

ADwin-X-A20

Handbuch



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
Rheinstraße 2-4
D-64653 Lorsch

Inhaltsverzeichnis

| | |
|--|-----|
| Inhaltsverzeichnis | III |
| 1 Typografische Konventionen | V |
| 1 Zu diesem Handbuch | 1 |
| 2 Systembeschreibung | 2 |
| 2.1 ADwin Systemkonzept | 2 |
| 2.2 ADwin-X-A20 | 3 |
| 3 Betriebliche Umgebung | 7 |
| 4 Inbetriebnahme der Hardware | 8 |
| 5 Übersicht Ein- und Ausgänge | 9 |
| 6 X-A20 Basis mit den Varianten -M1 und -F | 11 |
| 6.1 Mehrfarbige LED | 11 |
| 6.2 Analogeingänge 18 Bit | 11 |
| 6.3 Analogausgänge 12 Bit | 14 |
| 6.4 Analogausgänge 16 Bit | 15 |
| 6.5 TTL-Digitalkanäle DIO39:DIO32 | 15 |
| 6.6 Event-Eingang | 16 |
| 6.7 LS-Bus | 16 |
| 6.8 Synchrone Aktionen | 16 |
| 7 Option CO1 | 18 |
| 8 Option D | 19 |
| 8.1 Diff. Digitalkanäle DIO47:DIO40 | 19 |
| 8.2 Diff. Zähler 4, 5 | 19 |
| 8.3 SSI-Schnittstelle | 20 |
| 9 Option DCT | 22 |
| 9.1 TTL-Digitalkanäle DIO31:DIO00 | 22 |
| 9.2 Komparatoreingänge DIO59:DIO48 | 22 |
| 9.3 Flankenüberwachung und Flankenausgabe | 23 |
| 9.4 TTL-Zähler 2, 3 | 24 |
| 9.5 Komparatorzähler 6, 7 | 24 |
| 10 Option COM | 25 |
| 10.1 CAN-Schnittstellen | 25 |
| 10.2 RS232-Schnittstelle | 27 |

| | |
|--|-----|
| 11 Option Profibus | 28 |
| 12 Option Profinet-IRT | 31 |
| 13 Option EtherCAT | 35 |
| 14 Option Boot | 38 |
| 15 Zählerblock | 39 |
| 15.1 Auswerten des Zählerinhalts | 41 |
| 15.2 Ereigniszähler einsetzen | 42 |
| 15.3 PWM-Zähler einsetzen | 44 |
| 16 Software | 46 |
| 16.1 Allgemeine Befehle | 47 |
| 16.2 Analoge Ein- und Ausgänge | 54 |
| 16.3 Digitale Ein- und Ausgänge | 78 |
| 16.4 Zähler | 114 |
| 16.5 SSI-Schnittstelle | 131 |
| 16.6 CAN-Schnittstelle | 139 |
| 16.7 RSxxx-Schnittstelle | 149 |
| 16.8 Profibus-Schnittstelle | 155 |
| 16.9 Profinet-Schnittstelle | 159 |
| 16.10 EtherCAT-Schnittstelle | 163 |
| 16.11 LS-Bus + ADwin-X-A20 | 167 |
| Anhang | A-1 |
| A.1 Technische Daten | A-1 |
| A.2 Hardware-Revisionen | A-6 |
| A.3 RoHS Konformitätserklärung | A-6 |

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-X-A20-Systems. Es wird ergänzt durch

- das Handbuch „ADwin Installation“, das die Installation von Software und Hardware beschreibt.
Beginnen Sie hier die Installation Ihres Systems!
- die Beschreibung des Konfigurationsprogramms *ADconfig*, mit dem Sie die Kommunikation von der jeweiligen Schnittstelle zu Ihrem ADwin-X-A20-Gerät einrichten.
- das Handbuch *ADbasic*, das alle Befehle für den gleichnamigen Compiler enthält sowie das Funktionsprinzip von ADwin-Systemen erläutert.
Die Online-Hilfe von *ADbasic* enthält die gleichen Informationen.
- die Installations- und Befehlsbeschreibungen für die Treiber der gängigen Entwicklungsumgebungen.
- das Handbuch „ADwin HSM-24V“, das ein Modul am LS-Bus beschreibt.

Beachten Sie: Das Handbuch ist noch in Bearbeitung, es können Fehler enthalten sein.

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.

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).

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.

Diese Dokumentation ist einschließlich aller Abbildungen urheberrechtlich geschützt. Reproduktion, Übersetzung sowie elektronische und fotografische Archivierung und Veränderung bedürfen der schriftlichen Genehmigung der Firma *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch.

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

Hotline-Adresse siehe vordere Umschlagseite, innen.



Einschränkung der Anwendergruppe

Verfügbarkeit der Unterlagen



Rechtliche Grundlagen

Änderungen vorbehalten.

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.

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, DIAdem, 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.

Software Schnittstellen

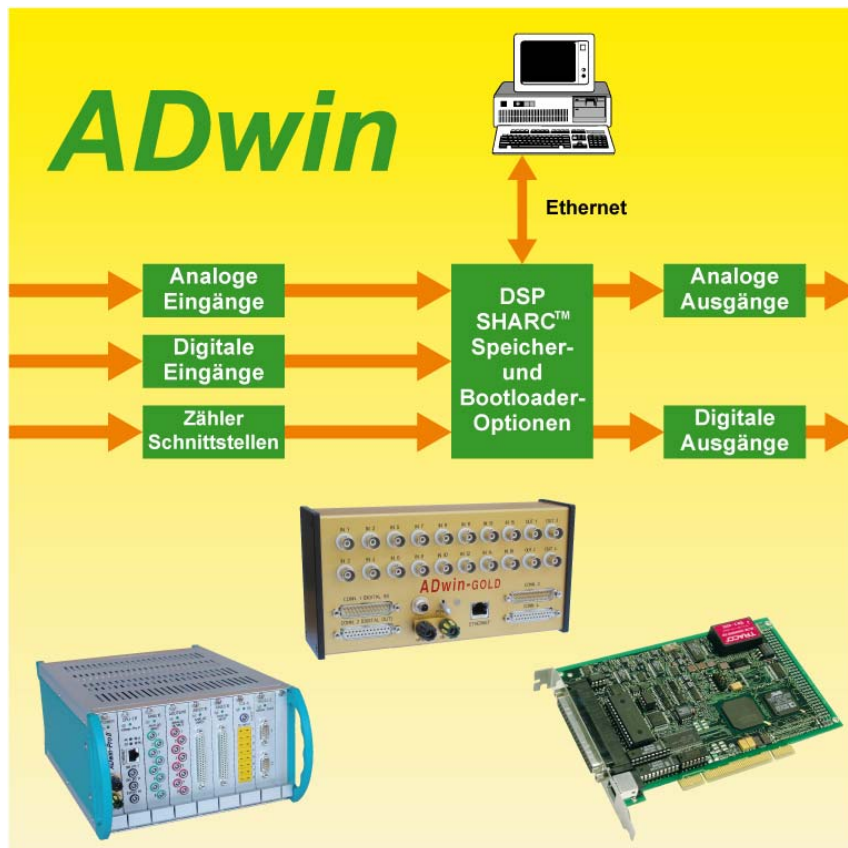


Abb. 1 – Konzept der ADwin-Systeme

2.2 ADwin-X-A20

In ADwin-X-A20 arbeitet der digitale **Signalprozessor** XILINX ZYNQ™ mit Dual-Core ARM Cortex-A9 (666MHz), der 64 Bit-Float und 32 Bit-Integer verarbeitet. Der Prozessor übernimmt die gesamte Messwerterfassung, Online-Verarbeitung und Signalausgabe und kann in Verbindung mit dem A/D-Wandler jeden Messwert mit Abtastraten im Bereich von 200kHz bis 800kHz sofort verarbeiten.

Der **Speicher mit 1 GiB** ist für alle Aufgaben und auch größere Datenmengen ausreichend. Ein integrierter Cache-Speicher erlaubt eine sehr kurze Zugriffszeit und nimmt das ADwin-Betriebssystem, die ADbasic-Prozesse und alle Variablen auf.

Für maximale Zugriffsgeschwindigkeit liegen alle Ein- und Ausgänge direkt im Adressbereich des DSP.

Die Anzahl und Funktion der Ein- und Ausgänge unterscheiden sich je nach gewählter Variante von X-A20. Nachfolgend sind alle verfügbaren Funktionen beschrieben.

Prozessor und Speicher

Analoge Eingänge

In einer Sub-D-Buchse sind **8 analoge Eingänge** (differentiell) bereit gestellt. Das Eingangssignal wird mit einem 18 Bit Analog-Digitalwandler (ADC) konvertiert (siehe Abbildung unten). Je nach Version kann ADwin-X pro Wandlungssequenz den Digitalwert für einen Kanal wandeln (X-A20-M1) oder für alle acht Kanäle synchron (X-A20-F).

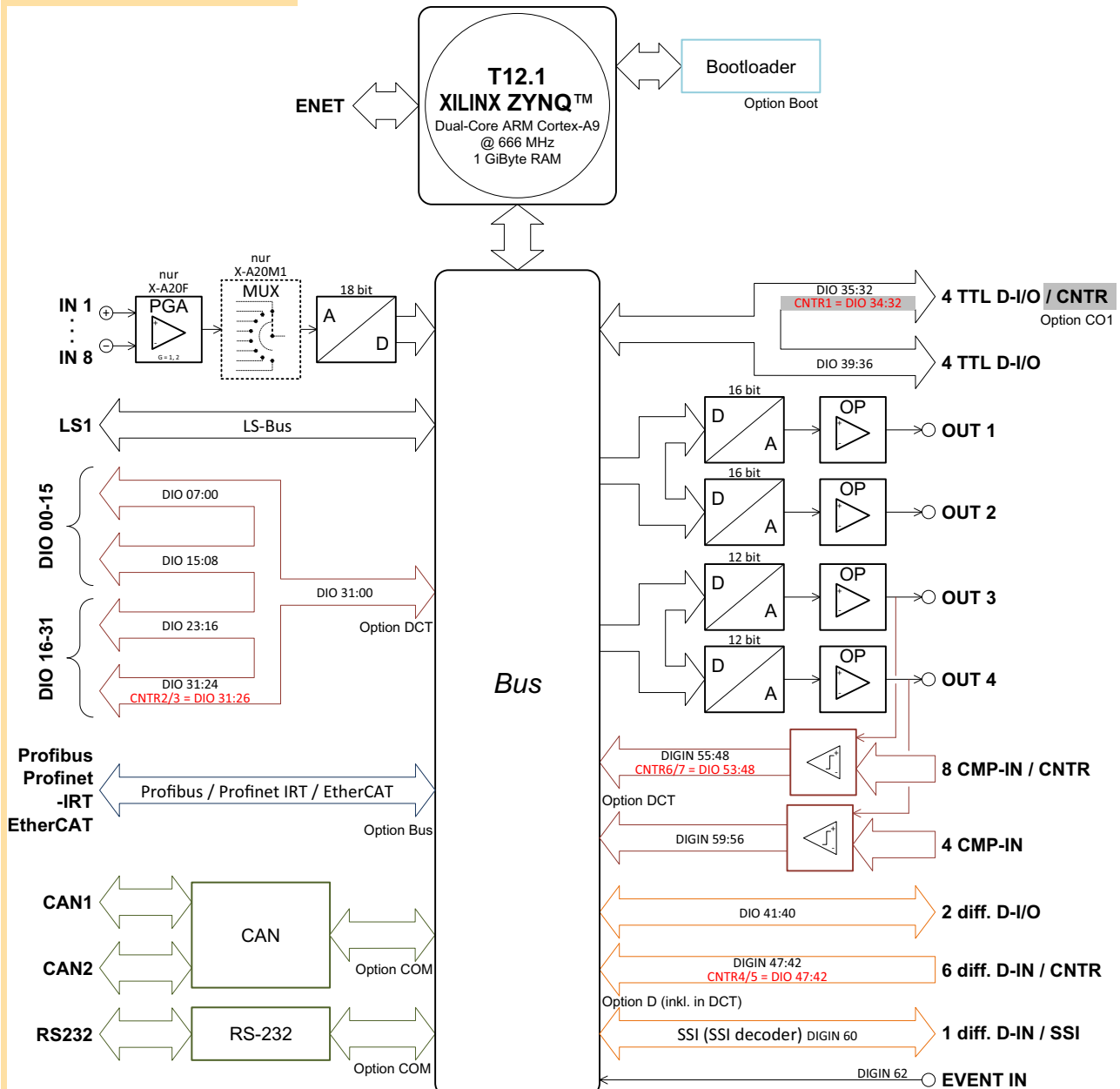


Abb. 2 – Funktionsschema

Analoge Ausgänge

ADwin-X-A20 verfügt über **2 Analogausgänge mit 16 Bit** Auflösung, Spannungsbereich -10 V...+10 V. Per Software können Sie die Ausgabe der Spannung aller DAC synchronisieren. Das Ausgangssignal durchläuft zur Glättung einen Tiefpassfilter mit einer Eckfrequenz von $f_g = 700$ kHz.

Es gibt außerdem **2 Analogausgänge mit 12 Bit** Auflösung, Wandlungsrate 1 ms. Die Ausgänge werden intern für Komparator-Eingänge und -Zähler verwendet.

Digitale Ein- und Ausgänge

Auf 37-poligen Sub-D-Buchsen stehen zwischen 8 und 61 **digitale Eingänge und Ausgänge** zur Verfügung, je nach Version von ADwin-X. Die digitalen Ein- und Ausgänge haben in der Regel TTL-ähnliche Signale, aber es gibt auch 8 differentielle Kanäle und 12 Komparator-Eingänge. Die meisten Kanäle können gruppenweise als Ein- oder Ausgänge konfiguriert werden, einige sind jedoch bereits festgelegt. Zum Teil sind digitale Kanäle doppelt belegt und werden alternativ als Zählereingänge verwendet.

Über einen FIFO können die Flanken der Digitaleingänge überwacht werden. Bei Digitalausgängen können Pegel zu bestimmten Zeitpunkten ausgegeben werden.

Es sind Eingänge für insgesamt **7 Impuls-/Ereigniszählerblöcke** mit 32 Bit vorhanden; ein Zählerblock hat einen Vor-/ Rückwärtszähler sowie einen PWM-Zähler. Die Zählerblöcke sind funktionsgleich, unterscheiden sich aber bei den verwertbaren Eingangssignalen: TTL-ähnliche Signale, differentielle Signale, Komparatorsignale.

ADwin-X-A20 besitzt einen Trigger-Eingang (EVENT, siehe auch [Kapitel 6.6 "Event-Eingang"](#)). Hiermit können Prozesse durch ein Signal (Trigger) ausgelöst und sofort vollständig abgearbeitet werden (siehe *ADbasic*-Handbuch oder Online-Hilfe: Kapitel „Prozesse im Betriebssystem“).

Über eine serielle Schnittstelle (LS-Bus, siehe [Seite 16](#)) können bis zu 15 LS-Bus-Module angesteuert werden. Das LS-Bus-Modul HSM-24V ermöglicht den Anschluss von 24V-Signalen auf 32 digitalen Kanälen.

Es gibt Schnittstellen für CAN (High Speed), RS232, SSI, Profibus, Profinet-IRT und EtherCAT. Weitere Schnittstellen sind auf Anfrage möglich.

Ein Bootloader erlaubt, eine zuvor programmierte Anwendung automatisch nach dem Einschalten zu starten. Damit ist nach dem Einrichten der Anwendung ein Betrieb ohne PC möglich.

Der Standardlieferungsumfang eines ADwin-X-A20 beinhaltet

- Hardware ADwin-X-A20 ([Bauform](#) and [Funktionen](#) wie bestellt).
- Ethernet-Anschlusskabel, Länge 1,8 Meter
- 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 Bauform

ADwin-X-A20 ist in folgenden Bauformen verfügbar:

- Standard: Metallgehäuse als Tischgerät.
- A20-R: Metallgehäuse zum Einbau in 19“-Rahmen. Sämtliche +Bus-Optionen (PROFIBUS, PROFINET, ECAT) stehen nicht zur Verfügung.

Zähler

Trigger-Eingang

24 Volt-Signale

Schnittstellen

Bootloader

Standardlieferungsumfang

2.2.2 Funktionen

ADwin-X-A20 gibt es in der Basisversion X-A20-M1 oder X-A20-F. Die Basisversion ist beliebig kombinierbar mit zahlreichen Optionen.

| Optionen | Funktionen | Seite |
|--------------------|--|--|
| X-A20-Basis | 8 Analogeingänge, 18 Bit 2 Analogausgänge, 16 Bit 2 Analogausgänge, 12 Bit 8 TTL-Digitalkanäle 1 Event-Eingang 1 LS-Bus-Schnittstelle | Seite 11 Seite 11 Seite 16 |
| M1 | Analogeingang mit Multiplexer: 1 Messwert pro Wandlung, 200 kHz. | |
| F | Analogeingang Fast: bis zu 8 Messwerte synchron wandeln, 200 kHz...800 kHz; einstellbare Verstärkung. | |
| +CO1 | 1 TTL-Zählerblock: 32 Bit Vor-/Rückwärtszähler und PWM-Zähler | Seite 18 |
| +DCT (inklusive D) | 32 TTL-Digitalkanäle 12 Komparatoreingänge 2 TTL-Zählerblöcke: 32 Bit Vor-/Rückwärtszähler und PWM-Zähler 2 Komparatorzählerblöcke: 32 Bit Vor-/Rückwärtszähler und PWM-Zähler Eingangs-FIFO und Ausgangs-FIFO für digitale Kanäle | Seite 22 |
| +D | 6 diff. Digitaleingänge + 2 diff. Digitalkanäle 2 diff. Zählerblöcke: 32 Bit Vor-/Rückwärtszähler und PWM-Zähler 1 SSI-Schnittstelle | Seite 19 |
| +COM | 2 CAN-Schnittstellen (high speed) 1 RS232-Schnittstelle | Seite 25 |
| +Bus +PROFIBUS | 1 Profibus-Schnittstelle (Slave) | Seite 28 |
| +PROFINET | 1 Profinet-IRT-Schnittstelle (Slave), zwei Steckverbinder-Varianten | Seite 31 |
| +ECAT | 1 EtherCAT-Schnittstelle (Slave) | Seite 35 |
| +Boot | Flash-EPROM-Bootloader zum eigenständigen Betrieb ohne PC | Seite 38 |

Option DCT ist eine Erweiterung der Option D, die beiden Optionen sind also nicht kombinierbar.

Die Optionen PROFIBUS, PROFINET und ECAT sind nicht miteinander kombinierbar.

2.2.3 Zubehör

Für ADwin-X-A20-System ist das folgende Zubehör (auch nachträglich) erhältlich:

- *ADbasic*, Echtzeit-Entwicklungstool für alle ADwin-Systeme
- *A20-Mount*: Gehäuse zur Hutschienenmontage in einem Schaltschrank mit isolierten Clipsen.
- *A20-Pow*: externes Netzteil.
- *A20-Pow-Mount*: externes Netzteil für DIN-Hutschienen.
- *HSM-24V*: Hutschienenmodul für LS-Bus-Schnittstelle, 32 Digital-I/Os, 24V-Pegel, in 8er Gruppen konfigurierbar, Schraubklemmanschluss.

3 Betriebliche Umgebung

Je nach Variante und Ausrüstung kann das Gerät in 19"-Systemgehäusen, in Schaltschränken oder im mobilen Betrieb (z.B. im Kfz) eingesetzt werden.

Das ADwin-X-A20-Gerät **muss geerdet werden**, um

- einen Massebezugspunkt für die Elektronik herzustellen und
- Störungsenergie auf die Erde ableiten zu können.

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

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

Ausgleichsströme, die über das Gehäuse oder die Schirmung abfließen, beeinflussen das Messsignal.

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

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

Das ADwin-X-A20-System wird intern mit einer Spannung von +5 V und ±15 V gegen GND betrieben und stellt von dieser Seite keine Gefahr für Leib und Leben dar. Für den Betrieb mit einem externem Netzteil gelten die Angaben des Herstellers.

ADwin-X-A20 ist für den Betrieb in trockenen Räumen konzipiert. Am Einbauort (im PC oder 19"-Gehäuse) sollen eine Umgebungstemperatur von +5 °C ... +50 °C und eine relative Luftfeuchte von 0 ... 80 % (nicht kondensierend, s.a. Anhang) vorhanden sein.

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



Galvanische Kopplung

Ausgleichströme ausschließen



Schutzkleinspannung

Umgebungs-klima

Gehäusetemperatur



4 Inbetriebnahme der Hardware



Schließen Sie bei der Inbetriebnahme keine Kabel an das ADwin-X-A20-System an, bevor Sie nicht folgende Schritte durchgeführt haben:

1. Software-Installation / Einbau im PC oder 19"-Gehäuse:
Folgen Sie der Anleitung im Handbuch: „ADwin-Installation“.
2. Richten Sie die betriebliche Umgebung ein wie in [Kapitel 3](#) beschrieben.
3. Lesen Sie das [Kapitel 5](#) „Übersicht Ein- und Ausgänge“ in diesem Handbuch.
4. Schließen Sie erst jetzt Signalleitungen an die Ein- und Ausgänge an.

Hinweise

Achten Sie auf eine zuverlässige Spannungsversorgung.

Im Standardlieferungsumfang betrifft das den PC, ansonsten auch das externe Netzteil, bei Betrieb im Fahrzeug die Batteriespannung.

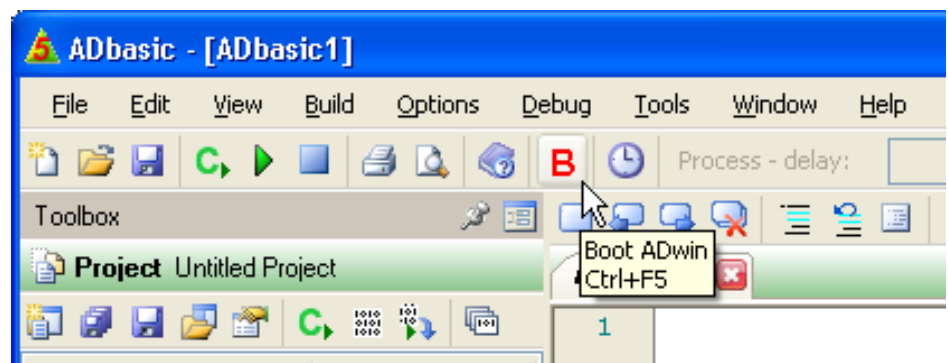
Achten Sie bei der Verwendung strombegrenzender Netzteile darauf, dass beim Einschalten der Strombedarf ein Mehrfaches des Betriebsstroms betragen kann. Genaue Angaben finden Sie bei den Technischen Daten (Anhang).

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

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

Datenverbindung testen

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



Der Boot-Vorgang dauert etwa vier Sekunden, in seltenen Einzelfällen auch bis zu fünfzehn Sekunden.

Die grün blinkende LED1 am Gerät und die Anzeige in der Statuszeile: „ADwin is booted“ zeigt an, dass das Betriebssystem richtig geladen ist und *ADbasic* eine Verbindung zum ADwin-System herstellen kann.

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

Beginnen Sie mit Programmbeispielen im *ADbasic*-Tutorial.

Sicherstellen der Spannungsversorgung



Programme mit ADbasic



5 Übersicht Ein- und Ausgänge

ADwin-X-A20 besitzt die folgenden Steckverbindungen (Pinbelegung siehe nächste Seite). Je nach Bauvariante ist nur ein Teil der Steckverbindungen aktiv belegt.

- Anschlussbuchse für Ethernet
- Strom-Eingangsstecker
- GND-Buchse
- 3 Sub-D-Buchsen 37-polig, Conn. 1, Conn. 2, Conn. 3
 - analoge Eingänge, analoge Ausgänge
 - digitale Ein- und Ausgänge: TTL, differentiell, Komparator
 - Eingänge für Impuls-/Ereigniszähler: TTL, differentiell, Komparator
 - SSI-Schnittstelle
 - digitaler Trigger-Eingang (Event)
 - Stromversorgungs-Ausgang +5V

Manche Pins sind doppelt belegt.

- 2 Sub-D-Stecker 9-polig, CAN1, CAN2
- 1 Sub-D-Stecker 9-polig, RS232
- 1 Sub-D-Buchse 9-polig, LS-BUS

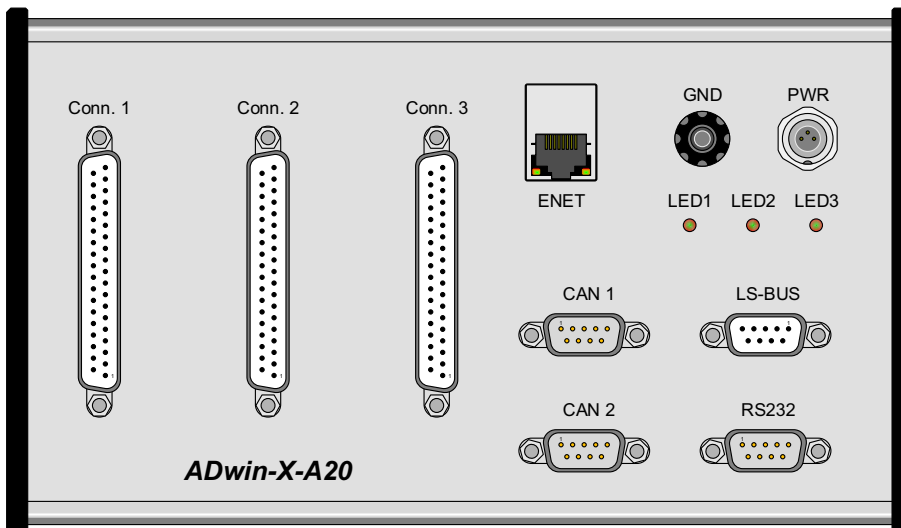


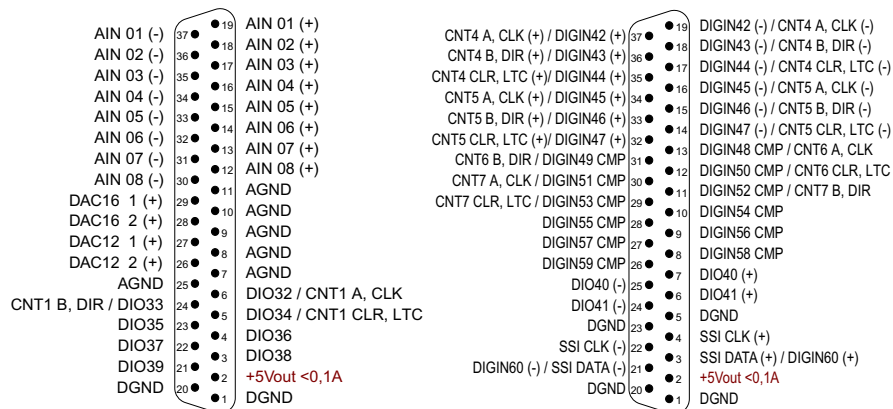
Abb. 3 – Steckverbindungen ADwin-X-A20

Alle Ein- und Ausgänge dürfen nur im Bereich der angegebenen Spezifikation betrieben werden (siehe Anhang A.1 Technische Daten). Im Zweifel wenden Sie sich bitte an den Hersteller des Gerätes, das Sie an das ADwin-X-A20-System anschließen wollen.

Offene Eingänge können zu Fehlern führen – vor allem in einer nicht störungsfreien Umgebung. Zu Ihrer Sicherheit legen Sie nicht benutzte Eingänge möglichst nah an der Sub-D-Buchse auf einen definierten Pegel (z.B. GND). Trennen Sie diese Eingänge auch von offenen Leitungen.

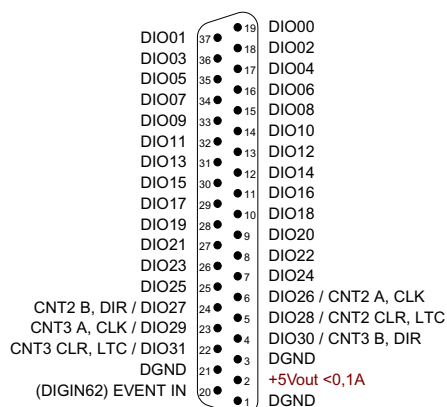
Ausnahme hierzu ist der Event-Eingang, der bereits einen internen Pull-down-Widerstand (4,7 kΩ) besitzt.





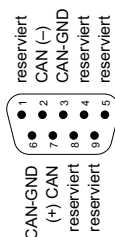
Conn. 1

Conn. 2

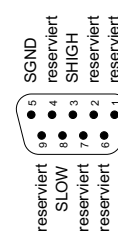


Conn. 3

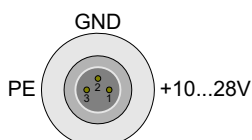
Abb. 4 – Pin-Belegung Ein-/Ausgänge (Buchse)



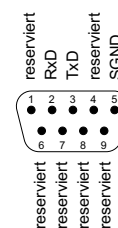
Pin-Belegung CAN1, CAN2



Pin-Belegung LS-BUS (Buchse)



Stromeingang (Stecker)



Pin-Belegung RS232

Abb. 5 – Pin-Belegung Schnittstellen, Stromeingang

6 X-A20 Basis mit den Varianten -M1 und -F

Zur Basisversion X-A20 gehören:

- 3 Mehrfarbige LED
- 8 Analogeingänge 18 Bit
 - X-A20-M1: Multiplexer
 - X-A20-F: Synchronwandler
- 2 Analogausgänge 12 Bit
- 2 Analogausgänge 16 Bit
- 8 TTL-Digitalkanäle DIO39:DIO32
- 1 Event-Eingang
- 1 LS-Bus
- Synchrone Aktionen

6.1 Mehrfarbige LED

X-A20 besitzt 3 mehrfarbige LED, die Sie ein- und ausschalten können.

LED 1 dient nach dem Einschalten als Status-LED und leuchtet rot; sobald der Boot-Prozess beendet ist (etwa vier Sekunden, selten bis zu fünfzehn Sekunden), läuft Prozess 15 und lässt LED 1 grün blinken.

Befehle zur Programmierung der LED ab [Seite 48](#) beschrieben. Die Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle |
|---|------------------------|
| LED-Status prüfen | <code>Check_LED</code> |
| LED ein- oder ausschalten, Farbe setzen | <code>Set_LED</code> |

6.2 Analogeingänge 18 Bit

X-A20 hat 8 differentielle analoge Messeingänge, die über 18 Bit Analog-Digitalwandler (ADC) geführt werden. Es gibt zwei Varianten:

- X-A20-M1: Multiplexer

Die 8 Messeingänge werden über einen Multiplexer auf den ADC geführt. Die Wandlungszeit (inklusive Einschwingzeit des Multiplexers) beträgt 5 µs. Der Verstärkungsfaktor ist auf 1 festgelegt.

- X-A20-F: Synchronwandler

Bis zu 8 Messeingänge werden synchron gewandelt. Die Verstärkung (PGA) ist programmierbar auf 1 oder 2. Die Wandlungszeit (für alle Kanäle zusammen) ist abhängig von der Anzahl der gemessenen Kanäle:

- 1 Kanal: max. 800kHz = 1,25 µs.
- 2 Kanäle: max. 550kHz = 1,82 µs.
- 3 Kanäle: max. 425kHz = 2,35 µs.
- 4 Kanäle: max. 350kHz = 2,86 µs.
- 5 Kanäle: max. 300kHz = 3,3 µs.
- 6 Kanäle: max. 250kHz = 4,0 µs.
- 7 Kanäle: max. 225kHz = 4,44 µs.
- 8 Kanäle: max. 200kHz = 5 µs.

Wenn Sie mit einem Wandlungsbefehl (`ADC...` / `Start_Conv`) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Der Eingangs-Spannungsbereich ist ±10V (bei Verstärkungsfaktor 1).

Die analogen Eingänge sind differentiell. Für jeden Messeingang sind jeweils ein Plus- und ein Minuseingang vorhanden, zwischen denen die Spannungsdifferenz gemessen wird (jedoch nicht potentialfrei), siehe [Abb. 4 – Pin-Belegung Ein-/Ausgänge \(Buchse\)](#).

Beachten Sie, dass zusätzlich zu Plus- und Minuseingang des Kanals immer eine Masseverbindung zwischen X-A20 (GND) und der Signalquelle bestehen muss.

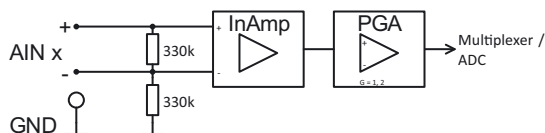


Abb. 6 – Eingangsbeschaltung eines analogen Eingangs

Signale werden mit dem 18 Bit Analog-Digital-Wandler (ADC) schnell und sehr genau ($76\mu\text{V}$) konvertiert. Messwerte können mit 16 Bit oder 24 Bit Auflösung gelesen werden.

Bitte beachten Sie die [Berechnungsgrundlagen](#) zur Auswertung der Messwerte.

Bei X-A20-F gibt es zwei Methoden, mehrere Messwerte zu wandeln, die alternativ einsetzbar sind. Bei X-A20-M1 sind die Methoden ebenfalls einsetzbar, aber nur mit einem einzelnen Kanal:

- Einzelwertmessung: Die Wandlung wird auf einem oder mehreren Kanälen zu einem definierten Zeitpunkt gestartet und der Messwert (bzw. die Messwerte) nach der entsprechenden Zeit zurückgegeben. Jede Wandlung wird einzeln gestartet.
- Dauermessung: Eine Ablaufsteuerung wandelt kontinuierlich Messwerte. Man kann den jeweils aktuellsten Wert ohne Wartezeit lesen, kennt aber den genauen Zeitpunkt der Messung nicht. Dadurch kann der Prozessor stark entlastet werden, der die fertig gewandelten Werte nur noch aus dem Zwischenspeicher der Ablaufsteuerung ausliest.

Erdung



X-A20 muss geerdet werden, um Messungen störungsfrei durchführen zu können. Verbinden Sie dazu die GND-Buchse über ein impedanz-armes Masseband mit dem zentralen Erdungspunkt Ihrer Anlage.

Das Gehäuse ist über die GND-Leitung des Stromversorgungskabels als auch über die Schirmung des Ethernet-Steckers mit GND verbunden.

Die Spannungsversorgung vom Power-Adapter am PC verbindet auch die Erdung des X-A20 mit der Erdung des PC. Wenn Sie den PC und das System nicht am selben Ort betreiben, können unterschiedliche Massepotenziale am X-A20 und am Messobjekt bzw. den Messleitungen Störungen verursachen. Vermeiden Sie solche Einflüsse, indem Sie ein externes Netzteil benutzen.

Standard-Befehl

Der Befehl **ADC** () führt mit dem ADC eine komplette Messung auf einem analogen Eingang durch (siehe [Seite 59](#)) und liefert einen 16 Bit-Wert zurück.

Messwerte mit der Auflösung 18 Bit sind mit dem Befehl **ADC24** möglich (siehe [Seite 60](#)); der Wert wird im Format 24 Bit zurückgegeben (siehe [Seite 14](#)).

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



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 10Ω etwa 1 Digit).

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

Programmieren

Befehle zur Programmierung der analogen Eingänge sind – für beide Versionen X-A20-M1 und X-A20-F – ab [Seite 59](#) beschrieben. Die Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle | -M1 | -F |
|--|---|-----|----|
| vollständige Messung durchführen | ADC, ADC24 | x | x |
| Messung auf mehreren Kanälen durchführen | ADC2, ADC4, ADC8 ADC_2_24, ADC4_24 ADC8_24 | - | x |
| Messung in Teilschritten durchführen, Dauermessung starten | Start_Conv, Wait_EOC Read_ADC, Read_ADC24 | x | x |
| Verstärkung einstellen | Start_Conv_PGA | - | x |

| Bereich | Befehle | -M1 | -F |
|--|---|-----|----|
| Mehrere Werte lesen (Messung in Teilschritten) | <code>Read_ADC_Packed</code> <code>Read_ADC8</code> <code>Read_ADC8_24</code> | - | x |
| Mehrere Funktionen synchron starten. | <code>Sync_All</code> | x | x |

6.2.1 Berechnungsgrundlagen

Das ADwin-X-A20-System arbeitet bei den analogen Ein- und Ausgängen mit einem Spannungsbereich von -10 V bis $+10\text{ V}$ oder bipolar 10 V .

Bei einem 16 Bit-Messwert sind die 65536 (2^{16}) Digits dem Spannungsbereich so zugeordnet, dass

- 0 (Null) Digit der maximalen negativen Spannung und
- 65535 Digit der maximalen positiven Spannung entspricht¹.

Der Wert für 65.536 Digit, genau $+10\text{ Volt}$, liegt gerade außerhalb des Messbereichs, womit sich für die 16 Bit-Wandlung ein maximaler Spannungswert von $+9,999695\text{ Volt}$ ergibt.

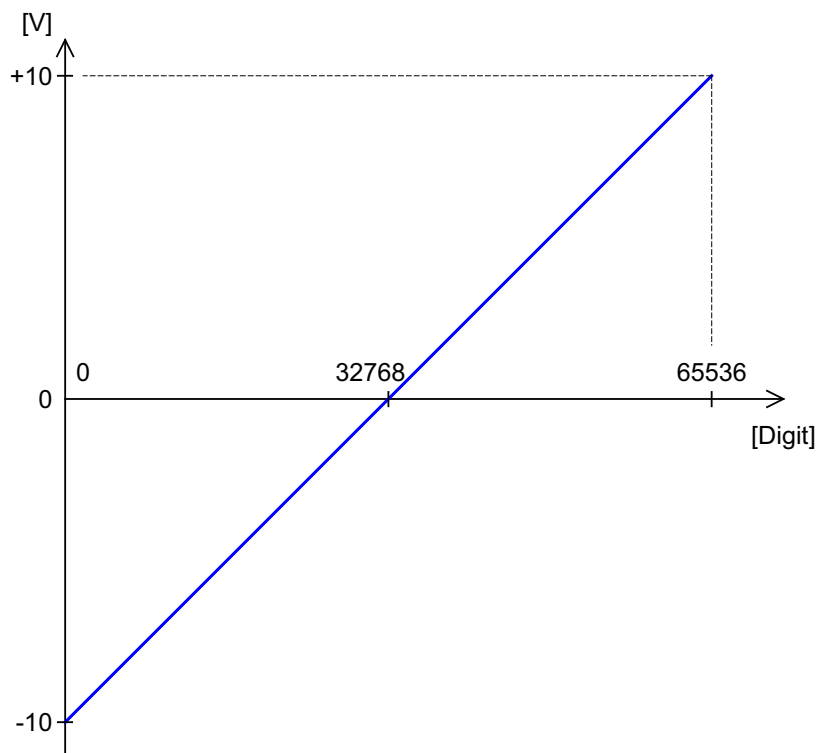


Abb. 7 – Nullpunktverschiebung bei Standardeinstellung bipolar 10 Volt

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

Beim Spannungsbereich $-10\text{ V} \dots +10\text{ V}$ gilt $U_{\text{OFF}} = -10\text{ V}$.

X-A20-F besitzt einen programmierbaren Verstärker (PGA), mit dem Sie die Eingangsspannung um die Faktoren 1 ($\pm 10\text{ V}$) oder 2 ($\pm 5\text{ V}$) verstärken können. Mit der Verstärkung ändert sich der Messbereich.

Beachten Sie bei $k_v = 2$, dass auch die Störsignale entsprechend mit verstärkt werden.

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

Spannungsbereich

Zuordnung von Digit und Spannung



Verstärkungsfaktor k_v

Quantisierungsstufe U_{LSB}

1. Bei einem 24 Bit-Wert sind dem Spannungsbereich 16777216 (2^{24}) Digits zugeordnet.

Der Messwert des 18 Bit-ADC kann im Format 16 Bit oder 24 Bit zurückgegeben werden. Die DAC verarbeiten Werte mit 12 Bit und 16 Bit.

- 24 Bit-Format: In dem 24 Bit-Wert ist der 18 Bit-Messwert in den Bits 23:6 enthalten, d. h. der Messwert wird um 6 Bits nach links verschoben, die Bits 5:0 sind immer Null.

$$U_{\text{LSB } 24\text{Bit}} = 20\text{V} / 2^{24} = 1,19\mu\text{V}$$

- 16 Bit-Format: Der Messwert steht in den unteren 16 Bits des Rückgabewerts, die oberen 16 Bits sind immer Null.

$$U_{\text{LSB } 16\text{Bit}} = 20\text{V} / 2^{16} = 305,175\mu\text{V}$$

Das gleiche gilt entsprechend für einen auszugebenden DAC-Wert.

Auch der Digitalwert für den 12 Bit-DAC wird im 16 Bit-Format angegeben, er steht linksbündig in den unteren 16 Bits, die Bits 3:0 sind immer Null.

| Bit-Nr. | 31:24 | 23:16 | 15:6 | 5:4 | 3:0 |
|---------|-------|-------------------------------|------------------------------|-----|-----|
| Inhalt | 0 | 18-Bit Wert in den Bits 23:6. | | | 0 |
| | 0 | 0 | 16 Bit-Wert in den Bits 15:0 | | |
| | 0 | 0 | 12 Bit-Wert in den Bits 15:4 | | 0 |

Abb. 8 – Ablage der ADC/DAC-Bits im Speicher

Werte im gleichen Format kann man direkt miteinander verrechnen, hier also 12 Bit- und 16 Bit-Werte. Zum Verrechnen von Werten mit verschiedenen Formaten muss der genauere 24 Bit-Wert um 8 Bit nach rechts oder der 16 Bit-Wert nach links geschoben werden.

Umrechnung Digit in Spannung

Für einen DAC gilt:

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

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

Für einen ADC gilt:

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

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

Toleranzbereiche

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

INL

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

DNL

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

6.3 Analogausgänge 12 Bit

ADwin-X-A20 hat 2 analoge Ausgänge mit 12 Bit-Wandlern (DAC12-1, DAC12-2), siehe [Seite 10](#). Jedem Ausgang ist ein eigener Digital-Analog-Wandler (DAC) zugeordnet.

Der DAC hat eine max. Wandlungszeit von 1000µs.

Die 12 Bit-ADC sind intern mit den Komparatoreingängen DIO59:DIO48 verbunden (siehe [Option DCT, Seite 24](#)). Die eingestellte DAC-Spannung dient als Komparatorsignal (0V...5V), d. h. ein anliegendes Digitalsignal mit einer kleineren Spannung wird als Pegel Low verarbeitet, mit einer höheren Spannung als Pegel High.

