C-Control Pro IDE

Inhalt

Kapitel 1

1 Wichtige Hinweise

Dieses Kapitel behandelt wichtige Informationen zur Gewährleistung, und zum Support und Betrieb der C-Control-Pro Hardware und Software.

1.1 Einleitung

Die C-Control Pro Systeme basieren auf dem Atmel AVR32 und der Atmel Mega Serie (Mega 32, Mega 128, AT90CAN). Diese Mikrocontroller werden 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 (2000/XP/Vista/Win7/Win8) 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 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 SE
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). 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:

ANTLR 2.73	http://www.antlr.org
Inno Setup 5.5.2	http://www.jrsoftware.org
GPP (Generic Preprocessor)	http://www.nothingisreal.com/gpp
avra-1.2.3a Assembler	http://avra.sourceforge.net/

Gemäß den Bestimmungen der "LESSER GPL" (www.gnu.org/copyleft/lesser) wird bei der Installation der IDE auch der Original Sourcecode des avra Assemblers, 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.

1.8 Demo Programme

Die aktuellen Demoprogramme sind im Verzeichnis "C:\Dokumente und Einstellungen\Alle Benutzer \Gemeinsame Dokumente\C-Control Pro Demos" (XP und früher) bzw. "C:\Benutzer\Öffentlich\Öffentliche Dokumente\C-Control Pro Demos" (Vista und später) zu finden. Die aktuellen Demos sind im Ordner "Demos Ver 2.31" gespeichert. Die alten Demoprogramme werden damit nicht überschrieben.

➔ Es gibt in der IDE unter dem Hilfe Menü die Funktion Demo Programme, die ein Explorer Fenster an der Stelle öffnet, an der die Demo Programme gespeichert sind. Auch kann mit Öffne Demos direkt aus dem Projekt Menü ein Demoprojekt geöffnet werden.

1.9 Historie

- Version 2.31 vom 20.09.2013

neue Features
- AVR32Bit Unterstützung
- Ethernet Support (AVR32Bit)
- Webserver (AVR32Bit)
- Tab Interface für Editor
- neue Kommunikationsroutinen
- Direkter Zugriff auf COM Port über Toolbar
- Ein- und Ausschalten des COM Ports in Toolbar

Fehlerkorrekturen
- Handbuch Korrekturen
- Teilweise fehlerhafte Inkrementierung der Clock Variablen im Interrupt Kontext
- Fehler in der Typerkennung von Konstanten behoben
- Fehler in Onewire_Read behoben
- falsche Definitionen PORT_ON und PORT_OFF korrigiert

- Version 2.13 vom 04.04.2011

neue Features
- Alle Demo Programme überarbeitet

Fehlerkorrekturen
- Dokumentations Korrekturen
- Fehlerabfrage im Linker verbessert
- Registernutzung im Mega32 Interpreter korrigiert
- Felder in Projektoptionen werden jetzt immer initialisiert
- Fehler beim Setzen von PIN Codes behoben
- Ausdrücke wie "a[fun(2)]=b" funktionieren wieder

- Version 2.12 vom 6.01.2011

neue Features
- 32-Bit Integer (nur Mega128)
- neues Zeitscheiben Multithreading
- #thread Parameter in Source Code

- SD card Unterstützung
- Support des C-Control Pro Mega128 CAN Modul
- CAN-BUS Bibliothek
- Direkter Zugriff auf Flash-Arrays
- Array Tooltips im Debugger
- IDE Style änderbar
- Vista und Win7 Theme Support
- Übertragung bei Programmstart einschaltbar
- erhöhte serielle Geschwindigkeit bei Modul Kommunikation
- VT100 Emulation für Terminal
- rand(), srand() Random Funktionen

Fehlerkorrekturen

- Dokumentations Korrekturen
- funktionierende floats in Tabellen
- negative Zahlen in Tabellen korrigiert
- Klammerfehler in konstanten Ausdrücken behoben
- Funktionsaufrufe in return Anweisungen korrigiert
- "__DEBUG__" ist nun korrekt "DEBUG"
- "#pragma Warn" heißt nun korrekt "'#pragma Warning"
- Editor Undo nach speichern arbeitet jetzt richtig
- Problem in Servo-Routinen korrigiert
- Fehler in Serial_IRQ_Info behoben
- Schwachstelle bei serieller Übertragung beseitigt
- externes Interrupt Acknowledge jetzt in richtiger Reihenfolge
- Übersetzungsfehler korrigiert
- falsche Obergrenze bei einigen TimerXTime() Funktionen
- Löschen aller Breakpoints funktioniert jetzt immer
- Problem beim Überschreiten der 64kb Grenze behoben
- Programm im Debugger kann jenseits der 64kb Grenze gestoppt werden
- Problem in round() korrigiert
- Anomalie in BASIC For-Loop korrigiert

Version 2.01 vom 27.06.2009

neue Features

- Suchen Funktion dem Editor Popupmenü hinzugefügt

Fehlerkorrekturen

- Dokumentations Korrekturen
- Fehler bei "Unbenutzten Code erkennen" korrigiert
- Überschreiten der 64kb Grenze bei internen Compiler Strukturen läuft wieder
- Fehler beim Aufruf in Werkzeugmenü behoben
- Übersetzungsfehler im Suchen Dialog
- Zeilenoffset bei Projekt Suchen Dialog
- Timeout in I2C Routinen
- Fehlermeldung "...tbSetRowCount:new count too small"

Version 2.00 vom 14.05.2009

neue Features

- Assembler Support
- Verbesserte Suchfunktionen im Editor

- Neue konfigurierbare Oberfläche in der IDE
- Todo Liste
- Compiler Warnungen abschaltbar
- Programm Transfer nur mit Bytecode ohne Projekt
- erweiterte Programminfo
- Schneller Transfer, wenn Interpreter schon übertragen
- Verbesserte Vervollständigung von Befehlswörtern und Namen der Bibliotheksfunktionen
- Funktions Parameter Hilfe
- Optimizer um nicht benutzten Code zu entfernen
- Peephole Optimizer
- Unterstützung von vordefinierten Arrays im Flash Speicher
- Array Grenzen Check zur Laufzeit
- Optimierte Array Zugriffe
- exaktere Überprüfung von konstanten Array Indizes
- Aufruf von Funktionen mit Stringkonstanten
- Binärzahlen Definition mit 0b....
- Addition und Subtraktion bei Zeigern
- Optimierung der Port OUT, PIN und DDR Zugriffe
- Direkte Atmel Register Zugriffe
- Formatierte Ausgabe mit Str_Printf()
- Konvertierungsroutinen um ASCII Zeichen in numerische Werte zu wandeln
- ++/-- für BASIC
- Funktionen um Ports zu toggeln
- RC5 Sende- und Empfangsroutinen
- Software Uhr (Zeit & Datum) mit Quarzkorrektur Faktor
- Servo Routinen
- mathematische Rundungsfunktion
- Atmel Mega Sleep Funktion

Fehlerkorrekturen

- Initialisierung Timer_T0FRQ korrigiert
- Initialisierung Timer_T0PWM korrigiert
- Initialisierung Timer_T1FRQ korrigiert
- Initialisierung Timer_T1FRQX korrigiert
- Initialisierung Timer_T1PWM korrigiert
- Initialisierung Timer_T1PWMX korrigiert
- Initialisierung Timer_T1PWMXY korrigiert
- Initialisierung Timer_T3FRQ korrigiert
- Refresh für Array Fenster korrigiert
- Desktop zurücksetzen korrigiert
- Bug bei Modul zurücksetzen korrigiert
- Bug bei Debugdateien >30000 Bytes korrigiert
- Fehler bei bedingter Bewertung in CompactC behoben
- Fehler in Timer_Disable() behoben

Version 1.72 vom 22.10.2008

neue Features

- SPI Funktionen hinzugefügt
- RP6 AutoConnect

Fehlerkorrekturen

- serielle Übertragungsqualität verbessert

Version 1.71 vom 25.06.2008

neue Features
- neuer Editor in der IDE
- Editor hat Funktionsnamen Übersicht
- Code folding
- Internes Terminalprogramm
- Werkzeugmenü mit Optional erweiterbarer Toolliste
- Syntaxhervorhebung aller Standard Bibliotheksaufrufe
- Konfiguration der Syntaxhervorhebung
- Erweiterung von Select .. Case in BASIC
- Groß-Kleinschreibung wird bei Befehlswörtern und Namen der Bibliotheksfunktionen automatisch korrigiert
- Einfache automatische Vervollständigung von Befehlswörtern und Namen der Bibliotheksfunktionen
- OneWire Bibliotheksfunktionen
- Auskommentieren von Blöcken in Basic mit /* , */
- Neue FTDI Treiber

Fehlerkorrekturen
- globale For-Schleifenzähler Variablen in BASIC arbeiten nun korrekt
- char Variablen arbeiten jetzt korrekt bei negativen Zahlen
- "u" hinter Integerzahl definiert nun korrekt vorzeichenlose Zahl
- Projektnamen können nun auch Sonderzeichen enthalten
- Thread_Wait() arbeitet jetzt korrekt mit dem thread Parameter
- return Befehl in CompactC ohne Rückgabeparameter arbeitete fehlerhaft
- Vertauschte Fehlermeldungen bei Funktionsaufrufen mit Zeigern
- Korrekte Fehlermeldung bei Zuweisung, wenn Funktionsaufruf keinen Rückgabewert hat
- Geschachtelte switch/Select Anweisungen funktionieren jetzt
- Sehr lange switch/Select Anweisungen funktionieren jetzt
- Bessere Fehlerbehandlung wenn selektierter COM Port schon benutzt
- Kein Absturz mehr, wenn über USB oder COM Port aufgrund fehlerhafter Übertragung große Datenmengen empfangen werden
- "Exit" bei BASIC in For-Loop funktioniert jetzt.
- Compilerfehler bei Deklarationen von Array Variablen behoben

Version 1.63 vom 21.12.2007

Fehlerkorrekturen
- Dokumentations Änderungen

Version 1.62 vom 08.12.2007

neue Features
- Vista Kompatibilität

Fehlerkorrekturen
- Eckige Klammern funktionieren
- Der Compiler stürzt nicht mehr ab, wenn Variablennamen nicht stimmen
- Der Compiler gibt einen korrekten Syntaxfehler, wenn mehrere Klammerebenen geöffnet sind, und ein Operand fehlt
- "Exit" funktionierte in BASIC For-Next Schleifen nicht immer korrekt

- Man konnte nur 16mal das Array Fenster öffnen, auch wenn eines vorher geschlossen wurde
- Aus dem Text "Compiler" unter Optionen wurde "Compiler Voreinstellungen"

Version 1.60 vom 03.04.2007

neue Features
- englische Sprache in der IDE - umschaltbar zur Laufzeit
- englische Sprache in den Compiler Meldungen
- englische Version von Hilfedateien und Handbuch
- drucken von Programmdateien aus der IDE
- Druckvorschau von Programmdateien
- Thread_Wait() um thread Parameter erweitert
- ADC_Set() ist performanter
- In den seriellen Routinen kann der DoubleClock Modus aktiviert werden

Fehlerkorrekturen
- ExtIntEnable funktionierte nur bei den IRQs 0 und 4 korrekt
- Serial_Init() und Serial_Init_IRQ() nahmen als divider nur ein byte statt ein word
- EEPROM_WriteFloat und EEPROM_ReadFloat() arbeiteten fehlerhaft
- Thread_Kill() arbeitete im Hauptthread fehlerhaft
- Lese Zugriffe auf global definierte floating-point arrays waren fehlerhaft
- Die 2. serielle Schnittstelle auf dem Mega128 arbeitete nicht korrekt
- EEPROM Schreibzugriffe mit zu hohen Adressen konnten reservierten Bereich überschreiben
- Mit einer sehr kleinen Wahrscheinlichkeit konnten LCD Zugriffe fehlerhafte Zeichen auf das LCD Display schreiben

Version 1.50 vom 08.11.2005

neue Features
- IDE Unterstützung für Mega128
- verbesserter Cache Algorithmus bei Zugriff der IDE auf Laufzeitdaten im Debugger
- neue Bibliotheksroutinen für Timer 3 (Mega128)
- Programme nutzen den erweiterten (>64kb) Adressraum (Mega128)
- Unterstützung des externen 64kb SRAM
- externe Interrupts 3-7 werden unterstützt (Mega128)
- Routinen für 2. serielle Schnittstelle (Mega128)
- mathematische Funktionen (Mega128)
- Anzeige der Speichergröße bei Start des Interpreters
- interner RAM Check zur Erkennung wenn globale Variablen zu groß für Hauptspeicher
- interner RAM Check zur Erkennung wenn Thread Konfiguration zu groß für Hauptspeicher
- Laufzeitüberprüfung ob Stackgrenzen verletzt werden
- Quelldateien können in der Projekthierarchie nach oben und unten bewegt werden
- Warnung bei Zuweisung von zu langen Strings
- der Compiler erzeugt auf Wunsch eine Map-Datei, die die Größe aller Programmvariablen beschreibt
- neues Adressmodell für globale Variablen (das gleiche Programm läuft bei verschiedenen RAM Größen)
- Interruptroutinen für serielle Schnittstelle (bis zu 256 Byte Empfangspuffer / 256 Byte Sendepuffer)
- festverdrahtete IRQ Routinen um eine Periodenmessung kleiner Zeiträume zu ermöglichen
- Rekursionen sind nun uneingeschränkt nutzbar
- beliebig große Arrays können im Debugger in eigenem Fenster angezeigt werden
- Strings (character arrays) werden nun als Tooltip im Debugger gezeigt

- SPI kann ausgeschaltet werden um die Pins als I/O zu nutzen
- Die serielle Schnittstelle kann ausgeschaltet werden um die Pins als I/O zu nutzen
- Der Hexwert wird nun zusätzlich als Tooltip im Debugger angezeigt
- neue Funktion Thread_MemFree()
- Zusätzliche EEPROM Routinen für Wort- und Fließkommazugriff
- Zeitmessung mit Timer_TickCount()
- #pragma Kommandos um Fehler oder Warnungen zu erzeugen
- vordefinierte Symbol im Preprozessor: __DATE__, __TIME__ __FILE__, __FUNCTION__, __LINE__
- Versionsnummer im Splashscreen
- erweiterte Dokumentation
- interaktive Grafik bei "Jumper Application Board" in Hilfe Datei
- neue Demoprogramme
- Ctrl-F1 startet Kontexthilfe

Fehlerkorrekturen
- es wird nun ein Fehler erzeugt, wenn eine return Anweisung am Ende einer Funktion fehlt
- Breakpoint Markierungen wurden nicht immer gelöscht
- Grenzen bei EEPROM Zugriff genauer überprüft (interner Überlauf abgefangen)
- Einzelschritt kann im Debugger nicht mehr zu früh den nächsten Befehl absetzen

Version 1.39 vom 09.06.2005

neue Features
- BASIC Unterstützung
- CompactC und BASIC können in einem Projekt gemischt werden
- erweiterte Dokumentation
- Schleifenoptimierung für For - Next in BASIC
- ThreadInfo() Funktion
- neue Demoprogramme

Fehlerkorrekturen
- Bei Umlauten stürzt Compiler nicht mehr ab
- interner Bytecode Befehl StoreRel32XT korrigiert
- Offset in Stringtabelle verbessert

Version 1.28 vom 26.04.2005

- Initialversion

Kapitel 2

2 Hardware

2.1 Mega Serie

Es wird die Hardware vorgestellt, die bei der C-Control Pro Mega Serie zur Anwendung kommt. Beschrieben werden die Module von C-Control Pro Mega32, C-Control Pro Mega128 und Control Pro Mega128 CAN. Weitere Abschnitte erklären Aufbau und Funktion der zugehörigen Application Boards, und die mitgelieferten LCD Module, sowie Tastatur.

2.1.1 Installation

Dieses Kapitel beschreibt die Installation der C-Control Pro Mega Unit auf dem Applicationboard, und die Installation der Software und USB Treiber.

2.1.1.1 Software

Die aktuelle Entwicklungssoftware, Beispielprogramme sowie das Handbuch und nützliche Informationen finden Sie unter: www.c-control.de Das Handbuch gibt es auch als Hilfedatei in der Entwicklungsumgebung der C-Control PRO IDE und als PDF im Installationsordner der C-Control Pro im "Manual" Verzeichnis.

Direkter IDE Download Link: www.c-control-pro.de/updates/C-ControlSetup.exe

➔ 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. Am 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.

2.1.1.2 Hardware

Wichtiger Hinweis zum Ein-Ausbau eines Mega Moduls

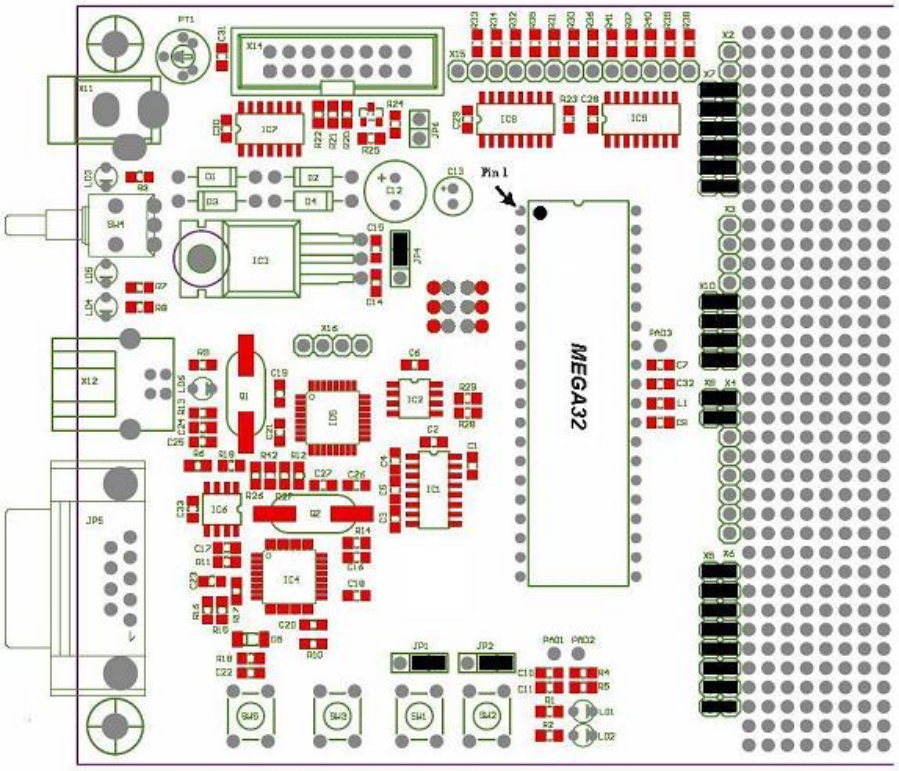
Für die Verbindung zwischen dem Modul und dem Application Board sind hochwertige Steckverbinder verwendet worden, die eine gute Kontaktierung sicherstellen. Der Ein- und Ausbau eines Moduls darf nur bei ausgeschalteter Versorgungsspannung (spannungsfrei) durchgeführt werden, da sonst Zerstörungen auf dem Application Board bzw. Modul auftreten können. Durch die Kontaktanzahl (40/64 Pin) ist eine erhebliche Kraft beim Ein- und Ausbau des Moduls erforderlich. Beim Einbau ist darauf zu achten, daß das Modul gleichmäßig, d.h. nicht verkantet in die Fassung gedrückt wird. Legen sie das Application Board dazu auf eine ebene Unterfläche.

Mega128 Modul

Das Modul Mega128 (CAN) hat die Steckverbinder so angeordnet, daß ein falscher Einbau des Moduls nicht möglich ist. Der Ausbau erfolgt durch vorsichtiges Heraushebeln des Moduls mit einem geeigneten Werkzeug aus der Fassung. Um ein Verbiegen der Anschlüsse zu vermeiden sollte das Hebeln von mehreren Seiten erfolgen.

Einbaurichtung Mega32 Modul

Das Modul Mega32 in der richtigen Orientierung montieren. Dazu die Pin 1 Markierung beachten. Die Beschriftung des Moduls zeigt dann zu den Bedienungselementen auf dem Application Board.



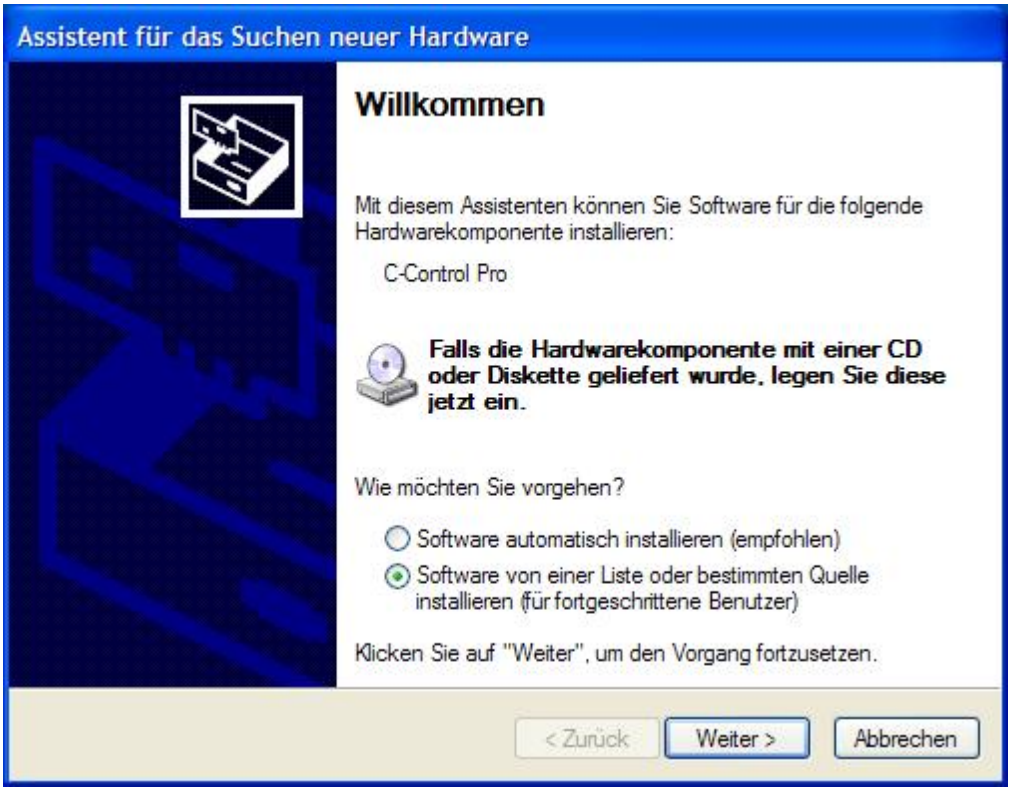
2.1.1.3 USB und seriell

Installation des USB Treibers

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.

➔ Treiber und Software der C-Control Pro unterstützen kein Windows Betriebssystem vor Windows 2000.

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.

Nun kann man in der PC-Software die Schnittstelle selektieren.

➔ Der Treiber von FTDI unterstützt 32 Bit und 64 Bit Windows Systeme. Die spezifischen Treiber sind in den Unterverzeichnissen "USB Driver\CCPro Mega USB Driver".

Serieller Anschluss

Aufgrund der langsamen Übertragungsgeschwindigkeit der seriellen Schnittstelle ist ein USB-Interface 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 die Schnittstelle selektieren.

2.1.2 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 (Port **M32**:D.2 bzw. **M128**:E.4 auf low) 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.

➔ Da der SW1 beim Starten des Moduls den seriellen Bootloader einleitet, sollte auf dem Port **M32**:D.2 bzw. **M128**:E.4 beim Einschalten der Applikation kein Signal sein. Man kann diese Ports ja auch als Ausgänge benutzen.

SPI Abschaltung (nur Mega128)

Ein Signal auf der SPI Schnittstelle beim Einschalten des Moduls kann die USB Kommunikation aktivieren. Um dies zu unterbinden kann man PortG.4 (LED 2) beim Einschalten auf low setzen. Dann wird die SPI Schnittstelle nicht konfiguriert. Die SPI Schnittstelle kann auch später vom Interpreter manuell mit SPI_Disable() abgeschaltet werden.

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.

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. Das heißt, wird das Modul in eine Hardware Applikation eingebaut, so reicht ein Anlegen der Betriebsspannung, um das Anwenderprogramm automatisch zu starten.

➔ Ein Signal auf **Mega32**:INT_0 bzw. **Mega128**:INT_4 beim einschalten des C-Control Pro Moduls kann das Autostartverhalten stören. Nach der Pinzuordnung von M32 und M128 liegt der INT_0 (bzw. INT_4) 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.

2.1.3 Mega32 Modul

Modulspeicher

In dem C-Control Pro Modul sind 32kB FLASH, 1kB EEPROM 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.

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, das 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).

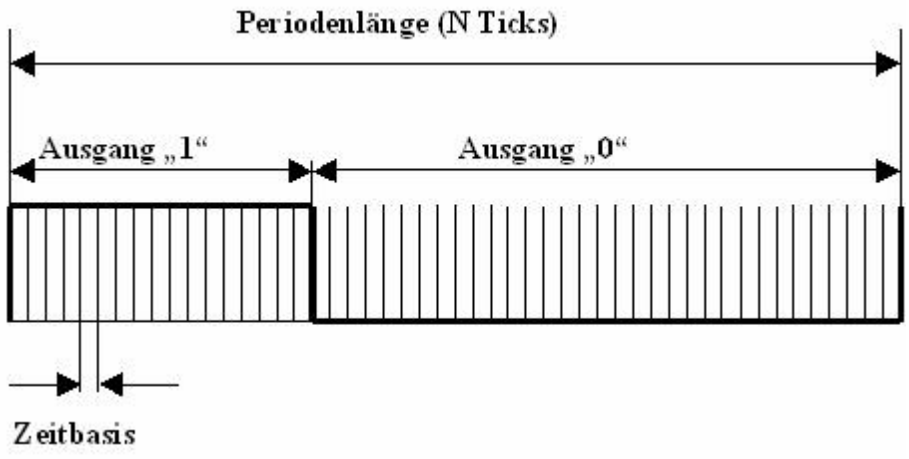
➔ Den maximal zulässigen Laststrom für einen einzelnen Port und für alle Ports in der Summe be-

achten. 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 von M32 und M128 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

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

2.1.3.1 CPU

Mega32 Übersicht

Der Mikrocontroller ATmega32 stammt aus der AVR-Familie von ATMEL. Es handelt sich um einen low-power Mikrocontroller 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

2.1.3.2 Pinzuordnung

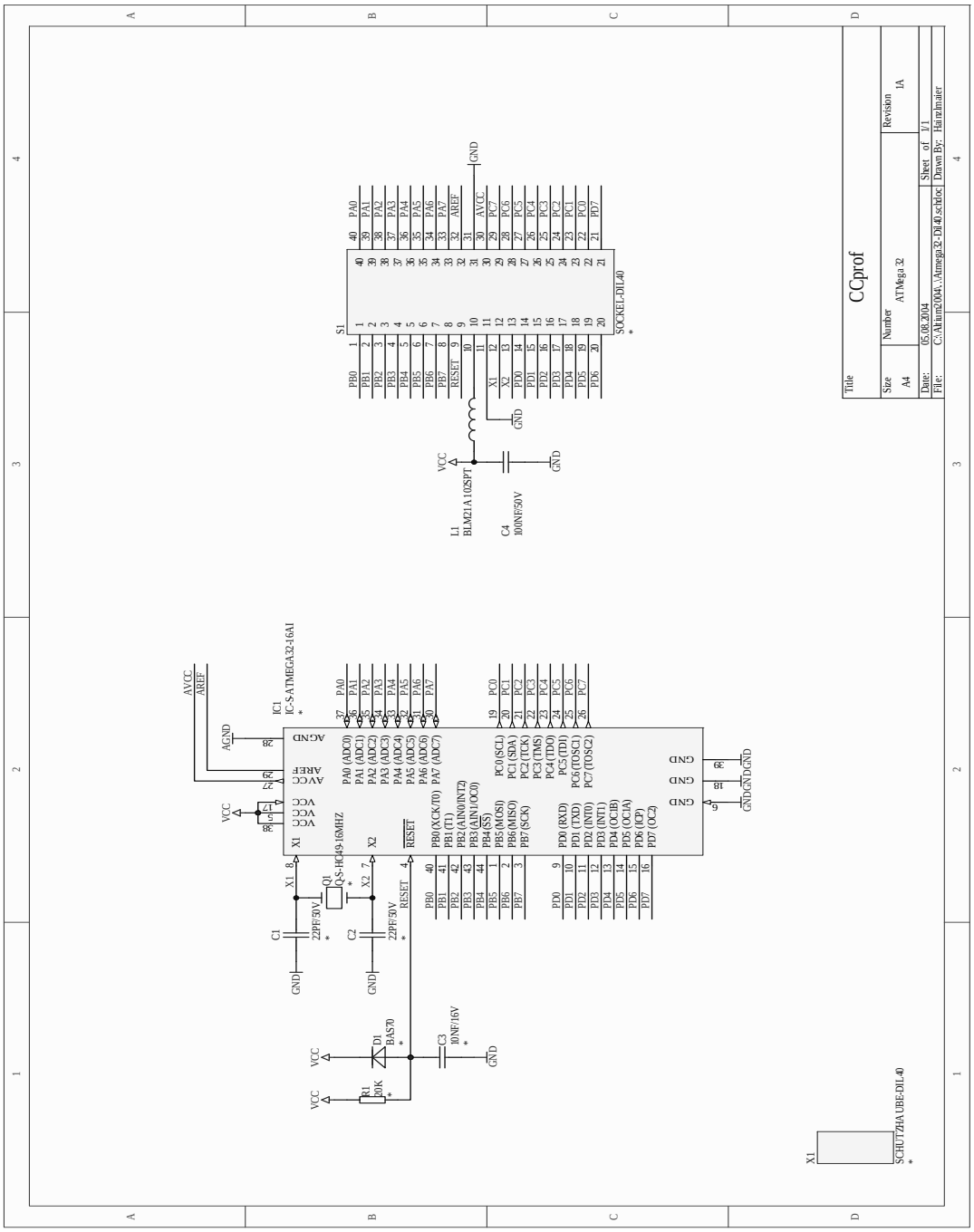
PortA bis PortD werden für direkte Pin-Funktionen (z.B. Port_WriteBit) von 0 bis 31 gezählt, siehe "PortBit".

Pinbelegung für Application Board Mega32

M32 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 für Puls/Periodenmessung
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	PC7	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

2.1.3.3 Schaltplan

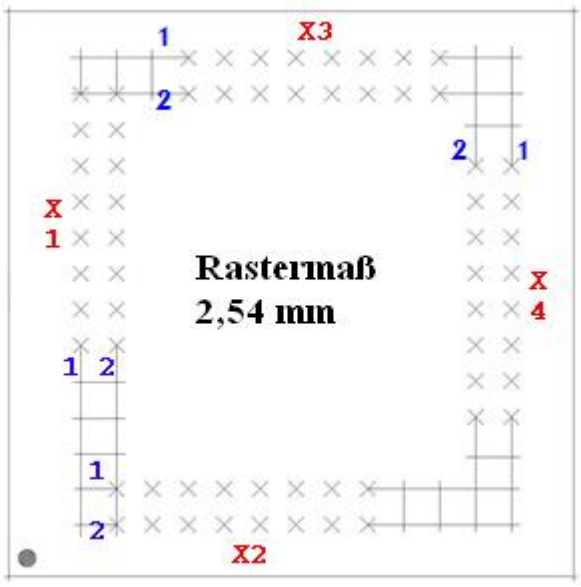


2.1.4 Mega128 Modul

Pinlayout des Moduls

Das Mega128 Modul wird auf 4 doppelreihigen (2x8) Vierkantstiften ausgeliefert. Für eine Hardware

Applikation müssen die entsprechenden Buchsenleisten im folgenden Rasterformat angeordnet werden:



In der Grafik sieht man die Buchsenleisten X1-X4 und dann die ersten beiden Pins der Buchsenleiste. Pin 1 von Leiste X1 entspricht dem Anschluß X1_1 (siehe Mega128 Pinzuordnung).

Modulspeicher

In dem C-Control Pro 128 Modul sind 128kB FLASH, 4kB EEPROM und 4kB SRAM integriert. Auf dem Application Board befindet sich ein zusätzliches EEPROM mit einer Speichertiefe von 8kB und ein SRAM mit 64kB Speichertiefe. Das EEPROM ist über eine I2C Schnittstelle ansprechbar.

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 Con-

trollers 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 (X2_3) 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, PortE, PortF, PortG)

Das C-Control Pro Modul verfügt über 6 digitale Ports mit je 8 Pins und einem digitalen Port mit 5 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.

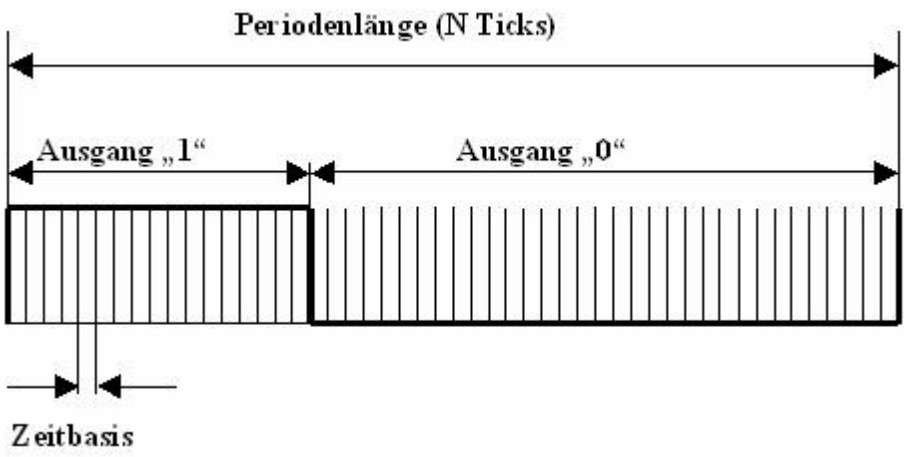
➔ Niemals zwei Ports direkt zusammenschalten, die gleichzeitig als Ausgang arbeiten sollen!

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 von M32 und M128 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 drei Timer für PLM zur Verfügung. *Timer_0* mit 8 bit und *Timer_1* und *Timer_3* mit jeweils 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, Timer_1* und *Timer_3* 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

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.	40 mm x 40mm x 8 mm
Masse	ca. 90g
Pinraster	2,54mm
Pinanzahl (zweireihig)	64

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

2.1.4.1 CPU

Mega128 Übersicht

Der Mikrocontroller ATmega128 stammt aus der AVR-Familie von ATMEL. Es handelt sich um einen low-power Mikrocontroller mit Advanced RISC Architecture. Hier folgt eine kurze Zusammenstellung der Hardwareressourcen:

- **133 Powerful Instructions – Most Single Clock Cycle Execution**
- **32 x 8 General Purpose Working Registers + Peripheral Control Registers**
- **Fully Static Operation**
- **Up to 16 MIPS Throughput at 16 MHz**
- **On-chip 2-cycle Multiplier**

- **Nonvolatile Program and Data Memories**
  128K Bytes of In-System Reprogrammable Flash
  Endurance: 10,000 Write/Erase Cycles
  Optional Boot Code Section with Independent Lock Bits
  In-System Programming by On-chip Boot Program

- **True Read-While-Write Operation**
  4K Bytes EEPROM
  Endurance: 100,000 Write/Erase Cycles
  4K Bytes Internal SRAM
  Up to 64K Bytes Optional External Memory Space
  Programming Lock for Software Security
  SPI Interface for In-System Programming

- **JTAG (IEEE std. 1149.1 Compliant) Interface**
  Boundary-scan Capabilities According to the JTAG Standard
  Extensive On-chip Debug Support
  Programming of Flash, EEPROM, Fuses and Lock Bits through the JTAG Interface

- **Peripheral Features**

Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
Two Expanded 16-bit Timer/Counters with Separate Prescaler, Compare Mode and Capture Mode
Real Time Counter with Separate Oscillator
Two 8-bit PWM Channels
6 PWM Channels with Programmable Resolution from 2 to 16 Bits
Output Compare Modulator
8-channel, 10-bit ADC
8 Single-ended Channels
7 Differential Channels
2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
Byte-oriented Two-wire Serial Interface
Dual Programmable Serial USARTs
Master/Slave SPI Serial Interface
Programmable Watchdog Timer with On-chip Oscillator
On-chip Analog Comparator

- Special Microcontroller Features
  Power-on Reset and Programmable Brown-out Detection
  Internal Calibrated RC Oscillator
  External and Internal Interrupt Sources
  Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
  Software Selectable Clock Frequency
  ATmega103 Compatibility Mode Selected by a Fuse
  Global Pull-up Disable

- I/O and Packages
  53 Programmable I/O Lines
  64-lead TQFP and 64-pad MLF

- Operating Voltages
  2.7 - 5.5V for ATmega128L
  4.5 - 5.5V for ATmega128

2.1.4.2 Pinzuordnung

PortA bis PortG werden für direkte Pin-Funktionen (z.B. Port_WriteBit) von 0 bis 52 gezählt, siehe "PortBit".

Pinbelegung für Application Board Mega128

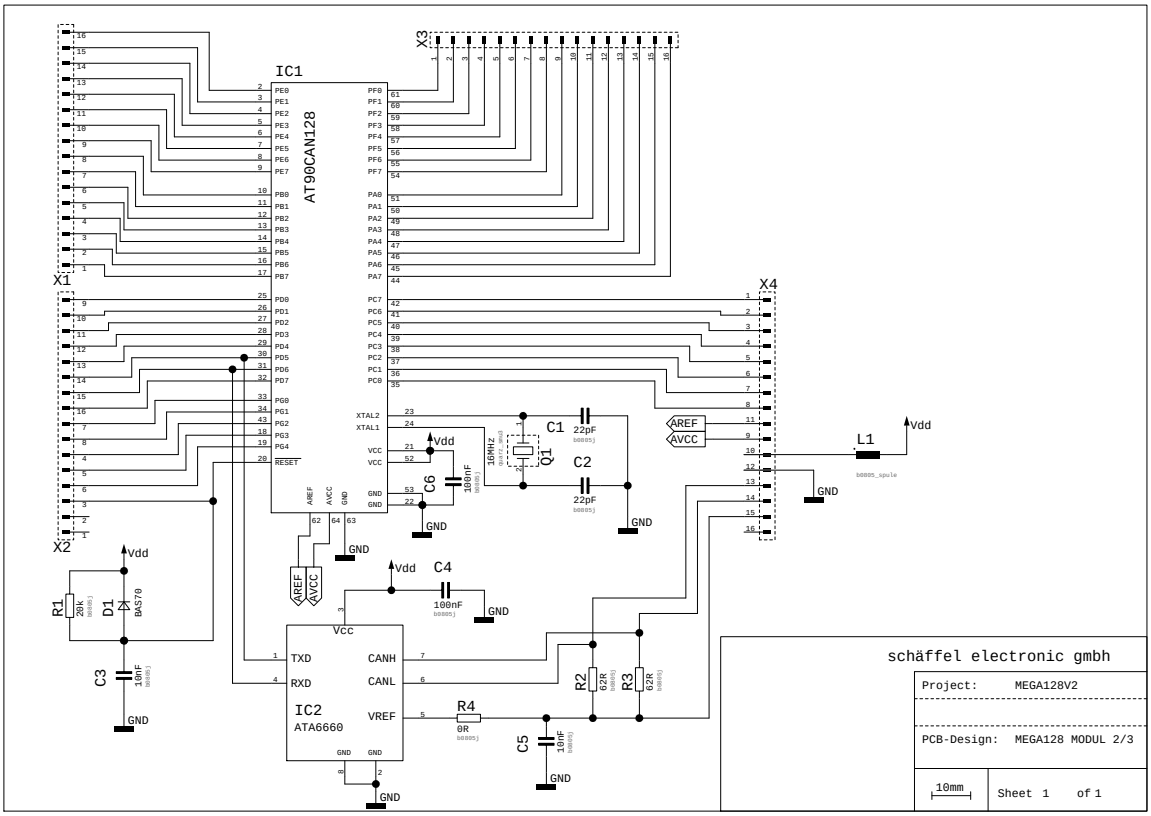
Modul	M128	Port	Port#	PortBit	Name1	Name2	Intern	Bemerkungen
	1				PEN			prog. Enable
X1_16	2	PE0	4	32	RXD0	PDI	EXT-RXD0	RS232
X1_15	3	PE1	4	33	TXD0	PDO	EXT-TXD0	RS232
X1_14	4	PE2	4	34	AIN0	XCK0		Analog Comparator
X1_13	5	PE3	4	35	AIN1	OC3A		Analog Comparator
X1_12	6	PE4	4	36	INT4	OC3B	EXT-T1	Taste1 (SW1)
X1_11	7	PE5	4	37	INT5	OC3C	TX-REQ	SPI_TX_REQ
X1_10	8	PE6	4	38	INT6	T3	EXT-T2	Taste2 (SW2) / Eingang Ti-

								mer 3
X1_9	9	PE7	4	39	INT7	IC3	EXT-DATA	LCD_Interface
X1_8	10	PB0	1	8	SS			SPI
X1_7	11	PB1	1	9	SCK			SPI
X1_6	12	PB2	1	10	MOSI			SPI
X1_5	13	PB3	1	11	MISO			SPI
X1_4	14	PB4	1	12	OC0		RX-BUSY	SPI_RX_BUSY
X1_3	15	PB5	1	13	OC1A		EXT-A1	DAC1
X1_2	16	PB6	1	14	OC1B		EXT-A2	DAC2
X1_1	17	PB7	1	15	OC1C	OC2	EXT-SCK	LCD_Interface
X2_5	18	PG3	6	51	TOSC2		LED1	Leuchtdiode
X2_6	19	PG4	6	52	TOSC1		LED2	Leuchtdiode
X2_3	20				RESET			
X4_10	21				VCC			
X4_12	22				GND			
	23				XTAL2			Oscillator
	24				XTAL1			Oscillator
X2_9	25	PD0	3	24	INT0	SCL	EXT-SCL	I2C
X2_10	26	PD1	3	25	INT1	SDA	EXT-SDA	I2C
X2_11	27	PD2	3	26	INT2	RXD1	EXT-RXD1	RS232
X2_12	28	PD3	3	27	INT3	TXD1	EXT-TXD1	RS232
X2_13	29	PD4	3	28	IC1	A16		IC Timer 1, SRAM bank select
X2_14	30	PD5	3	29	XCK1		LCD-E	LCD_Interface
X2_15	31	PD6	3	30	T1			Eingang Timer 1
X2_16	32	PD7	3	31	T2		KEY-E	LCD_Interface / Eingang Timer 2
X2_7	33	PG0	6	48	WR			WR SRAM
X2_8	34	PG1	6	49	RD			RD SRAM
X4_8	35	PC0	2	16	A8			ADR SRAM
X4_7	36	PC1	2	17	A9			ADR SRAM
X4_6	37	PC2	2	18	A10			ADR SRAM
X4_5	38	PC3	2	19	A11			ADR SRAM
X4_4	39	PC4	2	20	A12			ADR SRAM
X4_3	40	PC5	2	21	A13			ADR SRAM
X4_2	41	PC6	2	22	A14			ADR SRAM
X4_1	42	PC7	2	23	A15			ADR SRAM
X2_4	43	PG2	6	50	ALE			Latch
X3_16	44	PA7	0	7	AD7			A/D SRAM
X3_15	45	PA6	0	6	AD6			A/D SRAM
X3_14	46	PA5	0	5	AD5			A/D SRAM
X3_13	47	PA4	0	4	AD4			A/D SRAM
X3_12	48	PA3	0	3	AD3			A/D SRAM
X3_11	49	PA2	0	2	AD2			A/D SRAM
X3_10	50	PA1	0	1	AD1			A/D SRAM
X3_9	51	PA0	0	0	AD0			A/D SRAM
X4_10	52				VCC			
X4_12	53				GND			
X3_8	54	PF7	5	47	ADC7	TDI-JTAG		
X3_7	55	PF6	5	46	ADC6	TDO-JTAG		
X3_6	56	PF5	5	45	ADC5	TMS-		

						JTAG		
X3_5	57	PF4	5	44	ADC4	TCK-JTAG		
X3_4	58	PF3	5	43	ADC3			
X3_3	59	PF2	5	42	ADC2			
X3_2	60	PF1	5	41	ADC1			
X3_1	61	PF0	5	40	ADC0			
X4_11	62				AREF			
X4_12	63				GND			
X4_9	64				AVCC			

2.1.4.3 Schaltplan

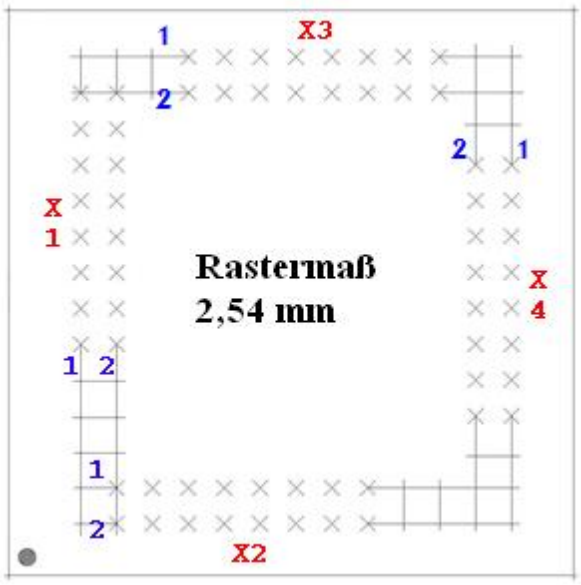
➡ Der hier abgebildete Schaltplan zeigt das geplante C-Control Pro Modul mit CAN Bus, was aber bisher nicht gebaut wurde. Im C-Control Pro 128 Modul ist eine Mega 128 CPU eingebaut, und kein AT90CAN128 wie hier abgebildet. Auch der ATA6660 CAN-Bus Transceiver ist nicht vorhanden.



2.1.5 Mega128 CAN Modul

Pinlayout des Moduls

Das Mega128 CAN Modul wird auf 4 doppelreihigen (2x8) Vierkantstiften ausgeliefert. Für eine Hardware Applikation müssen die entsprechenden Buchsenleisten im folgenden Rasterformat angeordnet werden:



In der Grafik sieht man die Buchsenleisten X1-X4 und dann die ersten beiden Pins der Buchsenleiste. Pin 1 von Leiste X1 entspricht dem Anschluß X1_1 (siehe Mega128 Pinzuordnung).

➔ Damit auf dem Applicationboard mit dem C-Control Mega128 CAN Modul der CAN-Bus und das LCD-Display gleichzeitig ansprechbar sind, wurden die Anschlüsse von PD5 und PF7 getauscht! Im C-Control Mega128 CAN ist PD5 mit X3_8 verbunden und PF7 mit X2_14!

Modulspeicher

In dem C-Control Pro Mega128 CAN Modul sind 128kB FLASH, 4kB EEPROM und 4kB SRAM integriert. Auf dem Application Board befindet sich ein zusätzliches EEPROM mit einer Speichertiefe von 8kB und ein SRAM mit 64kB Speichertiefe. Das 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 16MHz-Quarzoszillator. Alle zeitlichen Abläufe des Controllers sind von dieser Taktfrequenz
abgeleitet.

➔ Das normale C-Control Pro Mega128 Modul läuft mit 14,7456 Mhz. Für die Einhaltung eines genauen CAN-Bus Taktes, wird hier ein 16Mhz Quarz eingesetzt.

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 (X2_3) 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, PortE, PortF, PortG)

Das C-Control Pro Modul verfügt über 6 digitale Ports mit je 8 Pins und einem digitalen Port mit 5 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!

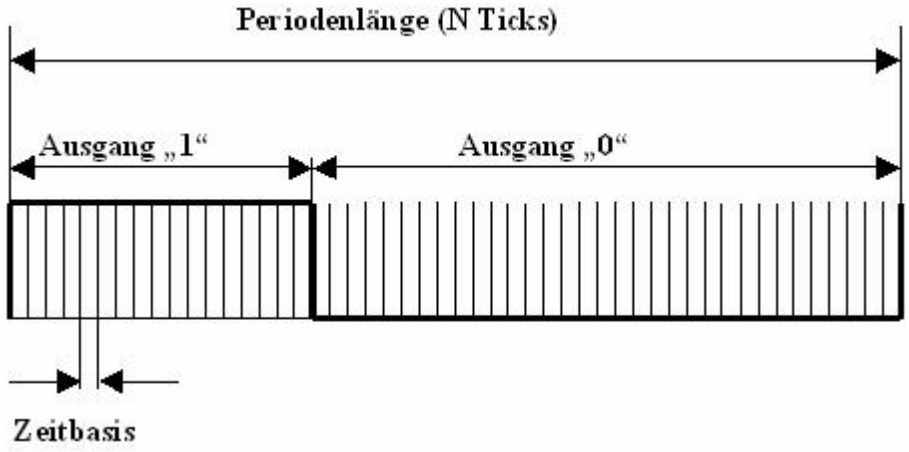
Digitale Eingangspins sind hochohmig oder mit internem Pullup-Widerstand beschaltet und überführen ein anliegendes Spannungssignal in einen logischen Wert. Voraussetzung dafür ist, daß sich das Spannungssignal innerhalb der für TTL-bzw. CMOS-ICs definierten Bereiche für Low- oder Highpegel befindet. 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 des Mega128 CAN 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 Entwicklungsumge-

bung 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 drei Timer für PLM zur Verfügung. *Timer_0* mit 8 bit und *Timer_1* und *Timer_3* mit jeweils 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, Timer_1* und *Timer_3* 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. 30mA
Stromaufnahme CAN-Bus Treiber	max. 150mA

Takt	
Taktfrequenz (Quarzoszillator)	16MHz
Mechanik	
äußere Abmessungen ohne Pins ca.	40 mm x 40mm x 8 mm
Masse	ca. 90g
Pinraster	2,54mm
Pinanzahl (zweireihig)	64

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

2.1.5.1 CPU

Mega128 CAN Übersicht

Der Mikrocontroller AT90CAN stammt aus der AVR-Familie von ATMEL. Es handelt sich um einen low-power Mikrocontroller mit Advanced RISC Architecture. Hier folgt eine kurze Zusammenstellung der Hardwareressourcen:

- **Advanced RISC Architecture**
  133 Powerful Instructions – Most Single Clock Cycle Execution
  32 x 8 General Purpose Working Registers + Peripheral Control Registers
  Fully Static Operation
  Up to 16 MIPS Throughput at 16 MHz
  On-chip 2-cycle Multiplier

- **Non volatile Program and Data Memories**
  32K/64K/128K Bytes of In-System Reprogrammable Flash (AT90CAN32/64/128)
  • Endurance: 10,000 Write/Erase Cycles
  Optional Boot Code Section with Independent Lock Bits
  • Selectable Boot Size: 1K Bytes, 2K Bytes, 4K Bytes or 8K Bytes
  • In-System Programming by On-Chip Boot Program (CAN, UART, ...)
  • True Read-While-Write Operation
  1K/2K/4K Bytes EEPROM (Endurance: 100,000 Write/Erase Cycles) (AT90CAN32/64/128)
  2K/4K/4K Bytes Internal SRAM (AT90CAN32/64/128)
  Up to 64K Bytes Optional External Memory Space

Programming Lock for Software Security

- **JTAG (IEEE std. 1149.1 Compliant) Interface**
  Boundary-scan Capabilities According to the JTAG Standard
  Programming Flash (Hardware ISP), EEPROM, Lock & Fuse Bits
  Extensive On-chip Debug Support

- **CAN Controller 2.0A & 2.0B - ISO 16845 Certified** (1)
  15 Full Message Objects with Separate Identifier Tags and Masks
  Transmit, Receive, Automatic Reply and Frame Buffer Receive Modes
  1Mbits/s Maximum Transfer Rate at 8 MHz
  Time stamping, TTC & Listening Mode (Spying or Autobaud)

- **Peripheral Features**
  Programmable Watchdog Timer with On-chip Oscillator
  8-bit Synchronous Timer/Counter-0
  • 10-bit Prescaler
  • External Event Counter
  • Output Compare or 8-bit PWM Output
  8-bit Asynchronous Timer/Counter-2
  • 10-bit Prescaler
  • External Event Counter
  • Output Compare or 8-Bit PWM Output
  • 32Khz Oscillator for RTC Operation
  Dual 16-bit Synchronous Timer/Counters-1 & 3
  • 10-bit Prescaler
  • Input Capture with Noise Canceler
  • External Event Counter
  • 3-Output Compare or 16-Bit PWM Output
  • Output Compare Modulation
  8-channel, 10-bit SAR ADC
  • 8 Single-ended Channels
  • 7 Differential Channels
  • 2 Differential Channels With Programmable Gain at 1x, 10x, or 200x
  On-chip Analog Comparator
  Byte-oriented Two-wire Serial Interface
  Dual Programmable Serial USART
  Master/Slave SPI Serial Interface
  • Programming Flash (Hardware ISP)

- **Special Microcontroller Features**
  Power-on Reset and Programmable Brown-out Detection
  Internal Calibrated RC Oscillator
  8 External Interrupt Sources
  5 Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down & Standby
  Software Selectable Clock Frequency
  Global Pull-up Disable

- **I/O and Packages**
  53 Programmable I/O Lines
  64-lead TQFP and 64-lead QFN

- **Operating Voltages: 2.7 - 5.5V**

- **Operating temperature: Industrial (-40°C to +85°C)**
- **Maximum Frequency: 8 MHz at 2.7V, 16 MHz at 4.5V**

2.1.5.2 Pinzuordnung

PortA bis PortG werden für direkte Pin-Funktionen (z.B. Port_WriteBit) von 0 bis 52 gezählt, siehe "PortBit".

➔ Damit auf dem Applicationboard mit dem C-Control Mega128 CAN Modul der CAN-Bus und das LCD-Display gleichzeitig ansprechbar sind, wurden die Anschlüsse von PD5 und PF7 getauscht! Im C-Control Mega128 CAN ist PD5 mit X3_8 verbunden und PF7 mit X2_14! Vergleiche dazu Pinzuordnung.

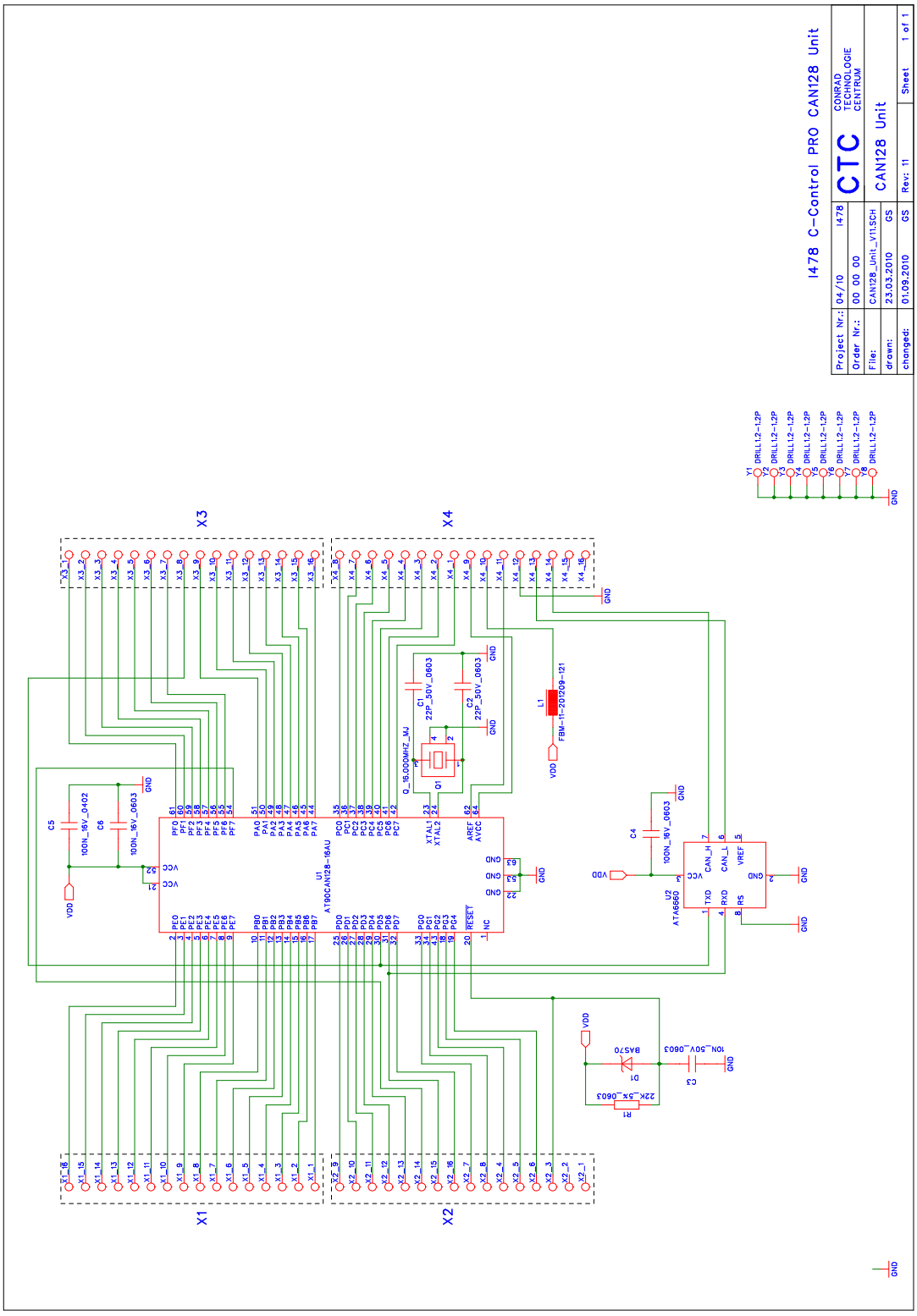
Pinbelegung für Application Board Mega128 CAN

Modul	M128	Port	Port#	PortBit	Name1	Name2	Intern	Bemerkungen
	1				PEN			prog. Enable
X1_16	2	PE0	4	32	RXD0	PDI	EXT-RXD0	RS232
X1_15	3	PE1	4	33	TXD0	PDO	EXT-TXD0	RS232
X1_14	4	PE2	4	34	AIN0	XCK0		Analog Comparator
X1_13	5	PE3	4	35	AIN1	OC3A		Analog Comparator
X1_12	6	PE4	4	36	INT4	OC3B	EXT-T1	Taste1 (SW1)
X1_11	7	PE5	4	37	INT5	OC3C	TX-REQ	SPI_TX_REQ
X1_10	8	PE6	4	38	INT6	T3	EXT-T2	Taste2 (SW2) / Eingang Timer 3
X1_9	9	PE7	4	39	INT7	IC3	EXT-DATA	LCD_Interface
X1_8	10	PB0	1	8	SS			SPI
X1_7	11	PB1	1	9	SCK			SPI
X1_6	12	PB2	1	10	MOSI			SPI
X1_5	13	PB3	1	11	MISO			SPI
X1_4	14	PB4	1	12	OC0		RX-BUSY	SPI_RX_BUSY
X1_3	15	PB5	1	13	OC1A		EXT-A1	DAC1
X1_2	16	PB6	1	14	OC1B		EXT-A2	DAC2
X1_1	17	PB7	1	15	OC1C	OC2	EXT-SCK	LCD_Interface
X2_5	18	PG3	6	51	TOSC2		LED1	Leuchtdiode
X2_6	19	PG4	6	52	TOSC1		LED2	Leuchtdiode
X2_3	20				RESET			
X4_10	21				VCC			
X4_12	22				GND			
	23				XTAL2			Oscillator
	24				XTAL1			Oscillator
X2_9	25	PD0	3	24	INT0	SCL	EXT-SCL	I2C
X2_10	26	PD1	3	25	INT1	SDA	EXT-SDA	I2C
X2_11	27	PD2	3	26	INT2	RXD1	EXT-RXD1	RS232
X2_12	28	PD3	3	27	INT3	TXD1	EXT-TXD1	RS232
X2_13	29	PD4	3	28	IC1	A16		IC Timer 1, SRAM bank select
X3_8	30	PD5	3	29	XCK1	TXCAN	LCD-E	LCD_Interface

X2_15	31	PD6	3	30	T1	RXCAN		Eingang Timer 1
X2_16	32	PD7	3	31	T2		KEY-E	LCD_Interface / Eingang Timer 2
X2_7	33	PG0	6	48	WR			WR SRAM
X2_8	34	PG1	6	49	RD			RD SRAM
X4_8	35	PC0	2	16	A8			ADR SRAM
X4_7	36	PC1	2	17	A9			ADR SRAM
X4_6	37	PC2	2	18	A10			ADR SRAM
X4_5	38	PC3	2	19	A11			ADR SRAM
X4_4	39	PC4	2	20	A12			ADR SRAM
X4_3	40	PC5	2	21	A13			ADR SRAM
X4_2	41	PC6	2	22	A14			ADR SRAM
X4_1	42	PC7	2	23	A15			ADR SRAM
X2_4	43	PG2	6	50	ALE			Latch
X3_16	44	PA7	0	7	AD7			A/D SRAM
X3_15	45	PA6	0	6	AD6			A/D SRAM
X3_14	46	PA5	0	5	AD5			A/D SRAM
X3_13	47	PA4	0	4	AD4			A/D SRAM
X3_12	48	PA3	0	3	AD3			A/D SRAM
X3_11	49	PA2	0	2	AD2			A/D SRAM
X3_10	50	PA1	0	1	AD1			A/D SRAM
X3_9	51	PA0	0	0	AD0			A/D SRAM
X4_10	52				VCC			
X4_12	53				GND			
X2_14	54	PF7	5	47	ADC7	TDI-JTAG		im CAN Modul getauscht mit X3_8
X3_7	55	PF6	5	46	ADC6	TDO-JTAG		
X3_6	56	PF5	5	45	ADC5	TMS-JTAG		
X3_5	57	PF4	5	44	ADC4	TCK-JTAG		
X3_4	58	PF3	5	43	ADC3			
X3_3	59	PF2	5	42	ADC2			
X3_2	60	PF1	5	41	ADC1			
X3_1	61	PF0	5	40	ADC0			
X4_11	62				AREF			
X4_12	63				GND			
X4_9	64				AVCC			
X4_13					CAN-L			
X4_14					CAN-H			

2.1.5.3 Schaltplan

➔ Der hier abgebildete Schaltplan zeigt das neue C-Control Pro Mega128 CAN Modul **mit** CAN Bus.



2.1.6 Mega32 Application Board

USB

Das "C-Control Pro Application Board MEGA 32" (Conrad Artikel-Nr. 198245) verfügt über eine USB Schnittstelle zum Laden und Debuggen des Programms. 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. Wird ein Programm im Interpreter gestartet, dann leuchtet die rote LED während der Laufzeit. 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.

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 ei-

ne 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.

Das Display wird im 4-Bit Modus betrieben. Daten werden am Ausgang EXT-Data angelegt, und dann der Reihe nach in das Schieberegister 74HC164 mit EXT-SCK hinein getaktet. Mit setzen von LCD-E werden die 4-Bit dann an das Display übergeben.

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.

Die einzelnen 12 Leitungen der Tastatur werden über PL(parallel load - KEY-E) in die Schieberegister der 74HC165 übernommen. Dann wird über einzelnes Triggern von CP (clock input - EXT-SCK) die einzelnen Bits zu Q7
gelatched. Dort können sie mit EXT-Data eingelesen werden. Damit alle Bits von einem 74HC165 in den anderen 74HC165 gelangen, ist der eine Q7 des 74HC165 mit dem DS des anderen 74HC165 verbunden.

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

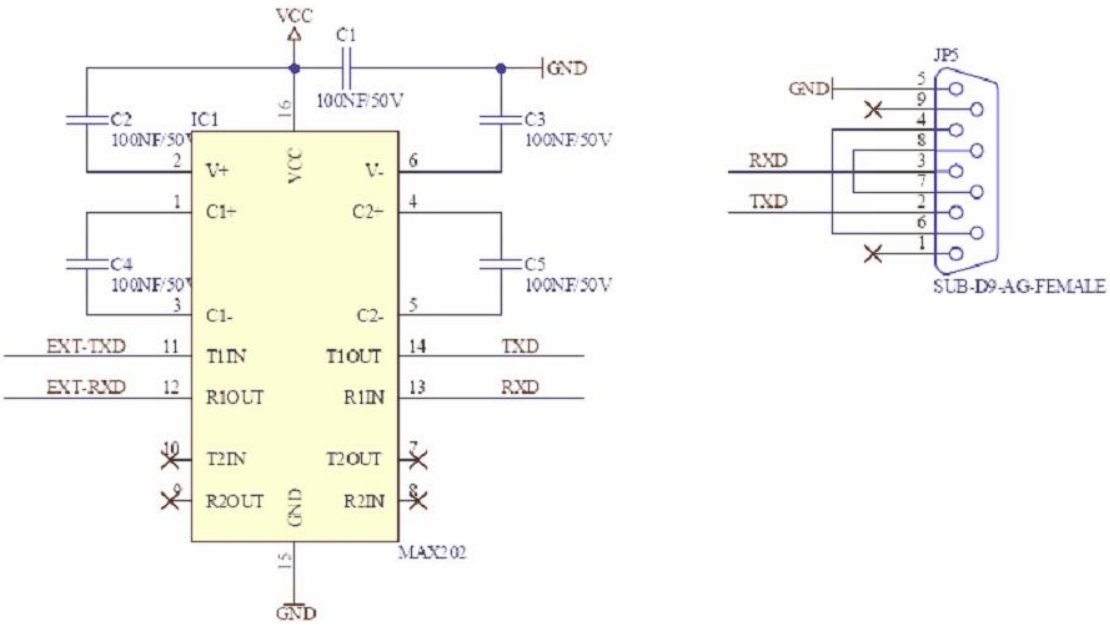
5V. Alle Schaltungsteile auf dem Application Board werden mit dieser Spannung versorgt. Durch die Leistungsreserven des Netzteils stehen diese 5V auch zur Versorgung externer ICs zur Verfügung.

➔ 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.

➔ Hält man das Application Board so, das die Anschlüsse nach oben zeigen, dann ist die linke Lochrasterfeldreihe mit GND verbunden und die rechte Lochrasterfeldreihe mit VCC.

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 Verbin-

dungskabel 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

2.1.6.1 Jumper

Jumper

Durch Jumper können bestimmte Optionen ausgewählt werden. Das betrifft einige Ports, welche mit speziellen Funktionen belegt sind (siehe Tabelle der Pinzuordnung des M32). 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 an-

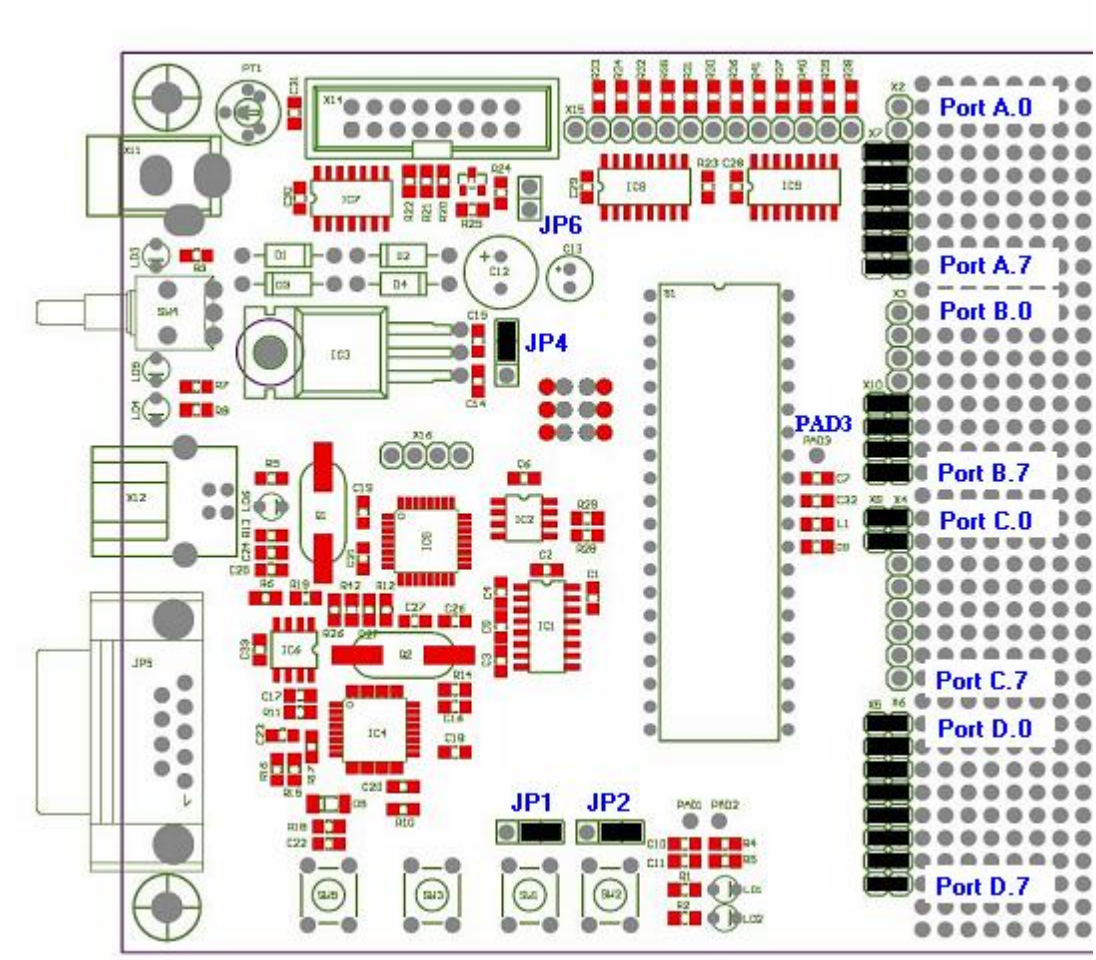
dere Funktionen zur Verfügung. Neben den Jumpern für die Ports gibt es noch zusätzliche Jumper, welche nachfolgend beschrieben werden.

Ports A bis D

Die dem Mega32 Modul zur Verfügung stehenden Ports sind in dieser Grafik eingezeichnet. Dabei ist die rechte Seite dem Modul verbunden, die linke Seite verbindet zu Bauteilen des Application Boards. Wird ein Jumper gezogen, so wird die Verbindung zum Application Board unterbrochen. Dies kann zur Störung von USB, RS232 etc. auf dem Board führen.

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.



Jumperpositionen im Auslieferzustand

JP4

JP4 dient zum Umschalten der Betriebsspannung (Netzteil oder USB). Das Application Board sollte

mit Netzteil und Spannungsregler betrieben werden (Auslieferzustand). Der maximal entnehmbare Strom der USB Schnittstelle ist kleiner als der des Netzteils. Ein Überschreiten kann zu Schäden am USB Interface des Computers führen.

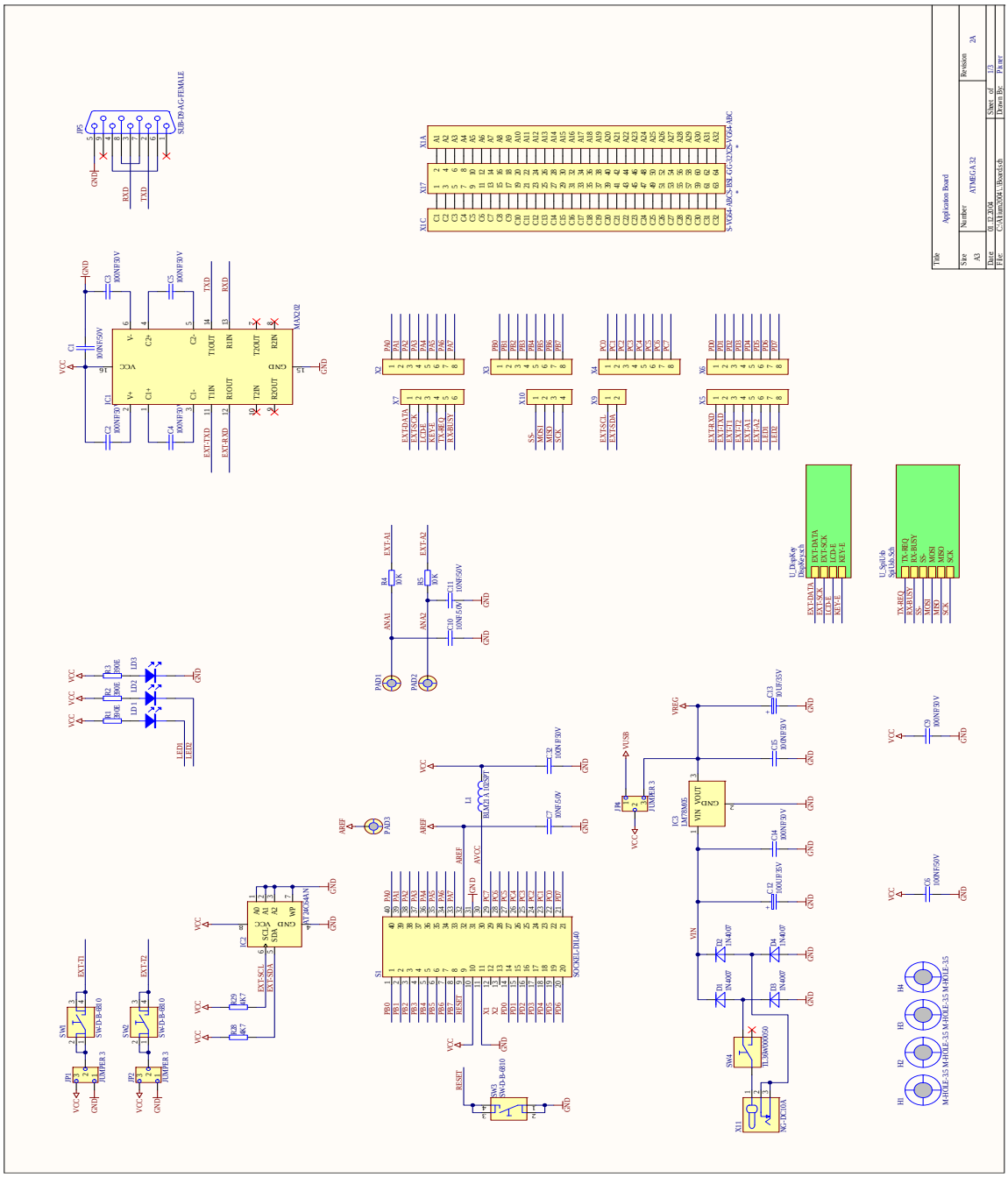
JP6

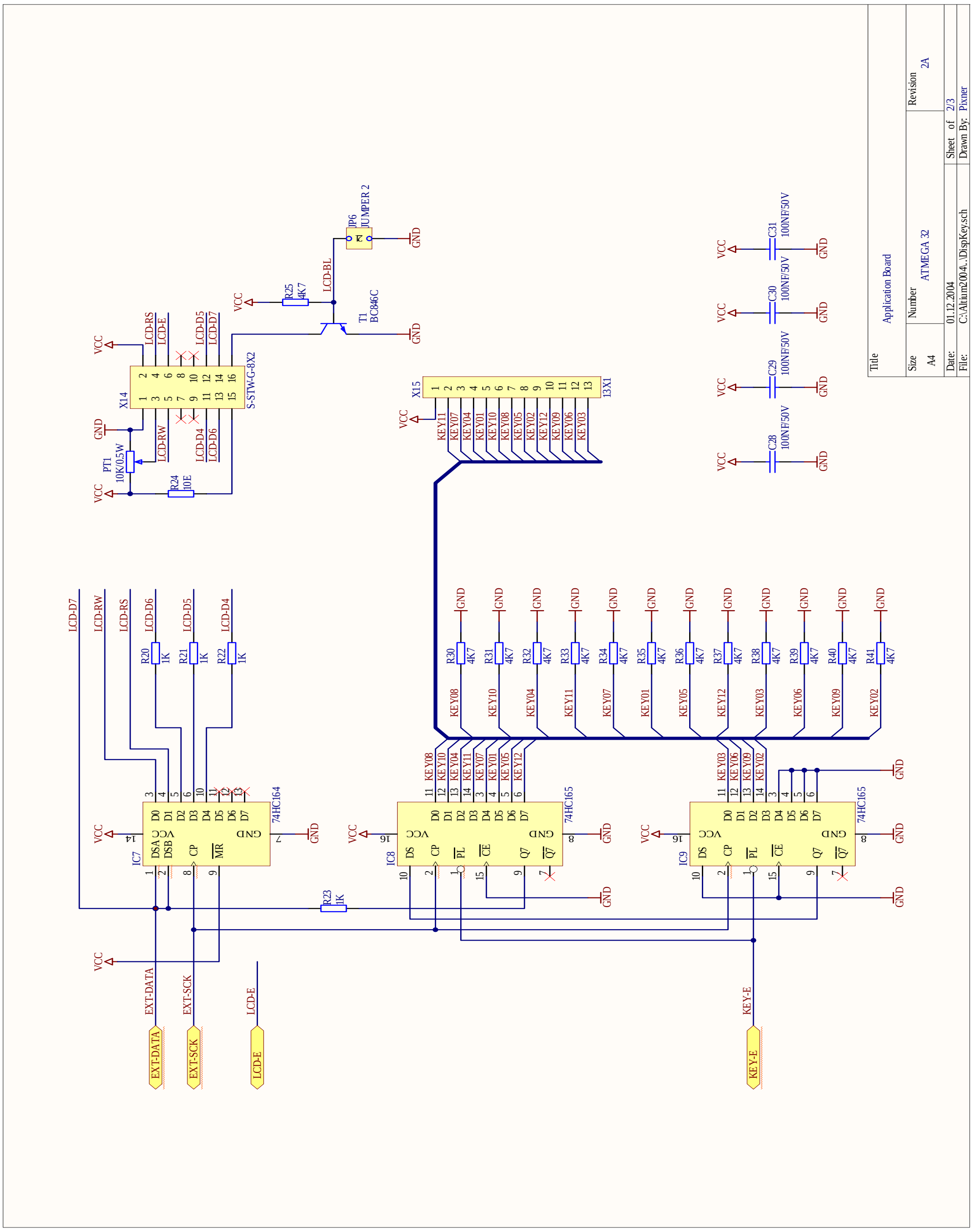
Bei Verwendung des Displays kann mit JP6 die Beleuchtung (back light) abgeschaltet werden.

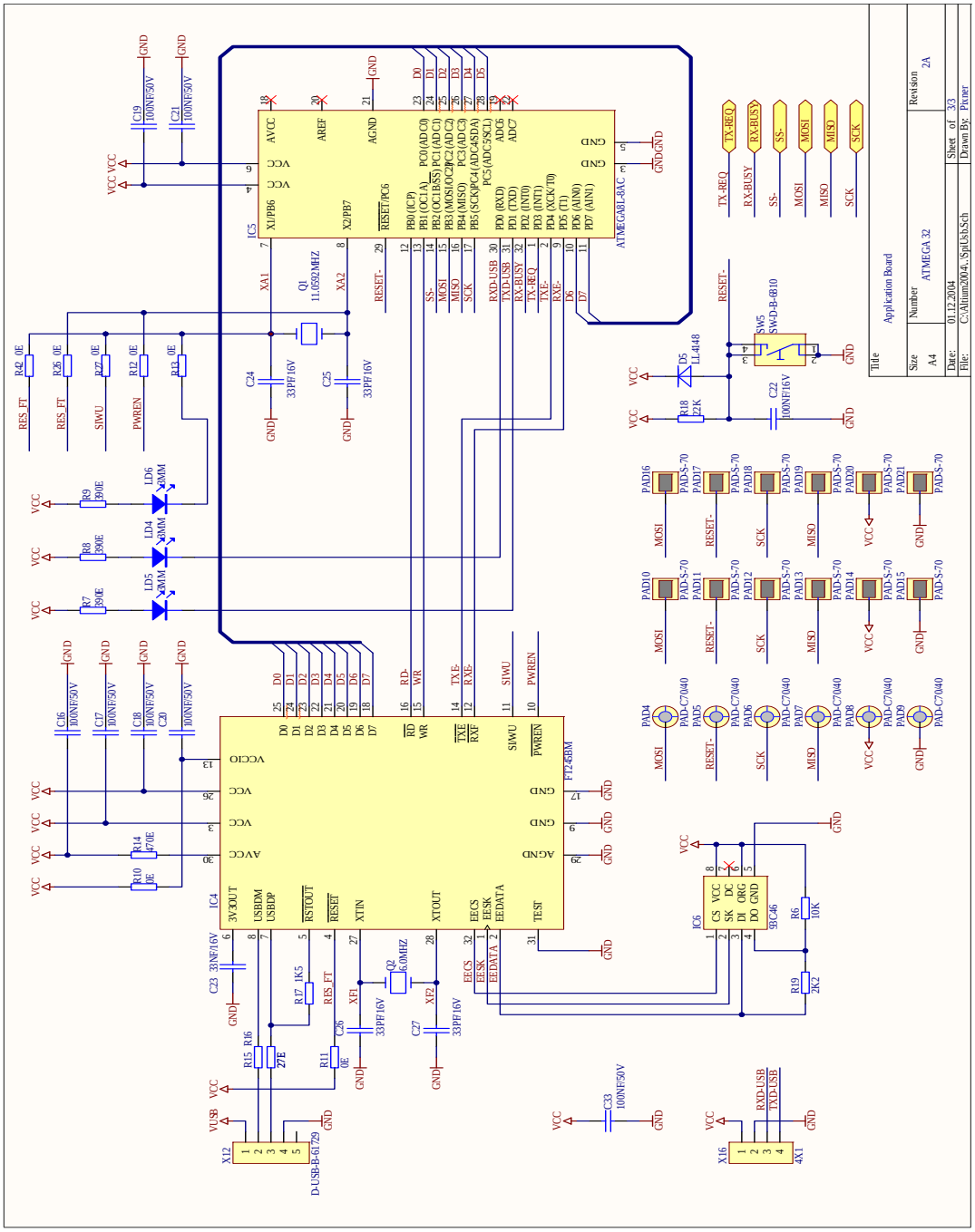
PAD3

PAD3 (rechts neben dem Modul, unter der blauen Beschriftung) wird als ADC_VREF_EXT für die Funktionen ADC_Set und ADC_SetInt benötigt.

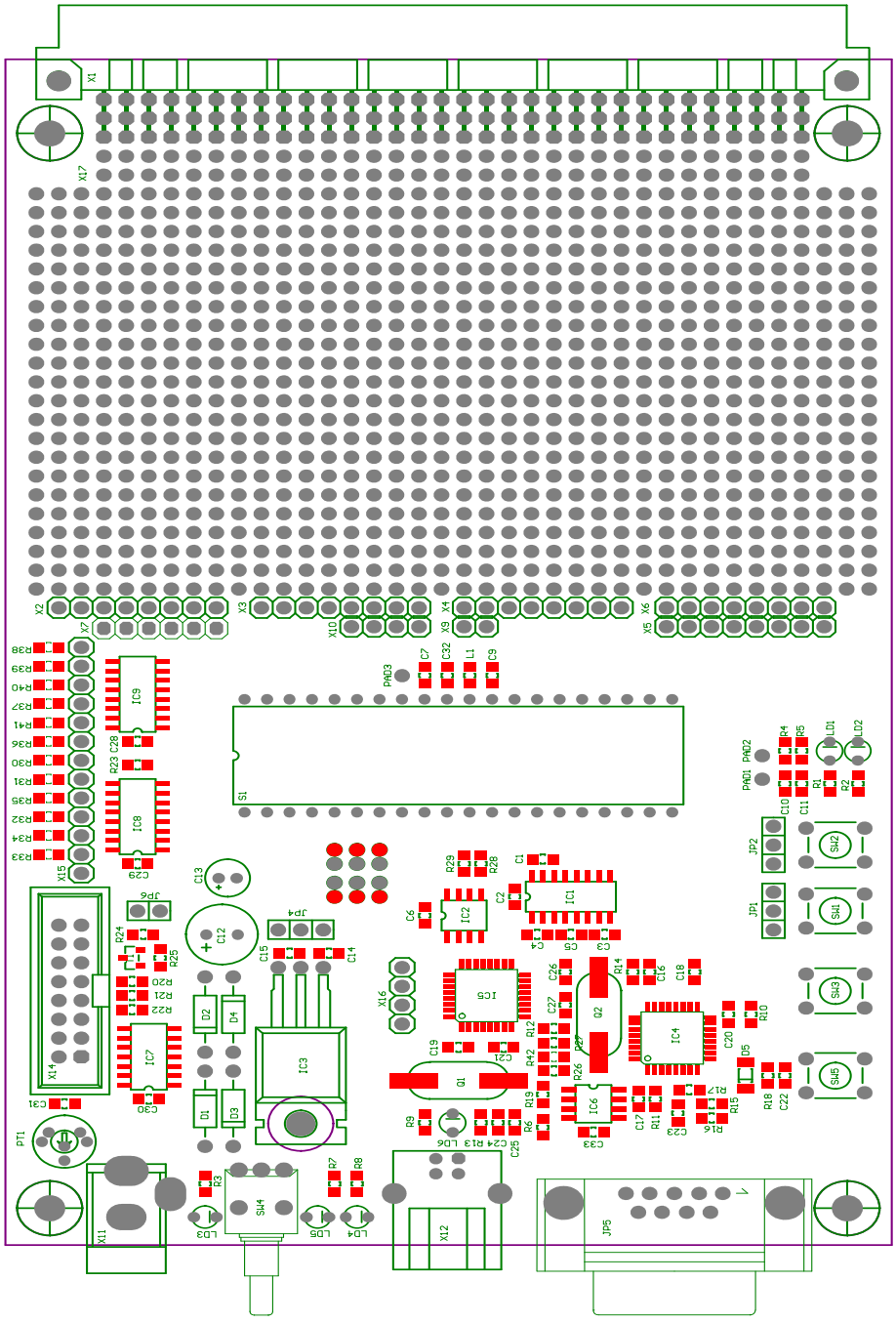
2.1.6.2 Schaltplan




Application Board
ATMEGA 32
Revision 2A
Size A4
Date: 01.12.2004
Sheet of 2/3
Drawn By: Pixner



2.1.6.3 Bestückungsplan



2.1.7 Mega128 Application Board

USB

Das "C-Control Pro Application Board MEGA 128" (Conrad Artikel-Nr. 198258) verfügt über eine USB Schnittstelle zum Laden und Debuggen des Programms. 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. Wird ein Programm im Interpreter gestartet, dann leuchtet die rote LED während der Laufzeit. Für die USB-Kommunikation wird die SPI-Schnittstelle des Mega128 verwendet (PortB.0 bis PortB.4, PortE.5) und müssen über die entsprechenden Jumper verbunden sein.

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 PortG.3 und LD2 an PortG.4 angeschlossen werden. Die Leuchtdioden leuchten wenn der entsprechende Port Pin low (GND) ist.

Taster

Es sind vier Taster vorgesehen. Mit SW3 (RESET1) wird beim Mega128 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 PortE.4 gelegt werden und entsprechend SW2 an PortE.6. 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.

Das Display wird im 4-Bit Modus betrieben. Daten werden am Ausgang EXT-Data angelegt, und dann der Reihe nach in das Schieberegister 74HC164 mit EXT-SCK hinein getaktet. Mit setzen von LCD-E werden die 4-Bit dann an das Display übergeben.

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.

Die einzelnen 12 Leitungen der Tastatur werden über PL(parallel load - KEY-E) in die Schieberegister der 74HC165 übernommen. Dann wird über einzelnes Triggern von CP (clock input - EXT-SCK) die einzelnen Bits zu Q7
gelatched. Dort können sie mit EXT-Data eingelesen werden. Damit alle Bits von einem 74HC165 in den anderen 74HC165 gelangen, ist der eine Q7 des 74HC165 mit dem DS des anderen 74HC165 verbunden.

CAN-Bus Anschluß

J3 ist der CAN-Anschluß wenn ein C-Control Pro Mega128 CAN Modul angeschlossen ist. Der obere Stift ist CAN-Hi, und der untere Stift ist CAN-Lo (Orientierung: serielle Schnittstelle zeigt nach links). Je nachdem wann das Applicationboard gekauft wurde, sind die beiden Stifte nicht bestückt, und müssen selbst aufgelötet werden.

SRAM

Auf dem Application Board befindet sich ein SRAM-Chip (K6X1008C2D) von Samsung. Dadurch wird der verfügbare SRAM-Speicher auf 64kByte erweitert. Das SRAM belegt zur Ansteuerung die Ports A , C und teilweise Port G. Wird das SRAM nicht benötigt, dann kann es mit JP7 de-

aktiviert werden und diese Ports stehen dann dem Anwender zur Verfügung.

➔ Um das SRAM zu deaktivieren muss der Jumper nach links umgelegt werden (Orientierung: serielle Schnittstelle zeigt nach links), so das die linken beiden Stifte von JP7 verbunden sind.

➔ Obwohl der eingesetzte RAM Chip 128kb Kapazität hat, sind aus Gründen des Speichermodells nur 64kb davon nutzbar.

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	PortD.0
I2C SDA	I2C-Bus Datenleitung	PortD.1

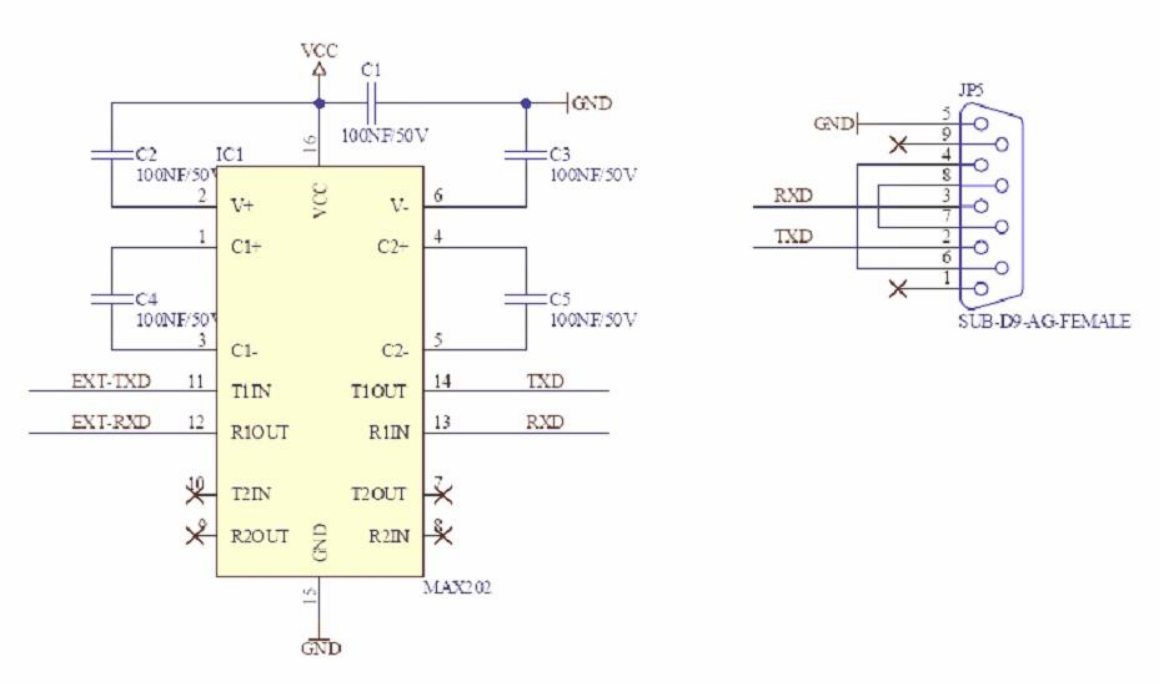
Spannungsversorgung (POWER, 5 Volt, GND)

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.

