siehe Tabelle). 1: TTL-Pegel High setzen. 0: Pegel nicht verändern.	LONG
clear	Bitmuster, das die Ausgänge festlegt, die auf den TTL-Pegeln Low gesetzt werden (siehe Tabelle). 1: TTL-Pegel Low setzen. 0: Pegel nicht verändern.	LONG

Bitnr.	31	30	…	1	0
Kanal	DIO31	DIO30	…	DIO01	DIO00

Bemerkungen

Für digitale Kanäle, die als Eingang geschaltet sind, hat **Digout_Bits** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Wenn Bits sowohl in set als auch in clear gesetzt sind, wird der entsprechende Kanal auf den TTL-Pegel Low gesetzt.

Siehe auch

Conf_DIO, Digout, Digout_Long, Get_Digout_Long, Digout_Reset, Digout_Set

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim value As Long

Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(0011b)
  Processdelay = 10000

Event:
  value = ADC(1)            'Messwerterfassung
  IF (value > 3000) Then'Grenzwert überschritten?
    REM Ausgänge DIO00 und DIO02 auf Pegel High setzen, Ausgänge
    REM DIO01,
    REM DIO03 und DIO04 auf Pegel Low
    Digout_Bits(00101b, 11010b)
  EndIf
```

Digout_Long

T11 TiCo

Digout_Long setzt alle digitalen Kanäle DIO31:DIO00 auf definierte TTL-Pegel.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Long(pattern)
```

Parameter

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

Bitnr. in pattern	31	30	…	1	0
Kanal	DIO31	DIO30	…	DIO01	DIO00

Bemerkungen

Für digitale Kanäle, die als Eingang geschaltet sind, hat **Digout_Long** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Wenn einzelne Kanäle gesetzt, die übrigen aber unverändert bleiben sollen, ist der Befehl **Digout_Bits** geeignet.

Siehe auch

Conf_DIO, Digin_Long, Digout, Digout_Bits, Digout_Word1, Digout_Word2, Get_Digout_Long

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim value As Long

Init:
  REM Alle Kanäle als Ausgänge konfigurieren
  CONF_DIO(1111b)
  Processdelay = 10000

Event:
  value = ADC(1)              'Messwerterfassung
  IF (value > 1500) Then'Grenzwert überschritten?
    REM Ausgänge DIO00, DIO02 und DIO06 setzen, alle anderen
    REM löschen
    Digout_Long(1000101b)
  EndIf
```

Digout_Reset

T11 TiCo

Digout_Reset setzt ausgewählte digitale Kanäle auf den TTL-Pegel Low.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Reset(clear)
```

Parameter

clear	Bitmuster, das die Ausgänge festlegt, die auf den TTL-Pegeln Low gesetzt werden (siehe Tabelle). 1: TTL-Pegel Low setzen. 0: Pegel nicht verändern.	LONG

Bitnr.	31	30	…	1	0
Kanal	DIO31	DIO30	…	DIO01	DIO00

Bemerkungen

Für digitale Kanäle, die als Eingang geschaltet sind, hat **Digout_Reset** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Siehe auch

Conf_DIO, Digout, Digout_Bits, Digout_Long, Get_Digout_Long, Digout_Set

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim value As Long

Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(0011b)
  Processdelay = 10000

Event:
  value = ADC(1)            'Messwerterfassung
  IF (value > 3000) Then'Grenzwert überschritten?
    REM Ausgänge DIO01, DIO03 und DIO04 auf Pegel Low setzen
    Digout_Reset(11010b)
  EndIf
```

Digout_Set

T11 TiCo

Digout_Set setzt ausgewählte digitale Kanäle auf den TTL-Pegel High.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Set(set)
```

Parameter

set	Bitmuster, das die Ausgänge festlegt, die auf den TTL-Pegeln High gesetzt werden (siehe Tabelle). 1: TTL-Pegel High setzen. 0: Pegel nicht verändern.	LONG

Bitnr.	31	30	…	1	0
Kanal	DIO31	DIO30	…	DIO01	DIO00

Bemerkungen

Für digitale Kanäle, die als Eingang geschaltet sind, hat **Digout_Set** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Siehe auch

Conf_DIO, Digout, Digout_Bits, Digout_Long, Get_Digout_Long, Digout_Reset

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim value As Long

Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(0011b)
  Processdelay = 10000

Event:
  value = ADC(1)              'Messwerterfassung
  IF (value > 3000) Then'Grenzwert überschritten?
    REM Ausgänge DIO00 und DIO02 auf Pegel High setzen
    Digout_Set(00101b)
  EndIf
```

Digout_Word1

T11 TiCo

Digout_Word1 setzt die digitalen Kanäle DIO15:DIO00 auf definierte TTL-Pegel.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Word1(pattern)
```

Parameter

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

Bitnr. in pattern	31:16	15	14	…	1	0
Kanal	–	DIO15	DIO14	…	DIO01	DIO00

Bemerkungen

Für digitale Kanäle, die als Eingang geschaltet sind, hat **Digout_Word1** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Wenn einzelne Kanäle gesetzt, die übrigen aber unverändert bleiben sollen, ist der Befehl **Digout_Bits** geeignet.

Siehe auch

Conf_DIO, Digin_Word1, Digout, Digout_Bits, Digout_Long, Digout_Word2, Get_Digout_Word1

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim value As Long

Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(0011b)
  Processdelay = 10000

Event:
  value = ADC(1)              'Messwerterfassung
  IF (value > 3000) Then'Grenzwert überschritten?
    REM Ausgänge DIO00 und DIO02 setzen, alle anderen Ausgänge
    REM löschen
    Digout_Word1(101b)
  EndIf
```

Digout_Word2

T11 TiCo

Digout_Word2 setzt die digitalen Kanäle DIO31:DIO16 auf definierte TTL-Pegel.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Word2(pattern)
```

Parameter

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

Bitnr. in pattern	31:16	15	14	…	1	0
Kanal	–	DIO31	DIO30	…	DIO17	DIO16

Bemerkungen

Für digitale Kanäle, die als Eingang geschaltet sind, hat **Digout_Word2** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Wenn einzelne Kanäle gesetzt, die übrigen aber unverändert bleiben sollen, ist der Befehl **Digout_Bits** geeignet.

Siehe auch

Conf_DIO, Digin_Word2, Digout, Digout_Bits, Digout_Long, Digout_Word1, Get_Digout_Word2

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic
Dim value As Long

Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(1100b)
  Processdelay = 10000

Event:
  value = ADC(1)              'Messwerterfassung
  IF (value > 2500) Then'Grenzwert überschritten?
    REM Ausgänge DIO17 und DIO20 setzen, alle anderen Ausgänge
    REM löschen
    Digout_Word2(10010b)
  EndIf
```

Get_Digout_Long

T11 TiCo

`Get_Digout_Long` gibt den Inhalt des Registers für die digitalen Ausgänge DIO31:DIO00 zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Get_Digout_Long()
```

Parameter

`ret_val` Inhalt des Ausgangsregisters, Zuordnung der Bits zu den Ausgängen siehe Tabelle. `LONG`
1: TTL-level high.
0: TTL-level low.

Bitnr. in `ret_val`	31	30	…	1	0
Kanal	DIO31	DIO30	…	DIO01	DIO00

Bemerkungen

Der Rückgabewert gibt nur den Zustand des Ausgangsregisters wieder, ein Rücklesen der tatsächlichen Ausgangszustände ist technisch nicht möglich.

Für Kanäle, die als Eingang konfiguriert sind, ist der Rückgabewert nicht definiert. Der Befehl `Conf_DIO` konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Siehe auch

`Conf_DIO`, `Digin_Long`, `Digout_Bits`, `Digout_Long`, `Get_Digout_Word1`, `Get_Digout_Word2`

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic

Init:
  REM Alle Kanäle als Ausgänge konfigurieren
  CONF_DIO(1111b)
  Processdelay = 10000

Event:
  Par_1 = Get_Digout_Long()'Bits 31:0 aus dem Register
                           'zurücklesen
```

Get_Digout_Word1

T11 TiCo

`Get_Digout_Word1` gibt den Inhalt des Registers für die digitalen Ausgänge DIO15:DIO00 zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Get_Digout_Word1()
```

Parameter

`ret_val` Inhalt des Ausgangsregisters, Zuordnung der Bits zu den Ausgängen siehe Tabelle. `LONG`
1: TTL-level high.
0: TTL-level low.

Bitnr. in `ret_val`	31:16	15	14	…	1	0
Kanal	–	DIO15	DIO14	…	DIO01	DIO00

Bemerkungen

Der Rückgabewert gibt nur den Zustand des Ausgangsregisters wieder, ein Rücklesen der tatsächlichen Ausgangszustände ist technisch nicht möglich.

Für Kanäle, die nicht als Ausgang konfiguriert sind, ist der Rückgabewert nicht definiert. Der Befehl `Conf_DIO` konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Siehe auch

`Conf_DIO`, `Digin_Word1`, `Digout_Bits`, `Digout_Word1`, `Get_Digout_Long`, `Get_Digout_Word1`, `Get_Digout_Word2`

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic

Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(0011b)
  Processdelay = 10000

Event:
  Par_1 = Get_Digout_Word1()'Bits 15:0 aus dem Register
                            'zurücklesen
```

Get_Digout_Word2

T11 TiCo

`Get_Digout_Word2` gibt den Inhalt des Registers für die digitalen Ausgänge DIO31:DIO16 zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Get_Digout_Word2()
```

Parameter

`ret_val` Inhalt des Ausgangsregisters, Zuordnung der Bits zu den Ausgängen siehe Tabelle. `LONG`
1: TTL-level high.
0: TTL-level low.

Bitnr. in `ret_val`	31:16	15	14	…	1	0
Kanal	–	DIO31	DIO30	…	DIO17	DIO16

Bemerkungen

Der Rückgabewert gibt nur den Zustand des Ausgangsregisters wieder, ein Rücklesen der tatsächlichen Ausgangszustände ist technisch nicht möglich.

Für Kanäle, die nicht als Ausgang konfiguriert sind, ist der Rückgabewert nicht definiert. Der Befehl `Conf_DIO` konfiguriert die digitalen Kanäle in Gruppen zu je 8 als Eingänge oder Ausgänge.

Siehe auch

Conf_DIO, Digin_Word2, Digout_Bits, Digout_Word2, Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Gültig für

Gold II

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic

Init:
  REM Ein- und Ausgänge konfigurieren
  CONF_DIO(1100b)
  Processdelay = 10000

Event:
  Par_1 = Get_Digout_Word2()'Bits 31:16 aus dem Register
                            'zurücklesen
```

16.4 Zähler

Dieser Abschnitt beschreibt Befehle zum Ansprechen der Zähler auf *ADwin-Gold II*:

Cnt_Clear

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Clear(pattern)
```

Parameter

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

Bitnr.	31:4	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

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

Achten Sie darauf, in `Cnt_Mode` im Bitmuster `pattern` Bit 1 = 0 für die entsprechenden Zähler einzustellen. Mit Bit 1 = 1 müssen sonst auch die Zählereingänge `A` und `B` auf TTL-Pegel high stehen, damit der Zähler gelöscht wird.

Siehe auch

Cnt_Enable, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_Read_Latch, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Cnt_Enable(0000b)            'alle Zähler stoppen
  Cnt_SE_Diff(0011b) 'Zähler 1+2 diff. (3+4 single ended)
  Rem Zähler 1+2: Modus Takt-Richtung, CLR freigeben
  Cnt_Mode(1,110000b)          'Zähler 1+2: externen
  Cnt_Mode(2,110000b)          ' Takteingang LATCH freigeben
  Cnt_Clear(11b)               'Zähler 1+2 auf 0 zurücksetzen
  Rem Zähler 1+2 starten, Zähler 3+4 und PWM1-4 stoppen
  Cnt_Enable(11b)

Event:
  Cnt_Latch(0011b)             'Zähler 1+2 latchen
  Par_1 = Cnt_Read_Latch(1)    'Latch A Zähler 1 und
  Par_2 = Cnt_Read_Latch(2)    ' Latch A Zähler 2 lesen
```

Cnt_Enable

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Enable(pattern)
```

Parameter

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

Bitnr.	31:12	11	10	9	8	7:4	3	2	1	0
Zähler-Nr.	–	PW4	PW3	PW2	PW1	–	VR4	VR3	VR2	VR1

Bemerkungen

Die Standard-Zähler und die PWM-Zähler arbeiten unabhängig voneinander.

Siehe auch

Cnt_Clear, Cnt_Get_Status, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Latch, Cnt_Mode, Cnt_PW_Latch, Cnt_Read, Cnt_Read_Latch, Cnt_Read_Int_Register, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Cnt_Enable(0000b)            'alle Zähler stoppen
  Cnt_SE_Diff(0011b) 'Zähler 1+2 diff. (3+4 single ended)
  Rem Zähler 1+2: Modus Takt-Richtung, CLR freigeben
  Cnt_Mode(1,110000b)          'Zähler 1+2: externen
  Cnt_Mode(2,110000b)          ' Takteingang LATCH freigeben
  Cnt_Clear(11b)               'Zähler 1+2 auf 0 zurücksetzen
  Rem Zähler 1+2 starten, Zähler 3+4 und PWM1-4 stoppen
  Cnt_Enable(11b)

Event:
  Cnt_Latch(0011b)             'Zähler 1+2 latchen
  Par_1 = Cnt_Read_Latch(1)    'Latch A Zähler 1 und
  Par_2 = Cnt_Read_Latch(2)    ' Latch A Zähler 2 lesen
```

Cnt_Get_Status

T11 TiCo

`Cnt_Get_Status` gibt den Inhalt des Statusregisters für einen Zähler zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Cnt_Get_Status(counter_no)
```

Parameter

`counter_no`	Zählernummer: 1…4.	LONG
`ret_val`	Inhalt des Statusregisters für den Zähler: Hinweise auf mögliche Fehlerquellen. Bedeutung der Bits 4:0 siehe Tabelle.	LONG

Bitnr.	31:5	4	3	2	1	0
Signal	–	C	L	N	B	A

- :don't care (Signalzustände undefiniert, mit 01Fh ausmaskieren)
A:Signal A (statisch)
B: Signal B (statisch)
N:CLR-/LATCH-Eingang (statisch)
L: Leitungsfehler (Kabel abgezogen oder Leitung unterbrochen)
C:Korrelationsfehler (Signal A und B sind identisch, d.h. nicht um ca. 90° phasenverschoben)

Bemerkungen

Ein Leitungsfehler (L) kann nur bei differentiellen Eingängen detektiert werden! Bei TTL-Eingängen sind diese Bits stets 0.

Das Statusregister wird beim Auslesen automatisch zurückgesetzt.

