



## **C-Control Pro Mega Series**

© 2009 Conrad Electronic

# Inhalt

<b>Kapitel 1 Wichtige Hinweise</b>	<b>2</b>
1 Einleitung .....	2
2 Lesen dieser Anleitung .....	2
3 Handhabung .....	3
4 Bestimmungsgemäße Verwendung .....	3
5 Gewährleistung und Haftung .....	3
6 Service .....	4
7 Open Source .....	4
8 Historie .....	5
<b>Kapitel 2 Installation</b>	<b>11</b>
1 Applicationboard .....	11
2 Software .....	15
<b>Kapitel 3 Hardware</b>	<b>17</b>
1 Firmware .....	17
2 LCD Matrix .....	19
3 Mega32 .....	19
3.1 Modul .....	20
3.2 Application Board .....	23
3.3 Pinzuordnung .....	27
3.4 Jumper Application Board .....	28
3.5 Schaltpläne .....	30
4 Mega128 .....	36
4.1 Modul .....	37
4.2 Application Board .....	40
4.3 Pinzuordnung .....	45
4.4 Jumper Application Board .....	46
4.5 Schaltpläne .....	48
<b>Kapitel 4 IDE</b>	<b>55</b>
1 Projekte .....	56
1.1 Projekterstellung .....	56
1.2 Projekte Kompilieren .....	56
1.3 Projektverwaltung .....	57
1.4 Projektoptionen .....	59

1.5 Threadoptionen .....	60
1.6 Bibliotheksverwaltung .....	60
2 Editor .....	61
2.1 Editorfunktionen .....	64
2.2 Druckvorschau .....	65
2.3 Tastaturkürzel .....	66
2.4 Reguläre Ausdrücke .....	68
3 C-Control Hardware .....	69
3.1 Programm starten .....	69
3.2 Ausgaben .....	70
3.3 PIN Funktionen .....	70
3.4 Versionsüberprüfung .....	71
4 Debugger .....	71
4.1 Haltepunkte .....	72
4.2 Variablen Fenster .....	73
4.3 Array Fenster .....	75
5 Werkzeuge .....	76
6 Optionen .....	77
6.1 Editoreinstellungen .....	78
6.2 Syntaxhervorhebung .....	79
6.3 Compilervoreinstellung .....	81
6.4 IDE Einstellungen .....	82
7 Fenster .....	87
8 Hilfe .....	88

## **Kapitel 5 Compiler 90**

1 Allgemeine Features .....	90
1.1 externes RAM .....	90
1.2 Prozessor .....	90
1.3 Pragma Anweisungen .....	92
1.4 Map Datei .....	92
2 CompactC .....	93
2.1 Programm .....	93
2.2 Anweisungen .....	94
2.3 Datentypen .....	96
2.4 Variablen .....	97
2.5 Operatoren .....	101
2.6 Kontrollstrukturen .....	103
2.7 Funktionen .....	109
2.8 Tabellen .....	112

3 BASIC .....	115
3.1 Programm .....	115
3.2 Anweisungen .....	115
3.3 Datentypen .....	117
3.4 Variablen .....	118
3.5 Operatoren .....	122
3.6 Kontrollstrukturen .....	124
3.7 Funktionen .....	130
3.8 Tabellen .....	132
4 Assembler .....	134
4.1 Ein Beispiel .....	134
4.2 Datenzugriff .....	136
4.3 Leitfaden .....	138
5 ASCII Tabelle .....	138

## Kapitel 6 Bibliotheken 145

1 Interne Funktionen .....	145
2 Allgemein .....	145
2.1 AbsDelay .....	145
2.2 Sleep .....	146
3 Analog-Comparator .....	146
3.1 AComp .....	147
3.2 AComp Beispiel .....	147
4 Analog-Digital-Wandler .....	148
4.1 ADC_Disable .....	149
4.2 ADC_Read .....	149
4.3 ADC_ReadInt .....	150
4.4 ADC_Set .....	150
4.5 ADC_SetInt .....	151
4.6 ADC_StartInt .....	152
5 Clock .....	152
5.1 Clock_GetVal .....	152
5.2 Clock_SetDate .....	153
5.3 Clock_SetTime .....	153
6 DCF 77 .....	154
6.1 DCF_FRAME .....	156
6.2 DCF_INIT .....	156
6.3 DCF_PULS .....	157
6.4 DCF_START .....	157
6.5 DCF_SYNC .....	158

7	Debug .....	158
7.1	Msg_WriteChar .....	158
7.2	Msg_WriteFloat .....	159
7.3	Msg_WriteHex .....	159
7.4	Msg_WriteInt .....	159
7.5	Msg_WriteText .....	160
7.6	Msg_WriteWord .....	160
8	Direct Access .....	161
8.1	DirAcc_Read .....	161
8.2	DirAcc_Write .....	161
9	EEPROM .....	162
9.1	EEPROM_Read .....	162
9.2	EEPROM_ReadWord .....	162
9.3	EEPROM_ReadFloat .....	163
9.4	EEPROM_Write .....	163
9.5	EEPROM_WriteWord .....	164
9.6	EEPROM_WriteFloat .....	164
10	I2C .....	165
10.1	I2C_Init .....	165
10.2	I2C_Read_ACK .....	165
10.3	I2C_Read_NACK .....	166
10.4	I2C_Start .....	166
10.5	I2C_Status .....	166
10.6	I2C_Stop .....	167
10.7	I2C_Write .....	167
10.8	I2C Status Tabelle .....	168
10.9	I2C Beispiel .....	168
11	Interrupt .....	169
11.1	Ext_IntEnable .....	170
11.2	Ext_IntDisable .....	171
11.3	Irq_GetCount .....	171
11.4	Irq_SetVect .....	172
11.5	IRQ Beispiel .....	172
12	Keyboard .....	173
12.1	Key_Init .....	173
12.2	Key_Scan .....	173
12.3	Key_TranslateKey .....	174
13	LCD .....	174
13.1	LCD_ClearLCD .....	174

13.2	LCD_CursorOff .....	175
13.3	LCD_CursorOn .....	175
13.4	LCD_CursorPos .....	175
13.5	LCD_Init .....	176
13.6	LCD_Locate .....	176
13.7	LCD_SubInit .....	177
13.8	LCD_TestBusy .....	177
13.9	LCD_WriteChar .....	178
13.10	LCD_WriteCTRRegister .....	178
13.11	LCD_WriteDataRegister .....	178
13.12	LCD_WriteFloat .....	179
13.13	LCD_WriteRegister .....	179
13.14	LCD_WriteText .....	180
13.15	LCD_WriteWord .....	180
14	Mathematik .....	181
14.1	acos .....	181
14.2	asin .....	181
14.3	atan .....	182
14.4	ceil .....	182
14.5	cos .....	183
14.6	exp .....	183
14.7	fabs .....	183
14.8	floor .....	184
14.9	ldexp .....	184
14.10	ln .....	185
14.11	log .....	185
14.12	pow .....	186
14.13	round .....	186
14.14	sin .....	186
14.15	sqrt .....	187
14.16	tan .....	187
15	OneWire .....	188
15.1	Onewire_Read .....	188
15.2	Onewire_Reset .....	188
15.3	Onewire_Write .....	189
15.4	Onewire Beispiel .....	190
16	Port .....	191
16.1	Port_DataDir .....	192
16.2	Port_DataDirBit .....	193

16.3	Port_Read .....	194
16.4	Port_ReadBit .....	194
16.5	Port_Toggle .....	195
16.6	Port_ToggleBit .....	196
16.7	Port_Write .....	197
16.8	Port_WriteBit .....	198
16.9	Port Beispiel .....	199
17	RC5 .....	199
17.1	RC5_Init .....	202
17.2	RC5_Read .....	203
17.3	RC5_Write .....	204
18	RS232 .....	204
18.1	Divider .....	204
18.2	Serial_Disable .....	205
18.3	Serial_Init .....	206
18.4	Serial_Init_IRQ .....	207
18.5	Serial_IRQ_Info .....	208
18.6	Serial_Read .....	209
18.7	Serial_ReadExt .....	209
18.8	Serial_Write .....	210
18.9	Serial_WriteText .....	210
18.10	Serial Beispiel .....	211
18.11	Serial Beispiel (IRQ) .....	211
19	Servo .....	211
19.1	Servo_Init .....	212
19.2	Servo_Set .....	213
19.3	Servo Beispiel .....	214
20	SPI .....	214
20.1	SPI_Disable .....	214
20.2	SPI_Enable .....	215
20.3	SPI_Read .....	216
20.4	SPI_ReadBuf .....	216
20.5	SPI_Write .....	216
20.6	SPI_WriteBuf .....	217
21	Strings .....	217
21.1	Str_Comp .....	217
21.2	Str_Copy .....	218
21.3	Str_Fill .....	218
21.4	Str_Isalnum .....	219

21.5	Str_Isalpha .....	219
21.6	Str_Len .....	220
21.7	Str_Printf .....	220
21.8	Str_ReadFloat .....	221
21.9	Str_ReadInt .....	221
21.10	Str_ReadNum .....	222
21.11	Str_Substr .....	222
21.12	Str_WriteFloat .....	223
21.13	Str_WriteInt .....	223
21.14	Str_WriteWord .....	224
21.15	Str_Printf Beispiel .....	225
22	Threads .....	225
22.1	Thread_Cycles .....	227
22.2	Thread_Delay .....	227
22.3	Thread_Info .....	228
22.4	Thread_Kill .....	228
22.5	Thread_Lock .....	229
22.6	Thread_MemFree .....	229
22.7	Thread_Resume .....	230
22.8	Thread_Signal .....	230
22.9	Thread_Start .....	230
22.10	Thread_Wait .....	231
22.11	Thread Beispiel .....	231
22.12	Thread Beispiel 2 .....	232
23	Timer .....	233
23.1	Ereigniszähler .....	233
23.2	Frequenzerzeugung .....	233
23.3	Frequenzmessung .....	234
23.4	Pulsweitenmodulation .....	235
23.5	Puls & Periodenmessung .....	235
23.6	Timerfunktionen .....	236
23.7	Timer_Disable .....	237
23.8	Timer_T0CNT .....	237
23.9	Timer_T0FRQ .....	238
23.10	Timer_T0GetCNT .....	239
23.11	Timer_T0PW .....	239
23.12	Timer_T0PWM .....	239
23.13	Timer_T0Start .....	240
23.14	Timer_T0Stop .....	241



23.15	Timer_T0Time .....	241
23.16	Timer_T1CNT .....	242
23.17	Timer_T1CNT_Int .....	242
23.18	Timer_T1FRQ .....	243
23.19	Timer_T1FRQX .....	244
23.20	Timer_T1GetCNT .....	244
23.21	Timer_T1GetPM .....	244
23.22	Timer_T1PWA .....	245
23.23	Timer_T1PM .....	245
23.24	Timer_T1PWB .....	246
23.25	Timer_T1PWM .....	246
23.26	Timer_T1PWMX .....	247
23.27	Timer_T1PWMY .....	247
23.28	Timer_T1Start .....	248
23.29	Timer_T1Stop .....	248
23.30	Timer_T1Time .....	249
23.31	Timer_T3CNT .....	249
23.32	Timer_T3CNT_Int .....	250
23.33	Timer_T3FRQ .....	250
23.34	Timer_T3FRQX .....	251
23.35	Timer_T3GetCNT .....	251
23.36	Timer_T3GetPM .....	252
23.37	Timer_T3PWA .....	252
23.38	Timer_T3PM .....	253
23.39	Timer_T3PWB .....	253
23.40	Timer_T3PWM .....	254
23.41	Timer_T3PWMX .....	254
23.42	Timer_T3PWMY .....	255
23.43	Timer_T3Start .....	255
23.44	Timer_T3Stop .....	256
23.45	Timer_T3Time .....	256
23.46	Timer_TickCount .....	257

## Kapitel 7 FAQ

**259**

# **Kapitel**



# 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 Mega 32, bzw. dem Atmel Mega 128 RISC Mikrocontroller. Dieser Mikrocontroller wird in sehr vielen Geräten in großen Stückzahlen eingesetzt. Von der Unterhaltungselektronik, über Haushaltsmaschinen bis hin zu verschiedenen Einsatzmöglichkeiten in der Industrie. Dort übernimmt der Controller wichtige Steuerungsaufgaben. C-Control Pro bietet Ihnen diese hochmoderne Technologie zur Lösung Ihrer Steuerungsprobleme. Sie können analoge Meßwerte und Schalterstellungen erfassen und abhängig von diesen Eingangsbedingungen entsprechende Schaltsignale ausgeben, z.B. für Relais oder Stellmotoren. In Verbindung mit einer DCF77-Funkantenne kann C-Control Pro die atomgenaue Uhrzeit empfangen und präzise Schaltuhrfunktionen übernehmen. Verschiedene Hardware-Schnittstellen und Bussysteme erlauben die Vernetzung von C-Control Pro mit Sensoren, Aktoren und anderen Steuerungssystemen. Wir wollen unsere Technologie einem breiten Anwenderkreis zur Verfügung stellen. Aus unserer bisherigen Arbeit im C-Control-Service wissen wir, daß sich auch lernbereite Kunden ohne jegliche Elektronik- und Programmiererfahrungen für C-Control interessieren. Sollten Sie zu dieser Anwendergruppe gehören, gestatten Sie uns an dieser Stelle bitte einen Hinweis:

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

## 1.2 Lesen dieser Anleitung

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

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

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

## 1.3 Handhabung

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

## 1.4 Bestimmungsgemäße Verwendung

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

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

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

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

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

## 1.5 Gewährleistung und Haftung

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

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

Der Einsatz der C-Control Pro Unit in Systemen, die direkt oder indirekt medizinischen, gesundheits- oder lebenssichernden Zwecken dienen, ist nicht zulässig.

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

## 1.6 Service

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

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

Fax-Nr.:      09604 / 40-8848  
Mail:          [webmaster@c-control.de](mailto: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](http://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	<a href="http://www.antlr.org">http://www.antlr.org</a>
Inno Setup 5.2.3	<a href="http://www.jrsoftware.org">http://www.jrsoftware.org</a>
GPP (Generic Preprocessor)	<a href="http://www.nothingisreal.com/gpp">http://www.nothingisreal.com/gpp</a>
avra-1.2.3a Assembler	<a href="http://avra.sourceforge.net/">http://avra.sourceforge.net/</a>

Gemäß den Bestimmungen der "LESSER GPL" ([www.gnu.org/copyleft/lesser](http://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 Historie

**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
- Fehler beim Verkleinern des Ausgabenfensters 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 Installation

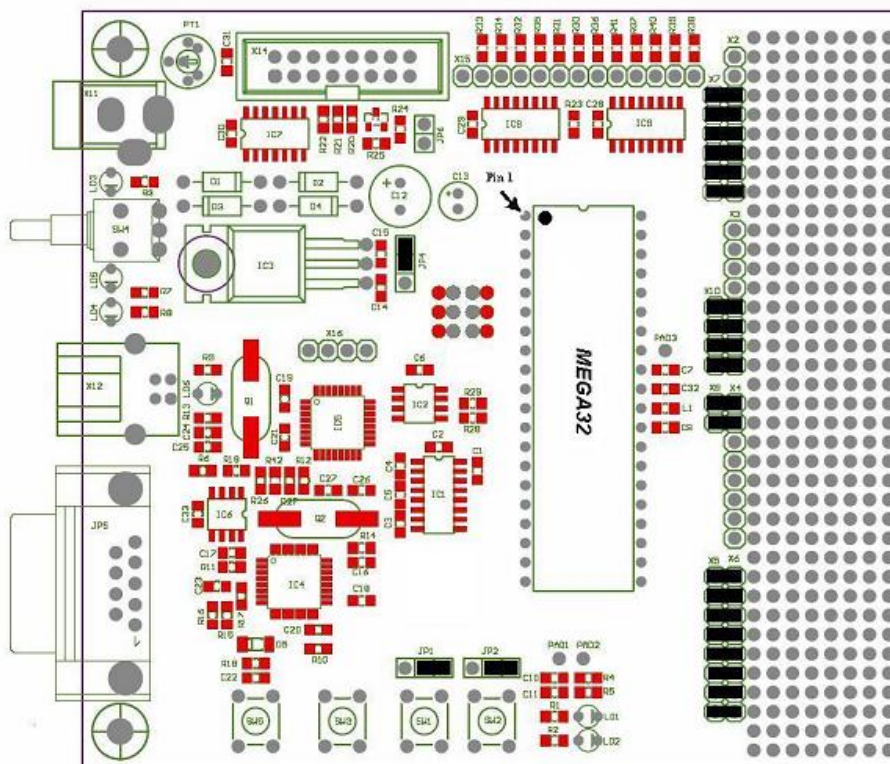
In diesem Abschnitt wird die Installation von Hard- und Software erläutert.