Bitte beachten Sie die [Berechnungsgrundlagen](#) zur Verarbeitung der DAC-Werte.

Zur Programmierung der analogen Ausgänge gibt es nur einen Befehl, der auf [Seite 56](#) beschrieben ist. Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle |
|---------------------------------------|--------------|
| vollständige Werteausgabe durchführen | DAC12 |

6.4 Analogausgänge 16 Bit

ADwin-X-A20 hat 2 analoge Ausgänge mit 16 Bit-Wandlern (DAC16-1, DAC16-2), Pinbelegung siehe [Seite 10](#). Jedem Ausgang ist ein eigener Digital-Analog-Wandler (DAC) zugeordnet.

Der DAC hat eine Wandlungszeit von 1 µs.

Bitte beachten Sie die [Berechnungsgrundlagen](#) zur Verarbeitung der DAC-Werte.

Befehle zur Programmierung der analogen Ausgänge sind ab [Seite 55](#) beschrieben. Die Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle |
|---|--------------------------------------|
| vollständige Werteausgabe durchführen | DAC |
| Werteausgabe in Teilschritten durchführen | Write_DAC Start_DAC |
| Mehrere Funktionen synchron starten. | Sync_All |

6.5 TTL-Digitalkanäle DIO39:DIO32

Auf der Sub-D-Buchse `Conn. 1` stehen 8 Digitalkanäle (DIO39:DIO32) zur Verfügung. Die Kanäle werden in 4er-Gruppen als Eingänge oder Ausgänge konfiguriert. Die Pinbelegung der Sub-D-Buchse finden Sie in Abbildung 4 auf [Seite 10](#).

Die Kanäle DIO34:DIO32 können als Zählereingang doppelt belegt sein (siehe [Option CO1](#)). In diesem Fall kann nur einer der beiden Zwecke (Digitalkanal oder Zählereingang) genutzt werden.

Die digitalen Kanäle sind TTL-ähnlich und nicht gegen Überspannung geschützt. Jeder Eingang ist mit einem Pull-down-Widerstand (10kΩ) bestückt.

Befehle zur Programmierung der analogen Eingänge sind ab [Seite 79](#) beschrieben. Die Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle |
|--|---|
| Kanäle konfigurieren. | Conf_DIO |
| Eingangsfiler konfigurieren | Digin_Filter_Init |
| Digitaleingänge lesen. | Digin, Digin_Long2 |
| Flanken an Digitaleingängen überwachen. | Digin_Edge2 |
| Digitalausgänge setzen. | Digout Digout_Long2 Digout_Bits2 Get_Digout_Long2 |
| Digitalkanäle über Latch-Register lesen und setzen. | Dig_Latch Digin_Read_Latch2 Digout_Write_Latch2 |
| Ausgangs-Fifo verwenden (nur in Verbindung mit Option DCT). | Digout_Fifo_Read_Timer Digout_Fifo_Clear Digout_Fifo_Enable Digout_Fifo_Empty Digout_Fifo_Mode Digout_Fifo_Start Digout_Fifo_Write |

Programmieren



Programmierung

| Bereich | Befehle |
|--|---|
| Eingangs-Fifo verwenden (nur in Verbindung mit Option DCT). | <code>Digin_Fifo_Read_Timer</code> <code>Digin_Fifo_Clear</code> <code>Digin_Fifo_Enable</code> <code>Digin_Fifo_Full</code> <code>Digin_Fifo_Read</code> |
| Mehrere Funktionen synchron starten. | <code>Sync_All</code> |

6.6 Event-Eingang

ADwin-X-A20 besitzt einen externen Trigger-Eingang (EVENT) am Pin 20 der Sub-D-Buchse Conn. 3 (siehe [Abb. 4 – Pin-Belegung Ein-/Ausgänge \(Buchse\)](#)).

Ein externes Signal (Trigger) mit steigender Flanke an diesem Eingang kann einen Prozesszyklus aufrufen, der sofort und vollständig abgearbeitet wird (siehe *ADbasic*-Handbuch oder -Online-Hilfe, Kapitel „Prozesse im Betriebssystem“).

Sie können mit dem Befehl `CPU_Event_Config` genau konfigurieren, welche Flanken am Event-Eingang einen Prozesszyklus starten.

Der Event-Eingang besitzt einen internen Pull-down-Widerstand (4,7 kΩ).

Per Software kann der Pegel des Event-Eingangs als Digitaleingang DIGIN62 abgerufen werden.

6.7 LS-Bus

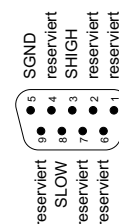
ADwin-X-A20 stellt einen LS-Bus-Anschluss auf dem 9-poligen Sub-D-Verbinder (Buchse) LS-BUS zur Verfügung.

Der LS-Bus (Low Speed) ist ein bidirektionaler, serieller Bus mit 5MHz Taktrate. Der Bus ist eine Eigenentwicklung für den Anschluss externer Module. Es steht der Modultyp HSM-24V zur Verfügung, mit dem 24V-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 5m.

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.

Befehle zur Programmierung der LS-Bus-Module sind ab [Seite 168](#) beschrieben und in der Online-Hilfe. Die Befehle sind in der Include-Datei `ADwin-X.inc` enthalten.



| Bereich | Befehle |
|--|---|
| LS-Bus-Modul initialisieren | <code>LS_DIO_Init</code> |
| Digitalkanäle am LS-Bus-Modul 1 setzen und zurücklesen | <code>LS_Dig_IO</code> (keine anderen Befehle nutzbar) |
| Digitalsignale ausgeben | <code>LS_Digout_Long</code> <code>LS_Digout_Long_BS</code> |
| Digitalsignale einlesen | <code>LS_Digin_Long</code> <code>m</code> |
| Überstrom-Status der Modul-Ausgänge lesen | <code>LS_Get_Output</code> |
| Schnittstelle zurücksetzen | <code>LS_Reset</code> |
| Watchdog am LS-Bus nutzen | <code>LS_Watchdog_Init</code> <code>LS_Watchdog_Reset</code> |

6.8 Synchrone Aktionen

ADwin-X-A20 erlaubt, mit dem Befehl `Sync_All` mehrere Aktionen an verschiedenen Ein- und Ausgängen synchron zu starten. Der Befehl ist auf [Seite 52](#) beschrieben.

Folgende Aktionen sind (je nach Version) möglich: Analogsignale einlesen und ausgeben, Digitalsignale einlesen und ausgeben, Flankenausgabe und Flankenüberwa-

chung an Digitalkanälen starten, Zählerwerte kopieren, Zähler zurücksetzen, SSI-Signal auslesen.

Programmierung

7 Option CO1

Die Option X-A20-CO1 stellt einen zusätzlichen TTL-Zählerblock mit der Nummer 1 zur Verfügung.

Die Zählereingänge liegen auf den Pins DIO34:DIO32 auf der Sub-D-Buchse Conn. 1. Die Pin-Belegung der Sub-D-Buchse finden Sie in Abbildung 4 auf Seite 10.

Die Pins sind als Digitalkanäle doppelt belegt, siehe TTL-Digitalkanäle DIO39:DIO32. Die Kanäle müssen mit **Conf_DIO** als Digitaleingänge konfiguriert werden, damit sie als Zählereingang genutzt werden können.

Die Zählereingänge A/CLK, B/DIR und CLR/LATCH sind TTL-ähnlich und gegen Überspannung nicht geschützt.

Die Funktionen des Zählerblocks 1 sind in Kapitel 15 "Zählerblock" beschrieben.

Befehle zur Programmierung der analogen Eingänge sind ab Seite 115 beschrieben. Die Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle |
|---|---|
| Kanäle konfigurieren. | Conf_DIO |
| Zähler löschen. | Cnt_Clear |
| Zähler sperren oder freigeben (achten Sie auf bereits laufende Zähler). | Cnt_Enable Cnt_PW_Enable |
| Statusregister auslesen. | Cnt_Get_Status |
| Zählerstand in Latch A kopieren. | Cnt_Latch |
| Zähler und PWM-Zähler gleichzeitig in Zwischenspeicher kopieren. | Cnt_Sync_Latch |
| Betriebsart eines Zählers festlegen. | Cnt_Mode |
| Zählerstand in Latch A kopieren und auslesen. | Cnt_Read |
| Latch A (getriggert durch pos. Flanke) auslesen. | Cnt_Read_Latch |
| Inhalt eines Zählerregisters zurückgeben. | Cnt_Read_Int_Register |
| Zählerstand von PWM-Zählern in Latch kopieren. | Cnt_PW_Latch |
| Frequenz und Tastverhältnis eines PWM-Zählers lesen. | Cnt_Get_PW |
| Eintast- und Austastzeit eines PWM-Zählers lesen. | Cnt_Get_PW_HL |
| Mehrere Funktionen synchron starten. | Sync_All |

8 Option D

Option X-A20-D stellt zusätzlich zur Verfügung:

- 8 Diff. Digitalkanäle DIO47:DIO40
- 2 Diff. Zähler 4, 5
- 1 SSI-Schnittstelle

8.1 Diff. Digitalkanäle DIO47:DIO40

Auf der Sub-D-Buchse Conn. 2 stehen 8 differentielle Digitalkanäle zur Verfügung (DIGIN47: DIGIN42, DIO41:DIO40). Die Kanäle DIO41:DIO40 können einzeln als Eingang oder Ausgang konfiguriert werden, alle anderen Kanäle sind fest als Eingänge gesetzt. Die Pin-Belegung der Sub-D-Buchse finden Sie in Abbildung 4 auf Seite 10.

Die Kanäle DIGIN47:DIGIN42 sind als Zählereingänge doppelt belegt (siehe Diff. Zähler 4, 5). Es kann nur einer der beiden Zwecke (Digitalkanal oder Zählereingang) genutzt werden.

Die Kanäle sind differentiell und gegen Überspannung nicht geschützt. Für jeden Kanal sind je ein Plus- und ein Minus-Pin vorhanden; die Differenz der beiden Spannungen ergibt das Signal (jedoch nicht potenzialfrei). Die Kennwerte der Eingänge DIGIN47: DIGIN42 unterscheiden sich von denen der Kanäle DIO41:DIO40, siehe Anhang.

Zwischen den Pins der Eingänge DIGIN47: DIGIN42 ist jeweils ein Busabschluss von 120Ω vorhanden.

An den Eingängen sind TTL-ähnliche Signale erforderlich.

Die Funktionalität der Digitalkanäle wird mit *ADbasic*-Befehlen komfortabel programmiert (Beschreibung ab Seite 79):

| Bereich | Befehle |
|---|--|
| Kanäle gruppenweise konfigurieren. | Conf_DIO |
| Eingangsfiler konfigurieren | Digin_Filter_Init |
| Digitaleingänge lesen. | Digin, Digin_Long2 |
| Flanken an Digitaleingängen überwachen. | Digin_Edge2 |
| Digitalausgänge setzen. | Digout Digout_Long2 Digout_Bits2 Get_Digout_Long2 |
| Digitalkanäle über Latch-Register lesen und setzen. | Dig_Latch Digin_Read_Latch2 Digout_Write_Latch2 |
| Mehrere Funktionen synchron starten. | Sync_All |

8.2 Diff. Zähler 4, 5

Option X-A20-D stellt zwei zusätzliche differentielle Zähler mit den Nummer 4 und 5 zur Verfügung.

Die Zählereingänge liegen auf den Pins DIGIN47:DIGIN42 auf der Sub-D-Buchse Conn. 2. Die Pin-Belegung der Sub-D-Buchse finden Sie in Abbildung 4 auf Seite 10.

An den Eingängen sind TTL-ähnliche Signale erforderlich.

Die Pins sind als Digitalkanäle doppelt belegt, siehe Diff. Digitalkanäle DIO47:DIO40. Die Kanäle müssen mit **Conf_DIO** als Digitaleingänge konfiguriert werden, damit sie als Zählereingang genutzt werden können.

Die Zählereingänge A/CLK, B/DIR und CLR/LATCH sind differentiell und gegen Überspannung nicht geschützt. Für jeden Eingang sind je ein Plus- und ein Minus-Pin vorhanden, zwischen denen das Eingangssignal gemessen wird (jedoch nicht potenzialfrei). Zwischen den Pins ist jeweils ein Busabschluss von 120Ω vorhanden. An jedem Eingang müssen Plus- und Minus-Pin angeschlossen werden.

Alle Funktionen der Zählerblöcke 4 und 5 sind in Kapitel 15 "Zählerblock" beschrieben.



Programmierung

Programmierung

Befehle zur Programmierung der Zähler sind ab [Seite 115](#) beschrieben. Die Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle |
|---|---|
| Kanäle gruppenweise konfigurieren. | <code>Conf_DIO</code> |
| Zähler löschen. | <code>Cnt_Clear</code> |
| Zähler sperren oder freigeben (achten Sie auf bereits laufende Zähler). | <code>Cnt_Enable</code> <code>Cnt_PW_Enable</code> |
| Statusregister auslesen. | <code>Cnt_Get_Status</code> |
| Zählerstand in Latch A kopieren. | <code>Cnt_Latch</code> |
| Zähler und PWM-Zähler gleichzeitig in Zwischenspeicher kopieren. | <code>Cnt_Sync_Latch</code> |
| Betriebsart eines Zählers festlegen. | <code>Cnt_Mode</code> |
| Zählerstand in Latch A kopieren und auslesen. | <code>Cnt_Read</code> |
| Latch A (getriggert durch pos. Flanke) auslesen. | <code>Cnt_Read_Latch</code> |
| Inhalt eines Zählerregisters zurückgeben. | <code>Cnt_Read_Int_Register</code> |
| Zählerstand von PWM-Zählern in Latch kopieren. | <code>Cnt_PW_Latch</code> |
| Frequenz und Tastverhältnis eines PWM-Zählers lesen. | <code>Cnt_Get_PW</code> |
| Eintast- und Austastzeit eines PWM-Zählers lesen. | <code>Cnt_Get_PW_HL</code> |
| Mehrere Funktionen synchron starten. | <code>Sync_All</code> |

8.3 SSI-Schnittstelle

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

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

Die Anschlüsse des Decoders stehen auf der Buchse `Conn. 2` zur Verfügung, Pins `SSI_CLK`, `SSI_DATA`. Die Pinbelegung der Sub-D-Buchse finden Sie in [Abbildung 4](#) auf [Seite 10](#).

Der Anschluss `SSI_DATA / DIGIN60` kann auch als Digitaleingang verwendet werden, mit dem der SSI-Pegel ausgewertet werden kann.

Eigenschaften einstellen

Folgende Eigenschaften des SSI-Decoders sind per Software einstellbar:

- Taktrate: Über einen Vor-Teiler sind Taktraten von ca. 100kHz bis 2,5 MHz möglich mit `SSI_Set_Clock`.
- Zeitabstand: Zeitabstand zwischen dem Einlesen von zwei Encoder-Werten mit `SSI_Set_Delay`.
- Auflösung: Einstellbar bis 32 Bit mit `SSI_Set_Bits`.

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. 9 – Listing: Konvertierung von Gray- in Binär-Code

Die SSI-Decoder werden mit Befehlen aus <ADwin-X.inc> komfortabel programmiert; Beschreibung ab [Seite 132](#) oder in der Online-Hilfe.

| Bereich | Befehle |
|--------------------------------------|--|
| Decoder initialisieren | SSI_Mode SSI_Set_Bits SSI_Set_Clock SSI_Set_Delay |
| Encoder-Daten auslesen | SSI_Read SSI_Start SSI_Status |
| Mehrere Funktionen synchron starten. | Sync_All |



Beispiel:
Umsetzung von
Gray-Code

Programmierung

9 Option DCT

Option X-A20-DCT umfasst alle Funktionen der [Option D](#). Zur Verfügung stehen damit zusätzlich:

- 52 Digitalkanäle
 - 32 [TTL-Digitalkanäle DIO31:DIO00](#), [Seite 22](#)
 - 8 [Diff. Digitalkanäle DIO47:DIO40](#), [Seite 19](#) (siehe Option D)
 - 12 [Komparatoreingänge DIO59:DIO48](#), [Seite 22](#)
 - [Flankenüberwachung und Flankenausgabe](#) für Digitalkanäle, [Seite 23](#)
- Zähler
 - 2 [TTL-Zähler 2, 3](#), [Seite 24](#)
 - 2 [Diff. Zähler 4, 5](#), [Seite 19](#) (siehe Option D)
 - 2 [Komparatorzähler 6, 7](#), [Seite 24](#)
- 1 [SSI-Schnittstelle](#), [Seite 20](#) (siehe Option D)

9.1 TTL-Digitalkanäle DIO31:DIO00

Auf der Sub-D-Buchse `Conn. 3` stehen 32 Digitalkanäle zur Verfügung (DIO31:DIO00). Die Kanäle können in Gruppen von 8 Kanälen als Eingang oder Ausgang konfiguriert werden. Die Pin-Belegung der Sub-D-Buchse finden Sie in [Abbildung 4](#) auf [Seite 10](#).

Die Kanäle DIO31:DIO26 sind als Zählereingänge doppelt belegt (siehe [TTL-Zähler 2, 3](#)). Es kann nur einer der beiden Zwecke (Digitalkanal oder Zählereingang) genutzt werden.



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

Eingangs- und Ausgangs-FIFOs ermöglichen Flankenüberwachung und -ausgabe an Digitalkanälen, siehe [Flankenüberwachung und Flankenausgabe](#) auf [Seite 23](#).

Befehle zur Programmierung siehe unten ([Kapitel 9.2](#) und [Kapitel 9.3](#)).

9.2 Komparatoreingänge DIO59:DIO48

Auf der Sub-D-Buchse `Conn. 2` stehen 12 Digitaleingänge DIGIN59:DIGIN48 mit Komparatorfunktion zur Verfügung, Pinbelegung siehe [Seite 10](#). Die Digitalkanäle sind fest als Eingänge geschaltet und können nicht als Ausgänge konfiguriert werden.

DIGIN53:DIGIN48 sind doppelt belegt und können als Eingänge für Komparatorzähler genutzt werden (siehe [Komparatorzähler 6, 7](#)). Es kann nur einer der beiden Zwecke (Digitalkanal oder Zählereingang) genutzt werden.

Die Komparatoreingänge sind intern mit den Analogausgängen der 12 Bit-DAC verbunden, DAC12-1 mit DIGIN55:DIGIN48 und DAC12-2 mit DIGIN59:DIGIN56. Die eingestellte DAC-Spannung (0V...5V) dient als Komparatorsignal, d.h. ein anliegendes Digitalsignal mit einer kleineren Spannung wird als Pegel Low verarbeitet, mit einer höheren Spannung als Pegel High.

Die maximale Messfrequenz hängt vom (am DAC) eingestellten Komparatorsignal und von der Spannung des Eingangssignals ab. Bei Eingangsspannungen um +24V liegt die maximale Messfrequenz unter 30kHz.

Für eine genaue Messung der Pulsbreite des Eingangssignals ist eine Messfrequenz deutlich unter dem Maximum zu wählen.

Die Komparatoreingänge können mit der Flankenüberwachung kombiniert werden, siehe [Flankenüberwachung und Flankenausgabe](#) auf [Seite 23](#).

Befehle zur Programmierung der Digitalkanäle sind ab [Seite 79](#) beschrieben. Die Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle |
|---|--|
| Kanäle gruppenweise konfigurieren. | <code>Conf_DIO</code> |
| Eingangsfilter konfigurieren | <code>Digin_Filter_Init</code> |
| Digitaleingänge lesen. | <code>Digin</code> <code>Digin_Long1/2</code> |
| Flanken an Digitaleingängen überwachen. | <code>Digin_Edge1/2</code> |

Programmierung

| Bereich | Befehle |
|---|--|
| Digitalausgänge setzen. | <code>Digout</code> <code>Digout_Long1/2</code> <code>Digout_Bits1/2</code> <code>Get_Digout_Long1/2</code> |
| Digitalkanäle über Latch-Register lesen und setzen. | <code>Dig_Latch</code> <code>Digin_Read_Latch1/2</code> <code>Digout_Write_Latch1/2</code> |
| Mehrere Funktionen synchron starten. | <code>Sync_All</code> |

9.3 Flankenüberwachung und Flankenausgabe

Option X-A20-DCT kann mit Hilfe von Eingangs- und Ausgangs-FIFOs automatisch die Flanken an den ausgewählten Eingangskanälen überwachen sowie selbstständig Pegel zu bestimmten Zeitpunkten auf den Digitalausgängen ausgeben.

Überwachung und Ausgabe sind für alle Digitalkanäle von X-A20 verfügbar, nicht nur für Kanäle der Option DCT.

Es gibt zwei Eingangs-FIFOs und zwei Ausgangs-FIFOs, jeder FIFO bezieht sich auf 32 Digitalkanäle (DIO31:DIO00 oder DIO60:DIO32).

Eine Flankenüberwachung prüft alle 10ns, ob sich an den festgelegten Eingangskanälen ein Pegel geändert hat bzw. ob eine Flanke aufgetreten ist. Mit jeder Änderung wird ein Wertepaar in den entsprechenden Eingangs-FIFO kopiert:

- Wert 1 enthält den Pegelzustand aller 32 Kanäle als Bitmuster.
- Wert 2 enthält einen Zeitstempel, den aktuellen Stand eines 100MHz-Zählers. Jede Flankenüberwachung hat ihren eigenen Zähler.

Ein Eingangs-FIFO 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.

Die Flankenüberwachungen für die Kanäle DIO31:DIO00 und DIO60:DIO32 arbeiten unabhängig voneinander.

Alternativ können Sie mit `Digin_Edge1/2` das Auftreten von Flanken an ausgewählten Eingangskanälen registrieren (ohne Zeitstempel). Wenn eine positive oder eine negative Flanke an einem Eingang auftritt, wird in einem Zwischenspeicher das entsprechende Bit des Eingangskanals gesetzt. Der Inhalt des Zwischenspeichers kann jederzeit abgefragt werden. Die Anzahl und der Zeitpunkt aufgetretener Flanken werden nicht festgehalten.

Die Flankenausgabe kann selbstständig Flanken zu bestimmten Zeitpunkten auf den Digitalausgängen ausgeben. Ein Ausgangs-FIFO dient als Zwischenspeicher für die vom Benutzer festgelegten Pegel und Zeitpunkte, maximal 511 Wertepaare. Der Ausgabezeitpunkt kann auf 10ns genau festgelegt werden. Bei einer Ausgabe werden die Pegel aller Digitalausgänge ausgegeben.

Befehle zur Programmierung der FIFOs sind ab [Seite 79](#) beschrieben. Die Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle |
|--------------------------------------|---|
| Ausgangs-Fifo verwenden. | <code>Digout_Fifo_Read_Timer</code> <code>Digout_Fifo_Clear</code> <code>Digout_Fifo_Enable</code> <code>Digout_Fifo_Empty</code> <code>Digout_Fifo_Mode</code> <code>Digout_Fifo_Start</code> <code>Digout_Fifo_Write</code> |
| Eingangs-Fifo verwenden. | <code>Digin_Fifo_Read_Timer</code> <code>Digin_Fifo_Clear</code> <code>Digin_Fifo_Enable</code> <code>Digin_Fifo_Full</code> <code>Digin_Fifo_Read</code> |
| Mehrere Funktionen synchron starten. | <code>Sync_All</code> |

Flankenüberwachung

Zeitgesteuerte Flankenausgabe

Programmierung

9.4 TTL-Zähler 2, 3

Option X-A20-DCT stellt zwei zusätzliche TTL-Zähler mit den Nummer 2 und 3 zur Verfügung.

Die Zählereingänge liegen auf den Pins DIO31:DIO26 auf der Sub-D-Buchse Conn. 3. Die Pin-Belegung der Sub-D-Buchse finden Sie in Abbildung 4 auf Seite 10.

Die Pins DIO31:DIO26 sind als Digitaleingänge doppelt belegt, siehe [TTL-Digitalkanäle DIO31:DIO00](#). Die Kanäle müssen mit **Conf_DIO** als Digitaleingänge konfiguriert werden, damit sie als Zählereingang genutzt werden können.

Die Funktion der Zähler ist in [Kapitel 15 auf Seite 39](#) beschrieben.

Befehle zur Programmierung der Zähler sind ab [Seite 115](#) beschrieben, Übersicht siehe unten ([Kapitel 9.5](#)).

9.5 Komparatorzähler 6, 7

Option X-A20-DCT stellt zwei zusätzliche Komparatorzähler mit den Nummer 6 und 7 zur Verfügung.

Die Zählereingänge liegen auf den Pins DIGIN53:DIGIN48 auf der Sub-D-Buchse Conn. 2. Die Pin-Belegung der Sub-D-Buchse finden Sie in Abbildung 4 auf Seite 10.

Die Pins sind als Digitalkanäle doppelt belegt, siehe [Komparatoreingänge DIO59:DIO48](#). Die Kanäle müssen mit **Conf_DIO** als Digitaleingänge konfiguriert werden, damit sie als Zählereingang genutzt werden können.

Die Komparator-Zählereingänge DIGIN53:DIGIN48 sind intern mit dem Analogausgang des 12 Bit-DAC DAC12-1 verbunden (vergleiche auch [Komparatoreingänge DIO59:DIO48](#)). Die eingestellte DAC-Spannung (0V...5V) dient als Komparatorsignal, d.h. ein anliegendes Digitalsignal mit einer kleineren Spannung wird als Pegel Low verarbeitet, mit einer höheren Spannung als Pegel High.

Alle Funktionen der Zählerblöcke 6 und 7 sind in [Kapitel 15 "Zählerblock"](#) beschrieben.

Befehle zur Programmierung der Zähler sind ab [Seite 115](#) beschrieben. Die Befehle sind in der Include-Datei `ADwin-X.inc` für *ADbasic* enthalten und werden auch in der Online-Hilfe erläutert.

| Bereich | Befehle |
|---|---|
| Kanäle gruppenweise konfigurieren. | Conf_DIO |
| Zähler löschen. | Cnt_Clear |
| Zähler sperren oder freigeben (achten Sie auf bereits laufende Zähler). | Cnt_Enable Cnt_PW_Enable |
| Statusregister auslesen. | Cnt_Get_Status |
| Zählerstand in Latch A kopieren. | Cnt_Latch |
| Zähler und PWM-Zähler gleichzeitig in Zwischenspeicher kopieren. | Cnt_Sync_Latch |
| Betriebsart eines Zählers festlegen. | Cnt_Mode |
| Zählerstand in Latch A kopieren und auslesen. | Cnt_Read |
| Latch A (getriggert durch pos. Flanke) auslesen. | Cnt_Read_Latch |
| Inhalt eines Zählerregisters zurückgeben. | Cnt_Read_Int_Register |
| Zählerstand von PWM-Zählern in Latch kopieren. | Cnt_PW_Latch |
| Frequenz und Tastverhältnis eines PWM-Zählers lesen. | Cnt_Get_PW |
| Eintast- und Austastzeit eines PWM-Zählers lesen. | Cnt_Get_PW_HL |
| Mehrere Funktionen synchron starten. | Sync_All |

10 Option COM

Option X-A20-COM beinhaltet zusätzliche Schnittstellen, die unabhängig voneinander konfiguriert und betrieben werden:

- 2 CAN-Schnittstellen „High-speed“, Seite 25.
- 1 RS232-Schnittstelle, Seite 27.

10.1 CAN-Schnittstellen

Die CAN-Schnittstellen 1 und 2 („High-speed“) werden voneinander unabhängig betrieben.

10.1.1 Hardware-Beschreibung

Die Anschlüsse der Schnittstellen stehen auf 9-poligen Sub-D-Verbindern CAN1 und CAN2 zur Verfügung, die Belegung ist gleich.

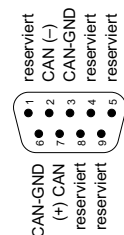


Abb. 10 – CAN: Pinbelegung

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

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.

10.1.2 Beschreibung der CAN-Schnittstelle

Die CAN-Schnittstelle 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 einen Eingangs-FIFO und einen Ausgangs-FIFO.

Der CAN-Bus (high speed) ist auf Frequenzen bis 1 MHz einstellbar und wird standardmäßig mit 1 MHz betrieben. Der CAN-Bus ist durch Optokoppler vom ADwin-System galvanisch getrennt.

Nachrichten verwalten

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 $0 \dots 2^{11}-1$ bzw. $0 \dots 2^{29}-1$.

Zu sendende Nachrichten speichert der Controller in einem Ausgangs-FIFO, empfangene Nachrichten in einem Eingangs-FIFO. Nach der Initialisierung des CAN-Controllers sind beide FIFOs nicht konfiguriert und beteiligen sich nicht am Busverkehr.

Zusätzlich können Nachrichten mit hoher Priorität verschickt werden. Im Ausgangs-FIFO zum Senden gespeicherte CAN-Nachrichten werden dabei zurückgestellt.

Identifier

Eingangs-/Ausgangs-FIFO

Nachricht senden

In *ADbasic* erhalten Sie eine CAN-Nachricht nach dem Empfang über das Feld `can_msg []` aus dem Eingangs-FIFO. Das Feld enthält 8 Datenbytes, die Anzahl der Datenbytes, den Identifier und (nur beim Empfangen) einen Empfangs-Zeitstempel (11 Elemente). Ebenso wird eine Nachricht beim Senden über das Feld `can_msg []` übertragen.

Das Versenden einer Nachricht läuft in folgenden Schritten ab:

- Sie speichern die Nachricht mit Identifier im Feld `can_msg []`.
- Sie übergeben das Feld `can_msg []` mit `CAN_Transmit` als Nachricht an die CAN-Schnittstelle. Sobald der Bus frei ist, wird die Nachricht gesendet.
 - Bei normaler Priorität speichert die Schnittstelle die Nachricht im Ausgangs-FIFO. Die Nachrichten werden in der Reihenfolge gesendet, wie sie gespeichert wurden.
 - Bei hoher Priorität wird die Nachricht gesendet, sobald die CAN-Schnittstelle Zugriffsrecht auf den CAN-Bus hat. Im Ausgangs-FIFO bereits zum Senden gespeicherte CAN-Nachrichten müssen warten.

Nachricht empfangen

Das Empfangen einer Nachricht läuft in folgenden Schritten ab:

- Sie stellen den Empfangsfilter für ausgewählte Identifier ein (`CAN_RX_Set_Filter`). Wenn Sie den Empfangsfilter nicht setzen, werden alle CAN-Nachrichten akzeptiert.
- Der Controller überwacht den CAN-Bus auf eingehende Nachrichten, filtert sie gemäß Empfangsfilter und speichert sie im Eingangs-FIFO.
- Sie übertragen die Nachricht aus dem FIFO in das Feld `can_msg []` (`CAN_Receive`) und lesen den zugehörigen Identifier aus.

Im Eingangs-FIFO können bis zu 64 CAN-Nachrichten gespeichert werden. Sind alle FIFO-Plätze belegt, überschreibt eine eingehende Nachricht alte Daten, 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.

Eingehende Nachrichten filtern

Sie können mit `CAN_RX_Set_Filter` bis zu vier Empfangsfilter für eingehende Nachrichten festlegen. Filter können einzeln aktiviert und deaktiviert werden. Der Identifier der eingehenden Nachricht wird mit den aktiven Empfangsfiltern verglichen und akzeptiert oder verworfen:

- Wenn der Identifier der Nachricht gleich dem Empfangsfilter ist, wird die Nachricht in den Eingangs-FIFO übernommen.
- Sobald eine Nachricht einen der aktiven Empfangsfilter erfolgreich durchlaufen hat, wird sie im Empfangs-FIFO gespeichert.

Busfrequenz einstellen

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

Bei der Initialisierung mit `CAN_Init` wird die CAN-Bus-Frequenz festgelegt. In Sonderfällen kann es vorteilhaft sein, die Einstellungen anders zu wählen, als es mit `CAN_Init` möglich ist. Wenden Sie sich in diesem Fall an unseren Support.

Programmierung

Die CAN-Schnittstellen werden mit Befehlen aus `ADwin-X.inc` für *ADbasic* komfortabel programmiert; Beschreibung ab [Seite 140](#) oder in der Online-Hilfe:

| Bereich | Befehle |
|--------------------------------|--|
| Initialisierung | <code>CAN_Init</code> |
| Empfangen und Senden von Daten | <code>can_msg []</code> <code>CAN_Receive</code> <code>CAN_RX_Set_Filter</code> <code>CAN_Transmit</code> |

10.2 RS232-Schnittstelle

Die RS232-Schnittstelle wird ohne Handshake betrieben. Die Anschlüsse der Schnittstelle stehen auf dem 9-poligen Sub-D-Verbinder RS232 zur Verfügung.

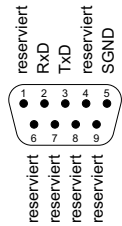
Die Schnittstelle verfügt über einen Eingangs- und einen Ausgangs-FIFO mit einer Länge von jeweils 64 Byte.

Folgende Schnittstellen-Parameter können eingestellt werden:

- 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.
- Datenbits: Die Nutzdaten können aus 6...8 Bits bestehen.
- Stopp-Bits: Die Anzahl der Stopp-Bits kann auf 1, 1½ oder 2 eingestellt werden.
- Baudrate: Die erreichbaren Werte liegen bei 35 Baud ... 115,2kBaud. Typische Baudraten sind 300, 600, 1200, 2400, ..., 115200 Baud.

Die Funktionalität und Programmierung der Schnittstelle beruhen auf dem eingebauten Schnittstellen-Controller. Der Controller wird mit Befehlen aus ADwin-X.inc für ADbasic komfortabel programmiert; Beschreibung ab Seite 150 oder in der Online-Hilfe:

| Bereich | Befehle |
|--------------------------------|--|
| Initialisierung | RS_Init |
| Empfangen und Senden von Daten | RS_Read_FIFO RS_Write_FIFO RS_Write_FIFO_Full RS_Write_FIFO_Empty |



Programmierung

11 Option Profibus

Die Option *X-A20-Profibus* stellt einen Feldbusknoten mit der Funktionalität eines Profibus-Slave zur Verfügung. Alle Einstellungen werden per Software vorgenommen.

Die Option kann nicht mit [Option Profinet-IRT](#) oder [Option EtherCAT](#) kombiniert werden.

Funktionsbeschreibung

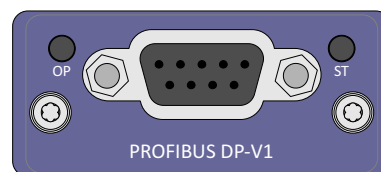
Nach dem Einschalten muss der Feldbusknoten per Software initialisiert werden. Mit der Initialisierung werden die Stationsadresse (Slave-Knotenadresse) auf dem Profibus und die Größe des Eingangs- und des Ausgangsbereichs festgelegt.

Es gibt je einen Bereich für eingehende und für ausgehende Daten; die Größe jedes Bereichs kann separat festgelegt werden und zwar auf 1, 2, 4, 8, 16, 32 oder 61 Doppelworte.

Hardware

Die Pinbelegung der 9-poligen D-Sub-Buchse DP-V1 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. |
| | blinkt grün | Feldbusknoten online, Status Clear. |
| | blinkt rot, einfach | Fehler: Ein/Ausgangskonfiguration stimmt nicht mit der Masterkonfiguration überein. |
| | blinkt rot, doppelt | Fehler bei der Profibus-Konfiguration. |
| ST | aus | Nicht online oder keine Stromversorgung. |
| | grün | Initialisiert. |
| | blinkt grün | Initialisiert, Diagnosemeldung liegt an. |
| | rot | Ausnahmefehler. |

Abb. 11 – 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 Siemens und das zugehörige Programm *SIMATIC* verwendet.

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

- Installieren Sie im Programm *SIMATIC* die GSD-Datei *hmsb1815.gsd* (Gerättestammdaten-Datei), Verzeichnis *C:\ADwin\Fieldbus\Profibus*.

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 erscheint der Slave in *SIMATIC* im Profil-Baum und kann angesprochen werden.

- Stellen Sie die Stationsadresse des Slaves in *SIMATIC* genauso ein wie bei der Initialisierung in *ADbasic* mit **Init_Profibus**.

GSD-Datei installieren

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

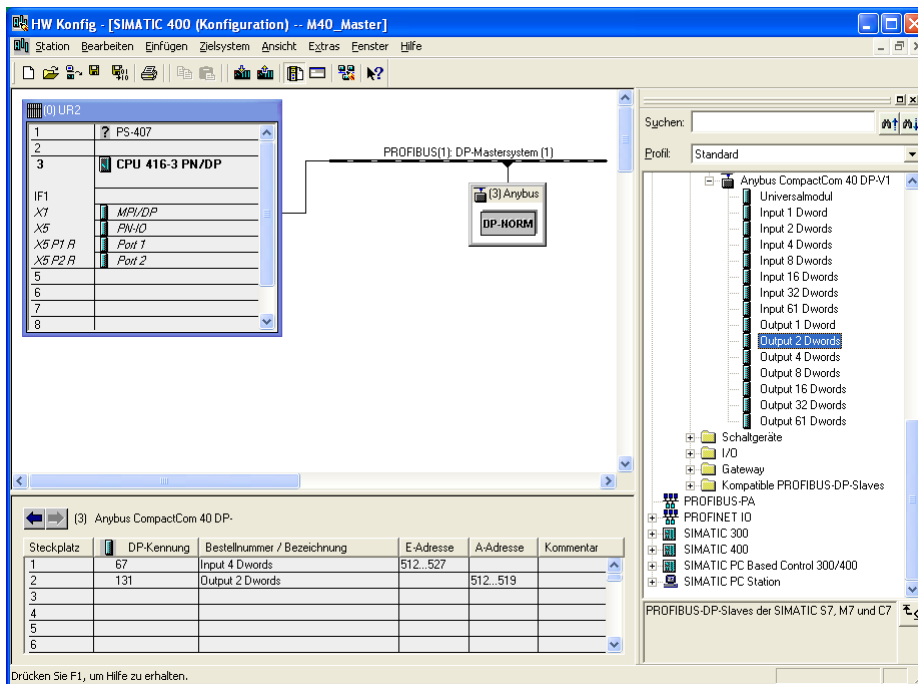
Beachten Sie dabei folgende Regeln:

- Die Begriffe „Eingang“ und „Ausgang“ in *ADbasic* (Slave) und im Konfigurations-Tool (Master) sind vertauscht.
- Konfigurieren Sie (aus Sicht des Konfigurations-Tools) zuerst den Eingang und dann den Ausgang.
Wenn also in *ADbasic* die Eingangsgröße initialisiert wurde, muss dafür im Master der Ausgang mit der gleichen Größe konfiguriert werden.
- Die Größe der Datenbereiche muss die gleiche sein wie bei der Initialisierung in *ADbasic* mit **Init_Profibus**.
- Datenbereiche können in Größen von 1, 2, 4, 8, 16, 32, 61 Doppelworten (dword) angelegt werden.

Beispielhaft wird der Slave in *ADbasic* initialisiert mit Stationsadresse 5, Eingang 2 Doppelworte und Ausgang 4 Doppelworte:

```
Par_31 = Init_Profibus(5, 2, 4, conf_Arr)
```

Um den Slave im Konfigurations-Tool richtig einzurichten, müssen nun zuerst der Eingang mit 4 Doppelworten und dann der Ausgang mit 2 Doppelworten angelegt werden. Die unten stehende Grafik zeigt die Beispielkonfiguration.



Slave konfigurieren

Programmieren in *ADbasic*

Das Modul wird komfortabel mit *ADbasic*-Befehlen programmiert. Die Befehle sind im Handbuch Pro II-Software und in der Online-Hilfe *ADbasic* erläutert.

Die Include-Datei `ADwin-X.inc` enthält Befehle für folgende Bereiche:

| Bereich | Befehle |
|--|----------------------|
| Stationsadresse und Datenbereiche initialisieren | Init_Profibus |
| Daten schreiben und lesen | Run_Profibus |

Die Initialisierung muss mit niedriger Priorität ablaufen, da sie einige Zeit in Anspruch nimmt; bei hoher Priorität würde der PC nach einer bestimmten Zeit (time-out) die Kommunikation abbrechen. Das Schreiben und Lesen von Daten kann hingegen mit hoher Priorität ablaufen.

**Betriebszustände des
Feldbusknotens****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

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

| Betriebs- Zustand | Verhalten |
|----------------------|---|
| Operate | Der Profibus-Slave 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. 12 – Betriebszustände

12 Option Profinet-IRT

Die Option *X-A20-Profinet-IRT* stellt einen Feldbusknoten mit der Funktionalität eines Profinet-IRT-Slave zur Verfügung. Alle Einstellungen werden per Software vorgenommen.

Es gibt die Option mit verschiedenen Anschlüssen:

- Profinet-IRT-Cu: Schnittstelle mit Kupferleiter, 2 Buchsen RJ-45, handelsübliche Steckverbinder.
- Profinet-IRT-FO: Schnittstelle mit Lichtwellenleiter, 2 Duplex-Buchsen SC-RJ (Lichtwellenleiter).

Die Option kann nicht mit [Option Profibus](#) oder [Option EtherCAT](#) kombiniert werden.

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 1280 Byte. Die Begriffe „Eingang“ und „Ausgang“ sind aus Sicht des Feldbus-Masters zu sehen.

Bei der Initialisierung legen Sie für beide Bereiche separat fest, wieviele Datenbereiche in jedem Bereich vorhanden sind und wie groß die Datenbereiche sind. Im Betrieb kann jedoch nur eine Bereichsgröße genutzt werden.

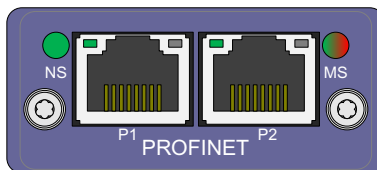
Hardware

An die Buchsen werden handelsübliche Stecker angeschlossen:

- Ethernet-Stecker RJ-45 (IRT-Cu)

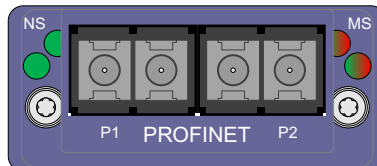
Neben den beiden Buchsen befinden sich zwei LEDs, die den Betriebszustand des Knotens im Profinet anzeigen: Netzwerk-Status (NS) und Modul-Status (MS).

In jeder Buchse befindet sich jeweils eine LINK-LED.