Siehe auch

Cnt_Clear, Cnt_Enable, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_Read_Latch, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim error As Long

Init:
  Cnt_Enable(0000b)             'alle Zähler stoppen
  Rem Zähler 1: Modus Takt-Richtung
  Cnt_Mode(1,0)
  Cnt_Clear(0001b)              'Zähler 1 auf 0 zurücksetzen
  Cnt_Enable(0001b)             'Zähler 1 starten
  error = 0                     'Fehlerindikator zurücksetzen

Event:
  PAR_1 = Cnt_Read(1)           'Zähler 1 lesen
  PAR_2 = Cnt_GetStatus(1) And 11111b 'Status
  REM Leitungs- bzw. Kabelfehler am Zähler 1?
  If (PAR_2 And 10000b = 10000b) Then
    REM Anzahl Leitungs- bzw. Kabelfehler
    Inc PAR_3
    error = 1                   'Fehlerindikator setzen
  EndIf
  REM Korrelationsfehler Zähler 1?
  If (PAR_2 AND 01000b = 01000b) Then
    Inc PAR_4                   'Anzahl Korrelationsfehler
    error = 1                   'Fehlerindikator setzen
  EndIf
  PAR_5 = Shift_Right(PAR_2 And 100b,2) 'Zustand Eingang CLR
  PAR_6 = PAR_2 And 1b          'Zustand Eingang A
  PAR_7 = Shift_Right(PAR_2 And 10b,1) 'Zustand Eingang B
```

Cnt_Get_PW

T11

`Cnt_Get_PW` gibt Frequenz und Tastverhältnis eines PWM-Zählers zurück.

Syntax

```
#Include ADwinGoldII.inc
Cnt_Get_PW(pwm_no,frequency,dutycycle)
```

Parameter

pwm_no	Nummer (1…4) des PWM-Zählers.	LONG
frequency	Frequenz in Hertz: 0,025 Hz …100 MHz.	FLOAT CONST
dutycycle	Tastverhältnis in Prozent (0.0…100.0).	FLOAT CONST

Bemerkungen

Die Rückgabewerte werden in den Parametern `frequency` und `dutycycle` übergeben.

Siehe auch

Cnt_Enable, Cnt_Get_PW_HL, Cnt_Mode, Cnt_PW_Latch, Cnt_Read_Int_Register, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
#Include ADwinGoldII.inc

Init:
  Cnt_Enable(0)              'Zähler stoppen
  Cnt_SE_Diff(0)             'Zählereingänge single ended (TTL)
  Rem Betriebsmodus PWM-Zähler einstellen:
  Rem Bits 0..5: ohne Bedeutung
  Rem Bit 6=0: steigende Flanke als PWM-Signal
  Rem Bit 7,8: Auswahl A, B oder CLR als PWM-Eingang
  Cnt_Mode(1,010000000b)     'Zähler 1: PWM-Messung, Eingang B
  Cnt_Mode(2,110000000b)     'Zähler 2: PWM-Messung, Eingang CLR
  Cnt_Enable(1100000000b)    'PWM-Zähler 1+2 starten

Event:
  Cnt_PW_Latch(11b)          'Zähler 1+2 gleichzeitig latchen
  Cnt_Get_PW_HL(1,Par_1,Par_2)'High-/Low-Zeit lesen
  Cnt_Get_PW(1,FPar_1,FPar_2)'Frequenz und Taktverhältnis lesen
```

Cnt_Get_PW_HL

T11 TiCo

Cnt_Get_PW_HL gibt die Eintastzeit und die Austastzeit eines PWM-Zählers zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Get_PW_HL(counter_no,hightime,lowtime)
```

Parameter

counter_no	Zählernummer: 1…4.	LONG
hightime	Eintastzeit des PWM-Signals in Einheiten von 10ns.	LONG CONST
lowtime	Austastzeit des PWM-Signals in Einheiten von 10ns.	LONG CONST

Bemerkungen

Die Rückgabewerte werden in den Parametern hightime und lowtime übergeben.

Siehe auch

Cnt_Enable, Cnt_Get_PW, Cnt_Mode, Cnt_PW_Latch, Cnt_Read_Int_Register, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Cnt_Enable(0)              'Zähler stoppen
  Cnt_SE_Diff(0)             'Zählereingänge single ended (TTL)
  Rem Betriebsmodus PWM-Zähler einstellen:
  Rem Bits 0..5: ohne Bedeutung
  Rem Bit 6=0: steigende Flanke als PWM-Signal
  Rem Bit 7,8: Auswahl A, B oder CLR als PWM-Eingang
  Cnt_Mode(1,010000000b)     'Zähler 1: PWM-Messung, Eingang B
  Cnt_Mode(2,110000000b)     'Zähler 2: PWM-Messung, Eingang CLR
  Cnt_Enable(1100000000b)    'Zähler 1+2 starten

Event:
  Cnt_PW_Latch(11b)          'Zähler 1+2 gleichzeitig latchen
  Cnt_Get_PW_HL(1,Par_1,Par_2)'High-/Low-Zeit lesen
```

Cnt_Latch

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Latch(pattern)
```

Parameter

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

Bitnr.	31:4	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

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

Das Latch A wird mit `Cnt_Read_Latch` in eine Variable ausgelesen.

Siehe auch

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Mode, Cnt_Read, Cnt_Read_Latch, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Cnt_Enable(0000b)            'alle Zähler stoppen
  Cnt_SE_Diff(0011b) 'Zähler 1+2 diff. (3+4 single ended)
  Rem Zähler 1+2: Modus Takt-Richtung, CLR freigeben
  Cnt_Mode(1,00000b)           'externen Takteingang sperren
  Cnt_Mode(2,00000b)
  Cnt_Clear(11b)               'Zähler 1+2 auf 0 zurücksetzen
  Rem Zähler 1+2 starten, Zähler 3+4 und PWM1-4 stoppen
  Cnt_Enable(11b)

Event:
  Cnt_Latch(0011b)             'Zähler 1 und 2 latchen
  Par_1 = Cnt_Read_Latch(1)    'Latch A Zähler 1 und
  Par_2 = Cnt_Read_Latch(2)    ' Latch A Zähler 2 lesen
```

Cnt_Mode

T11 TiCo

Cnt_Mode definiert die Betriebsart eines Zählers.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Mode(counter_no,pattern)
```

Parameter

counter_no	Zählernummer: 1…4.	LONG
pattern	Bitmuster zur Einstellung des Betriebsmodus des Zählers.	LONG

Bitnr.	Bedeutung
Bit 0	Zählermodus: Bit = 0: Takt-Richtungs-Modus. Bit = 1: A-B-Modus.
Bit 1	Löschmodus: Bit = 0: TTL-Pegel high am Eingang CLR setzt den Zählerstand auf Null. Bit = 1: Zähler löschen, wenn an allen Eingängen A, B, CLR der TTL-Pegel high anliegt. Nur im A-B-Modus.
Bit 2	Eingang A / CLK invertieren im Takt-Richtungs-Modus: Bit = 0: Eingang ist nicht invertiert. Bit = 1: Eingang ist invertiert.
Bit 3	Eingang B / DIR invertieren im Takt-Richtungs-Modus: Bit = 0: Eingang ist nicht invertiert. Bit = 1: Eingang ist invertiert.
Bit 4	Eingang CLR / LATCH einstellen. Bit = 0: Eingang CLR. Bit = 1: Eingang LATCH.
Bit 5	Eingang CLR / LATCH freigeben. Bit = 0: Eingang ist gesperrt. Bit = 1: Eingang ist freigegeben.
Bit 6	Auswahl der Referenzflanke für PWM-Auswertung. Bit = 0: steigende Flanke. Bit = 1: fallende Flanke.
Bit 8:7	Auswahl eines Eingangs für PWM-Auswertung. 00b: Eingang A / CLK 01b: Eingang B / DIR 10b: Eingang CLR / LATCH
Bits 31:9	reserviert.

Bemerkungen

Verwenden Sie **Cnt_Mode** nur bei gesperrtem Zähler, siehe **Cnt_Enable**.

Im Standard-Löschmodus (Bit 1=0) wird der Zählerstand so lange auf Null gesetzt, wie der TTL-Pegel high anliegt. Zum Löschen muss der Eingang CLR mit Bit 5=1 freigegeben werden.

Wenn Sie einen Zähler mit **Cnt_Clear** löschen wollen, sollten Sie im Bitmuster pattern Bit 1 = 0 einstellen. Mit Bit 1 = 1 müssen sonst auch die Eingänge A und B auf TTL-Pegel high stehen, damit der Zähler gelöscht wird.

Siehe auch

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Latch, Cnt_PW_Latch, Cnt_Read, Cnt_Read_Latch, Cnt_Read_Int_Register, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Cnt_Enable(0000b)            'alle Zähler stoppen
  Cnt_SE_Diff(0011b) 'Zähler 1+2 diff. (3+4 single ended)
  Rem Zähler 1+2: Modus Takt-Richtung, CLR freigeben
  Cnt_Mode(1,110000b)          'Zähler 1+2: externen
  Cnt_Mode(2,110000b)          ' Takteingang LATCH freigeben
  Cnt_Clear(11b)               'Zähler 1+2 auf 0 zurücksetzen
  Rem Zähler 1+2 starten, Zähler 3+4 und PWM1-4 stoppen
  Cnt_Enable(11b)

Event:
  Cnt_Latch(0011b)             'Zähler 1+2 latchen
  Par_1 = Cnt_Read_Latch(1)    'Latch A Zähler 1 und
  Par_2 = Cnt_Read_Latch(2)    ' Latch A Zähler 2 lesen
```

Cnt_PW_Latch

T11 TiCo

Cnt_PW_Latch kopiert den Inhalt eines oder mehrerer PWM-Zähler in einen Zwischenspeicher.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_PW_Latch(pattern)
```

Parameter

pattern	Bitmuster Bit = 0: Kein Einfluss. Bit = 1: PWM-Zählerinhalt latchen.	LONG

Bitnr.	31:4	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

Der Zwischenspeicher wird mit **Cnt_Get_PW** oder **Cnt_Get_PW_HL** ausgelesen.

Siehe auch

Cnt_Enable, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Mode, Cnt_Read_Int_Register, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

Init:
  Cnt_Enable(0000b)          'alle Zähler stoppen
  Cnt_SE_Diff(0)             'Alle Zählereingänge single ended
  Rem Betriebsmodus PWM-Zähler einstellen:
  Rem Bits 0..5: ohne Bedeutung
  Rem Bit 6=0: steigende Flanke als PWM-Signal
  Rem Bit 7,8: Auswahl A, B oder CLR als PWM-Eingang
  Cnt_Mode(1,010000000b)     'Zähler 1: PWM-Messung, Eingang B
  Cnt_Mode(2,110000000b)     'Zähler 2: PWM-Messung, Eingang CLR
  Cnt_Enable(1100000000b)    'Zähler 1+2 starten

Event:
  Cnt_PW_Latch(11b)          'Zähler 1+2 gleichzeitig latchen
  Cnt_Get_PW_HL(1,Par_1,Par_2)'High-/Low-Zeit lesen
```

Cnt_Read

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Cnt_Read(counter_no)
```

Parameter

counter_no	Zählernummer: 1…4.	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_Enable, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read_Latch, Cnt_Read_Int_Register, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Cnt_Enable(0000b)            'Zähler stoppen
  Cnt_SE_Diff(0011b)'Zähler 1+2 diff. (3+4 single ended)
  Rem Zähler 1+2: Modus Takt-Richtung, CLR freigeben
  Cnt_Mode(1,10000b)
  Cnt_Mode(2,10000b)
  Cnt_Clear(11b)               'Zähler 1+2 auf 0 zurücksetzen
  Rem Zähler 1+2 starten, Zähler 3+4 und PWM1-4 stoppen
  Cnt_Enable(11b)

Event:
  Par_1 = Cnt_Read(1)          'Zähler 1 lesen
  Par_2 = Cnt_Read(2)          'Zähler 2 lesen
```

Cnt_Read_Int_Register

T11 TiCo

Cnt_Read_Int_Register gibt den Inhalt eines Zählerregisters zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Cnt_Read_Int_Register(counter_no,reg_no)
```

Parameter

counter_no	Zählernummer: 1…4.	LONG
reg_no	Kennzahl (0…15) für ein Zählerregister, Zuordnungstabelle siehe unten.	LONG
ret_val	Inhalt des Zählerregisters.	LONG

reg_no	Register
0	Latch 1 für positive Flanken.
1	Latch 2 für positive Flanken.
2	Latch 3 für positive Flanken.
3	Latch 1 für negative Flanken.
4	Latch 2 für negative Flanken.
5	Latch 3 für negative Flanken.
6	Software-Latch für VR-Zähler.
7	Software-Latch für PWM-Zähler.
8	Schattenregister für Latch 1, positive Flanken.
9	Schattenregister für Latch 2, positive Flanken.
10	Schattenregister für Latch 3, positive Flanken.
11	Schattenregister für Latch 1, negative Flanken.
12	Schattenregister für Latch 2, negative Flanken.
13	Schattenregister für Latch 3, negative Flanken.
14	Schattenregister für Software-Latch, VR-Zähler.
15	Zählerstatus.

Bemerkungen

Zu jedem PWM-Zähler gehören die oben angegebenen Register. Wenn Sie die PWM-Zähler mit den Standard-Befehlen **Cnt_Get_PW** und **Cnt_Get_PW_HL** auswerten, benötigen Sie keine Kenntnisse über die PWM-Register. Nur für spezielle Lösungen ist es sinnvoll, wenn Sie die PWM-Register selbst auswerten.

Registerinhalte werden mit **Cnt_PW_Latch** oder **Cnt_Sync_Latch** gesetzt.

Zur Auswertung der PWM-Register beachten Sie die Hinweise in Kapitel 7.4 auf Seite 29.

Siehe auch

Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_PW_Latch, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

siehe Cnt_Sync_Latch

Cnt_Read_Latch

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Cnt_Read_Latch(counter_no)
```

Parameter

counter_no	Zählernummer: 1…4.	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`.

Siehe auch

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_SE_Diff, Cnt_Sync_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Init:
  Cnt_Enable(0000b)            'alle Zähler stoppen
  Cnt_SE_Diff(0011b) 'Zähler 1+2 diff. (3+4 single ended)
  Rem Zähler 1+2: Modus Takt-Richtung, CLR freigeben
  Cnt_Mode(1,110000b)          'Zähler 1+2: externen
  Cnt_Mode(2,110000b)          ' Takteingang LATCH freigeben
  Cnt_Clear(11b)               'Zähler 1+2 auf 0 zurücksetzen
  Rem Zähler 1+2 starten, Zähler 3+4 und PWM1-4 stoppen
  Cnt_Enable(11b)

Event:
  Cnt_Latch(0011b)             'Zähler 1+2 latchen
  Par_1 = Cnt_Read_Latch(1)    'Latch A Zähler 1 und
  Par_2 = Cnt_Read_Latch(2)    ' Latch A Zähler 2 lesen
```

Cnt_SE_Diff

T11 TiCo

`Cnt_SE_Diff` stellt die Eingänge aller Zähler auf den Betriebsmodus single-ended oder differentiell ein.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_SE_Diff(pattern)
```

Parameter

pattern	Bitmuster zur Auswahl der Zählerpaare (siehe Tabelle) und des Betriebsmodus der Eingänge: Bit = 0: Betriebsmodus single-ended Bit = 1: Betriebsmodus differentiell	LONG

Bitnr. in pattern	31:4	3	2	1	0
Nr. des Zählereingangs	–	4	3	2	1

Bemerkungen

Nach dem Start ist der Betriebsmodus single-ended eingestellt.