### 2.1 Applicationboard

#### **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. 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.

## Einbaurichtung Modul MEGA32



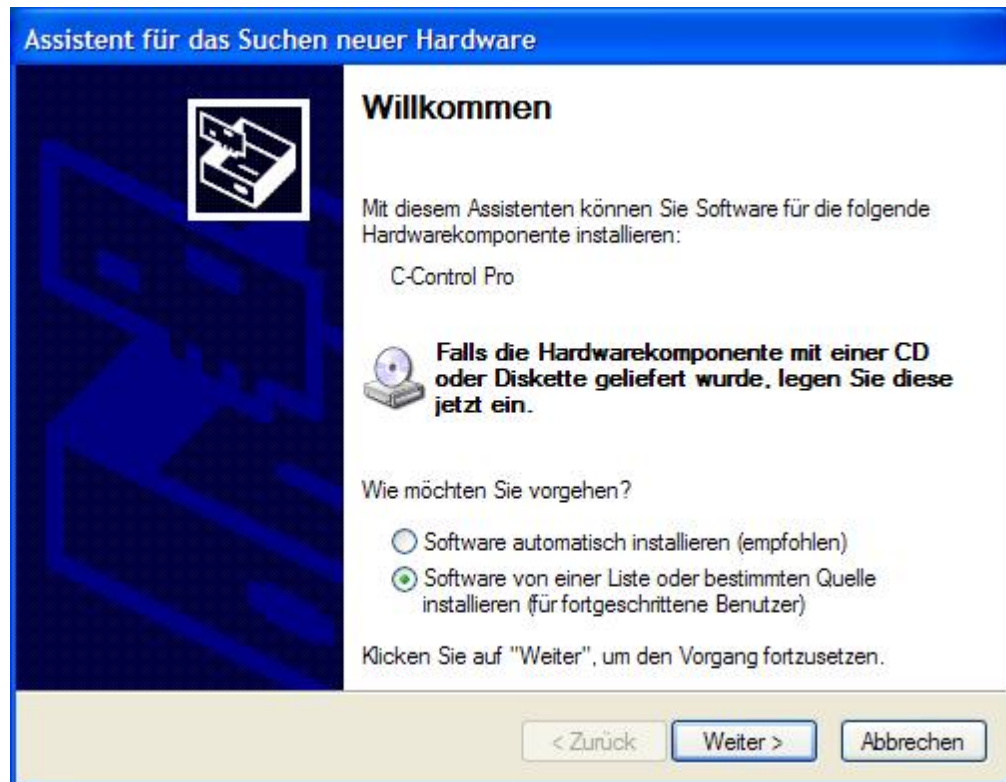
Das Modul Mega128 hat die Steckverbinder so angeordnet, daß ein falscher Einbau des Moduls nicht möglich ist. Der Ausbau erfolgt durch vorsichtiges Heraushebeln des Moduls mit einem geeigneten Werkzeug aus der Fassung. Um ein Verbiegen der Anschlüsse zu vermeiden sollte das Hebeln von mehreren Seiten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

## Installation des USB Treibers

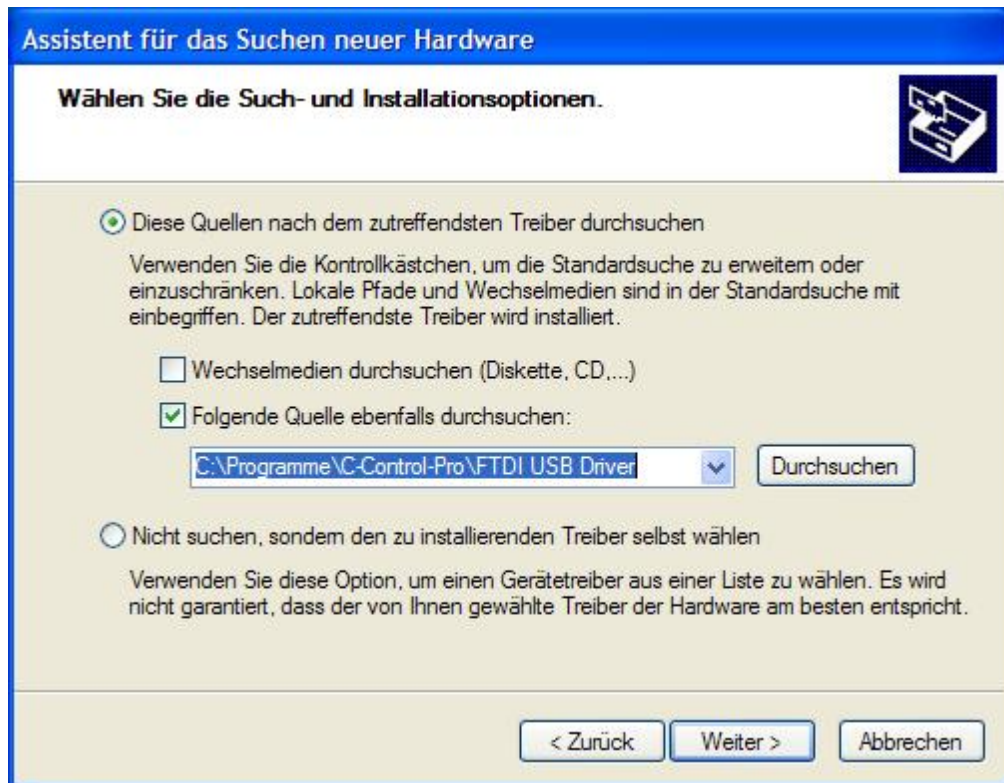
Bitte verbinden Sie das Application Board mit einem Netzgerät. Sie können hierzu ein Standard Steckernetzteil mit 9V/250mA verwenden. Die Polung ist beliebig, sie wird durch Dioden immer richtig umgesetzt. Je nach zusätzlicher Beschaltung kann es später notwendig sein ein Netzteil mit höherer Leistung 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.

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

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



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



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



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

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

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

➔ Der Treiber von FTDI unterstützt 32 Bit und 64 Bit Windows Systeme. Die spezifischen Treiber sind in den Unterverzeichnissen "FTDI USB Driver\i386" und "FTDI USB Driver\amd64".

### Serieller Anschluß

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

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

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

## 2.2 Software

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

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

➔ Um die Konsistenz der Demo Programme zu erhalten, wird bei der Installation auf eine bestehende Installation das alte Verzeichnis Demoprogramme gelöscht, und durch ein neues ersetzt. Deshalb bitte eigene Programme außerhalb des C-Control-Pro Verzeichnisses erstellen.

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.



# **Kapitel**



## 3 Hardware

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

### 3.1 Firmware

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

- *Bootloader*
- *Interpreter*

#### Bootloader

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

➔ Soll statt dem USB Interface eine serielle Verbindung von der IDE zum C-Control Pro Modul aufgebaut werden, so ist beim Einschalten des Moduls der Taster SW1 (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.

## 3.2 LCD Matrix

Die vollständigen Datenblätter finden sich auf der CD-ROM im Verzeichnis "Datasheets".

### 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)			0	@	P	`	F		ー	タ	ミ	α	p
0001	(2)	!	1	A	Q	a	4	。	ア	チ	4	ä	q	
0010	(3)	"	2	B	R	b	r	「	イ	ウ	×	β	θ	
0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	ω	
0100	(5)	\$	4	D	T	d	t	、	エ	ト	7	μ	Ω	
0101	(6)	%	5	E	U	e	u	・	オ	ナ	1	ε	Ü	
0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ	
0111	(8)	'	7	G	W	g	w	7	キ	ヌ	ラ	g	π	
1000	(1)	(	8	H	X	h	x	ィ	ク	ネ	リ	7	Σ	
1001	(2)	)	9	I	Y	i	y	6	7	ル	7	7	7	
1010	(3)	*	:	J	Z	j	z	エ	コ	ハ	レ	j	7	
1011	(4)	+	;	K	[	k	(	オ	サ	ヒ	ロ	*	7	
1100	(5)	,	<	L	¥	l	l	7	シ	フ	ワ	φ	7	
1101	(6)	ー	=	M	]	m	)	ユ	ズ	ハ	ン	モ	÷	
1110	(7)	。	>	N	^	n	→	ヨ	セ	ホ	7	7		
1111	(8)	/	?	O	_	o	←	ッ	リ	マ	"	Ö		

## 3.3 Mega32

### 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**

### 3.3.1 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.

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:

$$u = x \cdot \text{Referenzspannung} / 1024$$

## 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 bytewise 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).

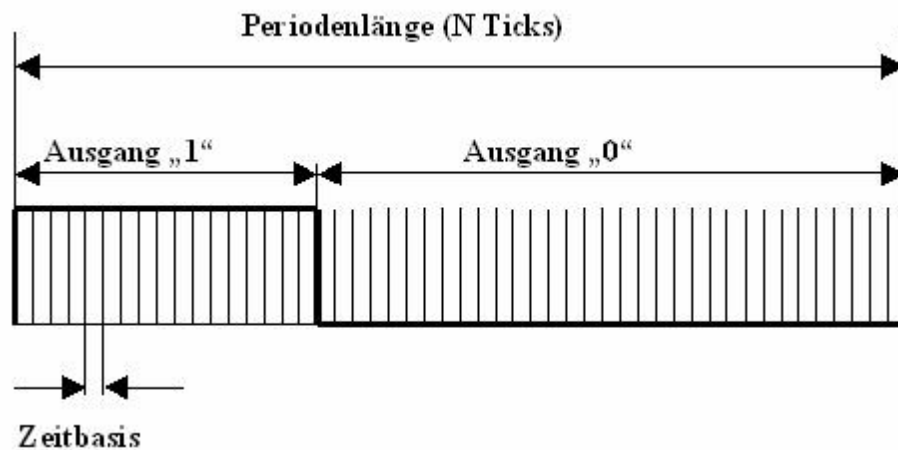
➔ Den [maximal zulässigen Laststrom](#) für einen einzelnen Port und für alle Ports in der Summe beachten. Eine Überschreitung der Maximalwerte kann zur Zerstörung des C-Control Pro Moduls führen. Nach dem Reset ist zunächst jeder Digitalport als Eingangsport konfiguriert. Über bestimmte Befehle kann 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 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 pulswidenmodulierten Digital-Analogwandlung werden Zeitbasis und Periodenlänge einmalig eingestellt, und dann nur der Ausgabewert verändert. Die PLM-Ports sind nach ihren elektrischen Eigenschaften Digitalports. Die technischen Randbedingungen für Digitalports beachten (max. Strom).

## Technische Daten Modul

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

Alle Spannungsangaben beziehen sich auf Gleichspannung (DC).

Umgebungsbedingungen	
Bereich der zulässigen Umgebungstemperatur	0°C ... 70°C
Bereich der zulässigen relativen Umgebungsluftfeuchte	20% ... 60%
Versorgungsspannung	
Bereich der zulässigen Versorgungsspannung	4,5V ... 5,5V
Stromaufnahme des Moduls ohne externe Lasten	ca. 20mA

<b>Takt</b>	
Taktfrequenz (Quarzoszillator)	14,7456MHz
<b>Mechanik</b>	
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

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

### 3.3.2 Application Board

#### USB

Das Application Board 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. Für die USB-Kommunikation wird die SPI-Schnittstelle des Mega32 verwendet (PortB.4 bis PortB.7, PortA.6, PortA.7) und müssen über die entsprechenden Jumper verbunden sein.

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

#### Ein- Ausschalter

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



## Leuchtdioden

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

## Taster

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

➔ Ein Drücken von SW1 beim Einschalten des Boards aktiviert den [seriellen Bootloadermodus](#).

## LCD

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

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

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.

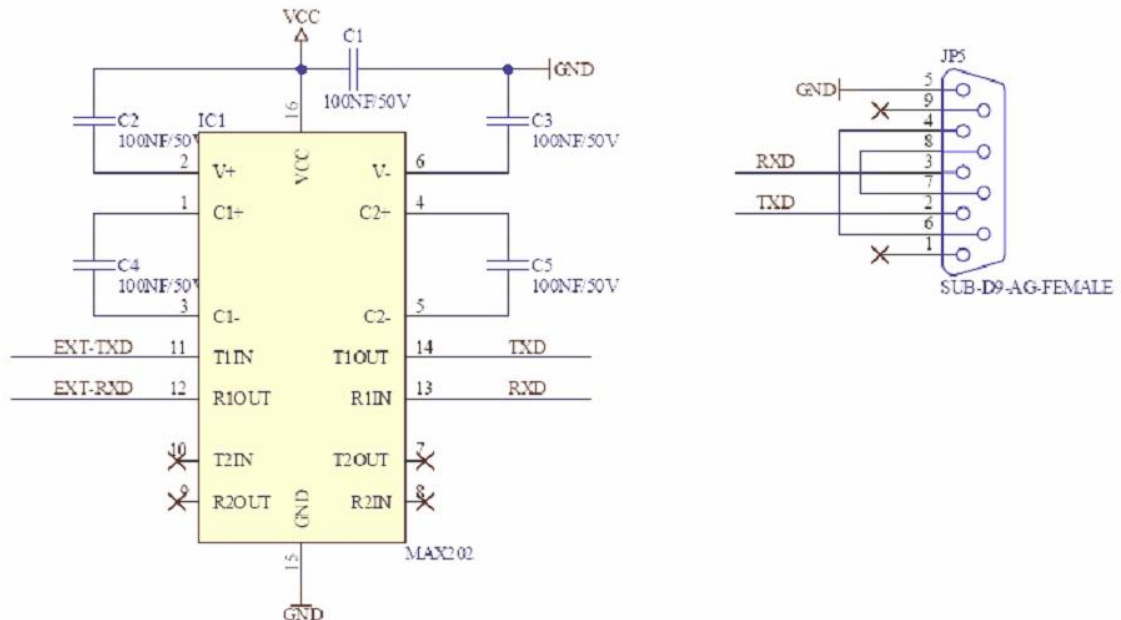
➔ Bitte den [maximal entnehmbaren Strom](#) beachten. Eine Überschreitung kann zur Zerstörung führen! Wegen der relativ hohen Stromaufnahme des Application Boards im Bereich von 125 mA ist sie für den Einsatz in dauerhaft batteriebetriebenen Geräten nicht zu empfehlen. Bitte den Hinweis zu kurzzeitigen 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 Verbindungskabel verbinden, deren Anschlußbelegung bekannt ist.

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

## Testschnittstellen

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

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

## Technische Daten Application Board

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

Alle Spannungsangaben beziehen sich auf Gleichspannung (DC).

<b>Mechanik</b>	
äußere Abmessungen ca.	160 mm x 100 mm
Pinrastr Verdrahtungsfeld	2,54 mm
<b>Umgebungsbedingungen</b>	
Bereich der zulässigen Umgebungstemperatur	0°C ... 70°C
Bereich der zulässigen relativen Umgebungsluftfeuchte	20% ... 60%

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

### 3.3.3 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	5	ADC6	TX_REQ	ADC6 Eingang; USB-Kommunikation
35	PA5	PortA.5	5	ADC5	KEY_EN	ADC5 Eingang; LCD/Tastatur Interface
36	PA4	PortA.4	4	ADC4	LCD_EN	ADC4 Eingang; LCD/Tastatur Interface
37	PA3	PortA.3	3	ADC3	EXT_SCK	ADC3 Eingang; LCD/Tastatur Interface
38	PA2	PortA.2	2	ADC2	EXT_DATA	ADC2 Eingang; LCD/Tastatur Interface
39	PA1	PortA.1	1	ADC1		ADC1 Eingang
40	PA0	PortA.0	0	ADC0		ADC0 Eingang

### 3.3.4 Jumper Application Board

#### Jumper

Durch Jumper können bestimmte Optionen ausgewählt werden. Das betrifft einige Ports, welche mit speziellen Funktionen belegt sind (siehe Tabelle der Pinzuordnung von [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 andere Funktionen zur Verfügung. Neben den Jumpern für die Ports gibt es noch zusätzliche Jumper, welche nachfolgend beschrieben werden.

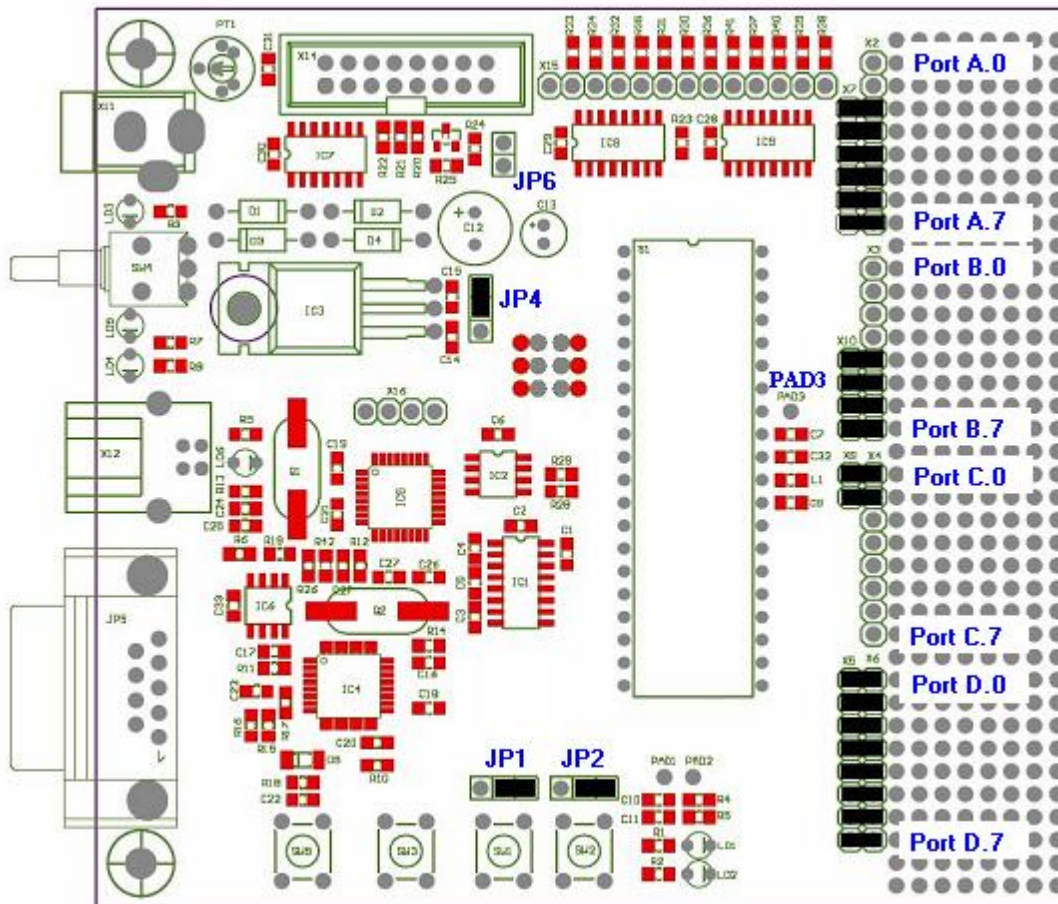
#### Ports A bis D

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 Auslieferungszustand

## JP4

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

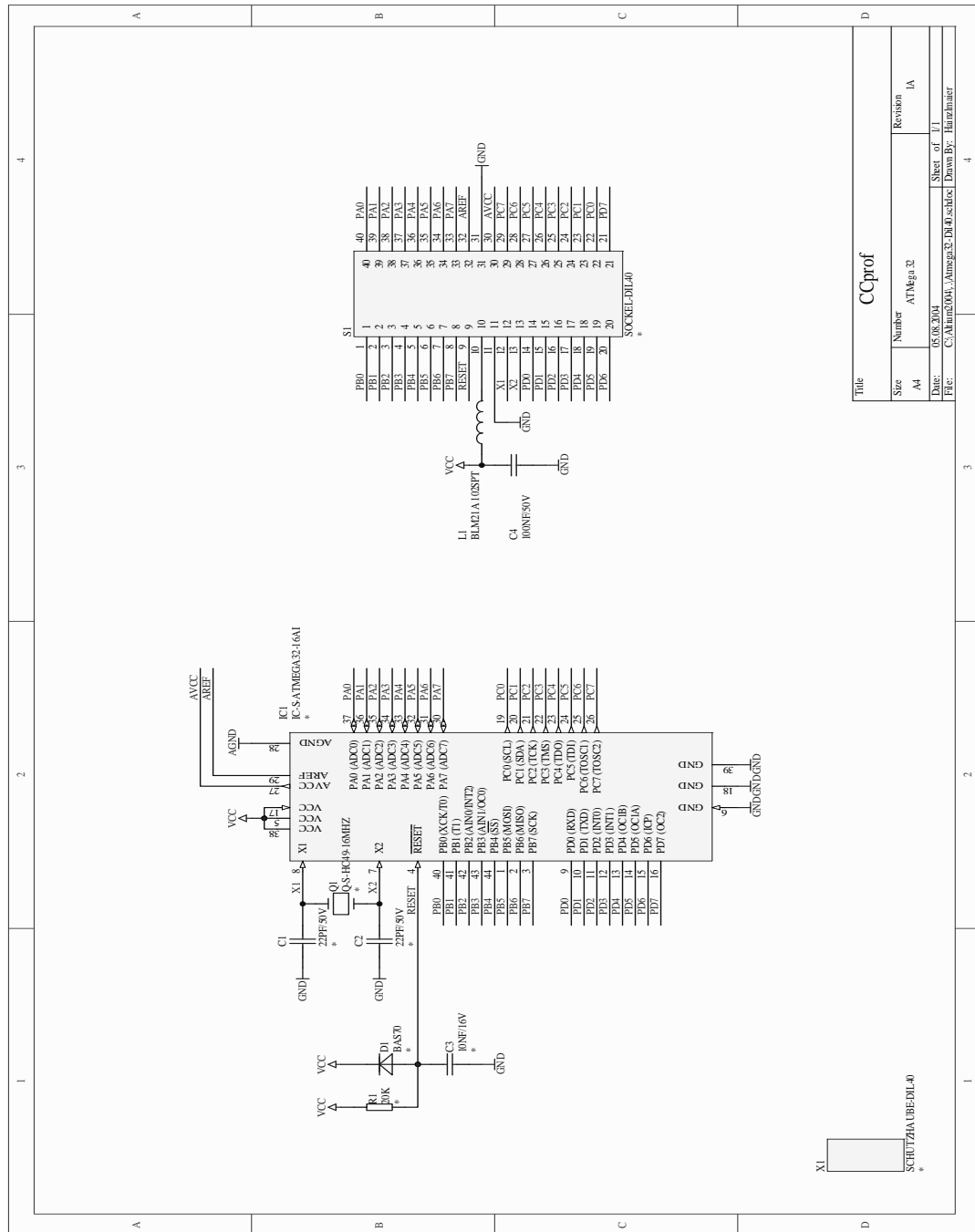
### PAD3

PAD3 (rechts neben dem Modul, unter der [blauen](#) Beschriftung) wird als ADC\_VREF\_EXT für die Funktionen [ADC\\_Set](#) und [ADC\\_SetInt](#) benötigt.