➔ Hält man das Application Board so, das die Anschlüsse nach oben zeigen, dann ist die linke Lochrasterfeldreihe mit GND verbunden und die rechte Lochrasterfeldreihe mit VCC.

Serielle Schnittstellen



Der Mikrocontroller Atmega128 besitzt hardwareseitig zwei asynchrone serielle Schnittstellen 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 RS232 Standard (positive Spannung für Lowbits, negative Spannung für Highbits) für beide Schnittstellen. 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 RxD0 (PortE.0), TxD0 (PortE.1) und RxD1(PortD.2), TxD1 (PortD.3) vom Controller mit dem Pegelwandler 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 RxD0 an Pin 3 und TxD0 an Pin 2. Der GND-Anschluß liegt auf Pin 5. Es werden für die serielle Datenübertragung keine Handshakesignale verwendet. Die zweite serielle Schnittstelle ist auf eine 3-polige Stiftleiste geführt. RxD1 liegt an Pin 2 und TxD1 an Pin 1, Pin3=GND.

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) rechts unten neben

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

2.1.7.1 externes RAM

Auf dem Application Board der **Mega128** und **Mega128 CAN** ist externes RAM vorhanden. Dieses RAM wird vom Interpreter automatisch erkannt und für das auszuführende Programm genutzt. Statt ca. 2665 Bytes stehen dann ca. 63848 Bytes als Programmspeicher zur Verfügung. Hierfür muss das Programm nicht neu kompiliert werden.

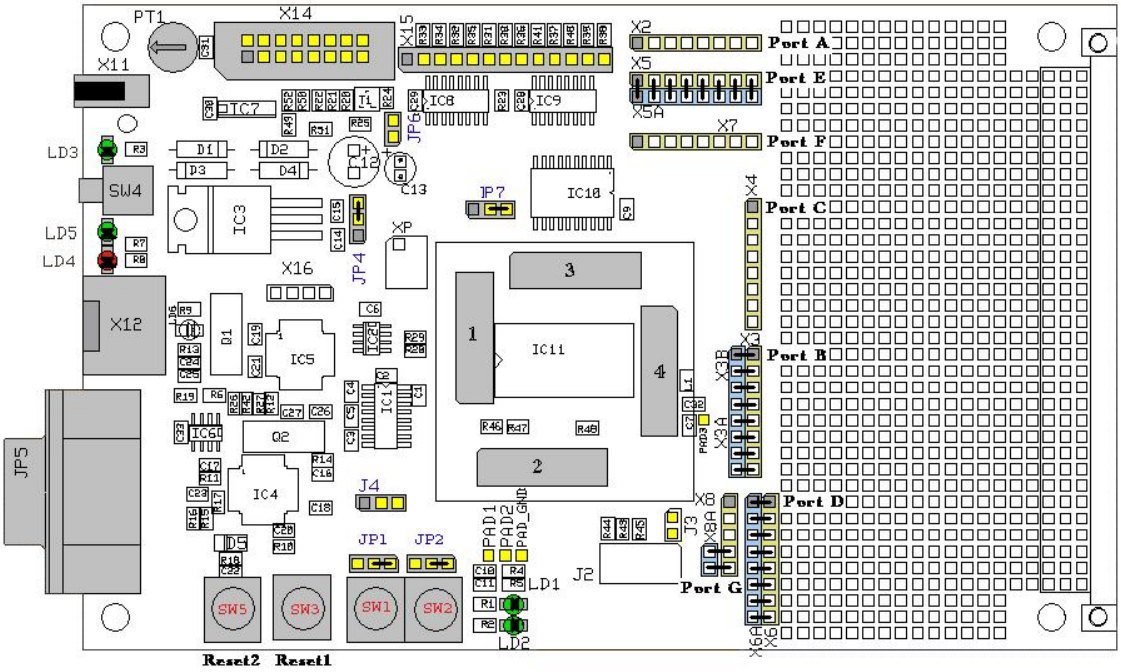
➔ Wird das SRAM nicht benötigt, dann kann es mit JP7 deaktiviert werden und diese Ports sind dann frei. Um das SRAM zu deaktivieren muss der Jumper nach links umgelegt werden (Orientierung: serielle Schnittstelle zeigt nach links), so das die linken beiden Stifte von JP7 verbunden sind.

2.1.7.2 Jumper

Jumper

Durch Jumper können bestimmte Optionen ausgewählt werden. Das betrifft einige Ports, welche mit speziellen Funktionen belegt sind (siehe Tabelle der Pinzuordnung von M128). Beispielsweise ist die serielle Schnittstelle über die Pins PortE.0 und PortE.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

Ports A bis G

Die dem Mega128 Modul zur Verfügung stehenden Ports sind in dieser Grafik eingezeichnet. Dabei ist die gelbe Seite dem Modul verbunden, die hellblaue Seite verbindet zu Bauteilen des Application Boards. Wird ein Jumper gezogen, so wird die Verbindung zum Application Board unterbrochen. Dies kann zur Störung von USB, RS232 etc. auf dem Board führen. Die grau eingezeichnete Markierung stellt den ersten Pin (Pin 0) des Ports dar.

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). Der maximal entnehmbare Strom der USB Schnittstelle ist kleiner als der des Netzteils. Ein Überschreiten kann zu Schäden am USB Interface des Computers führen.

JP6

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

JP7

Wird das SRAM auf dem Application Board nicht benötigt, dann kann es mit JP7 deaktiviert werden und diese Ports stehen dann dem Anwender zur Verfügung.

➔ Um das SRAM zu deaktivieren muss der Jumper nach links umgelegt werden (Orientierung: serielle Schnittstelle zeigt nach links), so das die linken beiden Stifte von JP7 verbunden sind.

J3

J3 ist der CAN-Anschluß wenn ein C-Control Pro Mega128 CAN Modul angeschlossen ist. Der obere Stift ist CAN-Hi, und der untere Stift ist CAN-Lo (Orientierung: serielle Schnittstelle zeigt nach links). Je nachdem wann das Applicationboard gekauft wurde, sind die beiden Stifte nicht bestückt, und müssen selbst aufgelötet werden.

J4

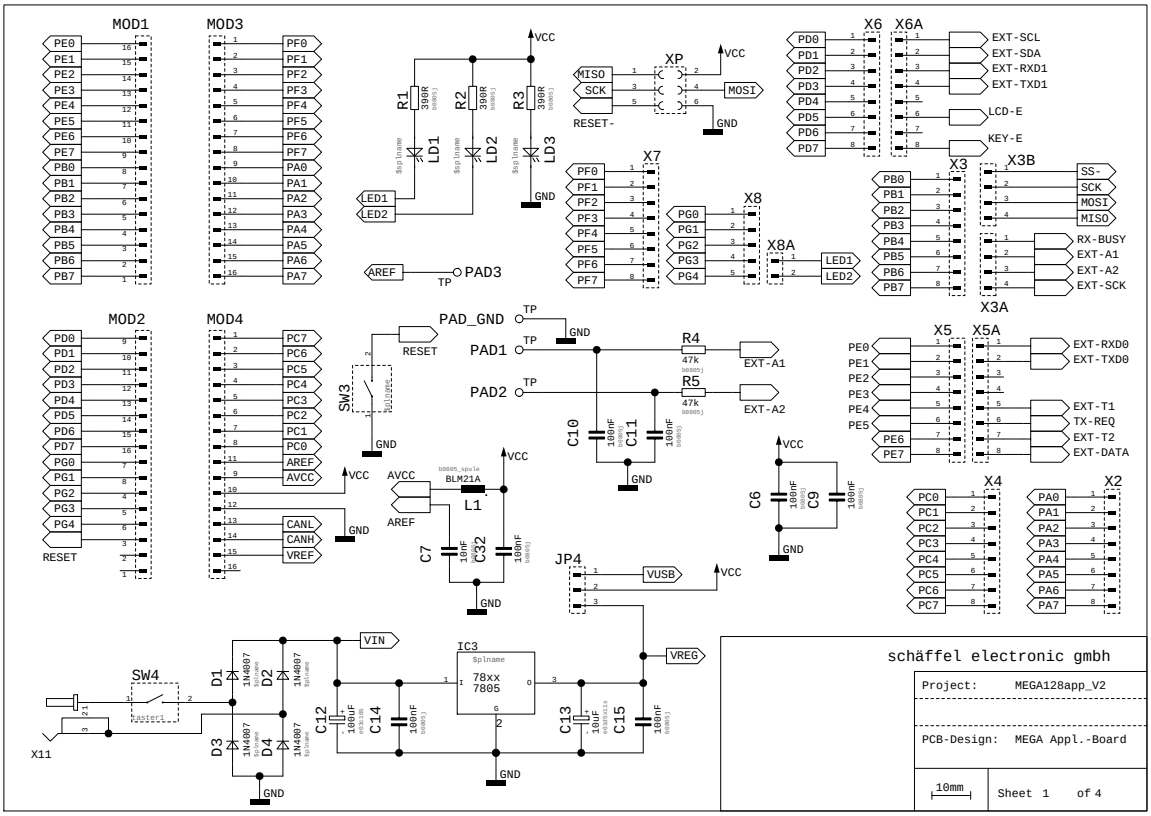
Am Jumper J4 ist die 2. serielle Schnittstelle des Mega128 über einen Pegelwandler angeschlossen.

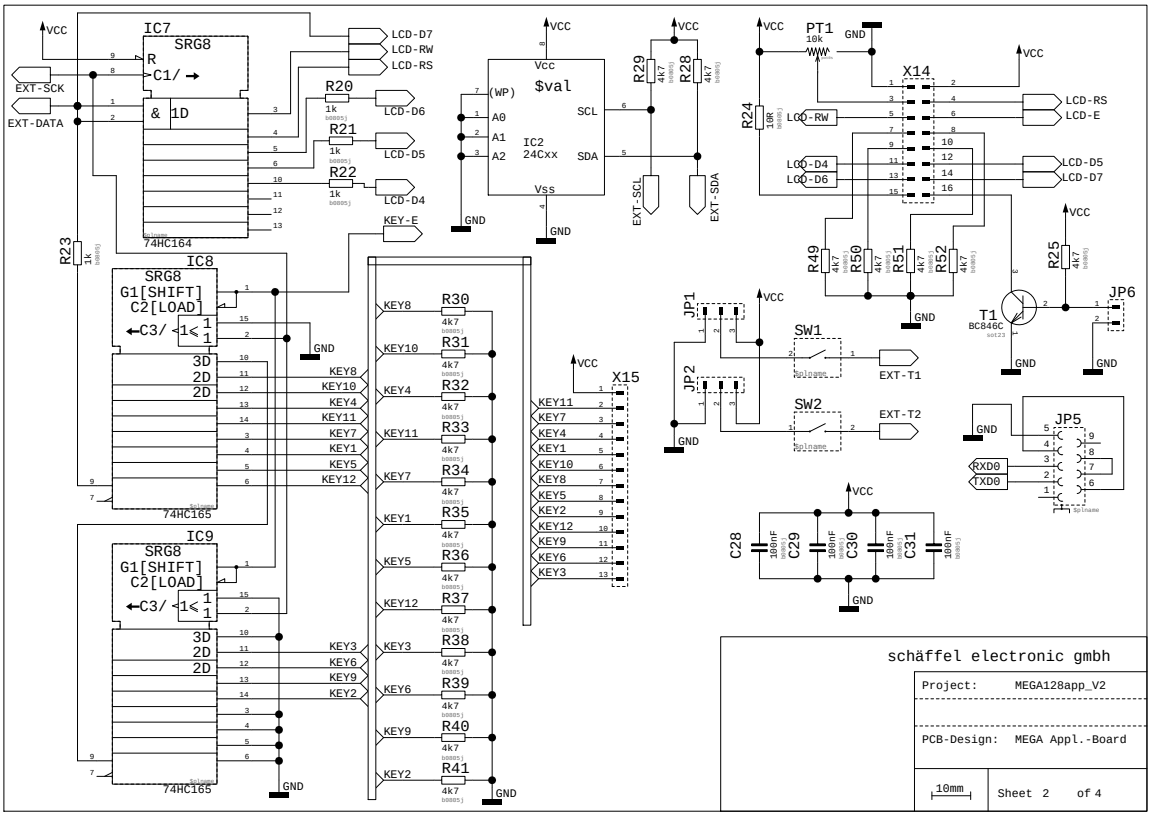
Pin 1 (links, grau)	**TxD**
Pin 2 (mitte)	**RxD**
Pin 3 (rechts)	**GND**

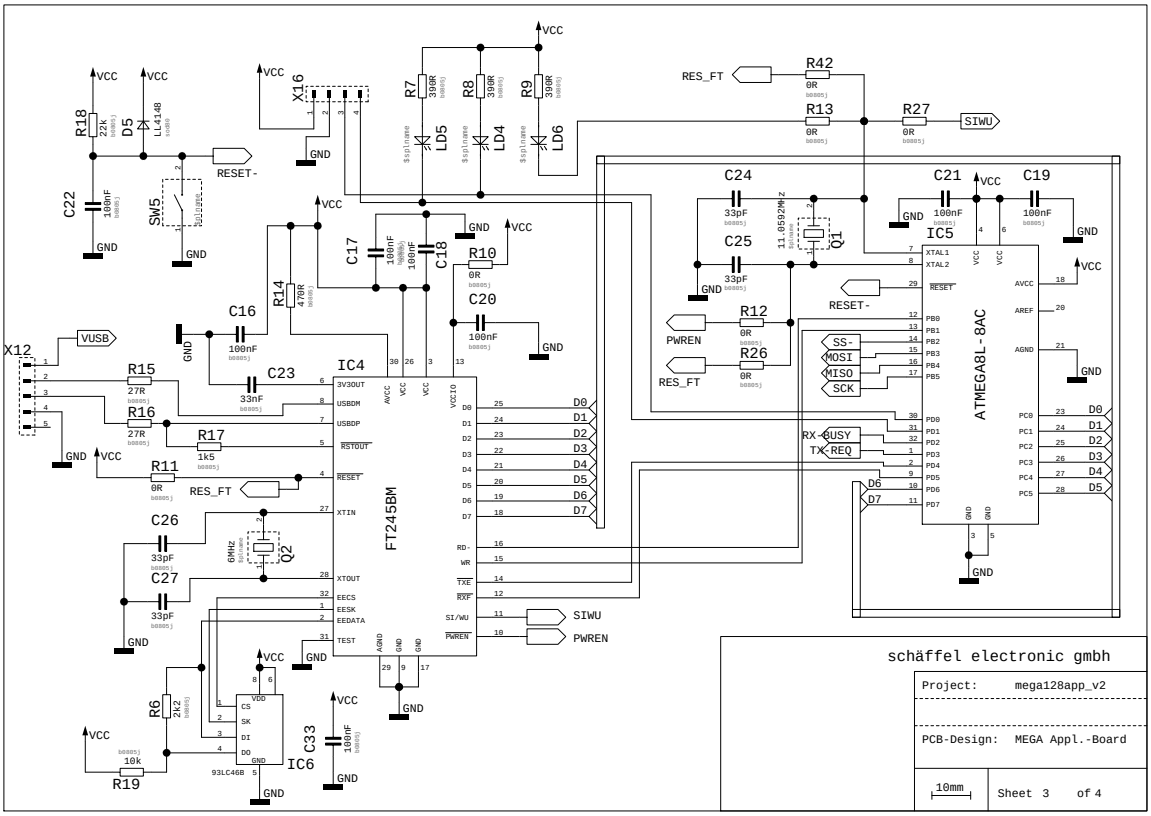
PAD3

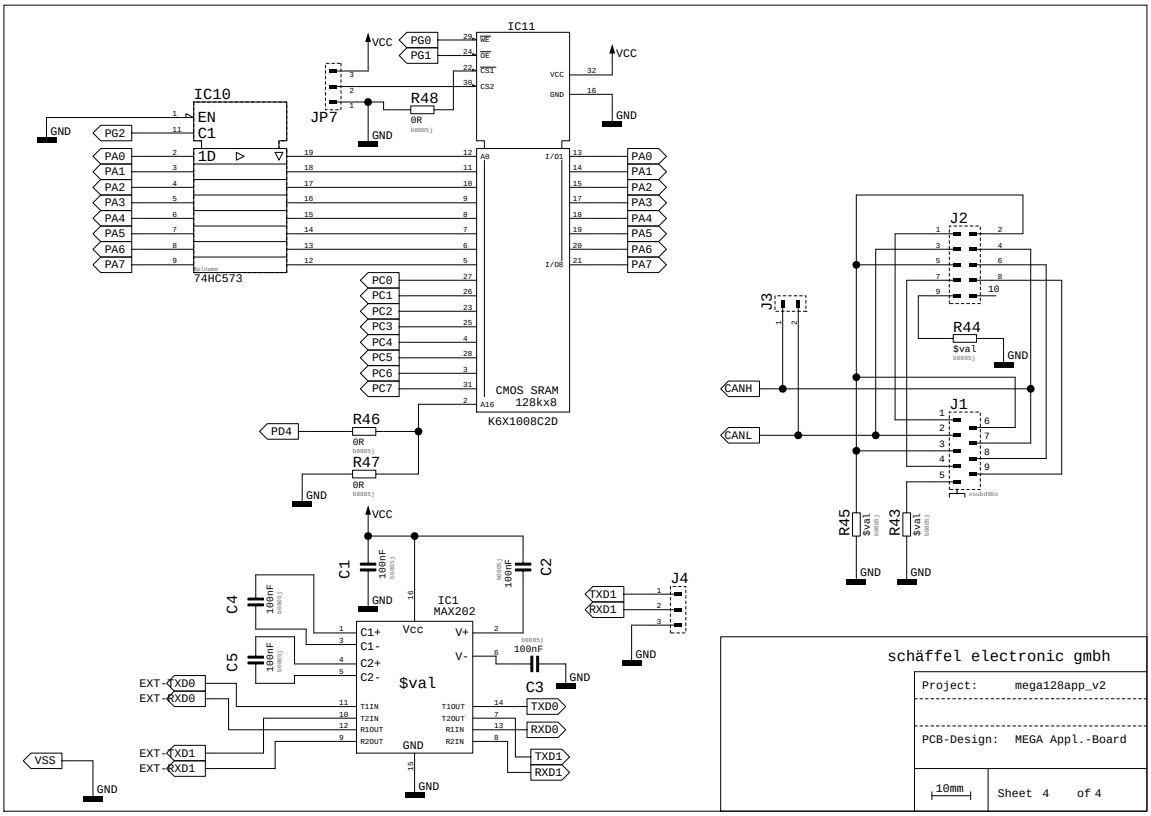
PAD3 (rechts neben dem Modul) wird als ADC_VREF_EXT für die Funktionen ADC_Set und ADC_SetInt benötigt.

2.1.7.3 Schaltplan

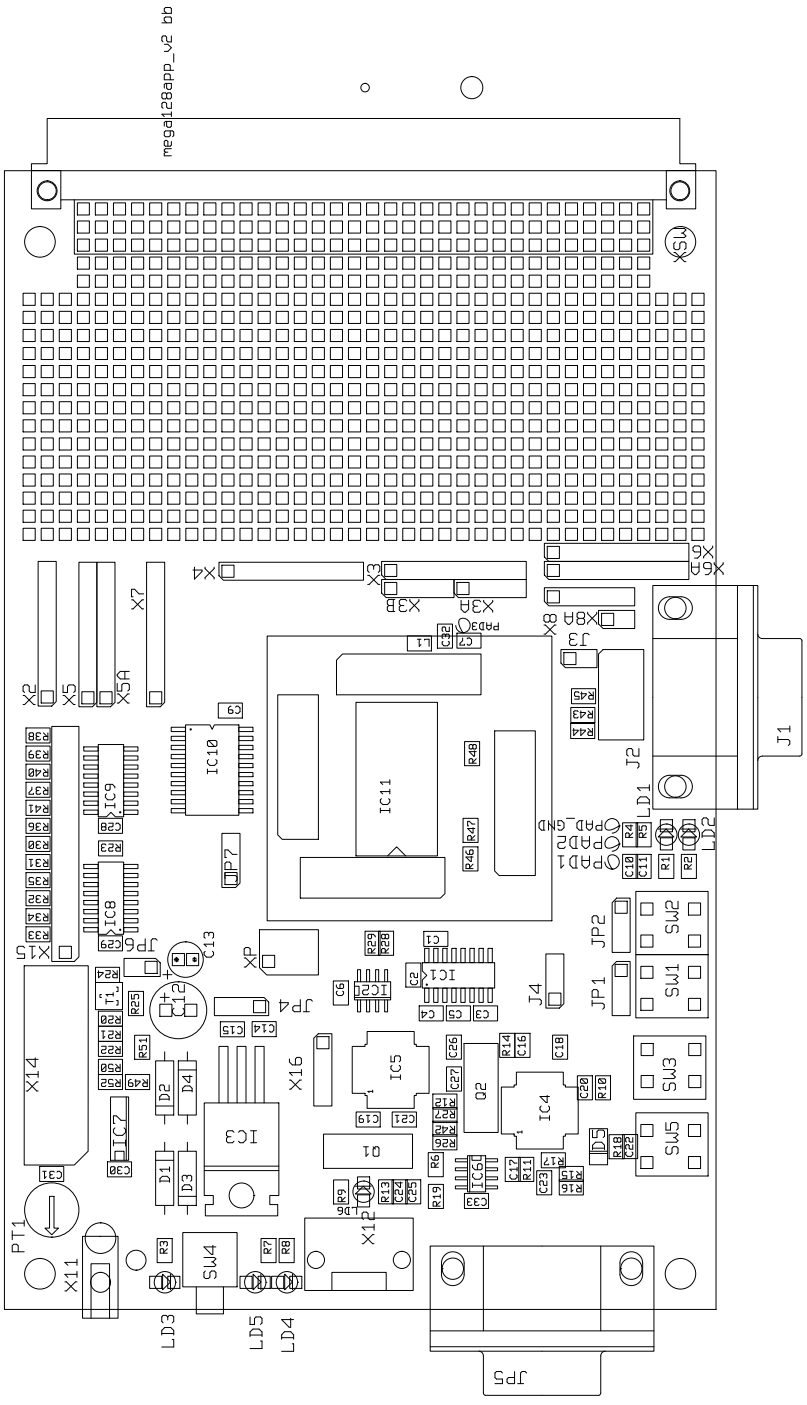






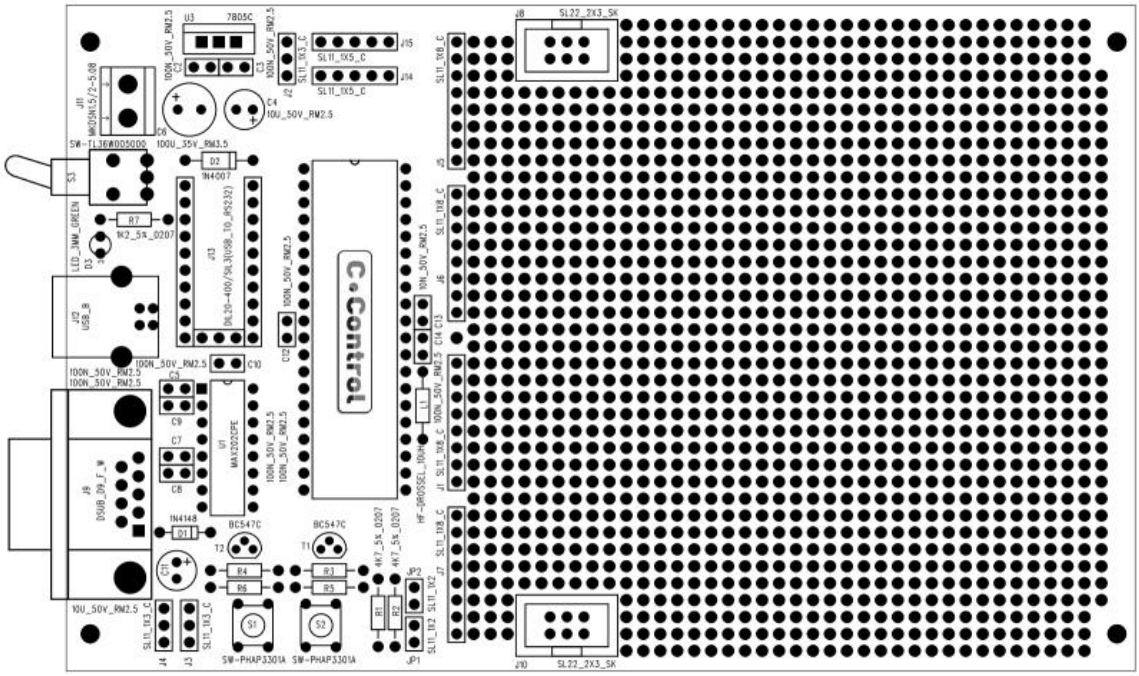


2.1.7.4 Bestückungsplan



2.1.8 Mega32 Projectboard

Das "C-Control Projectboard PRO32" (Conrad Artikel-Nr. 197287) stellt eine günstige Alternative zum Applikationsboard MEGA32 (Conrad-Best.-Nr. 198245) dar. Es ist vom Funktionsumfang her zum C-Control PRO Applikation-Board deutlich eingeschränkt und dient hauptsächlich für eigene Hardwareentwicklungen in Verbindung mit der der MEGA32 UNIT. Das Projectboard enthält die wichtigsten Komponenten, die zum Betrieb der MEGA32 UNIT benötigt werden. Weiterhin ist auf dem Projectboard eine Spannungsversorgung (USB/Netzadapter), Schnittstellenwandler (RS232) und ein großes Lochrasterfeld für eigene Entwicklungen vorhanden. Standardmäßig ist das Projectboard für die Programmierung über RS232 ausgelegt. Optional kann über den RS232-USB-Converter (Conrad-Best.-Nr. 197257) die Programmierung der MEGA32 UNIT auch über USB geschehen. Die Programmierung erfolgt in diesen Fall ebenfalls über die serielle Verbindung der MEGA32 UNIT (UART), deshalb ist die Programmübertragung nicht so schnell wie die USB-Übertragung auf dem Applikation-Board MEGA32.



- Die MEGA32 UNIT wird so aufgesteckt, daß der Schriftzug der UNIT zu lesen ist, wenn die Programmier und Versorgungsstecker zu Ihnen hin zeigen.

- Im Basiszustand ohne RS232-USB-Converter sind die Jumper J4/J3 so zu stecken, wie auf der Abbildung zu sehen ist.

➔ Wird der RS232-USB-Converter verwendet (nicht im Lieferumfang), so müssen diese Jumper auf USB umgesteckt werden.

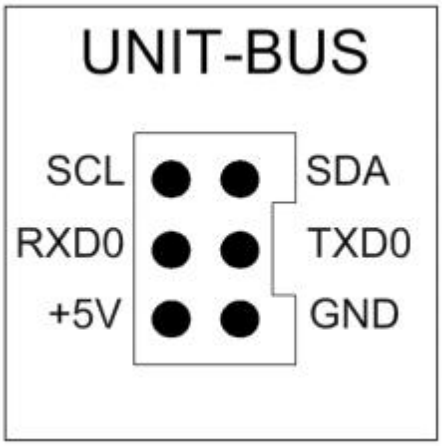
- Der Jumper J2 dient zur Auswahl der Versorgungsspannung. Bei Jumperstellung „Netz“ werden die Klemmen J11 zur Versorgung benutzt (Labornetzteil oder stabilisiertes Steckernetzgerät, min. 100mA, je nach Applikation). Wird der Jumper J2 auf USB umgesteckt, kann das Board auch über die USB-Stromversorgung des Rechners betrieben werden.

➔ Achtung! Eine maximale Stromaufnahme von 100mA über USB darf nicht überschritten werden!

- Der Schalter S3 sowie die Stromversorgungsstiftleisten JP7/JP5 und die Stifte für Vcc/GND auf den Lochrasterfeld sind bei USB-Betrieb nicht mehr unter Spannung. Diese Versorgung dient lediglich für Testanwendungen, sollte keine externe Stromversorgung zur Verfügung stehen.

- In der IDE der C-Control PRO muss der entsprechende COM-Port (serielle Schnittstelle) ausgewählt werden. Auch die USB-Programmierung erfolgt über die serielle Schnittstelle der C-Control PRO32 UNIT. Kontrollieren Sie ggf. vorher im Gerätemanager von Windows, welche COM-Ports zur Verfügung stehen oder welcher vom RS232-USB-Converter installiert wurde.

- Wird der I2C-Bus verwendet, müssen die Jumper JP2 und JP1 gesteckt werden, sofern keine externen Pull-Up-Widerstände von Ihnen vorgesehen sind.



- Der Unit-Bus dient zum Anschluss von I2C-Bus-Erweiterungsmodulen der CC1-Familie und kann auch für eigene Anwendungen genutzt werden. Die Schnittstellenbelegung dazu finden Sie in der Abbildung.

- Die Ports der MEGA32 UNIT sind über die Stiftleisten J1, J5, J6 und J7 herausgeführt.

- Bevor man ein Programm in die UNIT übertragen kann, muss der Taster (BOOT/STOP) betätigt werden, um die C-Control PRO32 in den Programmiermodus zu schalten.

- Beim Anlegen der Versorgungsspannung läuft das im Programmspeicher der C-Control MEGA32 abgelegte User-Programm automatisch ab. Dieses kann mit dem Taster (BOOT/STOP) angehalten werden, dadurch befindet sich die C-Control PRO32 im BOOT-Modus, der zur Programmübertragung benötigt wird.

- Der Programmstart kann über die IDE oder über den Taster (RESET/START) ausgelöst werden.

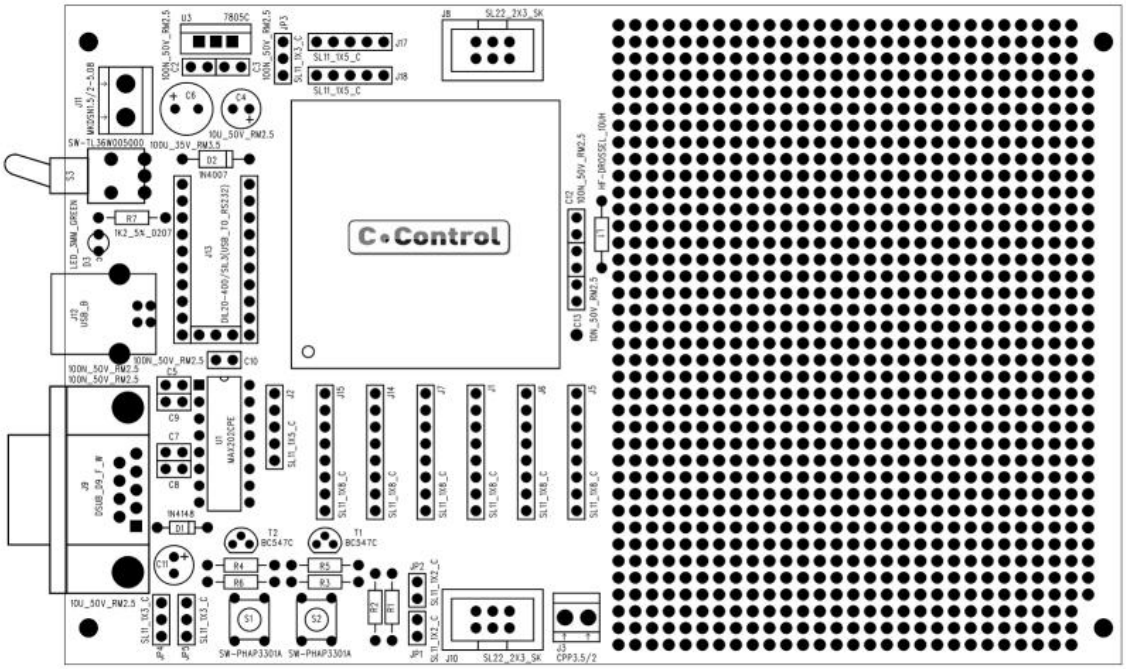
- Werden Variable über Msg_Write… ausgegeben, empfiehlt es sich, den softwaremäßigen Start in der IDE zu verwenden.

Technische Daten
Betriebsspannung: 8 - 16V DC
Stromaufnahme ohne externe Lasten und ohne RS232-USB-Converter: ca. 40mA

Max. zulässiger Dauerstrom aus der stabilisierten 5V-Spannung: 100mA (ohne Kühlung)
Lochrasterfeld: 2,54mm
Bereich der zulässigen Umgebungstemperatur: 0°C bis 70°C
Bereich der zulässigen relativen Umgebungsluftfeuchte: .. 20-60% nicht kondensierend
Abmessungen: 60*100*21mm (inkl. MEGA32 UNIT)

2.1.9 Mega128 Projectboard

Das „C-Control PRO 128 Projectboard“ (Conrad Artikel-Nr. 197313) stellt eine günstige Alternative zum Applikationsboard „MEGA128“ (Conrad-Best.-Nr. 198258) dar. Es ist vom Funktionsumfang her zum „C-Control PRO Applikation-Board“ deutlich eingeschränkt und dient hauptsächlich für eigene Hardware Entwicklungen in Verbindung mit der „MEGA128 UNIT“ und „MEGA128CAN UNIT“. Das Projectboard bietet zusätzlich eine Schraubklemme „J3“ an, die den CAN-Bus der „MEGA128CAN“ herausführt. Auf dem Projectboard kann wahlweise die „MEGA128“ oder die „MEGA128CAN“ verwendet werden. Das „Projectboard PRO 128“ enthält die wichtigsten Komponenten, die zum Betrieb der „MEGA128 UNIT“ benötigt werden. Weiterhin ist auf dem Projectboard eine Spannungsversorgung (USB/Netzadapter), Schnittstellenwandler (RS232) und ein großes Lochrasterfeld für eigene Entwicklungen vorhanden. Standardmäßig ist das Projectboard für die Programmierung über RS232 ausgelegt. Optional kann über den RS232-USB-Converter (Conrad-Best.-Nr. 197257) die Programmierung der UNIT auch über USB geschehen. Die Programmierung erfolgt in diesen Fall ebenfalls über die serielle Verbindung der UNIT (UART), deshalb ist die Programmübertragung nicht so schnell wie die USB-Übertragung auf dem „Applikation-Board MEGA128“.



- Die „MEGA128 UNIT“ wird so aufgesteckt, daß der Schriftzug der UNIT zu lesen ist, wenn die Taster (RESET/RUN & BOOT/STOP) zu Ihnen zeigen.

- Im Basiszustand ohne RS232-USB-Converter sind die Jumper JP4/JP5 so zu stecken, wie auf der Abbildung auf der nächsten Seite zu sehen ist.

➔ Wird der RS232-USB-Converter verwendet (nicht im Lieferumfang), so müssen diese Jumper auf USB umgesteckt werden.

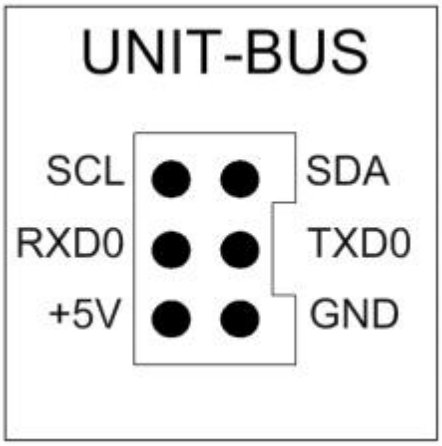
- Der Jumper JP3 dient zur Auswahl der Versorgungsspannung. Bei Jumperstellung „Netz“ werden die Klemmen J11 zur Versorgung benutzt (Labornetzteil oder stabilisiertes Steckernetzgerät, min. 100mA, je nach Applikation). Wird der Jumper JP3 auf USB umgesteckt, kann das Board auch über die USB-Stromversorgung des Rechners betrieben werden.

➔ Achtung! Eine maximale Stromaufnahme von 100mA über USB darf nicht überschritten werden!

- Der Schalter S3 sowie die Stromversorgungsstiftleisten J17/J18 und die Stifte für Vcc/GND auf den Lochrasterfeld sind bei USB-Betrieb nicht mehr unter Spannung. Diese Versorgung dient lediglich für Testanwendungen, sollte keine externe Stromversorgung zur Verfügung stehen.

- In der IDE der „C-Control PRO“ muss der entsprechende COM-Port (serielle Schnittstelle) ausgewählt werden. Auch die USB-Programmierung erfolgt über die serielle Schnittstelle der „MEGA128 UNIT“. Kontrollieren Sie ggf. vorher im Gerätemanager von Windows, welche COM-Ports zur Verfügung stehen oder welcher vom RS232-USB-Converter installiert wurde.

- Wird der I2C-Bus verwendet, müssen die Jumper JP2 und JP1 gesteckt werden, sofern keine externen Pull-Up-Widerstände von Ihnen vorgesehen sind.



- Der Unit-Bus dient zum Anschluß von I2C-Bus-Erweiterungsmodulen der CC1-Familie und kann auch für eigene Anwendungen genutzt werden. Die Schnittstellenbelegung dazu finden Sie in der Abbildung.

- Die Ports der „MEGA128 UNIT“ sind über die Stiftleisten J1, J2, J5, J6, J7, J14 und J15 herausgeführt.

➔ Weitere Informationen zu den genauen Eigenschaften der Ports finden Sie in den Unterlagen zur Anleitung/Hilfe-Datei der „C-Control PRO“.

- Bevor man ein Programm in die UNIT übertragen kann, muss der Taster (BOOT/STOP) betätigt

werden, um die „MEGA128 UNIT“ in den Programmiermodus zu schalten.

- Beim Anlegen der Versorgungsspannung läuft das im Programmspeicher der „MEGA128 UNIT“ abgelegte User-Programm automatisch ab. Dieses kann mit dem Taster (BOOT/STOP) angehalten werden, dadurch befindet sich die „MEGA128 UNIT“ im BOOT-Modus, der zur Programmübertragung benötigt wird.

- Der Programmstart kann über die IDE oder über den Taster (RESET/START) ausgelöst werden. Werden Variable über „Msg_Write…“ ausgegeben, empfiehlt es sich, den softwaremäßigen Start in der IDE zu verwenden.

Technische Daten
Betriebsspannung: 8 - 16 V/DC
Stromaufnahme ohne externe Lasten und ohne RS232-USB-Converter: ca. 50 mA
Max. zulässiger Dauerstrom aus der stabilisierten 5V-Spannung: 100 mA (ohne Kühlung)
Lochrasterfeld: 2,54 mm
Bereich der zulässigen Umgebungstemperatur: 0 °C bis +70 °C
Bereich der zulässigen relativen Umgebungsluftfeuchte: .. 20 - 60% nicht kondensierend
Abmessungen: 160 x 100 x 23 mm (inkl. „MEGA128 UNIT“ oder „MEGA128CAN UNIT“)

2.2 AVR32Bit

Es wird die Hardware vorgestellt, die bei der C-Control Pro AVR32Bit Serie zur Anwendung kommt. Weitere Abschnitte erklären Aufbau und Funktion des zugehörigen Application Boards.

2.2.1 Installation

Dieses Kapitel beschreibt die Installation der C-Control Pro AVR32Bit Unit auf dem Applicationboard, und die Installation der Software und USB Treiber.

➔ Im Auslieferungszustand ist der Autostart Jumper gesetzt. Bitte abziehen, da sonst keine Übertragung möglich ist.

2.2.1.1 Software

Die aktuelle Entwicklungssoftware, Beispielprogramme sowie das Handbuch und nützliche Informationen finden Sie unter: www.c-control.de im Bereich "C-Control Pro" unter "IDE und Treiber Download". Das Handbuch gibt es auch als Hilfedatei in der Entwicklungsumgebung der C-Control PRO IDE und als PDF im Installationsordner der C-Control Pro unter Manual.

Direkter IDE Download Link: http://www.c-control-pro.de/updates/C-ControlSetup.exe

➔ 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. Am 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.

MAC Adresse

Um Verbindungsprobleme zu vermeiden sollte vor dem Einschalten der Ethernet Unterstützung die MAC-Adresse in der C-Control Konfiguration auf einen neuen Wert eingestellt werden. Zu diesem Zweck wird für jede C-Control Pro AVR32Bit eine eigene MAC-Adresse erzeugt und auf einem Aufkleber mitgeliefert. Diesen Aufkleber finden Sie auf der Unterseite der UNIT.

Siehe Abbildung:

2.2.1.2 USB

Treiberinstallation

- Verbinden Sie nun die UNIT mit den mitgelieferten Mini-USB Kabel mit dem PC (Dieses Kabel liegt dem Applicationboard oder Mainboard bei). Der PC versucht nun einen Treiber für ein "C-Control Pro AVR32" Device zu installieren. Den passenden Treiber finden Sie unter USB Driver\AVR32 USB Driver im Installationsverzeichnis der C-Control Pro IDE.
- Sind alle Verbindungen getätigt, so starten Sie nun die Entwicklungsumgebung.
- In der IDE der C-Control Pro muss der entsprechende COM-Port (virtuelle serielle Schnittstelle) ausgewählt werden. Kontrollieren Sie ggf. vorher im Gerätemanager von Windows, welche COM-Port Nummer vergeben wurde (siehe Abbildung).

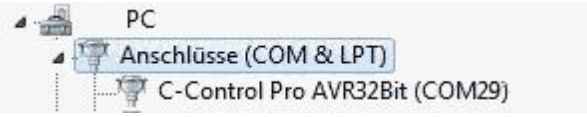
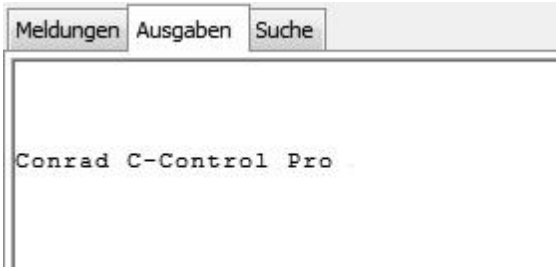


Bild Gerätemanager Comport

➔ Treiber und Software der C-Control Pro unterstützen kein Windows Betriebssystem vor Windows 2000.

Drücken Sie nun auf der C-Control PRO AVR32Bit UNIT den Reset-Taster. In der Ausgabe der IDE muss nun folgende Meldung erscheinen:



C-Control PRO IDE Ausgabe nach erfolgreicher Installation der UNIT

Nun können Sie bereits ein Programm, z. B. eines der Beispiele, auf die UNIT übertragen. Die Beispielprogramme finden Sie, wenn Sie in der IDE unter „Hilfe“ auf „Demo Programme“ klicken.

2.2.2 Firmware

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

- *Bootloader*
- *Interpreter*

Bootloader

Der Bootloader ist immer verfügbar. Er startet den Interpreter oder führt ein Upload durch, wenn eine neue Interpreterversion verfügbar ist.

- Ein Power-On Reset (Aus- Einschalten am Schalter) bringt das AVR32Bit Modul immer zuerst in den Bootloader (wenn der Autostart-Jumper nicht gesetzt ist). Dies ist ein Sicherheitsfeature, um immer Zugang zu ermöglichen, auch wenn der Interpreter fehlerhaft arbeiten sollte. In dem Zustand kann die UNIT immer über "Modul zurücksetzen" in den Originalzustand gebracht werden.
- Ein Druck auf den Reset Taster bringt das Modul direkt vom Bootloader in die Firmware, wenn ein gültiger Interpreter geladen ist. Dadurch wird im normalen Entwicklungsbetrieb die Anzahl der USB Treiber Unterbrechungen minimiert.

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.

2.2.2.1 Autostart

Autostart

Ist der Autostart-Jumper (J1 auf AVR32Bit UNIT) gesteckt, so startet das Userprogramm nach einem Reset oder Power On direkt. Da der Autostart-Jumper die Verbindung zum Start/Stop Taster dauerhaft brückt, hat der Start/Stop Taster keine Wirkung mehr, wenn der Jumper gesetzt ist.

➔ Die Bibliotheksfunktion ForceBootloader(), sowie eine Änderung der AVR32Bit Unit Optionen mit "C-Control Konfiguration" in der IDE führen zu einem internen Reset, bei dem der Autostart ignoriert wird. Dies geschieht mit Absicht um eine Fernwartung möglich zu machen. Das Userprogramm kann aber über die IDE gestartet werden, oder ein Druck auf den Reset-Taster löst den Autostart wieder aus.

➔ Im Auslieferungszustand ist der Autostart Jumper gesetzt. Bitte abziehen, da sonst keine Übertragung möglich ist.

Fernwartung

Um eine Applikation mit der AVR32Bit aus der Ferne zu warten, kann die Applikation mit ForceBootloader() in den Bootloader springen. Ein Update der Applikation würde aber bei gesetztem Autostart-Jumper gleich wieder das Programm starten. Man kann deshalb nach ForceBootloader in der C-Control Konfiguration die Option Autostart verhindern aktivieren. Danach die Applikation updaten und gewünschte Änderungen an den Optionen durchführen. Am Ende Autostart verhindern zurücksetzen und von der IDE die Applikation starten.

2.2.2.2 USB Troubleshooting

Der USB Support der C-Control Pro AVR32Bit wird durch den Prozessor selbst ausgeführt, und nicht durch einen externen Chip, wie z.B. auf dem C-Control Pro Mega Applicationboard. Dies ist soweit problematisch, da das Windows Betriebssystem nicht immer Unterbrechungen des USB Systems korrekt verarbeitet. Man merkt das im Alltag, wenn hin- und wieder ein USB-Stick, eine USB-Harddisk oder USB-Seriell Wandler erst beim zweiten Anstecken funktioniert. Um dem entgegen zu wirken, sind mehrere Maßnahmen getroffen worden, um die Anzahl der USB Neustarts zu minimieren:

- Die C-Control Pro AVR32Bit Unit bleibt so lange wie möglich in der Firmware und springt seltener in den Bootloader als die C-Control Pro Mega Units.
- Man kann mit dem Start/Stop Taster die Unit anhalten, ohne einen Reset durchführen zu müssen.
- Ein Druck auf den Reset Taster überspringt die USB Initialisierung im Bootloader, und startet direkt die Firmware. Nur ein Power-On Reset belässt das AVR32Bit Modul im Bootloader (wenn der Autostart-Jumper nicht gesetzt ist).

➔ In seltenen Fällen kann es vorkommen, das die Unit bei einem Power-On nicht erkannt wird. Dies kann man im Device Manager erkennen, wenn dort beim Einschalten der Unit der C-Control AVR32Bit COM Port nicht im Gerätemanager auftaucht. Dann bitte (falls vorhanden) den USB-Hub ab- und anstecken, oder falls das nicht hilft, einen Restart von Windows durchführen. Danach wird die C-Control Pro Unit wieder erkannt.

➔ Wird das Benutzerprogramm durch einen Autostart direkt gestartet, so wird keine Meldung "Interpreter gestartet" ausgegeben. Das liegt daran, dass das USB-Subsystem bis zu 2 Sekunden benötigt um den virtuellen COM Port zu aktivieren. Da das Benutzerprogramm aber sofort los läuft, gehen alle Ausgaben der ersten 2 Sekunden verloren. Auch Debugausgaben sind in dieser Zeit bei aktivem Autostart nicht sichtbar. Ein Start des Programms durch den Start-Taster, wenn die Unit im Bootloader ist (z.B. nach einem Power-On Reset), verhält sich wie ein Autostart. Deswegen gibt es auch dort in den ersten 2 Sekunden keine Ausgaben.

IDE antwortet nicht

Wird bei der Abarbeitung der Programme auf der AVR32Bit fremder Speicher überschreiben, kann das Auswirkungen bis auf die IDE haben. In diesem Fall wird das USB CDC Protokoll vom AVR32 nicht mehr fehlerfrei ausgeführt, und der virtuelle COM Port auf dem PC kann in einen blockierenden Zustand geraten, der der IDE nur mit Verzögerungen (Timeouts) erlaubt die Daten entgegenzunehmen. Die IDE arbeitet dann nicht mehr richtig. Im Normalfall lässt sich mit einem Druck auf den Reset Taster des AVR32Bit Moduls die IDE aus dieser Situation zu befreien, manchmal hilft es aber nur, die IDE mit dem Task Manager zu beenden.

2.2.3 Modul

Die C-Control Pro AVR32Bit UNIT (Conrad Best.-Nr.: 192573) ist die derzeit schnellste Mikrocontroller-Einheit der C-Control Pro Familie (Atmel AT32UC3C1512C). Die Unit ist mit einem leistungsstarken AVR32 32-Bit DSP-Mikrocontroller mit FPU (Floating Point Unit) zur Berechnung von Fließkommazahlen ausgestattet. Dieser Mikrocontroller wurde speziell für Industrielle und Automotive Anwendungen konzipiert und erfüllt so einen hohen Standard an Leistung und Zuverlässigkeit. Die C-Control Pro AVR32Bit UNIT verfügt über eine umfangreiche Ausstattung an Peripherie, so ist bereits ein Webserver, CAN-, µSD-, USB-Interface uvm. zur Programmierung und Debugging auf dieser kleinen Einheit enthalten.

Zum Betrieb der UNIT benötigen Sie nur eine stabilisierte 3.3V / 200mA Stromversorgung und ein Mini-USB Kabel um die UNIT mit Ihrem PC zu verbinden. Am einfachsten funktioniert dies zu Entwicklungszwecken über das optional erhältliche Applicationboard (Conrad Best.-Nr.: 192587). Dieses Board wurde speziell für die Soft- & Hardware Entwicklung konstruiert und stellt eine Vielzahl an zusätzlicher Peripherie bereit.

Für Rapid Prototyping und Kleinserien kann auch das AVR32Bit Mainboard (Conrad Best.-Nr.: 192702) verwendet werden. Dieses Board kann mit Zusatzboards je nach Applikation erweitert werden.

Die Programmierung der AVR32Bit UNIT erfolgt in der bereits seit mehreren Jahren bewährten und stetig weiterentwickelten C-Control Pro Entwicklungsumgebung in Basic, CompactC und Graphisch.

Die UNIT bietet folgende Features:

- Leistungsstarker 32 Bit Mikrocontroller (91MIPS intern) 66 MHz Takt
- 512 KB High Speed FLASH (160 KB reserviert für Interpreter)
- 64 KB High Speed SRAM (14 KB reserviert für Interpreter)
- 1x CAN-Bus (2.0A & 2.0B) mit CAN Treiber + jumperbaren Abschlusswiderstand
- 2x SPI-Schnittstellen
- 1x I2C-Bus (TWI)
- 2x Referenzspannungseingang für ADC

- 1x 16 Kanal 12 Bit ADC
- 1x USB-Interface (Mini-USB) zum Programmieren und Debuggen
- 3x USART-Interface (serielle-Schnittstelle)
- 1x Externes I2C EEPROM 512 KBit
- 1x Real Time Clock (RTC) mit 32.768 kHz Uhrenquarz
- 1x LAN-Interface (LAN Buchse extern)
- 1x µSD-Karten Halter (unterstützt SDHC)
- 1x Referenzsspannungseingang für DAC
- 2x Analogkomparator
- 1x 4 Kanal 20 Bit PWM Controller
- 2x 16 Bit Timer mit 3 Kanälen
- 7x Interrupteingänge
- 57x digitale Ein- Ausgänge (je nach Verwendung der anderen Funktionen)
- Autostart optional über Jumper auswählbar
- Start- Stopptaster
- Reset-Taster
- 2 Stiftleisten mit je 2x23 Pins im Rastermaß 2.54mm
- Pinout im Rastermaß 2.54mm, auch ideal für Lochrasterplatinen

Schema der AVR32Bit Unit

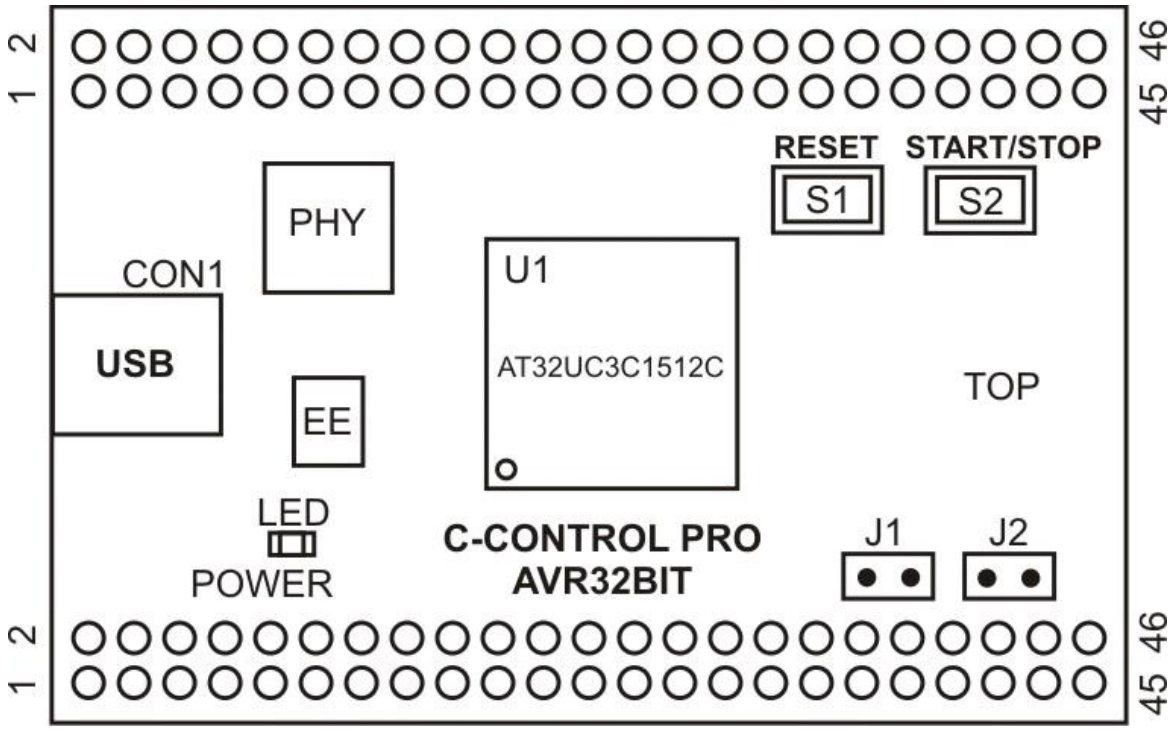


Bild UNIT (Sicht von oben)

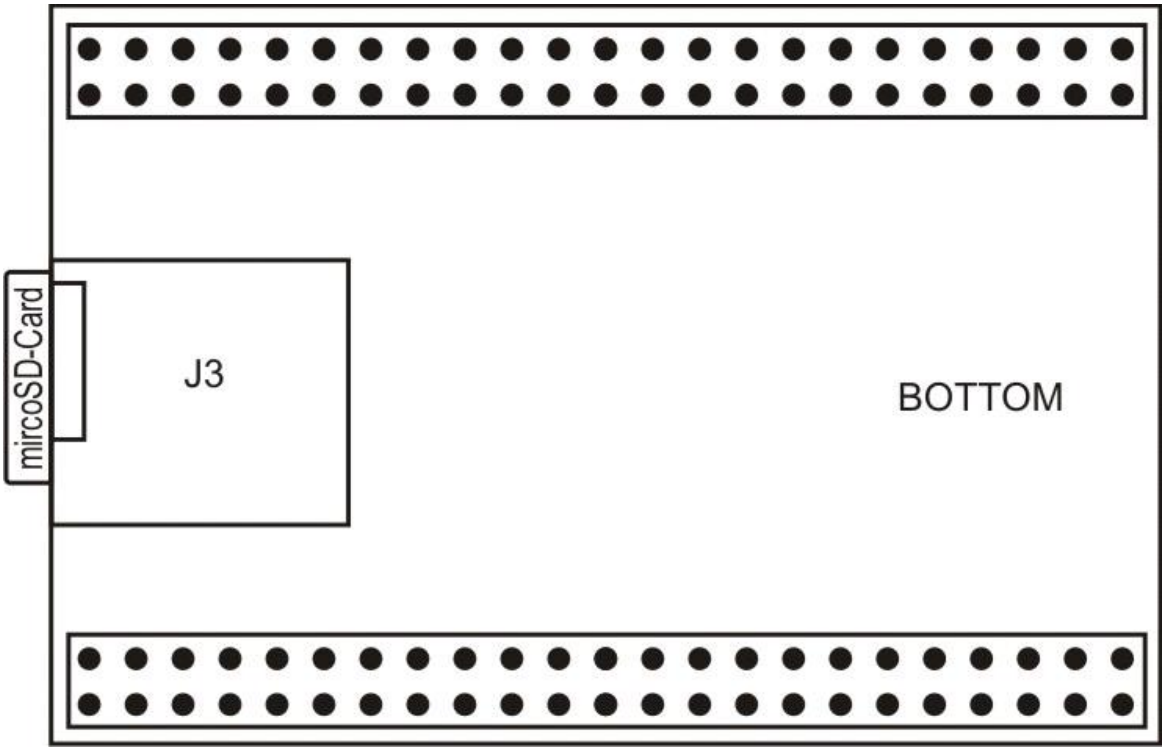


Bild UNIT (Sicht von unten)

Jumper:
J1: Autostart der Userapplikation
J2: CAN 120Ohm Abschlusswiderstand aktiviert

Pinlayout des Moduls

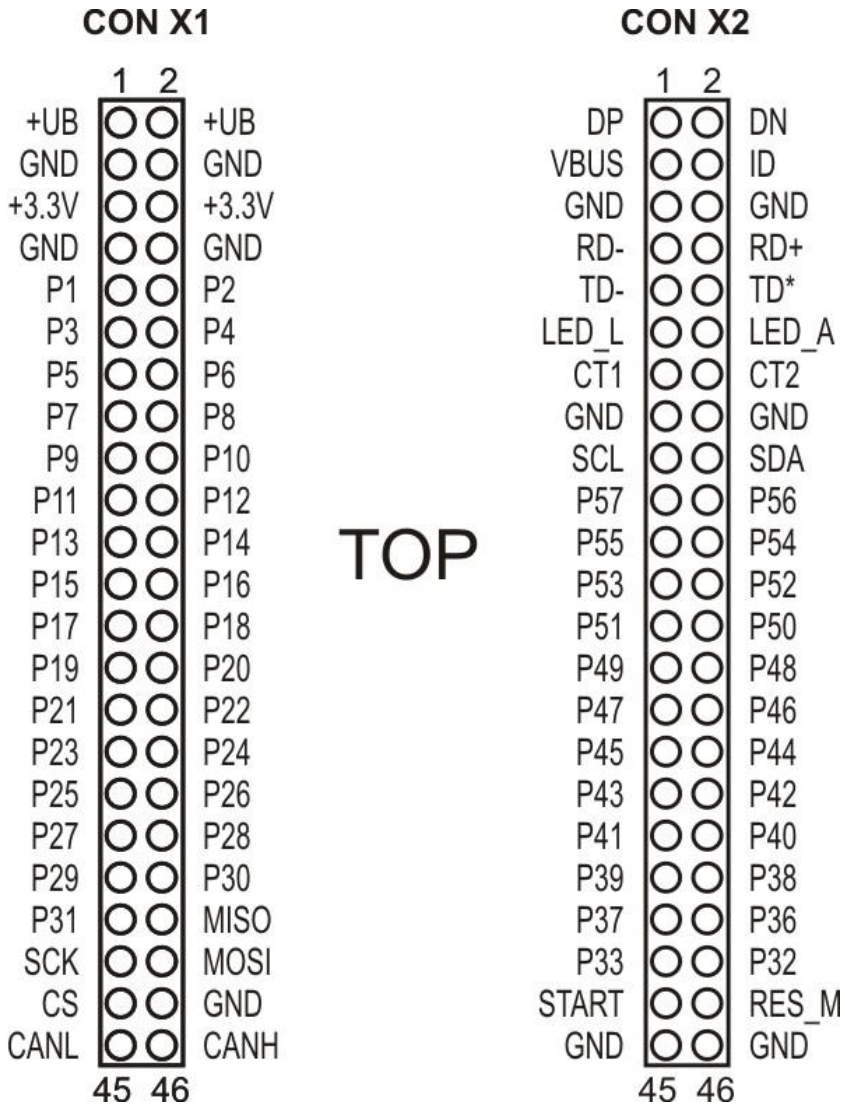


Bild UNIT Pinout

➔ Eine Portübersicht finden Sie im Kapitel Pinzuordnung

Stromversorgung

An den UNIT Pins CON X1 3.3V und GND muss eine stabilisierte Versorgungsspannung angeschlossen werden. Die jeweils vier 3.3V und GND Pins sind untereinander verbunden! Die LED „POWER“ signalisiert, dass die UNIT mit Strom versorgt wird.

➔ **Die C-Control PRO UNIT besitzt keinen Verpolungsschutz, daher wird die UNIT bei verkehrter Polung der Stromversorgung zerstört!**

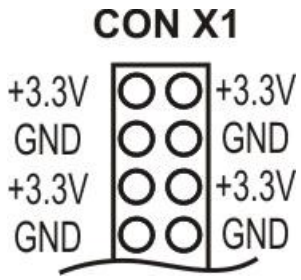


Bild UNIT
Stromversorgung

USB

Über die Mini-USB Buchse wird das C-Control Pro AVR32Bit Modul mit dem PC verbunden. Der USB-Anschluss dient zur Programmierung und zum Debuggen der Usersoftware. Die C-Control Pro UNITs besitzen alle einen Debugger. Über diesen können Breakpoints gesetzt werden und Variablen zur Laufzeit überwacht und analysiert werden.

➔ **Das Modul wird nicht über USB mit Strom versorgt!**

Reset

Ein Reset bewirkt die Rückkehr des Microcontrollersystems in einen definierten Anfangszustand. Das C-Control Pro AVR32Bit Modul kennt grundsätzlich 3 Reset-Quellen:

- Power-On-Reset: Wird automatisch nach dem Einschalten der Betriebsspannung ausgeführt. Die UNIT befindet sich danach wieder im Bootloader-Modus. Es kann das Modul zurückgesetzt werden oder ein neuer Interpreter auf die Unit übertragen werden.
- Brown-Out -Reset: Wird automatisch ausgeführt, wenn die Core-Spannung kleiner 1.65V ist. Dadurch wird verhindert, dass der Controller bei Absinken der Versorgungsspannung in undefinierte Zustände gerät. Ist die Spannung wieder deutlich höher, dann startet das Modul von neuem.
- Hardware-Reset: Wird ausgeführt wenn der RESET-Taster des Moduls gedrückt wurde.

Start/Stop-Taster

Mit dem Start/Stopp-Taster wird das Programm gestartet. Bei einem erneuten Druck wird das Programm gestoppt. Ein Stopp mit diesem Taster ist einem Reset vorzuziehen, da bei einem Reset das USB-Subsystem wieder komplett neu gestartet und die Verbindung neu ausgehandelt wird. Ist der Autostart-Jumper (J1) gesteckt, so wird die Applikation nach einem Reset direkt gestartet und der Taster bleibt dann ohne Wirkung.

Autostart

Ist der Autostart-Jumper (J1) gesteckt, so startet das Userprogramm nach einem Reset sofort neu.

➔ Im Auslieferungszustand ist der Autostart Jumper gesetzt. Bitte abziehen, da sonst keine Übertragung möglich ist.

Takterzeugung

Die Takterzeugung des Mikrocontrollers erfolgt durch einen 12 MHz-Quarz. Im Controller werden die 12 MHz über den PLL-Oszillator auf 66 MHz hochgetaktet. Alle zeitlichen Abläufe des Controllers, wie auch die 48Mhz des USB-Subsystems sind von dieser Taktfrequenz abgeleitet.

Real-Time Clock

Die C-Control Pro AVR32Bit UNIT besitzt einen separaten Oszillator mit einen 32.768 kHz Uhrenquarz. Diese genaue Quarzuhr kann per Software gestellt und ausgelesen werden. Diese Uhr ist ideal für zeitpräzise Anwendungen wie Zeitschaltuhren usw.

Digitalports

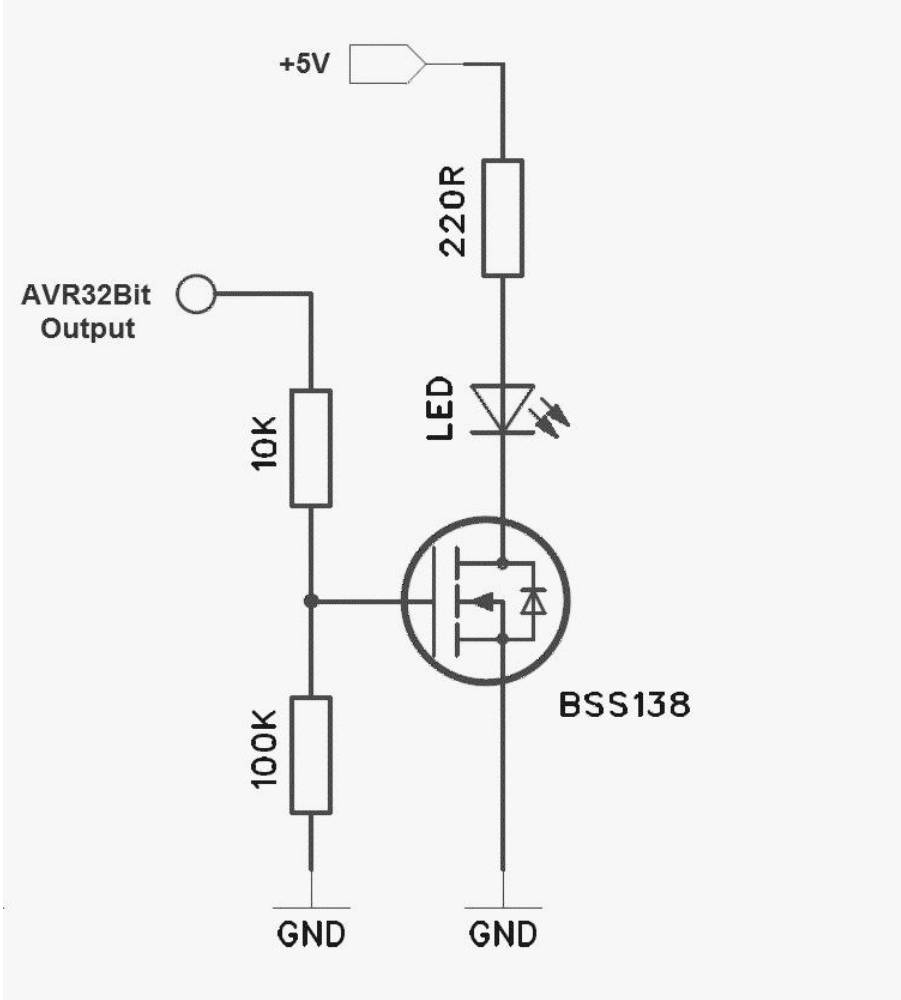
Das C-Control Pro AVR32Bit Modul verfügt über 57 digitale Ein- Ausgänge, die je nach Konfiguration der Sonderfunktionen wie PWM, ADC usw. verwendet werden können. An den digitalen Ein- Ausgängen können z.B. Taster mit Pull-up/down-Widerständen, Digital-ICs, Optokoppler oder Treiberschaltungen für Relais angeschlossen werden. Die Pins werden einzeln, d.h pinweise angesprochen. Jeder Pin kann entweder Eingang oder Ausgang sein.

➔ **Niemals zwei Pins direkt zusammenschalten, die gleichzeitig als Ausgang arbeiten sollen! Dies kann die C-Control Pro AVR32Bit UNIT zerstören!**

Digitale Eingangspins sind hochohmig oder mit internem Pull-up/down-Widerstand beschaltet und überführen ein anliegendes Spannungssignal in einen logischen Wert. Voraussetzung dafür ist, dass sich das Spannungssignal innerhalb des definierten Bereichs für Low- oder Highpegel befindet. In der weiteren Verarbeitung im Programm werden die logischen Werte von einzelnen Eingangspins als 0 ("Low") oder -1 ("High") dargestellt. 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).

➔ **Niemals eine größere Spannung als 3.6V an einen der Pins der C-Control Pro AVR32Bit UNIT anschließen!**

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 AVR32Bit Moduls führen. Nach dem Reset ist zunächst jeder Digitalpin als Eingang konfiguriert. Über bestimmte Befehle kann die Datenrichtung umgeschaltet werden.



Da die Ausgänge der AVR32Bit UNIT nicht sonderlich belastet werden können, sollte immer eine kleine Treiberstufe (siehe Bild) nachgeschaltet werden. Im Beispiel wird eine LED angesteuert, je nach Verbraucher muss ein entsprechender FET oder Transistor verwendet werden. Diese Schaltung wird für Verbraucher bis 100mA eingesetzt. Bei induktiven Lasten muss parallel zum Verbraucher eine Freilaufdiode zugeschaltet werden.

Viele nützliche und interessante Informationen zu diesem Thema, finden Sie unter folgenden Link:
http://www.mikrocontroller.net/articles/Relais_mit_Logik_ansteuern

➔ **Bei der C-Control Pro AVR32Bit UNIT werden die Pins nicht mehr über „Port_DataDir“ bzw. „Port_DataDirBit“ konfiguriert! Da die AVR32Bit UNIT mehr Möglichkeiten bietet die Pins zu konfigurieren, wurde hier der Befehl „Port_Attribute“ eingeführt.**

➔ **Es ist wichtig, vor der Programmierung die Pinzuordnung der AVR32Bit zu studieren, da wichtige Funktionen der Programmentwicklung (z.B. LAN, USB) auf bestimmten Pins liegen.**

ADC-Referenzspannung

Der Mikrocontroller verfügt über einen Analog-Digital-Wandler mit einer umschaltbaren Auflösung von 8/10/12 Bit. Das heißt, gemessene Spannungen können maximal als ganze Zahlen von -2048 bis 2048 dargestellt werden, da der A/D-Wandler immer differentiell arbeitet. Zudem kann die Verstärkung des ADC-Vorverstärkers von 1, 2, 4, 8, 16, 32, 64 per Software eingestellt werden.

Folgende Referenzspannungsquellen stehen zur Verfügung:

- 0,6 * VDDANA intern (0,6 * 3.3V = 1,98V)
- interne Referenzspannung von 1V
- zwei externe Referenzspannungseingänge, z.B. 2.048V, durch Referenzspannungs-IC erzeugt

Ist „x“ ein digitaler Messwert, dann errechnet sich der entsprechende Spannungswert „u“ wie folgt: Die Auflösung ist von der Konfiguration des ADCs abhängig.

Auflösung	Maximaler Wert
8 Bit	-128 bis +127
10 Bit	-512 bis +511
12 Bit	-2048 bis +2047

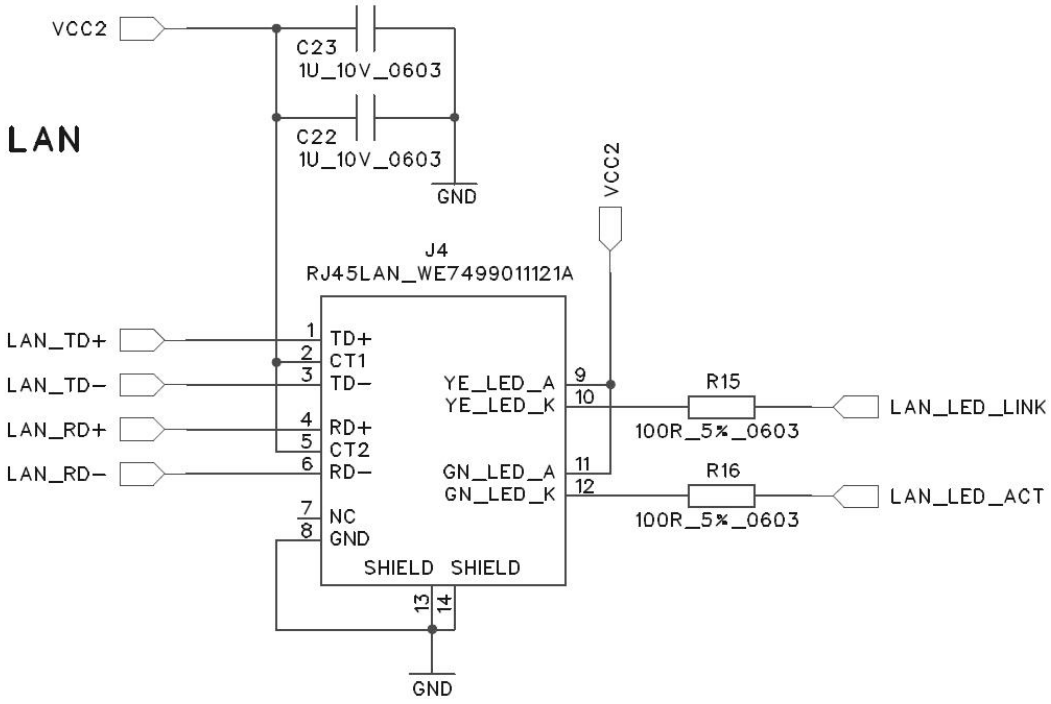
Formel zur Berechnung der anliegenden ADC-Spannung:
u = x * Referenzspannung / Auflösung

CAN-Abschlusswiderstand

Über den Jumper (J2) wird der 120 Ohm Abschlusswiderstand für den CAN-Bus aktiviert. Mehr Informationen zum CAN-BUS und seine Eigenschaften finden Sie in der Rubrik CAN Bus!

LAN

An den Pins VCC2 (+3,3 V), GND, RD-, RD+, RD-, TD+, LED_L und LEDA sind die Anschlüsse für die LAN-Buchse herausgeführt. Auf dem Applicationboard bzw. Mainboard ist bereits eine LAN-Buchse vorhanden, die fest mit den Anschlüssen der UNIT verdrahtet ist. Innerhalb eigener Applikationen in deren die UNIT „stand alone“ eingesetzt wird, kann eine Ethernet-Buchse selber nach unten aufgeführtem Schaltplan nachgerüstet werden.



Technische Daten

Stromversorgung VCC	3 bis 3.6V Nominal 3.3V / >200mA (stabilisiert)
Maximale Spannung an den Pins	-0.3V bis 3.6V
Maximaler Strom an allen Pins (Summe)	120mA
R_{PULLUP} I/O	5 bis 26 kOhm
$R_{PULLDOWN}$ I/O	2 bis 16 kOhm
Input Low-level voltage I/O	0.3 * VCC
Input high-level voltage I/O	0.7 * VCC
Output low-level voltage I/O	-3.5 bis -14mA je nach Konfiguration(1)
Output high-level voltage I/O	3.5 bis 14mA je nach Konfiguration(1)
Abweichung RTC	+/- 20ppm
Umgebungstemperatur (Ta)	0 bis 70°C
Abmessungen	60x40x8mm (ohne Stiftleiste)
Gewicht	ca. 18g

(1) *-3.5/ 3.5mA fähige Pins sind*: **PB02** (PORT57), **PC04** (PORT34), **PC05** (PORT35), **PC06** (PORT33).
-7/ 7mA und -14/ 14mA fähige Pins sind: **PB06** (PORT7), **PB21** (PORT29), **PD02** (SPI0-SCK).

Die verbleibenden Pins von PAxx, PBxx, PCxx, PDxx arbeiten mit Strömen von -3.5/ 3.5 mA bzw. -7/ 7mA. Die Stromstärke der Pins wird über die Anweisung „Port_Attribute“ programmiert.

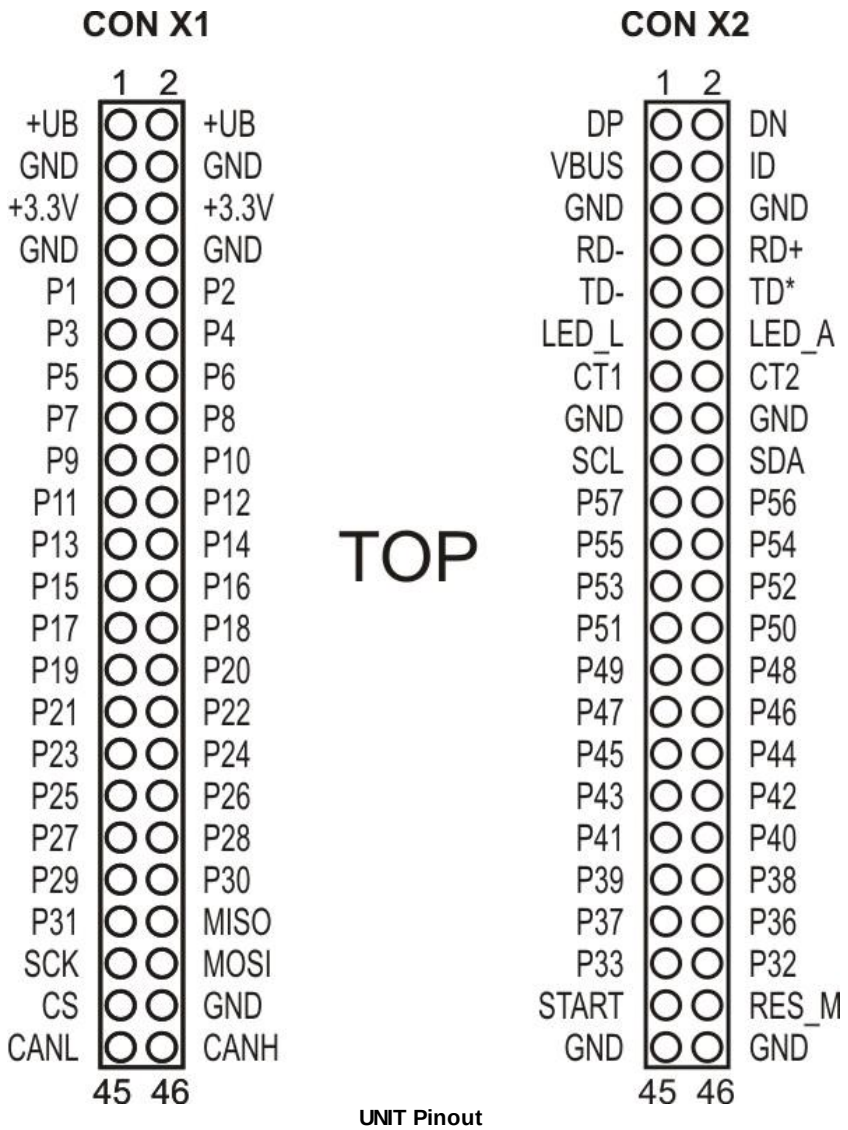
2.2.3.1 Pinzuordnung

PortA bis PortD werden für direkte Pin-Funktionen (z.B. Port_WriteBit) von 0 bis 127 gezählt, siehe "PortBit".

Pinbelegung für C-Control Pro AVR32Bit Unit und Application Board

C-Control Unit Name	C-Control Modul Pin	AVR32 Portname	TQFP 100	GPIO	Function1	Function2	Appl. Board Function
P1	X1.09	PA00	1	0	CAN1-TX		
P2	X1.10	PA01	2	1	CAN1-RX		
P3	X1.11	PA02	3	2			
P4	X1.12	PA03	4	3		Ext Int1	
P5	X1.13	PB04	7	36	SPI1-MOSI		
P6	X1.14	PB05	8	37	SPI1-MISO		
P7	X1.15	PB06	9	38	SPI1-SCK		
P8	X1.16	PA16	22	16		ADCREF0	
P9	X1.17	PA04	10	4	ADC0		
P10	X1.18	PA05	11	5	ADC1		
P11	X1.19	PA06	12	6	ADC2	AC1AP1	
P12	X1.20	PA07	13	7	ADC3	AC1AN1	
P13	X1.21	PA08	14	8	ADC4	Ext Int2 AC1BP1	
P14	X1.22	PA09	15	9	ADC5		
P15	X1.23	PA10	16	10	ADC6	Ext Int4	
P16	X1.24	PA11	17	11	ADC7	ADCREF1	
P17	X1.25	PA19	25	19	ADC8		
P18	X1.26	PA20	28	20	ADC9	AC0AP0	
P19	X1.27	PA21	29	21	ADC10		
P20	X1.28	PA22	30	22	ADC11	AC0AN0	
P21	X1.29	PA23	31	23	ADC12	AC0BP0	
P22	X1.30	PA24	32	24	ADC13		
P23	X1.31	PA25	33	25	ADC14		
P24	X1.32	PA13	19	13	ADC15	AC1AN0	
P25	X1.33	PA12	18	12		AC1AP0	
P26	X1.34	PA14	20	14		AC1BP0	

P27	X1.35	PA15	21	15			
P28	X1.36	PB20	43	52	TIMER0-B		
P29	X1.37	PB21	44	53	COUNTA-1		
P30	X1.38	PB22	45	54	TIMER2-A		
P31	X1.39	PB23	46	55	TIMER2-B		
P32	X2.42	PC01	50	65	TIMER5-A		
P33	X2.41	PC06	57	70	COUNTA-2		
P34	X2.18	PC04	55	68	SDA (I2C)	Ext Int3	
P35	X2.17	PC05	56	69	SCL (I2C)		
P36	X2.40	PC17	65	81	USART3-TX	PWMH_0	
P37	X2.39	PC18	66	82	USART3-RX	PWML_0	
P38	X2.38	PC15	63	79	USART0-RX	PWMH_1	
P39	X2.37	PC16	64	80	USART0-TX	PWML_1	
P40	X2.36	PC19	67	83		PWML_2	
P41	X2.35	PC20	68	84		PWMH_2	PORT_T1
P42	X2.34	PC12	60	76		PWML_3	PORT_T2
P43	X2.33	PC11	59	75	COUNTA-0	PWMH_3	PORT_T3
P44	X2.32	PC13	61	77		Ext Int7	PORT_T4
P45	X2.31	PC14	62	78			PORT_T5
P46	X2.30	PC21	69	85			
P47	X2.29	PC22	70	86			
P48	X2.28	PC23	71	87			PORT_LED1
P49	X2.27	PC24	72	88			PORT_LED2
P50	X2.26	PC31	73	95	TIMER1-B		
P51	X2.25	PD07	78	103	USART4-TX	Ext Int5	
P52	X2.24	PD08	79	104	USART4-RX COUNTB-2	Ext Int6	
P53	X2.23	PD21	88	117			
P54	X2.22	PD22	89	118	TIMER4-A		
P55	X2.21	PD23	90	119			
P56	X2.20	PB19	42	51	TIMER0-A		
P57	X2.19	PB02	99	34	TIMER3-A		
---	X1.43	PD03	77	99			



UNIT Pinout

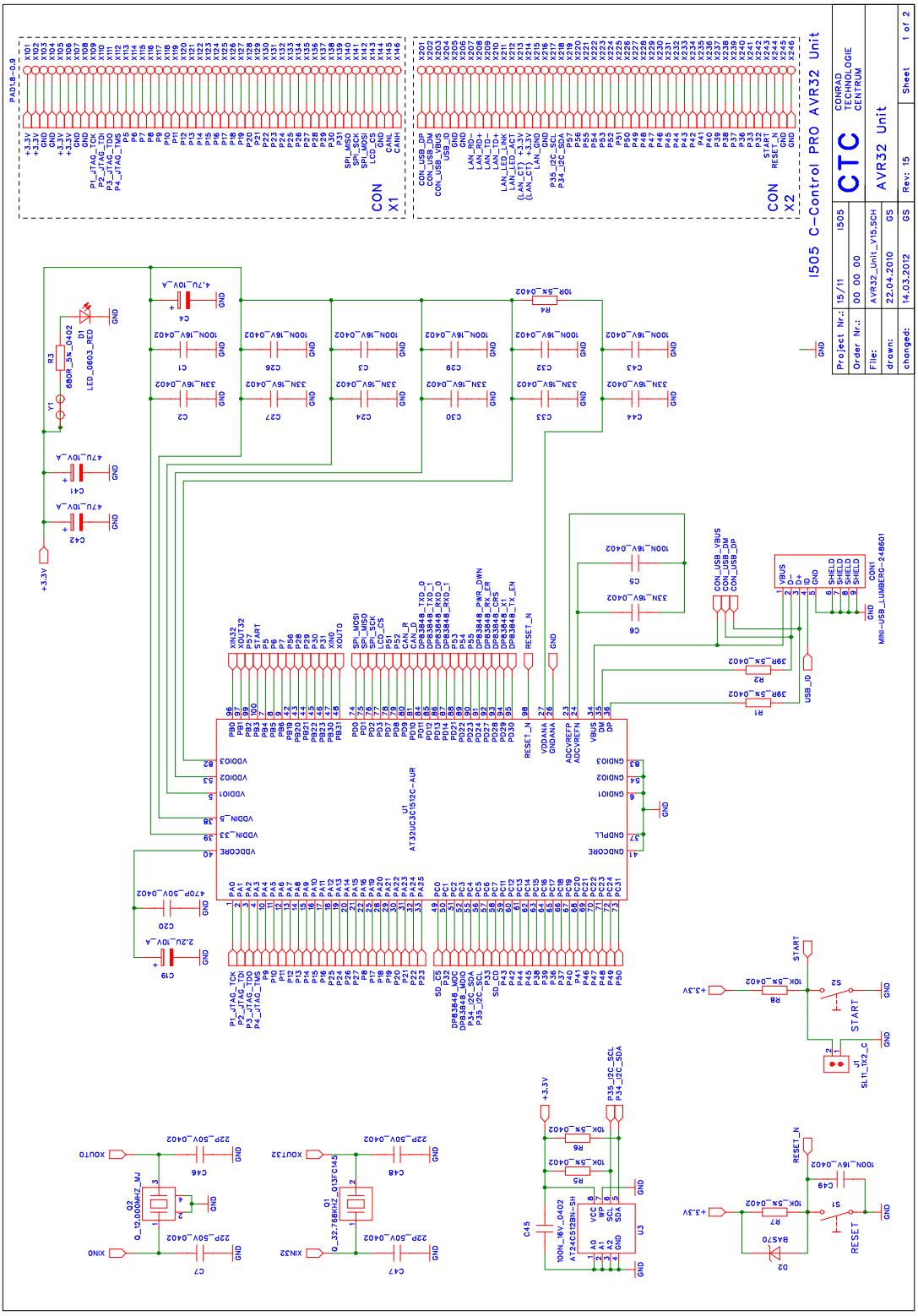
Festverdrahtete Pins

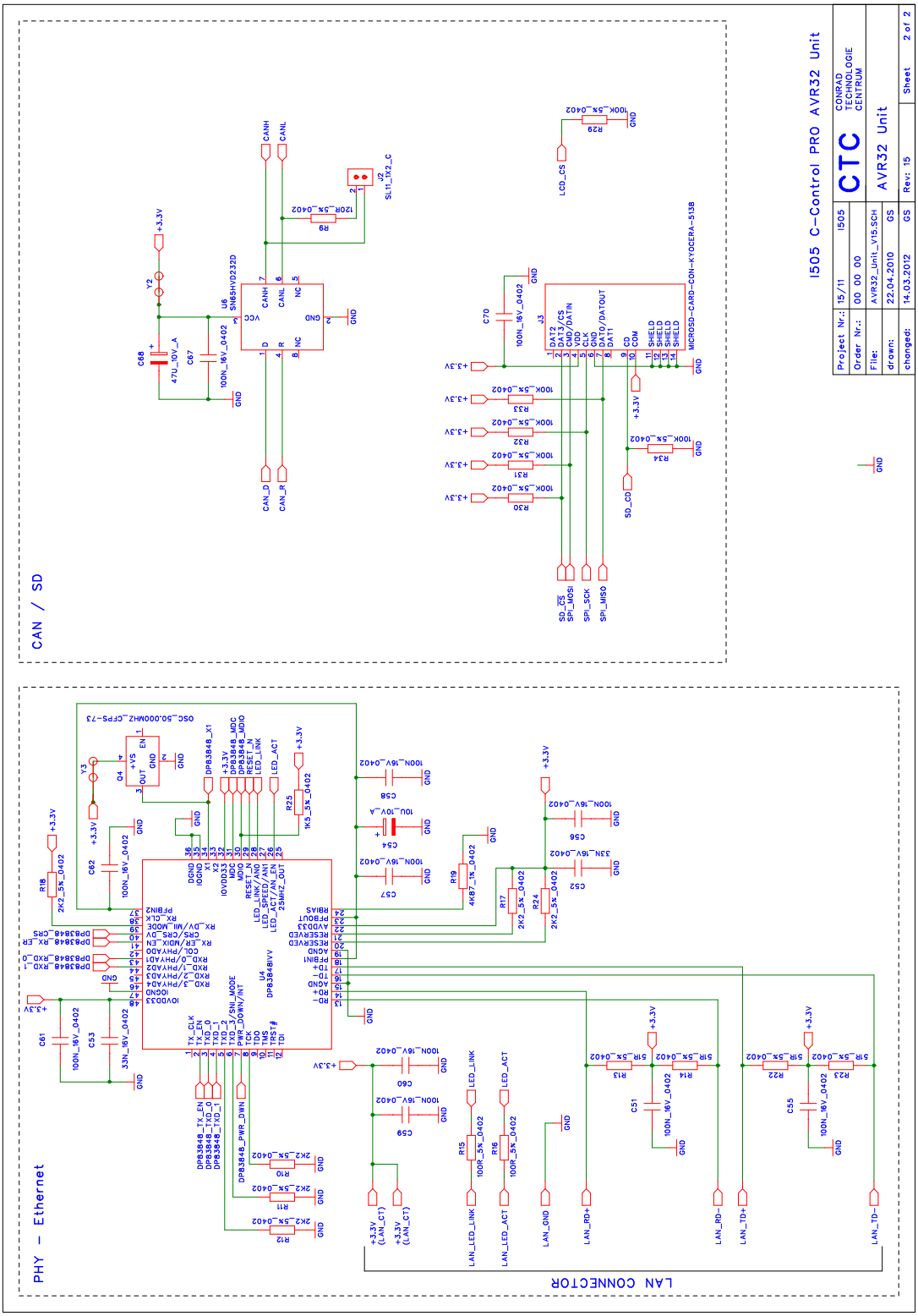
C-Control Modul Pin	TQFP 100	AVR32 Portname	GPIO	Function	I/O
X1.05, X1.06	5	VDDIO1			3,3V
X1.07, X1.08	6	GNDIO1			GND
	23	ADCVREFP			REF
	24	ADCVREFN			REF
	27	VDDANA			3,3V
	26	GNDANA			GND
	37	GNDPLL			GND
	38	VDDIN_5			3,3V

	39	VDDIN_33			3,3V
	40	VDDCORE			3,3V
	41	GNDCORE			GND
	53	VDDIO2			3,3V
	54	GNDIO2			GND
	82	VDDIO3			3,3V
	83	GNDIO			GND
	96	PB00	32	RTC	Xin32
	97	PB01	33	RTC	Xout32
	47	PB30	62	System Clock	Xin0
	48	PB31	63	System Clock	Xout0
X2.03	34	VBUS			USB-Debug
X2.02	35	DM			USB-Debug
X2.01	36	DP			USB-Debug
X2.43	100	PB03	35		Start-Button
X2.44	98	Reset_n			Reset-Button
	58	PC07	71	SD card	Card-Detect
X1.42	74	PD00	96	SPI	SPI0-MOSI
X1.40	75	PD01	97	SPI	SPI0-MISO
X1.41	76	PD02	98	SPI	SPI0-SCK
	49	PC00	64	SD-Card	SPI0-NPCS[1]
	93	PD28	124	MAC	MACB_CRS
	95	PD30	126	MAC	MACB_TX_EN
	51	PC02	66	MAC	MACB_MDC
	52	PC03	67	MAC	MACB_MDIO
	84	PD11	107	MAC	MACB_TXD[0]
	86	PD13	109	MAC	MACB_RXD[0]
	85	PD12	108	MAC	MACB_TXD[1]
	87	PD14	110	MAC	MACB_RXD[1]
	91	PD24	120	MAC	POWER_DOWN
	92	PD27	123	MAC	MACB_RX_ER
	94	PD29	125	MAC	MACB_TX_CLK
	80	PD09	105	CAN	CAN0_RX
	81	PD10	106	CAN	CAN0_TX
X1.45				CAN Driver	CAN0_LO

X1.46				CAN Driver	CAN0_HI
X2.07				PHY	LAN_RD-
X2.08				PHY	LAN_RD+
X2.09				PHY	LAN_TD-
X2.10				PHY	LAN_TD+
X2.11				PHY	LAN_LED_LNK
X2.12				PHY	LAN_LED_ACT
X2.13, X2.14				PHY	+3.3V (LAN_CT)
X2.15				PHY	LAN_GND

2.2.3.2 Schaltplan



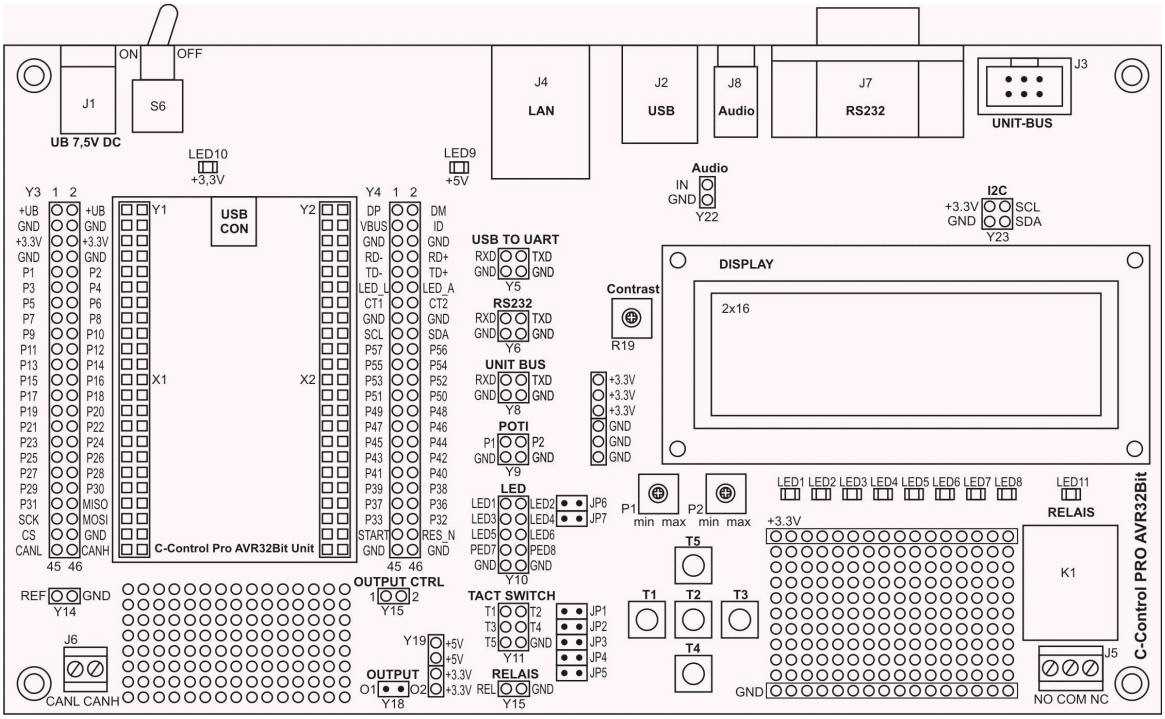


2.2.4 Applicationboard

Das C-Control PRO AVR32Bit Applicationboard (Conrad Best.-Nr.: 192587) ist das Standard Entwicklungsboard zur C-Control PRO AVR32Bit UNIT. Das Applicationboard enthält alle wichtigen Komponenten, die zum Betrieb des C-Control PRO AVR32Bit UNIT benötigt werden. Zudem verfügt das Board über eine sehr gute und große Ausstattung an Peripherie.