Siehe auch

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_Read_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim error As Long               'Variablen

Init:
  Cnt_Enable(0)                 'Alle Zähler stoppen
  Cnt_SE_Diff(0001b)            'Zähler 1 diff.
                                'Zähler 2-4 single ended
  Cnt_Mode(1,0)                 'Zähler 1: Modus Takt-Richtung
  Cnt_Clear(0001b)              'Zähler 1 auf 0 setzen
  Cnt_Enable(0001b)             'Zähler 1 starten
  error = 0                     'Fehlerindikator zurücksetzen

Event:
  Par_1 = Cnt_Read(1)           'Zähler 1 auslesen
  Rem Statusregister auslesen und maskieren
  Par_2 = Cnt_Get_Status(1) And 01Fh
  If (Par_2 And 01000b = 01000b) Then 'Leitungsfehler Zähler 1?
    Inc Par_3                   'Anzahl Leitungsfehler
    error = 1                   'Fehlerindikator setzen
  EndIf
  If (Par_2 And 10000b = 10000b) Then 'Korrelationsfehler
    Inc Par_4                   'Anzahl Korrelationsfehler
    error = 1                   'Fehlerindikator setzen
  EndIf
  Par_5 = Shift_Right(Par_2 And 100b,2) 'Zustand CLR-Eingg
  Par_6 = Par_2 And 1b          'Zustand Eingang A
  Par_7 = Shift_Right(Par_2 And 10b,1) 'Zustand Eingang B
```

Cnt_Sync_Latch

T11 TiCo

Cnt_Sync_Latch kopiert die Inhalte der gewählten Zähler und PWM-Zähler in Zwischenspeicher.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Sync_Latch(pattern)
```

Parameter

pattern	Bitmuster Bit = 0: Kein Einfluss. Bit = 1: Zählerinhalt in Zwischenspeicher kopieren.	LONG

Bitnr.	31:4	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

Jedem Bit sind sowohl ein VR-Zähler als auch ein PWM-Zähler zugeordnet. Beide Zählerinhalte werden gleichzeitig kopiert. Der Befehl hat damit die gleiche Funktion wie **Cnt_Latch** und **Cnt_PW_Latch** zusammen.

Die Zwischenspeicher werden beispielsweise mit **Cnt_Read_Latch** oder **Cnt_Get_PW** ausgelesen.

Siehe auch

Cnt_Get_PW, Cnt_Latch, Cnt_Mode, Cnt_PW_Latch, Cnt_Read_Int_Register, Cnt_Read_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
#Define frequency Par_1
Dim time, edges, oldpw, oldvr As Long
Dim vr, pw As Long

Init:
  Processdelay = 3000000        '100Hz with T11 processor
  Cnt_Enable(0)                 'counters off
  Cnt_Mode(1,00000000b)         'mode: clock/dir
  Cnt_Clear(0001b)              'clear counter 1
  Cnt_Enable(0101h)             'enable counters V/R 1 and PWM 1
  Cnt_Sync_Latch(0001b)         'latch counter 1 (V/R and PWM)
  oldvr = Cnt_Read_Int_Register(1,6)  'V/R counter 1
  oldpw = Cnt_Read_Int_Register(1,8)  'PWM counter 1
  frequency = 0

Event:
  REM latch values of counter 1 (V/R and PWM)
  Cnt_Sync_Latch(0001b)
  vr = Cnt_Read_Int_Register(1,6)  'value of clock/dir counter
  edges = Abs(vr - oldvr)       'number of edges between events
  If (edges <> 0) Then
    pw = Cnt_Read_Int_Register(1,8)  'positive edges latch 1
    time = pw - oldpw           'calculate timebase
    Rem frequency: 100000000=timer frequency
    frequency = edges*100000000/time
    oldcnt = cnt                'store VR counter value
    oldpw = pw                  'store PW counter value
  EndIf
```

16.5 SSI-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen der SSI-Schnittstellen auf *ADwin-Gold II*:

SSI_Mode

T11 TiCo

`SSI_Mode` stellt den Modus aller SSI-Decoder ein, entweder „single shot“ (einzeln lesen) und „continuous“ (kontinuierlich lesen).

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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.	31:2	3	2	1	0
SSI-Decoder	–	4	3	2	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

Gold II-CNT

Beispiel

```
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
REM Decoder 1 läuft mit 2.5 MHz, Decoder 2 mit 1,0 MHz
  INIT:
    SSI_Set_Clock(1,10)          'Taktrate einstellen, Decoder 1
    SSI_Set_Clock(2,25)          'Taktrate einstellen, Decoder 2
    SSI_Mode(11b) 'Continuous-Mode setzen, Decoder 1+2
    SSI_Set_Bits(1,10)           '10 Encoder-Bits,  Encoder 1
    SSI_Set_Bits(2,25)           '25 Encoder-Bits, Encoder 2

  EVENT:
    PAR_1 = SSI_Read(1)          'Pos.wert auslesen, Encoder 1
    PAR_2 = SSI_Read(2)          'Pos.wert auslesen, Encoder 2
```

SSI_Read

T11 TiCo

SSI_Read gibt den zuletzt gespeicherten Zählerstand eines bestimmten SSI-Zählers zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = SSI_Read(dcdr_no)
```

Parameter

dcdr_no	Nummer (1…4) 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

Gold II-CNT

Beispiel

```
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
REM Decoder 1 läuft mit 500 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

T11 TiCo

`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 ADwinGoldII.inc / GoldIITiCo.inc
SSI_Set_Bits(dcdr_no,bit_count)
```

Parameter

dcdr_no	Nummer (1…4) des SSI-Decoders, dessen Status gefragt 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

Gold II-CNT

Beispiel

```
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
REM Decoder 1 läuft mit 2.5 MHz, Decoder 2 mit 1,0 MHz
  INIT:
    SSI_Set_Clock(1,10)          'Taktrate einstellen, Decoder 1
    SSI_Set_Clock(2,25)          'Taktrate einstellen, Decoder 2
    SSI_Mode(11b) 'Continuous-Mode setzen, Decoder 1+2
    SSI_Set_Bits(1,10)           '10 Encoder-Bits,  Encoder 1
    SSI_Set_Bits(2,25)           '25 Encoder-Bits, Encoder 2

  EVENT:
    PAR_1 = SSI_Read(1)          'Pos.wert auslesen, Encoder 1
    PAR_2 = SSI_Read(2)          'Pos.wert auslesen, Encoder 2
```

SSI_Set_Clock

T11 TiCo

`SSI_Set_Clock` stellt die Taktrate (ca. 100 kHz bis 2,5 MHz) ein, mit der der Encoder getaktet wird.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
SSI_Set_Clock(dcdr_no,prescale)
```

Parameter

dcdr_no	Nummer (1…4) des SSI-Decoders, dessen Status gefragt ist.	LONG
prescale	Teilerfaktor (10…255) zur Einstellung der Taktrate nach der Formel: Taktrate = 25 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

Gold II-CNT

Beispiel

```
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
REM Decoder 1 läuft mit 2.5 MHz, Decoder 2 mit 1,0 MHz
  INIT:
    SSI_Set_Clock(1,10)          'Taktrate einstellen, Decoder 1
    SSI_Set_Clock(2,25)          'Taktrate einstellen, Decoder 2
    SSI_Mode(11b) 'Continuous-Mode setzen, Decoder 1+2
    SSI_Set_Bits(1,10)           '10 Encoder-Bits,  Encoder 1
    SSI_Set_Bits(2,25)           '25 Encoder-Bits, Encoder 2

  EVENT:
    PAR_1 = SSI_Read(1)          'Pos.wert auslesen, Encoder 1
    PAR_2 = SSI_Read(2)          'Pos.wert auslesen, Encoder 2
```

SSI_Start

T11 TiCo

`SSI_Start` startet das Auslesen der gewählten SSI-Encoder (nur im Modus „single shot“).

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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

Bitnr.	31:2	3	2	1	0
SSI-Decoder	–	4	3	2	1

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

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
REM Beide Decoder laufen mit 100 kHz
  INIT:
    SSI_Set_Clock(1,250)          'Taktrate Decoder 1
    SSI_Set_Clock(2,250)          'Taktrate Decoder 2
    SSI_Mode(0)                   'Single shot-Mode
                                  'für alle Zähler
    SSI_Set_Bits(1,23)            '23 Encoder-Bits auf Encoder 1
    SSI_Set_Bits(2,23)            '23 Encoder-Bits auf Encoder 2

  EVENT:
    SSI_Start(11b)                'Positionswert Encoder 1&2 lesen
    DO                            'Für Encoder 1:
    UNTIL (SSI_Status(1) = 0)  'Wenn Positionswert komplett
                                  'gelesen ist …
    PAR_1 = SSI_Read(1)           'Positionswert auslesen
    DO                            'Für Encoder 2:
    UNTIL (SSI_Status(2) = 0)  'Wenn Positionswert komplett
                                  'gelesen ist …
    PAR_1 = SSI_Read(2)           'Positionswert auslesen
```

SSI_Status

T11 TiCo

`SSI_Status` liefert für einen bestimmten Decoder den aktuellen Lese-Status zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = SSI_Status(dcdr_no)
```

Parameter

dcdr_no	Nummer (1…4) des SSI-Decoders, dessen Status gefragt 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

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
REM Beide Decoder laufen mit 100 kHz
  INIT:
    SSI_Set_Clock(1,250)          'Taktrate Decoder 1
    SSI_Set_Clock(2,250)          'Taktrate Decoder 2
    SSI_Mode(0)                   'Single shot-Mode
                                  'für alle Zähler
    SSI_Set_Bits(1,23)            '23 Encoder-Bits auf Encoder 1
    SSI_Set_Bits(2,23)            '23 Encoder-Bits auf Encoder 2

  EVENT:
    SSI_Start(11b)                'Positionswert Encoder 1&2 lesen
    DO                            'Für Encoder 1:
    UNTIL (SSI_Status(1) = 0)  'Wenn Positionswert komplett
                                  'gelesen ist …
    PAR_1 = SSI_Read(1)           'Positionswert auslesen
    DO                            'Für Encoder 2:
    UNTIL (SSI_Status(2) = 0)  'Wenn Positionswert komplett
                                  'gelesen ist …
    PAR_1 = SSI_Read(2)           'Positionswert auslesen
```

16.6 PWM-Ausgänge

Dieser Abschnitt beschreibt Befehle zum Ansprechen der PWM-Ausgänge auf *ADwin-Gold II*:

PWM_Enable

T11 TiCo

PWM_Enable gibt einen oder mehrere PWM-Ausgänge zur Ausgabe frei.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Enable(pattern)
```

Parameter

pattern	Bitmuster zur Auswahl der PWM-Ausgänge: Bit = 0: PWM-Ausgabe sperren. Bit = 1: PWM-Ausgabe freigeben.	LONG

Bitnr.	31:6	5	4	3	2	1	0
PWM-Ausgang	–	6	5	4	3	2	1

Bemerkungen

Wann die PWM-Ausgänge gesperrt werden – sofort oder nach dem nächsten Periodenende – hängt von der Einstellung ab, die mit **PWM_Init** gemacht wurde (Parameter mode).

Siehe auch

PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Gültig für

Gold II-CNT

Beispiel

siehe PWM_Init (Seite 143)

PWM_Get_Status

T11 TiCo

`PWM_Get_Status` liest den aktuellen Betriebsstatus für alle PWM-Ausgänge.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = PWM_Get_Status()
```

Parameter

ret_val	Statusbits für alle PWM-Ausgänge. Bit = 0: PWM-Ausgabe ist abgeschlossen. Bit = 1: PWM-Ausgabe läuft.	LONG

Bitnr.	31:6	5	4	3	2	1	0
PWM-Ausgang	–	6	5	4	3	2	1

Bemerkungen

- / -

Siehe auch

PWM_Enable, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Gültig für

Gold II-CNT

Beispiel

- / -

PWM_Init

T11 TiCo

PWM_Init setzt die Voreinstellungen für den angegebenen PWM-Ausgang.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Init(pwm_output, startdelay, startvalue, mode,
    count)
```

Parameter

pwm_output	Nummer des PWM-Ausgabekanals (1…6).	LONG
startdelay	Startverzögerung in Einheiten von 20 ns.	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 2:0 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

Die Voreinstellungen werden aktiv, sobald PWM-Ausgänge mit **PWM_Enable** zur Ausgabe freigegeben werden.

Die Änderung der Voreinstellungen bei laufender Ausgabe ist nicht möglich. Stattdessen stoppen Sie die PWM-Ausgänge mit **PWM_Reset** oder sperren sie mit **PWM_Enable**, um die Voreinstellungen zu ändern. Anschließend geben Sie die PWM-Ausgänge wieder zur Ausgabe frei.

auch

PWM_Enable, PWM_Get_Status, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Gültig für

Gold II-CNT

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
#Define freq1 FPar_1
#Define freq2 FPar_2
#Define pw1 FPar_3
#Define pw2 FPar_4
Dim channel As Long

Init:
  freq1 = 1000                  '1000 Hz
  freq2 = 2000                  '2000 Hz
  pw1 = 50                      '50 %
  pw2 = 70                      '70 %
  PWM_Reset(011b)               'stop channels 1 und 2

  For channel = 1 To 2
    PWM_Init(channel,0,0,0,0)
  Next

  PWM_Write_Latch(1,pw1,freq1)
  PWM_Write_Latch(2,pw2,freq2)
  PWM_Latch(11b)
  PWM_Enable(011b)              'start output

Event:
  PWM_Write_Latch(1,pw1,freq1)
  PWM_Write_Latch(2,pw2,freq2)

  PWM_Latch(11b)
```

PWM_Latch

T11 TiCo

PWM_Latch gibt Frequenz und Tastverhältnis eines oder mehrerer PWM-Ausgänge für die Ausgabe frei.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Latch(pattern)
```

Parameter

pattern	Bitmuster zur Auswahl der PWM-Ausgänge: Bit = 0: Kein Einfluss. Bit = 1: Latchen = für Ausgabe freigeben.	LONG

Bitnr.	31:6	5	4	3	2	1	0
PWM-Ausgang	–	6	5	4	3	2	1

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_Enable, PWM_Get_Status, PWM_Init, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Gültig für

Gold II-CNT

Beispiel

siehe PWM_Init (Seite 143)

PWM_Reset

T11 TiCo

`PWM_Reset` stoppt die Ausgabe auf einem oder mehreren Ausgängen sofort.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Reset(pattern)
```

Parameter

pattern	Bitmuster zur Auswahl der PWM-Ausgänge: Bit = 0: Kein Einfluss Bit = 1: Ausgabe sofort stoppen	LONG

Bitnr.	31:6	5	4	3	2	1	0
PWM-Ausgang	–	6	5	4	3	2	1

Bemerkungen

Die Ausgabe wird auch dann sofort gestoppt, wenn mit `PWM_Init` ein anderer Modus eingestellt ist.

Siehe auch

PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Standby_Value, PWM_Write_Latch

Gültig für

Gold II-CNT

Beispiel

siehe `PWM_Init` (Seite 143)

PWM_Standby_Value

T11 TiCo

PWM_Standby_Value setzt den Vorgabewert (TTL-Pegel) für einen PWM-Ausgang.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Standby_Value(pattern)
```

Parameter

pattern	Vorgabewert für PWM-Ausgänge: Bit = 0: TTL-Pegel low Bit = 1: TTL-Pegel high	LONG

Bitnr.	31:6	5	4	3	2	1	0
PWM-Ausgang	–	6	5	4	3	2	1

Bemerkungen

Mit dem Befehl **PWM_Standby_Value** können PWM-Ausgänge auch als einfache TTL-Ausgänge benutzt werden.

Wenn ein 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 sind die Ausgänge zunächst auf TTL-Pegel low gesetzt.

Siehe auch

PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Write_Latch

Gültig für

Gold II-CNT

Beispiel

- / -

PWM_Write_Latch

T11 TiCo

PWM_Write_Latch schreibt Frequenz und Tastverhältnis in das Latch-Register.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Write_Latch(pwm_output, dutycycle, frequency)
```

Parameter

pwm_output	Nummer des PWM-Ausgabekanals (1…6).	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: 0,025Hz …25MHz.	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 ca. 500kHz.