### 3.3.5 Schaltpläne

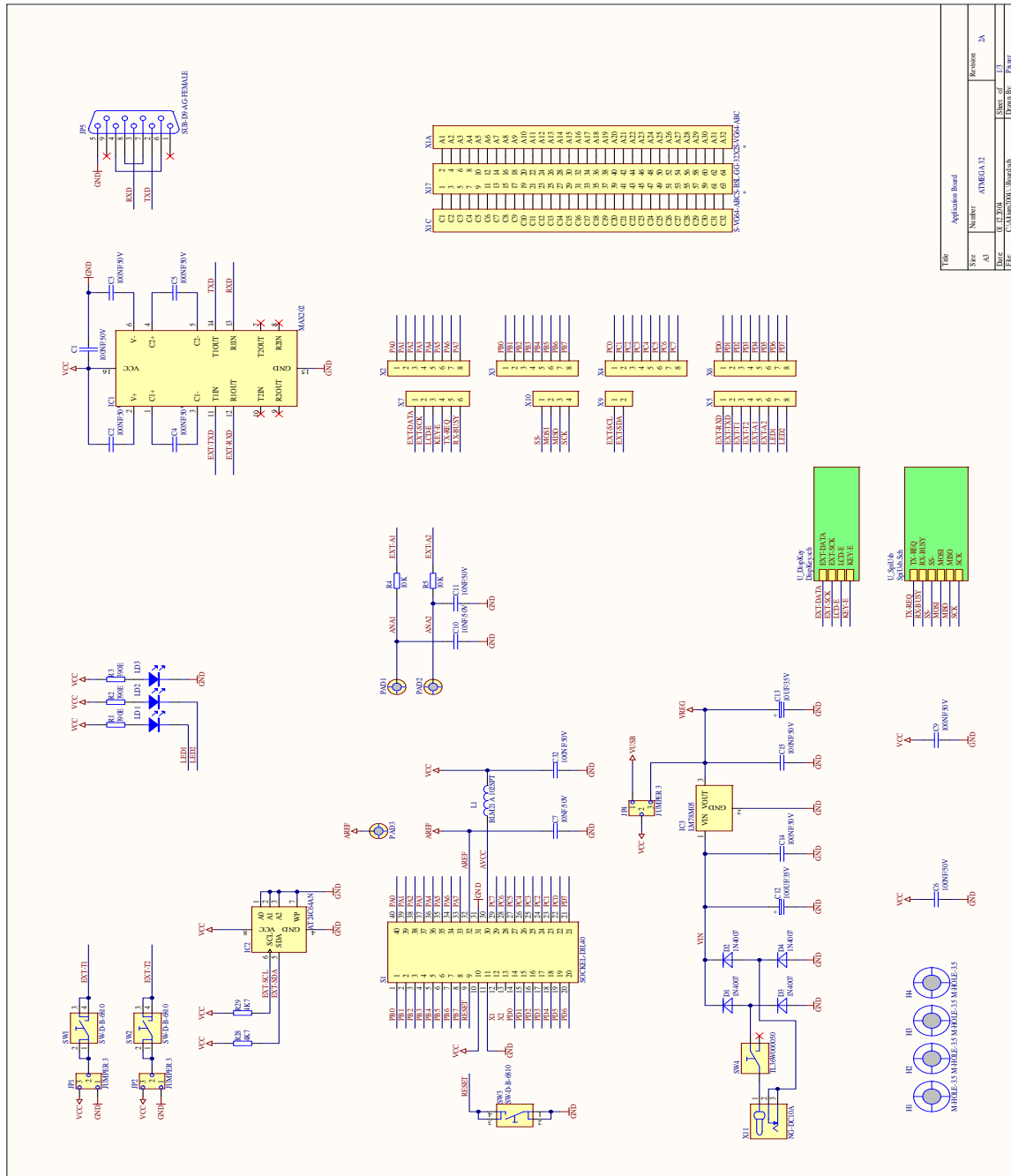
Die Schaltpläne sind zusätzlich als pdf Dateien auf der Installations CD.

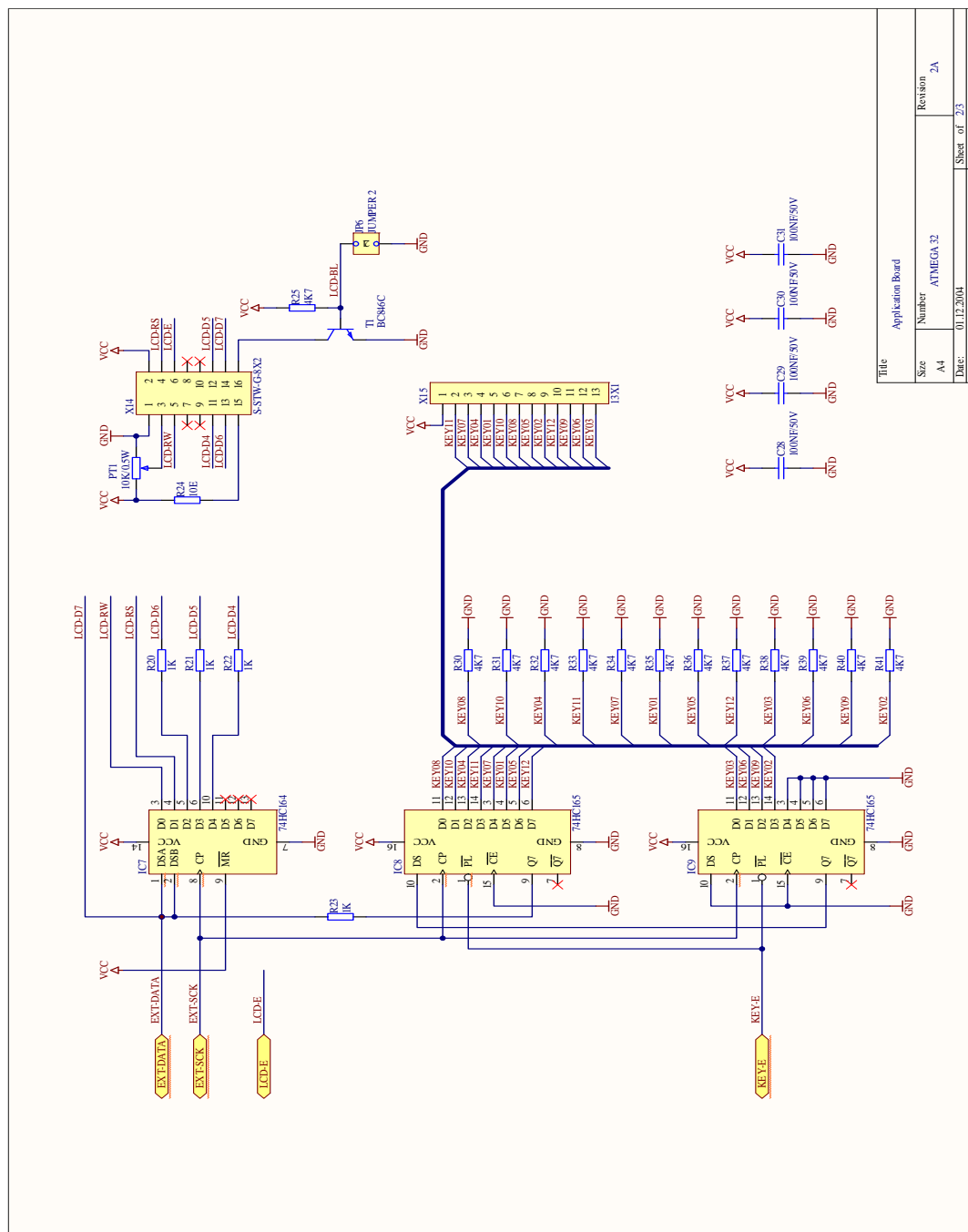
## 3.3.5.1 Mega 32 Modul

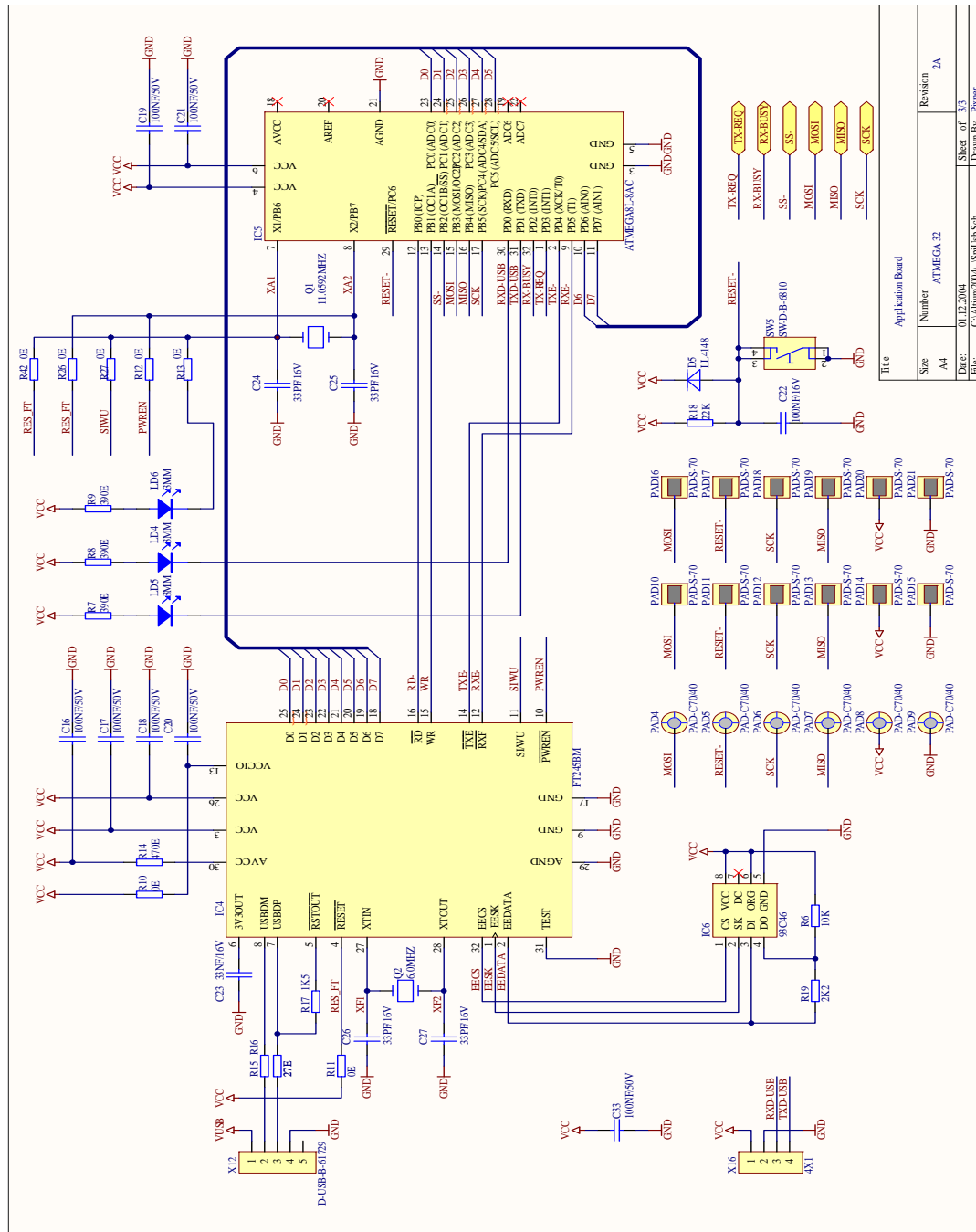




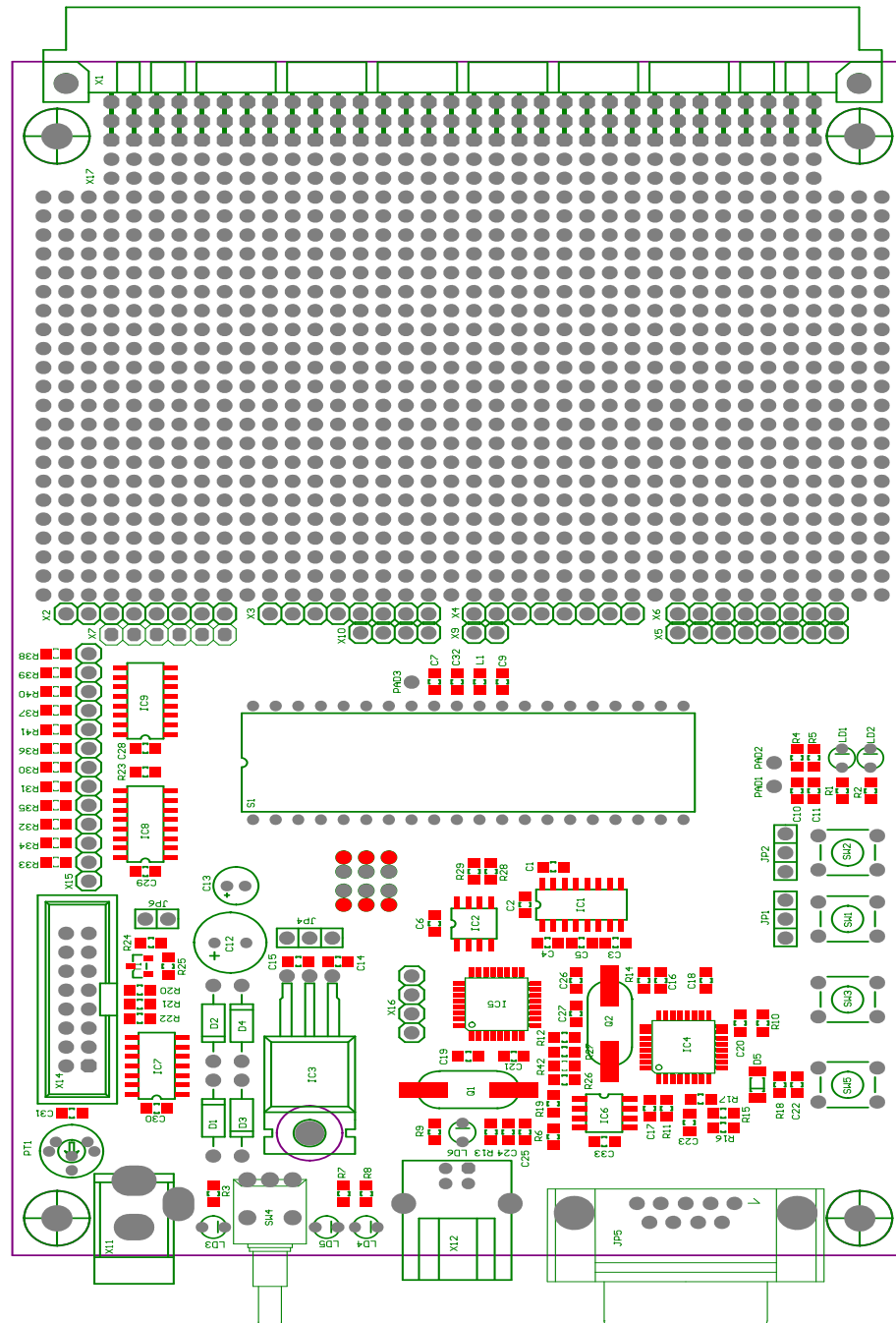
### 3.3.5.2 Applicationboard







## 3.3.5.3 Bestückungsplan



## 3.4 Mega128

### 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**

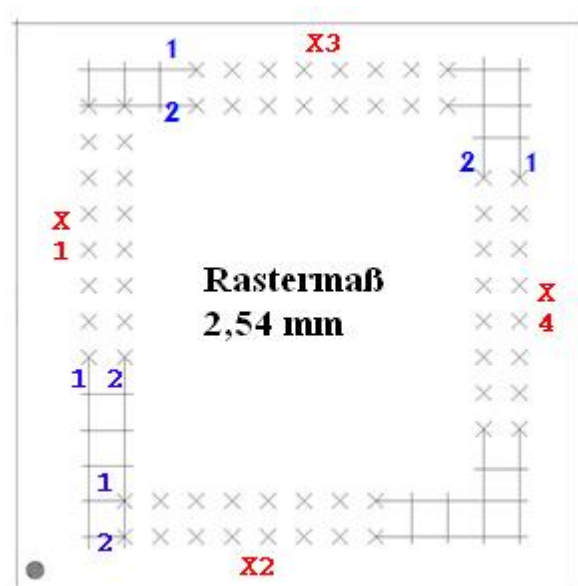
**Global Pull-up Disable**

- **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

### 3.4.1 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.

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:

$$u = x * \text{Referenzspannung} / 1024$$

### 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 (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 bytewise angesprochen werden. Jeder Pin kann entweder Eingang oder Ausgang sein.

