C-Control Pro
Mega 32

Inhalt

Kapitel

1 Wichtige Hinweise

1.1 Einleitung

Das C-Control Pro System basiert auf dem Atmel Mega 32 RISC Mikrocontroller. Dieser Mikrocontroller wird in sehr vielen Geräten in großen Stückzahlen eingesetzt. Von der Unterhaltungselektronik, über Haushaltsmaschinen bis hin zu verschiedenen Einsatzmöglichkeiten in der Industrie. Dort übernimmt der Controller wichtige Steuerungsaufgaben. C-Control Pro bietet Ihnen diese hochmoderne Technologie zur Lösung Ihrer Steuerungsprobleme. Sie können analoge Meßwerte und Schalterstellungen erfassen und abhängig von diesen Eingangsbedingungen entsprechende Schaltsignale ausgeben, z.B. für Relais oder Stellmotoren. In Verbindung mit einer DCF77-Funkantenne kann C-Control Pro die atomgenaue Uhrzeit empfangen und präzise Schaltuhrfunktionen übernehmen. Verschiedene Hardware-Schnittstellen und Bussysteme erlauben die Vernetzung von C-Control Pro mit Sensoren, Aktoren und anderen Steuerungssystemen. Wir wollen unsere Technologie einem breiten Anwenderkreis zur Verfügung stellen. Aus unserer bisherigen Arbeit im C-Control-Service wissen wir, daß sich auch lernbereite Kunden ohne jegliche Elektronik- und Programmiererfahrungen für C-Control interessieren. Sollten Sie zu dieser Anwendergruppe gehören, gestatten Sie uns an dieser Stelle bitte einen Hinweis:

C-Control Pro ist nur bedingt für den Einstieg in die Programmierung von Mikrocomputern und die elektronische Schaltungstechnik geeignet! Wir setzen voraus, daß Sie zumindest über Grundkenntnisse in einer höheren Programmiersprache, wie z.B. BASIC, PASCAL, C, C++ oder Java verfügen. Außerdem nehmen wir an, daß Ihnen die Bedienung eines PCs unter einem der Microsoft Windows Betriebssysteme (98SE/NT/2000/ME/XP) geläufig ist. Sie sollten auch einige Erfahrungen im Umgang mit dem Lötkolben, Multimetern und elektronischen Bauelementen haben. Wir haben uns bemüht, alle Beschreibungen so einfach wie möglich zu formulieren. Leider können wir in einer Bedienungsanleitung zum hier vorliegenden Thema nicht immer auf den Gebrauch von Fachausdrücken und Anglizismen verzichten. Schlagen Sie diese bei Bedarf bitte in entsprechenden Fachbüchern nach.

1.2 Lesen dieser Anleitung

Bitte lesen Sie diese Anleitung, bevor Sie die C-Control Pro Unit in Betrieb nehmen. Während einige Kapitel nur für das Verständnis der tieferen Zusammenhänge von Interesse sind, enthalten andere wichtige Informationen, deren Nichtbeachtung zu Fehlfunktionen oder Beschädigungen führen kann.

» Kapitel und Absätze, die wichtige Themen enthalten, sind durch das Symbol » gekennzeichnet. Bitte lesen Sie diese Anmerkungen besonders intensiv durch.

Lesen Sie bitte vor Inbetriebnahme die komplette Anleitung durch, sie enthält wichtige Hinweise zum korrekten Betrieb. Bei Sach- oder Personenschäden, die durch unsachgemäße Handhabung oder Nichtbeachten dieser Bedienungsanleitung verursacht werden, erlischt der Garantieanspruch! Für Folgeschäden übernehmen wir keine Haftung!

1.3 Handhabung

Die C-Control Pro Unit enthält empfindliche Bauteile. Diese können durch elektrostatische Entladungen zerstört werden! Beachten Sie die allgemeinen Regeln zur Handhabung elektronischer Bauelemente. Richten Sie Ihren Arbeitsplatz fachgerecht ein. Erden Sie Ihren Körper vor der Arbeit, z.B. durch Berühren eines geerdeten, leitenden Gegenstandes(z.B. Heizkörper). Vermeiden Sie die Berührung der Anschlußpins der C-Control Pro Unit.

1.4 Bestimmungsgemäße Verwendung

Die C-Control Pro Unit ist ein elektronisches Bauelement im Sinne eines integrierten Schaltkreises. Die C-Control Pro Unit dient zur programmierbaren Ansteuerung elektrischer und elektronischer Geräte. Der Aufbau und Betrieb dieser Geräte muß konform zu geltenden europäischen Zulassungsrichtlinien (CE) erfolgen.

Die C-Control Pro Unit darf nicht in galvanischer Verbindung zu Spannungen über Schutzkleinspannung stehen. Die Ankoppelung an Systeme mit höherer Spannung darf ausschließlich über Komponenten mit VDE-Zulassung erfolgen. Dabei müssen die vorgeschriebenen Luft- und Kriechstrecken eingehalten sowie ausreichende Maßnahmen zum Schutz vor Berührung gefährlicher Spannungen getroffen werden.

Auf der Platine der C-Control Pro Unit arbeiten elektronische Bauelemente mit hochfrequenten Taktsignalen und steilen Pulsflanken. Bei unsachgemäßem Einsatz der Unit kann das zur Aussendung elektromagnetischer Störsignale führen. Die Ergreifung entsprechender Maßnahmen (z.B. Verwendung von Drosselspulen, Begrenzungswiderständen, Blockkondensatoren und Abschirmungen) zur Einhaltung gesetzlich vorgeschriebener Maximalwerte liegt in der Verantwortung des Anwenders.

Die maximal zulässige Länge angeschlossener Leitungen ohne zusätzliche Maßnahmen beträgt 0,25 Meter (Ausnahme serielle Schnittstelle). Unter dem Einfluß von starken elektromagnetischen Wechselfeldern oder Störimpulsen kann die Funktion der C-Control Pro Unit beeinträchtigt werden. Gegebenenfalls sind ein Reset und ein Neustart des Systems erforderlich.

Achten Sie beim Anschluß von externen Baugruppen auf die zulässigen maximalen Strom- und Spannungswerte der einzelnen Pins. Das Anlegen einer verpolten oder zu hohen Spannung oder die Belastung mit einem zu hohen Strom kann zur sofortigen Zerstörung der Unit führen. Bitte halten Sie die C-Control Pro Unit von Spritzwasser und Kondensationsfeuchtigkeit fern. Beachten Sie den zulässigen Betriebstemperaturbereich in den Technischen Daten im Anhang.

1.5 Gewährleistung und Haftung

Conrad Electronic bietet für die C-Control Pro Unit eine Gewährleistungsdauer von 24 Monaten ab Rechnungsdatum. Innerhalb dieses Zeitraums werden defekte Units kostenfrei umgetauscht, wenn der Defekt nachweislich auf einen Produktionsfehler oder Transportschaden zurückzuführen ist.

Die Software im Betriebssystem des Mikrocontrollers sowie die PC-Software auf CD-ROM werden in der vorliegenden Form geliefert. Conrad Electronic übernimmt keine Garantie dafür, daß die Leistungsmerkmale dieser Software individuellen Anforderungen genügen und daß die Software in jedem Fall unterbrechungs- und fehlerfrei arbeitet. Conrad Electronic übernimmt keine Haftung für Schäden, die unmittelbar durch oder in Folge der Anwendung der C-Control Pro Unit entstehen. Der Einsatz der C-Control Pro Unit in Systemen, die direkt oder indirekt medizinischen, gesundheits- oder lebenssichernden Zwecken dienen, ist nicht zulässig.

Sollte die C-Control Pro Unit inklusive Software Ihre Ansprüche nicht befriedigen, oder sollten Sie mit den Gewährleistungs- und Haftungsbedingungen nicht einverstanden sein, nutzen Sie unsere 14tägige Geld-Zurück-Garantie. Bitte geben Sie uns die Unit dann innerhalb dieser Frist ohne Gebrauchsspuren, in unbeschädigter Originalverpackung und mit allem Zubehör zur Erstattung oder Verrechnung des Warenwertes zurück!

1.6 Service

Conrad Electronic stellt Ihnen ein Team von erfahrenen Servicemitarbeitern zur Seite. Sollten Sie Fragen zur C-Control Pro Unit haben, erreichen Sie unsere Technische Kundenbetreuung per Brief, Fax oder E-Mail.

per Brief Conrad Electronic
Technische Anfrage
Klaus-Conrad-Straße 2
92530 Wernberg-Köblitz

Fax-Nr.: 09604 / 40-8848
Mail: webmaster@c-control.de

Bitte nutzen Sie vorzugsweise die Kommunikation per E-Mail. Wenn Sie ein Problem haben, geben Sie uns nach Möglichkeit eine Skizze Ihrer Anschlußschaltung als angehängte Bilddatei (im JPG-Format) sowie den auf das Problem reduzierten Teil Ihres Programmquelltextes (maximal 20 Zeilen). Sollten Sie Interesse an einer kundenspezifischen Softwarelösung haben, vermitteln wir Ihnen gern ein entsprechendes Angebot. Weiterführende Informationen und aktuelle Software zum Download finden Sie auf der C-Control Homepage im Internet unter www.c-control.de.

1.7 Open Source

Bei Erstellung von C-Control Pro ist auch Open Source Software zum Einsatz gekommen:

ANTRL 2.73	http://www.antlr.org
Inno Setup 5.08	http://www.jrsoftware.org
GPP (Generic Preprocessor)	http://www.nothingisreal.com/gpp

Gemäß den Bestimmungen der "LESSER GPL" (www.gnu.org/copyleft/lesser) wird bei der Installation der IDE auch der Orginal Sourcecode des Generic Preprocessors, sowie der Quelltext der modifizierten Version mitgeliefert, der bei C-Control Pro zum Einsatz kommt. Beide Quelltexte sind im "GNU" Unterverzeichnis in einem ZIP Archiv zu finden.

Kapitel 2

2 Installation

2.1 Software

Wird die mitgelieferte CD in den Computer eingelegt, sollte automatisch der Installer gestartet werden, um die C-Control Pro Software zu installieren. Geschieht dies nicht, weil z.B. die Autostart Funktion für CD oder DVD in Windows abgeschaltet ist, so starten Sie bitte den Installer 'C-ControlSetup.exe' im Hauptverzeichnis der CD-ROM per Hand.

 Für den Zeitraum der Software Installation und der Installation der USB Treiber muß der Anwender sich als Administrator angemeldet haben. Bei der normalen Arbeit mit C-Control Pro ist dies nicht nötig.

Am Anfang der Installation wählen Sie in welcher Sprache die Installation durchgeführt werden soll. Danach können Sie aussuchen, ob C-Control Pro im Standard Pfad installiert werden soll, oder ob Sie ein eigenes Zielverzeichnis angeben möchten. Zu Ende des Installationsvorgangs werden Sie noch gefragt, ob Icons auf Ihrem Desktop kreiert werden sollen.

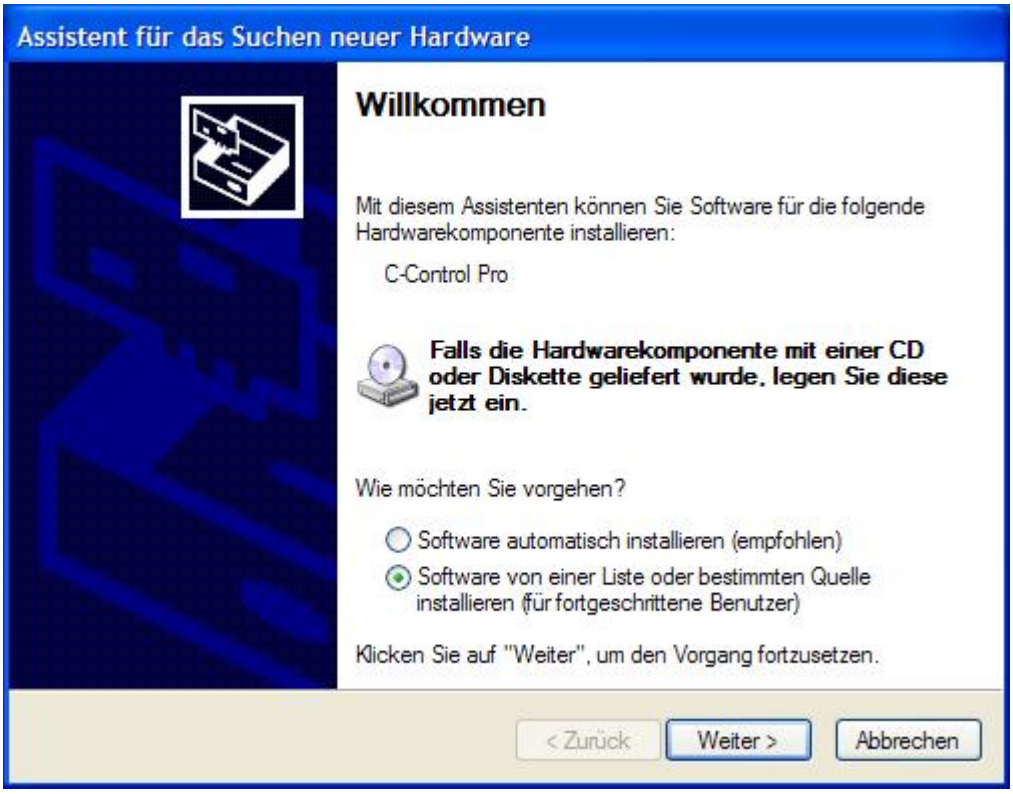
Ist der Installationsvorgang abgeschlossen, so können Sie sich auf Wunsch direkt das "ReadMe" anzeigen lassen, oder die C-Control Pro Entwicklungsumgebung starten.

2.2 Applicationboard

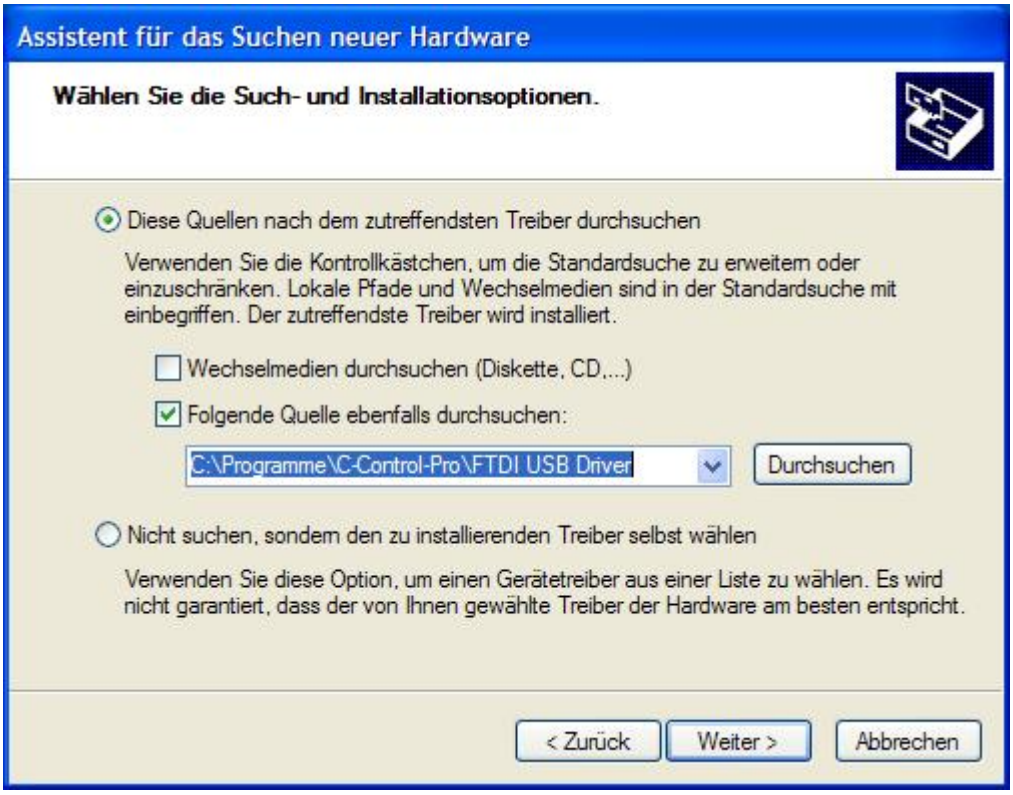
Bitte verbinden Sie das Application Board mit einem Netzgerät. Sie können hierzu ein Standard Steckernetzteil mit 9V/250mA verwenden. Die Polung ist beliebig, sie wird durch Dioden immer richtig umgesetzt. Je nach zusätzlicher Beschaltung kann es später notwendig sein ein Netzteil mit höherer Leistung zu verwenden. Stellen Sie eine Verbindung zwischen dem Application Board und Ihrem PC mit Hilfe eines USB Kabels her. Schalten Sie das Application Board ein.

Ein Windowsbetriebssystem vor Win98 SE ("*Second Edition*") wird vermutlich keine zuverlässige USB Verbindung zwischen PC und Application Board ermöglichen. Die USB Treiber von Microsoft funktionieren erst ab Win98 SE zuverlässig mit allen USB Geräten. In einem solchen Fall kann man nur raten auf ein aktuelleres Betriebssystem aufzurüsten, oder nur die serielle Verbindung zum Application Board zu benutzen.

Ist das Application Board zum ersten Mal angeschlossen worden, so ist kein Treiber für den FTDI Chip vorhanden. Unter Windows XP wird dann folgendes Fenster angezeigt:



Es ist hier dann "Software von einer Liste oder bestimmten Quelle installieren" anzuwählen und auf "Weiter" zu klicken.



Danach ist der Pfad zum Verzeichnis des Treibers anzugeben. Hat man die Software nach "C:\Programme" installiert, ist der Pfad "C:\Programme\C-Control-Pro\FTDI USB Driver".

Die Nachricht "C-Control Pro USB Device hat den Windows-Logo-Test nicht bestanden...." ist ganz normal. Sie besagt **nicht,** daß der Treiber beim Windows-Logo-Test versagt hat, sondern daß der Treiber am (ziemlich kostspieligen) Test in Redmond nicht teilgenommen hat.

An dieser Stelle einfach "Installation fortsetzen" drücken. Nach ein paar Sekunden sollte der USB Treiber dann fertig installiert sein.

In der PC-Software im Menü Optionen auf IDE klicken und den Bereich Schnittstellen selektieren. Dort den Kommunikationsport "USB0" auswählen.

Serieller Anschluß

Aufgrund der langsamen Übertragungsgeschwindigkeit der seriellen Schnittstelle ist ein USB Anschluß zu bevorzugen. Ist jedoch aus Hardwaregründen die USB Schnittstelle nicht verfügbar, so kann der Bootloader in den seriellen Modus gebracht werden.

Hierzu ist beim Einschalten des Application Boards der Taster SW1 gedrückt zu halten. Danach ist der serielle Bootloader Modus aktiviert.

In der PC-Software den Punkt IDE im Menü Optionen anwählen und dort den Bereich Schnittstellen auswählen. Dort einen Kommunikationsport "COMx" wählen, der zu der Schnittstelle am PC passt, an der das Board angeschlossen wurde.

Kapitel 3

3 Hardware

3.1 Firmware

Das Betriebssystem des C-Control Pro besteht aus folgenden Komponenten:

- *Bootloader*
- *Interpreter*

Bootloader

Der Bootloader ist immer verfügbar. Er sorgt für die USB oder serielle Kommunikation mit der IDE. Über Kommandozeilenbefehle kann der Interpreter und das Anwenderprogramm vom PC in den Atmel Risc Chip übertragen werden. Wird ein Programm kompiliert und in den Mega Chip übertragen wird gleichzeitig auch der aktuelle Interpreter mit übertragen.

» Soll statt dem USB Interface eine serielle Verbindung von der IDE zum C-Control Pro Modul aufgebaut werden, so ist beim Einschalten des Moduls der Taster SW1 gedrückt zu halten. In diesem Modus wird jegliche Kommunikation über die serielle Schnittstelle geleitet. Dies ist praktisch, wenn das Modul schon in die Hardware Applikation eingebaut wurde und das Application Board daher nicht zur Verfügung steht. Die serielle Kommunikation ist jedoch um einiges langsamer als eine USB Verbindung. Im seriellen Modus werden die Pins für USB nicht benutzt und stehen dem Anwender für andere Aufgaben zur Verfügung.

Interpreter

Der Interpreter besteht aus mehreren Komponenten:

- Bytecode Interpreter
- Multithreading Unterstützung
- Interruptverarbeitung
- Anwenderfunktionen
- RAM und EEPROM Schnittstelle

In der Hauptsache arbeitet der Interpreter den Bytecode ab, der vom Compiler generiert wurde. Weiter sind die meisten Bibliotheksfunktionen in ihm integriert, damit das Bytecodeprogramm z.B. auf Hardwareports zugreifen kann. Die RAM und EEPROM Schnittstelle wird vom Debugger der IDE benutzt, um Zugang zu Variablen zu bekommen, wenn der Debugger bei einem Breakpoint angehalten hat.

» Ist kein USB Interface angeschlossen, und wurde beim Einschalten nicht SW1 gedrückt, um in den seriellen Bootloadermodus zu kommen, wird der Bytecode (sofern vorhanden) im Interpreter gestartet. Das heißt, wird das Modul in eine Hardware Applikation eingebaut, so reicht ein Anlegen der Betriebsspannung, um das Anwenderprogramm automatisch zu starten.

3.2 Mega32

3.2.1 Microcontroller

Mega32 Übersicht

Der Microcontroller ATmega32 stammt aus der AVR-Familie von ATMEL. Es handelt sich um

einen low-power Microcontroller mit Advanced RISC Architecture. Hier folgt eine kurze Zusammenstellung der Hardwareressourcen:

- **131 Powerful Instructions – Most Single-clock Cycle Execution**
- **32 x 8 General Purpose Working Registers**
- **Up to 16 MIPS Throughput at 16 MHz**

- **Nonvolatile Program and Data Memories**
  32K Bytes of In-System Self-Programmable Flash
  Endurance: 10,000 Write/Erase Cycles
  In-System Programming by On-chip Boot Program

- **1024 Bytes EEPROM**
- **2K Byte Internal SRAM**

- **Peripheral Features:**
  Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  Four PWM Channels
  8-channel, 10-bit ADC
  8 Single-ended Channels
  2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  Byte-oriented Two-wire Serial Interface (I2C)
  Programmable Serial USART
  On-chip Analog Comparator
  External and Internal Interrupt Sources
  32 Programmable I/O Lines

- **40-pin DIP**
- **Operating Voltages 4.5 - 5.5V**

3.2.2 Modul

Modulspeicher

In dem C-Control Pro Modul sind 32kB FLASH und 2kB SRAM integriert. Auf dem Application Board befindet sich ein zusätzliches EEPROM mit einer Speichertiefe von 8kB. Dieses EEPROM ist über eine I2C Schnittstelle ansprechbar.

Hinweis: Detailliertere Informationen findet man in den PDF-Dateien der IC-Hersteller auf der C-Control Pro Software CD.

ADC-Referenzspannungserzeugung

Der Mikrocontroller verfügt über einen Analog-Digital-Wandler mit einer Auflösung von 10 Bit. Das heißt, gemessene Spannungen können als ganze Zahlen von 0 bis 1023 dargestellt werden. Die Referenzspannung für die untere Grenze ist der GND-Pegel, also 0V. Die Referenzspannung für die obere Grenze kann vom Benutzer gewählt werden:

- 5V Versorgungsspannung (VCC)
- interne Referenzspannung von 2,56V
- externe Referenzspannung z.B. 4,096V durch Referenzspannungs-IC erzeugt

Ist *x* ein digitaler Meßwert, dann errechnet sich der entsprechende Spannungswert u wie folgt:

u = *x* * Referenzspannung / 1024

Takterzeugung

Die Takterzeugung erfolgt durch einen 14,7456MHz-Quarzoszillator. Alle zeitlichen Abläufe des Controllers sind von dieser Taktfrequenz abgeleitet.

Reset

Ein Reset bewirkt die Rückkehr des Microcontrollersystems in einen definierten Anfangszustand. Das C-Control Pro Modul kennt grundsätzlich 2 Reset-Quellen:

- Power-On-Reset: wird automatisch nach dem Einschalten der Betriebsspannung ausgeführt
- Hardware-Reset: wird ausgeführt wenn der RESET (Pin 9) des Moduls "low" gezogen und wieder losgelassen wird, z.B. durch kurzes Drücken des angeschlossenen Reset-Tasters RESET1 (SW3)

Durch eine „Brown-Out-Detection" wird verhindert, daß der Controller bei Absinken der Versorgungsspannung in undefinierte Zustände kommen kann.

Digitalports (PortA, PortB, PortC, PortD)

Das C-Control Pro Modul verfügt über vier digitale Ports mit je 8 Pins. An den Digitalports können z.B. Taster mit Pull-Up-Widerständen, Digital-ICs, Optokoppler oder Treiberschaltungen für Relais angeschlossen werden. Die Ports können einzeln, d.h pinweise oder byteweise angesprochen werden. Jeder Pin kann entweder Eingang oder Ausgang sein.

» Niemals zwei Ports direkt zusammenschalten, die gleichzeitig als Ausgang arbeiten sollen!

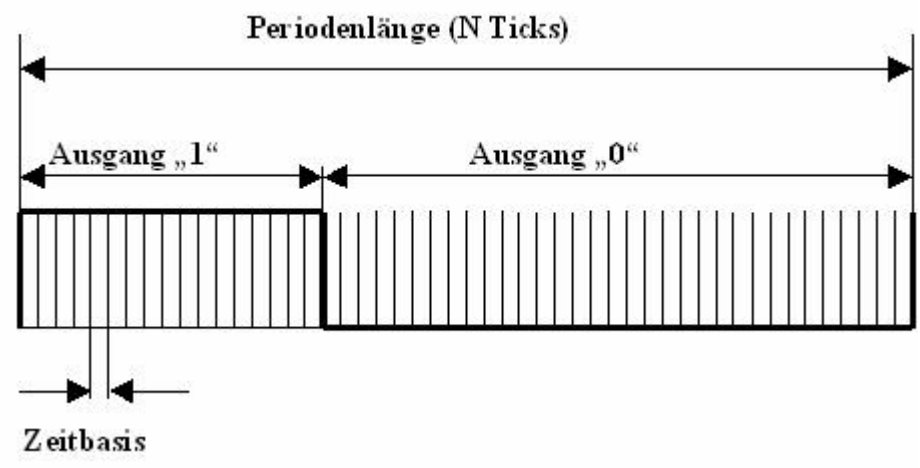
Digitale Eingangspins sind hochohmig oder mit internem Pullup-Widerstand beschaltet und überführen ein anliegendes Spannungssignal in einen logischen Wert. Voraussetzung dafür ist, daß sich das Spannungssignal innerhalb der für TTL-bzw. CMOS-ICs definierten Bereiche für Low- oder Highpegel befindet. In der weiteren Verarbeitung im Programm werden die logischen Werte von einzelnen Eingangsports als 0 ("low") oder -1 ("high") dargestellt. Pins nehmen Werte von 0 oder 1 an, Byteports 0 bis 255. Ausgangsports können über eine interne Treiberschaltung digitale Spannungssignale ausgeben. Angeschlossene Schaltungen können einen bestimmten Strom aus den Ports ziehen (bei High-Pegel) bzw. in diesen speisen (bei Low-Pegel).