Das Board bietet folgende Features:

- 1x Stromversorgung (3.3V & 5V)
- 1x Ein-/ Ausschalter
- 1x LAN-Anschluss (RJ45)
- 1x 2.048V Spannungsreferenz
- 1x CAN-Anschluss
- 1x Dual Power MOS-FET (2x Open Drain)
- 1x Tastaturkreuz (5 Tasten)
- 2x Analogwertgeber (Trimmer)
- 2x16 Zeichen LC-Display (blau/weiss)
- 1x Kontrastregler für LC-Display
- 8x LEDs mit Treiber zur Signalkontrolle
- 1x Lastrelais (24V/ 7A)
- 1x USB zu UART Konverter
- 1x RS232 zu UART Konverter
- 1x Audioverstärker
- 1x UNIT-Bus (3.3V auf 5V)
- 2x Lochrasterfeld für eigene Schaltungen



Applicationboard mit Komponentenbeschriftung

Montage/ Inbetriebnahme

- Die C-Control PRO AVR32Bit UNIT wird so aufgesteckt, dass die Mini-USB Buchse der UNIT in Richtung Ein-/ Ausschalter zeigt (Markierung Applicationboard).
- Im Basiszustand sind die Jumper (JP1 bis JP7) für LED1, LED2 und Tastatur nicht gesteckt.
- Die Stromversorgung des Applicationboards erfolgt über eine Stabilisiertes Stecker- oder Labornetzgerät mit einer Ausgangsspannung von 7.5V und einem Mindeststrom von 500mA.
- Installieren Sie die C-Control PRO Entwicklungsumgebung „IDE“ (engl. integrated development environment). Siehe Installation Software.
- Installieren Sie den USB-Treiber.

Stromversorgung

Das Applicationboard wird über ein stabilisiertes Stecker-/ Labornetzgerät (7,5V/500mA) versorgt. Je nach zusätzlicher Beschaltung des Applicationboards kann es später notwendig sein, ein Netzteil mit höherer Leistung zu verwenden. Zwei Festspannungsregler auf dem Applicationboard erzeugen die interne stabilisierte Versorgungsspannung von 3.3V und 5V. Die beiden LED´s LED9 und LED10 signalisieren die Funktionalität der Stromversorgung. Alle Schaltungsteile auf dem Applicationboard werden mit diesen Spannungen versorgt (siehe Schaltplan). Auf dem Board stehen einige Anschlüsse zur Verfügung, an denen Sie die Spannungen abgreifen können. Achten Sie darauf, dass die beiden Spannungsregler bei größerer Belastung durch eigene Beschaltungen nicht zu heiß werden. Bei größeren Verbrauchern ist es empfehlenswert, diese extern zu versorgen!

➔ Die Masse zwischen externer Schaltung (Stromversorgung) und dem Applicationboard muss gleich sein!

➔ Die Kühlfläche der Spannungsregler wird bei Betrieb warm bis heiß, je nach angeschlossenem Verbraucher!

Ein-/ Ausschalter

Der Schalter S6 befindet sich an der Rückseite neben der Stromversorgungsbuchse des Applicationboards, und dient zum Ein-/ Ausschalten der Hauptstromversorgung.

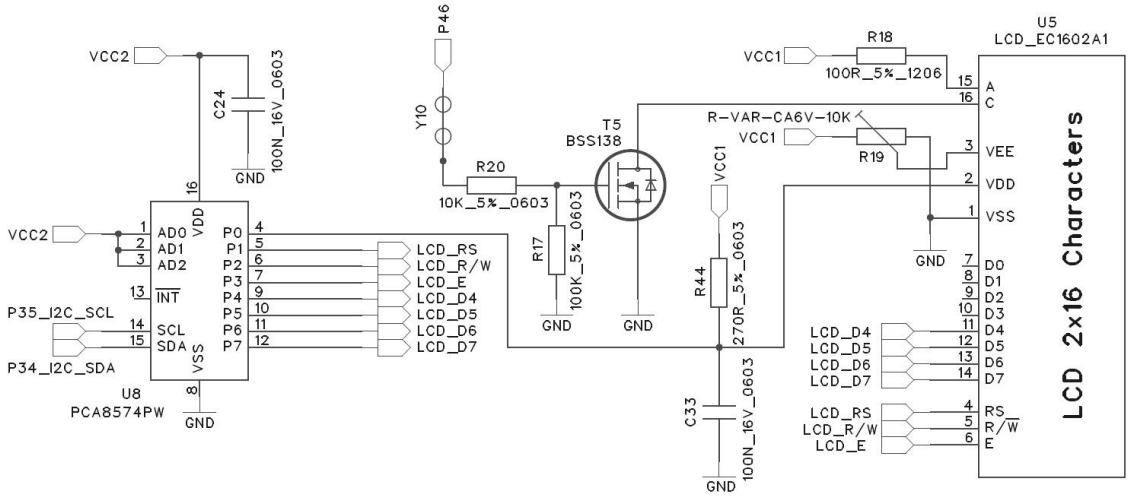
Jumper

Die Jumper JP1 bis JP5 verbinden die Taster der Tastatur mit den Pins der UNIT (T1 = P41, T2 = P42, T3 = P43, T4 = P44, T5 = P45). Die Jumper JP6 und JP7 verbinden die LEDs LED1 und LED2 mit den Pins der UNIT (LED1 = P48, LED2 = P49). Siehe auch in der Pinzuordnungstabelle unter Application Board Function.

LC-Display

Das LC-Display dient zur Darstellung von Variablen und Zeichen. Es wird über den I2C-Bus angesteuert. Auf dem Applicationboard ist hierzu ein Portexpander (PCA8574) vorhanden, der die Kommunikation zwischen UNIT und LC-Display über den I2C-Bus übernimmt. Das Display wird im 4-Bit Modus betrieben. Über den P46 wird die Hintergrundbeleuchtung ein-/ ausgeschaltet. Das Betriebssystem bietet eine einfache Softwareschnittstelle für Ausgaben auf das Display. Die kleine Schaltung zur Ansteuerung des LCDs kann für eigene Schaltungen einfach übernommen werden. Es werden die meisten „Standard Dot Matrix“ LCDs unterstützt. (siehe Schaltplan und LC-Display Daten-

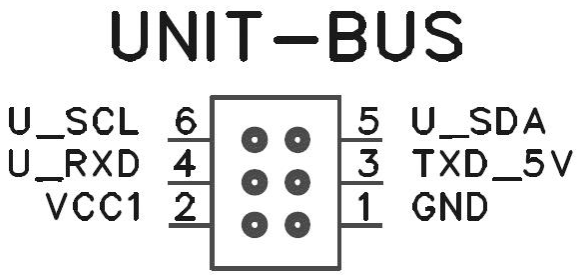
blatt).



Schaltplanausschnitt zum I2C-LCD

LC-Display Kontrastregler

Die beste Sichtbarkeit der LCD-Zeichen ergibt sich bei frontaler Betrachtung. Gegebenenfalls muss der Kontrast nachgeregelt werden. Der Kontrast kann über den Trimmer R19 („Contrast“ links neben dem LC-Display) eingestellt werden.

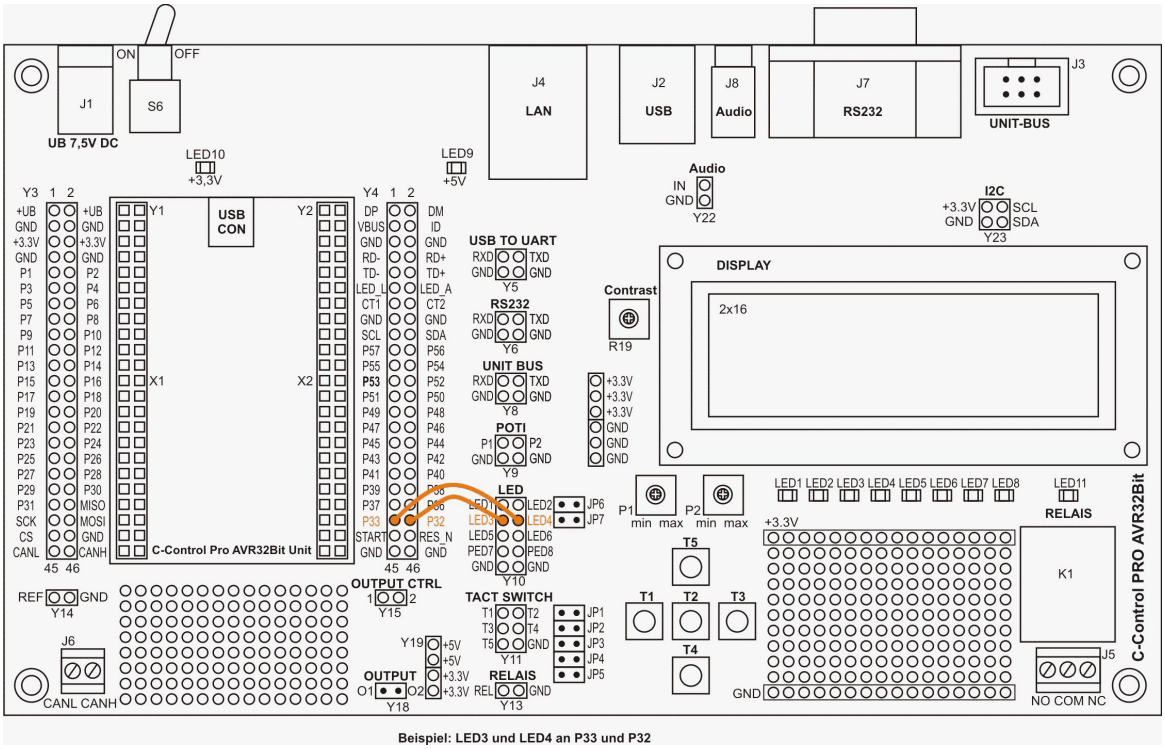


UNIT-BUS Belegung

I2C und UNIT-Bus

Die Pins der Buchsenleiste Y23 (4er Block) sind mit den Pins P34 (SDA) und P35 (SCL) fest verbunden. An den Pins der Buchsenleiste Y8 (4er Block) kann eine freie UART-Schnittstelle auf den UNIT-Bus gelegt werden. Der UNIT-Bus setzt die 3.3V der UNIT nach 5V um und die 5V Signale des UNIT-Bus auf 3.3V (Bidirektionaler Pegel-Shifter). An diesen BUS können z.B. die I2C-Module der C-

Control I und andere 5V I2C-Bus bzw. UART Bausteine angeschlossen werden.



Beispiel: LED3 und LED4 an P33 und P32

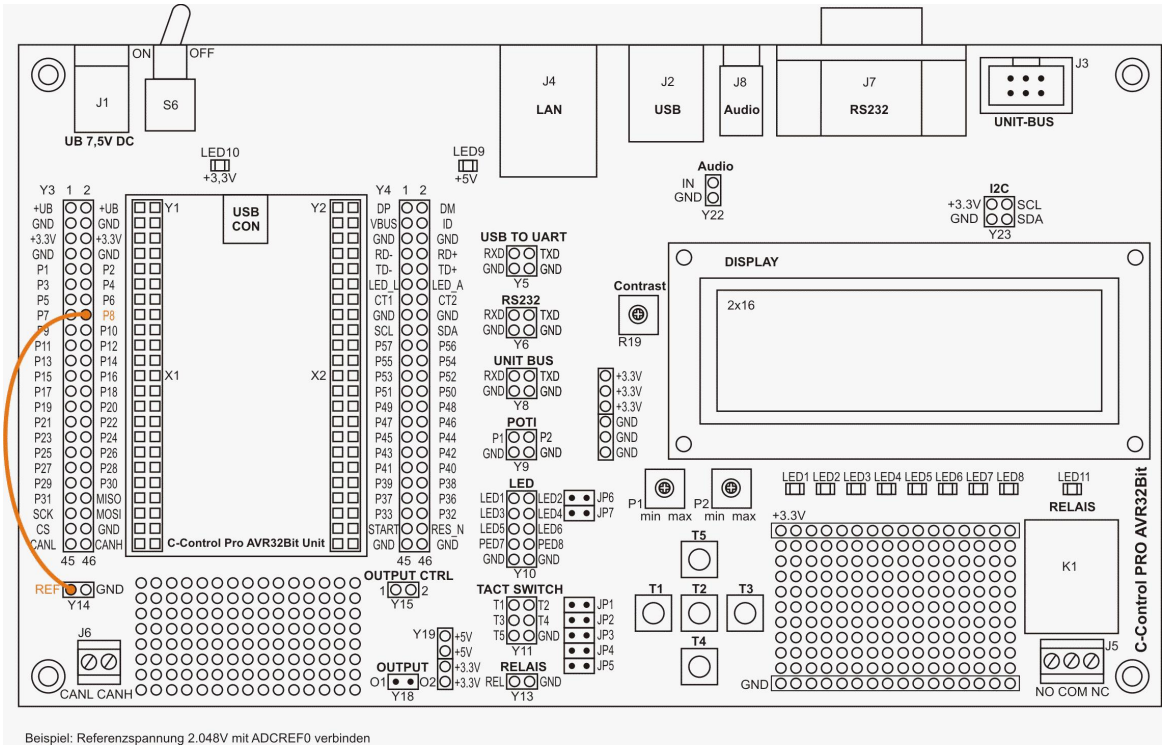
Beispiel LED3 und LED4 an P32 und P33

LEDs

Die Pins der Buchsenleisten Y10 (10er Block) sind fest mit den LEDs LED1 bis LED8 verbunden. Die LEDs werden hochohmig über FETs angesteuert (ca. 100K). Damit kann der Port auch noch für andere Zwecke benutzt werden, und die LEDs zeigen zusätzlich den Portzustand an. Die Jumper JP6 und JP7 verbinden die ersten beiden LEDs **LED1**, **LED2** mit den Pins der UNIT P48 und P49.

Referenzspannung

Die Pins der Buchsenleiste Y14 stellen eine stabile Referenzspannung für den ADC (Analog Digital Konverter) bereit. Diese kann mit den ADCREFx Eingängen der UNIT verbunden werden. Hiermit können Sie für den ADC eine stabile ext. Referenzspannung bereitstellen.



Beispiel Referenzspannung am ADCREF0

CAN-Bus

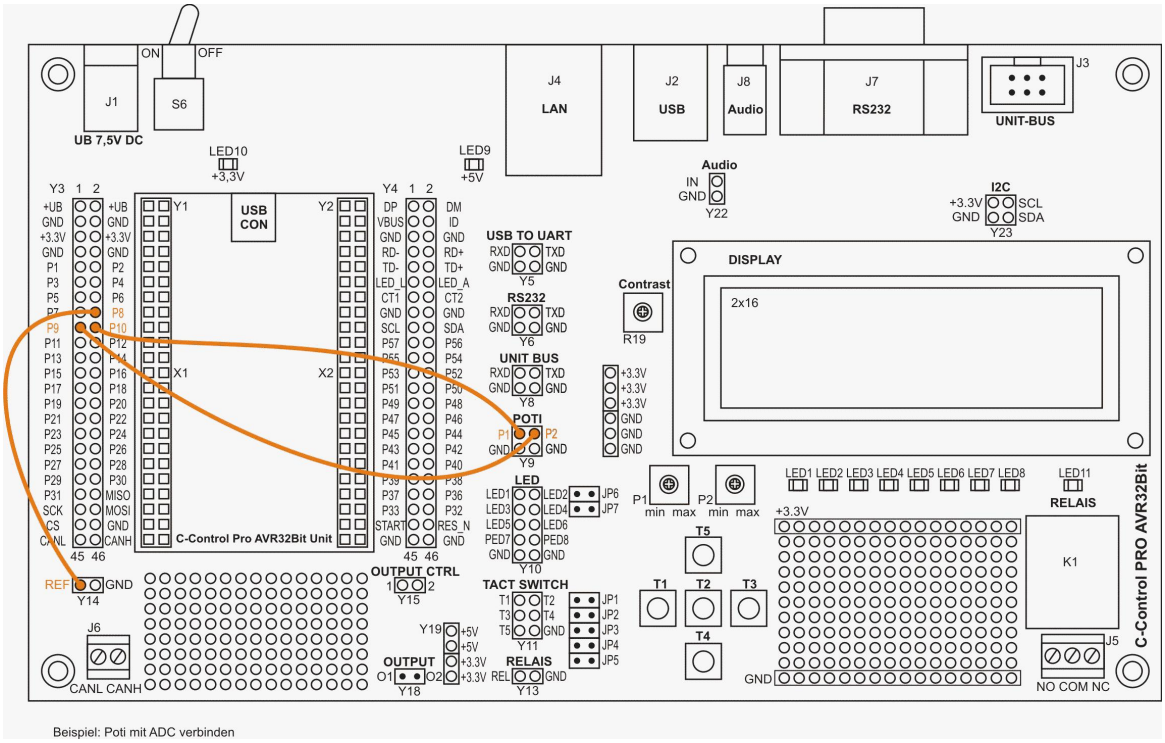
An der Klemme J6 ist der CAN-Bus (CAN0) der UNIT herausgeführt und kann direkt verwendet werden. Es muss kein zusätzlicher Treiber nachgeschaltet werden, da dieser bereits auf der UNIT vorhanden ist.

Audioverstärker

An den Pins der Buchsenleiste Y22 kann direkt von der UNIT ein PWM Signal zur Tonausgabe angeschlossen werden. An der Klinkenbuchse kann ein Kopfhörer, kleiner Lautsprecher (min. 8 Ohm) bzw. ein Aktivlautsprecher angeschlossen werden. Achtung die Ausgabe kann je nach Signal sehr laut sein und bei unsachgemäßer Verwendung zur Gehörschäden führen!

Analogwertgeber

Die Pins der Buchsenleiste Y9 (4er Block) sind mit den Trimmern P1 und P2 verbunden. Die Trimmer sind als variabler Spannungsteiler geschaltet und werden aus der 2.048V Referenzspannung versorgt. Somit lässt sich eine Ausgangsspannung zwischen 0 und 2.048V einstellen. Die Ausgänge an der Buchsenleiste können direkt mit den Analogeingängen der UNIT verbunden werden.



Beispiel Trimmer am ADC0 und ADC1 mit externer Referenzspannung 2.048V

Tastatur

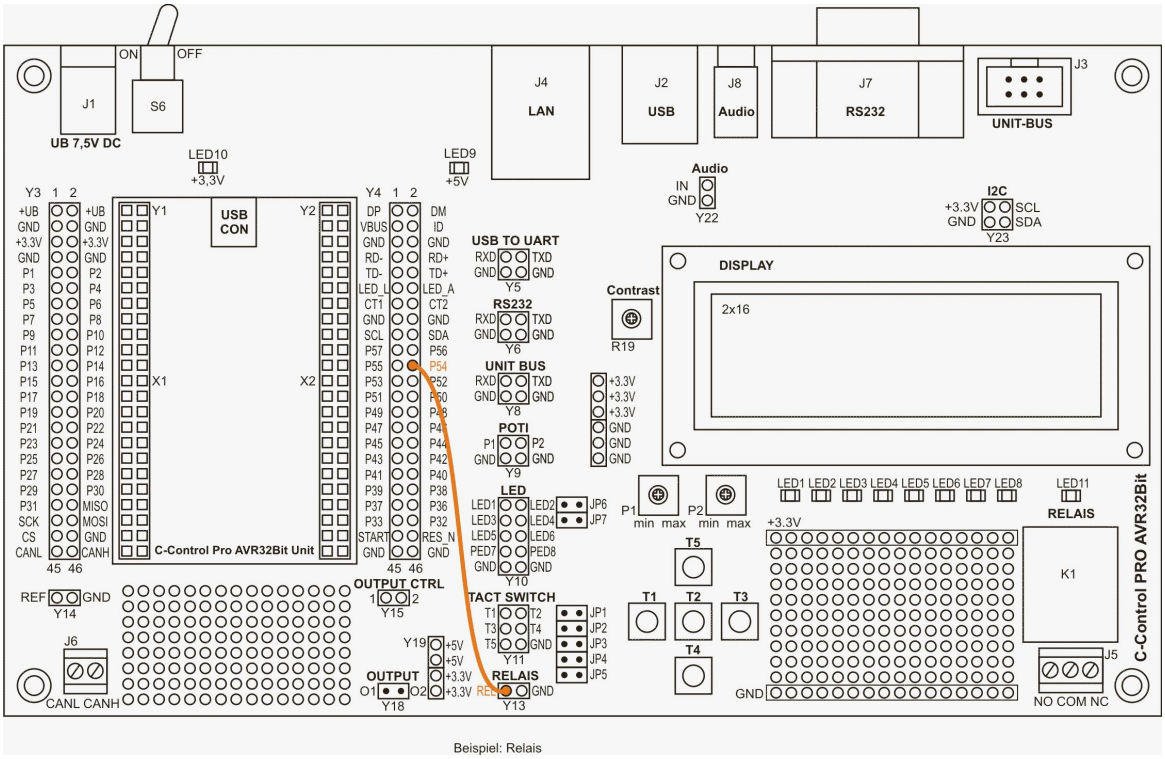
Für Benutzereingaben steht eine 5-Tasten Tastatur (Tastaturkreuz) zur Verfügung. Die Pins der Buchsenleiste Y11 (6er Block) sind mit den Tastern T1 bis T5 verbunden. Über die Jumper JP1 bis JP5 können diese direkt mit den Pins der UNIT (P41 bis PT45) verbunden werden. Ist eine andere Belegung gewünscht, so müssen die Jumper abgezogen werden und die Taster über Drahtbrücken (Jumpwire) von der Buchsenleiste Y11 mit der UNIT verbunden werden. Die Taster sind auf dem Applicationboard mit 47 kOhm Pull-Up Widerständen verbunden. Es müssen in der Software keine Pull-Up/Down Widerstände aktiviert werden. Liest man einen Schalter im Ruhezustand (nicht gedrückt), wird am Port eine "1" erkannt, da über den Pull-Up Widerstand die 3,3V an den Pin geführt werden.

LAN-Buchse/ Ethernet

Die LAN-Buchse kann direkt mit einem FTP-Kabel an einen Switch oder Router angeschlossen werden. Mit dem Ethernet-Interface der C-Control PRO AVR32Bit kann auf einfache Weise ein Webserver realisiert werden (siehe Beispiele). Des Weiteren kann über den Ethernet-Bootloader die UNIT über das Netzwerk programmiert werden. Die LAN-Buchse ist fest mit den Pins der UNIT verbunden (siehe Pinzuordnung im Handbuch).

Relais

Die Pins an der Buchsenleiste Y13 sind mit dem Relais K1 verbunden. Das Relais wird über einen FET-Treiber geschaltet, und der „REL“ Anschluss von Y13 kann direkt mit einem Port der UNIT verbunden werden. Das Relais dient zum Schalten von kleineren Verbrauchern.

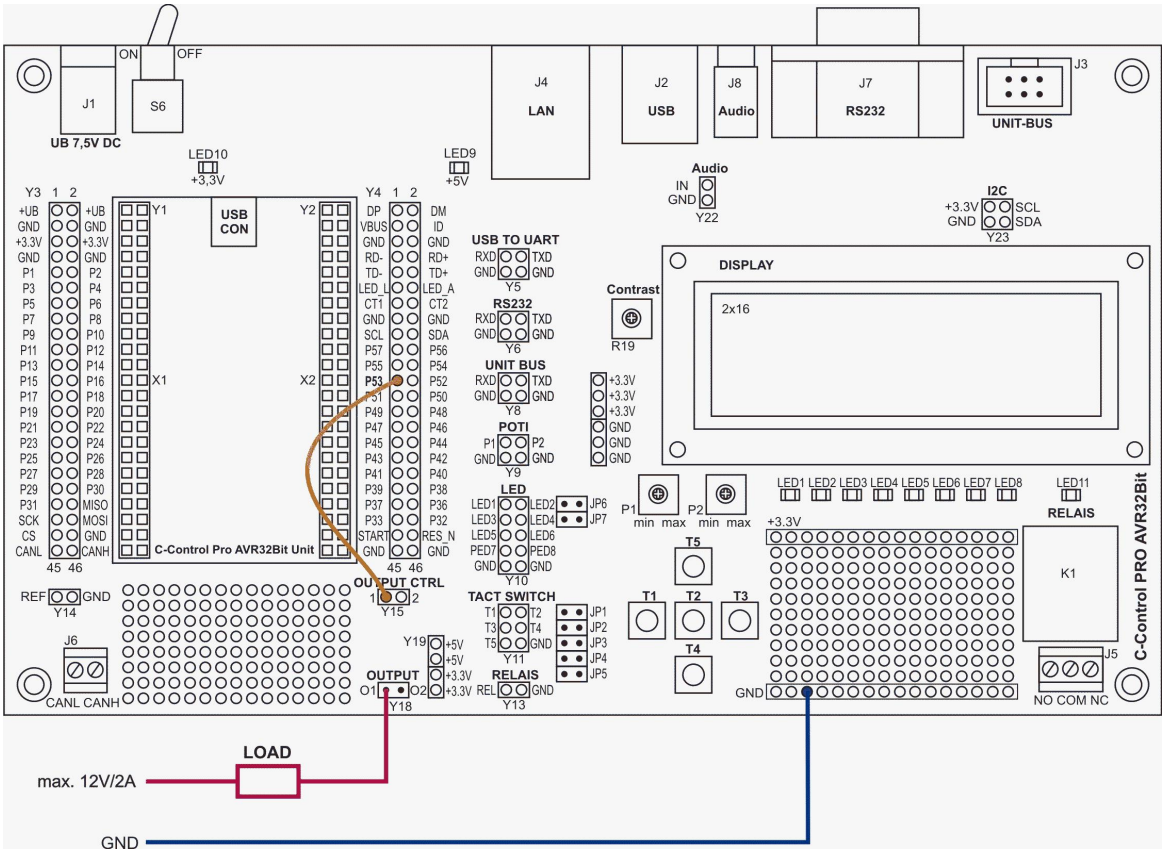


Beispiel Relais an P54

FET-Treiber

Die Pins der Buchsenleiste Y15 sind mit dem Open-Drain FET-Treiber verbunden. Hiermit können Ohmsche Verbraucher (max. 12VDC / 2A) direkt angesteuert werden. Die OUTPUT CTRL-Pins können direkt mit einem Port der UNIT verbunden werden. Es können auch PWM Signale zur Ansteuerung verwendet werden. An der Stiftleiste Y18 stehen die Open-Drain Anschlüsse bereit. Diese werden mit dem Verbraucher – Stromversorgung (+) verbunden.

➔ **Beim Schalten von induktiven Verbrauchern muss eine externe Freilaufdiode zu den Open-Drain Ausgängen angebracht werden. Die Diode ist im Idealfall so nahe wie möglich am Verbraucher anzubringen.**



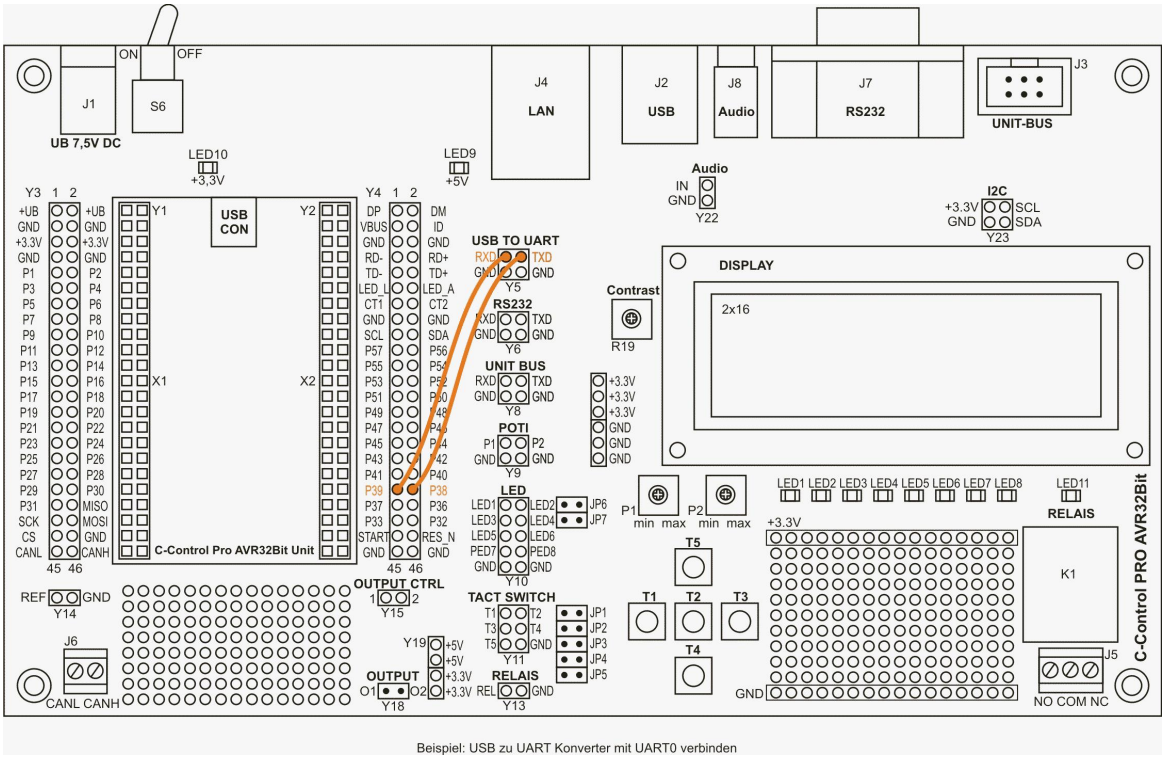
Beispiel FET-Treiber mit Last gesteuert über P53

USB zu UART Konverter

Die Pins der Buchsenleiste Y5 (4er Block) sind mit dem UART zu USB Konverter (Silabs CP2104) verbunden. An der USB-Buchse (Type B) wird das Board mit dem PC verbunden. Der Konverter dient zur seriellen Ausgabe von Daten der UNIT an den PC.

➔ **Installieren Sie zuerst die Treiber bevor Sie die Verbindung herstellen.**

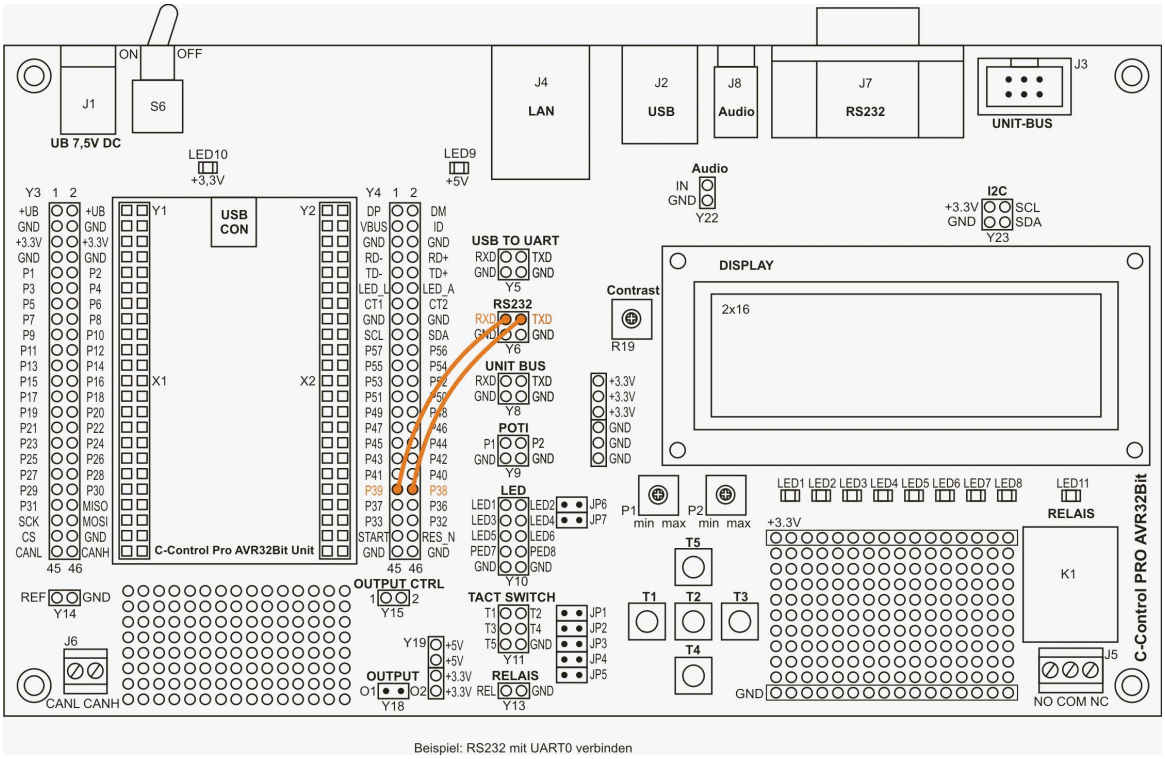
Die Treiber zum Konverterbaustein finden Sie unter:
http://www.silabs.com/PROducts/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx
http://www.silabs.com



UART Konverter an UART0

RS232 zu UART Konverter

Die Pins der Buchsenleiste Y6 (4er Block) sind mit dem RS232 Konverter (MAX3232) verbunden. An der 9 pol. SUB-D Buchse wird das Board mit dem PC oder einem RS232 Gerät verbunden. Der Konverter dient zur Pegelwandlung der 3.3V UNIT auf den Normpegel der seriellen RS232 Schnittstelle (+/-12V). Über diese Schnittstelle können Daten zwischen einem PC oder einem Gerät mit RS232 (z.B. Messgerät) ausgetauscht werden.



Beispiel: RS232 mit UART0 verbinden

RS232 an UART0

Dieses Produkt erfüllt die gesetzlichen, nationalen und europäischen Anforderungen. Der „I2C-Bus“ ist ein eingetragenes Markenzeichen von Philips Semiconductors. Alle anderen enthaltenen Firmennamen und Produktbezeichnungen sind Warenzeichen der jeweiligen Inhaber. Alle Rechte vorbehalten.

➔ **Die Kühlfläche bei den Spannungsreglern (zwischen Ein-/ Ausschalter und LAN-Anschluss) wird beim Betrieb heiß!**

Technische Daten

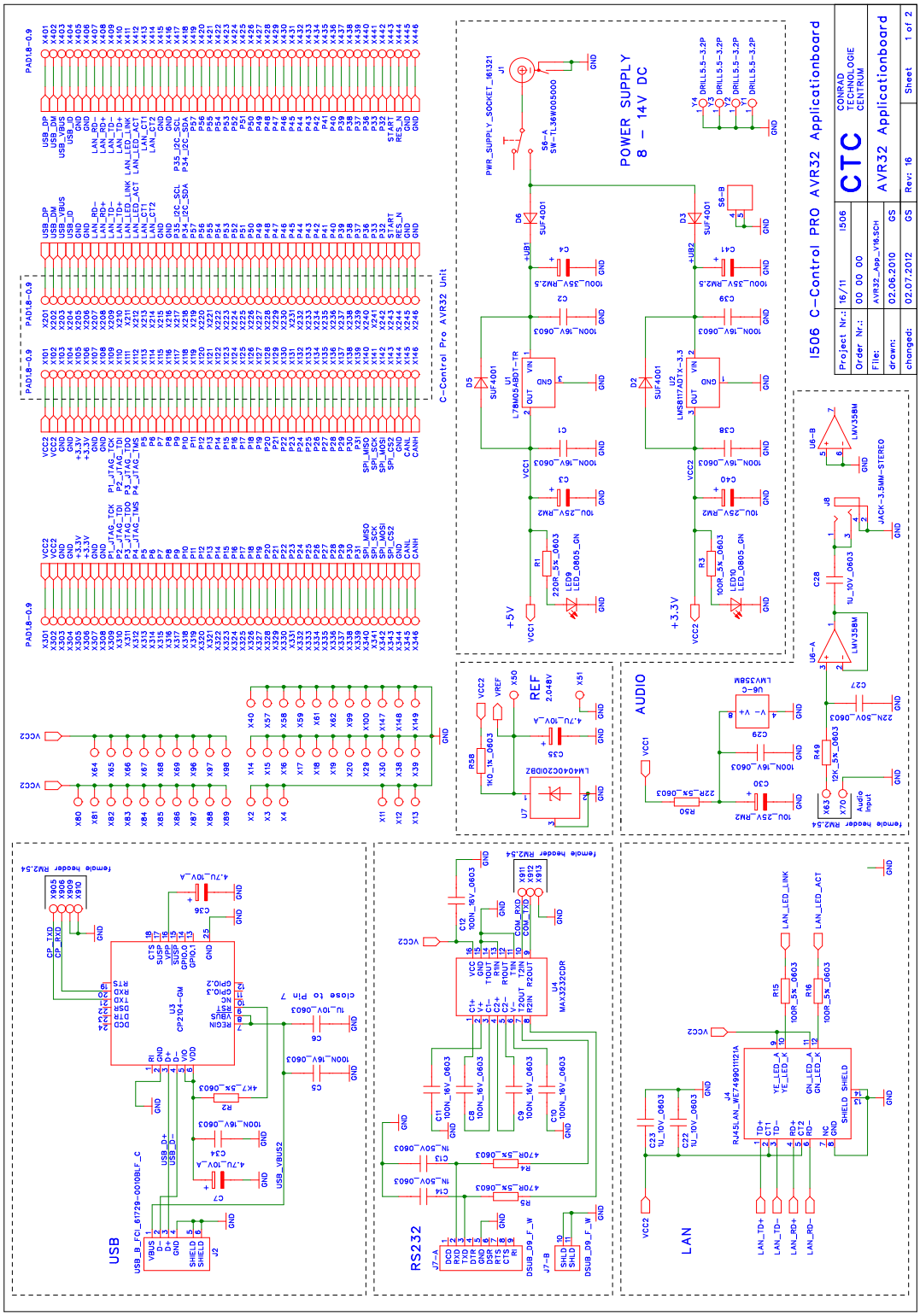
Stromversorgung extern	7,5VDC / 500mA (stabilisiert)
Stromversorgung intern	3.3V und 5V
Umgebungstemperatur	0 bis 60°C
Abmessungen	190x110mm
Gewicht ohne UNIT	ca. 160g

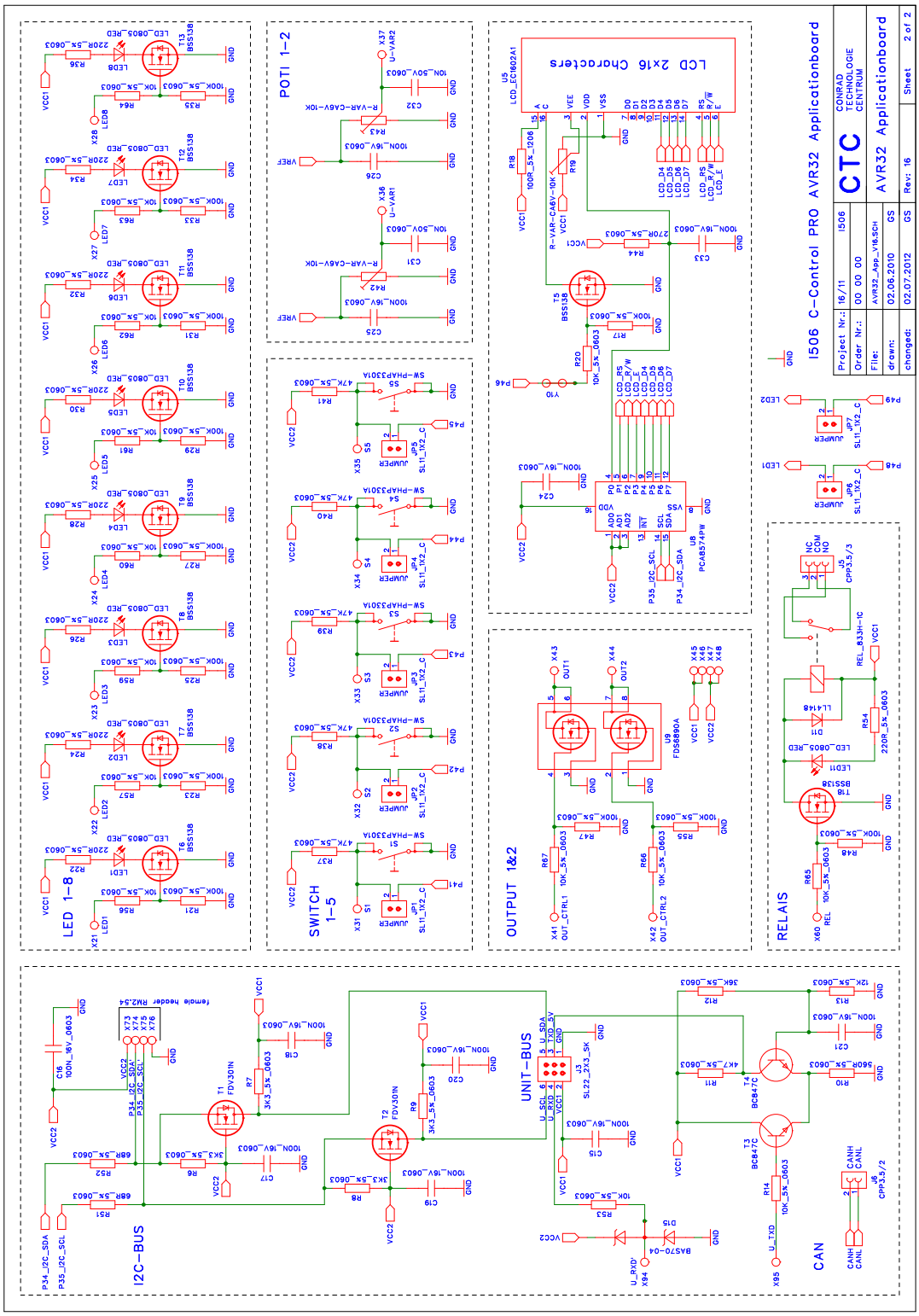
Lieferumfang

- 1x C-Control PRO AVR32Bit Applicationboard
- 1x Mini-USB Kabel
- 7x Jumper
- 1m Drahtwickel für Steckbrücken

- Kurzanleitung

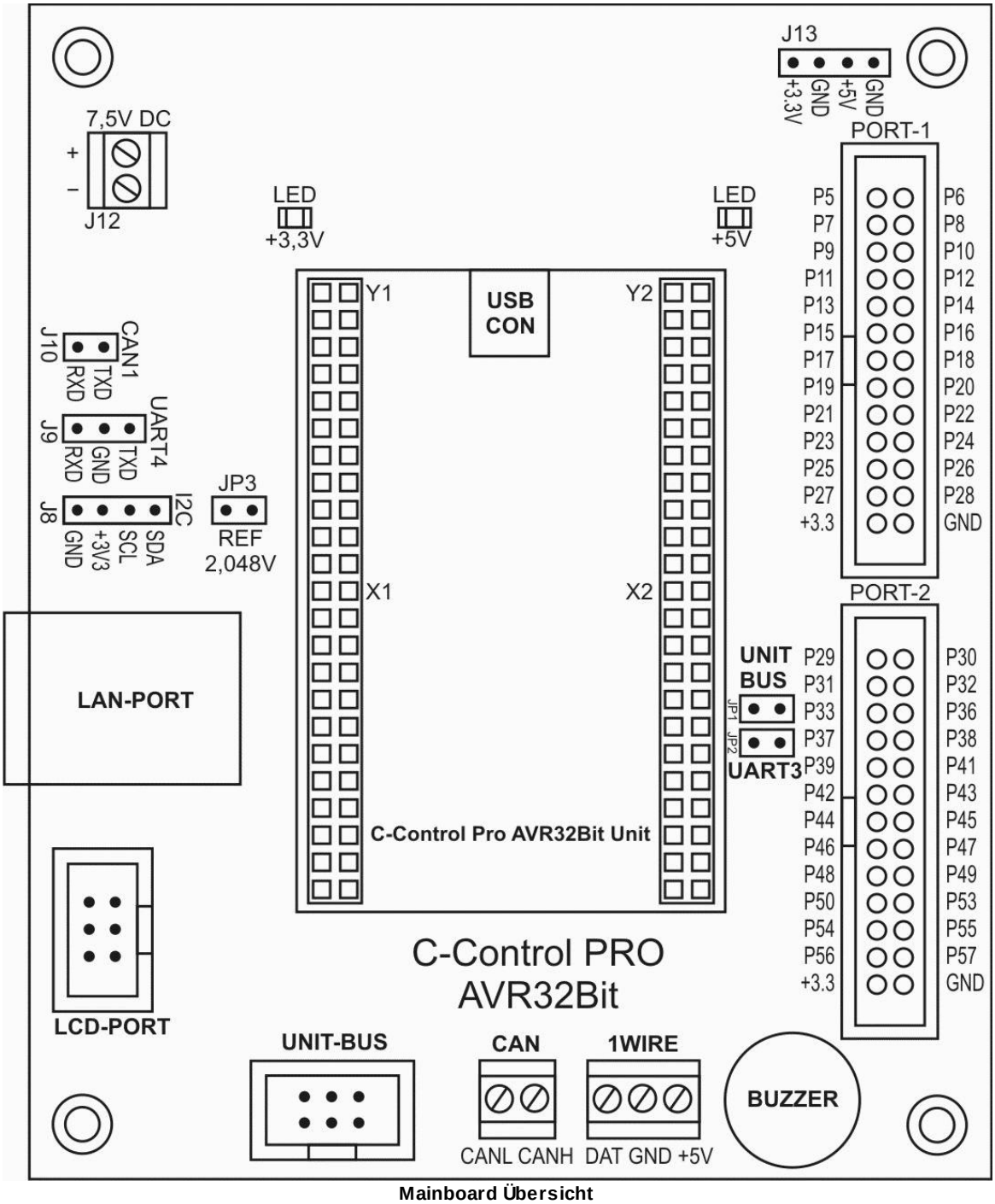
2.2.4.1 Schaltplan





2.2.5 Mainboard

Das C-Control PRO AVR32Bit Mainboard (Conrad Best.-Nr.: 192702) ist das kompakte Experimentier- und Entwicklungsboard zur C-Control PRO AVR32Bit „UNIT“. Das C-Control PRO AVR32Bit Mainboard enthält alle wichtigen Komponenten die zum Betrieb des C-Control PRO AVR32Bit UNIT benötigt werden. Zudem verfügt das Board über eine gute Grundausstattung an Peripherie.



Mainboard Übersicht

Das Board bietet folgende Features:

- 1x Stromversorgung (3.3V & 5V)
- 1x LAN-Anschluss (RJ45)
- 1x 2.048V externe Spannungsreferenz
- 1x Signalgeber (Buzzer)
- 2x CAN-Anschluss
- 1x LCD-PORT zum Anschluss des I2C-LCD´s (Conrad BestNr.: 192602)
- 1x I2C-BUS Anschluss
- 1x 1-Wire Anschluss
- 2x I/O-PORT mit je 26 Pins
- 1x UNIT-Bus (3.3V auf 5V) für diverse Sensoren und Aktuatoren

Montage/ Inbetriebnahme

- Die C-Control PRO AVR32Bit UNIT wird so aufgesteckt, dass die Mini-USB Buchse der UNIT mit der Markierung auf dem Mainboard übereinstimmt (USB CON).
- Im Basiszustand sind die Jumper (JP1, JP2 und JP3) nicht gesteckt.
- Die Stromversorgung des Mainboards erfolgt über ein stabilisiertes Stecker- oder Labornetzgerät mit einer Ausgangsspannung von 7.5V und einem Mindeststrom von 500mA.
- Installieren Sie die C-Control PRO Entwicklungsumgebung „IDE“ (engl. integrated development environment). Siehe Installation Software.
- Installieren Sie den USB-Treiber.

Stromversorgung

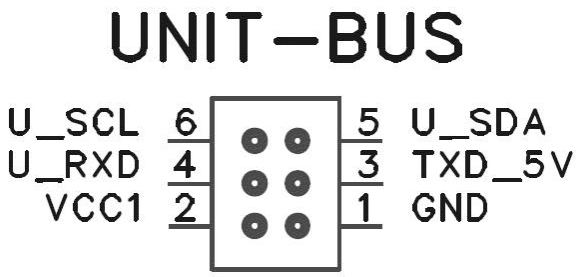
Das Mainboard wird über ein stabilisiertes Stecker Netzgerät (7,5V/500mA) versorgt. Je nach zusätzlicher Beschaltung des Mainboards kann es später notwendig sein, ein Netzteil mit höherer Leistung zu verwenden. Zwei Festspannungsregler auf dem Mainboard erzeugen die interne stabilisierte Versorgungsspannung von 3.3V und 5V. Die beiden LEDs +3.3V und +5V signalisieren die Funktionalität der Stromversorgung. Alle Schaltungsteile auf dem Mainboard werden mit diesen Spannungen versorgt (siehe Schaltplan). Auf dem Board stehen einige Buchsen zur Verfügung an denen Sie die Spannungen abgreifen können. Achten Sie darauf, dass die beiden Spannungsregler bei größerer Belastung nicht zu heiß werden. Bei größeren Verbrauchern ist es empfehlenswert, diese Verbraucher extern zu versorgen!

➔ **Achtung**: Die Masse zwischen externer Schaltung (Stromversorgung) und dem Mainboard muss gleich sein!

➔ **Achtung**: Die Kühlfläche der Spannungsregler wird bei Betrieb warm bis heiß, je nach angeschlossene Verbraucher!

I2C, UART und UNIT-Bus

Die Pins von J8 sind mit den Pins P34 (I2C-SDA) und P35 (I2C-SCL) fest verbunden. An den Pins von J9 ist die UART4-Schnittstelle herausgeführt. Der I2C-BUS ist zudem fest mit dem UNIT-BUS verbunden. Der UNIT-Bus setzte die 3.3V Pegel der UNIT auf 5V um und die 5V Signale des UNIT-BUS (Sensoren und Module) auf 3.3V (Bidirektionaler Pegel-Shifter). An diesen BUS können z.B. die I2C-Module der C-Control I oder diverse andere 5V I2C-Bus bzw. UART Bausteine angeschlossen werden. Der UART3-Schnittstelle kann über die Jumper JP1 und JP2 auf den UNIT-BUS gelegt werden.



UNIT-BUS Belegung

Referenzspannung

Der Jumper JP3 legt die externe 2.048V Referenzspannung auf den ADCREF0 (P8) Pin der UNIT.

CAN-Bus

An der Schraubklemme mit der Bezeichnung „CAN“ ist der CAN-Bus (CAN0) der UNIT herausgeführt und kann direkt verwendet werden. Es muss kein zusätzlicher Treiber nachgeschaltet werden, da dieser bereits auf der UNIT vorhanden ist. Auf der Buchsenleiste J10 ist der CAN-BUS (CAN1) herausgeführt. Dieser verfügt über keinen CAN-Treiber und kann auch als normaler Ein-/ Ausgang (I/O) verwendet werden.

LAN-Buchse

Die LAN-Buchse kann direkt mit einen Switch oder Router verbunden werden. Die LAN-Buchse ist fest mit den Pins der UNIT verbunden (siehe Pinzuordnung der AVR32Bit UNIT). Mit dem Ethernet-Interface der C-Control PRO AVR32Bit kann auf einfache Weise ein Webserver realisiert werden (siehe Demo Beispiele). Des Weiteren kann über den Ethernet-Bootloader (siehe Handbuch unter AVR32Bit UNIT Bootloader) die UNIT aus der Ferne programmiert werden.

LCD-PORT

An den 6poligen Pfosten-Steckverbinder mit der Bezeichnung „LCD-PORT“ kann das C-Control PRO AVR32Bit LCD1602 Board mit der Conrad BestNr.: 192602 angeschlossen werden. Verbunden wird das Mainboard mit dem LCD-Modul mittels einen 6poligen Flachbandkabel mit Pfosten-Steckverbinder (female). Da je nach Anwendung die Kabellänge variieren kann, bieten wir Ihnen diese Komponenten zur Selbstmontage unter folgenden Bestellnummern an.

- Flachbandkabel RM1.27 0.05mm²: Best.-Nr. 607237

- Pfostensteckverbinder 2x3 RM:2.54mm: Best.-Nr. 742063
- Passendes Anschlusskabel bereits konfektioniert (Länge: 35cm) Best.-Nr. 19 88 76.

1-WIRE

An der Schraubklemme mit der Bezeichnung „1WIRE“ ist der Digitalpin P3 der UNIT herausgeführt. An diesen kann ein 1-WIRE Sensor wie z. B. Conrad Best.-Nr.: 198284 angeschlossen werden. Dieser Pin kann auch als normaler digitaler Ein-/ Ausgang (I/O) verwendet werden.

PORT-1, PORT-2

An den beiden 26poligen Pfosten-Steckverbindern mit der Bezeichnung „PORT-1“ und „PORT-2“ sind die freien Pins der AVR32Bit UNIT herausgeführt. Hier kann das Experimentierboard (Conrad-Best.-Nr.: 192615) über zwei 26-polige Flachbandkabel mit Pfosten-Steckverbinder angeschlossen werden.

- Flachbandkabel 26-polig RM1.27 0.05mm²: Best.-Nr. 607222
- Pfostensteckverbinder 2x13 RM:2.54mm: Best.-Nr. 742185

Herausgeführte Pin´s am Pfostensteckverbinder PORT-1:
P5, P6, P7, P8, P9 , P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, +3.3V, GND

Herausgeführte Pin´s am Pfostensteckverbinder PORT-2:
P29, P30, P31, P32, P33, P36, P37, P38, P39, P41, P42, P43, P44, P45, P46, P47, P48, P49, P50, P53, P54, P55, P56, P57, +3.3V, GND

Dieses Produkt erfüllt die gesetzlichen, nationalen und europäischen Anforderungen. „I2C-Bus“ ist ein eingetragenes Markenzeichen von Philips Semiconductors. Alle anderen enthaltenen Firmennamen und Produktbezeichnungen sind Warenzeichen der jeweiligen Inhaber. Alle Rechte vorbehalten.

➔ **Achtung**: Die Kühlfläche bei den Spannungsreglern (nahe der Mini-USB Buchse der UNIT) wird beim Betrieb heiß!

Technische Daten

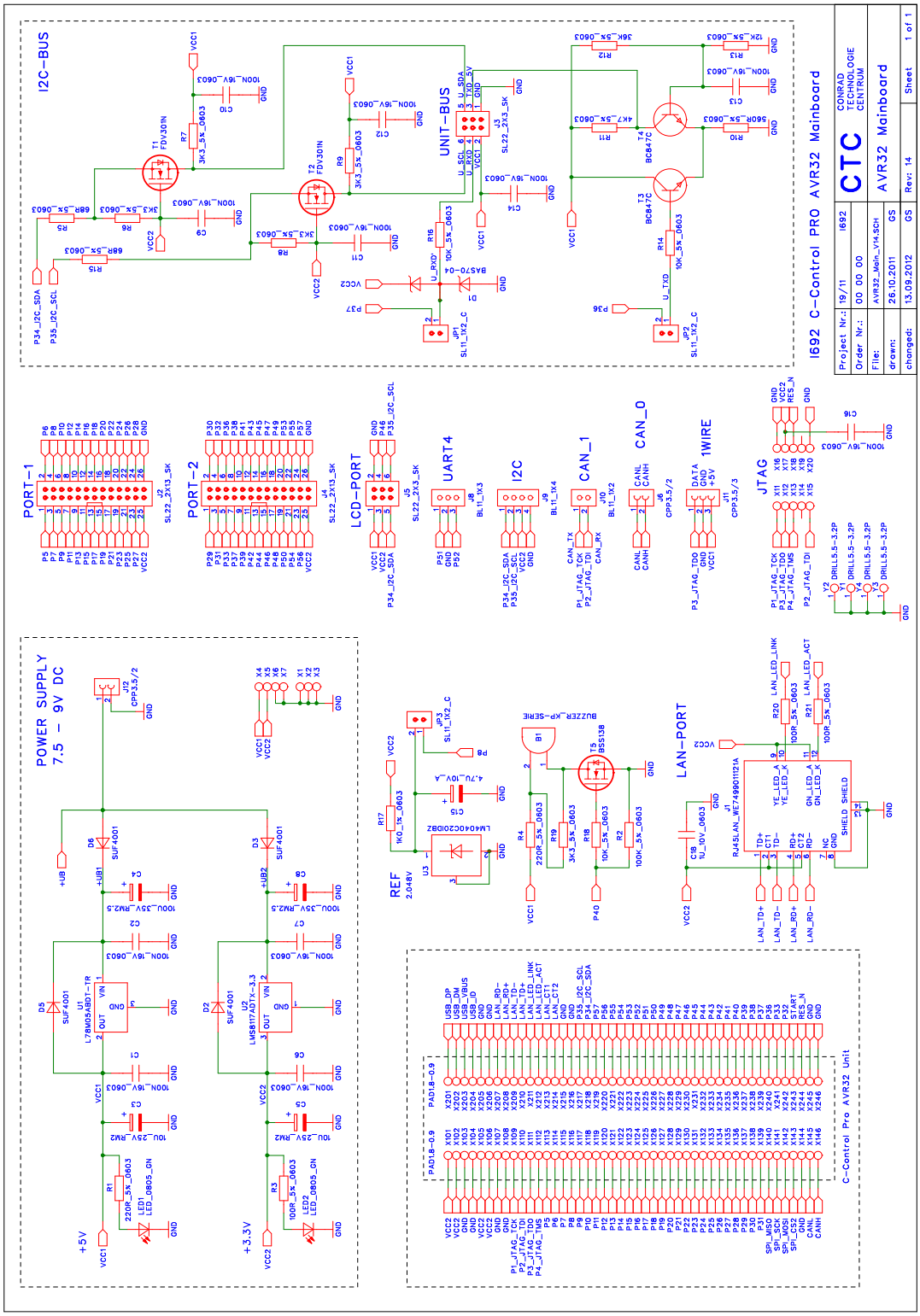
Stromversorgung extern	7,5VDC / 500mA (stabilisiert)
Stromversorgung intern	3.3V und 5V
Umgebungstemperatur	0 bis 60°C
Abmessungen	110x95mm
Gewicht ohne UNIT	ca. 65g

Lieferumfang

- 1x C-Control PRO AVR32Bit Mainboard
- 1x Mini-USB Kabel

- 3x Jumper
- Kurzanleitung

2.2.5.1 Schaltplan



2.2.6 UNIT-BUS Exp. Board

Das C-Control PRO AVR32Bit UNIT-BUS Exp. Board (Conrad Best.-Nr.: 192659) ist zur Erweiterung des Funktionsumfanges der C-Control PRO AVR32Bit Produkte bestimmt. Das Produkt ist als offene Leiterplatte mit sechs einzelne 6poligen Pfosten-Steckverbinder ausgeführt, und nur für den UNIT-BUS der C-Control PRO AVR32 und der C-Control I Produktfamilie (Erweiterungen) und deren Anschlussbuchsen bestimmt. Die Ansteuerung der einzelnen Zusatzmodule erfolgt über Software. Die Software Beispiele finden Sie im Ordner der Beispielprogramme (siehe Demo Programme) und unter www.c-control.de.

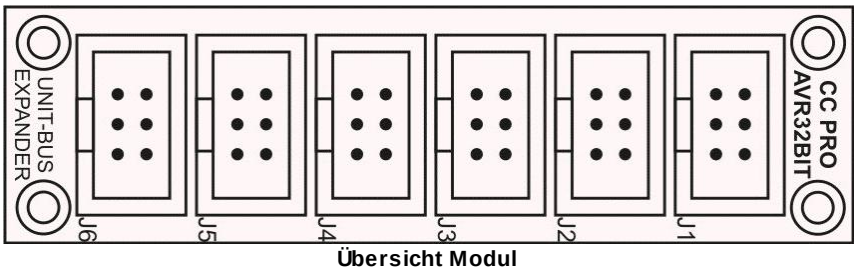
Anschluss und Inbetriebnahme

Vergewissern Sie sich, dass vor dem Anschluss der Module an Ihr C-Control PRO AVR32Bit Basisprodukt (z. B. AVR32Bit Applicationboard Conrad Best.-Nr.: 192587, AVR32Bit Mainboard Conrad Best.-Nr.: 192702) alle Verbindungen zu angeschlossenen Geräten getrennt und spannungsfrei sind. Auf den C-Control PRO AVR32Bit Basisprodukten befindet sich ebenfalls ein 6-poliger Pfosten-Steckverbinder mit der Bezeichnung UNIT-BUS. Dieser Pfosten-Steckverbinder ist für den Anschluss der UNIT-BUS Erweiterungsmodule geeignet. Jede dieser Buchsen führt die Leitungen SDA, SCL, RxD, TxD, +5V und GND. Die C-Control PRO AVR32Bit UNIT arbeitet mit 3.3V Pegel, und die UNIT-BUS Erweiterungen, wie auch die älteren C-Control I I2C-Bus Module, benutzen 5V. Deshalb ist zwischen der C-Control PRO AVR32Bit UNIT und dem UNIT-BUS ein Pegelwandler vorhanden, der die 3.3V Signale der UNIT auf 5V Signale des UNIT-BUS umsetzt. Der UNIT-BUS passiv Verteiler dient zur Verteilung der I2C-Bus Signale SDA und SCL sowie der UART Signale RxD und TxD. Hinzu kommen die +5V Versorgung und GND. Der Verteiler kann mit seinen äußeren Befestigungslöchern (Bohrungsdurchmesser: 2.5mm) in Ihrer Applikation montiert werden.

Bei der Verwendung von C-Control I Erweiterungsmodulen nehmen Sie bitte die Anleitung der C-Control I Erweiterungsmodule. Hier finden Sie weitere Technische Informationen zu den einzelnen Produkten. Wenn nicht anders angegeben, werden alle Erweiterungsmodule über ihre Steckverbinder vom jeweiligem Basisgerät mit der erforderlichen Betriebsspannung versorgt. Da je nach Anwendung die Kabellänge variieren kann, bieten wir Ihnen diese Komponenten zur Selbstmontage unter folgenden Bestellnummern an:

Flachbandkabel RM1.27 0.05mm²: Best.-Nr. 607237
Pfostensteckverbinder 2x3 RM:2.54mm: Best.-Nr. 742063

Ein bereits fertig konfektioniertes Anschlusskabel (Länge: 35cm) erhalten Sie mit der Bestellnummer Best.-Nr. 198876.

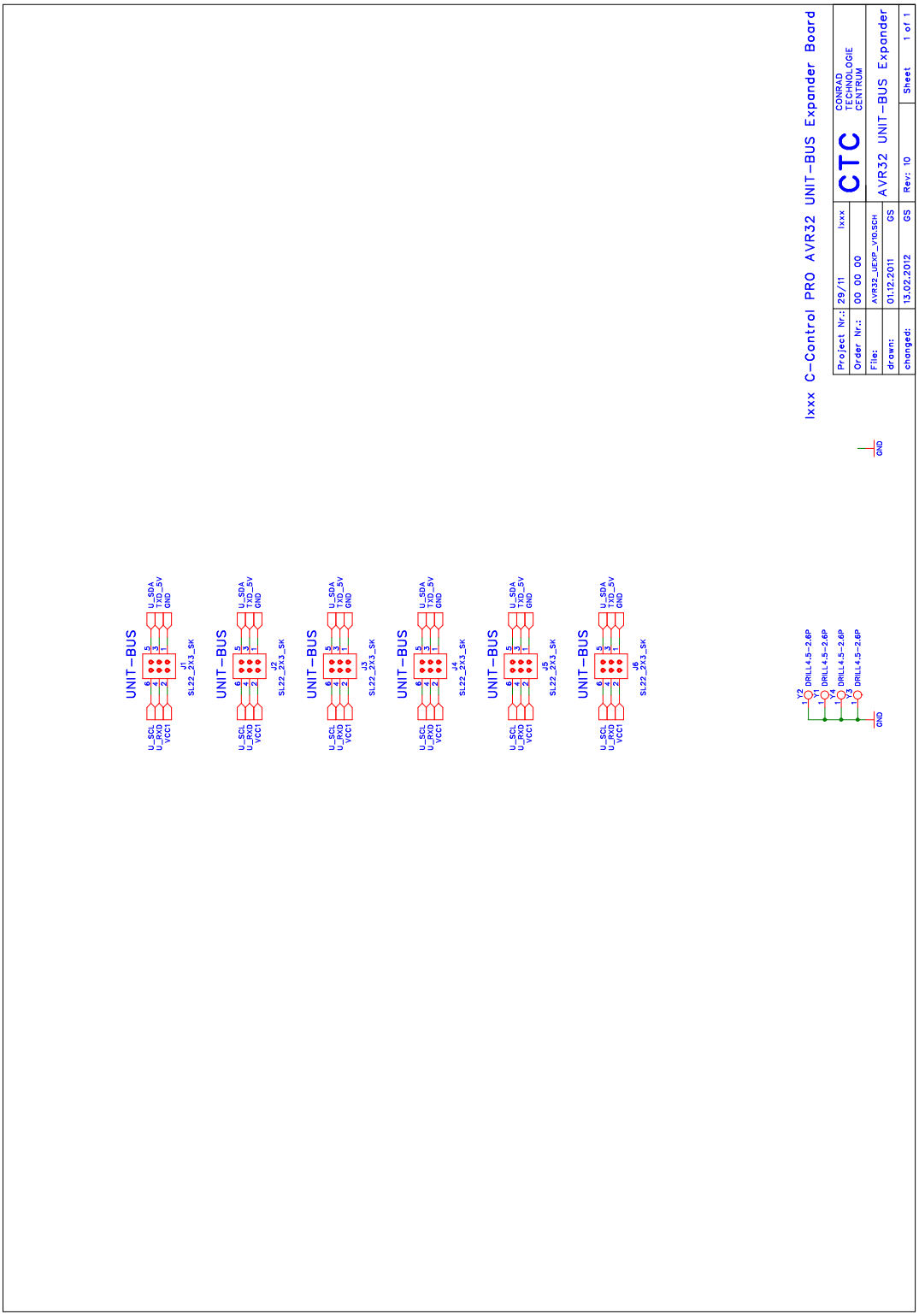


Übersicht Modul

Technische Daten

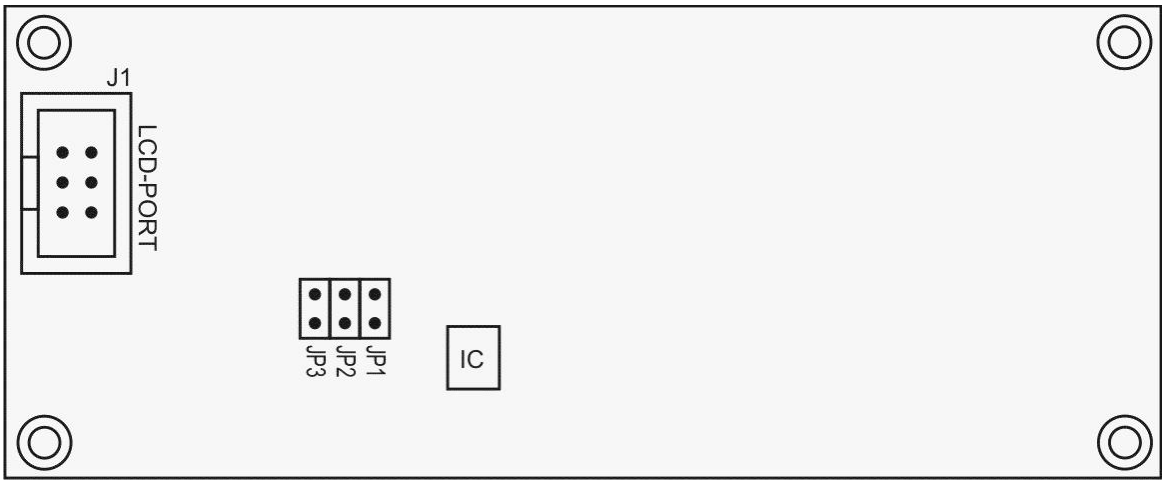
Abmessung	72mm x 20mm x 12mm (LxBxH)
Pfosten-Stecker Raster	2.54mm
Gewicht	12g

2.2.6.1 Schaltplan



2.2.7 LCD1602 Board

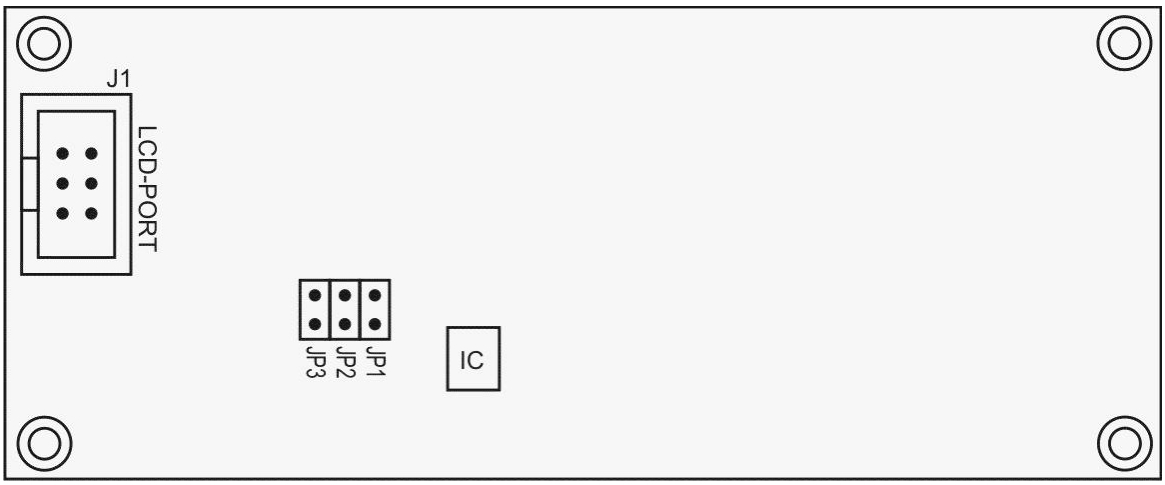
Das C-Control PRO AVR32Bit LCD1602 Board (Conrad Best.-Nr.: 192602) ist zur Erweiterung des Funktionsumfanges der C-Control PRO AVR32Bit Produkte bestimmt. Das Produkt ist als offene Leiterplatte ausgeführt. Das Modul ist mit einem zweizeiligen 16 Zeichen LC-Display mit Hintergrundbeleuchtung und einem 6polige Pfosten-Steckverbinder ausgestattet, und nur für das C-Control PRO AVR32Bit Mainboard (Conrad Best.-Nr.: 192702) bestimmt. Die Platine dient zur Anzeige von Daten der AVR32Bit UNIT (Conrad Best.-Nr.: 192573) in Verbindung mit dem AVR32Bit Mainboard. Die Ansteuerung der einzelnen Zusatzmodule erfolgt über Software. Die Software Beispiele finden Sie im Ordner der Beispielprogramme (siehe Demo Programme) und unter www.c-control.de.



Platine Vorderansicht

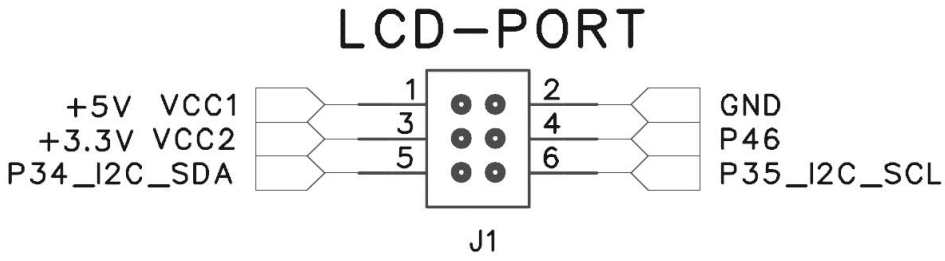
Anschluss und Inbetriebnahme

Vergewissern Sie sich, dass vor dem Anschluss der Module an Ihr C-Control PRO AVR32Bit Mainboard alle Verbindungen zu angeschlossenen Geräten getrennt und spannungsfrei sind. Auf dem C-Control PRO AVR32Bit Mainboard befindet sich ein 6poliger Pfosten-Steckverbinder mit der Bezeichnung LCD-PORT. Dieser Pfosten-Steckverbinder ist für den Anschluss des LCD1602-Boards geeignet. Auf diesen Pins sind I2C, P46, 3.3V und +5V herausgeführt. Das LCD1602 Board besitzt einen I2C-BUS Portexpander der für die Ansteuerung des LCDs zuständig ist. Dadurch werden weniger Pins der UNIT belegt.



Platine Rückansicht

➔ Wird die Basisadresse des LCD verwendet (gleich mit dem Applicationboard), so müssen die Jumper auf der LCD Platinen entfernt werden. Siehe auch LCD_SetDispAddr.



Anschlüsse des LCD-Port

Verbunden wird das Mainboard mit dem LCD1602 Board mittels einen 6poligen Flachbandkabel mit Pfosten-Steckverbinder (female). Da je nach Anwendung die Kabellänge variieren kann, bieten wir Ihnen diese Komponenten zur Selbstmontage unter folgenden Bestellnummern an.

Flachbandkabel RM1.27 0.05mm²: Best.-Nr. 607237
Pfostensteckverbinder 2x3 RM:2.54mm: Best.-Nr. 742063

Ein bereits fertig konfektioniertes Anschlusskabel (Länge: 35cm) erhalten Sie mit der Bestellnummer Best.-Nr. 198876.

➔ **Tipp**: Die Pfostensteckverbinder, lassen sich leicht mit einen kleinen Schraubstock zusammenpressen. Das Kabel auf die passende Länge kürzen und im Stecker gerade ausrichten (Führungsrillen im Stecker). Dann zwischen den beiden Schraubstock-Backen einspannen und diesen vorsichtig zudrehen, bis die Stecker Verriegelung einrastet.

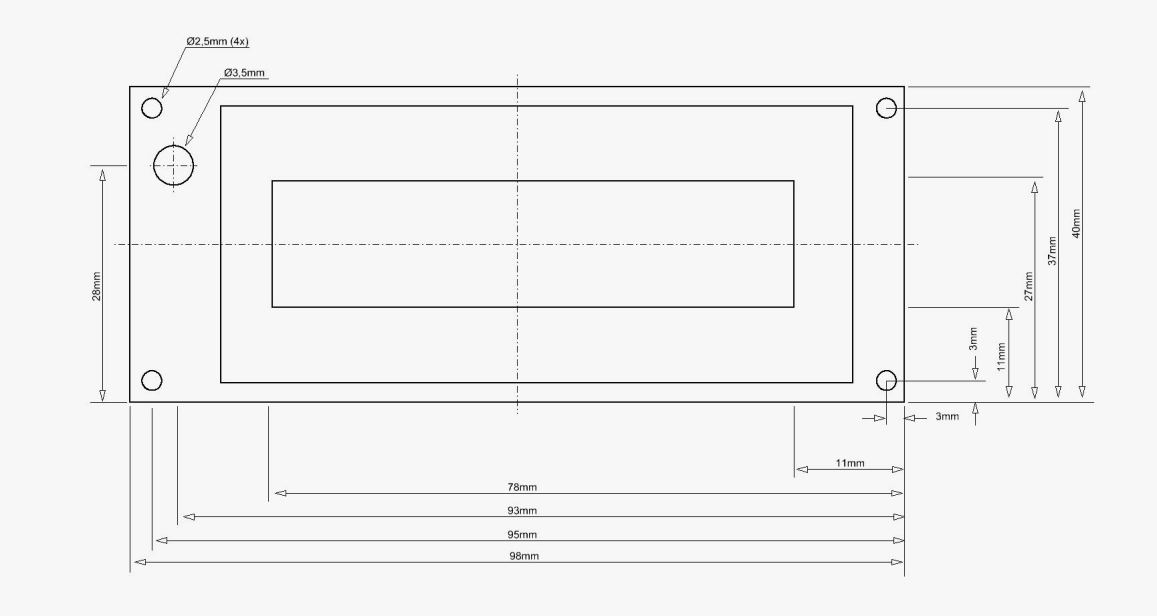
Adressierung:

Die I2C-Adressjumper JP1 bis JP3 befinden sich auf der Rückseite des LCD1602 Moduls. Bei der Verwendung von mehreren LCD-Modulen (max. 8) müssen die Jumper je nach gewünschter Adresse wie folgt gesteckt werden (Jumper JP1 = zeigt Richtung IC):

JP3	JP2	JP1	Adresse
-	-	-	Hex 27
-	-	x	Hex 26
-	x	-	Hex 25
-	x	x	Hex 24
x	-	-	Hex 23
x	-	x	Hex 22
x	x	-	Hex 21
x	x	x	Hex 20

x=Jumper gesteckt / - = nicht gesteckt

➔ Andere I2C-Bus Module der C-Control PRO verwenden den gleichen I2C-Expander Bausteine (PCA8574). Die Anzahl der maximalen Module mit diesem IC ist auf 8 Stück beschränkt!



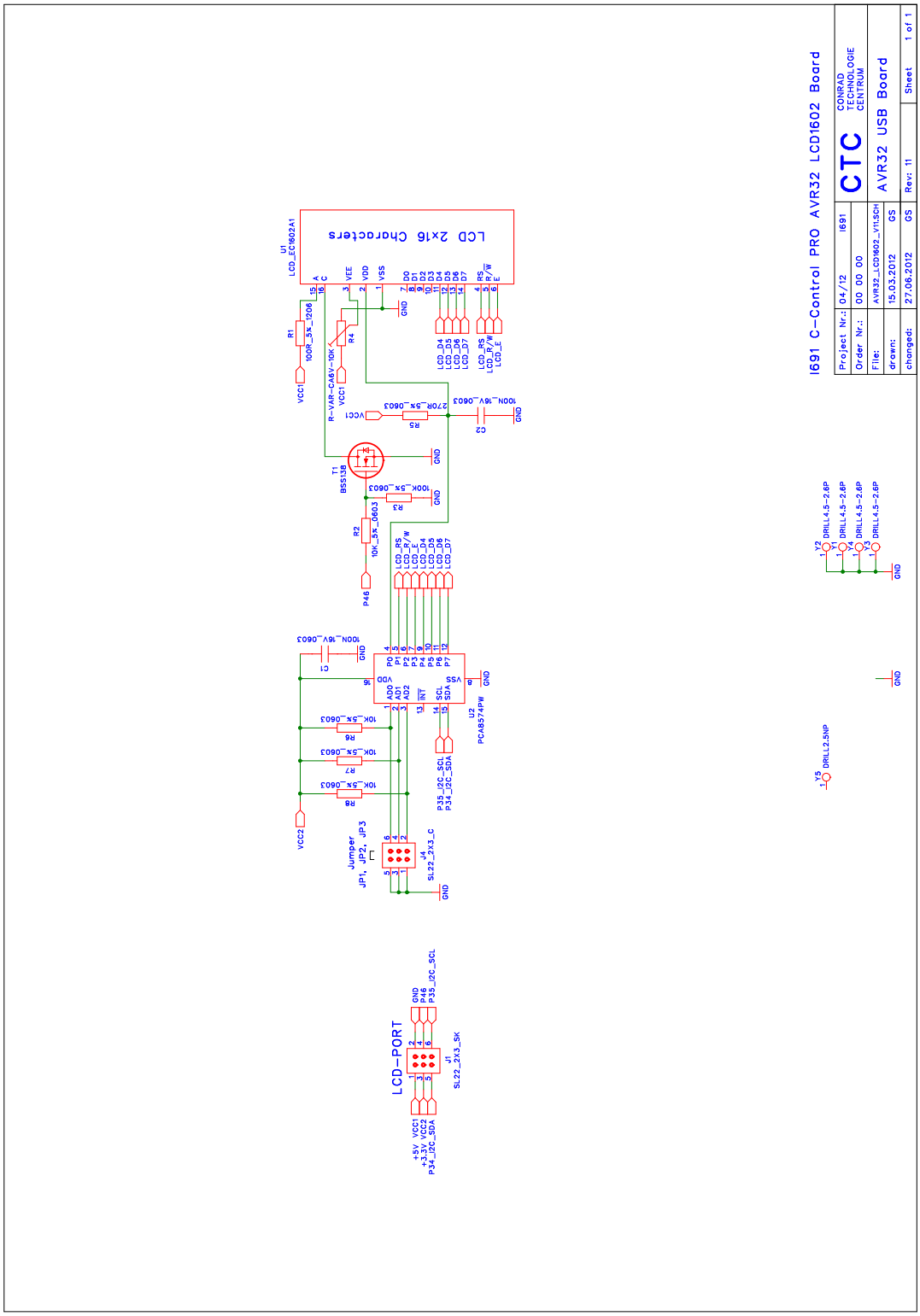
CAD

Technische Daten:

Abmessung	98 mm x 40 mm x 26 mm (LxBxH)
Pfosten-Stecker Raster	2.54 mm

Betriebsspannung	3.3 V (LCD) und 5 V (Beleuchtung)
Stromaufnahme ohne Beleuchtung	3 mA
Stromaufnahme mit Beleuchtung	13 mA
Gewicht	55 g

2.2.7.1 Schaltplan



2.2.8 Port-Ext-Board

Das Port-Ext-Board (Conrad Best.-Nr.: 192615) ist zur Erweiterung des Funktionsumfanges des C-Control PRO AVR32Bit Mainboards (Conrad Best.-Nr.: 192702) bestimmt. Das Produkt ist als offene Lochraster Experimentierleiterplatte mit einen Raster von 2.54mm ausgeführt. Sie ist mit zwei einzelnen 26-poligen Pfosten-Steckverbindern ausgestattet und nur für die Portausgänge (PORT-1/ PORT-2) geeignet. Die Platine dient zum Aufbau eigener Schaltungen in Verbindung mit dem C-Control PRO AVR32Bit Mainboard. Für den komfortablen Schaltungsauf- und Nachbau ist die Platine mit einen Koordinatensystem bedruckt. Die Platine kann direkt über, unter oder neben dem Mainboard montiert werden. Bei der „Sandwich“ Montage werden Platinen-Abstandshalter ab 20mm Länge und einem Gewindedurchmesser von 3mm benötigt.

Anschluss und Inbetriebnahme

.

Vergewissern Sie sich, dass vor dem Anschluss der Module an Ihr C-Control PRO AVR32Bit Mainboard alle Verbindungen zu angeschlossenen Geräten getrennt und spannungsfrei sind. Auf dem C-Control PRO AVR32Bit Mainboard befinden sich zwei 26-polige Pfosten-Steckverbinder mit der Bezeichnung PORT-1 und PORT-2. Diese Pfosten-Steckverbinder sind für den Anschluss des Port-Extension-Board geeignet. Auf diese Pins sind die freien Ports der C-Control PRO AVR32Bit UNIT herausgeführt (siehe Port-Extension-Board Übersicht).