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

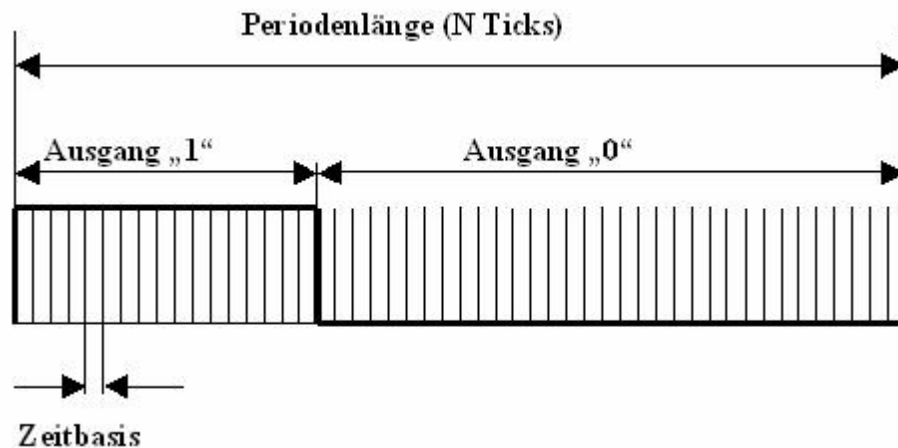
Digitale Eingangspins sind hochohmig oder mit internem Pullup-Widerstand beschaltet und überführen ein anliegendes Spannungssignal in einen logischen Wert. Voraussetzung dafür ist, daß sich das Spannungssignal innerhalb der für TTL-bzw. CMOS-ICs definierten Bereiche für Low- oder

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 pulswertenmodulierten 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).

<b>Umgebungsbedingungen</b>	
Bereich der zulässigen Umgebungstemperatur	0°C ... 70°C
Bereich der zulässigen relativen Umgebungsluftfeuchte	20% ... 60%
<b>Versorgungsspannung</b>	
Bereich der zulässigen Versorgungsspannung	4,5V ... 5,5V
Stromaufnahme des Moduls ohne externe Lasten	ca. 20mA

<b>Takt</b>	
Taktfrequenz (Quarzoszillator)	14,7456MHz
<b>Mechanik</b>	
äußere Abmessungen ohne Pins ca.	40 mm x 40mm x 8 mm
Masse	ca. 90g
Pinraster	2,54mm
Pinanzahl (zweireihig)	64

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

### 3.4.2 Application Board

#### USB

Das Application Board 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. 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.

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

### Ein- Ausschalter

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

### Leuchtdioden

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

### 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 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.

➔ 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

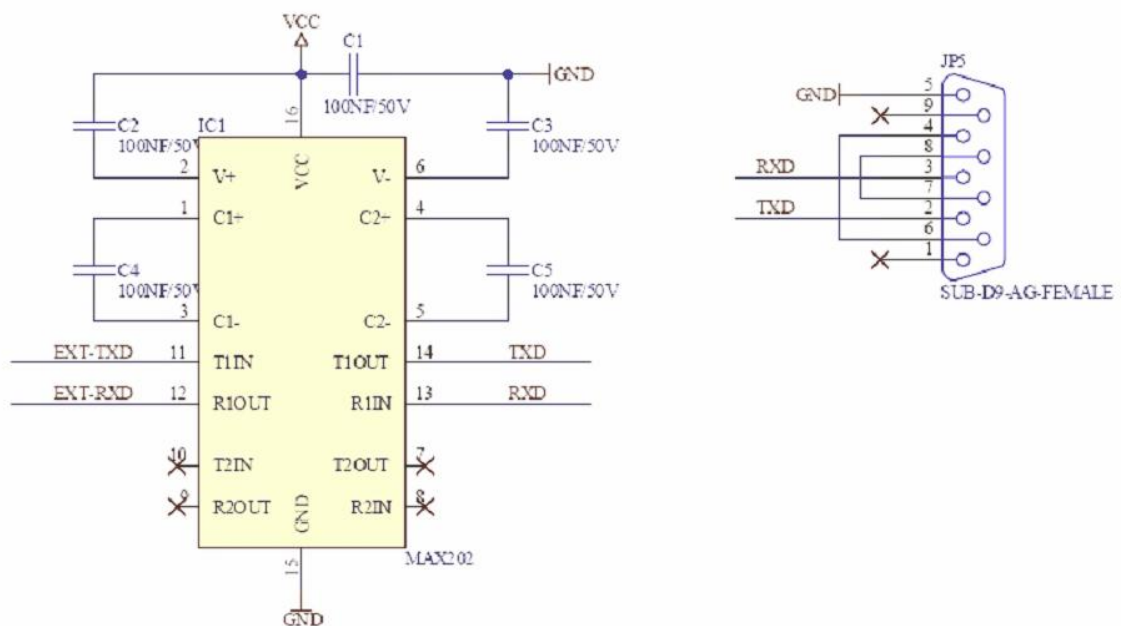
## Spannungsversorgung (POWER, 5 Volt, GND)

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).

<b>Mechanik</b>	
äußere Abmessungen ca.	160 mm x 100 mm
Pinraaster Verdrahtungsfeld	2,54 mm
<b>Umgebungsbedingungen</b>	
Bereich der zulässigen Umgebungstemperatur	0°C ... 70°C
Bereich der zulässigen relativen Umgebungsluftfeuchte	20% ... 60%

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

### 3.4.3 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

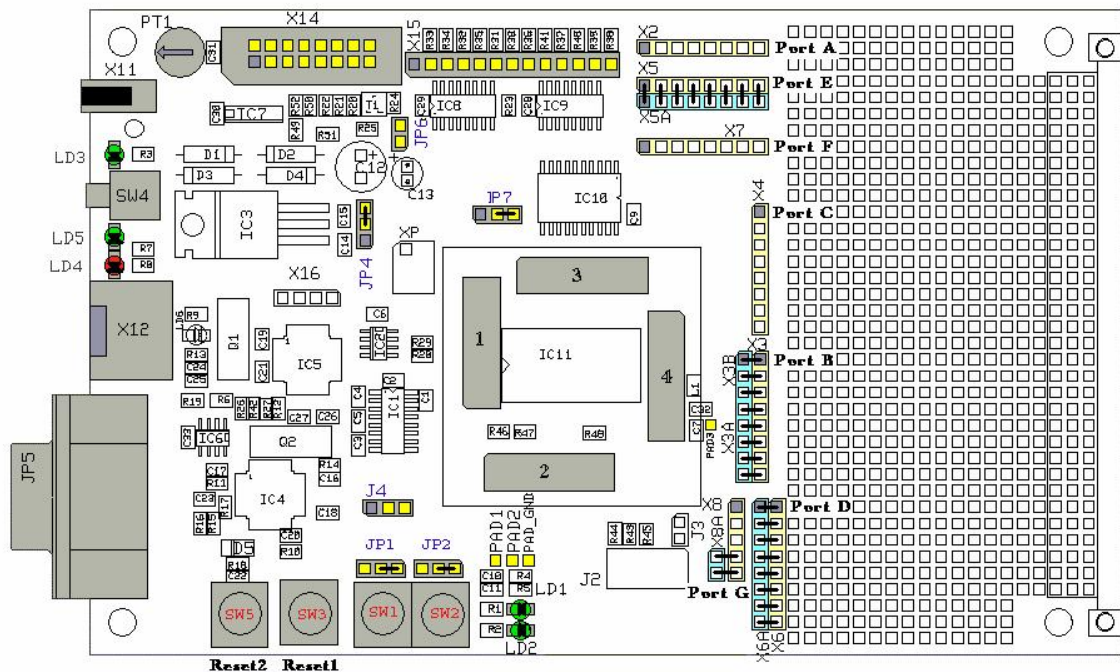
Modul	M128	Port	Port #	PortBit	Name1	Name2	Intern	Bemerkungen
	1				PEN			prog. Enable
X1_16	2	PE0	4	32	RXD0	PDI	EXT-RXD0	RS232
X1_15	3	PE1	4	33	TXD0	PDO	EXT-TXD0	RS232
X1_14	4	PE2	4	34	AIN0	XCK0		Analog Comparator
X1_13	5	PE3	4	35	AIN1	OC3A		Analog Comparator
X1_12	6	PE4	4	36	INT4	OC3B	EXT-T1	Taste1 (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 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
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			

### 3.4.4 Jumper Application Board

#### Jumper

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

## 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 (Auslieferungszustand). 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 dass die linken beiden Stifte von JP7 verbunden sind.

### J4

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

Pin 1 (links, grau)	<b>TxD</b>
Pin 2 (mitte)	<b>RxD</b>
Pin 3 (rechts)	<b>GND</b>

### PAD3

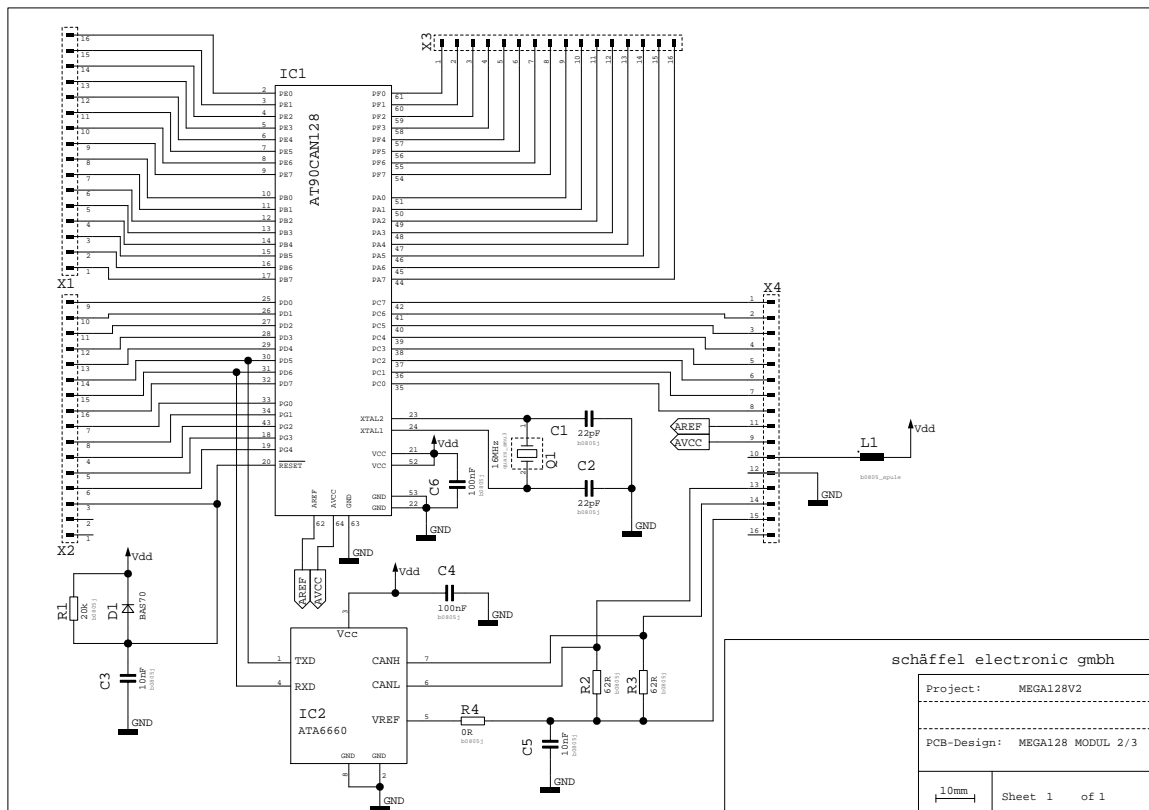
PAD3 (rechts neben dem Modul) wird als ADC\_VREF\_EXT für die Funktionen [ADC\\_Set](#) und [ADC\\_SetInt](#) benötigt.

## 3.4.5 Schaltpläne

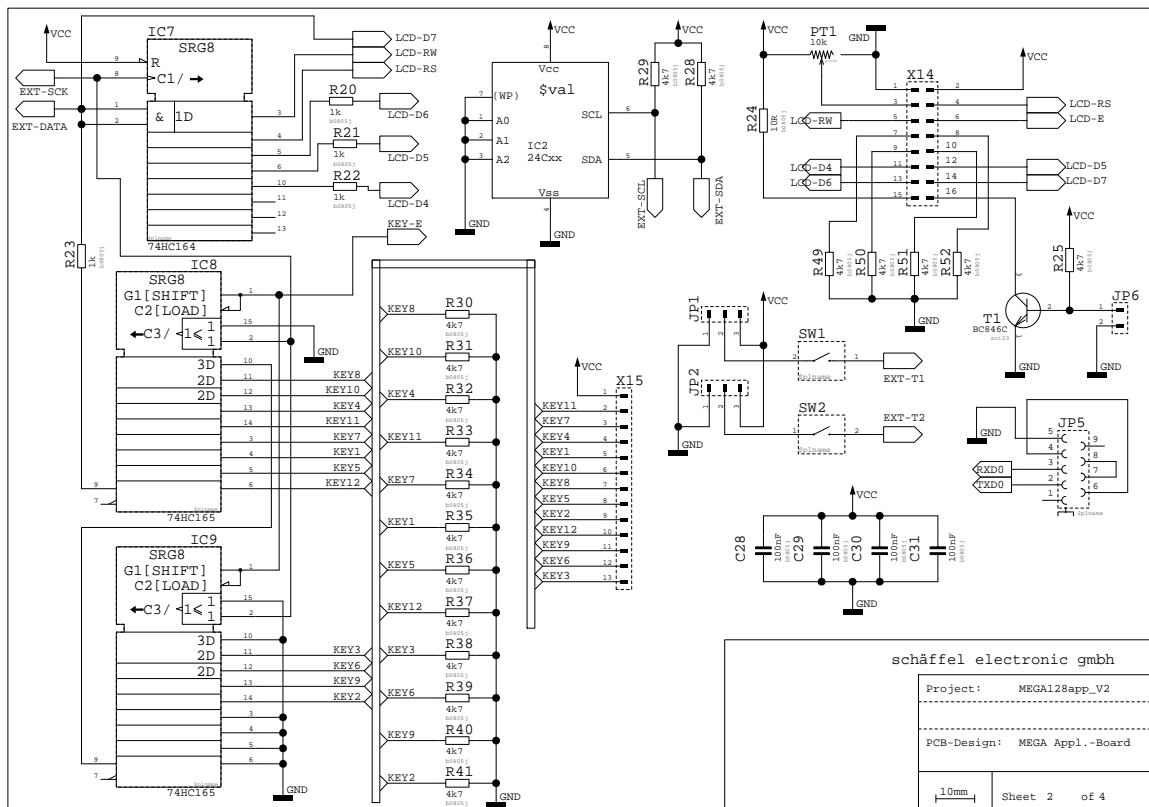
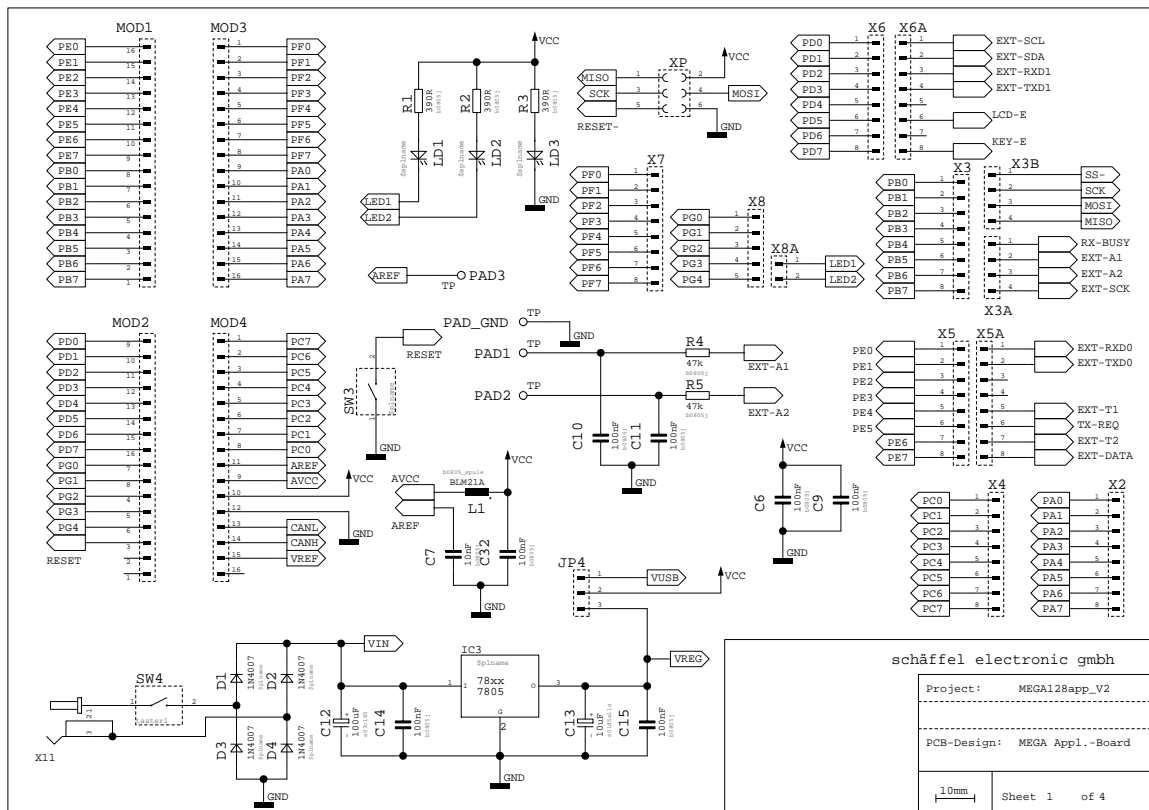
Die Schaltpläne sind zusätzlich als pdf Dateien auf der Installations CD.