» Den maximal zulässigen Laststrom für einen einzelnen Port und für alle Ports in der Summe beachten. Eine Überschreitung der Maximalwerte kann zur Zerstörung des C-Control Pro Moduls führen. Nach dem Reset ist zunächst jeder Digitalport als Eingangsport konfiguriert. Über bestimmte Befehle kann die Datenrichtung umgeschaltet werden.

» Es ist wichtig vor der Programmierung die Pinzuordnung zu studieren, da wichtige Funktionen der Programmentwicklung (z.B. die USB Schnittstelle des Application Boards) auf bestimmten Ports liegen. Werden diese Ports umprogrammiert oder sind die zugehörigen Jumper auf dem Application Board nicht mehr gesetzt, kann es passieren, daß die Entwicklungsumgebung keine Programme mehr zum C-Control Pro übertragen kann. Auch Ein- und Ausgänge der Timer, A/D Wandler, I2C und die serielle Schnittstelle sind mit einigen Port Pins verbunden.

PLM-Ports

Es stehen zwei Timer für PLM zur Verfügung. *Timer_0* mit 8 bit und *Timer_1* mit 16 bit. Diese können zur D/A-Wandlung, zur Ansteuerung von Servomotoren im Modellbau oder zur Ausgabe von Tonfrequenzen benutzt werden. Ein pulslängenmoduliertes Signal hat eine Periode von N sogenannten "Ticks". Die Dauer eines Ticks ist die Zeitbasis. Setzt man den Ausgabewert eines PLM-Ports auf X, dann hält dieser für X Ticks einer Periode Highpegel und fällt für den Rest der Periode auf low. Zur Programmierung der PLM-Kanäle siehe Timer.



Die PLM-Kanäle für *Timer_0* und *Timer_1* haben unabhängige Zeitbasis und Periodenlänge. In Anwendungen zur pulsweitenmodulierten Digital-Analogwandlung werden Zeitbasis und Periodenlänge einmalig eingestellt und dann nur der Ausgabewert verändert. Die PLM-Ports sind nach ihren elektrischen Eigenschaften Digitalports. Die technischen Randbedingungen für Digitalports beachten (max. Strom).

Technische Daten Modul

Hinweis: detailliertere Informationen findet man in den PDF-Dateien der IC-Hersteller auf der C-Control Pro Software CD.

Alle Spannungsangaben beziehen sich auf Gleichspannung (DC).

Umgebungsbedingungen	
Bereich der zulässigen Umgebungstemperatur	0°C … 70°C
Bereich der zulässigen relativen Umgebungsluftfeuchte	20% … 60%

Versorgungsspannung	
Bereich der zulässigen Versorgungsspannung	4,5V … 5,5V
Stromaufnahme des Moduls ohne externe Lasten	ca. 20mA

Takt	
Taktfrequenz (Quarzoszillator)	14,7456MHz

Mechanik	
äußere Abmessungen ohne Pins ca.	53 mm x 21mm x 8 mm
Masse	ca. 90g
Pinraster	2,54mm
Pinanzahl (zweireihig)	40
Abstand der Reihen	15,24mm

Ports	
Max. zulässiger Strom aus digitalen Ports	± 20 mA
Zulässige Summe der Ströme an digitalen Ports	200mA
Zulässige Eingangsspannung an Portpins (digital und A/D)	–0,5V ... 5,5V
Interne Pullup Widerstände (abschaltbar)	20 - 50 kOhm

3.2.3 Application Board

USB

Das Application Board verfügt über eine USB Schnittstelle zum Programmladen und Debuggen. Durch die hohe Datenrate dieser Schnittstelle sind die Datenübertragungszeiten gegenüber der seriellen Schnittstelle erheblich kürzer. Die Kommunikation erfolgt über einen USB-Controller von FTDI und einen AVR Mega8 Controller. Der Mega8 hat einen eigenen Reset-Taster (SW5). Im USB-Betrieb wird der Status der Schnittstelle über zwei Leuchtdioden angezeigt (LD4 rot, LD5 grün). Leuchtet nur die grüne LED, so ist die USB-Schnittstelle bereit. Erfolgt eine Datenübertragung, so leuchten beide LEDs. Das gilt auch für den Debugmodus. Ein Blinken der roten LED zeigt einen Fehlerzustand an. Für die USB-Kommunikation wird die SPI-Schnittstelle des Mega32 verwendet (PortB.4 bis PortB.7, PortA.6, PortA.7) und müssen über die entsprechenden Jumper verbunden sein.

Hinweis: Detailliertere Informationen zum Mega 8 findet man in den PDF-Dateien der IC-Hersteller auf der C-Control Pro Software CD.

Ein- Ausschalter

Der Schalter SW4 befindet sich an der Frontseite des Application Boards und dient zum Ein/Ausschalten der Spannungsversorgung.

Leuchtdioden

Es stehen 5 Leuchtdioden zur Verfügung. LD3 (grün) befindet sich an der Frontseite unter dem DC-Anschluß und leuchtet, wenn die Versorgungsspannung vorhanden ist. LD4 und LD5 zeigen den Status der USB-Schnittstelle an (siehe Abschnitt USB). Die grünen Leuchtdioden LD1 und LD2 befinden sich neben den vier Tasten und stehen dem Anwender frei zur Verfügung. Sie sind über einen Vorwiderstand an VCC gelegt. Über Jumper kann LD1 an PortD.6 und LD2 an PortD.7 angeschlossen werden. Die Leuchtdioden leuchten wenn der entsprechende Port Pin low (GND) ist.

Taster

Es sind vier Taster vorgesehen. Mit SW3 (RESET1) wird beim Mega32 ein Reset ausgelöst und mit SW5 (RESET2) ein Reset für den Mega8. Die Taster SW1 und SW2 stehen dem Anwender zur Verfügung. SW1 kann über einen Jumper an PortD.2 gelegt werden und entsprechend SW2 an PortD.3. Es besteht die Möglichkeit SW1/2 entweder gegen GND oder VCC zu schalten. Diese Wahlmöglichkeit wird mit JP1 bzw. JP2 festgelegt. Um bei offenem Taster einen definierten Pegel am Eingangsport zu haben, sollte der entsprechende Pullup eingeschaltet sein (siehe Abschnitt Digitalports).

Ein Drücken von SW1 beim Einschalten des Boards aktiviert den seriellen Bootloadermodus.

LCD

Ein LCD-Modul kann an das Application Board angesteckt werden. Es stellt 2 Zeilen zu je 8 Zeichen dar. Auch anders organisierte Displays können grundsätzlich über diese Schnittstelle betrieben werden. Jedes Zeichen besteht aus einer monochromen Matrix von 5x7 Punkten. Ein blinkender Cursor unter einem der Zeichen kann die aktuelle Ausgabeposition anzeigen. Das Betriebssystem bietet eine einfache Softwareschnittstelle für Ausgaben auf das Display. Das Display wird an den Stecker X14 (16-polig, zweireihig) angeschlossen. Durch einen mechanischen Verpolungsschutz ist ein falsches Einstecken nicht möglich.

Das verwendete LCD Modul ist vom Typ Hantronix HDM08216L-3. Weitere Informationen findet man auf der Hantronix Webseite http://www.hantronix.com und im Datasheets Verzeichnis auf der CD-ROM.

LCD-Kontrast (LCD ADJ)

Die beste Sichtbarkeit der LCD-Zeichen ergibt sich bei frontaler Betrachtung. Gegebenenfalls muß der Kontrast etwas nachgeregelt werden. Der Kontrast kann über den Drehwiderstand PT1 eingestellt werden.

Tastatur

Für Benutzereingaben steht eine 12-er Tastatur (0..9,*,#) zur Verfügung. (X15: 13-poliger Stecker). Die Tastatur ist 1 aus 12 organisiert, d.h. jeder Taste ist eine Leitung zugeordnet. Die Tasteninformation wird seriell über ein Schieberegister eingelesen. Wird keine Tastatur verwendet, so können die 12 Eingänge als zusätzliche Digitaleingänge verwendet werden. Die Tastatur verfügt über einen 13-poligen Anschluß (einreihig) und wird an X15 so angesteckt, dass das Tastenfeld zum Application Board zeigt.

I2C-Schnittstelle

Über diese Schnittstelle können mit hoher Geschwindigkeit serielle Daten übertragen werden. Es werden dazu nur zwei Signalleitungen benötigt. Die Datenübertragung geschieht gemäß dem I2C-Protokoll. Zur effektiven Nutzung dieser Schnittstelle werden spezielle Funktionen zur Verfügung gestellt. (siehe Softwarebeschreibung I2C)

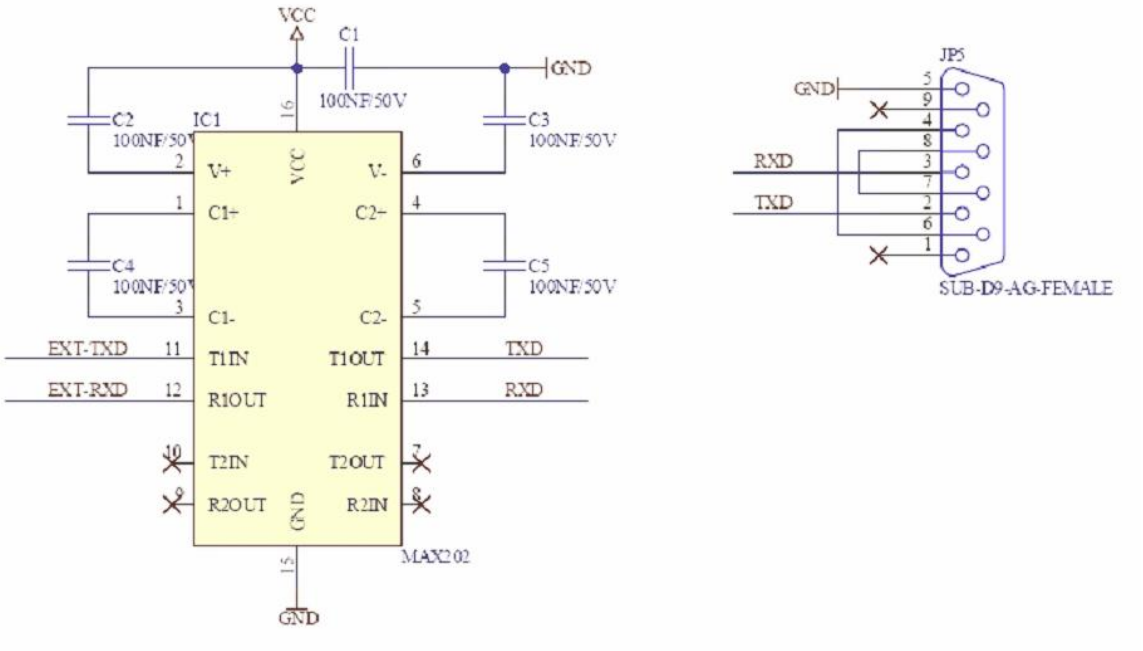
I2C SCL	I2C-Bus Taktleitung	PortC.0
I2C SDA	I2C-Bus Datenleitung	PortC.1

Spannungsversorgung (POWER, 5 Volt, GND)

Das Application Board wird über ein Steckernetzteil (9V/250mA) versorgt. Je nach zusätzlicher Beschaltung des Application Boards kann es später notwendig sein ein Netzteil mit höherer Leistung zu verwenden. Ein Festspannungsregler erzeugt die interne stabilisierte Versorgungsspannung von 5V. Alle Schaltungsteile auf dem Application Board werden mit dieser Spannung versorgt. Durch die Leistungsreserven des Netzteils stehen diese 5V auch zur Versorgung externer ICs zur Verfügung.

Bitte den maximal entnehmbaren Strom beachten. Eine Überschreitung kann zur Zerstörung führen! Wegen der relativ hohen Stromaufnahme des Application Boards im Bereich von 125 mA ist sie für den Einsatz in dauerhaft batteriebetriebenen Geräten nicht zu empfehlen. Bitte den Hinweis zu kurzzeitigen Ausfällen der Versorgungsspannung ("siehe Resetverhalten") beachten.

Serielle Schnittstelle



Der Mikrocontroller Atmega32 besitzt hardwareseitig eine asynchrone serielle Schnittstelle nach RS232-Standard. Das Format kann bei der Initialisierung der Schnittstelle festgelegt werden (Datenbits, Paritätsbit, Stopbit). Auf dem Application Board befindet sich ein hochwertiges Pegelwandler-IC zur Umsetzung der digitalen Bitströme in Non-Return-Zero-Signale nach dem

RS232Standard (positive Spannung für Lowbits, negative Spannung für Highbits). Das Pegelwandler-IC verfügt über einen erhöhten Schutz vor Spannungsspitzen. Spannungsspitzen können in elektromagnetisch belastetem Umfeld, z.B. in industriellen Anwendungen, in die Schnittstellenkabel induziert werden und angeschlossene Schaltkreise zerstören. Über Jumper können die Datenleitungen RxD und TxD mit dem Controller PortD.0 und PortD.1 verbunden werden. Im Ruhezustand (keine aktive Datenübertragung) können Sie am Pin TxD eine negative Spannung von einigen Volt gegen GND messen. RxD ist hochohmig. An der 9-poligen SUB-D Buchse des Application Boards liegt RxD an Pin 3 und TxD an Pin 2. Der GND-Anschluß liegt auf Pin 5. Es werden für die serielle Datenübertragung keine Handshakesignale verwendet.

Eine Kabelverbindung mit Anschluß an die NRZ-Pins TxD, RxD, RTS darf bis zu 10 Metern lang sein. Es sind nach Möglichkeit geschirmte Normkabel zu verwenden. Bei längeren Leitungen oder ungeschirmten Kabeln können Störeinflüsse die Datenübertragung beeinträchtigen. Nur Verbindungskabel verbinden, deren Anschlußbelegung bekannt ist.

» Niemals die seriellen Sendeausgänge zweier Geräte miteinander verbinden! Man erkennt die Sendeausgänge in der Regel an der negativen Ausgangsspannung im Ruhezustand.

Testschnittstellen

Die 4-polige Stiftleiste X16 wird nur für interne Testzwecke verwendet und wird auch nicht auf allen Application Boards bestückt werden. Für den Anwender ist diese Stiftleiste ohne Bedeutung.

Eine weitere Testschnittstelle ist die 6-polige Stiftleiste (zweireihig mit je 3 Pin) bei JP4. Auch diese Stiftleiste ist nur für den internen Gebrauch und wird in späteren Board Serien vermutlich nicht mehr bestückt.

Technische Daten Application Board

Hinweis: Detailliertere Informationen findet man in den PDF-Dateien der IC-Hersteller auf der C-Control Pro Software CD.
Alle Spannungsangaben beziehen sich auf Gleichspannung (DC).

Mechanik	
äußere Abmessungen ca.	160 mm x 100 mm
Pinraster Verdrahtungsfeld	2,54 mm

Umgebungsbedingungen	
Bereich der zulässigen Umgebungstemperatur	0°C … 70°C
Bereich der zulässigen relativen Umgebungsluftfeuchte	20% … 60%

Versorgungsspannung	
Bereich der zulässigen Versorgungsspannung	8V … 24V
Stromaufnahme ohne externe Lasten	ca. 125mA
max. zulässiger Dauerstrom aus der stabilisierten 5V-Spannung	200mA

3.2.4 Pinzuordnung

PortA bis PortD werden für direkte Pin-Funktionen (z.B. Port_WriteBit) von 0 bis 31 gezählt, siehe "PortBit".

Pinbelegung für Application Board Mega32 (Pins 1-20)

PIN	Port	Port	PortBit	Name	Schaltplan	Bemerkungen
1	PB0	PortB.0	8	T0		Eingang Timer/Counter0
2	PB1	PortB.1	9	T1		Eingang Timer/Counter1
3	PB2	PortB.2	10	INT2/AIN0		(+)Analog Comparator, externer Interrupt2
4	PB3	PortB.3	11	OT0/AIN1		(-)Analog Comparator, Ausgang Timer0
5	PB4	PortB.4	12		SS	USB-Kommunikation
6	PB5	PortB.5	13		MOSI	USB-Kommunikation
7	PB6	PortB.6	14		MISO	USB-Kommunikation
8	PB7	PortB.7	15		SCK	USB-Kommunikation
9				RESET		
10				VCC		
11				GND		
12				XTAL2		Oszillator : 14,7456MHz
13				XTAL1		Oszillator : 14,7456MHz
14	PD0	PortD.0	24	RXD	EXT-RXD	RS232, serielle Schnittstelle
15	PD1	PortD.1	25	TXD	EXT-TXD	RS232, serielle Schnittstelle
16	PD2	PortD.2	26	INT0	EXT-T1	SW1 (Taster1); externer Interrupt0
17	PD3	PortD.3	27	INT1	EXT-T2	SW2 (Taster2); externer Interrupt1
18	PD4	PortD.4	28	OT1B	EXT-A1	Ausgang B Timer1
19	PD5	PortD.5	29	OT1A	EXT-A2	Ausgang A Timer1
20	PD6	PortD.6	30	ICP	LED1	Leuchtdiode; Input Capture Pin Puls/Periodenmess.

Pinbelegung für Application Board Mega32 (Pins 21-40)

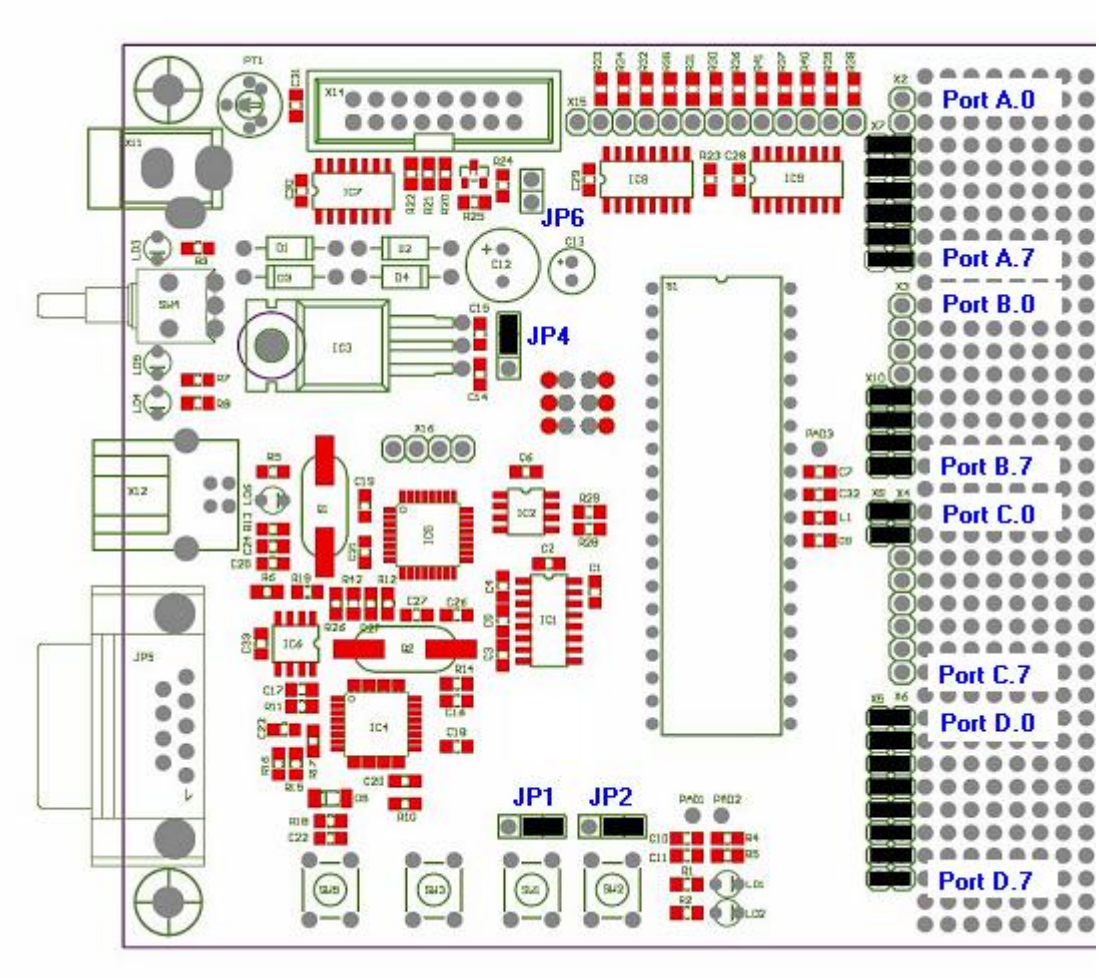
21	PD7	PortD.7	31		LED2	Leuchtdiode
22	PC0	PortC.0	16	SCL	EXT-SCL	I2C-Interface
23	PC1	PortC.1	17	SDA	EXT-SDA	I2C-Interface
24	PC2	PortC.2	18			
25	PC3	PortC.3	19			
26	PC4	PortC.4	20			
27	PC5	PortC.5	21			
28	PC6	PortC.6	22			
29		PortC.7	23			
30				AVCC		
31				GND		
32				AREF		
33	PA7	PortA.7	7	ADC7	RX_BUSY	ADC7 Eingang;USB-Kommunikation
34	PA6	PortA.6	6	ADC6	TX_REQ	ADC6 Eingang;USB-Kommunikation
35	PA5	PortA.5	5	ADC5	KEY_EN	ADC5 Eingang;LCD/Tastatur Interface
36	PA4	PortA.4	4	ADC4	LCD_EN	ADC4 Eingang;LCD/Tastatur Interface
37	PA3	PortA.3	3	ADC3	EXT_SCK	ADC3 Eingang;LCD/Tastatur Interface
38	PA2	PortA.2	2	ADC2	EXT_DATA	ADC2 Eingang;LCD/Tastatur Interface
39	PA1	PortA.1	1	ADC1		ADC1 Eingang;
40	PA0	PortA.0	0	ADC0		ADC0 Eingang;

3.2.5 Jumper Application Board

Jumper

Durch Jumper können bestimmte Optionen ausgewählt werden. Das betrifft einige Ports, welche mit speziellen Funktionen belegt sind (siehe Tabelle der Pinzuordnung). Beispielsweise ist die serielle Schnittstelle über die Pins PortD.0 und PortD.1 realisiert. Wird die serielle Schnittstelle nicht benutzt, so können die entsprechenden Jumper entfernt werden und diese Pins stehen dann für andere Funktionen zur Verfügung. Neben den Jumpern für die Ports gibt es noch zusätzliche Jumper, welche nachfolgend beschrieben werden.

Jumperpositionen im Auslieferzustand



JP1 und JP2

Die Jumper sind den Tastern SW1 und SW2 zugeordnet. Es besteht die Möglichkeit die Taster gegen GND oder VCC zu betreiben. In der Grundeinstellung schalten die Taster gegen GND.

JP4

JP4 dient zum Umschalten der Betriebsspannung (Netzteil oder USB). Das Application Board sollte mit Netzteil und Spannungsregler betrieben werden (Auslieferzustand).

JP6

Bei Verwendung des Displays kann mit JP6 die Beleuchtung (back light) abgeschaltet werden.

3.2.6 Schaltpläne

3.2.6.1 PDF Schaltpläne

Aus Konvertierungsgründen sind die Schaltpläne nicht im PDF Handbuch. Sie sind unter "Manual\Mega32 Schaltpläne" zu finden.

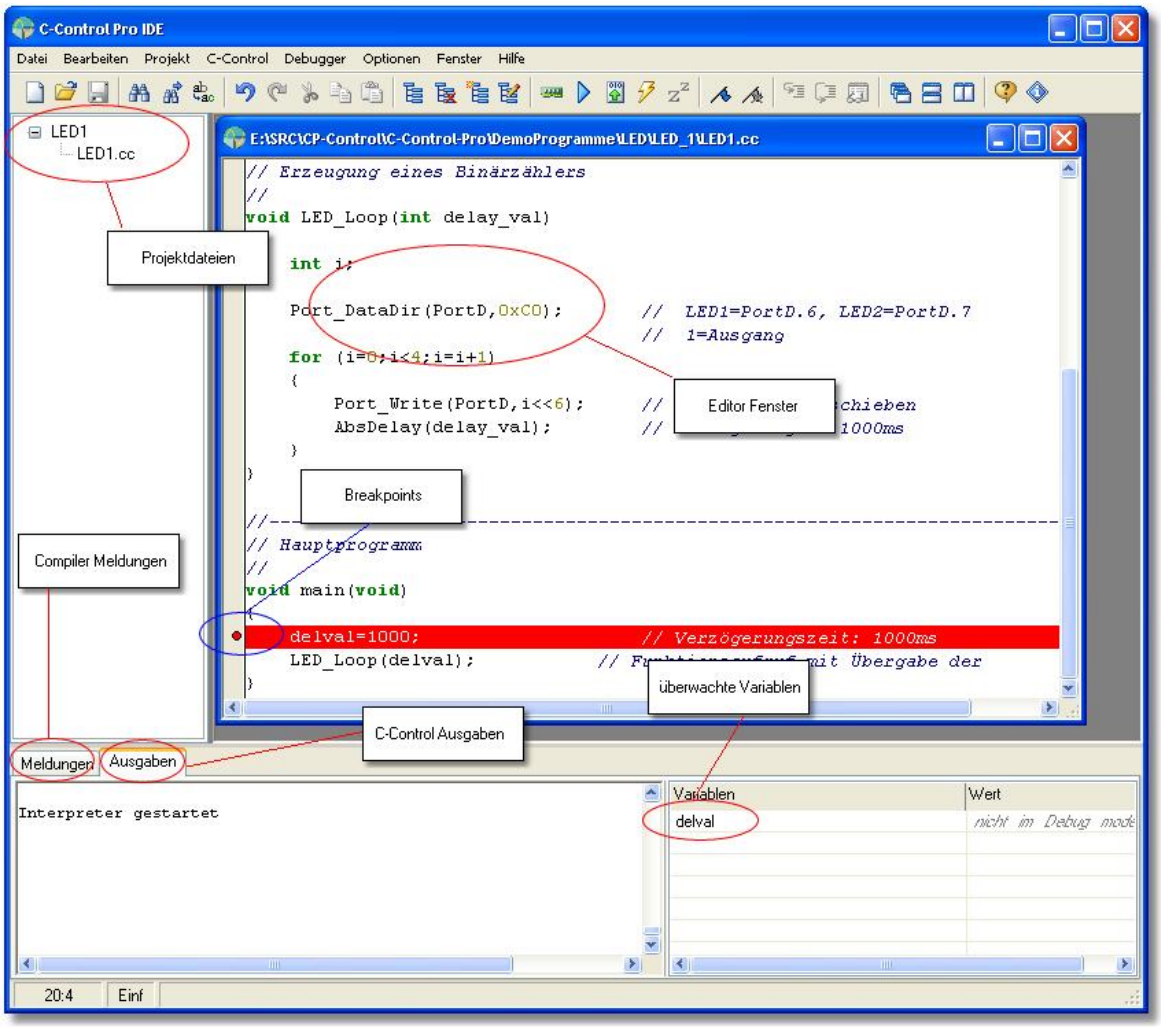
Kapitel 4

4 IDE

4.1 Übersicht

Die C-Control Pro Benutzeroberfläche (IDE) besteht aus folgenden Hauptelementen:

Sidebar für Projekt Dateien	Mehrere Dateien können hier zu einem Projekt abgelegt werden.
Editor Fenster	Es können beliebig viele Editor Fenster geöffnet werden um Dateien zu editieren.
Compiler Meldungen	Fehlermeldungen und allgemeine Compilerinformationen werden hier angezeigt.
C-Control Ausgaben	Ausgabe von Debug Nachrichten der CompactC Programme.
Variablenfenster	Überwachte Variablen werden hier angezeigt



4.2 Projekte

4.2.1 Projekterstellung

Unter dem Menü Projekt kann man mit dem Aufruf von Neu die *Projekt erstellen* Dialogbox aufrufen. Es wird dort für das Projekt ein Projektname angegeben und das Projekt wird in der Sidebar erstellt.