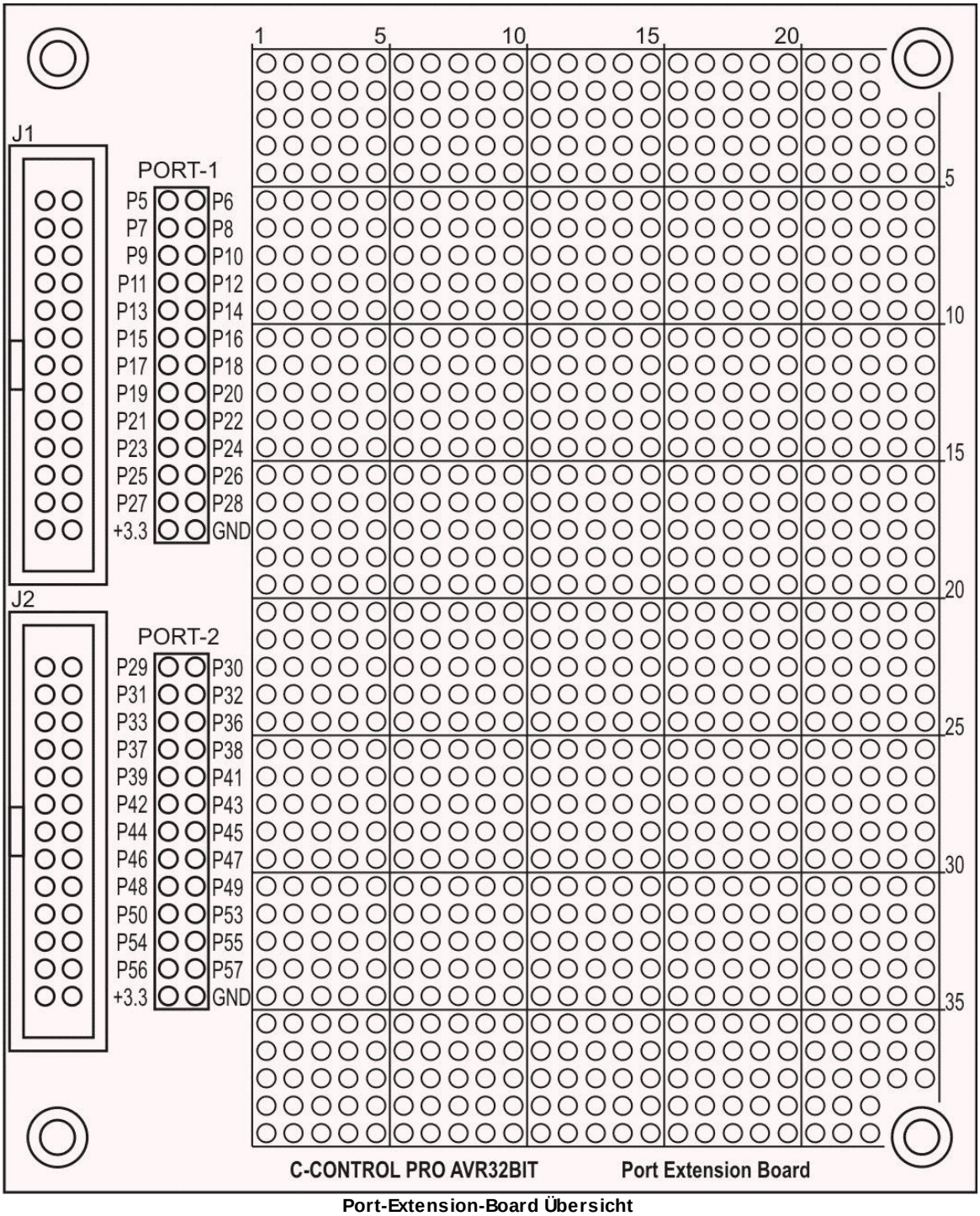
Herausgeführte Ports an Pfostensteckverbinder PORT-1:
P5, P6, P7, P8, P9 , P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, +3.3V, GND

Herausgeführte Ports an Pfostensteckverbinder PORT-2:
P29, P30, P31, P32, P33, P36, P37, P38, P39, P41, P42, P43, P44, P45, P46, P47, P48, P49, P50, P53, P54, P55, P56, P57, +3.3V, GND

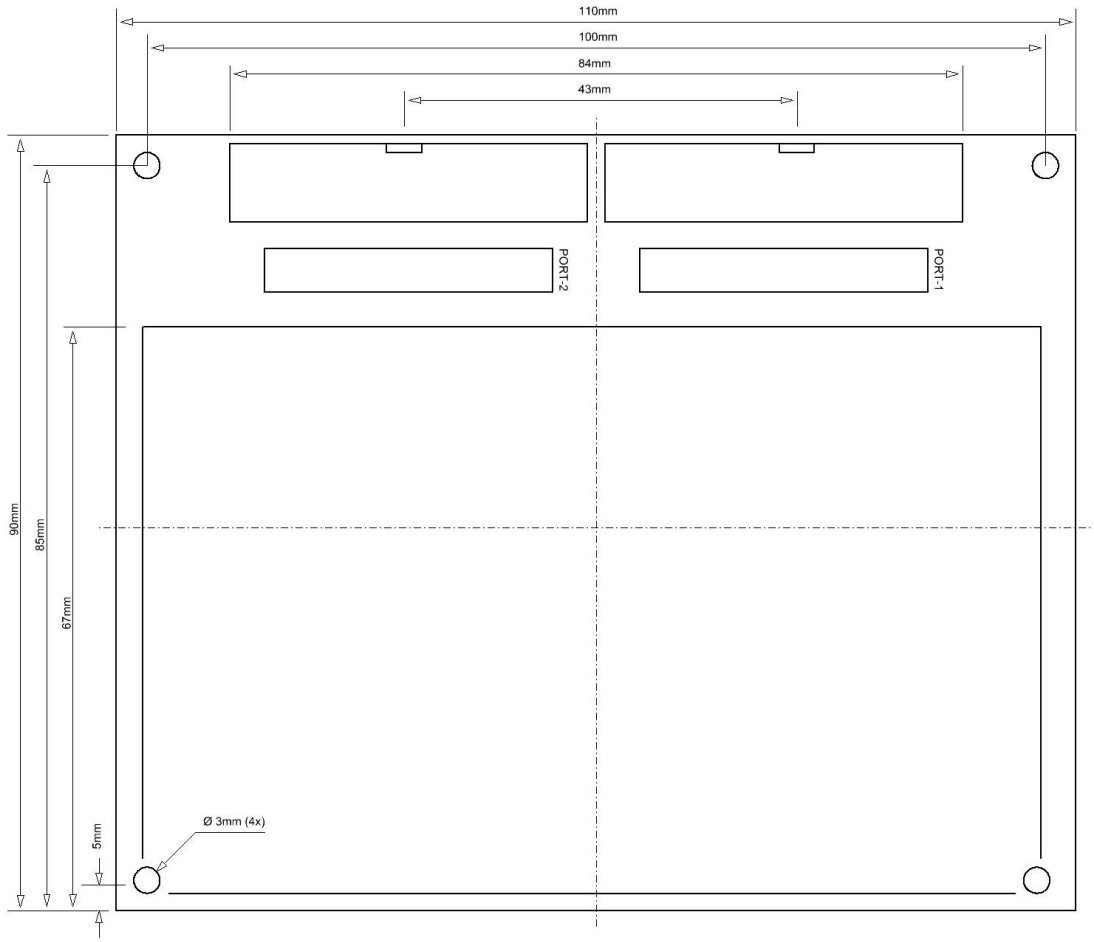
Verbunden wird das Mainboard mit dem Port-Extension-Board mittels zweier Flachbandkabel mit Pfosten-Steckverbinder (female). Da je nach Anwendung die Kabellänge variieren kann, bieten wir Ihnen diese Komponenten zur Selbstmontage unter folgenden Bestellnummern an.

Flachbandkabel 26polig RM1.27 0.05mm²: Best.-Nr. 607222
Pfostensteckverbinder 2x13 RM:2.54mm: Best.-Nr. 742185

Tipp: Die Pfostensteckverbinder, lassen sich leicht mit einen kleinen Schraubstock zusammenpressen. Das Kabel auf die passende Länge kürzen und im Stecker gerade ausrichten (Führungsrillen im Stecker), und dann zwischen den beiden Schraubstock-Backen einspannen, und diesen vorsichtig zudrehen bis die Stecker Verriegelung einrastet.



Port-Extension-Board Übersicht



CAD

Technische Daten:

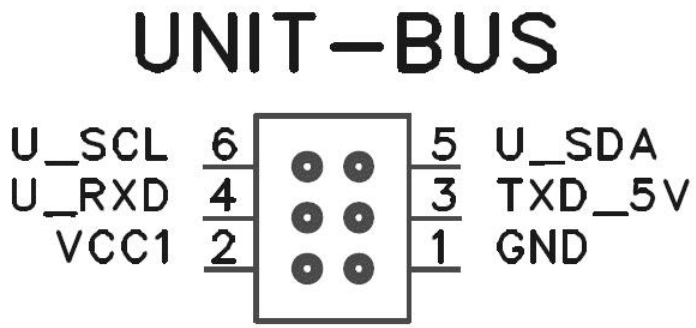
Abmessung	110mm x 90mm x 13mm (LxBxH)
Pfosten-Stecker Raster	2.54 mm
Lochraster der Platine	2.54 mm
Größe Lochrasterfeld	25x40 Pins
Gewicht	35g

2.2.9 REL4-Board

Das C-Control PRO AVR32Bit REL4-Board (Conrad Best.-Nr.: 192631) ist zur Erweiterung des Funktionsumfanges der C-Control PRO AVR32Bit Produkte bestimmt. Das Produkt ist als offene Leiterplatte ausgeführt. Die Platime ist mit 4 Relais zum Schalten von Verbrauchern ausgestattet, und nur für die C-Control Produkte bestimmt.

Anschluss und Inbetriebnahme

Vergewissern Sie sich, dass vor dem Anschluss der Module an Ihr C-Control PRO z.B. AVR32Bit Applicationboard bzw. Mainboard alle Verbindungen zu angeschlossenen Geräten getrennt und spannungsfrei sind. Auf den C-Control PRO Systemen, befindet sich ein 6-poliger Pfosten-Steckverbinder mit der Bezeichnung UNIT-BUS. Dieser Pfosten-Steckverbinder ist für den Anschluss des REL4-Boards geeignet. An diesen Pins sind I2C, UART und +5V herausgeführt. Das REL4-Board besitzt einen I2C-BUS Portexpander, der für die Ansteuerung der Relais zuständig ist. Durch die I2C-Bus Ansteuerung werden keine zusätzlichen Ein-/ Ausgabepins (I/O) Pins der UNIT benötigt.



UNIT-BUS Belegung

Verbunden wird das REL4-Board mittels einen 6poligen Flachbandkabel mit Pfosten-Steckverbinder (UNIT-BUS) und einer Schraubklemme mit der Bezeichnung „VREL“. Da je nach Anwendung die Kabellänge des UNIT-BUS variieren kann, bieten wir Ihnen diese Komponenten zur Selbstmontage unter folgenden Bestellnummern an:

Flachbandkabel RM1.27 0.05mm²: Best.-Nr. 607237
Pfostensteckverbinder 2x13 RM:2.54mm: Best.-Nr. 742063

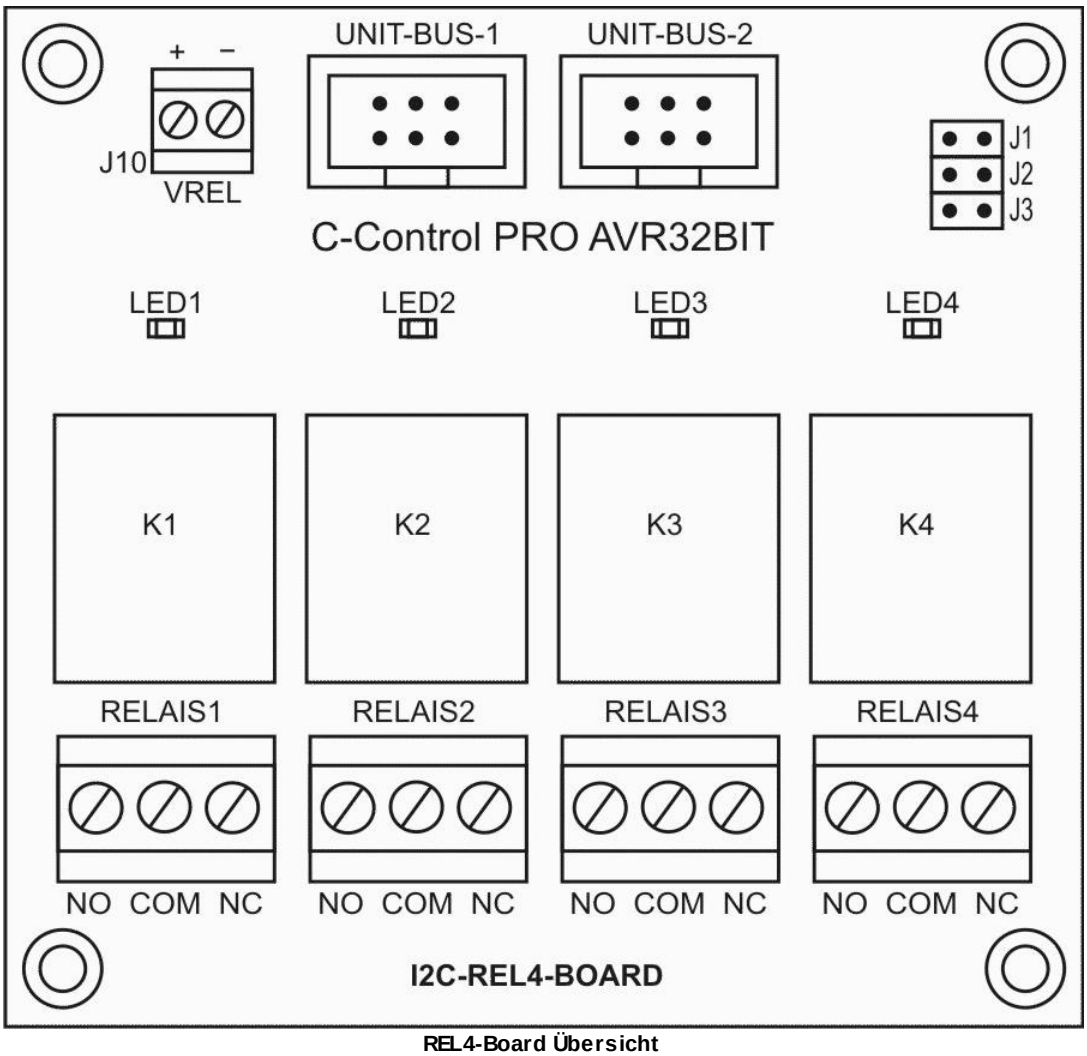
Das REL4-Board Module besitzt zwei UNIT-BUS Anschlüsse die untereinander 1:1 verbunden sind. Somit kann einer der Anschlüsse z.B. mit dem Applicationboard bzw. Mainboard verbunden werden, und der zweite kann als Verteiler dienen an den man ein weiteres UNIT-BUS Modul ansteckt. Die Leitungslänge darf 2m nicht überschreiten, da es sonst zu Kommunikationsfehlern kommt. Werden dennoch längere Leitungen benötigt, so ist es hilfreiche den I2C-Leistungstreiber Best.-Nr. 19 82 80

dazwischen zu schalten.

Das REL4-Board wird über die Schraubklemme „VREL“ mit Strom versorgt. Der UNIT-BUS dient bei diesem Modul nur als Kommunikationsschnittstelle und zur Versorgung des Digitalen Elektronikteils der Schaltung. Die Relais werden extern über die „VREL“ versorgt!

Hinweis: Passendes Anschlusskabel bereits konfektioniert (Länge: 35cm) Best.-Nr. 198876.

Tipp: Die Pfostensteckverbinder, lassen sich leicht mit einen kleinen Schraubstock zusammenpressen. Das Kabel auf die passende Länge kürzen und im Stecker gerade ausrichten (Führungsrillen im Stecker), und dann zwischen den beiden Schraubstock-Backen einspannen, und diesen vorsichtig zudrehen bis die Stecker Verriegelung einrastet.



REL4-Board Übersicht

Adressierung:

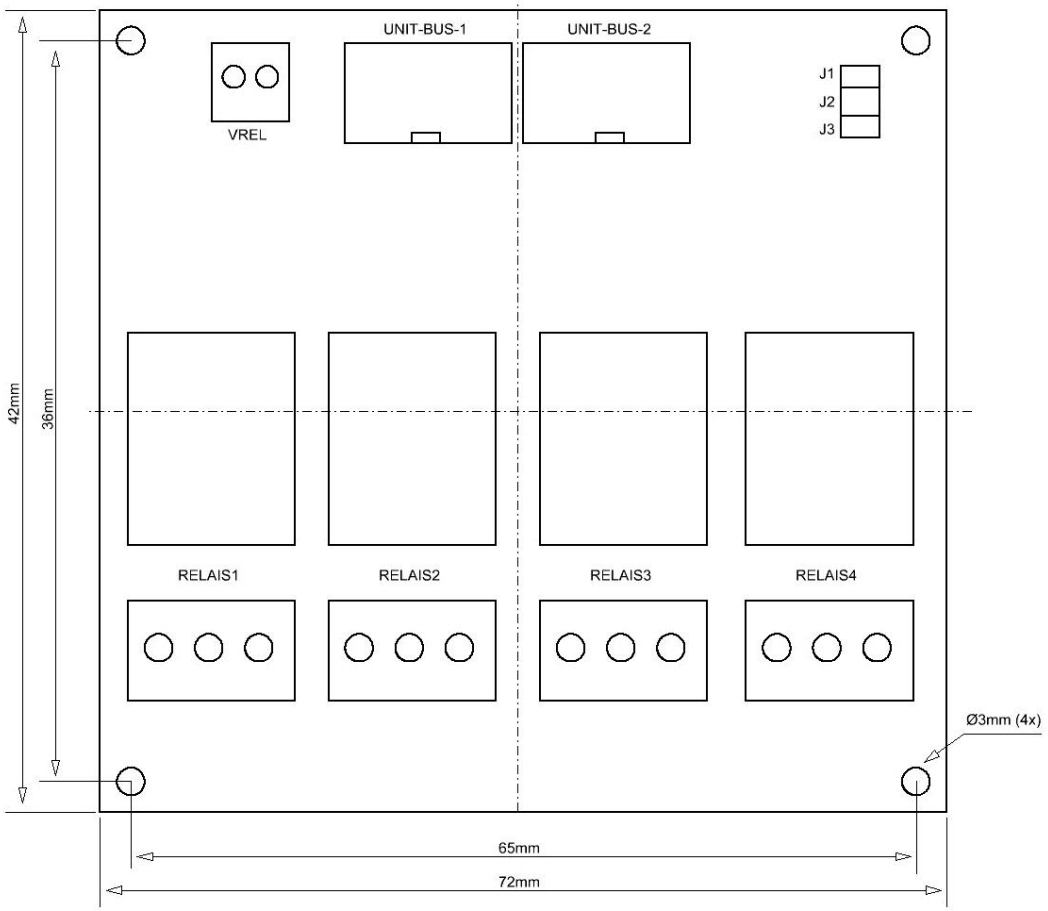
Die I2C-Adressjumper J1 bis J3 befinden sich auf der Vorderseite des REL4-Boards. Bei der Verwendung von mehreren Modulen müssen die Jumper je nach gewünschter Adresse wie folgt gesteckt

werden:

J3	J2	J1	Adresse
-	-	-	Hex 27
-	-	x	Hex 26
-	x	-	Hex 25
-	x	x	Hex 24
x	-	-	Hex 23
x	-	x	Hex 22
x	x	-	Hex 21
x	x	x	Hex 20

x=Jumper gesteckt / - = nicht gesteckt
Bild: Tabelle der möglichen Adressen

Achtung: Andere I2C-Bus Module der C-Control PRO, verwenden den gleichen I2C-Expander Bausteine (PCA8574), die Anzahl der maximalen Module mit diesem IC ist auf 8 Stück beschränkt!



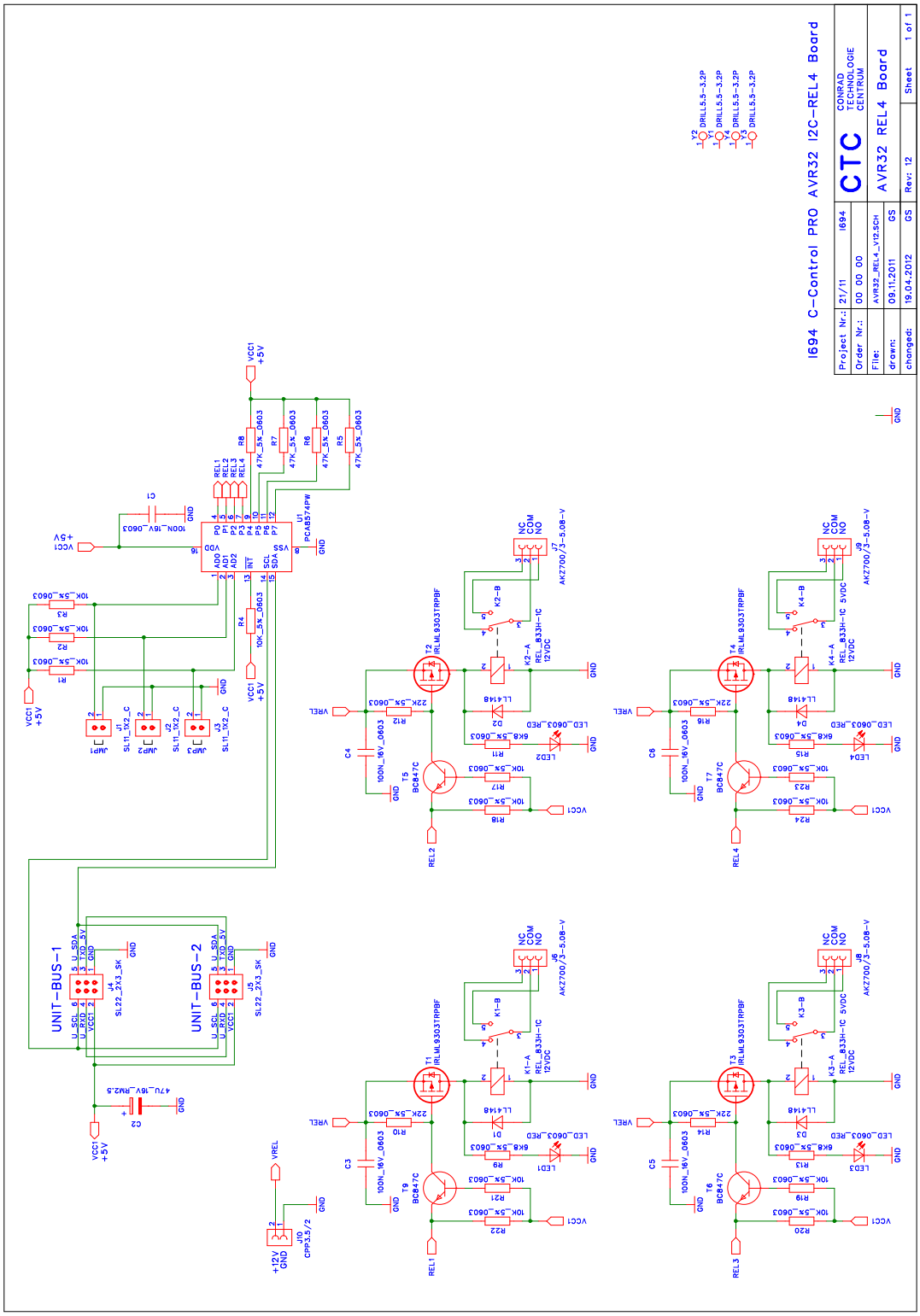
CAD

Info: Das REL4-Board kann mittels Phönix Contact Hutschienenträger der Serie „UMK“ auf einer Normschiene montiert werden.

Technische Daten:

Abmessung	76mm x 72mm x 18mm (LxBxH)
Pfosten-Stecker Raster	2.54 mm
Relais	NC/NO, max. 24V/7A
Betriebsspannung	12V
Stromaufnahme	120mA
Gewicht	70g

2.2.9.1 Schaltplan

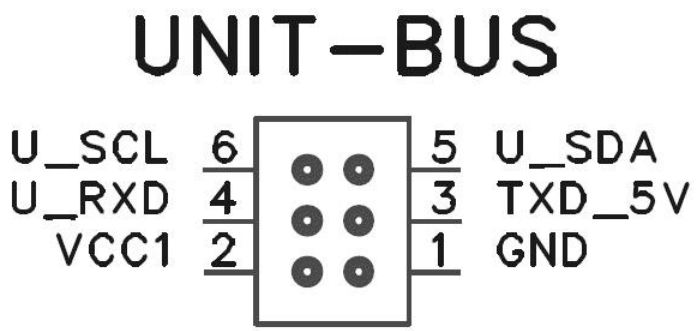


2.2.10 RELBUS-Board

Das C-Control PRO AVR32Bit RELBUS-Board (Conrad Best.-Nr.: 192645) ist zur Erweiterung des Funktionsumfanges der C-Control PRO AVR32Bit Produkte bestimmt. Das Produkt ist als offene Leiterplatte ausgeführt. Sie ist mit 8 Open-Source Schaltstufen (High-Side Switch) zum Schalten von 8 Kleinverbrauchern wie z.B. Relais ausgestattet und nur für den C-Control PRO UNIT-BUS bestimmt.

Anschluss und Inbetriebnahme

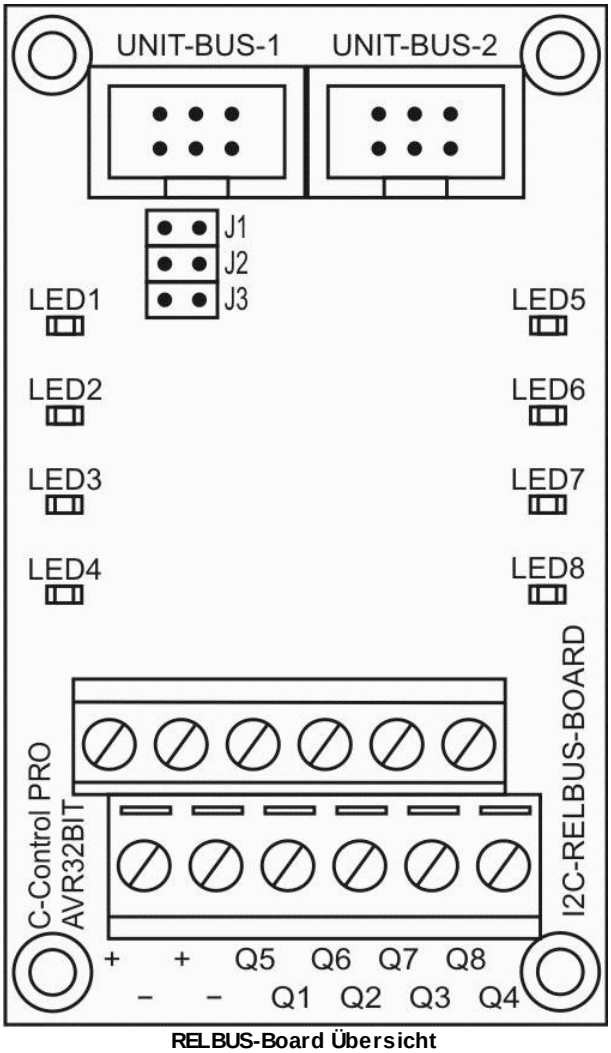
Vergewissern Sie sich, dass vor dem Anschluss der Module an Ihr C-Control PRO z.B. AVR32Bit Applicationboard bzw. Mainboard alle Verbindungen zu angeschlossenen Geräten getrennt und spannungsfrei sind. Auf den C-Control PRO Systemen befindet sich ein 6-poliger Pfosten-Steckverbinder mit der Bezeichnung UNIT-BUS. Dieser Pfosten-Steckverbinder ist für den Anschluss des RELBUS-Boards geeignet. An diesen Pins sind I2C, UART und +5V herausgeführt. Das RELBUS-Board besitzt einen I2C-BUS Portexpander, der für die Ansteuerung der Verbraucher (z.B. Relais) zuständig ist. Durch die I2C-Bus Ansteuerung werden keine zusätzlichen Ein-/Ausgabepins (I/O) Pins der UNIT benötigt.



UNIT-BUS Belegung

Verbunden wird das RELBUS-Board mittels eines 6-poligen Flachbandkabels mit Pfosten-Steckverbinder (UNIT-BUS). Da je nach Anwendung die Kabellänge des UNIT-BUS variieren kann, bieten wir Ihnen diese Komponenten zur Selbstmontage unter folgenden Bestellnummern an.

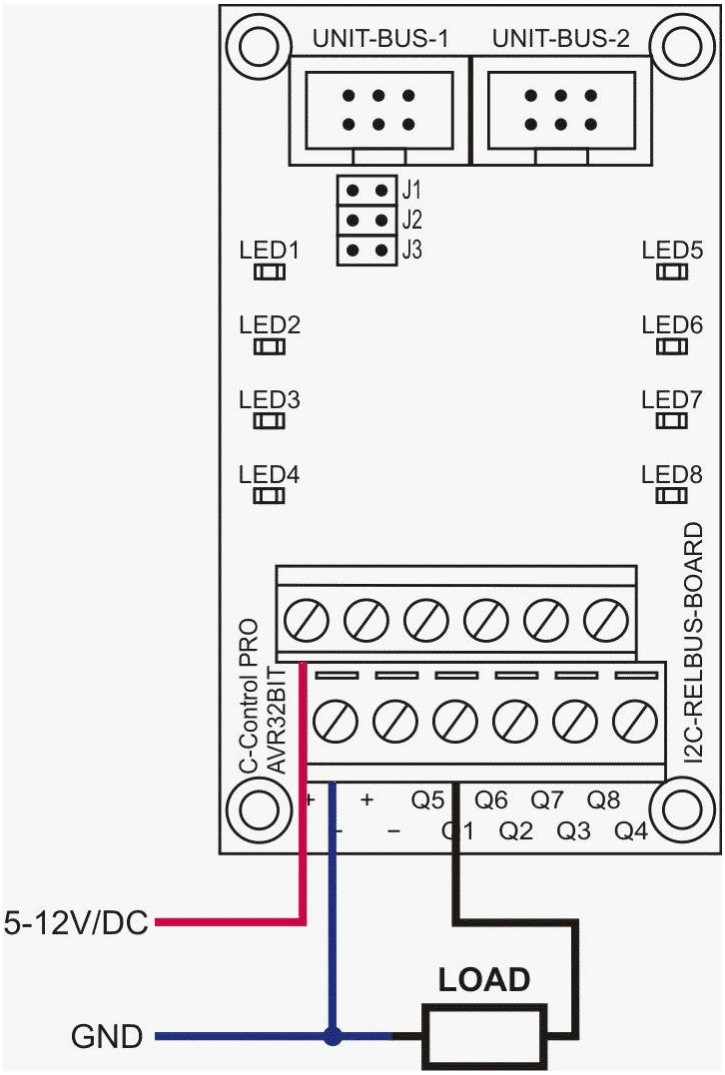
Flachbandkabel RM1.27 0.05mm²: Best.-Nr. 607237
Pfostensteckverbinder 2x3 RM:2.54mm: Best.-Nr. 742063



RELBUS-Board Übersicht

Das RELBUS-Board besitzt zwei UNIT-BUS Anschlüsse die untereinander 1:1 verbunden sind. Somit kann einer der Anschlüsse z.B. mit dem Applicationboard bzw. Mainboard verbunden werden, und der zweite Anschluss kann als Verteiler dienen, an den man ein weiteres UNIT-BUS Modul ansteckt. Die Leitungslänge darf 2m nicht überschreiten, da es sonst zu Kommunikationsfehlern kommt. Werden dennoch längere Leitungen benötigt, so ist es hilfreiche den I2C-Leistungstreiber Best.-Nr. 19 82 80 dazwischen zu schalten.

Das RELBUS-Board wird über den UNIT-BUS mit Strom versorgt. An den Schraubklemme mit der Bezeichnung „+ und -“ wird die Verbraucher Stromversorgung hergestellt. Der UNIT-BUS dient bei diesem Modul nur als Kommunikationsschnittstelle und zur Versorgung des Digitalen Elektronikteils der Schaltung. Die Verbraucher werden extern über die Anschlüsse „+ und -“ versorgt!



Beispiel Verbraucheranschluss an Ausgang Q1

➔ **Hinweis**: Passendes Anschlusskabel bereits konfektioniert (Länge: 35cm) Best.-Nr. 198876.

➔ **Tipp**: Die Pfostensteckverbinder, lassen sich leicht mit einen kleinen Schraubstock zusammenpressen. Das Kabel auf die passende Länge kürzen und im Stecker gerade ausrichten (Führungsrillen im Stecker), und dann zwischen den beiden Schraubstock-Backen einspannen, und diesen vorsichtig zudrehen bis die Stecker Verriegelung einrastet.

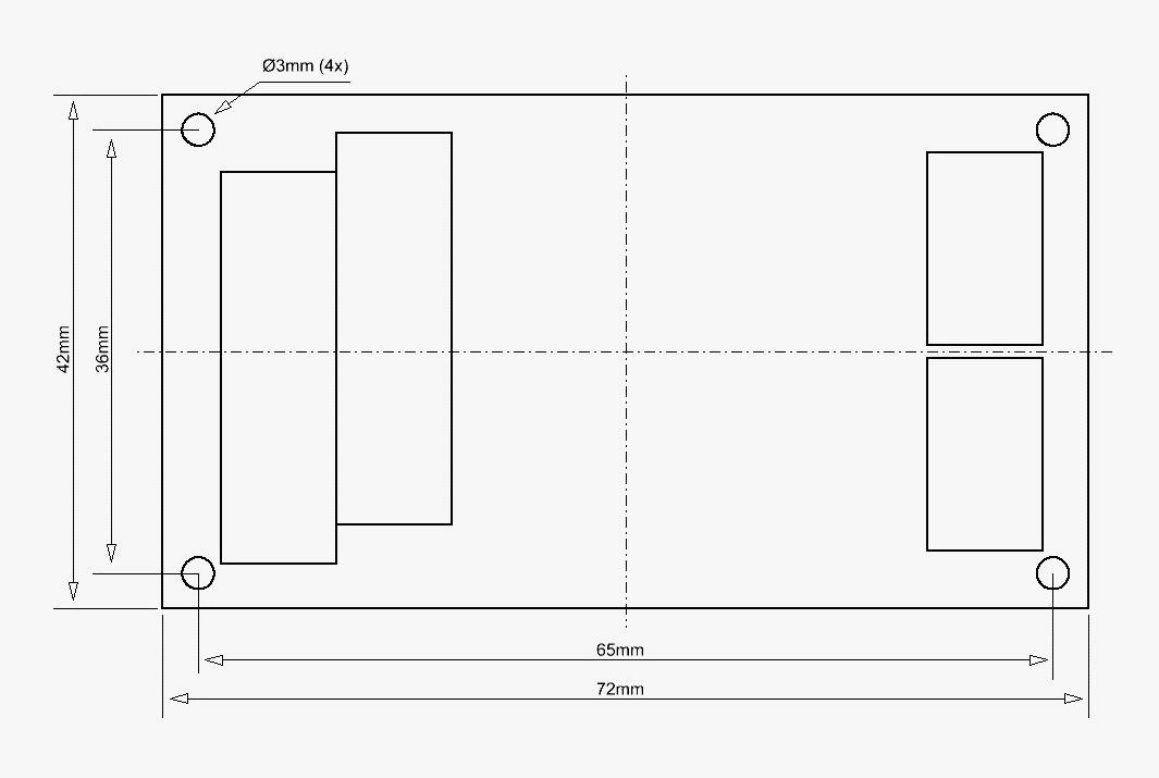
➔ **Achtung**: Andere I2C-Bus Module verwenden den gleichen I2C-Expander Bausteine (PCA8574), die Anzahl der maximalen Module mit diesem IC ist auf 8 Stück beschränkt!

Adressierung:

Die I2C-Adressjumper J1 bis J3 befinden sich auf der Vorderseite des REL4-Boards. Bei der Verwendung von mehreren Modulen müssen die Jumper je nach gewünschter Adresse wie folgt gesteckt werden.

J3	J2	J1	Adresse
-	-	-	Hex 27
-	-	x	Hex 26
-	x	-	Hex 25
-	x	x	Hex 24
x	-	-	Hex 23
x	-	x	Hex 22
x	x	-	Hex 21
x	x	x	Hex 20

x=Jumper gesteckt / - = nicht gesteckt



CAD

➔ **Info**: Das REL4-Board kann mittels Phönix Contact Hutschienenträger der Serie „UMK“ auf einer Normschiene montiert werden.

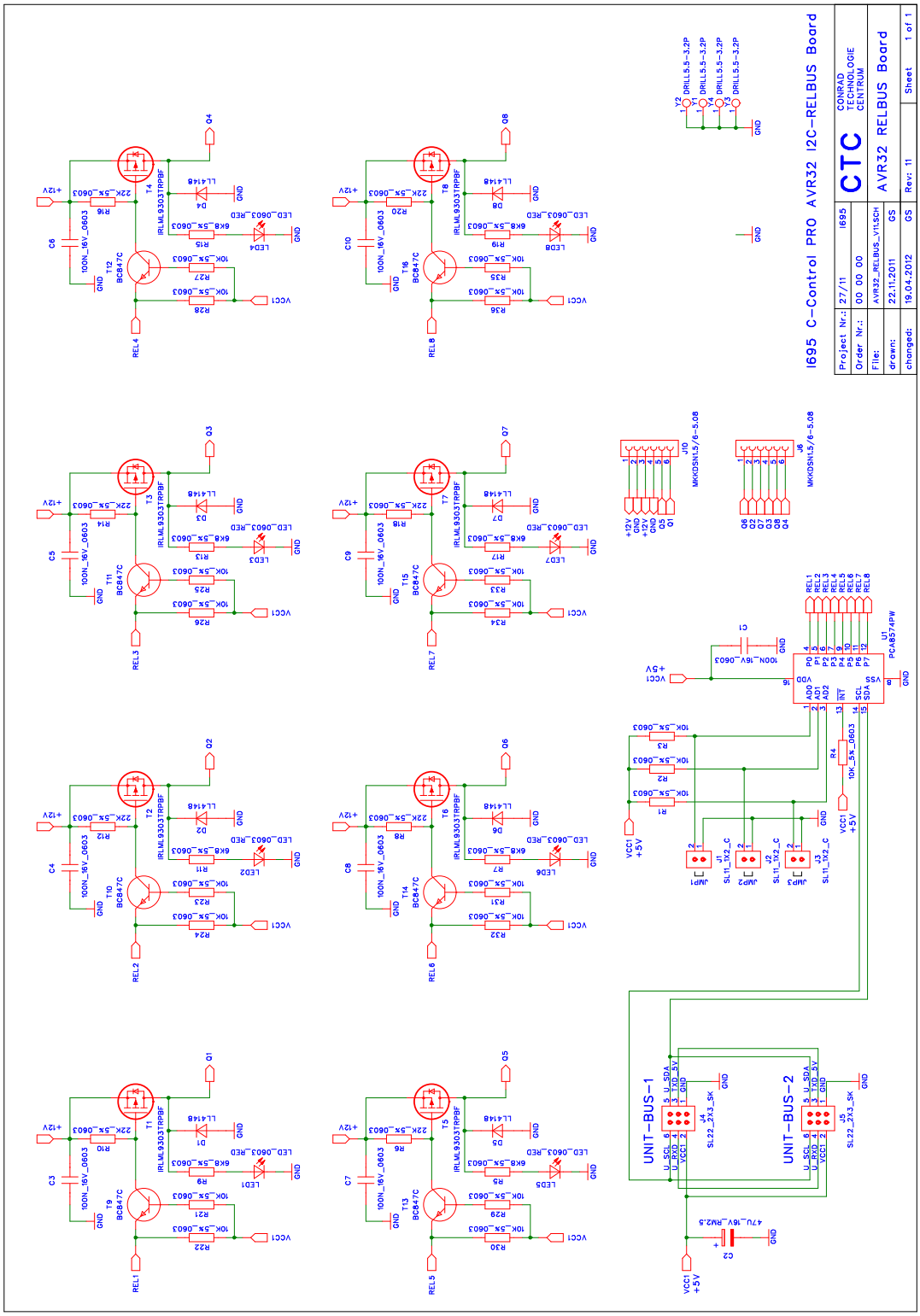
➔ **Achtung**: Die Ausgänge sind nicht kurzschlussfest und werden bei einem Kurzschluss gegen Masse zerstört!

Technische Daten:

Abmessung	42mm x 72mm x 22mm (LxBxH)
Pfosten-Stecker Raster	2.54 mm
Ausführung Ausgänge	Open-Source (High-Side Switch)

Ausgang Last	max je Ausgang. 5 bis 12VDC/200mA
Betriebsspannung	5V über UNIT-BUS
Stromaufnahme	20mA
Gewicht	30g

2.2.10.1 Schaltplan

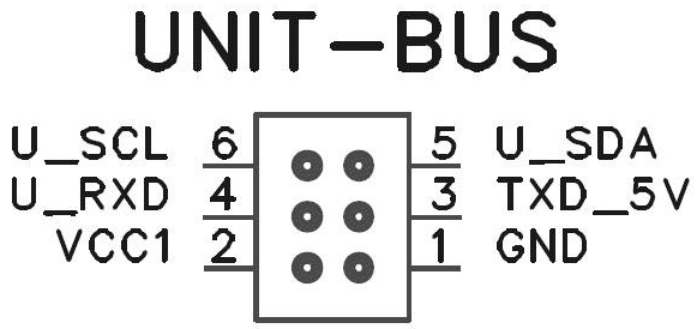


2.2.11 UNIT-BUS Ext-Board

Das C-Control PRO AVR32Bit UNIT-BUS Ext-Board (Conrad Best.-Nr.: 192673) ist zur Erweiterung des Funktionsumfanges der C-Control PRO AVR32Bit Produkte bestimmt. Das Produkt ist als offene Lochraster Experimentierleiterplatte mit einen Raster von 2.54mm ausgeführt. Sie ist mit zwei 6poligen Pfosten-Steckverbinder bestückt und nur für den C-Control UNIT-BUS bestimmt. Die Platine dient zum Aufbau eigener Schaltungen. Für den komfortablen Schaltungsauf- und Nachbau ist die Platine mit einem Koordinatensystem bedruckt. Die Platine kann direkt über, unter oder neben dem Mainboard montiert werden. Bei der „Sandwich“ Montage werden Platinen-Abstandshalter ab 20mm Länge und einem Gewindedurchmesser von 3mm benötigt.

Anschluss und Inbetriebnahme

Vergewissern Sie sich, dass vor dem Anschluss der Module an Ihr C-Control Pro AVR32Bit Mainboard alle Verbindungen zu angeschlossenen Geräten getrennt und spannungsfrei sind. Auf den C-Control PRO AVR32Bit Applicationboard und Mainboard befindet sich ein 6-poliger Pfosten-Steckverbinder mit der Bezeichnung UNIT-BUS. Dieser Pfosten-Steckverbinder ist für den Anschluss des UNIT-BUS-Extension-Boards geeignet. Auf diesen Pins sind I2C, UART und +5V herausgeführt. Die C-Control PRO AVR32Bit UNIT arbeitet mit 3.3V Pegel, und die Erweiterungen, wie auch die älteren C-Control I I2C-Bus Module, benutzen 5V. Deshalb ist zwischen der C-Control PRO AVR32Bit UNIT und dem UNIT-BUS ein Pegelwandler vorhanden, der die 3.3V Signale der UNIT auf 5V Signale des UNIT-BUS umsetzt.



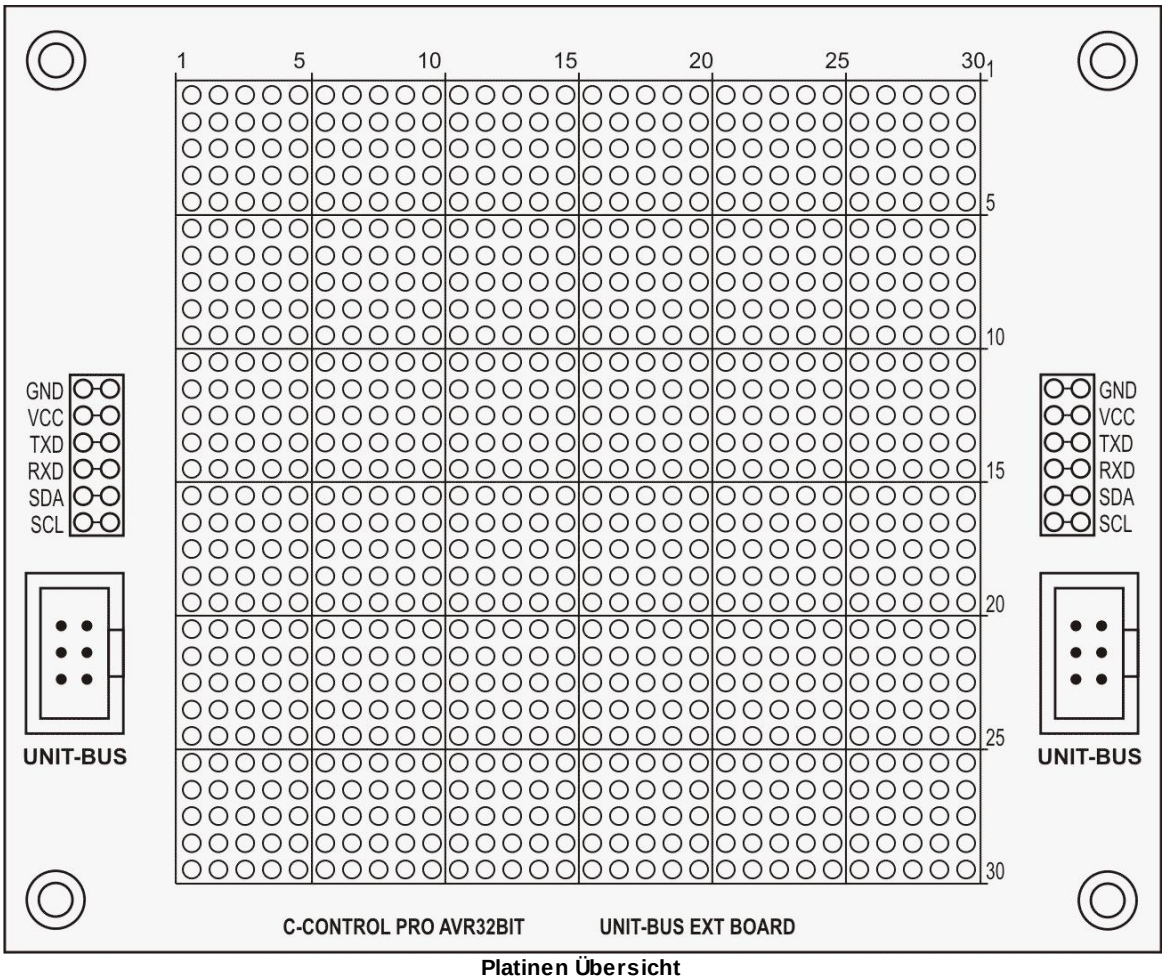
UNIT-BUS Belegung

Verbunden wird das UNIT-BUS-Extension-Board mittels eines 6-poligen Flachbandkabels mit Pfosten-Steckverbinder (female). Da je nach Anwendung die Kabellänge variieren kann, bieten wir Ihnen diese Komponenten zur Selbstmontage unter folgenden Bestellnummern an:

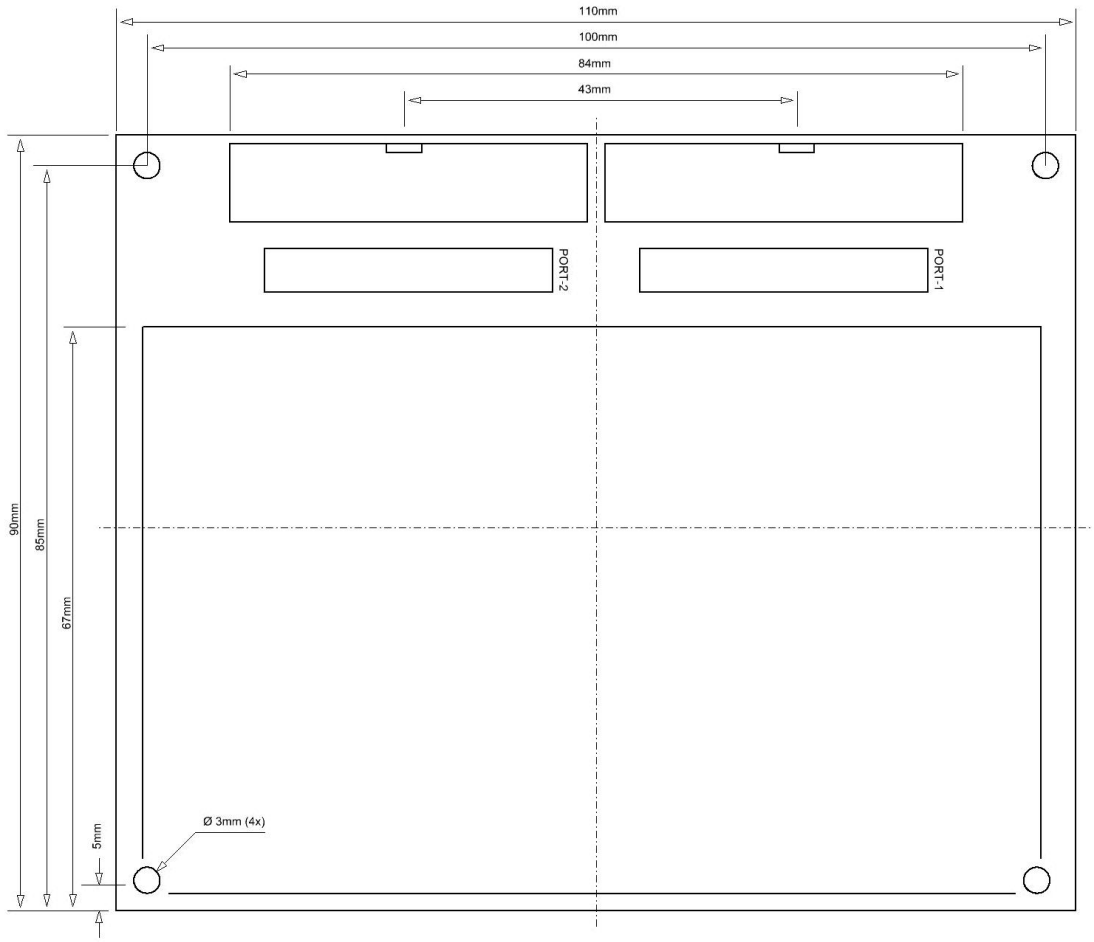
Flachbandkabel RM1.27 0.05mm²:	Best.-Nr. 607237
Pfostensteckverbinder 2x3 RM:2.54mm:	Best.-Nr. 742063

- Passendes Anschlusskabel bereits konfektioniert (Länge: 35cm): Best.-Nr. 198876
- Verwenden Sie um ein sauberes Signal zu erhalten, bei langen Leitungslängen und hohen Leitungskapazitäten den I2C-Leitungstreiber Best.-Nr. 198280

Tipp: Die Pfostensteckverbinder, lassen sich leicht mit einen kleinen Schraubstock zusammenpressen. Das Kabel auf die passende Länge kürzen und im Stecker gerade ausrichten (Führungsrillen im Stecker), und dann zwischen den beiden Schraubstock-Backen einspannen, und diesen vorsichtig zudrehen bis die Stecker Verriegelung einrastet.



Platinen Übersicht



CAD

Technische Daten:

Abmessung	110mm x 90mm x 13mm (LxBxH)
Pfosten-Stecker Raster	2.54 mm
Lochraster der Platine	2.54 mm
Größe Lochrasterfeld	30x30 Pins
Gewicht	30g

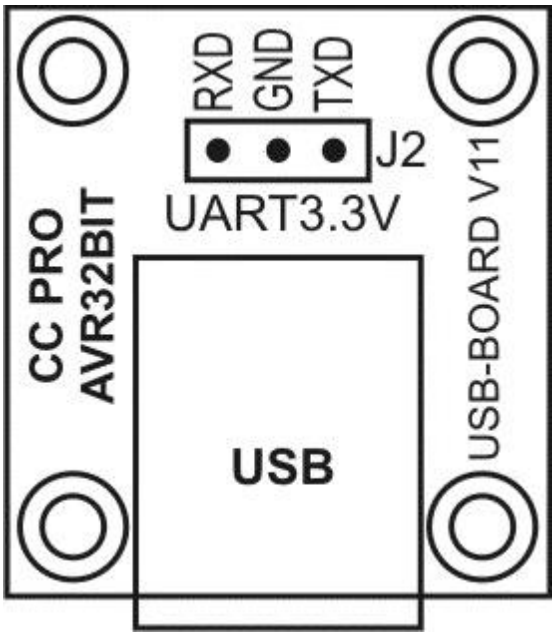
2.2.12 USB-Board

Das C-Control PRO AVR32Bit USB-Board (Conrad Best.-Nr.: 192688) ist zur Erweiterung des Funktionsumfanges der C-Control PRO AVR32Bit Produkte bestimmt. Das Produkt ist als offene Leiterplatte ausgeführt. Sie ist mit einem USB-Steckverbinder Type B und einer 3 polige Stiftleiste (Rastermaß: 2.54mm) zum Anschluss an einer 3.3V UART-Schnittstellen der C-Control PRO AVR32Bit ausgestattet.

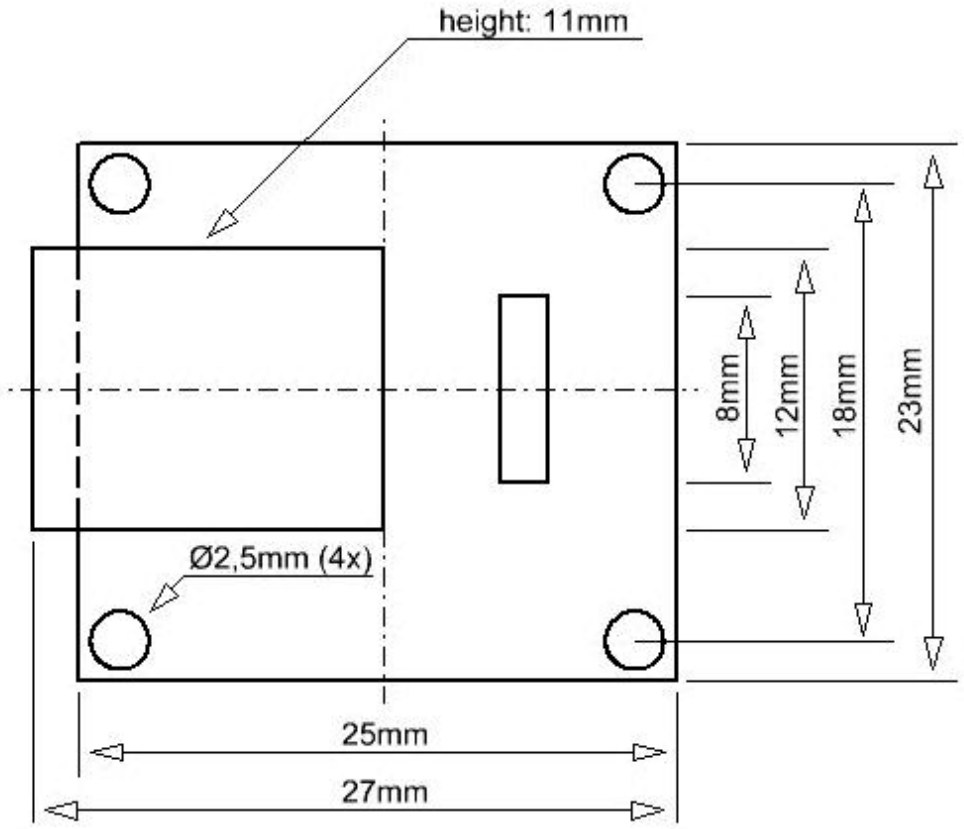
Anschluss und Inbetriebnahme

Vergewissern Sie sich, dass vor dem Anschluss der Module an Ihr C-Control PRO System alle Verbindungen zu angeschlossenen Geräten getrennt und spannungsfrei sind. Die C-Control PRO AVR32Bit UNIT besitzt mehrere UART-Schnittstellen (siehe Anleitung), die mit den USB-Board verbunden werden können. Auf den Boards z.B. Applicationboard oder Mainboard sind die UART-Schnittstellen über Kontakte erreichbar, an dem das Board angeschlossen werden kann. Das USB-Board dient als Schnittstelle zwischen UNIT UART-Schnittstelle und einen PC USB-Anschluss um mit diesen Daten auszutauschen.

➔ Unter folgenden Link können Sie die Treiber für das Produkt herunterladen: http://www.silabs.com



Platinenübersicht

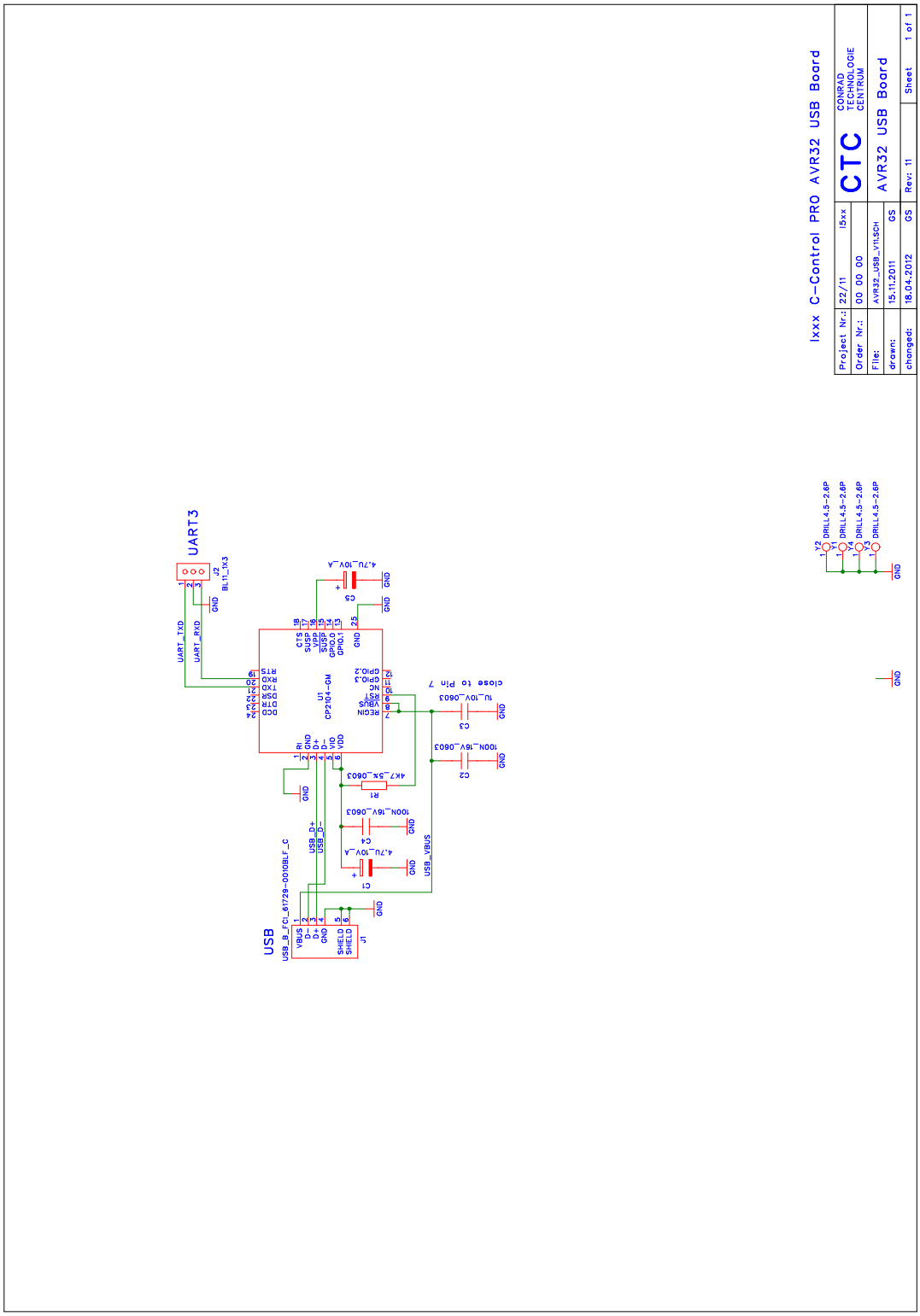


CAD

Technische Daten:

Abmessung	23mm x 25mm x 12mm (LxBxH)
UART PIN Raster	2.54 mm
USB	Type-B
Betriebsspannung	Versorgung über USB-Anschluss
UART Pegel	3.3V RxD/TxD
Gewicht	5g

2.2.12.1 Schaltplan



2.3 LCD Matrix

CHARACTER MODULE FONT TABLE (Standard font)

Character modules with built in controllers and Character Generator (CG) ROM & RAM will display 96 ASCII and special characters in a dot matrix format. Then first 16 locations are occupied by the character generator RAM. These locations can be loaded with the user designed symbols and then displayed along with the characters stored in the CG ROM.

CHARACTER FONT TABLE

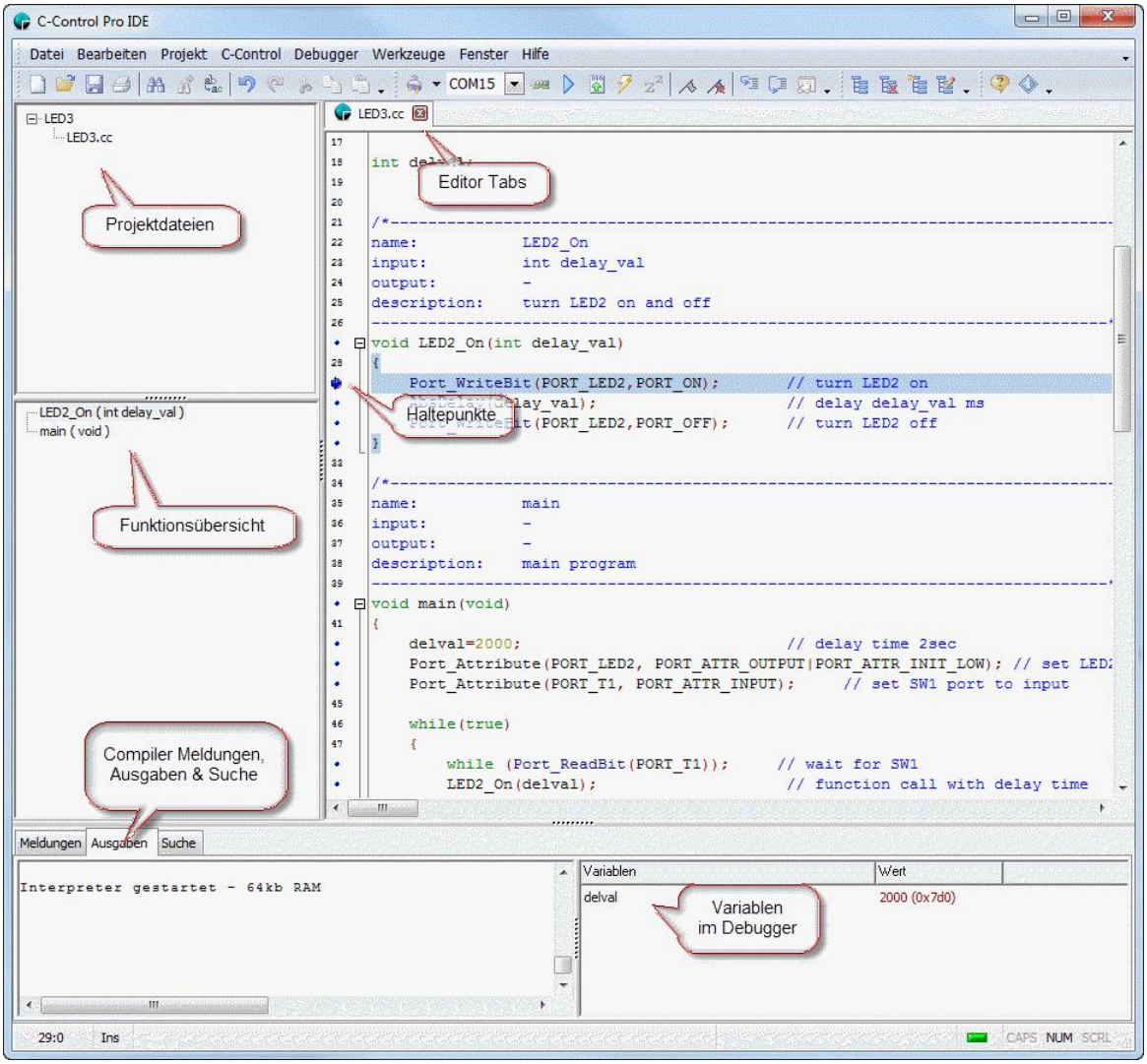
LOWER 4 BITS \ UPPER 4 BITS	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)												
0001	(2)												
0010	(3)												
0011	(4)												
0100	(5)												
0101	(6)												
0110	(7)												
0111	(8)												
1000	(1)												
1001	(2)												
1010	(3)												
1011	(4)												
1100	(5)												
1101	(6)												
1110	(7)												
1111	(8)												

Kapitel 3

3 IDE

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.



3.1 Projekte

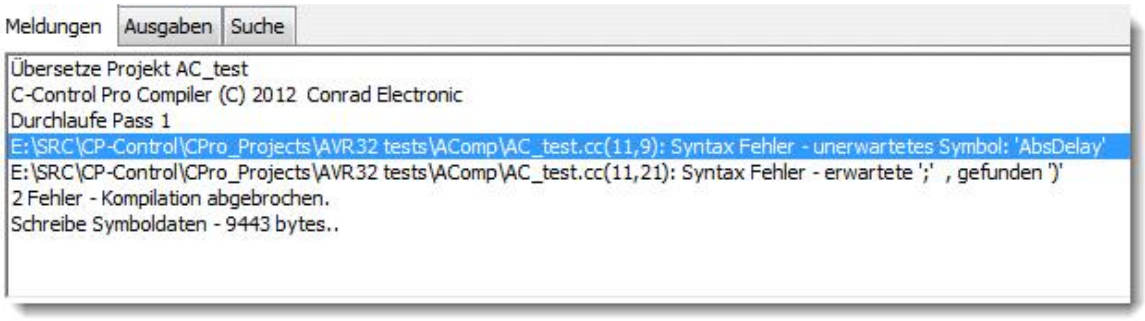
Jedes Programm für das C-Control Pro Modul wird durch ein Projekt konfiguriert. In einem Projekt steht, welche Quelldateien und Bibliotheken benutzt werden. Auch sind hier die Einstellungen des Compilers vermerkt. Ein Projekt besteht aus der Projektdatei mit der Endung ".cprj" und den dazugehörigen Quelldateien.

3.1.1 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. Die Quelltext Dateien in einem Projekt bestimmen welche Programmiersprache zum Einsatz kommt. Dateien mit der Endung *.cc laufen in einem CompactC Kontext, Dateien mit der Endung *.cbas werden mit BASIC übersetzt.

3.1.2 Projekte 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 bei der Fehlerstelle 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

3.1.3 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.

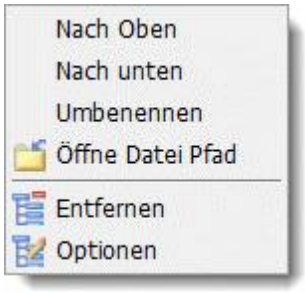
Hinzufügen von Projektdateien

Drückt man auf Hinzufügen von Projektdateien, so wird ein Datei Öffnen Dialog angezeigt, in dem man die Dateien anwählen kann, die dem Projekt hinzugefügt werden. Man kann mehrere Dateien auswählen.

Alternativ kann man mittels Drag&Drop Dateien aus dem Windows Explorer in die Projektverwaltung übernehmen.

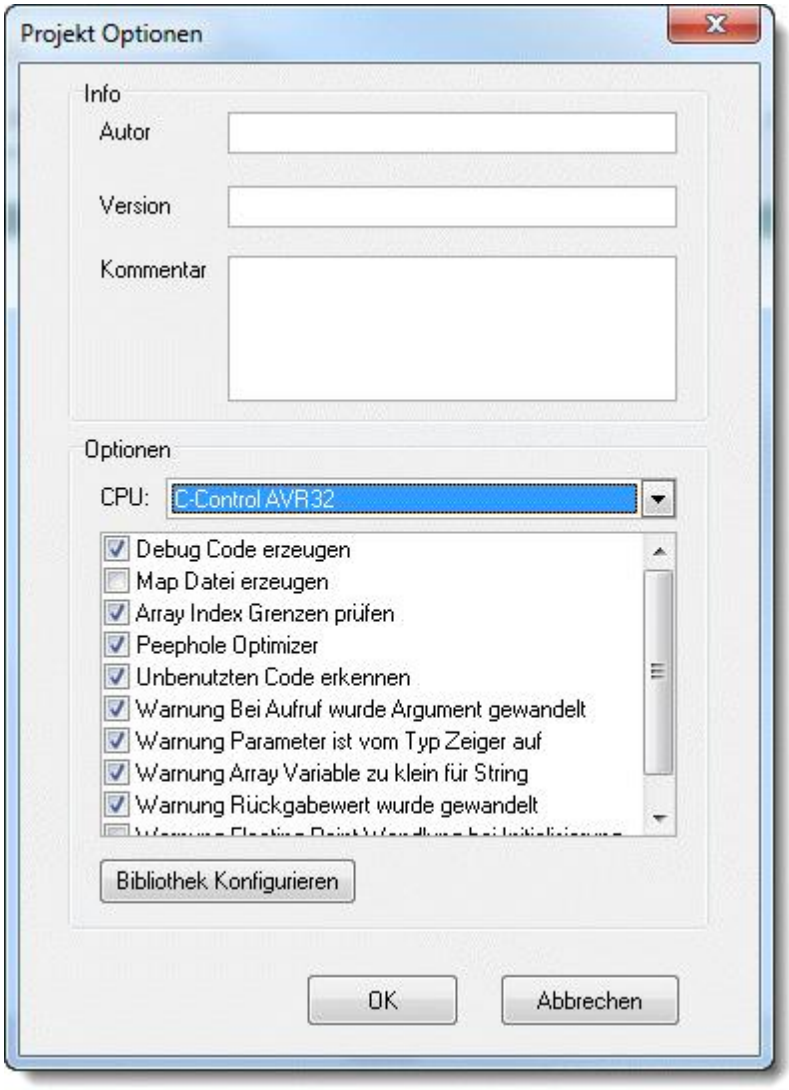
Projektdateien

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



- Nach Oben - Die Projektdatei wandert in der Liste nach oben (auch mit Strg - Pfeil rauf)
- Nach Unten - Die Projektdatei wandert unten (auch mit Strg - Pfeil runter)
- Umbenennen - Der Name der Projektdatei wird geändert
- Öffne Datei Pfad - Öffnet ein Explorer Fenster an der Stelle, an der die Datei steht
- Entfernen - Die Datei wird aus dem Projekt entfernt.
- Optionen - Die Projektoptionen können geändert werden.

3.1.4 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 Projektes erinnern zu können.

In "CPU" legt man die Zielplattform des Projekts fest.

Bibliothek Konfigurieren ruft die Bibliotheksverwaltung auf.

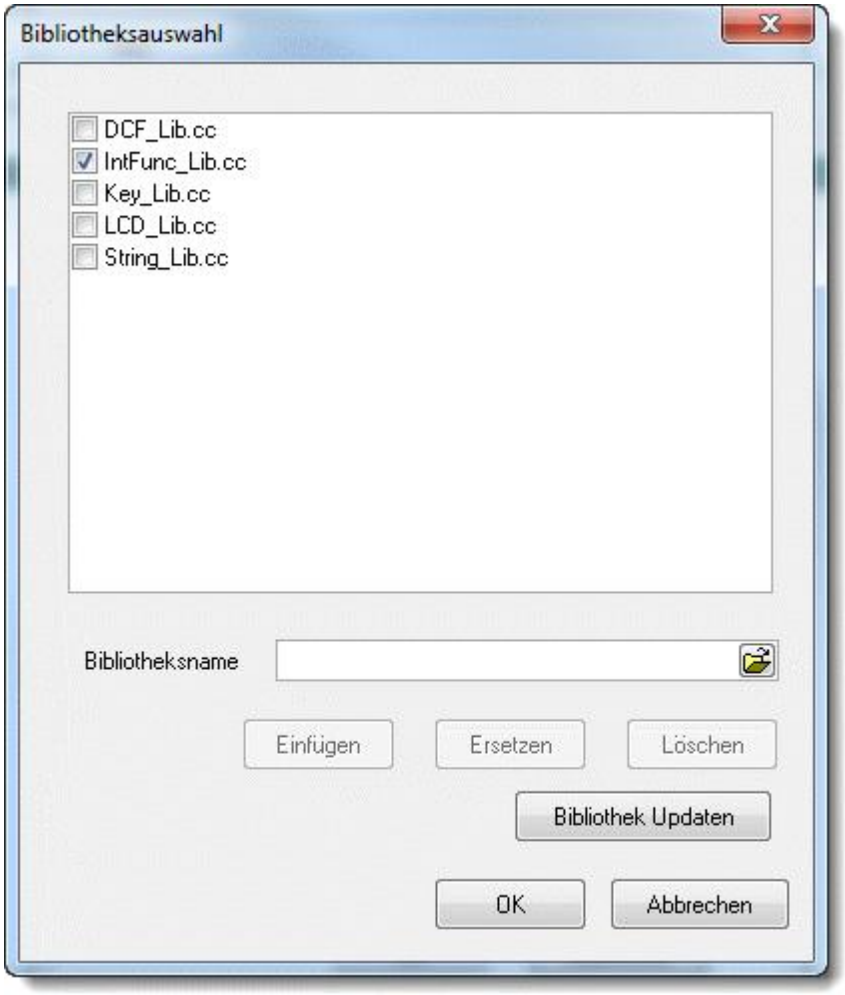
Optionen

Option	Bedeutung

Debug Code erzeugen	Erzeugt Debugcode. 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.
Map Datei erzeugen	Erzeugt eine Map Datei, in der Adressen und Länge von Variablen geschrieben werden.
Array Index Grenzen prüfen	Es wird Code eingefügt, der bei Array Zugriff den Index prüft. Nur bei Tests verwenden, da die Laufzeit erhöht wird.
Peephole Optimizer	Optimiert spezielle Codefolgen. Immer einschalten.
Unbenutzten Code erkennen	Nicht benutzter Code wird wegoptimiert. Immer einschalten.
Warnung bei Aufruf wurde Argument gewandelt	Der Typ einer Variable wurde bei einem Funktionsaufruf umgewandelt.
Warnung Parameter ist vom Typ Zeiger auf	Der Typ einer Zeigervariable (Array) ist von einem anderen Typ als die aufgerufene Funktion erwartet.
Warnung Array Variable zu klein für String	Der String passt nicht komplett in die zugewiesene Array Variable.
Warnung Rückgabewert wurde gewandelt	Der Rückgabewert is von einem anderen Typ als im Ausdruck erwartet.
Warnung Floating Point Wandlung bei Initialisierung	Der Floating Point Wert wird bei der Initialisierung in einen anderen Typ umgewandelt.

3.1.5 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:

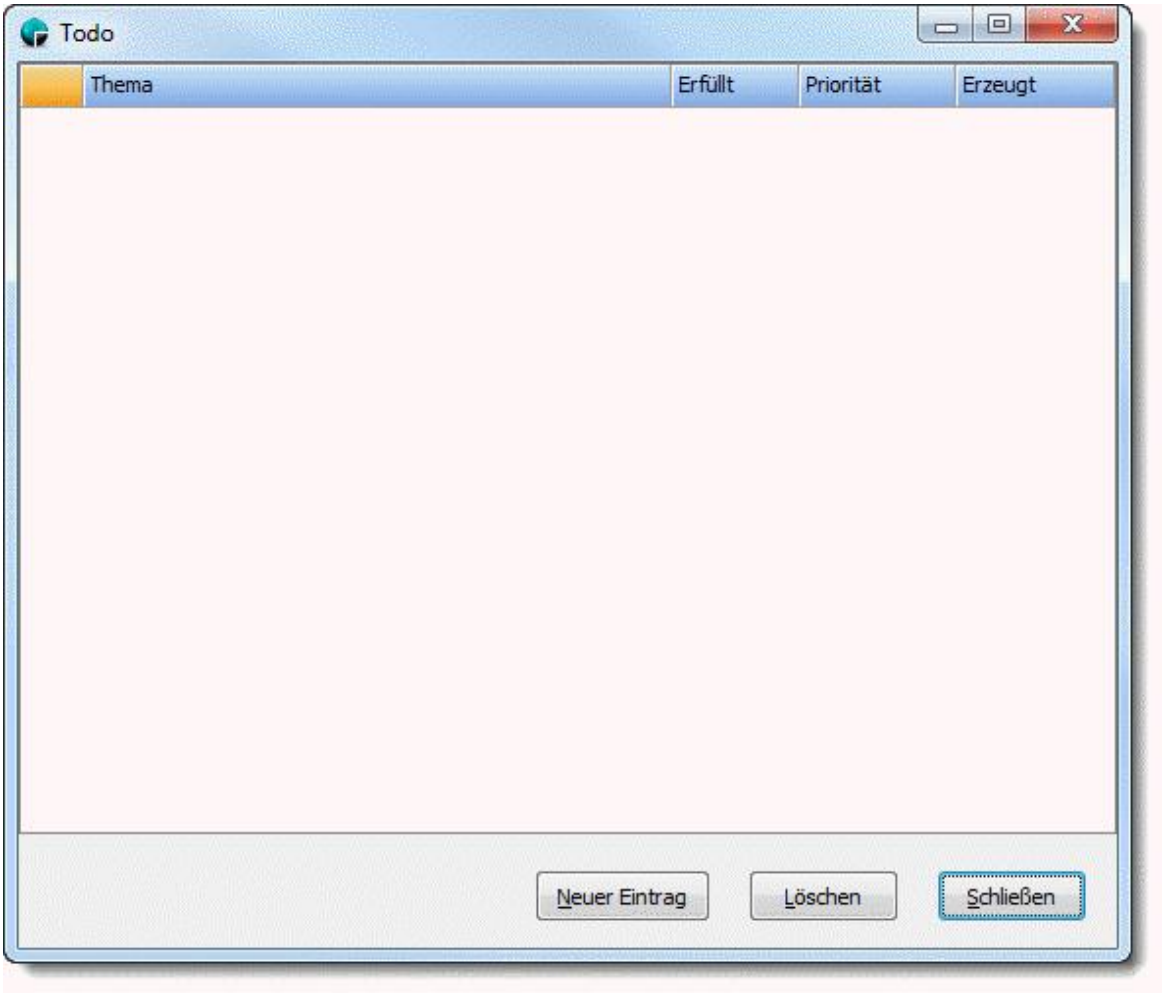
- Einfügen - Der Pfad wird zur Liste hinzugefügt.
- Ersetzen - Der selektierte Eintrag in der Liste wird durch den Pfadnamen ersetzt.
- Löschen - Der selektierte Listeneintrag wird gelöscht.
- Bibliothek Updaten - Dateien die in der Compilervoreinstellung vorhanden sind, aber in dieser Liste nicht, werden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

3.1.6 Threadoptionen

Seit Version 2.12 der IDE werden Threads nicht mehr in den Projektoptionen konfiguriert. Bitte die neue Syntax in Threads ansehen.

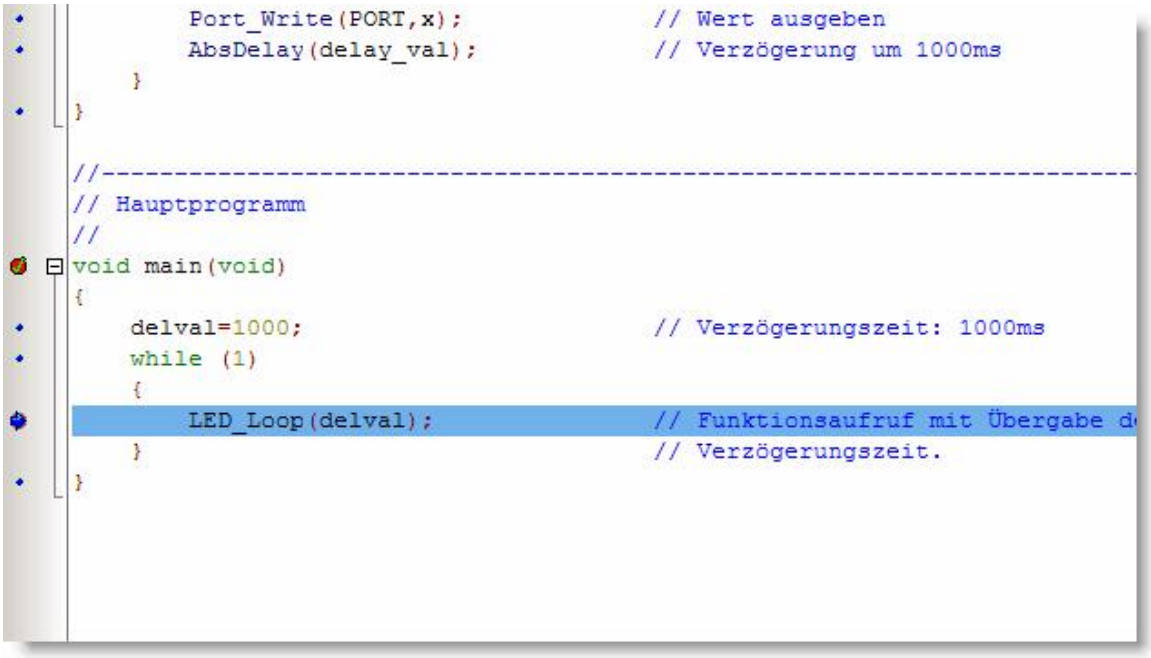
3.1.7 Todo Liste

Im Projekt Menü kann eine einfache Todo Liste aufgerufen werden. Der Inhalt wird zusammen mit dem Projekt gespeichert.



3.2 Editor

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.



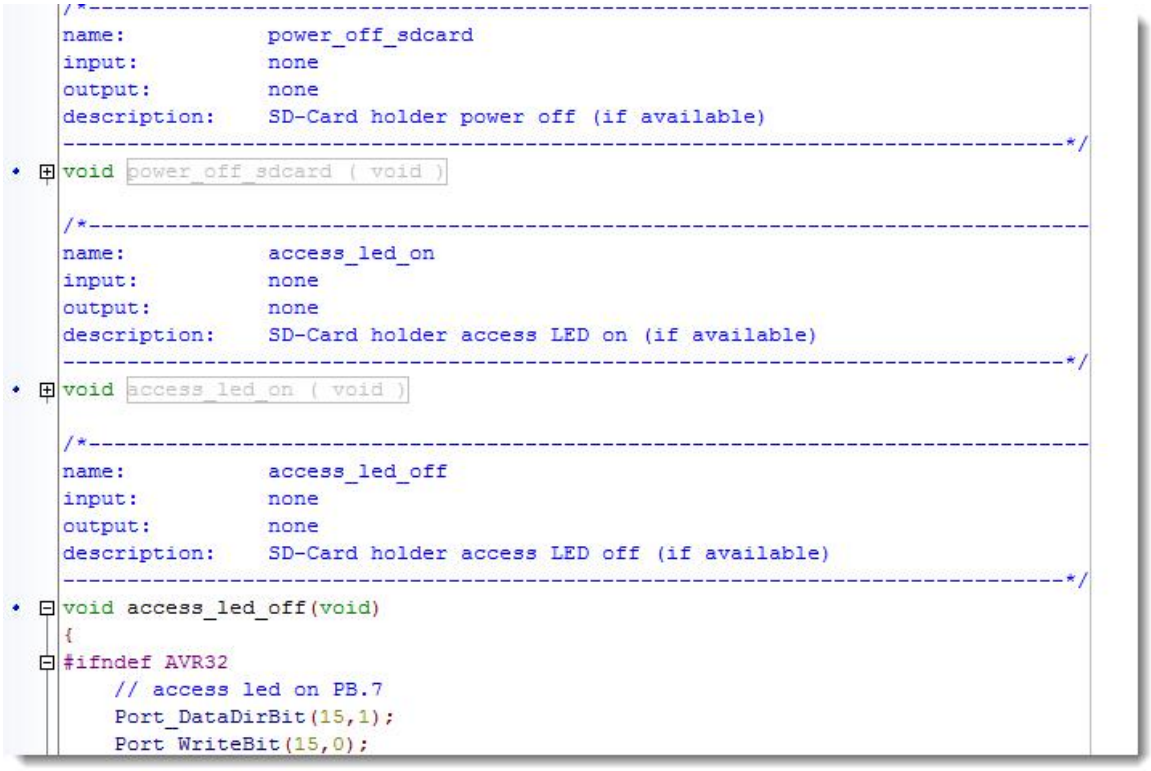
Ein Klick auf den Bereich links neben den Textanfang setzt dort einen Haltepunkt (Breakpoint). Dazu muss vorher der Quelltext fehlerfrei mit "*Debug Info*" kompiliert worden sein, und in der entsprechenden Zeile tatsächlich ausführbarer Programmtext stehen (z.B. keine Kommentarzeile o. ä.).

Funktionsübersicht

Auf der linken Seite findet man eine Übersicht aller definierten syntaktisch korrekten Funktionen. Die Funktionsnamen samt Parameter werden dort dargestellt. Die Funktion in der sich der Cursor befindet, wird in der Funktionsliste grau hinterlegt dargestellt. Man kann mit einem Doppelklick auf den Funktionsnamen auch direkt an den Anfang der Funktion im Editor springen.

Code-Falten

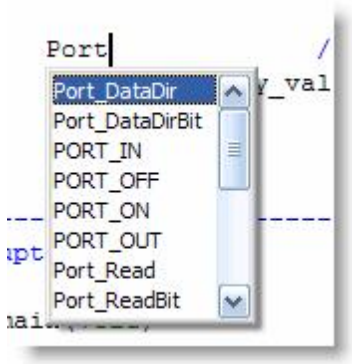
Um im Editor eine bessere Übersicht zu behalten, kann der Quelltext gefaltet werden. Sobald der im Editor enthaltene syntaktische Analyzer eine definierte Funktion erkennt, werden auf der linken Seite der Bereich der Funktion durch Balken signalisiert. Ein Klick auf das Minuszeichen im quadratischen Kästchen, und von der Funktion ist nur noch die erste Zeile zu sehen. Ein weiterer Klick auf das Pluszeichen, und der Quelltext entfaltet sich wieder.



Um alle Funktionen in einer Datei zu falten, oder zu entfalten, kann man mit Rechtsklick im Editor im Menü Zeilen falten oder Zeilen entfalten auswählen.

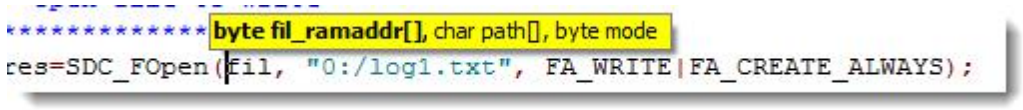
Syntaktische Eingabehilfe

Der Editor hat eine syntaktische Eingabehilfe erhalten. Tippt man den Anfang eines Befehlswortes, oder einer Funktion aus der Standardbibliothek so kann man mit Ctrl-Space (Strg-Leerzeichen) die Eingabehilfe aktivieren. Abhängig von den bereits getippten Buchstaben öffnet sich eine Auswahlbox, und man kann das gesuchte Wort in den Quelltext einfügen. Nach einer erfolgreichen Kompilierung werden auch selbst definierte Funktionen und Variablen des Programms in der Eingabehilfe angezeigt.



Parameter Eingabehilfe

Nach einer erfolgreichen Kompilierung werden auch die Parameter einer Funktion analysiert. Tippt man einen bekannten Funktionsnamen und Klammer auf "(", so wird in gelb die erwarteten Typen der Funktionsparameter angezeigt.

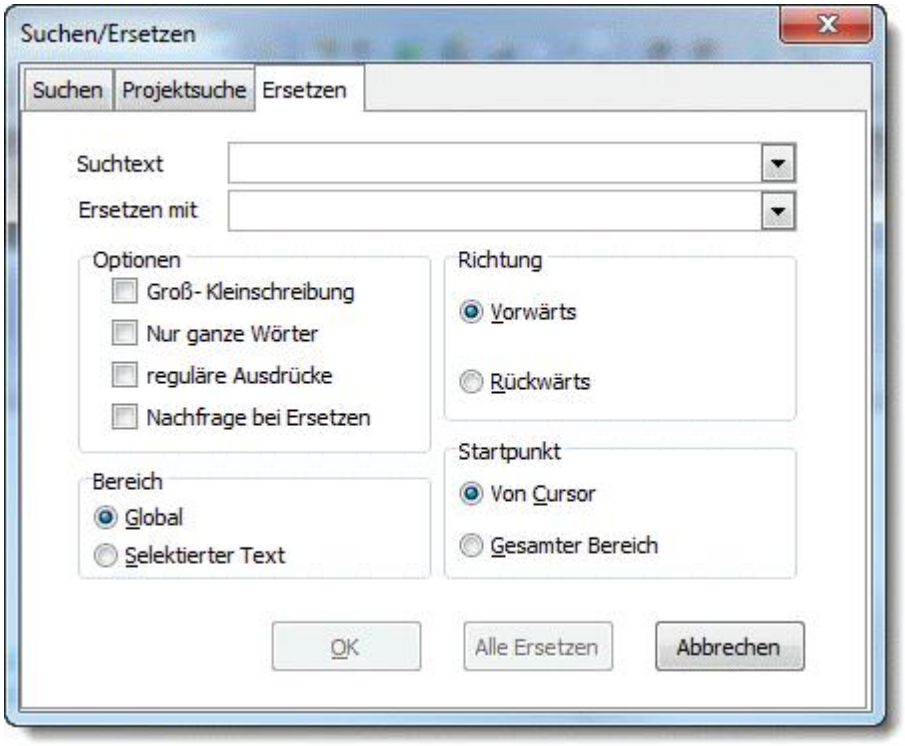


3.2.1 Editorfunktionen

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 Ersetzen - vor dem Ersetzen wird beim Benutzer nachgefragt.

Weiter kann die Suchrichtung (Vorwärts , Rückwärts) vorbestimmt werden, ob der gesamte Text (Global) , oder nur ein selektierter Bereich (Selektierter Text) durchsucht wird. Auch wird eingestellt, ob die Suche am Cursor beginnt (Von Cursor) oder am Textanfang (Gesamter Bereich).

Projektsuche

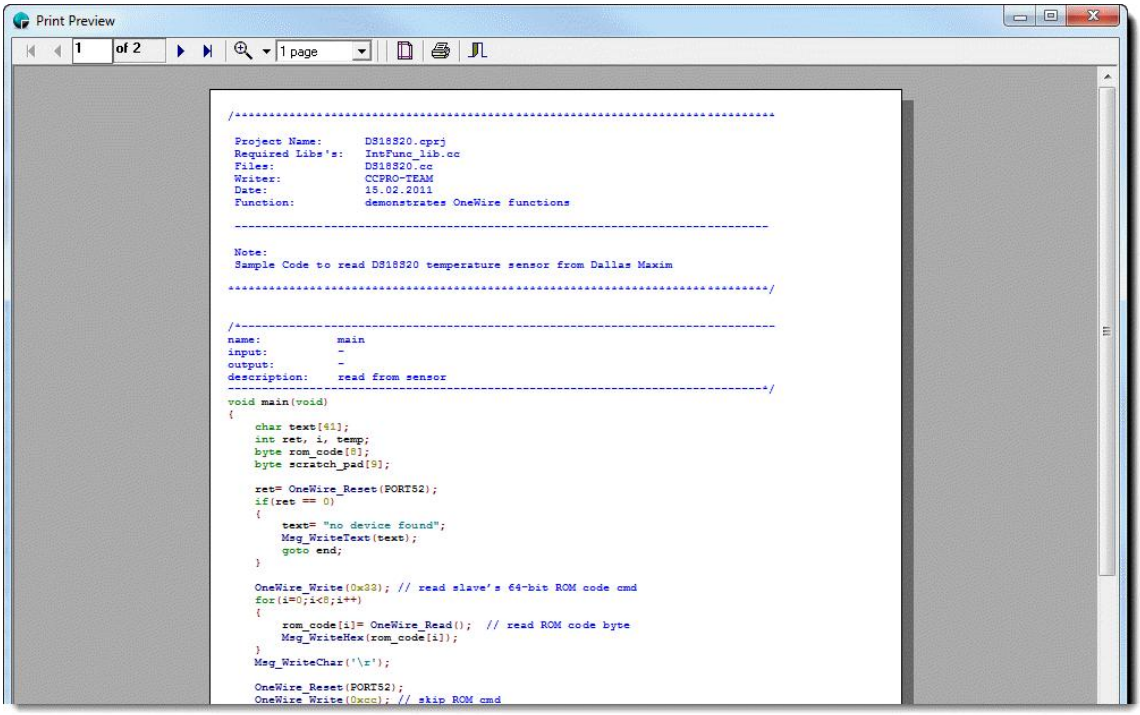
In der Projektsuche wird ein Text in mehr als nur einer Datei gesucht.

- Alle Projektdateien - Sucht den Text in allen gespeicherten Projektdateien, auch wenn sie nicht im Editor geöffnet sind.
- Alle offenen Dateien - Durchsucht alle im Editor offenen Dateien. Es werden darin auch nicht gespeicherte Änderungen berücksichtigt.