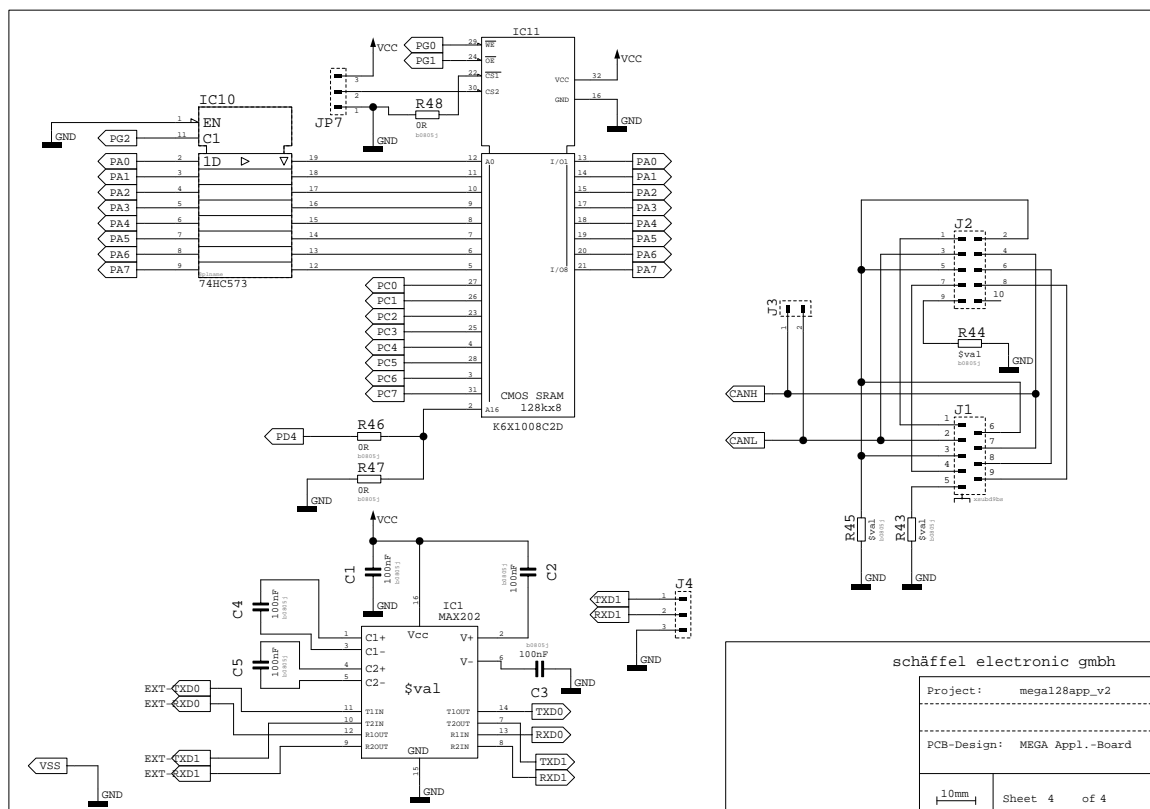
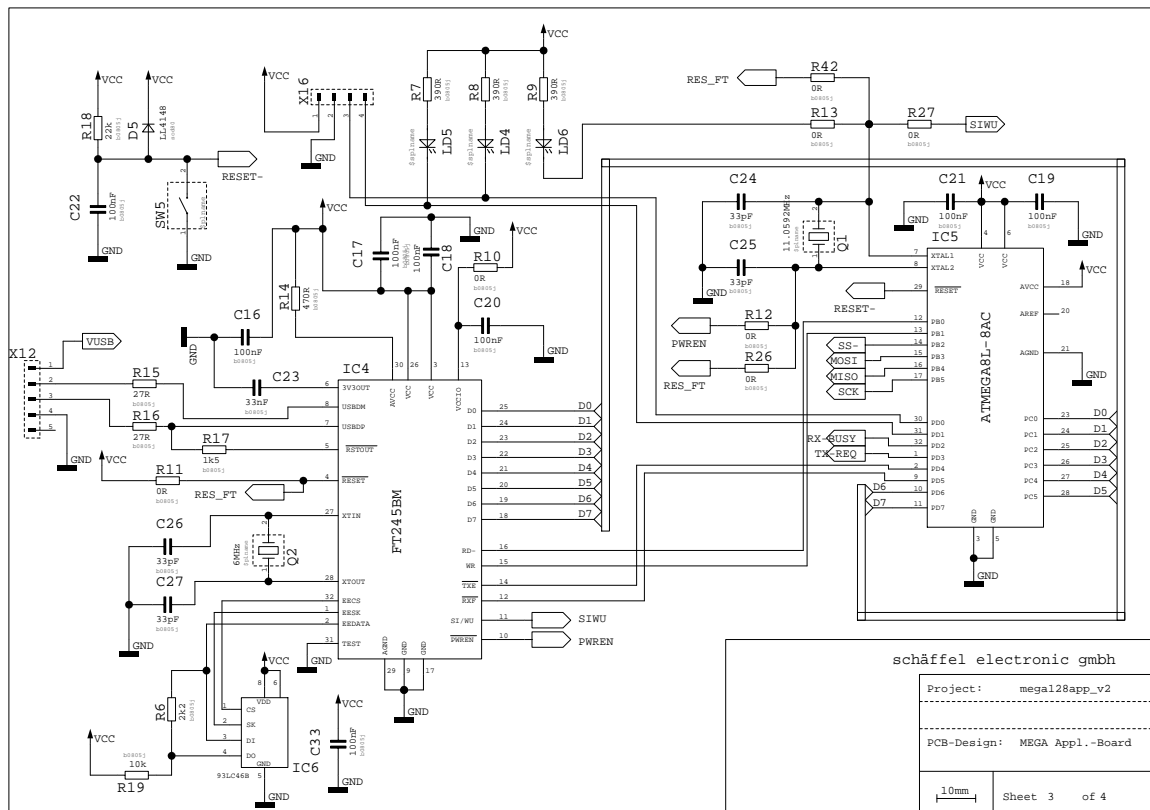
### 3.4.5.1 Mega 128 Modul

➔ 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.

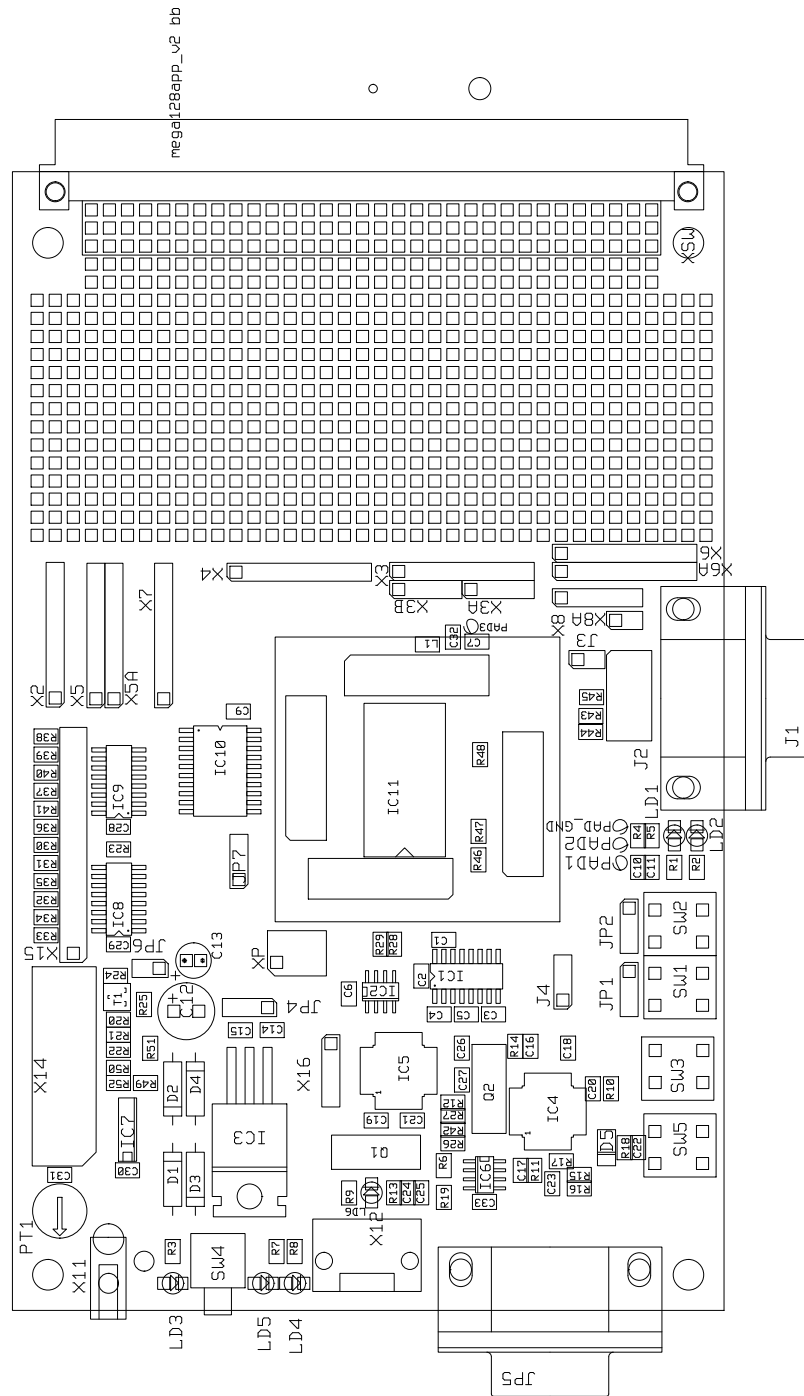


### 3.4.5.2 Applicationboard





### 3.4.5.3 Bestückungsplan



# **Kapitel**



## 4 IDE

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

Sidebar für [Projekt](#)  
[Dateien](#)  
[Editor Fenster](#)

[Compiler Meldungen](#)

[C-Control Ausgaben](#)  
[Variablenfenster](#)

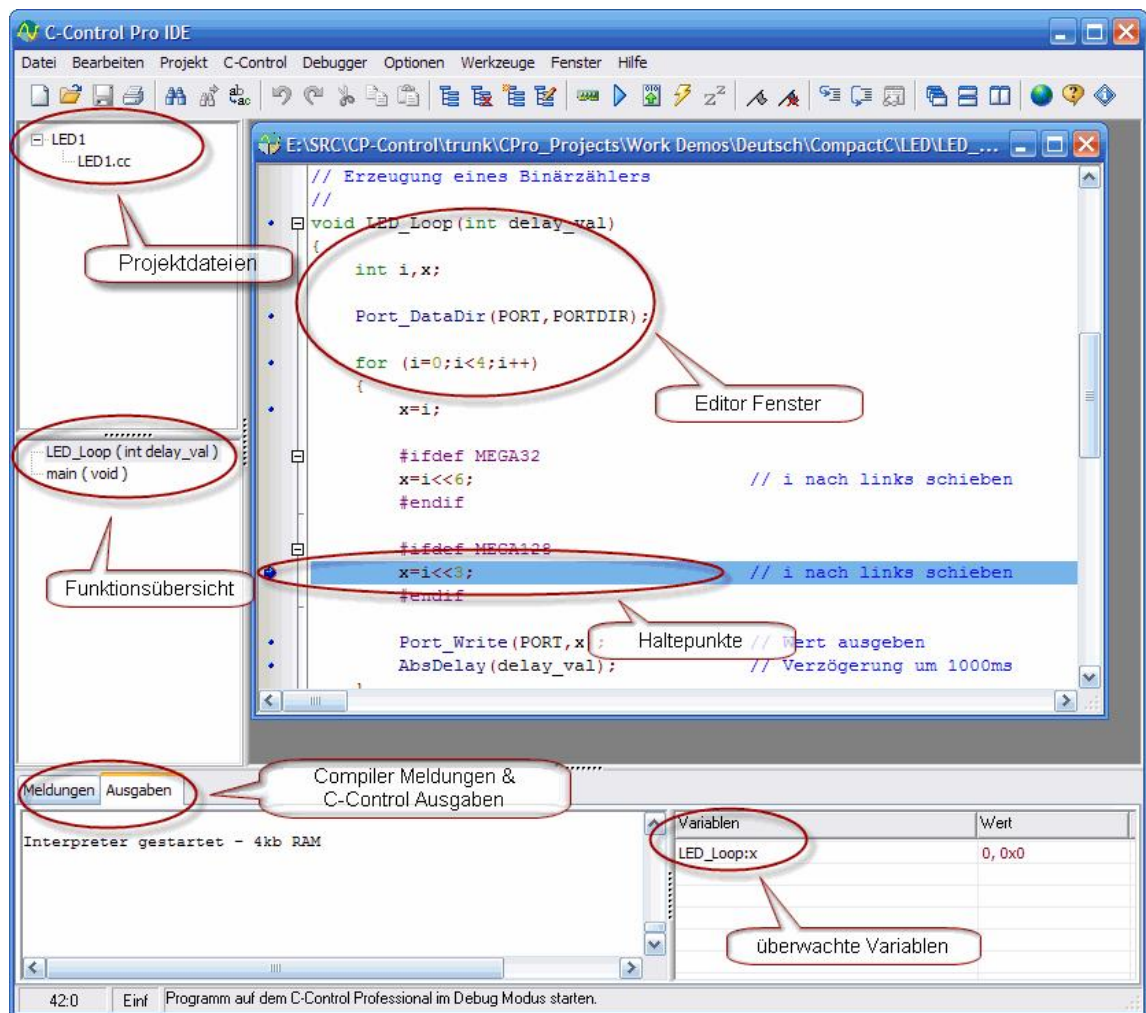
Mehrere Dateien können hier zu einem Projekt abgelegt werden.

Es können beliebig viele Editor Fenster geöffnet werden, um Dateien zu editieren.

Fehlermeldungen und allgemeine Compilerinformationen werden hier angezeigt.

Ausgabe von Debug Nachrichten der CompactC Programme.

Überwachte Variablen werden hier angezeigt.





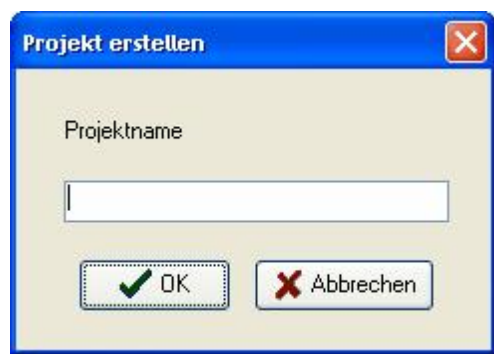
## 4.1 Projekte

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.

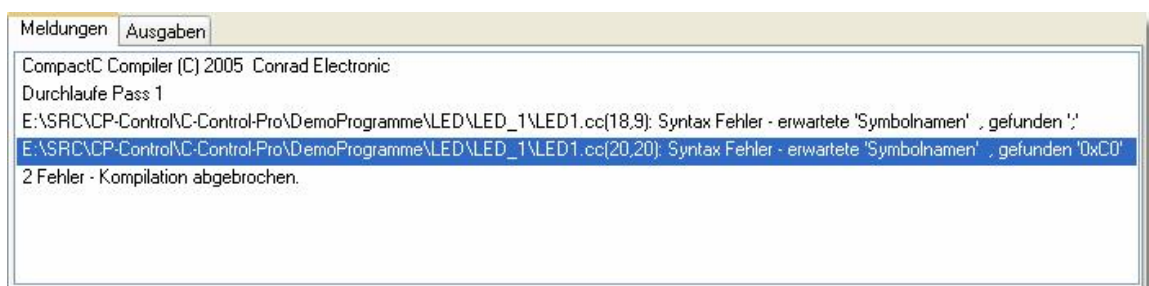
### 4.1.1 Projekterstellung

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.



### 4.1.2 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

### 4.1.3 Projektverwaltung

Klickt man mit der rechten Maustaste auf das neu erstellte Projekt in der Sidebar, so erscheint ein Pop-up-Menü 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.

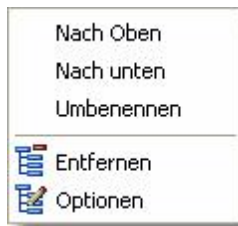
#### Hinzufügen von Projektdateien

Drückt man auf **Hinzufügen** von Projektdateien, so wird ein Datei Öffnen Dialog angezeigt, in dem man die Dateien auswählen 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
- **Entfernen** - Die Datei wird aus dem Projekt entfernt.
- **Optionen** - Die Projektoptionen können geändert werden.

#### 4.1.4 Projektoptionen

The dialog box 'Projekt Optionen' has a blue title bar with a close button. It contains three input fields: 'Autor', 'Version', and 'Kommentar'. Below these are two main sections. The 'Optionen' section has three checkboxes: 'Multithreading' (unchecked), 'Debug Code erzeugen' (checked), and 'Map Datei erzeugen' (checked). To the right of each checkbox is a button: 'Threads Konfigurieren', 'Bibliothek Konfigurieren', and 'Bibliothek Konfigurieren' respectively. The 'CPU Auswahl' section has three radio buttons: 'C-Control 32' (selected), 'C-Control 128', and 'C-Control 128 CAN'. To the right of these is a button 'Hardware abfragen'. At the bottom of the dialog are two buttons: 'OK' with a green checkmark and 'Abbrechen' with a red X.

Für jedes Projekt können die Compilereinstellungen einzeln geändert werden.

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

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

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

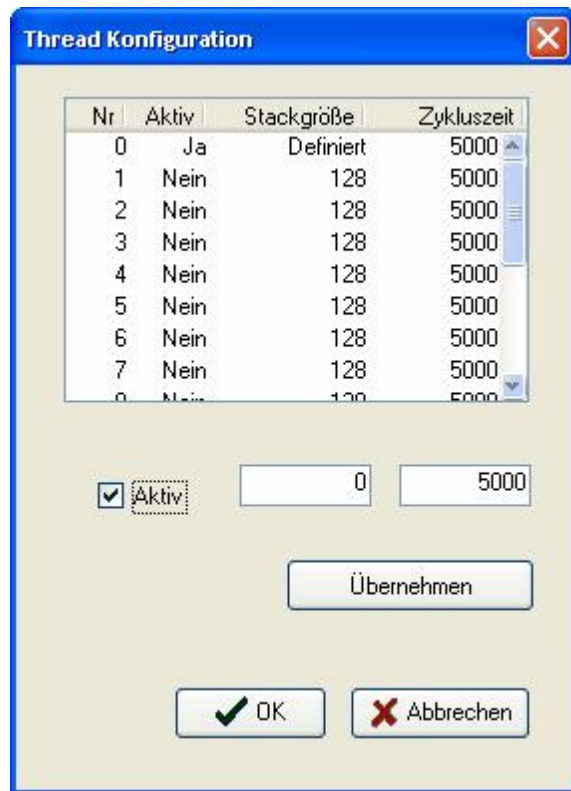
➔ Wird mit Debug Code kompiliert, wird der Bytecode geringfügig länger. Pro Zeile im Quelltext die ausführbare Anweisungen enthält, wird der Bytecode um ein Byte größer.