» Man muß sich vorher nicht entscheiden ob man ein CompactC oder ein Basic Projekt erstellt. In einem Projekt kann man als Projektdateien CompactC und Basic Dateien gemischt anlegen und daraus ein Programm erzeugen.

4.2.2 Projektverwaltung

Klickt man mit der rechten Maustaste auf das Neu erstellte Projekt in der Sidebar, so erscheint ein Popupmenü mit den Optionen

- Neu Hinzufügen - Es wird eine neue Datei angelegt und gleichzeitig ein Editorfenster geöffnet
- Hinzufügen - Eine bestehende Datei wird dem Projekt hinzugefügt
- Umbenennen - Der Name des Projektes wird geändert (Dies ist nicht unbedingt der Name der Projektdatei)
- Kompilieren - Der Compiler wird für das Projekt gestartet
- Optionen - Die Projektoptionen können geändert werden

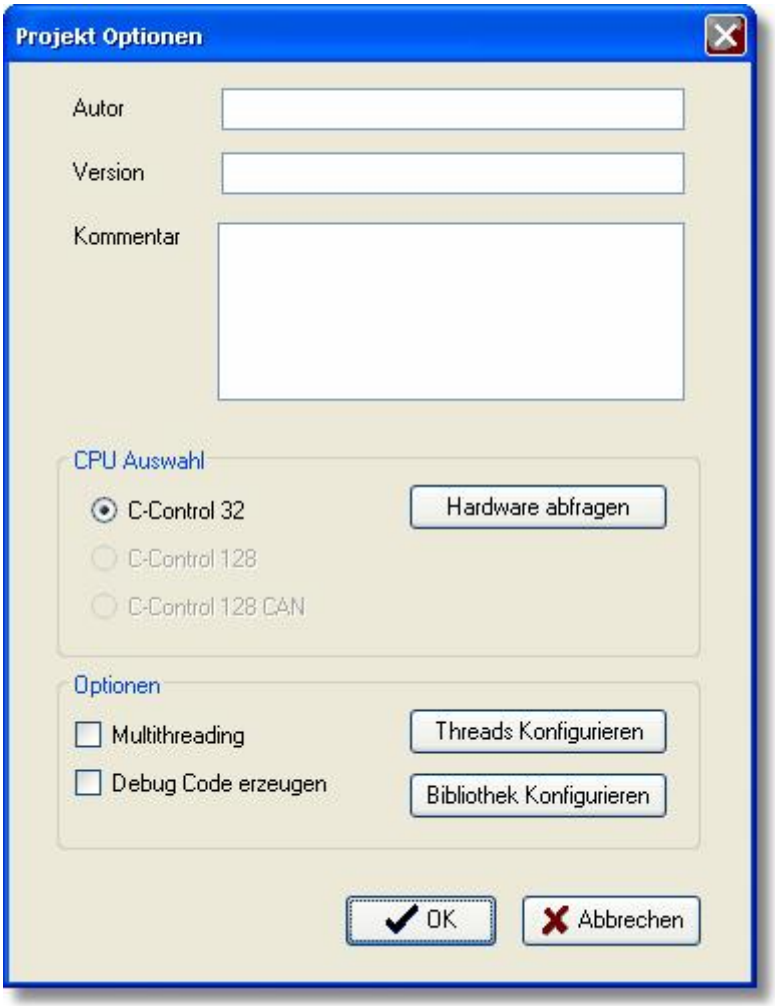
Projektdateien

Hat man zum Projekt Dateien hinzugefügt, dann kann man die Dateien mit einem Doppelklick auf den Dateinamen öffnen. Mit einem Rechtsklick erscheinen weitere Optionen:

- Umbenennen - Der Name der Projektdatei wird geändert
- Entfernen - Die Datei wird aus dem Projekt entfernt
- Optionen - Die Projektoptionen können geändert werden.

4.2.3 Projektoptionen

Für jedes Projekt können die Compilereinstellungen einzeln geändert werden.



Die Einträge *Autor*, *Version*, *Kommentar* können frei beschriftet werden, sie dienen nur als Erinnerungsstütze um sich später einmal besser an Einzelheiten des Projekts erinnern zu können.

In "CPU Auswahl" legt man die Zielplattform des Projekts fest. Klickt man auf "*Hardware Abfragen*" dann wird das angeschlossene C-Control Pro Modul abgefragt und die CPU richtig ausgewählt.

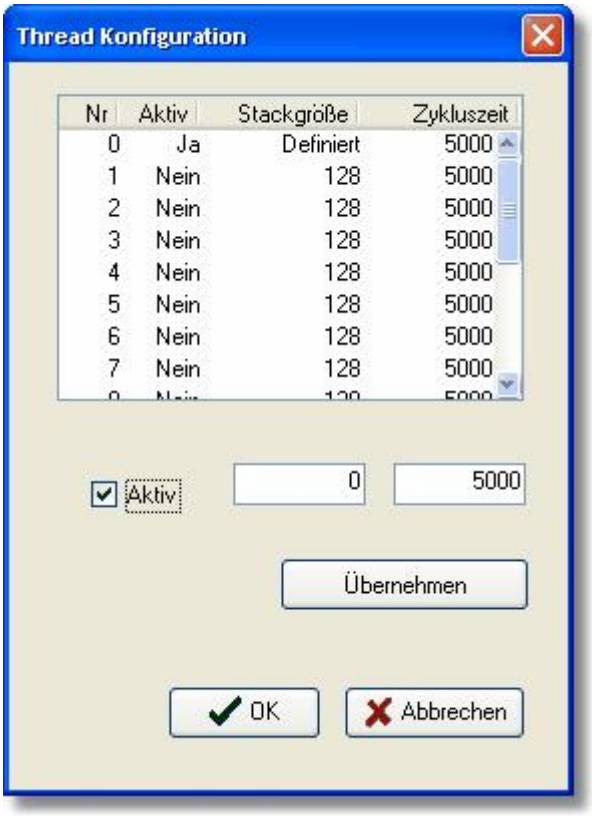
Bei den "Optionen" konfiguriert man das Multithreading und ob Debug Code erzeugt werden soll.

» Wird mit Debug Code kompiliert wird der Bytecode geringfügig länger. Pro Zeile im Quelltext die ausführbare Anweisungen enthält, wird der Bytecode um ein Byte größer.

» Soll Multithreading genutzt werden, muß in den Projekt Optionen die Auswahlbox selektiert werden und zusätzlich müssen die Threads unter "Threads Konfigurieren" einzeln parametrisiert werden.

4.2.4 Threadoptionen

Um einen Thread zur Laufzeit aktivieren zu können muß er in dieser Auswahlbox aktiviert werden und die Parameter *Stackgröße* und *Zykluszeit* eingestellt werden.



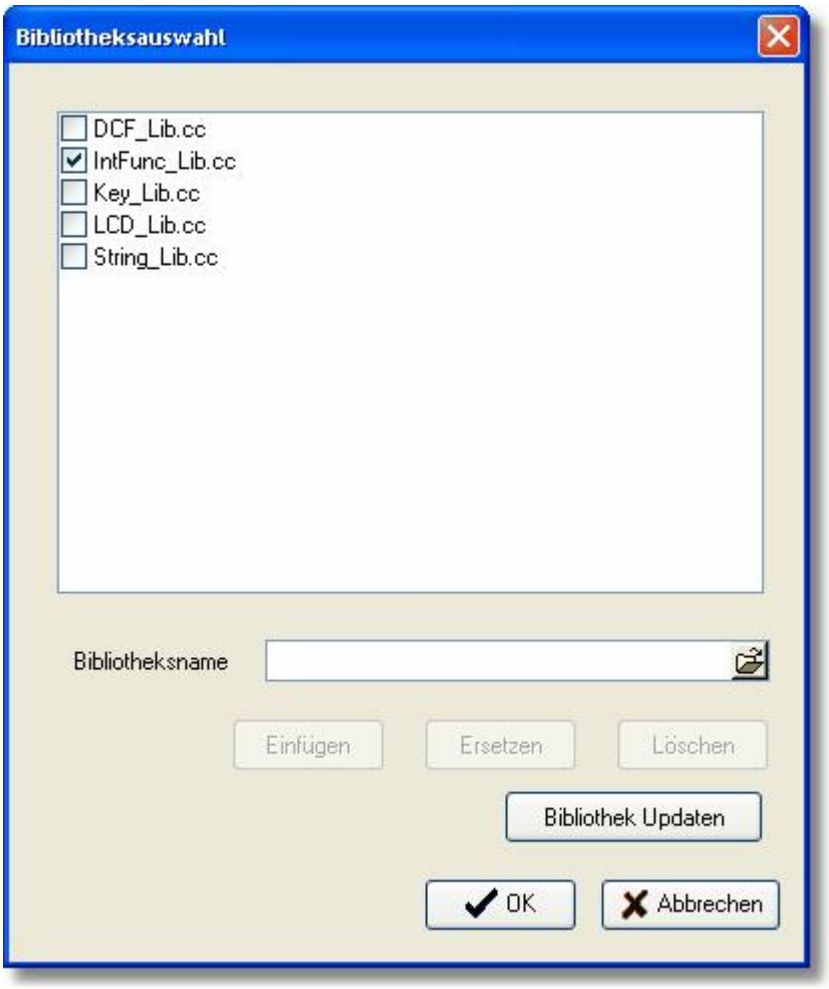
Jedem zusätzlichen Thread außer dem Hauptprogramm wird ein Platz auf dem Stack zugewiesen, den er nicht überschreiten darf.

» Benutzt ein Thread mehr Platz als zugewiesen, wird der Speicherplatz der anderen Threads in Mitleidenschaft gezogen und ein Absturz des Programms ist sehr wahrscheinlich.

Die Zykluszeit ist die Anzahl der Zyklen (Bytecode Operationen) die ein Thread verarbeiten darf bis zu einem anderen Thread gewechselt wird. Über die Anzahl der Zyklen bis zum Threadwechsel wird auch die Priorität der Threads gesteuert. Siehe auch Threads.

4.2.5 Bibliotheksverwaltung

In der Bibliotheksverwaltung können die Quelltext Bibliotheken abgewählt werden, die zusätzlich zu den Projektdateien kompiliert werden.



Es werden nur die Dateien zur Kompilierung hinzugezogen deren CheckBox auch selektiert wurde.

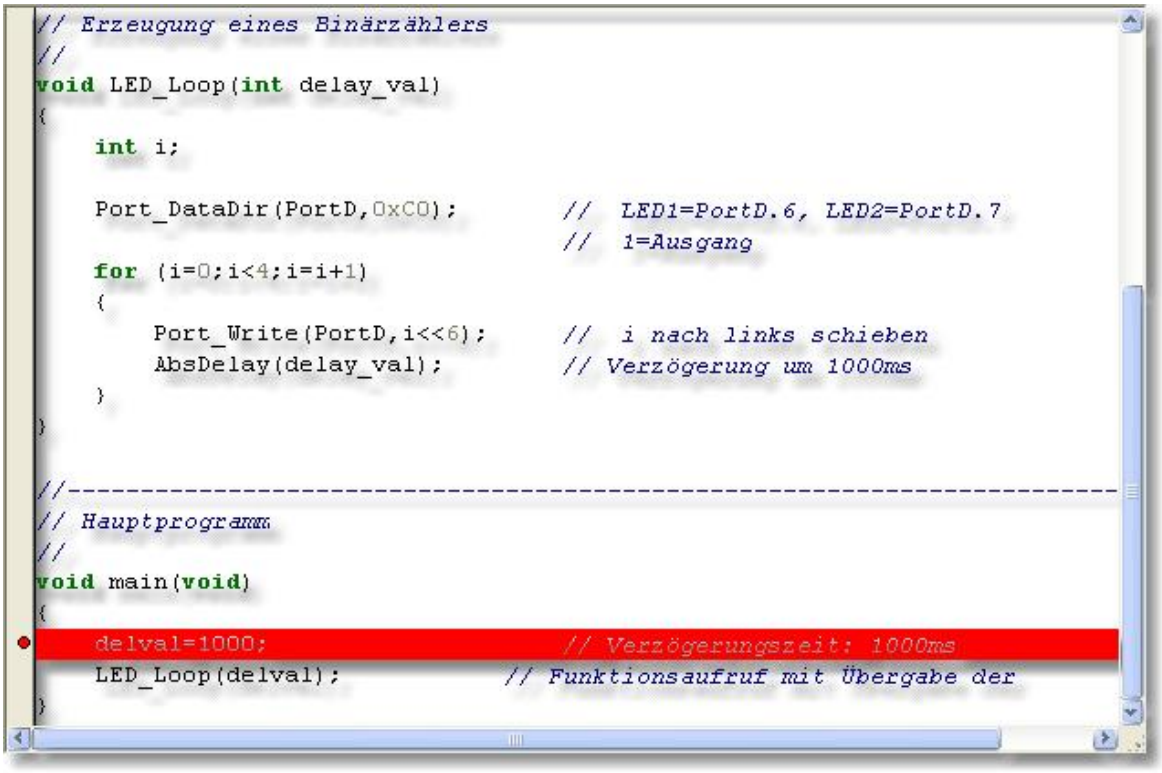
Die Liste kann mit Hilfe des Pfad Texteingabefeldes "Bibliotheksname" und den Buttons im Dialog geändert werden:

- Einfügen - Der Pfad wird zur Liste hinzugefügt
- Ersetzen - Der selektierte Eintrag in der Liste wird durch den Pfadnamen ersetzt
- Löschen - Der selektierte Listeneintrag wird gelöscht
- Bibliothek Updaten - Dateien die in der Compilervoreinstellung vorhanden sind, aber in dieser Liste nicht, werden hinzugefügt

4.3 Editor

4.3.1 Editorfenster

Man kann in der C-Control Pro Oberfläche mehrere Editorfenster öffnen. Jedes Fenster läßt sich in der Größe und im gezeigten Textauschnitt verändern. Ein Doppelklick auf die Titelzeile maximiert das Fenster.



Ein Klick auf den Bereich links neben den Textanfang setzt dort einen Haltepunkt (Breakpoint). Dazu muß vorher der Quelltext fehlerfrei mit "*Debug Info*" kompiliert worden sein, und in der entsprechenden Zeile tatsächlich ausführbarer Programmtext stehen (z.B. keine Kommentarzeile o. ä.).

4.3.2 Editorfunktionen

Unter dem Menüpunkt Bearbeiten sind die wichtigsten Editorfunktionen zu finden:

- Rückgängig (Ctrl-Z) - führt eine Undo Operation aus. Wieviel dieser Befehl rückgängig macht hängt auch von der Einstellung von Gruppen Rückgängig ab.
- Wiederherstellen (Ctrl-Y) - stellt den Editorzustand wieder her, der vorher durch Rückgängig verändert wurde
- Ausschneiden (Ctrl-X) - schneidet selektierten Text aus und kopiert ihn in die Ablage
- Kopieren (Ctrl-C) - kopiert selektierten Text in die Ablage
- Einfügen (Ctrl-V) - kopiert den Inhalt der Ablage an die Cursorposition
- Alles Markieren (Ctrl-A) - selektiert den gesamten Text

- Suchen (Ctrl-F) - Öffnet den Suchen Dialog
- Weitersuchen (F3) - sucht weiter mit den gleichen Suchkriterien
- Ersetzen (Ctrl-R) - Öffnet den Ersetzen Dialog
- Gehe zu (Alt-G) - man kann zu einer bestimmten Zeile springen

Suchen/Ersetzen Dialog

- Suchtext - Eingabefeld für den zu suchenden Text
- Ersetzen mit - der Text der den gefunden Text ersetzt
- Groß/Kleinschreibung - unterscheidet Groß- und Kleinschreibung
- Nur ganze Wörter - findet nur ganze Wörter und keine Teilzeichenketten
- reguläre Ausdrücke - aktiviert die Eingabe von regulären Ausdrücken in der Suchmaske
- Nachfrage bei Treffer - Vor dem Ersetzen wird beim Benutzer nachgefragt

Weiter kann die Suchrichtung bestimmt werden, ob der gesamte Text oder nur ein selektierter Bereich durchsucht wird, und ob die Suche am Cursor beginnt oder am Textanfang.

4.3.3 Editoreinstellungen

- Automatisches Einrücken - drückt man Enter wird der Cursor auf der nächsten Zeile bis zum Anfang der vorherigen Zeile eingerückt

- Einfügen - ist diese Option aus, ist Überschreiben als Standard eingestellt
- Benutze Tabulator - ist dies aktiviert werden Tab Zeichen eingefügt, sonst werden Leerzeichen benutzt
- Smart Tabulator - mit Tabulator springt man an die Stelle an der die Zeichen der vorherige Zeile beginnen
- Optimales Füllen - "Automatisches Einrücken" füllt zuerst mit Tabulatoren und den Rest mit Leerzeichen
- Backspace rückt aus - mit Backspace springt man an die Stelle an der die Zeichen der vorherige Zeile beginnen
- Cursor geht durch Tabulatoren - man geht durch Tabulatoren wie durch Leerzeichen
- Gruppen Rückgängig - eine Undo Operation wird nicht in kleinen Schritten, sondern in Blöcken durchgeführt
- Cursor hinter Dateiende - man kann den Cursor hinter das Dateiende positionieren
- Cursor hinter Zeilenende - man kann den Cursor hinter das Zeilenende bewegen
- Erlaube Undo nach speichern - der Undo Puffer wird nach dem Speichern nicht geleert
- Folgende Leerzeichen behalten - ist dies aktiviert, werden Leerzeichen am Ende der Zeile nicht gelöscht
- Blöcke überschreiben - ist ein Block selektiert, so ersetzt die nächste Eingabe den Block
- Erlaube Selektion - Text kann selektiert werden
- Erlaube Draggen - Selektierter Text kann mit der Maus "gedragged" (bei gedrückter linker Maustaste verschoben) werden
- Markierung bei Suchoperation - wird ein gesuchter Text gefunden, ist das Ergebnis selektiert
- Doppelklick selektiert Zeile - normalerweise selektiert ein Doppelklick ein Wort
- Suchtext Text von Cursor - der Text beim "Suchtext" Eingabefeld wird von der Cursorposition übernommen
- Dreifachklick selektiert Zeile - wenn Doppelklick ein Wort selektiert, kann mit dieser Option mit Dreifachklick eine Zeile selektiert werden
- autom. Rechtschreibprüfung - diese Option schaltet die Rechtschreibprüfung in den Kommentaren ein
- Benutze Syntax Einfärbung - das Syntax Highlighting für *.cc und *.cbas Dateien wird eingeschaltet

In der Auswahlbox Tastaturbelegung kann man das Tastenlayout von gängigen Editoren einstellen. Diese Emulation ist aber nur unvollständig da das Verhalten der verschiedenen Editoren sehr komplex ist. Die wichtigsten Tastaturbefehle werden aber meist unterstützt.

Bei Block Einfügen wird die Anzahl der Leerzeichen eingetragen mit der ein selektierter Block mit der Tabulator Taste eingerückt bzw. ausgerückt wird.

Das Eingabefeld Tabulatoren bestimmt wieviele Zeichen ein Tabulator breit ist.

4.3.4 reguläre Ausdrücke

Die Suchfunktion im Editor unterstützt reguläre Ausdrücke. Damit können Zeichenketten sehr flexibel gesucht oder ersetzt werden.

^	Ein Circumflex am Anfang eines Wortes findet das Wort am Anfang einer Zeile
$	Ein Dollarzeichen repräsentiert das Ende einer Zeile
.	Ein Punkt symbolisiert ein beliebiges Zeichen
*	Ein Stern steht für ein mehrfaches Auftreten eines Musters. Die Anzahl darf aber auch null sein
+	Ein Plus steht für ein mehrfaches aber mindestens einmaliges Auftreten eines Musters
[ ]	Zeichen in eckigen Klammern repräsentieren das Auftauchen eines der Zeichen
[^]	Ein Circumflex innerhalb einer eckigen Klammer negiert die Auswahl
[-]	Ein Minuszeichen innerhalb einer eckigen Klammer symbolisiert einen Buchstabenbereich
{ }	geschweifte Klammern gruppieren einzelne Ausdrücke. Es dürfen bis zu 10 Ebenen geschachtelt werden
\	der Rückwärtsschrägstrich nimmt dem folgenden Zeichen die besondere Bedeutung

Beispiele

Beispiel	**findet**
^void	das Wort "void" nur am Zeilenanfang
;$	das Semikolon nur am Zeilenende
^void$	in der Zeile darf nur "void" stehen
vo.*d	z.B. "vod","void","vqqd"
vo.+d	z.B. "void","vqqd" aber nicht "vod"
[qs]	die Buchstaben 'q' oder 's'
[qs]port	"qport" oder "sport"
[^qs]	alle Buchstaben außer 'q' oder 's'
[a-g]	alle Buchstaben zwischen 'a' und 'g' (inklusive)
{tg}+	z.B. "tg", "tgtg", "tgtgtg" usw.
\$	'$'

4.4 Compiler

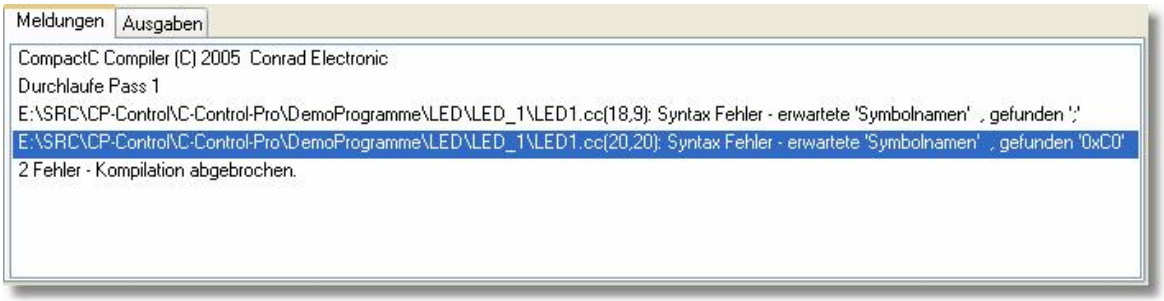
4.4.1 Compilervoreinstellung

In der Compilervoreinstellung können die Standardwerte konfiguriert werden die beim erzeugen eines neues Projekts gespeichert werden.

Die Auswahlboxen "Threads Konfigurieren" und "Bibliothek Konfigurieren" sind identisch zu den Einstellungs Parametern bei Projektoptionen.

4.4.2 Kompilieren

Unter dem Menüpunkt Projekt kann mit Kompilieren (F9) das aktuelle Projekt vom Compiler übersetzt werden. Die Compiler Meldungen werden in einem eigenen Fensterbereich ausgegeben.



Kommt es bei der Kompilierung zu Fehlern, so wird pro Zeile ein Fehler beschrieben. Die Form ist:

```
Dateiname(Zeile,Spalte): Fehlerbeschreibung
```

Man kann die Fehlerposition im Quelltext über die Befehle Nächster Fehler (F11) oder Vorheriger Fehler (Shift-F11) finden. Beide Befehle sind im Menüpunkt Projekt. Alternativ kann man auch mit einem Doppelklick auf eine Fehlermeldung des Compilers den Cursor im beim Fehler im Editor positionieren.

Bei erfolgreicher Kompilierung wird der Bytecode als Datei mit der Endung "*.bc" im Projektverzeichnis abgelegt.

Mit einem Rechtsklick im Bereich der Compiler Meldungen lassen sich folgende Vorgänge auslösen:

- löschen - löscht die Liste der Compiler Meldungen
- in Ablage kopieren - kopiert alle Textnachrichten in die Zwischenablage

4.5 C-Control Hardware

4.5.1 Programm starten

Programmübertragung

Ist ein Projekt fehlerfrei übersetzt worden, so muß der Bytecode erst auf den Mega 32 übertragen werden bevor er ausgeführt werden kann. Dies geschieht mit dem Befehl Übertragen (Shift-F9) aus dem Menü C-Control.

» Es wird nicht nur der Bytecode zum Mega 32 Modul übertragen, sondern gleichzeitig wird die neueste Version des Interpreters mit zum C-Control Modul geschickt.

Starten

Durch Starten (F10) wird dann die Ausführung des Bytecode auf dem Mega 32 veranlaßt.

Stoppen

Im normalen Betrieb wird ein Programm durch Drücken auf den Taster RESET1 gestoppt. Aufgrund von Performancegründen wird die Programmausführung auf dem Modul im normalen Betrieb nicht per Software angehalten. Dies ist aber mit der IDE Funktion Programm Stoppen möglich, wenn das Programm im Debugmodus läuft.

» In seltenen Fällen kann sich im USB Betrieb beim Druck auf den Taster RESET1 das System verklemmen. Bitte dann den Taster RESET2 betätigen um auch dem Mega8 einen Reset Impuls zu geben. Der Mega8 kümmert sich auf dem Application Board um die USB Schnittstelle.