Siehe auch

PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value

Gültig für

Gold II-CNT

Beispiel

siehe PWM_Init (Seite 143)

16.7 CAN-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen der CAN-Schnittstellen auf *ADwin-Gold II*:

CAN_Msg

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 Datenbytes und ihre Anzahl in das Feld **CAN_Msg**[] ein, *bevor* Sie diese mit **Transmit** übertragen.

Siehe auch

En_CAN_Interrupt, En_Receive, En_Transmit, Get_CAN_Reg, Init_CAN, Read_Msg, Set_CAN_Baudrate, Set_CAN_Reg, Transmit

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
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(1)                'CAN-Controller 1 initialisieren

  Rem Message-Objekt 6 der Schnittstelle 1 initialisieren
  Rem zum Senden von CAN-Nachrichten mit dem Identifier 40
  En_Transmit(1,6,40,0)

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

  Rem Bitmuster (32 Bit) in 4 Bytes aufteilen
  For i = 0 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(1,6)              'Message-Objekt 6 senden
  REM Empfangen einer Fließkomma-Zahl siehe Bsp. bei Read_Msg
```

En_CAN_Interrupt

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
En_CAN_Interrupt(can_no, msg_no)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG
msg_no	Nummer (1…15) des Message-Objektes	LONG

Bemerkungen

Nur *ADbasic*: Die Befehle **Event_Enable** und **En_CAN_Interrupt** dürfen nicht gemeinsam verwendet werden. Beide Befehle legen die verwendete Event-Quelle fest, so dass nur die zuletzt festgelegte Quelle aktiv ist.

CAN als Event-Quelle kann mit **Event_Enable** wieder deaktiviert werden.

Siehe auch

CAN_Msg, En_Receive, En_Transmit, Event_Enable

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
  INIT:
    Init_CAN(1)                  'CAN-Controller 1 initialisieren
    Rem Initialisiere das Message-Objekt 1 der CAN-
    Rem Schnittstelle 1 zum Empfangen von CAN-Nachrichten
    Rem mit dem Identifier 200
    En_Receive(1,1,200,0)
    Rem Gibt das Auslösen von Interrupts (ext. EVENT) beim
    Rem Empfang des Message-Objektes 1 frei
    En_CAN_Interrupt(1,1)
```

En_Receive

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
En_Receive(can_no, msg_no, id, id_extend)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG
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_CAN_Interrupt, En_Transmit, Get_CAN_Reg

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

  INIT:
    Init_CAN(1)                 'CAN-Controller 1 initialisieren
    Rem Initialisiere Message-Objekt 1 der Schnittstelle 1
    Rem zum Empfangen von Nachrichten mit dem Identifier 200
    En_Receive(1,1,200,0)
```

En_Transmit

T11 TiCo

En_Transmit gibt ein bestimmtes Message-Objekt einer CAN-Schnittstelle für das Senden von Nachrichten frei.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
En_Transmit(can_no, msg_no, id, id_extend)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG
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_CAN_Interrupt, En_Receive, Get_CAN_Reg, Transmit

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

  INIT:
    Init_CAN(1)                  'CAN-Controller 1 initialisieren
    Rem Initialisere Message-Objekte der Schnittstelle 1:
    Rem Objekt 2 zum Empfangen mit Identifier 200,
    Rem Objekt 6 zum Senden mit Identifier 40
    En_Receive(1,2,200,0)
    En_Transmit(1,6,40,0)
```

Get_CAN_Reg

T11 TiCo

`Get_CAN_Reg` gibt den Wert eines bestimmten Registers im Controller einer CAN-Schnittstelle zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Get_CAN_Reg(can_no, regno)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG
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 (Address map). 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

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
  INIT:
    Init_CAN(1)                  'CAN-Controller 1 initialisieren
    PAR_1 = Get_CAN_Reg(1,0)'Control-Register auslesen
```

Init_CAN

T11 TiCo

Init_CAN initialisiert den Controller einer CAN-Schnittstelle.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Init_CAN(can_no)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG

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_CAN_Interrupt, En_Receive, En_Transmit, Get_CAN_Reg, Read_Msg

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

  INIT:
    Init_CAN(1) 'Initialisiere den CAN-Controller 1
```

Read_Msg

T11 TiCo

Read_Msg gibt zurück, ob eine neue Nachricht in einem Message-Objekt einer CAN-Schnittstelle empfangen wurde.

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Read_Msg(can_no, msg_no)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG
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_Receive, En_Transmit, Get_CAN_Reg, Read_Msg_Con, Transmit

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
  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(1)                  'CAN-Controller 1 initialisieren
    Rem Message-Objekt 1 initialisieren zum Empfangen von
    Rem CAN-Nachrichten mit dem Identifier 40
    En_Receive(1,1,40,0)

  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(1,1)

    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, mit
      For n = 2 TO 4             ' restlichen 3 Bytes zu 32
  Bit-Zahl
        PAR_1 = Shift_Left(PAR_1,8) + CAN_Msg[n] 'zusammenfügen
      Next n
      Rem Bitmuster in PAR_1 in den Datentyp FLOAT wandeln und
      Rem der Variablen FPAR_1 zuweisen.
      FPAR_1 = Cast_LongToFloat(PAR_1)
    EndIf
REM Senden einer Fließkomma-Zahl siehe Bsp. bei Transmit.
```

Read_Msg_Con

T11 TiCo

Read_Msg_Con prüft, ob eine vollständige neue Nachricht in einem bestimmten Message-Objekt in einer CAN-Schnittstelle empfangen wurde.

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Read_Msg_Con(can_no,msg_no)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG
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_Con**.

Sie können eine empfangene Nachricht nur einmal auslesen.

Siehe auch

CAN_Msg, En_CAN_Interrupt, En_Receive, En_Transmit, Read_Msg

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
  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(1)                  'CAN-Controller 1 initialisieren
    Rem Message-Objekt 1 initialisieren zum Empfangen von
    Rem CAN-Nachrichten mit Identifier 40
    En_Receive(1,1,40,0)


  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(1,1)

    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 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

T11 TiCo

`Set_CAN_Baudrate` stellt die Baudrate des Controllers einer CAN-Schnittstelle ein.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Set_CAN_Baudrate(can_no, rate)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG
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 (1 MBit/s) überschrieben wird.

Siehe auch

Init_CAN, Get_CAN_Reg, Set_CAN_Reg

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

  #Define status Par_1          'Status in PAR_1

  INIT:
    Init_CAN(1)                 'CAN-Controller 1 initialisieren
    Rem Baudrate 125 kBit/s setzen
    status = Set_CAN_Baudrate(1,125000)
```

Set_CAN_Reg

T11 TiCo

Set_CAN_Reg schreibt einen Wert in ein bestimmtes Register des Controllers einer CAN-Schnittstelle.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Set_CAN_Reg(can_no, regno, value)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG
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

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

  INIT:
    Init_CAN(1)                 'CAN-Controller 1 initialisieren
    Set_CAN_Reg(1,0,1)          'Control-Register auf den Wert 1
                                'setzen
```

Transmit

T11 TiCo

Transmit sendet die Nachricht in CAN_Msg über ein bestimmtes Message-Objekt einer CAN-Schnittstelle.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Transmit(can_no, msg_no)
```

Parameter

can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG
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.
- Vor dem Senden: Fragen Sie mit **Transmit_Status** ab, ob das Message-Objekt zum Senden bereit ist.
- 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, Read_Msg, Transmit_Status

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
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(2)                 'CAN-Controller 2
                              'initialisieren

  Rem Initialisiere das Message-Objekt 6 der Schnitt-
  Rem stelle 2 zum Senden von CAN-Nachrichten mit dem
  Rem Identifier 40
  En_Transmit(2, 6,40,0)

  Rem Bitmuster von Pi mit Datenformat Long erzeugen
  PAR_1 = Cast_FloatToLong(pi)

  Rem Bitmuster (32 Bit) in 4 Bytes aufteilen
  For i = 0 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(2,6)               'Message-Objekt 6 senden
```

Empfangen einer Fließkomma-Zahl siehe Bsp. bei Read_Msg.

Transmit_Status

T11 TiCo

`Transmit_Status` gibt zurück, ob ein Message-Objekt bereit ist zum Senden.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Transmit_Status(channel, msg_no)
```

Parameter

channel	Nummer (1, 2) der CAN-Schnittstelle	LONG
msg_no	Nummer (1…14) des Message-Objektes im CAN-Controller	LONG
ret_val	Status des Message-Objekts. 0: Bereit zum Senden. 1: Nicht bereit zum Senden.	LONG

Bemerkungen

Der Rückgabewert ist nur für solche Message-Objekte sinnvoll, die zum Senden konfiguriert sind.

Ein Message-Objekt, das nicht bereit zum Senden ist, enthält noch eine Nachricht, die gesendet werden soll oder gerade gesendet wird.

Die CAN-Schnittstelle sendet die Nachricht, sobald das Message-Objekt Zugriffsrecht auf den CAN-Bus hat.

Sie können Nachrichten auch verschicken, ohne vorher den Status des Message-Objekts abzufragen. Wenn Sie jedoch Nachrichten schneller bereitstellen als der CAN-Controller sie verschicken kann, gehen einzelne Nachrichten verloren.

Siehe auch

CAN_Msg, En_Transmit, Init_CAN, Read_Msg, Transmit

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc 'für TiCoBasic

Init:
  Init_CAN(1)                 'CAN-Controller initialisieren
  En_Transmit(1,6,40,0)       'Message-Objekt 6 initialisieren
  Par_1 = 0
  CAN_Msg[1] = Par_1          'Wert setzen
  CAN_Msg[9] = 1              'Länge der Nachricht in Bytes

Event:
  Inc(Par_1)
  CAN_Msg[1] = Par_1          'Wert setzen
  If (Transmit_Status(1,6) = 0) Then 'bereit zum Senden?
    Transmit(1,6)             'Message-Objekt 6 senden
  EndIf
  If (Par_1 = 255) Then Par_1 = 0
```

16.8 RSxxx-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen der RSxxx-Schnittstellen auf *ADwin-Gold II*:

Check_Shift_Reg

T11 TiCo

Check_Shift_Reg gibt zurück, ob alle Daten gesendet sind, die in den Sende-FIFO der RSxxx-Schnittstelle geschrieben wurden.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Check_Shift_Reg(channel)
```

Parameter

channel	Nummer (1, 2) der Schnittstelle, deren Sende-Status geprüft wird.	LONG
ret_val	Sende-Status: 0: Daten sind gesendet (= keine Daten im Sende-FIFO vorhanden). 1: Noch nicht alle Daten gesendet (= im Sende-FIFO sind noch Daten vorhanden).	LONG

Bemerkungen

Bei dem Rückgabewert 0 ist sowohl das Sende-FIFO als auch das Ausgangs-Shiftregister leer. Bei dem Rückgabewert 1 ist mindestens ein Bit noch nicht gesendet.

Benutzen Sie diesen Befehl nur, wenn Sie sich bereits eingehend mit dem eingesetzten Controller vertraut gemacht haben (Datenblatt des Herstellers Texas Instruments). Für allgemeine Anwendungen stehen Ihnen komfortablere Befehle aus der Include-Datei zur Verfügung.

Siehe auch

Get_RS, RS_Init, RS_Reset, Write_Fifo, Write_Fifo_Full

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

  EVENT:
    Rem …
    Rem Prüft, ob Schnittstelle 1 noch Daten zu senden hat
    PAR_1 = Check_Shift_Reg(1)
    Rem …
```

Get_RS

T11 TiCo

`Get_RS` liest den Inhalt eines bestimmten Controller-Registers aus.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Get_RS(reg_addr)
```

Parameter

`reg_addr`	Adresse des zu lesenden Controller-Registers.	LONG
`ret_val`	Inhalt des Controller-Registers.	LONG

Bemerkungen

Benutzen Sie diesen Befehl nur, wenn Sie sich bereits eingehend mit dem eingesetzten Controller vertraut gemacht haben (Datenblatt des Herstellers Texas Instruments). Für allgemeine Anwendungen stehen Ihnen komfortablere Befehle aus der Include-Datei zur Verfügung.

Siehe auch

Check_Shift_Reg, RS_Init, RS_Reset, Set_RS

Gültig für

Gold II-CAN

Beispiel

- / -

Read_Fifo

T11 TiCo

Read_Fifo liest einen Wert aus dem Eingangs-FIFO einer bestimmten Schnittstelle.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Read_Fifo(channel)
```

Parameter

channel	Nummer (1, 2) der auszulesenden Schnittstelle.	LONG
ret_val	Inhalt des Eingangs-FIFO: -1: FIFO ist leer ≥0: Übertragener Datenwert	LONG

Bemerkungen

-/-

Siehe auch

RS_Init, RS_Reset, RS485_Send, Write_Fifo, Write_Fifo_Full

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

INIT:
  RS_Reset()
  Rem Schnittstelle 1 initialisieren:  9600 Baud, ohne Parität,
  Rem 8 Datenbits, 1 Stoppbit und Hardwarehandshake.
  RS_Init(1,9600,0,8,0,1)

EVENT:
  Rem Einen Wert aus dem FIFO holen. Wenn der FIFO leer ist,
wird -1
  Rem zurückgeliefert.
  PAR_1 = Read_Fifo(1)
```

RS485_Send

T11 TiCo

`RS485_Send` legt die Übertragungsrichtung für eine bestimmte Schnittstelle fest.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
RS485_Send(channel,dir)
```

Parameter

`channel`	Einzustellende Schnittstelle (1, 2)	LONG
`dir`	Übertragungsrichtung: 0: Schnittstelle als Empfänger einstellen. 1: Schnittstelle als Sender einstellen. 2: Schnittstelle als Sender einstellen, der gleichzeitig die gesendeten Daten empfängt. 3: Schnittstelle stumm schalten, d.h. die Schnittstelle arbeitet als Empfänger, nimmt aber keine Daten in den Eingangs-FIFO auf.	LONG

Bemerkungen

Die Einstellung der Übertragungsrichtung bedeutet:

- Empfänger: Der Controller kann Daten auf dem Bus ausschließlich lesen, auch wenn Daten im Ausgangs-FIFO liegen.
- Sender: Der Controller kann Daten auf den Bus legen, die von anderen Teilnehmern gelesen werden können.
- Sender/Empfänger: Der Controller kann Daten auf den Bus legen und gleichzeitig zurücklesen. Dadurch ist eine Überprüfung der ausgegebenen Daten möglich.

Siehe auch

Check_Shift_Reg, Get_RS, RS_Init, RS_Reset, Set_RS, Write_Fifo, Write_Fifo_Full

Gültig für

Gold II-CAN

Beispiel

- / -

RS_Init

T11 TiCo

RS_Init initialisiert die angegebene Schnittstelle.

Folgende Kennwerte werden gesetzt:

- Übertragungsgeschwindigkeit in Baud
- Anwendung von Prüf-Bits
- Datenlänge
- Anzahl der Stopp-Bits
- Übertragungs-Protokoll (Handshake)

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
RS_Init(channel,baud,parity,bits,stop,handshake)
```

Parameter

channel	Nummer (1, 2) der Schnittstelle, die initialisiert werden soll.	LONG
baud	Übertragungsgeschwindigkeit in Baud: RS232: 35 … 115.200 RS485: 35 … 2.304.000	LONG
parity	Anwendung von Prüf-Bits: 0: ohne Paritäts-Bit 1: gerade Parität (even) 2: ungerade Parität (odd)	LONG
bits	Anzahl der Daten-Bits (5, 6, 7 oder 8).	LONG
stop	Anzahl der Stopp-Bits 0: 1 Stopp-Bit 1: 1½ Stopp-Bits bei 5 Daten-Bits; 2 Stopp-Bits bei 6, 7 oder 8 Daten-Bits	LONG
handshake	Übertragungs-Protokoll: 0: RS232, kein Handshake 1: RS232, Hardware Handshake (RTS/CTS) 2: RS232, Software-Handshake (Xon/Xoff) 3: RS485 (Voreinstellung)	LONG

Bemerkungen

RS_Init ist vor dem ersten Arbeiten mit der gewählten Schnittstelle notwendig, um deren Parameter einzustellen. Sie müssen für eine korrekte Übertragung mit der Gegenstelle identisch sein.