- Glasfaser-Duplexstecker SC-RJ (IRT-FO)

Neben den Buchsen befinden sich vier LEDs. Die halb verdeckten LED zeigen den Betriebszustand des Knotens im Profinet anzeigen: Netzwerk-Status (NS, links) und Modul-Status (MS, rechts).



Die weiter außen liegenden LED für jede Buchse den LINK-Status an.

| LED | Status | Bedeutung |
|------|----------------------|--|
| NS | Aus | Offline: Nicht online oder keine Stromversorgung. |
| | Grün | Online (RUN): Feldbusknoten online, IO-Master arbeitet. |
| | Grün blinkt 1-fach | Online (STOP): Feldbusknoten online, aber IO-Master ruht, Datenfehler oder IRT-Synchronisierung nicht abgeschlossen. |
| | Grün blinkend | Blink-Modus: wird von Prüfwerkzeugen zur Identifizierung des Knotens im Netzwerk genutzt. |
| | Rot (mit MS rot) | Ausnahmefehler: Schwerer interner Fehler; die MS LED ist ebenfalls rot. |
| | Rot blinkt 1-fach | Stationsname ist nicht gesetzt. |
| | Rot blinkt 2-fach | IP-Adresse ist nicht gesetzt. |
| | Rot blinkt 3-fach | Konfigurationsfehler: Erwartete unterscheidet sich von der tatsächlichen Identifizierung. |
| MS | Aus | Nicht initialisiert: Keine Stromversorgung oder Modul im Modus SETUP / NW_INIT. |
| | Grün | Normaler Betrieb. |
| | Grün blinkt 1-fach | Diagnosemeldung liegt an. |
| | Rot | Ausnahmefehler, Gerät im Status EXCEPTION. |
| | Rot (mit NS rot) | Ausnahmefehler: Schwerer interner Fehler; die MS LED ist ebenfalls rot. |
| | Rot / Grün wechselnd | Firmware-Update. Schalten Sie das Gerät nicht ab. Abschalten während dieser Phase könnte dauerhafte Schäden verursachen. |
| LINK | Aus | Kein Verbindung vorhanden. |
| | Grün | Ethernet-Verbindung vorhanden, keine Kommunikation. |
| | Grün flackernd | Ethernet-Verbindung vorhanden, Kommunikation findet statt. |

Abb. 13 – 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.

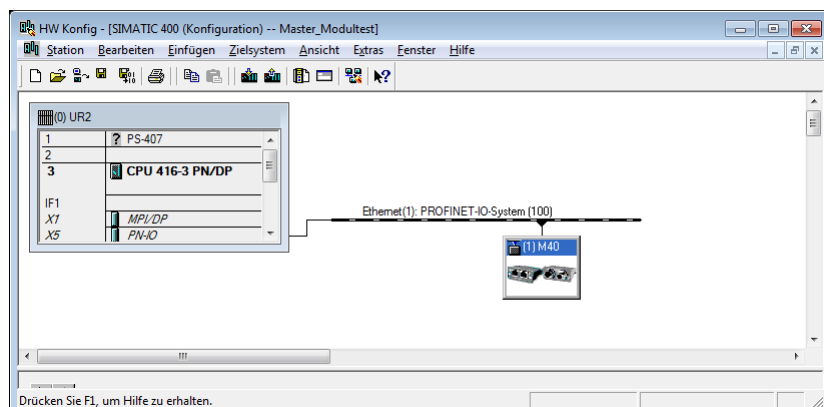
GSD-Datei installieren

- Installieren Sie im Programm *SIMATIC-Manager* die GSD-Datei *GSDML-V2.33-HMS-ABCC40-PIR-ADwin-20180620.xml* (Gerätestammdaten-Datei), Verzeichnis *C:\ADwin\Fieldbus\Profinet*.

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

- Fügen Sie im Konfigurations-Tool den Slave, also den Feldbusknoten zum Profinet hinzu.

Danach könnte das Profinet-Layout wie folgt aussehen:



- Konfigurieren Sie die 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.
- Die Größe der Datenbereiche müssen so eingestellt sein wie bei der Initialisierung in *ADbasic* mit **Init_ProfinetIO**.
- Die Datenbereichsgrößen für Eingang und Ausgang können voneinander unabhängig initialisiert werden. Datenbereiche können angelegt werden in einer der folgenden Größen (1 Doppelwort = 4 Byte):
1, 2, 4, ... 64 Doppelworte; 64, 128, ... 320 Doppelworte
Die 64 Doppelwort-Blöcke sind nummeriert und dürfen nur in aufsteigender Reihenfolge IN0...IN5 / OUT0...OUT5 angelegt werden.

Mit der folgenden Zeile werden in *ADbasic* sowohl der Eingangsbereich als auch der Ausgangsbereich mit 320 Doppelworten (=1280 Byte) initialisiert:

```
Init_ProfinetIO(320, 320, work_arr)
```

Um den Slave im Konfigurations-Tool richtig einzurichten, müssen im Eingangs- und im Ausgangsbereich jeweils fünf Blöcke zu 256 Byte (=64 Doppelworte) angelegt werden. Beachten Sie die Reihenfolge der 64 Doppelwort-Blöcke.

| Steckplatz | Baugruppe | Bestellnummer | E-Adresse | A-Adresse | Diagnoseadresse |
|------------|---------------|---------------|-------------|-------------|-----------------|
| 0 | M40 | ABCC40-PIR | | | 16378* |
| X1 | Interface | | | | 16377* |
| P1 | Port 1 | | | | 16376* |
| P2 | Port 2 | | | | 16375* |
| 1 | IN0064UINT32 | | 0...255 | | |
| 2 | IN1064UINT32 | | 256...511 | | |
| 3 | IN2064UINT32 | | 512...767 | | |
| 4 | IN3064UINT32 | | 768...1023 | | |
| 5 | IN4064UINT32 | | 1024...1279 | | |
| 6 | OUT0064UINT32 | | | 0...255 | |
| 7 | OUT1064UINT32 | | | 256...511 | |
| 8 | OUT2064UINT32 | | | 512...767 | |
| 9 | OUT3064UINT32 | | | 768...1023 | |
| 10 | OUT4064UINT32 | | | 1024...1279 | |
| 11 | | | | | |

Slave einbinden

Slave konfigurieren

Beispiel-Konfiguration

**Betriebszustände des
Feldbusknotens****Programmieren in ADbasic**

Der Feldbusknoten wird mit Befehlen aus `ADwin-X.inc` für *ADbasic* komfortabel programmiert; Beschreibung ab [Seite 160](#) oder in der Online-Hilfe:

| Bereich | Befehle |
|---|------------------------------|
| Reset, Datenbereiche initialisieren | <code>Init_ProfinetIO</code> |
| Daten schreiben und lesen, Nachrichtenverwaltung | <code>Run_ProfinetIO</code> |

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

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. 14 – Profinet: Betriebszustände

13 Option EtherCAT

Die Option X-A20-EtherCAT stellt einen Feldbusknoten mit der Funktionalität eines EtherCAT-Slave zur Verfügung. Alle Einstellungen werden per Software vorgenommen.

Die Option kann nicht mit [Option Profibus](#) oder [Option Profinet-IRT](#) kombiniert werden.

Funktionsbeschreibung

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

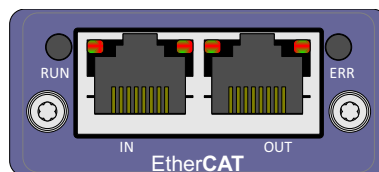
Es gibt je einen Bereich für eingehende und für ausgehende Daten; jeder Bereich hat eine maximale Größe von 1440 Byte. Die Begriffe „Eingang“ und „Ausgang“ sind aus Sicht des Feldbus-Masters zu sehen.

Bei der Initialisierung legen Sie für beide Bereiche separat fest, wieviele Datenbereiche in jedem Bereich vorhanden sind und wie groß die Datenbereiche sind. Im Betrieb kann jedoch nur eine Bereichsgröße genutzt werden.

Hardware

Die Schnittstelle hat zwei Buchsen vom Typ RJ45, die mit IN und OUT bezeichnet sind.

In jeder Buchse befindet sich oben links eine LED „Link / Activity“, die den Betriebszustand des Knotens im EtherCAT-Bus anzeigt. Die LED oben rechts ist ohne Funktion.



Neben den Buchsen befinden sich LEDs, die den Status der EtherCAT-Zustandsmaschine (RUN) und das Auftreten von Kommunikationsfehlern (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. 15 – 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 (Version 3.1) 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.

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

- Kopieren Sie die Beschreibungsdatei des Feldbusknotens HMS CompactCom 40 EtherCAT 2_08.xml aus dem Ordner C:\ADwin\Feldbus\EtherCAT in das Quellverzeichnis des Konfigurations-Tools.

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

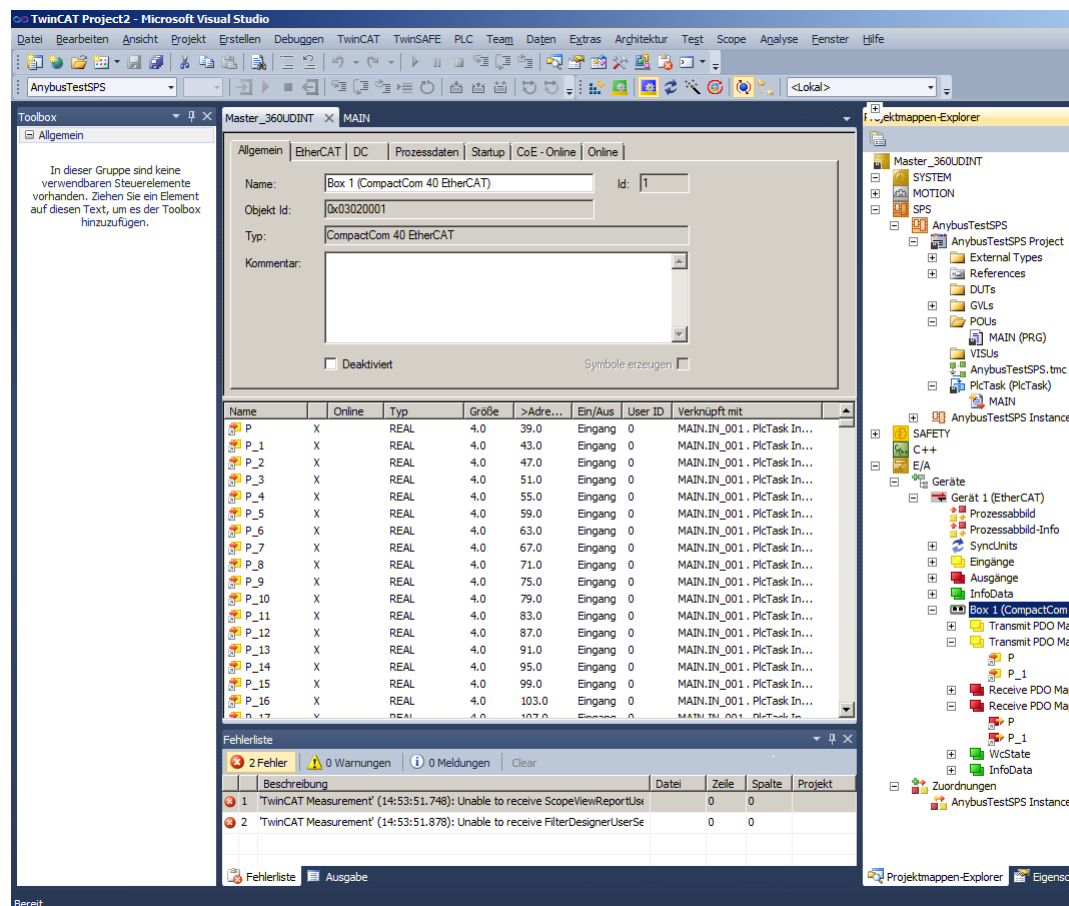
- Fügen Sie den *ADwin*-EtherCAT-Slave als Busteilnehmer zum EtherCAT-Bus hinzu.

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

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

- 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.



- Damit ist die Busprojektierung abgeschlossen und das Modul betriebsbereit.

Programmieren in ADbasic

Der Feldbusknoten wird mit Befehlen aus `ADwin-X.inc` für ADbasic komfortabel programmiert; Beschreibung ab [Seite 164](#) oder in der Online-Hilfe:

| Bereich | Befehle |
|---|----------------------------|
| Reset, Datenbereiche initialisieren | <code>Init_EtherCAT</code> |
| Daten schreiben und lesen, Nachrichtenverwaltung | <code>Run_EtherCAT</code> |

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/>

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

| Betriebs-Zustand | Verhalten |
|------------------|--|
| Init | Der EtherCAT-Slave wird vom Bus-Master initialisiert. |
| Boot | Der EtherCAT-Slave befindet sich im Boot-Modus. |
| 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. 16 – Pro II-EtherCAT-SL Rev. E: Betriebszustände

Betriebszustände der EtherCAT-Schnittstelle

14 Option Boot

ADwin-X-A20-Boot startet eine zuvor programmierte Anwendung automatisch nach dem Einschalten. Damit ist nach dem Einrichten der Anwendung ein Betrieb ohne PC möglich.

Sie programmieren den Bootloader in der Entwicklungsumgebung *ADbasic*, Menüeintrag **Tools / Bootlader**.

Folgende Schritte führt *ADwin-X-A20*-Boot nach dem Einschalten aus:

- Betriebssystem laden.
- Kompilierte Prozesse (max. 10) laden. Prozesse werden mit dem *ADbasic*-Compiler kompiliert.
- Prozess Nr. 10 automatisch starten. Im Prozess 10 können Sie auch weitere Prozesse starten.

Wenn Sie nicht mit der Bootloader-Option arbeiten wollen:

- Vorübergehend deaktivieren:
 - Schalten Sie *ADwin-X-A20* ein.
 - Booten Sie *ADwin-X-A20* aus *ADbasic*. Die gespeicherten Prozesse werden deaktiviert.
 - Nach dem Ausschalten und erneuten Einschalten ist die Bootloader-Option wieder aktiv.
- Dauerhaft deaktivieren:
 - Deaktivieren Sie den Bootloader in *ADbasic* Menüeintrag **Tools / Bootlader**, Tab **Enable/Disable**.
 - Schalten Sie *ADwin-X-A20* aus und wieder ein.

ADwin-X-A20-Boot beinhaltet kein EEPROM (im Unterschied zu anderer *ADwin*-Hardware).

15 Zählerblock

Jeder der bis zu 7 Zähler in ADwin-X-A20 ist als Zählerblock aufgebaut. In einem Zählerblock arbeiten 2 parallele 32 Bit-Zähler, die voneinander unabhängig sind:

- ein Vor-/ Rückwärtszähler für externe Takte mit Takt/Richtung- oder Vierflanken-auswertung
- ein PWM-Zähler mit internem Takteingang zur Pulsweitenmessung.

Sie konfigurieren die Zähler per Software, die Daten der Zähler werden in Latches zum Auslesen zur Verfügung gestellt.

Insgesamt kann ADwin-X-A20 mit bis zu 7 Zählern ausgerüstet sein. Die Zählernummern sind folgenden Optionen zugeordnet:

- Zähler 1 (TTL): [Option CO1, Seite 18](#)
- Zähler 2...3 (TTL): [Option DCT, Seite 24](#)
- Zähler 4...5 (differenziell): [Option D / Option DCT, Seite 19](#)
- Zähler 6...7 (Komparator): [Option DCT, Seite 24](#)

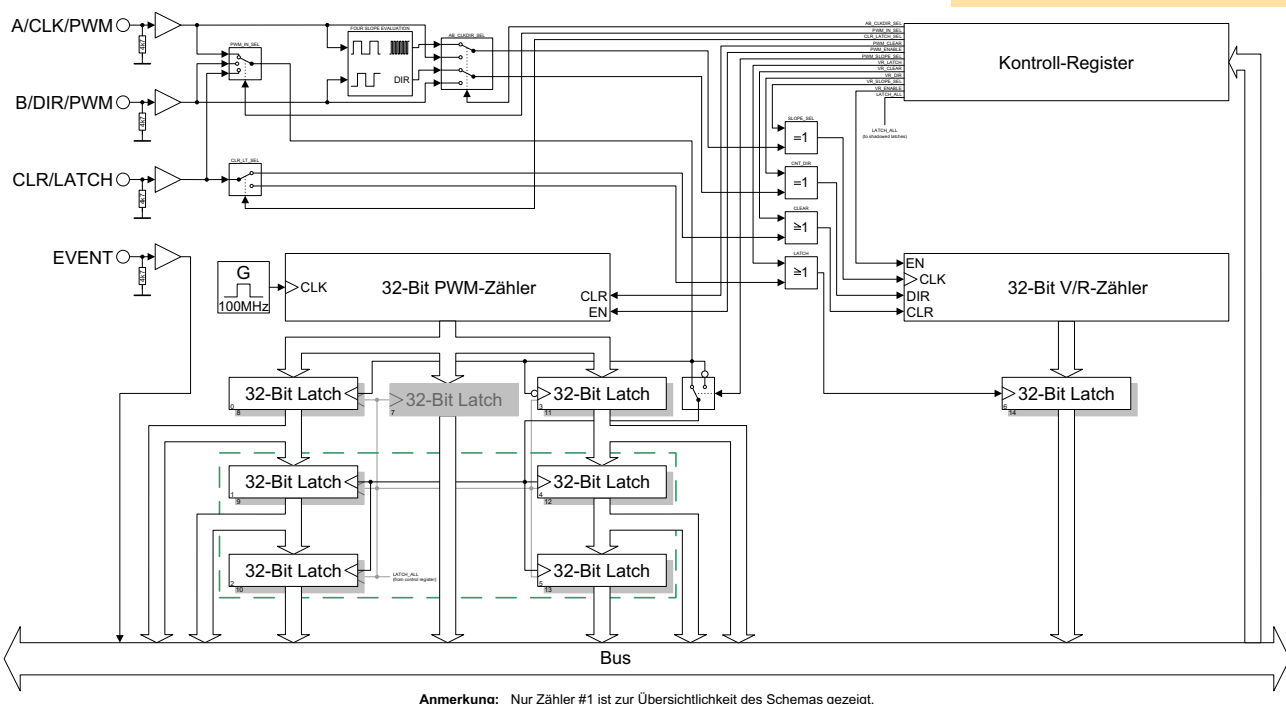


Abb. 17 – Schema des Zählerblocks

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 15.2](#).

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 100MHz ausgelöst; Näheres siehe [Kapitel 15.3 auf Seite 44](#).

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.

Eingangssignale

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

An den Eingängen A/CLK, B/DIR und CLR/LATCH sind TTL-ähnliche Signale erforderlich. Die Pinbelegungen sind auf [Seite 10](#) dargestellt.

Obwohl alle Zählereingänge einen Pull-down-Widerstand von 10kΩ 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 +5V und den Minuseingang auf GND.

Zähler programmieren

Die für den Zugriff auf die Zähler benötigten Funktionen befinden sich in der Include-Datei *ADwin-X.inc* für *ADbasic*.

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 115](#) oder in der Online-Hilfe beschrieben.

| Befehl | Funktion |
|---|---|
| Cnt_Clear | Zähler löschen. |
| Cnt_Enable Cnt_PW_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_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. |
| Sync_All | Mehrere Funktionen synchron starten. |

Abb. 18 – Befehle für einen Zählerblock

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.

Konfigurieren Sie die Zähler bitte in dieser Reihenfolge:

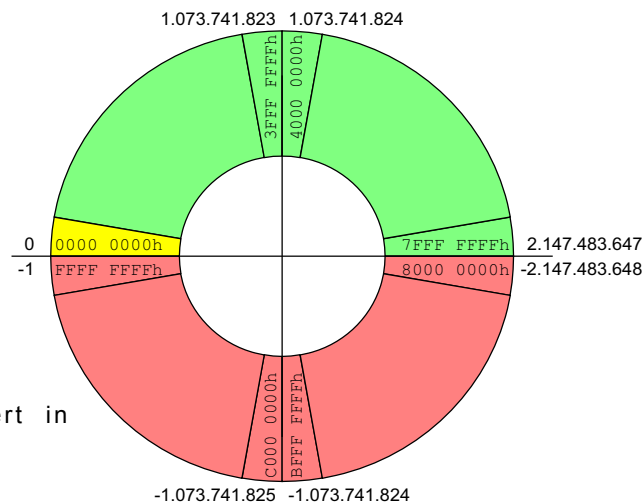
1. Gewünschten Zähler sperren (**Cnt_Enable**)
2. Betriebsmodus einstellen (**Cnt_Mode**)
3. Zähler löschen (**Cnt_Clear**)
4. Zähler freigeben (**Cnt_Enable**)

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.

15.1 Auswerten des Zählerinhalts

Die Binärzähler 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).



innen:Wert des
Binärzählers
außen:Zahlenwert in
ADbasic

Abb. 19 – Zahlenkreis als Interpretation von Zählerwerten

Beachten Sie deswegen bei der Programmierung die nachstehenden Regeln:

1. 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:
2. 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.

Berücksichtigen Sie bei der Programmierung, dass ein „Überlauf“ zwischen dem Auslesen von zwei Zählerständen - d.h. der aktuelle Zählerstand „übereignet“ den zuletzt gelesenen - nicht erfasst wird. Ein solcher Überlauf tritt bei einer Eingangsfrequenz von 100MHz nach etwas mehr als 42 Sekunden ein.

Befehlsfolge

Zahlenkreis

Zählrichtung

„Überlauf“

15.2 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).

15.2.1 Takt und Richtung

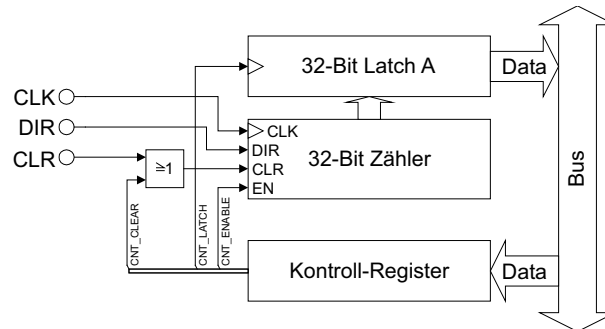


Abb. 20 – Schema CNT-Erweiterung im Modus „Takt und Richtung“

Jede positive Flanke eines Rechtecksignals auf dem CLK-Eingang (Clock) wird bis zu einer maximalen Frequenz von 20MHz 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 ADwin-X.inc
Dim val As Long

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_Enable(0001b)       'Zähler 1 starten
...

Event:
...
Cnt_Latch(0001b)        'Zähler 1 latchen
val = Cnt_Read_Latch(0001b) 'Latch-Wert lesen
```


15.2.2 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.

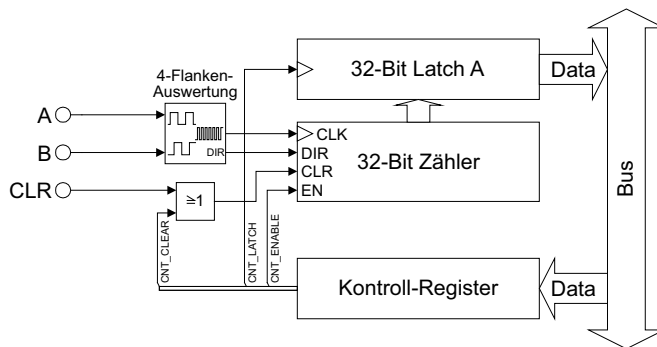


Abb. 21 – Schema CNT-Erweiterung im Modus „4-Flanken-Auswertung“

Berücksichtigen Sie bitte:

- Der Zähler registriert bei einem Zyklus des A/B-Signals 4 Flanken.
- Die maximale Zählfrequenz beträgt 20MHz. Gemeinsam mit den 4 Flanken je Zyklus ergibt sich daraus eine maximale Eingangsfrequenz von 5MHz.
- Der Abstand zwischen einer Flanke an A und einer Flanke an B darf 50ns nicht unterschreiten. Impulsbreiten oder Pausenzeiten kürzer als 100ns 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 5MHz beispielsweise bei 45 Grad auf 2,5MHz.



```
#Include ADwin-X.inc
Dim val As Long
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_Enable(0001b)       'Zähler 1 starten
...
Event:
...
Cnt_Latch(0001b)        'Zähler 1 latchen
val = Cnt_Read_Latch(0001b) 'Latch-Wert lesen
```

Programmbeispiel

Referenztaktgeber

Programmbeispiel

Ausnahme:
PWM-Register selbst
auswerten

15.3 PWM-Zähler einsetzen

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

```
#Include ADwin-X.inc
#Define frequency FPAR_1
#Define dutycycle FPAR_2
#Define hightime PAR_1
#Define lowtime PAR_2
Init:
...
Cnt_PW_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(1,01000000b)
Cnt_PW_Enable(00000001b) '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)
```

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“.

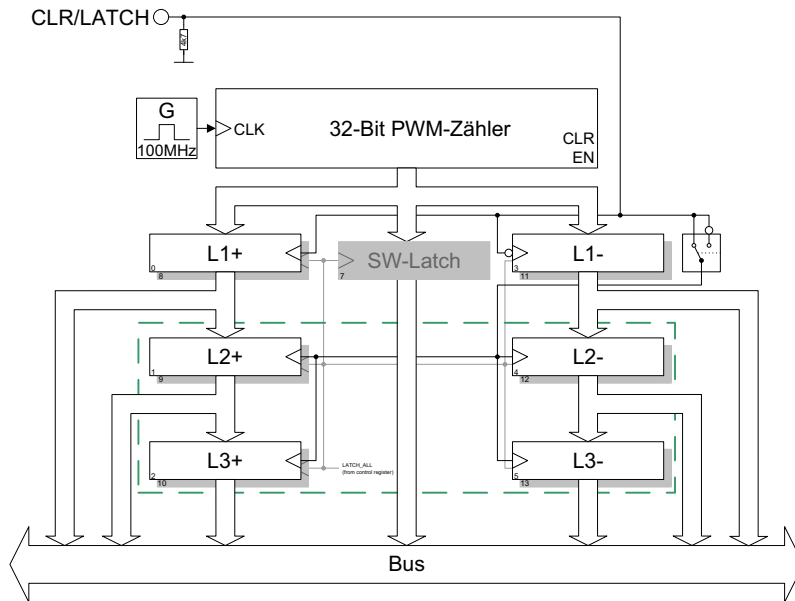
| Register | Latch | Schattenregister |
|--|-------|------------------|
| Latch 1 für positive Flanken (aktuell) | L1+ | SL1+ |
| Latch 2 für positive Flanken | L2+ | SL2+ |
| Latch 3 für positive Flanken | L3+ | SL3+ |
| Latch 1 für negative Flanken (aktuell) | L1– | SL1– |
| Latch 2 für negative Flanken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



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 | $t_H = L3- - L3+$ | $t_H = L2- - L3+$ |
| Pausen- dauer | $t_L = T - t_H = L2+ - L3-$ | $t_L = T - t_H = L3+ - L3-$ |
| Frequenz | $f = 1 / T = 1 / (L2+ - L3+)$ | $f = 1 / T = 1 / (L2- - L3-)$ |
| Tastver- hältnis | $g = t_H / T = (L3- - L3+) / (L2+ - L3+)$ | $g = t_H / T = (L2- - L3+) / (L2- - L3-)$ |

Beispiel: Auswertung der PWM-Register

16 Software

Sie programmieren *ADwin-X-A20* inklusive aller Erweiterungen mit einfachen *ADbasic*-Befehlen. Die Befehle sind im *ADbasic*-Handbuch und im folgenden beschrieben.

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

- [Allgemeine Befehle: Seite 47](#)
- [Analoge Ein- und Ausgänge: Seite 54](#)
- [Digitale Ein- und Ausgänge: Seite 78](#)
- [Zähler: Seite 114](#)
- [SSI-Schnittstelle: Seite 131](#)
- [CAN-Schnittstelle: Seite 139](#)
- [RSxxx-Schnittstelle: Seite 149](#)
- [Profibus-Schnittstelle: Seite 155](#)
- [Profinet-Schnittstelle: Seite 159](#)
- [EtherCAT-Schnittstelle: Seite 163](#)
- [LS-Bus + ADwin-X-A20: Seite 167](#)

16.1 Allgemeine Befehle

Dieser Abschnitt beschreibt allgemeine Befehle für X-A20:

- [Check_LED](#) (Seite 48)
- [Set_LED](#) (Seite 49)
- [Calc_Processdelay](#) (Seite 50)
- [CPU_Event_Config](#) (Seite 51)
- [Sync_All](#) (Seite 52)

Check_LED

Check_LED gibt den Status einer LED zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = Check_LED(led_no)
```

Parameter

| | | |
|----------------|---|------|
| led_no | Nummer (1...3) der LED. | LONG |
| ret_val | 0: LED ist aus. 1: LED leuchtet grün. 2: LED leuchtet rot. 3: LED leuchtet orange. | LONG |

Bemerkungen

Nach dem Einschalten dient LED 1 als Status-LED und leuchtet rot. Nach dem Booten läuft Prozess 15 und lässt LED 1 grün blinken. Falls erforderlich, können Sie den Prozess mit **Stop_Process** stoppen.

Siehe auch

[Set_LED](#)

Gültig für

X-A20

Beispiel

```
#Include ADwin-X.inc

Init:
If (Check_LED(2)=0) Then 'Falls LED aus ist ...
    Set_LED(2,3)          '... dann LED auf orange setzen
EndIf
```

Set_LED schaltet eine LED ein oder aus.

Syntax

```
#Include ADwin-X.inc
Set_LED(led_no, color)
```

Parameter

| | | |
|--------|---|--------|
| led_no | Nummer (1...3) der LED. | LONG |
| color | 0: LED ausschalten. 1: LED einschalten, Farbe grün. 2: LED einschalten, Farbe rot. 3: LED einschalten, Farbe orange. | __LONG |

Bemerkungen

Nach dem Einschalten leuchtet LED 1 rot. Nach dem Booten läuft Prozess 15 und lässt LED 1 grün blinken. Falls erforderlich, können Sie den Prozess mit **Stop_Process** stoppen.

Siehe auch

[Check_LED](#)

Gültig für

X-A20

Beispiel

```
#Include ADwin-X.inc
Init:
    Set_LED(2,1)                'LED 2 einschalten

Event:
Rem ...

Finish:
    Set_LED(2,0)                'LED 2 ausschalten
```

Set_LED

Calc_ Processdelay

Calc_Processdelay gibt die Anzahl der Prozesszyklen zu einer Prozessfrequenz zurück.

Syntax

```
#Include ADwin-X.Inc
```

```
ret_val = Calc_Processdelay(frequency)
```

Parameter

| | | |
|-----------|---------------------------|--------|
| frequency | Prozessfrequenz in Hertz. | __LONG |
|-----------|---------------------------|--------|

| | | |
|---------|--|------|
| ret_val | Anzahl Prozesszyklen (= Processdelay). | LONG |
|---------|--|------|

Bemerkungen

- / -

Siehe auch

Processdelay

Gültig für

X-A20

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Rem Processdelay für 150kHz einstellen
```

```
Processdelay = Calc_Processdelay(150000)
```


CPU_Event_Config konfiguriert den EVENT-Eingang.

Syntax

```
#Include ADwin-X.Inc

CPU_Event_Config(min_hold, edge, prescale)
```

Parameter

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

Bemerkungen

Am Eingang **EVENT** werden TTL-Signale erwartet.

Wenn die Eingangssignale zu häufig Störimpulse enthalten – soweit die Störimpulse nicht vermeidbar sind –, können Sie Folgendes tun:

- Parameter **min_hold** auf 1 setzen, um kurze Störimpulse auszufiltern.
- Das Eingangssignal vorher über einen Optokoppler leiten.

Siehe auch

- / -

Gültig für

X-A20

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Rem Eingang EVENT konfigurieren für
Rem Mindestzeit 15 ns, neg. Flanken, 4 Flanken
CPU_Event_Config(0,2,4)
```

Event:

```
Rem Event-gesteuerter Prozess startet jeweils, wenn 4 negative
Rem Flanken am Eingang EVENT angelegen haben.
Rem ...
```

CPU_Event_Config

Sync_All

Sync_All startet ausgewählte Aktionen synchron.

Syntax

```
#Include ADwin-X.Inc
```

```
Sync_All(pattern)
```

Parameter

pattern Bitmuster zur Auswahl von Aktionen (siehe Tabelle [LONG](#) unten), die gestartet werden:
 Bit = 0: Keine Auswirkung.
 Bit = 1: Aktion synchron starten.
 Die Bits 31:18 sind reserviert.

Bemerkungen

Die gestartete Aktion entspricht in der Regel einem Standardbefehl. Für die Aktion gelten die zuvor festgelegten Einstellungen, z.B. für Multiplexer oder Ausgabewert.

Es kommt auf die Variante von X-A20 an, welche Aktionen verfügbar sind.

| | Bit-Nr. | Aktion | entspricht |
|---------------------------|---------|---|-----------------------------------|
| Analoge Eingänge | 0 | A/D-Wandlung starten, Modus single shot. | Start_Conv |
| | 1 | A/D-Wandlung starten, Modus continuous. | Start_Conv |
| Analoge Ausgänge | 2 | D/A-Wandlung auf DAC1 und DAC2 mit den Werten der DAC-Register starten. | Start_DAC |
| Digitale Eingänge | 3 | Aktuellen Zustand der Eingänge DIO31:DIO00 in das Eingangs-Zwischenregister übertragen. | Dig_Latch (0001b) |
| | 4 | Aktuellen Zustand der Eingänge DIO63:DIO32 in das Eingangs-Zwischenregister übertragen. | Dig_Latch (0010b) |
| Digitale Ausgänge | 5 | Ausgangs-Zwischenregister auf die digitalen Ausgänge DIO31:DIO00 übertragen. | Dig_Latch (0100b) |
| | 6 | Ausgangs-Zwischenregister auf die digitalen Ausgänge DIO63:DIO32 übertragen. | Dig_Latch (1000b) |
| Flanken-ausgabe-FIFO | 7 | Flankenausgabe am Ausgangs-FIFO 1 starten. | Digout_Fifo_Start |
| | 8 | Flankenausgabe am Ausgangs-FIFO 2 starten. | Digout_Fifo_Start |
| | 9 | Flankenausgabe am Ausgangs-FIFO 1 stoppen und FIFO 1 löschen. | Digout_Fifo_Clear |
| | 10 | Flankenausgabe am Ausgangs-FIFO 2 stoppen und FIFO 2 löschen. | Digout_Fifo_Clear |
| Flanken-überwachungs-FIFO | 11 | Eingangs-FIFO 1 zur Flankenüberwachung löschen. | Digin_Fifo_Clear |
| | 12 | Eingangs-FIFO 2 zur Flankenüberwachung löschen. | Digin_Fifo_Clear |
| | 13 | Referenzzähler des Eingangs-FIFO 1 löschen. | - / - |
| | 14 | Referenzzähler des Eingangs-FIFO 2 löschen. | - / - |
| Zähler | 15 | Zähler 1...7 auf Null setzen. | Cnt_Clear |
| Sync | 16 | Inhalte aller Zähler und PWM-Zähler in Zwischenspeicher kopieren. | Cnt_Sync_Latch |

| | Bit-Nr. | Aktion | entspricht |
|-----|---------|---|------------|
| SSI | 17 | Auslesen des SSI-Decoders starten (nur einmalig). | SSI_Start |

Bits 0 und 1 dürfen nicht gleichzeitig gesetzt werden. Sie müssen vorher **Start_Conv** im Modus Einzelmessung ausführen, um die verwendeten Kanäle und ggf. die Verstärkung festzulegen.

Die Bits 13 und 14 setzen die Zähler zurück, die die Zeitstempel bei der Flankenüberwachung der Eingangs-FIFOs erzeugen; siehe auch [Digin_Fifo_Read_Timer](#).

Siehe auch

[Start_Conv](#), [Start_DAC](#), [Dig_Latch](#), [Digout_Fifo_Start](#), [Digout_Fifo_Clear](#), [Digin_Fifo_Clear](#), [Cnt_Clear](#), [Cnt_Sync_Latch](#), [SSI_Start](#)

Gültig für

X-A20

Beispiel

```
#Include ADwin-X.Inc
Dim i As Long
```

Init:

```
Write_DAC(1,3500)           'initialize DAC 1
Write_DAC(2,65535)          'initialize DAC 2
REM Set channels DIO15:DIO00 as outputs, DIO31:DIO16 as inputs
Conf_DIO(0011b)
Digout_Write_Latch1(0)      'Set all output bits to 0
i=1                          'initialize index
Rem initialize A/D conversion for Sync_All:
Rem ADC 1, gain 1, single shot (!)
Start_Conv(1b, 0, 1)
Wait_EOC()                  'wait for end of conversion
```

Event:

```
Rem start ADC (single shot), both DAC, latch digital channels
Rem DIO31:0 synchronously
Sync_All(101101b)
Wait_EOC()                  'Wait for end of conversion

Rem Read ADC
Par_1 = Read_ADC(1)
Write_DAC(1,Par_1)          'set DAC 1
Write_DAC(2,Par_1 * 3.5)    'set DAC 2

Par_2 = Digin_Read_Latch1() 'read input bits and ...
Digout_Write_Latch1(Par_1) 'output in next event cycle

If (i=1000) Then End        'End process after 1000 repetitions
Inc(i)                      'Increment index
```

16.2 Analoge Ein- und Ausgänge

Dieser Abschnitt beschreibt folgende Befehle:

- [DAC](#) (Seite 55)
- [DAC12](#) (Seite 56)
- [Start_DAC](#) (Seite 57)
- [Write_DAC](#) (Seite 58)
- [ADC](#) (Seite 59)
- [ADC24](#) (Seite 60)
- [ADC2](#) (Seite 62)
- [ADC4](#) (Seite 63)
- [ADC8](#) (Seite 64)
- [ADC2_24](#) (Seite 65)
- [ADC4_24](#) (Seite 66)
- [ADC8_24](#) (Seite 67)
- [Read_ADC](#) (Seite 68)
- [Read_ADC24](#) (Seite 69)
- [Read_ADC_Packed](#) (Seite 70)
- [Read_ADC8](#) (Seite 71)
- [Read_ADC8_24](#) (Seite 72)
- [Start_Conv](#) (Seite 73)
- [Start_Conv_PGA](#) (Seite 75)
- [Wait_EOC](#) (Seite 77)

DAC gibt eine definierte Spannung auf einem analogen 16 Bit-Ausgang aus.

Syntax

```
#Include ADwin-X.inc
```

```
DAC (dac_no, value)
```

Parameter

| | | |
|---------------------|--|------|
| <code>dac_no</code> | Nummer (1...2) des analogen 16 Bit-Ausgangs. | LONG |
| <code>value</code> | Wert in Digits (0...65535), der die auszugebende Spannung definiert. | LONG |

Beschreibung

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

Die Wandlungszeit beträgt 1 µs.

Der Spannungsbereich ist -10V...+10V = 20V. Mit der folgenden Formel berechnen Sie die ausgegebene Spannung zu einem Digitalwert:

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

Siehe auch

[DAC12](#), [ADC](#), [Start_DAC](#), [Write_DAC](#)

Gültig für

X-A20M1, X-A20F

Beispiel

```
#Include ADwin-X.inc
Rem Digitaler P-Regler
#Define set_to Par_1      'Sollwert
#Define gain Par_2        'Verstärkungsfaktor
#Define diff Par_3        'Regelabweichung
#Define out Par_4         'Stellgröße

Init:
    Processdelay = 10000

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

DAC

DAC12

DAC12 gibt eine definierte Spannung auf einem analogen 12 Bit-Ausgang aus.

Syntax

```
#Include ADwin-X.inc
```

```
DAC12 (dac_no, value)
```

Parameter

| | | |
|---------------------|--|------|
| <code>dac_no</code> | Nummer (1...2) des analogen 12 Bit-Ausgangs. | LONG |
| <code>value</code> | Wert in Digits (0, 16, ...65520), der die auszugebende Spannung definiert. | LONG |

Beschreibung

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

Die Wandlungszeit liegt im Bereich von 500...1000µs.

Wenn der DAC als Geber für das Komparatorsignal verwendet wird (siehe unten), empfehlen wir, die Spannung im Abschnitt `Init:` oder `LowInit:` zu setzen und anschließend mit `IO_Sleep` so lange zu warten, bis die Spannung sicher ausgegeben wird.

Die Bits 15:4 von `value` werden als Digitwert verarbeitet, die Bits 3:0 werden ignoriert und zu Null gesetzt.

| Bitnr. | 31:24 | 15:4 | 3:0 |
|--------|-------|-------------|-----|
| Inhalt | 0 | 12 Bit-Wert | – |

Der Spannungsbereich ist $-10V \dots +10V = 20V$. Mit der folgenden Formel berechnen Sie die ausgegebene Spannung zu einem Digitalwert:

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

Die Ausgänge der DAC sind intern mit den Komparatoreingängen verbunden. Die eingestellte DAC-Spannung dient als Komparatorsignal, d.h. ein anliegendes Digitalsignal mit einer kleineren Spannung wird als Pegel Low verarbeitet, mit einer höheren Spannung als Pegel High.

Das Komparatorsignal muss im Bereich 0...5 Volt liegen, damit der Komparator korrekt arbeitet.

Siehe auch

[DAC](#), [ADC](#), [Start_DAC](#), [Write_DAC](#), [IO_Sleep](#)

Gültig für

X-A20M1, X-A20F

Beispiel

```
#Include ADwin-X.inc
```

```
Rem example for the use of comparator inputs
```

Init:

```
DAC12(1, 42598)           'set +3V comp. level. channels 1..8
DAC12(2, 39321)           'set +2V comp. level, channels 9..12
IO_Sleep(100000)         'wait 1 ms
```

Event:

```
Rem use comparator inputs
```

Start_DAC startet die Wandlung bzw. Ausgabe aller 16 Bit-DAC.

Syntax

```
#Include ADwin-X.inc
Start_DAC()
```

Parameter

- / -

Bemerkungen

Der Wert im Ausgaberegister eines 16 Bit-DAC wird mit **Write_DAC** gesetzt.

Sie können Wandlung auch mit **Sync_All** starten.

Siehe auch

[DAC](#), [DAC12](#), [Write_DAC](#), [Sync_All](#)

Gültig für

X-A20M1, X-A20F

Beispiel

REM Simultane Ausgabe von verschiedenen Signalverläufen

REM auf den Ausgängen DAC 1 und 2.