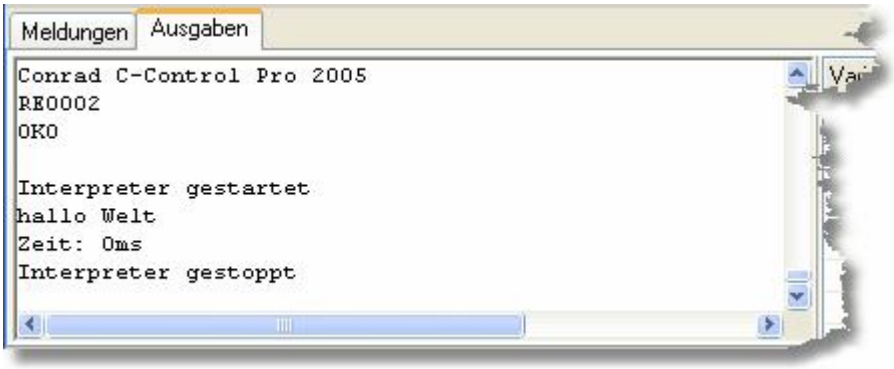
Autostart

Ist kein USB Interface angeschlossen, und wurde beim Einschalten nicht SW1 gedrückt um in den seriellen Bootloadermodus zu kommen, wird der Bytecode (sofern vorhanden) im Interpreter gestartet. D.h. wird das Modul in eine Hardware Applikation eingebaut, so reicht ein Anlegen der Betriebsspannung um das Anwenderprogramm automatisch zu starten.

» Ein Signal auf INT_0 beim einschalten des C-Control Pro Moduls kann das Autostartverhalten stören. Nach der Pinzuordnung liegt der INT_0 auf dem gleichen Pin wie der SW1. Wird der SW1 beim einschalten des Moduls gedrückt, führt dies zur Aktivierung des seriellen Bootloader Modus und das Programm wird nicht automatisch gestartet.

4.5.2 Ausgaben

Um Debug Nachrichten anzuzeigen gibt es einen "Ausgaben" Fensterbereich.



Es wird hier angezeigt wann der Bytecode Interpreter gestartet und beendet worden ist, und wie lange (in Millisekunden) der Interpreter ausgeführt wurde. Die Ausführungszeit ist natürlich nicht aussagekräftig wenn der Interpreter im Debug Modus angehalten wurde.

Im Ausgaben Fenster kann aber auch der Benutzer seine eigenen Debugnachrichten anzeigen lassen. Zu diesem Zweck existieren mehrere Debug Funktionen.

Mit einem Rechtsklick im Bereich der Debug Ausgaben lassen sich folgende Befehle anwählen:

- löschen - löscht die Liste der Debug Ausgaben
- in Ablage kopieren - kopiert alle Textnachrichten in die Zwischenablage

4.5.3 PIN Funktionen

Einzelne Funktionen des Interpreters lassen sich mit einer alphanumerischen PIN schützen. Ist ein Interpreter durch eine PIN gesichert, so sind normale Operationen verboten. Er kann durch eine erneute Übertragung überschrieben werden, aber die PIN bleibt erhalten. Auch ein normales Starten ist nicht mehr erlaubt, mit Ausnahme des Autostart Verhaltens. Auch die Abfrage der Versionsnummern von Hardware und Firmware ist gesperrt.

Möchte man auf eine verbotene Funktion zugreifen, kommt ein Dialog mit dem Text "C-Control ist Passwortgeschützt. Operation nicht erlaubt!".

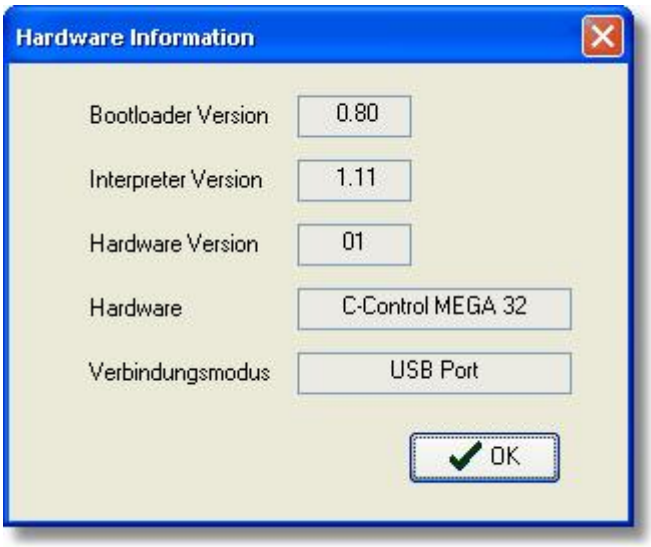
Durch Eingabe der PIN über PIN Eingeben im C-Control kann man alle Operationen freischalten.

Um eine neue PIN einzutragen oder eine gesetzte PIN zu löschen existieren die Befehle PIN Setzen und PIN Löschen im C-Control Menü. War schon eine PIN gesetzt, so muß das Modul natürlich erst durch Eingabe der alten PIN entsperrt werden. Eine PIN darf bis zu 6 alphanumerische Zeichen lang sein.

» Hat man das Passwort vergessen, gibt es eine Notfallfunktion um das Modul in den Ausgangszustand zurückzusetzen. Unter C-Control existiert die Option Modul zurücksetzen, mit der man PIN, Interpreter und Programm löschen kann.

4.5.4 Versionsüberprüfung

Da es geplant ist, nach dem C-Control Pro MEGA 32 weitere Hardware Plattformen zu unterstützen, ist es wichtig die aktuellen Versionsnummern von Bootloader, Interpreter und Hardwareversion im Auge zu behalten. Dies ist mit Hardware Version im Menü C-Control möglich.



4.6 Debugger

4.6.1 Breakpoints

Um den Debugger zu aktivieren muß das Projekt erst fehlerfrei mit Debug Code kompiliert worden und zum Modul übertragen worden sein. Die Datei mit dem Debug Code (*.dbg) muß im Projektverzeichnis vorliegen.

Der Editor erlaubt es bis zu 16 Haltepunkte (Breakpoints) zu setzen. Ein Breakpoint wird eingetragen in dem links neben den Anfang einer Zeile mit der Maus geklickt wird (siehe Übersicht oder Editorfenster).

» Die Anzahl der Breakpoints ist auf 16 begrenzt weil diese Information beim Lauf des Bytecode Interpreters im RAM mitgeführt wird. Andere Debugger setzen Haltepunkte direkt in den Programmcode. Dies ist hier nicht erwünscht, da es die Lebenszeit des Flashspeichers (ca. 10000 Schreibzugriffe) drastisch reduzieren würde.

Im Debugger Menü sind alle Debugger Befehle zu finden. Mit Debug Modus (Shift-F10) startet man den Debugger. Ist zu diesem Zeitpunkt kein Breakpoint gesetzt so hält der Debugger auf der ersten ausführbaren Anweisung.

Ist man im Debug Modus, so springt man mit Starten (F10) zum nächsten Haltepunkt. Ist kein Breakpoint gesetzt so wird das Programm normal abgearbeitet, mit der Ausnahme daß der Programmlauf mit Programm Stoppen angehalten werden kann. Dies funktioniert aber nur wenn das Programm aus dem Debug Modus heraus gestartet wurde.

Hat der Debugger im Programm angehalten (der grüne Balken ist sichtbar) so kann man das Programm im Einzelschritt (Singlestep) ausführen lassen. Die Befehle Einzelschritt (Shift-F8) und Prozedurschritt (F8) führen jeweils den Programmcode bis zur nächsten Codezeilen aus und bleiben dann stehen. Im Unterschied zu Einzelschritt springt Prozedurschritt nicht in Funktionsaufrufe sondern geht über sie hinweg.

Ist in einer Schleife nur eine Codezeile so führt ein Einzelschritt die ganze Schleife aus, da erst dann zu einer neuen Codezeile verzweigt wird.

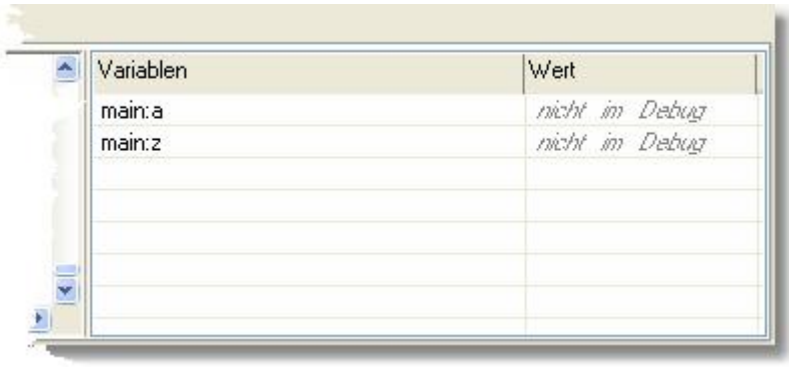
Mit der Anweisung Debug Modus verlassen wird der Debug Modus beendet.

Während der Debug Modus aktiv ist, kann der Programmtext nicht geändert werden. Dies geschieht damit sich die Zeilennummern nicht verschieben können wo Breakpoints gesetzt wurden. Der Debugger wäre sonst nicht in der Lage sich mit dem Bytecode auf dem C-Control Modul zu synchronisieren.

4.6.2 Variablen

Man kann sich im Debugger den Inhalt von Variablen anzeigen lassen. Dafür positioniert man den Mauszeiger über der Variablen, und nach ca. 2 Sekunden wird der Inhalt der Variablen als Tooltip angezeigt.

Möchte man mehrere Variablen überwachen, so kann man die Variablen in einer Liste zusammenfassen.



Um eine Variable in die Liste der überwachten Variablen einzutragen, existieren zwei Möglichkeiten. Man kann zum einen den Cursor am Beginn einer Variable im Texteditor positionieren, und dann mit Rechtsklick Variable Einfügen anwählen.

Die andere Variante geht über das Kontextmenü in der Variablenliste, das man auch über Rechtsklick aktivieren kann:

Wählt man dort Variable Einfügen an, so kann man die zu überwachende Variable in der Liste als Text eintragen. Ist es eine lokale Variable, so wird dort der Funktionsname mit einem Doppelpunkt vorangestellt (**Funktionsname** : **Variablenname**). Mit Variable Ändern kann man den Texteintrag in der Liste ändern, und mit Variable entfernen die Variable aus der Liste entfernen. Dabei muß vorher die Zeile mit der zu löschenden Variable selektiert worden sein. Das Kommando Alle Variablen entfernen löscht alle Einträge aus der Liste.

» Es ist nicht möglich sich den Inhalt von Arrays im Debugger anzusehen.

Unter bestimmten Bedingungen wird anstatt einem Wert in der Liste eine Fehlermeldung angezeigt:

kein Debug Code	es wurde kein Debug Code generiert
falsche Syntax	bei der Texteingabe wurden ungültige Zeichen für die Variable eingegeben
Funktion unbekannt	der Funktionsname ist nicht bekannt
Variable unbekannt	der Variablenname ist nicht bekannt
nicht im Debug mode	der Debug Modus wurde nicht aktiviert
kein Kontext	lokale Variablen können nur angezeigt werden, wenn man sich in der Funktion befindet
nicht aktuell	der Variableninhalt wurde nicht aktualisiert

Sind viele Variablen in die Überwachungsliste eingetragen, so kann es bei einem Einzelschritt lange dauern bis alle Variableninhalte aus dem Modul abgefragt worden sind. Zu diesem Zweck kann die Option Auto Aktualisieren für einzelne Variablen ausschalten. Dann werden die Inhalte dieser Variablen erst dann angezeigt wenn der Befehl Variablen Aktualisieren durchgeführt wird. So läßt sich schnell im Debugger mit Einzelschritt fortfahren und die Inhalte werden erst bei Bedarf aktualisiert.

Man kann sich die Werte der Variablen als Dezimal Zahl oder als Hexzahl anzeigen lassen. Variablen vom Typ **char** werden im Dezimalmodus als ASCII Zeichen dargestellt.

4.7 IDE Einstellungen

Man kann einzelne Aspekte der IDE konfigurieren.

- Übertragung nach Kompilieren Abfrage - Wurde ein Programm kompiliert aber nicht zum C-Control Modul übertragen, erfolgt ein Nachfrage beim Benutzer ob das Programm gestartet werden soll
- Letztes Projekt wieder öffnen - Das letzte offene Projekt wird beim Starten der C-Control Pro IDE wieder geöffnet
- Editorfenster maximiert öffnen - Bei dem Öffnen einer Datei wird automatisch das Editorfenster auf volle Größe geschaltet
- Splashscreen nur kurz zeigen - Der Splashscreen wird dann nur bis zum Öffnen des Hauptfenster angezeigt

- Mehrere Instanzen von C-Control Pro zulassen - Wird die C-Control Pro Oberfläche mehrfach gestartet kann es zu Konflikten bezüglich der USB Schnittstelle kommen.

Zusätzlich lassen sich hier auch die Listen der "zuletzt geöffneten Projekte", sowie der "zuletzt geöffneten Dateien" löschen.

4.7.1 Kommunikation

Über eine Auswahlbox läßt sich die Verbindung zum Application Board einstellen. USB Verbindungen beginnen mit dem Kürzel "USB" und werden dann durchnummeriert: USB0, USB1 ... Serielle Schnittstellen werden genauso behandelt. Sie beginnen mit dem Kürzel "COM": COM0, COM1 .. usw.

Mit der Taste "Schnittstellensuche" werden alle Schnittstellen durchsucht bis die Kommandozeilen Schnittstelle des C-Control Pro reagiert. Damit ein Application Board erkannt wird muß der Strom eingeschaltet sein und die Firmware darf sich nicht aufgehangen haben. Am besten vor der Suchaktion einmal aus- und wieder einschalten.

Die Knöpfe "C-Control Test" und "Hardware Version" ermöglichen es sofort zu sehen ob die ausgewählte Schnittstelle auch sinnvoll mit dem C-Control Pro Modul kommunizieren kann.

4.7.2 Internet Update

Um zu überprüfen ob Verbesserungen oder Fehlerkorrekturen von Conrad veröffentlicht wurden, kann man das Internet Update aktivieren. Wird die Auswahlbox "Alle n Tage auf Update prüfen" angewählt, so wird im Intervall von n Tagen beim Start der IDE im Internet nach einem Update gesucht. Der Parameter n läßt sich im Eingabefeld rechts daneben einstellen.

Der Knopf "Jetzt auf Update prüfen" aktiviert die Suche nach Updates sofort.

» Damit das Internet Update ordnungsgemäß funktioniert, darf der MS Internet Explorer nicht im "offline" Modus stehen.

Ist z.B. wegen einer Firewall, der Zugang auf das Internet durch einen Proxy beschränkt, so können die Proxy Einstellungen wie Adresse, Benutzername und Passwort in diesem Dialog angegeben werden.

» Sind im MS Internet Explorer Proxy Daten eingetragen, so haben diese eine höhere Priorität und überschreiben die Einstellungen in diesem Dialog.

4.8 Fenster

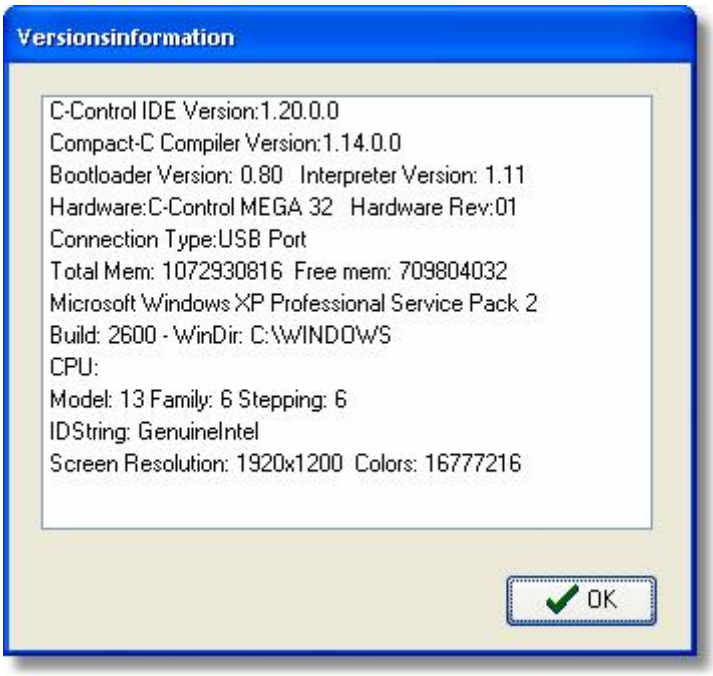
Sind im Editorbereich mehrere Fenster geöffnet so kann man über Kommandos im Fenster Menü die Editorfenster automatisch anordnen lassen.

- Überlappend - die Fenster werden übereinander angeordnet, jedes Fenster ist dabei etwas weiter nach unten rechts verschoben als das vorhergehende
- Untereinander - die Fenster werden vertikal untereinander positioniert
- Nebeneinander - ordnet die Fenster von links nach rechts nebeneinander
- Alle Minimieren - verkleinert alle Fenster auf Symbolgröße
- Schließen - schließt das aktive Fenster

4.9 Hilfe

Unter dem Menüpunkt Hilfe kann man sich mit Inhalt (Taste F1) die Hilfedatei aufrufen.

Der Menüpunkt Programmversion öffnet folgendes "Versionsinformation" Fenster und kopiert gleichzeitig den Inhalt in die Ablage.
Soll eine Support email an Conrad geschrieben werden, so sind diese Informationen wichtig. Da sie beim Aufruf von Programmversion auch gleich in der Ablage sind, kann man diese Daten bequem an das Ende einer email einfügen.



Will man nach einem bestimmten Suchbegriff in der Hilfedatei suchen, so kann die Kontexthilfe die Suche erleichtern. Steht man z.B. im Editor mit dem Cursor in dem Wort "AbsDelay" und sucht nach den richtigen Parametern so kann man einfach Kontexthilfe anwählen. Diese Funktion nimmt das Wort an dem der Cursor steht als Suchbegriff und zeigt die Ergebnisse in der Hilfedatei an.

Der Befehl Kontexthilfe steht auch bei einem Rechtsklick im Editorfenster zur Verfügung.

Kapitel 5

5 Compiler

5.1 Compact C

5.1.1 Programm

Ein Programm besteht aus einer Menge von Anweisungen (wie z.B. "a=5;") die auf verschiedene Funktionen verteilt sind. Die Startfunktion die in jedem Programm vorhanden sein muß, ist die Funktion "main()". Ein minimalistisches Programm welches eine Zahl in das Ausgabenfenster druckt:

```
void main(void)
{
      Msg_WriteInt(42);  // die Antwort auf alles
}
```

Projekte

Man kann ein Programm auf mehrere Dateien aufteilen die in einem Projekt (siehe Projektverwaltung) zusammengefasst sind. Zusätzlich zu diesen Dateien kann man Bibliotheken zu einem Projekt hinzufügen, die Funktionen bereitstellen die vom Programm genutzt werden.

5.1.2 Anweisungen

Anweisung

Eine Anweisung besteht aus mehreren reservierten Befehlswörtern, Bezeichnern und Operatoren, die mit einem Semikolon (';') am Ende abgeschlossen wird. Um verschiedene Elemente einer Anweisung zu trennen, existiert zwischen den einzelnen Anweisungselementen Zwischenraum im engl. auch "*Whitespaces*" genannt.
Unter Zwischenraum versteht man Leerzeichen, Tabulatoren und Zeilenvorschübe ("C/R und LF"). Dabei ist es egal ob ein oder mehrere "*Whitespaces*" den Zwischenraum bilden.

Einfache Anweisung:

```
a=  5;
```

 Eine Anweisung muß nicht notwendigerweise komplett in einer Zeile stehen. Da auch Zeilenvorschübe zum Zwischenraum gehören, ist es legitim eine Anweisung über mehrere Zeilen zu verteilen.

```
if(a==5)   // Anweisung über 2 Zeilen
a=a+10;
```

Anweisungsblock

Man kann mehrere Anweisungen in einem Block gruppieren. Dabei wird der Block mit einer linken geschweiften Klammer ("{") geöffnet, danach folgen die Anweisungen, und am Ende wird der Block mit einer rechten geschweiften Klammer ("}") geschlossen. Ein Block muß nicht mit einem Semikolon beendet werden. Das heißt,. das wenn ein Block das Ende einer Anweisung bildet, ist das letzte Zeichen der Anweisung die geschweifte Klammer zu.

```
if(a>5)
{
    a=a+1;      //  Anweisungsblock
    b=a+2;
}
```

Kommentare

Es existieren zwei Arten von Kommentaren, einzeilige und mehrzeilige Kommentare. Dabei wird der Text in den Kommentaren vom Compiler ignoriert.

- Einzeilige Kommentare beginnen mit "//" und hören mit dem Zeilenende auf.
- Mehrzeilige Kommentare beginnen mit "/*" und hören mit "*/" auf.

```
/* Ein
mehrzeiliger
Kommentar */

// Ein einzeiliger Kommentar
```

Bezeichner

Bezeichner sind die Namen von Funktionen oder Variablen.

- gültige Zeichen sind die Buchstaben (A-Z,a-z), die Ziffern (0-9) und der Unterstrich ('_')
- ein Bezeichner beginnt immer mit einem Buchstaben
- Groß- und Kleinschreibung werden unterschieden
- reservierte Worte sind als Bezeichner nicht erlaubt
- die Länge von Bezeichnern ist nicht begrenzt

arithmetische Ausdrücke

Ein arithmetischer Ausdruck ist eine Menge von Werten die mit Operatoren verbunden sind. Unter Werten versteht man in diesem Zusammenhang Zahlen, Variablen und Funktionen.

Ein einfaches Beispiel:

```
2 + 3
```

Hier werden die Zahlenwerte 2 und 3 mit dem Operator "+" verknüpft. Ein arithmetischer Ausdruck repräsentiert wieder einen Wert. Hier ist der Wert 5.

Weitere Beispiele:

```
a - 3

b + f(5)

2 + 3 * 6
```

Nach "Punkt vor Strich" wird hier erst 3 mal 6 gerechnet und danach 2 addiert. Dieser Vorrang von Operatoren heißt bei Operatoren Präzedenz. Eine Aufstellung der Prioritäten findet sich in der Präzedenz Tabelle.

 Auch Vergleiche sind arithmetische Ausdrücke. Die Vergleichsoperatoren liefern einen

Wahrheitswert von "1" oder "0" zurück, je nach dem ob der Vergleich korrekt war. Der Ausdruck "3 < 5" liefert den Wert "1" (wahr; true).

konstante Ausdrücke

Ein Ausdruck oder Teile eines Ausdrucks können konstant sein. Diese Teilausdrücke können schon zu Compilerlaufzeit berechnet werden.

So wird z.B.

```
12 + 123 - 15
```

vom Compiler zu

```
120
```

zusammengefaßt. Manchmal müssen Ausdrücke konstant sein damit sie gültig sind. Siehe z.B. Deklarierung von Array Variablen.

5.1.3 Datentypen

Werte haben immer einen bestimmten Datentyp. Die Integerwerte (ganzzahlige Werte) haben in CompactC einen 8 oder 16 Bit breiten Datentyp, floating point Zahlen sind immer 4 byte lang.

Datentyp	Vorzeichen	Wertebereich	Bit
char	Ja	-128 ... +127	8
unsigned char	Nein	0 ... 255 8	8
byte	Nein	0 ... 255 8	8
int	Ja	-32768 ... +32767	16
unsigned int	Nein	0 ... 65535	16
word	Nein	0 ... 65535	16
float	Ja	±1.175e-38 to ±3.402e38	32

Wie man sieht sind die Datentypen "**unsigned char**" und **byte**, sowie "**unsigned int**" und **word** identisch.

Strings

Es existiert kein expliziter "String" Datentyp. Ein String basiert auf einem character array. Man muß die Größe des arrays so wählen das alle Zeichen des Strings in das character array passen. Zusätzlich wird Platz für ein Terminierungszeichen (dezimal Null) benötigt, um das Ende der Zeichenkette anzuzeigen.

Typkonvertierung

Bei arithmetischen Ausdrücken passiert es sehr oft daß einzelne Werte nicht vom gleichen Typ sind. So sind die Datentypen im folgenden Ausdruck gemischt (a ist integer variable).

```
a + 5.5
```

In diesem Fall wird a zuerst in den Datentyp **float** konvertiert und danach 5.5 addiert. Der Datentyp des Ergebnis ist auch **float**. Es gelten bei der Typkonvertierung folgende Regeln:

- Ist bei der Verknüpfung von zwei 8 Bit oder 16 Bit Integerwerten einer der beiden Datentypen vorzeichenbehaftet ("**signed**") so ist auch das Ergebnis des Ausdrucks vorzeichenbehaftet. D.h. die Operation wird "**signed"** ausgeführt.
- Ist einer der beiden Operanden vom Typ **float,** so ist auch das Ergebnis vom Typ **float**. Sollte einer der beiden Operanden einen 8 Bit oder 16 Bit Datentyp haben, so wird er vor der Operation in einen **float** Datentyp umgewandelt.

5.1.4 Variablen

Variablen können verschiedene Werte annehmen abhängig vom Datentyp mit denen sie definiert wurden. Eine Variablendefinition sieht folgendermaßen aus:

```
Typ Variablenname;
```

Möchte man mehrere Variablen des gleichen Typs definieren, so kann man mehrere Variablennamen durch Komma getrennt angeben:

```
Typ Name1, Name2, Name3, ...;
```

Als Typ sind erlaubt: **char**, **unsigned char**, **byte**, **int**, **unsigned** int, **word** ,**float**

Beispiele:

```
int a;

int i,j;

float xyz;
```

Integer Variablen lassen sich Zahlenwerte dezimal oder als Hexzahl zuweisen. Bei einer Hexzahl werden vor die Zahl die Buchstaben "**0x**" gesetzt. Bei Variablen mit vorzeichenbehafteten Datentyp lassen sich negative Dezimalzahlen zuweisen indem ein Minuszeichen vor die Zahl geschrieben wird.

Beispiele:

```
word a;
int i,j;

a=0x3ff;
i=15;
j=-22;
```

Fließkommazahlen (Datentyp **float**) dürfen ein Dezimalkomma und einen Exponenten beinhalten:

```
float x,y;

x=5.70;
```

```
y=2.3e+2;
x=-5.33e-1;
```

sizeof Operator

Mit dem Operator **sizeof**() kann die Anzahl der Bytes bestimmt werden die eine Variable im Speicher belegt.

```
Beispiel:

int s;
float f:

s=sizeof(f);  // der Wert von s ist 4
```

» Bei Arrays wird auch nur die Bytelänge des Grunddatentyps zurückgegeben. Man muß den Wert mit der Anzahl der Elemente multiplizieren um den Speicherverbrauch des Arrays zu berechnen.