3.2.2 Druckvorschau

Um den Source Code von Projekten anderen in Papierform zu übergeben, oder aber auch zu archivieren, sind Druckfunktionen in die C-Control Pro IDE eingebaut worden. Folgende Optionen kann man aus dem Datei Pull-Down Menü aussuchen:

Drucken: Druckt die angegeben Seiten
Druckvorschau: Zeigt eine Vorschau auf den Druck
Druckereinstellung: Wahl des Druckers, Papiergröße und Orientierung
Seiteneinstellung: Es lassen sich Kopf- und Fußzeilen, Zeilennummern sowie weitere Druckparameter einstellen



3.2.3 Tastaturkürzel

Taste	Funktion
Left	Move cursor left one char
Right	Move cursor right one char
Up	Move cursor up one line
Down	Move cursor down one line
Ctrl + Left	Move cursor left one word
Ctrl + Right	Move cursor right one word
PgUp	Move cursor up one page
PgDn	Move cursor down one page
Ctrl + PgUp	Move cursor to top of page
Ctrl + PgDn	Move cursor to bottom of page
Ctrl + Home	Move cursor to absolute beginning
Ctrl + End	Move cursor to absolute end
Home	Move cursor to first char of line

End	Move cursor to last char of line
Shift + Left	Move cursor and select left one char
Shift + Right	Move cursor and select right one char
Shift + Up	Move cursor and select up one line
Shift + Down	Move cursor and select down one line
Shift + Ctrl + Left	Move cursor and select left one word
Shift + Ctrl + Right	Move cursor and select right one word
Shift + PgUp	Move cursor and select up one page
Shift + PgDn	Move cursor and select down one page
Shift + Ctrl + PgUp	Move cursor and select to top of page
Shift + Ctrl + PgDn	Move cursor and select to bottom of page
Shift + Ctrl + Home	Move cursor and select to absolute beginning
Shift + Ctrl + End	Move cursor and select to absolute end
Shift + Home	Move cursor and select to first char of line
Shift + End	Move cursor and select left and up at line start
Alt + Shift + Left	Move cursor and column select left one char
Alt + Shift + Right	Move cursor and column select right one char
Alt + Shift + Up	Move cursor and column select up one line
Alt + Shift + Down	Move cursor and column select down one line
Alt + Shift + Ctrl + Left	Move cursor and column select left one word
Alt + Shift + Ctrl + Right	Move cursor and column select right one word
Alt + Shift + PgUp	Move cursor and column select up one page
Alt + Shift + PgDn	Move cursor and column select down one page
Alt + Shift + Ctrl + PgUp	Move cursor and column select to top of page
Alt + Shift + Ctrl + Alt + PgDn	Move cursor and column select to bottom of page
Alt + Shift + Ctrl + Home	Move cursor and column select to absolute beginning
Alt + Shift + Ctrl + End	Move cursor and column select to absolute end
Alt + Shift + Home	Move cursor and column select to first char of line
Alt + Shift + End	Move cursor and column select to last char of line
Ctrl + C; Ctrl + Ins	Copy selection to clipboard
Ctrl + X	Cut selection to clipboard
Ctrl + V; Shift + Ins	Paste clipboard to current position
Ctrl + Z; Alt + Backspace	Perform undo if available
Shift +Ctrl + Z	Perform redo if available
Ctrl + A	Select entire contents of editor
Ctrl + Del	Clear current selection
Ctrl + Up	Scroll up one line leaving cursor position unchanged
Ctrl + Down	Scroll down one line leaving cursor position unchanged
Backspace	Delete last char
Del	Delete char at cursor
Ctrl + T	Delete from cursor to next word
Ctrl + Backspace	Delete from cursor to start of word
Ctrl + B	Delete from cursor to beginning of line
Ctrl + Y	Delete current line
Enter	Break line at current position, move caret to new line
Ctrl + N	Break line at current position, leave caret
Tab	Tab key
Tab (block selected)	Indent selection
Shift + Tab	Unindent selection
Ctrl + K + N	Upper case to current selection or current char
Ctrl + K + O	Lower case to current selection or current char
Ins	Toggle insert/overwrite mode
Ctrl + O + K	Normal selection mode

Ctrl + O + C	Column selection mode
Ctrl + K + B	Marks the beginning of a block
Ctrl + K + K	Marks the end of a block
Esc	Reset selection
Ctrl + digit (0-9)	Go to Bookmark digit (0-9)
Shift + Ctrl + (0-9)	Set Bookmark digit (0-9)
Ctrl + Space	Auto completion popup

3.2.4 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

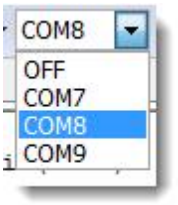
Beispiel	findet
^void	das Wort "void" nur am Zeilenanfang
;$	das Semikolon nur am Zeilenende
^void$	in der Zeile darf nur "void" stehen
vo.*d	z.B. "vod","void","vqqd"
vo.+d	z.B. "void","vqqd" aber nicht "vod"
[qs]	die Buchstaben 'q' oder 's'
[qs]port	"qport" oder "sport"
[^qs]	alle Buchstaben außer 'q'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

3.3 C-Control Hardware

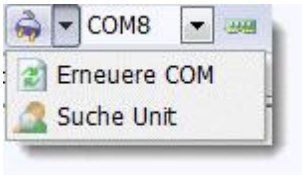
Unter dem Menüpunkt C-Control können die Hardware relevanten Funktionen ausgeführt werden. Dies beinhaltet Übertragen und Starten des Programms auf der Hardware, sowie Passwortfunktionen.

3.3.1 Interface selektieren

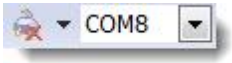
In der Toolbar lässt sich in einem Drop-down Menü direkt das Interface auswählen, mit der das C-Control Pro Modul angesprochen werden kann.



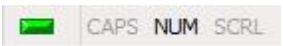
Dabei werden alle Interfaces mit **COM** bezeichnet, egal ob es wirklich um eine serielle Schnittstelle handelt, oder um einen über USB angeschlossenen Virtual Comport.



Mit Suche Unit wird nach einem C-Control Pro Modul gesucht. Die Funktion Erneuere COM sucht nach Änderungen der angeschlossenen COM Schnittstellen. Wenn z.B. ein USB-Kabel angeschlossen wurde, und eine neuer COM Port verfügbar ist.



Ein Klick auf das Steckersymbol schaltet den COM Port aus (rotes Kreuz), ein erneuter Klick wieder ein.



Die grüne "LED" Anzeige signalisiert, das der COM Port offen ist. Ist die Anzeige rot, so ist der COM Port geschlossen.

➔ Ein offener COM Port (grün) bedeutet nicht unbedingt, das eine C-Control angeschlossen ist, es könnte z.B. auch ein anderes Gerät wie USB-seriell Konverter sein. Erst eine Unit Suche überprüft, ob dort ein C-Control Modul ist.

3.3.2 Programm starten

Programmübertragung

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

➔ Nach einem Update der IDE, wird bei Bedarf, nicht nur der Bytecode zum Modul übertragen, sondern gleichzeitig wird die neueste Version des Interpreters mit zum C-Control Modul geschickt.

Programmübertragung ohne Source

Möchte man ein Programm übertragen, von dem man nur den Bytecode (.bc) hat und keinen Sourcecode, dann kann man den Bytecode mit Übertragen Datei auf das C-Control Pro Modul laden.

Starten

Durch Starten (F10) wird dann die Ausführung des Bytecode auf dem Modul veranlasst. Dies wird auf einem Mega Applicationboard durch ein Leuchten der roten LED signalisiert.

Stoppen

Im normalen Betrieb wird ein Programm durch Drücken auf den Taster RESET1 (Mega) oder dem Start/Stop Taster (AVR32Bit) 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.

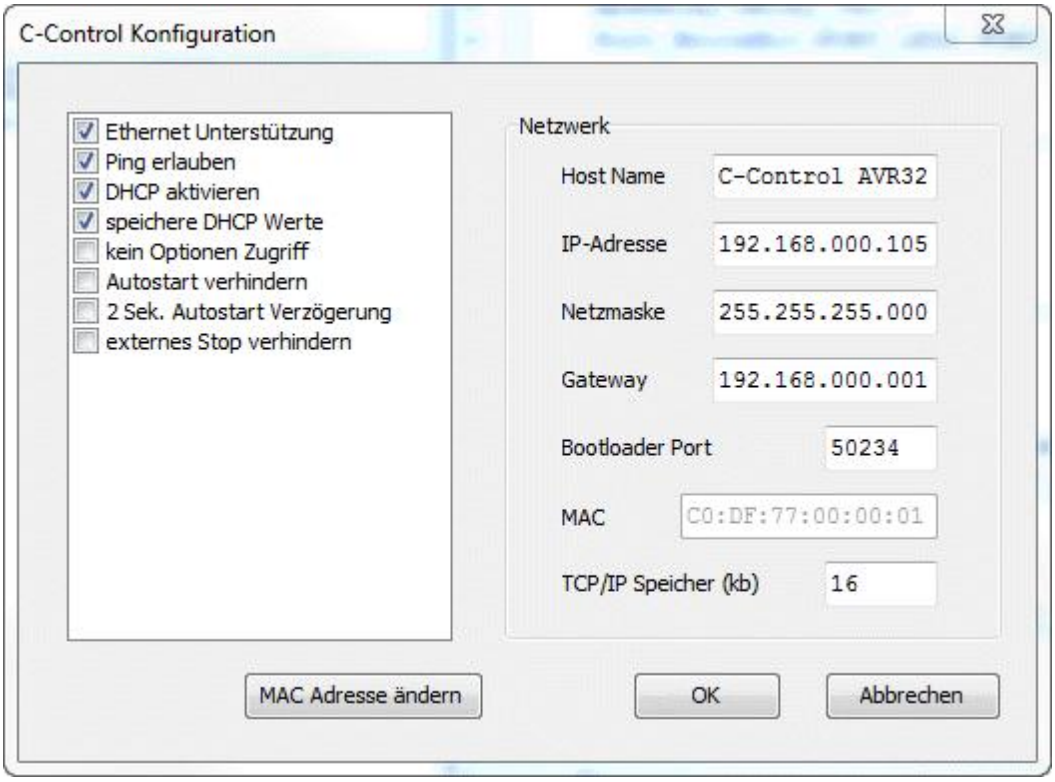
➔ In seltenen Fällen kann sich im USB Betrieb des C-Control Pro Mega Applicationboards beim Druck auf den Taster RESET1 das System verklemmen. Bitte dann den Taster RESET2 betätigen, um auch dem Mega8 einen Reset Impuls zu geben. Der Mega8 kümmert sich auf dem Application Board um die USB Schnittstelle.

Autostart

Wird das Modul in eine Hardware Applikation eingebaut, so ist es oft gewollt, das das Anwenderprogramm automatisch gestartet wird. Siehe hierzu Autostart (Mega) und Autostart (AVR32Bit).

3.3.3 C-Control konfigurieren

Die Funktion C-Control Konfiguration erlaubt es die Hardware Einstellungen der C-Control Pro AVR32Bit zu ändern. Hier kann man keine Einstellungen der C-Control Pro Mega Module kontrollieren.



Es können die gängigen Netzwerkeinstellungen eingegeben werden, der UDP-Port des Bootloaders und die MAC Adresse.

➔ Um Verbindungsprobleme zu vermeiden sollte vor dem Einschalten der Ethernet Unterstützung die MAC-Adresse auf einen neuen Wert eingestellt werden. Zu diesem Zweck wird für jede C-Control Pro AVR32Bit eine eigene MAC-Adresse erzeugt und auf einem Aufkleber mitgeliefert. Siehe Software Installation.

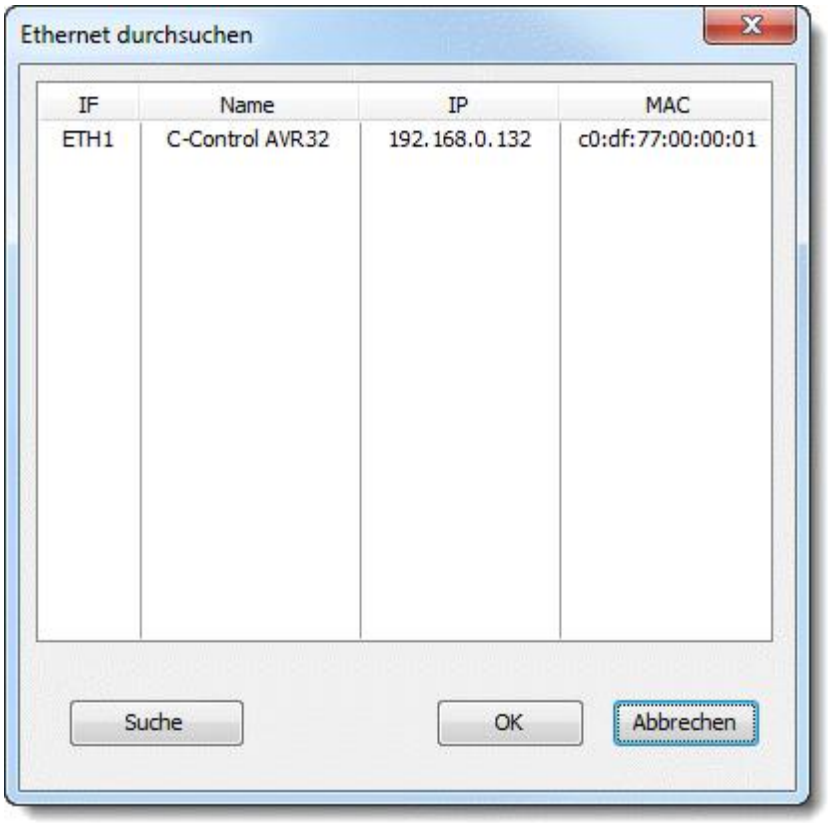
Optionen

Ethernet Unterstützung - Schaltet Ethernet Support ein.
Ping erlauben - ICMP Echo wird beantwortet.
kein Optionen Zugriff - Die C-Control Konfiguration kann nicht mehr geändert werden. Erst Modul zurücksetzen erlaubt dies wieder.
Autostart verhindern - Es wird kein Autostart ausgeführt (siehe Autostart).
DHCP aktivieren - Holt die Netzwerkdaten von einem DHCP Server.
speichere DHCP Werte - Geänderte DHCP Daten werden gespeichert.
externes Stop verhindern - Ein Programm kann nicht über Software gestoppt werden.
2 Sek. Autostart Verzögerung - Autostart wird um 2 Sekunden verzögert damit USB hochgefahren ist

3.3.4 Ethernet durchsuchen

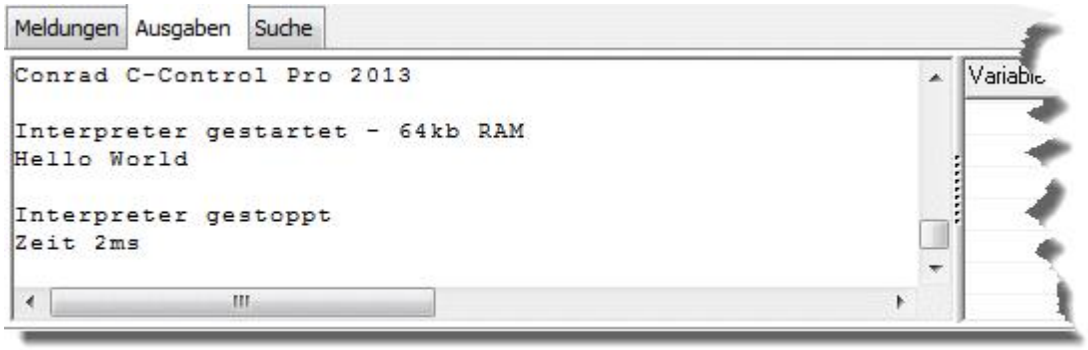
Wenn Ethernet Unterstützung aktiviert ist, dann ist das C-Control Pro AVR32Bit Modul auf dem lokalen Ethernet LAN sichtbar. Da die Suche per UDP-Broadcast durchgeführt wird, ist sie auf das lokale Subnetz beschränkt, da Router meist keine Broadcasts weiterleiten. Der Default UDP Port für den Ethernet Zugriff ist 50234. Dies darf durch eine lokale Firewall auf dem PC nicht eingeschränkt werden.

➔ Derzeit ist die Ethernet Unterstützung auf das Programm Update beschränkt.



3.3.5 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

3.3.6 PIN Funktionen

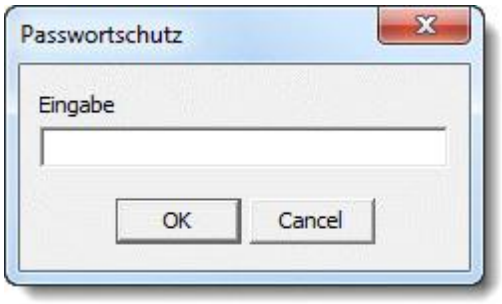
Einzelne Funktionen des Interpreters lassen sich mit einer alphanumerischen PIN schützen. Ist ein Interpreter durch eine PIN gesichert, so sind normale Operationen verboten. Er kann durch eine erneute Übertragung überschrieben werden, aber die PIN bleibt erhalten. Auch ein normales Starten ist nicht 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 Menü kann man alle Operationen freischalten.

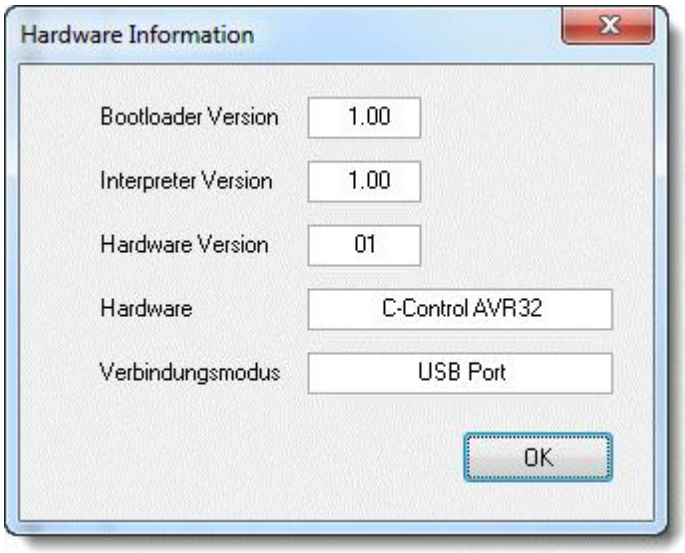
Um eine neue PIN einzutragen, oder eine gesetzte PIN zu löschen, existieren die Befehle PIN Setzen und PIN Löschen im C-Control Menü. War schon eine PIN gesetzt, so muss das Modul natürlich erst durch Eingabe der alten PIN entsperrt werden. Eine PIN darf bis zu 6 alphanumerische Zeichen lang sein.

➔ 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.



3.3.7 Versionsüberprüfung

Da die C-Control Pro Mega Serie mehrere Hardware Plattformen zu unterstützt, 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.



3.4 Debugger

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

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 blaue 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 Codezeile aus und bleiben dann stehen. Im Unterschied zu Einzelschritt springt Prozedurschritt nicht in Funktionsaufrufe, sondern geht über sie hinweg. Während das Programm hält, können die Breakpoints verändert werden.

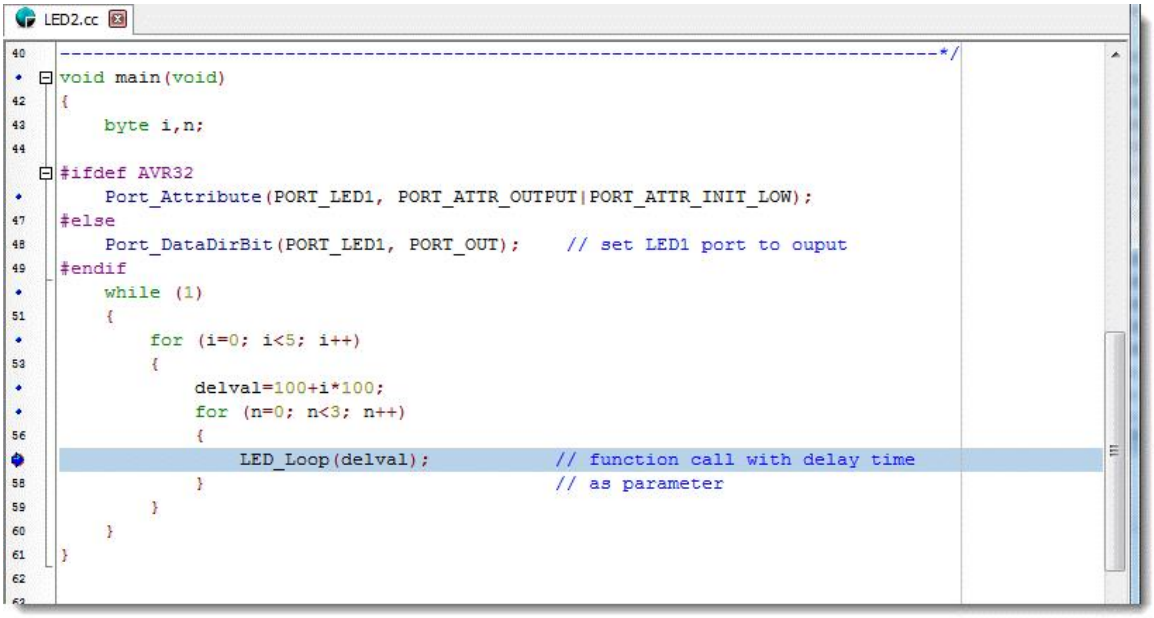
➔ Ist in einer Schleife nur eine Codezeile, so führt ein Einzelschritt die ganze Schleife aus, da erst dann zu einer neuen Codezeile verzweigt wird.

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 wo Breakpoints gesetzt wurden, nicht verschieben können. Der Debugger wäre sonst nicht in der Lage, sich mit dem Bytecode auf dem C-Control Modul zu synchronisieren.

3.4.1 Haltepunkte

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 IDE oder Editorfenster).

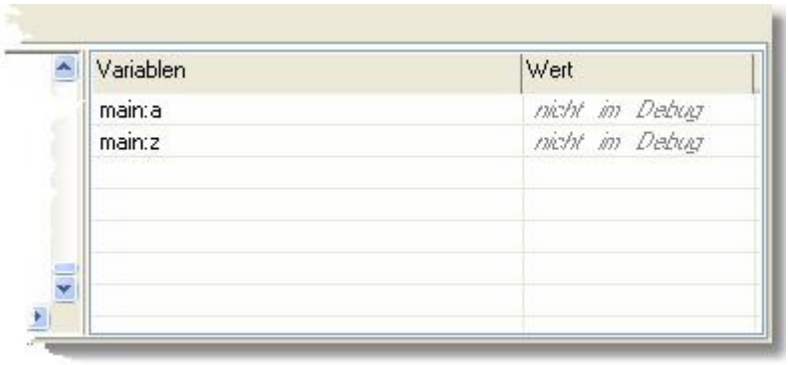


➔ Die Anzahl der Breakpoints ist auf 16 begrenzt, weil diese Information beim Lauf des Bytecode Interpreters im RAM mitgeführt wird. Andere Debugger setzen Haltepunkte direkt in den Programmcode. Dies ist hier nicht erwünscht, da es die Lebenszeit des Flashspeichers (ca. 10000 Schreibzugriffe) drastisch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

3.4.2 Variablen Fenster

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. Die Variable wird zuerst gemäß ihrem Datentyp dargestellt, und dann durch Komma getrennt, als Hexzahl mit einem vorangestellten "0x".

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

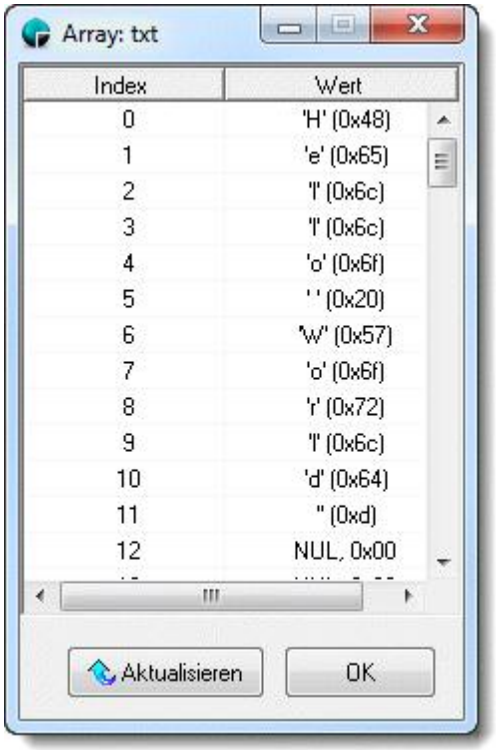
Sind viele Variablen in die Überwachungsliste eingetragen, so kann es bei einem Einzelschritt lange dauern, bis alle Variableninhalte aus dem Modul abgefragt worden sind. Zu diesem Zweck läßt sich die Option Auto Aktualisieren für einzelne Variablen ausschalten. Dann werden die Inhalte dieser Variablen 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.

➔ Variablen vom Typ character werden als einzelnes ASCII Zeichen dargestellt.

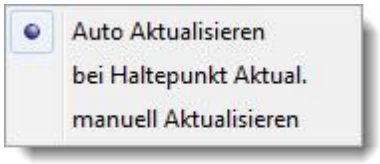
3.4.3 Array Fenster

Um die Inhalte von Array Variablen zu betrachten ist es möglich ein Fenster mit dem Inhalt des Arrays aufzurufen. Dafür wird der Cursor auf der Variablen positioniert, und mit Rechtsklick Array anzeigen angewählt.

Auf der linken Seite werden die Indizes des Arrays angezeigt, auf der rechten Seite der Inhalt. Man beachte, daß bei multidimensionalen Arrays die Indizes auf der rechten Seite am schnellsten wachsen.



Der Inhalt eines Array Fensters kann bei jedem Halt des Debuggers, oder bei einem Einzelschritt nicht mehr aktuell sein. Wird bei jedem Einzelschritt im Debugger mehrere Array Fenster neu aktualisiert, so können Verzögerungen auftreten, da die Daten immer vom Modul geladen werden müssen. Es gibt daher drei Arbeitsmodi:

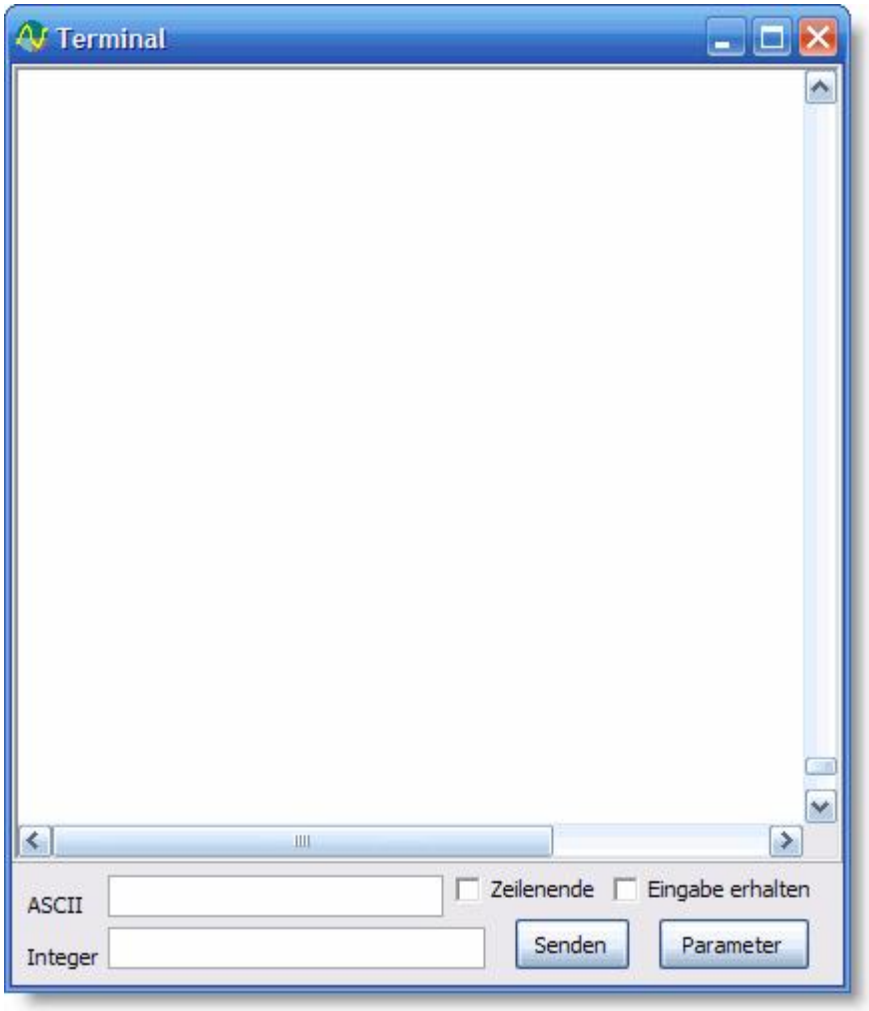


Auto Aktualisieren	Aktualisierung bei Einzelschritt und Haltepunkt
bei Haltepunkt Aktualisieren	Aktualisierung nur bei Haltepunkt (Breakpoint)
manuell Aktualisieren	nur bei Klick auf den Schalter "Aktualisieren"

3.5 Werkzeuge

Unter dem Menüpunkt Werkzeuge kann man das einfache integrierte Terminalprogramm starten, eigene Programme hinzufügen und die IDE Optionen ändern.

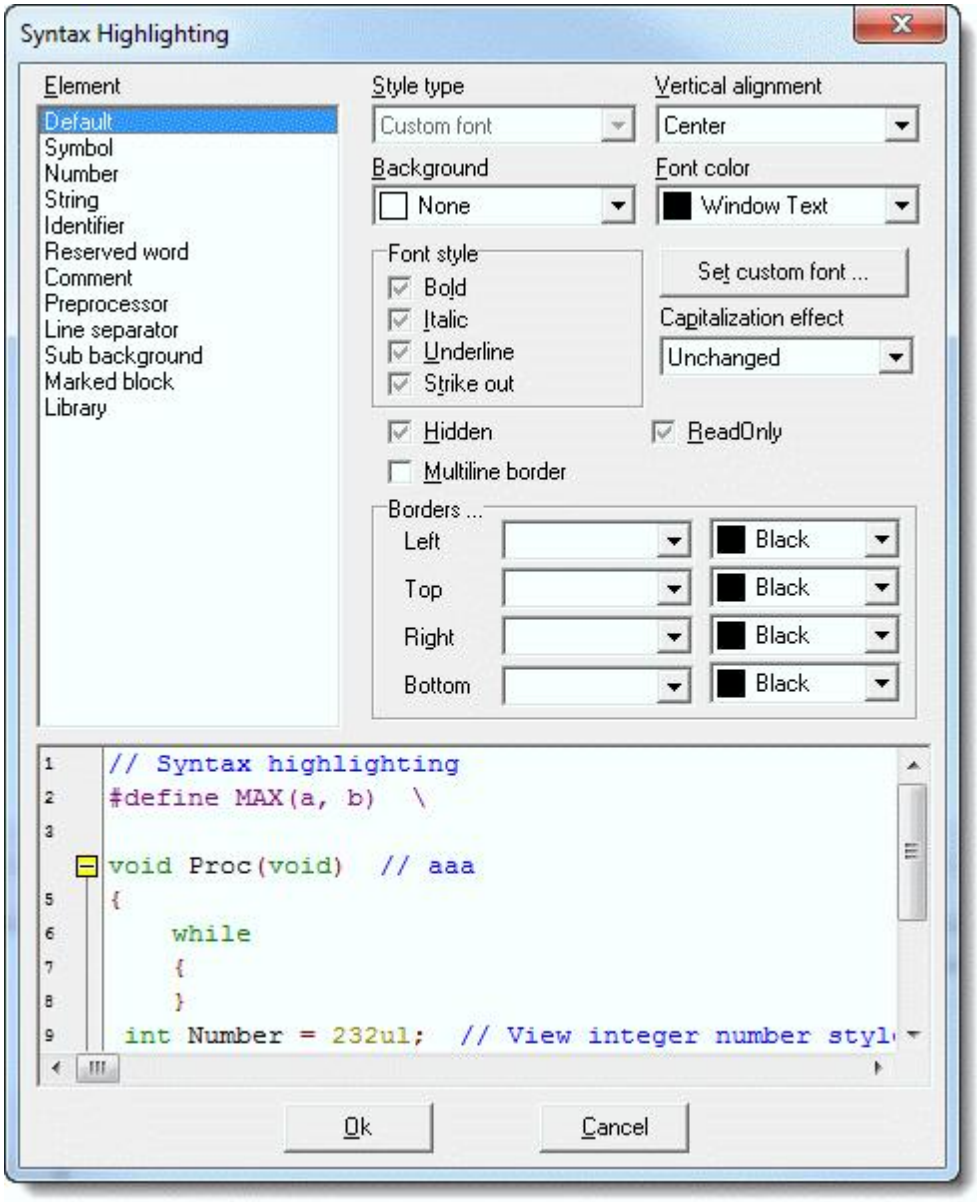
Terminal Fenster



Empfangene Zeichen werden direkt im Terminalfenster dargestellt. Man kann Zeichen auf zwei verschiedene Arten senden. Zum einen kann man in das Fenster klicken und die Zeichen direkt über die Tastatur eingeben, oder man schreibt die Zeichen in die ASCII Eingabezeile und drückt den Senden Knopf. Die ASCII Zeichen können auch als Integerzahlen in die Integer Eingabezeile. Selektiert man Zeilenende, wird immer ein Carriage Return (13) mitgeschickt. Mit der Option Eingabe erhalten bewirkt man, das nach dem Senden der Text in den Eingabezeilen nicht gelöscht wird. Der Knopf Parameter öffnet den Terminal Konfigurationsdialog in den IDE Einstellungen.

3.5.1 Syntaxhervorhebung

In diesem Dialog kann die spezifische Syntaxeinfärbung von CompactC und BASIC verändert werden. Es wird die Einstellung von CompactC oder BASIC geändert, abhängig davon welche Sprache gerade in dem selektierten Editor Fenster aktuell war.



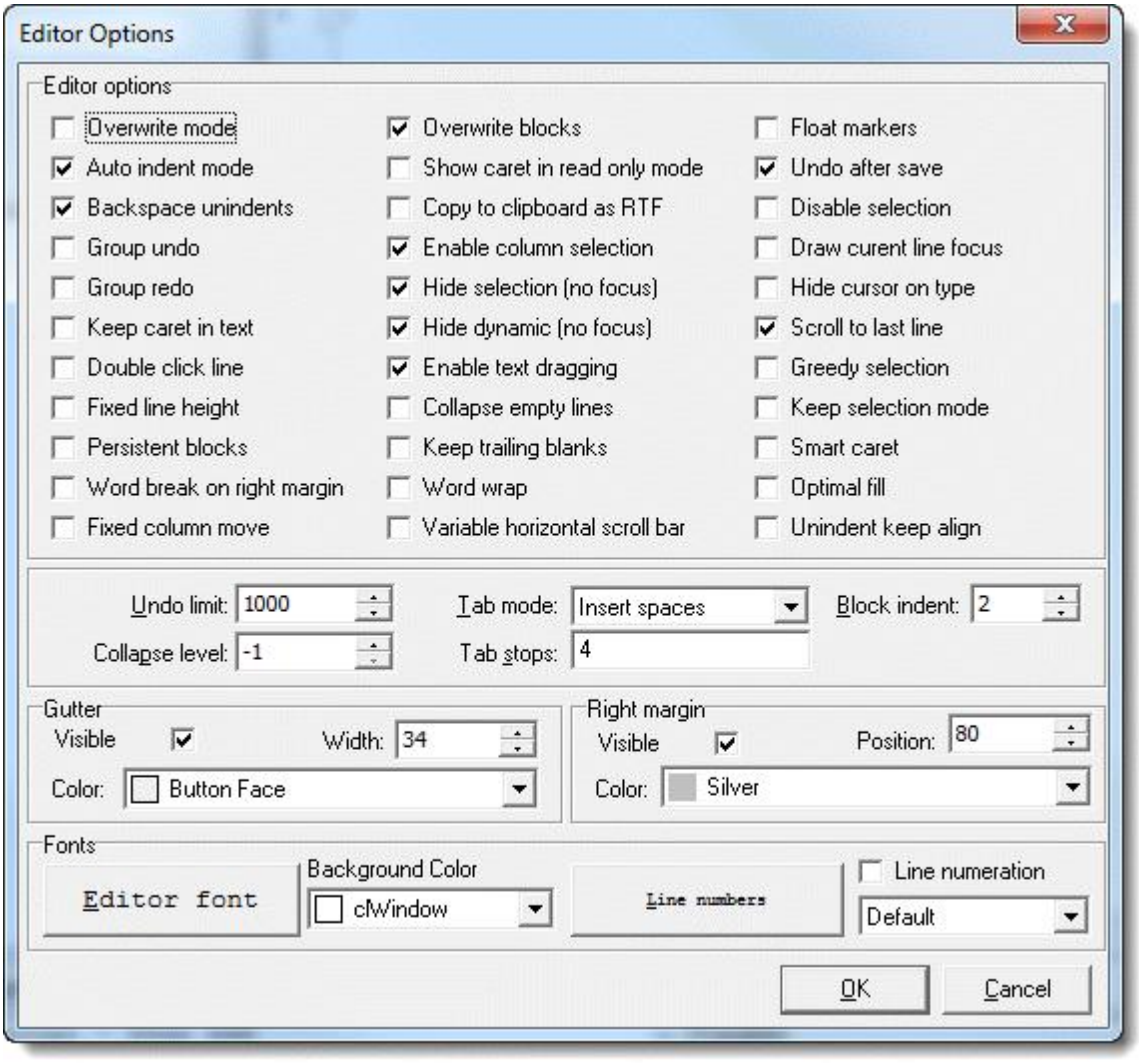
In dem Dialog können die Fontattribute der Schrift, die Vordergrund und Hintergrundfarbe geändert werden. Mit Multiline border kann ein Rahmen um die entsprechenden Zeichen oder Wörter gesetzt werden. Auch ist es mit Capitalization Effect möglich auf die Groß-Kleinschreibung Einfluß zu nehmen. Die einstellbaren Elemente haben folgende Bedeutung:

Symbol:	Alle nicht alpha-numerischen Zeichen
Number:	alle numerischen Zeichen
String:	Zeichen die von der Zielsprache als Zeichenkette interpretiert werden ("String")
Identifier:	Alle Namen die nicht Befehlswörter sind oder aus der Standardbibliothek stammen
Reserved Word:	Befehlswörter der Zielsprache
Comment:	Kommentare
Preprocessor:	Preprozessor Anweisungen
Marked Block:	markierte Blöcke
Library:	Funktionsnamen der Standardbibliothek

Default, Line separator and Sub background werden nicht benutzt.

➔ Aufgrund technischer Einschränkungen, wird dieser Dialog immer auf englisch angezeigt!

3.5.2 Editoreinstellungen



- Auto indent mode - drückt man Enter wird der Cursor auf der nächsten Zeile bis zum Anfang der vorherigen Zeile eingerückt.
- Overwrite mode - ist diese Option an, ist Überschreiben als Standard eingestellt.
- Optimal fill - "Automatisches Einrücken" füllt zuerst mit Tabulatoren und den Rest mit Leerzeichen.
- Backspace unindents - mit Backspace springt man an die Stelle an der die Zeichen der vorherigen Zeile beginnen.
- Group Undo - eine Undo Operation wird nicht in kleinen Schritten, sondern in Blöcken durchge-

führt.

- Group Redo - eine Redo Operation wird nicht in kleinen Schritten, sondern in Blöcken durchgeführt.
- Keep caret in text - man kann den Cursor hinter das Dateiende positionieren.
- Keep trailing blanks - ist dies aktiviert, werden Leerzeichen am Ende der Zeile nicht gelöscht.
- Overwrite blocks - ist ein Block selektiert, so ersetzt die nächste Eingabe den Block.
- Disable Selection - Text kann nicht selektiert werden.
- Enable text dragging - Selektierter Text kann mit der Maus "gedragged" (bei gedrückter linker Maustaste verschoben) werden.
- Double click line - normalerweise selektiert ein Doppelklick ein Wort.
- Fixed line height - Verhindert Zeilenhöhen Berechnung. Zeilenhöhe ist definiert durch den "Default Style".
- Persistent blocks - Lässt markierte Blöcke selektiert, auch wenn der Cursor mit den Pfeiltasten bewegt wird, bis ein neuer Block selektiert ist.
- Overwrite blocks - Replaces a marked block of text with whatever is typed next. If Persistent Blocks is also selected, text you enter is appended following the currently selected block.
- Show caret in read only mode - Shows caret in read only mode.
- Copy to clipboard as RTF - Kopiert selektierten Text im RTF Format.
- Enable column selection - Ermöglicht Spaltenselektion.
- Hide selection - Versteckt Selektion wenn der Editor den Fokus verliert.
- Hide dynamic - Versteckt dynamisches Hervorhebungen wenn der Editor den Fokus verliert.
- Enable text dragging - Ermöglicht Drag & Drop für Text Bewegungen.
- Collapse empty lines - Faltet leere Zeilen nach dem Textbereich, wenn der Textbereich gefaltet wurde.
- Keep trailing blanks - Erhält Leerzeichen am Ende der Zeile.
- Float markers - Wenn Marker mit dem Text verbunden sind, bewegen sie sich bei Änderungen mit dem Text.
- Undo after save - Der Undo Puffer wird auch nach dem Speichern erhalten.
- Disable selection - Sperrt jede Selektion.
- Draw current line focus - Zeichnet ein "Fokus Rechteck" um die aktuelle Zeile wenn der Editor den Fokus hat.
- Hide cursor on type - Hides mouse cursor when user type text and mouse cursor within client area.
- Scroll to last line - When it is true you may scroll to last line of text, otherwise you can scroll to last page. When this option is off and total text height less then client height vertical scroll bar will be hidden.
- Greedy selection - If this option is set selection will contain extra column/line during column/line selection modes.
- Keep selection mode - Selection enabled for caret movement commands (like in BRIEF).
- Smart caret - Acts on the caret movement (up, down, line start, line end). Caret is moved to the nearest position on the screen.
- Word wrap - Determines whether the editor wraps text at the right side of text area.
- Word break on right margin - Determines whether text wraps (word-wrap mode) on the right margin instead of right side of client area.
- Optimal fill - Begins every auto indented line with the minimum number of characters possible, using tabs and spaces as necessary.
- Fixed column move - Keeps X position of caret before editing text, this position is used when moving up/down caret.
- Variable horizontal scroll bar - Sets range of horizontal scroll bar to the maximal width of only visible lines. Hides horizontal scroll bar if visible lines fit client width.
- Unindent keep align - Restricts unindent operation when at least one of lines can not be unindented.

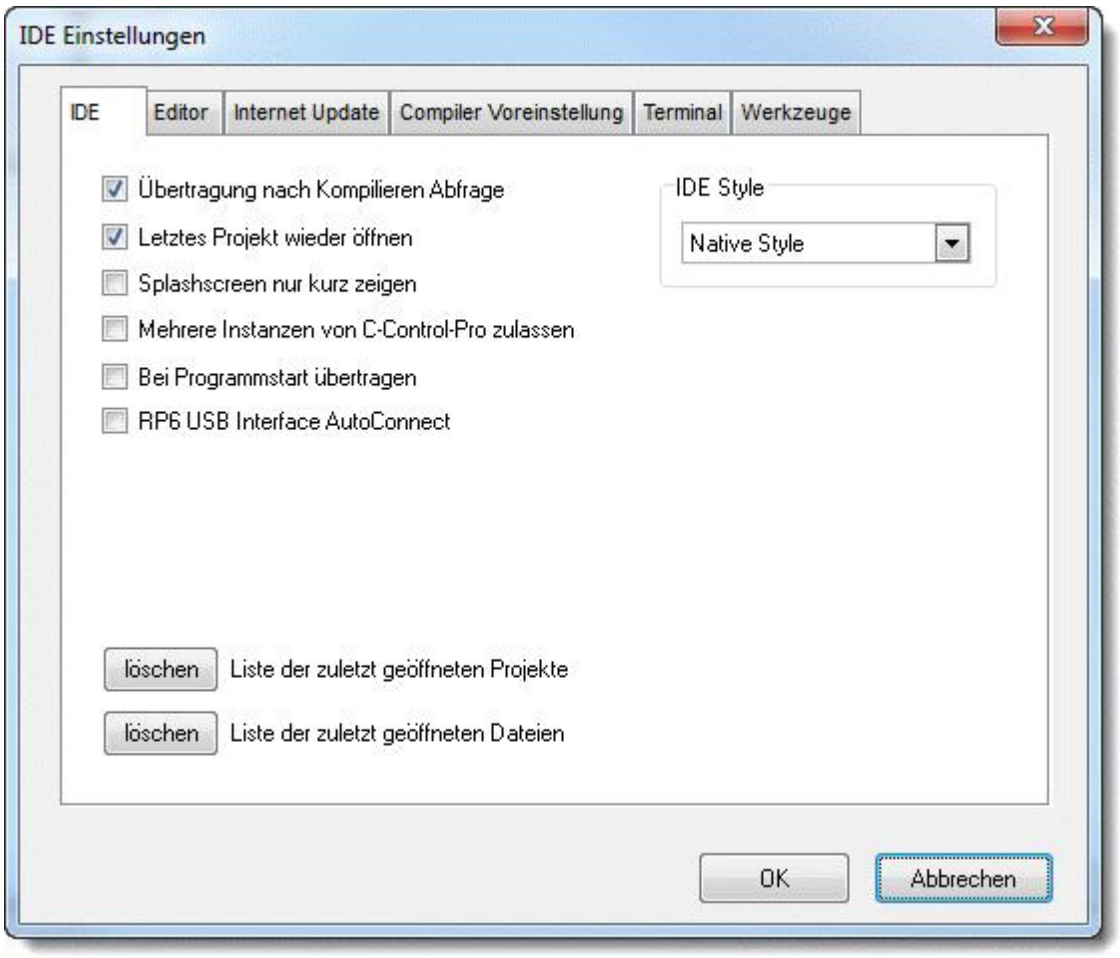
Bei Block indent wird die Anzahl der Leerzeichen eingetragen, mit der ein selektierter Block mit der Tabulator Taste eingerückt bzw. ausgerückt wird.

Das Eingabefeld Tab stops bestimmt wie viele Zeichen ein Tabulator breit ist.

➔ Aufgrund technischer Einschränkungen, wird dieser Dialog immer auf englisch angezeigt!

3.5.3 IDE Einstellungen

Man kann einzelne Aspekte der IDE konfigurieren.

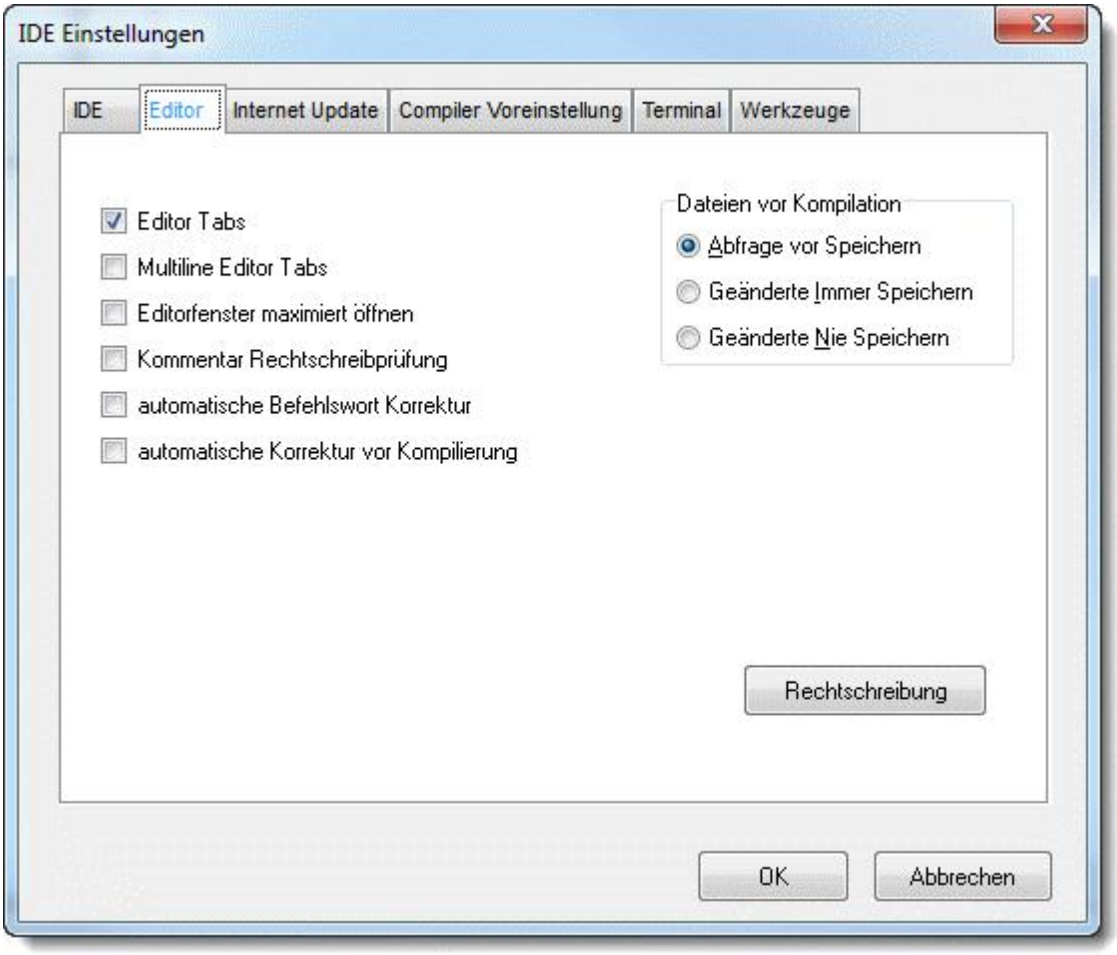


- Übertragung nach Kompilieren Abfrage - Wurde ein Programm kompiliert, aber nicht zum C-Control Modul übertragen, erfolgt eine Nachfrage beim Benutzer ob das Programm gestartet werden soll.
- Letztes Projekt wieder öffnen - Das letzte offene Projekt wird beim Starten der C-Control Pro IDE wieder geöffnet.
- 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.
- Bei Programmstart übertragen - Übertragt das Programm automatisch wenn Programmstart in der IDE gedrückt wird.
- RP6 USB Interface AutoConnect - Unterstützt Hardware Interface des RP6 Roboters

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

3.5.3.1 Editor



- Editor Tabs - Verschiedene Dateien werden als Editor Tabs angezeigt.
- Multiline Editor Tabs - Die Tabs werden mehrreihig dargestellt.
- Editorfenster maximiert öffnen - Bei dem Öffnen einer Datei wird automatisch das Editorfenster auf volle Größe geschaltet.
- Dateien vor Kompilation - Bestimmt die Aktion bei geänderten Dateien wenn kompiliert wird.
- Kommentar Rechtschreibprüfung - Die Kommentare innerhalb des Editors werden auf Rechtschreibfehler überprüft
- automatische Befehlswort Korrektur - Während des Schreibens werden bei allen Befehlswörtern

und bekannten Bibliotheksfunktionen die Groß-Kleinschreibung korrigiert

- automatische Korrektur vor Kompilierung - Bei dem Starten des Compilerlaufes werden bei allen Befehlswörtern und bekannten Bibliotheksfunktionen die Groß-Kleinschreibung korrigiert

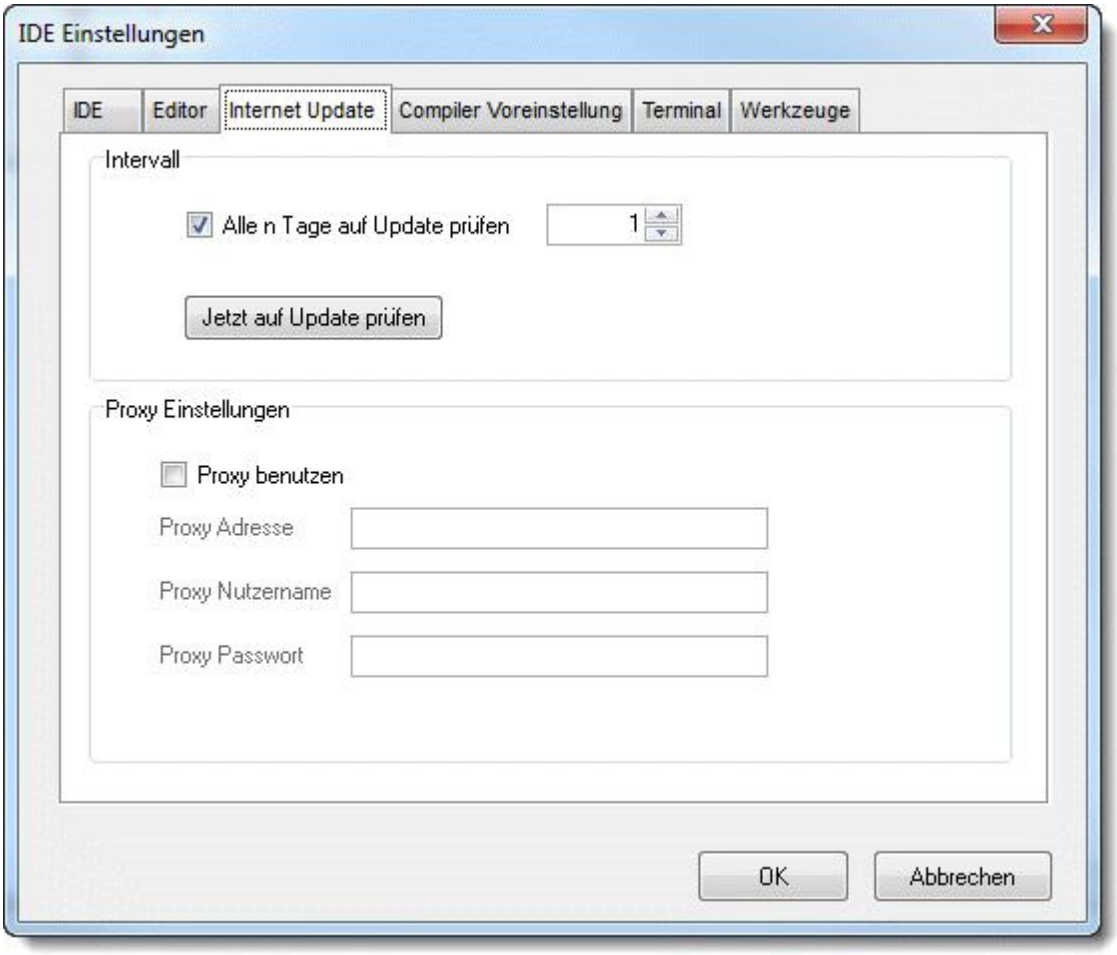
Der Knopf Rechtschreibung zeigt den Dialog zur Einstellung der Rechtschreibprüfung.

3.5.3.2 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.



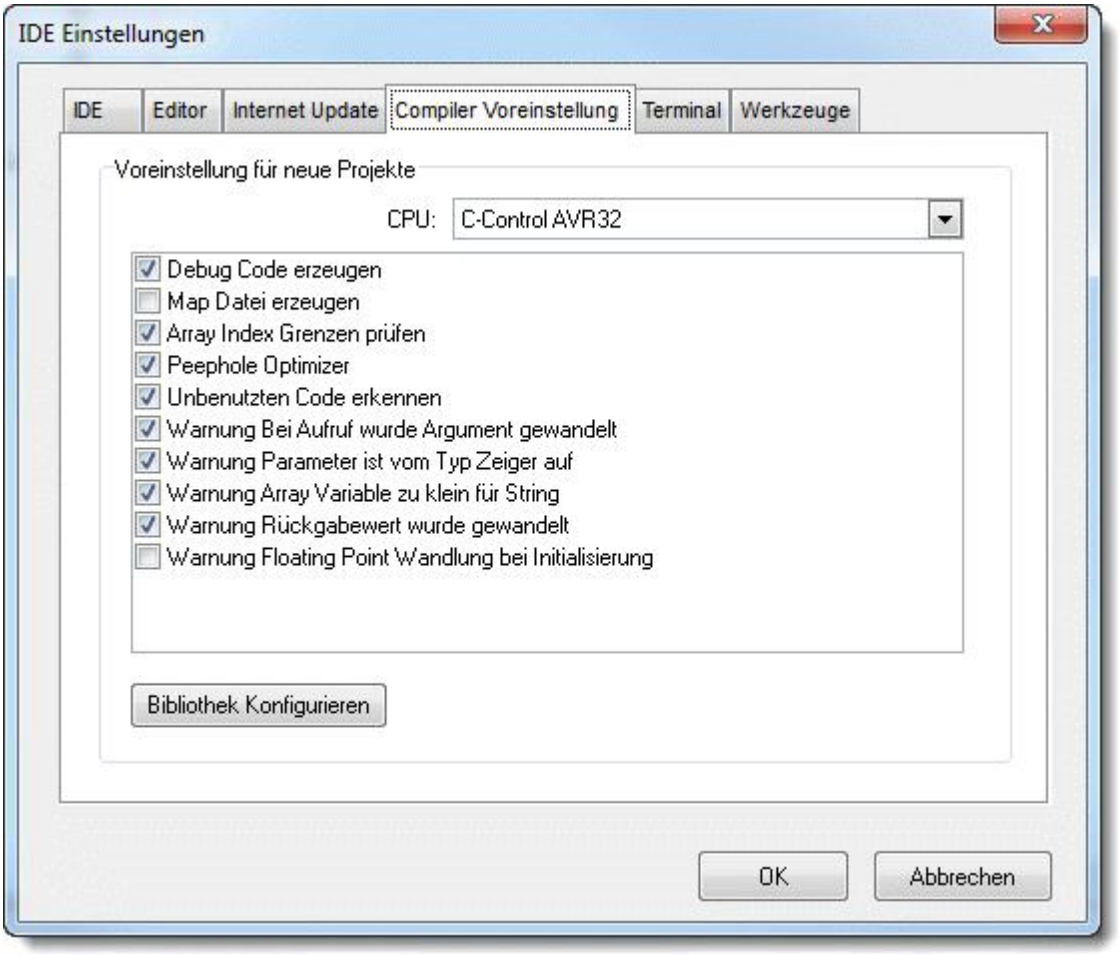
Ist z.B. wegen einer Firewall, der Zugang auf das Internet durch einen Proxy beschränkt, so können

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.

3.5.3.3 Compilervoreinstellung

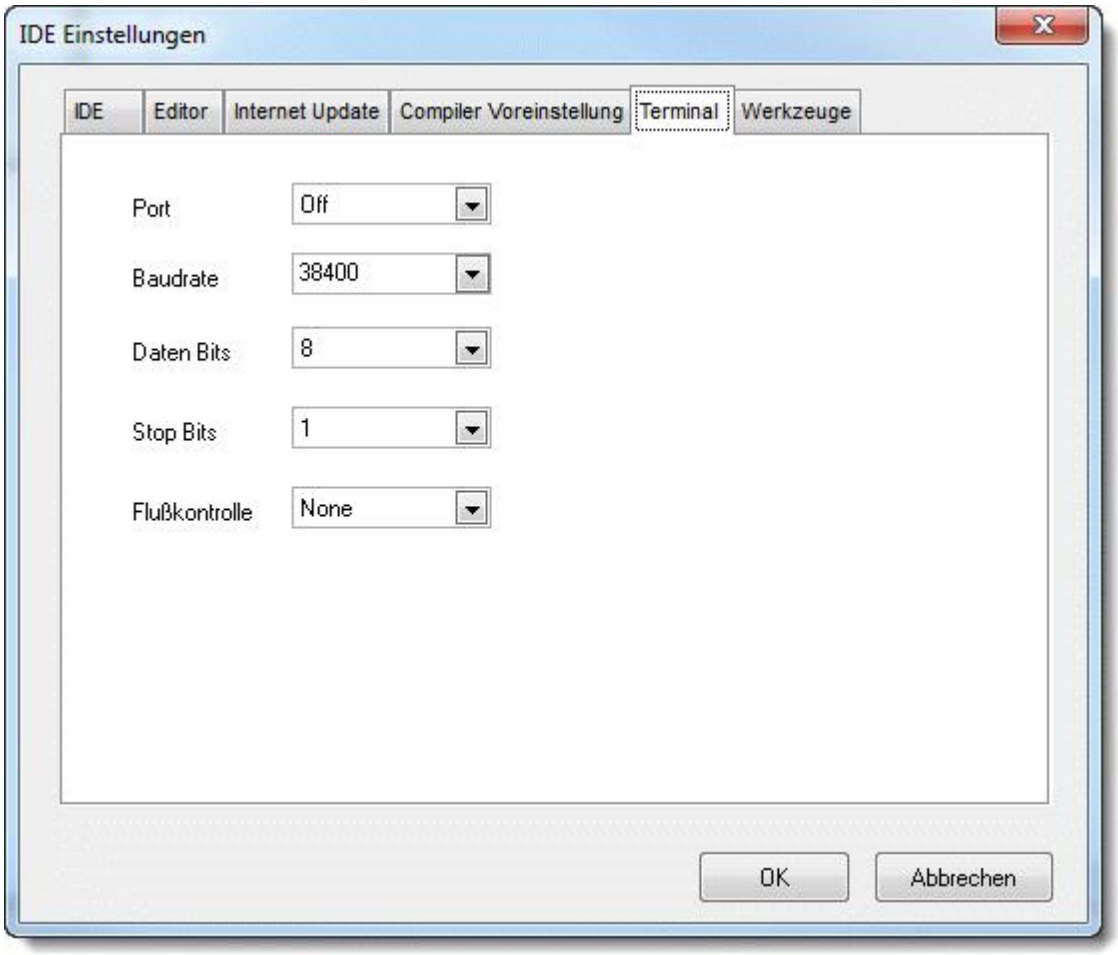
In der Compilervoreinstellung können die Standardwerte konfiguriert werden, die beim Erzeugen eines neues Projektes gespeichert werden. Die Voreinstellung ist unter Compiler im Menü Optionen zu erreichen.



Eine Beschreibung der Optionen befindet sich bei Projektoptionen. Die Auswahlbox "Bibliothek Konfigurieren" ist identisch zu der Beschreibung im Kapitel Projekte.

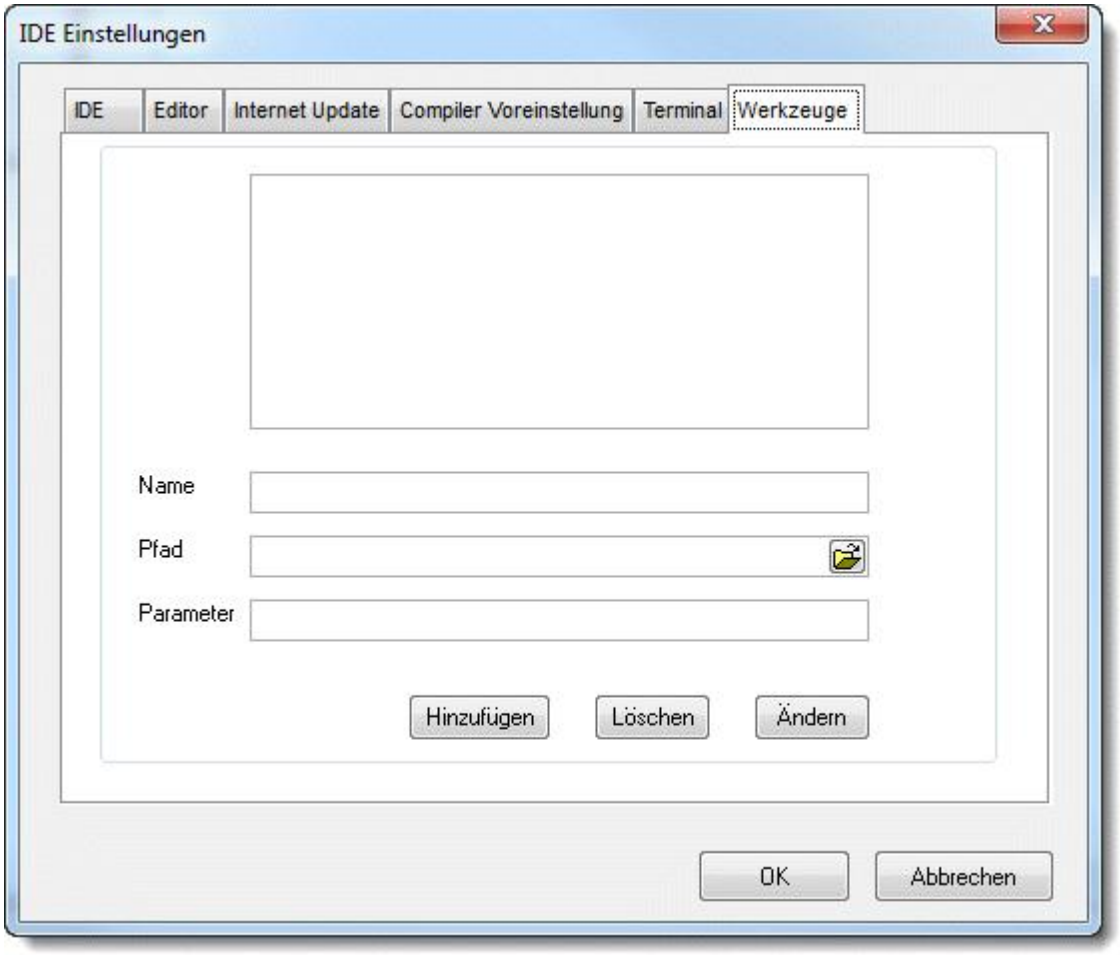
3.5.3.4 Terminal

Hier lassen sich die seriellen Parameter des eingebauten Terminalprogramms einstellen. Bei Port kann man den entsprechenden seriellen COM Port aussuchen. Weiter lassen sich die gängigen Baudraten, die Anzahl der Daten- und Stop Bits und die Flußkontrolle einstellen.



3.5.3.5 Werkzeuge

In den Werkzeug Einstellungen lassen sich die Einträge einfügen, löschen und verändern, mit denen man beliebige Programm aus der IDE schnell und einfach starten kann. Die Namen der Programme sind dann im "Werkzeug" Pulldown Menü zu finden und können mit einem Klick gestartet werden.



Für jedes Programm das hinzugefügt wird, kann man einen eigenen Namen, den Ausführungspfad zur Datei, sowie die zu übergebenden Parameter einstellen.

3.6 Fenster

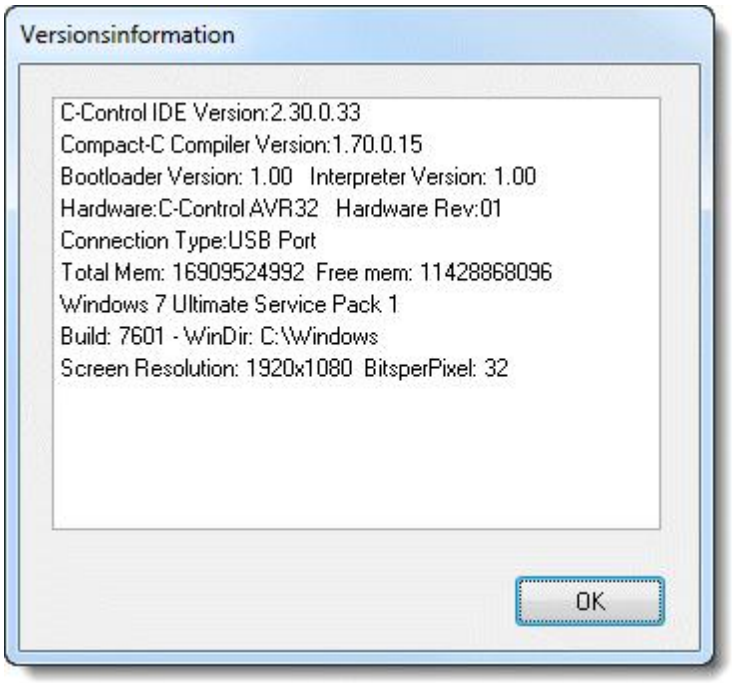
Sind im Editorbereich mehrere Fenster geöffnet, so kann man über Kommandos im Fenster Menü, die Editorfenster automatisch anordnen lassen.

- 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.

3.7 Hilfe

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

Der Menüpunkt Programmversion öffnet folgendes "Versionsinformations" 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 4

4 Compiler

4.1 Allgemeine Features

Dieser Bereich gibt Auskunft über Compiler Eigenschaften und Features die unabhängig von der benutzten Programmiersprache sind.

4.1.1 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, 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
```

➔ Besteht ein Projekt aus mehreren Quellen, so ist ein #define Konstante für alle Quelldateien existent ab der Datei, in der die Konstante definiert wurde. Daher ist es möglich, die Reihenfolge der Quelldateien in ein Projekt zu ändern.

Bedingte Kompilierung

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

Man kann kontrollieren, welche Teile 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!

4.1.1.1 Vordefinierte Symbole

Um die Arbeit mit verschiedenen Ausführungen der C-Control Pro Serie zu erleichtern, existieren eine Reihe von Definitionen die in Abhängigkeit von Zielsystem und Compiler Projektoptionen gesetzt werden. Diese Konstanten können mit #ifdef, #ifndef oder #if abgefragt werden.

Symbol	Bedeutung
MEGA32	Konfiguration für Mega 32
MEGA128	Konfiguration für Mega 128
MEGA128CAN	Konfiguration für Mega 128 CAN Bus
AVR32	Konfiguration für AVR 32
MEGA128_ARCH	Mega 128 oder Mega 128 CAN
CANBUS_SUPP	CAN Bus wird unterstützt
DEBUG	Debugdaten werden erzeugt
MAPFILE	Ein Speicherlayout wird berechnet

Die folgenden Konstanten enthalten einen String. Es macht Sinn sie in Verbindung mit Textausgaben zu verwenden.

Symbol	Bedeutung
__DATE__	aktuelles Datum
__TIME__	Uhrzeit der Kompilierung
__LINE__	aktuelle Zeile im Sourcecode
__FILE__	Name der aktuellen Quelldatei
__FUNCTION__	aktueller Funktionsname

Beispiel

Es werden Zeilennummer, Dateiname und Funktionsname ausgegeben. Da der Dateiname lang werden kann, bitte das character Array großzügig dimensionieren:

```
char txt[60];

txt=__LINE__;
Msg_WriteText(txt);  // Zeilennummer ausgeben
Msg_WriteChar(13);   // LF
txt=__FILE__;
Msg_WriteText(txt);  // Dateinamen ausgeben
Msg_WriteChar(13);   // LF
txt=__FUNCTION__;
Msg_WriteText(txt);  // Funktionsnamen ausgeben
Msg_WriteChar(13);   // LF
```

4.1.2 Pragma Anweisungen

Mit der Anweisung #pragma kann die Ausgabe und der Ablauf des Compilers gesteuert werden. Folgende Kommandos sind zulässig:

#pragma Error "xyz..."	Ein Fehler wird erzeugt und der Text "xyz..." ausgegeben
#pragma Warning "xyz..."	Eine Warnung wird erzeugt und der Text "xyz..." ausgegeben
#pragma Message "xyz..."	Der Text "xyz..." wird vom Compiler ausgegeben

Beispiel

Diese #pragma Anweisungen werden oft im Zusammenspiel mit Preprozessor Befehlen und vordefinierten Konstanten eingesetzt. Ein klassisches Beispiel ist die Produktion einer Fehlermeldung, wenn bestimme Hardwarekriterien nicht erfüllt werden:

```
#ifdef MEGA128
#pragma Error "Counter Funktionen nicht bei Timer0 und Mega128"
#endif
```

4.1.3 Map Datei

Ist bei der Kompilierung eine Map Datei generiert worden, kann man dort die Speichergröße der benutzten Variablen in Erfahrung bringen.

Beispiel

Das Projekt CNT0.cprj generiert bei der Kompilierung folgende Map Datei:

```
Globale Variablen               Laenge    Position (RAM Anfang)
-------------------------------------------------------------
Gesamtlaenge: 0 bytes

Lokale Variablen                Laenge    Position (Stackrelativ)
-------------------------------------------------------------
Funktion Pulse()
```

```
count                          2         4
i                              2         0
Gesamtlaenge: 4 bytes

Funktion main()
count                          2         2
n                              2         0
Gesamtlaenge: 4 bytes
```

Aus dieser Liste ist ersichtlich, daß keine globalen Variablen benutzt werden. Weiter existieren zwei Funktionen, "Pulse()" und "main()". Jede dieser Funktionen hat einen Speicherverbrauch von 4 Byte an lokalen Variablen.

4.2 CompactC

Eine Möglichkeit den C-Control Pro Mega 32 oder Mega 128 zu programmieren ist in der Programmiersprache CompactC. Der Compiler übersetzt die Sprache CompactC in einen Bytecode, der vom Interpreter des C-Control Pro abgearbeitet wird. Der Sprachumfang von CompactC entspricht im wesentlichen ANSI-C, ist aber an einigen Stellen reduziert, da die Firmware resourcensparend implementiert werden mußte. Folgende Sprachkonstrukte fehlen:

- **structs** / **unions**
- **typedef**
- **enum**
- Konstanten (**const** Anweisung)
- Zeigerarithemetik

Ausführliche Programmbeispiele sind im Verzeichnis "C-Control Pro Demos" zu finden, das mit der Entwicklungsumgebung installiert wurde. Dort sind für fast alle Aufgabenbereiche des C-Control Pro Moduls Beispiellösungen.

Die folgenden Kapitel beinhalten eine systematische Einführung in die Syntax und Semantik von CompactC.

4.2.1 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 Projektdateien kann man zu einem Projekt Bibliotheken hinzufügen, die Funktionen bereitstellen, die vom Programm genutzt werden.

4.2.2 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, 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 nachdem, 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.

4.2.3 Datentypen

Werte haben immer einen bestimmten Datentyp. Die Integerwerte (ganzzahlige Werte) haben in CompactC einen 8, 16 oder 32 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
byte	Nein	0 ... 255	8
int	Ja	-32768 ... +32767	16
unsigned int	Nein	0 ... 65535	16
word	Nein	0 ... 65535	16
long (kein Mega32)	Ja	-2147483648 ... 2147483647	32
unsigned long (kein Mega32)	Nein	0 ... 4294967295	32
dword (kein Mega32)	Nein	0 ... 4294967295	32
float	Ja	±1.175e-38 to ±3.402e38	32

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

➔ Da der Interpreter sonst zu groß werden würde, sind die 32-Bit Integer Datentypen nicht auf dem Mega32 verfügbar.

Strings

Es existiert kein expliziter "String" Datentyp. Ein String basiert auf einem character array. Man muss die Größe des arrays so wählen, daß 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 Ergebnisses 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 der andere der beiden Operanden einen 8 Bit oder 16 Bit Datentyp haben, so wird er vor der Operation in einen **float** Datentyp umgewandelt.

4.2.4 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. Binärzahlen können mit dem Prefix "**0b**" erzeugt werden. Bei Variablen mit vorzeichenbehaftetem Datentyp lassen sich negative Dezimalzahlen zuweisen, indem ein Minuszeichen vor die Zahl geschrieben wird.

➔ Für Zahlen ohne Dezimalpunkt oder Exponent wird angenommen, das sie vom Typ Integer mit Vorzeichen sind. Um eine Zahl explizit als vorzeichenlosen Integer zu definieren, so ist ein "u" direkt hinter die Zahl zu schreiben. Damit eine Zahl als 32-Bit (**long**) Typ gekennzeichnet ist, so ist der Wert entweder größer 65535 oder es wird ein "l" hinter die Zahl gesetzt.

Beispiele:

```
char c;
word a;
int i,j;
```

```
c=5;
c='a';       // Bei einfachen Anführungszeichen wird der ASCII Wert übernommen
a=0x3ff;     // hexadezimalzahlen sind immer unsigned
x=0b1001;    // Binärzahl
a=50000u;    // unsigned
a=100ul;     // unsigned 32 Bit (dword)
i=15;        // default ist immer signed
j=-22;       // signed
```

Fließkommazahlen (Datentyp **float**) dürfen ein Dezimalpunkt 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.

➔ Nur wenn die Compiler Option "Array Index Grenzen prüfen" gesetzt ist, findet während des Programmlaufs eine Überprüfung statt, ob die definierte Indexgrenze eines Arrays überschritten wurde. Wird ansonsten der Index während der Programmabarbeitung zu groß, so wird auf fremde Variablen zugegriffen, und die Chance ist groß, daß das Programm "abstürzt".

Tabellen mit vordefinierten Arrays

Seit Version 2.0 der IDE können Arrays mit Werten vorbelegt werden:

```
byte glob[10] = {1,2,3,4,5,6,7,8,9,10};
flash byte fglob[2][2]={10,11,12,13};

void main(void)
{
    byte loc[5]= {2,3,4,5,6};
    byte xloc[2][2];

    xloc= fglob;
}
```

Da bei der C-Control Pro Unit mehr Flash als RAM Speicher zur Verfügung steht, kann man mit dem **flash** Befehlswort Daten definieren, die nur im Flashspeicher stehen. Diese Daten können dann durch eine Zuweisung auf ein Array im RAM mit gleichen Dimensionen kopiert werden. Im Beispiel ist dies: "xloc= fglob". Diese Art der Zuweisung gibt es nicht in normalem "C".

Direkter Zugriff auf flash Array Einträge

Seit Version 2.12 ist es möglich auf einzelne Einträge in flash Arrays zuzugreifen:

```
flash byte glob[10] = {1,2,3,4,5,6,7,8,9,10};

void main(void)
{
    int a;

    a= glob[2];
}
```

➔ Eine Begrenzung bleibt bestehen: Nur normale Arrays die im RAM liegen, können als Referenz einer Funktion übergeben werden. Dies ist mit Referenzen auf flash Arrays nicht möglich.

Strings

Es existiert kein expliziter "String" Datentyp. Ein String basiert auf einem Array vom Datentyp **char**. Man muss die Größe des Arrays so wählen, daß 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!";
```

Man darf spezielle Zeichen in Strings einbetten die mit einem "\" (Backslash) gestartet werden. Folgende Sequenzen sind definiert:

Sequenz	Zeichen/Wert
\\	\
\'	'
\a	7
\b	8
\t	9
\n	10
\v	11
\f	12
\r	13

➔ Man kann keinen String einem mehrdimensionalen **Char** Array zuweisen. Es gibt aber Tricks für Fortgeschrittene:

```
char str_array[3][40];
char single_str[40];

single_str="A String";

// kopiert single_str in den zweiten String von str_array
Str_StrCopy(str_array,single_str,40);
```

Dies funktioniert, da mit einem Abstand von 40 Zeichen hinter dem ersten String, in str_array der Platz für den zweiten String liegt.

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 dagegen, sind beim Start der Funktion nicht initialisiert und können beliebige Werte haben!

Lokale Variablen werden, während der Berechnung einer Funktion, 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.

4.2.5 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.

4.2.5.1 Arithmetische Operatoren

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

➔ Es ist zu beachten, daß in einem Ausdruck die Zahl 7 einen Integer Datentyp zugewiesen bekommt. Möchte man explizit eine Zahl vom Datentyp **float** erzeugen, so ist ein Dezimalpunkt 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

4.2.5.2 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

4.2.5.3 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

4.2.5.4 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

4.2.5.5 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

4.2.5.6 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

4.2.6 Kontrollstrukturen

Kontrollstrukturen erlauben es den Programmablauf in Abhängigkeit von Ausdrücken, Variablen oder äußeren Einflüssen zu ändern.

4.2.6.1 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++;
```

4.2.6.2 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, daß 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 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:

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

```
} while(1);  // Endlosschleife
```

4.2.6.3 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, daß 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;
}
```

4.2.6.4 goto

Auch wenn man es innerhalb von strukturierten Programmiersprachen vermeiden sollte, so ist es möglich innerhalb einer Prozedur mit **goto** zu einem label zu springen:

```
// for Schleife mit goto realisiert
void main(void)
{
    int a;

    a=0;
label0:
    a++;
    if(a<10) goto label0;
}
```

4.2.6.5 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++;
}
```

4.2.6.6 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;
}
```

Die Abarbeitung der **switch** Anweisung ist im Interpreter optimiert, da alle Werte in einer Sprungtabelle abgelegt werden. Daraus resultiert die Einschränkung das der berechnete *Ausdruck* immer als vorzeichenbehafteter 16 Bit Integer (-32768 .. 32667) ausgewertet wird. Ein "**case** > 32767" ist daher nicht sinnvoll.

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.

4.2.6.7 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; // bricht Schleife ab
}
```

4.2.7 Funktionen

Um größere Programme zu strukturieren, teilt 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 Klam-

mer 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

➔ Man kann zur Zeit nur Arrays "by Reference" übergeben!

Zeigerarithmetik

In der aktuellen C-Control Pro Software ist auch Arithmetik auf einer Referenz (Zeiger) erlaubt, wie das folgende Beispiel zeigt. Die Arithmetik ist auf Addition, Subtraktion, Multiplikation und Division beschränkt.

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

    text="hallo welt";
    len=StringLength(text+2*3);
}
```

➔ Die Zeigerarithmetik ist zur Zeit experimentell und kann eventuell noch Fehler enthalten.

Strings als Argument

Seit Version 2.0 der IDE kann man nun Funktionen mit einem String als Argument aufrufen. Die aufgerufene Funktion bekommt die Zeichenkette als Referenz übergeben. Da aber Referenzen im RAM stehen müssen, und vordefinierte Zeichenketten im Flashspeicher stehen, erzeugt der Compiler intern vor Aufruf der Funktion einen anonymen Speicherplatz auf dem Stack und kopiert die Daten aus dem Flash dorthin.

```
int StringLength(char str[])
{
...
}

void main(void)
{
    int len;

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

4.2.8 Tabellen

4.2.8.1 Operator Präzedenz

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

4.2.8.2 Operatoren

	Arithmetische Operatoren
+	Addition
-	Subtraktion
*	Multiplikation
/	Division

%	Modulo
-	negatives Vorzeichen

	Vergleichsoperatoren
<	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

4.2.8.3 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	dword	long

4.3 BASIC

Die zweite Programmiersprache für das C-Control Pro Mega Modul ist BASIC. Der Compiler übersetzt die BASIC Befehle in einen Bytecode, der vom Interpreter des C-Control Pro abgearbeitet wird. Der Sprachumfang, des hier verwendeten BASIC Dialektes, entspricht in großen Teilen dem Industriestandard der großen Softwareanbieter. Folgende Sprachkonstrukte fehlen:

- Objektorientierte Programmierung
- Structures
- Konstanten

Ausführliche Programmbeispiele sind im Verzeichnis "C-Control Pro Demos" zu finden, das mit der Entwicklungsumgebung installiert wurde. Dort sind für fast alle Aufgabenbereiche des C-Control Pro Moduls Beispiellösungen zu finden.

Die folgenden Kapitel beinhalten eine systematische Einführung in die Syntax und Semantik des C-Control Pro BASIC.

4.3.1 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:

```
Sub main()
      Msg_WriteInt(42)  // die Antwort auf alles
End Sub
```

Projekte

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

4.3.2 Anweisungen

Anweisung

Eine Anweisung besteht aus mehreren reservierten Befehlswörtern, Bezeichnern und Operatoren, die vom Ende der Zeile 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. Mit dem "_" (Unterstrich) Zeichen ist es möglich, eine Anweisung auf die nächste Zeile auszudehnen.

```
If a=5 _ ' Anweisung über 2 Zeilen
a=a+10
```

➔ Auch ist es möglich mehr als eine Anweisung in einer Zeile zu platzieren. Das ":" (Doppelpunkt) Zeichen trennt dann die einzelnen Anweisungen. Aus Gründen der Lesbarkeit sollte von dieser Option aber nur selten Gebrauch gemacht werden.

```
a=1 : b=2 : c=3
```

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 einem einzelnen Anführungsstrich 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 nachdem, 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.

4.3.3 Datentypen

Werte haben immer einen bestimmten Datentyp. Die Integerwerte (ganzzahlige Werte) haben in BASIC einen 8, 16 oder 32 Bit breiten Datentyp, floating point Zahlen sind immer 4 byte lang.

Datentyp	Vorzeichen	Wertebereich	Bit
Char	Ja	-128 ... +127	8
Byte	Nein	0 ... 255	8
Integer	Ja	-32768 ... +32767	16
UInteger	Nein	0 ... 65535	16
Word	Nein	0 ... 65535	16
Long (kein Mega32)	Ja	-2147483648 ... 2147483647	32
ULong (kein	Nein	0 ... 4294967295	32

Mega32)			
Single	Ja	±1.175e-38 to ±3.402e38	32

➔ Da der Interpreter sonst zu groß werden würde, sind die 32-Bit Integer Datentypen nicht auf dem Mega32 verfügbar.

Strings

Es existiert kein expliziter "String" Datentyp. Ein String basiert auf einem character array. Man muss die Größe des arrays so wählen, daß 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 **Single** konvertiert und danach 5.5 addiert. Der Datentyp des Ergebnisses ist auch **Single**. Es gelten bei der Typkonvertierung folgende Regeln:

- Ist bei der Verknüpfung von zwei 8 Bit oder 16 Bit Integerwerten einer der beiden Datentypen vorzeichenbehaftet so ist auch das Ergebnis des Ausdrucks vorzeichenbehaftet.
- Ist einer der beiden Operanden vom Typ Single, so ist auch das Ergebnis vom Typ Single. Sollte der andere der beiden Operanden einen 8 Bit oder 16 Bit Datentyp haben, so wird er vor der Operation in einen Single Datentyp umgewandelt.

4.3.4 Variablen

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

```
Dim Variablenname As Typ
```

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

```
Dim Name1, Name2, Name3 As Integer
```

Als Typ sind erlaubt: **Char**, **Byte**, **Integer**, **Word**, **Single**

Beispiele:

```
Dim a As Integer
```

```
Dim i,j As Integer
```

```
Dim xyz As Single
```

Integer Variablen lassen sich Zahlenwerte dezimal oder als Hexzahl zuweisen. Bei einer Hexzahl werden vor die Zahl die Buchstaben "**&H**" gesetzt. Zusätzlich ist es erlaubt, wie bei C Hexadezimalzahlen mit dem Prefix "**0x**" beginnen zu lassen. Binärzahlen können mit dem Prefix "**0B**" erzeugt werden. Bei Variablen mit vorzeichenbehaftetem Datentyp lassen sich negative Dezimalzahlen zuweisen, indem ein Minuszeichen vor die Zahl geschrieben wird.

➔ Für Zahlen ohne Dezimalpunkt oder Exponent wird angenommen, das sie vom Typ Integer mit Vorzeichen sind. Um eine Zahl explizit als vorzeichenlosen Integer zu definieren, so ist ein "u" direkt hinter die Zahl zu schreiben. Damit eine Zahl als 32-Bit (**ULong**) Typ gekennzeichnet ist, so ist der Wert entweder größer 65535 oder es wird ein "l" hinter die Zahl gesetzt.

Beispiele:

```
Dim c As Char
Dim a As Word
Dim i,j As Integer

c=5;
c=&"a";     ' Syntax für ASCII Wert
a=&H3ff     ' hexadezimalzahlen sind immer unsigned
a=50000u    ' unsigned
x=0b1001    ' Binärzahl
a=100ul     ' unsigned 32 Bit (ULong)
i=15        ' default ist immer signed
j=-22       ' signed
a=0x3ff     ' hexadezimalzahlen sind immer unsigned
```

Fließkommazahlen (Datentyp **Single**) dürfen ein Dezimalpunkt und einen Exponenten beinhalten:

```
Dim x,y As Single

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:

```
Dim s As Integer
Dim f As Single

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 runden 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:

```
Dim x(10) As Integer
```

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 Indizes, durch Komma getrennt, bei der Variablendefinition angefügt werden:

```
Dim x(3,4) As Integer  ' Array mit 3*4 Einträgen
Dim y(2,2,2) As Integer ' Array mit 2*2*2 Einträgen
```

➔ Arrays dürfen in BASIC 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.

➔ Nur wenn die Compiler Option "Array Index Grenzen prüfen" gesetzt ist, findet während des Programmlaufs eine Überprüfung statt, ob die definierte Indexgrenze eines Arrays überschritten wurde. Wird ansonsten der Index während der Programmabarbeitung zu groß, so wird auf fremde Variablen zugegriffen, und die Chance ist groß, daß das Programm "abstürzt".

Tabellen mit vordefinierten Arrays

Seit Version 2.0 der IDE können Arrays mit Werten vorbelegt werden:

```
Dim glob(10) = {1,2,3,4,5,6,7,8,9,10} As Byte
Flash fglob(2,2)={10,11,12,13} As Byte

Sub main()
    Dim loc(5)= {2,3,4,5,6} As Byte
    Dim xloc(2,2) As Byte

    xloc= fglob
End Sub
```

Da bei der C-Control Pro Unit mehr Flash als RAM Speicher zur Verfügung steht, kann man mit dem F**lash** Befehlswort Daten definieren, die nur im Flashspeicher stehen. Diese Daten können dann durch eine Zuweisung auf ein Array im RAM mit gleichen Dimensionen kopiert werden. Im Beispiel ist dies: "xloc= fglob".

Direkter Zugriff auf flash Array Einträge

Seit Version 2.12 ist es möglich auf einzelne Einträge in flash Arrays zuzugreifen:

```
Flash glob(10) = {1,2,3,4,5,6,7,8,9,10} As Byte

Sub main()
    Dim a As Byte

    a= glob(2)
End Sub
```

➔ Eine Begrenzung bleibt bestehen: Nur normale Arrays die im RAM liegen, können als Referenz einer Funktion übergeben werden. Dies ist mit Referenzen auf flash Arrays nicht möglich.

Strings

Es existiert kein expliziter "String" Datentyp. Ein String basiert auf einem Array vom Datentyp **Char**. Man muss die Größe des Arrays so wählen, daß 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:

```
Dim str1(21) As Char
```

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

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

Man darf spezielle Zeichen in Strings einbetten die mit einem "\" (Backslash) gestartet werden. Folgende Sequenzen sind definiert:

Sequenz	Zeichen/Wert
\\	\
\'	'
\a	7
\b	8
\t	9
\n	10
\v	11
\f	12
\r	13

➔ Man kann keinen String einem mehrdimensionalen **Char** Array zuweisen. Es gibt aber Tricks für Fortgeschrittene:

```
Dim str_array(3,40) As Char
Dim single_str(40) As Char
```

```
single_str="A String"

' kopiert single_str in den zweiten String von str_array
Str_StrCopy(str_array,single_str,40)
```

Dies funktioniert, da mit einem Abstand von 40 Zeichen hinter dem ersten String, in str_array der Platz für den zweiten String liegt.

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:

```
Dim a,b As Integer

Sub func1()
    Dim a,x,y As Integer
    ' 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
End Sub

Sub main()
    Dim u As Integer
    ' globale a,b sind zugreifbar
    ' lokale u ist zugreifbar
    ' x,y nicht zugreifbar da lokal zu Funktion func1
End Sub
```

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 dagegen, sind beim Start der Funktion nicht initialisiert und können beliebige Werte haben!

Lokale Variablen werden, während der Berechnung einer Funktion, 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** für den Datentyp setzen.