```
#Include ADwin-X.inc
```

```
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
```

Start_DAC

Write_DAC

Write_DAC schreibt einen Digitalwert in das Ausgaberegister eines 16 Bit-DAC.

Syntax

```
#Include ADwin-X.inc

Write_DAC(dac_no,value)
```

Parameter

| | | |
|---------------|--|------|
| dac_no | Nummer (1...2) des analogen 16 Bit-Ausgangs. | 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.

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

Siehe auch

[DAC](#), [DAC12](#), [Start_DAC](#)

Gültig für

X-A20M1, X-A20F

Beispiel

REM Simultane Ausgabe von verschiedenen Signalverläufen
REM auf den DAC 1 und 2.
REM Die Signalverläufe sind in zwei DATA-Feldern abgelegt und
REM können vor dem Programmstart vom PC übergeben werden.

```
#Include ADwin-X.inc
Dim i As Long 'Deklaration
Dim Data_1[1000], Data_2[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

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
    INC(i)
    IF (i>1000) Then i=1
```


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

Syntax

```
#Include ADwin-X.inc

ret_val = ADC(channel)
```

Parameter

| | | |
|---------|---|------|
| channel | Nummer (1...8) des analogen Eingangskanals. | LONG |
| ret_val | Messergebnis in Digits (0...65535). | LONG |

Bemerkungen

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

ADC ist eine Zusammenstellung aufeinander folgender Funktionen:

- **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.

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

- Sehr kurze Zykluszeiten: **Processdelay** < 240 (s.o.).
- 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 157).

Der Messbereich ist -10V...+10V = 20V, die Verstärkung ist 1. Mit der folgenden Formel berechnen Sie aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Die Wandlungszeit beträgt 5µs bei X-A20M1 (inklusive Einschwingzeit des Multiplexers) und 1,25µs bei X-A20F.

Bei X-A20F gilt: Wenn Sie mit einem Wandlungsbefehl (ADC... / Start_Conv) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Beispiel: Der Wechsel von **ADC8** zu **ADC** dauert 5µs + 1,25µs; der Wechsel von **ADC(3)** zu **ADC(2)** dauert 1,25µs + 1,25µs.

Siehe auch

[ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20M1, X-A20F

Beispiel

```
#Include ADwin-X.inc
#define in_channel 1          'Eingangskanal
#define in_value Par_1
```

Event:

```
Rem Analogen Eingang messen
in_value = ADC(in_channel) * 10900
```

ADC

ADC24

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 ADwin-X.Inc

ret_val = ADC24(channel)
```

Parameter

| | | |
|---------|---|------|
| channel | Nummer (1...8) 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 sowie 31:24 sind immer Null.

| Bitnr. | 31:24 | 23:16 | 15:6 | 5:0 |
|--------|-------|----------------------------------|------|-----|
| Inhalt | 0 | 18-Bit Messwert in den Bits 32:6 | | 0 |

ADC24 ist eine Zusammenstellung aufeinander folgender Funktionen:

- **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.

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

In folgenden Fällen sollten Sie anstelle der Anweisung **ADC24** die Anweisungen **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 157).

Der Messbereich ist 20V (Eingangsspannung: -10V ... +10V). Mit der folgenden Formel berechnen Sie aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Die Wandlungszeit beträgt 5µs bei X-A20M1 (inklusive Einschwingzeit des Multiplexers) und 1,25µs bei X-A20F.

Bei X-A20F gilt: Wenn Sie mit einem Wandlungsbefehl (**ADC...** / **Start_Conv**) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Beispiel: Der Wechsel von **ADC8** zu **ADC24** dauert 5µs + 1,25µs; der Wechsel von **ADC24** (3) zu **ADC24** (2) dauert 1,25µs + 1,25µs.

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20M1, X-A20F

Beispiel

```
#Include ADwin-X.Inc
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
```

ADC2

ADC2 misst die Spannungen an 2 analogen Eingängen und gibt die Messwerte (16 Bit) in einem Feld zurück.

Syntax

```
#Include ADwin-X.inc
```

```
ADC2 (array [], array_idx, channel_group)
```

Parameter

| | | |
|----------------------------|---|----------------------|
| <code>array []</code> | Feld, das die Messwerte der zwei gewählten Eingangs-kanäle aufnimmt. | ARRAY LONG |
| <code>array_idx</code> | Feldelement, das den ersten der Messwerte aufnimmt. | LONG |
| <code>channel_group</code> | Gewählte Kanalgruppe: 0: Kanäle 1 und 2. 1: Kanäle 3 und 4. 2: Kanäle 5 und 6. 3: Kanäle 7 und 8. | LONG |

Bemerkungen

Der Messbereich ist $-10V \dots +10V = 20V$, die Verstärkung ist 1. Mit der folgenden Formel berechnen Sie aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Die Wandlungszeit (für beide Kanäle zusammen) beträgt $1,82\mu s$.

Wenn Sie mit einem Wandlungsbefehl (ADC... / Start_Conv) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Beispiel: Der Wechsel von **ADC8** zu **ADC2** dauert $5\mu s + 1,82\mu s$; der Wechsel von **ADC2 (0)** zu **ADC2 (3)** dauert $1,82\mu s + 1,82\mu s$.

Siehe auch

[ADC](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
```

```
#Define in_array Data_1
```

```
Dim in_array[2000] As Long
```

```
Dim array_idx As Long
```

Init:

```
array_idx = 1
```

Event:

```
Rem Eingänge 3..4 messen
```

```
ADC2 (in_array, array_idx, 1)
```

```
array_idx = array_idx + 2
```

```
If (array_idx > 1992) Then array_idx = 1
```

ADC4 misst die Spannungen an 4 analogen Eingängen und gibt die Messwerte (16 Bit) in einem Feld zurück.

Syntax

```
#Include ADwin-X.inc

ADC4(array[], array_idx, channel_group)
```

Parameter

| | | |
|----------------------------|---|---------------|
| <code>array[]</code> | Feld, das die Messwerte der vier gewählten Eingangskanäle aufnimmt. | ARRAY LONG |
| <code>array_idx</code> | Feldelement, das den ersten der Messwerte aufnimmt. | LONG |
| <code>channel_group</code> | Gewählte Kanalgruppe: 0: Kanäle 1...4. 1: Kanäle 5...8. | LONG |

Bemerkungen

Der Messbereich ist $-10V \dots +10V = 20V$, die Verstärkung ist 1. Mit der folgenden Formel berechnen Sie aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Die Wandlungszeit (für alle Kanäle zusammen) beträgt $2,86\mu s$.

Wenn Sie mit einem Wandlungsbefehl (ADC... / Start_Conv) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Beispiel: Der Wechsel von **ADC8** zu **ADC4** dauert $5\mu s + 2,86\mu s$; der Wechsel von **ADC4 (0)** zu **ADC4 (1)** dauert $2,86\mu s + 2,86\mu s$.

Siehe auch

[ADC](#), [ADC2](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
#Define in_array Data_1

Dim in_array[2000] As Long
Dim array_idx As Long

Init:
    array_idx = 1

Event:
    Rem Eingänge 1..4 messen
    ADC4(in_array, array_idx, 0)
    array_idx = array_idx + 4
    If (array_idx > 1992) Then array_idx = 1
```

ADC4

ADC8

ADC8 misst die Spannung an den analogen Eingängen 1...8 und gibt die Messwerte (16 Bit) in einem Feld zurück.

Syntax

```
#Include ADwin-X.inc

ADC8(array[], array_idx)
```

Parameter

| | | |
|------------------------|--|----------------------|
| <code>array[]</code> | Feld, das die Messwerte der Eingangskanäle 1...4 aufnimmt. | ARRAY LONG |
| <code>array_idx</code> | Feldelement, das den ersten der Messwerte aufnimmt. | LONG |

Bemerkungen

Der Messbereich ist $-10V \dots +10V = 20V$, die Verstärkung ist 1. Mit der folgenden Formel berechnen Sie aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Die Wandlungszeit (für alle Kanäle zusammen) beträgt 5µs.

Wenn Sie mit einem Wandlungsbefehl (ADC... / Start_Conv) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Beispiel: Der Wechsel von **ADC** zu **ADC8** dauert 1,25µs + 5µs.

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
#Define in_array Data_1

Dim in_array[2000] As Long
Dim array_idx As Long

Init:
    array_idx = 1

Event:
    Rem Eingänge 1..8 messen
    ADC8(in_array, array_idx)
    array_idx = array_idx + 8
    If (array_idx > 1992) Then array_idx = 1
```

ADC2_24 misst die Spannungen an 2 analogen Eingängen und gibt die Messwerte (18 Bit) in einem Feld zurück. Die Rückgabewerte sind auf 24 Bit normiert.

Syntax

```
#Include ADwin-X.inc
```

```
ADC2_24(array[], array_idx, channel_group)
```

Parameter

| | | |
|----------------------|---|---------------|
| array [] | Feld, das die Messwerte der Eingangskanäle 1...4 aufnimmt. | ARRAY LONG |
| array_idx | Feldelement, das den ersten der Messwerte aufnimmt. | LONG |
| channel_group | Gewählte Kanalgruppe: 0: Kanäle 1 und 2. 1: Kanäle 3 und 4. 2: Kanäle 5 und 6. 3: Kanäle 7 und 8. | LONG |

Bemerkungen

Der Messwerte enthalten in den Bits 23:6 den 18 Bit-Messwert, die Bits 5:0 sowie 31:24 sind immer Null.

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

Der Messbereich ist $-10V \dots +10V = 20V$, die Verstärkung ist 1. Mit der folgenden Formel berechnen Sie aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Die Wandlungszeit (für beide Kanäle zusammen) beträgt $1,82\mu s$.

Wenn Sie mit einem Wandlungsbefehl (ADC... / Start_Conv) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Beispiel: Der Wechsel von **ADC8** zu **ADC2_24** dauert $5\mu s + 1,82\mu s$; der Wechsel von **ADC2_24** (0) zu **ADC2_24** (3) dauert $1,82\mu s + 1,82\mu s$.

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
#Define in_array Data_1
```

```
Dim in_array[2000] As Long
Dim array_idx As Long
```

```
Init:
    array_idx = 1
```

```
Event:
    Rem Eingänge 3..4 messen
    ADC2_24(in_array, array_idx, 1)
    array_idx = array_idx + 2
    If (array_idx > 1992) Then array_idx = 1
```

ADC2_24

ADC4_24

ADC4_24 misst die Spannungen an 4 analogen Eingängen und gibt die Messwerte (18 Bit) in einem Feld zurück. Die Rückgabewerte sind auf 24 Bit normiert.

Syntax

```
#Include ADwin-X.inc
```

```
ADC4_24(array[], array_idx, channel_group)
```

Parameter

| | | |
|----------------------|---|----------------------|
| array [] | Feld, das die Messwerte der Eingangskanäle 1...4 aufnimmt. | ARRAY LONG |
| array_idx | Feldelement, das den ersten der Messwerte aufnimmt. | LONG |
| channel_group | Gewählte Kanalgruppe: 0: Kanäle 1...4. 1: Kanäle 5...8. | LONG |

Bemerkungen

Der Messwerte enthalten in den Bits 23:6 den 18 Bit-Messwert, die Bits 5:0 sowie 31:24 sind immer Null.

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

Der Messbereich ist $-10V \dots +10V = 20V$, die Verstärkung ist 1. Mit der folgenden Formel berechnen Sie aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Die Wandlungszeit (für alle Kanäle zusammen) beträgt 2,86µs.

Wenn Sie mit einem Wandlungsbefehl (ADC... / Start_Conv) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Beispiel: Der Wechsel von **ADC8** zu **ADC4_24** dauert 5µs + 2,86µs; der Wechsel von **ADC4_24** (0) zu **ADC4_24** (1) dauert 2,86µs + 2,86µs.

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
#Define in_array Data_1
```

```
Dim in_array[2000] As Long
Dim array_idx As Long
```

Init:

```
array_idx = 1
```

Event:

```
Rem Eingänge 1..4 messen
```

```
ADC4_24(in_array, array_idx, 0)
```

```
array_idx = array_idx + 4
```

```
If (array_idx > 1992) Then array_idx = 1
```


ADC8_24 misst die Spannungen an den analogen Eingängen 1...8 und gibt die Messwerte (18 Bit) in einem Feld zurück. Die Rückgabewerte sind auf 24 Bit normiert.

Syntax

```
#Include ADwin-X.inc
ADC8_24(array[], array_idx)
```

Parameter

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

Bemerkungen

Der Messwerte enthalten in den Bits 23:6 den 18 Bit-Messwert, die Bits 5:0 sowie 31:24 sind immer Null.

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

Der Messbereich ist $-10V...+10V = 20V$, die Verstärkung ist 1. Mit der folgenden Formel berechnen Sie aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Die Wandlungszeit (für alle Kanäle zusammen) beträgt 5µs.

Wenn Sie mit einem Wandlungsbefehl (ADC... / Start_Conv) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Beispiel: Der Wechsel von **ADC** zu **ADC8_24** dauert 1,25µs + 5µs.

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
#Define in_array Data_1

Dim in_array[2000] As Long
Dim array_idx As Long

Init:
    array_idx = 1

Event:
    Rem Eingänge 1..8 messen
    ADC8_24(in_array, array_idx)
    array_idx = array_idx + 8
    If (array_idx > 1992) Then array_idx = 1
```

ADC8_24

Read_ADC

Read_ADC gibt einen gewandelten Wert eines analogen Eingangs mit 16 Bit Auflösung zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = Read_ADC(channel)
```

Parameter

| | | |
|---------|---|------|
| channel | Nummer (1...8) des analogen Eingangskanals. | LONG |
| ret_val | Messergebnis in Digits (0...65535). | LONG |

Bemerkungen

Read_ADC24 gibt einen Analogwert mit 24 Bit Auflösung zurück.

Für den Messbereich $-10V \dots +10V = 20V$ und die Verstärkung 1 berechnen Sie mit der folgenden Formel aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20M1, X-A20F

Beispiel

```
#Include ADwin-X.inc
Init:
    Rem ADC-Wandlung auf Kanal 3 starten, Verstärkung 2, Einzelwert
    Start_Conv(100b, 0, 1)

Event:
    Wait_EOC()                'Ende der Wandlung abwarten
    Par_1 = Read_ADC(3)        'Wert von Kanal 3 einlesen
    Rem ADC-Einzelwandlung auf Kanal 3 starten
    Start_Conv(100b, 0, 1)
```

Read_ADC24 gibt einen gewandelten Wert eines analogen Eingangs mit 24 Bit Auflösung zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = Read_ADC24(channel)
```

Parameter

| | | |
|---------|---|------|
| channel | Nummer (1...8) des analogen Eingangskanals. | LONG |
| ret_val | Messergebnis in Digits (0...16777215 = $2^{24}-1$). | LONG |

Bemerkungen

Read_ADC gibt einen Analogwert mit 18 Bit Auflösung zurück.

Der Messwerte enthalten in den Bits 23:6 den 18 Bit-Messwert, die Bits 5:0 sowie 31:24 sind immer Null.

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

Für den Messbereich $-10V \dots +10V = 20V$ und die Verstärkung 1 berechnen Sie mit der folgenden Formel aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20M1, X-A20F

Beispiel

```
#Include ADwin-X.inc
Init:
    Rem ADC-Wandlung auf Kanal 5 starten, Verstärkung 2, Einzelwert
    Start_Conv(10000b,1,1)

Event:
    Wait_EOC()                'Ende der Wandlung abwarten
    Par_1 = Read_ADC24(5)      'Wert von Kanal 5 einlesen
    Rem ADC-Einzelwandlung auf Kanal 5 starten
    Start_Conv(10000b,1,1)
```

Read_ADC24

Read_ADC_Packed

Read_ADC_Packed gibt die gewandelten Werte von zwei analogen Eingängen in gepackter Form zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = Read_ADC_Packed(ch_pair)
```

Parameter

| | | |
|----------------|---|------|
| ch_pair | Zahl (1...3) zur Festlegung eines Kanalpaars: 1: Kanäle 1 und 2. 2: Kanäle 3 und 4. 3: Kanäle 5 und 6. 4: Kanäle 7 und 8. | LONG |
| ret_val | 32 Bit-Wert, der 2 Messwerte zu je 16 Bit enthält: Bits 15:0: Messwert des Kanals n. Bits 31:16: Messwert des Kanals n+1. | LONG |

Bemerkungen

Für den Messbereich $-10V \dots +10V = 20V$ und die Verstärkung 1 berechnen Sie mit der folgenden Formel aus einem zurückgegebenen Digitalwert die gemessene Spannung:

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

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
Init:
    Rem Wandlung auf Kanälen 3+4 starten
    Start_Conv(1100b, 0, 1)

Event:
    Wait_EOC() 'Ende der Wandlung abwarten
    Par_1 = Read_ADC_Packed(2) 'Wert von Kanälen 3+4 lesen
    Par_3 = Par_1 And 0FFFFh 'Wert Kanal 3
    Par_4 = Shift_Right(Par_1, 16) And 0FFFFh 'Wert Kanal 4
    Rem Neue Wandlung auf Kanälen 3+4 starten
    Start_Conv(1100b, 0, 1)
```

Read_ADC8 gibt gewandelte Werte mit 16 Bit Auflösung von den A/D-Wandlern 1...8 in einem Feld zurück.

Syntax

```
#Include ADwin-X.inc

Read_ADC8(array[], array_idx)
```

Parameter

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

Bemerkungen

Für den Messbereich $-10V \dots +10V = 20V$ und die Verstärkung 1 berechnen Sie mit der folgenden Formel aus dem zurückgegebenen Digitalwert die gemessene Spannung:

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

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
Dim Data_1[2000] As Long
Dim data_idx As Long
```

Init:

```
Rem ADC - Einzelwandlung auf Kanälen 1..8 starten
Start_Conv(0FFh, 0, 1)
data_idx = 1
```

Event:

```
Wait_EOC() 'Ende der Wandlung abwarten
Read_ADC8(Data_1, data_idx) 'Werte einlesen
data_idx = data_idx + 8
If (data_idx > 1992) Then data_idx = 1
Rem Einzelwandlung erneut starten
Start_Conv(0FFh, 0, 1)
```

Read_ADC8

Read_ADC8_24

Read_ADC8_24 gibt 8 gewandelte Werte mit 24 Bit Auflösung von den A/D-Wandlern 1...8 in einem Feld zurück.

Syntax

```
#Include ADwin-X.inc

Read_ADC8_24(array[], array_idx)
```

Parameter

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

Bemerkungen

Für den Messbereich $-10V \dots +10V = 20V$ und die Verstärkung 1 berechnen Sie mit der folgenden Formel aus einem zurückgegebenen Digitalwert die gemessene Spannung:

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

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Start_Conv](#), [Start_Conv_PGA](#), [Wait_EOC](#)

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
Dim Data_1[2000] As Long
Dim data_idx As Long

Init:
    Rem ADC - Einzelwandlung auf Kanälen 1..8 starten
    Start_Conv(0FFh, 0, 1)
    data_idx = 1

Event:
    Wait_EOC() 'Ende der Wandlung abwarten
    Read_ADC8_24(Data_1, data_idx) 'Werte einlesen
    data_idx = data_idx + 8
    If (data_idx > 1992) Then data_idx = 1
    Rem Einzelwandlung erneut starten
    Start_Conv(0FFh, 0, 1)
```

Start_Conv startet die Wandlung an ausgewählten A/D-Wandlern für eine Einzelmessung oder für dauerhafte Messungen.

Syntax

```
#Include ADwin-X.inc

Start_Conv(adc_pattern, gain, mode)
```

Parameter

| | | |
|--------------------|--|------|
| adc_pattern | Bitmuster, das den oder die zu startenden A/D-Wandler festlegt (nur Bits 0...7 verwendbar, siehe Tabelle): Bit=1: Wandler starten. Bit=0: Wandler nicht starten. | LONG |
| gain | Verstärkungsfaktor: 0: Faktor = 1, Spannungsbereich -10V...+10V. 1: Faktor = 2, Spannungsbereich -5V...+5V. | LONG |
| mode | Arbeitsmodus der Wandlung: 1: Einzelmessung. 2: Modus „continuous“, regelmäßiger Messzyklus. | LONG |

| Bitnr. in | 31:8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------------------|------|---|---|---|---|---|---|---|---|
| adc_pattern | | | | | | | | | |
| ADC-Nummer | – | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Beschreibung

Bei X-A20M1 darf in **adc_pattern** nur ein einzelnes Bit gesetzt sein; der Parameter **gain** wird ignoriert und der Verstärkungsfaktor immer auf 1 ($\pm 10V$) gesetzt. Die Wandlungszeit (inklusive Einschwingzeit des Multiplexers) beträgt 5 μ s.

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

Die Wandlungszeit (für alle Kanäle zusammen) ist abhängig von der Anzahl der gemessenen Kanäle:

- 1 Kanal: max. 800kHz = 1,25 μ s.
- 2 Kanäle: max. 550kHz = 1,82 μ s.
- 3 Kanäle: max. 425kHz = 2,35 μ s.
- 4 Kanäle: max. 350kHz = 2,86 μ s.
- 5 Kanäle: max. 300kHz = 3,3 μ s.
- 6 Kanäle: max. 250kHz = 4,0 μ s.
- 7 Kanäle: max. 225kHz = 4,44 μ s.
- 8 Kanäle: max. 200kHz = 5 μ s.

Bei X-A20F gilt: Wenn Sie mit einem Wandlungsbefehl (ADC... / Start_Conv) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen.

Beispiel: Der Wechsel von **ADC4** zu **Start_Conv(111b, ...)** dauert 2,86 μ s + 2,35 μ s; der Wechsel von **Start_Conv(11b, ...)** zu **Start_Conv(101b, ...)** dauert 1,82 μ s + 1,82 μ s.

Mit **Start_Conv_PGA** (nur X-A20F) können Sie den Verstärkungsfaktor für jeden ADC unterschiedlich setzen.

Beim Arbeitsmodus „continuous“ wandeln die ausgewählten ADC kontinuierlich Messwerte. Nach dem Start der kontinuierlichen Wandlung muss mit **Wait_EOC** zunächst einmalig das Ende der ersten Wandlung geprüft werden. Mit den Befehlen **Read_ADC...** können anschließend die jeweils neuesten Messwerte gelesen werden.

Alle **ADC**-Befehle stellen den Arbeitsmodus Einzelmessung ein und beenden damit einen regelmäßigen Messzyklus.

Sie können die Wandlung auch mit **Sync_All** starten.

Start_Conv

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv_PGA](#), [Wait_EOC](#), [Sync_All](#)

Gültig für

X-A20M1, X-A20F

Beispiel

```
#Include ADwin-X.inc
Dim Data_3[2000] As Long
Dim data_idx As Long
```

Init:

```
Rem Dauermessung mit ADC 3 starten, Verstärkungsfaktor 1
Start_Conv(00000100b, 0, 2)
Rem Einmalig Ende der Wandlung abwarten
Wait_EOC()
data_idx = 1
```

Event:

```
Data_3[data_idx] = Read_ADC(3)
data_idx = data_idx + 1
If (data_idx > 1992) Then data_idx = 1
Rem Konvertierung arbeitet automatisch weiter
```


Start_Conv_PGA startet die Wandlung an ausgewählten A/D-Wandlern mit separatem Verstärkungsfaktor für eine Einzelmessung oder für dauerhafte Messungen.

Syntax

```
#Include ADwin-X.inc

Start_Conv_PGA(adc_pattern, gain_pattern, mode)
```

Parameter

| | | |
|---------------------|---|------|
| adc_pattern | Bitmuster, das die zu startenden A/D-Wandler festlegt (nur Bits 7:0 verwendbar, siehe Tabelle): Bit=1: Wandler starten. Bit=0: Wandler nicht starten. | LONG |
| gain_pattern | Bitmuster, das die Verstärkungsfaktoren für alle A/D-Wandler festlegt. Je 2 Bits legen den Faktor für einen ADC fest: 00b: Faktor = 1, Spannungsbereich -10V...+10V. 01b: Faktor = 2, Spannungsbereich -5V...+5V. 10b, 11b: reserviert | LONG |
| mode | Arbeitsmodus der Wandlung: 1: Einzelmessung. 2: Modus „continuous“, regelmäßiger Messzyklus. | LONG |

| | | | | | | | | | |
|---------------------------------|------|---|---|---|---|---|---|---|---|
| Bitnr. in adc_pattern | 31:8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ADC-Nummer | – | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

| | | | | | | | | | |
|----------------------------------|------|-------|-------|-------|-----|-----|-----|-----|-----|
| Bitnr. in gain_pattern | 31:8 | 15:14 | 13:12 | 11:10 | 9:8 | 7:6 | 5:4 | 3:2 | 1:0 |
| ADC-Nummer | – | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Beschreibung

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

Die Wandlungszeit (für alle Kanäle zusammen) ist abhängig von der Anzahl der gemessenen Kanäle:

- 1 Kanal: max. 800kHz = 1,25µs.
- 2 Kanäle: max. 550kHz = 1,82µs.
- 3 Kanäle: max. 425kHz = 2,35µs.
- 4 Kanäle: max. 350kHz = 2,86µs.
- 5 Kanäle: max. 300kHz = 3,3µs.
- 6 Kanäle: max. 250kHz = 4,0µs.
- 7 Kanäle: max. 225kHz = 4,44µs.
- 8 Kanäle: max. 200kHz = 5µs.

Wenn Sie mit einem Wandlungsbefehl (ADC... / Start_Conv) andere Kanäle wählen als mit dem vorherigen Wandlungsbefehl, ist die Wandlungszeit verlängert: Bei der Kanaländerung wird die Wandlung (einmalig) doppelt ausgeführt, nämlich einmal mit den vorher gewählten Kanälen und einmal mit den neu gewählten Kanälen. Das gleiche geschieht, wenn Sie **gain_pattern** ändern.

Beispiel: Der Wechsel von **ADC4** zu **Start_Conv_PGA(111b, ...)** dauert 2,86µs + 2,35µs; der Wechsel von **Start_Conv_PGA(11b, 0101b, ...)** zu **Start_Conv_PGA(11b, 0100b, ...)** dauert 1,82µs + 1,82µs.

Beim Arbeitsmodus „continuous“ wandeln die ausgewählten ADC kontinuierlich Messwerte. Nach dem Start der kontinuierlichen Wandlung muss mit **Wait_EOC** zunächst einmalig das Ende der ersten Wandlung geprüft werden. Mit den Befehlen **Read_ADC...** können anschließend die jeweils neuesten Messwerte gelesen werden.

Alle **ADC**-Befehle stellen den Arbeitsmodus Einzelmessung ein und beenden damit einen regelmäßigen Messzyklus.

Siehe auch

Start_Conv_PGA

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Wait_EOC

Gültig für

X-A20F

Beispiel

```
#Include ADwin-X.inc
Dim Data_3[2000] As Long
Dim data_idx As Long
```

Init:

```
Rem Dauermessung mit ADC 1..8 starten,
Rem Verstärkung 1 für Kanäle 1..4, 2 für Kanäle 5..8
Start_Conv_PGA(0FFh, 0101010100000000b, 2)
Rem einmalig Ende der Wandlung abwarten
Wait_EOC()
data_idx = 1
```

Event:

```
Read_ADC8(Data_3, data_idx) '8 Werte lesen
data_idx = data_idx + 8
If (data_idx > 1992) Then data_idx = 1
Rem Konvertierung arbeitet automatisch weiter
```

Wait_EOC wartet auf das Ende der A/D-Wandlung an den ausgewählten ADC.

Syntax

```
#Include ADwin-X.inc

Wait_EOC()
```

Parameter

-/-

Beschreibung

Die ADC werden mit **Start_Conv** zum Wandeln ausgewählt.

Der Befehl **Wait_EOC** wartet im Modus Einzelmessung, bis alle eingestellten ADC ihre Wandlung beendet haben.

Im Modus „continuous“ muss das Wandlungsende nur einmalig nach dem ersten Wandlerstart abgefragt werden.

Siehe auch

[ADC](#), [ADC2](#), [ADC4](#), [ADC8](#), [ADC24](#), [ADC2_24](#), [ADC4_24](#), [ADC8_24](#), [Read_ADC](#), [Read_ADC24](#), [Read_ADC_Packed](#), [Read_ADC8](#), [Read_ADC8_24](#), [Start_Conv](#), [Start_Conv_PGA](#)

Gültig für

X-A20M1, X-A20F

Beispiel

```
#Include ADwin-X.inc
Dim Data_3[2000] As Long
Dim data_idx As Long
```

Init:

```
Rem Dauermessung mit ADC 3 starten, Verstärkungsfaktor 1
Start_Conv(00000100b, 0, 2)
Rem Einmalig Ende der Wandlung abwarten
Wait_EOC()
data_idx = 1
```

Event:

```
Data_3[data_idx] = Read_ADC(3)
data_idx = data_idx + 1
If (data_idx > 1992) Then data_idx = 1
Rem Konvertierung arbeitet automatisch weiter
```

Wait_EOC

16.3 Digitale Ein- und Ausgänge

Dieser Abschnitt beschreibt folgende Befehle:

- [Conf_DIO](#) (Seite 79)
- [Digin_Filter_Init](#) (Seite 88)
- [Dig_Latch](#) (Seite 80)
- [Digin_Read_Latch1](#) (Seite 81)
- [Digin_Read_Latch2](#) (Seite 82)
- [Digout_Write_Latch1](#) (Seite 83)
- [Digout_Write_Latch2](#) (Seite 84)
- [Digin](#) (Seite 85)
- [Digin_Long1](#) (Seite 86)
- [Digin_Long2](#) (Seite 87)
- [Digin_Edge1](#) (Seite 90)
- [Digin_Edge2](#) (Seite 91)
- [Digout](#) (Seite 92)
- [Digout_Long1](#) (Seite 93)
- [Digout_Long2](#) (Seite 94)
- [Digout_Bits1](#) (Seite 95)
- [Digout_Bits2](#) (Seite 96)
- [Get_Digout_Long1](#) (Seite 97)
- [Get_Digout_Long2](#) (Seite 98)
- [Digin_Fifo_Read_Timer](#) (Seite 99)
- [Digin_Fifo_Clear](#) (Seite 100)
- [Digin_Fifo_Enable](#) (Seite 101)
- [Digin_Fifo_Full](#) (Seite 103)
- [Digin_Fifo_Read](#) (Seite 104)
- [Digout_Fifo_Read_Timer](#) (Seite 105)
- [Digout_Fifo_Clear](#) (Seite 106)
- [Digout_Fifo_Enable](#) (Seite 107)
- [Digout_Fifo_Empty](#) (Seite 109)
- [Digout_Fifo_Mode](#) (Seite 110)
- [Digout_Fifo_Start](#) (Seite 111)
- [Digout_Fifo_Write](#) (Seite 112)

Conf_DIO konfiguriert die digitalen Kanäle DIO41:DIO00 gruppenweise als Ein- oder Ausgänge.

Syntax

```
#Include ADwin-X.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.

| Bitnr. in pattern | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----------------------|-------|-------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Kanäle | DIO41 | DIO40 | DIO39: DIO36 | DIO35: DIO32 | DIO31: DIO24 | DIO23: DIO16 | DIO15: DIO08 | DIO07: DIO00 |

Bemerkungen

Die digitalen Kanäle DIO41:DIO00 sind nach dem Einschalten als Eingänge konfiguriert (und können also auch noch nicht als Ausgänge angesprochen werden). Sie können gruppenweise als Ein- oder Ausgänge konfiguriert werden.

Die digitalen Kanäle DIO60:DIO42 sind bereits als Eingänge und Ausgänge festgelegt und können daher mit **Conf_DIO** nicht verändert werden. Die Digitalkanäle sind auf drei Steckverbinder verteilt.

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_Filter_Init](#), [Digin](#), [Digout](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Read](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Rem Configure DIO15:00 as inputs and DIO41:16 as outputs
Conf_DIO(11111100b)
```

Conf_DIO

Dig_Latch

Dig_Latch überträgt auf dem angegebenen Modul Digital-Informationen von den Eingängen in die Eingangs-Latches und von den Ausgangs-Latches zu den Ausgängen.

Syntax

```
#Include ADwin-X.Inc
```

```
Dig_Latch(pattern)
```

Parameter

pattern Bitmuster zur Auswahl der Kanalgruppen, die gelatcht `_LONG` werden.

| Bitnr. in pattern | 31:4 | 3 | 2 | 1 | 0 |
|----------------------|------|-----------------|-----------------|-----------------|-----------------|
| Kanäle | – | Ausgänge | Ausgänge | Eingänge | Eingänge |
| | | DIO63: DIO32 | DIO31: DIO00 | DIO63: DIO32 | DIO31: DIO00 |

Bemerkungen

Bei digitalen Eingängen überträgt die Anweisung die Eingangs-Signale an die Eingangs-Latches. Lesen Sie die Werte mit **Digin_Read_Latch1/2**.

Bei digitalen Ausgängen überträgt die Anweisung die Werte der Ausgangs-Latches an die Ausgänge. Schreiben Sie Werte mit **Digout_Write_Latch1/2** in die Ausgangs-Latches.

Sie können die Übertragung auch mit **Sync_All** auslösen.

Siehe auch

[Conf_DIO](#), [Digin_Read_Latch1](#), [Digin_Read_Latch2](#), [Digout_Write_Latch1](#), [Digout_Write_Latch2](#), [Digin](#), [Digout](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Sync_All](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
REM Set channels DIO15:DIO0 as outputs, DIO31:DIO16 as inputs
Conf_DIO(0011b)
Digout_Write_Latch1(0) 'Set all output bits to 0
```

Event:

```
Dig_Latch(0101b) 'latch inputs and outputs DIO31:DIO00
Rem further program
Par_1 = Digin_Read_Latch1() 'read input bits and ...
Digout_Write_Latch1(Par_1) 'output in next event cycle
```

Digin_Read_Latch1 liefert die Bitwerte aus dem Latch-Register für die digitalen Eingänge DIO31:DIO00.

Syntax

```
#Include ADwin-X.Inc

ret_val = Digin_Read_Latch1()
```

Parameter

ret_val Bitwerte des Latch-Registers. Jedes Bit entspricht `_LONG` einem digitalen Eingang.

| Bitnummer in <code>ret_val</code> | 31 | 30 | ... | 1 | 0 |
|--------------------------------------|-------|-------|-----|-------|-------|
| Eingang | DIO31 | DIO30 | ... | DIO01 | DIO00 |

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **Conf_DIO** als Eingänge zu programmieren.

Sie können den aktuellen Zustand der digitalen Eingänge mit folgenden Anweisungen in das Zwischenregister übernehmen:

- Dig_Latch
- Sync_All

Siehe auch

[Conf_DIO](#), [Dig_Latch](#), [Digin_Read_Latch2](#), [Digout_Write_Latch1](#), [Digout_Write_Latch2](#), [Digin](#), [Digout](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Sync_All](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
Dim value As Long
```

Init:

```
REM Set channels DIO15:DIO00 as outputs, DIO31:DIO16 as inputs
Conf_DIO(0011b)
Digout_Write_Latch1(0) 'Set all output bits to 0
```

Event:

```
Dig_Latch(0101b) 'latch inputs and outputs DIO31:DIO00
Rem further program
Par_1 = Digin_Read_Latch1() 'read input bits and ...
Digout_Write_Latch1(Par_1) 'output in next event cycle
```

Digin_Read_Latch1

Digin_Read_Latch2

Digin_Read_Latch2 liefert die Bitwerte aus dem Latch-Register für die digitalen Eingänge DIO60:DIO32.

Syntax

```
#Include ADwin-X.Inc

ret_val = Digin_Read_Latch2()
```

Parameter

ret_val Bitwerte des Latch-Registers. Jedes Bit entspricht `_LONG` | einem digitalen Eingang.

| Bitnummer in <code>ret_val</code> | 31:29 | 28 | 27 | ... | 1 | 0 |
|--------------------------------------|-------|-------|-------|-----|-------|-------|
| Eingang | – | DIO60 | DIO59 | ... | DIO33 | DIO32 |

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **Conf_DIO** als Eingänge zu programmieren.

Sie können den aktuellen Zustand der digitalen Eingänge mit folgenden Anweisungen in das Zwischenregister übernehmen:

- Dig_Latch
- Sync_All

Siehe auch

[Conf_DIO](#), [Dig_Latch](#), [Digin_Read_Latch1](#), [Digout_Write_Latch1](#), [Digout_Write_Latch2](#), [Digin](#), [Digout](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Sync_All](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
Dim value As Long
```

Init:

```
REM Set channels DIO39:DIO32 as outputs
Conf_DIO(110000b)
Digout_Write_Latch2(0) 'Set all output bits to 0
```

Event:

```
Dig_Latch(1010b) 'latch inputs and outputs DIO63:DIO32
Rem further program
Par_1 = Digin_Read_Latch2() 'read input bits and ...
Digout_Write_Latch1(Par_1) 'output in next event cycle
```


Digout_Write_Latch1 schreibt einen 32 Bit-Wert in das Latch-Register für die digitalen Ausgänge DIO31:DIO00.

Syntax

```
#Include ADwin-X.Inc

Digout_Write_Latch1(pattern)
```

Parameter

pattern Bitmuster. Jedes Bit entspricht einem digitalen Ausgang. _LONG_

| Bitnummer in pattern | 31 | 30 | ... | 1 | 0 |
|--------------------------------|-------|-------|-----|-------|-------|
| Ausgang | DIO31 | DIO30 | ... | DIO01 | DIO00 |

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **Conf_DIO** als Ausgänge programmiert werden.

Sie können die Werte aus dem Zwischenregister mit folgenden Anweisungen an die Ausgänge übertragen:

- Dig_Latch
- Sync_All

Sie können einen einzelnen digitalen Ausgang mit **Digout** direkt setzen.

Siehe auch

[Conf_DIO](#), [Dig_Latch](#), [Digin_Read_Latch1](#), [Digin_Read_Latch2](#), [Digout_Write_Latch2](#), [Digin](#), [Digout](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Sync_All](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
REM Set channels DIO15:DIO00 as outputs, DIO31:DIO16 as inputs
Conf_DIO(0011b)
Digout_Write_Latch1(0) 'Set all output bits to 0
```

Event:

```
Dig_Latch(0101b) 'latch inputs and outputs DIO31:DIO00
Rem further program
Par_1 = Digin_Read_Latch1() 'read input bits and ...
Digout_Write_Latch1(Par_1) 'output in next event cycle
```

Digout_Write_Latch1

Digout_Write_Latch2

Digout_Write_Latch2 schreibt einen 32 Bit-Wert in das Latch-Register für die digitalen Ausgänge DIO41:DIO32.

Syntax

```
#Include ADwin-X.Inc

Digout_Write_Latch2(pattern)
```

Parameter

pattern Bitmuster. Jedes Bit entspricht einem digitalen Ausgang. _LONG |

| Bitnummer in pattern | 31:10 | 9 | 8 | ... | 1 | 0 |
|--------------------------------|-------|-------|-------|-----|-------|-------|
| Ausgang | – | DIO41 | DIO40 | ... | DIO33 | DIO32 |

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **Conf_DIO** als Ausgänge programmiert werden.

Sie können die Werte aus dem Zwischenregister mit folgenden Anweisungen an die Ausgänge übertragen:

- Dig_Latch
- Sync_All

Sie können einen einzelnen digitalen Ausgang mit **Digout** direkt setzen.

Siehe auch

[Conf_DIO](#), [Dig_Latch](#), [Digin_Read_Latch1](#), [Digin_Read_Latch2](#), [Digout_Write_Latch1](#), [Digin](#), [Digout](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Sync_All](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
REM Set channels DIO39:DIO32 as outputs
Conf_DIO(110000b)
Digout_Write_Latch2(0) 'Set all output bits to 0
```

Event:

```
Dig_Latch(1010b) 'latch inputs and outputs DIO63:DIO32
Rem further program
Par_1 = Digin_Read_Latch2() 'read input bits and ...
Digout_Write_Latch1(Par_1) 'output in next event cycle
```

Digin gibt den Wert eines der digitalen Eingänge DIO60:DIO00 zurück.

Syntax

```
#Include ADwin-X.inc
ret_val = Digin(channel_no)
```

Parameter

| | | |
|-------------------|---|------|
| channel_no | Nummer (0...60) 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 DIO41:DIO00 gruppenweise als Eingänge oder Ausgänge. Die Kanäle DIO60:DIO42 sind als Eingänge festgelegt.

Diese Anweisung ist für das Auslesen weniger digitaler Eingänge geeignet. Für mehrere digitale Eingänge ist **Digin_Long1/2** deutlich schneller.

Siehe auch

[Conf_DIO](#), [Digin_Long1](#), [Digin_Long2](#), [Digin_Edge1](#), [Digin_Edge2](#), [Digout](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Read](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
Dim Data_1[10000] As Long As Fifo
```

Event:

```
Rem Check if input 0 has TTL level high
If (Digin(0) = 1) Then
    Data_1 = ADC(1)      'read value of ADC 1
EndIf
```

Digin

Digin_Long1

Digin_Long1 gibt die Werte der digitalen Eingänge DIO31:DIO00 zurück.

Syntax

```
#Include ADwin-X.Inc

ret_val = Digin_Long1()
```

Parameter

ret_val Bitmuster, das den TTL-Pegeln an den digitalen Eingängen entspricht (Zuordnung s.u.). **LONG**

1: TTL-Pegel high liegt an
0: TTL-Pegel low liegt an

| Bitnummer in ret_val | 31 | 30 | ... | 1 | 0 |
|--------------------------------|-------|-------|-----|-------|-------|
| Eingang | DIO31 | DIO30 | ... | DIO01 | DIO00 |

Bemerkungen

Für digitale Kanäle, die als Ausgang konfiguriert sind, hat **Digin_Long1** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle DIO31:DIO00 in Gruppen von jeweils 8 als Eingänge oder Ausgänge.