Die Initialisierung ist auch dann erforderlich, wenn Sie mit **RS_Reset** einen Hardware-Reset ausgeführt haben.

Wenn das Übertragungs-Protokoll RS485 eingestellt wird, muss auch die Übertragungsrichtung festgelegt werden (mit **RS485_Send**).

Eine Liste gängiger Baudraten finden Sie auf Seite 39 (Abb. 22).

Siehe auch

Check_Shift_Reg, Get_RS, RS485_Send, RS_Reset, Set_RS, Write_Fifo, Write_Fifo_Full

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

  INIT:
    RS_Reset()                  'RS-Controller zurücksetzen
    Rem Schnittstelle 1 initialisieren:  9600 Baud, ohne Parität,
    Rem 8 Datenbits, 1 Stoppbit, Hardwarehandshake.
    RS_Init(1,9600,0,8,0,1)
```

RS_Reset

T11 TiCo

`RS_Reset` führt einen Hardware-Reset des RSxxx-Controllers durch.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
RS_Reset()
```

Parameter

- / -

Bemerkungen

`RS_Reset` sendet einen Reset-Impuls auf den entsprechenden Eingang des Controllers TL16C754. Sie können dem Datenblatt des Controllers 16C754 von Texas Instruments entnehmen, auf welche Werte die Register durch den Hardware-Reset gesetzt werden.

Nach einem Hardware-Reset muss eine Initialisierung mit `RS_Init` folgen, um den Controller zu initialisieren und die gewünschten Schnittstellen-Parameter einzustellen.

Siehe auch

Check_Shift_Reg, Get_RS, RS_Init, Set_RS

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic

  INIT:
    RS_Reset()                  'RSxxx Controller zurücksetzen
    Rem Kanal 1 initialisieren:  9600 Baud, ohne Parität,
    Rem 8 Datenbits, 1 Stoppbit und Hardwarehandshake.
    RS_Init(1,9600,0,8,0,1)
```

Set_RS

T11 TiCo

`Set_RS` schreibt einen Wert in ein bestimmtes Register.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Set_RS(reg_addr,value)
```

Parameter

`reg_addr`	Nummer des zu beschreibenen Registers	LONG
`value`	Wert, der in das Register geschrieben werden soll	LONG

Bemerkungen

Benutzen Sie diesen Befehl nur, wenn Sie sich bereits eingehend mit dem eingesetzten Controller vertraut gemacht haben (Datenblatt des Herstellers: TL16C754 von Texas Instruments). Für allgemeine Anwendungen stehen Ihnen komfortablere Befehle aus der Include-Datei zur Verfügung.

Siehe auch

Get_RS, RS_Init, RS_Reset

Gültig für

Gold II-CAN

Beispiel

- / -

Write_Fifo

T11 TiCo

Write_Fifo schreibt einen Wert in den Sende-FIFO einer bestimmten Schnittstelle.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Write_Fifo(channel,value)
```

Parameter

channel	Nummer (1, 2) der Schnittstelle, deren Sende-FIFO beschrieben wird	LONG
value	Wert der ins Sende-FIFO geschrieben werden soll.	LONG
ret_val	Statusmeldung: 0: Daten wurden erfolgreich geschrieben. 1: Daten konnten nicht geschrieben werden, Sende-FIFO ist voll.	LONG

Bemerkungen

Die Anweisung prüft zuerst, ob noch mindestens ein Speicherplatz im Sende-FIFO frei ist. Ist dies der Fall, wird der übergebene Wert ins FIFO geschrieben (Rückgabewert 0); anderenfalls wird eine 1 zurückgeliefert, die angibt, dass das FIFO voll ist und ein Schreiben nicht möglich war.

Siehe auch

Check_Shift_Reg, Read_Fifo, RS_Init, RS_Reset, RS485_Send, Write_Fifo_Full

Gültig für

Gold II-CAN

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
  Dim val As Long

  INIT:
    RS_Reset()
    Rem Initialisierung von Schnittstelle 1 mit 9600 Baud,
    Rem keine Parität, 8 Datenbits, 1 Stoppbit und
    Rem Hardware-Handshake.
    RS_Init(1,9600,0,8,0,1)

  EVENT:
    Rem Ist das FIFO nicht voll, wird val ins FIFO geschrieben.
    Rem Wenn das FIFO-Feld voll ist, wird dies mit dem Wert 1
    Rem in PAR_1 angezeigt.
    PAR_1 = Write_Fifo(1,val)
```

Write_Fifo_Full

T11 TiCo

`Write_Fifo_Full` gibt zurück, ob noch mindestens ein Speicherplatz im Sende-Fifo einer bestimmten Schnittstelle frei ist.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Write_Fifo_Full(channel)
```

Parameter

`channel`	Nummer (1, 2) der Schnittstelle, deren Sende-Fifo beschrieben wird.	LONG
`ret_val`	Statusmeldung: 0: Im Sende-Fifo ist mindestens ein Element frei. 1: Sende-Fifo ist voll.	LONG

Bemerkungen

Der Rückgabewert ist der gleiche wie bei `Write_Fifo`.

Siehe auch

Check_Shift_Reg, Read_Fifo, RS_Init, RS_Reset, RS485_Send, Write_Fifo

Gültig für

Gold II-CAN

Beispiel

```
Rem sending data to and receiving data from the PC while using
Rem a Fifo in ADwin-Gold II
#Include ADwinGoldII.inc'for ADbasic
Rem #Include GoldIITiCo.inc 'for TiCoBasic
#Define outfifo Data_1
#Define infifo Data_2
#Define rs_channel 1
#Define sending Par_10     '0: not sending, 1:sending
#Define receiving Par_11   '0: not receiving, 1:receiving

Dim outfifo[1000] As Long As Fifo
Dim infifo[1000] As Long As Fifo
Dim send_time, rec_time, value, check As Long

Init:
  Rem reset and initialize channel
  RS_Reset(rs_channel)
  RS_Init(rs_channel, 9600, 0, 8, 0, 0)
  Fifo_Clear(1)
  Fifo_Clear(2)
  sending = 0 : receiving = 0

Event:
  Rem sending
  If (Fifo_Full(1) > 0) Then 'any data present?
    If (Write_Fifo_Full(rs_channel) = 0) Then'send Fifo empty?
      value = outFifo         'read value from Fifo
      check = Write_Fifo(rs_channel, value)
      Rem check is not used, since Write_Fifo_Full has
      Rem proved that Fifo has empty elements.

      If (sending = 0) Then
        sending = 1
        send_time = Read_Timer()
      EndIf
    EndIf
  EndIf

  Rem receiving
  If (Fifo_Empty(2) > 0) Then 'are there empty elements?
    check = Read_Fifo(rs_channel)
    If (check <> -1) Then    'is a value in the receiving buffer?
      inFifo = check          'get value into inFifo

      If (receiving = 0) Then
        receiving = 1
        rec_time = Read_Timer()
      EndIf
    EndIf
  EndIf

  'reset status after defined delay (10ms)
  If (sending > 0) Then
    If ((Read_Timer() - send_time) > 3000000) Then
      sending = 0
      send_time = Read_Timer()
    EndIf
  EndIf
  If (receiving > 0) Then
    If ((Read_Timer() - rec_time) > 3000000) Then
      receiving = 0
      rec_time = Read_Timer()
    EndIf
  EndIf
```

Siehe auch weitere Beispiele für RS232 und RS485 (Pro I) ab Seite 2058.

16.9 Profibus-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen des Profibusknotens auf *ADwin-Gold II*:

Init_Profibus

T11

`Init_Profibus` initialisiert den Profibus-Slave.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Init_Profibus(dev_adr, in_mod_cnt,
    in_mod_type, out_mod_cnt, out_mod_type,
    work_arr[], info[])
```

Parameter

`dev_adr`	Slave-Knotenadresse / Stationsadresse (1…125) auf dem Profibus.	LONG
`in_mod_cnt`	Anzahl (0…76) der Eingangs-Datenbereiche im Profibus-Slave. Die max. Anzahl hängt von der Kennzahl `in_mod_type` ab.	LONG
`in_mod_type`	Kennzahl (1…3, 16) für die Länge der Eingangs-Datenbereiche: 1: 1 Byte; max. Wert für `in_mod_cnt`: 76. 2: 2 Byte; max. Wert für `in_mod_cnt`: 38. 3: 4 Byte; max. Wert für `in_mod_cnt`: 19. 16: 8 Byte; max. Wert für `in_mod_cnt`: 9.	LONG
`out_mod_cnt`	Anzahl (0…76) der Ausgangs-Datenbereiche im Profibus-Slave. Die max. Anzahl hängt von der Kennzahl `out_mod_type` ab.	LONG
`out_mod_type`	Kennzahl (1…3, 16) für die Länge der Ausgangs-Datenbereiche: 1: 1 Byte; max. Wert für `out_mod_type`: 76. 2: 2 Byte; max. Wert für `out_mod_type`: 38. 3: 4 Byte; max. Wert für `out_mod_type`: 19. 16: 8 Byte; max. Wert für `out_mod_type`: 9.	LONG
`work_arr[]`	Feld, das Daten für den Betrieb des Profibus-Slave aufnimmt. Das Feld muss mind. 200 Elemente haben.	ARRAY LONG
`info[]`	Feld, das Daten über den Profibus-Slave enthält. Das Feld muss mind. 10 Elemente haben. Die Elemente `info[1]` und `info[2]` enthalten den Produktionstyp des Profibus-Slave: `info[1]=1`, `info[2]=4`	ARRAY LONG
`ret_val`	Status der Initialisierung: 0: kein Fehler. ≠0: Fehler; bitte melden Sie sich beim Support von Jäger Messtechnik.	LONG

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem Profibus-Slave ausgeführt werden.

`Init_Profibus` soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit (etwa 2-3 Sekunden) dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Stationsadresse, Anzahl und Größe der Datenbereiche müssen die gleichen sein wie bei der Projektierung des Profibus. Die Modullänge wird bei der Projektierung auch in Worten angegeben: 1 Wort = 2 Byte.

Gültig für

Gold II-Profibus

Siehe auch

Run_Profibus

Beispiel

```
#Include ADwinGoldII.INC
#Define node 2              'slave node address
#Define info Data_1         'info array
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[76] As Long At DM_Local
Dim in_arr[76] As Long At DM_Local
Dim conf_arr[200] As Long At DM_Local
Dim info[10] As Long At DM_Local
Dim i As Long
Dim error As Long

LowInit:
  Processdelay = 3000000    'set to 100 Hz
  For i = 1 To 10           'initialize info array
    info[i] = 0
  Next i
  Rem initialize profibus interface: 38 input data areas of 2 byte
  Rem and 76 output data bytes of 1 Byte
  error = Init_Profibus(node,38,2,76,1,conf_arr,info)
  If (error <> 0) Then      'initialization error
    Par_1 = error
    Exit
  EndIf

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 76
    out_arr[i] = (out_arr[i] + i) And 0FFh
  Next i

  Rem send and read data (output bytes: 76; input bytes: 76)
  error = Run_Profibus(out_arr,76,in_arr,76,conf_arr)And 7h
  Par_2 = error

  Rem here the received data in in_arr[] can be processed
```

Run_Profibus

T11

Run_Profibus tauscht Daten mit dem Profibus-Slave aus.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Run_Profibus(out_pd_arr[], out_pd_arr_len,
    in_pd_arr[], in_pd_arr_len, work_arr[])
```

Parameter

out_pd_arr[]	Feld, aus dem der Profibus-Slave Daten liest und auf den Profibus schreibt.	ARRAY LONG
out_pd_arr_len	Anzahl der Ausgangs-Bytes (1…76), deren Daten aus dem Feld out_pd_arr[] gelesen werden. Die Anzahl darf nicht größer sein als in out_mod_cnt bei **Init_Profibus** angegeben wurde.	LONG
in_pd_arr[]	Feld, in das der Profibus-Slave Daten schreibt, die vom Profibus gelesen werden.	ARRAY LONG
in_pd_arr_len	Anzahl der Eingangs-Bytes (1…76), deren Daten im Feld in_pd_arr[] zurückgegeben werden. Die Anzahl darf nicht größer sein als in in_mod_cnt bei **Init_Profibus** angegeben wurde.	LONG
work_arr[]	Feld, das Daten für den Betrieb des Profibus-Slave enthält, siehe **Init_Profibus**.	ARRAY LONG
ret_val	Bitmuster, das den Betriebszustand des Profibus-Slave angibt. Von Bedeutung sind die Bits 2:0: 100b: Slave ist aktiv und arbeitet korrekt. 010b: Profibus nicht aktiv, Slave im Wartezustand. 110b, 111b: Fehler.	LONG

Bemerkungen

Run_Profibus soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Jedes Feldelement in in_pd_arr[] und out_pd_arr[] enthält nur 1 Datenbyte (Bits 7:0). Datenbereiche aus mehreren Bytes werden in entsprechend vielen, aufeinander folgenden Feldelementen abgelegt.
Beispiel: 5 Datenbereiche mit je 4 Byte Länge werden in 5x4=20 Feldelementen gespeichert.

Gültig für

Gold II-Profibus

Siehe auch

Init_Profibus

Beispiel

siehe Init_Profibus

16.10Profinet-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen des Profinet-Knotens auf *ADwin-Gold II*:

Init_ProfinetIO

T11

`Init_ProfinetIO` initialisiert den Profinet-Slave.

Syntax

```
#Include ADwinGoldII.inc
Init_ProfinetIO(in_mod_cnt, in_mod_type,
    out_mod_cnt, out_mod_type, work_arr[])
```

Parameter

`in_mod_cnt`	Anzahl (0…76) der Eingangs-Datenbereiche im Profinet-Slave. Die max. Anzahl hängt von der Kennzahl `in_mod_type` ab.	LONG
`in_mod_type`	Kennzahl (1…3, 16) für die Länge der Eingangs-Datenbereiche: 1: 1 Byte; max. Wert für `in_mod_cnt`: 76. 2: 2 Byte; max. Wert für `in_mod_cnt`: 38. 3: 4 Byte; max. Wert für `in_mod_cnt`: 19. 16: 8 Byte; max. Wert für `in_mod_cnt`: 9.	LONG
`out_mod_cnt`	Anzahl (0…76) der Ausgangs-Datenbereiche im Profinet-Slave. Die max. Anzahl hängt von der Kennzahl `out_mod_type` ab.	LONG
`out_mod_type`	Kennzahl (1…3, 16) für die Länge der Ausgangs-Datenbereiche: 1: 1 Byte; max. Wert für `out_mod_type`: 76. 2: 2 Byte; max. Wert für `out_mod_type`: 38. 3: 4 Byte; max. Wert für `out_mod_type`: 19. 16: 8 Byte; max. Wert für `out_mod_type`: 9.	LONG
`work_arr[]`	Feld, das Daten für den Betrieb des Profinet-Slave aufnimmt. Das Feld muss mind. 4096 Elemente haben.	ARRAY LONG

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem Profinet-Slave ausgeführt werden.

`Init_ProfinetIO` soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Anzahl und Größe der Datenbereiche müssen die gleichen sein wie bei der Projektierung des Profinet. Achten Sie darauf, dass die Größe der Datenbereiche bei der Projektierung auch in anderen Einheiten als Byte angegeben werden kann (z.B. Word, QWord).

Gültig für

- / -

Siehe auch

Run_ProfinetIO

Beispiel