```
Sub func1()
    Static a As Integer
End Sub
```

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.

4.3.5 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.

4.3.5.1 Arithmetische Operatoren

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

➔ Es ist zu beachten, daß in einem Ausdruck die Zahl 7 einen Integer Datentyp zugewiesen bekommt. Möchte man explizit eine Zahl vom Datentyp **Single** erzeugen, so ist ein Dezimalpunkt 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
Mod	**Modulo**	15 Mod 4 17 Mod 2	3 1
-	**negatives Vorzeichen**	-(2+2)	-4

4.3.5.2 Bitoperatoren

Bitoperatoren sind nur für Integer Datentypen erlaubt.

Operator	Erklärung	Beispiel	Ergebnis
And	**Und**	&H0f And 3 &Hf0 And &H0f	3 0
Or	**Oder**	1 Or 3 &Hf0 Or &H0f	3 &Hff
Xor	**exclusives Oder**	&Hff Xor &H0f &Hf0 Xor &H0f	&Hf0 &Hff
Not	**Bitinvertierung**	Not &Hff Not &Hf0	0 &H0f

➔ All diese Operatoren arbeiten arithmetisch: Z.B. **Not** &H01 = &Hfe. Beide Werte werden in einem **If** Ausdruck zu wahr verarbeitet. Dies ist unterschiedlich zu einem logischen **Not** Operator, wo **Not** &H01 = &H00 ist.

4.3.5.3 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**	&Hff >> 6 16 >> 2	3 4

4.3.5.4 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

➔ Diese Operatoren sind normalerweise in Basic Dialekten nicht enthalten und kommen aus der Welt der C inspirierten Programmiersprachen.

4.3.5.5 Vergleichsoperatoren

Vergleichsoperatoren sind für **Single** 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

4.3.6 Kontrollstrukturen

Kontrollstrukturen erlauben es den Programmablauf in Abhängigkeit von Ausdrücken, Variablen oder äußeren Einflüssen zu ändern.

4.3.6.1 Do Loop While

Mit einem **Do ... Loop While** Konstrukt lassen sich abhängig von einer Bedingung Anweisungen in einer Schleife wiederholen:

```
Do
    Anweisungen
```

```
Loop While Ausdruck
```

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

Beispiele:

```
Do
    a=a+2
Loop While a<10
```

```
Do
    a=a*2
    x=a
Loop While a
```

➔ Der wesentliche Unterschied der **Do Loop while** Schleife zur normalen **Do While** Schleife ist der Umstand, daß in einer **Do Loop While** Schleife, die Anweisung mindestens einmal ausgeführt wird.

Exit Anweisung

Eine **Exit** Anweisung verläßt die Schleife, und die Programmausführung startet mit der nächsten Anweisung hinter der **Do Loop While** Schleife.

Beispiel:

```
Do
    a=a+1
    If a>10 Then
        Exit ' bricht Schleife ab
    End If
Loop While 1  ' Endlosschleife
```

4.3.6.2 Do While

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

```
Do While Ausdruck
    Anweisungen
End While
```

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.

Beispiele:

```
Do While a<10
    a=a+2
End While

Do While a
    a=a*2
    x=a
End While
```

Exit Anweisung

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

Beispiel:

```
Do While 1   ' Endlosschleife
    a=a+1
    If a>10 Then
        Exit ' bricht Schleife ab
    End If
End While
```

4.3.6.3 For Next

Eine **For Next** Schleife wird normalerweise benutzt, um eine bestimmte Anzahl von Schleifendurchläufen zu programmieren.

```
For Zählervariable=Startwert To Endwert Step Schrittweite
    Anweisungen
Next
```

Die Zählervariable wird auf den Startwert gesetzt, und danach die Anweisungen so oft wiederholt, bis der Endwert erreicht wird. Bei jedem Schleifendurchlauf, erhöht sich der Wert der Zählervariable um die Schrittweite, die auch negativ sein darf. Die Angabe der Schrittweite, hinter dem Endwert, ist optional. Wird die Schrittweite nicht angegeben, so hat sie den Wert 1.

➔ Da bei der **For Next** Schleife besonders optimiert wird, muss die Zählervariable vom Typ Integer sein.

Beispiele:

```
For i=1 To 10
    If i>a Then
        a=i
```

```
    End If
    a=a-1
Next


For i=1 To 10 Step 3  ' Erhöhe i in 3er Schritten
    If i>3 Then
        a=i
    End If
    a=a-1
Next
```

➔ An dieser Stelle nochmal der Hinweis, Arrays sind immer nullbasiert. Eine **For Next** Schleife, sollte daher bei einem Array Zugriff, eher von 0 nach 9 laufen.

Exit Anweisung

Eine **Exit** Anweisung verläßt die Schleife, und die Programmausführung startet mit der nächsten Anweisung hinter der **For** Schleife.

Beispiel:

```
For i=1 To 10
    If i=6 Then
        Exit
    End If
Next
```

4.3.6.4 Goto

Auch wenn man es innerhalb von strukturierten Programmiersprachen vermeiden sollte, so ist es möglich innerhalb einer Prozedur mit **Goto** zu einem label zu springen. Um ein label zu kennzeichnen wird das Befehlswort **Lab** vor den Labelnamen gesetzt.

```
' For Schleife mit Goto realisiert
Sub main()
    Dim a As Integer

    a=0
Lab label1
    a=a+1
    If a<10 Then
        Goto label1
    End If
End Sub
```

4.3.6.5 If .. Else

Eine **If** Anweisung hat folgende Syntax:

```
If Ausdruck1 Then
    Anweisungen1
ElseIf Ausdruck2 Then
    Anweisungen2
Else
    Anweisungen3
End If
```

Hinter der **If** Anweisung folgt ein arithmetischer Ausdruck. Wird dieser *Ausdruck* zu ungleich 0 ausgewertet, dann werden die Anweisungen1 ausgeführt. Man kann mit Hilfe des **Else** Befehlswortes alternative Anweisungen2 definieren, die dann ausgeführt wird, wenn der *Ausdruck* zu 0 berechnet wurde. Das Hinzufügen einer **Else** Anweisung ist optional und muss nicht geschehen.

Soll in dem **Else**-Zweig direkt wieder eine **If** Anweisung stehen, ist es möglich mit **ElseIf** direkt wieder ein **If** einzuleiten. Damit muss das neue **If** nicht in den **Else**-Block geschachtelt werden, und der Quelltext bleibt übersichtlicher.

Beispiele:

```
If a=2 Then
    b=b+1
End If


If x=y Then
    a=a+2
Else
    a=a-2
End If


If a<5 Then
    a=a-2
ElseIf a<10 Then
    a=a-1
Else
    a=a+1
End If
```

4.3.6.6 Select Case

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

```
Select Case Ausdruck
    Case konstanten_vergleich1
        Anweisungen_1
```

```
    Case konstanten_vergleich2
        Anweisungen_2
    .
    .
    Case konstanten_vergleich_x
        Anweisungen_x
    Else    ' Else ist optional
        Anweisungen
End Case
```

Der Wert von *Ausdruck* wird berechnet. Danach springt die Programmausführung zum dem Konstantenvergleich, der als erster zu wahr ausgewertet wird, und führt das Programm dort fort. Kann kein Konstantenvergleich erfüllt werden, so wird das **Select Case** Konstrukt verlassen.

Für den Konstantenvergleich können spezielle Vergleiche oder ganze Bereiche angeben werden. Hier Beispiele für alle Möglichkeiten:

Vergleich	Ausführung bei
Konstante, = Konstante	Ausdruck gleich Konstante
< Konstante	Ausdruck kleiner Konstante
<= Konstante	Ausdruck kleiner gleich Konstante
> Konstante	Ausdruck größer Konstante
>= Konstante	Ausdruck größer gleich Konstante
<> Konstante	Ausdruck ungleich Konstante
Konstante1 To Konstante2	Konstante1 <= Ausdruck <= Konstante2

➔ Die neuen Möglichkeiten Vergleiche in der Select Case Anweisung zu definieren sind neu in Version 1.71 eingeführt worden. Diese Erweiterung existiert nicht für CompactC switch Anweisungen.

➔ Die Abarbeitung der **Select Case** Anweisung ist im Interpreter optimiert, da alle Werte in einer Sprungtabelle abgelegt werden. Daraus resultiert die Einschränkung das der berechnete *Ausdruck* immer als vorzeichenbehafteter 16 Bit Integer (-32768 .. 32667) ausgewertet wird. Ein "**Case** > 32767" ist daher nicht sinnvoll.

Exit Anweisung

Ein **Exit** verläßt die **Select Case** Anweisung.

Ist in einer **Select Case** Anweisung ein **Else** definiert, so werden die Anweisungen hinter **Else** ausgeführt, wenn keine Konstantenvergleich gefunden wurde, der erfüllt werden konnte.

Beispiel:

```
Select Case a+2
    Case 1
        b=b*2
    Case = 5*5
        b=b+2
```

```
    Case 100 And &Hf
        b=b/c
    Case < 10
        b=10
    Case <= 10
        b=11
    Case 20 To 30
        b=12
    Case > 100
        b=13
    Case >= 100
        b=14
    Case <> 25
        b=15
    Else
        b=b+2
End Case
```

➔ In CompactC werden die Anweisungen hinter einer **case** Anweisung weitergeführt, bis ein **break** auftritt oder die **switch** Anweisung verlassen wird. Dies ist in BASIC anders: Hier bricht die Abarbeitung der Befehle hinter einem **Case** ab, wenn man bis zur nächsten **Case** Anweisung gelangt.

4.3.7 Funktionen

Um größere Programme zu strukturieren, teilt 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:

```
Sub func1()
    ' Anweisungen in Funktion func1
    .
    .
End Sub

Sub main()
    ' die Funktion func1 wird zweimal aufgerufen
    func1()
    func1()
End Sub
```

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 Parameternamen, und danach den Datentyp an. Will man keinen Parameter übergeben, so läßt man die Klammer leer. Ein Beispiel:

```
Sub func1(param1 As Word, param2 As Single)
    Msg_WriteHex(param1)  ' den ersten Parameter ausgeben
```

```
    Msg_WriteFloat(param2)  ' den zweiten Parameter ausgeben
End Sub
```

➔ 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.

```
Sub main()
    Dim a As Word
    Dim f As Single

    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
End Sub
```

➔ 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 hinter der Parameterliste der Funktion an.

```
Sub func1(a As Integer) As Integer
    Return a-10
End Sub
```

Der Rückgabewert wird innerhalb der Funktion mit der Anweisung "**Return** *Ausdruck*" angegeben. Hat man eine Funktion ohne Rückgabewert, so muss man die **Return** Anweisung 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 das Attribut "**ByRef**" vor den Parameternamen:

```
Sub StringLength(ByRef str As Char) As Integer
    Dim i As Integer

    i=0
    Do While str(i)
        i=i+1  ' wiederhole solange Zeichen nicht null
```

```
    End While
    Return i
End Sub

Sub main()
    Dim Len As Integer
    Dim Text(15) As Char

    Text="hallo welt"
    Len=StringLength(Text)
End Sub
```

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.

➔ Man kann zur Zeit nur Arrays "by Reference" übergeben!

Zeigerarithmetik

In der aktuellen C-Control Pro Software ist auch Arithmetik auf einer Referenz (Zeiger) erlaubt, wie das folgende Beispiel zeigt. Die Arithmetik ist auf Addition, Subtraktion, Multiplikation und Division beschränkt.

```
Sub main()
    Dim Len As Integer
    Dim Text(15) As Char

    Text="hallo welt"
    Len=StringLength(Text+2*3)
End Sub
```

➔ Die Zeigerarithmetik ist zur Zeit experimentell und kann eventuell noch Fehler enthalten.

Strings als Argument

Seit Version 2.0 der IDE kann man nun Funktionen mit einem String als Argument aufrufen. Die aufgerufene Funktion bekommt die Zeichenkette als Referenz übergeben. Da aber Referenzen im RAM stehen müssen, und vordefinierte Zeichenketten im Flashspeicher stehen, erzeugt der Compiler intern vor Aufruf der Funktion einen anonymen Speicherplatz auf dem Stack und kopiert die Daten aus dem Flash dorthin.

```
Sub StringLength(ByRef str As Char) As Integer
....
End Sub

Sub main()
    Dim Len As Integer
```

```
    Len=StringLength("hallo welt")
End Sub
```

4.3.8 Tabellen

4.3.8.1 Operatoren

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

	Vergleichsoperatoren
<	kleiner
>	größer
<=	kleiner gleich
>=	größer gleich
=	gleich
<>	ungleich

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

	Bitoperatoren
And	Und
Or	Oder
Xor	exclusives Oder
Not	Bitinvertierung

4.3.8.2 reservierte Worte

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

And	As	ByRef	Byte	Case
Char	Dim	Do	Else	ElseIf

End	Exit	False	For	Goto
If	Integer	Lab	Loop	Mod
Next	Not	Opc	Or	Return
Select	Single	SizeOf	Static	Step
Sub	Then	To	True	While
Word	Xor	Long	ULong	UInteger

4.3.8.3 Operator Präzedenz

Rang	Operator
10	( )
9	- (negatives Vorzeichen)
8	* /
7	Mod
6	+ -
5	<< >>
4	= <> < <= > >=
3	Not
2	And
1	Or Xor

4.4 Assembler

Mit IDE Version 2.0 wurde die Möglichkeit eröffnet auch Assembler Routinen in ein Projekt einzubinden. Als Assembler wir der Open Source Assembler AVRA eingesetzt. Die Sourcen des Assemblers sind im Installationsverzeichnis "GNU" zu finden. Mann kann von CompactC und Basic Assemblerroutinen aufrufen, die in voller CPU Geschwindigkeit laufen, im Gegensatz zum Bytecode Interpreter. Man kann den Assembler Prozeduren Parameter übergeben und Rückgabewerte bekommen. Auch der Zugriff auf globale Variablen des CompactC oder Basic Programms ist möglich. Der Compiler erkennt Assemblerdateien an der ".asm" Endung der Dateinamen. Assemblerdateien werden wie die CompactC oder Basic Dateien dem Projekt hinzugefügt.

➔ Die Programmierung von Assembler ist nur für fortgeschrittene Anwender des Systems gedacht. Die Programmierung ist sehr komplex und fehleranfällig, und sollte nur von denen verwendet werden, die das System sonst problemlos beherrschen.

➔ Es steht kein freier Assembler für die AVR32Bit Unit zur Verfügung. Da die C-Control Pro AVR32Bit auch deutlich schneller ist als die C-Control Pro Mega Serie, ist kein Assembler Support für die AVR32 geplant.

Literatur

Es gibt vielfältige Literatur über die Assembler Programmierung im Internet und auch im Buchhandel. Wichtig sind das "AVR Instruction Reference Manual" das man auf der Atmel Webseite und auch im "Manual" Verzeichnis der C-Control Pro Installation findet, und das "AVR Assembler User Guide" von der Atmel Webseite.

4.4.1 Ein Beispiel

Am folgenden Beispiel (ist auch in den Demo Programmen enthalten) wird die Struktur von Assemblerroutinen erklärt. Dabei muss in dem Projekt der CompactC Source Code die Endung ".cc" die Assembler Sourcen die Endung ".asm".

```
// CompactC Source
void proc1 $asm("tag1")(void);
int proc2 $asm("tag2")(int a, float b, byte c);

int glob1;
void main(void)
{
    int a;

    proc1();
    a= proc2(11, 2.71, 33);
}
```

Vor Aufruf der Assembler Prozeduren *proc1* und *proc2* müssen die beiden Prozeduren erstmal deklariert werden. Dies geschieht mit dem Befehlswort $asm. Die Deklaration sieht in Basic ähnlich aus:

```
' Basic Deklaration der Assembler Routinen
$Asm("tag1") proc1()
$Asm("tag2") proc2(a As Integer, b As Single, c As Byte) As Integer
```

Man sieht in der Deklaration die Strings "tag1" und "tag2". Diese Strings werden in einer ".def" Datei definiert, wenn tatsächlich ein Aufruf der deklarierten Funktionen stattfand. In diesem Fall sieht dann die ".def" Datei folgendermaßen aus:

```
; .def file
.equ glob1 = 2
.define tag1 1
.define tag2 1
```

Setzt man nun im Assembler Source die einzelnen Routinen in ".ifdef ..." Anweisungen, so werden die Routinen nur assembliert, wenn ein Funktionsaufruf wirklich stattfand. Dies spart Platz bei der Codegenerierung. Auch werden in der ".def" Datei die Positionen der globalen Variablen definiert. Die ".def" Datei wird automatisch zusammen mit den Assemblerdateien gemeinsam übersetzt, sie braucht nicht extra inkludiert zu werden.

Hier folgt nun der Assembler Source der Prozedur *proc1*. In diesem Source wird die globale Variable *glob1* auf den Wert 42 gesetzt.

```
; Assembler Source
.ifdef tag1
proc1:
    ; global variable access example
    ; write 42 to global variable glob1
```

```
    MOVW R26,R8          ; get RamTop from register 8,9
    SUBI R26,LOW(glob1)  ; subtract index from glob1 to get address
    SBCI R27,HIGH(glob1)

    LDI  R30,LOW(42)
    ST   X+,R30
    CLR  R30             ; the high byte is zero
    ST   X,R30

  ret
.endif
```

Im zweiten Teil des Assembler Sources werden die übergebenen Parameter "a" und "c" als integer addiert, und die Summe dann zurückgegeben.

```
.ifdef tag2
proc2:
    ; example for accessing and returning parameter
    ; we have int proc2(int a, float b, byte c);
    ; return a + c

    MOVW R30, R10   ; move parameter stack pointer into Z
    LDD R24, Z+5  ; load parameter "a" into R24,25
    LDD R25, Z+6

    LDD R26, Z+0  ; load byte parameter "c" into X (R26)
    CLR R27       ; hi byte zero because parameter is byte

    ADD R24, R26  ; add X to R24,25
    ADC R25, R27

    MOVW R30, R6      ; copy stack pointer from R6
    ADIW R30, 4       ; add 4 to sp  - ADIW only works for R24 and greater
    MOVW R6, R30      ; copy back to stack pointer location

    ST   Z+, R24      ; store R24,25 on stack
    ST   Z, R25

    ret
.endif
```

4.4.2 Datenzugriff

Globale Variablen

Im Bytecode Interpreter liegen in den Registern 8 und 9 ein 16-Bit Zeiger auf das Ende des Speicherbereichs der globalen Variablen. Möchte man auf eine globale Variable zugreifen, die in der ".def" Datei definiert wurde, so erhält man die Adresse der Variablen, wenn man die Position der Variablen von R8,R9 abzieht. Dies sieht dann so aus:

```
; global variable access example
; write 0042 to global variable glob1
MOVW R26,R8          ; get Ram Top from register 8,9
SUBI R26,LOW(glob1)  ; subtract index from glob1 to get address
SBCI R27,HIGH(glob1)
```

Liegt dann die Adresse der globalen Variable im X Registerpaar (R26,R27), dann kann man den gewünschten Wert (in unserem Beispiel 42) dort hineinschreiben:

```
LDI  R30,LOW(42)
ST   X+,R30
CLR  R30             ; the high byte of 42 is zero
ST   X,R30
```

Parameterübergabe

Parameter werden auf dem Stack des Bytecode Interpreters übergeben. Der Stackpointer (SP) sitzt im Registerpaar R10,R11. Werden Parameter übergeben, so werden sie der Reihe nach auf den Stack geschrieben. Da der Stack nach unten wächst, sieht in unserem Beispiel (integer a, floating point b, byte c) das Speicherlayout folgendermaßen aus:

```
SP+5: a  (typ integer, länge 2)
SP+1: b  (typ float, länge 4)
SP+0: c  (typ byte, länge 1)
```

Möchte man nun a und c addieren, so findet man *a* bei SP+5, und *c* bei SP. Im folgenden Assembler Code wird der SP (R10,R11) in das Registerpaar Z (R30,R31) kopiert, und dann indirekt über Z die beiden Parameter *a* und *c* geladen.

```
; example for accessing and returning parameter
; we have int proc2(int a, float b, byte c);
MOVW R30, R10 ; move parameter stack pointer into Z
LDD R24, Z+5  ; load parameter "a" into R24,25
LDD R25, Z+6

LDD R26, Z+0  ; load byte parameter "c" into X (R26)
CLR R27       ; hi byte zero because parameter is byte
```

Man hat jetzt die beiden Parameter *a* und *c* in den Registerpaaren X und R24,R25. Nun kann man die Zahlen addieren.

```
ADD R24, R26  ; add X to R24, R25
ADC R25, R27
```

Rückgabe von Werten

In der Routine *proc2* wird auch die Summe zurückgegeben. Rückgabewerte werden auf den Parameter Stack (PSP) des Bytecode Interpreter geschrieben. Der Zeiger auf den PSP liegt im Registerpaar R6,R7. Man muss vorher allerdings eine 4 auf den PSP Zeiger addieren, dann kann man den Wert dort speichern. Im Gegensatz zur Parameterübergabe spielt der Typ des Rückgabeparameters keine Rolle. Auf dem Parameterstack sind alle Parameter immer 4 Bytes lang.

➔ Auch bei einem 8-Bit Rückgabewert erwartet der Interpreter immer einen 16-Bit Wert, dies wird gemacht um Bytecodes im Interpreter einzusparen. Ist die Assemblerroutine als **byte** deklariert, muss ein **word** als Rückgabewert geschrieben werden, ist die Assemblerroutine vom Typ **char**, ist ein **int** erforderlich. In allen anderen Fällen ist keine Änderung erforderlich.

```
; return a + c
  MOVW R30, R6      ; copy stack pointer from R6
  ADIW R30, 4       ; add 4 to sp  - ADIW only works for R24 and greater
  MOVW R6, R30      ; copy back to stack pointer location

  ST   Z+, R26      ; store R26, R27 on stack
  ST   Z, R27
```

4.4.3 Leitfaden

Hier werden die wichtigsten Punkte erklärt, die man beim Programmieren in Assembler für die C-Control Pro beachten muss:

- Assembler Aufrufe sind atomar. Ein Assembleraufruf kann nicht vom Multithreading oder von der Bytecode Interruptroutine unterbrochen werden. Dies ist ähnlich wie bei den Aufrufen der Bibliothek. Ein Interrupt wird zwar von der internen Interruptstruktur registriert, aber die Bytecode Interrupt Routine wird erst nach Beendigung dem Assembler Prozedur gestartet.

- Das Y-Register (R28 und R29) darf nicht verändert werden, es wird vom Interpreter als data stack pointer genutzt. Assembler Interruptroutinen benutzen das Y-Registers um Daten aus Registern zu sichern und können sonst abstürzen.

- Die Register R0, R1, R22, R23, R24, R25, R26, R27, R30 und R31 können in Assembler Routinen ohne Sicherung benutzt werden. Benötigt man in Assembler andere Register, so muss man die Inhalte der Register vorher abspeichern. Man legt die Werte meist auf dem Prozessor Stack ab. Z.B:

```
am Anfang: PUSH R5
           PUSH R6
...
am Ende:   POP R6
           POP R5
```

- Man verläßt die Assembler Routine mit einer "RET" Anweisung. Zu diesem Zeitpunkt muss der Prozessorstack wieder in dem Zustand sein, wie er vor dem Aufruf war. Auch die Inhalte der zu si-

chernden Register muss wiederhergestellt sein.

- Das Debugging funktioniert nur im Bytecode Interpreter, ein Debuggen in Assembler ist nicht möglich.
- Der Bytecode Interpreter hat eine feste Speichereinteilung. Auf **keinen** Fall Assembler Befehle für Datensegmente wie .**byte**, .**db**, .**dw**, .**dseg** oder ähnliches benutzen. Dies würde den Assembler bei Zugriff auf diese Datensegmente dazu bringen Speicher zu überschreiben, der vom Bytecode Interpreter genutzt wird. Benötigt man globale Variablen, so sollte man diese in CompactC oder Basic deklarieren, und dann wie im Kapitel Datenzugriff beschrieben darauf zugreifen.
- **Nicht** mit .org die Adresse Assembler Routine festlegen. Die IDE generiert beim Aufruf des AVRA Assemblers selber einen .org Befehl, der eingehalten werden muss.

4.5 ASCII Tabelle

ASCII Tabelle				
CHAR	**DEC**	**HEX**	**BIN**	**Description**
NUL	000	000	00000000	Null Character
SOH	001	001	00000001	Start of Header
STX	002	002	00000010	Start of Text
ETX	003	003	00000011	End of Text
EOT	004	004	00000100	End of Transmission
ENQ	005	005	00000101	Enquiry
ACK	006	006	00000110	Acknowledgment
BEL	007	007	00000111	Bell
BS	008	008	00001000	Backspace
HAT	009	009	00001001	Horizontal TAB
LF	010	00A	00001010	Line Feed
VT	011	00B	00001011	Vertical TAB
FF	012	00C	00001100	Form Feed
CR	013	00D	00001101	Carriage Return
SO	014	00E	00001110	Shift Out
SI	015	00F	00001111	Shift In
DLE	016	010	00010000	Data Link Escape
DC1	017	011	00010001	Device Control 1

DC2	018	012	00010010	Device Control 2
DC3	019	013	00010011	Device Control 3
DC4	020	014	00010100	Device Control 4
NAK	021	015	00010101	Negative Acknowledgment
SYN	022	016	00010110	Synchronous Idle
ETB	023	017	00010111	End of Transmission Block
CAN	024	018	00011000	Cancel
EM	025	019	00011001	End of Medium
SUB	026	01A	00011010	Substitute
ESC	027	01B	00011011	Escape
FS	028	01C	00011100	File Separator
GS	029	01D	00011101	Group Separator
RS	030	01E	00011110	Request to Send, Record Separator
US	031	01F	00011111	Unit Separator
SP	032	020	00100000	Space
!	033	021	00100001	Exclamation Mark
“	034	022	00100010	Double Quote
#	035	023	00100011	Number Sign
$	036	024	00100100	Dollar Sign
%	037	025	00100101	Percent
&	038	026	00100110	Ampersand
‘	039	027	00100111	Single Quote
(	040	028	00101000	Left Opening Parenthesis
)	041	029	00101001	Right Closing Parenthesis
*****	042	02A	00101010	Asterisk
+	043	02B	00101011	Plus
,	044	02C	00101100	Comma
-	045	02D	00101101	Minus or Dash
.	046	02E	00101110	Dot

CHA	DEC	HEX	BIN	Description

R				
/	047	02F	00101111	Forward Slash
0	048	030	00110000	
1	049	031	00110001	
2	050	032	00110010	
3	051	033	00110011	
4	052	034	00110100	
5	053	035	00110101	
6	054	036	00110110	
7	055	037	00110111	
8	056	038	00111000	
9	057	039	00111001	
:	058	03A	00111010	Colon
;	059	03B	00111011	Semi-Colon
<	060	03C	00111100	Less Than
=	061	03D	00111101	Equal
>	062	03E	00111110	Greater Than
?	063	03F	00111111	Question Mark
@	064	040	01000000	AT Symbol
A	065	041	01000001	
B	066	042	01000010	
C	067	043	01000011	
D	068	044	01000100	
E	069	045	01000101	
F	070	046	01000110	
G	071	047	01000111	
H	072	048	01001000	
I	073	049	01001001	
J	074	04A	01001010	
K	075	04B	01001011	
L	076	04C	01001100	

M	077	04D	01001101	
N	078	04E	01001110	
O	079	04F	01001111	
P	080	050	01010000	
Q	081	051	01010001	
R	082	052	01010010	
S	083	053	01010011	
T	084	054	01010100	
U	085	055	01010101	
V	086	056	01010110	
W	087	057	01010111	
X	088	058	01011000	
Y	089	059	01011001	
Z	090	05A	01011010	
[	091	05B	01011011	Left Opening Bracket
****	092	05C	01011100	Back Slash
]	093	05D	01011101	Right Closing Bracket
^	094	05E	01011110	Caret

CHAR	DEC	HEX	BIN	Description
_	095	05F	01011111	Underscore
`	096	060	01100000	
a	097	061	01100001	
b	098	062	01100010	
c	099	063	01100011	
d	100	064	01100100	
e	101	065	01100101	
f	102	066	01100110	
g	103	067	01100111	
h	104	068	01101000	

i	105	069	01101001	
j	106	06A	01101010	
k	107	06B	01101011	
l	108	06C	01101100	
m	109	06D	01101101	
n	110	06E	01101110	
o	111	06F	01101111	
p	112	070	01110000	
q	113	071	01110001	
r	114	072	01110010	
s	115	073	01110011	
t	116	074	01110100	
u	117	075	01110101	
v	118	076	01110110	
w	119	077	01110111	
x	120	078	01111000	
y	121	079	01111001	
z	122	07A	01111010	
{	123	07B	01111011	Left Opening Brace
|	124	07C	01111100	Vertical Bar
}	125	07D	01111101	Right Closing Brace
~	126	07E	01111110	Tilde
DEL	127	07F	01111111	Delete

Kapitel 5

5 Bibliotheken

In diesem Teil der Dokumentation sind alle mitgelieferten Hilfsfunktionen beschrieben, die es dem Benutzer ermöglichen komfortabel auf die Hardware zuzugreifen. Am Anfang wird für jede Funktion die Syntax für CompactC und BASIC dargestellt. Dann folgt eine Beschreibung der Funktion und der beteiligten Parameter.

5.1 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, daß 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!

5.2 Allgemein

In diesen Bereich fallen Allgemeine Funktionen die sich nicht weiter in Kategorien fassen lassen.

5.2.1 AbsDelay

Allgemeine Funktionen

Syntax

```
void AbsDelay(word ms);

Sub AbsDelay(ms As Word);
```

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.

➔ Bei der AVR32Bit Unit wird bei AbsDelay auch intern Thread_Delay aufgerufen. Es wird sichergestellt werden, das auf Ethernetpakete direkt reagiert wird.

Beim arbeiten mit Threads immer Thread_Delay und nicht AbsDelay benutzen. Wird trotzdem z.B. ein AbsDelay(1000) benutzt, so tritt folgender Effekt auf: Da der Thread erst nach 5000 Zyklen (Default Wert) zum nächsten Thread wechselt, würde der Thread 5000 * 1000ms (5000 Sek.) laufen, bis der nächste Thread anfangen könnte zu arbeiten.

Parameter

ms Wartezeit in ms

5.2.2 ForceBootloader (AVR32Bit)

Allgemeine Funktionen

Syntax

```
void ForceBootloader(void);

Sub ForceBootloader();
```

Beschreibung

Springt in den Bootloader. Danach ist die Unit wieder zugänglich für Kommandos, um z.B. die Software upzudaten.

Parameter

Keine

5.2.3 Sleep (Mega)

Allgemeine Funktionen

Syntax

```
void Sleep(byte ctrl);

Sub Sleep(ctrl As Byte)
```

Beschreibung

Mit dieser Funktion läßt sich die Atmel CPU in eine der 6 verschiedenen Sleep Modi bringen. Die exakte Funktionalität wird im Atmel Mega Reference Manual im Kapitel "Power Management and Sleep Modes" beschrieben. Der Wert von ctrl wird in die Bits *SM0* bis *SM2* geschrieben. Das *sleep enable* Bit (*SE* in **MCUCR**) wird gesetzt und eine Assembler *sleep* Instruktion wird ausgeführt.

Parameter

ctrl Initialisierungsparameter (*SM0* bis *SM2*)

Sleep Modes

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby
1	1	1	Extended Standby

5.3 Analog-Comparator

5.3.1 Mega

Der Analog-Comparator ermöglicht, zwei analoge Signale zu vergleichen. Das Ergebnis dieses Vergleichs wird entweder als „0" oder „1" zurückgegeben. Es können an den positiven und negativen Eingängen, Spannungen zwischen 0 und 5V verglichen werden.

5.3.1.1 AComp

AComp Funktionen **Beispiel**

Syntax

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

```
Sub AComp(mode As Byte);
```

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 **Mega32**: AIN1 (PortB.3), **Mega128**: AIN1 (PortE.3). Der positive Eingang kann entweder **Mega32**: AIN0 (PortB.2), **Mega128**: AIN0 (PortE.2) sein, oder eine interne Referenzspannung von 1,22V.

Parameter

mode Arbeitsmodus

Moduswerte:

00 (Hex)	externe Eingänge (+)AIN0 und (-)AIN1 werden verwendet
40 (Hex)	externer Eingang (-)AIN1und interne Referenzspannung werden verwendet
80 (Hex)	Analog-Comparator wird abgeschaltet

5.3.1.2 AComp Beispiel

Beispiel: Verwendung des Analog-Comparators

```
// AComp: Analog Comparator
// Mega32:  Eingang (+) PB2 (PortB.2) bzw. band gap reference 1,22V
//          Eingang (-) PB3 (PortB.3)
// Mega128: Eingang (+) PE2 (PortE.2) bzw. band gap reference 1,22V
//          Eingang (-) PE3 (PortE.3)
// erforderliche Library: IntFunc_Lib.cc

// Die Funktion AComp gibt den Wert des Komparators zurück.
// Ist die Spannung am Eingang PB2/PE2 größer als am Eingang PB3/PE3
// hat die Funktion AComp den Wert 1.
// Mode:
// 0x00  externe Eingänge (+)AIN0 und (-)AIN1 werden verwendet
// 0x40  externer Eingang (-)AIN1 und interne Referenzspannung
// 0x80  Analog-Comparator wird abgeschaltet
// Der Aufruf kann mit dem Parameter 0 (beide Eingänge werden
// verwendet) oder 0x40 (interne Referenzspannung am (+) Eingang,
// externer Eingang PB3/PE3) erfolgen.


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

5.3.2 AVR32Bit

Der Analog-Comparator ermöglicht, zwei analoge Signale zu vergleichen. Das Ergebnis dieses Vergleichs wird entweder als „0" oder „1" zurückgegeben. Es können an den positiven und negativen Eingängen, Spannungen zwischen 0 und 3.3V verglichen werden.

5.3.2.1 AC_Disable

Analog Compare Funktionen Beispiel

Syntax

```
void AC_Disable(byte ctrl);

Sub AC_Disable(ctrl As Byte);
```

Beschreibung

Schaltet den betreffenden Analog Comparator ab.

Parameter

ctrl Analog Comparator (0 - 1)

5.3.2.2 AC_Enable

Analog Compare Funktionen Beispiel

Syntax

```
void AC_Enable(byte ctrl, byte in_pos, byte in_neg);

Sub AC_Enable(ctrl As Byte, in_pos As Byte, in_neg As Byte);
```

Beschreibung

Schaltet den Analog Comparator ein. Der Atmel AVR32 hat 2 Analog Comparatoren. Die Tabelle gibt Auskunft welche Porteingänge für welchen Comparator (Ctrl 0, Ctrl 1) und für die Eingänge in_pos oder in_neg genutzt werden dürfen. Nur Eingänge auf dem gleichen Comparator können verglichen werden, aber es dürfen beide Comparatoren gleichzeitig benutzt werden. Für den Chip ist eine Hysterese von 0 eingestellt.

➔ Aufgrund der eingesetzten TQFP100 Version des Prozessors hat Analog Comparator 0 weniger Auswahlmöglichkeiten für die Wahl des Eingangspins als Comparator 1.

Parameter

ctrl Analog Comparator (0 - 1)
in_pos Eingang V_ip
in_neg Eingang V_in

Tabelle Analog Comparator Pin Auswahl

#define	Port	Wert	Ctrl 0	Ctrl 1
AC_AC0AP0	P18	0	in_pos	-
AC_AC0AN0	P20	1	in_neg	-
AC_AC0BP0	P21	2	in_neg	-
AC_AC1AP0	P25	0	-	in_pos
AC_AC1AP1	P11	1	-	in_pos
AC_AC1AN0	P24	2	-	in_neg
AC_AC1AN1	P12	3	-	in_neg
AC_AC1BP0	P26	4	-	in_neg
AC_AC1BP1	P13	5	-	in_neg

5.3.2.3 AC_InpHigher

Analog Compare Funktionen **Beispiel**

Syntax

```
byte AC_InpHigher(byte ctrl);

Sub AC_Disable(ctrl As Byte) As Byte;
```

Beschreibung

Gibt zurück ob die Spannung von in_pos größer ist als die Spannung von in_neg.

Parameter

ctrl Analog Comparator (0 - 1)

Rückgabewert

Ungleich Null, wenn in_pos größer als in_neg.

5.3.2.4 AC Beispiel

```
// AVR32Bit Analog Comparator Example
void main(void)
{
    AC_Enable(0, AC_AC0AP0, AC_AC0AN0);

    while(1)
    {
        if(AC_InpHigher(0)) Msg_WriteText("AC_AC0AP0 > AC_AC0AN0\r");
        else Msg_WriteText("AC_AC0AP0 < AC_AC0AN0\r");

        AbsDelay(500); // 500ms delay
    }
}
```

5.4 Analog-Digital-Wandler

5.4.1 Mega

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 (Port A.0 bis A.7 bei **Mega32**, Port F.0 bis F.7 bei **Mega128)**, 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, erzeugt durch z.B. ein Referenzspannungs-IC, dann entspricht eine Differenz von einem Bit des digitalisierten Meßwertes einer Spannungsdifferenz von 4mV oder :

*u = x * 0,004V*

➔ Das Messergebnis einer A/D Wandlung kann verfälscht werden, wenn während der Messung, auf dem gleichen Port wie der A/D Kanal, der Zustand von irgendeinem Portbit geändert wird, das auf Ausgang geschaltet ist.

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*

5.4.1.1 ADC_Disable

ADC Funktionen

Syntax

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

```
Sub ADC_Disable()
```

Beschreibung

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

Parameter

Keine

5.4.1.2 ADC_Read

ADC Funktionen

Syntax

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

```
Sub ADC_Read() As Word
```

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

5.4.1.3 ADC_ReadInt

ADC Funktionen

Syntax

```
word ADC_ReadInt(void);

Sub ADC_ReadInt() As Word
```

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

5.4.1.4 ADC_Set

ADC Funktionen

Syntax

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

Sub ADC_Set(v_ref As Byte, channel As Byte) As Word
```

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.

➔ Das Messergebnis einer A/D Wandlung kann verfälscht werden, wenn während der Messung, auf dem gleichen Port wie der A/D Kanal, der Zustand von irgendeinem Portbit geändert wird, das auf Ausgang geschaltet ist.

Parameter

channel Portnummer (0..7) des ADC (Port A.0 bis A.7 bei **Mega32**, Port F.0 bis F.7 bei **Mega128)**
v_ref Referenzspannung (siehe Tabelle)

Name	Wert (Hex)	Beschreibung
ADC_VREF_BG	C0	2,56V interne Referenzspannung
ADC_VREF_VCC	40	Versorgungsspannung (5V)
ADC_VREF_EXT	00	externe Referenzspannung an PAD3

Für den Standort von PAD3 siehe Jumper Application Board M32 oder M128.

5.4.1.5 ADC_SetInt

ADC Funktionen

Syntax

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

Sub ADC_SetInt(v_ref As Byte, channel As Byte) As Word
```

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 .

➔ Das Messergebnis einer A/D Wandlung kann verfälscht werden, wenn während der Messung, auf dem gleichen Port wie der A/D Kanal, der Zustand von irgendeinem Portbit geändert wird, das auf Ausgang geschaltet ist.

Parameter

channel Portnummer (0..7) des ADC (Port A.0 bis A.7 bei **Mega32**, Port F.0 bis F.7 bei **Mega128)**
v_ref Referenzspannung (siehe Tabelle)

Name	Wert	Beschreibung
ADC_VREF_BG	C0 (Hex)	2,56V interne Referenzspannung
ADC_VREF_VCC	40 (Hex)	Versorgungsspannung (5V)
ADC_VREF_EXT	00 (Hex)	externe Referenzspannung an PAD3

Für den Standort von PAD3 siehe Jumper Application Board M32 oder M128.

5.4.1.6 ADC_StartInt

ADC Funktionen

Syntax

```
void ADC_StartInt(void);

Sub ADC_StartInt()
```

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

5.4.2 AVR32Bit

Der Mikrocontroller verfügt über einen Analog-Digital-Wandler mit einer umschaltbaren Auflösung von 8/10/12 Bit. Das heißt, gemessene Spannungen können maximal als ganze Zahlen von -2048 bis 2048 dargestellt werden, da der A/D-Wandler immer differentiell arbeitet. Zudem kann die Verstärkung des ADC-Vorverstärkers von 1, 2, 4, 8, 16, 32, 64 per Software eingestellt werden.

Folgende Referenzspannungsquellen stehen zur Verfügung:

- 0,6 * VDDANA intern (0,6 * 3.3V = 1,98V)
- interne Referenzspannung von 1V
- zwei externe Referenzspannungseingänge, z.B. 2.048V, durch Referenzspannungs-IC erzeugt

Ist „x“ ein digitaler Messwert, dann errechnet sich der entsprechende Spannungswert „u“ wie folgt:
Die Auflösung ist von der Konfiguration des ADCs abhängig.

Auflösung	Maximaler Wert
8 Bit	-128 bis +127
10 Bit	-512 bis +511
12 Bit	-2048 bis +2047

Formel zur Berechnung der anliegenden ADC-Spannung:
u = x * Referenzspannung / Auflösung

5.4.2.1 ADC_Disable

ADC Funktionen

Syntax

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

```
Sub ADC_Disable()
```

Beschreibung

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

Parameter

Keine

5.4.2.2 ADC_Enable

ADC Funktionen

Syntax

```
void ADC_Enable(byte mode, dword speed, byte ref, byte input_cnt, char offset);
```

```
Sub ADC_Enable(mode As Byte, speed As ULong, ref As Byte, input_cnt As Byte, offset As Char)
```

Beschreibung

Der ADC-Sequencer im AVR32 kann bis zu 8 AD-Konvertierungen am Stück durchführen. Es kann bei einer AD-Konvertierung eine Differenzmessung zwischen einem ADC Pin und GND durchgeführt werden, oder eine Differenzmessung zwischen zwei Pins. Siehe ADC_SetInput.

Im mode Parameter können verschiedene Eigenschaften oderiert werden (dabei macht natürlich nur eine ADC Auflösung Sinn). Oversampling und Sample & Hold kann ausgeschaltet werden. Falls aktiviert, wird ein Interrupt ausgelöst, wenn eine ADC Messung fertig ist (siehe Interrupt Tabelle).

Für jede neue Messung wird ADC_Start aufgerufen. Das Ende der Messung kann über den Interrupt angezeigt werden, oder man benutzt ADC_GetValues mit dem **ADC_GET_WAIT** Parameter. Ist der Free Running Mode selektiert, wird ADC_Start nur einmal aufgerufen, danach wird kontinuierlich gemessen, und ADC_GetValues liefert immer die Werte der letzten Messung.

➔ Für die genaue Bedeutung von Oversampling und Sample & Hold und die Auswirkungen auf die Messungen bitte in das Datenblatt des AT32UC3C schauen.

➔ Zu hohe ADC Geschwindigkeiten können den Interpreter überfordern, wenn der Interrupt aktiviert ist.

Parameter

mode	Arbeitsmodi (siehe Tabelle)
speed	ADC Clock (32khz - 1.5Mhz)
ref	Referenzspannung (siehe Tabelle)
input_cnt	Anzahl der zu messenden Eingänge (1-8)
offset	Korrekturfaktor (-128 bis 127)

Mode Tabelle

Definition	Funktion
ADC_MODE_12BIT	ADC 12-Bit Auflösung
ADC_MODE_8BIT	ADC 8-Bit Auflösung
ADC_MODE_10BIT	ADC 10-Bit Auflösung
ADC_MODE_NO_OVERSAMP	Schaltet Oversampling aus
ADC_MODE_ENAB_IRQ	Aktiviert ADC IRQ
ADC_MODE_NO_SAMPHOLD	Kein Sample & Hold
ADC_MODE_FREE_RUN	Aktiviert Free Running

Referenzspannung Tabelle

Definition	Funktion
ADC_REF1V	Interne 1V Referenz
ADC_REF06VDD	Interne 0.6 x VDDANA Referenz
ADC_ADCREF0	Externe ADCREF0 Referenz
ADC_ADCREF1	Externe ADCREF1 Referenz

5.4.2.3 ADC_GetValue

ADC Funktionen

Syntax

```
int ADC_GetValue(byte indx);

Sub ADC_GetValue(indx As Byte) As Integer
```

Beschreibung

Die Funktion liest einen gemessenen Wert aus dem A/D-Wandler. Der Parameter indx korrespondiert mit dem Eintrag im inputs Array bei ADC_Enable(). Wird zu `indx` der Wert **ADC_GET_WAIT** (80 Hex) oderiert, dann wird erst auf die Beendigung aller ADC Messungen gewartet, bevor der Wert zurückgegeben wird.

➔ Die **ADC_GET_WAIT** Funktionalität sollte nicht im "Free Running" Modus benutzt werden, oder wenn der ADC abgeschaltet ist.

Parameter

indx	Index des gemessenen A/D Werts

Rückgabewert

gemessener A/D Wert

5.4.2.4 ADC_GetValues

ADC Funktionen

Syntax

```
void ADC_GetValues(int values[], byte cnt);

Sub ADC_GetValues(Byref values As Integer, cnt As Byte)
```

Beschreibung

Die Funktion liest die gemessenen Werte aus dem A/D-Wandler und kopiert sie in ein 16-Bit Array. Wird zu cnt der Wert **ADC_GET_WAIT** (80 Hex) oderiert, dann wird erst auf die Beendigung aller ADC Messungen gewartet, bevor die Werte kopiert werden.

➔ Die **ADC_GET_WAIT** Funktionalität sollte nicht im "Free Running" Modus benutzt werden, oder wenn der ADC abgeschaltet ist.

Parameter

values	Zeiger auf das 16-Bit Array (0-7)
cnt	Anzahl der Werte die in das Array kopiert werden

5.4.2.5 ADC_SetInput

ADC Funktionen

Syntax

```
void ADC_SetInput(byte indx, byte inp1, byte inp2, byte gain);

Sub ADC_Enable(indx As Byte, inp1 As Byte, inp2 As Byte, gain As Byte)
```

Beschreibung

Der ADC-Sequencer im AVR32 kann bis zu 8 AD-Konvertierungen am Stück durchführen. Die Funktion ADC_SetInput definiert die ADC Eingänge zwischen denen eine Differenzmessung durchgeführt wird. Möchte man nur einen Eingang messen, nimmt man ADC_GND als zweiten Eingang. Zusätzlich kann ein GAIN Faktor definiert werden.

➔ Auch wenn eine Messung zwischen einem Eingang und ADC_GND nur positive Werte liefert, so bleibt trotzdem ein Bit der ADC Auflösung dem Vorzeichen reserviert.

Parameter

indx	Index für die Konvertierung (0-7)
inp1	Erster AD Eingang (0-15)
inp2	Zweiter AD Eingang (0-15)

gain GAIN Faktor

GAIN Tabelle

Definition	Funktion
ADC_SHG_1	Gain Faktor 1
ADC_SHG_2	Gain Faktor 2
ADC_SHG_4	Gain Faktor 4
ADC_SHG_8	Gain Faktor 8
ADC_SHG_16	Gain Faktor 16
ADC_SHG_32	Gain Faktor 32
ADC_SHG_64	Gain Faktor 64

5.4.2.6 ADC_Start

ADC0**ADC Funktionen**

Syntax

```
void ADC_Start(void);

Sub ADC_Start()
```

Beschreibung

Der eingebaute A/D Wandler beginnt Analog Daten zu konvertieren.

Parameter

Keine

5.4.2.7 ADC Beispiel

```
// Programm zum Auslesen der gemessenen Werte von zwei ADC Pins
void main(void)
{
    int result[2];
    char str[40];

    ADC_Disable();

    ADC_SetInput(0, 2, ADC_GND, ADC_SHG_1);  // aktiviere ADC2 - Gain 1
    ADC_SetInput(1, 5, ADC_GND, ADC_SHG_4);  // aktiviere ADC5 - Gain 4

    // 12Bit ADC, free running, 1MHz Abtastfreq., Referenz 1V, Offset 0
    ADC_Enable(ADC_MODE_12BIT | ADC_MODE_FREE_RUN, 1000000, ADC_REF1V,
```

```
        2, 0);
    ADC_Start();

    while(1)
    {
        ADC_GetValues(result, 2); // Werte auslesen

        Str_Printf(str, "adc2: %d\r", result[0]);
        Msg_WriteText(str);
        Str_Printf(str, "adc5: %d\r", result[1]);
        Msg_WriteText(str);
        AbsDelay(300);
    }
}
```

5.5 CAN Bus

Der CAN-Bus (engl. Controller Area Network) ist ein asynchrones, serielles Bussystem und gehört zu den Feldbussen. Er ist nach ISO 11898 international standardisiert und definiert die Layer 1 (physikalische Schicht) und 2 (Datensicherheitsschicht).

Der CAN-Bus wurde 1983 bei der Fa. Bosch entwickelt. Ursprünglich wurde der CAN-Bus für den Automobilsektor entwickelt, da mit zunehmender Elektronik im Fahrzeug die Kabelbäume immer größer wurde und eine Lösung zur Gewichts-/ und Kostenreduzierung gefunden werden musste. Mittlerweile wird dieses Erfolgreiche und sehr sichere Konzept nicht nur in der Automobilindustrie eingesetzt, sonder auch in den Bereichen: Automatisierung, Flugzeugbau, Raumfahrt und auch in der Medizintechnik.

➔ Das C-Control Handbuch kann aufgrund der Komplexität keine Einführung in den CAN Standard bieten. Vorwissen über den CAN Standard und die Full CAN Message Objects werden an dieser Stelle vorausgesetzt. Deshalb ist die Arbeit mit dem CAN-Bus nicht direkt für Anfänger im Bereich der Embedded Controller zu empfehlen. Eine gute Zusammenfassung über CAN und Message Objects bietet das "Atmel AT90CAN" Reference Manual Kapitel 19, "Controller Area Network - CAN".

MEGA128CAN

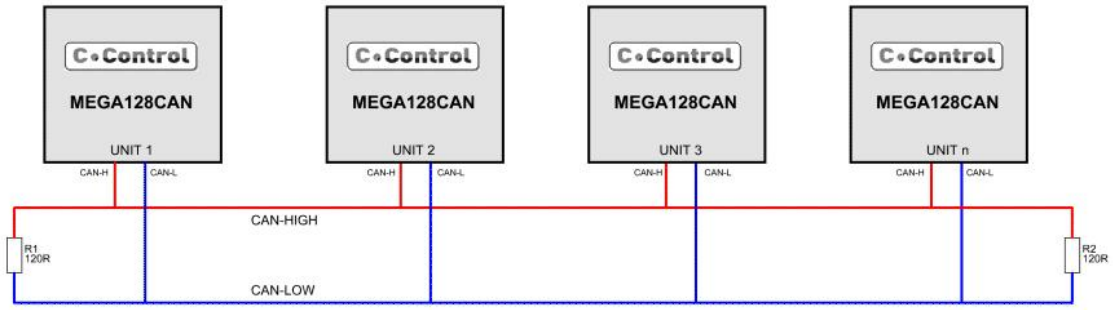
Die CAN-Signale der C-Control Pro MEGA128CAN stehen an den Pins X4_13 (CANL) und X4_14 (CANH) zur Verfügung.

AVR32Bit

Im C-Control Pro AVR32 arbeitet ein CAN Controller mit zwei Kanälen. Aber nur am ersten Kanal ist ein Transceiver angeschlossen, der auch auf dem Applicationboard herausgeführt ist. Auf dem Mainboard ist CAN1 über eine Buchsenleiste herausgeführt (ohne Transceiver). Die Leitungen CANH und CANL sind beim Modul am Connector X1 herausgeführt. Um den zweiten Kanal nutzen zu können, muss der Anwender selber einen Transceiver anschließen. Als Beispiel kann das Datenblatt des AVR32 Moduls dienen. Der 2. Controller liegt auf Port 1 (CAN_TX, PA00) und Port 2 (CAN_RX, PA01).

Netzwerk

Sie können mehrere CAN-Bus Teilnehmer über die Pins CAN-H und CAN-L vernetzen. Der erste und der letzte Teilnehmer muss mit einen 120Ohm Widerstand abgeschlossen werden. Als Datenkabel sollte ein verdrilltes Kabel (engl. Twist and pair) zur Verwendung kommen. Für kürzere Strecken von wenigen Zentimetern bis max. 2 Meter, kann auch ein einfaches Parallelkabel (engl. Twin lead) verwendet werden.



Die UNIT unterstützt den Low-/ sowie Highspeed Bus (MEGA128CAN 10 kbit/s bis 1 Mbit/s, AVR32Bit 50 kbit/s bis 1 Mbit/s). Die Theoretischen Leitungslängen je nach Busgeschwindigkeit entnehmen Sie der unten aufgeführten Tabelle.

Geschwindigkeit	Leitungslänge
1 Mbit/s	40 m
Bis 500 kbit/s	100 m
Bis 125 kbit/s	500 m
Kleiner 125 kbit/s	Bis zu 1000 m

Die Leitungslängen sind stark abhängig von den verwendeten Leitungen und Anzahl der Teilnehmer. Es ist möglich ein „Twist-Pair-Kabel“ mit einem Wellenwiderstand von 108 bis 132 Ohm zu verwenden. Es können maximal 32 UNITs an einem Bus betrieben werden. Bei der Inbetriebnahme des Busses beginnt man am besten bei der, passend zur Kabellänge, theoretischen maximal zulässigen Geschwindigkeit und senkt diese ab, wenn keine Übertragung stattfindet oder zu viele Fehler (Paketfehler) auftreten.

Der Controller unterstützt das “Base frame format” CAN 2.0A (11 Bit-Identifier) und das Extended frame format” CAN 2.0B (29 Bit-Identifier).

Um den CAN Bus in eigenen Projekten einsetzen zu können, ist es unabdingbar das CAN Datenformat und die technischen Details des CAN Bus zu verstehen. Hintergrundinformationen sind in Büchern und in der Wikipedia zu finden: http://de.wikipedia.org/wiki/Controller_Area_Network

Message Objects

Der aktive CAN Bus arbeitet mit 15 (MEGA128CAN) oder 16 (AVR32Bit) unabhängigen Message Objects (MOb) mit denen man Nachrichten mit bestimmten Identifiern senden und empfangen kann. Hierzu werden mit CAN_SetMOb () die Message Objects auf die entsprechende Operation parametrisiert.

➔ Message Objects mit einer niedrigen MOb Nummer haben immer Vorrang vor einer höheren

MOb Nummer. Wenn man zwei MObs hat, die eine bestimmtes Paket empfangen würden, wird es immer von dem Message Object empfangen was die niedrigere MOb Nummer hat.

CAN Protokoll

Der CAN Bus Controller kann gleichzeitig normale Pakete (CAN 2.0A) und erweiterte Pakete (CAN 2.0B) verarbeiten. CAN Bus Identifier werden als 32-Bit dword (ULong) übergeben. Je nach Typ der Pakete ist ein Identifier 11-Bit (V2.0 part A) oder 29-Bit lang (V2.0 part B). Die ungenutzten Bits werden dabei ignoriert. Die maskID bestimmt, welche Pakete bei einem bestimmten Identifier (ID) empfangen werden. Nur die Bits in der maskID die eins sind, werden bei einem Bitvergleich zwischen eingestelltem Identifier und der ID des eingehenden Paketes überprüft.

automatic reply

Ist ein Message Object auf automatic reply gestellt, so übernimmt der MOb den Data Length Code (DLC) von dem eingehenden Remote Trigger Paket. D.h. der Sender des Trigger Paketes bestimmt über den mitgesendeten DLC die Anzahl der Daten Bytes die in dem Reply Paket gesendet werden.

Message FIFO

Bei der Initialisierung der CAN Bibliothek stellt der Benutzer RAM für einen Message FIFO zur Verfügung, in dem alle eingehenden CAN Pakete gespeichert werden. Damit kann man dann die empfangenen Nachrichten asynchron entgegen nehmen.

5.5.1 CAN Beispiele

In diesem Kapitel werden einige Initialisierungsbeispiele gegeben, um die Funktionsweise der CAN Bibliothek zu verdeutlichen.

Initialisierung

In jedem Fall muss die CAN Bibliothek vor dem Betrieb initialisiert werden. Dieses Beispiel stellt für den CAN Bus eine Geschwindigkeit von 1 Mega bps ein, und stellt RAM für 10 FIFO Einträge zur Verfügung.

```
byte fifo_buf[CAN_BUF(10)];

CAN_Init(CAN_1MBPS, 10, fifo_buf);
```

Empfang

1. Auf MOb Nummer 2 werden CAN 2.0A Nachrichten empfangen, die exakt einen Identifier von 0x123 haben.

```
CAN_SetMOb(2, 0x123, 0x7ff, CAN_RECV);
```

2. Auf MOb Nummer 3 werden CAN 2.0B Nachrichten empfangen, die exakt einen Identifier von 0x12345 haben.

```
CAN_SetMOb(3, 0x12345, 0x1fffffff, CAN_RECV|CAN_EXTID);
```

3. Auf MOb Nummer 3 werden CAN2.0A und CAN 2.0B Nachrichten empfangen, da das CAN_IGN_EXTID flag gesetzt ist. Durch die maskID von null, werden Nachrichten mit allen Identifiern empfangen. Wegen CAN_IGN_RTR werden normale und Trigger Pakete empfangen.

```
CAN_SetMOb(3, 0x12345, 0, CAN_RECV|CAN_IGN_EXTID|CAN_IGN_RTR);
```

4. Auf MOb Nummer 2 werden CAN 2.0A Nachrichten empfangen, die einen Identifier von 0x120, 0x121, 0x122 oder 0x123 haben.

```
CAN_SetMOb(2, 0x120, 0x7fc, CAN_RECV);
```

Senden

1. Auf MOb Nummer 0 wird eine CAN 2.0A Nachricht mit ID 0x432 und 6 Datenbyte gesendet.

```
byte data[8], i;

for(i=0;i<8;i++) data[i]=i;
CAN_SetMOb(0, 0x432, 0, CAN_SEND);
CAN_MObSend(0, 6, data);
```

2. Auf MOb Nummer 1 wird eine CAN 2.0B Nachricht mit ID 0x12345678 und 8 Datenbyte gesendet.

```
byte data[8], i;

for(i=0;i<8;i++) data[i]=i;
CAN_SetMOb(1, 0x12345678, 0, CAN_SEND|CAN_EXTID);
CAN_MObSend(1, 8, data);
```

Automatic Reply

MOb Nummer 4 wird auf automatic reply eingestellt. Die mit CAN_MObSend() übergebenen Datenbytes werden gesendet, wenn eine CAN 2.0B Trigger Nachricht mit ID von 0x999 empfangen wird. Die Anzahl der gesendeten Datenbytes hängt vom DLC der eingehenden Trigger Nachricht ab.

```
byte data[5], i;

for(i=0;i<5;i++) data[i]=i;
CAN_SetMOb(4, 0x999, 0x1fffffff, CAN_REPL|CAN_EXTID);
CAN_MObSend(4, 5, data);
```

5.5.2 CAN_Exit

CAN Bus Funktionen

Syntax

```
void CAN_Exit(void);
```

```
Sub CAN_Exit()
```

Beschreibung

Die CAN Chipfunktionen werden ausgeschaltet.

5.5.3 CAN_GetInfo

CAN Bus Funktionen

Syntax

```
byte CAN_GetInfo(byte infotype);
```

```
Sub CAN_GetInfo(infotype As Byte) As Byte
```

Beschreibung

Gibt Informationen über die Anzahl der empfangenen CAN Nachrichten und CAN Fehler zurück.

Parameter

infotype gewünschte CAN Bus Information

Rückgabewert

CAN Bibliotheks Information

infotype Parameter:

Wert	Definition	Bedeutung
1	CAN_MSGS	Anzahl der schon empfangenen CAN Nachrichten im FIFO
2	CAN_ERR_RECV	Anzahl der CAN Empfangsfehler (max. 255)
3	CAN_ERR_TRAN	Anzahl der CAN Sendefehler (max. 255)

5.5.4 CAN_Init

CAN Bus Funktionen

Syntax

```
void CAN_Init(byte speed, byte fifo_len, byte fifo_addr[]);

Sub CAN_Init(speed As Byte, fifo_len As Byte, ByRef fifo_addr As Byte);
```

Beschreibung

Initialisiert die CAN Funktionen. Bei der Initialisierung wird vom Benutzer ein RAM Puffer für den Empfang von CAN Nachrichten zur Verfügung gestellt, in dem fifo_len Nachrichten gespeichert werden können. Für die Größe des RAM Bereiches gibt es ein #define **CAN_BUF**. Möchte man ein byte Array definieren mit Platz für fifo_len Nachrichten, schreibt man `"byte buf[CAN_BUF(fifo_len)];"` (siehe auch Beispiele). Ist der FIFO voll, werden ankommende CAN Nachrichten nicht gespeichert.

➔ Der vom Benutzer zur Verfügung gestellte RAM Puffer muss während der Benutzung der CAN Schnittstelle reserviert bleiben. Da lokale Variablen nach Verlassen der Funktion freigegeben werden, ist es meistens sinnvoll den Puffer als globale Variable zu deklarieren.

Parameter

speed	CAN Bus Geschwindigkeit
fifo_len	Anzahl der Einträge im Empfangsfifo
fifo_addr	Benutzerspeicher für den Empfangspuffer

speed Parameter:

Wert Mega	Wert AVR32Bit (Hex)	Definition	CAN Baudrate
0	-	CAN_10KBPS	10.000bps
1	-	CAN_20KBPS	20.000bps
2	-	CAN_40KBPS	40.000bps
-	e1b3c	CAN_50KBPS	50.000bps
3	e1b1d	CAN_100KBPS	100.000bps
4	e1b17	CAN_125KBPS	125.000bps
5	e1b0d	CAN_200KBPS	200.000bps
6	e1b0b	CAN_250KBPS	250.000bps
7	e1b05	CAN_500KBPS	500.000bps
8	c1204	CAN_800KBPS	800.000bps
9	e1b02	CAN_1MBPS	1.000.000bps

5.5.5 CAN_Receive

CAN Bus Funktionen

Syntax

```
byte CAN_Receive(byte data[]);

Sub CAN_Receive(ByRef data As Byte) As Byte
```

Beschreibung

Wenn Nachrichten im Empfangsfifo sind, so wird der 14-Byte Datensatz in ein Array des Benutzers kopiert, welches auch eine Länge von 14-Byte haben muss. Ist bei der IDT in der empfangenen Nachricht Bit 31 gesetzt, so hatte das CAN Paket RTR gesetzt.

Parameter

data Array in das die CAN Nachricht kopiert wird

Rückgabewert

Länge des CAN Pakets (0-8 Byte) oder ff (Hex) wenn kein Paket im Buffer war

Aufbau des Datensatz

Byte 0: MOb Nummer (0-14)
Byte 1-4: 29-Bit IDT (bei V2.0 part A Msgs sind die oberen Bits null)
Byte 5: Länge der CAN Daten (0-8)
Byte 6-13: Paketdaten

5.5.6 CAN_MObSend

CAN Bus Funktionen

Syntax

```
void CAN_MObSend(byte mob, byte len, byte data[]);

Sub CAN_MObSend(mob As Byte, len As Byte, ByRef data As Byte);
```

Beschreibung

Eine CAN Nachricht wird über den Bus gesendet. Wenn aber bei CAN_SetMOb() das CAN_REPL Flag gesetzt wurde, werden die Daten für das Automatic Reply gespeichert, und nicht sofort gesendet.

Parameter

mob MOb Nummer (0-14)
len Länge der zu sendenden Daten

data Array in der die Sendedaten stehen

5.5.7 CAN_SetChan (AVR32Bit)

CAN Bus Funktionen

Syntax

```
void CAN_SetChan(byte chan);
```

```
Sub CAN_SetChan(chan As Byte)
```

Beschreibung

Selektiert einen CAN Kanal (CAN0 oder CAN1) für den weiteren Zugriff.

➔ Der C-Control Pro Mega128 CAN hat nur einen CAN Kanal.

Parameter

chan CAN Bus Kanal (0 - 1)

5.5.8 CAN_SetMOb

CAN Bus Funktionen

Syntax

```
void CAN_SetMOb(byte mob, dword ID, dword maskID, byte flag);
```

```
Sub CAN_SetMOb(mob As Byte, ID As ULong, maskID As ULong, flag As Byte);
```

Beschreibung

Mit dieser Funktion werden die Parameter für eine Message Object (MOb) gesetzt. Der Identifier und die Identifier Maske werden als dword (ULong) übergeben. Bei einem 11-Bit Identifier werden die oberen Bits ignoriert. Die maskID wird nur beim Empfang genutzt. Nur wenn ein Bit in der maskID gesetzt ist, wird beim Nachrichtenempfang an der gleichen Bitposition im Identifier geprüft, ob der empfangene Identifier übereinstimmt.

Parameter

mob MOb Nummer (0-14)
ID Identifier
maskID Identifier Maske
flag Operationsparameter für das Message Object (MOb)

flag Parameter:

Wert (Hex)	Definition	Bedeutung
01	CAN_RECV	Nachrichtenempfang auf diesem MOb
02	CAN_RTR	Das Remote Trigger Bit wird gesetzt
04	CAN_EXTID	Die CAN Nachricht hat eine 29-Bit ID (V2.0 part B)
08	CAN_REPL	Automatic Reply wird initiiert
10	CAN_IGN_RTR	In der ID Maske wird RTR **nicht** gesetzt
20	CAN_IGN_EXTID	In der ID Maske wird IDEMSK **nicht** gesetzt
40	CAN_SEND	Auf diesem MOb soll gesendet werden

5.6 Clock

Mega

Die interne Software Uhr wird durch den 10ms Interrupt von Timer2 getaktet. Es können Uhrzeit und Datum gesetzt werden, die ab diesem Zeitpunkt selbständig weiterlaufen. Schaltjahre werden berücksichtigt. Der Fehler beträgt je nach Quartzungenauigkeit zwischen 4-6 Sekunden pro Tag. Man kann einen Korrekturfaktor in 10ms Ticks angeben, der jede volle Stunde auf den internen Zähler addiert wird.

Beispiel: Hat man eine Abweichung von 9,5 Sek. für 2 Tage gemessen, so muss man eine Abweichung von 9,5 / (2 * 24) = 0,197 Sek. korrigieren. Dies entspricht einem Korrekturfaktor von 20, wenn die Software Uhr nachgeht, oder -20 wenn die Uhr vorläuft.

➔ Wird Timer 2 abgeschaltet oder für einen anderen Zweck gebraucht, so ist die interne Software Uhr nicht funktionsfähig.

AVR32Bit

Im AVR32Bit wird bei den Uhrenfunktionen das eingebaute **Real Time Clock** Modul des AVR32 genutzt. Zudem bietet der externe 32khz Quarz hier einen viel genaueren Taktgeber als der Oszillator der C-Control Pro Mega Units. Deshalb bleibt der Korrekturaktor im C-Control Pro AVR32Bit ungenutzt.

5.6.1 Clock_GetVal

Clock Funktionen

Syntax

```
byte Clock_GetVal(byte indx);

Sub Clock_GetVal(indx As Byte) As Byte
```

Beschreibung

Die einzelnen Werte Zeit- und Datumswerte der internen Software Uhr können ausgelesen werden.

➔ Die Werte von Tag und Monat sind nullbasiert, in einer Ausgabe sollte auf diese Werte eine eins addiert werden.

Parameter

indx Indexparameter des auszulesenden Wertes

#define	Index	Bedeutung
CLOCK_SEC	0	Sekunde
CLOCK_MIN	1	Minute
CLOCK_HOUR	2	Stunde
CLOCK_DAY	3	Tag
CLOCK_MON	4	Monat
CLOCK_YEAR	5	Jahr

Rückgabewert

angeforderter Zeitwert

5.6.2 Clock_SetDate

Clock Funktionen

Syntax

```
void Clock_SetDate(byte day, byte mon, byte year);
```

```
Sub Clock_SetDate(day As Byte, mon As Byte, year As Byte)
```

Beschreibung

Setzt das Datum der internen Software Uhr.

➔ Die Werte von Tag und Monat sind nullbasiert, bei der Angabe muss daher eine eins subtrahiert werden.

Parameter

day Tag
mon Monat
year Jahr

5.6.3 Clock_SetTime

Clock Funktionen

Syntax

```
void Clock_SetTime(byte hour, byte min, byte sec, char corr);
```

```
Sub Clock_SetTime(hour As Byte, min As Byte, sec As Byte, corr As Char)
```

Beschreibung

Setzt die Uhrzeit in der internen Software Uhr. Für eine Beschreibung des Korrekturfaktors siehe Kapitel

Clock.

➔ Der Korrekturfaktor bleibt beim AVR32Bit ungenutzt, man kann dort einen beliebigen Wert angeben.

Parameter

hour	Stunde
min	Minute
sec	Sekunde
corr	Korrekturfaktor

5.7 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 mit einzubinden.

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 Beispielprogramms "DCF_RTC.cc"

Das Programm DCF_RTC.cc ist eine Uhr, die über DCF77 synchronisiert wird. Die Uhrzeit und das Datum werden 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 Eingangsport (**Mega32**: PortD.7 - **M128**: PortF.0 **AVR32Bit**: P27(PA15)) 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.

Ändern des Eingangs Pin

Der verwendete Eingangsport ist als DCF_IN in der Bibliothek "DCF_Lib.cc" definiert.

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. Jede 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, daß 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.

5.7.1 DCF_FRAME

DCF Funktionen

Syntax

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

```
Sub DCF_FRAME()
```

Beschreibung

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

Parameter

Keine

5.7.2 DCF_INIT

DCF Funktionen

Syntax

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

```
Sub DCF_INIT()
```

Beschreibung

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

Parameter

Keine

5.7.3 DCF_PULS

DCF Funktionen

Syntax

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

```
Sub DCF_PULS()
```

Beschreibung

DCF_Mode auf 1 schalten ("Puls suchen").

Parameter

Keine

5.7.4 DCF_START

DCF Funktionen

Syntax

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

```
Sub DCF_START()
```

Beschreibung

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

Parameter

Keine

5.7.5 DCF_SYNC

DCF Funktionen

Syntax

```
void DCF_SYNC(void);

Sub DCF_SYNC()
```

Beschreibung

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

Parameter

Keine

5.8 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 daß das Mega 32 oder Mega 128 Modul 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.

5.8.1 Msg_WriteChar

Debug Message Funktionen

Syntax

```
void Msg_WriteChar(char c);

Sub Msg_WriteChar(c As Char);
```

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

5.8.2 Msg_WriteFloat

Debug Message Funktionen

Syntax

```
void Msg_WriteFloat(float val);

Sub Msg_WriteFloat(val As Single)
```

Beschreibung

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

Parameter

val float Wert

5.8.3 Msg_WriteHex

Debug Message Funktionen

Syntax

```
void Msg_WriteHex(word val);

Sub Msg_WriteHex(val As Word)
```

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

5.8.4 Msg_WriteInt

Debug Message Funktionen

Syntax

```
void Msg_WriteInt(int val);

Sub Msg_WriteInt(val As Integer)
```

Beschreibung

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

Parameter

val 16bit integer Wert

5.8.5 Msg_WriteText

Debug Message Funktionen

Syntax

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

```
Sub Msg_WriteText(ByRef text As Char)
```

Beschreibung

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

Parameter

text Zeiger auf char array

5.8.6 Msg_WriteWord

Debug Message Funktionen

Syntax

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

```
Sub Msg_WriteWord(val As Word)
```

Beschreibung

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

Parameter

val 16bit unsigned integer Wert

5.9 Direct Access (Mega)

Die *Direct Access* Funktionen erlauben einen direkten Zugriff auf alle Register der Atmel Prozessoren. Die Registernummern der Atmel Mega32 und Mega128 Prozessoren können in den Reference Manuals im Kapitel "**Register Summary**" nachgeschlagen werden.

➔ **Vorsicht!** Ein unbedachter Lese- oder Schreibzugriff auf ein Register kann die Funktionalität aller Bibliotheksfunktionen stark beeinträchtigen. Nur wer weiß was er hier macht, sollte die *Direct Access* Funktionen auch benutzen!

5.9.1 DirAcc_Read

Direct Access Funktionen

Syntax

```
byte DirAcc_Read(byte register);

Sub DirAcc_Read(register As Byte) As Byte
```

Beschreibung

Ein Byte Wert wird aus einem Register der Atmel CPU gelesen.

Parameter

register Registernummer (siehe Kapitel "**Register Summary**" im Atmel Reference Manual)

Rückgabewert

Wert des Registers

5.9.2 DirAcc_Write

Direct Access Funktionen

Syntax

```
void DirAcc_Write(byte register, byte val);

Sub DirAcc_Write(register As Byte, val As Byte)
```

Beschreibung

Ein Byte Wert wird in ein Register der Atmel CPU geschrieben.

Parameter

register Registernummer (siehe Kapitel "**Register Summary**" im Atmel Reference Manual)
val Byte Wert

5.10 EEPROM

Auf dem C-Control Pro Modul sind **AVR32Bit**:64kB **M32**:1kB **M128**:4kB EEPROM integriert. Diese Bibliotheksfunktionen ermöglichen den Zugriff auf das EEPROM vom Interpreter.

5.10.1 EEPROM_Read

EEPROM Funktionen

Syntax

```
byte EEPROM_Read(word pos);

Sub EEPROM_Read(pos As Word) As Byte
```

Beschreibung

Liest ein byte von Position pos aus dem EEPROM.

➔ Auf einem C-Control Pro Mega sind die ersten 32 byte für das C-Control Pro System 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

5.10.2 EEPROM_ReadWord

EEPROM Funktionen

Syntax

```
word EEPROM_ReadWord(word pos);

Sub EEPROM_ReadWord(pos As Word) As Word
```

Beschreibung

Liest ein word von Position pos aus dem EEPROM. Der Wert von pos ist eine Byte Position im EEPROM. Dies sollte bei word oder Fließkommazugriffen beachtet werden.

➔ Auf einem C-Control Pro Mega sind die ersten 32 byte für das C-Control Pro System reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu.

Parameter

pos Byte Position im EEPROM

Rückgabewert

der Wert des word an Position pos im EEPROM

5.10.3 EEPROM_ReadFloat

EEPROM Funktionen

Syntax

```
float EEPROM_ReadFloat(word pos);

Sub EEPROM_ReadFloat(pos As Word) As Single
```

Beschreibung

Liest einen Fließkommawert von Position pos aus dem EEPROM. Der Wert von pos ist eine Byte Position im EEPROM. Dies sollte bei word oder Fließkommazugriffen beachtet werden.

➔ Auf einem C-Control Pro Mega sind die ersten 32 byte für das C-Control Pro System reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu.

Parameter

pos Byte Position im EEPROM

Rückgabewert

der Fließkommawert an Position pos im EEPROM

5.10.4 EEPROM_Write

EEPROM Funktionen

Syntax

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

Sub EEPROM_Write(pos As Word, val As Byte)
```

Beschreibung

Schreibt ein byte an Position pos in das EEPROM

➔ Auf einem C-Control Pro Mega sind die ersten 32 byte für das C-Control Pro System 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

5.10.5 EEPROM_WriteWord

EEPROM Funktionen

Syntax

```
void EEPROM_WriteWord(word pos, word val);

Sub EEPROM_WriteWord(pos As Word, val As Word)
```

Beschreibung

Schreibt ein word an Position pos in das EEPROM. Der Wert von pos ist eine Byte Position im EEPROM. Dies sollte bei word oder Fließkommazugriffen beachtet werden.

➔ Auf einem C-Control Pro Mega sind die ersten 32 byte für das C-Control Pro System reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu.

Parameter

pos	Byte Position im EEPROM
val	der ins EEPROM zu schreibende Wert

5.10.6 EEPROM_WriteFloat

EEPROM Funktionen

Syntax

```
void EEPROM_WriteFloat(word pos, float val);

Sub EEPROM_WriteFloat(pos As Word, val As Single)
```

Beschreibung

Schreibt einen Fließkommawert an Position pos in das EEPROM. Der Wert von pos ist eine Byte Position im EEPROM. Dies sollte bei word oder Fließkommazugriffen beachtet werden.

➔ Auf einem C-Control Pro Mega sind die ersten 32 byte für das C-Control Pro System reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu.

Parameter

pos	Byte Position im EEPROM
val	der ins EEPROM zu schreibende Wert

5.11 Ethernet (AVR32Bit)

Die C-Control Pro AVR32Bit Unit unterstützt Ethernet Hardware und Protokolle nach IEEE-Norm 802.3. Der angeschlossene PHY arbeitet mit Auto-Negotiation und stellt sich bei einer Verbindung auf eine Geschwindigkeit von 100Mbit oder 10Mbit ein, je nachdem was die Gegenseite (z.B. der Switch) anbietet. Power-Over-Ethernet wird nicht unterstützt. Folgende Protokolle sind momentan implementiert:

- ARP
- ICMP Echo (ping)
- DHCP
- TCP/IP
- UDP
- C-Control Pro (UDP Port 50234)
- HTTP (TCP/IP)

Vorkenntnisse

Dieses Kapitel und der erfolgreiche Einsatz der Bibliothek setzen ein Basiswissen über folgende Bereiche von IPv4 voraus:

- IP-Nummern
- Port Adressen und Bedeutung
- UDP Pakete
- TCP/IP Datenstrom

➔ Zu empfehlen wären Vorkenntnisse in einer Programmierumgebung mit TCP/IP z.B. dem BSD Socket Interface.

5.11.1 Ethernet Aktivierung

➔ Um Verbindungsprobleme zu vermeiden sollte vor dem Einschalten der Ethernet Unterstützung die MAC-Adresse ("MAC Adresse ändern") auf einen neuen Wert eingestellt werden (siehe C-Control Konfiguration). Zu diesem Zweck wird für jede verkaufte C-Control Pro AVR32Bit eine eigene MAC-Adresse erzeugt und auf einem Aufkleber mitgeliefert. Siehe Software Installation.

- In der C-Control Konfiguration muss die Ethernet Unterstützung aktiviert sein. Danach sollte beim Einstecken des Ethernet Steckers die gelbe LED dauerhaft angehen, und die grüne LED sporadisch blinken.
- Wenn DHCP nicht aktiviert ist, werden die Netzwerkparameter aus der C-Control Konfiguration benutzt. Die Einträge IP-Adresse, Netzmaske, und Gateway müssen dann manuell eingetragen werden.
- Ist DHCP im Einsatz, werden die Netzwerkparameter vom DHCP Server (z.B. DSL-Router oder ähnlich) abgerufen. Das DHCP Protokoll wird nicht vom Bootloader unterstützt, sondern nur vom Interpreter. Daher nach der DHCP Aktivierung z.B. einmal den Reset Taster betätigen, um in den Interpreter zu gelangen.
- Eine Änderung der Netzwerkdaten über DHCP wird direkt in der Konfiguration abgespeichert, wenn die Option speichere DHCP Werte eingeschaltet ist.
- Zum Test ob das Netzwerk richtig konfiguriert ist, vom PC einen ping an die AVR32Bit Unit schicken. Der Parameter Ping erlauben muss dafür aktiviert sein.

➔ Man kann beim Stoppen des Programms mit dem Start/Stop Taster den lwIP TCP/IP-Stack in den Zustand bringen, das noch dynamischer Speicher der aktuellen Verbindung nicht frei gegeben wird. Dieser Speicher kann beim Neustart des Programms fehlen. Bei Problemen sollte im Zweifelsfall der Reset-Taster betätigt werden, um einen kompletten Neustart des Systems auszulösen.

5.11.2 TCP/IP Programmierung

Eine TCP/IP Verbindung öffnen:

- Mit ETH_SetConnBuf einen Empfangspuffer anlegen.
- Der Aufruf von ETH_ConnectTCP baut eine Verbindung auf und setzt den internen State auf ES_CONNECTING.
- Man überwacht mit ETH_GetStateTCP den Zustand der Verbindung. Nach ES_CONNECTING kann der Wert sich auf ES_CONNECTED oder ES_DISCONNECTED ändern. Bei ES_CONNECTED ist die Verbindung aufgebaut, sonst fand eine eine Zeitüberschreitung statt, oder die Gegenstelle hat abgelehnt.
- Ist die Verbindung aufgebaut, kann man mit ETH_SendTCP Daten senden.
- Gleichzeitig ist mit ETH_CheckReceiveBuf periodisch zu überprüfen ob Daten eingegangen sind, und man muss mit ETH_GetStateTCP überwachen, ob die Verbindung nicht irgendwann in den Zustand ES_DISCONNECTED fällt.
- Ein Aufruf von ETH_DisconnectTCP beendet die Verbindung.

Auf einem TCP/IP Port auf eine eingehende Verbindung warten:

- Mit ETH_SetConnBuf einen Empfangspuffer anlegen.
- ETH_ListenTCP überwacht einen spezifizierten Port.
- Man überprüft mit ETH_CheckReceiveBuf periodisch ob Daten eingegangen sind und damit eine neue Verbindung von außen aufgemacht wurde. Der Zustand von ETH_GetStateTCP hat jetzt den Wert ES_LCONNECTED.
- Ist die Verbindung aufgebaut, kann man mit ETH_SendTCP Daten senden.
- Man muss mit ETH_GetStateTCP überwachen, ob die Verbindung nicht irgendwann in den Zustand ES_DISCONNECTED fällt.
- Ein Aufruf von ETH_DisconnectTCP beendet die Verbindung.

➔ Es ist zu empfehlen, sich die Demoprogramme für UDP und TCP/IP anzuschauen.

➔ Es können gleichzeitig bis zu 10 TCP/IP Verbindungen aufgemacht werden, und bei bis zu 3 Ports auf eingehende Verbindungen gelauscht werden.

➔ Als Default sind 4kb für den TCP/IP Stack reserviert. Je nach Nutzung braucht der Stack mehr oder weniger RAM. Der benötigte Speicher ist nur aufwendig zu berechnen, und sollte selbst durch Tests ermittelt werden.

Beispiele

Das Programm baut eine Verbindung zum HTTP Port auf, schickt ein "GET" Kommando ab, und empfängt die Antwort:

```
byte tcp_buf[ETH_BUF(4000,6)], rbuf[1461];

void main(void)
{
    word info[4], plen;
    char cmdtxt[50];
    dword ip;
    byte id, state;

    ETH_SetConnBuf(tcp_buf, 4000, 6);
    id= ETH_ConnectTCP(IP_ADDR(192,168,0,1), 80);

    state= ES_CONNECTING;
    while(state == ES_CONNECTING)
    {
        state= ETH_GetStateTCP(id);
    }

    if(state == ES_CONNECTED)
    {
        cmdtxt= "GET / HTTP/1.1\n\n";
        ETH_SendTCP(id, cmdtxt, Str_Len(cmdtxt));

        while(1)
        {
            ip= ETH_CheckReceiveBuf(info);
            if(ip)
            {
                plen= info[3];
                if(plen > 1460) plen= 1460;  // auf 1460 bytes begrenzen
                ETH_ReceiveData(rbuf, plen);
            }
        }
    }
}
```

Das folgende Beispiel wartet auf eingehende Verbindungen auf Port 23 (Telnet). Die Daten werden in rbuf abgeholt aber nicht weiter verarbeitet:

```
byte tcp_buf[ETH_BUF(4000,6)],  rbuf[200];  // 4000 byte Empfangspuffer

void main(void)
{
    word info[4], plen;
    dword ip;
```

```
    char text[10];

    ETH_SetConnBuf(tcp_buf, 4000, 6);  // 4000 byte Puffer und 6 Connections
    ETH_ListenTCP(23);  // Listen Telnet port

    while(1)
    {
        ip= ETH_CheckReceiveBuf(info);
        if(ip)
        {
            plen= info[3];  //
            if(plen > 200) plen= 200;  // Begrenze auf 200 bytes
            ETH_ReceiveData(rbuf, plen);
            txt= "Cmd:\n";
            ETH_SendTCP(info[0], txt, 5); // Sende Cmd String
        }
    }

}
```

5.11.3 UDP Programmierung

- UDP Pakete können direkt mit ETH_SendUDP versendet werden. Dabei beträgt die Maximalgröße 1460 Byte. Dies entspricht einer MTU von 1500 und einem 40 Byte UDP/IP header.
- Um UDP Pakete empfangen zu können, wird mit ETH_SetConnBuf ein Empfangspuffer (Ringpuffer) reserviert und mit ETH_ListenUDP ein listening Port eingerichtet. Danach landen alle Pakete im Empfangspuffer. Wenn der Puffer voll ist, gehen die weiteren empfangenen Daten verloren. Der Puffer sollte daher regelmäßig mit der Funktion ETH_CheckReceiveBuf überprüft werden. Der Aufruf von ETH_ReceiveData kopiert die Pufferdaten in ein Bytearray. Wird dabei eine kleinere Datenlänge angegeben, als Paketbytes im Puffer sind, werden die auch die übrigen Bytes des Pakets aus dem Ringpuffer gelöscht.

➔ Es ist zu empfehlen, sich die Demoprogramme für UDP und TCP/IP anzuschauen.

➔ Als Default sind 4kb für den TCP/IP Stack reserviert. Je nach Nutzung braucht der Stack mehr oder weniger RAM. Der benötigte Speicher ist nur aufwendig zu berechnen, und sollte selbst durch Tests ermittelt werden.

Beispiele

1. Programm Sendet jede Sekunde einen String auf den Syslog Port 514:

```
void SendSyslogMsg(dword ip, byte level, char text[])
{
    byte buf[100];

    Str_Printf(buf, "<%d>%s", 16*8+level, text);
    ETH_SendUDP(ip, 514, buf, Str_Len(buf));
}

void main(void)
{
    while(1)
    {
        SendSyslogMsg(IP_ADDR(192,168,0,1), 3, "test message");
        AbsDelay(1000);
    }
}
```

2. Programm sendet auf UDP Port 50000 empfangene Daten wieder zurück an den Absender:

```
byte buf[ETH_BUF(500,0)], rbuf[200];  // 500 byte Empfangspuffer

void main(void)
{
    word info[4], plen;
    dword ip;

    ETH_SetConnBuf(buf, 500, 0);
    ETH_ListenUDP(50000);  // Listen auf Port 50000

    while(1)
    {
        ip= ETH_CheckReceiveBuf(info);
        if(ip)
        {
            plen= info[3];  // Länge
            if(plen > 200) plen= 200;  // Begrenze auf 200 bytes
            ETH_ReceiveData(rbuf, plen);
            ETH_SendUDP(ip, 50000, rbuf, plen);
        }
    }
}
```

5.11.4 ETH_ConnectTCP

Ethernet Funktionen

Syntax

```
byte ETH_ConnectTCP(dword ip, word port);

Sub ETH_ConnectTCP(ip As ULong, port As Word) As Byte
```

Beschreibung

Öffnet ein TCP/IP Verbindung zu einem Port. Mit dem Macro IP_ADDR() kann man sich den 32-Bit Wert der IP-Adresse aus der gewöhnten Notation berechnen lassen: Z.B. IP_ADDR(192,168,1,1).

➔ Bei der Rückkehr aus ETH_ConnectTCP ist die Verbindung nicht direkt aufgebaut. Man muss den Zustand der Verbindung mit ETH_GetStateTCP überwachen.

Parameter

ip	IP-Adresse
port	UDP Port

Rückgabewert

sock_id (Socket Index), ff (Hex) im Fehlerfall

5.11.5 ETH_CheckReceiveBuf

Ethernet Funktionen

Syntax

```
dword ETH_CheckReceiveBuf(word info[]);

Sub ETH_CheckReceiveBuf(ByRef info As Word) As ULong
```

Beschreibung

Überprüft ob Pakete im Empfangspuffer vorhanden sind. Ist der Rückgabewert Null, so sind keine Ethernetpakete empfangen worden. Ist ein Paket da, so werden in das info array zusätzliche Parameter abgelegt. Das info array (16-Bit) sollte eine Größe von 4 haben. Wird ein UDP Paket empfangen, so ist der Socket Index (info[0]) gleich ff (Hex).

➔ Man sollte darauf achten, den Socket Index (sock_idx) nicht mit dem Sockethandle zu verwechseln. Die Listen Befehle (ListenTCP, CloseListenTCP etc.) arbeiten mit dem Sockethandle, alle anderen mit dem Socket Index.

Parameter

info

Rückgabewert

IP Adresse des Senders der das Paket im Puffer gesendet hat
0, wenn keine Pakete im Puffer sind

Info Array

info[0]	Socket Index

info[1]	IP Port Absender
info[2]	Sockethandle
info[3]	Paketlänge

5.11.6 ETH_CloseListenTCP

Ethernet Funktionen

Syntax

```
void ETH_CloseListenTCP(word handle);

Sub ETH_CloseListenTCP(handle As Word)
```

Beschreibung

Schließt einen TCP listening Socket der mit ETH_ListenTCP angelegt wurde.

Parameter

handle ETH_ListenTCP handle

5.11.7 ETH_CloseListenUDP

Ethernet Funktionen

Syntax

```
void ETH_CloseListenUDP(word handle);

Sub ETH_CloseListenUDP(handle As Word)
```

Beschreibung

Schließt einen UDP listening Socket der mit ETH_ListenUDP angelegt wurde.

Parameter

handle ETH_ListenUDP handle

5.11.8 ETH_DisconnectTCP

Ethernet Funktionen

Syntax

```
void ETH_DisconnectTCP(byte sock_id);

Sub ETH_DisconnectTCP(sock_id As Byte)
```

Beschreibung

Beendet eine aufgebaute Verbindung.

Parameter

sock_id Socket Index

5.11.9 ETH_GetIPInfo

Ethernet Funktionen

Syntax

```
void ETH_GetIPInfo(byte info, byte data[]);

Sub ETH_GetIPInfo(info As Byte, ByRef data As Byte)
```

Beschreibung

Gibt Ethernet Informationen in ein Byte Array zurück. Die Länge des Array muss so dimensioniert sein, das die Werte hineinpassen. Ist DHCP aktiviert und die IP-Adresse momentan 0.0.0.0, so wurde noch keine gültige IP-Adresse bisher per DHCP vergeben.

Parameter

info Infotyp
data Rückgabe Array

Infotyp	Bedeutung	Länge
EI_IP_ADDR	IP-Adresse	4
EI_NETMASK	Netzmaske	4
EI_GATEWAY	Gateway Adresse	4
EI_MACADDR	MAC Adresse	6

5.11.10 ETH_GetStateTCP

Ethernet Funktionen

Syntax

```
byte ETH_GetStateTCP(byte sock_id);

Sub ETH_GetStateTCP(sock_id As Byte) As Byte
```

Beschreibung

Informiert über den Zustand der Verbindung. Da die Gegenseite jederzeit eine TCP/IP Verbindung abbrechen kann, sollte der Status in der Hauptschleife des Programms periodisch überwacht werden.

➔ Der Parameter sock_id wird entweder von ETH_ConnectTCP zurückgegeben, oder man bekommt ihn als info[0] Wert von ETH_CheckReceiveBuf.

Parameter

`sock_id` Socket Index

Rückgabewert

Verbindungsstatus

State Tabelle

#define	Wert	Bedeutung
ES_DISCONNECTED	0	Keine TCP/IP Verbindung
ES_CONNECTING	1	Verbindung wurde gestartet (ETH_ConnectTCP)
ES_CONNECTED	2	Verbindung ist aufgebaut (ETH_ConnectTCP)
ES_LCONNECTED	3	Verbindung ist aufgebaut (ETH_ListenTCP)

5.11.11 ETH_ListenTCP

Ethernet Funktionen

Syntax

```
word ETH_ListenTCP(word port);

Sub ETH_ListenTCP(port As Word) As Word
```

Beschreibung

Öffnet einen listening Socket auf einem TCP Port. Empfangene Pakete werden in dem Puffer abgelegt, der mit ETH_SetConnBuf initialisiert wurde.

Parameter

port TCP Port

Rückgabewert

handle zum TCP listening Socket, 0 im Fehlerfall

5.11.12 ETH_ListenUDP

Ethernet Funktionen

Syntax

```
word ETH_ListenUDP(word port);

Sub ETH_ListenUDP(port As Word) As Word
```

Beschreibung

Öffnet einen listening Socket auf einem UDP Port.

Parameter

port UDP Port

Rückgabewert

handle zum UDP listening Socket, 0 im Fehlerfall

5.11.13 ETH_ReceiveData

Ethernet Funktionen

Syntax

```
void ETH_ReceiveData(byte buf[], word len);

Sub ETH_ReceiveData(ByRef buf As Byte, len As Word)
```

Beschreibung

Speichert ein Paket aus dem Ethernet Empfangspuffer an Adresse buf. Der Parameter len darf kleiner als die Länge der Paketdaten sein, die übrigen Bytes des Pakets werden verworfen. Möchte man die kompletten Paketdaten verwerfen, ohne das sie abgespeichert werden, so setzt man len auf Null.

Parameter

buf Arrayvariable in dem Paketdaten gespeichert werden
len Anzahl der Bytes die kopiert werden

5.11.14 ETH_SendTCP

Ethernet Funktionen

Syntax

```
byte ETH_SendTCP(byte sock_id, byte buf[], word len);

Sub ETH_SendTCP(sock_id As Byte, ByRef buf As Byte, len As Word) As Byte
```

Beschreibung

Sendet TCP Daten zu einer geöffneten TCP/IP Verbindung.

➔ Der Parameter sock_id wird entweder von ETH_ConnectTCP zurückgegeben, oder man bekommt ihn als info[0] Wert von ETH_CheckReceiveBuf.

Parameter

sock_id	socket index
buf	buffer address of TCP data
len	length of TCP data

Rückgabewert

0 wenn kein Fehler vorliegt

5.11.15 ETH_SendUDP

Ethernet Funktionen

Syntax

```
void ETH_SendUDP(dword ip, word port, byte buf[], word len);

Sub ETH_SendUDP(ip As ULong, port As Word, ByRef buf As Byte, len As
Word)
```

Beschreibung

Sendet ein UDP Paket zu einer IP-Adresse und Port.

Parameter

ip	IP-Adresse
port	UDP Port
buf	Adresse des Paket Puffers
len	Länge des UDP Pakets

5.11.16 ETH_SetConnBuf

Ethernet Funktionen

Syntax

```
void ETH_SetConnBuf(byte buf[], word size, byte TCP_conn);

Sub ETH_SetConnBuf(ByRef buf As Byte, size As Word, TCP_conn As Byte)
```

Beschreibung

Legt einen Ethernet Empfangspuffer an, in dem empfangene TCP/IP und UDP Pakete abgelegt werden.

Parameter

`buf` Adresse des Empfangspuffers
`size` Größe des Puffers

5.12 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.

5.12.1 Mega

5.12.1.1 I2C_Init

I2C Funktionen **Beispiel**

Syntax

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

```
Sub I2C_Init(I2C_BR As Byte)
```

Beschreibung

Diese Funktion initialisiert die I2C-Schnittstelle.

Parameter

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

I2C_100kHz
I2C_400kHz

Definition	14,7456 Mhz	16 Mhz
I2C_100kHz	66	72
I2C_400kHz	10	12

Die Bitrate kann wie folgt berechnet werden: Bitrate = ((CPU_CLOCK / TARGET_I2C_SPEED) - 16) / 2

5.12.1.2 I2C_Read_ACK

I2C Funktionen

Syntax

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

```
Sub I2C_Read_ACK() As Byte
```

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

5.12.1.3 I2C_Read_NACK

I2C Funktionen Beispiel

Syntax

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

```
Sub I2C_Read_NACK() As Byte
```

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

5.12.1.4 I2C_Start

I2C Funktionen Beispiel

Syntax

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

```
Sub I2C_Start()
```

Beschreibung

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

Parameter

Keine

5.12.1.5 I2C_Status

I2C Funktionen

Syntax

```
byte I2C_Status(void);

Sub I2C_Status()
```

Beschreibung

Mit I2C_Status kann der Status der Schnittstelle abgefragt werden. Die Bedeutung der Statusinformation ist in der I2C Status Tabelle dargestellt.

Rückgabewert

aktueller I2C Status

5.12.1.6 I2C_Stop

I2C Funktionen **Beispiel**

Syntax

```
void I2C_Stop(void);

Sub I2C_Stop()
```

Beschreibung

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

Parameter

Keine

5.12.1.7 I2C_Write

I2C Funktionen Beispiel

Syntax

```
void I2C_Write(byte data);

Sub I2C_Write(data As Byte)
```

Beschreibung

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

Parameter

data Datenbyte

5.12.1.8 I2C Status Tabelle

Tabelle: **Status Codes Master Transmitter Mode**

Status Code (Hex)	Beschreibung
08	eine START Sequenz wurde gesendet
10	eine "repeated" START Sequenz wurde gesendet
18	SLA+W wurde gesendet, ACK wurde empfangen
20	SLA+W wurde gesendet, NACK wurde empfangen
28	Data byte wurde gesendet, ACK wurde empfangen
30	Data byte wurde gesendet, NACK wurde empfangen
38	Konflikt in SLA+W or data bytes

Tabelle: **Status Codes Master Receiver Mode**

Status Code (Hex)	Beschreibung
08	eine START Sequenz wurde gesendet
10	eine "repeated" START Sequenz wurde gesendet
38	Konflikt in SLA+R or data bytes

40	SLA+R wurde gesendet, ACK wurde empfangen
48	SLA+R wurde gesendet, NACK wurde empfangen
50	Data byte wurde empfangen, ACK wurde gesendet
58	Data byte wurde empfangen, NACK wurde gesendet

5.12.1.9 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);
}
```

5.12.2 AVR32Bit

5.12.2.1 I2C_Probe

I2C Funktionen

Syntax

```
byte I2C_Probe(byte addr);

Sub I2C_Probe(addr As Byte) As Byte
```

Beschreibung

I2C_Probe versucht ein I2C Gerät anzusprechen und gibt als Ergebnis ob der Versuch erfolgreich war.