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

Man kann in den Optionen auch wählen, ob eine [Map Datei](#) erzeugt werden soll.

#### 4.1.5 Threadoptionen

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



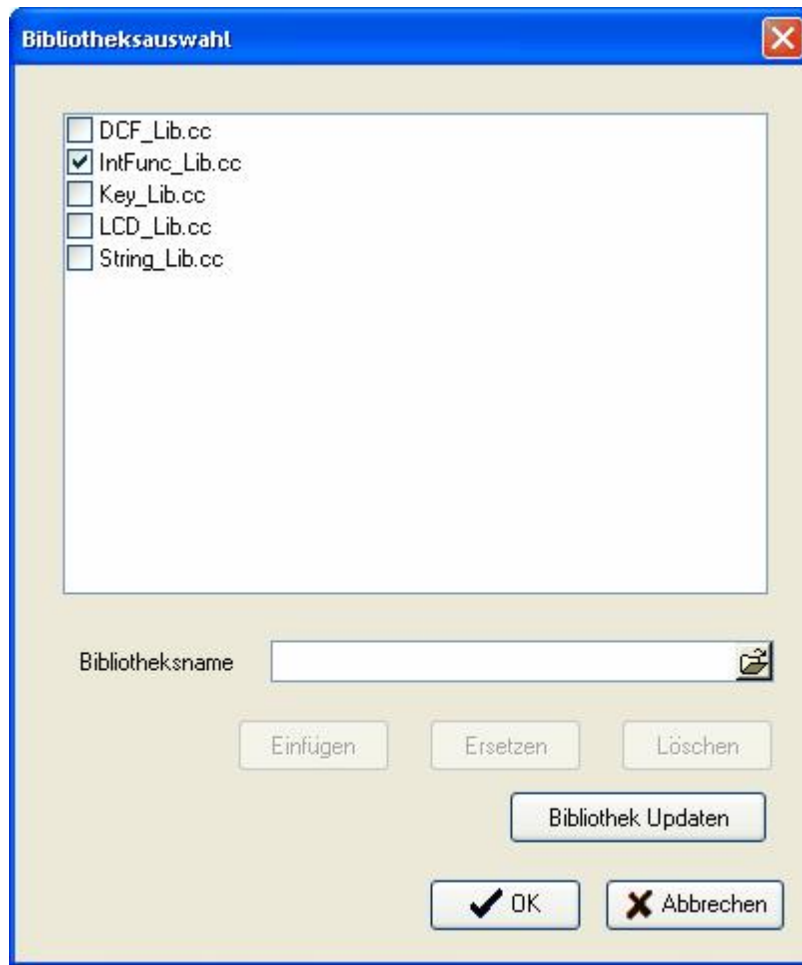
Jedem zusätzlichen Thread, außer dem Hauptprogramm, wird ein Platz auf dem Stack zugewiesen, den er nicht überschreiten darf.

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

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

#### 4.1.6 Bibliotheksverwaltung

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



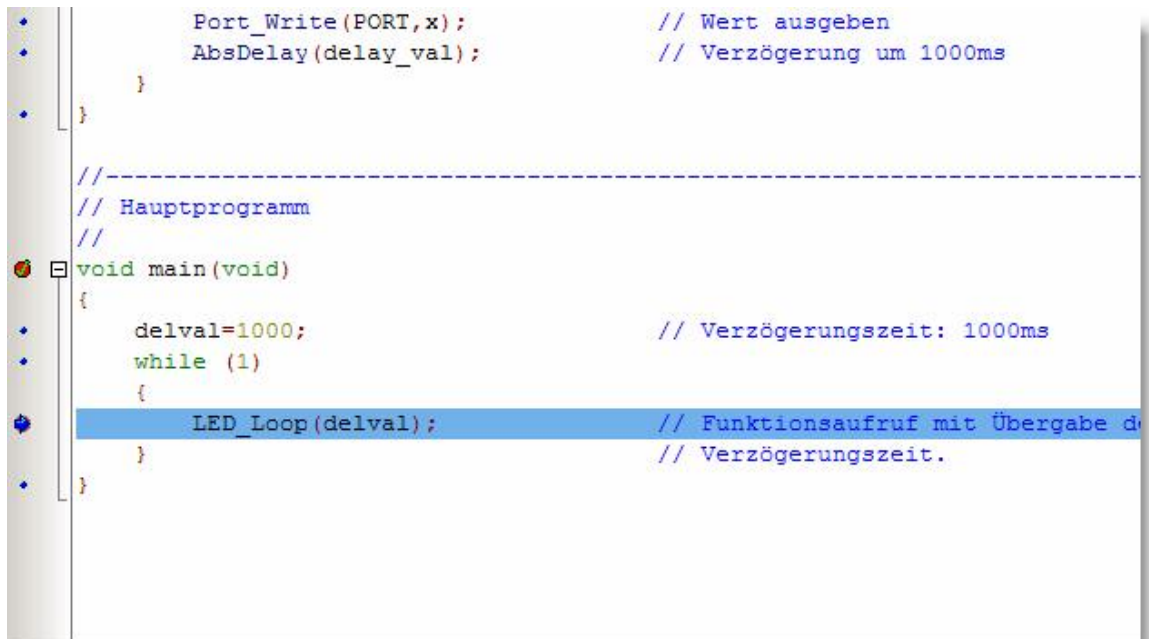
Es werden nur die Dateien zur Kompilierung hinzugezogen, deren CheckBox auch selektiert wurde.

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

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

## 4.2 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.



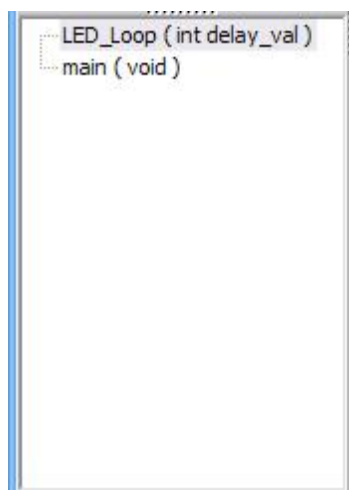
```
•      Port_Write(PORT,x);           // Wert ausgeben
•      AbsDelay(delay_val);          // Verzögerung um 1000ms
•  }

//-----
// Hauptprogramm
//
void main(void)
{
    delval=1000;                      // Verzögerungszeit: 1000ms
    while (1)
    {
        LED_Loop(delval);             // Funktionsaufruf mit Übergabe d
        // Verzögerungszeit.
    }
}
```

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.).

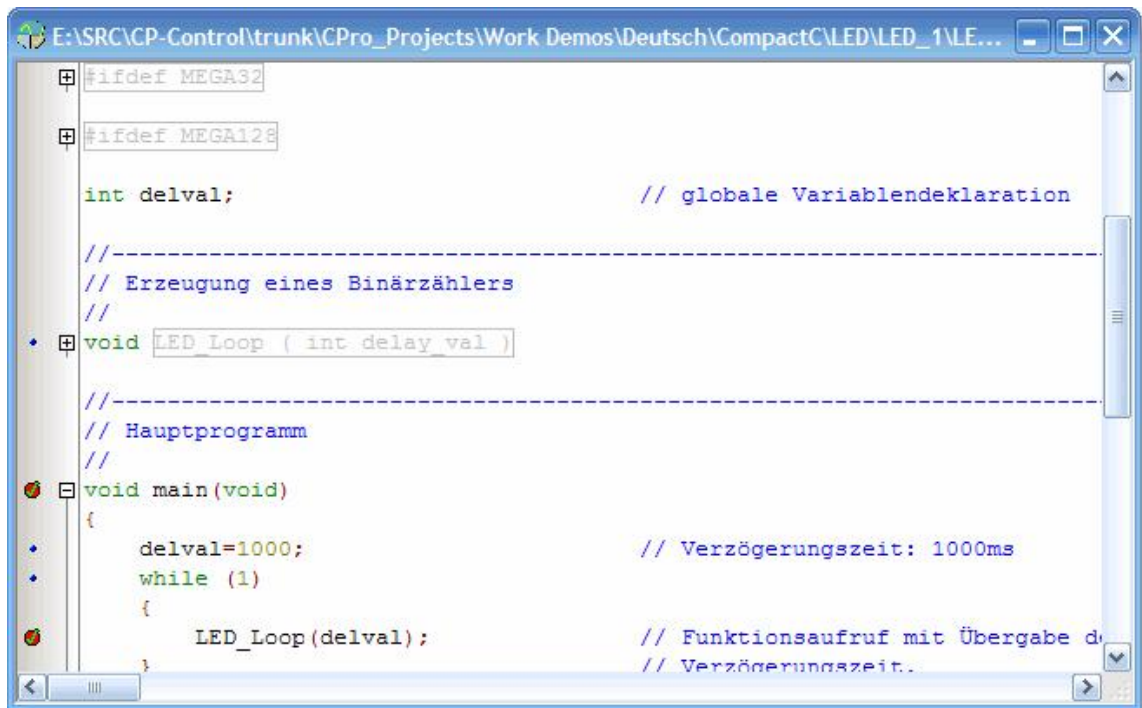
## Funktionsübersicht

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.

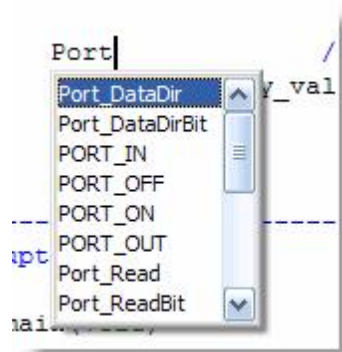


Um alle Funktionen in einer Datei zu falten, oder zu entfalten, kann man mit Rechtsklick im Editor im Menü **Zeilen falten** oder **Zeilen entfalten** auswählen.

## 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.





## Editor Ansicht Erneuern

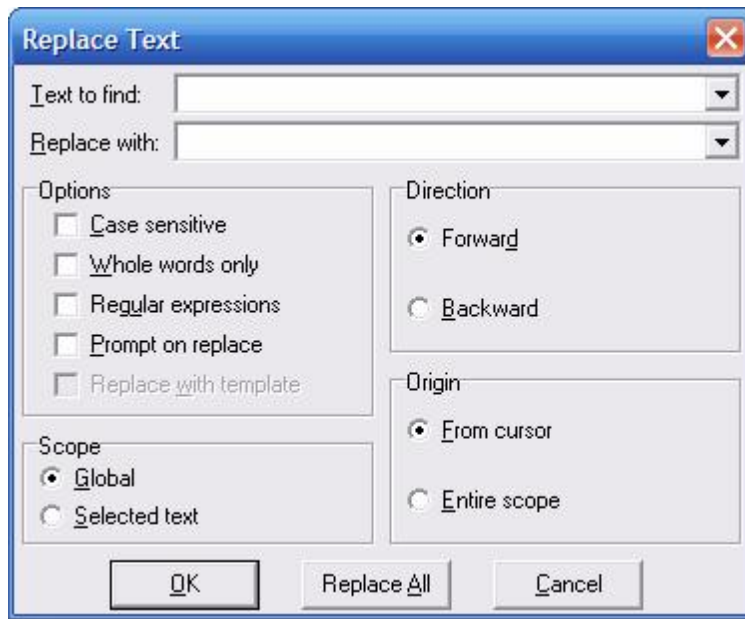
Sollte der syntaktische Analyzer einmal fehllaufen und definierte Funktionsblöcke nicht mehr erkennen (passiert selten mal bei Suchen - Ersetzen), dann kann man mit dem Befehl **Erneuern** aus dem **Bearbeiten** Pull-down Menü, die syntaktische Analyse wiederherstellen.

### 4.2.1 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



- **Text to find** - Eingabefeld für den zu suchenden Text.
- **Replace with** - der Text der den gefunden Text ersetzt.
- **Case Sensitive** - unterscheidet Groß- und Kleinschreibung.
- **Whole words only** - findet nur ganze Wörter und keine Teilzeichenketten.
- **Regular expressions** - aktiviert die Eingabe von [regulären Ausdrücken](#) in der Suchmaske.
- **Prompt on replace** - vor dem Ersetzen wird beim Benutzer nachgefragt.

Weiter kann die Suchrichtung ("**Forward**", "**Backward**") vorbestimmt werden, ob der gesamte Text ("**Global**") , oder nur ein selektierter Bereich ("**Selected Text**") durchsucht wird. Auch wird eingestellt ob die Suche am Cursor beginnt ("**From Cursor**") oder am Textanfang ("**Entire Scope**").

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

### 4.2.2 Druckvorschau

Um den Source Code von Projekten in Papierform anderen 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:

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

```

// LED1: Binary Counter
// A binary counter is shown at LED1/LED2.
// used Library: IntFunc_Lib.cc

// Mega32: LED1/2 are accessed from PortD
// Mega128: LED1/2 are accessed from PortG

// LED is lit when Port Pin is low
// 0 = PORT A, 1 = PORT B, 2 = PORT C, 3 = PORTD
// MEGA128: 4 = PORT E, 5 = PORT F, 6 = PORT G

#ifdef MEGA32
#define PORT 3
#define PORTDIR 0xC0
#endif

#ifdef MEGA128
#define PORT 6
#define PORTDIR 0x18
#endif

int delval;

//-----
// Create a binary counter
//
void LED_Loop(int delay_val)
{
    int i,x;

    Port_DataDir(PORT,PORTDIR);

    for (i=0;i<4;i++)
    {
        x=1;

        #ifdef MEGA32
        x=x<<6;
        #endif
    }
}

```

### 4.2.3 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 + E	Delete from cursor to end 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

#### 4.2.4 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.
\\$	'\$'

## 4.3 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.

### 4.3.1 Programm starten

#### Programmübertragung

Ist ein Projekt fehlerfrei übersetzt worden, so muß der Bytecode erst auf den Mega 32 oder Mega 128 übertragen werden, bevor er ausgeführt werden kann. Dies geschieht mit dem Befehl **Übertragen** (Shift-F9) aus dem Menü **C-Control**.

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

#### Starten

Durch **Starten** (F10) wird dann die Ausführung des Bytecode auf dem Mega 32 oder Mega 128 veranlaßt.

#### Stoppen

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

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

#### Autostart

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

➔ Ein Signal auf **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.

### 4.3.2 Ausgaben

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



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

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

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

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

### 4.3.3 PIN Funktionen

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

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

Durch Eingabe der PIN über **PIN Eingeben** im **C-Control** 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.

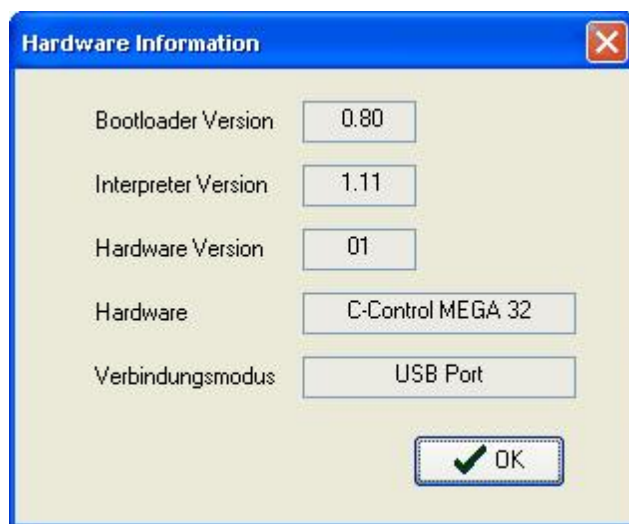
➔ Hat man das Passwort vergessen, gibt es eine Notfallfunktion, um das Modul in den Ausgangszustand zurückzusetzen. Unter **C-Control** existiert die Option **Modul zurücksetzen**, mit

der man PIN, Interpreter und Programm löschen kann.



#### 4.3.4 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.



### 4.4 Debugger

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

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



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

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

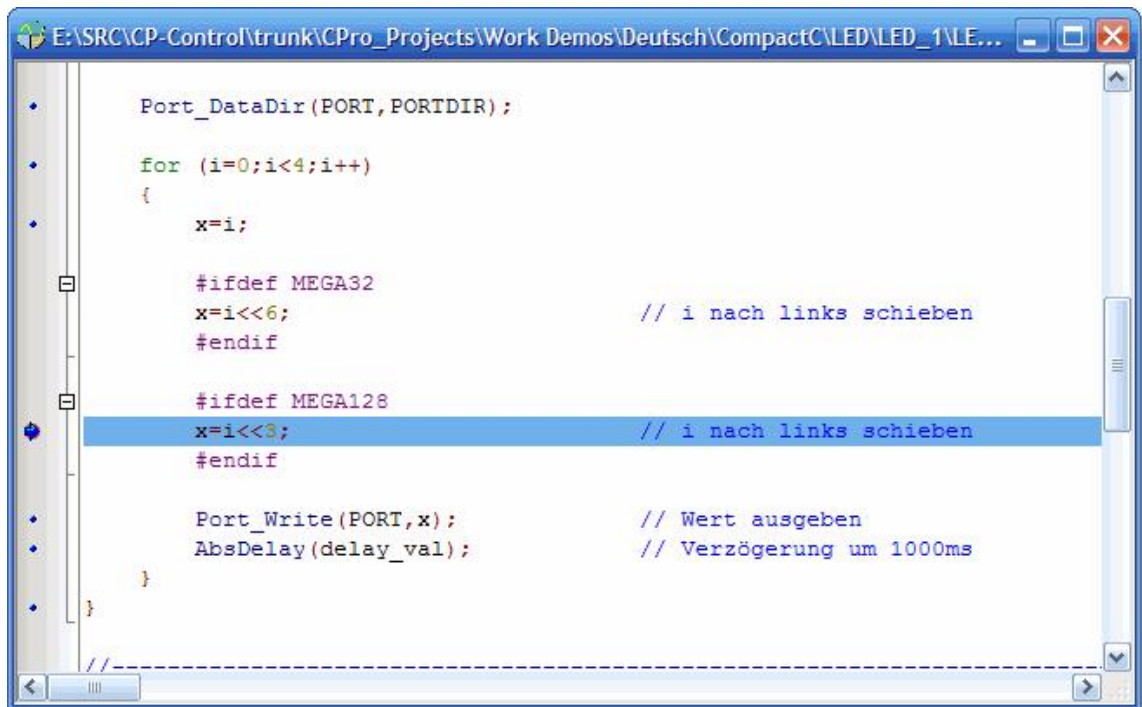
➔ Ist in einer Schleife nur eine Codezeile, so führt ein Einzelschritt die ganze Schleife aus, da erst dann zu einer neuen Codezeile verzweigt wird.