Siehe auch

[Conf_DIO](#), [Digin_Long2](#), [Digin_Edge1](#), [Digin_Edge2](#), [Digout](#), [Digout_Long1](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Read](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc

Init:
    Rem Configure DIO15:00 as inputs and DIO31:16 as outputs
    Conf_DIO(1100b)

Event:
    Par_1 = Digin_Long1() 'read values of inputs (DIO15:00)
```

Digin_Long2 gibt die Werte der digitalen Eingänge DIO60:DIO32 auf einmal zurück.

Syntax

```
#Include ADwin-X.Inc

ret_val = Digin_Long2()
```

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

| Bitnummer in ret_val | 31:29 | 28 | 27 | ... | 1 | 0 |
|--------------------------------|-------|-------|-------|-----|-------|-------|
| Eingang | – | DIO60 | DIO59 | ... | DIO33 | DIO32 |

Bemerkungen

Für digitale Kanäle, die als Ausgang konfiguriert sind, hat **Digin_Long2** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle DIO41:DIO32 gruppenweise als Eingänge oder Ausgänge. Die Kanäle DIO60:DIO42 sind immer als Eingänge festgelegt.

Siehe auch

[Conf_DIO](#), [Digin_Long1](#), [Digin_Edge1](#), [Digin_Edge2](#), [Digout](#), [Digout_Long2](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Read](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Rem Configure DIO15:00/DIO39:32 as inputs,
Rem and DIO31:16/DIO41:40 as outputs
Conf_DIO(11001100b)
```

Event:

```
Par_1 = Digin_Long1() 'read inputs DIO15:00
Par_2 = Digin_Long2() 'read inputs DIO39:32/DIO59:42
```

Digin_Long2

Digin_Filter_Init

Digin_Filter_Init stellt die Filter-Prüfdauer für alle Eingänge ein.

Syntax

```
#Include ADwin-X.inc  
  
Digin_Filter_Init(filter_value)
```

Parameter

filter_value Prüfdauer des Filters, angegeben in Einheiten **LONG** | (1...65535) von 20ns.
Der Wert 0 (Null) deaktiviert den Filter.

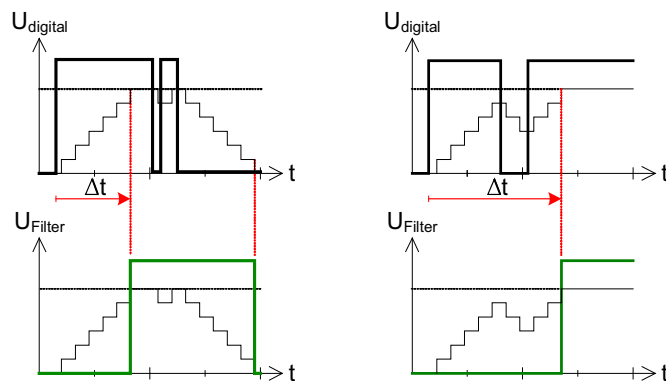
Bemerkungen

Der Filter unterdrückt einzelne Fehlpulse (Spikes) eines Signals. Die Anzahl der Fehlpulse sollte im Verhältnis zur Pulsbreite des Signals klein sein. Die Prüfdauer des Filters sollte etwas länger sein als die erwartete Breite der Fehlpulse.

Die Filtereinstellung gilt für alle Kanäle gleichermaßen und betrifft auch die Eingänge für Zähler und SSI-Decoder. Jeder Kanal hat seinen eigenen Filter. Nach dem Einschalten sind die Filter deaktiviert.

Der Filter überträgt einen Flankenwechsel des Eingangssignals nicht direkt auf das Ausgangssignal. Je nach Eingangssignal wird alle 20ns ein Zähler erhöht (High-Signal) oder erniedrigt (Low-Signal), in den Grenzen von 0... **filter_value**. Hat der Zähler den Wert 0, hat das Ausgangssignal den Pegel Low, bei **filter_value** ist es Pegel High.

Beachten Sie: Der Filter verzögert Flanken des resultierenden Signals um die eingestellte Prüfdauer. Falls Fehlpulse auftreten, könne sich die Flanken zusätzlich geringfügig verzögern.



Die Abbildung zeigt das Filtern von 2 Beispielsignalen (schwarzes Signal, oben) mit Fehlpulsen. Das Treppensmuster zeigt den Wert des Filterzählers. Im rechten Beispiel verzögert sich die resultierende Flanke durch den Fehlpuls. Der Filter (hier mit **filter_value**=6) verzögert die Flanken des resultierenden Signals; die Verzögerung Δt kann größer werden, wenn mehrere Fehlpulse vorkommen.

Beachten Sie: Der Eingangsfiler beeinflusst die Zeitstempel bei der Flankenüberwachung für einen Eingangs-FIFO (siehe **Digin_Fifo_Read_Timer**). Bei eingeschaltetem Eingangsfiler ist der Zeitabstand zwischen zwei Zeitstempeln ein ganzzahliges Vielfaches von 20ns. Anders gesagt: es gibt entweder nur gerade oder nur ungerade Zeitstempel.

Siehe auch

[Conf_DIO](#), [Digin_Long1](#), [Digin_Long2](#), [Digin_Edge1](#), [Digin_Edge2](#), [Cnt_Mode](#), [SSI_Mode](#), [Digin_Fifo_Read_Timer](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Conf_DIO(0000b)      'Set DIO31:00 as inputs  
Digin_Filter_Init(5)  'set spike filter to 100ns
```

Event:

```
Par_1 = Digin_Long1()  'Read inputs DIO31:00
```

Digin_Edge1

Digin_Edge1 gibt zurück, ob an den Digitaleingängen DIO31:DIO00 eine positive oder negative Flanke aufgetreten ist.

Syntax

```
#Include ADwin-X.inc

ret_val = Digin_Edge1(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 | DIO31 | DIO30 | ... | DIO02 | DIO01 | DIO00 |

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 Befehl **Conf_DIO** konfiguriert die digitalen Kanäle DIO31:DIO00 in Gruppen von jeweils 8 als Eingänge oder Ausgänge.

Der Aufruf von **Digin_Edge1** setzt alle Bits zurück auf 0.

Siehe auch

[Conf_DIO](#), [Digin_Long1](#), [Digin_Long2](#), [Digin_Edge2](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Read](#), [Digout_Fifo_Read_Timer](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.inc

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

Event:
    Rem check rising and falling edges, mask out outputs
    Par_1 = Digin_Edge1(1) And 0Fh
    Par_2 = Digin_Edge1(0) And 0Fh

    Rem output edge changes to outputs
    If (Par_1 + Par_2 > 0) Then
        Digout_Bits1(Shift_Left(Par_1,16), Shift_Left(Par_2,16))
    EndIf
```


Digin_Edge2 gibt zurück, ob an den Digitaleingängen DIO60:DIO32 eine positive oder negative Flanke aufgetreten ist.

Syntax

```
#Include ADwin-X.inc

ret_val = Digin_Edge2(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. | 31:29 | 28 | 27 | ... | 2 | 1 | 0 |
|---------|-------|-------|-------|-----|-------|-------|-------|
| Eingang | – | DIO60 | DIO59 | ... | DIO34 | DIO33 | DIO32 |

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 Befehl **Conf_DIO** konfiguriert die digitalen Kanäle DIO41:DIO32 gruppenweise als Eingänge oder Ausgänge. Die Kanäle DIO60:DIO42 sind als Eingänge festgelegt.

Der Aufruf von **Digin_Edge2** setzt alle Bits zurück auf 0.

Siehe auch

[Conf_DIO](#), [Digin_Long1](#), [Digin_Long2](#), [Digin_Edge1](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Read](#), [Digout_Fifo_Read_Timer](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
Rem Configure DIO15:00/DIO39:32 as inputs,
Rem and DIO31:16/DIO41:40 as outputs
Conf_DIO(11001100b)
```

Event:

```
Rem check rising and falling edges, mask out outputs
Par_1 = Digin_Edge2(1) And 0Fh
Par_2 = Digin_Edge2(0) And 0Fh

Rem output edge changes to outputs
If (Par_1 + Par_2 > 0) Then
    Digout_Bits2(Shift_Left(Par_1,16),Shift_Left(Par_2,16))
EndIf
```

Digin_Edge2

Digout

Digout setzt einen der digitalen Ausgänge DIO41:DIO00 auf einen definierten TTL-Pegel.

Syntax

```
#Include ADwin-X.Inc

Digout(channel_no, level)
```

Parameter

| | | |
|-------------------|---|------|
| channel_no | Nummer (0...41) des Digitalausgangs DIO41:DIO00. | LONG |
| level | TTL-Pegel des gewählten Digitaleingangs: 1: TTL-Pegel high 0: TTL-Pegel low | LONG |

Bemerkungen

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

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle DIO41:DIO32 gruppenweise als Eingänge oder Ausgänge. Die Kanäle DIO60:DIO42 sind als Eingänge festgelegt.

Die Digitalkanäle sind auf drei Steckverbinder verteilt.

Diese Anweisung ist für das Setzen weniger digitaler Eingänge geeignet. Für mehrere digitale Ausgänge ist **Digout_Long1/2** deutlich schneller. Beachten Sie dies insbesondere bei kritischen Anwendungen.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout_Long1](#), [Digout_Long2](#), [Digout_Bits1](#), [Digout_Bits2](#), [Get_Digout_Long1](#), [Get_Digout_Long2](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc

Init:
    Rem Configure DIO31:00 as outputs
    Conf_DIO(1111b)
    Par_2 = 0AAAAAAAAh          'Bit pattern for even bits

Event:
    Digout(Par_2, 1)            'set outputs to level high
```

Digout_Long1 setzt mit einem Bitmuster gleichzeitig die digitalen Ausgänge DIO31:DIO00 auf definierte TTL-Pegel.

Syntax

```
#Include ADwin-X.Inc
```

```
Digout_Long1(pattern)
```

Parameter

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

| Bitnummer in pattern | 31 | 30 | ... | 1 | 0 |
|-------------------------|-------|-------|-----|-------|-------|
| Ausgang | DIO31 | DIO30 | ... | DIO01 | DIO00 |

Bemerkungen

Für digitale Kanäle, die als Eingang konfiguriert sind, hat **Digout_Long1** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle DIO31:DIO00 in Gruppen von jeweils 8 als Eingänge oder Ausgänge.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Long2](#), [Digout_Bits1](#), [Digout_Bits2](#), [Get_Digout_Long1](#), [Get_Digout_Long2](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Rem Configure DIO15:00 as outputs and DIO31:16 as inputs
Conf_DIO(0011b)
Rem bit pattern for odd bits
Par_1 = 55555555h
```

Event:

```
Digout_Long1(Par_1)          'output bits DIO15:00
```

Digout_Long1

Digout_Long2

Digout_Long2 setzt mit einem Bitmuster gleichzeitig die digitalen Ausgänge DIO41:DIO32 auf definierte TTL-Pegel.

Syntax

```
#Include ADwin-X.Inc

Digout_Long2(pattern)
```

Parameter

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

| Bitnummer in pattern | 31:10 | 9 | 8 | ... | 1 | 0 |
|-------------------------|-------|-------|-------|-----|-------|-------|
| Ausgang | – | DIO41 | DIO40 | ... | DIO33 | DIO32 |

Bemerkungen

Für digitale Kanäle, die als Eingang konfiguriert sind, hat **Digout_Long2** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle DIO41:DIO32 gruppenweise als Eingänge oder Ausgänge. Die Kanäle DIO60:DIO42 sind als Eingänge festgelegt.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Long1](#), [Digout_Bits1](#), [Digout_Bits2](#), [Get_Digout_Long1](#), [Get_Digout_Long2](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Rem Configure DIO39:32 as outputs
Conf_DIO(00110000b)
Rem bit pattern for even bits
Par_2 = 0AAAAAAAAh
```

Event:

```
Digout_Long2(Par_2) 'output bits DIO39:32
```

Digout_Bits1 setzt mit einem Bitmuster gleichzeitig die digitalen Ausgänge DIO31:DIO00 auf definierte TTL-Pegel.

Syntax

```
#Include ADwin-X.Inc

Digout_Bits1(set,clear)
```

Parameter

set Bitmuster zur Auswahl der Ausgänge, an denen der TTL-Pegel High gesetzt wird (Zuordnung s.u.).
 Bit = 1: Setzen auf TTL-Pegel High.
 Bit = 0: TTL-Pegel nicht ändern.

clear Bitmuster zur Auswahl der Ausgänge, an denen der TTL-Pegel Low gesetzt wird (Zuordnung s.u.).
 Bit = 1: Setzen auf TTL-Pegel Low.
 Bit = 0: TTL-Pegel nicht ändern.

| Bitnummer in set/clear | 31 | 30 | ... | 1 | 0 |
|---------------------------|-------|-------|-----|-------|-------|
| Ausgang | DIO31 | DIO30 | ... | DIO01 | DIO00 |

Bemerkungen

Für digitale Kanäle, die als Eingang konfiguriert sind, hat **Digout_Bits1** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle DIO31:DIO00 in Gruppen von jeweils 8 als Eingänge oder Ausgänge.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Long1](#), [Digout_Long2](#), [Digout_Bits2](#), [Get_Digout_Long1](#), [Get_Digout_Long2](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Rem Configure DIO15:00 as outputs and DIO31:16 as inputs
Conf_DIO(0011b)
Rem bit pattern for odd bits
Par_1 = 55555555h
```

Event:

```
Rem set odd bits to level high, leave even bits unchanged
Digout_Bits1(Par_1,0)
```

Digout_Bits1

Digout_Bits2

Digout_Bits2 setzt mit einem Bitmuster gleichzeitig die digitalen Ausgänge DIO41:DIO32 auf definierte TTL-Pegel.

Syntax

```
#Include ADwin-X.Inc

Digout_Bits2(set,clear)
```

Parameter

| | | |
|--------------|---|------|
| set | Bitmuster zur Auswahl der Ausgänge, an denen der TTL-Pegel High gesetzt wird (Zuordnung s.u.). Bit = 1: Setzen auf TTL-Pegel High. Bit = 0: TTL-Pegel nicht ändern. | LONG |
| clear | Bitmuster zur Auswahl der Ausgänge, an denen der TTL-Pegel Low gesetzt wird (Zuordnung s.u.). Bit = 1: Setzen auf TTL-Pegel Low. Bit = 0: TTL-Pegel nicht ändern. | LONG |

| Bitnummer | 31:10 | 9 | 8 | ... | 1 | 0 |
|-----------|-------|-------|-------|-----|-------|-------|
| in | | | | | | |
| set/clear | | | | | | |
| Ausgang | – | DIO41 | DIO40 | ... | DIO33 | DIO32 |

Bemerkungen

Für digitale Kanäle, die als Eingang konfiguriert sind, hat **Digout_Bits2** keine Funktion.

Der Befehl **Conf_DIO** konfiguriert die digitalen Kanäle DIO41:DIO32 gruppenweise als Eingänge oder Ausgänge. Die Kanäle DIO60:DIO42 sind als Eingänge festgelegt.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Long1](#), [Digout_Long2](#), [Digout_Bits1](#), [Get_Digout_Long1](#), [Get_Digout_Long2](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Rem Configure DIO39:32 as outputs
Conf_DIO(00110000b)
Rem bit pattern for odd bits
Par_2 = 0AAAAAAAAh
```

Event:

```
Rem set even bits to level high, leave odd bits unchanged
Digout_Bits2(Par_1,0)
```

Get_Digout_Long1 gibt den Inhalt des Registers für die digitalen Ausgänge DIO31:DIO00 zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = Get_Digout_Long1()
```

Parameter

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

| Bitnr. in <code>ret_val</code> | 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 DIO31:DIO00 in Gruppen von jeweils 8 als Eingänge oder Ausgänge.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Long1](#), [Digout_Long2](#), [Digout_Bits1](#), [Digout_Bits2](#), [Get_Digout_Long2](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
REM Alle Kanäle als Ausgänge konfigurieren
Conf_DIO(11111111b)
Processdelay = 10000
```

Event:

```
Par_1 = Get_Digout_Long1() 'Bits 31:0 aus dem Register
                        'zurücklesen
```

Get_Digout_Long1

Get_Digout_Long2

Get_Digout_Long2 gibt den Inhalt des Registers für die digitalen Ausgänge DIO41:DIO32 zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = Get_Digout_Long2()
```

Parameter

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

| Bitnummer in ret_val | 31:10 | 9 | 8 | ... | 1 | 0 |
|-------------------------|-------|-------|-------|-----|-------|-------|
| Ausgang | – | DIO41 | DIO40 | ... | DIO33 | DIO32 |

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 DIO41:DIO32 gruppenweise als Eingänge oder Ausgänge. Die Kanäle DIO60:DIO42 sind als Eingänge festgelegt.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Long1](#), [Digout_Long2](#), [Digout_Bits1](#), [Digout_Bits2](#), [Get_Digout_Long1](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Write](#)

Gültig für

X-A20, X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
REM Alle Kanäle als Ausgänge konfigurieren
Conf_DIO(11111111b)
Processdelay = 10000
```

Event:

```
Par_1 = Get_Digout_Long2() 'Bits 61:32 aus dem Register
' zurücklesen
```


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

Syntax

```
#Include ADwin-X.inc

ret_val = Digin_Fifo_Read_Timer(fifo_no)
```

Parameter

| | | |
|----------------|--|------|
| fifo_no | Nummer (1, 2) des Eingangs-FIFOs zur Flankenüberwachung. | LONG |
| ret_val | Aktueller Stand ($-2^{31}-1 \dots 2^{31}$) des 100MHz-Zählers. | LONG |

Bemerkungen

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

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

Beachten Sie: Wenn der Eingangsfiler (siehe **Digin_Filter_Init**) eingeschaltet ist, ist der Zeitabstand zwischen zwei Zeitstempeln ein ganzzahliges Vielfaches von 20ns. Anders gesagt: es gibt entweder nur gerade oder nur ungerade Zeitstempel.

Die Zähler können mit **Sync_All** auf Null gesetzt werden.

Siehe auch

[Digin](#), [Digin_Edge1](#), [Digin_Edge2](#), [Digout](#), [Digin_Fifo_Clear](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Full](#), [Digin_Fifo_Read](#), [Digout_Fifo_Read_Timer](#), [Digout_Fifo_Enable](#), [Digin_Filter_Init](#), [Sync_All](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
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(1)
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(1) - 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

Digin_Fifo_Read_Timer

Digin_Fifo_Clear

Digin_Fifo_Clear löscht einen Eingangs-FIFO zur Flankenüberwachung.

Syntax

```
#Include ADwin-X.inc  
Digin_Fifo_Clear(fifo_no)
```

Parameter

| | | |
|----------------------|--|-------------------|
| <code>fifo_no</code> | Nummer (1, 2) des Eingangs-FIFOs zur Flankenüberwachung. | <code>LONG</code> |
|----------------------|--|-------------------|

Bemerkungen

Der Eingangs-FIFO 1 bezieht sich auf die Digitaleingänge DIO31:DIO00, der Eingangs-FIFO 2 auf die Digitaleingänge DIO60:DIO32.

Die Eingangs-FIFOs können auch mit **Sync_All** gelöscht werden.

Siehe auch

[Digin](#), [Digin_Edge1](#), [Digin_Edge2](#), [Digout](#), [Digin_Fifo_Read_Timer](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Full](#), [Digin_Fifo_Read](#), [Digout_Fifo_Clear](#), [Digout_Fifo_Enable](#), [Sync_All](#)

Gültig für

X-A20+DCT

Beispiel

siehe [Digin_Fifo_Enable](#)

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

Syntax

```
#Include ADwin-X.inc

Digin_Fifo_Enable(fifo_no, pattern)
```

Parameter

fifo_no Nummer (1, 2) des Eingangs-FIFOs zur Flankenüberwachung. LONG

pattern Bitmuster, das die zu überwachenden Eingangskanäle festlegt. LONG

Bemerkungen

Der Eingangs-FIFO 1 bezieht sich auf die Digitaleingänge DIO31:DIO00, der Eingangs-FIFO 2 auf die Digitaleingänge DIO60:DIO32.

| Bitnr. bei FIFO 1 | 31 | 30 | ... | 2 | 1 | 0 |
|-------------------|-------|-------|-----|-------|-------|-------|
| Eingang | DIO31 | DIO30 | ... | DIO02 | DIO01 | DIO00 |

| Bitnr. bei FIFO 2 | 31:29 | 28 | 27 | ... | 1 | 0 |
|-------------------|-------|-------|-------|-----|-------|-------|
| Eingang | – | DIO60 | DIO59 | ... | DIO33 | DIO32 |

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

Die Flankenüberwachung prüft alle 10ns, 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 Eingangs-FIFO kopiert:

- Wert 1 enthält den Pegelzustand der betroffenen 32 Kanäle als Bitmuster.
- Wert 2 enthält einen Zeitstempel, den aktuellen Stand eines 100MHz-Zählers.

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

Siehe auch

[Conf_DIO](#), [Digin](#), [Digin_Edge1](#), [Digin_Edge2](#), [Digout](#), [Digin_Fifo_Read_Timer](#), [Digin_Fifo_Clear](#), [Digin_Fifo_Full](#), [Digin_Fifo_Read](#), [Digout_Fifo_Enable](#)

Gültig für

X-A20+DCT

Digin_Fifo_Enable

Beispiel

```
#Include ADwin-X.inc

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(1,0)     'edge control off
  Digin_Fifo_Clear(1)        'clear FIFO 1
  Digin_Fifo_Enable(1,101010b) 'control channels 1,3,5
  index = 1

Event:
  num = Digin_Fifo_Full(1) '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(1,Data_1[index], Data_2[index])
      index = index+1
    Next i
  EndIf
```

Digin_Fifo_Full gibt die Anzahl der gespeicherten Wertepaare in einem FIFO der Flankenüberwachung zurück.

Syntax

```
#Include ADwin-X.inc  
  
ret_val = Digin_Fifo_Full(fifo_no)
```

Parameter

| | | |
|----------------|--|--------|
| fifo_no | Nummer (1, 2) des Eingangs-FIFOs zur Flankenüberwachung. | __LONG |
| ret_val | Anzahl (0...511) der belegten Wertepaare im FIFO. | LONG |

Bemerkungen

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

Siehe auch

[Digin](#), [Digin_Edge1](#), [Digin_Edge2](#), [Digout](#), [Digin_Fifo_Read_Timer](#), [Digin_Fifo_Clear](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Read](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Empty](#)

Gültig für

X-A20+DCT

Beispiel

siehe [Digin_Fifo_Enable](#)

Digin_Fifo_Full

Digin_Fifo_Read

Digin_Fifo_Read liest ein Wertepaar aus dem FIFO der Flankenüberwachung und gibt es in 2 Variablen zurück.

Syntax

```
#Include ADwin-X.inc

Digin_Fifo_Read(fifo_no, value_by_ref,
               timestamp_by_ref)
```

Parameter

| | | |
|-------------------------------|---|---------------|
| <code>fifo_no</code> | Nummer (1, 2) des Eingangs-FIFO. | LONG |
| <code>value_by_ref</code> | Variable, in die ein Bitmuster der Pegelzustände geschrieben wird. Die Zuordnung der Bits zu den Eingängen ist unten dargestellt. | LONG CONST |
| <code>timestamp_by_ref</code> | Variable, in die der zugehörige Zeitstempel geschrieben wird. | LONG CONST |

Bemerkungen

Der Eingangs-FIFO 1 bezieht sich auf die Digitaleingänge DIO31:DIO00, der Eingangs-FIFO 2 auf die Digitaleingänge DIO60:DIO32.

| | | | | | | |
|-------------------|-------|-------|-----|-------|-------|-------|
| Bitnr. bei FIFO 1 | 31 | 30 | ... | 2 | 1 | 0 |
| Eingang | DIO31 | DIO30 | ... | DIO02 | DIO01 | DIO00 |

| | | | | | | |
|-------------------|-------|-------|-------|-----|-------|-------|
| Bitnr. bei FIFO 2 | 31:29 | 28 | 27 | ... | 1 | 0 |
| Eingang | – | DIO60 | DIO59 | ... | DIO33 | DIO32 |

Vor dem Auslesen müssen Sie mit **Digin_Fifo_Full** sicherstellen, dass mindestens ein Wertepaar im FIFO gespeichert ist.

Die Übergabeparameter müssen Variablen (oder Feldelemente) sein, Konstanten sind nicht erlaubt.

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

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

Siehe auch

[Conf_DIO](#), [Digin](#), [Digin_Edge1](#), [Digin_Edge2](#), [Digout](#), [Digin_Fifo_Read_Timer](#), [Digin_Fifo_Clear](#), [Digin_Fifo_Enable](#), [Digin_Fifo_Full](#), [Digout_Fifo_Enable](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.inc

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(1,0)    'edge control off
  Digin_Fifo_Clear(1)       'clear FIFO
  Digin_Fifo_Enable(1,10011b) 'control channels 0,1,4
  index = 1

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

Digout_Fifo_Read_Timer gibt den aktuellen Stand eines 100MHz-Zählers zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = Digout_Fifo_Read_Timer(fifo_no)
```

Parameter

| | | |
|----------------|--|------|
| fifo_no | Nummer (1, 2) des Ausgangs-FIFO. | LONG |
| ret_val | Aktueller Stand ($-2^{31}-1 \dots 2^{31}$) des 100MHz-Zählers. | LONG |

Bemerkungen

Der Zähler wird für die exakte Ausgabe von Flanken zu vorgegebenen Zeitpunkten benutzt, siehe **Digout_Fifo_Write**.

Der Ausgangs-FIFO 1 bezieht sich auf die Digitalausgänge DIO31:DIO00, der Ausgangs-FIFO 2 auf die Digitalausgänge DIO60:DIO32.

Der Zählerstand kann nur im FIFO-Betriebsmodus mit absoluten Werten verwendet werden, d.h. Parameter **mode** = 1 bei **Digout_Fifo_Mode**.

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

Der Zähler wird mit **Digout_Fifo_Clear** auf Null gesetzt.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Bits1](#), [Digout_Bits2](#), [Digin_Fifo_Read_Timer](#), [Digout_Fifo_Clear](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Empty](#), [Digout_Fifo_Mode](#), [Digout_Fifo_Start](#), [Digout_Fifo_Write](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
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 = Digout_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 = Digout_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

Digout_Fifo_Read_Timer

Digout_Fifo_Clear

Digout_Fifo_Clear stoppt die Flankenausgabe und löscht den FIFO der Flanken-
ausgabe.

Syntax

```
#Include ADwin-X.inc  
  
Digout_Fifo_Clear(fifo_no)
```

Parameter

| | | |
|----------------------|--|-------------------|
| <code>fifo_no</code> | Nummer (1, 2) des Ausgangs-FIFOs zur Flankenüberwachung. | <code>LONG</code> |
|----------------------|--|-------------------|

Bemerkungen

Der FIFO muss vor der ersten Anwendung gelöscht werden. Anschließend kann der FIFO über **Digout_Fifo_Write** mit Daten befüllt werden.

Wenn die Flankenausgabe mit **Digout_Fifo_Clear** gestoppt wurde, kann sie nur mit **Digout_Fifo_Start** wieder gestartet werden.

Die Ausgangs-FIFOs können auch mit **Sync_All** gelöscht werden.

Siehe auch

[Digin](#), [Digout](#), [Digout_Bits1](#), [Digout_Bits2](#), [Digin_Fifo_Clear](#), [Digout_Fifo_Read_Timer](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Empty](#), [Digout_Fifo_Mode](#), [Digout_Fifo_Start](#), [Digout_Fifo_Write](#), [Sync_All](#)

Gültig für

X-A20+DCT

Beispiel

siehe [Digout_Fifo_Mode](#)

Digout_Fifo_Enable legt fest, an welchen Ausgangskanälen die Flankenausgabe stattfindet.

Syntax

```
#Include ADwin-X.inc
```

```
Digout_Fifo_Enable(fifo_no, pattern)
```

Parameter

| | | |
|----------------|--|------|
| fifo_no | Nummer (1, 2) des Ausgangs-FIFOs zur Flankenüberwachung. | LONG |
| pattern | Bitmuster, das die Ausgangskanäle für die Flankenausgabe festlegt. | LONG |

Bemerkungen

Der Ausgangs-FIFO 1 bezieht sich auf die Digitalausgänge DIO31:DIO00, der Ausgangs-FIFO 2 auf die Digitalausgänge DIO60:DIO32.

| | | | | | | |
|-------------------|-------|-------|-----|-------|-------|-------|
| Bitnr. bei FIFO 1 | 31 | 30 | ... | 2 | 1 | 0 |
| Ausgang | DIO31 | DIO30 | ... | DIO02 | DIO01 | DIO00 |

| | | | | | | |
|-------------------|-------|-------|-------|-----|-------|-------|
| Bitnr. bei FIFO 2 | 31:29 | 28 | 27 | ... | 1 | 0 |
| Ausgang | – | DIO60 | DIO59 | ... | DIO33 | DIO32 |

Flanken können nur auf Ausgangskanälen ausgegeben werden. Sie programmieren die Kanäle mit **Conf_DIO** als Eingänge oder Ausgänge.

Conf_DIO konfiguriert die digitalen Kanäle DIO41:DIO32 gruppenweise als Eingänge oder Ausgänge. Die Kanäle DIO60:DIO42 sind als Eingänge festgelegt.

Digout_Fifo_Enable wählt Kanäle zur Flankenausgabe über einen Ausgangs-FIFO aus. An den übrigen Ausgangskanälen – und zwar nur an diesen – können Sie die Pegel einzeln mit Befehlen wie **Digout_Long** setzen.

Die Pegel und Zeitpunkte für die Flankenausgabe werden mit **Digout_Fifo_Write** festgelegt.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Bits1](#), [Digout_Bits2](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Read_Timer](#), [Digout_Fifo_Clear](#), [Digout_Fifo_Empty](#), [Digout_Fifo_Mode](#), [Digout_Fifo_Start](#), [Digout_Fifo_Write](#)

Gültig für

X-A20+DCT

Digout_Fifo_Enable

Beispiel

```
#Include ADwin-X.inc
Dim Data_1[10000], Data_2[10000] As Long

Init:
  Conf_DIO(1111b)           'channels 31:0 as outputs
  Digout_Fifo_Clear(1)       'clear FIFO
  Digout_Fifo_Enable(1,101010b) 'edge output on channels 1,3,5
  Rem write 3 value pairs into output FIFO and start output
  Rem 100ns: channels 1,3,5
  Rem 300ns: channels 1,3
  Rem 500ns: channels 3,5
  Digout_Fifo_Write(1,101010b,10)
  Digout_Fifo_Write(1,001010b,30)
  Digout_Fifo_Write(1,101000b,50)
  Digout_Fifo_Start(1)       'clear FIFO

Event:
  Rem write new value pairs into FIFO, if possible
  If (Digout_Fifo_Empty(1) > 2) Then
    Digout_Fifo_Write(1,100000b,1)
  EndIf
  If (Digout_Fifo_Empty(1) > 20) Then
    Digout_Fifo_Write(1,101010b,10)
    Digout_Fifo_Write(1,001010b,30)
    Digout_Fifo_Write(1,101000b,50)
  EndIf
```

Digout_Fifo_Empty gibt die Anzahl der freien Wertepaare in einem Ausgangs-FIFO zurück.

Syntax

```
#Include ADwin-X.inc

ret_value = Digout_Fifo_Empty(fifo_no)
```

Parameter

| | | |
|------------------|---|-----------------------|
| fifo_no | Nummer (1, 2) des Ausgangs-FIFO. | _LONG |
| ret_value | Anzahl (0...511) der freien Wertepaare im FIFO. | LONG |

Bemerkungen

Der Ausgangs-FIFO 1 bezieht sich auf die Digitalausgänge DIO31:DIO00, der Ausgangs-FIFO 2 auf die Digitalausgänge DIO41:DIO32.

Ein FIFO kann maximal 511 Wertepaare (Pegelzustand und Zeitstempel) enthalten.

Siehe auch

[Digin](#), [Digout](#), [Digout_Bits1](#), [Digout_Bits2](#), [Digin_Fifo_Full](#), [Digout_Fifo_Read_Timer](#), [Digout_Fifo_Clear](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Mode](#), [Digout_Fifo_Start](#), [Digout_Fifo_Write](#)

Gültig für

X-A20+DCT

Beispiel

siehe [Digout_Fifo_Mode](#)

Digout_Fifo_Empty

Digout_Fifo_Mode

Digout_Fifo_Mode stellt den Betriebsmodus eines Ausgangs-FIFO ein.

Syntax

```
#Include ADwin-X.inc

Digout_Fifo_Mode(fifo_no,mode)
```

Parameter

| | | |
|----------------------|---|------|
| <code>fifo_no</code> | Nummer (1, 2) des Ausgangs-FIFO. | LONG |
| <code>mode</code> | Betriebsmodus des FIFO: 1: Ausgangs-FIFO zur Flankenausgabe, werte absolut. 3: Ausgangs-FIFO zur Flankenausgabe, werte relativ. | LONG |

Bemerkungen

Der Ausgangs-FIFO 1 bezieht sich auf die Digitalausgänge DIO31:DIO00, der Ausgangs-FIFO 2 auf die Digitalausgänge DIO41:DIO32.

stempel im Ausgangs-FIFO geben den Ausgabepunkt eines Flanken-Bitmusters (siehe **Digout_Fifo_Write**) an. Die Werte eines Zeitstempels können absolut oder relativ angegeben werden:

- Absolutwert: Der Zeitstempel bezieht sich auf den Startpunkt 0 des 100MHz-Zählers (**Digout_Fifo_Start**).
In diesem Modus kann der Zählerstand mit **Digout_Fifo_Read_Timer** gelesen werden.
- Relativwert: Der Zeitstempel wird relativ zum vorherigen Zeitstempel angegeben.

Solange Wertepaare im FIFO vorhanden sind, kann die Liste an Wertepaaren im FIFO aufgefüllt werden.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Bits1](#), [Digout_Bits2](#), [Digout_Fifo_Read_Timer](#), [Digout_Fifo_Clear](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Empty](#), [Digout_Fifo_Start](#), [Digout_Fifo_Write](#)

Gültig für

X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
Dim value[4] As Long

Init:
    Processdelay = 6000          '6000 x 1.5 ns = 9µs
    value[1] = 01b               'output value n
    value[2] = 5000              ' with output time 50 µs (relative)
    value[3] = 10b               'output value n+1
    value[4] = 7000              ' with output time 70 µs (relative)
    Conf_DIO(01111b)            'set DIO31:00 as output
    Digout_Fifo_Mode(1,3)        'Set FIFO1 as relative output
    Digout_Fifo_Clear(1)         'clear FIFO
    Digout_Fifo_Enable(1,11b)    'Enable output channels 0+1
    Rem write 2 value pairs into output FIFO and start output
    Digout_Fifo_Write(1,value[1],value[2])
    Digout_Fifo_Write(1,value[3],value[4])
    Digout_Fifo_Start(01b)

Event:
    Rem write new value pairs into FIFO, if possible
    If (Digout_Fifo_Empty(1) >= 2) Then
        Digout_Fifo_Write(1,value[1],value[2])
        Digout_Fifo_Write(1,value[3],value[4])
    EndIf
```

Digout_Fifo_Start startet die Flankenausgabe auf den Ausgangs-FIFOs.

Syntax

```
#Include ADwin-X.inc

Digout_Fifo_Start(fifo_pattern)
```

Parameter

fifo_pattern Bitmuster zum Ansprechen der Ausgangs-FIFOs: **LONG** |
 Bit = 0: FIFO ignorieren.
 Bit = 1: Flankenausgabe auf dem FIFO starten.

| | | | |
|-----------------------------|------|---|---|
| Bits in fifo_pattern | 31:2 | 1 | 0 |
| FIFO-Nummer | – | 2 | 1 |

Bemerkungen

Mit dem Start der Flankenausgabe beginnt der 100MHz-Zähler von 0 aus zu zählen. Der 100MHz-Zähler wird für die genaue Flankenausgabe benutzt, siehe **Digout_Fifo_Write**.

Der Zähler wird regelmäßig um 1 erhöht, so dass der Zähler nach 2^{32} Takten seinen ursprünglichen Wert erneut erreicht. Bei vergleichen muss dieser „Überlauf“ berücksichtigt werden, der Zählerstand muss daher im Programm regelmäßig vor dem Überlauf abgefragt werden. Der Zähler arbeitet mit einer Taktrate von 100MHz (Schrittweite 10ns).

Sie können die Flankenausgabe auch mit **Sync_All** starten.

Siehe auch

[Digin](#), [Digout](#), [Digout_Bits1](#), [Digout_Bits2](#), [Digout_Fifo_Read_Timer](#), [Digout_Fifo_Clear](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Empty](#), [Digout_Fifo_Mode](#), [Digout_Fifo_Write](#), [Sync_All](#)

Gültig für

X-A20+DCT

Beispiel

siehe [Digout_Fifo_Mode](#)

Digout_Fifo_Start

Digout_Fifo_Write

Digout_Fifo_Write schreibt Wertepaare in den FIFO der Flankenausgabe.

Syntax

```
#Include ADwin-X.inc
```

```
Digout_Fifo_Write(fifo_no, level_pattern, timestamp)
```

Parameter

| | | |
|----------------------|--|------|
| fifo_no | Nummer (1, 2) des Ausgangs-FIFO. | LONG |
| level_pattern | Bitmuster der Pegelzustände, die ausgegeben werden. Bit=0: TTL-Pegel low. Bit=1: TTL-Pegel high. Zuordnung der Bits zu Ausgängen siehe unten. | LONG |
| timestamp | Der Zeitstempel (in 10ns-Schritten) zu level_pattern , der den Zeitpunkt der Ausgabe festlegt. | LONG |

Bemerkungen

Der Ausgangs-FIFO 1 bezieht sich auf die Digitalausgänge DIO31:DIO00, der Ausgangs-FIFO 2 auf die Digitalausgänge DIO41:DIO32.

| | | | | | | |
|-------------------|-------|-------|-----|-------|-------|-------|
| Bitnr. bei FIFO 1 | 31 | 30 | ... | 2 | 1 | 0 |
| Ausgang | DIO31 | DIO30 | ... | DIO02 | DIO01 | DIO00 |

| | | | | | | |
|-------------------|-------|-------|-------|-----|-------|-------|
| Bitnr. bei FIFO 2 | 31:10 | 9 | 8 | ... | 1 | 0 |
| Ausgang | – | DIO41 | DIO40 | ... | DIO33 | DIO32 |

Es dürfen nicht mehr Wertepaare geschrieben werden als im FIFO frei sind. Die Anzahl der freien Wertepaare wird mit **Digout_Fifo_Empty** bestimmt.

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

Der Zeitstempel kann absolut oder relativ angegeben werden, siehe **Dig_Fifo_Mode**. Der Abstand zwischen zwei Zeitstempeln muss mindestens 20ns betragen. Der Wert eines Zeitstempels wird in Prozessortakten gezählt, also in Einheiten von 10ns.

Die Ausgabe läuft ab wie folgt:

- Der 100MHz-Zähler wird alle 10ns um 1 hochgezählt.
- Wenn der Zählerstand gleich dem Zeitstempel des aktuellen Wertepaars im FIFO ist, wird das Bitmuster auf den festgelegten Kanälen ausgegeben.
- Wenn ein Bitmuster ausgegeben wurde, wird das Wertepaar aus dem FIFO gelöscht.
- Die Wertepaare werden in der Reihenfolge abgearbeitet, wie sie in den FIFO geschrieben wurden.

Es gilt daher:

Ein Zeitstempel definiert den Ausgabepunkt und zwar in Einheiten von 10ns. Der Wert kann auf zwei Weisen angegeben werden:

- als Absolutwert mit Bezug zum Start des 100MHz-Zählers mit **Digout_Fifo_Start**.
Bei einem Zeitstempel von 152 wird das zugehörige Bitmuster genau 1,52µs nach Start des 100MHz-Zählers ausgegeben.
- als Relativwert mit Bezug zum vorherigen Zeitstempel.
Bei einem Zeitstempel von 152 wird das zugehörige Bitmuster genau 1,52µs nach der Ausgabe des vorherigen Bitmusters ausgegeben.

stempel müssen in aufsteigender Reihenfolge abgelegt werden.

Der FIFO muss immer so mit Daten gefüllt werden, dass der jeweils nächste Ausgabepunkt in der Zukunft liegt. Wenn der FIFO jedoch einmal leer läuft, ist Folgendes zu beachten:

- Bei Absolutwerten muss der Zeitstempel größer sein als der aktuelle Zählerstand. Anderenfalls wird die Flankenausgabe „verpasst“ und erst nach einem zusätzlichen Zählerumlauf von etwa 43 Sekunden ausgeführt.
- Bei Relativwerten muss der Zeitstempel größer sein als der raum seit dem vorherigen Ausgabepunkt (als der FIFO leer lief). Gelingt das nicht, wird das Bitmuster sofort ausgegeben (jedoch offensichtlich verspätet); der nächste Zeitstempel bezieht sich auf den verspäteten Ausgabepunkt.

Siehe auch

[Conf_DIO](#), [Digin](#), [Digout](#), [Digout_Bits1](#), [Digout_Bits2](#), [Digin_Fifo_Enable](#), [Digout_Fifo_Read_Timer](#), [Digout_Fifo_Clear](#), [Digout_Fifo_Enable](#), [Digout_Fifo_Empty](#), [Digout_Fifo_Mode](#), [Digout_Fifo_Start](#)

Gültig für

X-A20+DCT

Beispiel

siehe [Digout_Fifo_Mode](#)

16.4 Zähler

Dieser Abschnitt beschreibt folgende Befehle:

- [Cnt_Clear](#) (Seite 716)
- [Cnt_Enable](#) (Seite 116)
- [Cnt_PW_Enable](#) (Seite 117)
- [Cnt_Get_Status](#) (Seite 118)
- [Cnt_Latch](#) (Seite 120)
- [Cnt_Mode](#) (Seite 121)
- [Cnt_Read](#) (Seite 123)
- [Cnt_PW_Latch](#) (Seite 124)
- [Cnt_Read_Int_Register](#) (Seite 125)
- [Cnt_Get_PW](#) (Seite 126)
- [Cnt_Get_PW_HL](#) (Seite 127)
- [Cnt_Read_Latch](#) (Seite 128)
- [Cnt_Sync_Latch](#) (Seite 129)
- [SSI_Mode](#) (Seite 132)
- [SSI_Read](#) (Seite 133)
- [SSI_Set_Bits](#) (Seite 134)
- [SSI_Set_Clock](#) (Seite 135)
- [SSI_Start](#) (Seite 137)
- [SSI_Status](#) (Seite 138)

Cnt_Clear setzt einen oder mehrere Vor-/ Rückwärtszähler auf Null, gemäß dem Bitmuster in **pattern**.

Syntax

```
#Include ADwin-X.inc
```

```
Cnt_Clear(pattern)
```

Parameter

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

| Bitnr. | 31:7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------|------|---|---|---|---|---|---|---|
| Zähler-Nr. | – | 7 | 6 | 5 | 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** den Löschmodus mit 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.