Parameter

addr Adresse des I2C Gerät

Rückgabewert

1 = Gerät hat geantwortet
0 sonst

5.12.2.2 I2C_Read

I2C Funktionen

Syntax

```
byte I2C_Read(byte addr, dword hdr, byte hdr_len, byte mem_addr[],
              word length);

Sub I2C_Read(addr As Byte, hdr As ULong, hdr_len As Byte,
             ByRef mem_addr As Byte, length As Word) As Byte
```

Beschreibung

Zuerst werden an das I2C device mit Adresse addr (I2C 7-bit Adresse) bis zu 4 Bytes header Daten geschrieben. Die Daten werden in hdr übergeben, die Anzahl der Bytes in hdr_len. Dabei darf hdr_len Null sein, d.h. es wird kein header übertragen. Es werden immer zuerst die höherwertigen Bytes des header übertragen. Anschließend werden length bytes vom I2C device in das array mem_addr geschrieben.

➔ Der Begriff header steht hier nicht für einen spezifischen I2C Term, sondern für bis zu 4 Bytes die an das I2C device übertragen werden. Viele I2C Geräte benutzen einen solchen header um z.B. ein Register zu indizieren.

Parameter

addr	Adresse des I2C Gerätes
hdr	Bis zu 4 byte header Daten
hdr_len	Länge des headers

mem_addr Array in das die I2C Daten gelesen werden
length Anzahl der Daten die gelesen werden

Rückgabewert

-1 = Übertragungsfehler
0 = erfolgreich

5.12.2.3 I2C_SetSpeed

I2C Funktionen **Beispiel**

Syntax

```
void I2C_SetSpeed(dword I2C_BR);

Sub I2C_SetSpeed(I2C_BR As ULong)
```

Beschreibung

Diese Funktion setzt die Geschwindigkeit der I2C Schnittstelle

Parameter

I2C_BR gibt die Bitrate als 32-Bit Wert an. Folgende Werte sind schon vordefiniert:

I2C_100kHz
I2C_400kHz

Definition	Wert
I2C_100kHz	100000
I2C_400kHz	400000

Rückgabewert

-8 = Parameterfehler
0 = erfolgreich

5.12.2.4 I2C_Write

I2C Funktionen

Syntax

```
byte I2C_Write(byte addr, dword hdr, byte hdr_len, byte mem_addr[],
               word length);
```

```
Sub I2C_Write(addr As Byte, hdr As ULong, hdr_len As Byte,
              ByRef mem_addr As Byte, length As Word) As Byte
```

Beschreibung

Zuerst werden an das I2C device mit Adresse addr (I2C 7-bit Adresse) bis zu 4 Bytes header Daten geschrieben. Die Daten werden in hdr übergeben, die Anzahl der Bytes in hdr_len. Dabei darf hdr_len Null sein, d.h. es wird kein header übertragen. Es werden immer zuerst die höherwertigen Bytes des header übertragen. Anschließend werden length bytes aus array mem_addr auf das I2C device geschrieben.

➔ Der Begriff header steht hier nicht für einen spezifischen I2C Term, sondern für bis zu 4 Bytes die an das I2C device übertragen werden. Viele I2C Geräte benutzen einen solchen header um z.B. ein Register zu indizieren.

Parameter

addr	Adresse des I2C Gerätes
hdr	Bis zu 4 byte header Daten
hdr_len	Länge des headers
mem_addr	Array aus dem die I2C Daten geschrieben werden
length	Anzahl der Daten die geschrieben werden

Rückgabewert

-1 = Übertragungsfehler
0 = erfolgreich

5.12.2.5 I2C Beispiel

Beispiel: EEPROM 24C64 lesen und schreiben

```
// I2C Geräte Adresse = 0x50, Bit Rate 100kHz
// EEPROM hat 16bit Speicher adresse
byte data[10];

void main(void)
{
    // Lese 10 bytes von Speicheradresse 0x20 in array data[]
    I2C_Read(0x50, 0x20, 2, data, 10);

    // Schreibe 10 bytes aus array data[] nach EEPROM Speicheradresse 0x20
    I2C_Write(0x50, 0x20, 2, data, 10);
}
```

5.13 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	Mega32	Mega128 (CAN)
INT_0	externer Interrupt 0	externer Interrupt 0
INT_1	externer Interrupt 1	externer Interrupt 1
INT_2	externer Interrupt 2	externer Interrupt 2
INT_3	---	externer Interrupt 3
INT_4	---	externer Interrupt 4
INT_5	---	externer Interrupt 5
INT_6	---	externer Interrupt 6
INT_7	---	externer Interrupt 7
INT_TIM1CAPT	Timer 1 Capture	Timer 1 Capture
INT_TIM1CMPA	Timer 1 CompareA	Timer 1 CompareA
INT_TIM1CMPB	Timer 1 CompareB	Timer 1 CompareB
INT_TIM1OVF	Timer 1 Overflow	Timer 1 Overflow
INT_TIM0COMP	Timer 0 Compare	Timer 0 Compare
INT_TIM0OVF	Timer 0 Overflow	Timer 0 Overflow
INT_ANA_COMP	Analog Comparator	Analog Comparator
INT_ADC	ADC	ADC
INT_TIM2COMP	Timer 2 Compare	Timer 2 Compare
INT_TIM2OVF	Timer 2 Overflow	Timer 2 Overflow
INT_TIM3CAPT	---	Timer 3 Capture
INT_TIM3CMPA	---	Timer 3 CompareA
INT_TIM3CMPB	---	Timer 3 CompareB
INT_TIM3CMPC	---	Timer 3 CompareC
INT_TIM3OVF	---	Timer 3 Overflow

➔ Ein Signal auf INT_0 (**Mega32**) oder INT_4 (**Mega128 (CAN)**) kann beim Einschalten des C-Control Pro Mega Moduls das Autostartverhalten stören. Nach der Pinzuordnung von M32 und M128 liegen diese Interrupts 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.

Interrupt Name	AVR32Bit
INT_ANA_COMP	Analog Comparator
INT_1	externer Interrupt 1

INT_2	externer Interrupt 2
INT_3	externer Interrupt 3
INT_4	externer Interrupt 4
INT_5	externer Interrupt 5
INT_6	externer Interrupt 6
INT_7	externer Interrupt 7
INT_ADC	ADC
INT_100Hz	100 Hz Interrupt
INT_TIMER0	Timer 0
INT_TIMER1	Timer 1
INT_TIMER2	Timer 2
INT_TIMER3	Timer 3
INT_TIMER4	Timer 4
INT_TIMER5	Timer 5
INT_CAN	CAN

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.

5.13.1 Ext_IntEnable

INT_TIMER0Interrupt Funktionen

Syntax

```
void Ext_IntEnable(byte IRQ, byte Mode);

Sub Ext_IntEnable(IRQ As Byte, Mode As Byte)
```

Beschreibung

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

➔ Der Parameter IRQ hat einen numerischen Wert. Nicht verwechseln mit den #defines von Parameter irqnr bei Funktion Irq_SetVect().

Parameter

IRQ Nummer des freizuschaltenden Interrupts **Mega32** (0-2) , **Mega128** (0-7) , **AVR32** (1-7)
Mode Parameter für **Mega32** und **Mega128 (CAN)**:

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

➔ Ein Signal auf **Mega32**:IRQ 0 **Mega128**:IRQ 4 kann zu Autostart Problemen führen.

Mode Parameter nur für **Mega32** und IRQ2:

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

Mode Parameter für **AVR32:**

0: ein low Pegel löst einen Interrupt aus
1: ein high Pegel löst einen Interrupt aus
2: eine fallende Flanke löst einen Interrupt aus
3: eine steigende Flanke löst einen Interrupt aus

Oderiert man zum Mode Parameter (nur AVR32) den Wert 40 (Hex) wird ein interner PullDown gesetzt, oderiert man 80 (Hex) wird ein interner PullUp aktiviert.

5.13.2 Ext_IntDisable

Interrupt Funktionen

Syntax

```
void Ext_IntDisable(byte IRQ);

Sub Ext_IntDisable(IRQ As Byte)
```

Beschreibung

Der externe Interrupt IRQ wird gesperrt.

➔ Der Parameter IRQ hat Werte zwischen 0 und 2 auf dem Mega32 und zwischen 0 und 7 auf dem Mega128. Nicht verwechseln mit dem irqnr Parameter von Irq_SetVect().

Parameter

IRQ Nummer des zu sperrenden Interrupts **Mega32** (0-2) , **Mega128** (0-7) , **AVR32** (1-7)

5.13.3 Irq_GetCount

Interrupt Funktionen **Beispiel**

Syntax

```
byte Irq_GetCount(byte irqnr);

Sub Irq_GetCount(irqnr As Byte) As Byte
```

Beschreibung

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

Parameter

irqnr spezifiziert den Typ des Interrupts (siehe Tabelle)

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.

5.13.4 Irq_SetVect

Interrupt Funktionen **Beispiel**

Syntax

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

Sub Irq_SetVect(irqnr As Byte, vect As ULong)
```

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. Ein vect von Null setzt den Interrupt wieder inaktiv.

Parameter

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

5.13.5 IRQ Beispiel

Beispiel: Verwendung von Interrupt Routinen

```
// INT_100HZ (AVR32Bit) oder Timer 2 (MEGA) laufen normalerweise im 10ms
// Takt. Im Beispiel wird daher die Variable cnt alle 10ms um 1 erhöht

int cnt;

void ISR(void)
{
    cnt=cnt+1;
#if AVR32
    Irq_GetCount(INT_100HZ);
#else
    Irq_GetCount(INT_TIM2COMP);
#endif
}

void main(void)
{
    cnt=0;

#if AVR32
    Irq_SetVect(INT_100HZ, ISR);
#else
    Irq_SetVect(INT_TIM2COMP, ISR);
#endif
    while(true);  // Endlosschleife
}
```

5.14 Keyboard (Mega)

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, daß man bei Nichtgebrauch, diese Funktionen durch Weglassen der Bibliothek aus dem Projekt nimmt. Direkte Interpreterfunktionen sind immer präsent, kosten aber Flashspeicher.

➔ Beim AVR32Bit Applicationboard wird keine Tastatur mitgeliefert, daher gibt es keine Keyboard Routinen in der Bibliothek.

5.14.1 Key_Init

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

Syntax

```
void Key_Init(void);

Sub Key_Init()
```

Beschreibung

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

Parameter

Keine

5.14.2 Key_Scan

Keyboard Funktionen

Syntax

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

```
Sub Key_Scan() As Word
```

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

5.14.3 Key_TranslateKey

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

Syntax

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

```
Sub Key_TranslateKey(keys As Word) As Char
```

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

5.15 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, daß man bei Nichtgebrauch, diese Funktionen durch Weglassen der Bibliothek aus dem Projekt nimmt. Direkte Interpreterfunktionen sind immer präsent, kosten aber Flashspeicher.

5.15.1 Interne Funktionen

Die hier aufgeführten Funktionen werden intern verwendet, und sollten in der Regel nicht durch den Anwender genutzt werden.

5.15.1.1 LCD_SubInit

LCD Funktionen

Syntax

```
void LCD_SubInit(void);

Sub LCD_SubInit()
```

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

5.15.1.2 LCD_TestBusy

LCD Funktionen

Syntax

```
void LCD_TestBusy(void);

Sub LCD_TestBusy()
```

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

5.15.1.3 LCD_WriteCTRRegister

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

Syntax

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

```
Sub LCD_WriteCTRRegister(cmd As Byte)
```

Beschreibung

Schickt ein Kommando zum Display Controller.

Parameter

cmd Kommando in Byteform

5.15.1.4 LCD_WriteDataRegister

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

Syntax

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

```
Sub LCD_WriteDataRegister(x As Char)
```

Beschreibung

Schickt ein Datenbyte zum Display Controller.

Parameter

x Datenbyte

5.15.2 LCD_ClearLCD

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

Syntax

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

```
Sub LCD_ClearLCD()
```

Beschreibung

Löscht das Display und schaltet den Cursor ein.

Parameter

Keine

5.15.3 LCD_CursorOff

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

Syntax

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

```
Sub LCD_CursorOff()
```

Beschreibung

Schaltet den Cursor des Display aus.

Parameter

Keine

5.15.4 LCD_CursorOn

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

Syntax

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

```
Sub LCD_CursorOn()
```

Beschreibung

Schaltet den Cursor des Display ein.

Parameter

Keine

5.15.5 LCD_CursorPos

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

Syntax

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

```
Sub LCD_CursorPos(pos As Byte)
```

Beschreibung

Setzt den Cursor auf Position pos.

Parameter

pos Cursorposition

Wert von pos (Hex)	**Position im Display**
00-07	0-7 in der 1. Zeile
40-47	0-7 in der 2. Zeile

Für Display mit mehr als 2 Zeilen und bis zu 32 Zeichen pro Zeile gilt folgendes Schema:

Wert von pos (Hex)	**Position im Display**
00-1f	0-31 in der 1. Zeile
40-5f	0-31 in der 2. Zeile
20-3f	0-31 in der 3. Zeile
60-6f	0-31 in der 4. Zeile

5.15.6 LCD_Init

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

Syntax

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

```
Sub LCD_Init()
```

Beschreibung

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

Parameter

Keine

5.15.7 LCD_Locate

LCD Funktionen

Syntax

```
void LCD_Locate(int row, int column);

Sub LCD_Locate(row As Integer, column As Integer)
```

Beschreibung

Setzt den Cursor des LCD Displays auf ein bestimmte Zeile und Spalte.

Parameter

row	Zeile
column	Spalte

5.15.8 LCD_SetDispAddr (AVR32Bit)

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

Syntax

```
void LCD_SetDispAddr(byte addr);

Sub LCD_SetDispAddr(addr As Byte)
```

Beschreibung

Setzt eine neue Ziel-Adresse für die LCD Ausgaben. Auf diese Weise können mehrere LCD1602 Boards gleichzeitig adressiert werden. Siehe Adressierung im Kapitel "C-Control PRO AVR32 LCD1602 Board". Der Default Wert für die Adresse ist 27 (Hex).

Parameter

addr	neue Adresse

5.15.9 LCD_WriteChar

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

Syntax

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

```
Sub LCD_WriteChar(c As Char)
```

Beschreibung

Schreibt ein Zeichen an die Cursorposition im LCD Display.

Parameter

c ASCII Wert des Zeichens

5.15.10 LCD_WriteFloat

LCD Funktionen

Syntax

```
void LCD_WriteFloat(float value, byte length);
```

```
Sub LCD_WriteFloat(value As Single, length As Byte)
```

Beschreibung

Schreibt eine Fließkommazahl mit angegebener Länge auf das LCD Display.

Parameter

value Fließkommawert
length Ausgabelänge

5.15.11 LCD_WriteRegister

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

Syntax

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

```
Sub LCD_WriteRegister(y As Byte,x As Byte)
```

Beschreibung

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

Parameter

y Datenbyte
x Kommandonibble

5.15.12 LCD_WriteText

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

Syntax

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

Sub LCD_WriteText(ByRef Text As Char)
```

Beschreibung

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

Parameter

text char array

5.15.13 LCD_WriteWord

LCD Funktionen

Syntax

```
void LCD_WriteWord(word value, byte length);

Sub LCD_WriteWord(value As Word, length As Byte)
```

Beschreibung

Schreibt einen vorzeichenlosen Integer (word) mit angegebener Länge auf das LCD Display. Ist die LCD Ausgabe kürzer als die Länge, wird mit Nullen aufgefüllt.

Parameter

value Ausgabewert
length Ausgabelänge

5.16 Mathematik

Mathematische Funktionen

5.16.1 Fließkomma

Im folgenden sind die mathematischen Funktionen aufgeführt, die die C-Control Pro in einfacher Fließkommagenauigkeit (32 Bit) beherrscht. Diese Funktionen sind nicht in der Bibliothek des C-Control Pro 32, da sonst zu wenig Flash Speicher für Benutzerprogramme bleiben würden.

5.16.1.1 FPU (AVR32Bit)

Die AVR32Bit UNIT hat eine Floating Point Einheit (FPU) integriert, die Fließkomma Operationen stark beschleunigt. Eine Ausnahme ist die Fließkomma Division, die in Software durchgeführt wird. Bei einer Division durch eine Konstanten, sollte man daher überlegen lieber mit dem Kehrwert zu multiplizieren.

5.16.1.2 acos

Fließkomma Funktionen

Syntax

```
float acos(float val);
```

```
Sub acos(val As Single) As Single
```

Beschreibung

Der Arcus Cosinus wird berechnet. Der Winkel wird in Radiant angegeben. Ein- und Ausgabewerte liegen zwischen 0 und +pi.

Parameter

val Wert (-1 bis 1) von dem die Funktion berechnet wird

Rückgabewert

Arcus Cosinus des Eingabewertes.

5.16.1.3 asin

Fließkomma Funktionen

Syntax

```
float asin(float val);

Sub asin(val As Single) As Single
```

Beschreibung

Der Arcus Sinus wird berechnet. Der Winkel wird in Radiant angegeben. Ein- und Ausgabewerte liegen zwischen -pi/2 und +pi/2.

Parameter

val Wert (-1 bis 1) von dem die Funktion berechnet wird

Rückgabewert

Arcus Sinus des Eingabewertes.

5.16.1.4 atan

Fließkomma Funktionen

Syntax

```
float atan(float val);

Sub atan(val As Single) As Single
```

Beschreibung

Der Arcus Tangens wird berechnet. Der Winkel wird in Radiant angegeben. Ein- und Ausgabewerte liegen zwischen -pi/2 und +pi/2.

Parameter

val Wert von dem die Funktion berechnet wird

Rückgabewert

Arcus Tangens des Eingabewertes.

5.16.1.5 ceil

Fließkomma Funktionen

Syntax

```
float ceil(float val);

Sub ceil(val As Single) As Single
```

Beschreibung

Der nächst größere Integerwert (ganzzahlige Teil) der Fließkommazahl val wird berechnet.

Parameter

val Wert von dem der Integerwert berechnet wird

Rückgabewert

Ergebnis der Funktion

5.16.1.6 cos

Fließkomma Funktionen

Syntax

```
float cos(float val);
```

```
Sub cos(val As Single) As Single
```

Beschreibung

Der Cosinus wird berechnet. Der Winkel wird in Radiant angegeben.
Parameter

val Wert von dem die Funktion berechnet wird

Rückgabewert

Cosinus des Eingabewertes (-1 bis 1)

5.16.1.7 exp

Fließkomma Funktionen

Syntax

```
float exp(float val);
```

```
Sub exp(val As Single) As Single
```

Beschreibung

Die Funktion e ^ val wird berechnet

Parameter

val Exponent

Rückgabewert

Ergebnis der Funktion

5.16.1.8 fabs

Fließkomma Funktionen

Syntax

```
float fabs(float val);
```

```
Sub fabs(val As Single) As Single
```

Beschreibung

Der Absolutwert der Fließkommazahl wird berechnet.

Parameter

val Eingabewert

Rückgabewert

Ergebnis der Funktion

5.16.1.9 floor

Fließkomma Funktionen

Syntax

```
float floor(float val);
```

```
Sub floor(val As Single) As Single
```

Beschreibung

Der nächst kleinere Integerwert (ganzzahlige Teil) der Fließkommazahl val wird berechnet.

Parameter

val Wert von dem der Integerwert berechnet wird

Rückgabewert

Ergebnis der Funktion

5.16.1.10 ldexp

Fließkomma Funktionen

Syntax

```
float ldexp(float val,int expn);

Sub ldexp(val As Single,expn As Integer) As Single
```

Beschreibung

Die Funktion val * 2 ^ expn wird berechnet

Parameter

val Multiplikator
expn Exponent

Rückgabewert

Ergebnis der Funktion

5.16.1.11 ln

Fließkomma Funktionen

Syntax

```
float ln(float val);

Sub ln(val As Single) As Single
```

Beschreibung

Der natürliche Logarithmus wird berechnet.

Parameter

val Eingabewert

Rückgabewert

Ergebnis der Funktion

5.16.1.12 log

Fließkomma Funktionen

Syntax

```
float log(float val);
```

```
Sub log(val As Single) As Single
```

Beschreibung

Der Logarithmus zur Basis 10 wird berechnet.

Parameter

val Eingabewert

Rückgabewert

Ergebnis der Funktion

5.16.1.13 pow

Fließkomma Funktionen

Syntax

```
float pow(float x, float y);

Sub pow(x As Single, y As Single) As Single
```

Beschreibung

Potenzfunktion. Die Funktion x ^ y wird berechnet

Parameter

x Basis
y Exponent

Rückgabewert

Ergebnis der Funktion

5.16.1.14 round

Fließkomma Funktionen

Syntax

```
float round(float val);

Sub round(val As Single) As Single
```

Beschreibung

Rundungsfunktion. Die Fließkommazahl wird in eine Zahl ohne Nachkommastellen auf- oder abgerundet.

Parameter

val Eingabewert

Rückgabewert

Ergebnis der Funktion

5.16.1.15 sin

Fließkomma Funktionen

Syntax

```
float sin(float val);
```

```
Sub sin(val As Single) As Single
```

Beschreibung

Der Sinus wird berechnet. Der Winkel wird in Radiant angegeben.

Parameter

val Wert von dem die Funktion berechnet wird

Rückgabewert

Sinus des Eingabewertes (-1 bis 1)

5.16.1.16 sqrt

Fließkomma Funktionen

Syntax

```
float sqrt(float val);
```

```
Sub sqrt(val As Single) As Single
```

Beschreibung

Die Quadratwurzel wird berechnet.

Parameter

val Wert von dem die Quadratwurzel berechnet wird

5.16.1.17 tan

Fließkomma Funktionen

Syntax

```
float tan(float val);
```

```
Sub tan(val As Single) As Single
```

Beschreibung

Der Tangens wird berechnet. Der Winkel wird in Radiant angegeben.

Parameter

val Wert von dem die Funktion berechnet wird

Rückgabewert

Tangens des Eingabewertes.

5.16.2 Integer

Mathematische Integer Funktionen.

5.16.2.1 rand

Integer Funktionen

Syntax

```
int rand(void);
```

```
Sub rand() As Integer
```

Beschreibung

Diese Funktion gibt eine Pseudo Zufallszahl zwischen 0 und 32767 zurück. Für unterschiedliche Sequenzen die Funktion srand() mit verschiedenen Anfangswerten füttern.

Rückgabewert

Pseudo Zufallszahl

5.16.2.2 srand

Integer Funktionen

Syntax

```
void srand(int seed);

Sub srand(seed As Integer)
```

Beschreibung

Setzt einen Anfangswert für den Pseudo Zufallszahlengenerator. Mit einem gleichen Anfangswert können die gleichen Sequenzen an Zufallszahlen generiert werden.

Parameter

seed Anfangswert

5.17 OneWire

1-Wire bzw. One-Wire oder Eindraht-Bus ist eine serielle Schnittstelle, die mit einer Datenader auskommt, die sowohl als Stromversorgung als auch als Sende- und Empfangsleitung genutzt wird. Die Daten werden asynchron (ohne Taktsignal) in Blöcken von 64 Bit übertragen. Es können Daten entweder gesendet oder empfangen werden, nicht beides gleichzeitig (Halbduplex).

Das Besondere an 1-Wire-Geräten ist die parasitäre Stromversorgung, wobei die das Gerät über die Datenleitung versorgt wird: Bei inaktiver Kommunikation liegt die Datenleitung auf +5V High-Pegel und lädt einen Kondensator auf. Während der Low-Pulse in der Kommunikation wird der Slave aus seinem Kondensator gespeist. Je nach Ladung des Kondensators können Low-Zeiten bis ca. 960 µs überbrückt werden.

5.17.1 Onewire_Read

1-Wire Funktionen

Syntax

```
byte Onewire_Read(void);

Sub Onewire_Read() As Byte
```

Beschreibung

Ein Byte wird vom Eindraht-Bus gelesen.

Rückgabewert

gelesener Wert vom One-Wire Bus

5.17.2 Onewire_Reset

1-Wire Funktionen

Syntax

```
void Onewire_Reset(byte portbit);
```

```
Sub Onewire_Reset(portbit As Byte)
```

Beschreibung

Es wird auf dem Eindraht-Bus ein Reset ausgelöst. Es wird die Bitnummer des Ports angegeben, über den die Eindraht-Kommunikation geführt wird.

Parameter

portbit Bitnummer des Ports (siehe Port Tabellen)

5.17.3 Onewire_Write

1-Wire Funktionen

Syntax

```
void Onewire_Write(byte data);

Sub Onewire_Write(data As Byte)
```

Beschreibung

Es wird ein Byte auf den Eindraht-Bus geschrieben.

Parameter

data Datenbyte

5.17.4 Onewire Beispiel

CompactC

```
// Beispielprogramm um DS18S20 Temp. Sensor von Dallas Maxim zu lesen
void main(void)
{
    char text[40];
    int ret, i, temp;
    byte rom_code[8];
    byte scratch_pad[9];

    ret= OneWire_Reset(7); // PortA.7
    if(ret == 0)
    {
        text= "Kein Sensor gefunden";
        Msg_WriteText(text);
        goto end;
    }
```

```
    OneWire_Write(0xcc); // ROM überspringen Kommando
    OneWire_Write(0x44); // starte Temperatur Messung Kommando

    AbsDelay(3000);

    OneWire_Reset(7);    // PortA.7
    OneWire_Write(0xcc); // ROM überspringen
    OneWire_Write(0xbe); // lese scratch_pad Kommando
    for(i=0;i<9;i++)     // komplettes scratchpad lesen
    {
        scratch_pad[i]= OneWire_Read();
        Msg_WriteHex(scratch_pad[i]);
    }
    Msg_WriteChar('\r');

    text= "Temperatur: ";
    Msg_WriteText(text);

    temp= scratch_pad[1]*256 + scratch_pad[0];
    Msg_WriteFloat(temp* 0.5);
    Msg_WriteChar('C');
    Msg_WriteChar('\r');

    end:
}
```

BASIC

```
' Beispielprogramm um DS18S20 Temp. Sensor von Dallas Maxim zu lesen
Dim Text(40) As Char
Dim ret,i As Integer
Dim temp As Integer
Dim rom_code(8) As Byte
Dim scratch_pad(9) As Byte

Sub main()

    ret = OneWire_Reset(7) ' PortA.7

    If ret = 0 Then
       Text= "Kein Sensor gefunden"
       Msg_WriteText(Text)
       Goto Ende
    End If

    OneWire_Write(0xcc)   ' ROM überspringen Kommando
    OneWire_Write(0x44)   ' starte Temperatur Messung Kommando

    AbsDelay(3000)

    OneWire_Reset(7)      ' PortA.7
```

```
    OneWire_Write(0xcc)   ' ROM überspringen Kommando
    OneWire_Write(0xbe)   ' lese scratch_pad Kommando

    For i = 0 To 9        ' komplettes scratchpad lesen
        scratch_pad(i)= OneWire_Read()
        Msg_WriteHex(scratch_pad(i))
    Next
    Msg_WriteChar(13)

    Text = "Temperatur: "
    Msg_WriteText(Text)

    temp = scratch_pad(1) * 256 + scratch_pad(0)
    Msg_WriteFloat(temp * 0.5)
    Msg_WriteChar(99)
    Msg_WriteChar(13)

    Lab Ende
End Sub
```

5.18 Port

Atmel Mega

Der Atmel Mega 32 hat 4 Ein-/Ausgabeports zu je 8 Bit. Der Atmel Mega 128 hat 6 Ein-/Ausgabeports zu je 8 Bit und ein Ein-/Ausgabeport zu 5 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.

➔ 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 zur 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.

Atmel AVR32Bit

Im Atmel AVR32Bit gibt es die Ports A bis D, die jeweils 32-Bit breit sind. Jedes Bit der einzelnen Ports kann als Eingang oder als Ausgang konfiguriert werden. Zusätzlich ist es möglich einen PullUp, PullDown, und die Drive Strength einzustellen. Beim AVR32Bit wurden die vom Atmel Mega

bekannten Port_DataDir, Port_Toggle und Port_Write Funktionen weggelassen, da in der Praxis ein Schreiben der kompletten 32-Bit eines Ports sehr unhandlich ist.

➔ Es ist wichtig, vor der Programmierung die Pinzuordnung zu studieren, da wichtige Peripherie Funktionen auf bestimmten Ports liegen. Werden diese Ports umprogrammiert, kann es passieren, daß die Entwicklungsumgebung keine Programme mehr zur C-Control Pro übertragen kann.

➔ Beim AVR32 die Port_Attribute Funktion anstatt wie beim Atmel Mega Port_DataDirBit benutzen, um zwischen Eingang und Ausgang umzuschalten.

➔ Wird bei einem Portpin eine Funktion wie z.B. PWM nur temporär benutzt, empfiehlt es sich meist, nach der Abschaltung dem Pin wieder mit Port_Attribute einen definierten Pegel zu geben.

5.18.1 Port_Attribute (AVR32Bit)

Port Funktionen

Syntax

```
void Port_Attribute(byte portbit, word attribute);

Sub Port_Attribute(portbit As Byte, attribute As Word)
```

Beschreibung

Die Funktion Port_Attribute konfiguriert die Eigenschaften eines Portbits. Mehrere Eigenschaften können als Wert für attribute zusammenoderiert werden. Siehe Beispiel.

Parameter

portbit Bitnummer des Ports (siehe Port Port Tabellen)
attribute Portbit Attribut

Attribut Tabelle

Funktion	Definition	Wert (Hex)
Port auf Input	PORT_ATTR_INPUT	00
Port auf Output	PORT_ATTR_OUTPUT	01
Portausgang low	PORT_ATTR_INIT_LOW	00
Portausgang high	PORT_ATTR_INIT_HIGH	02
PullUp gesetzt	PORT_ATTR_PULL_UP	04
PullDown gesetzt	PORT_ATTR_PULL_DOWN	08
Drive Strength minimal	PORT_ATTR_DRIVE_MIN	00
Drive Strength normal	PORT_ATTR_DRIVE_LOW	10
Drive Strength hoch	PORT_ATTR_DRIVE_HIGH	20
Drive Strength maximal	PORT_ATTR_DRIVE_MAX	30

➔ Für exakte Werte der Drive Strength eines Portpin bitte im Atmel AT32UC3C Datenblatt das Kapitel "Electrical Characteristics" lesen.

5.18.2 Port_DataDir (Mega)

Port Funktionen **Beispiel**

Syntax

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

Sub Port_DataDir(port As Byte, val As Byte)
```

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 = 02, dann wird der Pin 2 des Atmel Mega (gleich PortB.1 - siehe Pinzuordnung von M32 und M128) auf Ausgang konfiguriert.

Parameter

port Portnummer (siehe Port Tabellen)
val Ausgabe byte

5.18.3 Port_DataDirBit (Mega)

Port Funktionen

Syntax

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

Sub Port_DataDirBit(portbit As Byte, val As Byte)
```

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 von M32 und M128) auf Eingang konfiguriert.

➔ Port Bit Zugriffe sind immer deutlich langsamer als die normalen Port Zugriffe die 8 Bit transferieren. Wenn man den gewünschten Wert aller Bits eines Ports kennt, sollte man immer 8-Bit Zugriffe machen.

➔ Bitte beim AVR32Bit die Port_Attribute Funktion anstatt Port_DataDirBit benutzen. Die AVR32 MCU bietet erweiterte Möglichkeiten wie PullDown oder den Drive Strength einzustellen.

Parameter

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

5.18.4 Port_Read (Mega)

Port Funktionen

Syntax

```
byte Port_Read(byte port);

Sub Port_Read(port As Byte) As Byte
```

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.

Parameter

port Portnummer (siehe Port Tabellen)

Rückgabewert

Wert des Ports

5.18.5 Port_ReadBit

Port Funktionen

Syntax

```
byte Port_ReadBit(byte port);

Sub Port_ReadBit(port As Byte) As Byte
```

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 von AVR32, M32 und M128.)

Bei der C-Control Pro Mega Serie sind Port Bit Zugriffe sind immer deutlich langsamer als die norma-

len Port Zugriffe die 8 Bit transferieren. Wenn man den gewünschten Wert aller Bits eines Ports kennt, sollte man immer 8-Bit Zugriffe machen.

Parameter

portbit Bitnummer des Ports (siehe Port Tabellen)

Rückgabewert

Bitwert des Ports (0 oder 1)

5.18.6 Port_Toggle (Mega)

Port Funktionen

Syntax

```
void Port_Toggle(byte port);

Sub Port_Toggle(port As Byte)
```

Beschreibung

Invertiert die Bits auf dem 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 wird der interne Pullup Widerstand invertiert. (Für die Abbildung zwischen Portbits und den Pins des Atmel Mega Chips siehe Pinzuordnung von M32 und M128).

Parameter

port Portnummer (siehe Port Tabellen)

5.18.7 Port_ToggleBit

Port Funktionen

Syntax

```
void Port_ToggleBit(byte portbit);

Sub Port_ToggleBit(portbit As Byte)
```

Beschreibung

Die Funktion Port_ToggleBit invertiert den Wert eines Pins, der auf Ausgang geschaltet ist. (Für die Abbildung zwischen Portbits und den Pins des Atmel Mega Chips siehe Pinzuordnung von AVR32, M32 und M128).

➔ Mega: Ist ein Pin auf Eingang geschaltet, so wird der interne Pullup Widerstand umgeschaltet.

➔ Mega: Port Bit Zugriffe sind immer deutlich langsamer als die normalen Port Zugriffe die alle 8 Bit verändern. Wenn man den gewünschten Wert aller Bits eines Ports kennt, sollte man immer 8-Bit Zugriffe machen.

Parameter

portbit Bitnummer des Ports (siehe Port Tabellen)

5.18.8 Port_Write (Mega)

Port Funktionen **Beispiel**

Syntax

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

Sub Port_Write(port As Byte, val As Byte)
```

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 von M32 und M128).

➔ In älteren IDE Versionen waren PORT_ON und PORT_OFF falsch definiert, das ist jetzt korrigiert.

Parameter

port Portnummer (siehe Port Tabellen)

5.18.9 Port_WriteBit

Port Funktionen

Syntax

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

Sub Port_WriteBit(portbit As Byte, val As Byte)
```

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 von AVR32, M32 und M128).

➔ Bei der C-Control Pro Mega Serie sind Port Bit Zugriffe sind immer deutlich langsamer als die normalen Port Zugriffe die 8 Bit transferieren. Wenn man den gewünschten Wert aller Bits eines Ports kennt, sollte man immer 8-Bit Zugriffe machen.

➔ Beim C-Control Pro AVR32Bit wird der interne PullUp über das Kommando Port_Attribute() geschaltet.

➔ In älteren IDE Versionen waren PORT_ON und PORT_OFF falsch definiert, das ist jetzt korrigiert.

Parameter

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

5.18.10 Port Tabellen

Portnummern Tabelle Mega32 und Mega128 (CAN)

Die Port Namen sind als #defines in der IntFunc_Lib.cc gespeichert.

Definition	Wert
PortA	0
PortB	1
PortC	2
PortD	3
PortE (Mega128)	4
PortF (Mega128)	5
PortG (Mega128)	6

Portbits Tabelle Mega32 und Mega128 (CAN)

Die Portbit Namen sind als #defines in der IntFunc_Lib.cc gespeichert.

Definition	Portbit	Portbit Name
PortA.0	0	PA0
...	...	...
PortA.7	7	PA7
PortB.0	8	PB0
...	...	...
PortB.7	15	PB7
PortC.0	16	PC0
...	...	...
PortC.7	23	PC7

PortD.0	24	PD0
...	...	...
PortD.7	31	PD7
ab hier nur Mega128		
PortE.0	32	PE0
...	...	...
PortE.7	39	PE7
PortF.0	40	PF0
...	...	...
PortF.7	47	PF7
PortG.0	48	PG0
...	...	...
PortG.4	52	PG4

Portbits Tabelle AVR32Bit

Die Portbit Namen sind als #defines in der IntFunc_Lib.cc gespeichert.

AVR32 Definition	Portbit	Portbit Name
PA00	0	PA00
...	...	...
PA31	31	PA31
PB00	32	PB00
...	...	...
PB31	63	PB31
PC00	64	PC00
...	...	...
PC31	95	PC31
PD00	96	PD00
...	...	...
PD31	127	PD31

AVR32 Application Board Port Tabelle

Die Board Port Namen und auch die Portbit Namen sind als #defines in der IntFunc_Lib.cc gespeichert.

Board Port Name	Portbit	Portbit Name
P1	0	PA00
P2	1	PA01
P3	2	PA02
P4	3	PA03
P5	36	PB04
P6	37	PB05
P7	38	PB06
P8	16	PA16
P9	4	PA04
P10	5	PA05

P11	6	PA06
P12	7	PA07
P13	8	PA08
P14	9	PA09
P15	10	PA10
P16	11	PA11
P17	19	PA19
P18	20	PA20
P19	21	PA21
P20	22	PA22
P21	23	PA23
P22	24	PA24
P23	25	PA25
P24	13	PA13
P25	12	PA12
P26	14	PA14
P27	15	PA15
P28	52	PB20
P29	53	PB21
P30	54	PB22
P31	55	PB23
P32	65	PC01
P33	70	PC06
P34	68	PC04
P35	69	PC05
P36	81	PC17
P37	82	PC18
P38	79	PC15
P39	80	PC16
P40	83	PC19
P41	84	PC20
P42	76	PC12
P43	75	PC11
P44	77	PC13
P45	78	PC14
P46	85	PC21
P47	86	PC22
P48	87	PC23
P49	88	PC24
P50	95	PC31
P51	103	PD07
P52	104	PD08
P53	117	PD21
P54	118	PD22
P55	119	PD23
P56	51	PB19
P57	34	PB02

5.18.11 Port Beispiel (Mega)

```
// Programm läßt abwechselnd die beiden LEDs auf dem
// Application Board im Sekunden Rhythmus blinken
// C-Control Pro Mega
void main(void)
{
    Port_DataDirBit(PORT_LED1,PORT_OUT);
    Port_DataDirBit(PORT_LED2,PORT_OUT);

    while(true)  // Endlosschleife
    {
        Port_WriteBit(PORT_LED1,PORT_ON);
        Port_WriteBit(PORT_LED2,PORT_OFF);
        AbsDelay(1000);
        Port_WriteBit(PORT_LED1,PORT_OFF);
        Port_WriteBit(PORT_LED2,PORT_ON);
        AbsDelay(1000);
    }
}
```

5.18.12 Port Beispiel (AVR32Bit)

Alle drei Programmbeispiele lassen LED1 leuchten solange Taster T1 gedrückt wird. Die Beispiele unterscheiden sich in der Adressierung der Port Namen. Wenn der Taster nicht gedrückt wird, so wird vom Port eine "1" gelesen, da auf dem Applicationboard jeder Taster mit einem Pull-Up Widerstand verbunden ist.

```
// Beispiel mit Function Name defines
void main(void)
{
    Port_Attribute(PORT_LED1, PORT_ATTR_OUTPUT | PORT_ATTR_INIT_LOW);
    Port_Attribute(PORT_T1, PORT_ATTR_INPUT);

    while(true)  // Endlosschleife
    {
        if(Port_ReadBit(PORT_T1))
        {
            Port_WriteBit(PORT_LED1, PORT_OFF);
        }
        else
        {
            Port_WriteBit(PORT_LED1, PORT_ON);
        }
    }
}
```

```
// Beispiel mit Unit Name defines
void main(void)
{
    Port_Attribute(P48, PORT_ATTR_OUTPUT | PORT_ATTR_INIT_LOW);
    Port_Attribute(P41, PORT_ATTR_INPUT);

    while(true)  // Endlosschleife
    {
        if(Port_ReadBit(P41))
        {
            Port_WriteBit(P48, PORT_OFF);
        }
        else
        {
            Port_WriteBit(P48, PORT_ON);
        }
    }
}

// Beispiel mit AVR32 Portname defines
void main(void)
{
    Port_Attribute(PC23, PORT_ATTR_OUTPUT | PORT_ATTR_INIT_LOW);
    Port_Attribute(PC20, PORT_ATTR_INPUT);

    while(true)  // Endlosschleife
    {
        if(Port_ReadBit(PC20))
        {
            Port_WriteBit(PC23, PORT_OFF);
        }
        else
        {
            Port_WriteBit(PC23, PORT_ON);
        }
    }
}
```

5.19 RC5

RC-5 ist der Fernbedienungscode der Firma Philips, der auch von einigen wenigen anderen Herstellern, z.B. Marantz oder auch Hauppauge (TV-Karten für PC), verwendet wird.
Jeder RC-5-Code besteht aus 14 Bit, die nacheinander an den Empfänger übertragen werden.

Das waren ursprünglich:

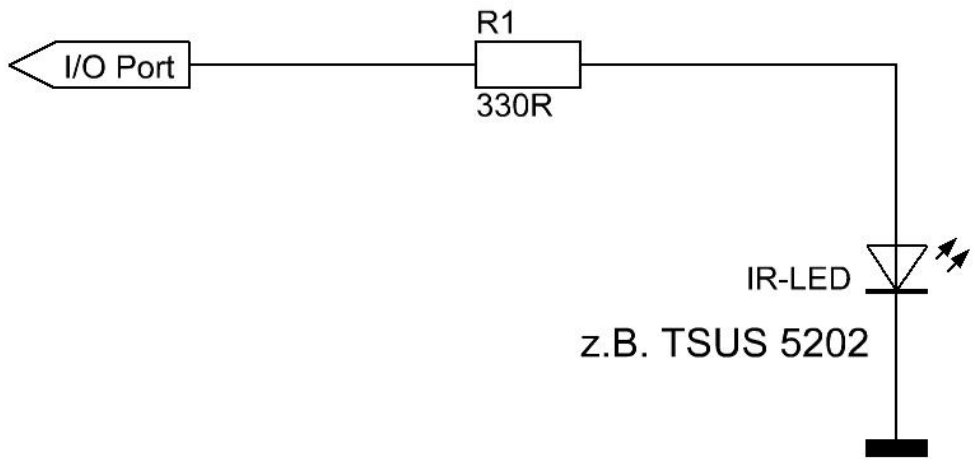
- 2 Startbits (immer "1")
- ein Togglebit (abwechselnd "1" oder "0")
- 5 Adressbits
- 6 Kommandobits

Die Startbits dienen dem Infrarotempfänger zur Synchronisation mit der Übertragung, sowie dazu, seine Verstärkungsregelung auf das Signal einzustellen. Das Togglebit ändert seinen Wert bei jedem Tastendruck. Dadurch kann man das lange Drücken einer Taste (und damit das wiederholte Senden eines Befehls) vom wiederholten Drücken derselben Taste unterscheiden. In den Adressbits ist das zu steuernde Gerät kodiert. Es können also 32 verschiedene Geräte gesteuert werden. Die Kommandobits enthalten das Kommando, das an das adressierte Gerät versendet wird. Damit können erst einmal 64 verschiedene Kommandos pro Gerät übertragen werden. Irgendwann fiel auf, das 64 Befehle für komplizierte Geräte etwas wenig sein könnten. Man brauchte ein weiteres Kommandobit. Im Interesse weitestgehender Kompatibilität, entschied man sich dafür, das zweite Startbit nun nicht mehr als Startbit, sondern als invertiertes 7. Kommandobit zu nutzen. Für die ersten 64 Kommandos ist es "1", als wäre es ein Startbit, für die 64 neuen Kommandos ist es "0". Damit sind nun für jedes Gerät je 128 unterschiedliche Befehle möglich.

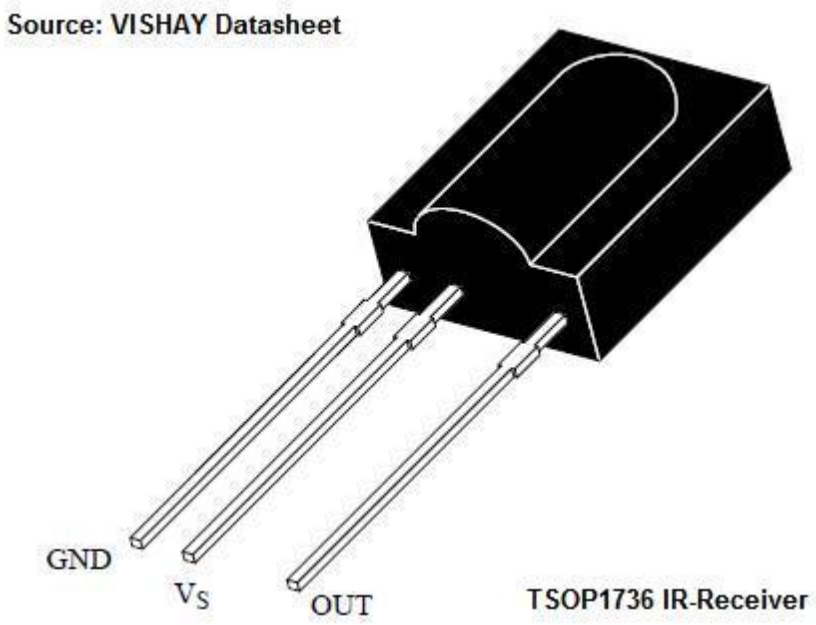
Wie werden nun die einzelnen Bits übertragen?

In der C-Control Pro werden an den Pin an den die IR-Diode angeschlossen und konfiguriert wurde, ca. 36 KHz als Trägerfrequenz erzeugt. Die Sendepulse sind 6,9444 µs lang. Zwischen den Sendepulsen ist jeweils eine Pause von 20,8332 µs. Für ein Bit mit dem Wert "1" wird der Sendergenerator für 889 µs ausgeschaltet und anschließend für 889 µs eingeschaltet das entspricht 32 IR-Impulse. Ein "0"-Bit beginnt dagegen mit 889 µs Sendezeit (32 IR-Impulse), gefolgt von 889 µs Pause. Folglich dauert ein Bit 1,778 ms und die Übertragung eines kompletten 14-Bit Datenworts 24,889 ms. Falls man die Taste auf der Fernbedienung gedrückt hält, wird das Datenwort alle 113,778 ms wiederholt dies entspricht der Dauer von 64 Bit.

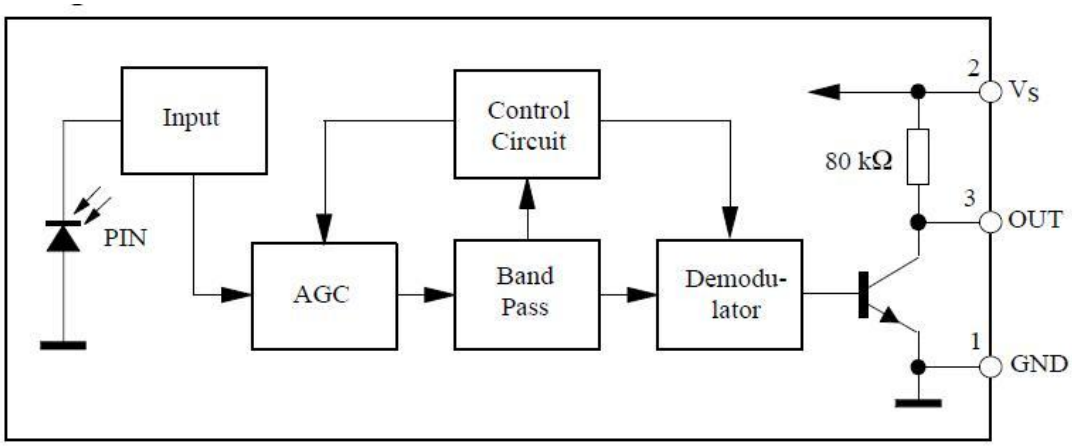
Anschluß an C-Control Pro (Sender Diode)



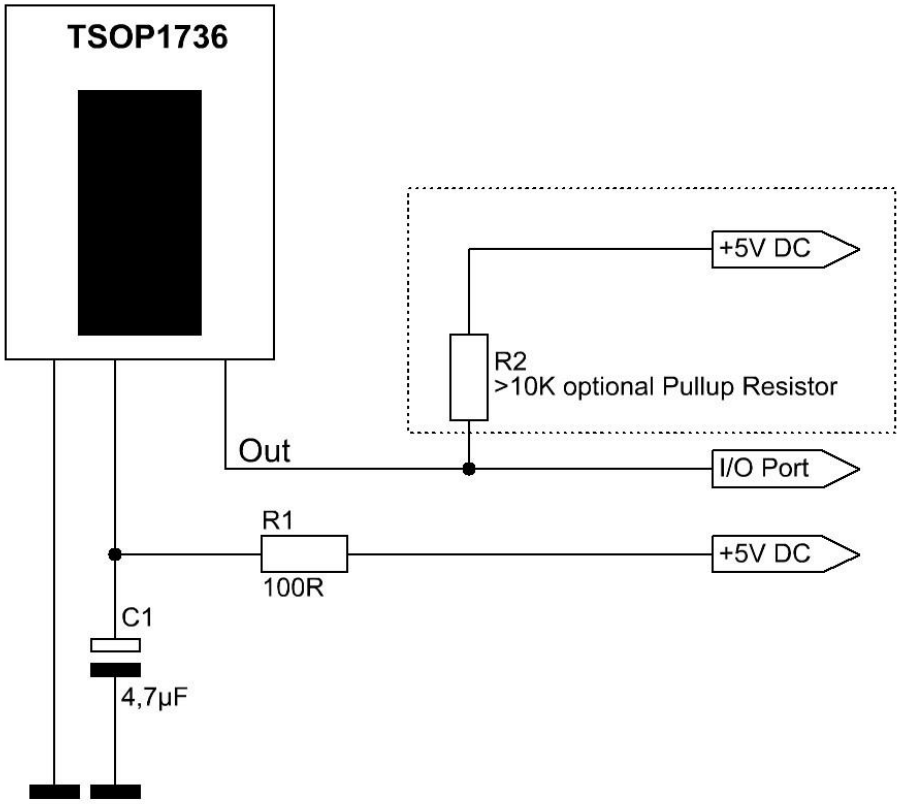
Anschluß an C-Control Pro (Empfänger)



Pinbelegung des TSOP1736 IR-Empfängers



Technologischer Aufbau des Empfängers



Externe Beschaltung des Empfängers zum Anschluss an die C-Control Pro

5.19.1 RC5_Init

RC5 Funktionen

Syntax

```
void RC5_Init(byte pin);
```

```
Sub RC5_Init(pin As Byte)
```

Beschreibung

Es wird der Portpin definiert, auf dem RC5 Kommandos empfangen oder gesendet werden.

Parameter

pin Bitnummer des Ports (siehe Port Tabellen)

5.19.2 RC5_Read

RC5 Funktionen

Syntax

```
word RC5_Read(void);
```

```
Sub RC5_Read() As Word
```

Beschreibung

Es werden auf dem mit RC5_Init() angegebenen Portpin empfangenen 14 Bit des RC5 Kommandos zurückgeliefert. Wird kein Signal empfangen, so wartet die Leseroutine bis zu 130ms, bis sie zurückkehrt. Diese Wartezeit ist länger als die 113ms die als Abstand zwischen 2 Wiederholungen eines Kommandos einer RC5 IR Fernbedienung definiert sind. Ein Rückgabewert von 0 signalisiert, das kein RC5 Signal empfangen wurde.

➔ Diese Funktion erkennt nicht, wenn ein anderes Format als RC5 benutzt wurde. Es werden dann im Zweifelsfall falsche Werte zurückgegeben.

Rückgabewert

14 Bit des empfangenen RC5 Kommandos

5.19.3 RC5_Write

RC5 Funktionen

Syntax

```
void RC5_Write(word data);
```

```
Sub RC5_Write(data As Word)
```

Beschreibung

Die 14 Bit eines RC5 Kommando werden auf den Portpin gesendet der mit RC5_Init() definiert wurde.

➔ Zum Treiben der Infrarot LED wird der Ausgangsport auf maximalen Drive Strength gesetzt. Aber nicht alle Portpin haben diese Ausgangsleistung. Siehe AVR32Bit Modul.

Parameter

data auszugebendes RC5 Datenwort

5.20 RS232

➔ Wenn man die "gepollten" seriellen Routinen benutzt, besteht, insbesondere bei hohen Baudraten, die Möglichkeit, das Zeichen nicht empfangen werden. Um dies zu verhindern bitte Serial_Init_IRQ() anstelle von Serial_Init() benutzen.

Mega

Die serielle Schnittstelle kann mit Geschwindigkeiten bis zu 230.4kbaud betrieben werden. Bei den Funktionen für die serielle Schnittstelle gibt der erste Parameter die Portnummer an (0 oder 1). Beim Mega32 steht nur eine serielle Schnittstelle zur Verfügung (0), für den Mega128 zwei (0, 1).

AVR32Bit

Die C-Control Pro AVR32Bit unterstützt 3 serielle Schnittstellen mit Geschwindigkeiten von bis zu 460.8kbaud. Die seriellen Schnittstellen sind von 0 bis 2 durchnummeriert. Diese Nummern sind abweichend von dem wie Atmel die Schnittstellen im Datenblatt benennt:

C-Control AVR32Bit	Atmel AVR32
0	USART0
1	USART3
2	USART4

5.20.1 Divider (Mega)

Die Funktionen Serial_Init() und Serial_Init_IRQ bekommen als Baudratenparameter einen Teiler (divider) der den Baudratentakt aus dem Prozessortakt ableitet. Der Prozessortakt beträgt 14,7456 MHz bei Mega32, Mega128 und 16 MHz bei dem Mega128 CAN.

Laut dem Atmel Prozessor Handbuch wird folgende Formel angewendet um einen divider für eine Baudrate zu ermitteln:

divider = (Prozessortakt / Baudrate / 16) -1

Beispiel: 15 = (14745600 / 57600 / 16) -1

➔ Es ist schwieriger aus dem 16 MHz Takt des Mega128 CAN die Standard Baudraten abzuleiten. Daher sind die divider Tabellen der Prozessoren leicht unterschiedlich.

DoubleClock Modus

Wenn man das High-Bit setzt, wird der DoubleClock Modus eingeschaltet. Man muss dann denn doppelten Wert als divider eintragen. Für 57600 Baud kann man z.B. statt 0f Hex (dezimal 15) auch 801e Hex (= 8000 Hex + 2 * 15) benutzen. Für MIDI (31250 Baud) bekäme man einen divider = (14745600 / 31250 / 16) -1 = 28,49 . Da man nur ganzzahlige Werte übergeben kann, bekommt man im DoubleClock Modus einen besseren Wert: 8039 Hex (= 8000 Hex + 2 * 28,5).

Tabelle divider Definitionen 14,7456 MHz (Mega32, Mega128):

divider	Definition	Baudrate
3071	SR_BD300	300bps
1535	SR_BD600	600bps
767	SR_BD1200	1200bps
383	SR_BD2400	2400bps
191	SR_BD4800	4800bps
95	SR_BD9600	9600bps
63	SR_BD14400	14400bps
47	SR_BD19200	19200bps
31	SR_BD28800	28800bps
8039 (Hex)	SR_BDMIDI	31250bps
23	SR_BD38400	38400bps
15	SR_BD57600	57600bps
11	SR_BD76800	76800bps
7	SR_BD115200	115200bps
3	SR_BD230400	230400bps

Tabelle divider Definitionen 16 MHz (Mega128 CAN):

divider	Definition	Baudrate
3332	SR_BD300	300bps
1666	SR_BD600	600bps
832	SR_BD1200	1200bps
416	SR_BD2400	2400bps
207	SR_BD4800	4800bps
103	SR_BD9600	9600bps
68	SR_BD14400	14400bps
51	SR_BD19200	19200bps
34	SR_BD28800	28800bps
31	SR_BDMIDI	31250bps
25	SR_BD38400	38400bps
8022 (Hex)	SR_BD57600	57600bps
12	SR_BD76800	76800bps
6	SR_BD125000	125000bps
3	SR_BD250000	250000bps

5.20.2 Serial_Disable

Serielle Funktionen

Syntax

```
void Serial_Disable(byte serport);
```

```
Sub Serial_Disable(serport As Byte)
```

Beschreibung

Die serielle Schnittstelle wird abgeschaltet und die dazugehörigen Ports können anders verwendet werden.

Parameter

serport Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)

5.20.3 Serial_Init (Mega)

Serielle Funktionen **Beispiel**

Syntax

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

Sub Serial_Init(serport As Byte, par As Byte, divider As Word)
```

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_2STOP | SR_EVEN_PAR" für 7 Bit pro Zeichen, 2 Stop Bit und gerade Parität (siehe auch Beispiel). Diese Werte sähen in BASIC Syntax wie folgt aus: "SR_7BIT Or SR_2STOP Or SR_EVEN_PAR". Die Baudrate wird als Teilerwert angegeben, wie in der Tabelle spezifiziert.

➔ Wenn man die "gepollten" seriellen Routinen benutzt, besteht, insbesondere bei hohen Baudraten, die Möglichkeit, das Zeichen nicht empfangen werden. Um dies zu verhindern bitte Serial_Init_IRQ() anstelle von Serial_Init() benutzen.

➔ Man kann den DoubleClock Modus des Atmel AVR einschalten. Dies geschieht, wenn das High Bit im Teiler gesetzt wird. Beim DoubleClock muss gegenüber der normalen Tabelle der Teiler verdoppelt werden, um die gleiche Baudrate zu erhalten. Dafür sind dann "krumme" Baudraten besser einstellbar. Z.B. MIDI: Der neue Wert SB_MIDI (=803a Hex) liegt jetzt sehr nahe an 31250baud. Beispiel für 19200baud: Der Teiler für 19200baud ist 002f (Hex). Für den DoubleClock Modus verdoppelt man nun den Teiler (= 005e Hex). Nun das Hi-Bit setzen, und man kann anstatt 2f (Hex) auch 805e (Hex) für 19200baud benutzen .

Parameter

serport Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)
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

5.20.4 Serial_Init (AVR32Bit)

Serielle Funktionen Beispiel

Syntax

```
void Serial_Init(byte serport, byte par, dword baud);

Sub Serial_Init(serport As Byte, par As Byte, baud As ULong)
```

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_2STOP | SR_EVEN_PAR" für 7 Bit pro Zeichen, 2 Stop Bit und gerade Parität (siehe auch Beispiel). Diese Werte sähen in BASIC Syntax wie folgt aus: "SR_7BIT Or SR_2STOP Or SR_EVEN_PAR".

Parameter

serport	Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)
par	Schnittstellenparameter (siehe Tabelle)
baudrate	Baudrate

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

5.20.5 Serial_Init_IRQ (Mega)

Serielle Funktionen **Beispiel**

Syntax

```
void Serial_Init_IRQ(byte serport, byte ramaddr[], byte recvlen,
                     byte sendlen, byte par, word divider);

Sub Serial_Init_IRQ(serport As Byte,ByRef ramaddr As Byte,recvlen As Byte,
                    sendlen As Byte, par As Byte, divider As Word)
```

Beschreibung

Die serielle Schnittstelle wird für die Benutzung im Interrupt Modus initialisiert. Der Anwender muss eine **globale** Variable als Puffer bereitstellen. In diesem Puffer werden die empfangenen Daten, sowie die zu sendenden Daten abgelegt. Für die Größe des Puffers gibt es ein #define **SERIAL_BUF**. Möchte man ein byte Array definieren mit X bytes Empfangspuffer und Y bytes Sendepuffer, schreibt man `"byte buf [SERIAL_BUF(X,Y)];"` (siehe auch Beispiel). Der Sende- und der Empfangspuffer kann maximal 255 Zeichen aufnehmen.

Für par wird der Wert durch Oderieren der vordefinierten Bitwerte zusammengestellt. Man oderiert erst Zeichenlänge, dann Anzahl der Stopbits und dann Parity. Z.B. "SR_7BIT | SR_2STOP | SR_EVEN_PAR" für 7 Bit pro Zeichen, 2 Stop Bit und gerade Parität. Diese Werte sähen in BASIC Syntax wie folgt aus: "SR_7BIT Or SR_2STOP Or SR_EVEN_PAR". Die Baudrate wird als Teilerwert angegeben, wie in der Tabelle spezifiziert.

➔ Der vom Benutzer zur Verfügung gestellte RAM Puffer muss während der Benutzung der seriellen Schnittstelle reserviert bleiben. Da lokale Variablen nach Verlassen der Funktion freigegeben werden, ist es meistens sinnvoll den Puffer als globale Variable zu deklarieren.

➔ Man kann den DoubleClock Modus des Atmel AVR einschalten. Siehe das Kapitel Divider.

➔ Wenn im seriellen Interrupt Modus gearbeitet wird, immer Serial_ReadExt() benutzen. Serial_Read() funktioniert nur im normalen (polled) Modus.

Parameter

serport	Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)
ramaddr	Adresse des Puffers
recvlen	Größe des Empfangspuffers
sendlen	Größe des Sendepuffers

par Schnittstellenparameter (siehe Tabelle)
div 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

5.20.6 Serial_Init_IRQ (AVR32Bit)

Serielle Funktionen Beispiel

Syntax

```
void Serial_Init_IRQ(byte serport, byte ramaddr[], word recvlen,
                     word sendlen, byte par, dword baud);

Sub Serial_Init_IRQ(serport As Byte,ByRef ramaddr As Byte,recvlen As Word,
                    sendlen As Word, par As Byte, baud As ULong)
```

Beschreibung

Die serielle Schnittstelle wird für die Benutzung im Interrupt Modus initialisiert. Der Anwender muss eine **globale** Variable als Puffer bereitstellen. In diesem Puffer werden die empfangenen Daten, sowie die zu sendenden Daten abgelegt. Für die Größe des Puffers gibt es ein #define **SERIAL_BUF**. Möchte man ein byte Array definieren mit X bytes Empfangspuffer und Y bytes Sendepuffer, schreibt man `"byte buf [SERIAL_BUF(X,Y)];"` (siehe auch Beispiel). Der Sende- und der Empfangspuffer kann jeweils bis zu 65535 Zeichen aufnehmen, ist aber natürlich durch die Grenze des RAMs beschränkt.

Für par wird der Wert durch Oderieren der vordefinierten Bitwerte zusammengestellt. Man oderiert erst Zeichenlänge, dann Anzahl der Stopbits und dann Parity. Z.B. "SR_7BIT | SR_2STOP | SR_EVEN_PAR" für 7 Bit pro Zeichen, 2 Stop Bit und gerade Parität. Diese Werte sähen in BASIC Syntax wie folgt aus: "SR_7BIT Or SR_2STOP Or SR_EVEN_PAR".

➔ Der vom Benutzer zur Verfügung gestellte RAM Puffer muss während der Benutzung der seriellen Schnittstelle reserviert bleiben. Da lokale Variablen nach Verlassen der Funktion freigegeben werden, ist es meistens sinnvoll den Puffer als globale Variable zu deklarieren.

➔ Wenn im seriellen Interrupt Modus gearbeitet wird, immer Serial_ReadExt() benutzen. Serial_Read() funktioniert nur im normalen (polled) Modus.

Parameter

serport	Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)
ramaddr	Adresse des Puffers
recvlen	Größe des Empfangspuffers
sendlen	Größe des Sendepuffers
par	Schnittstellenparameter (siehe Tabelle)
baud	Baudrate

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

5.20.7 Serial_IRQ_Info

Serielle Funktionen

Syntax

```
byte Serial_IRQ_Info(byte serport, byte info);

Sub Serial_IRQ_Info(serport As Byte, info As Byte) As Byte
```

Beschreibung

Abhängig vom Parameter info wird zurückgegeben, wieviele Bytes empfangen wurden, oder in den Sendepuffer geschrieben wurden.

Parameter

serport Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)

info Werte:

RS232_FIFO_RECV (0) Zeichen im Empfangspuffer
RS232_FIFO_SEND (1) Zeichen in den Sendepuffer geschrieben

Rückgabewert

in Bytes

5.20.8 Serial_Read (Mega)

Serielle Funktionen

Syntax

```
byte Serial_Read(byte serport);

Sub Serial_Read(serport As Byte) As Byte
```

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.

➔ Wenn im seriellen Interrupt Modus gearbeitet wird, immer Serial_ReadExt() benutzen. Serial_Read() funktioniert nur im normalen (polled) Modus.

➔ Die Funktion wird in der AVR32Bit nicht mehr unterstützt, da sie, wenn keine Daten empfangen werden, endlos wartet. Z.B. würden auch keine eingehenden Ethernetpakete mehr bearbeitet.

Parameter

serport Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)

Rückgabewert

empfangenes byte aus der seriellen Schnittstelle

5.20.9 Serial_ReadExt

Serielle Funktionen

Syntax

```
word Serial_ReadExt(byte serport);

Sub Serial_ReadExt(serport As Byte) As Word
```

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 der Wert 256 (100 Hex) zurückgegeben.

➔ Wenn im seriellen Interrupt Modus gearbeitet wird, immer Serial_ReadExt() benutzen. Serial_Read() funktioniert nur im normalen (polled) Modus.

Parameter

serport Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)

Rückgabewert

empfangenes byte aus der seriellen Schnittstelle
256 (100 Hex) kein Zeichen in der Schnittstelle

5.20.10 Serial_Write

Serielle Funktionen **Beispiel**

Syntax

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

Sub Serial_Write(serport As Byte, val As Byte)
```

Beschreibung

Ein byte wird zur seriellen Schnittstelle geschickt.

Parameter

serport Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)
val der auszugebende byte Wert

5.20.11 Serial_WriteText

Serielle Funktionen

Syntax

```
void Serial_WriteText(byte serport, char text[]);

Sub Serial_WriteText(serport As Byte, ByRef Text As Char)
```

Beschreibung

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

Parameter

serport Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)
text char array

5.20.12 Serial Beispiel

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

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

    while(str[i]) Serial_Write(0,str[i++]);  // Gib den String aus
}
```

5.20.13 Serial Beispiel (IRQ)

```
// 35 byte Sende + Empfungspuffer + SR_BUF byte interne FIFO Verwaltung
byte buffer[SERIAL_BUF(20,15)];                 // Array Deklarierung

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

    str="test";
    i=0;
    // Init Schnittstelle mit 19200baud, 8 Bit, 1 Stop Bit, keine Parität
    // 20 byte Empfangspuffer - 15 byte Sendepuffer
    Serial_Init_IRQ(0,buffer,20,15,SR_8BIT|SR_1STOP|SR_NO_PAR,SR_BD19200);

    while(str[i]) Serial_Write(0,str[i++]);  // Gib den String aus
    while(1);  // Endlosschleife
}
```

5.21 SDCard

SD-Card Support bei C-Control Pro AVR32Bit

Der Kartenhalter für Micro SD-Cards ist direkt unter der C-Control Pro AVR32Bit Unit. Siehe Beschreibung der AVR32Bit Unit. Für die Beschreibung der benutzten Signale bitte die Pinzuordnung konsultieren. Im Gegensatz zum Mega SD-Card Interface gibt es keine Enable Leitung über die ein Reset ausgelöst werden kann. In den Demoprogrammen ist dieser Teil für die AVR32Bit Unit auskommentiert.

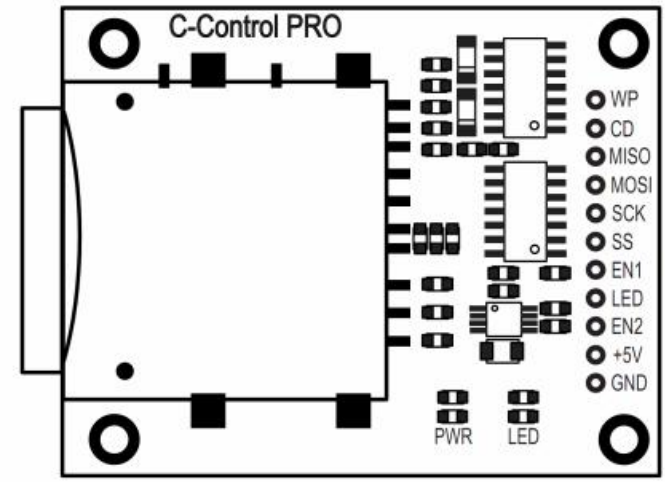
SD-Card Support bei C-Control Pro Mega 128 und Mega 128 CAN

Das C-Control Pro SD-Card Interface (Conrad Artikel-Nr. 197220) dient zum Anbinden eines Mikrocontrollers wie z.B. die C-Control Mega 128 Unit (Conrad Artikel-Nr. 198219) an eine 3.3V SD-Karte. Die SD-Card Erweiterung besitzt einen Pegelkonverter, welcher die Signale bidirektional konvertiert und somit eine direkte Anbindung der SD-Karte an einen 5V Mikrocontroller ermöglicht. Es können alle zur Zeit am Markt befindlichen Speicherkarten wie z.B. SD, SDHC, MMC und auch andere Karten über einen entsprechenden SD-Karten-Adapter verwendet werden.

➔ Die SD-Card wird nicht für die C-Control Pro Mega32 unterstützt, da dort der Platz im Flashspeicher (32kb) nicht ausreicht, um die FAT Filesystem Routinen zu übernehmen.

➔ Wird die SD-Card zusammen mit USB und dem Mega Applicationboard genutzt, kommt es zu einer Kollision auf dem SPI Bus. Leider erlaubt das USB Interface auf dem Applicationboard keine gemeinsame Nutzung. Die Projectboards sind nicht betroffen, da dort über das serielle Interface kommuniziert wird. Möchte man die SD-Card nutzen, so muss man die Jumper auf dem Applicationboard (**Mega128** PB.0 bis PB.4 und PE.5) abziehen.

➔ Die Signale PB.5 bis PB.7 müssen nicht zwingend genutzt werden, in den Demo Programmen werden darüber die Enable Leitungen und eine LED gesteuert. Man kann um Pins zu sparen diese Leitungen auch fest verdrahten.



Kartenhalter	PIN Mega128
WP	PE.5
CD	PB.4
MISO	PB.3
MOSI	PB.2
SCK	PB.1
SS	PB.0
EN1	PB.5
LED	PB.7
EN2	PB.6

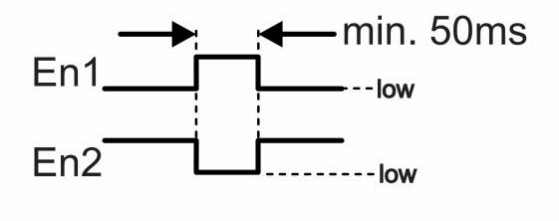
WP (Schreibschutz):
high = SD-Karte schreibgeschützt
low = schreiben erlaubt

CD (Kartenerkennung):
high = keine SD-Karte erkannt
low = SD-Karte erkannt

SPI- Schnittstelle:
MISO
MOSI
SCK
SS

Sonstige:
LED -> Benutzer Led (5V Pegel)

Resetbeschaltung:
En1 = Reset der SD-Karte (low = running mode / high = reset)
En2 = Versorgung SD-Kartenhalter (low = off / high = on)
Das untere Diagramm zeigt die Ausführung des Hardwarereset.



Einlegen der SD-Karte:
Die SD-Karte muss stets so eingeschoben werden, daß deren Kontakte zur Leiterplatte des SD-Card Interfaces zeigen. Ein verkehrtes Einlegen der SD-Karte führt zur Beschädigung des Kartenhalters.

Technische Daten:
Versorgungsspannung +5V/DC
Stromaufnahme: max. 150mA
SPI, Ein- und Ausgänge: 5V Pegel (TTL)
Zulässige Umgebungstemperatur: 0°C bis +70°C
Zulässige relative Umgebungsluftfeuchte: 20 - 80% r.F., nicht kondensierend
Abmessungen: ca. 53,5 x 42 x 4,5 mm
Gewicht: ca. 10g

5.21.1 FAT Unterstützung

FAT Spezifikation

- FAT Unterstützung: FAT12, FAT16 und FAT32.

- Offene Dateien: Unbegrenzt, abhängig vom verfügbaren Speicher
- Dateigröße: Abhängig vom FAT Typ (bis zu 4G-1 bytes)
- Volume Größe: Abhängig vom FAT Typ (bis zu 2T bytes bei 512 bytes/Sektor)
- Cluster Größe: Abhängig vom FAT Typ (bis zu 64K bytes bei 512 bytes/Sektor)
- Sektorgröße: Abhängig vom FAT Typ (bis zu 4K bytes)

➔ Die SD card Funktionen unterstützen unter FAT keine langen Dateinamen (LFN). Zum einen haben die langen Dateinamen einen erweiterten RAM und Flashspeicherbedarf, da sie auf Unicode basieren, zum anderen hält die Firma Microsoft (TM) ein Patent auf die Nutzung von LFN. Der Datei- oder Verzeichnisname muss daher das Format 8.3 haben.

5.21.2 SDC Rückgabe Werte

Alle SDC Funktionen geben Byte zurück, das den Erfolg des SDC Zugriff laut folgender Tabelle beschreibt.

Fehler	Wert	Beschreibung
FR_OK	0	Operation erfolgreich
FR_DISK_ERR	1	Physikalischer Zugriff fehlgeschlagen
FR_INT_ERR	2	Falsche FAT Struktur oder interner Fehler
FR_NOT_READY	3	Speichermedium nicht vorhanden
FR_NO_FILE	4	Datei nicht gefunden
FR_NO_PATH	5	Pfad nicht gefunden
FR_INVALID_NAME	6	Dateiname ungültig
FR_DENIED	7	Dateizugriff nicht erlaubt
FR_EXIST	8	Datei existiert schon
FR_INVALID_OBJECT	9	Datei nicht mit SDC_FOpen geöffnet
FR_WRITE_PROTECTED	10	Speichermedium schreibgeschützt
FR_INVALID_DRIVE	11	Drive-Nummer ungültig
FR_NOT_ENABLED	12	Filesystem nicht initialisiert
FR_NO_FILESYSTEM	13	Keine FAT Partition auf Speichermedium
FR_MKFS_ABORTED	14	nicht vorhanden, da mkfs nicht unterstützt
FR_TIMEOUT	15	Speichermedium antwortet nicht

5.21.3 SDC_FClose

SDCard Funktionen

Syntax

```
byte SDC_FClose(byte fil_ramaddr[]);

Sub SDC_FClose(ByRef fil_ramaddr As Byte) As Byte
```

Beschreibung

Schließt eine vorher geöffnete Datei.

Parameter

fil_ramaddr Adresse des FILE Puffers

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.4 SDC_FOpen

SDCard Funktionen

Syntax

```
byte SDC_FOpen(byte fil_ramaddr[], char path[], byte mode);

Sub SDC_FOpen(ByRef fil_ramaddr As Byte, ByRef path As Char,
              mode As Byte) As Byte;
```

Beschreibung

Öffnet eine Datei. Für jede geöffnete Datei muss eine FILE Puffer angelegt werden. Dafür definiert man ein Byte Array der Größe **SDC_FILE_BUF**. Bitte immer das #define **SDC_FILE_BUF** benutzen, da die Mega und AVR32Bit unterschiedlichen RAM Bedarf haben.

➔ Der vom Benutzer zur Verfügung gestellte RAM Puffer muss während dem Zugriff auf die SD card reserviert bleiben. Da lokale Variablen nach Verlassen der Funktion freigegeben werden, ist es meistens sinnvoll den Puffer als globale Variable zu deklarieren.

Parameter

fil_ramaddr Adresse des FILE Puffers
path Pfad zur Datei
mode Dateimodus

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

mode Parameter:

Die einzelnen Parameter werden oderiert wie z.B:

```
FA_CREATE_NEW | FA_WRITE  // CompactC
FA_CREATE_NEW Or FA_WRITE  ' BASIC
```

Modus	Wert (Hex)	Beschreibung
FA_OPEN_EXISTING	00	Öffnet Datei. Wenn Datei nicht existiert, dann Fehler
FA_READ	01	Man darf von Datei lesen
FA_WRITE	02	Man darf auf Datei schreiben

FA_CREATE_NEW	04	Legt Datei neu an. Wenn Datei schon existiert, dann Fehler
FA_CREATE_ALWAYS	08	Legt Datei neu an. Wenn Datei existiert, Dateilänge wird null
FA_OPEN_ALWAYS	10	Öffnet Datei. Wenn Datei nicht existiert, Datei wird erzeugt

5.21.5 SDC_FRead

SDCard Funktionen

Syntax

```
byte SDC_FRead(byte fil_ramaddr[], byte buf[], word btr, word br[]);

Sub SDC_FRead(ByRef fil_ramaddr As Byte, ByRef buf As Byte, btr As Word,
              ByRef br As Word) As Byte
```

Beschreibung

Liest Daten aus einer geöffneten Datei. Die Daten werden an der Leseposition aus der Datei in den Puffer buf geschrieben. Die Anzahl der zu lesenden Bytes ist btr, in das erste Element von br wird die Anzahl der tatsächlich gelesenen Bytes kopiert. Die Leseposition kann mit SDC_FSeek bestimmt werden.

Parameter

fil_ramaddr	Adresse des FILE Puffers
buf	Puffer in die die gelesenen Bytes geschrieben werden
btr	Anzahl der zu lesenden Bytes
br	Anzahl der tatsächlich gelesenen Bytes

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.6 SDC_FSeek

SDCard Funktionen

Syntax

```
byte SDC_FSeek(byte fil_ramaddr[], dword pos);

Sub SDC_FSeek(ByRef fil_ramaddr As Byte, pos As ULong) As Byte
```

Beschreibung

Setzt die Lese- bzw. Schreibposition der geöffneten Datei. Die Position pos wird immer vom Anfang der Datei gezählt.

Parameter

fil_ramaddr	Adresse des FILE Puffers
pos	Lese- bzw. Schreibposition

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.7 SDC_FSetDateTime

SDCard Funktionen

Syntax

```
byte SDC_FSetDateTime(char path[], byte day, byte mon, word year, byte min,
                      byte hours, byte sec);

Sub SDC_FSetDateTime(ByRef path As Char,day As Byte,mon As Byte,year As Word,
                     min As Byte, hours As Byte, sec As Byte) As Byte
```

Beschreibung

Setzt die Datums und Uhrzeit Attribute einer Datei.

Parameter

path	Pfad zur Datei.
day	Tag (1-31)
mon	Monat (1-12)
year	Jahr (1980-2107)
min	Minute (0-59)
hours	Stunde (0-23)
sec	Sekunde (0-59) (wird immer auf einen geraden Wert gesetzt)

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.8 SDC_FStat

SDCard Funktionen

Syntax

```
byte SDC_FStat(char path[], dword filinfo[]);

Sub SDC_FStat(ByRef path As Char, ByRef filinfo As ULong) As Byte
```

Beschreibung

Liest Attribute einer Datei in ein dword (ULong) Array mit 4 Elementen.

Parameter

path Pfad zur Datei.
filinfo Rückgabe Array.

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

Rückgabe Array:

fileinfo[0]	Dateilänge
fileinfo[1]	Datum
fileinfo[2]	Uhrzeit
fileinfo[3]	Dateiattribut

Kodierung Datum:
Bits 0:4 - Tag: 1...31
Bits 5:8 - Monat: 1...12
Bits 9:15 - Jahr beginnend mit1980: 0...127

Kodierung Zeit:
Bits 0:4 - Sekunde/2: 0...29
Bits 5:10 - Minute: 0...59
Bits 11:15 - Stunde: 0...23

Kodierung Dateiattribut:
Bit 1: Read Only (Nur Lesen)
Bit 2: Hidden (Versteckt)
Bit 3: Volume label (Disknamen)
Bit 4: Directory (Verzeichnis)
Bit 5: Archive (Archiv)

5.21.9 SDC_FSync

SDCard Funktionen

Syntax

```
byte SDC_FSync(byte fil_ramaddr[]);

Sub SDC_FSync(ByRef fil_ramaddr As Byte) As Byte
```

Beschreibung

Wartet darauf, das alle Daten aus dem Puffer in die Datei auf die SD card geschrieben wurden.

Parameter

fil_ramaddr Adresse des FILE Puffers

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.10 SDC_FTruncate

SDCard Funktionen

Syntax

```
byte SDC_FTruncate(byte fil_ramaddr[]);

Sub SDC_FTruncate(ByRef fil_ramaddr As Byte) As Byte
```

Beschreibung

Löscht den Rest der Datei ab der aktuellen Schreibposition.

Parameter

fil_ramaddr Adresse des FILE Puffers

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.11 SDC_FWrite

SDCard Funktionen

Syntax

```
byte SDC_FWrite(byte fil_ramaddr[], byte buf[], word btr, word br[]);

Sub SDC_FWrite(ByRef fil_ramaddr As Byte, ByRef buf As Byte, btr As Word,
               ByRef br As Word) As Byte
```

Beschreibung

Schreibt Daten in eine geöffnete Datei. Die Daten werden von dem Puffer buf in die Datei an der aktuellen Schreibposition geschrieben. Die Anzahl der zu lesenden Bytes ist btr, in das erste Element von br wird die Anzahl der tatsächlich geschriebenen Bytes kopiert. Die Schreibposition kann mit SDC_FSeek bestimmt werden.

Parameter

fil_ramaddr	Adresse des FILE Puffers
buf	Puffer aus dem die Bytes in die Datei geschrieben werden
btr	Anzahl der zu schreibenden Bytes
br	Anzahl der tatsächlich geschriebenen Bytes

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.12 SDC_GetFree

SDCard Funktionen

Syntax

```
byte SDC_GetFree(char path[], dword kbfree[]);

Sub SDC_GetFree(ByRef path As Char, ByRef kbfree As ULong) As Byte
```

Beschreibung

Gibt die Anzahl der freien Cluster auf der SD card zurück. Die Anzahl der freien Cluster wird in das erste Element des Array kbfree kopiert.

Parameter

path Pfad zum Wurzelverzeichnis (Root) der Disk.
kbfree Rückgabe Array

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.13 SDC_Init

SDCard Funktionen

Syntax

```
void SDC_Init(byte fat_ramaddr[]);

Sub SDC_Init(ByRef fat_ramaddr As Byte)
```

Beschreibung

Initialisiert die SD card Bibliothek. Für diese Operation muss eine FAT Puffer angelegt werden. Dafür definiert man ein Byte Array der Größe **SDC_FAT_BUF**. Bitte immer das #define **SDC_FAT_BUF** benutzen, da die Mega und AVR32Bit unterschiedlichen RAM Bedarf haben.

➔ Der vom Benutzer zur Verfügung gestellte RAM Puffer muss während dem Zugriff auf die SD-Card reserviert bleiben. Da lokale Variablen nach Verlassen der Funktion freigegeben werden, ist es meistens sinnvoll den Puffer als globale Variable zu deklarieren.

Parameter

fat_ramaddr Adresse des FAT Puffers

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.14 SDC_MkDir

SDCard Funktionen

Syntax

```
byte SDC_MkDir(char path[]);
```

```
Sub SDC_MkDir(ByRef path As Char) As Byte
```

Beschreibung

Erstellt ein Verzeichnis auf der SD card.

Parameter

path Pfad zum Verzeichnis.

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.15 SDC_Rename

SDCard Funktionen

Syntax

```
byte SDC_Rename(char oldpath[], char newpath[]);
```

```
Sub SDC_Rename(ByRef oldpath As Char, ByRef newpath As Char) As Byte
```

Beschreibung

Benennt eine Datei von oldpath nach newpath um.

Parameter

oldpath Pfad zur Datei.
newpath Pfad zur Datei mit neuem Namen.

Wenn newpath zu einem anderen Verzeichnis zeigt als oldpath, dann wird die Datei nicht nur umbenannt, sondern auch in das neue Verzeichnis bewegt. In newpath darf keine logische Disknummer stehen, nur in oldpath.

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.16 SDC_Unlink

SDCard Funktionen

Syntax

```
byte SDC_Unlink(char path[]);