Array Variablen

Wenn man hinter den Namen bei der Variablendefinition in eckigen Klammern einen Zahlenwert schreibt so hat man ein Array definiert. Ein Array legt den Platz für die definierte Variable mehrfach im Speicher an. Bei der Beispieldefinition:

```
int x[10];
```

Wird für die Variable x der 10-fache Speicherplatz angelegt. Den ersten Speicherplatz kann man mit `x[0]` ansprechen, den zweiten mit `x[1]`, den dritten mit `x[2]`, ... bis `x[9]`. Man darf bei der Definition natürlich auch andere Indexgrößen wählen. Die Limitierung ist nur der RAM Speicherplatz des C-Control Pro.

Man kann auch mehrdimensionale Arrays deklarieren, in dem weitere eckige Klammern bei der Variablendefinition angefügt werden:

```
int x[3][4];      // Array mit 3*4 Einträgen
int y[2][2][2];  // Array mit 2*2*2 Einträgen
```

» Arrays dürfen in CompactC bis zu 16 Indizes (Dimensionen) haben. Der Maximalwert für einen Index ist 65535. Die Indizes der Arrays sind immer nullbasiert, d.h. jeder Index beginnt mit 0.

» Es findet während des Programmlaufs keine Überprüfung statt, ob die definierte Indexgrenze eines Arrays überschritten wurde. Wird der Index während der Programmabarbeitung zu groß, wird auf fremde Variablen zugegriffen und die Chance ist groß das das Programm "abstürzt".

Strings

Es existiert kein expliziter "String" Datentyp. Ein String basiert auf einem Array vom Datentyp **char**. Man muß die Größe des Arrays so wählen das alle Zeichen des Strings in das character Array passen. Zusätzlich wird Platz für ein Terminierungszeichen (dezimal Null) benötigt, um das Ende der Zeichenkette anzuzeigen.

Beispiel für eine Zeichenkette mit maximal 20 Zeichen:

```
char str1[21];
```

Als Ausnahme darf man **char** Arrays Zeichenketten zuweisen. Dabei wird die Zeichenkette zwischen Anführungszeichen gesetzt.

```
str1="Hallo Welt!";
```

Sichtbarkeit von Variablen

Werden Variablen außerhalb von Funktionen deklariert so haben sie eine globale Sichtbarkeit. Das heißt, man kann sie aus jeder Funktion ansprechen. Variablendeklarationen innerhalb von Funktionen erzeugen lokale Variablen. Lokale Variablen sind nur innerhalb der Funktion erreichbar. Ein Beispiel:

```
int a,b;

void func1(void)
{
    int a,x,y;
    // globale b ist zugreifbar
    // globale a ist nicht zugreifbar da durch lokale a verdeckt
    // lokale x,y sind zugreifbar
    // u ist nicht zugreifbar da lokal zu Funktion main
}

void main(void)
{
    int u;
    // globale a,b sind zugreifbar
    // lokale u ist zugreifbar
    // x,y nicht zugreifbar da lokal zu Funktion func1
}
```

Globale Variablen haben einen definierten Speicherbereich der während des gesamten Programmlaufs zur Verfügung steht.

Bei Programmstart werden die globalen Variablen mit null initialisiert.

Lokale Variablen werden während der Berechnung einer Funktion von ihr auf dem Stack angelegt. Das heißt, lokale Variablen existieren im Speicher nur während des Zeitraums in der die Funktion abgearbeitet wird.

Wird bei lokalen Variablen der gleiche Name gewählt wie bei einer globalen Variable, so verdeckt die lokale Variable die globale Variable. Solange sich das Programm dann in der Funktion aufhält wo die namensgleiche lokale Variable definiert wurde, ist die globale Variable nicht ansprechbar.

Static Variablen

Man kann bei lokalen Variablen die Eigenschaft **static** vor den Datentyp setzen.

```
void func1(void)
{
    static int a;
}
```

Static Variablen behalten im Gegensatz zu normalen lokalen Variablen ihren Wert auch wenn die

Funktion verlassen wird. Bei einem weiteren Aufruf der Funktion hat die statische Variable den gleichen Inhalt wie beim Verlassen der Funktion. Damit der Inhalt einer **static** Variable bei dem ersten Zugriff definiert ist, werden statische Variablen wie globale auch bei Programmstart mit null initialisiert.

5.1.5 Operatoren

Prioritäten von Operatoren

Operatoren teilen arithmetische Ausdrücke in Teilausdrücke. Die Operatoren werden dann in der Reihenfolge ihrer Priorität (Präzedenz) ausgewertet. Ausdrücke mit Operatoren von gleicher Präzedenz werden von links nach rechts berechnet. Beispiel:

```
i= 2+3*4-5;  // Ergebnis 9 => erst 3*4, dann +2 danach -5
```

Mann kann die Reihenfolge der Abarbeitung beinflußen in dem man Klammern setzt. Klammern haben die größte Priorität. Möchte man das letzte Beispiel strikt von links nach rechts auswerten:

```
i= (2+3)*4-5;  // Ergebnis 15 => erst 2+3, dann *4, danach -5
```

Eine Aufstellung der Prioritäten findet sich in der Präzedenz Tabelle.

5.1.5.1 Arithmetische Operatoren

Alle arithmetischen Operatoren mit Ausnahme von Modulo sind für Integer und Fließkomma Datentypen definiert. Nur Modulo ist auf einen Integertyp beschränkt.

Es ist zu beachten das in einem Ausdruck die Zahl 7 einen Integer Datentyp zugewiesen bekommt. Möchte man explizit eine Zahl vom Datentyp **float** erzeugen, so ist ein Dezimalkomma einzufügen: 7.0

Operator	Erklärung	Beispiel	Ergebnis
+	**Addition**	2+1 3.2 + 4	3 7.2
-	**Subtraktion**	2 - 3 22 - 1.1e1	-1 11
*	**Multiplikation**	5 * 4	20
/	**Division**	7 / 2 7.0 / 2	3 3.5
%	**Modulo**	15 % 4 17 % 2	3 1
-	**negatives Vorzeichen**	-(2+2)	-4

5.1.5.2 Vergleichsoperatoren

Vergleichsoperatoren sind für **float** und Integer Datentypen erlaubt.

Operator	Erklärung	Beispiel	Ergebnis
<	**kleiner**	1 < 2 2 < 1 2 < 2	1 0 0
>	**größer**	-3 > 2 3 > 2	0 1
<=	**kleiner gleich**	2 <= 2 3 <= 2	1 0
>=	**größer gleich**	2 >= 3 3 >= 2	0 1
==	**gleich**	5 == 5 1 == 2	1 0
!=	**ungleich**	2 != 2 2 != 5	0 1

5.1.5.3 Logische Operatoren

Logische Operatoren sind nur für Integer Datentypen erlaubt. Jeder Wert ungleich **null** gilt als logisch 1. Die **null** gilt als logisch 0.

Operator	Erklärung	Beispiel	Ergebnis
&&	**logisches Und**	1 && 1 5 && 0	1 0
||	**logisches Oder**	0 || 0 1 || 0	0 1
!	**logisches Nicht**	!2 !0	0 1

5.1.5.4 Bitschiebe Operatoren

Bitschiebe Operatoren sind nur für Integer Datentypen erlaubt. Bei einer Bit-Shift Operation wird immer eine 0 an einem Ende hineingeschoben.

Operator	Erklärung	Beispiel	Ergebnis
<<	**um ein Bit nach links schieben**	1 << 2 3 << 3	4 24
>>	**um ein Bit nach rechts schieben**	0xff >> 6 16 >> 2	3 4

5.1.5.5 In- Dekrement Operatoren

Inkrement und Dekrement Operatoren sind nur für Variablen mit Integer Datentypen erlaubt.

Operator	Erklärung	Beispiel	Ergebnis
variable++	Wert der Variablen, danach Variable um eins erhöht (Postinkrement)	a++	a
variable--	Wert der Variablen, danach Variable um eins erniedrigt (Postdekrement)	a--	a
++variable	Wert der Variablen um eins erhöht (Preinkrement)	a++	a+1
variable--	Wert der Variablen um eins erniedrigt (Predekrement)	a--	a-1

5.1.5.6 Bitoperatoren

Bitoperatoren sind nur für Integer Datentypen erlaubt.

Operator	Erklärung	Beispiel	Ergebnis
&	**Und**	0x0f & 3 0xf0 & 0x0f	3 0
|	**Oder**	1 | 3 0xf0 | 0x0f	3 0xff
^	**exclusives Oder**	0xff ^ 0x0f 0xf0 ^ 0x0f	0xf0 0xff
~	**Bitinvertierung**	~0xff ~0xf0	0 0x0f

5.1.6 Kontrollstrukturen

5.1.6.1 if .. else

Eine **if** Anweisung hat folgende Syntax:

```
if( Ausdruck ) Anweisung1;
else Anweisung2;
```

Hinter der **if** Anweisung folgt in Klammern ein arithmetischer Ausdruck. Wird dieser *Ausdruck* zu ungleich 0 ausgewertet, dann wird die Anweisung1 ausgeführt. Man kann mit Hilfe des **else** Befehlswortes eine alternative Anweisung2 definieren, die dann ausgeführt wird, wenn der *Ausdruck* zu 0 berechnet wurde. Das Hinzufügen einer **else** Anweisung ist optional und muß nicht geschehen.

Beispiele:

```
if(a==2) b++;

if(x==y) a=a+2;
else a=a-2;
```

Statt einer einzelnen Anweisung kann auch ein Anweisungsblock definiert werden.

Beispiele:

```
if(x<y)
{
    c++;
    if(c==10) c=0;
}
else d--;

if(x>y)
{
    a=b*5;
    b--;
}
else
{
    a=b*4;
    y++;
}
```

5.1.6.2 while

Mit einer **while** Anweisung lassen sich abhängig von einer Bedingung Anweisungen in einer Schleife wiederholen:

```
while( Ausdruck ) Anweisung;
```

Zuerst wird der *Ausdruck* ausgewertet. Ist das Ergebnis ungleich 0 dann kommt es zur Ausführung der Anweisung. Danach erfolgt wieder die Berechnung des *Ausdrucks* und der ganze Vorgang wiederholt sich solange bis der *Ausdruck* den Wert 0 annimmt. Statt einer einzelnen Anweisung kann auch ein Anweisungsblock definiert werden.

Beispiele:

```
while(a<10) a=a+2;

while(a)
{
    a=a*2;
    x=a;
}
```

break Anweisung

Wird innerhalb der Schleife ein **break** ausgeführt, so wird die Schleife verlassen und die Programmausführung startet mit der nächsten Anweisung hinter der **while** Schleife.

continue Anweisung

Bei der Ausführung von **continue** innerhalb einer Schleife kommt es sofort zur erneuten Berechnung des *Ausdrucks*. In Abhängigkeit vom Ergebnis wird bei ungleich 0 die Schleife wiederholt. Ein Ergebnis von 0 bricht die Schleife ab.

Beispiel:

```
while(1)  // Endlosschleife
{
    a++;
    if(a>10) break; // brich Schleife ab
}
```

5.1.6.3 do .. while

Mit einem **do** .. **while** Konstrukt lassen sich abhängig von einer Bedingung Anweisungen in einer Schleife wiederholen:

```
do

Anweisung

while( Ausdruck );
```

Die Anweisung oder der Anweisungsblock wird ausgeführt. Am Ende wird der *Ausdruck* ausgewertet. Ist das Ergebnis ungleich 0 kommt es zur wiederholten Ausführung der Anweisung. Der ganze Vorgang wiederholt sich solange bis der *Ausdruck* den Wert 0 annimmt.

Beispiele:

```
do
a=a+2;
while(a<10);

do
{
    a=a*2;
    x=a;
} while(a);
```

Der wesentliche Unterschied der **do** .. **while** Schleife zur normalen **while** Schleife ist der Umstand, das in einer **do** .. **while** Schleife die Anweisung mindestens einmal ausgeführt wird.

break Anweisung

Eine **break** Anweisung verläßt die Schleife, und die Programmausführung startet mit der nächsten Anweisung hinter der **do** .. **while** Schleife.

continue Anweisung

Bei Ausführung von **continue** innerhalb einre Schleife, kommt es sofort zur erneuten Berechnung des *Ausdrucks*. In Abhängigkeit vom Ergebnis wird bei ungleich 0 die Schleife wiederholt. Ein Ergebnis von 0 bricht die Schleife ab.

Beispiel:

```
do
{
    a++;
    if(a>10) break; // bricht Schleife ab
} while(1);  // Endlosschleife
```

5.1.6.4 for

Eine for Schleife wird normalerweise benutzt um eine bestimmte Anzahl von Schleifendurchläufen zu programmieren.

```
for(Anweisung1; Ausdruck; Anweisung2) Anweisung3;
```

Als erstes wird Anweisung1 ausgeführt, die normalerweise eine Initialisierung beinhaltet. Danach erfolgt die Auswertung des *Ausdrucks*. Ist der *Ausdruck* ungleich 0 wird Anweisung2 und Anweisung3 ausgeführt, und die Schleife wiederholt sich. Hat der *Ausdruck* einen Wert von 0 kommt es zum Schleifenabbruch. Wie bei anderen Schleifentypen kann bei Anweisung3 statt einer einzelnen Anweisung ein Anweisungsblock benutzt werden.

```
for(i=0;i<10;i++)
{
    if(i>a) a=i;
    a--;
}
```

» Es gilt zu beachten das die Variable i innerhalb der Schleife die Werte von 0 bis 9 durchläuft, und nicht 1 bis 10!

Möchte man eine Schleife programmieren die eine andere Schrittweite hat, so ist Anweisung2 entsprechend zu modifizieren:

```
for(i=0;i<100;i=i+3)  // die Variable i inkrementiert sich nun in 3er Schritten
{
    a=5*i;
}
```

break Anweisung

Eine **break** Anweisung verläßt die Schleife, und die Programmausführung startet mit der nächsten Anweisung hinter der **for** Schleife.

continue Anweisung

continue veranlaßt die sofortige Neuberechnung des *Ausdrucks*. In Abhängigkeit vom Ergebnis wird bei ungleich 0 Anweisung2 ausgeführt und die Schleife wiederholt sich. Ein Ergebnis von 0 bricht die Schleife ab.

Beispiel:

```
for(i=0;i<10;i++)
{
    if(i==5) continue;
}
```

5.1.6.5 switch

Sollen in Abhängigkeit vom Wert eines Ausdrucks verschiedene Befehle ausgeführt werden, so ist eine **switch** Anweisung sehr elegant:

```
switch( Ausdruck )
{
    case konstante_1:
        Anweisung_1;
    break;

    case konstante_2:
        Anweisung_2;
    break;
    .
    .
   case konstante_n:
        Anweisung_n;
    break;
    default:   // default ist optional
        Anweisung_0;
};
```

Der Wert von *Ausdruck* wird berechnet. Danach springt die Programmausführung zur Konstante die dem Wert des *Ausdrucks* entspricht und führt das Programm dort fort. Entspricht keine Konstante dem Ausdruckswert so wird das **switch** Konstrukt verlassen.

Ist in einer **switch** Anweisung ein **default** definiert, so werden die Anweisungen hinter **default** ausgeführt wenn keine Konstante gefunden wurde die dem Wert des *Ausdrucks* entspricht.

Beispiel:

```
switch(a+2)
{
    case 1:
        b=b*2;
    break;

    case 5*5:
        b=b+2;
    break;

    case 100&0xf:
        b=b/c;
    break;
```

```
    default:
        b=b+2;
}
```

break Anweisung

Ein **break** verläßt die switch Anweisung. Läßt man vor **case** das **break** weg, so werden die Anweisungen auch ausgeführt wenn zum vorherigen **case** gesprungen wird:

```
switch(a)
{
    case 1:
        a++;

    case 2:
        a++;  // wird auch bei einem Wert von a==1 ausgeführt

    case 3:
        a++; // wird auch bei einem Wert von a==1 oder a==2 ausgeführt
}
```

In diesem Beispiel werden alle drei "a++" Anweisungen ausgeführt wenn a gleich 1 ist.

5.1.6.6 bedingte Bewertung

Mit einer bedingten Bewertung lassen sich Ausdrücke erzeugen die bedingt berechnet werden. Die Form ist:

```
( Ausdruck1 )  ? Ausdruck2 : Ausdruck3

Das Ergebnis dieses Ausdrucks ist Ausdruck2 wenn Ausdruck1 zu ungleich 0
berechnet wurde, sonst ist das Ergebnis Ausdruck3.

Beispiele:

a = (i>5) ? i : 0;

a= (i>b*2) ? i-5 : b+1;

while(i> ((x>y) ? x : y) ) i++;
```

5.1.7 Funktionen

Um größere Programme zu strukturieren teil man sie in mehrere Unterfunktionen auf. Dies erhöht nicht nur die Lesbarkeit, sondern erlaubt es Programmanweisungen die mehrfach vorkommen in Funktionen zusammenzufassen. Ein Programm besteht immer aus der Funktion "main" die als allererstes gestartet wird. Danach kann man von main aus andere Funktionen aufrufen. Ein einfaches Beispiel:

```
void func1(void)
{
    // Anweisungen in Funktion func1
    .
    .
```

```
}

void main(void)
{
    // die Funktion func1 wird zweimal aufgerufen
    func1();
    func1();
}
```

Parameterübergabe

Damit Funktionen flexibel nutzbar sind, kann man sie parametrisieren. Hierfür werden in der Klammer nach dem Funktionsnamen die Parameter für die Funktion durch Komma getrennt übergeben. Man gibt ähnlich wie in der Variablendeklaration erst den Datentyp und danach den Parameternamen an. Will man keinen Parameter übergeben schreibt man **void** in die runden Klammern. Ein Beispiel:

```
void func1(word param1, float param2)
{
    Msg_WriteHex(param1);  // den ersten Parameter ausgeben
    Msg_WriteFloat(param2);  // den zweiten Parameter ausgeben
}
```

» Wie lokale Variablen sind übergebene Parameter nur in der Funktion selber sichtbar.

Um die Funktion func1 mit den Parametern aufzurufen schreibt man beim Aufruf die Parameter in der gleichen Reihenfolge wie sie bei func1 definiert wurden. Bekommt die Funktion keine Parameter läßt man die Klammer leer.

```
void main(void)
{
    word a;
    float f;

    func1(128,12.0);  // man kann numerische Konstanten übergeben ...
    a=100;
    f=12.0;
    func1(a+28,f); // oder aber auch Variablen und sogar numerische Ausdrücke
}
```

» Man muß bei dem Aufruf einer Funktion immer alle Parameter angeben. Folgende Aufrufe wären unzulässig:

```
func1();        // func1 bekommt 2 Parameter!
func1(128);     // func1 bekommt 2 Parameter!
```

Rückgabeparameter

Es ist nicht nur möglich Parameter zu übergeben, eine Funktion kann auch einen Rückgabewert haben. Den Datentyp dieses Wertes gibt man bei der Funktionsdefinition vor dem Namen der Funktion an. Möchte man keinen Wert zurückgeben, benutzt man **void** als Datentyp.

```
int func1(int a)
{
    return a-10;
}
```

Der Rückgabewert wird innerhalb der Funktion mit der Anweisung "**return** *Ausdruck*" angegeben. Hat man eine Funktion vom Typ **void** so kann man die **return** Anweisung auch ohne Parameter anwenden um die Funktion zu verlassen.

Referenzen

Da es nicht möglich ist Arrays als Parameter zu übergeben, kann man auf Arrays über Referenzen zugreifen. Dafür schreibt man in der Parameterdeklaration einer Funktion ein eckiges Paar Klammern hinter den Parameternamen:

```
int StringLength(char str[])
{
    int i;

    i=0;
    while(str[i]) i++;  // wiederhole solange Zeichen nicht null
    return(i);
}

void main(void)
{
    int len;
    char text[15];

    text="hallo welt";
    len=StringLength(text);
}
```

In main wird die Referenz von Text als Parameter an die Funktion StringLength übergeben. Ändert man in einer Funktion einen normalen Parameter, so ist die Änderung außerhalb dieser Funktion nicht sichtbar. Bei Referenzen ist dies anders. Über den Parameter str kann man in StringLength den Inhalt von text ändern, da str nur eine Referenz (ein Zeiger) auf die Array Variable text ist.

5.1.8 Tabellen

5.1.8.1 reservierte Worte

Folgende Worte sind **reserviert** und können nicht als Namen für Bezeichner benutzt werden:

break	**byte**	**case**	**char**	**continue**
default	**do**	**else**	**false**	**float**
for	**goto**	**if**	**int**	**return**
signed	**static**	**switch**	**true**	**unsigned**
void	**while**	**word**		

5.1.8.2 Operator Präzedenz

Rang	Operator
13	( )
12	++ -- ! ~ - (negatives Vorzeichen)
11	* / %
10	+ -
9	<< >>
8	< <= > >=
7	== !=
6	&
5	^
4	|
3	&&
2	||
1	? :

5.1.8.3 Operatoren

	Arithmetische Operatoren
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo
-	negatives Vorzeichen

	Vegleichsoperatoren
<	kleiner
>	größer
<=	kleiner gleich
>=	größer gleich
==	gleich
!=	ungleich

	Bitschiebeoperatoren
<<	um ein Bit nach links schieben
>>	um ein Bit nach rechts schieben

	Inkrement/Dekrement Operatoren
++	Post/Pre Inkrement
--	Post/Pre Dekrement

	Logische Operatoren
&&	logisches Und
||	logisches Oder
!	logisches Nicht

	Bitoperatoren
&	Und
|	Oder
^	exclusives Oder
~	Bitinvertierung

5.2 Preprozessor

 Der Gnu Generic Preprocessor der hier zum Einsatz kommt hat noch weitere Funktionen die unter http://nothingisreal.com/gpp/gpp.html dokumentiert sind. Allerdings sind nur die hier beschriebenen Funktionen auch im Zusammenspiel mit dem C-Control Pro Compiler auch ausführlich getestet. Ein benutzen der hier undokumentierten Funktionen geschieht auf eigene Gefahr!

Im C-Control Entwicklungssystem ist ein vollständiger C-Preprozessor enthalten. Der Preprozessor bearbeitet den Quelltext bevor der Compiler gestartet wird. Folgende Befehle werden unterstützt:

Definitionen

Man definiert mit dem Befehl "#define" Textkonstanten.

```
#define symbol textkonstante
```

Da der Preprozessor vor dem Compiler läuft, wird bei jedem Auftauchen von symbol im Quelltext symbol durch textkonstante ersetzt.

Ein Beispiel

#define PI 3.141

Bedingte Kompilierung

```
#ifdef symbol
...
#else  // optional
...
#endif
```

Man kann kontrollieren welche Teil eines Quelltextes wirklich kompiliert werden. Nach einer #ifdef symbol Anweisung wird der folgende Text nur kompiliert wenn das symbol auch durch #define symbol definiert wurde.

Ist eine optionale #else Anweisung angegeben, so wird der Quelltext nach #else bearbeitet wenn das symbol nicht definiert ist.

Einfügen von Text

```
#include pfad\dateiname
```

Mit dieser Anweisung läßt sich eine Textdatei in den Quellcode einfügen.

» Aufgrund einer Limitierung des Preprozessors darf der Pfad in einer #include Anweisung keine Leerzeichen enthalten!

5.3 Bibliotheken

5.3.1 Interne Funktionen

Damit der Compiler die im Interpreter vorhandenen internen Funktionen erkennen kann, müssen diese Funktionen in der Bibliothek "IntFunc_Lib.cc" definiert sein. Ist diese Bibliothek nicht eingebunden so können keine Ausgaben vom Programm getätigt werden. Ein typischer Eintrag in "IntFunc_Lib.cc" sieht z.B. so aus:

```
void Msg_WriteHex$opc(0x23)(word val);
```

Diese Definition besagt, das die Funktion("Msg_WriteHex") im Interpreter mit einem Sprungvektor von 0x23
aufgerufen wird, und als Parameter ein word auf dem Stack zu übergeben ist.

» Änderungen in der Bibliothek "IntFunc_Lib.cc" können dazu führen, daß die dort deklarierten Funktionen nicht mehr korrekt ausführbar sind!

5.3.2 AbsDelay

Allgemeine Funktionen

Syntax

```
void AbsDelay(word ms);
```

Beschreibung

Die Funktion Absdelay() wartet eine bestimmte Anzahl von Millisekunden.

» Die Funktion arbeitet zwar sehr genau, aber unterbricht nicht nur die Abarbeitung des aktuellen Threads sondern läßt den Bytecode Interpreter insgesamt warten. Interrupts werden zwar registriert, aber die Interruptroutinen in dieser Zeit nicht abgearbeitet, da auch dafür der Bytecode Interpreter nötig ist.

Soll nur der aktuelle Thread warten, ist die Funktion Thread_Delay() zu benutzen.

Parameter

ms Wartezeit in ms

5.3.3 Analog-Comparator

5.3.3.1 AComp

AComp Funktionen **Beispiel**

Syntax

```
void AComp(byte mode);
```

Beschreibung

Der Analog-Comparator ermöglicht zwei analoge Signale zu vergleichen. Das Ergebnis dieses Vergleichs wird entweder als „0" oder „1" zurückgegeben (Ausgang des Komparators). Der negative Eingang ist AIN1 (PortB.3). Der positive Eingang kann entweder AIN0 (PortB.2) sein, oder eine interne Referenzspannung von 1,22V.

Parameter

mode Arbeitsmodus

Moduswerte:

0x00	externe Eingänge (+)AIN0 und (-)AIN1 werden verwendet
0x40	externer Eingang (-)AIN1und interne Referenzspannung werden verwendet
0x80	Analog-Comparator wird abgeschaltet

5.3.3.2 AComp Beispiel

Beispiel: Verwendung des Analog-Comparators

```
// AComp: Analog Comparator
// Eingang (+) PB2 bzw. band gap reference 1,22V
// Eingang (-) PB3
// Die Funktion AComp gibt den Wert des Komparators zurück.
// Der Aufruf kann mit dem Parameter 0 (beide Eingänge werden verwendet) oder
// oder 0x40 (interne Referenzspannung am (+) Eingang, externer Eingang PB3.

void main(void)
{
    // Der Komparator wird alle 500ms gelesen und ausgegeben
    while (1)
    {
        if (AComp(0x40)==1)       // Eingang (+) interne band gap reference
1,22V
        {
            Msg_WriteChar('1');
        }
        else
        {
           Msg_WriteChar('0');
        }
        AbsDelay(500);
    }
}
```

5.3.4 Analog-Digital-Wandler

Der Mikrocontroller verfügt über einen Analog-Digital-Wandler mit einer Auflösung von 10 Bit. Das heißt, gemessene Spannungen können als ganze Zahlen von 0 bis 1023 dargestellt werden. Die Referenzspannung für die untere Grenze ist der GND-Pegel, also 0V. Die Referenzspannung für die obere Grenze kann ausgewählt werden.

- externe Referenzspannung
- AVCC mit Kondensator an AREF
- Interne Spannungsreferenz 2,56V mit Kondensator an AREF

Analogeingänge ADC0 ... ADC7, ADC_BG, ADC_GND

Als Eingänge für den ADC stehen die Eingänge ADC0 ... ADC7 , eine interne Bandgap (1,22V) oder GND (0V) zur Verfügung. ADC_BG und ADC_GND können zur Überprüfung des ADC verwendet werden.