Mit **Sync_All** können Sie alle 7 Zähler (und die 7 PWM-Zähler) auf einmal auf Null setzen.

Siehe auch

[Cnt_Enable](#), [Cnt_Get_Status](#), [Cnt_Latch](#), [Cnt_Mode](#), [Cnt_Read](#), [Cnt_Read_Int_Register](#), [Cnt_Read_Latch](#), [Cnt_Sync_Latch](#), [Sync_All](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
Cnt_Enable(0)           'alle Zähler stoppen
Cnt_Mode(1,0b)          'Zähler 1 Takt-Richtung
Cnt_Mode(2,0b)          'Zähler 2 Takt-Richtung
Cnt_Clear(11b)          'Zähler 1+2 auf 0 zurücksetzen
Cnt_Enable(11b)         'Zähler 1+2 starten
```

Event:

```
Cnt_Latch(11b)          'Zähler 1+2 gleichzeitig latchen
Par_1 = Cnt_Read_Latch(1) 'Latch Zähler 1 und ...
Par_2 = Cnt_Read_Latch(2) 'Latch Zähler 2 auslesen
```

Cnt_Clear

Cnt_Enable

Cnt_Enable hält die gewählten Vor-/ Rückwärtszähler an oder gibt sie frei, um eingehende Impulse zu zählen.

Syntax

```
#Include ADwin-X.inc
```

```
Cnt_Enable(pattern)
```

Parameter

| | |
|----------------|---|
| pattern | Bitmuster _LONG Bit = 0: Zähler anhalten Bit = 1: Zähler freigeben |
|----------------|---|

| Bitnr. | 31:7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------|------|---|---|---|---|---|---|---|
| Zähler-Nr. | – | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Bemerkungen

PWM-Zähler werden mit **Cnt_PW_Enable** angehalten oder freigegeben.

Bei den Zählern 1, 2 und 3 müssen Sie die Eingänge mit **Conf_DIO** als Digitaleingänge konfigurieren, damit die Zähler arbeiten können.

Siehe auch

[Cnt_Clear](#), [Cnt_Get_Status](#), [Cnt_Latch](#), [Cnt_Mode](#), [Cnt_Read](#), [Cnt_Read_Int_Register](#), [Cnt_Read_Latch](#), [Cnt_Sync_Latch](#), [Cnt_Sync_Latch](#), [Conf_DIO](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
Cnt_Enable(0)           'alle Zähler stoppen
Cnt_Mode(1,0b)          'Zähler 1 Takt-Richtung
Cnt_Mode(2,0b)          'Zähler 2 Takt-Richtung
Cnt_Clear(11b)          'Zähler 1+2 auf 0 zurücksetzen
Cnt_Enable(11b)         'Zähler 1+2 starten
```

Event:

```
Cnt_Latch(11b)          'Zähler 1+2 gleichzeitig latchen
Par_1 = Cnt_Read_Latch(1) 'Latch Zähler 1 und ...
Par_2 = Cnt_Read_Latch(2) 'Latch Zähler 2 auslesen
```

Cnt_PW_Enable hält die gewählten PWM-Zähler an oder gibt sie frei, um eingehende Impulse zu zählen.

Syntax

```
#Include ADwin-X.inc

Cnt_PW_Enable(pattern)
```

Parameter

pattern Bitmuster _LONG |
 Bit = 0: Zähler anhalten
 Bit = 1: Zähler freigeben

| Bitnr. | 31:7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----------------|------|---|---|---|---|---|---|---|
| PWM-Zähler-Nr. | – | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Bemerkungen

Vor-/ Rückwärtszähler werden mit **Cnt_Enable** angehalten oder freigegeben.
 Der PWM-Zählereingang wird mit **Cnt_Mode** festgelegt.

Siehe auch

[Cnt_Clear](#), [Cnt_Get_Status](#), [Cnt_Mode](#), [Cnt_PW_Latch](#), [Cnt_Read_Int_Register](#), [Cnt_Get_PW](#), [Cnt_Get_PW_HL](#), [Cnt_Sync_Latch](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc

Init:
    Cnt_PW_Enable(0)           'alle PWM-Zähler stoppen
    Cnt_Mode(1,0b) 'Zähler 1 Takt-Richtung, PWM-Eingang CLK
    Cnt_Mode(2,0b) 'Zähler 2 Takt-Richtung, PWM-Eingang CLK
    Cnt_Clear(11b)            'Zähler 1+2 auf 0 zurücksetzen
    Cnt_PW_Enable(11b)        'Zähler 1+2 starten

Event:
    Cnt_PW_Latch(11b)         'Zähler 1+2 gleichzeitig latches
    Cnt_Get_PW_HL(1, Par_1, Par_2) 'High-/Low- lesen
    Cnt_Get_PW_HL(2, Par_11, Par_12)
    Cnt_Get_PW(1, FPar_1, FPar_2) 'Frequenz, Taktverhältnis lesen
    Cnt_Get_PW(2, FPar_11, FPar_12)
```

Cnt_PW_Enable

Cnt_Get_Status

Cnt_Get_Status gibt den Inhalt des Statusregisters für einen Zählerblock zurück.

Syntax

```
#Include ADwin-X.inc
```

```
ret_val = Cnt_Get_Status(counter_no)
```

Parameter

counter_no Nummer des Zählerblocks: 1...7. LONG |

ret_val Inhalt des Statusregisters für den Zähler: Hinweise auf mögliche Fehlerquellen. LONG |

Bedeutung der Bits 0...4 siehe Tabelle.

| 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_Enable](#), [Cnt_Get_PW](#), [Cnt_Mode](#), [Cnt_Read](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc
Dim error As Long

Init:
    Cnt_Enable(0)           'Zähler stoppen
    Rem Zähler 1: Modus Takt-Richtung
    Cnt_Mode(1,0)
    Cnt_Clear(1b)           'Zähler 1 auf 0 zurücksetzen
    Cnt_Enable(1b)          'Zähler 1 starten
    error = 0               'Fehlerindikator zurücksetzen

Event:
    PAR_1 = Cnt_Read(1)     'Zähler 1 lesen
    PAR_2 = Cnt_Get_Status(1) And 11111b 'Status
    REM Leitungs- bzw. Kabelfehler Zähler 1?
    If (PAR_2 AND 01000b = 01000b) Then
        REM Anzahl Leitungs- bzw. Kabelfehler
        Inc PAR_3
        error = 1           'Fehlerindikator setzen
    EndIf
    REM Korrelationsfehler Zähler 1?
    If (PAR_2 And 10000b = 10000b) Then
        Inc PAR_4           'Anzahl Korrelationsfehler
        error = 1           'Fehlerindikator setzen
    EndIf
    REM Zustand Eingang CLR
    PAR_5 = Shift_Right(PAR_2 And 100b,2)
    REM Zustand Eingang A
    PAR_6 = Shift_Right(PAR_2 And 10b,1)
    REM Zustand Eingang B
    PAR_7 = PAR_2 And 1b
```

Cnt_Latch

Cnt_Latch überträgt den aktuellen Zählerstand eines oder mehrerer Vor-/ Rückwärtszähler in das zugehörige Latch, je nach Bitmuster in **pattern**.

Syntax

```
#Include ADwin-X.inc
```

```
Cnt_Latch(pattern)
```

Parameter

pattern Bitmuster _LONG |
 Bit = 0: keine Funktion
 Bit = 1: Zählerstand in Latch übertragen

| Bitnr. | 31:6 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------|------|---|---|---|---|---|---|---|
| Zähler-Nr. | – | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Bemerkungen

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

Das Latch wird mit **Cnt_Read_Latch** in eine Variable ausgelesen.

Für PWM-Zähler verwenden Sie **Cnt_PW_Latch**. Um den Stand mehrerer Zähler synchron in das Latch zu übertragen, verwenden Sie **Cnt_Sync_Latch**.

Siehe auch

[Cnt_Clear](#), [Cnt_Enable](#), [Cnt_Get_Status](#), [Cnt_PW_Latch](#), [Cnt_Mode](#), [Cnt_Read](#), [Cnt_Read_Latch](#), [Cnt_Sync_Latch](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
Cnt_Enable(0)           'alle Zähler stoppen
Cnt_Mode(1,0b)          'Zähler 1 Takt-Richtung
Cnt_Mode(2,0b)          'Zähler 2 Takt-Richtung
Cnt_Clear(11b)          'Zähler 1+2 auf 0 zurücksetzen
Cnt_Enable(11b)         'Zähler 1+2 starten
```

Event:

```
Cnt_Latch(11b)          'Zähler 1+2 gleichzeitig latchen
Par_1 = Cnt_Read_Latch(1) 'Latch Zähler 1 und ...
Par_2 = Cnt_Read_Latch(2) 'Latch Zähler 2 auslesen
```

Cnt_Mode definiert die Betriebsart eines Zählerblocks.

Syntax

```
#Include ADwin-X.inc

Cnt_Mode (cnt_no, pattern)
```

Parameter

| | | |
|----------------|---|------|
| cnt_no | Nummer des Zählerblocks: 1...7. | LONG |
| pattern | Bitmuster zur Einstellung des Betriebsmodus des Zählerblocks. | LONG |

| Bitnr. | Bedeutung |
|-----------|---|
| Bit 0 | Modus des Vor-/ Rückwärtszählers: Bit = 0: Takt-Richtungs-Modus. Bit = 1: Vier-Flanken-Auswertung (A-B-Modus). |
| Bit 1 | Löschmodus: Bit = 0: TTL-Pegel high am Eingang CLR setzt den Vor-/ Rückwärtszähler auf Null. Bit = 1: Zähler löschen, wenn an allen Eingänge A, B, CLR der TTL-Pegel high anliegt. Nur mit Vier-Flanken-Auswertung. |
| 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 / LTC einstellen. Bit = 0: Eingang CLR. Bit = 1: Eingang LTC. |
| Bit 5 | Eingang CLR / LTC freigeben. Bit = 0: Eingang ist gesperrt. Bit = 1: Eingang ist freigegeben. |
| Bit 6 | Auswahl der Signalflanke für PWM-Zähler. Bit = 0: steigende Flanke. Bit = 1: fallende Flanke. |
| Bit 7,8 | Auswahl eines Eingangs für PWM-Zähler. 00b: Eingang A / CLK 01b: Eingang B / DIR 10b: Eingang CLR / LTC |
| Bits 31:9 | reserviert. |

Bemerkungen

Verwenden Sie **Cnt_Mode** möglichst nur bei gesperrtem Zählerblock, siehe **Cnt_Enable** und **Cnt_PW_Enable**.

Bei den Zählern 1, 2 und 3 müssen Sie die Eingänge mit **Conf_DIO** als Digitaleingänge konfigurieren, damit die Zähler arbeiten können.

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.

Beachten Sie, dass mit **Digin_Filter_Init** Fehlpulse in Eingangssignalen unterdrückt werden können.

Siehe auch

[Cnt_Clear](#), [Cnt_Enable](#), [Cnt_PW_Enable](#), [Cnt_Get_Status](#), [Digin_Filter_Init](#), [Conf_DIO](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Cnt_Mode

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
Cnt_Enable(0)           'alle Zähler stoppen  
Cnt_Mode(1,0b)          'Zähler 1 Takt-Richtung  
Cnt_Mode(2,0b)          'Zähler 2 Takt-Richtung  
Cnt_Clear(11b)          'Zähler 1+2 auf 0 zurücksetzen  
Cnt_Enable(11b)         'Zähler 1+2 starten
```

Event:

```
Cnt_Latch(11b)          'Zähler 1+2 gleichzeitig latchen  
Par_1 = Cnt_Read_Latch(1) 'Latch Zähler 1 und ...  
Par_2 = Cnt_Read_Latch(2) 'Latch Zähler 2 auslesen
```


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

Syntax

```
#Include ADwin-X.inc

ret_val = Cnt_Read(counter_no)
```

Parameter

| | | |
|------------|---|------|
| counter_no | Nummer des Vor-/ Rückwärtszählers: 1...7. | 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_Sync_Latch](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc

Init:
    Cnt_Enable(0)                'alle Zähler stoppen
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    Cnt_Mode(1,100000b)
    Rem Zähler 2: Modus Takt-Richtung, LTC freigeben
    Cnt_Mode(2,110000b)
    Cnt_Clear(11b)               'Zähler 1+2 auf 0 zurücksetzen
    Cnt_Enable(11b)              'Zähler 1+2 starten

Event:
    Par_1 = Cnt_Read(1) 'Zähler 1 und ...
    Par_2 = Cnt_Read(2) 'Zähler 2 auslesen
```

Cnt_Read

Cnt_PW_Latch

Cnt_PW_Latch kopiert den Inhalt eines oder mehrerer PWM-Zähler in einen Zwischenspeicher.

Syntax

```
#Include ADwin-X.inc

Cnt_PW_Latch(pattern)
```

Parameter

pattern Bitmuster _LONG |
 Bit = 0: Kein Einfluss.
 Bit = 1: PWM-Zählerinhalt latchen.

| Bitnr. | 31:6 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------|------|---|---|---|---|---|---|---|
| Zähler-Nr. | – | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Bemerkungen

Der Zwischenspeicher wird mit **Cnt_Get_PW** oder **Cnt_Get_PW_HL** ausgelesen.

Siehe auch

[Cnt_Clear](#), [Cnt_PW_Enable](#), [Cnt_Get_Status](#), [Cnt_Mode](#), [Cnt_Read_Int_Register](#), [Cnt_Get_PW](#), [Cnt_Get_PW_HL](#), [Cnt_Sync_Latch](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc

Init:
    Cnt_PW_Enable(0)                'alle Zähler stoppen
    Rem Zähler 1+2: Modus Takt-Richtung, PWM-Msg. am Eingang CLK
    Cnt_Mode(1,0)
    Cnt_Mode(2,0)
    Cnt_PW_Enable(11b)             'PWM-Zähler 1+2 starten

Event:
    Cnt_PW_Latch(11b)              'Zähler 1+2 gleichzeitig latchen
    REM High-/Low- lesen
    Cnt_Get_PW_HL(1,Par_1,Par_2)
    Cnt_Get_PW_HL(2,Par_11,Par_12)
    REM Frequenz und Taktverhältnis lesen
    Cnt_Get_PW(1,FPar_1,FPar_2)
    Cnt_Get_PW(2,FPar_11,FPar_12)
```

Cnt_Read_Int_Register gibt den Inhalt eines Zählerregisters zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = Cnt_Read_Int_Register(counter_no, reg_no)
```

Parameter

| | | |
|------------|--|------|
| counter_no | Nummer des Zählerblocks: 1...7. | 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 Vor-/ Rückwärtszähler. |
| 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 im Handbuch *ADwin-X*.

Siehe auch

[Cnt_Get_PW](#), [Cnt_Get_PW_HL](#), [Cnt_PW_Latch](#), [Cnt_Sync_Latch](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

siehe [Cnt_Sync_Latch](#)

Cnt_Read_Int_Register

Cnt_Get_PW

Cnt_Get_PW gibt Frequenz und Tastverhältnis eines PWM-Zählers zurück.

Syntax

```
#Include ADwin-X.inc
```

```
Cnt_Get_PW(pwm_no, frequency, dutycycle)
```

Parameter

| | | |
|-----------|--|-------|
| pwm_no | Nummer (1...7) des PWM-Zählers. | LONG |
| frequency | Frequenz in Hertz: 0,023 Hz ...20MHz. | FLOAT |
| | | CONST |
| dutycycle | Tastverhältnis in Prozent (0.0...100.0). | FLOAT |
| | | CONST |

Bemerkungen

Verwenden Sie zuerst **Cnt_PW_Latch**, um aktuelle Werte zu erhalten.

Die Rückgabewerte werden in den Parametern **frequency** und **dutycycle** übergeben.

Siehe auch

[Cnt_Clear](#), [Cnt_PW_Enable](#), [Cnt_Get_Status](#), [Cnt_Mode](#), [Cnt_PW_Latch](#), [Cnt_Read_Int_Register](#), [Cnt_Get_PW_HL](#), [Cnt_Sync_Latch](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
Cnt_PW_Enable(0)           'alle Zähler stoppen
Rem Zähler 1+2: Modus Takt-Richtung, PWM-Msg. am Eingang CLK
Cnt_Mode(1,0)
Cnt_Mode(2,0)
Cnt_PW_Enable(11b)         'PWM-Zähler 1+2 starten
```

Event:

```
Cnt_PW_Latch(11b)          'Zähler 1+2 gleichzeitig latches
REM High-/Low- lesen
Cnt_Get_PW_HL(1,Par_1,Par_2)
Cnt_Get_PW_HL(2,Par_11,Par_12)
REM Frequenz und Taktverhältnis lesen
Cnt_Get_PW(1,FPar_1,FPar_2)
Cnt_Get_PW(2,FPar_11,FPar_12)
```

Cnt_Get_PW_HL gibt die Eintast und die Austast eines PWM-Zählers zurück.

Syntax

```
#Include ADwin-X.inc

Cnt_Get_PW_HL(counter_no, hightime, lowtime)
```

Parameter

| | | |
|-------------------|---|---------------|
| counter_no | Nummer (1...7) des PWM-Zählers. | LONG |
| hightime | Eintast (Pegel High) des PWM-Signals in Einheiten von 10ns. | LONG CONST |
| lowtime | Austast (Pegel Low) des PWM-Signals in Einheiten von 10ns. | LONG CONST |

Bemerkungen

Verwenden Sie zuerst **Cnt_PW_Latch**, um aktuelle Werte zu erhalten.

Die Rückgabewerte werden in den Parametern **hightime** und **lowtime** übergeben.

Siehe auch

[Cnt_Clear](#), [Cnt_PW_Enable](#), [Cnt_Get_Status](#), [Cnt_Mode](#), [Cnt_PW_Latch](#), [Cnt_Read_Int_Register](#), [Cnt_Get_PW](#), [Cnt_Sync_Latch](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc

Init:
    Cnt_PW_Enable(0)           'alle Zähler stoppen
    Rem Zähler 1+2: Modus Takt-Richtung, PWM-Msg. am Eingang CLK
    Cnt_Mode(1,0)
    Cnt_Mode(2,0)
    Cnt_PW_Enable(11b)        'PWM-Zähler 1+2 starten

Event:
    Cnt_PW_Latch(11b)         'Zähler 1+2 gleichzeitig latches
    Rem High-/Low- lesen
    Cnt_Get_PW_HL(1,Par_1,Par_2)
    Cnt_Get_PW_HL(2,Par_11,Par_12)
    Rem Frequenz und Taktverhältnis lesen
    Cnt_Get_PW(1,FPar_1,FPar_2)
    Cnt_Get_PW(2,FPar_11,FPar_12)
```

Cnt_Get_PW_HL

Cnt_Read_Latch

Cnt_Read_Latch gibt den Wert aus dem Latch eines Vor-/ Rückwärtszählers als Rückgabewert zurück.

Syntax

```
#Include ADwin-X.inc  
  
ret_val = Cnt_Read_Latch(counter_no)
```

Parameter

| | | |
|------------|-------------------------------|------|
| counter_no | Zählernummer: 1...7. | LONG |
| ret_val | Inhalt des Latch 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_Sync_Latch](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

Beispiel

```
#Include ADwin-X.inc  
  
Init:  
  Cnt_Enable(0)                'alle Zähler stoppen  
  Rem Zähler 2 Takt-Richtung, Latch-Eingang freigeben  
  Cnt_Mode(2,110000b)  
  Cnt_Clear(10b)               'Zähler 2 auf 0 zurücksetzen  
  Cnt_Enable(10b)              'Zähler 2 starten  
  
Event:  
  Cnt_Latch(10b)               'Zähler 2 latchen  
  Par_10 = Cnt_Read_Latch(2)   'Latch Zähler 2 lesen
```

Cnt_Sync_Latch kopiert die Inhalte der gewählten Vor-/ Rückwärtszähler und PWM-Zähler in Zwischenspeicher.

Syntax

```
#Include ADwin-X.inc

Cnt_Sync_Latch(pattern)
```

Parameter

pattern Bitmuster _LONG |
 Bit = 0: Kein Einfluss.
 Bit = 1: Zählerinhalt in Zwischenspeicher kopieren.

| Bitnr. | 31:6 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------|------|---|---|---|---|---|---|---|
| Zähler-Nr. | – | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Bemerkungen

Jedem Bit sind sowohl ein Vor-/ Rückwärtszähler 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.

Mit **Sync_All** können Sie alle 7 Zähler und 7 PWM-Zähler auf einmal latchen.

Siehe auch

[Cnt_Get_PW](#), [Cnt_Latch](#), [Cnt_Mode](#), [Cnt_PW_Latch](#), [Cnt_Read_Int_Register](#), [Cnt_Read_Latch](#), [Sync_All](#)

Gültig für

X-A20+D, X-A20+DCT, X-A20+CO1

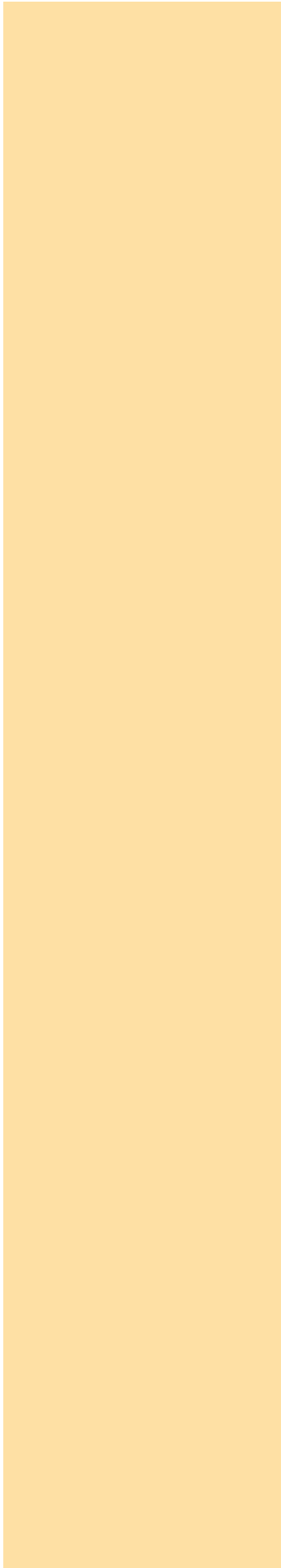
Beispiel

```
#Include ADwin-X.inc
#Define frequency PAR_1
Dim time, edges As Long
Dim pw, oldpw As Long
Dim vr, oldvr As Long

Init:
    Processdelay = 3000000          '200Hz with T12 processor
    Cnt_Enable(0)                  'counters off
    Cnt_PW_Enable(0)               'PWM counters off
    Cnt_Mode(1,00000000b)          'mode: clock/dir
    Cnt_Clear(0001b)               'clear counter 1
    Cnt_Enable(1b)                 'enable V/R 1
    Cnt_PW_Enable(1b)             'enable PWM 1
    Cnt_Sync_Latch(0001b)          'latch counter 1 (V/R + 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:
    Cnt_Sync_Latch(0001b)          'latch counter 1 (V/R + PWM)
    vr = Cnt_Read_Int_Register(1,6) 'V/R counter 1
    edges = Abs(vr - oldvr)        'number of edges between events
    If (edges <> 0) Then
        Rem get positive edges latch 1
        pw = Cnt_Read_Int_Register(1,8)
        time = pw - oldpw          'calculate time base
        Rem frequency: 100000000=timer frequency of CNT module
        frequency = edges * 100000000 / time
        oldvr = vr                 'store VR counter value
        oldpw = pw                 'store PW counter value
    EndIf
```

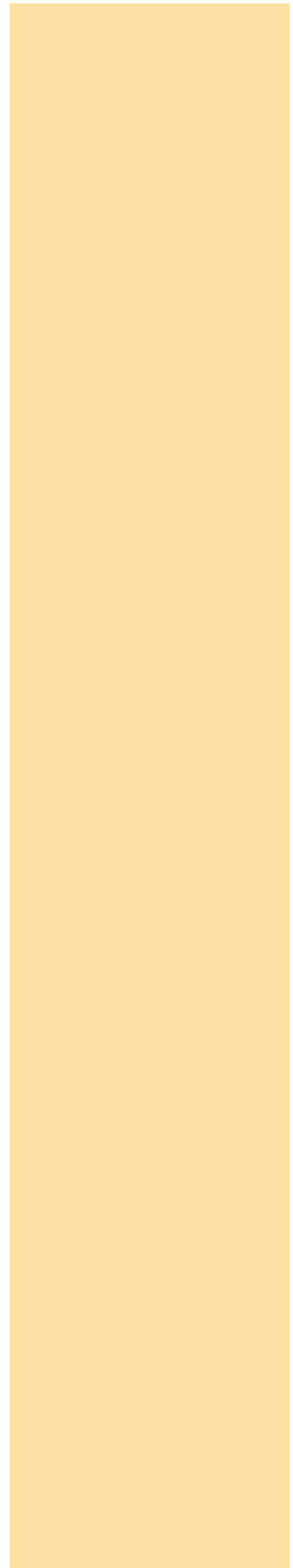
Cnt_Sync_Latch



16.5 SSI-Schnittstelle

Dieser Abschnitt beschreibt folgende Befehle:

- [SSI_Mode](#) (Seite 132)
- [SSI_Read](#) (Seite 133)
- [SSI_Set_Bits](#) (Seite 134)
- [SSI_Set_Clock](#) (Seite 135)
- [SSI_Set_Delay](#) (Seite 136)
- [SSI_Start](#) (Seite 137)
- [SSI_Status](#) (Seite 138)



SSI_Mode

SSI_Mode stellt den Modus des SSI-Decoders ein, entweder „single shot“ (einzelnen Wert lesen) und „continuous“ (kontinuierlich lesen).

Syntax

```
#Include ADwin-X.inc
```

```
SSI_Mode(pattern)
```

Parameter

| | |
|----------------|---|
| pattern | Betriebsmodus des SSI-Decoders. LONG |
| | 0: Modus „single shot“, der Decoder wird einmal ausgelesen. |
| | 1: Modus „continuous“, der Decoder wird kontinuierlich ausgelesen. |

Bemerkungen

Wenn Sie den Modus „continuous“ wählen, startet das Auslesen des entsprechenden Decoders sofort. **SSI_Start** ist hierzu nicht erforderlich. Mit **SSI_Set_Delay** stellen Sie den Zeitabstand zwischen dem Einlesen von zwei Encoder-Werten ein.

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.

Beachten Sie, dass mit **Digin_Filter_Init** Fehlpulse in Eingangssignalen unterdrückt werden können.

Siehe auch

[SSI_Read](#), [SSI_Set_Bits](#), [SSI_Set_Clock](#), [SSI_Set_Delay](#), [SSI_Start](#), [SSI_Status](#), [Digin_Filter_Init](#)

Gültig für

X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
SSI_Set_Clock(200)      'CLK (Taktrate) = 125 kHz
SSI_Set_Bits(1,23)      'Anzahl Bits = 23
SSI_Mode(1)             'Continuous-Modus
```

Event:

```
Par_1 = SSI_Read(1)     'Positionswert lesen
```

SSI_Read gibt den zuletzt gespeicherten Zählerstand des SSI-Decoders zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = SSI_Read(dcdr_no)
```

Parameter

| | | |
|----------------|---|------|
| dcdr_no | Nummer (1) des SSI-Decoders. | LONG |
| ret_val | Letzter Zählerstand des SSI-Decoders (= Absolutwert-Position des Encoders). | LONG |

Bemerkungen

Ein Encoder-Wert wird dann gespeichert, wenn die durch **SSI_Set_Bits** angegebene Anzahl von Bits eingelesen wurde.

Es wird immer diejenige Anzahl an Bits zurückgegeben, die mit der Anweisung **SSI_Set_Bits** eingestellt wurde, auch wenn dies nicht mit der Auflösung des Encoders übereinstimmt.

In diesem Fall ist der zurückgegebene Zählerstand abhängig vom Encoder (siehe Dokumentation des Herstellers). In der Regel gilt:

- Wenn der Encoder eine größere Auflösung besitzt, werden dessen überzählige niederwertigste Bits nicht genutzt.
- Besitzt der Encoder eine kleinere als die eingestellte Auflösung, wird für jedes fehlende höchstwertige Bit eine 0 (Null) gelesen.

Siehe auch

[SSI_Mode](#), [SSI_Set_Bits](#), [SSI_Set_Clock](#), [SSI_Set_Delay](#), [SSI_Start](#), [SSI_Status](#)

Gültig für

X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
Dim m, n, y As Long
```

Init:

```
SSI_Set_Clock(50)      'CLK (Taktrate) = 500 kHz
SSI_Set_Bits(1,23)     'Anzahl Bits = 23
SSI_Mode(1)            'Continuous-Mode setzen
```

Event:

```
Par_1 = SSI_Read(1)    'Positionswert lesen
Rem Falls es sich um einen Encoder mit Gray-Code handelt:
m = 0                  'Werte der letzten Wandlung 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
Par_9 = y              'Das Ergebnis der Gray-/Binär-
                        'Wandlung in Par_9
```

SSI_Read



SSI_Set_Bits

SSI_Set_Bits stellt für einen SSI-Decoder 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 ADwin-X.inc

SSI_Set_Bits(dcdr_no, no_bits)
```

Parameter

| | | |
|----------------|---|------|
| dcdr_no | Nummer (1) des SSI-Decoders. | LONG |
| no_bits | Anzahl (1...32) der zu lesenden Bits für einen Encoder-Wert (entspricht der Encoder-Auflösung). | LONG |

Bemerkungen

Die Auflösung (Anzahl der Bits) des SSI-Decoders sollte mit der Anzahl der zu übertragenden Bits übereinstimmen.

Es wird immer diejenige Anzahl an Bits für einen Encoder-Wert erwartet, die mit **SSI_Set_Bits** eingestellt wurde, auch wenn dies nicht mit der Auflösung des Encoders übereinstimmt. In diesem Fall ist der zurückgegebene Zählerstand abhängig vom Encoder (siehe Dokumentation des Herstellers). In der Regel gilt:

- Wenn der Encoder eine größere Auflösung besitzt, werden dessen überzählige niederwertigste Bits nicht genutzt.
- Besitzt der Encoder eine kleinere als die eingestellte Auflösung, wird für jedes fehlende höchstwertige Bit eine 0 (Null) gelesen.

Siehe auch

[SSI_Mode](#), [SSI_Read](#), [SSI_Set_Clock](#), [SSI_Set_Delay](#), [SSI_Start](#), [SSI_Status](#)

Gültig für

X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
SSI_Set_Clock(50)           'CLK (Taktrate) = 500 kHz
SSI_Mode(1)                 'set continuous mode
SSI_Set_Bits(1,10)          '10 Bits
```

Event:

```
Par_1 = SSI_Read(1)         'read value of decoder
```



SSI_Set_Clock stellt die Taktrate (6,1kHz bis 12,5MHz) ein, mit der der Decoder getaktet wird.

Syntax

```
#Include ADwin-X.inc

SSI_Set_Clock(dcdr_no, prescale)
```

Parameter

| | | |
|-----------------|---|------|
| dcdr_no | Nummer (1) des SSI-Decoders. | LONG |
| prescale | Teilerfaktor (2...4095) zur Einstellung der Taktrate nach der Formel: Taktrate = 25MHz / prescale . | LONG |

Bemerkungen

Nach dem Einschalten wird als Voreinstellung der Teilerfaktor 100 verwendet, das entspricht einer Taktrate von 250kHz.

Bei Teilerfaktoren über 4095 werden die niederwertigsten 12 Bits als Teilerfaktor verwendet.

Die mögliche Taktfrequenz ist abhängig von Kabellänge, Kabeltyp und den jeweils verwendeten Send- und Empfangsbausteinen des Encoders bzw. Decoders. Grundsätzlich gilt als Regel: Je höher die Taktfrequenz, desto kürzer die mögliche Kabellänge.

Siehe auch

[SSI_Mode](#), [SSI_Read](#), [SSI_Set_Bits](#), [SSI_Set_Delay](#), [SSI_Start](#), [SSI_Status](#)

Gültig für

X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
SSI_Set_Clock(10)      'CLK (Taktrate) = 2.5 MHz
SSI_Set_Bits(1,10)     '10 bits
SSI_Set_Delay(1,2500)  '50 µs
SSI_Mode(1)            'set continuous mode
```

Event:

```
Par_1 = SSI_Read(1)    'read value of decoder
```

SSI_Set_Clock

SSI_Set_Delay

SSI_Set_Delay stellt für einen bestimmten SSI-Zähler den Zeitabstand zwischen dem Einlesen von zwei Encoder-Werten ein.

Syntax

```
#Include ADwin-X.Inc

SSI_Set_Delay(dcdr_no, delay)
```

Parameter

| | |
|----------------|--|
| dcdr_no | Nummer (1) des SSI-Decoders, dessen Werte einzustellen ist. <small>LONG</small> |
| delay | abstand (1...65535) in Einheiten von 20ns; der einstellbare Bereich ist 20ns...1310,7µs. <small>LONG</small> |

Bemerkungen

Der Zeitabstand **delay** beginnt nach dem Einlesen eines Encoder-Werts und endet mit dem Einlesen des nächsten Encoder-Werts.

Nach dem Einschalten des Moduls wird als Voreinstellung der Wert 1250 verwendet, das entspricht 25µs.

Siehe auch

[SSI_Mode](#), [SSI_Read](#), [SSI_Set_Bits](#), [SSI_Set_Clock](#), [SSI_Start](#), [SSI_Status](#)

Gültig für

X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
SSI_Set_Clock(50)           'CLK (clock rate) = 1 MHz
SSI_Set_Delay(1,400)        'waiting time 8µs for decoder 1
SSI_Set_Bits(1,10)          '10 bits for decoder 1
SSI_Mode(1)                 'Set continuous-mode
```

Event:

```
Par_1 = SSI_Read(1)         'Read position value decoder 1
```

SSI_Start startet das Auslesen des SSI-Decoders (nur im Modus single shot).

Syntax

```
#Include ADwin-X.inc
SSI_Start(dcdr_no)
```

Parameter

| | | |
|----------------|------------------------------|------|
| dcdr_no | Nummer (1) des SSI-Decoders. | LONG |
|----------------|------------------------------|------|

Bemerkungen

Im Modus „continuous“ ist die 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.

Sie können das Auslesen auch mit **Sync_All** starten.

Siehe auch

[SSI_Mode](#), [SSI_Read](#), [SSI_Set_Bits](#), [SSI_Set_Clock](#), [SSI_Set_Delay](#), [SSI_Status](#), [Sync_All](#)

Gültig für

X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
SSI_Set_Clock(250)      'CLK (Taktrate) = 100 kHz
SSI_Mode(0)             'Single shot-Mode einstellen
SSI_Set_Bits(1,23)      'Anzahl Bits = 23
```

Event:

```
SSI_Start(1)            'Positionswert lesen
Do : Until (SSI_Status(1) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = SSI_Read(1)      'Positionswert auslesen
```

SSI_Start



SSI_Status

SSI_Status liefert den aktuellen Lese-Status des Decoders zurück.

Syntax

```
#Include ADwin-X.inc

ret_val = SSI_Status(dcdr_no)
```

Parameter

| | | |
|----------------|--|------|
| dcdr_no | Nummer (1) des SSI-Decoders. | 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_Set_Delay](#), [SSI_Start](#)

Gültig für

X-A20+D, X-A20+DCT

Beispiel

```
#Include ADwin-X.inc
```

Init:

```
SSI_Set_Clock(250)      'CLK (Taktrate) = 100 kHz
SSI_Mode(0)             'Single shot-Mode einstellen
SSI_Set_Bits(1,23)      'Anzahl Bits = 23
```

Event:

```
SSI_Start(1)            'Positionswert lesen
Do : Until (SSI_Status(1) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = SSI_Read(1)      'Positionswert auslesen und anzeigen
```


16.6 CAN-Schnittstelle

Dieser Abschnitt beschreibt zum Ansprechen der CAN-Schnittstellen auf *ADwin-X-A20-COM*:

- [CAN_Msg](#) (Seite 140)
- [CAN_Init](#) (Seite 141)
- [CAN_Receive](#) (Seite 142)
- [CAN_RX_Set_Filter](#) (Seite 144)
- [CAN_Transmit](#) (Seite 145)

CAN_Msg

CAN_Msg ist ein eindimensionales Feld mit 11 Elementen, in dem eine CAN-Nachricht beim Senden oder Empfangen zwischengespeichert wird.

Syntax

```
#Include ADwin-X.Inc  
  
CAN_Msg[n] = value  
oder  
value = CAN_Msg[n]
```

Parameter

| | | |
|--------------|--|------|
| n | Elementnummer im Feld CAN_Msg (1...11) | LONG |
| value | Byte-Wert (0...256) als Teil der CAN-Nachricht, siehe Tabelle. | LONG |

Bemerkungen

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

| Elementnr. in | Inhalt |
|------------------------|--|
| CAN_Msg [1...8] | CAN-Nachricht aus 1...8 Datenbytes |
| CAN_Msg [9] | Anzahl (0...8) belegter Datenbytes |
| CAN_Msg [10] | ID der CAN-Nachricht (11 Bit oder 29 Bit) |
| CAN_Msg [11] | Empfangs-stempel (16 Bit) |

Tragen Sie die zu übertragenden Datenbytes und ihre Anzahl sowie die ID in das Feld **CAN_Msg []** ein, *bevor* Sie diese mit **CAN_Transmit** übertragen. Beim Senden hat der Empfangs-stempel keine Bedeutung.

Siehe auch

[CAN_Init](#), [CAN_Receive](#), [CAN_RX_Set_Filter](#), [CAN_Transmit](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc  
  
Rem Sendet eine 32 Bit-FLOAT-Zahl (hier: Pi) als Folge  
Rem von 4 Bytes in einem Message-Objekt  
  
#Define pi 3.14159265  
Dim i As Long  
  
Init:  
Rem CAN-Controller 1 initialisieren, Baudrate 10kBaud  
Par_1 = CAN_Init(1,10000)  
If (Par_1 <> 0) Then Exit  
  
Rem Bitmuster von Pi mit Datenformat Long erzeugen  
Par_1 = Cast_Float32ToLong(pi)  
  
Rem Bitmuster (32 Bit) in 4 Bytes aufteilen  
CAN_Msg[4] = Par_1 And 0FFh 'LSB zuweisen  
For i = 1 To 3  
    CAN_Msg[4-i] = Shift_Right(Par_1,8*i) And 0FFh  
Next i  
CAN_Msg[9] = 4 'Länge der Nachricht in Bytes  
CAN_Msg[10] = 40 'ID  
  
Event:  
Rem Nachricht senden, niedrige Priorität, 11 Bit-ID  
Par_2 = CAN_Transmit(1,0,0)
```

CAN_Init initialisiert den Controller einer CAN-Schnittstelle.

Syntax

```
#Include ADwin-X.Inc
ret_val = CAN_Init(can_no, baudrate)
```

Parameter

| | | |
|-----------------|---|------|
| can_no | Nummer (1, 2) der CAN-Schnittstelle | LONG |
| baudrate | Baudrate des CAN-Controllers in Bit/Sekunde. | LONG |
| ret_val | Status der Befehlsausführung: -1: Keine CAN-Schnittstelle vorhanden.. 0: Baudrate wurde eingestellt. 1: Baudrate unzulässig. | LONG |

Bemerkungen

Die Anweisung führt bei der Initialisierung folgende Aktionen aus:

- Reset (Hardware-Reset des CAN-Controllers).
- Empfangs-FIFO und Send-FIFO leeren.
- Empfangsfilter deaktivieren (siehe **CAN_RX_Set_Filter**).
- Baudrate einstellen.

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](#), [CAN_Receive](#), [CAN_RX_Set_Filter](#), [CAN_Transmit](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc

Init:
  Rem Initialisiere CAN - Controller 1 mit 50000 Baud
  Par_1 = CAN_Init(1,50000)
  If (Par_1 <> 0) Then Exit
```

CAN_Init

CAN_Receive

CAN_Receive prüft, ob Nachrichten im Eingangs-FIFO eines CAN-Controllers empfangen wurden.

Falls ja, wird die älteste Nachricht aus dem FIFO in **CAN_Msg** gespeichert und der Identifier der Nachricht zurückgegeben.

Syntax

```
#Include ADwin-X.Inc  
  
ret_val = CAN_Receive(can_no)
```

Parameter

| | | |
|----------------|--|------|
| can_no | Nummer (1, 2) der CAN-Schnittstelle | LONG |
| ret_val | -1: keine CAN-Nachricht im FIFO. >0: Neue Nachricht; Wert = Identifier der Nachricht. | LONG |

Bemerkungen

Sie können eine empfangene Nachricht nur einmal auslesen, anschließend wird die Nachricht aus dem Empfangs-FIFO gelöscht.

Wenn mehr als 64 Nachrichten eingegangen sind, ohne dass Nachrichten abgeholt werden, werden die ältesten Daten im Eingangs-FIFO überschrieben, die dadurch unwiderruflich verloren sind. Achten Sie daher beim Programmieren darauf, dass die CAN-Nachrichten schneller ausgelesen als empfangen werden. Ein Datenverlust wird nicht angezeigt.

Die Länge der CAN-ID (11 Bit oder 29 Bit) wird nicht zurückgegeben.

Sie können mit **CAN_RX_Set_Filter** festlegen, dass nur CAN-Nachrichten mit bestimmten Identifiern empfangen werden.

Siehe auch

[CAN_Msg](#), [CAN_Init](#), [CAN_RX_Set_Filter](#), [CAN_Transmit](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc
Rem Wenn eine neue Nachricht mit dem passenden Identifier
Rem empfangen wurde, werden die Daten gelesen. Die
Rem ersten 4 Bytes der Nachricht werden zu einer Fließkomma-
Rem Zahl mit 32 Bit Länge zusammengesetzt.
Dim n As Long
Dim valuePi As Float32

INIT:
  PAR_1 = 0
  Rem Initialisiere CAN-Controller 1 mit 50000 Baud
  Par_1 = CAN_Init(1,50000)
  If (Par_1 <> 0) Then Exit

EVENT:
  Rem Wenn eine Nachricht empfangen wurde, werden die
  Rem empfangenen Daten gelesen und der Identifier an PAR_9
  Rem übergeben.
  Rem Die Daten stehen im Feld CAN_Msg[] bereit.
  PAR_9 = CAN_Receive(1)

  If (PAR_9 = 40) Then
    Rem neue Nachricht mit dem Identifier 40.
    Rem High-Byte auslesen, mit restlichen 3 Bytes zu
    Rem 32 Bit-Zahl zusammenfügen
    PAR_1 = CAN_Msg[1]
    For n = 2 TO 4
      PAR_1 = Shift_Left(PAR_1,8) + CAN_Msg[n]
    Next n
    Rem Bitmuster in PAR_1 in den Datentyp FLOAT32 wandeln.
    valuePi = Cast_LongToFloat32(PAR_1)
    FPAR_1 = valuePi
    PAR_10 = CAN_Msg[11]      'stempel (16 Bit)
  EndIf
```

Senden einer Fließkomma-Zahl siehe Bsp. bei [CAN_Transmit](#).