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

➔ Während der Debug Modus aktiv ist, kann der Programmtext nicht geändert werden. Dies geschieht, damit sich die Zeilennummern 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.

#### 4.4.1 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.

#### 4.4.2 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.

Variablen	Wert
main:a	nicht im Debug
main:z	nicht im Debug

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.

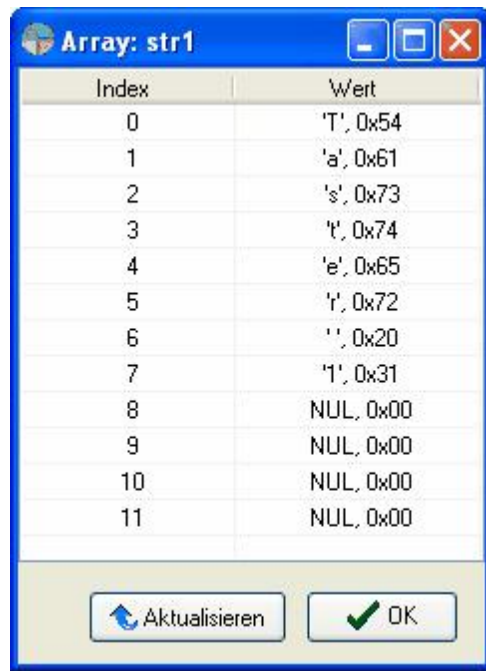
➔ Variablen vom Typ character werden als einzelnes ASCII Zeichen dargestellt.

#### 4.4.3 Array Fenster

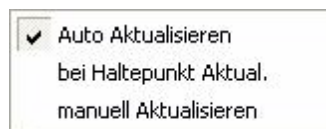
Um die Inhalte von Array Variablen zu betrachten ist es möglich ein Fenster mit dem Inhalt des Arrays aufzurufen. Dafür wird der Cursor auf der Variablen positioniert, und mit Rechtsklick **Array anzeigen** angewählt.



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:

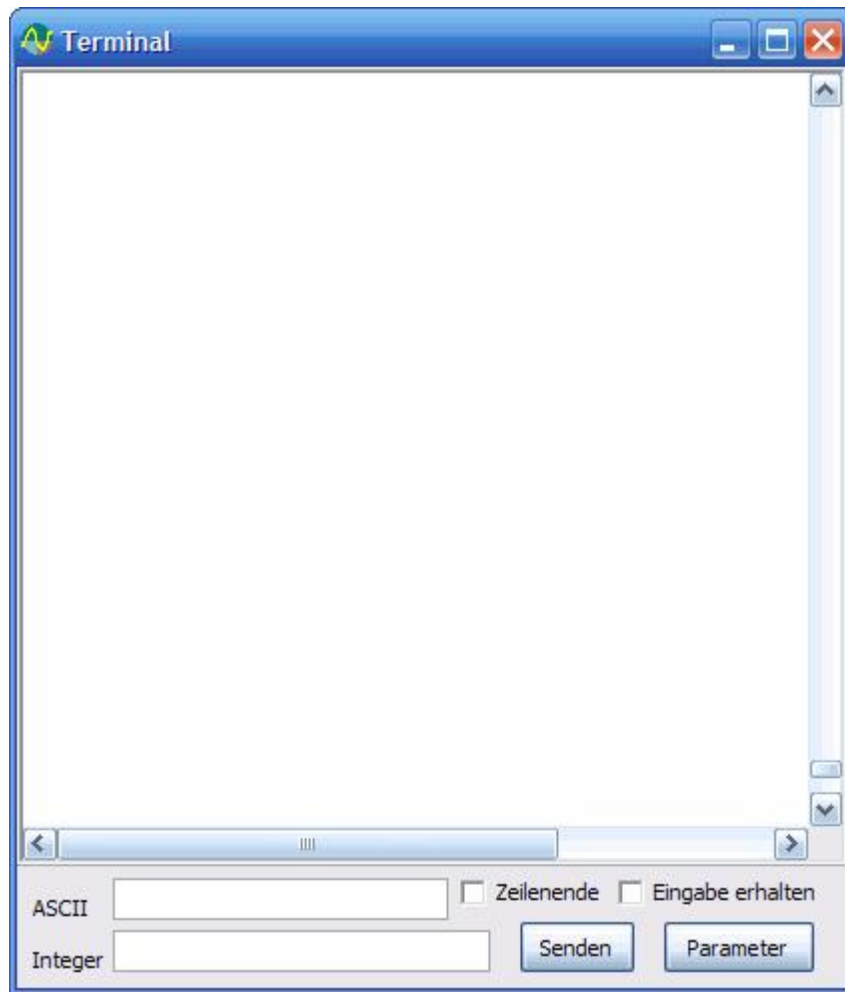


Auto Aktualisieren	Aktualisierung bei Einzelschritt und Haltepunkt
bei Haltepunkt Aktualisieren	Aktualisierung nur bei Haltepunkt (Breakpoint)
manuell Aktualisieren	nur bei Klick auf den Schalter "Aktualisieren"

## 4.5 Werkzeuge

### Terminal Fenster

Unter dem Menüpunkt **Werkzeuge** kann man das einfache integrierte Terminalprogramm starten.

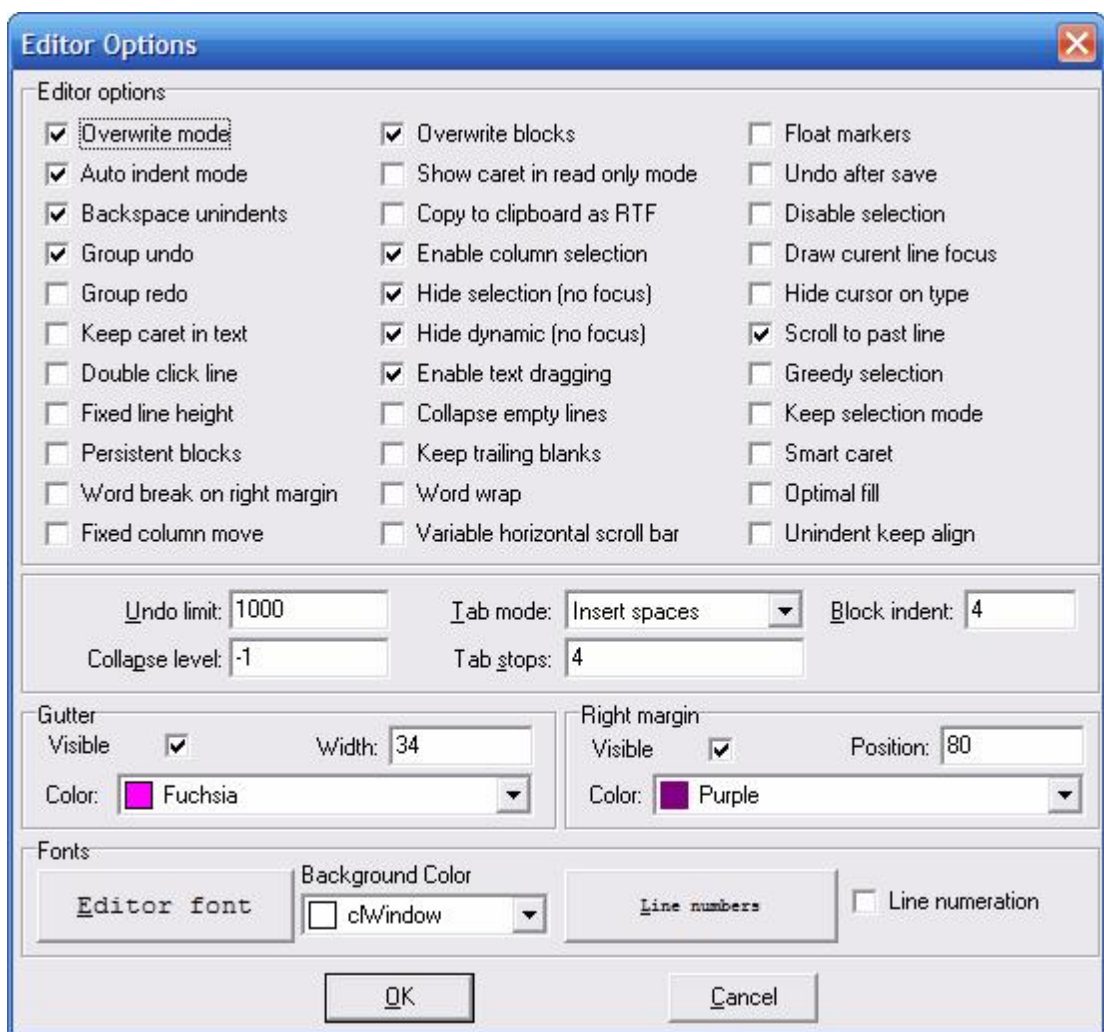


Empfangene Zeichen werden direkt im Terminalfenster dargestellt. Man kann Zeichen auf zwei verschiedene Arten senden. Zum einen kann man in das Fenster klicken und die Zeichen direkt über die Tastatur eingeben, oder man schreibt die Zeichen in die **ASCII** Eingabezeile und drückt den **Senden** Knopf. 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, dass nach dem **Senden** der Text in den Eingabezeilen nicht gelöscht wird. Der Knopf **Parameter** öffnet den [Terminal Konfigurationsdialog](#) in den IDE Einstellungen.

## 4.6 Optionen

Im Menü **Optionen** sind die Einstellungen der IDE und die Voreinstellungen für den Compiler zu finden.

### 4.6.1 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 durchgeführt.
- **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.

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

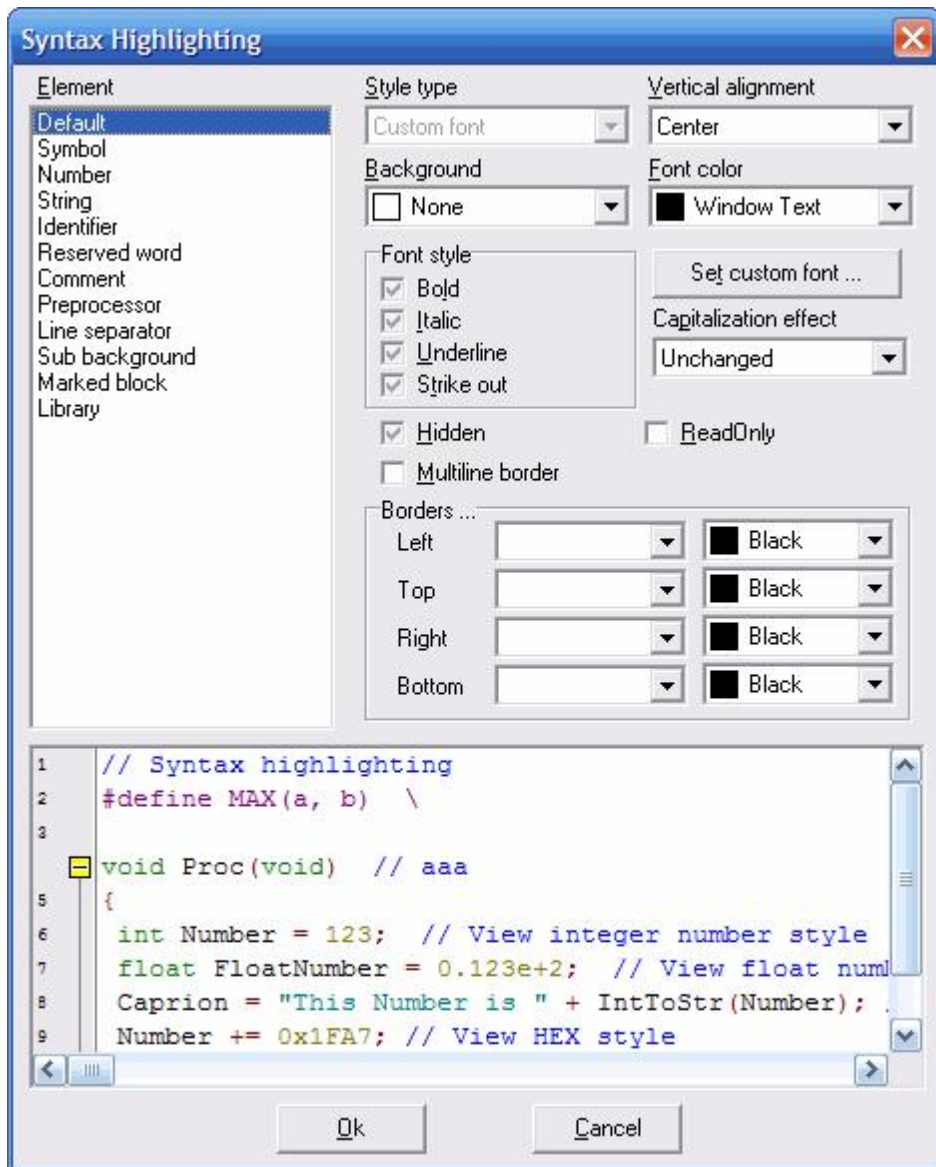
Das Eingabefeld **Tab stops** bestimmt wie viele Zeichen ein Tabulator breit ist.

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

## 4.6.2 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:

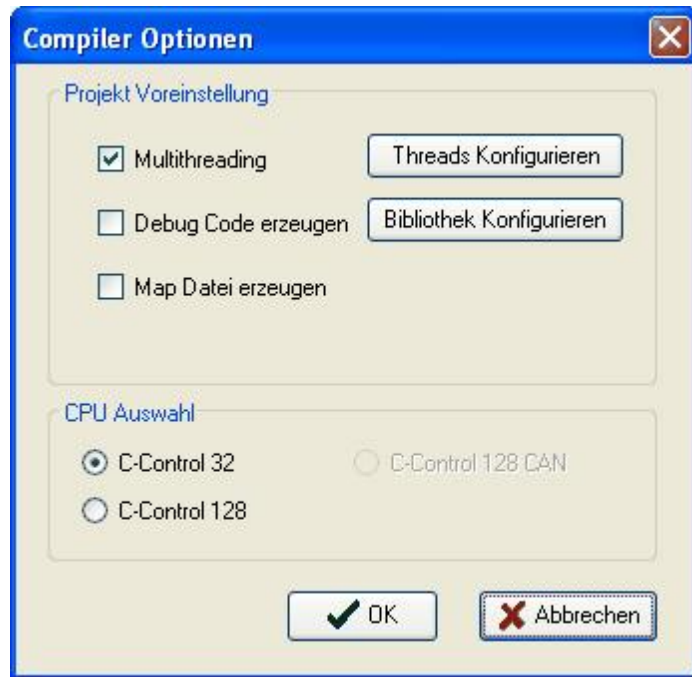
<b>Symbol:</b>	Alle nicht alpha-numerischen Zeichen
<b>Number:</b>	alle numerischen Zeichen
<b>String:</b>	Zeichen die von der Zielsprache als Zeichenkette interpretiert werden ("String")
<b>Identifier:</b>	Alle Namen die nicht Befehlswörter sind oder aus der Standardbibliothek stammen
<b>Reserved Word: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

Default, Line separator and Sub background werden nicht benutzt.

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

### 4.6.3 Compilervoreinstellung

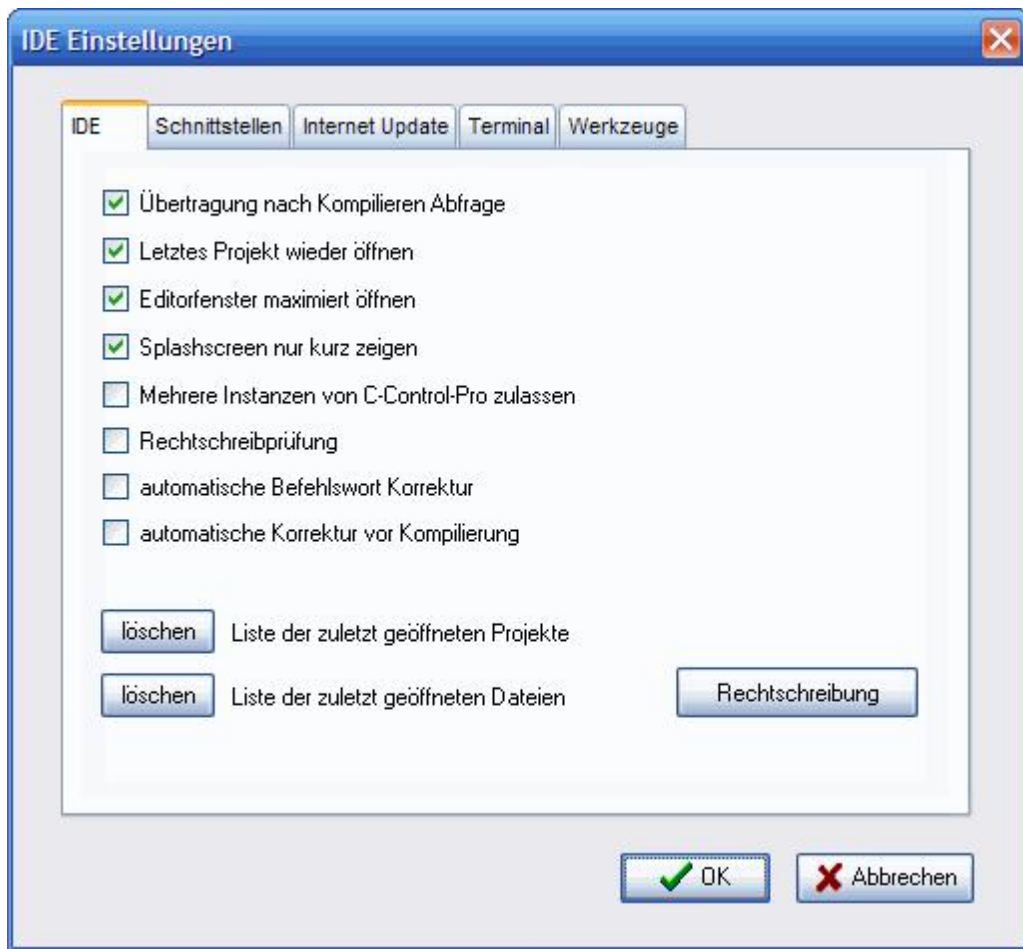
In der Compilervoreinstellung können die Standardwerte konfiguriert werden, die beim Erzeugen eines neuen Projektes gespeichert werden. Die Voreinstellung ist unter Compiler im Menü Optionen zu erreichen.