Ist x ein digitaler Meßwert, dann errechnet sich der entsprechende Spannungswert u wie folgt:

*u = x * Referenzspannung / 1024*

Beträgt die externe Referenzspannung 4,096V z.B. durch ein Referenzspannungs-IC erzeugt, dann entspricht eine Differenz von einem Bit des digitalisierten Meßwertes einer Spannungsdifferenz von 4mV oder :

*u = x * 0,004V*

Differenzeingänge

ADC22x10	Differenzeingänge ADC2, ADC2, Verstärkung 10	; Offsetmessung
ADC23x10	Differenzeingänge ADC2, ADC3, Verstärkung 10	
ADC22x200	Differenzeingänge ADC2, ADC2, Verstärkung 200	; Offsetmessung
ADC23x200	Differenzeingänge ADC2, ADC3, Verstärkung 200	
ADC20x1	Differenzeingänge ADC2, ADC0, Verstärkung 1	
ADC21x1	Differenzeingänge ADC2, ADC1, Verstärkung 1	
ADC22x1	Differenzeingänge ADC2, ADC2, Verstärkung 1	; Offsetmessung
ADC23x1	Differenzeingänge ADC2, ADC3, Verstärkung 1	
ADC24x1	Differenzeingänge ADC2, ADC4, Verstärkung 1	
ADC25x1	Differenzeingänge ADC2, ADC5, Verstärkung 1	

ADC2 ist der negative Eingang.

Der ADC kann auch Differenzmessungen durchführen. Das Ergebnis kann positiv oder negativ sein. Die Auflösung beträgt im Differenzbetrieb +/- 9 Bit und wird als two's complement dargestellt. Im Differenzbetrieb steht ein Verstärker zur Verfügung mit den Verstärkungen V : x1, x10, x200. Ist x ein digitaler Meßwert, dann errechnet sich der entsprechende Spannungswert u wie folgt:

*u = x * Referenzspannung / 512 / V*

5.3.4.1 ADC_Disable

ADC Funktionen

Syntax

```
void ADC_Disable(void);
```

Beschreibung

Die Funktion ADC_Disable schaltet den A/D-Wandler ab um den Stromverbrauch zu reduzieren.

Parameter

Keine

5.3.4.2 ADC_Read

ADC Funktionen

Syntax

```
word ADC_Read(void);
```

Beschreibung

Die Funktion ADC_Read liefert den digitalisierten Meßwert von einem der 8 ADC-Ports. Die Nummer des Ports (0..7) wurde beim Aufruf von ADC_Set() als Parameter übergeben. Das Ergebnis ist im Bereich von

0 bis 1023 - entsprechend der 10bit-Auflösung des A/D-Wandlers. Es können die Analogeingänge ADC0 bis ADC7 gegen GND gemessen werden oder Differenzmessungen mit den Verstärkungsfaktoren 1/10/200 durchgeführt werden.

Rückgabewert

gemessener Wert des ADC-Ports

5.3.4.3 ADC_ReadInt

ADC Funktionen

Syntax

```
word ADC_ReadInt(void);
```

Beschreibung

Diese Funktion wird verwendet um nach einem ADC-Interrupt den Meßwert zu lesen. Der ADC-Interrupt wird ausgelöst, wenn die AD_Wandlung abgeschlossen ist und somit ein neuer Messwert zur Verfügung steht. Siehe auch ADC_SetInt und ADC_StartInt. Die Funktion ADC_Read liefert den digitalisierten Meßwert von einem der 8 ADC-Ports. Die Nummer des Ports (0..7) wurde beim Aufruf von ADC_SetInt als Parameter übergeben. Das Ergebnis ist im Bereich von 0 bis 1023 - entsprechend der 10bit-Auflösung des A/D-Wandlers. Es können die Analogeingänge ADC0 bis ADC7 gegen GND gemessen werden oder Differenzmessungen mit den Verstärkungsfaktoren 1/10/200 durchgeführt werden.

Rückgabewert

gemessener Wert des ADC-Ports

5.3.4.4 ADC_Set

ADC Funktionen

Syntax

```
word ADC_Set(byte v_ref,byte channel);
```

Beschreibung

Die Funktion ADC_Set initialisiert den Analog-Digital_Wandler. Die Referenzspannung und der Messkanal werden ausgewählt und der A/D Wandler für die Messungen vorbereitet. Der Meßwert wird danach mit ADC_Read() ausgelesen.

Parameter

```
channel  Portnummer (0..7) des ADC
v_ref    Referenzspannung (siehe Tabelle)
```

Name	Wert	Beschreibung
ADC_VREF_BG	0xC0	2,56V interne Referenzspannung
ADC_VREF_VCC	0x40	Versorgungsspannung (5V)
ADC_VREF_EXT	0x00	externe Referenzspannung an PAD3

5.3.4.5 ADC_SetInt

ADC Funktionen

Syntax

```
word ADC_SetInt(byte v_ref,byte channel);
```

Beschreibung

Die Funktion ADC_SetInt initialisiert den Analog-Digital_Wandler für den Interruptbetrieb. Die Referenzspannung und der Messkanal werden ausgewählt und der A/D Wandler für die Messungen vorbereitet. Die Interrupt-Service-Routine für den ADC muß definiert sein. Nach erfolgtem Interrupt kann der Meßwert mit ADC_ReadInt() ausgelesen werden .

Parameter

```
channel  Portnummer (0..7) des ADC
v_ref    Referenzspannung (siehe Tabelle)
```

Name	Wert	Beschreibung
ADC_VREF_BG	0xC0	2,56V interne Referenzspannung
ADC_VREF_VCC	0x40	Versorgungsspannung (5V)
ADC_VREF_EXT	0x00	externe Referenzspannung an PAD3

5.3.4.6 ADC_StartInt

ADC Funktionen

Syntax

```
void ADC_StartInt(void);
```

Beschreibung

Die Messung wird gestartet, wenn vorher der A/D Wandler mit Hilfe von ADC_SetInt() auf Interruptbetrieb

initialisert wurde. Liegt das Messergebnis bereit, wird ein ADC_Interrupt ausgelöst.

Parameter

```
Keine
```

5.3.5 DCF 77

Alle DCF-Routinen sind in der Bibliothek "LCD_Lib.cc" realisiert. Für den Gebrauch dieser Funktionen ist die Bibliothek "DCF_Lib.cc" in das Projekt miteinzubinden.

RTC mit DCF77 Zeitsynchronisation

Das DCF77 Zeitsignal

Die logischen Informationen (die Zeitinformationen) werden zusätzlich zur Normalfrequenz (der Trägerfrequenz des Senders, also 77,5 kHz) übertragen. Das geschieht durch negative Modulation des Signals (Absenken der Trägeramplitude auf 25%). Der Beginn der Absenkung liegt jeweils auf dem Beginn der Sekunden 0...58 innerhalb einer Minute. In der 59. Sekunde erfolgt keine Absenkung, wodurch die nachfolgende Sekundenmarke den Beginn einer Minute kennzeichnet, und der Empfänger synchronisiert werden kann. Der logische Wert der Zeichen ergibt sich aus der Zeichendauer: 100 ms sind die "0", 200 ms sind die "1". Damit stehen innerhalb einer Minute 59 Bit für Informationen zur Verfügung. Davon werden die Sekundenmarken 1 bis 14 für Betriebsinformationen verwendet, die nicht für DCF77-Nutzer bestimmt sind. Die Sekundenmarken 15 bis 19 kennzeichnen die Sendeantenne, die Zeitzone und kündigen Zeitumstellungen an:

Von der 20. bis zur 58. Sekunde wird die Zeitinformation für die jeweils nachfolgende Minute seriell in Form von BCD-Zahlen übertragen, wobei jeweils mit dem niederwertigsten Bit begonnen wird:

Bits	**Bedeutung**
20	Startbit (ist immer "1")
21 - 27	Minute
28	Parität Minute
29 - 34	Stunde
35	Parität Stunde
36 - 41	Monatstag
42 - 44	Wochentag
45 - 49	Monat
50 - 57	Jahr
58	Parität Datum

Das bedeutet, daß der Empfang mindestens eine volle Minute laufen muss, bevor die Zeitinformation zur Verfügung stehen kann. Die innerhalb dieser Minute dekodierte Information ist lediglich durch drei Paritätsbits gesichert, somit führen bereits zwei fehlerhaft empfangene Bits zu einem auf diese Weise nicht zu erkennenden Übertragungsfehler. Bei höheren Anforderungen

können zusätzliche Prüfmechanismen verwendet werden, z.B. Plausibilitätsprüfung (ist die empfangene Zeit innerhalb der zulässigen Grenzen) oder mehrmaliges lesen der DCF77-Zeitinformation und Vergleich der Daten. Eine andere Möglichkeit wäre die DCF-Zeit mit der aktuellen Zeit der RTC vergleichen und nur eine bestimmte Abweichung zulassen. Dieses Verfahren geht nicht nach dem Programmstart, da die RTC erst gesetzt werden muß.

Beschreibung des Beispielprogramm "DCF_RTC.cc"

Das Programm DCF_RTC.cc ist eine Uhr, die über DCF77 synchronisiert wird. Die Uhrzeit und das Datum wird auf einem LCD-Display angezeigt. Die Synchronisation erfolgt nach dem Programmstart und dann täglich zu einer im Programm festgelegten Zeit (Update_Stunden, Update_Minuten). Es werden zwei Libraries verwendet: DCF_Lib.cc und LCD_Lib.cc.
Für den Funkempfang des Zeitsignals ist ein DCF77-Empfänger erforderlich. Der Ausgang des DCF-Empfängers wird an den PortD.7 angeschlossen. Zuerst muß der Anfang einer Zeitinformation gefunden werden. Es wird auf die Pulslücke (59.Bit) synchronisiert. Danach werden die Bits im Sekundentakt aufgenommen. Es erfolgt eine Parity-Prüfung nach der Minuten und Stunden Information und ebenfalls am Ende der Übertragung. Das Ergebnis der Parity-Prüfung wird im DCF_ARRAY[6] gespeichert. Zur Übergabe der Zeitinformation wird das DCF_ARRAY[0..6] verwendet. Nach dem Empfang einer gültigen Zeitinformation wir die RTC mit der neuen Zeit gesetzt und läuft dann selbständig weiter. Die RTC als auch die DCF77-Dekodierung ist über einen 10ms Interrupt gesteuert. Diese Zeitbasis ist von der Quarzfrequenz des Controllers abgeleitet. DCF_Mode steuert den Ablauf für die DCF77-Zeitaufnahme.

Tabelle DCF-Modi

DCF_Mode	Beschreibung
0	kein DCF77-Betrieb
1	Puls suchen
2	Synchronisation auf Frameanfang
3	Daten dekodieren und speichern, Paritätsprüfung

RTC (Real Time Clock)

Die RTC wird mit einem 10ms Interrupt gesteuert und läuft im Hintergrund unabhängig vom Anwenderprogramm. Alle Sekunde wird die Anzeige auf dem LCD-Display ausgegeben. Das Anzeigeformat ist 1. Zeile: Stunde : Minute : Sekunde
2. Zeile: Tag . Monat . Jahr

Die LED1 blinkt einmal pro Sekunde.
Nach dem Programmstart beginnt die RTC mit der festgelegten Uhrzeit. Das Datum ist auf Null gesetzt und zeigt an. Das noch kein DCF-Zeitabgleich erfolgt ist. Nach dem Empfang der DCF-Zeit wird die RTC mit den aktuellen Daten aktualisiert. Die RTC ist nicht batteriegepuffert, d.h. die Uhrzeit läuft ohne Spannungsversorgung des Controllers nicht weiter.

5.3.5.1 DCF_FRAME

DCF Funktionen

Syntax

```
void DCF_FRAME(void);
```

Beschreibung

DCF_Mode auf 3 schalten ("Daten dekodieren und speichern, Paritätsprüfung").

Parameter

Keine

5.3.5.2 DCF_INIT

DCF Funktionen

Syntax

```
void DCF_INIT(void);
```

Beschreibung

DCF_INIT bereitet den DCF-Betrieb vor. Es wird der Eingang für das DCF-Signal eingestellt. DCF_Mode =0.

Parameter

Keine

5.3.5.3 DCF_PULS

DCF Funktionen

Syntax

```
void DCF_PULS(void);
```

Beschreibung

DCF_Mode auf 1 schalten ("Puls suchen").

Parameter

Keine

5.3.5.4 DCF_START

DCF Funktionen

Syntax

```
void DCF_START(void);
```

Beschreibung

DCF_START initialisiert alle verwendeten Variablen und setzt DCF_Mode auf 1. Die DCF-Zeiterfassung läuft jetzt automatisch ab.

Parameter

Keine

5.3.5.5 DCF_SYNC

DCF Funktionen

Syntax

```
void DCF_SYNC(void);
```

Beschreibung

DCF_Mode auf 2 schalten ("Synchronisation auf Frameanfang").

Parameter

Keine

5.3.6 Debug

Die Debug Message Funktionen erlauben es formatierten Text auf das Ausgabefenster der IDE zu senden. Diese Funktionen sind interruptgetrieben mit einem Puffer von bis zu 128 Byte. D.h. 128 Byte können über die Debug Schnittstelle abgesetzt werden ohne das der Mega 32 auf die Vollendung der Ausgabe warten muß. Die Übertragung der einzelnen Zeichen geschieht im Hintergrund. Wird versucht mehr als 128 zu senden, dann muß die Mega Risc CPU warten bis alle Zeichen die nicht mehr in den Puffer hineinpassen übertragen wurden.

5.3.6.1 Msg_WriteChar

Debug Message Funktionen

Syntax

```
void Msg_WriteChar(char c);
```

Beschreibung

Ein Zeichen wird zum Ausgabenfenster geschickt. Ein C/R (Carriage Return - Wert:13) löst einen Sprung zum Anfang der nächsten Zeile aus.

Parameter

c das auszugebende Zeichen

5.3.6.2 Msg_WriteFloat

Debug Message Funktionen

Syntax

```
void Msg_WriteFloat(float val);
```

Beschreibung

Die übergebene floating point Zahl wird im Ausgabenfenster mit Vorzeichen dargestellt.

Parameter

val float Wert

5.3.6.3 Msg_WriteHex

Debug Message Funktionen

Syntax

```
void Msg_WriteHex(word val);
```

Beschreibung

Der übergebene 16bit Wert wird im Ausgabenfenster dargestellt. Die Ausgabe wird als Hexzahl mit 4 Stellen formatiert. Ist die Zahl kleiner als vierstellig werden die ersten Stellen mit Nullen aufgefüllt.

Parameter

val 16bit Wert

5.3.6.4 Msg_WriteInt

Debug Message Funktionen

Syntax

```
void Msg_WriteInt(int val);
```

Beschreibung

Der übergebene Integer wird im Ausgabenfenster dargestellt. Negativen Werten wird ein Minuszeichen vorangestellt.

Parameter

val 16bit integer Wert

5.3.6.5 Msg_WriteText

Debug Message Funktionen

Syntax

```
void Msg_WriteText(char text[]);
```

Beschreibung

Es werden alle Zeichen des char array bis zur terminierenden Null ausgegeben.

Parameter

text Zeiger auf char array

5.3.6.6 Msg_WriteWord

Debug Message Funktionen

Syntax

```
void Msg_WriteWord(word val);
```

Beschreibung

Der Parameter val wird als vorzeichenlose Zahl in das Ausgabenfenster geschrieben.

Parameter

val 16bit unsigned integer Wert

5.3.7 EEPROM

5.3.7.1 EEPROM_Read

EEPROM Funktionen

Syntax

```
byte EEPROM_Read(word pos);
```

Beschreibung

Liest ein byte von Position pos aus dem EEPROM. Die ersten 32 byte sind für das C-Control Pro OS reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu.

Parameter

pos Position im EEPROM

Rückgabewert

der Wert des byte an Position pos im EEPROM

5.3.7.2 EEPROM_Write

EEPROM Funktionen

Syntax

```
void EEPROM_Write(word pos,byte val);
```

Beschreibung

Schreibt ein byte an Position pos in das EEPROM. Die ersten 32 byte sind für das C-Control Pro OS reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu.

Parameter

pos Position im EEPROM
val der ins EEPROM zu schreibende Wert

5.3.8 I2C

Der Controller verfügt über eine I2C-Logik, die eine effektive Kommunikation ermöglicht. Der Controller arbeitet als I2C-Master (single master system). Eine Betriebsart als Slave ist möglich, aber in der jetzigen Version nicht implementiert.

5.3.8.1 I2C_Init

I2C Funktionen **Beispiel**

Syntax

```
void I2C_Init(byte I2C_BR);
```

Beschreibung

Diese Funktion initialisiert die I2C-Schnittstelle.

Parameter

I2C_BR gibt die Bitrate an. Folgende Werte sind schon vordefiniert:

```
I2C_100kHz
I2C_400kHz
```

5.3.8.2 I2C_Read_ACK

I2C Funktionen

Syntax

```
byte I2C_Read_ACK(void);
```

Beschreibung

Diese Funktion empfängt ein Byte und quittiert mit ACK. Danach kann mit I2C_Status der Status der Schnittstelle abgefragt werden.

Rückgabewert

gelesener Wert vom I2C Bus

5.3.8.3 I2C_Read_NACK

I2C Funktionen **Beispiel**

Syntax

```
byte I2C_Read_NACK(void);
```

Beschreibung

Diese Funktion empfängt ein Byte und quittiert mit NACK. Danach kann mit I2C_Status der Status der Schnittstelle abgefragt werden.

Rückgabewert

gelesener Wert vom I2C Bus

5.3.8.4 I2C_Start

I2C Funktionen **Beispiel**

Syntax

```
void I2C_Start(void);
```

Beschreibung

Diese Funktion leitet die Kommunikation mit einer Startsequenz ein. Danach kann mit I2C_Status der Status der Schnittstelle abgefragt werden.

Parameter

Keine

5.3.8.5 I2C_Status

I2C Funktionen

Syntax

```
byte I2C_Status(void);
```

Beschreibung

Mit I2C_Status kann der Status der Schnittstelle abgefragt werden. Die Bedeutung der Statusinformation ist in der Tabelle dargestellt.

Rückgabewert

aktueller I2C Status

5.3.8.6 I2C_Stop

I2C Funktionen **Beispiel**

Syntax

```
void I2C_Stop(void);
```

Beschreibung

Diese Funktion beendet die Kommunikation mit einer Stopsequenz. Danach kann mit I2C_Status der Status der Schnittstelle abgefragt werden.

Parameter

Keine

5.3.8.7 I2C_Write

I2C Funktionen **Beispiel**

Syntax

```
void I2C_Write(byte data);
```

Beschreibung

Diese Funktion sendet ein Byte. Danach kann mit I2C_Status der Status der Schnittstelle abgefragt werden.

Parameter

data Datenbyte

5.3.8.8 I2C Status Codes

Tabelle: **Status Codes Master Transmitter Mode**

Status Code	Beschreibung
0x08	eine START Sequenz wurde gesendet
0x10	eine "repeated" START Sequenz wurde gesendet
0x18	SLA+W wurde gesendet, ACK wurde empfangen
0x20	SLA+W wurde gesendet, NACK wurde empfangen
0x28	Data byte wurde gesendet, ACK wurde empfangen
0x30	Data byte wurde gesendet, NACK wurde empfangen
0x38	Konflikt in SLA+W or data bytes

Tabelle: **Status Codes Master Receiver Mode**

Status Code	Beschreibung
0x08	eine START Sequenz wurde gesendet
0x10	eine "repeated" START Sequenz wurde gesendet
0x38	Konflikt in SLA+R or data bytes
0x40	SLA+R wurde gesendet, ACK wurde empfangen
0x48	SLA+R wurde gesendet, NACK wurde empfangen
0x50	Data byte wurde empfangen, ACK wurde gesendet
0x58	Data byte wurde empfangen, NACK wurde gesendet

5.3.8.9 I2C Beispiel

Beispiel: EEPROM 24C64 lesen und schreiben ohne I2C_Status Abfrage

```
// I2C Initialization, Bit Rate 100kHz

main(void)
{
    word address;
    byte data,EEPROM_data;

    address=0x20;
    data=0x42;

    I2C_Init(I2C_100kHz );
    // write data to 24C64 (8k x 8) EEPROM
    I2C_Start();
    I2C_Write(0xA0);                          // DEVICE ADDRESS : A0
    I2C_Write(address>>8);                    // HIGH WORD ADDRESS
    I2C_Write(address);                       // LOW WORD ADDRESS
    I2C_Write(data);                          // write Data
    I2C_Stop();
    AbsDelay(5);                              // delay for EEPROM Write
Cycle

    // read data from 24C64 (8k x 8) EEPROM
    I2C_Start();
    I2C_Write(0xA0);                          // DEVICE ADDRESS : A0
    I2C_Write(address>>8);                    // HIGH WORD ADDRESS
    I2C_Write(address);                       // LOW WORD ADDRESS
    I2C_Start();                              // RESTART
    I2C_Write(0xA1);                          // DEVICE ADDRESS : A1
    EEPROM_data=I2C_Read_NACK();
    I2C_Stop();
    Msg_WriteHex(EEPROM_data);
}
```

5.3.9 Interrupt

Der Controller stellt eine Vielzahl an Interrupts zur Verfügung. Einige davon werden für Systemfunktionen verwendet und stehen dem Anwender nicht zur Verfügung. Folgende Interrupts können vom Anwender genutzt werden:

Tabelle Interrupts

Interrupt Name	Beschreibung
INT_0	externer Interrupt0, Eingang PortD.2
INT_1	externer Interrupt1, Eingang PortD.3
INT_2	externer Interrupt2, Eingang PortB.2
INT_TIM1CAPT	Timer1 Capture, Eingang PortD.6
INT_TIM1CMPA	Timer1 CompareA
INT_TIM1CMPB	Timer1 CompareB
INT_TIM1OVF	Timer1 Overflow
INT_TIM0COMP	Timer0 Compare
INT_TIM0OVF	Timer0 Overflow
INT_ANA_COMP	Analog Comparator
INT_ADC	ADC
INT_TIM2COMP	Timer2 Compare
INT_TIM2OVF	Timer2 Overflow

Der betreffende Interrupt muß in einer Interrupt Service Routine (ISR) die entsprechenden Anweisungen erhalten und der Interrupt muß freigegeben sein. Siehe **Beispiel.** Während der Abarbeitung einer Interruptroutine wird das Multithreading ausgesetzt.

Ein Signal auf INT_0 beim einschalten des C-Control Pro Moduls kann das Autostartverhalten stören. Nach der Pinzuordnung liegt der INT_0 auf dem gleichen Pin wie der SW1. Wird der SW1 beim einschalten des Moduls gedrückt, führt dies zur Aktivierung des seriellen Bootloader Modus und das Programm wird nicht automatisch gestartet.

5.3.9.1 Ext_Int0

Interrupt Funktionen

Syntax

```
void Ext_Int0(byte Mode);
```

Beschreibung

Diese Funktion schaltet den externen Interrupt 0 frei. Der Parameter Mode legt fest, wann ein Interrupt erzeugt werden soll. Ein Signal auf INT_0 kann zu Autostart Problemen führen.

Parameter

Mode Parameter:

1: ein low Pegel löst einen Interrupt aus
2: jede Flankenwechsel löst einen Interrupt aus
3: eine fallende Flanke löst einen Interrupt aus
4: eine steigende Flanke löst einen Interrupt aus

5.3.9.2 Ext_Int0Disable

Interrupt Funktionen

Syntax

```
void Ext_Int0Disable(void);
```

Beschreibung

Der externe Interrupt 0 wird gesperrt.

Parameter

Keine

5.3.9.3 Ext_Int1

Interrupt Funktionen

Syntax

```
void Ext_Int1(byte Mode);
```

Beschreibung

Diese Funktion schaltet den externen Interrupt 1 frei. Der Parameter Mode legt fest, wann ein Interrupt erzeugt werden soll.

Parameter

Mode Parameter:

1: ein low Pegel löst einen Interrupt aus
2: jede Flankenwechsel löst einen Interrupt aus
3: eine fallende Flanke löst einen Interrupt aus
4: eine steigende Flanke löst einen Interrupt aus

5.3.9.4 Ext_Int1Disable

Interrupt Funktionen

Syntax

```
void Ext_Int1Disable(void);
```

Beschreibung

Der externe Interrupt 1 wird gesperrt.

Parameter

Keine

5.3.9.5 Ext_Int2

Interrupt Funktionen

Syntax

```
void Ext_Int2(byte Mode);
```

Beschreibung

Diese Funktion schaltet den externen Interrupt 0 frei. Der Parameter Mode legt fest, wann ein Interrupt erzeugt werden soll.

Parameter

Mode Parameter:

0: eine fallende Flanke löst einen Interrupt aus
1: eine steigende Flanke löst einen Interrupt aus

5.3.9.6 Ext_Int2Disable

Interrupt Funktionen

Syntax

```
void Ext_Int2Disable(void);
```

Beschreibung

Der externe Interrupt 2 wird gesperrt.

Parameter

Keine

5.3.9.7 Irq_GetCount

Interrupt Funktionen **Beispiel**

Syntax

```
byte Irq_GetCount(void);
```

Beschreibung

Signalisiert das der Interrupt abgearbeitet wurde (interrupt acknowledge). Wird die Funktion nicht am Ende einer Interruptroutine aufgerufen, wird ununterbrochen in den Interrupt gesprungen.

Rückgabewert

gibt an wie oft der Interrupt von der Hardware bis zum Aufruf von Irq_GetCount() ausgelöst wurde. Ein Wert größer 1 kann dann auftreten wenn die Hardware schneller Interrupts generiert als der Interpreter die Interruptroutine abarbeiten kann.

5.3.9.8 Irq_SetVect

Interrupt Funktionen **Beispiel**

Syntax

```
void Irq_SetVect(byte irqnr,word vect);
```

Beschreibung

Setzt die aufzurufende Interrupt Funktion für einen bestimmten Interrupt. Am Ende der Interruptroutine muß die Funktion Irq_GetCount() aufgerufen werden, ansonsten wird ununterbrochen in die Interrupt Funktion gesprungen.