CAN_RX_Set_Filter

CAN_RX_Set_Filter legt einen Empfangsfilter für CAN-Nachrichten mit einem ausgewählten Identifier fest.

Syntax

```
#Include ADwin-X.Inc  
  
ret_val = CAN_RX_Set_Filter(channel, filter_no,  
                           filter_enable, id, id_extend)
```

Parameter

| | | |
|----------------------------|--|------|
| <code>channel</code> | Nummer (1, 2) der CAN-Schnittstelle. | LONG |
| <code>filter_no</code> | Nummer (1...4) des Filters. | LONG |
| <code>filter_enable</code> | Status des Filters: 0: Filter deaktivieren. 1: Filter aktivieren. | LONG |
| <code>id</code> | Identifier (0...2 ¹¹ oder 0...2 ²⁹) der CAN-Nachrichten, die empfangen werden können. | LONG |
| <code>id_extend</code> | Länge des Identifiers <code>id</code> : 0: 11 Bit 1: 29 Bit | LONG |
| <code>ret_val</code> | 0: Filter wurde konfiguriert. <>0:Fehler beim Konfigurieren. | LONG |

Bemerkungen

Sie sollten diese Anweisung im Prozessabschnitt **LowInit:** oder **Init:** ausführen.

Eine Nachricht kann bei aktivierten Filtern nur empfangen werden, wenn der Identifier der Nachricht und der Identifier des Filters genau gleich sind.

Für jede CAN-Schnittstelle können bis zu 4 Empfangsfilter festgelegt und aktiviert werden. Sobald eine CAN-Nachricht einen der aktiven Empfangsfilter erfolgreich durchlaufen hat, wird sie im Empfangs-FIFO gespeichert.

Siehe auch

[CAN_Msg](#), [CAN_Init](#), [CAN_Receive](#), [CAN_Transmit](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc  
  
INIT:  
  PAR_1 = 0  
  Rem Initialisiere CAN-Controller 1 mit 50000 Baud  
  Par_1 = CAN_Init(1,50000)  
  If (Par_1 <> 0) Then Exit  
  Rem Filter 3 auf Identifier 40 festlegen und freigeben  
  PAR_1 = CAN_RX_Set_Filter(1,3,1,40,0)  
  
EVENT:  
  REM auf Nachricht prüfen, nur ID 40 ist gültig  
  PAR_9 = CAN_Receive(1)
```

CAN_Transmit übergibt die Nachricht in **CAN_Msg** zum Senden an die CAN-Schnittstelle.

Syntax

```
#Include ADwin-X.Inc
CAN_Transmit(can_no, priority, id_extend)
```

Parameter

| | | |
|------------------|---|------|
| can_no | Nummer (1, 2) der CAN-Schnittstelle | LONG |
| priority | Sendepriorität der Nachricht: 0: Normale Priorität, Nachricht wird über Ausgangs-FIFO verschickt. 1: Hohe Priorität, Nachricht wird als nächste verschickt. | LONG |
| id_extend | Länge des Identifiers: 0: 11 Bit 1: 29 Bit | LONG |
| ret_val | 0: Nachricht ist an CAN-Schnittstelle übergeben. -1: Sendespeicher ist voll, Nachricht muss erneut übergeben werden. | LONG |

Bemerkungen

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

- Geben Sie die Nachricht in das Feld **CAN_Msg** ein: Die Datenbytes, Anzahl der Datenbytes und Identifier. Der Zeitstempel hat keine Funktion.
- Übergeben Sie die Nachricht mit **CAN_Transmit** an die CAN-Schnittstelle.
- Prüfen Sie, ob die Nachricht korrekt übergeben wurde.

Die CAN-Schnittstelle sendet die nächste Nachricht, sobald sie Zugriffsrecht auf den CAN-Bus hat. Eine Nachricht mit hoher Priorität wird bei der nächsten Gelegenheit versendet, auch wenn im Ausgangs-FIFO bereits Nachrichten mit normaler Priorität warten. Nachrichten mit normaler Priorität werden in der Reihenfolge verschickt, in der sie in den Ausgangs-FIFO geschrieben werden.

Siehe auch

[CAN_Msg](#), [CAN_Init](#), [CAN_Receive](#), [CAN_RX_Set_Filter](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc
Rem Sendet eine 32 Bit-FLOAT-Zahl (hier: Pi) als Folge von
Rem 4 Bytes in einem Message-Objekt
#define pi 3.14159265
Dim i As Long

Init:
Rem Initialisiere CAN-Controller 2 mit 50000 Baud
Par_1 = CAN_Init(2,50000)
If (Par_1 <> 0) Then Exit

Rem Bitmuster von Pi mit Datenformat Long erzeugen
PAR_1 = Cast_Float32ToLong(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
CAN_Msg[10] = 40 'Identifier der Nachricht

Event:
Par_2 = CAN_Transmit(2,0,0) 'Nachricht mit 11Bit-ID senden
Empfangen einer Fließkomma-Zahl siehe Bsp. bei CAN_Receive.
```

CAN_Transmit

Einstellbare Baudraten

| Einstellbare Baudraten [Bit/s] | | | | |
|--------------------------------|-------------|-------------------|-------------|-------------------|
| 1000000.0000 | 888888.8889 | 800000.0000 | 727272.7273 | 666666.6667 |
| 615384.6154 | 571428.5714 | 533333.3333 | 500000.0000 | 470588.2353 |
| 444444.4444 | 421052.6316 | 400000.0000 | 380952.3810 | 363636.3636 |
| 347826.0870 | 333333.3333 | 320000.0000 | 307692.3077 | 296296.2963 |
| 285714.2857 | 266666.6667 | 250000.0000 | 242424.2424 | 235294.1176 |
| 222222.2222 | 210526.3158 | 205128.2051 | 200000.0000 | 190476.1905 |
| 181818.1818 | 177777.7778 | 173913.0435 | 166666.6667 | 160000.0000 |
| 156862.7451 | 153846.1538 | 148148.1481 | 145454.5455 | 142857.1429 |
| 140350.8772 | 133333.3333 | 126984.1270 | 125000.0000 | 123076.9231 |
| 121212.1212 | 117647.0588 | 115942.0290 | 114285.7143 | 111111.1111 |
| 106666.6667 | 105263.1579 | 103896.1039 | 102564.1026 | 100000.0000 |
| 98765.4321 | 95238.0952 | 94117.6471 | 90909.0909 | 88888.8889 |
| 87912.0879 | 86956.5217 | 84210.5263 | 83333.3333 | 81632.6531 |
| 80808.0808 | 80000.0000 | 78431.3725 | 76923.0769 | 76190.4762 |
| 74074.0741 | 72727.2727 | 71428.5714 | 70175.4386 | 69565.2174 |
| 68376.0684 | 67226.8908 | 66666.6667 | 66115.7025 | 64000.0000 |
| 63492.0635 | 62500.0000 | 61538.4615 | 60606.0606 | 60150.3759 |
| 59259.2593 | 58823.5294 | 57971.0145 | 57142.8571 | 55944.0559 |
| 55555.5556 | 54421.7687 | 53333.3333 | 52631.5789 | 52287.5817 |
| 51948.0519 | 51282.0513 | 50000.0000 | 49689.4410 | 49382.7160 |
| 48484.8485 | 47619.0476 | 47337.2781 | 47058.8235 | 46783.6257 |
| 45714.2857 | 45454.5455 | 44444.4444 | 43956.0440 | 43478.2609 |
| 42780.7487 | 42328.0423 | 42105.2632 | 41666.6667 | 41025.6410 |
| 40816.3265 | 40404.0404 | 40000.0000 | 39215.6863 | 38647.3430 |
| 38461.5385 | 38277.5120 | 38095.2381 | 37037.0370 | 36363.6364 |
| 36199.0950 | 35714.2857 | 35555.5556 | 35087.7193 | 34782.6087 |
| 34632.0346 | 34482.7586 | 34188.0342 | 33613.4454 | 33333.3333 |
| 33057.8512 | 32921.8107 | 32388.6640 | 32258.0645 | 32000.0000 |
| 31746.0317 | 31620.5534 | 31372.5490 | 31250.0000 | 30769.2308 |
| 30651.3410 | 30303.0303 | 30075.1880 | 29629.6296 | 29411.7647 |
| 29304.0293 | 29090.9091 | 28985.5072 | 28673.8351 | 28571.4286 |
| 28070.1754 | 27972.0280 | 27777.7778 | 27681.6609 | 27586.2069 |
| 27210.8844 | 27027.0270 | 26936.0269 | 26755.8528 | 26666.6667 |
| 26315.7895 | 26143.7908 | 25974.0260 | 25806.4516 | 25641.0256 |
| 25396.8254 | 25078.3699 | 25000.0000 | 24844.7205 | 24767.8019 |
| 24691.3580 | 24615.3846 | 24390.2439 | 24242.4242 | 24024.0240 |
| 23809.5238 | 23668.6391 | 23529.4118 | 23460.4106 | 23391.8129 |
| 23255.8140 | 23188.4058 | 22988.5057 | 22857.1429 | 22792.0228 |
| 22727.2727 | 22408.9636 | 22222.2222 | 22160.6648 | 22038.5675 |
| 21978.0220 | 21739.1304 | 21680.2168 | 21621.6216 | 21505.3763 |
| 21390.3743 | 21333.3333 | 21276.5957 | 21220.1592 | 21164.0212 |
| 21052.6316 | 20833.3333 | 20779.2208 | 20671.8346 | 20512.8205 |
| 20460.3581 | 20408.1633 | 20202.0202 | 20050.1253 | 20000.0000 |
| 19851.1166 | 19753.0864 | 19704.4335 | 19656.0197 | 19607.8431 |
| 19512.1951 | 19323.6715 | 19230.7692 | 19138.7560 | 19047.6190 |

| Einstellbare Baudraten [Bit/s] | | | | |
|--------------------------------|------------|-------------------|-------------------|------------|
| 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 |
| 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 |

| Einstellbare Baudraten [Bit/s] | | | | |
|--------------------------------|-----------|-----------|------------------|------------------|
| 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 | |

16.7 RSxxx-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen der RSxxx-Schnittstellen auf *ADwin-X-A20-COM*:

- [RS_Init](#) (Seite 150)
- [RS_Read_Fifo](#) (Seite 151)
- [RS_Write_Fifo](#) (Seite 152)
- [RS_Write_Fifo_Full](#) (Seite 153)
- [RS_Write_Fifo_Empty](#) (Seite 154)

RS_Init

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 ADwin-X.Inc
```

```
RS_Init(channel, baud, parity, bits, stop, handshake)
```

Parameter

| | | |
|------------------|--|------|
| channel | Nummer (1) der RSxxx-Schnittstelle. | LONG |
| baud | Übertragungsgeschwindigkeit in Baud: 35 ... 115200. | LONG |
| parity | Anwendung von Prüf-Bits: 0: ohne Paritäts-Bit. 1: gerade Parität (even). 2: ungerade Parität (odd). | LONG |
| bits | Anzahl der Daten-Bits (6...8). | LONG |
| stop | Anzahl der Stopp-Bits 0: 1 Stopp-Bit. 1: 1½ Stopp-Bits. 2: 2 Stopp-Bits. | LONG |
| handshake | Übertragungs-Protokoll: 0: RS232, kein Handshake. | 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.

Siehe auch

[RS_Read_Fifo](#), [RS_Write_Fifo](#), [RS_Write_Fifo_Full](#), [RS_Write_Fifo_Empty](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc
```

Init:

```
Rem Schnittstelle initialisieren: 9600 Baud, ohne Parität,  
Rem 8 Datenbits, 1 Stoppbit, kein Hardware-handshake  
RS_Init(1, 9600, 0, 8, 0, 0)
```



RS_Read_Fifo liest einen Wert aus dem Empfangs-FIFO der Schnittstelle.

Syntax

```
#Include ADwin-X.Inc

ret_val = RS_Read_Fifo(channel)
```

Parameter

| | | |
|----------------|---|------|
| channel | Nummer (1) der RSxxx-Schnittstelle. | LONG |
| ret_val | Inhalt des Eingangs-FIFO: -1: FIFO ist leer ≥0: Übertragener Datenwert (0...255). | LONG |

Bemerkungen

Der Empfangs-FIFO hat Platz für insgesamt 64 Werte.

Siehe auch

[RS_Init](#), [RS_Write_Fifo](#), [RS_Write_Fifo_Full](#), [RS_Write_Fifo_Empty](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc

Init:
    Rem Schnittstelle 1 initialisieren: 9600 Baud, ohne Parität,
    Rem 8 Datenbits, 1 Stoppbit und kein Hardware-handshake
    RS_Init(1,9600,0,8,0,0)

Event:
    Rem Einen Wert aus dem FIFO holen. Wenn der FIFO leer ist,
    Rem wird -1 zurückgeliefert.
    PAR_1 = RS_Read_Fifo(1)
```

RS_Read_Fifo

RS_Write_Fifo

RS_Write_Fifo schreibt einen Wert in den Sende-FIFO der Schnittstelle.

Syntax

```
#Include ADwin-X.Inc

ret_val = RS_Write_Fifo(channel,value)
```

Parameter

| | | |
|----------------|--|------|
| channel | Nummer (1) der RSxxx-Schnittstelle. | 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.

Sie können mit **RS_Write_Fifo_Full** vorab prüfen, ob noch Platz im Sende-FIFO ist. Der Sende-FIFO hat Platz für insgesamt 64 Werte.

Siehe auch

[RS_Init](#), [RS_Read_Fifo](#), [RS_Write_Fifo_Full](#), [RS_Write_Fifo_Empty](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc
Dim val As Long
```

Init:

```
Rem Initialisierung von Schnittstelle 1 mit 9600 Baud,
Rem keine Parität, 8 Datenbits, 1 Stoppbit und
Rem kein Hardware-handshake
RS_Init(1,9600,0,8,0,0)
```

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 = RS_Write_Fifo(1,val)
```

RS_Write_Fifo_Full prüft, ob der Sende-FIFO der RSxxx-Schnittstelle voll ist.

Syntax

```
#Include ADwin-X.Inc

ret_val = RS_Write_Fifo_Full(channel)
```

Parameter

| | | |
|----------------|---|------|
| channel | Nummer (1) der RSxxx-Schnittstelle. | LONG |
| ret_val | Status, ob der Sende-FIFO voll ist: 0: Falsch, Sende-FIFO ist nicht voll. 0: Wahr, Sende-FIFO ist voll. | LONG |

Bemerkungen

Der Sende-FIFO hat Platz für insgesamt 64 Werte.

Siehe auch

[RS_Init](#), [RS_Read_Fifo](#), [RS_Write_Fifo](#), [RS_Write_Fifo_Empty](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc
Dim val As Long

Init:
    Rem Initialisierung von Schnittstelle 1 mit 9600 Baud,
    Rem keine Parität, 8 Datenbits, 1 Stoppbit und
    Rem kein Hardware-Handshake.
    RS_Init(1,9600,0,8,0,0)

Event:
    Rem Ist der FIFO nicht voll, wird val ins FIFO geschrieben.
    If (RS_Write_Fifo_Full(1) <> 1) Then
        PAR_1 = RS_Write_Fifo(1, val)
    EndIf
```

RS_Write_Fifo_Full

RS_Write_Fifo_Empty

RS_Write_Fifo_Empty prüft, ob der Sende-FIFO der RSxxx-Schnittstelle leer ist.

Syntax

```
#Include ADwin-X.Inc

ret_val = RS_Write_Fifo_Empty(channel)
```

Parameter

| | | |
|----------------|---|------|
| channel | Nummer (1) der RSxxx-Schnittstelle. | LONG |
| ret_val | Status, ob der Sende-FIFO leer ist: 0: Falsch, Sende-FIFO ist nicht leer. 1: Wahr, Sende-FIFO ist leer. | LONG |

Bemerkungen

Der Sende-FIFO hat Platz für insgesamt 64 Werte.

Siehe auch

[RS_Init](#), [RS_Read_Fifo](#), [RS_Write_Fifo](#), [RS_Write_Fifo_Full](#)

Gültig für

X-A20+COM

Beispiel

```
#Include ADwin-X.Inc
Dim val As Long
```

Init:

```
Rem Initialisierung von Schnittstelle 1 mit 9600 Baud,
Rem keine Parität, 8 Datenbits, 1 Stoppbit und
Rem kein Hardware-Handshake.
RS_Init(1,9600,0,8,0,0)
```

Event:

```
Rem Ist das FIFO leer, wird val ins FIFO geschrieben.
If (RS_Write_Fifo_Empty(1) = 1) Then
    PAR_1 = RS_Write_Fifo(1, val)
EndIf
Rem Warten bis Sende-FIFO leer ist, d.h. val gesendet ist.
Do : Until (RS_Write_Fifo_Empty(1) = 1)
```


16.8 Profibus-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen des Profibus-Knotens auf *ADwin-X-A20*:

- [Init_Profibus](#) (Seite 156)
- [Run_Profibus](#) (Seite 158)

Init_Profibus

Init_Profibus initialisiert den Profibus-Slave.

Syntax

```
#Include ADwin-X.inc
```

```
ret_val = Init_Profibus(dev_adr, in_mod_cnt, out_mod_cnt,  
work_arr[])
```

Parameter

| | | |
|--------------------------|---|---------------|
| <code>dev_adr</code> | Slave-Knotenadresse / Stationsadresse (1...125) auf dem Profibus. | LONG |
| <code>in_mod_cnt</code> | Größe (1, 2, 4, 8, 16, 32, 61) des Eingangs-Datenbereichs im Profibus-Slave in Doppelworten. | LONG |
| <code>out_mod_cnt</code> | Größe (1, 2, 4, 8, 16, 32, 61) des Ausgangs-Datenbereichs im Profibus-Slave in Doppelworten. | LONG |
| <code>work_arr[]</code> | Feld, das für den Betrieb des Profibus-Slave initialisiert wird. Das Feld muss mindestens <code>AB_WORK_ARR_LEN</code> (5000) Elemente haben. | ARRAY LONG |
| <code>ret_val</code> | Status zur Befehlsausführung: 0: Initialisierung war erfolgreich. -1: Fehler, die Größen der Datenbereiche sind falsch. | 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 (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.

Kleinere Datenbereiche beschleunigen den Datenverkehr über den Profibus-Bus.

Stationsadresse und Größe der Datenbereiche muss die gleichen sein wie bei der Projektierung des Profibus.

Siehe auch

[Run_Profibus](#)

Gültig für

- / -

Beispiel

```
#Include ADwin-X.inc
#Define node 2 'slave node address
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[1000] As Long
Dim in_arr[1000] As Long
Dim conf_arr[AB_WORK_ARR_LEN] As Long
Dim i As Long
Dim state As Long

LowInit:
    Processdelay = 3000000 'set to 100 Hz
    Rem init Profibus interface: input data area = 8 DWords
    Rem and output data area = 16 DWords
    Par_10 = Init_Profibus(node, 8, 16, conf_arr)

Event:
    Rem set data in out_arr[] to be transferred
    For i = 1 To 16
        out_arr[i] = (out_arr[i] + i)
    Next i

    Rem send and read data
    state = Run_Profibus(out_arr,in_arr,16,conf_arr)
    Par_2 = state

    Rem now process received data stored in in_arr[1..8];
    Rem in_arr[9..16] has been filled with unusable data,
    Rem in_arr[17..1000] remains unused here.
```

Run_Profibus

Run_Profibus tauscht Daten mit dem Profibus-Slave aus.

Syntax

```
#Include ADwin-X.inc
```

```
ret_val = Run_Profibus(out_pd_arr[], in_pd_arr[],  
pd_arr_len, work_arr[])
```

Parameter

| | | |
|---------------------------|--|---------------|
| <code>out_pd_arr[]</code> | Feld, aus dem der Profibus-Slave Daten liest (Anzahl <code>pd_arr_len</code>) und auf den Profibus-Bus schreibt. | ARRAY LONG |
| <code>in_pd_arr[]</code> | Feld, in das der Profibus-Slave Daten schreibt (Anzahl <code>pd_arr_len</code>), die vom Profibus-Bus gelesen werden. | ARRAY LONG |
| <code>pd_arr_len</code> | Anzahl der Doppelworte (1, 2, 4, 8, 16, 32, 61), die jeweils in <code>in_pd_arr[]</code> und <code>out_pd_arr[]</code> übergeben werden. | LONG |
| <code>work_arr[]</code> | Feld, das Daten für den Betrieb des Profibus-Slave enthält, siehe Init_Profibus . | ARRAY LONG |
| <code>ret_val</code> | Betriebszustand des Profibus-Slave: 0: Initialisierung. 2: Slave wartet auf Busstart durch den Master.. 4: Normaler Betrieb. | LONG |

Bemerkungen

Die Anzahl `pd_arr_len` wird sowohl für das Lesen als auch das Schreiben von Daten verwendet, auch wenn der Eingangsbereich mit einer anderen Größe initialisiert sein kann als der Ausgangsbereich. Die Felder `in_pd_arr[]` und `out_pd_arr[]` müssen wenigstens mit der Größe `pd_arr_len` deklariert sein.

In jedem Feldelement in `in_pd_arr[]` und `out_pd_arr[]` wird 1 Doppelwort gespeichert. Ein Doppelwort entspricht einem Wert vom Datentyp `Long`.

Siehe auch

[Init_Profibus](#)

Gültig für

- / -

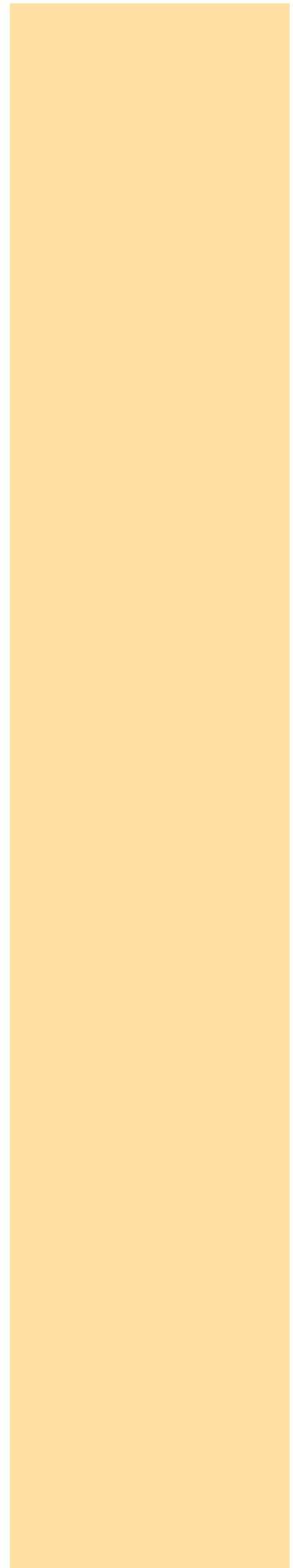
Beispiel

siehe [Init_Profibus](#)

16.9 Profinet-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen des Profinet-Knotens auf *ADwin-X-A20*:

- [Init_ProfinetIO](#) (Seite 160)
- [Run_ProfinetIO](#) (Seite 162)



Init_ProfinetIO

Init_ProfinetIO initialisiert ein Feld für den Betrieb mit dem Profinet-Slave.

Syntax

```
#Include ADwin-X.inc
```

```
ret_val = Init_ProfinetIO(in_size, out_size, work_arr[])
```

Parameter

| | | |
|-------------------------|---|---------------|
| <code>in_size</code> | Größe (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320) des Eingangsbereichs im Profinet-Slave in Doppelworten; 1 Doppelwort = 4 Byte. | LONG |
| <code>out_size</code> | Größe (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320) des Ausgangsbereichs im Profinet-Slave in Doppelworten; 1 Doppelwort = 4 Byte. | LONG |
| <code>work_arr[]</code> | Feld, das für den Betrieb des Profinet-Slave initialisiert wird. Das Feld muss mindestens <code>AB_WORK_ARR_LEN</code> (5000) Elemente haben. | ARRAY LONG |
| <code>ret_val</code> | Status zur Befehlsausführung: 0: Initialisierung war erfolgreich. -1: Fehler, die Größen der Datenbereiche sind falsch. | LONG |

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem Profinet-Slave ausgeführt werden.

Kleinere Datenbereiche beschleunigen den Datenverkehr über den Profinet-Bus.

Die Größe der Datenbereiche muss die gleiche 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).

Siehe auch

[Run_ProfinetIO](#)

Gültig für

- / -

Beispiel

```
#Include ADwin-X.inc
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[1000] As Long
Dim in_arr[1000] As Long
Dim conf_arr[AB_WORK_ARR_LEN] As Long
Dim i As Long
Dim state As Long

LowInit:
    Processdelay = 3000000 'set to 100 Hz
    Rem init Profinet interface: input data area = 128 DWords
    Rem and output data area = 256 DWords
    Par_10 = Init_ProfinetIO(128, 256, conf_arr)

Event:
    Rem set data in out_arr[] to be transferred
    For i = 1 To 256
        out_arr[i] = (out_arr[i] + i)
    Next i

    Rem send and read data
    state = Run_ProfinetIO(out_arr,in_arr,256,conf_arr)
    Par_2 = state

    Rem now process received data stored in in_arr[1..128];
    Rem in_arr[129..256] has been filled with unusable data,
    Rem in_arr[257..1000] remains unused here.
```

Run_ProfinetIO

Run_ProfinetIO tauscht Daten mit dem Profinet-Slave aus.

Syntax

```
#Include ADwin-X.inc

ret_val = Run_ProfinetIO(out_pd_arr[], in_pd_arr[],
    pd_arr_len, work_arr[])
```

Parameter

| | | |
|---------------------------|--|---------------|
| <code>out_pd_arr[]</code> | Feld, aus dem der Profinet-Slave Daten liest (Anzahl <code>pd_arr_len</code>) und auf den Profinet-Bus schreibt. | ARRAY LONG |
| <code>in_pd_arr[]</code> | Feld, in das der Profinet-Slave Daten schreibt (Anzahl <code>pd_arr_len</code>), die vom Profinet-Bus gelesen werden. | ARRAY LONG |
| <code>pd_arr_len</code> | Anzahl der Doppelworte (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320), die jeweils in <code>in_pd_arr[]</code> und <code>out_pd_arr[]</code> übergeben werden. | LONG |
| <code>work_arr[]</code> | Feld, das Daten für den Betrieb des Profinet-Slave enthält, siehe Init_ProfinetIO . | ARRAY LONG |
| <code>ret_val</code> | Betriebszustand des Profinet-Slave: 0: Initialisierung. 2: Slave wartet auf Busstart durch den Master.. 4: Normaler Betrieb. | LONG |

Bemerkungen

Die Anzahl `pd_arr_len` wird sowohl für das Lesen als auch das Schreiben von Daten verwendet, auch wenn der Eingangsbereich mit einer anderen Größe initialisiert sein kann als der Ausgangsbereich. Die Felder `in_pd_arr[]` und `out_pd_arr[]` müssen wenigstens mit der Größe `pd_arr_len` deklariert sein.

In jedem Feldelement in `in_pd_arr[]` und `out_pd_arr[]` werden 4 Datenbytes = 1 Doppelwort gespeichert. Ein Doppelwort entspricht einem Wert vom Datentyp `Long`.

Siehe auch

[Init_ProfinetIO](#)

Gültig für

- / -

Beispiel

siehe [Init_ProfinetIO](#)

16.10 EtherCAT-Schnittstelle

Dieser Abschnitt beschreibt Befehle zum Ansprechen des EtherCAT-Knotens auf *ADwin-X-A20*:

- [Init_EtherCAT](#) (Seite 164)
- [Run_EtherCAT](#) (Seite 166)

Init_EtherCAT

INIT_EtherCAT initialisiert ein Feld für den Betrieb mit dem EtherCAT-Slave.

Syntax

```
#Include ADwin-X.inc

ret_val = Init_EtherCAT(in_size, out_size, data_type,
                        work_arr[])
```

Parameter

| | | |
|-------------------------|--|---------------|
| <code>in_size</code> | Größe (0...360) des Eingangs-Datenbereichs im EtherCAT-Slave in Doppelworten. | LONG |
| <code>out_size</code> | Größe (0...360) des Ausgangs-Datenbereichs im EtherCAT-Slave in Doppelworten. | LONG |
| <code>data_type</code> | Datentyp für Eingangs- und Ausgangsbereich. Ein Wert belegt ein Doppelwort (=4 Byte). Verfügbar sind: AB_DATA_TYPE_SINT32: Signed integer, 32 Bit. AB_DATA_TYPE_FLOAT: Floating point, 32 Bit. | LONG |
| <code>work_arr[]</code> | Feld, das für den Betrieb des EtherCAT-Slave initialisiert wird. Das Feld muss mindestens AB_WORK_ARR_LEN (5000) Elemente haben. | ARRAY LONG |
| <code>ret_val</code> | Status zur Befehlsausführung: 0: Initialisierung war erfolgreich. -1: Fehler, die Größen der Datenbereiche sind falsch. | LONG |

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem EtherCAT-Slave ausgeführt werden.

Die Größe der Datenbereiche muss die gleiche sein wie bei der Projektierung des EtherCAT. Achten Sie darauf, dass die Größe der Datenbereiche bei der Projektierung auch in anderen Einheiten als Byte angegeben werden kann (z.B. Byte, Word).

Siehe auch

[Run_EtherCAT](#)

Gültig für

X-A20+ECAT

Beispiel

```
#Include ADwin-X.inc
#Define out_arr Data_2
#Define in_arr Data_3

'Data type according to Init_EtherCAT:
' AB_DATA_TYPE_SINT32 -> Long
' AB_DATA_TYPE_FLOAT -> Float32
Dim out_arr[1000] As Long
Dim in_arr[1000] As Long
Dim conf_arr[AB_WORK_ARR_LEN] As Long
Dim i As Long
Dim state As Long

LowInit:
    Processdelay = 3000000 'set to 100 Hz
    Rem init EtherCAT interface: input data area = 76 values
    Rem and output data area = 92 values
    Par_10 = Init_EtherCAT(76, 92, AB_DATA_TYPE_SINT32,
        conf_arr)

Event:
    Rem set data in out_arr[] to be transferred
    For i = 1 To 92
        out_arr[i] = (out_arr[i] + i)
    Next i

    Rem send and read data
    state = Run_EtherCAT(out_arr, in_arr, 92, conf_arr)
    Par_2 = state

    Rem now process received data stored in in_arr[1..76];
    Rem in_arr[77..92] has been filled with unusable data,
    Rem in_arr[93..1000] remains unused here.
```

Run_EtherCAT

Run_EtherCAT tauscht Daten mit dem EtherCAT-Slave aus.

Syntax

```
#Include ADwin-X.inc
```

```
ret_val = Run_EtherCAT(out_pd_arr[], in_pd_arr[],  
pd_arr_len, work_arr[])
```

Parameter

| | | |
|---------------------|---|------------------------|
| out_pd_arr[] | Feld, aus dem der EtherCAT-Slave Daten liest (Anzahl pd_arr_len) und auf den EtherCAT-Bus schreibt. | ARRAY LONG FLOAT |
| in_pd_arr[] | Feld, in das der EtherCAT-Slave Daten schreibt (Anzahl pd_arr_len), die vom EtherCAT-Bus gelesen werden. | ARRAY LONG FLOAT |
| pd_arr_len | Anzahl der Werte (1...360), die jeweils in in_pd_arr[] und out_pd_arr[] übergeben werden. | LONG |
| work_arr[] | Feld, das Daten für den Betrieb des EtherCAT-Slave enthält, siehe INIT_EtherCAT . | ARRAY LONG |
| ret_val | Betriebszustand des EtherCAT-Slave: 0: Initialisierung. 2: Slave wartet auf Busstart durch den Master.. 4: Normaler Betrieb. | LONG |

Bemerkungen

Die Anzahl **pd_arr_len** wird sowohl für das Lesen als auch das Schreiben von Daten verwendet, auch wenn der Eingangsbereich mit einer anderen Größe initialisiert sein kann als der Ausgangsbereich. Die Felder **in_pd_arr[]** und **out_pd_arr[]** müssen wenigstens mit der Größe **pd_arr_len** deklariert sein.

Deklarieren Sie die Felder **in_pd_arr[]** und **out_pd_arr[]** mit dem Datentyp, der zu der Einstellung mit **Init_EtherCAT**, Parameter **data_type** passt:

- **Long** für **AB_DATA_TYPE_SINT32**
- **Float32** für **AB_DATA_TYPE_FLOAT**

In jedem Feldelement in **in_pd_arr[]** und **out_pd_arr[]** werden 4 Datenbytes = 1 Doppelwort gespeichert.

Siehe auch

[Init_EtherCAT](#)

Gültig für

X-A20+ECAT

Beispiel

siehe [Init_EtherCAT](#)

16.11LS-Bus + ADwin-X-A20

Dieser Abschnitt beschreibt Befehle für die LS-Bus-Schnittstelle an ADwin-X-A20:

- [LS_DIO_Init](#) (Seite 168)
- [LS_DigProg](#) (Seite 170)
- [LS_Dig_IO](#) (Seite 172)
- [LS_Digout_Long](#) (Seite 174)
- [LS_Digout_Long_BS](#) (Seite 175)
- [LS_Digin_Long](#) (Seite 177)
- [LS_Digin_Long_BS](#) (Seite 178)
- [LS_Get_Output_Status](#) (Seite 180)
- [LS_Reset](#) (Seite 182)
- [LS_Watchdog_Init](#) (Seite 183)
- [LS_Watchdog_Reset](#) (Seite 185)

LS_DIO_Init

LS_DIO_Init initialisiert ein Modul vom Typ HSM-24V am LS-Bus und gibt den Fehlerstatus zurück.

Syntax

```
#Include ADwin-X.Inc

ret_val = LS_DIO_Init(ls_module)
```

Parameter

| | | |
|------------------|---|------|
| ls_module | Eingestellte Moduladresse (1...15) am HSM-Modul auf dem LS-Bus. | LONG |
| ret_val | Rückgabewert, das den Fehlerstatus angibt: -1: Keine Kommunikation mit dem Modul möglich. >0: Bitmuster mit mehreren Fehlerbits. Bit = 0: kein Fehler. Bit = 1: Fehler aufgetreten. | LONG |

| Bit-Nr. | 31...8 | 7 | 6 | 5...4 | 3 | 2 | 1 | 0 |
|---------|--------|-------|-------|-------|----|------|-----|-----|
| Status | – | Temp2 | Temp1 | – | WD | Time | Ovr | Par |

- :don't care (mit 0CFh ausmaskieren)

Par:Parity-Fehler bei der Datenübertragung auf dem LS-Bus.

Ovr: Overrun-Fehler bei der Datenübertragung auf dem LS-Bus.

Time: Timeout-Fehler bei der Datenübertragung auf dem LS-Bus.

WD: Watchdog hat ausgelöst. Die Kanaltreiber sind deaktiviert.

Temp1: Übertemperatur am Treiber für Kanäle 1...16. Treiber ist deaktiviert.

Temp2: Übertemperatur am Treiber für Kanäle 17...32. Treiber ist deaktiviert.

Bemerkungen

Die Anweisung soll nur im Abschnitt **INIT**: verwendet werden, weil sie eine lange Ausführungszeit hat.

Die Initialisierung setzt folgende Einstellungen:

- Alle DIO-Kanäle werden als Eingang programmiert.
Andere Einstellungen siehe **LS_DigProg**.
- Der Status für Überstrom (> ca. 500mA) wird zurückgesetzt.
- Der Fehlerstatus für Übertemperatur wird zurückgesetzt.
- Der Fehlerstatus für Timeout auf dem LS-Bus wird zurückgesetzt.
- Der Status des Watchdog-Zählers bleibt unverändert.

Das Modul speichert auftretende Fehler unabhängig vom ADwin-System. Fehlerbits im Rückgabewert können sich daher auf einen Fehler beziehen, der zu einem früheren Zeitpunkt aufgetreten ist.

Beachten Sie: Ignorieren Sie beim Programmstart das Fehlerbit WD so lange, bis der Watchdog-Zähler mit **LS_Watchdog_Init** konfiguriert ist. Der Watchdog-Zähler startet mit dem Einschalten des Moduls und hat bis zum Programmstart in der Regel schon einen Fehler ausgelöst.

Der Fehler „Übertemperatur“ eines Treibers kann nur auftreten, wenn auf mehreren Kanälen gleichzeitig ein Überstrom im Bereich von 150...500mA anliegt. Unabhängig davon wird bei einem Überstrom über 500mA der betroffene Kanal automatisch abgeschaltet.

Die Kanäle des Moduls HSM-24V dürfen nur im Bereich von 0...150mA betrieben werden. Damit ist sichergestellt, dass das Modul HSM-24V dauerhaft ohne Unterbrechung arbeitet, selbst wenn alle Kanäle gleichzeitig aktiv sind.

Siehe auch

[LS_DigProg](#), [LS_Dig_IO](#), [LS_Digout_Long](#), [LS_Digin_Long](#), [LS_Get_Output_Status](#), [LS_Reset](#), [LS_Watchdog_Init](#), [LS_Watchdog_Reset](#)

Gültig für

HSM-24V + X-A20



Beispiel

REM Example process for one module HSM-24V and ADwin-X

```
#Include ADwin-X.inc
```

Init:

```
Processdelay = 4000000    '10Hz HP
Par_1 = LS_DIO_Init(1)
REM enable watchdog time with 1.1 sec
Par_2 = LS_Watchdog_Init(1, 1, 1100)
REM LS channels 1...32 as outputs
Par_3 = LS_Digprog(1, 0Fh)
```

Event:

```
REM set one LS channel to high, rotating from 1 to 32
Inc Par_10
If (Par_10 >= 32) Then Par_10 = 0
Par_11 = Shift_Left(1, Par_10)
REM reset watchdog, set LS channels, and read back real state
REM (note: LS_Dig_IO uses LS address 1)
Par_12 = LS_Dig_IO(Par_11)
```

LS_DigProg

LS_DigProg programmiert die digitalen Kanäle 1...32 eines Moduls vom Typ HSM-24V am LS-Bus in Gruppen zu 8 als Ein- oder Ausgang.

Syntax

```
#Include ADwin-X.Inc

ret_val = LS_DigProg(ls_module, pattern)
```

Parameter

ls_module Eingestellte Moduladresse (1...15) am HSM-Modul auf dem LS-Bus. LONG |

pattern Bitmuster, nach dem die Kanäle als Ein- oder Ausgang gesetzt werden: LONG |
 Bit = 0: Kanal als Eingang setzen.
 Bit = 1: Kanal als Ausgang setzen.

| Bitnr. | 31...4 | 3 | 2 | 1 | 0 |
|----------|--------|-------|-------|------|-----|
| Kanalnr. | – | 32:25 | 24:17 | 16:9 | 8:1 |

ret_val Rückgabewert, das den Fehlerstatus angibt: LONG |
 -1: Keine Kommunikation mit dem Modul möglich.
 >0: Bitmuster mit mehreren Fehlerbits.
 Bit = 0: kein Fehler.
 Bit = 1: Fehler aufgetreten.

| Bit-Nr. | 31...8 | 7 | 6 | 5...4 | 3 | 2 | 1 | 0 |
|---------|--------|-------|-------|-------|----|------|-----|-----|
| Status | – | Temp2 | Temp1 | – | WD | Time | Ovr | Par |

- :don't care (mit **0CFh** ausmaskieren)

Par: Parity-Fehler bei der Datenübertragung auf dem LS-Bus.

Ovr: Overrun-Fehler bei der Datenübertragung auf dem LS-Bus.

Time: Timeout-Fehler bei der Datenübertragung auf dem LS-Bus.

WD: Watchdog hat ausgelöst. Die Kanaltreiber sind deaktiviert.

Temp1: Übertemperatur am Treiber für Kanäle 1...16. Treiber ist deaktiviert.

Temp2: Übertemperatur am Treiber für Kanäle 17...32. Treiber ist deaktiviert.

Bemerkungen

Die Anweisung soll nur im Abschnitt **Init**: verwendet werden, weil sie eine lange Ausführungszeit hat.

Nach der Initialisierung mit **LS_DIO_Init** sind alle Kanäle als Eingänge konfiguriert.

Die Kanäle können nur in Gruppen zu je 8 als Ein- oder Ausgang gesetzt werden (nur 4 relevante Bits, die anderen Bits werden ignoriert).

Siehe auch

[LS_DIO_Init](#), [LS_Dig_IO](#), [LS_Digout_Long](#), [LS_Digin_Long](#), [LS_Get_Output_Status](#), [LS_Reset](#), [LS_Watchdog_Init](#), [LS_Watchdog_Reset](#)

Gültig für

HSM-24V + X-A20

Beispiel

REM Example process for one module HSM-24V and ADwin-X

```
#Include ADwin-X.inc
```

Init:

```
Processdelay = 4000000    '10Hz HP
Par_1 = LS_DIO_Init(1)
REM enable watchdog time with 1.1 sec
Par_2 = LS_Watchdog_Init(1, 1, 1100)
REM LS channels 1...32 as outputs
Par_3 = LS_Digprog(1, 0Fh)
```

Event:

```
REM set one LS channel to high, rotating from 1 to 32
Inc Par_10
If (Par_10 >= 32) Then Par_10 = 0
Par_11 = Shift_Left(1, Par_10)
REM reset watchdog, set LS channels, and read back real state
REM (note: LS_Dig_IO uses LS address 1)
Par_12 = LS_Dig_IO(Par_11)
```

LS_Dig_IO

LS_Dig_IO setzt alle Digital-Ausgänge des Moduls HSM-24V am LS-Bus auf den Pegel High oder Low und gibt den Zustand aller Kanäle als Bitmuster zurück.