```
#Include ADwinGoldII.INC
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[76] As Long At DM_Local
Dim in_arr[76] As Long At DM_Local
Dim conf_arr[4096] As Long At DM_Local
Dim i As Long
Dim state As Long

LowInit:
  Processdelay = 3000000    'set to 100 Hz
  Rem initialize Profinet interface: 38 input data areas of 2 byte
  Rem and 76 output data bytes of 1 Byte
  error = Init_ProfinetIO(38,2,76,1,conf_arr)

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 76
    out_arr[i] = (out_arr[i] + i) And 0FFh
  Next i

  Rem send and read data (output bytes: 76; input bytes: 76)
  state = Run_ProfinetIO(out_arr,76,in_arr,76,conf_arr)And 7h
  Par_2 = state

  Rem here the received data in in_arr[] can be processed
```

Run_ProfinetIO

T11

Run_ProfinetIO tauscht Daten mit dem Profinet-Slave aus.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Run_ProfinetIO(out_pd_arr[],
    out_pd_arr_len, in_pd_arr[], in_pd_arr_len,
    work_arr[])
```

Parameter

out_pd_arr[]	Feld, aus dem der Profinet-Slave Daten liest und auf den Profinet schreibt.	ARRAY LONG
out_pd_arr_len	Anzahl der Ausgangs-Bytes (1…76), deren Daten aus dem Feld out_pd_arr[] gelesen werden. Die Anzahl darf nicht größer sein als in out_mod_cnt bei **Init_ProfinetIO** angegeben wurde.	LONG
in_pd_arr[]	Feld, in das der Profinet-Slave Daten schreibt, die vom Profinet gelesen werden.	ARRAY LONG
in_pd_arr_len	Anzahl der Eingangs-Bytes (1…76), deren Daten im Feld in_pd_arr[] zurückgegeben werden. Die Anzahl darf nicht größer sein als in in_mod_cnt bei **Init_ProfinetIO** angegeben wurde.	LONG
work_arr[]	Feld, das Daten für den Betrieb des Profinet-Slave enthält, siehe **Init_ProfinetIO**.	ARRAY LONG
ret_val	Betriebszustand des Profinet-Slave: 0: Initialisierung. 2: Slave wartet auf Busstart durch den Master.. 4: Normaler Betrieb. 5: Fehler im Betrieb. 7: Exception.	LONG

Bemerkungen

Run_ProfinetIO soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Jedes Feldelement in in_pd_arr[] und out_pd_arr[] enthält nur 1 Datenbyte (Bits 7:0). Datenbereiche aus mehreren Bytes werden in entsprechend vielen, aufeinander folgenden Feldelementen abgelegt.
Beispiel: 5 Datenbereiche mit je 4 Byte Länge werden in 5x4=20 Feldelementen gespeichert.

Gültig für

- / -

Siehe auch

Init_ProfinetIO

Beispiel

siehe Init_ProfinetIO

16.11DeviceNet-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen der DeviceNet-Schnittstelle auf *ADwin-Gold II*:

Init_DeviceNet

T11

`Init_DeviceNet` initialisiert den DeviceNet-Slave.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Init_DeviceNet(dev_adr, baudrate,
    in_mod_cnt, in_mod_type, out_mod_cnt,
    out_mod_type, work_arr[], info[])
```

Parameter

`dev_adr`	Slave-Knotenadresse / Stationsadresse (1…125) auf dem DeviceNet.	LONG
`baudrate`	Kennzahl für die Baudrate, mit der der DeviceNet-Bus betrieben wird: 0: 125 kBit 1: 250 kBit 2: 500 kBit 3: automatische Anpassung an die aktuelle Baudrate des DeviceNet-Bus (Autobaud).	LONG
`in_mod_cnt`	Anzahl (1…255) der Eingangs-Datenbereiche im DeviceNet-Slave. Die max. Anzahl hängt von der Kennzahl `in_mod_type` ab.	LONG
`in_mod_type`	Kennzahl (1…3, 16) für die Länge der Eingangs-Datenbereiche: 1: 1 Byte; max. Wert für `in_mod_cnt`: 255. 2: 2 Byte; max. Wert für `in_mod_cnt`: 127. 3: 4 Byte; max. Wert für `in_mod_cnt`: 63. 16: 8 Byte; max. Wert für `in_mod_cnt`: 31.	LONG
`out_mod_cnt`	Anzahl (1…255) der Ausgangs-Datenbereiche im DeviceNet-Slave. Die max. Anzahl hängt von der Kennzahl `out_mod_type` ab.	LONG
`out_mod_type`	Kennzahl (1…3, 16) für die Länge der Ausgangs-Datenbereiche: 1: 1 Byte; max. Wert für `out_mod_type`: 255. 2: 2 Byte; max. Wert für `out_mod_type`: 127. 3: 4 Byte; max. Wert für `out_mod_type`: 63. 16: 8 Byte; max. Wert für `out_mod_type`: 31.	LONG
`work_arr[]`	Feld, das Daten für den Betrieb des DeviceNet-Slave aufnimmt. Das Feld muss mind. 200 Elemente haben.	ARRAY LONG
`info[]`	Feld, das Daten über den DeviceNet-Slave enthält. Das Feld muss mind. 10 Elemente haben. Die Elemente `info[1]` und `info[2]` enthalten den Produktionstyp des DeviceNet-Slave: `info[1]`=1, `info[2]`=4	ARRAY LONG
`ret_val`	Status der Initialisierung: 0: kein Fehler. ≠0: Fehler; bitte melden Sie sich beim Support von Jäger Messtechnik.	LONG

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem DeviceNet-Slave ausgeführt werden.

`Init_DeviceNet` soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit (etwa 2-3 Sekunden) dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Stationsadresse, Anzahl und Größe der Datenbereiche müssen die gleichen sein wie bei der Projektierung des DeviceNet. Die Modullänge wird bei der Projektierung auch in Worten angegeben: 1 Wort = 2 Byte.

Gültig für

Gold II-DeviceNet

Siehe auch

Run_DeviceNet

Beispiel

```
#Include ADwinGoldII.INC
#Define node 2              'slave node address
#Define info DATA_1         'info array
#Define out_arr DATA_2
#Define in_arr DATA_3

Dim out_arr[255] As Long At DM_Local
Dim in_arr[254] As Long At DM_Local
Dim conf_arr[200] As Long At DM_Local
Dim info[10] As Long At DM_Local
Dim i As Long
Dim error As Long

LowInit:
  Processdelay = 3000000    'set to 100 Hz
  For i = 1 To 10           'initialize info array
    info[i] = 0
  Next i
  Rem initialize DeviceNet interface: autobaud, 127 input data
  Rem areas of 2 byte and 255 output data bytes of 1 Byte
  error = Init_DeviceNet(node,3,127,2,255,1,conf_arr,info)
  If (error <> 0) Then      'initialization error
    PAR_1 = error
    Exit
  EndIf

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 255
    out_arr[i] = (out_arr[i] + i) And 0FFh
  Next i

  Rem send and read data (output bytes: 255; input bytes: 254)
  error = Run_DeviceNet(out_arr,255,in_arr,254,conf_arr) And 7
  PAR_2 = error

  Rem here the received data in in_arr[] can be processed
```

Run_DeviceNet

T11

Run_DeviceNet tauscht Daten mit dem DeviceNet-Slave aus.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Run_DeviceNet(in_pd_arr[], in_pd_arr_len,
    out_pd_arr[], out_pd_arr_len, work_arr[])
```

Parameter

in_pd_arr[]	Feld, in das der DeviceNet-Slave Daten schreibt, die vom DeviceNet gelesen werden.	ARRAY LONG
in_pd_arr_len	Anzahl der Eingangs-Bytes (1…255), deren Daten im Feld in_pd_arr[] zurückgegeben werden. Die Anzahl darf nicht größer sein als in in_mod_cnt bei **Init_DeviceNet** angegeben wurde.	LONG
out_pd_arr[]	Feld, aus dem der DeviceNet-Slave Daten liest und auf den DeviceNet schreibt.	ARRAY LONG
out_pd_arr_len	Anzahl der Ausgangs-Bytes (1…255), deren Daten aus dem Feld out_pd_arr[] gelesen werden. Die Anzahl darf nicht größer sein als in out_mod_cnt bei **Init_DeviceNet** angegeben wurde.	LONG
work_arr[]	Feld, das Daten für den Betrieb des DeviceNet-Slave enthält, siehe **Init_DeviceNet**.	ARRAY LONG
ret_val	Kennzahl, die den Betriebszustand des DeviceNet-Slave angibt: 0: Slave wird initialisiert. 2: DeviceNet nicht aktiv, Slave im Wartezustand. 4: DeviceNet ist aktiv. andere Werte: Fehler.	LONG

Bemerkungen

Run_DeviceNet soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Jedes Feldelement in in_pd_arr[] und out_pd_arr[] enthält nur 1 Datenbyte (Bits 7:0). Datenbereiche aus mehreren Bytes werden in entsprechend vielen, aufeinander folgenden Feldelementen abgelegt.
Beispiel: 5 Datenbereiche mit je 4 Byte Länge werden in 5x4=20 Feldelementen gespeichert.

Gültig für

Gold II-DeviceNet

Siehe auch

Init_DeviceNet

Beispiel

siehe Init_DeviceNet

16.12EtherCAT-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen der EtherCAT-Schnittstelle auf *ADwin-Gold II*:

ECAT_Init

T11

`ECAT_Init` initialisiert die EtherCAT-Schnittstelle.

Syntax

```
#Include ADwinGoldII.inc
ret_val = ECAT_Init(in_mod_cnt, in_mod_type,
    out_mod_cnt, out_mod_type, work_arr[], info[])
```

Parameter

`in_mod_cnt`	Anzahl (1…254) der Eingangs-Datenbereiche in der EtherCAT-Schnittstelle. Die max. Anzahl hängt von der Kennzahl `in_mod_type` ab.	LONG
`in_mod_type`	Kennzahl (1…3, 16) für die Länge der Eingangs-Datenbereiche: 1: 1 Byte; max. Wert für `in_mod_cnt`: 254. 2: 2 Byte; max. Wert für `in_mod_cnt`: 127. 3: 4 Byte; max. Wert für `in_mod_cnt`: 63. 16: 8 Byte; max. Wert für `in_mod_cnt`: 15.	LONG
`out_mod_cnt`	Anzahl (1…254) der Ausgangs-Datenbereiche in der EtherCAT-Schnittstelle. Die max. Anzahl hängt von der Kennzahl `out_mod_type` ab.	LONG
`out_mod_type`	Kennzahl (1…3, 16) für die Länge der Ausgangs-Datenbereiche: 1: 1 Byte; max. Wert für `out_mod_type`: 254. 2: 2 Byte; max. Wert für `out_mod_type`: 127. 3: 4 Byte; max. Wert für `out_mod_type`: 63. 16: 8 Byte; max. Wert für `out_mod_type`: 15.	LONG
`work_arr[]`	Feld, das Daten für den Betrieb der EtherCAT-Schnittstelle aufnimmt. Das Feld muss mind. 200 Elemente haben.	ARRAY LONG
`info[]`	Feld, das Daten über die EtherCAT-Schnittstelle enthält. Das Feld muss mind. 10 Elemente haben. Die Elemente `info[1]` und `info[2]` enthalten den Produktionstyp der EtherCAT-Schnittstelle: `info[1]=1`, `info[2]=4`	ARRAY LONG
`ret_val`	Status der Initialisierung: 0: kein Fehler. ≠0: Fehler; bitte melden Sie sich beim Support von Jäger Messtechnik.	LONG

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit der EtherCAT-Schnittstelle ausgeführt werden.

`ECAT_Init` soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit (etwa 2-3 Sekunden) dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Wenn Sie die Schnittstelle extern (z.B. mit dem Programm TwinCAT System Manager) konfigurieren, müssen Sie die Schnittstelle dennoch in *ADbasic* konfigurieren und dabei die gleichen Einstellungen verwenden.

Gültig für

Gold II-EtherCAT

Siehe auch

ECAT_Run

Beispiel

```
#Include ADwinGoldII.INC
#Define info Data_1        'info array
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[76] As Long At DM_Local
Dim in_arr[76] As Long At DM_Local
Dim conf_arr[200] As Long At DM_Local
Dim info[10] As Long At DM_Local
Dim i As Long
Dim error As Long

LowInit:
  Processdelay = 3000000    'set to 100 Hz
  For i = 1 To 10           'initialize info array
    info[i] = 0
  Next i
  Rem initialize EtherCAT interface: 38 input data areas of 2 byte
  Rem and 76 output data bytes of 1 Byte
  error = ECAT_Init(38,2,76,1,conf_arr,info)
  If (error <> 0) Then      'initialization error
    Par_1 = error
    Exit
  EndIf

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 76
    out_arr[i] = (out_arr[i] + i) And 0FFh
  Next i

  Rem send and read data (output bytes: 76; input bytes: 76)
  error = ECAT_Run(in_arr,76,out_arr,76,conf_arr)And 7h
  Par_2 = error

  Rem Here the received data in in_arr[] can be processed
```

ECAT_Run

T11

ECAT_Run tauscht Daten mit der EtherCAT-Schnittstelle aus.

Syntax

```
#Include ADwinGoldII.inc
ret_val = ECAT_Run(in_pd_arr[], in_pd_arr_len,
    out_pd_arr[], out_pd_arr_len, work_arr[])
```

Parameter

in_pd_arr[]	Feld, in das die EtherCAT-Schnittstelle Daten schreibt, die vom Bus gelesen werden.	ARRAY LONG
in_pd_arr_len	Anzahl der Eingangs-Bytes (1…254), deren Daten im Feld in_pd_arr[] zurückgegeben werden. Die Anzahl darf nicht größer sein als in in_mod_cnt bei **ECAT_Init** angegeben wurde.	LONG
out_pd_arr[]	Feld, aus dem die EtherCAT-Schnittstelle Daten liest und auf den Bus schreibt.	ARRAY LONG
out_pd_arr_len	Anzahl der Ausgangs-Bytes (1…254), deren Daten aus dem Feld out_pd_arr[] gelesen werden. Die Anzahl darf nicht größer sein als in out_mod_cnt bei **ECAT_Init** angegeben wurde.	LONG
work_arr[]	Feld, das Daten für den Betrieb der EtherCAT-Schnittstelle enthält, siehe **ECAT_Init**.	ARRAY LONG
ret_val	Bitmuster, das den Betriebszustand der EtherCAT-Schnittstelle angibt. Von Bedeutung sind die Bits 0…2: 000b: Schnittstelle wird initialisiert. 010b: Schnittstelle nicht aktiv, im Wartezustand. 100b: Schnittstelle ist aktiv und arbeitet korrekt. 110b, 111b: Fehler; evtl. Konfiguration prüfen.	LONG

Bemerkungen

ECAT_Run soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Jedes Feldelement in in_pd_arr[] und out_pd_arr[] enthält nur 1 Datenbyte (Bits 7:0). Datenbereiche aus mehreren Bytes werden in entsprechend vielen, aufeinander folgenden Feldelementen abgelegt.
Beispiel: 5 Datenbereiche mit je 4 Byte Länge werden in 5x4=20 Feldelementen gespeichert.