Parameter

`irqnr` spezifiziert den Typ des Interrupts (siehe Tabelle)
`vect` ist der Name der aufzurufenden Interrupt Funktion

Tabelle Interrupt Vektoren:

Nr	Interrupt Name	Beschreibung
0	INT_0	externer Interrupt0
1	INT_1	externer Interrupt1
2	INT_2	externer Interrupt2
3	INT_TIM1CAPT	Timer1 Capture
4	INT_TIM1CMPA	Timer1 CompareA
5	INT_TIM1CMPB	Timer1 CompareB
6	INT_TIM1OVF	Timer1 Overflow
7	INT_TIM0COMP	Timer0 Compare
8	INT_TIM0OVF	Timer0 Overflow
9	INT_ANA_COMP	Analog Comparator
10	INT_ADC	ADC
11	INT_TIM2COMP	Timer2 Compare
12	INT_TIM2OVF	Timer2 Overflow

5.3.9.9 IRQ Beispiel

Beispiel: Verwendung von Interrupt Routinen

```
// Timer 2 läuft normalerweise im 10ms Takt. In diesem
// Beispiel wird daher die Variable cnt alle 10ms um 1 erhöht

int cnt;

void ISR(void)
{
    int irqcnt;

    cnt=cnt+1;
    irqcnt=Irq_GetCount(INT_TIM2COMP);
}

void main(void)
{
    cnt=0;

    Irq_SetVect(INT_TIM2COMP,ISR);
    while(true);  // Endlosschleife
}
```

5.3.10 Keyboard

Ein Teil dieser Routinen sind im Interpreter implementiert, ein anderer Teil wird durch Hinzufügen der Bibliothek "Key_Lib.cc" aufrufbar. Da die Funktionen in "LCD_Lib.cc" durch Bytecode realisiert werden, sind sie langsamer in der Abarbeitung. Bibliotheksfunktionen haben allerdings den Vorteil, das man bei Nichtgebrauch diese Funktionen durch Weglassen der Bibliothek aus dem Projekt nimmt. Direkte Interpreterfunktionen sind immer präsent aber kosten Flashspeicher.

5.3.10.1 Key_Init

Keyboard Funktionen (Bibliothek "*Key_Lib.cc*")

Syntax

```
void Key_Init(void);
```

Beschreibung

Das globale array keymap wird mit den ASCII Werten der Tastatur initialisiert.

Parameter

Keine

5.3.10.2 Key_Scan

Keyboard Funktionen

Syntax

```
word Key_Scan(void);
```

Beschreibung

Key_Scan sucht sequentiell die Eingabepins der angeschlossenen Tastatur ab und gibt das Ergebnis als Bitfeld zurück. Die "1" Bits repräsentieren die Tasten die zum Zeitpunkt des Scans gedrückt wurden.

Rückgabewert

16 Bits welche die einzelnen Eingabeleitungen der Tastatur repräsentieren

5.3.10.3 Key_TranslateKey

Keyboard Funktionen (Bibliothek "*Key_Lib.cc*")

Syntax

```
char Key_TranslateKey(word keys);
```

Beschreibung

Diese Hilfsfunktion liefert das Zeichen zurück das dem ersten
Auftauchen einer "1" im Bitfeld des Eingabeparameters entspricht.

Parameter

keys Bitfeld das von Key_Scan() zurückgeliefert wird

Rückgabewert

ASCII Wert der erkannten Taste
-1 wenn keine Taste gedrückt wird

5.3.11 LCD

Ein Teil dieser Routinen sind im Interpreter implementiert, ein anderer Teil wird durch Hinzufügen der Bibliothek "LCD_Lib.cc" aufrufbar. Da die Funktionen in "LCD_Lib.cc" durch Bytecode realisiert werden, sind sie langsamer in der Abarbeitung. Bibliotheksfunktionen haben allerdings den Vorteil, das man bei Nichtgebrauch diese Funktionen durch Weglassen der Bibliothek aus dem Projekt nimmt. Direkte Interpreterfunktionen sind immer präsent aber kosten Flashspeicher.

5.3.11.1 LCD_ClearLCD

LCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_ClearLCD(void);
```

Beschreibung

Löscht das Display und schaltet den Cursor ein.

Parameter

Keine

5.3.11.2 LCD_CursorOff

LCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_CursorOff(void);
```

Beschreibung

Schaltet den Cursor des Display aus.

Parameter

Keine

5.3.11.3 LCD_CursorOn

LLCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_CursorOn(void);
```

Beschreibung

Schaltet den Cursor des Display ein.

Parameter

Keine

5.3.11.4 LCD_CursorPos

LCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_CursorPos(byte pos);
```

Beschreibung

Setzt den Cursor auf Position pos.

Parameter

pos Cursorposition

Wert von pos	**Position im Display**
0x00-0x07	0-7 in der 1. Zeile
0x40-0x47	0-7 in der 2. Zeile

5.3.11.5 LCD_Init

LCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_Init(void);
```

Beschreibung

"Highlevel" Initialisierung des LCD Display. Ruft als erstes LCD_InitDisplay() auf.

Parameter

Keine

5.3.11.6 LCD_SubInit

LCD Funktionen

Syntax

```
void LCD_SubInit(void);
```

Beschreibung

Initialisiert die Ports für die Displaysteuerung auf Assemblerebene. Muß als erste Routine vor allen anderen LCD Ausgabefunktionen aufgerufen werden. Wird als erstes Kommando von LCD_Init() benutzt.

Parameter

Keine

5.3.11.7 LCD_WriteChar

LCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_WriteChar(char c);
```

Beschreibung

Schreibt ein Zeichen an die Cursorposition im LCD Display.

Parameter

c ASCII Wert des Zeichens

5.3.11.8 LCD_TestBusy

LCD Funktionen

Syntax

```
void LCD_TestBusy(void);
```

Beschreibung

Die Funktion wartet bis der Display Controller nicht mehr "*Busy*" ist. Wird vorher auf den Controller zugegriffen wird der Datenaufbau im Display gestört.

Parameter

Keine

5.3.11.9 LCD_WriteCTRRegister

LCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_WriteCTRRegister(byte cmd);
```

Beschreibung

Schickt ein Kommando zum Display Controller.

Parameter

cmd Kommando in Byteform

5.3.11.10 LCD_WriteDataRegister

LCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_WriteDataRegister(char x);
```

Beschreibung

Schickt ein Datenbyte zum Display Controller.

Parameter

x Datenbyte

5.3.11.11 LCD_WriteNibble

LCD Funktionen

Syntax

```
void LCD_WriteNibble(byte x,byte cmd);
```

Beschreibung

Sendet ein 4-Bit Kommando zum Display Controller. x enthält dabei eine "1" für das höherwertige Nibble und eine "2" für das niederwertige Nibble. Das Kommando Nibble steht in Bit 4 bis 7 von Byte cmd. Für weitere Details ist das Datenblatt des Display Controllers zu studieren.

Parameter

x Datennibble
cmd Kommandonibble

5.3.11.11 LCD_WriteRegister

LCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_WriteRegister(byte y,byte x);
```

Beschreibung

LCD_WriteRegister zerlegt das Datenbyte y in zwei Nibble und schickt sie zum Display Controller.

Parameter

y Datenbyte
x Kommandonibble

5.3.11.12 LCD_WriteText

LCD Funktionen (Bibliothek "*LCD_Lib.cc*")

Syntax

```
void LCD_WriteText(char text[]);
```

Beschreibung

Es werden alle Zeichen des char array bis zur terminierenden Null ausgegeben.

Parameter

text char array

5.3.12 Port

Der Atmel Mega 32 hat 4 Ein-/Ausgabeports zu je 8 Bit. Jedes Bit der einzelnen Ports kann als Eingang oder als Ausgang konfiguriert werden. Da aber die Anzahl der Pins der Mega 32 Risc CPU begrenzt ist, sind zusätzliche Funktionen einzelnen Ports zugeordnet. Eine Tabelle der Pinzuordnung findet sich hier.

Es ist wichtig vor der Programmierung die Pinzuordnung zu studieren, da wichtige Funktionen der Programmentwicklung (z.B. die USB Schnittstelle des Application Boards) auf bestimmten Ports liegen. Werden diese Ports umprogrammiert oder sind die zugehörigen Jumper auf dem Application Board nicht mehr gesetzt, kann es passieren das die Entwicklungsumgebung keine Programme mehr zum C-Control Pro übertragen kann.

Die Datenrichtung (Eingang/Ausgang) kann mit der Funktion Port_DataDir oder Port_DataDirBit festgelegt werden. Ist ein Pin als Eingang konfiguriert, so kann dieser Pin entweder hochohmig ("floatend") oder mit einem internen Pullup betrieben werden. Schreibt man mit Port_Write oder Port_WriteBit eine "1" auf einen Eingang, so wird der Pullup Widerstand (Bezugspegel VCC) aktiviert und der Eingang ist definiert.

5.3.12.1 Port_DataDir

Port Funktionen Beispiel

Syntax

```
void Port_DataDir(byte port,byte val);
```

Beschreibung

Die Funktion Port_DataDir konfiguriert die Bits des Ports zur Ein- oder Ausgabe. Ist das Bit '1' dann wird der Pin der entsprechenden Bitposition auf Ausgang geschaltet. Ein Beispiel: Ist `port` = PortB und val = 0x02, dann wird der Pin 2 des Atmel Mega (gleich PortB.1 - siehe Pinzuordnung) auf Ausgang konfiguriert.

Parameter

port Portnummer (siehe Tabelle)
val Ausgabe byte

Portnummern Tabelle

Definition	Wert
PortA	0
PortB	1
PortC	2
PortD	3

5.3.12.2 Port_DataDirBit

Port Funktionen

Syntax

```
void Port_DataDirBit(byte portbit,byte val);
```

Beschreibung

Die Funktion Port_DataDirBit konfiguriert ein Bit (Pin) eines Ports zur Ein- oder Ausgabe. Ist das Bit '1' dann wird der Pin auf Ausgang geschaltet, sonst auf Eingang. Ein Beispiel: Ist `portbit` = 9 und val = 0, dann wird der Pin 2 des Atmel Mega (gleich PortB.1 - siehe Pinzuordnung) auf Eingang konfiguriert.

Parameter

portbit Bitnummer des Ports (siehe Tabelle)
val 0=Eingang, 1= Ausgang

Portbits Tabelle

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15

Definition	Portbit
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31

5.3.12.3 Port_Read

Port Funktionen

Syntax

```
byte Port_Read(byte port);
```

Beschreibung

Liest ein Byte vom spezifizierten Port. Nur die Pins des Ports die auf Eingang geschaltet sind, liefern einen gültigen Wert an der entsprechenden Bitposition in dem gelesenen Byte zurück. (Für die Abbildung zwischen Portbits und den Pins des Atmel Mega Chips siehe Pinzuordnung.)

Parameter

port Portnummer (siehe Tabelle)

Rückgabewert

Wert des Ports

Portnummern Tabelle

Definition	Wert
PortA	0
PortB	1
PortC	2
PortD	3

5.3.12.4 Port_ReadBit

Port Funktionen

Syntax

```
byte Port_ReadBit(byte port);
```

Beschreibung

Liest einen Bitwert des spezifizierten Ports. Der entsprechende Pin des Ports muß auf Eingang geschaltet sein. (Für die Abbildung zwischen Portbits und den Pins des Atmel Mega Chips siehe Pinzuordnung.)

Parameter

portbit Bitnummer des Ports (siehe Tabelle)

Rückgabewert

Bitwert des Ports (0 oder 1)

Portbits Tabelle

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15

Definition	Portbit
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31

5.3.12.5 Port_Write

Port Funktionen Beispiel

Syntax

```
void Port_Write(byte port,byte val);
```

Beschreibung

Schreibt ein Byte auf den spezifizierten Port. Nur die Pins des Ports die auf Ausgang geschaltet sind, übernehmen die Bitwerte des übergebenen Parameters.
Ist ein Pin auf Eingang geschaltet, so kann der interne Pullup Widerstand eingeschaltet (1) oder abgeschaltet (0) werden. (Für die Abbildung zwischen Portbits und den Pins des Atmel Mega Chips siehe Pinzuordnung.)

Parameter

port Portnummer (siehe Tabelle)
val Ausgabe byte

Portnummern Tabelle

Definition	Wert
PortA	0
PortB	1
PortC	2
PortD	3

5.3.12.6 Port_WriteBit

Port Funktionen

Syntax

```
void Port_WriteBit(byte portbit,byte val);
```

Beschreibung

Die Funktion Port_WriteBit setzt den Wert eines Pins der auf Ausgang geschaltet ist. Ist ein Pin auf Eingang geschaltet, so kann der interne Pullup Widerstand eingeschaltet (1) oder abgeschaltet (0) werden. (Für die Abbildung zwischen Portbits und den Pins des Atmel Mega Chips siehe Pinzuordnung.)

Parameter

portbit Bitnummer des Ports (siehe Tabelle)
val darf 0 oder 1 sein

Portbits Tabelle

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15

Definition	Portbit
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31

5.3.12.7 Port Beispiel

```
// Programm läßt abwechselnd die beiden LEDs auf dem
// Application Board im 1 Sekunden Rhythmus blinken

void main(void)
{
    Port_DataDir(PortD,0xc0);  // die obersten beiden Bits von Port D werden
auf Ausgang geschaltet

    while(true)  // Endlosschleife
    {
        Port_Write(PortD,0x40);
        AbsDelay(1000);
        Port_Write(PortD,0x80);
        AbsDelay(1000);
    }
}
```

5.3.13 RS232

Im Gegensatz zu den Debug Message Funktionen arbeiten alle seriellen Routinen nicht mit Interrupt sondern "pollend". Das heißt das die Funktionen erst dann zurückkehren wenn das Zeichen oder Text geschrieben bzw. gelesen wurde. Die serielle Schnittstelle kann mit Geschwindigkeiten bis zu 230.4kbaud betrieben werden.

5.3.13.1 Serial_Init

Serielle Funktionen **Beispiel**

Syntax

```
void Serial_Init(byte par,byte divider);
```

Beschreibung

Die serielle Schnittstelle wird initialisiert. Der Wert par wird durch Oderieren der vordefinierten Bitwerte zusammengestellt. Man oderiert erst Zeichenlänge, dann Anzahl der Stopbits und dann Parity. Z.B. "SR_7BIT|SR_2STOPSR_EVEN_PAR" für 7 Bit pro Zeichen, 2 Stop Bit und gerade Parität (siehe auch Beispiel). Die Baudrate wird als Teilerwert angegeben, wie auch in der Tabelle spezifiziert.

Parameter

par Schnittstellenparameter (siehe Tabelle)
divider Baudrateninitialisierung mittels Teiler (siehe Tabelle)

Tabelle par Definitionen:

Definition	Funktion
SR_5BIT	5 Bit Zeichenlänge
SR_6BIT	6 Bit Zeichenlänge
SR_7BIT	7 Bit Zeichenlänge
SR_8BIT	8 Bit Zeichenlänge
SR_1STOP	1 Stop Bit
SR_2STOP	2 Stop Bit
SR_NO_PAR	no Parity
SR_EVEN_PAR	even Parity
SR_ODD_PAR	odd Parity

Tabelle divider Definitionen:

divider	Definition	Baudrate
383	SR_BD2400	2400bps
191	SR_BD4800	4800bps
95	SR_BD9600	9600bps
63	SR_BD14400	14400bps
47	SR_BD19200	19200bps
31	SR_BD28800	28800bps
23	SR_BD38400	38400bps
15	SR_BD57600	57600bps
11	SR_BD76800	76800bps
7	SR_BD115200	115200bps
3	SR_BD230400	230400bps

5.3.13.2 Serial_Read

Serielle Funktionen

Syntax

```
byte Serial_Read(void);
```

Beschreibung

Ein byte wird von der seriellen Schnittstelle gelesen. Ist kein byte im seriellen Puffer, kehrt die Funktion erst dann zurück wenn ein Zeichen empfangen wurde.

Rückgabewert

empfangenes byte aus der seriellen Schnittstelle

5.3.13.3 Serial_ReadExt

Serielle Funktionen

Syntax

```
word Serial_ReadExt(void);
```

Beschreibung

Ein byte wird von der seriellen Schnittstelle gelesen. Im Gegensatz zu Serial_Read(), kehrt die Funktion auch dann sofort zurück wenn kein Zeichen in der seriellen Schnittstelle ist. In diesem Fall wird dann der Wert 256 (0x100) zurückgegeben.

Rückgabewert

empfangenes byte aus der seriellen Schnittstelle

5.3.13.4 Serial_Write

Serielle Funktionen **Beispiel**

Syntax

```
void Serial_Write(byte val);
```

Beschreibung

Ein byte wird zur seriellen Schnittstelle geschickt.

Parameter

val der auszugebende byte Wert

5.3.13.5 Serial_WriteText

Serielle Funktionen

Syntax

```
void Serial_WriteText(char text[]);
```

Beschreibung

Es werden alle Zeichen des char array bis zur terminierenden Null auf der seriellen ausgegeben.

Parameter

text char array

5.3.13.6 Serial Beispiel

```
// Stringausgabe auf der seriellen Schnittstelle
void main(void)
{
    int i;
    char str[10];

    str="test";
    i=0;
    // Initialisiere Schnittstelle mit 19200baud, 8 Bit, 1 Stop Bit, keine
Parität
     Serial_Init(SR_8BIT|SR_1STOP|SR_NO_PAR,SR_BD19200);

    while(str[i]) Serial_Write(str[i++]);  // Gib den String aus
}
```

5.3.14 Strings

Ein Teil dieser Stringroutinen sind im Interpreter implementiert, ein anderer Teil wird durch Hinzufügen der Bibliothek "String_Lib.cc" aufrufbar. Da die Funktionen in "String_Lib.cc" durch Bytecode realisiert werden, sind sie langsamer in der Abarbeitung. Bibliotheksfunktionen haben allerdings den Vorteil, das man bei Nichtgebrauch diese Funktionen durch Weglassen der Bibliothek aus dem Projekt nimmt. Direkte Interpreterfunktionen sind immer präsent aber kosten Flashspeicher.

Es existiert kein expliziter "String" Datentyp. Ein String basiert auf einem character array. Man muß die Größe des arrays so wählen das alle Zeichen des Strings in das character array passen. Zusätzlich wird Platz für ein Terminierungszeichen (dezimal Null) benötigt, um das Ende der Zeichenkette anzuzeigen.

5.3.14.1 Str_Comp

String Funktionen

Syntax

```
char Str_Comp(char str1[],char str2[]);
```

Beschreibung

Zwei Strings werden miteinander verglichen.

Parameter

str1 Zeiger auf char array 1
str2 Zeiger auf char array 2

Rückgabewert

0 wenn beide Strings gleich sind
<0 wenn an der Unterscheidungsstelle der 1. String kleiner ist
>0 wenn an der Unterscheidungsstelle der 1. String größer ist

5.3.14.2 Str_Copy

String Funktionen

Syntax

```
void Str_Copy(char destination[],char source[],word offset);
```

Beschreibung

Der Quellstring (source) wird auf den Zielstring (destination) kopiert. Bei der Kopieraktion wird aber in jedem Fall daß String Terminierungszeichen der Quellzeichenkette mit kopiert.

Parameter

`destination` Zeiger auf den Zielstring
`source` Zeiger auf die Quellstring
offset Anzahl der Zeichen um die der Quellstring verschoben auf den Zielstring kopiert wird.

Hat offset den Wert **STR_APPEND** (0xffff), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird der Source String hinter den Destination String kopiert.

5.3.14.3 Str_Fill

String Funktionen (Bibliothek "*String_Lib.cc*")

Syntax

```
void Str_Fill(char dest[],char c,word len);
```

Beschreibung

Der String dest wird mit dem Zeichen c aufgefüllt.

Parameter

dest Zeiger auf den Zielstring
c das Zeichen das wiederholt in den String kopiert wird
len Anzahl wie oft c in den Zielstring geschrieben wird

5.3.14.4 Str_Isalnum

String Funktionen (Bibliothek "*String_Lib.cc*")

Syntax

```
byte Str_Isalnum(char c);
```

Beschreibung

Ein Zeichen c wird darauf überprüft, ob es aus dem Alphabet stammt oder eine Ziffer ist.

Parameter

c das zu überprüfende Zeichen

Rückgabewert

1 wenn das Zeichen numerisch oder alphabetisch ist (in Groß- oder Kleinschreibung)
0 sonst

5.3.14.5 Str_Isalpha

String Funktionen (Bibliothek "*String_Lib.cc*")

Syntax

```
byte Str_Isalpha(char c);
```

Beschreibung

Ein Zeichen c wird darauf überprüft, ob es aus dem Alphabet stammt.

Parameter

c das zu überprüfende Zeichen

Rückgabewert

1 wenn das Zeichen alphabetisch ist (in Groß- oder Kleinschreibung)
0 sonst

5.3.14.6 Str_Len

String Funktionen

Syntax

```
word Str_Len(char str[]);
```

Beschreibung

Die Länge der Zeichenkette (des character arrays) wird zurückgegeben.

Parameter

str Zeiger auf String

Rückgabewert

Anzahl der Zeichen im String (ohne die terminierende Null).

5.3.14.7 Str_Substr

String Funktionen (Bibliothek "*String_Lib.cc*")

Syntax

```
int Str_Substr(char source[],char search[]);
```

Beschreibung

Ein String search wird im String source gesucht. Wird die gesuchte Zeichenkette gefunden, so wird ihre Position zurückgegeben.

Parameter

source String der durchsucht wird
search Zeichenkette die gesucht wird

Rückgabewert

Position des Suchstrings in der untersuchten Zeichenkette
-1 sonst

5.3.14.8 Str_WriteFloat

String Funktionen

Syntax

```
void Str_WriteFloat(float n,int decimal,char text[],word offset);
```

Beschreibung

Die float Zahl n wird in einen ASCII String mit decimal Dezimalstellen konvertiert. Das Ergebnis wird im String text mit einem Versatz von offset abgespeichert.

Parameter

n float Zahl
decimal Anzahl der Dezimalstellen auf die n konvertiert wird
text Zeiger auf den Zielstring
offset Anzahl der Zeichen mit der die ASCII Darstellung der float Zahl verschoben in den Text String kopiert wird

Hat offset den Wert **STR_APPEND** (0xffff), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird die float Zahl an den Text String angehängt.

5.3.14.9 Str_WriteInt

String Funktionen

Syntax

```
void Str_WriteFloat(int n,char text[],word offset);
```

Beschreibung

Die Integer Zahl n wird in einen vorzeichenbehafteten ASCII String konvertiert. Das Ergebnis wird im String text mit einem Versatz von offset abgespeichert.

Parameter

n integer Zahl
text Zeiger auf den Zielstring
offset Anzahl der Zeichen mit der die ASCII Darstellung der Zahl verschoben in den Text String kopiert wird

Hat offset den Wert **STR_APPEND** (0xffff), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird die Integer Zahl an den Text String angehängt.

5.3.14.10 Str_WriteWord

String Funktionen

Syntax

```
void Str_WriteWord(word n,byte base,char text[],word offset,byte
minwidth);
```

Beschreibung

Das Wort n wird in einen ASCII String konvertiert. Das Ergebnis wird im String text mit einem Versatz von offset abgespeichert. Man kann für die Ausgabe eine beliebige Basis angeben. Mit einer base von 2 erhält man Binärzahlen, mit 8 Oktalzahlen und bei 16 werden Hexzahlen ausgegeben, etc. Ist die Basis größer als 16, werden weitere Buchstaben des Alphabets herangezogen. Ist z.B. die Basis 18, so hat die Zahl die Ziffern 0-9, und 'A' - 'H'. Ist der ASCII String kürzer als minwidth, so wird der Beginn des Strings mit Nullen aufgefüllt.

Parameter

n 16 Bit Wort
base Basis des Zahlensystems
text Zeiger auf den Zielstring
offset Anzahl der Zeichen mit der die ASCII Darstellung der Zahl verschoben in den Text String kopiert wird
minwidth minimale Breite des Strings

Hat offset den Wert **STR_APPEND** (0xffff), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird die Integer Zahl an den Text String angehängt.

5.3.15 Threads

Multithreading

Unter Multithreading versteht man die quasi parallele Abarbeitung mehrerer Abläufe in einem Programm. Einer von diesen Abläufen wird Thread (engl. Faden) genannt. Beim Multithreading wird in schnellen Abständen zwischen den verschiedenen Threads gewechselt, so daß beim Anwender der Eindruck von Gleichzeitigkeit einsteht.

Die C-Control Pro Firmware unterstützt außer dem Hauptprogramm (Thread "0") bis zu 15 zusätzliche Threads. Beim Multithreading wird nach einer bestimmten Anzahl von verarbeiteten Byte Instruktionen der aktuelle Thread auf den Status "*inaktiv*" gesetzt und der nächste ausführbare Thread wird gesucht. Danach startet die Abarbeitung des neuen Threads. Der neue Thread kann wieder derselbe wie vorher sein, je nachdem wie viele Threads aktiviert wurden oder für eine Ausführung bereit sind. Das Hauptprogramm gilt als erster Thread. Daher ist Thread "0" immer aktiv, auch wenn explizit keine Threads gestartet worden sind.

Die Priorität eines Threads kann beeinflußt werden, in dem man ändert wie viele Bytecodes ein Thread bis zum nächsten Threadwechsel ausführen darf (siehe Threadoptionen). Je kleiner die Anzahl der Zyklen bis zum Wechsel, desto geringer die Priorität des Threads. Die Ausführungszeit eines Bytecodes ist im Mittel ca. 7-9 µsec. Bei einzelnen Bytecode Befehlen dauert es jedoch länger, z.B. Floatingpoint Operationen.

Auch interne Interpreterfunktionen gelten als ein Zyklus. Da z.B. Serial_Read wartet bis ein Zeichen von der seriellen Schnittstelle ankommt, kann in Ausnahmefällen ein Zyklus sehr lange dauern.

Ein Thread bekommt für seine lokalen Variablen soviel Platz wie ihm in den Threadoptionen des Projekts zugewiesen wird. Eine Ausnahme ist Thread "0" (das Hauptprogramm). Dieser Thread erhält den restlichen Speicherplatz, den die anderen Threads übrig lassen. Man sollte daher vorher planen wie viel Speicherplatz jeder zusätzliche Thread wirklich benötigt.

Damit zusätzliche Threads gestartet werden können muß "*Multithreading*" in den Projektoptionen eingeschaltet werden, und die Parameter für die weiteren Threads in den Threadoptionen auf korrekte Wert gesetzt werden.

Thread Synchronisation

Manchmal ist es nötig daß ein Thread auf den anderen wartet. Dies kann z.B. eine gemeinsame

Hardwareresource sein, die nur ein Thread bearbeiten kann. Oder manchmal definiert man kritische Programmbereiche die nur ein Thread betreten darf. Diese Funktionen werden durch die Anweisungen Thread_Wait und Thread_Signal realisiert.

Ein Thread der warten soll führt die Anweisung Thread_Wait mit einer Signal Nummer aus. Der Zustand des Threads wird auf *wartend* gesetzt. Dies bedeutet das dieser Thread bei einem möglichen Threadwechsel übergangen wird. Hat der andere Thread seine kritische Arbeit beendet gibt er den Befehl Thread_Signal mit der gleichen Signalnummer die der andere Thread für Thread_Wait benutzt hat. Der Threadzustand des wartenden Threads wechselt dann von *wartend* zu *inaktiv*. Jetzt wird er bei einem möglichen Threadwechsel wieder berücksichtigt.

Deadlocks

Begeben sich alle aktiven Threads in einen Wartezustand mit Thread_Wait so gibt es keinen Thread mehr der die anderen Threads aus dem wartenden Zustand befreien könnte. Diese Konstellationen sind bei der Programmierung zu vermeiden.

Tabelle Threadzustände:

Zustand	Bedeutung
aktiv	Der Thread wird momentan abgearbeitet
inaktiv	Kann nach einem Threadwechsel wieder aktiviert werden
schlafend	Wird nach einer Anzahl von Ticks wieder auf "inaktiv" gesetzt
wartend	Der Thread wartet auf ein Signal

5.3.15.1 Thread_Cycles

Thread Funktionen

Syntax