Syntax

```
#Include ADwin-X.Inc

ret_val = LS_Dig_IO(pattern)
```

Parameter

pattern Bitmuster, mit dem die digitalen Ausgänge gesetzt werden (siehe Tabelle).
Bit = 0: Ausgang auf Pegel Low setzen.
Bit = 1: Ausgang auf Pegel High setzen.

ret_val Bitmuster mit dem Ist-Zustand aller digitalen Kanäle (siehe Tabelle).
Bit = 0: Pegel Low liegt an.
Bit = 1: Pegel High liegt an.

| Bitnr. | 31 | 30 | 29 | ... | 2 | 1 | 0 |
|---------|----|----|----|-----|---|---|---|
| Eingang | 32 | 31 | 30 | ... | 3 | 2 | 1 |

Bemerkungen

LS_Dig_IO arbeitet nur korrekt, wenn folgende Voraussetzungen erfüllt sind:

- Am LS-Bus ist nur ein Modul angeschlossen.
- Das Modul ist vom Typ HSM-24V.
- Am Modul ist die Moduladresse 1 eingestellt.
- Nach Aufruf des Befehls **LS_Dig_IO** wird kein anderer LS-Bus-Befehl mehr verwendet.

Die Kanäle werden mit **LS_DigProg** als Ein- oder Ausgänge programmiert.

Das Bitmuster **pattern** wird nur für die Kanäle angewendet, die als Ausgang programmiert sind. Bits für Eingänge werden ignoriert.

Der Rückgabewert enthält den tatsächlichen Schaltzustand sowohl von Eingängen wie von Ausgängen. Beachten Sie: Die Eingänge haben einen Filter mit ca. 12µs Verzögerung.

LS_Dig_IO setzt den Watchdog-Zähler des Moduls auf den Startwert zurück. Der Zähler bleibt aktiv. Der Startwert wird mit **LS_Watchdog_Init** eingestellt.

Setzen Sie den aktiven Watchdog-Zähler während seines Zählvorgangs mindestens einmal zurück, um die Funktion des Moduls zu gewährleisten.

Gültig für

HSM-24V + X-A20

Siehe auch

[LS_DIO_Init](#), [LS_DigProg](#), [LS_Digout_Long](#), [LS_Digin_Long](#), [LS_Get_Output_Status](#), [LS_Reset](#), [LS_Watchdog_Init](#), [LS_Watchdog_Reset](#)



Beispiel

REM Example process for one module HSM-24V and ADwin-X

```
#Include ADwin-X.inc
```

Init:

```
Processdelay = 4000000    '10Hz HP
Par_1 = LS_DIO_Init(1)
REM enable watchdog time with 1.1 sec
Par_2 = LS_Watchdog_Init(1, 1, 1100)
REM LS channels 1...32 as outputs
Par_3 = LS_Digprog(1, 0Fh)
```

Event:

```
REM set one LS channel to high, rotating from 1 to 32
Inc Par_10
If (Par_10 >= 32) Then Par_10 = 0
Par_11 = Shift_Left(1, Par_10)
REM reset watchdog, set LS channels, and read back real state
REM (note: LS_Dig_IO uses LS address 1)
Par_12 = LS_Dig_IO(Par_11)
```

LS_Digout_Long

LS_Digout_Long setzt oder löscht durch den übergebenen 32 Bit-Wert alle Ausgänge auf einem Modul vom Typ HSM-24V am LS-Bus.

Syntax

```
#Include ADwin-X.Inc

LS_Digout_Long(ls_module, pattern)
```

Parameter

| | | |
|------------------|---|------|
| ls_module | Eingestellte Moduladresse (1...15) am HSM-Modul auf dem LS-Bus. | LONG |
| pattern | Bitmuster, nach dem die digitalen Ausgänge gesetzt werden: Bit = 0: Ausgang auf Pegel Low setzen. Bit = 1: Ausgang auf Pegel High setzen. | LONG |

| Bitnr. | 31 | 30 | ... | 2 | 1 | 0 |
|---------|----|----|-----|---|---|---|
| Ausgang | 32 | 31 | ... | 3 | 2 | 1 |

Bemerkungen

Die Kanäle werden mit **LS_DigProg** als Ein- oder Ausgänge programmiert.

Das Bitmuster **pattern** wird nur für die Kanäle angewendet, die als Ausgang programmiert sind. Bits für Eingänge werden ignoriert.

Siehe auch

[LS_DIO_Init](#), [LS_DigProg](#), [LS_Dig_IO](#), [LS_Digin_Long](#), [LS_Digout_Long_BS](#), [LS_Get_Output_Status](#), [LS_Reset](#), [LS_Watchdog_Init](#), [LS_Watchdog_Reset](#)

Gültig für

HSM-24V + X-A20

Beispiel

REM Example process for ADwin-X-A20 and 2 modules HSM-24V

REM Set process to low priority!

```
#Include ADwin-X.inc
```

Init:

```
Processdelay = 4000000 '10Hz HP
REM settings for LS module 1
Par_1 = LS_DIO_Init(1)
REM enable watchdog with 1.1 s
Par_2 = LS_Watchdog_Init(1, 1, 1100)
REM set LS channels 1...32 as outputs
Par_3 = LS_Digprog(1, 01111b)

REM settings for LS module 3
Par_11 = LS_DIO_Init(3)
REM enable watchdog with 1.1 s
Par_12 = LS_Watchdog_Init(3, 1, 1100)
REM set LS channels 1...32 as inputs
Par_13 = LS_Digprog(3, 0h)
```

Event:

```
REM set one LS channel to high, rotating from 1 to 32
Inc Par_15
If (Par_15 >= 32) Then Par_15 = 0
Par_16 = Shift_Left(1, PAR_15)
REM set LS channels of module 1
LS_Digout_Long(1, PAR_16)
REM read LS channels of module 3
Par_17 = LS_Digin_Long(3)
REM reset watchdog
LS_Watchdog_Reset()
```

LS_Digout_Long_BS setzt oder löscht durch den übergebenen 32 Bit-Wert alle Ausgänge auf einem Modul vom Typ HSM-24V am LS-Bus und gibt den Fehlerstatus zurück.

Syntax

```
#Include ADwin-X.Inc
```

```
LS_Digout_Long_BS(ls_module, pattern, status)
```

Parameter

ls_module Eingestellte Moduladresse (1...15) am HSM-Modul auf **LONG** dem LS-Bus.

pattern Bitmuster, nach dem die digitalen Ausgänge gesetzt **LONG** werden:
Bit = 0: Ausgang auf Pegel Low setzen.
Bit = 1: Ausgang auf Pegel High setzen.

| Bitnr. | 31 | 30 | ... | 2 | 1 | 0 |
|---------|----|----|-----|---|---|---|
| Ausgang | 32 | 31 | ... | 3 | 2 | 1 |

status Fehlerstatus: **LONG**
0: O.k.
-1: Keine Kommunikation mit dem Modul möglich.
>0: Bitmuster mit mehreren Fehlerbits.
Bit = 0: kein Fehler.
Bit = 1: Fehler aufgetreten.

| Bitnr. | 31:8 | 7 | 6 | 5:4 | 3 | 2 | 1 | 0 |
|--------|------|-------|-------|-----|----|------|-----|-----|
| Status | – | Temp2 | Temp1 | – | WD | Time | Ovr | Par |

- :don't care (mit **0CFh** ausmaskieren)

Par: Parity-Fehler bei der Datenübertragung auf dem LS-Bus.

Ovr: Overrun-Fehler bei der Datenübertragung auf dem LS-Bus.

Time: Timeout-Fehler bei der Datenübertragung auf dem LS-Bus.

WD: Watchdog hat ausgelöst. Die Kanaltreiber sind deaktiviert.

Temp1: Übertemperatur am Treiber für Kanäle 1...16. Treiber ist deaktiviert.

Temp2: Übertemperatur am Treiber für Kanäle 17...32. Treiber ist deaktiviert.

Bemerkungen

Die Kanäle werden mit **LS_DigProg** als Ein- oder Ausgänge programmiert.

Das Bitmuster **pattern** wird nur für die Kanäle angewendet, die als Ausgang programmiert sind. Bits für Eingänge werden ignoriert.

Siehe auch

[LS_DIO_Init](#), [LS_DigProg](#), [LS_Dig_IO](#), [LS_Digin_Long_BS](#), [LS_Digout_Long](#),
[LS_Get_Output_Status](#), [LS_Reset](#), [LS_Watchdog_Init](#), [LS_Watchdog_Reset](#)

Gültig für

HSM-24V + X-A20

LS_Digout_Long_BS

Beispiel

REM Example process for ADwin-X-A20 and 2 modules HSM-24V

REM Set process to low priority!

#Include ADwin-X.inc

Init:

Processdelay = 4000000 '10Hz HP

REM settings for LS module 1

Par_1 = **LS_DIO_Init**(1)

REM enable watchdog with 1.1 s

Par_2 = **LS_Watchdog_Init**(1, 1, 1100)

REM set LS channels 1...32 as outputs

Par_3 = **LS_Digprog**(1, 01111b)

REM settings for LS module 3

Par_11 = **LS_DIO_Init**(3)

REM enable watchdog with 1.1 s

Par_12 = **LS_Watchdog_Init**(3, 1, 1100)

REM set LS channels 1...32 as inputs

Par_13 = **LS_Digprog**(3, 0h)

Event:

REM set one LS channel to high, rotating from 1 to 32

Inc **Par_15**

If (**Par_15** >= 32) **Then** **Par_15** = 0

Par_16 = **Shift_Left**(1, **Par_15**)

REM set LS channels of module 1

LS_Digout_Long_BS(1, **Par_16**, **Par_5**)

If (**Par_5** <> 0) **Then** **End** 'exit on error)

REM read LS channels of module 3

Par_17 = **LS_Digin_Long_BS**(3, **Par_6**)

If (**Par_6** <> 0) **Then** **End** 'exit on error)

REM reset watchdog

LS_Watchdog_Reset()

LS_Digin_Long gibt den Zustand aller Kanäle auf einem Modul vom Typ HSM-24V am LS-Bus als Bitmuster zurück.

Syntax

```
#Include ADwin-X.Inc

ret_val = LS_Digin_Long(ls_module)
```

Parameter

| | | |
|------------------|---|------|
| ls_module | Eingestellte Moduladresse (1...15) am HSM-Modul auf dem LS-Bus. | LONG |
| ret_val | Bitmuster. Jedes Bit (31...0) entspricht dem Zustand eines digitalen Kanals (siehe Tabelle). Bit = 0: Pegel Low liegt an. Bit = 1: Pegel High liegt an. | LONG |

| Bitnr. | 31 | 30 | ... | 2 | 1 | 0 |
|----------|----|----|-----|---|---|---|
| Kanalnr. | 32 | 31 | ... | 3 | 2 | 1 |

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **LS_DigProg** als Eingänge zu programmieren.

Siehe auch

[LS_DIO_Init](#), [LS_DigProg](#), [LS_Dig_IO](#), [LS_Digin_Long_BS](#), [LS_Digout_Long](#), [LS_Get_Output_Status](#), [LS_Reset](#), [LS_Watchdog_Init](#), [LS_Watchdog_Reset](#)

Gültig für

HSM-24V + X-A20

Beispiel

REM Example process for ADwin-X-A20 and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwin-X.inc

Init:

```
Processdelay = 4000000    '10Hz HP
REM settings for LS module 1
Par_1 = LS_DIO_Init(1)
REM enable watchdog with 1.1 s
Par_2 = LS_Watchdog_Init(1, 1, 1100)
REM set LS channels 1...32 as outputs
Par_3 = LS_Digprog(1, 01111b)

REM settings for LS module 3
Par_11 = LS_DIO_Init(3)
REM enable watchdog with 1.1 s
Par_12 = LS_Watchdog_Init(3, 1, 1100)
REM set LS channels 1...32 as inputs
Par_13 = LS_Digprog(3, 0h)
```

Event:

```
REM set one LS channel to high, rotating from 1 to 32
Inc Par_15
If (Par_15 >= 32) Then Par_15 = 0
Par_16 = Shift_Left(1, PAR_15)
REM set LS channels of module 1
LS_Digout_Long(1, PAR_16)
REM read LS channels of module 3
Par_17 = LS_Digin_Long(3)
REM reset watchdog
LS_Watchdog_Reset()
```

LS_Digin_Long

LS_Digin_Long_BS

LS_Digin_Long_BS gibt den Zustand aller Kanäle auf einem Modul vom Typ HSM-24V am LS-Bus als Bitmuster zurück sowie den Fehlerstatus.

Syntax

```
#Include ADwin-X.Inc
```

```
ret_val = LS_Digin_Long_BS(ls_module, status)
```

Parameter

ls_module Eingestellte Moduladresse (1...15) am HSM-Modul auf **LONG** | dem LS-Bus.

status Fehlerstatus: **LONG** |

0: O.k.
-1: Keine Kommunikation mit dem Modul möglich.
>0: Bitmuster mit mehreren Fehlerbits.
Bit = 0: kein Fehler.
Bit = 1: Fehler aufgetreten.

| Bitnr. | 31:8 | 7 | 6 | 5:4 | 3 | 2 | 1 | 0 |
|--------|------|-------|-------|-----|----|------|-----|-----|
| Status | – | Temp2 | Temp1 | – | WD | Time | Ovr | Par |

- :don't care (mit **0CFh** ausmaskieren)

Par: Parity-Fehler bei der Datenübertragung auf dem LS-Bus.

Ovr: Overrun-Fehler bei der Datenübertragung auf dem LS-Bus.

Time: Timeout-Fehler bei der Datenübertragung auf dem LS-Bus.

WD: Watchdog hat ausgelöst. Die Kanaltreiber sind deaktiviert.

Temp1: Übertemperatur am Treiber für Kanäle 1...16. Treiber ist deaktiviert.

Temp2: Übertemperatur am Treiber für Kanäle 17...32. Treiber ist deaktiviert.

ret_val Bitmuster. Jedes Bit (31...0) entspricht dem Zustand **LONG** | eines digitalen Kanals (siehe Tabelle).
Bit = 0: Pegel Low liegt an.
Bit = 1: Pegel High liegt an.

| Bitnr. | 31 | 30 | ... | 2 | 1 | 0 |
|----------|----|----|-----|---|---|---|
| Kanalnr. | 32 | 31 | ... | 3 | 2 | 1 |

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **LS_DigProg** als Eingänge zu programmieren.

Siehe auch

[LS_DIO_Init](#), [LS_DigProg](#), [LS_Dig_IO](#), [LS_Digin_Long](#), [LS_Digout_Long_BS](#), [LS_Get_Output_Status](#), [LS_Reset](#), [LS_Watchdog_Init](#), [LS_Watchdog_Reset](#)

Gültig für

HSM-24V + X-A20

Beispiel

REM Example process for ADwin-X-A20 and 2 modules HSM-24V

REM Set process to low priority!

```
#Include ADwin-X.inc
```

Init:

```
Processdelay = 4000000      '10Hz HP
```

```
REM settings for LS module 1
```

```
Par_1 = LS_DIO_Init(1)
```

```
REM enable watchdog with 1.1 s
```

```
Par_2 = LS_Watchdog_Init(1, 1, 1100)
```

```
REM set LS channels 1...32 as outputs
```

```
Par_3 = LS_Digprog(1, 01111b)
```

```
REM settings for LS module 3
```

```
Par_11 = LS_DIO_Init(3)
```

```
REM enable watchdog with 1.1 s
```

```
Par_12 = LS_Watchdog_Init(3, 1, 1100)
```

```
REM set LS channels 1...32 as inputs
```

```
Par_13 = LS_Digprog(3, 0h)
```

Event:

```
REM set one LS channel to high, rotating from 1 to 32
```

```
Inc Par_15
```

```
If (Par_15 >= 32) Then Par_15 = 0
```

```
Par_16 = Shift_Left(1, PAR_15)
```

```
REM set LS channels of module 1
```

```
LS_Digout_Long_BS(1, PAR_16, Par_5)
```

```
If (Par_5 <> 0) Then End 'exit on error)
```

```
REM read LS channels of module 3
```

```
Par_17 = LS_Digin_Long_BS(3, Par_6)
```

```
If (Par_6 <> 0) Then End 'exit on error)
```

```
REM reset watchdog
```

```
LS_Watchdog_Reset()
```

LS_Get_Output_Status

LS_Get_Output_Status gibt den Überstrom-Status der Ausgänge auf einem Modul vom Typ HSM-24V am LS-Bus als Bitmuster zurück.

Syntax

```
#Include ADwin-X.Inc
```

```
ret_val = LS_Get_Output_Status(ls_module)
```

Parameter

ls_module Eingestellte Moduladresse (1...15) am HSM-Modul auf dem LS-Bus. LONG |

ret_val Bitmuster. Jedes Bit (31...0) entspricht dem Überstrom-Status eines digitalen Ausganges (siehe Tabelle). LONG |
Bit = 0: Normalzustand.
Bit = 1: Ausgang wegen Überstrom deaktiviert.

| Bitnr. | 31 | 30 | ... | 2 | 1 | 0 |
|---------|----|----|-----|---|---|---|
| Ausgang | 32 | 31 | ... | 3 | 2 | 1 |

Bemerkungen

Der Fehler „Übertemperatur“ eines Treibers kann nur auftreten, wenn auf mehreren Kanälen gleichzeitig ein Überstrom im Bereich von 150...500mA anliegt. Unabhängig davon wird bei einem Überstrom über 500mA der betroffene Kanal automatisch abgeschaltet.

Nach dem Fehler „Übertemperatur“ wird das Modul mit **LS_DIO_Init** wieder zurückgesetzt.

Die Kanäle des Moduls HSM-24V dürfen nur im Bereich von 0...150mA betrieben werden. Damit ist sichergestellt, dass das Modul HSM-24V dauerhaft ohne Unterbrechung arbeitet, selbst wenn alle Kanäle gleichzeitig aktiv sind.

Siehe auch

[LS_DIO_Init](#), [LS_DigProg](#), [LS_Dig_IO](#), [LS_Digout_Long](#), [LS_Digin_Long](#), [LS_Reset](#), [LS_Watchdog_Init](#), [LS_Watchdog_Reset](#)

Gültig für

HSM-24V + X-A20



Beispiel

REM Example process for ADwin-X-A20 and 2 modules HSM-24V

REM Set process to low priority!

#Include ADwin-X.inc

Init:

Processdelay = 4000000 '10Hz HP

REM settings for LS module 1

Par_1 = LS_DIO_Init(1)

REM enable watchdog with 1.1 s

Par_2 = LS_Watchdog_Init(1, 1, 1100)

REM set LS channels 1...32 as outputs

Par_3 = LS_Digprog(1, 01111b)

REM settings for LS module 3

Par_11 = LS_DIO_Init(3)

REM enable watchdog with 1.1 s

Par_12 = LS_Watchdog_Init(3, 1, 1100)

REM set LS channels 1...32 as inputs

Par_13 = LS_Digprog(3, 0h)

Event:

REM check for over-current

Par_5 = LS_Get_Output_Status(1) + LS_Get_Output_Status(3)

If (Par_5 > 0) Then End 'over-current: Exit program

REM set one channel to high, rotating from 1 To 32

Inc Par_15

If (Par_15 >= 32) Then Par_15 = 0

Par_16 = Shift_Left(1, Par_15)

REM set LS channels of module 1

LS_Digout_Long(1, Par_16)

REM read LS channels of module 3

Par_17 = LS_Digin_Long(3)

REM reset watchdog

LS_Watchdog_Reset()

LS_Reset

LS_Reset setzt die Schnittstelle zum LS-Bus zurück.

Syntax

```
#Include ADwin-X.Inc  
  
LS_Reset()
```

Parameter

-/-

Bemerkungen

Die Anweisung soll nur im Abschnitt **INIT:** verwendet werden, weil sie eine lange Ausführungs hat.

Siehe auch

[LS_DIO_Init](#), [LS_DigProg](#), [LS_Dig_IO](#), [LS_Digout_Long](#), [LS_Digin_Long](#), [LS_Get_Output_Status](#), [LS_Watchdog_Init](#), [LS_Watchdog_Reset](#)

Gültig für

HSM-24V + X-A20

Beispiel

REM Example process for one module HSM-24V and ADwin-X

```
#Include ADwin-X.inc
```

Init:

```
Processdelay = 4000000    '10Hz HP  
LS_Reset()  
LS_Watchdog_Reset()  
Par_1 = LS_DIO_Init(1)  
REM enable watchdog time with 1.1 sec  
Par_2 = LS_Watchdog_Init(1, 1, 1100)  
REM LS channels 1...32 as outputs  
Par_3 = LS_Digprog(1, 0Fh)
```

Event:

```
REM set one LS channel to high, rotating from 1 to 32  
Inc Par_10  
If (Par_10 >= 32) Then Par_10 = 0  
Par_11 = Shift_Left(1, Par_10)  
REM reset watchdog, set LS channels, and read back real state  
REM (note: LS_Dig_IO uses LS address 1)  
Par_12 = LS_Dig_IO(Par_11)
```

LS_Watchdog_Init aktiviert oder deaktiviert den Watchdog-Zähler eines Moduls am LS-Bus. Beim Aktivieren erhält der Zähler seinen Startwert und wird gestartet.

Syntax

```
#Include ADwin-X.Inc
```

```
ret_val = LS_Watchdog_Init(ls_module, enable, time)
```

Parameter

| | | |
|------------------|---|------|
| ls_module | Eingestellte Moduladresse (1...15) am HSM-Modul auf dem LS-Bus. | LONG |
| enable | Status des Watchdog-Zählers einstellen: 0 : Watchdog-Zähler deaktivieren. 1 : Watchdog-Zähler aktivieren. | LONG |
| time | Auslösezeit (0...107374) des Zählers in Millisekunden. | LONG |
| ret_val | Rückgabewert, das den Fehlerstatus angibt: -1: Keine Kommunikation mit dem Modul möglich. >0: Bitmuster mit mehreren Fehlerbits. Bit = 0: kein Fehler. Bit = 1: Fehler aufgetreten. | LONG |

| Bit-Nr. | 31...8 | 7 | 6 | 5...4 | 3 | 2 | 1 | 0 |
|---------|--------|-------|-------|-------|----|------|-----|-----|
| Status | – | Temp2 | Temp1 | – | WD | Time | Ovr | Par |

- :don't care (mit **0CFh** ausmaskieren)

Par:Parity-Fehler bei der Datenübertragung auf dem LS-Bus.

Ovr: Overrun-Fehler bei der Datenübertragung auf dem LS-Bus.

Time: Timeout-Fehler bei der Datenübertragung auf dem LS-Bus.

WD: Watchdog hat ausgelöst. Die Kanaltreiber sind deaktiviert.

Temp1: Übertemperatur am Treiber für Kanäle 1...16. Treiber ist deaktiviert.

Temp2: Übertemperatur am Treiber für Kanäle 17...32. Treiber ist deaktiviert.

Bemerkungen

Die Anweisung soll nur im Abschnitt **INIT**: verwendet werden, weil sie eine lange Ausführungszeit hat.

Solange der Watchdog-Zähler aktiv ist, dekrementiert er den Zählerstand kontinuierlich. Nach der eingestellten Auslösezeit erreicht der Zählerstand 0 (Null). Das Modul nimmt nun eine Fehlfunktion an und wird gestoppt; dadurch werden alle Ausgangssignale zurückgesetzt.

Nach dem Einschalten des Moduls ist der Zähler auf den Startwert 10 ms eingestellt und der Watchdog ist aktiv.

Setzen Sie den aktiven Watchdog-Zähler während seines Zählvorgangs mindestens einmal zurück, um die Funktion des Moduls zu gewährleisten. Zum Zurücksetzen können modulspezifische Befehle oder **LS_Watchdog_Reset** verwendet werden.

Die Watchdog-Funktion dient zur Verbindungsüberwachung zwischen ADwin-System und LS-Bus-Modul.

Siehe auch

[LS_DIO_Init](#), [LS_DigProg](#), [LS_Dig_IO](#), [LS_Digout_Long](#), [LS_Digin_Long](#), [LS_Get_Output_Status](#), [LS_Reset](#), [LS_Watchdog_Reset](#)

Gültig für

HSM-24V + X-A20

LS_Watchdog_Init



Beispiel

REM Example process for one module HSM-24V and ADwin-X

```
#Include ADwin-X.inc
```

Init:

```
Processdelay = 4000000    '10Hz HP
```

```
Par_1 = LS_DIO_Init(1)
```

```
REM enable watchdog time with 1.1 sec
```

```
Par_2 = LS_Watchdog_Init(1, 1, 1100)
```

```
REM LS channels 1...32 as outputs
```

```
Par_3 = LS_Digprog(1, 0Fh)
```

Event:

```
REM set one LS channel to high, rotating from 1 to 32
```

```
Inc Par_10
```

```
If (Par_10 >= 32) Then Par_10 = 0
```

```
Par_11 = Shift_Left(1, Par_10)
```

```
REM reset watchdog, set LS channels, and read back real state
```

```
REM (note: LS_Dig_IO uses LS address 1)
```

```
Par_12 = LS_Dig_IO(Par_11)
```

LS_Watchdog_Reset setzt die Watchdog-Zähler auf allen Modulen am LS-Bus auf den jeweiligen Startwert zurück. Die Zähler bleiben aktiv.

Syntax

```
#Include ADwin-X.Inc

LS_Watchdog_Reset()
```

Parameter

- / -

Bemerkungen

Solange der Watchdog-Zähler aktiv ist, dekrementiert er den Zählerstand kontinuierlich. Wenn der Zählerstand 0 (Null) erreicht, nimmt das Modul eine Fehlfunktion an und wird gestoppt. Dadurch werden alle Ausgangssignale zurückgesetzt (pull-down Stromsenke).

Setzen Sie den aktiven Watchdog-Zähler während seines Zählvorgangs mindestens einmal zurück auf den Startwert, um die Funktion des Moduls zu gewährleisten. Zum Zurücksetzen können auch modulspezifische Befehle verwendet werden.

Die Watchdog-Funktion dient zur Überwachung des LS-Bus-Moduls.

Siehe auch

[LS_DIO_Init](#), [LS_DigProg](#), [LS_Dig_IO](#), [LS_Digout_Long](#), [LS_Digin_Long](#), [LS_Get_Output_Status](#), [LS_Reset](#), [LS_Watchdog_Init](#)

Gültig für

HSM-24V + X-A20

Beispiel

REM Example process for ADwin-X-A20 and 2 modules HSM-24V

REM Set process to low priority!

```
#Include ADwin-X.inc
```

Init:

```
Processdelay = 4000000    '10Hz HP
REM settings for LS module 1
Par_1 = LS_DIO_Init(1)
REM enable watchdog with 1.1 s
Par_2 = LS_Watchdog_Init(1, 1, 1100)
REM set LS channels 1...32 as outputs
Par_3 = LS_Digprog(1, 01111b)

REM settings for LS module 3
Par_11 = LS_DIO_Init(3)
REM enable watchdog with 1.1 s
Par_12 = LS_Watchdog_Init(3, 1, 1100)
REM set LS channels 1...32 as inputs
Par_13 = LS_Digprog(3, 0h)
```

Event:

```
REM set one LS channel to high, rotating from 1 to 32
Inc Par_15
If (Par_15 >= 32) Then Par_15 = 0
Par_16 = Shift_Left(1, PAR_15)
REM set LS channels of module 1
LS_Digout_Long(1, PAR_16)
REM read LS channels of module 3
Par_17 = LS_Digin_Long(3)
REM reset watchdog
LS_Watchdog_Reset()
```

LS_Watchdog_Reset



Anhang

A.1 Technische Daten

| Allgemeine Daten / Grenzwerte | | | | | | |
|-------------------------------|--|--|----------------------------------|----------|------|---------|
| | Symbol | Konditionen | min. | typ. | max. | Einheit |
| Versorgungs-Spannung / -Strom | | | | | | |
| Spannung | U _b | | 10 | 12 | 28 | V |
| Betriebsstrom | I _{idle} bei U _b , typ. | | 0,6 | 0,8 | 1,5 | A |
| Einschaltstrombedarf | I _{power-on} | | | 3,0 | | A |
| Betrieb | | | | | | |
| Temperatur | T _{Umgebung} | | +5 | | +50 | °C |
| | T _{Gehäuse} | | +5 | | +55 | |
| rel. Feuchte | F _{rel} | nicht kondensierend | 0 | | 80 | % |
| Lagerung | | | | | | |
| Temperatur | T | | -20 | | +70 | °C |
| Feuchte | R _H | nicht kondensierend oder aggressive Atmosphäre | | | | |
| Abmessungen | | | | | | |
| Breite x Höhe x Tiefe | B x H x T | Standard | 215 x 125 x 47 | | | mm |
| | | Variante -R | 42TE x 3HE 213,5 x 129 x 29,5 | | | mm |
| Nettogewicht | | | | | | |
| Gewicht | m _{Netto} | Standard | 760 | | | g |
| | | Variante -R | 580 | | | g |
| Steckverbinder | | | | | | |
| Sub-D-Verbinder | Metrisches ISO-Gewinde; UNC-Gewinde als Bestelloption erhältlich | | | | | |
| Montage | | | | | | |
| optional | Hutschienen- und Wandmontage | | | | | |
| Prozessor T12.1 | | | | | | |
| Parameter | Symbol | Konditionen | min. | typ. | max. | Einheit |
| Typ | ZYNQ™ mit Dual-Core ARM Cortex-A9 | | | | | |
| Hersteller | XILINX | | | | | |
| Taktfrequenz | f _{CLK} | | | 666 | | MHz |
| Register-Breite | | Floating point Integer | | 64 32 | | Bit |
| Speicher | DRAM | | | 1 | | GByte |

| Basisversion | | | | | | |
|--|---|--|-------|------|-----------|---------|
| Parameter | Symbol | Konditionen | min. | typ. | max. | Einheit |
| Analoge Eingänge (ADC 18 Bit) | | | | | | |
| Anzahl | 8, differentiell. Version M1: Wandlung mit Multiplexer, Version F: Wandlung synchron | | | | | |
| Eingangsspannung | U _{out} | Version M1/F, k _V =1 | -10 | | +9,999695 | V |
| | | Version F, k _V =2 | -5 | | +4,999847 | V |
| Eingangswiderstand | R _i | | 323,4 | 330 | 336,6 | kΩ |
| Spannungsfestigkeit | U _{in max.} | ON & OFF | | | ±35 | V |
| Wandlungszeit | t _{Conv} | Version M1 | | 5 | | µs |
| | | Version F abh. von Kanalzahl | 1,25 | | 5 | µs |
| Integrale Nichtlinearität | INL | k _V =1 | | ±1 | ±3,5 | LSB |
| Different. Nichtlinearität | DNL | | | ±0,2 | ±0,9 | |
| Analoge Ausgänge 16 Bit | | | | | | |
| Anzahl | 2 | | | | | |
| Ausgangsspannung | U _{out} | | -10 | | +9,999695 | V |
| Aktualisierungszeit | t _{update} | | | 1 | | µs |
| Einschwingzeit | t _{settle} | 2V-Sprung | | 2 | | µs |
| | | FSR ^a (20V) | | 4 | | µs |
| Zulässiger Strom | | | | | ±5 | mA |
| Integrale Nichtlinearität | INL | | | | ±2 | LSB |
| Differentielle Nichtlinearität | DNL | | | | ±1 | |
| Offset | Fehler | abgleichbar mit ADtest.exe | | | | |
| Gain | Fehler | abgleichbar mit ADtest.exe | | | | |
| Analoge Ausgänge 12 Bit | | | | | | |
| Anzahl | 2 | | | | | |
| Ausgangsspannung | U _{out} | | -10 | | +9,995117 | V |
| Aktualisierungszeit | t _{update} | | 500 | | 1000 | µs |
| Zulässiger Strom | | | | | ±5 | mA |
| Integrale Nichtlinearität | INL | | | | ±2 | LSB |
| Different. Nichtlinearität | DNL | | | | ±1 | |
| Offset | Fehler | abgleichbar mit ADtest.exe | | | | |
| Gain | Fehler | abgleichbar mit ADtest.exe | | | | |
| Digitale Ein-/Ausgänge | | | | | | |
| Anzahl | DIO39:32 | 8, TTL, in 4er-Gruppen als Ein- oder Ausgang programmierbar. | | | | |
| | EVENT (Digin 62) | 1 ext. Trigger-Eingang. | | | | |
| Beschaltung siehe „ Beschaltung digitaler Ein-/Ausgänge “, TTL-Ein-/Ausgänge, Event-Eingang, Seite A-4 | | | | | | |
| LS-Bus | | | | | | |
| | 1 serielle Schnittstelle für bis zu 15 LS-Bus-Module. | | | | | |

^a Full Scale Range

| Option CO1 | | | | | | |
|---------------------|---|-------------|------|------|------|---------|
| Parameter | Symbol | Konditionen | min. | typ. | max. | Einheit |
| Zähler | | | | | | |
| Anzahl und Funktion | 1 Zählerblock mit Vor-/Rückwärtszähler (Ereigniszählung mit Takt / Richtung oder Vierflanken-Auswertung) und PWM-Zähler (Pulsweitenmessung). Pins mit TTL-Pegel, doppelt belegt. Eigenschaften siehe „Zähler“, Seite A-5 | | | | | |

| Option D | | | | | | |
|---|--|---|------|------|------|---------|
| Parameter | Symbol | Konditionen | min. | typ. | max. | Einheit |
| Digitale Ein-/Ausgänge | | | | | | |
| Anzahl | DIO41:DIO40 DIGIN47: DIGIN42 DIGIN60 | 2 programmierbare Ein- oder Ausgänge, differentiell. 6 Eingänge, differentiell, doppelt belegt. 1 Eingang, differentiell, doppelt belegt. | | | | |
| Beschaltung siehe „Beschaltung digitaler Ein-/Ausgänge“, differentiell, Seite A-4 | | | | | | |
| Zähler | | | | | | |
| Anzahl und Funktion | 2 Zählerblöcke mit Vor-/Rückwärtszähler (Ereigniszählung mit Takt / Richtung oder Vierflanken-Auswertung) und PWM-Zähler (Pulsweitenmessung). Pins differentiell, doppelt belegt. Eigenschaften siehe „Zähler“, Seite A-5 | | | | | |
| Schnittstellen | | | | | | |
| SSI | 1 SSI-Decoder, Pins differentiell, Eingang doppelt belegt. | | | | | |

| Option DCT | | | | | | |
|--|---|---|------|------|------|---------|
| Parameter | Symbol | Konditionen | min. | typ. | max. | Einheit |
| Digitale Ein-/Ausgänge | | | | | | |
| Anzahl | DIO41:DIO40 DIGIN47: DIGIN42 DIGIN60 DIO31:DIO00 DIGIN55: DIGIN48 DIGIN59: DIGIN56 | 2 programmierbare Ein- oder Ausgänge, differentiell. 6 Eingänge, differentiell, doppelt belegt. 1 Eingang, differentiell, doppelt belegt. 32 Ein- oder Ausgänge, TTL-Pegel, in 8er-Gruppen als Ein- oder Ausgang programmierbar. DIO31:DIO26 doppelt belegt. 8 Eingänge, Komparatorpegel mit DAC12-1 einstellbar, doppelt belegt. 4 Eingänge, Komparatorpegel mit DAC12-2 einstellbar. | | | | |
| Beschaltung siehe „Beschaltung digitaler Ein-/Ausgänge“, TTL, differentiell, Seite A-4 | | | | | | |
| Zähler | | | | | | |
| Anzahl und Funktion | 2 Zählerblöcke, Eingänge differentiell, doppelt belegt. 2 Zählerblöcke, Eingänge mit TTL-Pegel, doppelt belegt. 2 Komparator-Zählerblöcke, Eingänge mit TTL-Pegel, doppelt belegt. Jeder Zählerblock jeweils mit Vor-/Rückwärtszähler (Ereigniszählung mit Takt / Richtung oder Vierflanken-Auswertung) und PWM-Zähler (Pulsweitenmessung). Eigenschaften siehe „Zähler“, Seite A-5 | | | | | |
| Schnittstellen | | | | | | |
| SSI | SSI-Decoder, 1 Schnittstelle. Pins TTL, doppelt belegt. | | | | | |

| Option COM | | | | | | |
|--|---|--|---------|------|---------|---------|
| Schnittstellen | | | | | | |
| CAN | CAN High speed, 2 Schnittstellen | | | | | |
| RS232 | RS232, 1 Schnittstelle | | | | | |
| Option Profinet | | | | | | |
| Schnittstellen | | | | | | |
| Profinett | Profinet, 1 Schnittstelle | | | | | |
| Option Profibus-IRT | | | | | | |
| Schnittstellen | | | | | | |
| Profibus-IRT | Profibus-IRT, 1 Schnittstelle 2 Buchsen RJ-45 (Kupfer) oder 2 Duplex-Buchsen SC-RJ (Lichtwellenleiter) | | | | | |
| Option EtherCat | | | | | | |
| Schnittstellen | | | | | | |
| EtherCat | EtherCat, 1 Schnittstelle | | | | | |
| Beschaltung digitaler Ein-/Ausgänge | | | | | | |
| Parameter | Symbol | Konditionen | min. | typ. | max. | Einheit |
| EVENT-Eingang | | | | | | |
| Flankenerkennung, pos. | V _{T+} (Low) | V _{CC} = 3,3V | 1,65 | 1,9 | 2,15 | V |
| Flankenerkennung, neg. | V _{T-} (High) | V _{CC} = 3,3V | 0,75 | 1,0 | 1,25 | |
| Schalthysterese | V _{T+} - V _{T-} | | 0,4 | 0,9 | | |
| Eingangsstrom | I _{IH} | V _I = 2,7V | | | 20 | µA |
| | I _{IL} | V _I = 0,4V | | | -50 | |
| TTL-Eingänge | | | | | | |
| Eingangsspannung | | | -0,5 | | +6,0 | V |
| Logik-Eingangsspannung | V _{IH} (High) | V _{CC} = 5V | 3,5 | | | |
| | V _{IL} (Low) | V _{CC} = 5V | | | 1,5 | |
| Logik-Eingangsstrom | I _I | 10kΩ Pull-down | | | | |
| Komparator-Eingänge | | | | | | |
| Eingangsspannung | | | -0,1 | | +30 | V |
| Logik-Schaltswelle | V _{IH} (High) | U _{out} DAC12 + 0,4V | 0,4...5 | | | |
| | V _{IL} (Low) | U _{out} DAC12 - 0,4V | | | 0...4,6 | |
| Schalthysterese | V _{IH} - V _{IL} | abhängig von U _{out} DAC12 | ±30 | | ±70 | mV |
| max. Messfrequenz | f | abhängig von U _{out} DAC12 und V _I | 10 | 30 | 200 | kHz |
| Logik-Eingangsstrom | I _I | 14,7kΩ Pull-up (zu +5V) | | | | |
| differentielle Kanäle DIO41:DIO40 | | | | | | |
| Differentielle Eingangsschwellenspannung | V _{TH} | -7V ≤ V _{CM} ≤ 12V | -300 | | +300 | mV |
| Schalthysterese | ΔV _{TH} | -7V ≤ V _{CM} ≤ 12V | | 25 | | mV |
| Bereich der Gleichtaktspannung | V _{CM} | | -7 | | +12 | V |
| Differentielle Ausgangsspannung | V _{OD1} | | 1,5 | | 5 | V |

| Beschaltung digitaler Ein-/Ausgänge | | | | | | |
|---|------------------------|--|------|-----|-------|------|
| differentielle Eingänge DIGIN47:DIGIN42 | | | | | | |
| Differentielle Eingangs-Schwellenspannung | V _{TH} | -10V ≤ V _{CM} ≤ 13,2V | -200 | | +200 | mV |
| Schalthysterese | ΔV _{TH} | -10V ≤ V _{CM} ≤ 13,2V | | 40 | | mV |
| Bereich der Gleichtaktspannung | V _{CM} | | -10 | | +13,2 | V |
| Anstieg-/Abfallgeschw. der differentiellen Spg. | | | 0,33 | | | V/μs |
| Zulässige differentielle Eingangsspannung | | für jeden Eingang | | | ±3,9 | V |
| Busabschluss | | bei Verwendung als Zählereingang | | 120 | | Ω |
| TTL-Ausgänge | | | | | | |
| Logik-Ausgangsspannung | V _{OH} (High) | I _{OH} = -32mA | 3,8 | | | V |
| | V _{OL} (Low) | I _{OL} = +6mA, V _{CC} = 4,5V | | | 0,55 | |
| Logik-Ausgangsstrom | I _O | je DIO-Leitung | | | ±32 | mA |
| | I _{TOTAL} | je DIO-Gruppe (8) über V _{CC} / GND | | | ±100 | |

| Zähler | | | | | | |
|---------------------------------------|---|---------------|------|------|------|---------|
| Parameter | Symbol | Konditionen | min. | typ. | max. | Einheit |
| Referenz-Quarzoszillator | | | | | | |
| Referenzfrequenz | f _{ref} | | | 33,3 | | MHz |
| Genauigkeit und Drift | | | | | ±50 | ppm |
| Funktion | Zählerblock mit <ul style="list-style-type: none">– Vor-/Rückwärtszähler zur Messung von Tastverhältnis, Impuls- und Periodendauer sowie zur Ereigniszählung mit Takt / Richtung oder Vierflanken-Auswertung.– PWM-Zähler mit internem Takt zur Pulsweitenmessung. | | | | | |
| Zählereingänge | Je Zähler 3 Eingänge (A/CLK, B/DIR, CLR/LATCH). Eingangsbeschaltungen siehe: „ Beschaltung digitaler Ein-/Ausgänge “. | | | | | |
| Zähler- und Latch-Breite | | | | 32 | | Bit |
| Zählfrequenz Zähler 1...5 | f _{CLK} | Eingang CLK | | 20 | | MHz |
| | | Eingang A/B | | 5 | | |
| Zählfrequenz Komparatorzähler 6, 7 | f _{CLK} | Eingang CLK | | 200 | | kHz |
| | | Eingang A/B | | 50 | | |
| PWM-Zählertakt | f _{CLK} | interner Takt | | 100 | | MHz |

A.2 Hardware-Revisionen

Auf dem Gerät befindet sich ein Aufkleber mit der Revisionsbezeichnung. Die Unterschiede der Revisionsstände sind nachfolgend dargestellt:

| Revision | Erst- ausgabe | Änderung zur Vorgänger-Version |
|----------|------------------|--------------------------------|
| A | 1 / 2019 | Erst-Version. |

A.3 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-X-A20* erfüllt die Voraussetzungen der RoHS-Richtlinie in allen gelieferten Varianten.