Gültig für

Gold II-EtherCAT

Siehe auch

ECAT_Init

Beispiel

siehe ECAT_Init

16.13Echtzeituhr

Dieser Abschnitt beschreibt Befehle zum Ansprechen der Echtzeituhr auf *ADwin-Gold II*:

RTC_Get

T11 TiCo

RTC_Get gibt das Datum und die Zeit von der Echtzeituhr zurück.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
RTC_Get(year,month,day,hour,minute,second)
```

Parameter

year	Jahreszahl (0…99), entspricht 2000…2099.	LONG
month	Monat (1…12).	LONG
day	Tag (1…31); gültiger Wertebereich je nach Monat und Schaltjahr.	LONG
hour	Stunde (0…23).	LONG
minute	Minute (0…59).	LONG
second	Sekunde (0…59).	LONG

Bemerkungen

Alle Parameter sind Rückgabeparameter; es müssen also Variablen als Parameter übergeben werden.

Siehe auch

RTC_Set

Gültig für

Gold II-Storage

Beispiel

```
Rem Wählen Sie das passende Include für ADbasic / TiCoBasic
#Include ADwinGoldII.inc'für ADbasic
Rem #Include GoldIITiCo.inc für TiCoBasic
Dim year,mon,day,h,m,s As Long

Init:
  Rem Echtzeituhr lesen
  RTC_Get(year,mon,day,h,m,s)
```

RTC_Set

T11

RTC_Set setzt das Datum und die Zeit auf der Echtzeituhr des angegebenen Moduls. Ungültige Werte werden nicht akzeptiert.

Syntax

```
#Include ADwinGoldII.inc
ret_val = RTC_Set(year,month,day,hour,minute,second)
```

Parameter

year	Jahreszahl (0…99), entspricht 2000…2099.	LONG
month	Monat (1…12).	LONG
day	Tag (1…31); gültiger Wertebereich je nach Monat und Schaltjahr.	LONG
hour	Stunde (0…23).	LONG
minute	Minute (0…59).	LONG
second	Sekunde (0…59).	LONG
ret_val	Status beim Setzen der Echtzeituhr: 0: Datum und Zeit sind gesetzt. -1: Fehler, Befehl darf nur mit niedriger Priorität genutzt werden. -2: Fehler, keine Echtzeituhr vorhanden.	LONG

Bemerkungen

Der Befehl RTC_Set darf nur in einem Prozessabschnitt mit niedriger Priorität genutzt werden.

Siehe auch

RTC_Get

Gültig für

Gold II-Storage

Beispiel

```
#Include ADwinGoldII.inc

LowInit:
  Rem Echtzeituhr auf 4.7.2003 9:17:30 setzen
  Par_1 = RTC_Set(3,7,4,9,17,30)
```

16.14Storage-Erweiterung (ADbasic)

Dieser Abschnitt beschreibt Befehle zum Ansprechen der Speicherkarte auf *ADwin-Gold II*:

Media_Init

T11

Media_Init initialisiert die Speicherkarte des *ADwin-Gold II*.

Syntax

```
#Include ADwinGoldII.Inc
ret_val = Media_Init(media_datatable[])
```

Parameter

media_datatable[]	Feld, das Daten für den Betrieb der Speicherkarte aufnimmt. Das Feld muss mind. 100 Elemente haben.	ARRAY LONG
ret_val	Status der Initialisierung: < 0: ein Fehler ist aufgetreten. Das Bitmuster gibt Hinweise auf die Fehlerursache. > 0: Initialisierung erfolgreich. ret_val enthält die Anzahl der verfügbaren Blöcke auf der Speicherkarte.	LONG

Bemerkungen

Der Befehl **Media_Init** steht nur in *ADbasic* zur Verfügung, nicht aber in *TiCoBasic*.

Media_Init muss in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit (etwa 2-3 Sekunden) dauert:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LowInit:** oder **Finish:** eines hoch-prioren Prozesses.

Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Jeder Datenblock auf der Speicherkarte enthält 128 Werte der Länge 32 Bit.

Siehe auch

Media_Erase, Media_Write, Media_Read

Gültig für

Gold II-Storage-16

Beispiel

```
#Include ADwinGoldII.inc
Dim err, num_blocks As Long
Dim media_datatable[100] As Long

LowInit:
  Rem initialize media card
  num_blocks = Media_Init(media_datatable)
  If (num_blocks < 0) Then Exit
  err = Media_Erase(media_datatable)
```

Media_Erase

T11

Media_Erase löscht die Speicherkarte vollständig.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Media_Erase(media_datatable[])
```

Parameter

media_datatable[]	Feld, das Daten für den Betrieb der Speicherkarte enthält, siehe **Media_Init**.	ARRAY LONG
ret_val	Status des Löschvorgangs: = 0: Die Speicherkarte wurde erfolgreich gelöscht. > 0: ein Hardware-Fehler ist aufgetreten. Bitte melden Sie sich bei unserem Support (Adresse siehe vordere Umschlagseite innen).	LONG

Bemerkungen

Der Befehl **Media_Erase** steht nur in *ADbasic* zur Verfügung, nicht aber in *TiCoBasic*.

Bevor die Anweisung genutzt werden kann, muss das Feld media_datatable[] mit **Media_Init** initialisiert werden. Der Befehl **Media_Init** steht nur in *ADbasic* zur Verfügung, nicht aber in *TiCoBasic*.

Media_Erase sollte nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LowInit:** oder **Finish:** eines hoch-prioren Prozesses.

Das Löschen der Speicherkarte kann das Lesen und Schreiben von Daten beschleunigen.

Siehe auch

Media_Init, Media_Write, Media_Read

Gültig für

Gold II-Storage-16

Beispiel

```
#Include ADwinGoldII.inc
Dim err, num_blocks As Long
Dim media_datatable[100] As Long

LowInit:
  Rem initialize media card
  num_blocks = Media_Init(media_datatable)
  If (num_blocks < 0) Then Exit
  err = Media_Erase(media_datatable)
```

Media_Read

T11

Media_Read kopiert eine Anzahl von Datenblöcken von der Speicherkarte des *ADwin-Gold II* in ein Feld.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Media_Read(media_datatable[], start_block,
    count_blocks128, dest_array[],
    array_start_index)
```

Parameter

media_datatable[]	Feld, das Daten für den Betrieb der Speicherkarte enthält, siehe **Media_Init**.	ARRAY LONG
start_block	Nummer (0…m-1) des ersten Datenblocks der Speicherkarte, der gelesen wird. m ist der Rückgabewert von **Media_Init**.	LONG
count_blocks128	Anzahl (1…m-1) der Datenblöcke, die gelesen werden. Ein Datenblock enthält 128 Werte zu 32 Bit. m ist der Rückgabewert von **Media_Init**.	LONG
dest_array[]	Feld, in das Daten übertragen werden. Es muss der gleiche Datentyp (Long oder Float) wie beim Schreiben der Daten verwendet werden.	ARRAY LONG FLOAT
array_start_index	Index (1…n) des ersten Feldelements, in das geschrieben wird.	LONG
ret_val	Status der Datenübertragung: = 0: Die Daten wurden erfolgreich übertragen. > 0: ein Hardware-Fehler ist aufgetreten. Bitte melden Sie sich bei unserem Support (Adresse siehe vordere Umschlagseite innen).	LONG

Bemerkungen

Bevor die Anweisung genutzt werden kann, muss die Speicherkarte mit **Media_Init** initialisiert werden. Der Befehl **Media_Init** steht nur in *ADbasic* zur Verfügung, nicht aber in *TiCoBasic*.

Die Anweisung sollte in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LowInit:** oder **Finish:** eines hoch-prioren Prozesses.

Der Aufruf in einem hoch-prioren Prozessabschnitt ist nur möglich, wenn nur wenige Datenblöcke übertragen werden. Anderenfalls würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Die Übertragungsgeschwindigkeit je Datenblock steigt mit der Anzahl der übertragenen Datenblöcke.

Das Feld dest_array[] muss mindestens count_blocks128 × 128 Elemente enthalten.

Der Datentyp des Felds dest_array[] muss mit dem ursprünglichen Datentyp übereinstimmen, mit dem die Werte auf die Speicherkarte übertragen wurden. Anderenfalls werden die Werte falsch interpretiert.

Werte auf der Speicherkarte haben immer eine Größe von 32 Bit, auch wenn es sich um Werte vom Datentyp Float handelt.

Siehe auch

Media_Init, Media_Erase, Media_Write

Gültig für

Gold II-Storage-16

Beispiel

- / -

Media_Write

T11

Media_Write kopiert Werte aus einem Feld blockweise auf die Speicherkarte des *ADwin-Gold II*.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Media_Write(media_datatable[],
    start_block, count_blocks128, source_array[],
    array_start_index)
```

Parameter

media_datatable[]	Feld, das Daten für den Betrieb der Speicherkarte enthält, siehe **Media_Init**.	ARRAY LONG
start_block	Nummer (0…m-1) des ersten Datenblocks der Speicherkarte, auf den geschrieben wird. m ist der Rückgabewert von **Media_Init**.	LONG
count_blocks128	Anzahl (1…m-1) der Datenblöcke, die geschrieben werden. Ein Datenblock enthält 128 Werte zu 32 Bit. m ist der Rückgabewert von **Media_Init**.	LONG
source_array[]	Feld, dessen Daten übertragen werden. Es ist der Datentyp Long oder Float erlaubt.	ARRAY LONG FLOAT
array_start_index	Index (1…n) des ersten zu übertragenden Feldelements.	LONG
ret_val	Status der Datenübertragung: = 0: Die Daten wurden erfolgreich übertragen. > 0: ein Hardware-Fehler ist aufgetreten. Bitte melden Sie sich bei unserem Support (Adresse siehe vordere Umschlagseite innen).	LONG

Bemerkungen

Bevor die Anweisung genutzt werden kann, muss die Speicherkarte mit **Media_Init** initialisiert werden. Der Befehl **Media_Init** steht nur in *ADbasic* zur Verfügung, nicht aber in *TiCoBasic*.

Die Anweisung sollte in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LowInit:** oder **Finish:** eines hochprioren Prozesses.

Der Aufruf in einem hoch-prioren Prozessabschnitt ist nur möglich, wenn nur wenige Datenblöcke übertragen werden. Anderenfalls würde die Kommunikation zwischen PC und *ADwin*-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Die Übertragungsgeschwindigkeit je Datenblock steigt mit der Anzahl der übertragenen Datenblöcke.

Das Feld source_array[] muss mindestens count_blocks128 × 128 Elemente enthalten.

Jeder Datenblock auf der Speicherkarte enthält 128 Werte der Länge 32 Bit, unabhängig vom Datentyp (Long oder Float). Wenn das Feld source_array[] Daten vom Typ Float – mit der Länge 40 Bit – enthält, werden die Fließkomma-Werte bei der Datenübertragung in das 32 Bit-Format transformiert.

Siehe auch

Media_Init, Media_Erase, Media_Read

Gültig für

Gold II-Storage-16

Beispiel

```
Rem store a sine table of about 3700 values
#Include ADwinGoldII.inc

#Define blocks Par_52      'number of medium data blocks
#Define val_per_block 128  'values per medium data block
#Define media_info Data_197'array with media information
#Define sine Data_1        'array for sine values
REM number of sine values, which is a multiple of val_per_block
#Define nds 3584           '3584 = 28 * val_per_block
#Define pi2 6.2831853      '2*pi

Dim media_info[100] As Long
Dim LUT[nds] As Float
Dim err, idx, blk_num As Long

LowInit:
  Rem initialize media card
  blocks = Media_Init(media_info)
  If (blocks < 0) Then Exit
  Rem erase memory to get maximum write speed (not necessary)
  Par_53 = Media_Erase(media_info)
  If (Par_53 > 0) Then Exit

  For idx = 1 To nds        'calculate sine values
    LUT[idx] = 32767.5 * Sin((idx-1) * pi2 / nds)
  Next idx

  Rem write values
  blk_num = nds/val_per_block
  If (blk_num > blocks) Then Exit
  err = Media_Write(media_info,1,blocks, LUT,1)
  If (err > 0) Then Exit
```

16.15Storage-Erweiterung (TiCoBasic)

Dieser Abschnitt beschreibt Befehle zum Ansprechen der Speicherkarte auf *ADwin-Gold II* mit dem *TiCo*-Prozessor:

Media_Read

TiCo

Media_Read kopiert eine Anzahl von Datenblöcken aus einem Feld auf die Speicherkarte des *ADwin-Gold II*.

Syntax

```
#Include GoldIITiCo.inc
ret_val = Media_Read(start_block, count_blocks128,
    dest_array[], array_start_index)
```

Parameter

start_block	Nummer (1…) des ersten Datenblocks der Speicherkarte, der gelesen wird.	LONG
count_blocks128	Anzahl (1…) der Datenblöcke, die gelesen werden. Ein Datenblock enthält 128 Werte zu 32 Bit.	LONG
dest_array[]	Feld, in das Daten übertragen werden.	ARRAY LONG
array_start_index	Index (1…n) des ersten zu Feldelements, in das geschrieben wird.	LONG
ret_val	Status der Datenübertragung: = 0: Die Daten wurden erfolgreich übertragen. > 0: ein Hardware-Fehler ist aufgetreten. Bitte melden Sie sich bei unserem Support (Adresse siehe vordere Umschlagseite innen).	LONG

Bemerkungen

Bevor die Anweisung genutzt werden kann, muss das Feld media_datatable[] mit **Media_Init** initialisiert werden. Der Befehl **Media_Init** steht nur in *ADbasic* zur Verfügung, nicht aber in *TiCoBasic*.

Der Befehl **Media_Read** hat eine relativ lange Bearbeitungsdauer; wir empfehlen deswegen die Verwendung in einem Prozess mit niedriger Priorität.

Die Übertragungsgeschwindigkeit je Datenblock steigt mit der Anzahl der übertragenen Datenblöcke.

Das Feld dest_array[] muss mindestens count_blocks128 × 128 Elemente enthalten.

Siehe auch

Media_Write

Gültig für

Gold II-Storage-16

Beispiel

- / -

Media_Write

TiCo

Media_Write kopiert Werte aus einem Feld blockweise auf die Speicherkarte des *ADwin-Gold II*.

Syntax

```
#Include GoldIITiCo.inc
ret_val = Media_Write(start_block, count_blocks128,
    source_array[], array_start_index)
```

Parameter

start_block	Nummer (1…) des ersten Datenblocks der Speicherkarte, auf den geschrieben wird.	LONG
count_blocks128	Anzahl (1…) der Datenblöcke, die geschrieben werden. Ein Datenblock enthält 128 Werte zu 32 Bit.	LONG
source_array[]	Feld, dessen Daten übertragen werden.	ARRAY LONG
array_start_index	Index (1…n) des ersten zu übertragenden Feldelements.	LONG
ret_val	Status der Datenübertragung: = 0: Die Daten wurden erfolgreich übertragen. > 0: ein Hardware-Fehler ist aufgetreten. Bitte melden Sie sich bei unserem Support (Adresse siehe vordere Umschlagseite innen).	LONG

Bemerkungen

Bevor die Anweisung genutzt werden kann, muss das Feld media_datatable[] mit **Media_Init** initialisiert werden. Der Befehl **Media_Init** steht nur in *ADbasic* zur Verfügung, nicht aber in *TiCoBasic*.

Der Befehl **Media_Write** hat eine relativ lange Bearbeitungsdauer; wir empfehlen deswegen die Verwendung in einem Prozess mit niedriger Priorität.

Die Übertragungsgeschwindigkeit je Datenblock steigt mit der Anzahl der übertragenen Datenblöcke.