Sub SDC_Unlink(ByRef path As Char) As Byte
```

Beschreibung

Löscht eine Datei.

Parameter

path Pfad zur Datei.

Rückgabewert

Erfolg der aufgerufenen SDC Funktion. Siehe SDC Rückgabe Werte.

5.21.17 SD-Card Beispiel

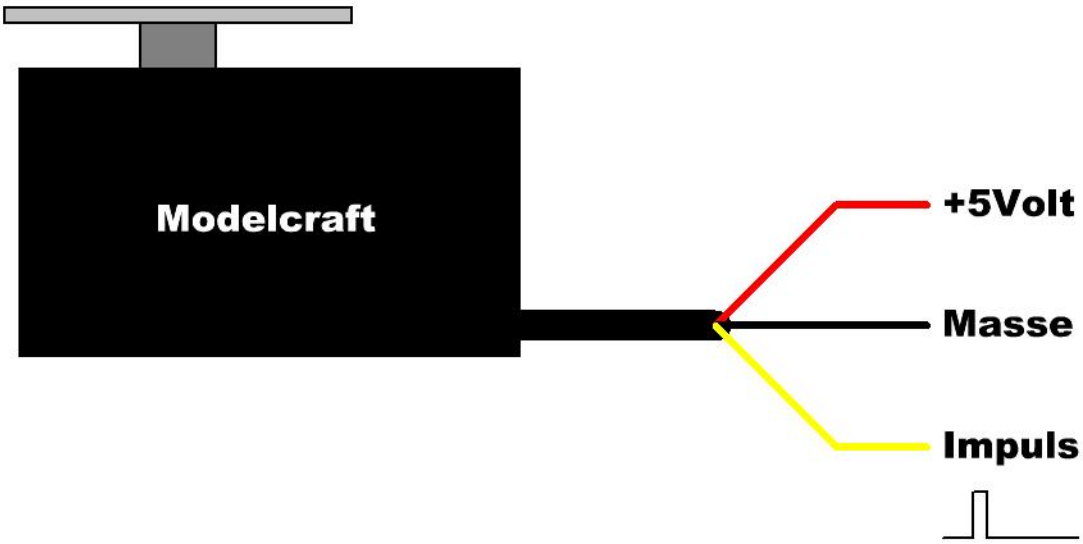
```
// Globale Variablen
byte fat[SDC_FAT_BUF];
byte fil[SDC_FILE_BUF];

void main(void)
{
    // Lokale Variable
    byte res;
    char buf[100];
    word bytes_written[1];

    // SD-Card reset
    Port_DataDirBit(13,1);          // PB.5 = Ausgang (EN1)
    Port_DataDirBit(14,1);          // PB.6 = Ausgang (EN2)

    Port_WriteBit(13,1);            // EN1 für 50ms auf +5V (PB.5)
    Port_WriteBit(14,0);            // EN2 für 50ms auf GND (PB.6)

    AbsDelay(50);                   // 50ms Pause

    Port_WriteBit(13,0);            // EN1 GND
    Port_WriteBit(14,1);            // EN2 +5V

    // Power on -> SD-Card
```

```
    Port_WriteBit(14,1);            // EN2 (PB.6) +5V

    AbsDelay(50);                   // 50ms Pause

    // SD-Card Fat init
    SDC_Init        (fat);


    // Neuen Dateiordner erstellen
    SDC_MkDir("0:/CC-PRO");


    // Ist die Datei bereits vorhanden?
    // Wenn nicht dann wird die Datei angelegt
    res=SDC_FOpen(fil, "0:/CC-PRO/test.txt", FA_READ|FA_WRITE|FA_OPEN_EXISTING);
    if(res!=0)SDC_FOpen(fil, "0:/CC-PRO/test.txt", FA_WRITE|FA_CREATE_ALWAYS);

    // Schreibt einen Text in die Datei
    buf= "Hallo... 123!\r\n";
    SDC_FWrite(fil, buf, Str_Len(buf), bytes_written);
    SDC_FSync(fil);

    // Datei wird geschlossen
    SDC_FClose(fil);
}
```

5.22 Servo

Ein Servo bezeichnet in der Elektrotechnik einen Verbund aus Ansteuerungseinheit und Antriebseinheit. Es handelt sich um einen DC-angesteuerten Motor. Zu diesem gehören drei Anschlüsse: Die Versorgung (VCC), Masse (GND) und eine Kontrollleitung (PW). Servos werden über eine Pulsweitenmodulation (PWM) angesteuert. Über die Breite der Pulse wird der Winkel, auf den der Servoarm gestellt werden soll, gesteuert. Gängig ist ein 50-Hz-Signal (20 ms Periodenlänge), welches zwischen 1 ms (linker Anschlag) und 2 ms (rechter Anschlag) auf High-Pegel und den Rest der Periodenlänge auf Lo-Pegel ist.

Anschluß an C-Control Pro



+5Volt ist die Versorgungsspannung des Servos, diese muss genügend Strom für das Servo liefern können. Die Masse des Servos muss gleich der Masse der Versorgungsspannung der C-Control Pro Unit sein. An der Impuls Leitung des Servos, wird das PWM Signal von der C-Control Pro in das Servo eingespeist.

5.22.1 Servo_Init

Servo Funktionen Beispiel

Syntax

```
void Servo_Init(byte servo_cnt, byte servo_interval, byte ramaddr[],
                byte timer);

Sub Servo_Init(servo_cnt As Byte, servo_interval As Byte,
               ByRef ramaddr As Byte, timer As Byte)
```

Beschreibung

Initialisiert die Servoroutinen. Der Parameter servo_cnt gibt an wieviele Servos gleichzeitig betrieben werden. Die Periodenlänge (10 oder 20ms) wird mit servo_interval gesetzt, der Parameter timer bestimmt, welcher 16-Bit Timer eingesetzt wird. Timer 3 steht allerdings nur bei dem Mega128 zur Verfügung. Der Anwender muss den Servoroutinen Speicher zur Verfügung stellen. Für die Größe des Puffers gibt es ein #define **SERVO_BUF**. Möchte man ein byte Array definieren um X Servos zu betreiben, schreibt man `"byte buf[SERVO_BUF(X)];"`

➔ Das vom Benutzer zur Verfügung gestellte RAM muss während der Servosteuerung reserviert bleiben. Da lokale Variablen nach Verlassen der Funktion freigegeben werden, ist es meistens sinn-

voll das byte Array global zu deklarieren.

➔ Für die Servoansteuerung wird ein 16-Bit Timer benötigt. Wird der Timer abgeschaltet oder für andere Timerfunktionen genutzt, so wird die Servoansteuerung nicht arbeiten.

Parameter

servo_cnt	Anzahl der möglichen Servos (maximal 20)
servo_interval	Periodenlänge (0=10ms, 1=20ms)
ramaddr	Adresse des Speicherblocks
timer	Für die Servosteuerung benutzter 16-Bit Timer Mega32: 0 = Timer 1 Mega128 & Mega128 CAN: 0=Timer 1, 1=Timer 3 AVR32: alle Timer (0 - 5)

5.22.2 Servo_Set

Servo Funktionen **Beispiel**

Syntax

```
void Servo_Set(byte portbit, word pos);

Sub Serial_Init(portbit As Byte, pos As Word)
```

Beschreibung

Setzt die Pulslänge zur Steuerung des Servoarms. Der Ausgangsport wird über den portbit Parameter angegeben. (Für die Abbildung zwischen Portbits und den Pins des Controllers siehe Port Tabellen).

➔ Alle Pulslängen der gestellten Servos dürfen als Summe nicht die Periodenlänge (servo_interval Parameter) überschreiten, da sonst ein erratisches Verhalten auftritt. Man kann daher z.B. 8 Servos auf 2500µs Pulslänge bei einer Periodenlänge von 20ms setzen. Zur Sicherheit sollte man allerdings ein wenig unter der Periodenlänge bleiben.

Parameter

portbit	Bitnummer des Ports (siehe Port Tabellen)
pos	Pulslänge zur Steuerung des Servoarms in µsec (500 - 2500)

5.22.3 Servo Beispiel

```
byte servo_var[SERVO_BUF(10)]; // Servo interne Variablen

// Ansteuerung von 3 Servos und beenden nach 10 Sek.
void main(void)
{
    // Max. 10 Servos, 20ms Intervall, Timer 3
    Servo_Init(10, 1, servo_var, 1);

    Servo_Set(7, 2000);  // Servo Portbit 7   2000µs
    Servo_Set(6, 1800);  // Servo Portbit 6   1800µs
```

```
    Servo_Set(5, 1600);  // Servo Portbit 5   1600µs

    AbsDelay(5000);

    Servo_Set(7, 1000);  // Servo Portbit 7   1000µs

    AbsDelay(5000);

    Servo_Set(7, 0);     // alle Servos aus
    Servo_Set(6, 0);
    Servo_Set(5, 0);
}
```

5.23 SPI

Das Serial Peripheral Interface (SPI) ist ein von Motorola entwickeltes Bus-System mit einem Standard für einen synchronen seriellen Datenbus, mit dem digitale Komponenten nach dem Master-Slave-Prinzip miteinander verbunden werden können.

5.23.1 Mega

Wird mit dem Application Board gearbeitet, wird der SPI Bus genutzt um Daten mit dem Mega8 Controller auszutauschen, der für die USB Anbindung zuständig ist. In eigenen Anwendungen kann der Benutzer die SPI Funktionen zum Datentransfer nutzen. Dann ist aber der serielle Bootloadermodus zu nutzen.

5.23.1.1 SPI_Disable

SPI Funktionen

Syntax

```
void SPI_Disable(void);

Sub SPI_Disable()
```

Beschreibung

Die SPI Schnittstelle wird abgeschaltet und die dazugehörigen Ports können anders verwendet werden.

➔ Das deaktivieren der SPI Schnittstelle auf dem C-Control Pro Mega verhindert die Benutzung USB Schnittstelle auf dem Application Board. Andererseits, wenn man die USB Schnittstelle nicht benötigt, kann man mit SPI_Disable() die Ports für andere Zwecke einsetzen.

Parameter

Keine

5.23.1.2 SPI_Enable

SPI Funktionen

Syntax

```
void SPI_Enable(byte ctrl);

Sub SPI_Enable(ctrl As Byte)
```

Beschreibung

Die SPI Schnittstelle wird mit dem Wert von ctrl initialisiert (siehe **SPCR** Register in Atmel Mega Reference Manual).

Parameter

ctrl Initialisierungsparameter (Mega SPCR Register)

Bit 7 - SPI Interrupt Enable (nicht einschalten, kann von C-Control Pro nicht genutzt werden)
Bit 6 - SPI Enable (muss gesetzt sein)
Bit 5 - Data Order (1 = LSB first, 0 = MSB first)
Bit 4 - Master/Slave Select (1 = Master, 0 = Slave)
Bit 3 - Clock polarity (1 = leading edge falling, 0 = leading edge rising)
Bit 2 - Clock Phase (1 = sample on trailing edge, 0 = sample on leading edge)

Bit 1	Bit 0	SCK Frequency
0	0	f_{Osc} / 4
0	1	f_{Osc} / 16
1	0	f_{Osc} / 64
1	1	f_{Osc} / 128

➔ Es ist zu beachten, das bei C-Control Pro Mega 32 und Mega128 f_{Osc} = 14,7456 Mhz ist, während die C-Control Pro Mega128 CAN mit 16 Mhz arbeitet.

5.23.1.3 SPI_Read

SPI Funktionen

Syntax

```
byte SPI_Read();

Sub SPI_Read() As Byte
```

Beschreibung

Ein Byte wird von der SPI Schnittstelle gelesen.

Rückgabewert

empfangenes byte aus der SPI Schnittstelle

5.23.1.4 SPI_ReadBuf

SPI Funktionen

Syntax

```
void SPI_ReadBuf(byte buf[], word length);

Sub SPI_ReadBuf(ByRef buf As Byte, length As Word)
```

Beschreibung

Ein Anzahl von Bytes wird von der SPI Schnittstelle in ein Array gelesen.

Parameter

buf	Zeiger auf byte array
length	Anzahl der einzulesenden bytes

5.23.1.5 SPI_Write

SPI Funktionen

Syntax

```
void SPI_Write(byte data);

Sub SPI_Write(data As Byte)
```

Beschreibung

Ein Byte wird auf die SPI Schnittstelle geschrieben.

Parameter

data	auszugebendes Datenbyte

5.23.1.6 SPI_WriteBuf

SPI Funktionen

Syntax

```
void SPI_WriteBuf(byte buf[], word length);

Sub SPI_WriteBuf(ByRef buf As Byte, length As Word)
```

Beschreibung

Ein Anzahl von Bytes wird auf die SPI Schnittstelle geschrieben.

Parameter

buf Zeiger auf byte array
length Anzahl der auszugebenden bytes

5.23.2 AVR32Bit

5.23.2.1 SPI_Disable

SPI Funktionen

Syntax

```
void SPI_Disable(byte chan);

Sub SPI_Disable(chan As Byte)
```

Beschreibung

Die SPI Schnittstelle wird abgeschaltet und die dazugehörigen Ports können anders verwendet werden.

Parameter

chan SPI Kanal (0 - 1)

5.23.2.2 SPI_Enable

SPI Funktionen

Syntax

```
void SPI_Enable(byte chan, dword speed, byte bits, byte mode);

Sub SPI_Enable(chan As Byte, speed As ULong, bits As Byte, mode As Byte)
```

Beschreibung

Die SPI Schnittstelle wird auf eine Taktrate, Anzahl Datenbits und SPI Mode initialisiert. Aus der Taktrate speed wird dann intern ein Divisor berechnet der im Chip die gewünschte Geschwindigkeit setzt. Da der Divisor nur einen Wert zwischen 1 und 255 annehmen kann, wird der angegebene speed Parameter nur grob eingehalten. Es wird der Divisor gewählt der die gewünschte Taktrate am nächsten erfüllt: Divisor = 66Mhz / speed. Die tatsächliche Geschwindigkeit ist dann 66Mhz / Divisor. Dies führt dazu, das Geschwindigkeiten kleiner 259.000 nicht genutzt werden können.

Parameter

chan	SPI Kanal (0 - 1)
speed	SPI Takt (259000 - 66000000)
bits	Anzahl der Datenbits
mode	SPI Mode

SPI Mode	CPOL	NCPHA
0	0	1
1	0	0
2	1	1
3	1	0

5.23.2.3 SPI_Read

SPI Funktionen

Syntax

```
word SPI_Read();
```

```
Sub SPI_Read() As Word
```

Beschreibung

Daten werden aus der SPI Schnittstelle gelesen.

Rückgabewert

empfangene Daten (4-16 Bit) aus der SPI Schnittstelle

5.23.2.4 SPI_ReadBuf

SPI Funktionen

Syntax

```
void SPI_ReadBuf(byte buf[], word length);
```

```
Sub SPI_ReadBuf(ByRef buf As Byte, length As Word)
```

Beschreibung

Ein Anzahl von Bytes wird von der SPI Schnittstelle in ein Array gelesen. Die Funktion arbeitet mit bis zu 8 Bit, ungeachtet ob die SPI Schnittstelle mit mehr Bit initialisiert wurde.

Parameter

buf Zeiger auf byte array
length Anzahl der einzulesenden bytes

5.23.2.5 SPI_SetChan

SPI Funktionen

Syntax

```
void SPI_SetChan(byte chan);

Sub SPI_SetChan(chan As Byte)
```

Beschreibung

Selektiert eine SPI Schnittstelle (SPI0 oder SPI1) für den weiteren Zugriff.

Parameter

chan SPI Kanal (0 - 1)

5.23.2.6 SPI_Write

SPI Funktionen

Syntax

```
void SPI_Write(word data);

Sub SPI_Write(data As Word)
```

Beschreibung

Daten werden auf die SPI Schnittstelle geschrieben.

Parameter

data auszugebende Daten (4-16 Bit)

5.23.2.7 SPI_WriteBuf

SPI Funktionen

Syntax

```
void SPI_WriteBuf(byte buf[], word length);

Sub SPI_WriteBuf(ByRef buf As Byte, length As Word)
```

Beschreibung

Ein Anzahl von Bytes wird auf die SPI Schnittstelle geschrieben. Die Funktion arbeitet mit bis zu 8 Bit, ungeachtet ob die SPI Schnittstelle mit mehr Bit initialisiert wurde.

Parameter

buf Zeiger auf byte array
length Anzahl der auszugebenden bytes

5.24 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, daß man bei Nichtgebrauch diese Funktionen durch Weglassen der Bibliothek aus dem Projekt nimmt. Direkte Interpreterfunktionen sind immer präsent, kosten aber Flashspeicher.

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

5.24.1 Str_Comp

String Funktionen

Syntax

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

Sub Str_Comp(ByRef str1 As Char, ByRef str2 As Char) As Char
```

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

5.24.2 Str_Copy

String Funktionen

Syntax

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

Sub Str_Copy(ByRef destination As Char, ByRef source As Char, offset As Word)
```

Beschreibung

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

Parameter

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

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

5.24.3 Str_Fill

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

Syntax

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

Sub Str_Fill(ByRef dest As Char, c As Char, len As Word)
```

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

5.24.4 Str_Isalnum

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

Syntax

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

```
Sub Str_Isalnum(c As Char) As Byte
```

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

5.24.5 Str_Isalpha

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

Syntax

```
byte Str_Isalpha(char c);

Sub Str_Isalpha(c As Char) As Byte
```

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

5.24.6 Str_Len

String Funktionen

Syntax

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

Sub Str_Len(ByRef str As Char) As Word
```

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).

5.24.7 Str_Printf

String Funktionen Beispiel

Syntax

```
void Str_Printf(char str[], char format[], ...);

Sub Str_Printf(ByRef str As Char, ByRef format As Char, ...)
```

Beschreibung

Die Funktion erstellt eine formatierte Zeichenkette in den String str. Der Formatierungsstring ist der Funktionsweise von printf() angelehnt. Die Formatierung beginnt immer mit **%**, es folgen optionale **flags** (**0**,**l**), und endet mit dem **Typ** (**d**,**x**,**s**,**f**,**u**). In der folgenden Tabelle sind die möglichen Typen erklärt. Zwischen **%** und **Typ** können optional eine Weite und eine Genauigkeit angegeben werden.

%[**flags**][*width*][.*prec*]**Typ** (Die eckigen Klammern bezeichnen den optionalen Teil)

Die Weite (**width**) ist der minimale Platz die die Ausgabe der Zahl einnimmt. Ist die Zahl kürzer als die Weite, so wird von links mit Leerzeichen aufgefüllt. Beginnt **width** mit einer "0", so wird mit "0" anstatt von Leerzeichen gefüllt. Ist ein Punkt "." angegeben, so wird eine Genauigkeit (**prec**) definiert, die bei Fließkommazahlen (**%f**) die Anzahl der Nachkommastellen darstellt, und bei vorzeichenlosen Integerzahlen (**%u**) die Basis der Zahl. Siehe auch Str_Printf Beispiel.

➔ Vergisst man bei der Ausgabe einer 32-Bit Zahl den Flag "l" anzugeben, so werden nur die unteren 16-Bit der Zahl ausgegeben.

Flags	Beschreibung
0	mit "0" auffüllen
l	32-Bit Integer

Formatierung	Beschreibung
%[*width*]**d**	Integerzahl
%[*width*][.*prec*]**u**	vorzeichenloser Integer
%[*width*]**x**	Hexzahl
%[*width*][.*prec*]**f**	Fließkommazahl
%[*width*]**s**	String
%[width]**c**	Zeichen

Parameter

str Zeiger auf Zielzeichenkette
format Zeiger auf Format String

5.24.8 Str_ReadFloat

String Funktionen

Syntax

```
float Str_ReadFloat(char str[]);

Sub Str_ReadFloat(ByRef str As Char) As Single
```

Beschreibung

Der Fließkommawert einer Zeichenkette auf die *str* zeigt, wird zurückgegeben. Nach der Zahl dürfen auch andere Zeichen in dem String stehen.

Parameter

str Zeiger auf String

Rückgabewert

Fließkommawert der Zeichenkette.

5.24.9 Str_ReadInt

String Funktionen

Syntax

```
int Str_ReadInt(char str[]);

Sub Str_ReadInt(ByRef str As Char) As Integer
```

Beschreibung

Der Integerwert einer Zeichenkette auf die *str* zeigt, wird zurückgegeben. Nach der Zahl dürfen auch andere Zeichen in dem String stehen.

Parameter

str Zeiger auf String

Rückgabewert

Integerwert der Zeichenkette.

5.24.10 Str_ReadNum

String Funktionen

Syntax

```
word Str_ReadNum(char str[], byte base);

Sub Str_ReadNum(ByRef str As Char, base As Byte) As Word
```

Beschreibung

Der Wert einer Zeichenkette auf die *str* zeigt, wird zurückgegeben. Nach der Zahl dürfen auch andere Zeichen in dem String stehen. Der Parameter base ist die Basis der Zahl. Möchte man z.B. eine Hexzahl einlesen, ist als base 16 anzugeben, bei einer Binärzahl ist die Basis 2.

Parameter

str Zeiger auf String
base Basis der einzulesenden Zahl

Rückgabewert

Integerwert der Zeichenkette.

5.24.11 Str_Substr

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

Syntax

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

Sub Str_SubStr(ByRef source As Char, ByRef search As Char) As Integer
```

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

5.24.12 Str_WriteFloat

String Funktionen

Syntax

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

Sub Str_WriteFloat(n As Single, decimal As Byte, ByRef text As Char,offset As Word)
```

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. Mit Hilfe von offset kann man den Anfang eines Strings intakt lassen.

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 (ffff Hex), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird die float Zahl an den Text String angehängt.

5.24.13 Str_WriteInt

String Funktionen

Syntax

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

Sub Str_WriteInt(n As Integer, ByRef text As Char, offset As Word)
```

Beschreibung

Die Integer Zahl n wird in einen vorzeichenbehafteten ASCII String konvertiert. Das Ergebnis wird im String text mit einem Versatz von offset abgespeichert. Mit Hilfe von offset kann man den Anfang eines Strings intakt lassen.

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** (ffff Hex), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird die Integer Zahl an den Text String angehängt.

5.24.14 Str_WriteWord

String Funktionen

Syntax

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

Sub Str_WriteWord(n As Word, base As Byte, ByRef text As Char, offset As Word, minwidth As Byte)
```

Beschreibung

Das Wort n wird in einen ASCII String konvertiert. Das Ergebnis wird im String text mit einem Versatz von offset abgespeichert. Mit Hilfe von offset kann man den Anfang eines Strings intakt lassen.

Man kann für die Ausgabe eine beliebige Basis angeben. Mit einer Basis von 2 erhält man Binärzahlen, mit 8 Oktalzahlen, 10 Dezimalzahlen 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 (ffff Hex), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird die Integer Zahl an den Text String angehängt.

5.24.15 Str_Printf Beispiel

```
// CompactC
void main(void)
{
    char str[80];

    // Integerzahl
    Str_Printf(str, "arg1: %d\r", 1234);
    Msg_WriteText(str);
```

```
    // Ausgabe von Integer, Fließkomma, String und Hexzahl
    Str_Printf(str, "arg1: %8d arg2:%10.3f arg3:%20s arg4: %x\r",
        1234, 2.34567, "hallo welt", 256);
    Msg_WriteText(str);
    Str_Printf(str, "arg1: %u arg2: %.2u\r", 65000, 0xff);
    Msg_WriteText(str);
}


' Basic
Sub main()
    Dim str(80) As Char

    Str_Printf(str, "arg1: %08d arg2:%10.3f arg3:%20s arg4: %x\r",
      1234, 2.34567, "hallo welt", 256)
    Msg_WriteText(str)
    Str_Printf(str, "arg1: %u arg2: %.2u\r", 65000, &Hff)
    Msg_WriteText(str)
End Sub
```

5.25 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 entsteht.

Die C-Control Pro Firmware unterstützt außer dem Hauptprogramm (Thread "0") bis zu 13 zusätzliche Threads. Mit der Version 2.12 der IDE wurde das Multithreading geändert. Vor 2.12 wurde in den Projekt Optionen jedem Thread eine Anzahl von Bytecodes zugeteilt, nach der der Thread gewechselt wurde. Dies führte of zu einer ungerechten Verteilung, da z.B. Fließkomma Operationen viel mehr CPU Zeit benötigen als andere Bytecodes. Jetzt arbeitet das Multithreading mit Zeitscheiben. Der Anwender kann jedem Thread nun eine Anzahl von 10ms Zeitzyklen zuweisen, nachdem der Threadwechsel stattfindet.

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.

➔ Wird das Hauptprogramm (Thread "0") beendet, stoppen auch alle anderen Threads.

Die Priorität eines Threads kann beeinflußt werden, in dem man ändert, wie viele Zeitzyklen ein Thread bis zum nächsten Threadwechsel ausführen darf. Je kleiner die Anzahl der Zyklen bis zum Wechsel, desto geringer die Priorität des Threads.

Thread Konfiguration

Vor IDE Version 2.12 die Konfiguration der Threads wurde in den Projekt Optionen eingestellt. Dies hat sich geändert. Jetzt wird die Konfiguration der threads mit Hilfe des neues "**#thread**" Befehls im Source Code vorgenommen:

```
#thread thread_nummer, benutztes_ram, anzahl_zeit_zyklen
```

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

➔ Die "**#thread**" Anweisungen müssen nicht bei den Thread Funktionen sein, sondern dürfen überall im Programm stehen. Benutzt man keine Threads, so ist ein "**#thread 0**" Befehl unnötig. Vergisst man einen Thread zu definieren, so wird das Thread_Start ignoriert.

Beispiel CompactC:

```
#thread 0, 0, 20   // Hauptthread mit Task Wechsel alle 20 * 10ms =200ms
#thread 1, 128, 10 // Thread 1 mit 128 byte & Task Wechsel 10*10ms =100ms
#thread 2, 256, 10 // Thread 2 mit 256 byte & Task Wechsel 10*10ms =100ms
```

Beispiel BASIC (Syntax identisch zu CompactC):

```
#thread 0, 0, 20   ' Hauptthread mit Task Wechsel alle 20 * 10ms =200ms
#thread 1, 128, 10 ' Thread 1 mit 128 byte & Task Wechsel 10*10ms =100ms
#thread 2, 256, 10 ' Thread 2 mit 256 byte & Task Wechsel 10*10ms =100ms
```

➔ Da z.B. Serial_Read wartet, bis ein Zeichen von der seriellen Schnittstelle ankommt, kann es passieren, das der Thread länger als die ihm zugewiesenen Zeitzyklen arbeitet.

➔ Beim arbeiten mit Threads sollte man immer Thread_Delay und nicht AbsDelay benutzen. Wird trotzdem z.B. ein AbsDelay(1000) aufgerufen, so wartet der Thread 1000ms, auch wenn ihm weniger Zeit zugewiesen wurde.

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, daß 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

5.25.1 Thread_Cycles

Thread Funktionen

Syntax

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

Sub Thread_Cycles(thread As Byte, cycles As Word)
```

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 (0-13) Nummer des Threads dessen Zyklus geändert werden soll
cycles Anzahl der Zyklen bis zum Threadwechsel

5.25.2 Thread_Delay

Thread Funktionen **Beispiel**

Syntax

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

```
Sub Thread_Delay(delay As Word)
```

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

5.25.3 Thread_Info

Thread Funktionen

Syntax

```
word Thread_Info(byte info);

Sub Thread_Info(info As Byte) As Word
```

Beschreibung

Liefert Informationen über den Thread, der die Funktion Thread_Info aufruft. Der info Parameter bestimmt, welche Information zurückgegeben wird.

Parameter

info Werte:

TI_THREADNUM	Nummer des aufrufenden Threads
TI_STACKSIZE	Definierte Stackgröße
TI_CYCLES	Anzahl der auszuführenden Cycles vor einem Threadwechsel

Rückgabewert

angeforderter Parameter

5.25.4 Thread_Kill

Thread Funktionen

Syntax

```
void Thread_Kill(byte thread);

Sub Thread_Kill(thread As Byte)
```

Beschreibung

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

Parameter

thread (0-13) Nummer des Threads

5.25.5 Thread_Lock

Thread Funktionen

Syntax

```
void Thread_Lock(byte lock);

Sub Thread_Lock(lock As Byte)
```

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, findet kein Multithreading mehr statt.

Parameter

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

5.25.6 Thread_MemFree

Thread Funktionen

Syntax

```
word Thread_MemFree(void);
```

```
Sub Thread_MemFree() As Word
```

Beschreibung

Gibt den freien Speicher zurück, die dem Thread noch zur Verfügung steht.

Parameter

Keine

Rückgabewert

freier Speicher in bytes

5.25.7 Thread_Resume

Thread Funktionen

Syntax

```
void Thread_Resume(byte thread);

Sub Thread_Resume(thread As Byte)
```

Beschreibung

Hat ein Thread den Zustand "*wartend*", kann er mit der Resume Funktion wieder auf "*inaktiv*" gesetzt werden. Der Status "*inaktiv*" bedeutet, das der Thread bereit ist, bei einem Threadwechsel wieder aktiviert zu werden.

Parameter

thread (0-13) Nummer des Threads

5.25.8 Thread_Signal

Thread Funktionen

Syntax

```
void Thread_Signal(byte signal);

Sub Thread_Signal(signal As Byte)
```

Beschreibung

Wurde ein Thread mittels Thread_Wait() auf "*wartend*" gesetzt, 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

5.25.9 Thread_Start

Thread Funktionen **Beispiel**

Syntax

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

Sub Thread_Start(thread As Byte, func As ULong)
```

Beschreibung

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

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

Parameter

thread (0-13) Nummer des Threads, der gestartet werden soll
func Name der Funktion, in welcher der neue Thread gestartet wird

5.25.10 Thread_Wait

Thread Funktionen

Syntax

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

Sub Thread_Wait(thread As Byte, signal As Byte)
```

Beschreibung

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

Parameter

thread (0-13) Nummer des Threads

signal Wert des Signals

5.25.11 Thread Beispiel

```
// Demoprogramm zum Multithreading für Mega und AVR32
// Das Programm ist nicht entprellt, ein kurzes Tasten führt daher zu
// mehrfacher Ausgabe des Strings
#ifdef AVR32
#define PORT_SW1 PORT_T1
#define PORT_SW2 PORT_T2
#endif

#thread 0, 0, 10   // Hauptthread mit Task Wechsel alle 10 * 10ms =100ms
#thread 1, 128, 10 // Thread 1 mit 128 byte & Task Wechsel 10*10ms =100ms

void thread1(void)
{
    while(true)  // Endlosschleife
    {
        if(!Port_ReadBit(PORT_SW2)) Msg_WriteText("Taster 2"); // SW2 gedrückt
    }
}

void main(void)
{
#ifdef AVR32
    // Pin jeweils auf Eingang & Pullup
    Port_Attribute(PORT_T1, PORT_ATTR_INPUT | PORT_ATTR_PULL_UP);
    Port_Attribute(PORT_T2, PORT_ATTR_INPUT | PORT_ATTR_PULL_UP);
#else
    Port_DataDirBit(PORT_SW1, PORT_IN);  // Pin auf Eingang
    Port_DataDirBit(PORT_SW2, PORT_IN);  // Pin auf Eingang
    Port_WriteBit(PORT_SW1, 1);  // Pullup setzen
    Port_WriteBit(PORT_SW1, 1);  // Pullup setzen
#endif
    Thread_Start(1,thread1); // Thread 1 starten

    while(true)     // Endlosschleife
    {
        if(!Port_ReadBit(PORT_SW1)) Msg_WriteText("Taster 1"); // SW1 gedrückt
    }
}
```

5.25.12 Thread Beispiel 2

```
// multithread2: Multithreading mit Thread_Delay
// erforderliche Library: IntFunc_Lib.cc

#thread 0, 0, 10   // Hauptthread mit Task Wechsel alle 10 * 10ms =100ms
#thread 1, 128, 10 // Thread 1 mit 128 byte & Task Wechsel 10*10ms =100ms

void thread1(void)
{
    while(true)
    {
        Msg_WriteText("Thread2");  // "Thread2" wird ausgegeben.
        Thread_Delay(200); // Danach ist der Thread für 200ms "schlafend".
    }
}
//---------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
    Thread_Start(1,thread1);       // Funktionsaufruf mit Angabe der
                                   // Threadnummer.
    while(true)                    // Endlosschleife
    {
        Thread_Delay(100);         // Der Thread ist für 100ms "schlafend".
        Msg_WriteText("Thread1");  // Danach wird "Thread1" ausgegeben.
    }
}
```

5.26 Timer

5.26.1 Mega

Es stehen im C-Control Pro Mega 32 zwei, Mega128 drei unabhängige Timer-Counter zur Verfügung. *Timer_0* mit 8 Bit und *Timer_1* mit 16 Bit *Timer_3* mit 16 Bit (nur Mega128). *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

5.26.1.1 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();
```

Beim **Mega128** ist aus Hardwaregründen die Benutzung von *Timer_0* als Zähler nicht möglich!

Timer1 (16 Bit)

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

5.26.1.2 Frequenzerzeugung

Zur Frequenzerzeugung können *Timer_0*, *Timer_1* und *Timer_3* folgendermaßen eingesetzt werden:

Timer0 (8 Bit)

1. Beispiel:

```
// Rechtecksignal mit 10*1,085 µs = 10,85 µs Periodendauer
Timer_T0FRQ(10, PS0_8)
```

2. Beispiel: gepulste Frequenzblöcke (Projekt FRQ0)

```
void main(void)
{
    int delval;              // Variable für die Ein-/Ausschaltzeit
    delval=200;              // Wertzuweisung der Variablen delval

    Timer_T0FRQ(100,PS0_1024); // Der Timer wird auf die Frequenz
                               // Periode=138,9µs*100=13,9ms,Frequenz= 2Hz
    while (1)
    {
        AbsDelay(delval);        // Zeitverzögerung um 200ms
        Timer_T0Stop();          // Der Timer wird angehalten.
        AbsDelay(delval);        // Zeitverzögerung um 200ms
        Timer_T0Start(PS0_1024); // Der Timer wird mit dem Timer Prescaler
                                 // PS0_1024 eingeschaltet.
    }
}
```

➔ Das Programm ist auf dem **Mega128** nicht im USB Modus funktionsfähig, da der Ausgang PB4 im Zusammenhang mit dem USB Interface auf dem Application Board genutzt wird.

Timer1 (16 Bit)

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

```
Timer_T1FRQ(125,PS_64);
```

Timer3 (16 Bit) (nur Mega128)

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

```
Timer_T3FRQX(10,2,PS_8);
```

5.26.1.3 Frequenzmessung

Zur direkten Messung einer Frequenz, kann *Timer_1* (16Bit) bzw. *Timer_3* (16Bit) (nur Mega128) 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.

5.26.1.4 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. Auf dem Mega128 kann zusätzlich *Timer_3* genutzt werden.

Timer0 (8 Bit)

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

```
//  Puls: 10*542,5 ns = 5,42 µs, Periode: 256*542,5 ns = 138,9 µs
Timer_T0PWM(10,PS0_8);

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(100,20,10,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
```

➔ Bei den Timer PWM Funktionen ist ein Wert von Null für den Pulsweiten Parameter nicht erlaubt, und schaltet den Ausgang des PIN auch nicht aus. Um ein low Signal zu erzeugen, muss der Timer ausgeschaltet werden (Timer_Disable) und der PIN auf Ausgang geschaltet werden. Benutzt man eine PWM Funktion die mehrere PWM Signale erzeugt, dann eine PWM Funktion aufrufen (z.B.Timer_T1PWM), die den PIN, der auf low geschaltet werden soll, nicht umfaßt.

Ein Beispiel:

```
while(1)
{
  Timer_T1PWMX(255,128,128,PS_8);
  Timer_T1PWA(128);
  Timer_T1PWB(128);

  AbsDelay(1000);

  // OC1B ausschalten
  Timer_Disable(1);
  Timer_T1PWM(255,128,PS_8);
  Port_DataDirBit(14,1);
  Port_WriteBit(14,0);
}
```

5.26.1.5 Puls & Periodenmessung

Mit *Timer_1* oder *Timer_3* (nur Mega128) 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 (Projekt PMessung) 434 µs (100 x 4,34 µs, siehe Tabelle) einschalten

```
word PM_Wert;

void Timer1_ISR(void)
{
    int irqcnt;
```

```
    PM_Wert=Timer_T1GetPM();   // Pulsweite auslesen
    irqcnt=Irq_GetCount(INT_TIM1CAPT);
}

void main(void)
{
    byte n;

 // Interrupt Service Routine definieren
    Irq_SetVect(INT_TIM1CAPT,Timer1_ISR);

    Timer_T0PWM(100,PS0_64);        // Pulsgenerator Timer 0 starten

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

    PM_Wert=0;
 // Pulsweite messen einstellen, Vorteiler für Messung festlegen
    Timer_T1PM(0,PS_64);

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

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

➔ Aus Übersichtsgründen ist hier eine vereinfachte Version angegeben. Beim **Mega128** wird wegen einer Kollision auf Pin B.4 der *Timer_0* zur Pulserzeugung benutzt. Das vollständige Programm ist im Ordner PW_Messung zu finden.

5.26.1.6 Timerfunktionen

Es stehen zwei (Mega32) bzw. drei (Mega128) unabhängige Timer zur Verfügung. *Timer_0* mit 8 Bit, *Timer_1* und *Timer_3* mit 16 Bit (nur Mega128). Die Timer verfügen über einen programmierbaren Vorteiler. 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 Routine definieren
    Timer_T0Time(100,PS0_1024);           // Zeit festlegen und Timer0 starten
    //  weiterer Programmablauf...
}
```

5.26.1.7 Timer_Disable

Timer Funktionen

Syntax

```
void Timer_Disable(byte timer);

Sub Timer_Disable(timer As Byte)
```

Beschreibung

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

Parameter

0 = *Timer_0*
1 = *Timer_1*
3 = *Timer_3* (nur Mega128)

5.26.1.8 Timer_T0CNT

Timer Funktionen

Syntax

```
void Timer_T0CNT(void);

Sub Timer_T0CNT()
```

Beschreibung

Diese Funktion initialisiert den Counter0. Der Counter0 wird bei einer positiven Signalflanke an Eingang **Mega32**:T0 (PIN1)
inkrementiert.

➔ Beim **Mega128** ist aus Hardwaregründen die Benutzung von *Timer_0* als Zähler nicht möglich!

Parameter

Keine

5.26.1.9 Timer_T0FRQ

Timer Funktionen

Syntax

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

Sub Timer_T0FRQ(period As Byte, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer0, mit dem angegebenen Vorteiler und Periodendauer, siehe Tabelle . Das Ausgangssignal erscheint an **Mega32**: PortB.3 (PIN4), **Mega128**: PortB.4 (X1_4). Die Frequenzerzeugung wird automatisch gestartet. Der Mega128 verfügt über erweiterte Vorteilerdefinitionen siehe Tabelle.

Parameter

period Periodendauer
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega32
PS0_1 (1)	135,6 ns
PS0_8 (2)	1,085 µs
PS0_64 (3)	8,681 µs
PS0_256 (4)	34,72 µs
PS0_1024 (5)	138,9 µs

Vorteiler (prescaler)	Tickdauer Mega128	Tickdauer Mega128 CAN
PS0_1 (1)	135,6 ns	125 ns
PS0_8 (2)	1,085 µs	1 µs
PS0_32 (3)	4,340 µs	4 µs
PS0_64 (4)	8,681 µs	8µs
PS0_128 (5)	17,36 µs	16 µs
PS0_256 (6)	34,72 µs	32 µs
PS0_1024 (7)	138,9 µs	128 µs

5.26.1.10 Timer_T0GetCNT

Timer Funktionen

Syntax

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

```
Sub Timer_T0GetCNT() As Byte
```

Beschreibung

Der Wert des Counter0 wird gelesen. Erfolgte ein Überlauf, dann wird der Wert FF (Hex) übergeben.

➔ Beim **Mega128** ist aus Hardwaregründen die Benutzung von *Timer_0* als Zähler nicht möglich!

Rückgabewert

der gemessene Zählerwert

5.26.1.11 Timer_T0PW

Timer Funktionen

Syntax

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

```
Sub Timer_T0PW(PW As Byte)
```

Beschreibung

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

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

PW Pulsweite

5.26.1.12 Timer_T0PWM

Timer Funktionen

Syntax

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

```
Sub Timer_T0PWM(PW As Byte, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer0, mit dem angegebenen Vorteiler und Pulsweite, siehe Tabelle . Das Ausgangssignal erscheint an **Mega32**: PortB.3 (PIN4) **Mega128**: PortB.4(X1_4). Der Mega128 verfügt über erweiterte Vorteilerdefinitionen siehe Tabelle.

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

PW Pulsweite
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega32
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_64 (3)	4,34 µs
PS0_256 (4)	17,36 µs
PS0_1024 (5)	69,44 µs

Vorteiler (prescaler)	Tickdauer Mega128	Tickdauer Mega128 CAN
PS0_1 (1)	67,8 ns	62,5 ns
PS0_8 (2)	542,5 ns	500 ns
PS0_32 (3)	2,17 µs	2 µs
PS0_64 (4)	4,34 µs	4 µs
PS0_128 (5)	8,68 µs	8 µs
PS0_256 (6)	17,36 µs	16 µs
PS0_1024 (7)	69,44 µs	64 µs

5.26.1.13 Timer_T0Start

Timer Funktionen

Syntax

```
void Timer_T0Start(byte prescaler);

Sub Timer_T0Start(prescaler As Byte)
```

Beschreibung

Der Timer läuft mit der vorherigen Einstellung weiter. Der Vorteiler muss neu angegeben werden.´

Parameter

prescaler Vorteiler (Tabelle prescaler)

5.26.1.14 Timer_T0Stop

Timer Funktionen

Syntax

```
void Timer_T0Stop(void);

Sub Timer_T0Stop()
```

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

5.26.1.15 Timer_T0Time

Timer Funktionen

Syntax

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

Sub Timer_T0Time(Time As Byte, PS As Byte)
```

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. Der Mega128 verfügt über erweiterte Vorteilerdefinitionen siehe Tabelle.

Parameter

Time Zeitwert bei dem Interrupt ausgelöst wird
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega32
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_64 (3)	4,34 µs

PS0 256 (4)	17,36 µs
PS0 1024 (5)	69,44 µs

Vorteiler (prescaler)	Tickdauer Mega128	Tickdauer Mega128 CAN
PS0 1 (1)	67,8 ns	62,5 ns
PS0 8 (2)	542,5 ns	500 ns
PS0 32 (3)	2,17 µs	2 µs
PS0 64 (4)	4,34 µs	4 µs
PS0 128 (5)	8,68 µs	8 µs
PS0 256 (6)	17,36 µs	16 µs
PS0 1024 (7)	69,44 µs	64 µs

5.26.1.16 Timer_T1CNT

Timer Funktionen

Syntax

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

```
Sub Timer_T1CNT()
```

Beschreibung

Diese Funktion initialisiert den Counter1. Der Counter1 wird bei einer positiven Signalflanke an Eingang **Mega32**: PortB.1 (PIN2) **Mega128**: PortD.6 (X2_15). inkrementiert.

Parameter

Keine

5.26.1.17 Timer_T1CNT_Int

Timer Funktionen

Syntax

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

```
Sub Timer_T1CNT_Int(limit As Word)
```

Beschreibung

Diese Funktion initialisiert den Counter1. Der Counter1 wird bei einer positiven Signalflanke an Eingang **Mega32**: PortB.1 (PIN2) **Mega128**: PortD.6 (X2_15). inkrementiert. Wenn das Limit erreicht ist, wird ein Interrupt ("Timer1 CompareA" - define: INT_TIM1CMPA) ausgelöst. Die entsprechende Interrupt_Service_Routine muss vorher definiert sein.

Parameter

limit

5.26.1.18 Timer_T1FRQ

Timer Funktionen

Syntax

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

Sub Timer_T1FRQ(period As Word, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer1, mit dem angegebenen Vorteiler und Periodendauer, siehe Tabelle . Das Ausgangssignal erscheint an **Mega32**: PortD.5 (PIN19). **Mega128**: PortB.5 (X1_3). Die Frequenzerzeugung wird automatisch gestartet.

Parameter

period Periodendauer
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega32 + Mega128	Tickdauer Mega128 CAN
PS_1 (1)	135,6 ns	125 ns
PS_8 (2)	1,085 µs	1 µs
PS_64 (3)	8,681 µs	8 µs
PS_256 (4)	34,72 µs	32 µs
PS_1024 (5)	138,9 µs	128 µs

5.26.1.19 Timer_T1FRQX

Timer Funktionen

Syntax

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

Sub Timer_T1FRQX(period As Word, skew As Word, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer1, mit dem angegebenen Vorteiler, Periodendauer und Phasenverschiebung der beiden Ausgangssignale, siehe Tabelle . Die Ausgangssignale erscheinen an **Mega32**: PortD.4 (PIN18) und PortD.5 (PIN19). **Mega128**: PortB.5 (X1_3) und PortB.6 (X1_2). Die Frequenzerzeu-

gung wird automatisch gestartet. Der Wert für die Phasenverschiebung muss kleiner sein als die halbe Periode.

Parameter

period Periodendauer
skew Phasenverschiebung
PS Vorteiler (Tabelle prescaler)

5.26.1.20 Timer_T1GetCNT

Timer Funktionen

Syntax

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

```
Sub Timer_T1GetCNT() As Word
```

Beschreibung

Der Wert des Counter1 wird gelesen. Erfolgte ein Überlauf, dann wird der Wert FFFF (Hex) übergeben.

Rückgabewert

der gemessene Zählerwert

5.26.1.21 Timer_T1GetPM

Timer Funktionen

Syntax

```
word Timer_T1GetPM(void);
```

```
Sub Timer_T1GetPM() As Word
```

Beschreibung

Diese Funktion liefert das Messergebnis zurück.

Parameter

Keine

Rückgabewert

Ergebnis der Messung

➔ Um das Meßergebnis zu errechnen, wird der zurückgegebene 16bit Wert mit dem Eintrag aus der prescaler_Tabelle multipliziert, der beim Aufruf von `Timer_T1PM` angegeben wurde (siehe

auch Beispiel).

5.26.1.22 Timer_T1PWA

Timer Funktionen

Syntax

```
void Timer_T1PWA(word PW0);

Sub Timer_T1PWA(PW0 As Word)
```

Beschreibung

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

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

PW0 Pulsweite

5.26.1.23 Timer_T1PM

Timer Funktionen

Syntax

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

void Timer_T1PM(Mode As Byte, PS As Byte)
```

Beschreibung

Diese Funktion legt fest, ob eine Pulsbreiten- oder Periodenmessung durchgeführt werden soll, initialisiert den Timer_1 für die Messung und setzt den Vorteiler.

Parameter

Mode 0 = Pulsweitenmessung, 1 = Periodenmessung
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega32 + Mega128	Tickdauer Mega128 CAN
PS 1 (1)	67,8 ns	62,5 ns
PS 8 (2)	542,5 ns	500 ns
PS 64 (3)	4,34 µs	4 µs

PS_256 (4)	17,36 µs	16 µs
PS_1024 (5)	69,44 µs	64 µs

5.26.1.24 Timer_T1PWB

Timer Funktionen

Syntax

```
void Timer_T1PWB(word PW1);

Sub Timer_T1PWB(PW1 As Word)
```

Beschreibung

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

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

PW1 Pulsweite

5.26.1.25 Timer_T1PWM

Timer Funktionen

Syntax

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

Sub Timer_T1PWM(period As Word, PW0 As Word, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer1 mit dem angegebenen Vorteiler, Pulsweite und Periodendauer, siehe Tabelle . Die Ausgangssignale erscheinen an **Mega32**: PortD.4 (PIN18). **Mega128**: PortB.5 (X1_3) .

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

period Periodendauer
PW0 Pulsweite
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega32 + Mega128	Tickdauer Mega128 CAN

PS_1 (1)	67,8 ns	62,5 ns
PS_8 (2)	542,5 ns	500 ns
PS_64 (3)	4,34 µs	4 µs
PS_256 (4)	17,36 µs	16 µs
PS_1024 (5)	69,44 µs	64 µs

5.26.1.26 Timer_T1PWMX

Timer Funktionen

Syntax

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

Sub Timer_T1PWMX(period As Word, PW0 As Word, PW1 As Word, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer1 mit dem angegebenen Vorteiler, Pulsweite für Kanal A und B und Periodendauer, siehe Tabelle. Die Ausgangssignale erscheinen an **Mega32**: PortD.4 (PIN18) und PortD.5 (PIN19). **Mega128**: PortB.5 (X1_3) und PortB.6 (X1_2).

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

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

5.26.1.27 Timer_T1PWMY

Timer Funktionen

Syntax

```
void Timer_T1PWMY(word period, word PW0, word PW1, word PW2, byte PS);

Sub Timer_T1PWMY(period As Word, PW0 As Word, PW1 As Word, PW2 As Word,
PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer1 mit dem angegebenen Vorteiler, Pulsweite für Kanal A ,B und C und Periodendauer, siehe Tabelle . Die Ausgangssignale erscheinen an PortB.5 (X1_3) , PortB.6 (X1_2) und PortB.7 (X1_1).

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

period	Periodendauer
PW0	Pulsweite Kanal A
PW1	Pulsweite Kanal B
PW2	Pulsweite Kanal C
PS	Vorteiler (Tabelle prescaler)

5.26.1.28 Timer_T1Start

Timer Funktionen

Syntax

```
void Timer_T1Start(byte prescaler);

Sub Timer_T1Start(prescaler As Byte)
```

Beschreibung

Der Timer läuft mit der vorherigen Einstellung weiter. Der Vorteiler muss neu angegeben werden.

Parameter

prescaler	Vorteiler (Tabelle prescaler)

5.26.1.29 Timer_T1Stop

Timer Funktionen

Syntax

```
void Timer_T1Stop(void);

Sub Timer_T1Stop()
```

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

5.26.1.30 Timer_T1Time

Timer Funktionen

Syntax

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

Sub Timer_T1Time(Time As Word, PS As Byte)
```

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_TIM1CMPA) ausgelöst.

Parameter

Time Zeitwert, bei dem Interrupt ausgelöst wird
PS Vorteiler

Tabelle prescaler:´

Vorteiler (prescaler)	Tickdauer Mega32 + Mega128	Tickdauer Mega128 CAN
PS_1 (1)	67,8 ns	62,5 ns
PS_8 (2)	542,5 ns	500 ns
PS_64 (3)	4,34 µs	4 µs
PS_256 (4)	17,36 µs	16 µs
PS_1024 (5)	69,44 µs	64 µs

5.26.1.31 Timer_T3CNT

Timer Funktionen

Syntax

```
void Timer_T3CNT(void);

Sub Timer_T3CNT()
```

Beschreibung

Diese Funktion initialisiert den Counter3. Der Counter3 wird bei einer positiven Signalflanke an Eingang PortE.6 (X1_10) inkrementiert.

Parameter

Keine

5.26.1.32 Timer_T3CNT_Int

Timer Funktionen

Syntax

```
void Timer_T3CNT_Int(word limit);

Sub Timer_T3CNT_Int(limit As Word)
```

Beschreibung

Diese Funktion initialisiert den *Counter_3*. Der *Counter_3* wird bei einer positiven Signalflanke an Eingang PortE.6 (X1_10) inkrementiert. Wenn das Limit erreicht ist, wird ein Interrupt ("Timer3 CompareA" - define: INT_TIM3CMPA) ausgelöst. Die entsprechende Interrupt Service Routine muss vorher definiert sein.

Parameter

limit

5.26.1.33 Timer_T3FRQ

Timer Funktionen

Syntax

```
void Timer_T3FRQ(word period, byte PS);

Sub Timer_T3FRQ(period As Word, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer3, mit dem angegebenen Vorteiler und Periodendauer, siehe Tabelle . Das Ausgangssignal erscheint an PortE.3 (X1_13). Die Frequenzerzeugung wird automatisch gestartet.

Parameter

period Periodendauer
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega128	Tickdauer Mega128 CAN
PS_1 (1)	135,6 ns	125 ns
PS_8 (2)	1,085 µs	1 µs
PS_64 (3)	8,681 µs	8 µs
PS_256 (4)	34,72 µs	32 µs
PS_1024 (5)	138,9 µs	128 µs

5.26.1.34 Timer_T3FRQX

Timer Funktionen

Syntax

```
void Timer_T3FRQX(word period, word skew, byte PS);

Sub Timer_T3FRQX(period As Word, skew As Word, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer3, mit dem angegebenen Vorteiler, Periodendauer und Phasenverschiebung der beiden Ausgangssignale, siehe Tabelle . Die Ausgangssignale erscheinen an PortE.3 (X1_13) und PortE.4 (X1_12). 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)

5.26.1.35 Timer_T3GetCNT

Timer Funktionen

Syntax

```
word Timer_T3GetCNT(void);

Sub Timer_T3GetCNT() As Word
```

Beschreibung

Der Wert des Counter3 wird gelesen. Erfolgte ein Überlauf, dann wird der Wert FFFF (Hex) übergeben.

Rückgabewert

der gemessene Zählerwert

5.26.1.36 Timer_T3GetPM

Timer Funktionen

Syntax

```
word Timer_T3GetPM(void);

Sub Timer_T3GetPM() As Word
```

Beschreibung

Diese Funktion liefert das Messergebnis zurück.

Parameter

Keine

Rückgabewert

Ergebnis der Messung

➔ Um das Meßergebnis zu errechnen, wird der zurückgegebene 16bit Wert mit dem Eintrag aus der prescaler Tabelle multipliziert, der beim Aufruf von `Timer_T3PM` angegeben wurde (siehe auch `Beispiel`).

5.26.1.37 Timer_T3PWA

Timer Funktionen

Syntax

```
void Timer_T3PWA(word PW0);

Sub Timer_T3PWA(PW0 As Word)
```

Beschreibung

Diese Funktion stellt eine neue Pulsweite (Kanal_A) für den Timer3 ein, ohne den Vorteiler zu verändern.

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

PW0 Pulsweite

5.26.1.38 Timer_T3PM

Timer Funktionen

Syntax

```
void Timer_T3PM(byte Mode, byte PS);

void Timer_T3PM(Mode As Byte, PS As Byte)
```

Beschreibung

Diese Funktion legt fest, ob eine Pulsbreiten- oder Periodenmessung durchgeführt werden soll, initialisiert den *Timer_3* für die Messung und setzt den Vorteiler.

Parameter

Mode 0 = Pulsweitenmessung, 1 = Periodenmessung
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega128	Tickdauer Mega128 CAN
PS_1 (1)	67,8 ns	62,5 ns
PS_8 (2)	542,5 ns	500 ns
PS_64 (3)	4,34 µs	4 µs
PS_256 (4)	17,36 µs	16 µs
PS_1024 (5)	69,44 µs	64 µs

5.26.1.39 Timer_T3PWB

Timer Funktionen

Syntax

```
void Timer_T3PWB(word PW1);

Sub Timer_T3PWB(PW1 As Word)
```

Beschreibung

Diese Funktion stellt eine neue Pulsweite (Kanal_B) für den Timer3 ein, ohne den Vorteiler zu verändern.

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

PW1 Pulsweite

5.26.1.40 Timer_T3PWM

Timer Funktionen

Syntax

```
void Timer_T3PWM(word period,word PW0,byte PS);

Sub Timer_T3PWM(period As Word,PW0 As Word,PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer3 mit dem angegebenen Vorteiler, Pulsweite und Periodendauer, siehe Tabelle . Das Ausgangssignal erscheint an PortE.3 (X1_13).

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

period Periodendauer
PW0 Pulsweite
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega128	Tickdauer Mega128 CAN
PS_1 (1)	67,8 ns	62,5 ns
PS_8 (2)	542,5 ns	500 ns
PS_64 (3)	4,34 µs	4 µs
PS_256 (4)	17,36 µs	16 µs
PS_1024 (5)	69,44 µs	64 µs

5.26.1.41 Timer_T3PWMX

Timer Funktionen

Syntax

```
void Timer_T3PWMX(word period, word PW0, word PW1, byte PS);

Sub Timer_T3PWMX(period As Word, PW0 As Word, PW1 As Word, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer3 mit dem angegebenen Vorteiler, Pulsweite für Kanal A und B und Periodendauer, siehe Tabelle . Die Ausgangssignale erscheinen an PortE.3 (X1_13) und PortE.4 (X1_12).

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

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

5.26.1.42 Timer_T3PWMY

Timer Funktionen

Syntax

```
void Timer_T3PWMY(word period, word PW0, word PW1, word PW2, byte PS);

Sub Timer_T3PWMY(period As Word, PW0 As Word, PW1 As Word, PW2 As Word,
PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den Timer3 mit dem angegebenen Vorteiler, Pulsweite für Kanal A ,B und C und Periodendauer, siehe Tabelle . Die Ausgangssignale erscheinen an PortE.3 (X1_13), PortE.4 (X1_12) und PortE.5 (X1_11) .

➔ Für den Pulsweitenparameter nicht den Wert Null benutzen. Siehe Pulsweitenmodulation

Parameter

period	Periodendauer
PW0	Pulsweite Kanal A
PW1	Pulsweite Kanal B
PW2	Pulsweite Kanal C
PS	Vorteiler (Tabelle prescaler)

5.26.1.43 Timer_T3Start

Timer Funktionen

Syntax

```
void Timer_T3Start(byte prescaler);

Sub Timer_T3Start(prescaler As Byte)
```

Beschreibung

Der Timer läuft mit der vorherigen Einstellung weiter. Der Vorteiler muss neu angegeben werden.

Parameter

prescaler	Vorteiler (Tabelle prescaler)

5.26.1.44 Timer_T3Stop

Timer Funktionen

Syntax

```
void Timer_T3Stop(void);

Sub Timer_T3Stop()
```

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

5.26.1.45 Timer_T3Time

Timer Funktionen

Syntax

```
void Timer_T3Time(word Time, byte PS);

Sub Timer_T3Time(Time As Word, PS As Byte)
```

Beschreibung

Diese Funktion initialisiert den *Timer_3* mit dem angegebenen Vorteiler, und dem Wert (16 Bit) für die Zeit, siehe Tabelle . Ist der Wert erreicht, dann wird der Timer3- Interrupt (INT_TIM3CMPA) ausgelöst.

Parameter

Time Zeitwert, bei dem Interrupt ausgelöst wird
PS Vorteiler

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer Mega128	Tickdauer Mega128 CAN
PS_1 (1)	67,8 ns	62,5 ns
PS_8 (2)	542,5 ns	500 ns
PS_64 (3)	4,34 µs	4 µs
PS_256 (4)	17,36 µs	16 µs
PS_1024 (5)	69,44 µs	64 µs

5.26.1.46 Timer_TickCount

Timer Funktionen

Syntax

```
word Timer_TickCount(void);

Sub Timer_TickCount() As Word
```

Beschreibung

Mißt die Zeit in 10ms Ticks zwischen zwei Aufrufen von Timer_TickCount() und gibt den Wert beim zweiten Aufruf von Timer_TickCount() zurück. Der Rückgabewert beim ersten Aufruf kann ignoriert werden.

Parameter

Keine

Rückgabewert

Zeitdifferenz zwischen zwei Aufrufen

Beispiel

```
void main(void)
{
    word time;
    Timer_TickCount();
    AbsDelay(500);  // 500 ms warten
    time=Timer_TickCount();  // der Wert von time sollte 50 sein
}
```

5.26.2 AVR32Bit

Es stehen im C-Control Pro AVR32Bit 2 Timer mit 3 Kanälen zur Verfügung. Diese werden in der Bibliothek auf 6 Timer abgebildet. Man kann die internen Timer für vielfältige Aufgaben einsetzen:

- Ereigniszähler
- Frequenzerzeugung

Zusätzlich gibt es 3 dedizierte Funktionseinheiten für die Pulsweitenmodulation.

5.26.2.1 Ereigniszähler

Auf jedem der 6 Timerkanäle können Ereignisse (max. 16-Bit) gezählt werden. In diesem Beispiel werden die steigenden Flanken auf Eingang COUNTA_1 (siehe Pinzuordnung) in Timer 2 gezählt. Alle 300 Ereignisse wird ein Interrupt ausgelöst, dabei wird der Zähler zurückgesetzt. Das Beispielprogramm gibt nach 10 Sekunden den aktuellen Stand des Zählers aus.

Beispiel

```
word cnt;

void count_irq(void)
{
    cnt++;
    Irq_GetCount(INT_TIMER0);
}

void main(void)
{
    cnt= 0;
    Irq_SetVect(INT_TIMER2, count_irq);
    Timer_ConfigCounter(2, COUNTA_1, CNT_RISING, 300);

    AbsDelay(10000);
    Msg_WriteWord(Timer_GetCounterVal(2));
    Msg_WriteChar('\r');

    while(1);
}
```

5.26.2.2 Frequenzerzeugung

Auf jedem der 6 Timerkanäle kann ein Rechtecksignal erzeugt werden. Im folgenden Beispiel wird ein 50Hz Signal erzeugt, das auf den Pin TIMER0-A und TIMER0-B (siehe Pinzuordnung) ausgegeben wird. Zusätzlich wird ein Interrupt ausgelöst. Der Vorteiler TIM_128 bestimmt eine Tickdauer von 1,939µs (= 128 / 66.000.000 Mhz). Durch 5157 * 1,939µs = 10ms ergeben sich 100 Flankenwechsel pro Sekunde = 50Hz.

Beispiel

```
word cnt;

void irq(void)
{
    cnt++;
    Irq_GetCount(INT_TIMER0);
}

void main(void)
{
    cnt= 0;
    Irq_SetVect(INT_TIMER0, irq);
    Timer_Set(0, TIM_128, 5157, TIMFLG_IRQ|TIMFLG_PINA|TIMFLG_PINB);
    while(1);
}
```

5.26.2.3 Pulsweitenmodulation

Der C-Control AVR32Bit kann auf bis zu 4 Kanälen ein pulsweitenmoduliertes Signal ausgeben. Im folgenden Beispiel wird auf PWM Kanal 1 ein Signal mit 1,65Mhz Periode und 50% Duty erzeugt. Nach 10 Sekunden wird der PWM Kanal abgeschaltet. Das Signal wird auf den Pin PWMH_1 und PWML_1 ausgegeben (siehe Pinzuordnung).

Beispiel

```
void main(void)
{
    PWM_Init(1, PWM_1, PWM_ENAB_HIGH|PWM_ENAB_LOW);
    PWM_Update(1, 40L, 20L, 0, 0);

    AbsDelay(10000);
    PWM_Disable(1);
}
```

5.26.2.4 PWM_Disable

Timer Funktionen

Syntax

```
void PWM_Disable(byte chan);

Sub PWM_Disable(chan As Byte)
```

Beschreibung

Die Funktion schaltet den selektierten PWM Kanal ab.

Parameter

chan Nummer des PWM Kanals (0 - 3)

5.26.2.5 PWM_Init

Timer Funktionen

Syntax

```
void PWM_Init(byte chan, byte PS, byte mode);

Sub PWM_Init(chan As Byte, PS As Byte, mode As Byte)
```

Beschreibung

Initialisiert einen PWM Kanal. Es kann mit mode einzeln gewählt werden, ob das Signal auf **PWMH_x** und/oder **PWML_x** ausgegeben wird. Es kann eine **deadtime** aktiviert werden, oder die Polarität

negiert. Zum erstellen des mode Parameter werden die Bitwerte aus der mode Tabelle oderiert (siehe PWM_Beispiel).

➔ Für detailliertere Informationen über PWMH_x, PWML_x und deadtime, bitte das AT32UC3C Datenblatt konsultieren.

Parameter

chan	Nummer des PWM Kanals (0 - 3)
PS	Vorteiler
mode	Arbeitsmodus des PWM Kanals

Tabelle PS:

Vorteiler (prescaler)	Tickdauer
PWM_1(0)	15,15 ns
PWM_2(1)	30,30 ns
PWM_4(2)	60,60 ns
PWM_8(3)	121,21 ns
PWM_16(4)	242,42 ns
PWM_32(5)	484,84 ns
PWM_64(6)	969,69 ns
PWM_128(7)	1,939 µs
PWM_256(8)	3,878 µs
PWM_512(9)	7,757 µs
PWM_1024(10)	15,51 µs

Tabelle mode:

mode	Beschreibung
PWM_ENAB_HIGH (1)	Signal auf Pin PWMH_x
PWM_ENAB_LOW (2)	Signal auf Pin PWML_x
PWM_ENAB_DEAD (4)	Aktiviert Deadtime
PWM_CPOL (8)	Negiert Polarität

5.26.2.6 PWM_Update

Timer Funktionen

Syntax

```
void PWM_Update(byte chan, dword period, dword duty, word dtl, word dth);

Sub PWM_Update(chan As Byte, period As ULong, duty As ULong, dtl As Word,
               dth As Word)
```

Beschreibung

Im laufenden Betrieb kann period (Frequenz), duty (Breite) und **deadtime** des PWM Signals angegeben werden.

➔ Für die Frequenz gilt folgende Formel: Frq = 66.000.000 / Vorteiler / period. Erlaubte Werte für duty sind 0 bis zu period. Bei einem Duty von 0 ist das Signal aber dauerhaft aus, bei einem Duty von period dauerhaft an. Ein Duty von 50% sind demnach period / 2. Damit die Breite (duty) des PWM Signals möglichst fein eingestellt werden kann, sollte ein Vorteiler so gewählt werden, das für die gewünschte Frequenz der period Parameter möglichst groß ist (maximal 20 Bit).

Parameter

chan	Nummer des PWM Kanals (0 - 3)
period	Frequenz des PWM Signals (20 Bit)
duty	Breite des PWM Signals (20 Bit)
dtl	deadtime des **PWML_x** Signals
dth	deadtime des **PWMH_x** Signals

5.26.2.7 Timer_ConfigCounter

Timer Funktionen

Syntax

```
void Timer_ConfigCounter(byte timer, byte portbit, byte edge, word irq_threshold);

Sub Timer_ConfigCounter(timer As Byte, portbit As Byte, edge As Byte, irq_threshold As Word)
```

Beschreibung

Die Funktion initialisiert einen Timer als Zähler. Es stehen die Eingänge COUNTA-0, COUNTA-1, COUNTA-2 und COUNTB-2 zur Verfügung (siehe Pinzuordnung). Es können für die COUNTA-x Eingänge nur die Timer 0,2,4 genutzt werden, für COUNTB-2 die Timer 1,3,5. Ist der Parameter irq_threshold ungleich Null, dann wird ein Interrupt ausgelöst, wenn der Zähler gleich dem Wert von irq_threshold ist. Nach einem Interrupt wird der Zähler wieder auf Null zurückgesetzt.

➔ Nach der Initialisierung behält der Zähler erstmal seinen alten Wert. Bei der ersten Flanke wird dann der Zähler auf Null gesetzt. Beim Auslesen des Zählers sieht es deshalb so aus, als ob eine Flanke zu wenig gezählt würde. Dieses Verhalten ist bedingt durch den Aufbau des internen Zählers des AVR32 Controllers.

Parameter

timer	Nummer des Timers (0 - 5)
portbit	(**GPIO** in Pinzuordnung)
edge	Flankentyp CNT_FALLING (fallend) oder CNT_RISING (steigend)
irq_threshold	Anzahl der Ereignisse wann ein IRQ ausgelöst wird

5.26.2.8 Timer_CPUCycles

Timer Funktionen

Syntax

```
dword Timer_CPUCycles(void);

Sub Timer_CPUCycles() As ULong
```

Beschreibung

Misst die CPU Zyklen zwischen zwei Aufrufen von Timer_CPUCycles() und gibt den Wert beim zweiten Aufruf von Timer_CPUCycles() zurück. Der Rückgabewert beim ersten Aufruf kann ignoriert werden.

➔ Da der Prozessor mit 66Mhz getaktet ist, können nur Zeiträume von bis zu 65 Sekunden gemessen werden.

Parameter

Keine

Rückgabewert

CPU Zyklen zwischen zwei Aufrufen

5.26.2.9 Timer_Disable

Timer Funktionen

Syntax

```
void Timer_Disable(byte timer);

Sub Timer_Disable(timer As Byte)
```

Beschreibung

Die Funktion schaltet den selektierten Timer bzw. Counter ab.

Parameter

timer Nummer des Timers (0 - 5)

5.26.2.10 Timer_GetCounterVal

Timer Funktionen

Syntax

```
word Timer_GetCounterVal(byte timer);
```

```
Sub Timer_GetCounterVal(timer As Byte) As Word
```

Beschreibung

Gibt den 16-Bit Counter eines Timers zurück.

Parameter

timer Nummer des Timers (0 - 5)

Rückgabewert

Zählerwert

5.26.2.11 Timer_Set

Timer Funktionen

Syntax

```
void Timer_Set(byte timer, byte PS, word period, word flags);

Sub Timer_Set(timer As Byte, PS As Byte, period As Word, flags As Word)
```

Beschreibung

Diese Funktion initialisiert den Timer, mit dem angegebenen Vorteiler und Periodendauer, siehe Tabelle. Durch die Anwendung der flags (man kann mehrere Werte oderieren), wird ein Interrupt ausgelöst und/ oder das Signal auf den Pins TIMERx-A bzw. TIMERx-B erzeugt (siehe Pinzuordnung).

➔ Aufgrund der Konfiguration der angeschlossenen Peripherie sind nicht alle Pin TIMERx-A und TIMERx-B verfügbar.

Parameter

timer Nummer des Timers (0 - 5)
PS Vorteiler
period Periodendauer
flags Timer Optionen

Tabelle prescaler:

Vorteiler (prescaler)	Tickdauer
TIM_32KHZ (0)	31,25 µs
TIM_2 (1)	30,30 ns
TIM_8 (2)	121,21 ns
TIM_32 (3)	484,84 ns

TIM_128 (4)	1,939 µs

Tabelle flags:

Definition	Bedeutung
TIMFLG_IRQ	Es werden Interrupts generiert
TIMFLG_PINA	Das Signal wird auf Pin TIMERx-A ausgegeben
TIMFLG_PINB	Das Signal wird auf Pin TIMERx-B ausgegeben

5.26.2.12 Timer_TickCount

Timer Funktionen

Syntax

```
dword Timer_TickCount(void);

Sub Timer_TickCount() As ULong
```

Beschreibung

Mißt die Zeit in 10ms Ticks zwischen zwei Aufrufen von Timer_TickCount() und gibt den Wert beim zweiten Aufruf von Timer_TickCount() zurück. Der Rückgabewert beim ersten Aufruf kann ignoriert werden.

Parameter

Keine

Rückgabewert

Zeitdifferenz zwischen zwei Aufrufen

Beispiel

```
void main(void)
{
    dword time;
    Timer_TickCount();
    AbsDelay(500);  // 500 ms warten
    time=Timer_TickCount();  // der Wert von time sollte 50 sein
}
```

5.27 Webserver (AVR32Bit)

Der Webserver der C-Control Pro AVR32Bit wird mit WEB_StartServer gestartet. Es kann dafür ein beliebiger TCP/IP Port ausgesucht werden. Beim Start des Webservers wird die Anzahl der dynamischen Variablen definiert, mit denen man arbeiten möchte. Die dynamischen Variablen übernehmen die Werte von URL Variablen beim Aufruf der Webseite und man kann dynamische Variablen zur Ausgabe von Werten innerhalb von Webseiten benutzen.

Alle Webseiten die vom Server zurückgeliefert werden, müssen im Hauptverzeichnis auf der SD card liegen, die in die C-Control Pro Unit eingesteckt ist. Da die SDCard Bibliothek keine langen Dateinamen unterstützt, müssen alle Dateinamen der Webseiten im DOS Format (8.3) vorliegen. Die Hauptseite heißt deshalb "index.htm". Man beachte die verkürzte Endung.

HTTP Header

Für Dateien mit einer bekannten Endung (siehe Tabelle) wird automatisch ein HTTP Header erzeugt, der vor den Inhalt der Datei gesetzt wird. Dabei wird immer

```
HTTP/1.1 200 OK\r\n
Connection: close\r\n
Content-Type: Type\r\n
\r\n
```

vorangesetzt. Dabei ist "\r\n" Carriage Return Linefeed, und **Type** der korrespondierende Inhalt aus der Tabelle. Für die Endung ".htm" wird z.B. ein "Content-Type: text/html" im Header generiert.

Datei Endung	Type
.htm	text/html
.js	application/x-javascript
.txt	text/plain
.css	text/css
.gif	image/gif
.ico	image/x-icon
.jpg	image/jpeg
.bmp	image/bmp
.png	image/png

➔ Ist die Dateiendung nicht in der Tabelle vorhanden, muss der Header selbst an den Anfang der Datei auf der SD-Card gesetzt werden.

Dynamische Variablen

Man kann mit der Funktion WEB_SetDynVar() dem Webserver die Adresse und Typ einer normalen Variablen mitgeben. Hat man sich z.B. eine Integer Variable mit "int a;" definiert, so würde man mit "WEB_SetDynVar(0, a, DYN_INT, 0);" die Variable **a** als dynamische Variable mit Index 0 definieren. Schreibt man in eine Webseite den Text **$var0$**, so würde dann **$var0$** durch den numerischen Wert von **a** ersetzt. Die Zahl nach "var" ist der Index der dynamischen Variable.

URL (CGI) Variablen

Werden bei einem Webrequest keine URL Variablen angegeben, so läuft der gesamte Prozess im Hintergrund, und es muss keine Interaktion erfolgen. Wird eine URL Variable angegeben (z.B. "?var0=5") so überprüft der Webserver, ob der Variablenname dem Schema "var" + Zahl entspricht. Die Zahl darf den maximalen Index der definierten dynamischen Variablen nicht überschreiten. Wird das Schema erfüllt, wird der Wert "5" der dynamischen Variablen zugewiesen. Die Integer Variable **a** bekommt dann den Wert 5.

Es gibt einen spezielle URL ("setvars.js") die nur die URL Variablen übernimmt, aber keinen Webseiten Inhalt zurückliefert. Damit lassen sich mit Javascript Variableninhalte übergeben, ohne viel TCP/IP Traffic zu erzeugen.

➔ Eine Variable kann nur über die URL modifiziert werden, wenn bei WEB_SetDynVar() das Flag *DYN_CGIVAR* gesetzt wird. Dadurch können normale Variablen vor einer Änderung von außen geschützt werden.

JSON

Wenn man mit Javascript arbeiten möchte, können dynamische Variablen auch als JSON Liste ausgegeben werden. Hierfür wird bei der der Definition mit WEB_SetDynVar() zusätzlich das Flag *DYN_JSONVAR* gesetzt. Ein Zugriff auf "getvars.js" liefert dann die JSON Daten. Z.B. "{"1":"123","3":"0"}". Dies ist eine Liste von zwei dynamischen Variablen mit den Indizes 1 und 3. Dabei hat die Variable den Inhalt "123", die zweite Variable den Inhalt "0".

Interaktion

Bei der normalen Arbeitsweise, wird in der Hauptschleife des Programms dauernd mit WEB_GetRequest() abgefragt, ob ein Request mit URL Variablen vorliegt. Ist der Rückgabewert ungleich Null, so liegt ein Request an. Man kann mit WEB_GetFileHash() den Hash des Dateinamens erfragen, und die übergebenen URL Variablen auswerten. Danach sollten die Ausgabevariablen (dynamischen Variablen der Webseite) neu gesetzt werden. Am Ende wird dem Webserver mit WEB_ReleaseRequest() signalisiert, das die Webseite ausgeliefert wird.

5.27.1 Webserver Tips

Webserver Checkliste

- Die aufzurufenden Webseiten müssen auf die Micro SD-Karte kopiert worden sein, und die Karte muss im SD-Karten Slot der AVR32Bit Unit eingerastet sein.
- Für die SD-Karte wird nur das FAT Dateisystem unterstützt (siehe FAT Unterstützung).
- Der Webserver wird gestartet, nach dem im Benutzerprogramm ETH_StartWebserver() aufgerufen wird. Der TCP/IP Port im Web-Browser (Default: 80) muss mit dem Port im Aufruf von ETH_StartWebserver übereinstimmen.
- Die Anzahl der in WEB_SetDynVar benutzten dynamischen Variablen muss mit der Definition im WEB_BUF Makro und mit dynvar_cnt in WEB_StartServer korrespondieren.

➔ Man kann beim Stoppen des Programms mit dem Start/Stop Taster den lwIP TCP/IP-Stack in den Zustand bringen, das noch dynamischer Speicher der aktuellen Verbindung nicht frei gegeben wird. Dieser Speicher kann beim Neustart des Programms fehlen. Bei Problemen sollte im Zweifels-

fall der Reset-Taster betätigt werden, um einen kompletten Neustart des Systems auszulösen.

Webserver Optimierung

Der lwIP TCP/IP-Stack ist darauf optimiert, bei embedded Devices möglichst gut mit Webseiten zu arbeiten, die im Flash-Speicher abgelegt sind. Mit der im C-Control Pro AVR32Bit Modul eingebauten SD card kann man sehr viel mehr Daten speichern als im Flash, aber es birgt den Nachteil, das die Webseiten im RAM zwischengeladen werden müssen, bevor sie über das Ethernet verschickt werden. Um den "RAM Hunger" des lwIP-Stacks zu begrenzen, sind mehrere Dinge zu beachten:

- Im normalen Webserverbetrieb sollte der "TCP/IP Speicher" in der C-Control Konfiguration auf ca. 16kb gesetzt werden.
- Alle GET-Requests des Webservers, die keine CGI Variablen in der URL übergeben, werden in einer Warteschlange serialisiert. Dies geschieht, damit immer nur RAM zum Senden einer Webseite gleichzeitig angefordert wird.
- Da die SD-Card in der C-Control Pro über SPI angeschlossen ist, und nicht wie bei einem PC im 4-Bit Parallelmodus, sind nur geringere Übertragungsraten zu realisieren. In Tests erreicht man mit wget.exe ein mittlere Übertragungsrate zwischen 140-150 kbyte/Sek. Daher sollten z.B. Bilder und andere Resourcen nicht wesentlich über 100kb hinausgehen, da sonst die Webseite nur langsam aufgebaut wird.
- Der Webserver unterstützt das "If-Modified-Since" Caching Protokoll der aktuellen Webbrowser. Daher sollte das Caching im Webbrowser eingeschaltet sein, sowie Uhrzeit und Datum der Dateien auf der SD-Card nicht in der Zukunft liegen.
- Möchte man nur CGI Variablen in der URL bei einem Javascript GET Request übergeben, ohne das man eine Antwort vom Server benötigt, sollte die URL "setvars.js" genutzt werden. Dieser Request übernimmt nur die Variablenwerte und erzeugt keine Antwort.
- Fordert man in Javascript nur Variablenwerte im JSON Format an, so erzeugt die URL "getvars.js" nur die JSON Ausgabe, mit dem Vorteil, das auf die SD-Card nicht zugegriffen werden muss.
- Es ist auf jeden Fall empfehlenswert sich die vorhandenen Demoprogramme für den Webserver anzusehen.
- Bei WEB_StartServer die Flags WEB_CACHE_HTML und WEB_CACHE_TEXT nur angeben, wenn man sich sicher ist, das HTML bzw. Text Webseiten wirklich gecached werden sollen!

5.27.2 WEB_GetRequest

Ethernet Funktionen

Syntax

```
byte WEB_GetRequest(void);

Sub WEB_GetRequest() As Byte
```

Beschreibung

Fragt den Webserver ab, ob ein HTTP Request zum Ausliefern einer Webseite gestellt wurde. Ein Wert von Null besagt, das kein Request vorliegt. Nach einem gültigen Request, sollte man die dynamischen Variablen auswerten und eventuell neu Werte setzen.

Rückgabewert

Request Parameter (0 = nichts empfangen)

5.27.3 WEB_GetFileHash

Ethernet Funktionen

Syntax

```
word WEB_GetFileHash(byte request);

Sub WEB_GetFileHash(request As Byte) As Word
```

Beschreibung

Gibt den 16 Bit CRC hash des Dateinamens zurück. Der request Parameter muss identisch mit dem Wert sein, den man von WEB_GetRequest() erhalten hat.

Parameter

request Request Parameter

Rückgabewert

16 Bit CRC hash des Dateinamens (8.3)

5.27.4 WEB_ReleaseRequest

Ethernet Funktionen

Syntax

```
void WEB_ReleaseRequest(byte request);

Sub WEB_ReleaseRequest(request As Byte)
```

Beschreibung

Signalisiert dem Webserver, dass die übergebenen URL Variablen ausgewertet wurden, und jetzt der Webserver die angeforderte Webseite über TCP/IP ausliefert. Der request Parameter muss identisch mit dem Wert sein, den man von WEB_Getrequest() erhalten hat.

Parameter

request Request Parameter

5.27.5 WEB_SetDynVar

Ethernet Funktionen

Syntax

```
void WEB_SetDynVar(word indx, ptr var_addr[], byte type, byte flags,
byte len);
```

```
Sub WEB_SetDynVar(indx As Word, var_addr As Pointer, type As Byte, flags
 As Byte, len As Byte)
```

Beschreibung

Definiert eine dynamische Variable über ihren index, Adresse und Variablentyp. Der len Parameter ist für Stringvariablen wichtig, andere Typen ignorieren diesen Parameter. Es können mehrere flags gleichzeitig angegeben werden, indem die Werte oderiert werden.

➔ Die Anzahl der benutzten dynamischen Variablen muss mit der Definition im WEB_BUF Makro und mit dynvar_cnt in WEB_StartServer korrespondieren.

Parameter

indx	Index der Variablen
var_addr	Adresse der Variablen
type	Variablentyp
flags	Eigenschaften der Variablen
len	Länge (0 bis 255) nur für Strings

Typdefinitionen

Definition	Bedeutung
DYN_BYTE	8-Bit ohne Vorzeichen
DYN_CHAR	8-Bit mit Vorzeichen
DYN_INT	16-Bit mit Vorzeichen
DYN_INTEGER	16-Bit mit Vorzeichen
DYN_WORD	16-Bit ohne Vorzeichen
DYN_UINTEGER	16-Bit ohne Vorzeichen
DYN_LONG	32-Bit mit Vorzeichen
DYN_DWORD	32-Bit ohne Vorzeichen
DYN_ULONG	32-Bit ohne Vorzeichen
DYN_STR	Character Array
DYN_FLOAT	Gleitkomma
DYN_SINGLE	Gleitkomma

Flagsdefinitionen

Definition	Bedeutung
DYN_CGIVAR	Darf über URL verändert werden
DYN_JSONVAR	Variable in JSON Liste

5.27.6 WEB_StartServer

Ethernet Funktionen

Syntax

```
void WEB_StartServer(word port, byte ramaddr[], word dynvar_cnt, word flags);

Sub WEB_StartServer(port As Word, ByRef ramaddr As Byte, dynvar_cnt As Word, flags As Word)
```

Beschreibung

Startet den Webserver auf TCP/IP Port port. Man definiert wieviele dynamischen Variable man nutzen möchte. Der Anwender muss eine **globale** Variable als Puffer bereitstellen. In diesem Puffer wird der Arbeitszustand des Webservers und Speicher für Kopiervorgänge abgelegt. Für die Größe des Puffers gibt es ein #define **WEB_BUF**. Möchte man ein byte Array definieren mit Platz für X dynamische Variablen, schreibt man `"byte buf[WEB_BUF(X)];"`. Es können mehrere flags gleichzeitig angegeben werden, indem die Werte oderiert werden.

➔ Der vom Benutzer zur Verfügung gestellte RAM Puffer muss während der gesamten Benutzung des Webservers reserviert bleiben. Da lokale Variablen nach Verlassen der Funktion freigegeben werden, ist es meistens sinnvoll den Puffer als globale Variable zu deklarieren.

Parameter

port	TCP/IP Port auf dem der Webserver antwortet
ramaddr	Adresse des Puffers
dynvar_cnt	Anzahl der dynamischen Variablen
flags	Eigenschaften des Webservers

Flagsdefinitionen

Definition	Bedeutung
WEB_CACHE_NORM	HTML und Text Seiten werden nicht gecached
WEB_CACHE_HTML	HTML Seiten werden gecached
WEB_CACHE_TEXT	Text Seiten werden gecached

5.27.7 WEB_StopServer

Ethernet Funktionen

Syntax

```
void WEB_StopServer(void);

Sub WEB_StopServer()
```

Beschreibung

Stoppt den Webserver.

Parameter

Keine

Kapitel 6

6 FAQ

6.1 Allgemein

1. 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.

2. Wo bestimmt man, ob das neue Projekt ein BASIC oder C Projekt ist?

Es gibt keine Unterschiede im Projekttyp. Die Quelltext Dateien in einem Projekt bestimmen welche Programmiersprache zum Einsatz kommt. Dateien mit der Endung *.cc laufen in einem CompactC Kontext, Dateien mit der Endung *.cbas werden mit BASIC übersetzt. Man kann in einem Project auch C und BASIC mischen.

3. Ich benutze ein andere LCD-Anzeige als die mitgelieferte mit aber demselben Controller. Die Cursorpositionierung funktioniert nicht richtig.

- Der Controller kann 4 Zeilen mit 32 Zeichen anzeigen. Die Zeilenanfänge liegen versetzt im Speicher nach folgenden Schema:

Wert von pos (Hex)	**Position im Display**
00-1f	0-31 in der 1. Zeile
40-5f	0-31 in der 2. Zeile
20-3f	0-31 in der 3. Zeile
60-6f	0-31 in der 4. Zeile

4. Wo sind die Demoprogramme?

- Wegen der geänderten Zugriffsrechte bei Windows Vista, wird bei der Installation auf eine bestehende Installation, das alte Verzeichnis Demos gelöscht. Die aktuellen Demoprogramme sind nun im Verzeichnis "C:\Dokumente und Einstellungen\Alle Benutzer\Gemeinsame Dokumente\C-Control Pro Demos" (XP und früher) bzw. "C:\Benutzer\Öffentlich\Öffentliche Dokumente\C-Control Pro Demos" (Vista und später) zu finden. Siehe auch das Kapitel Demo Programme.

5. Kann man die C-Control Pro mit IDE auch unter Linux programmieren?

- Es existiert keine native IDE für Linux, allerdings haben Anwender schon gute Erfahrungen damit gemacht die IDE unter Wine zu starten und das Modul im seriellen Modus (Mega) zu programmieren.

6. Kann man das C-Control Pro Modul auch mit anderen Compilern programmieren?

- Es existieren mehrere Entwicklungssysteme für die Atmel Mega oder AVR32 CPUs. Teilweise sind diese Compiler kommerziell oder frei. Ein Beispiel einer freien Entwicklungsumgebung ist der GNU C-Compiler. Mit Hilfe eines AVR Programmers kann man dann mit dem GNU C-Compiler geschriebene Programme übertragen. Ist aber einmal der Bootloader überschrieben, dann gibt es keinen Weg zurück mehr, dann ist die normale C-Control Pro Software nicht mehr weiter zu benutzen.

6.2 Mega

1. Wie schalte ich den Pull-Up Widerstand eines Porteingangs ein?

- Erst den Port mit PortDataDir() (bzw. PortDataDirBit()) auf Eingang schalten, dann eine "1" mit PortWrite() (bzw. PortWriteBit()) in den Port schreiben.

2. 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 der richtige Kommunikationsport eingestellt?
- Werden die Ports **M32**:B.4-B.7,A.6-A.7 bzw. **M128**:B.0-B.4,E.5 versehentlich in der Software benutzt? (siehe Pinzuordnung von M32 und M128). Sind die Jumper auf dem Application Board bei diesen Ports auch gesetzt?
- Ein Signal auf **M32**:PortD.2 **M128**:PortE.4 (SW1) beim Starten aktiviert den seriellen Bootloader.
- (nur Mega128) Ist vielleicht Port.G4 (LED2) beim Reset auf low? Siehe SPI Abschaltung im Kapitel "Firmware".
- Wird die SD card zusammen mit USB und dem Applicationboard genutzt, kommt es zu einer Kollision auf dem SPI Bus. Möchte man die SD-Card nutzen, so muss man die Jumper auf dem Applicationboard (**Mega128** PB.0 bis PB.4 und PE.5) abziehen und den seriellen Modus benutzen.

3. 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 von M32 und M128). Sind die Jumper auf dem Application Board bei diesen Ports auch gesetzt?

4. 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 **M32**:PortD.2 **M128**:PortE.4 (SW1) auch fest auf GND gelegt werden.

5. Die Hardware Applikation startet nicht von alleine (Autostart Verhalten).

- Ein Signal auf der SPI Schnittstelle beim Starten kann die USB Kommunikation aktivieren
- Ein Signal auf **M32**:PortD.2 **M128**:PortE.4 (SW1) beim Starten aktiviert den seriellen Bootloader.

6. Wieviel RAM Speicher haben meine Programme zur Verfügung?

- Auf dem Mega32 stehen 930 Bytes für eigene Programme zur Verfügung, auf dem Mega128 bleiben 2494 Bytes. Der Interpreter mit Debugger benötigt Puffer für die interruptgesteuerte Ein- und Ausgabe, und selber einen 256 Byte Datenstack. Außerdem werden mehrere interne Tabellen für Interruptverarbeitung und Multithreading gepflegt. Zusätzlich benötigen zahlreiche Bibliotheksfunktionen Variablen um sich Zustände zu merken.

7. Wo ist auf dem Mega128 Application Board die 2. serielle Schnittstelle?

- Siehe J4 im Kapitel Jumper Application Board M128.

8. Ich brauche keine USB Verbindung zum Application Board, wie kann ich die belegten Ports nutzen?

- Die USB Verbindung wird über die SPI Schnittstelle hergestellt. Die SPI Schnittstelle kann mit SPI_Disable() deaktiviert werden. Beim arbeiten mit dem Application Board dann nicht vergessen die Jumper abzuziehen, die die SPI mit dem Mega8 verbindet.

9. Wo finde ich die Versorgungsspannung für das Lochrasterfeld auf dem Application Board?

- Hält man das Application Board so, das die Anschlüsse nach oben zeigen, dann ist die linke Lochrasterfeldreihe mit GND verbunden und die rechte Lochrasterfeldreihe mit VCC. Dies ist gut zu erkennen, wenn man die Platine von unten betrachtet.

10. Ich brauche für meine Anwendung mehr freie Ports. Die meisten sind ja mit Funktionen belegt.

- Schaut man sich die Pinzuordnung von M32 und M128 an, so können die Ports genutzt werden, deren Funktionalität man nicht braucht. Bei Pins die beim Application Board mit Peripherie verbunden sind (SPI, RS232, LCD, Keyboard etc.) nicht vergessen die Jumper am Application Board abzuziehen. Ansonsten wird das Portverhalten beeinträchtigt.

6.3 AVR32Bit

1. Es existiert keine USB Verbindung zum Application Board.

- Ist der USB (usbser.sys) Treiber auf dem PC geladen? Oder erscheint vielleicht beim Einstecken des USB Steckers ein "unbekanntes Gerät" im Hardware Manager?
- Ist der richtige Kommunikationsport eingestellt?
- Bitte den USB Troubleshooting Leitfaden lesen!
- Im Auslieferungszustand ist der Autostart Jumper gesetzt. Bitte abziehen, da sonst keine Übertragung möglich ist.

2. Wie schalte ich den Pull-Up Widerstand eines Porteingangs ein?

- Siehe Port_Attribute()

3. Ich brauche für meine Anwendung mehr freie Ports. Die meisten sind ja mit Funktionen belegt.

- Schaut man sich die Pinzuordnung der AVR32Bit an, so können die Ports genutzt werden, deren Funktionalität man nicht braucht. Bei Pins die beim Application Board mit Peripherie verbunden sind (SPI, RS232, LCD, Keyboard etc.) nicht vergessen die Jumper am Application Board abzuziehen. Ansonsten wird das Portverhalten beeinträchtigt.

4. Ich kann das Modul nicht mehr zurücksetzen oder ein Übertragen des Interpreters nach einem Software Update funktioniert nicht mehr.

- Ein Aus- und Einschalten bringt das Modul wieder sicher in den Bootloader um ein Rücksetzen des Moduls zu erlauben. Siehe Firmware.

Sachverzeichnis

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- K -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- U -

- V -

- W -

- Z -