```
void Thread_Cycles(byte thread, word cycles);
```

Beschreibung

Setzt die Anzahl der Bytecode Instruktionen bis zum nächsten Threadwechsel auf cycles .

Wird ein Thread neu gestartet, erhält er immer die Anzahl der Zyklen zugewiesen, die in den Projektoptionen definiert wurden. Es macht also nur Sinn Thread_Cyles() aufzurufen, nachdem ein Thread gestartet wurde.

Parameter

thread (0-15) Nummer des Threads dessen Zyklus geändert werden soll
cycles Anzahl der Zyklen bis zum Threadwechsel

5.3.15.2 Thread_Delay

Thread Funktionen **Beispiel**

Syntax

```
void Thread_Delay(word delay);
```

Beschreibung

Hiermit wird ein Thread für eine bestimmte Zeit auf "*schlafend*" geschaltet. Nach dem angegebenen Zeitraum ist er wieder für die Abarbeitung bereit. Der Zeitraum wird in Ticks angegeben, die von Timer 2 erzeugt werden. Wird Timer 2 abgeschaltet oder für einen anderen Zweck gebraucht ist die Funktionsweise von Thread_Delay() undefiniert.

 Auch wenn Thread_Delay() normalerweise wie eine Wartefunktion arbeitet, so muß man doch beachten, daß nach der Wartezeit der Thread nicht immer automatisch wieder ausgeführt wird. Er ist dann zwar bereit, muß aber erst durch einen Threadwechsel wieder Ausführungszeit bekommen.

Parameter

delay Anzahl von 10ms Ticks die gewartet werden soll

5.3.15.3 Thread_Kill

Thread Funktionen

Syntax

```
void Thread_Kill(byte thread);
```

Beschreibung

Beendet die Abarbeitung eines Threads. Wird als Threadnummer 0 übergeben, so wird das Hauptprogramm und damit der ganze Interpreterlauf angehalten.

Parameter

thread (0-15) Nummer des Threads

5.3.15.4 Thread_Lock

Thread Funktionen

Syntax

```
void Thread_Lock(byte lock);
```

Beschreibung

Mit dieser Funktion kann ein Thread seinen Threadwechsel unterbinden. Dies ist sinnvoll wenn bei einer Serie von Portausgaben oder anderen Hardware Befehlen die zeitliche Trennung durch einen Threadwechsel vermieden werden soll.

 Wird vergessen das "Lock" wieder auszuschalten so findet kein Multithreading mehr statt.

Parameter

lock bei 1 wird der Threadwechsel unterbunden, bei 0 wieder zugelassen

5.3.15.5 Thread_Resume

Thread Funktionen

Syntax

```
void Thread_Resume(byte thread);
```

Beschreibung

Hat ein Thread des Zustand "*wartend*", so kann er hiermit wieder auf "*inaktiv*" gesetzt werden. Der Status "*inaktiv*" bedeutet, das der Thread bereit ist um bei einem Threadwechsel wieder aktiviert zu werden.

Parameter

thread (0-15) Nummer des Threads

5.3.15.6 Thread_Signal

Thread Funktionen

Syntax

```
void Thread_Signal(byte signal);
```

Beschreibung

Wurde ein Thread mittels Thread_Wait() auf "*wartend*" gesetzt, so kann der Zustand mit Hilfe von Thread_Signal() wieder auf "*inaktiv*" geändert werden. Der Parameter signal muß den gleichen Wert haben der bei Thread_Wait() benutzt wurde.

Parameter

signal Wert des Signals

5.3.15.7 Thread_Start

Thread Funktionen **Beispiel**

Syntax

```
void Thread_Start(byte thread, word func);
```

Beschreibung

Ein neuer Thread wird gestartet. Als Startfunktion für den Thread kann eine beliebige Funktion genutzt werden.

 Wird eine Funktion ausgesucht die Übergabeparameter enthält, so ist beim Start des Threads der Inhalt dieser Parameter nicht definiert!

Parameter

thread (0-15) Nummer des Threads der gestartet werden soll
func Name der Funktion in welcher der neue Thread gestartet wird

5.3.15.8 Thread_Wait

Thread Funktionen

Syntax

```
void Thread_Wait(byte signal);
```

Beschreibung

Der Thread bekommt den Status "*wartend*". Mittels Thread_Resume() oder Thread_Signal() kann der Thread wieder in einen inaktiven Zustand kommen.

Parameter

signal Wert des Signals

5.3.15.9 Thread Beispiel

```
// Demoprogramm zum Multithreading - Bit 26 ist SW1 und Bit 27 SW2
// das Programm ist nicht entprellt, ein kurzes Tasten führt daher zu
// mehrfacher Ausgabe des Strings

void thread1(void)
{
    while(true)  // Endlosschleife
    {
        if(!Port_ReadBit(27)) Msg_WriteText(str2);  // SW2 wurde gedrückt
    }
}
```

```
char str1[12],str2[12];

void main(void)
{
    str1="Taster 1";
    str2="Taster 2";

    Port_DataDir(PortD,0);  // Port D auf Eingang
    Port_Write(PortD,0xff);  // Pullup für alle Eingänge setzen

    Thread_Start(1,thread1); // Thread 1 starten

    while(true)     // Endlosschleife
    {
        if(!Port_ReadBit(26)) Msg_WriteText(str1); // SW1 wurde gedrückt
    }
}
```

5.3.15.1 Thread Beispiel 2

```
// Demoprogramm für Multithreading mit Thread_Delay
void thread1(void)
{
    while(true)     // Endlosschleife
    {
        Msg_WriteText(str2); Thread_Delay(200); // Text wird alle 2 Sekunden
ausgegeben
    }
}

char str1[12],str2[12];

void main(void)
{
    str1="Thread1";
    str2="Thread2";

    Thread_Start(1,thread1);   // Thread 1 wird gestartet

    while(true)     // Endlosschleife
    {
        Thread_Delay(100); Msg_WriteText(str1); // Text wird jede Sekunde
ausgegeben

    }
}
```

5.3.16 Timer

Es stehen im C-Control Pro Mega 32 zwei unabhängige Timer-Counter zur Verfügung. *Timer_0* mit 8 Bit und *Timer_1* mit 16 Bit. *Timer_2* wird von der Firmware als interne Zeitbasis verwendet, und ist fest auf einen 10ms Interrupt eingestellt. Mann kann die internen Timer für vielfältige Aufgaben einsetzen:

- Ereigniszähler
- Frequenzerzeugung
- Pulsweitenmodulation
- Timerfunktionen

- Puls & Periodenmessung
- Frequenzmessung

5.3.16.1 Ereigniszähler

Hier zwei Beispiele wie die Timer als Ereigniszähler genutzt werden:

Timer0 (8 Bit)

```
//  Beispiel: Pulszählung mit CNT0
Timer_T0CNT();
pulse(n);             //  n Pulse generieren
count=Timer_T0GetCNT();
```

Timer1 (16 Bit)

```
//  Beispiel: Pulszählung mit CNT1
Timer_T1CNT();
pulse(n);             //  n Pulse generieren
count=Timer_T1GetCNT();
```

5.3.16.2 Frequenzerzeugung

Zur Frequenzerzeugung können *Timer_0* und *Timer_1* folgendermaßen eingesetzt werden:

Timer0 (8 Bit)

1. Beispiel:

```
Timer_T0FRQ(10, ps_8)   // Rechtecksignal mit 10*1,085 µs = 10,85 µs
Periodendauer
```

2. Beispiel: gepulste Frequenzblöcke

```
int delval;
void main(void)
{
    delval=200;
    Timer_T0FRQ(10,2);

    while (1)
    {
        AbsDelay(delval);
        Timer_T0Stop();
        AbsDelay(delval);
        Timer_T0Start(2);
    }
}
```

Timer1 (16 Bit)

1. Beispiel: Frequenzerzeugung mit 125 * 4,34 µs = 1085µs Periode

```
Timer_T1FRQ(125,ps_64);
```

2. Beispiel: Frequenzerzeugung mit 10*1,085 µs =10,85 µs Periode und 2*1,085µs =2,17 µs Phasenverschiebung

```
Timer_T1FRQX(10,2,ps_8);
```

5.3.16.3 Pulsweitenmodulation

Es stehen zwei unabhängige Timer für die Pulsweitenmodulation zur Verfügung. *Timer_0* mit 8 Bit und *Timer_1* mit 16 Bit. Mit einer Pulsweitenmodulation lässt sich sehr einfach ein Digital-Analog-Wandler realisieren.

Timer0 (8 Bit)

Beispiel: Pulsweitenmodulation mit 138,9 µs Periode und 5,42 µs Pulsweite, geändert auf 10,84 µs Pulsweite

```
Timer_T0PWM(10,2);  //  Puls: 10*542,5 ns = 5,42 µs, Periode: 256*542,5 ns =
138,9 µs
Timer_T0PW(20);   //  Puls: 20*542,5 ns = 10,84 µs
```

Timer1 (16 Bit)

Beispiel: Pulsweitenmodulation mit 6,4 ms Periode und 1,28 ms Pulsweite Kanal A und 640 µs Pulsweite Kanal B

```
Timer_T1PWMX(10,20,10,ps_1024); //  Periode: 100*69,44 µs = 6,94 ms
                                //  PulsA: 20*69,44 µs = 1,389 ms
                                //  PulsB: 10*69,44 µs = 694,4 µs
```

5.3.16.4 Timerfunktionen

Es stehen zwei unabhängige Timer zur Verfügung. *Timer_0* mit 8 Bit und *Timer_1* mit 16 Bit. Die Timer verfügen über einen programmierbaren Vorteiler, siehe Tabelle. Mit dem Timer lässt sich eine Zeit festlegen, nach der ein Interrupt ausgelöst wird. In der Interruptroutine lassen sich dann bestimmte Verarbeitungsschritte ausführen.

Timer_T0Time (8 Bit)

Beispiel: Timer0: Ausgang mit einer Verzögerung von 6,94 ms (100x 69,44 µs, siehe Tabelle) einschalten

```
void Timer0_ISR(void)
{
    int  irqcnt;

    Port_WriteBit(0,1);
    Timer_T0Stop() ;                    // Timer0 anhalten
    irqcnt=Irq_GetCount(INT_TIM0COMP);
}
```

```
void main(void)
{
    Port_DataDirBit(0,0);                   // PortA.0 Ausgang
    Port_WriteBit(0,0);                     // PortA.0 Ausgang=0
    Irq_SetVect(INT_TIM0COMP,Timer0_ISR);  // Interrupt Service Routine
definieren
    Timer_T0Time(100,ps_1024);              // Zeit festlegen und Timer0
starten
    //  weiterer Programmablauf...
}
```

5.3.16.5 Puls & Periodenmessung

Mit dem *Timer_1* können Pulsweiten oder Signalperioden gemessen werden. Es wird mit Hilfe der Input Capture Funktion (spezielles Register des Controllers) die Zeit zwischen zwei Flanken gemessen. Diese Funktion nutzt den Capture-Interrupt (INT_TIM1CAPT). Der Puls wird zwischen einer steigenden und der nächsten fallenden Signalflanke gemessen. Die Periode wird zwischen zwei steigenden Signalflanken gemessen. Durch die Input Capture Funktion gehen Programmlaufzeiten nicht als Ungenauigkeit in das Messergebnis ein. Mit dem programmierbaren Vorteiler kann die Auflösung des *Timer_1* festgelegt werden. Vorteiler siehe **Tabelle**.

Beispiel: Pulsbreitenmessung 434 µs (100 x 4,34 µs, siehe Tabelle) einschalten

```
word PM_Wert;

void Timer1_ISR(void)
{
    int irqcnt;

    PM_Wert=Timer_T1GetPM(0);   // Pulsweite messen
    irqcnt=Irq_GetCount(INT_TIM1CAPT);
}

void main(void)
{
    byte n;

    Irq_SetVect(INT_TIM1CAPT,Timer1_ISR);  // Interrupt Service Routine
definieren
    Timer_T0PWM(100,ps_64);               // Pulsgenerator starten

 // die Messung beginnt hier
 // Output Timer0 OC0(PortB.3) verbinden mit ICP (input capture pin) (PortD.6)

    PM_Wert=0;
    Timer_T1PM(ps_64);      // Vorteiler für Messung festlegen

    while(PM_Wert==0);     // Pulsbreite oder Periode messen

    Msg_WriteHex(PM_Wert);  // Messwert ausgeben
}
```

5.3.16.6 Frequenzmessung

Zur direkten Messung einer Frequenz kann der Timer1(16Bit) verwendet werden. Es werden die Pulse innerhalb einer Sekunde gezählt und das Ergebnis ist dann in Herz. Die maximale Messfrequenz ist 64kHz und ergibt sich durch den 16Bit Zähler. Ein Beispiel für diese Art der Frequenzmessung findet man unter "Demo Programme/FreqMessung". Durch verkürzen der Messzeit lassen sich auch höhere Frequenzen messen. Das Ergebnis muß dann entsprechend umgerechnet werden.

5.3.16.7 Timer_T0CNT

Timer Funktionen

Syntax

```
void Timer_T0CNT(void);
```

Beschreibung

Diese Funktion initialisiert den Counter0. Der Counter0 wird bei einer positiven Signalflanke an dem Eingang T0 (PIN1) inkrementiert.

Parameter

Keine

5.3.16.8 Timer_T0Disable

Timer Funktionen

Syntax

```
void Timer_T0Disable(void);
```

Beschreibung

Die Funktion schaltet Timer 0 ab. Timerfunktionen belegen I/O Ports. Wird ein Timer nicht mehr benötigt und die Ports sollen als normale digitale I/Os verwendet werden, so muß die Timerfunktion abgeschaltet werden.

Parameter

Keine

5.3.16.9 Timer_T0FRQ

Timer Funktionen

Syntax

```
void Timer_T0FRQ(byte period,byte PS);
```

Beschreibung

Diese Funktion initialisiert den Timer0 mit dem angegebenen Vorteiler und Periodendauer, siehe Tabelle . Das Ausgangssignal erscheint an PortB.3 (PIN4). Die Frequenzerzeugung wird automatisch gestartet.

Parameter

period Periodendauer
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS_1 (1)	135,6 ns
PS_8 (2)	1,085 µs
PS_64 (3)	8,681 µs
PS_256 (4)	34,72 µs
PS_1024 (5)	138,9 µs

5.3.16.1 Timer_T0GetCNT

Timer Funktionen

Syntax

```
byte Timer_T0GetCNT(void);
```

Beschreibung

Der Wert des Counter0 wird gelesen. Erfolgte ein Überlauf, dann wird der Wert 0xFF übergeben.

Rückgabewert

der gemessene Zählerwert

5.3.16.1 Timer_T0PW

Timer Funktionen

Syntax

```
void Timer_T0PW(byte PW);
```

Beschreibung

Diese Funktion stellt eine neue Pulsweite für den Timer0 ein, ohne den Vorteiler zu verändern.

Parameter

PW Pulsweite

5.3.16.1.Timer_T0PWM

Timer Funktionen

Syntax

```
void Timer_T0PWM(byte PW,byte PS);
```

Beschreibung

Diese Funktion initialisiert den Timer0 mit dem angegebenen Vorteiler und Pulsweite, siehe Tabelle . Das Ausgangssignal erscheint an PortB.3 (PIN4).

Parameter

PW Pulsweite
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS_1 (1)	67,8 ns
PS_8 (2)	542,5 ns
PS_64 (3)	4,34 µs
PS_256 (4)	17,36 µs
PS_1024 (5)	69,44 µs

5.3.16.1.Timer_T0Start

Timer Funktionen

Syntax

```
void Timer_T0Start(byte prescaler);
```

Beschreibung

Die Frequenzerzeugung wird mit der vorherigen Einstellung gestartet. Der Vorteiler muß neu angegeben werden.

Parameter

prescaler Vorteiler (Tabelle prescaler)

5.3.16.1 Timer_T0Stop

Timer Funktionen

Syntax

```
void Timer_T0Stop(void);
```

Beschreibung

Die Frequenzerzeugung wird angehalten. Das Ausgangssignal kann 0 oder 1 sein, entsprechend dem letzten Zustand. Es wird nur der Takt für den Timer angehalten. Sonstige Einstellungen bleiben erhalten.

Parameter

Keine

5.3.16.1 Timer_T0Time

Timer Funktionen

Syntax

```
void Timer_T0Time(byte Time,byte PS);
```

Beschreibung

Diese Funktion initialisiert den Timer0 mit dem angegebenen Vorteiler und dem Wert (8 Bit) für die Zeit, siehe Tabelle . Ist der Wert erreicht, dann wird der Timer0 Interrupt (INT_TIM0COMP) ausgelöst.

Parameter

Time Zeitwert bei dem Interrupt ausgelöst wird
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS_1 (1)	67,8 ns
PS_8 (2)	542,5 ns
PS_64 (3)	4,34 µs
PS_256 (4)	17,36 µs
PS_1024 (5)	69,44 µs

5.3.16.1 Timer_T1CNT

Timer Funktionen

Syntax

```
void Timer_T1CNT(void);
```

Beschreibung

Diese Funktion initialisiert den Counter1. Der Counter1 wird bei einer positiven Signalflanke an dem Eingang T1 (PIN2) inkrementiert.

Parameter

Keine

5.3.16.1 Timer_T1CNT_Int

Timer Funktionen

Syntax

```
void Timer_T1CNT_Int(word limit);
```

Beschreibung

Diese Funktion initialisiert den Counter1. Der Counter1 wird bei einer positiven Signalflanke an dem Eingang T1 (PIN2) inkrementiert. Wenn das Limit erreicht ist wird ein Interrupt ausgelöst. Die entsprechende Interrupt_Service_Routine muß vorher definiert sein.

Parameter

limit

5.3.16.1 Timer_T1Disable

Timer Funktionen

Syntax

```
void Timer_T1Disable(void);
```

Beschreibung

Die Funktion schaltet Timer 1 ab. Timerfunktionen belegen I/O Ports. Wird ein Timer nicht mehr benötigt und die Ports sollen als normale digitale I/Os verwendet werden, so muß die Timerfunktion abgeschaltet werden.

Parameter

Keine

5.3.16.1 Timer_T1FRQ

Timer Funktionen

Syntax

```
void Timer_T1FRQ(word period,byte PS);
```

Beschreibung

Diese Funktion initialisiert den Timer1 mit dem angegebenen Vorteiler und Periodendauer, siehe Tabelle . Das Ausgangssignal erscheint an PortD.5 (PIN19). Die Frequenzerzeugung wird automatisch gestartet.

Parameter

period Periodendauer
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS_1 (1)	135,6 ns
PS_8 (2)	1,085 µs
PS_64 (3)	8,681 µs
PS_256 (4)	34,72 µs
PS_1024 (5)	138,9 µs

5.3.16.2 Timer_T1FRQX

Timer Funktionen

Syntax

```
void Timer_T1FRQX(word period,word skew,byte PS);
```

Beschreibung

Diese Funktion initialisiert den Timer1 mit dem angegebenen Vorteiler Periodendauer und Phasenverschiebung der beiden Ausgangssignale, siehe Tabelle . Die Ausgangssignale erscheinen an PortD.4 (PIN18) und PortD.5 (PIN19). Die Frequenzerzeugung wird automatisch gestartet. Der Wert für die Phasenverschiebung muß kleiner sein als die halbe Periode.

Parameter

period Periodendauer
skew Phasenverschiebung

PS Vorteiler (Tabelle prescaler)

5.3.16.2 Timer_T1GetCNT

Timer Funktionen

Syntax

```
word Timer_T1GetCNT(void);
```

Beschreibung

Der Wert des Counter1 wird gelesen. Erfolgte ein Überlauf, dann wird der Wert 0xFFFF übergeben.

Rückgabewert

der gemessene Zählerwert

5.3.16.2 Timer_T1GetPM

Timer Funktionen

Syntax

```
word Timer_T1GetPM(byte Mode);
```

Beschreibung

Diese Funktion legt fest ob eine Pulsbreiten- oder Periodenmessung durchgeführt werden soll, und liefert das Messergebnis zurück.

Parameter

Mode

0 Pulsweitenmessung
1 Periodenmessung

Rückgabewert

Ergebnis der Messung

5.3.16.2 Timer_T1PM

Timer Funktionen

Syntax

```
void Timer_T1PM(byte PS);
```

Beschreibung

Diese Funktion initialisiert den Timer_1 für die Messung und setzt den Vorteiler.

Parameter

PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS_1 (1)	67,8 ns
PS_8 (2)	542,5 ns
PS_64 (3)	4,34 µs
PS_256 (4)	17,36 µs
PS_1024 (5)	69,44 µs

5.3.16.2 Timer_T1PWA

Timer Funktionen

Syntax

```
void Timer_T1PWA(word PW0);
```

Beschreibung

Diese Funktion stellt eine neue Pulsweite (Kanal_A) für den Timer1 ein, ohne den Vorteiler zu verändern.

Parameter

PW0 Pulsweite

5.3.16.2 Timer_T1PWB

Timer Funktionen

Syntax

```
void Timer_T1PWB(word PW1);
```

Beschreibung

Diese Funktion stellt eine neue Pulsweite (Kanal_B) für den Timer1 ein, ohne den Vorteiler zu verändern.

Parameter

PW1 Pulsweite

5.3.16.2 Timer_T1PWM

Timer Funktionen

Syntax

```
void Timer_T1PWM(word period,word PW0,byte PS);
```

Beschreibung

Diese Funktion initialisiert den Timer1 mit dem angegebenen Vorteiler, Pulsweite und Periodendauer, siehe Tabelle . Das Ausgangssignal erscheint an PortD.5 (PIN19).

Parameter

period Periodendauer
PW0 Pulsweite
PS Vorteiler (Tabelle prescaler)

5.3.16.2 Timer_T1PWMX

Timer Funktionen

Syntax

```
void Timer_T1PWMX(word period,word PW0,word PW1,byte PS);
```

Beschreibung

Diese Funktion initialisiert den Timer1 mit dem angegebenen Vorteiler, Pulsweite für Kanal A und B und Periodendauer, siehe Tabelle . Die Ausgangssignale erscheinen an PortD.4 (PIN18) und PortD.5 (PIN19).

Parameter

period Periodendauer
PW0 Pulsweite Kanal A
PW1 Pulsweite Kanal B
PS Vorteiler (Tabelle prescaler)

5.3.16.2 Timer_T1Stop

Timer Funktionen

Syntax

```
void Timer_T1Stop(void);
```

Beschreibung

Die Frequenzerzeugung wird angehalten. Das Ausgangssignal kann 0 oder 1 sein, entsprechend dem letzten Zustand. Es wird nur der Takt für den Timer angehalten. Sonstige Einstellungen bleiben erhalten.

Parameter

Keine

5.3.16.2 Timer_T1Time

Timer Funktionen

Syntax

```
void Timer_T1Time(word Time,byte PS);
```

Beschreibung

Diese Funktion initialisiert den Timer1 mit dem angegebenen Vorteiler und dem Wert (16 Bit) für die Zeit, siehe Tabelle . Ist der Wert erreicht, dann wird der Timer1- Interrupt (INT_TIM1COMP) ausgelöst.

Parameter

Time Zeitwert bei dem Interrupt ausgelöst wird
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS_1 (1)	67,8 ns
PS_8 (2)	542,5 ns
PS_64 (3)	4,34 µs
PS_256 (4)	17,36 µs
PS_1024 (5)	69,44 µs

5.3.16.3 Timer_T1Start

Timer Funktionen

Syntax

```
void Timer_T1Start(byte prescaler);
```

Beschreibung

Die Frequenzerzeugung wird mit der vorherigen Einstellung gestartet. Der Vorteiler muß neu angegeben werden.

Parameter

prescaler Vorteiler (Tabelle prescaler)

Kapitel 6

6 Anhang

6.1 FAQ

Probleme

1. Es existiert keine USB Verbindung zum Application Board.

- Ist der FTDI USB Treiber auf dem PC geladen? Oder erscheint vielleicht beim Einstecken des USB Steckers ein "unbekanntes Gerät" im Hardware Manager?
- Ist in Optionen->IDE->Schnittstellen der richtige Kommunikationsport eingestellt?
- Wird eine Windowsversion vor Windows 98 SE ("*Second Edition*") benutzt? Die USB Treiber von Microsoft funktionieren erst ab Win98SE zuverlässig mit USB Geräten.
- Werden die Ports B.4-B.7 oder A.6-A.7 versehentlich in der Software benutzt? (siehe Pinzuordnung). Sind die Jumper auf dem Application Board bei diesen Ports auch gesetzt?

2. Die serielle Schnittstelle gibt keine Zeichen aus oder empfängt keine Zeichen.

- Werden die Ports D.0-D.1 versehentlich in der Software benutzt? (siehe Pinzuordnung). Sind die Jumper auf dem Application Board bei diesen Ports auch gesetzt?

3. Das Application Board reagiert nicht auf Kommandos wenn es seriell angeschlossen ist.

- Um den Bootloader in den seriellen Modus zu bekommen muß beim Einschalten des Application Boards der Taster SW1 gedrückt werden. (Jumper für SW1 beachten). Für den seriellen Mode kann PortD.2 (SW1) auch fest auf GND gelegt werden.

4. Es wurde die Tastenbelegung des Editors "**xyz**" eingestellt aber manche Tastaturbefehle funktionieren nicht.

- Die Möglichkeit die Tastenbelegung eines bestimmten Editors in der IDE einzuschalten ist nur eine Näherung. Manchmal ist es zu aufwendig die entsprechenden Funktionen des "fremden" Editors zu unterstützen, ein anderes Mal können Tastaturbefehle mit den Keyboard Shortcuts in der IDE kollidieren.

5. Die Rechtschreibprüfung funktioniert nicht.

- Ist die Rechtschreibprüfung in Optionen->Editor eingeschaltet?
- Die Rechtschreibprüfung zeigt nur Schreibfehler in den Kommentaren an. Die Prüfung für andere Bereiche wäre sinnlos.

Sachverzeichnis

- - -

- # -

- + -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- K -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- U -

- V -

- W -

- Z -