Das Feld source_array[] muss mindestens count_blocks128 × 128 Elemente enthalten.

Siehe auch

Media_Read

Gültig für

Gold II-Storage-16

Beispiel

Teil 1: ADbasic

```
Rem ------------------------------
Rem Part 1: ADbasic program
Rem initialize media card
#Include ADwinGoldII.inc
#Define blocks PAR_52      'number of medium data blocks
#Define media_info DATA_197'array with media information

Dim media_info[100] As Long

LowInit:
  blocks = Media_Init(media_info)
  If (blocks < 0) Then Exit
  PAR_53 = Media_Erase(media_info)
  If (PAR_53 > 0) Then Exit
  Rem Set flag on TiCo processor: card is initialized
  Set_Par(1, 12, blocks)
```

Teil 2: TiCoBasic

```
Rem ------------------------------
Rem Part 2: TiCoBasic program
Rem store measurement values
#Include GoldIITiCo.inc

#Define val_per_blk 128    'values per medium data block
#Define blk_group 40       'number of blocks to write
#Define blk_total PAR_12   'number of medium data blocks
Dim values[5120] As Long    '5120=blk_group*val_per_blk
Dim idx, blk_no As Long

Init:
  Do                        'wait for flag: media card initialized
  Until (blk_total > 0)
  idx = 0                   'index of measurement values
  blk_no = 1                'first block to write

Event:
  Inc idx                   'get next value
  values[idx] = ADC(1)
  If (idx = val_per_blk*blk_group) Then
    Rem write 40 x 128 values
   PAR_1 = Media_Write(blk_no, blk_group, values, 1)
    If (PAR_1 > 0) Then End
    Rem adjust counters
    idx = 0
    blk_no = blk_no + blk_group
    If (blk_no+blk_group > blk_total) Then End
  EndIf
```

Anhang

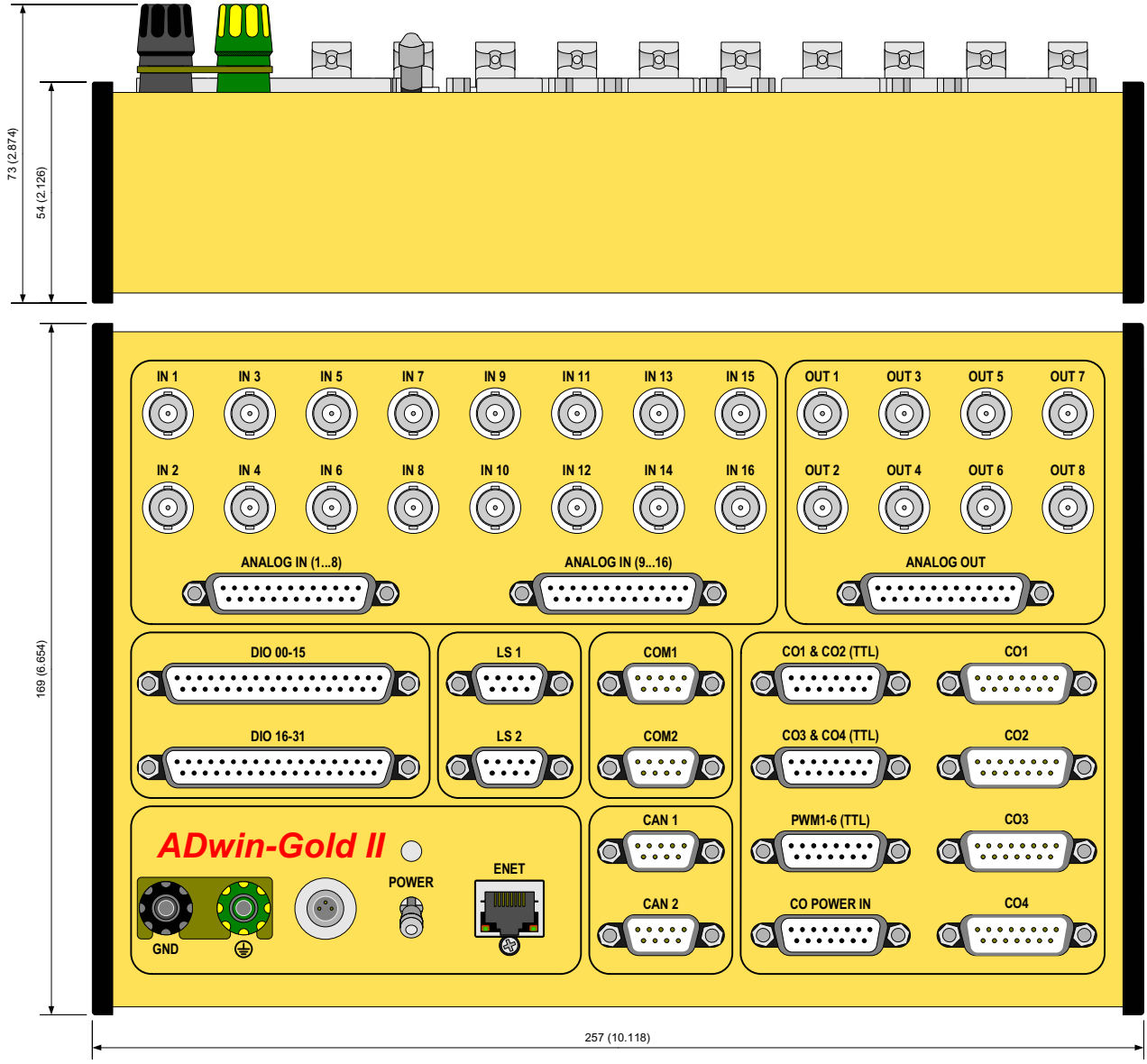
A.1 Technische Daten

Sämtliche technischen Daten beziehen sich auf ein eingeschaltetes *ADwin-Gold II*.

Allgemeine Daten / Grenzwerte						
	Symbol	Konditionen	min.	typ.	max.	Einheit
Versorgungs-Spannnung						
Spannung	U_b		10	12	28	V
Ruhestrom	I_{idle}	U_b=10V		1,3		A
		U_b=12V		0,9		
		U_b=24V		0,5		
Einschaltstrombedarf	$I_{power-on}$	U_b=12V	3			
Zulässiger Betriebsbereich						
Temperatur	$T_{Gehäuse}$		+5		+60	°C
rel. Feuchte	F_{rel}	nicht kondensierend	0		80	%
Lagerung						
Temperatur	T		-20		+70	°C
Feuchte	R_H	nicht kondensierend, keine aggressive Atmosphäre				
Steckverbinder						
Sub-D-Verbinder	Metrisches ISO-Gewinde; UNC-Gewinde als Bestelloption erhältlich.					
BNC-Steckverbinder	Standard, 50Ω					
Abmessungen						
Breite x Höhe x Tiefe	B x H x T	Gold II	257 × 73 × 169			mm
Nettogewicht						
Gewicht	m_{Netto}	Gold II	2050			g
		Clipse[a]	32			

a Zubehör zur Hutschienenmontage: *Gold II-Mount*

(all dimensions are in millimeters [mm], values in parentheses are in inches, 25.4mm = 1")



Digitale Ein- / Ausgänge						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
I/O-Leitungen						
Anzahl	DIO00:DIO31	32 (in Gruppen zu 8 als Ein-oder Ausgang programmierbar)				
	EVENT	ext. Trigger-Eingang (TTL-Logik, Polarität einstellbar)				
Eingänge						
max. Eingangsspanng.		V_{CC} = 5V	-0,5		+5,5	V
Logik-Eingangsspannung	V_{IH} (High)	V_{CC} = 5V	2,4			
	V_{IL} (Low)	V_{CC} = 5V			0,8	
Logik-Eingangsstrom	I_{IH}	V_I = 5V			2,4	mA
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}	alle DIGIN bzw. alle DIGOUT über V_{CC} / GND			±70	
EVENT-Eingang						
Flankenerkennung, pos.	V_{T+} (Low)	V_{CC} = 5V	1,65	1,9	2,15	V
Schalthysterese	V_{T+} - V_{T-}		0,4	0,9		
Eingangsstrom	I_{IH}	V_I = 2,7V			1,1	mA
	I_{IL}	V_I = 0,4V			1,1	

Analoge Ein- / Ausgänge						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Eingänge						
Anzahl	2 × 8 über Multiplexer, differentiell					
Eingangswiderstand	R_i		323,4	330	336,6	kΩ
Spannungsfestigkeit	U_{in} max.	ON & OFF			±35	V
Multiplexer-Einschwingzeit	t_{MUX}	1 LSB 18Bit		2,0		µs
ADC 18 Bit						
Konvertierungszeit	t_{conv}				2	µs
Messbereich	U_{in}	F_V=1	-10		+9,999695	V
		F_V=2	-5		+4,999847	
		F_V=4	-2,5		+2,499924	
		F_V=8	-1,25		+1,249962	
Diff. Gleichtaktspanng.					±2,5	
Integrale Nichtlinearität	INL			±1	±3	LSB
Different. Nichtlinearität	DNL			±0,25	±0,5	

Analoge Ein- / Ausgänge						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Offset	Drift[a]			±2		ppm/K
	Fehler	abgleichbar ±1%				
Gain	Drift[a]			±5		ppm/K
	Fehler	abgleichbar ±1%				
Ausgänge: DAC 16 Bit						
Anzahl	2 (mit DA-Erweiterung: 4 oder 8)					
Ausgangsspannung	U_{out}		-10		+9,999695	V
Einschwingzeit	t_{settle}	2V-Sprung		2		µs
		FSR[b] (20V)		4		
Zulässiger Strom					10	mA
Integrale Nichtlinearität	INL				±2	LSB
Different. Nichtlinearität	DNL				±1	
Offset	Drift[a]			±1		ppm/K
	Fehler	abgleichbar				
Gain	Drift[a]			±3		ppm/K
	Fehler	abgleichbar				

a bezogen auf den Gesamt-Spannungsbereich (FSR)

b Full Scale Range

Prozessor						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Typ	ADSP TS610S (TIGER-SHARC™)					
Hersteller	Analog Devices					
Taktfrequenz	f_{CLK}			300		MHz
Register-Breite				32		Bit
Interner Speicher	SRAM	für Programm		256		kByte
		für Daten		256		
Externer Speicher	SDRAM			256		MByte

CNT-Erweiterung						
Parameter	Symbol	Konditionen	min.	typ.	max.	Einheit
Zähler						
Anzahl	4 Zähler (CNTR1 … CNTR4)					
Eingänge	Je Zähler 3 differentielle Eingänge (A/CLK, B/DIR, CLR/LATCH); Zähler paarweise programmierbar mit differentiellen oder single-ended Eingängen.					
Zählerbreite				32		Bit
Zählfrequenz	f_{CLK}	Eingang CLK		20		MHz
		Eingang A/B		5		
Latch-Breite	LATCH			32		Bit
Referenz-Quarzoszillator						
Referenzfrequenz	f_{ref}			100		MHz
Genauigkeit und Drift					±20	ppm
Zählereingänge differentiell[a]						
Diff. Eingangs-Schwellenspannung	V_{TH}	$-10V \le V_{CM} \le 13,2V$	-200		+200	mV
Schalthysterese	ΔV_{TH}	$-10V \le V_{CM} \le 13,2V$		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		±5		V
Zählereingänge single ended[b] (mit Schmitt-Trigger)						
Flankenerkennung, pos.	V_{T+} (Low)	V_{CC} = 5V	1,65	1,9	2,15	V
Flankenerkennung, neg.	V_{T-} (Low)		0,75	1,0	1,25	
Schalthysterese	$V_{T+} - V_{T-}$		0,4	0,9		
Eingangsstrom	I_H	VI = 2,7V			0,5	mA
	I_L	VI = 0,4V			0,04	

a siehe auch Datenblatt MAX3098 von MAXIM

b siehe auch Datenblatt 74LS19 von Texas Instruments

A.2 Hardware-Adressen

Für *ADbasic* sind keine Hardware-Adressen dokumentiert, weil deren Anwendung im Vergleich zu den vorhandenen *ADbasic*-Befehlen keinen Vorteil bietet.

Die folgenden Hardware-Adressen sind gültig für den *TiCo*-Prozessor. Die Adressen der ersten Spalte sind geeignet, um damit einen extern gesteuerten Prozess anzusteuern. Die Adressen der zweiten Spalte werden im extern gesteuerten Prozess benötigt. Beispiele hierzu finden Sie unter `C:\ADwin\TiCoBasic\samples_ADwin_GoldII`.

Adresse *TiCoBasic* [HEX] Prozess triggern		Funktion	Bit Nr. 31:24	23:17	16	15:9	8	7	6	5	4	3	2	1	0	Kommentar
Digitale Ein- und Ausgänge																
000000D0	–	DIO00:DIO31 lesen	x	x	x	x	x	x	x	x	x	x	x	x	x	x: gelesener Digitalwert
000000E4	000000E0	Flankenüberwachung: steigende Flanke[a]	x	x	x	x	x	x	x	x	x	x	x	x	x	x=1: Flanke ist aufgetreten x=0: Flanke ist nicht aufgetreten
000000E6	000000E2	Flankenüberwachung: fallende Flanke[a]	x	x	x	x	x	x	x	x	x	x	x	x	x	
000000E8	–	DIO: FIFO der Flankenüberwachung	-	-	y	-	x									x: Anzahl Werte (0…511) im FIFO der Flankenüberwachung y=1: FIFO full
000000FE	000000FC	Event-Register[a]	-	-	-	-	-	-	-	-	-	c	b	a	x	Bit=1: Event-Signal ist aufgetreten. x: externer Event-Eingang b: Event Anybus c: Event CAN2 d: Event CAN1

a Die erste Adresse verändert das Register beim Lesen nicht, die zweite Adresse setzt das Register auf Null zurück.

A.3 Hardware-Revisionen

Auf der Rückseite des Gold II-Systems befindet sich ein Aufkleber mit der Revisionsbezeichnung des Geräts. Die Unterschiede der Revisionsstände sind nachfolgend dargestellt:

Revision	Erst-ausgabe	Änderung zur Vorgänger-Version
A1	Nov. 2007	Erstversion.
A2	Jun. 2008	Freigabe des *TiCo*-Prozessors. Option *Gold II-Storage-16* verfügbar, siehe Seite 57
A10	Okt. 2014	Einführung Schnittstelle ENET-3 (Ethernet-Kommunikation zum PC)

A.4 RoHS Konformitätserklärung

Die RoHS-Richtlinie 2011/65/EU der Europäischen Union zur Beschränkung und Verwendung gefährlicher Stoffe in elektrischen und elektronischen Geräten ist am 3. Januar 2013 in Kraft getreten.

Die Richtlinie betrifft folgende Substanzen:

– Blei (Pb)
– Cadmium (Cd)
– Hexavalentes Chrom (Cr VI)
– Polybromierte Biphenyle (PBB)
– Polybromierte Diphenylether (PBDE)
– Quecksilber (Hg)
– Bis(2-ethylhexyl)phthalat (DEHP)
– Benzylbutylphthalat (BBP)
– Dibutylphthalat (DBP)
– Diisobutylphthalat (DIBP)

Die Produktlinie *ADwin-Gold II* erfüllt die Voraussetzungen der RoHS-Richtlinie in allen gelieferten Varianten.

A.5 Baudraten für den CAN-Bus

ADwin-Gold II-CAN 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

Einstellbare Baudraten [Bit/s]				
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
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

Einstellbare Baudraten [Bit/s]				
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
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 Abbildungsverzeichnis

A.7 Index

W

Z