Eine Beschreibung der Optionen befindet sich bei [Projektoptionen](#). Die Auswahlboxen "[Threads Konfigurieren](#)" und "[Bibliothek Konfigurieren](#)" sind identisch zu den Beschreibungen im Kapitel Projekte.

#### 4.6.4 IDE Einstellungen

Man kann einzelne Aspekte der IDE konfigurieren.



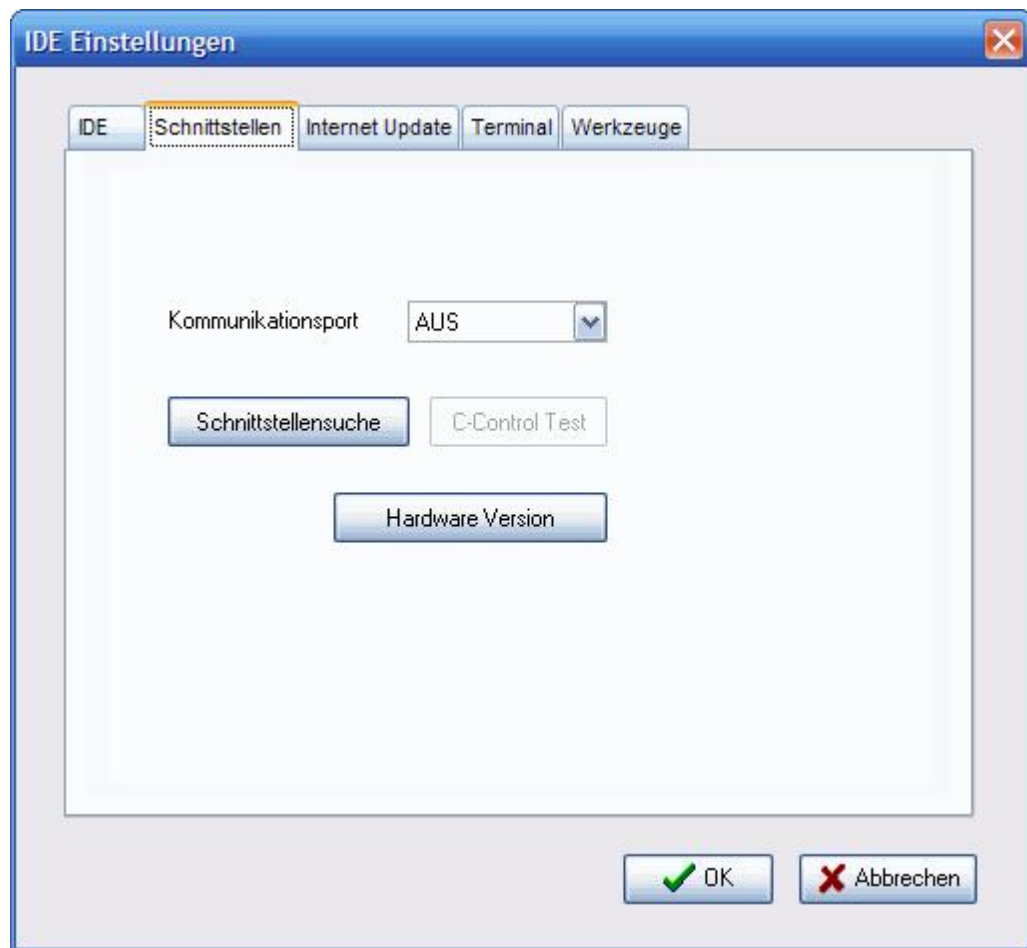
- **Übertragung nach Kompilieren Abfrage** - Wurde ein Programm kompiliert, aber nicht zum C-Control Modul übertragen, erfolgt eine Nachfrage beim Benutzer ob das Programm gestartet werden soll.
- **Letztes Projekt wieder öffnen** - Das letzte offene Projekt wird beim Starten der C-Control Pro IDE wieder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

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

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

#### 4.6.4.1 Schnittstellen

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



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

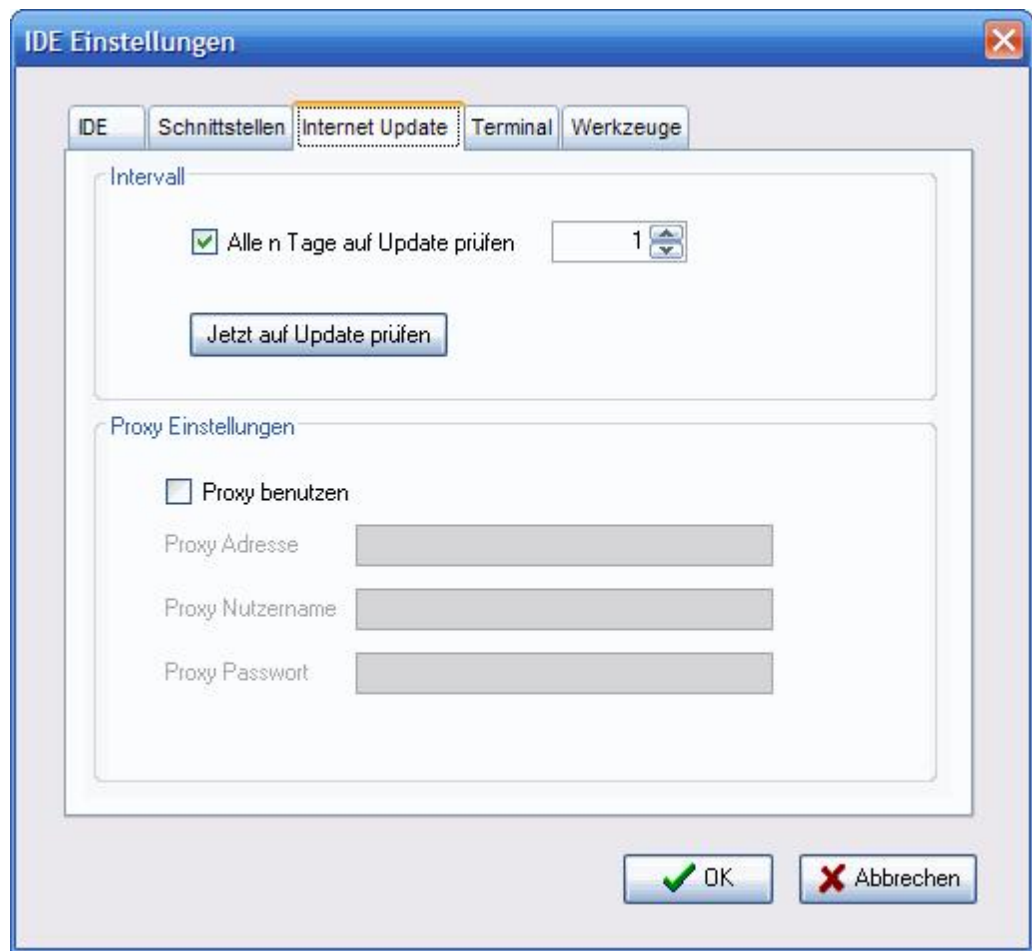
Die Knöpfe "C-Control Test" und "Hardware Version" ermöglichen es sofort zu sehen, ob die ausgewählte Schnittstelle auch sinnvoll mit dem C-Control Pro Modul kommunizieren kann.

#### 4.6.4.2 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.

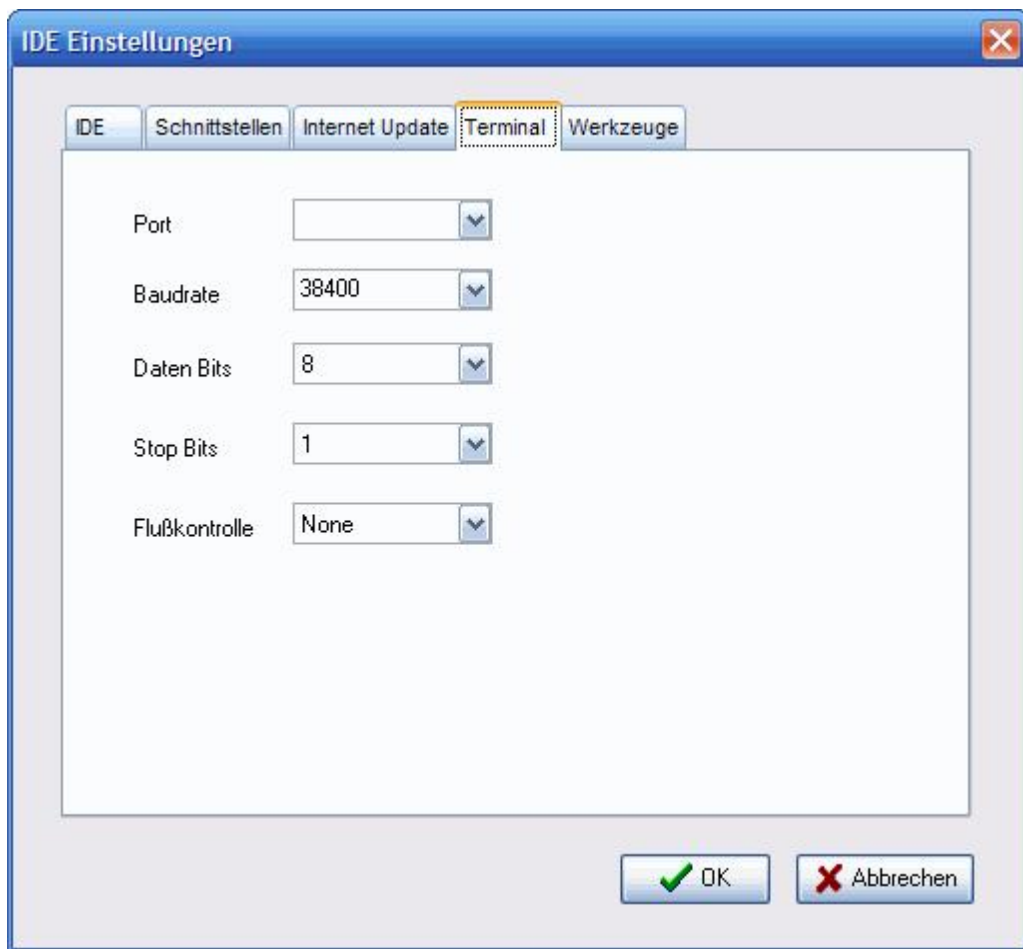


Ist z.B. wegen einer Firewall, der Zugang auf das Internet durch einen Proxy beschränkt, so können die Proxy Einstellungen wie Adresse, Benutzernamen und Passwort in diesem Dialog angegeben werden.

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

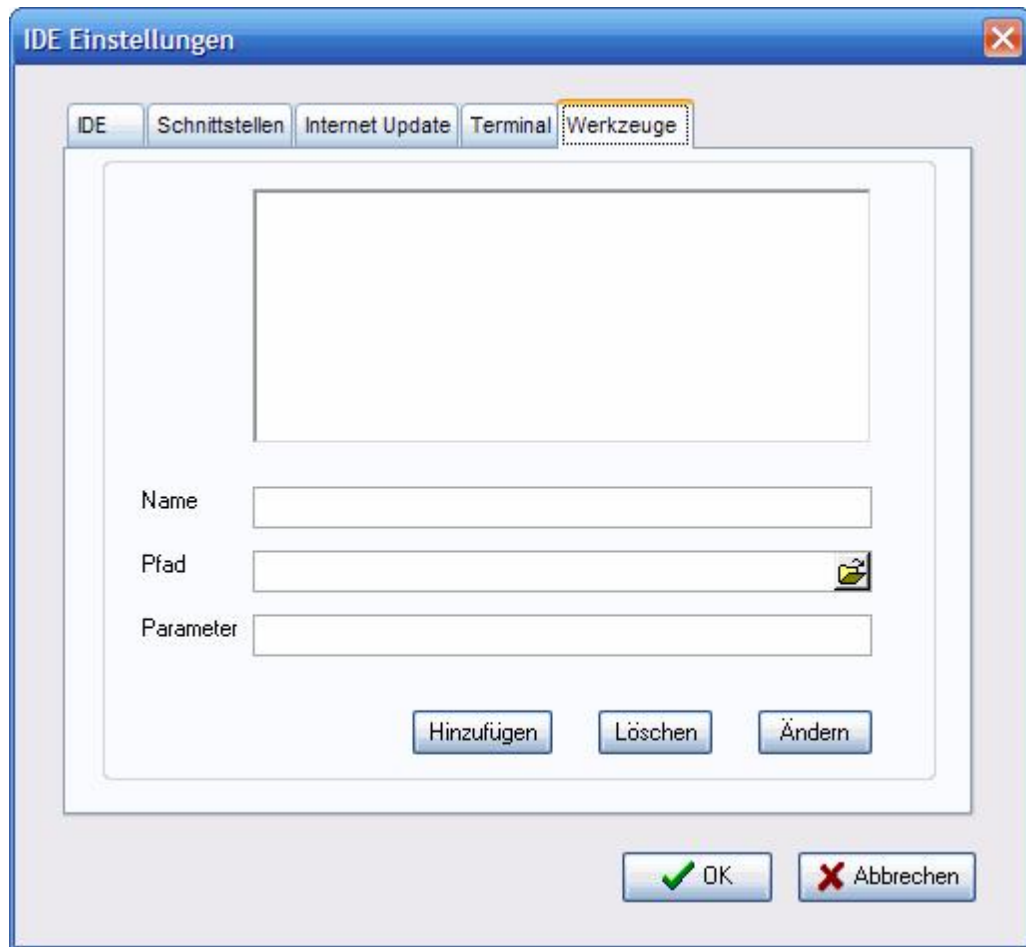
#### 4.6.4.3 Terminal

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.



#### 4.6.4.4 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.

## 4.7 Fenster

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

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

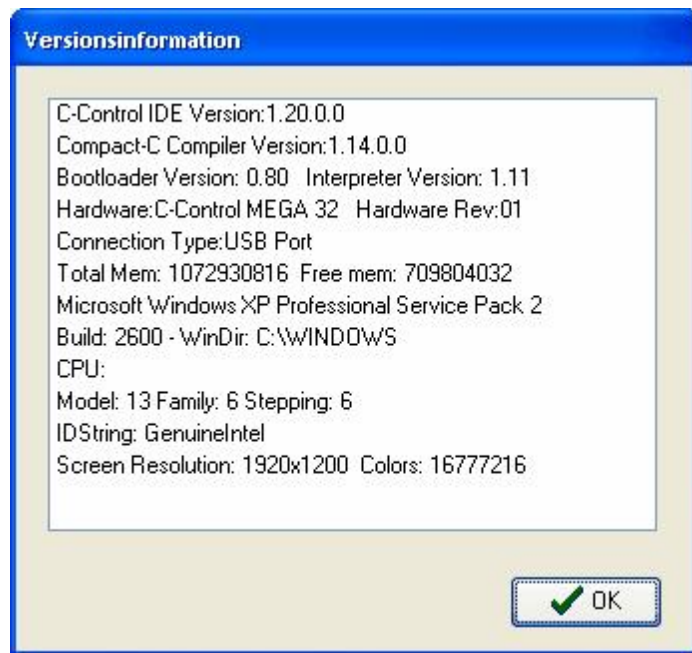


## 4.8 Hilfe

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

Der Menüpunkt **Programmversion** öffnet folgendes "Versionsinformationen" Fenster und kopiert gleichzeitig den Inhalt in die Ablage.

Soll eine Support email an Conrad geschrieben werden, so sind diese Informationen wichtig. Da sie beim Aufruf von **Programmversion** auch gleich in der Ablage sind, kann man diese Daten bequem an das Ende einer email einfügen.



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



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

# **Kapitel**



## 5 Compiler

### 5.1 Allgemeine Features

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

#### 5.1.1 externes RAM

Auf dem Application Board des **Mega128** 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.

#### 5.1.2 Preprozessor

➔ Der Gnu Generic Preprocessor, der hier zum Einsatz, kommt hat noch weitere Funktionen, die unter <http://nothingisreal.com/gpp/gpp.html> dokumentiert sind. Allerdings sind nur die hier beschriebenen Funktionen, auch im Zusammenspiel mit dem C-Control Pro Compiler, 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!

### 5.1.2.1 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
<b>MEGA32</b>	Konfiguration für Mega 32
<b>MEGA128</b>	Konfiguration für Mega 128
<b>MEGA128CAN</b>	Konfiguration für Mega 128 CAN Bus
<b>__DEBUG__</b>	Debugdaten werden erzeugt
<b>__MAPFILE__</b>	Ein Speicherlayout wird berechnet

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

Symbol	Bedeutung
<b>__DATE__</b>	aktuelles Datum
<b>__TIME__</b>	Uhrzeit der Kompilierung
<b>__LINE__</b>	aktuelle Zeile im Sourcecode

<b>FILE</b>	Name der aktuellen Quelldatei
<b>FUNCTION</b>	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
```

## 5.1.3 Pragma Anweisungen

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

<code>#pragma Error "xyz..."</code>	Ein Fehler wird erzeugt und der Text "xyz..." ausgegeben
<code>#pragma Warning "xyz..."</code>	Eine Warnung wird erzeugt und der Text "xyz..." ausgegeben
<code>#pragma Message "xyz..."</code>	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 bestimmte Hardwarekriterien nicht erfüllt werden:

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

## 5.1.4 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. Weiters existieren zwei Funktionen, "Pulse()" und "main()". Jede dieser Funktionen hat einen Speicherverbrauch von 4 Byte an lokalen Variablen.

## 5.2 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 ressourcensparend implementiert werden mußte. Folgende Sprachkonstrukte fehlen:

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

Ausführliche Programmbeispiele sind im Verzeichnis "Demoprogramme" 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.

### 5.2.1 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.

### 5.2.2 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. Deklaration von Array [Variablen](#).

## 5.2.3 Datentypen

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

Datentyp	Vorzeichen	Wertebereich	Bit
<b>char</b>	Ja	-128 ... +127	8
<b>unsigned char</b>	Nein	0 ... 255	8
<b>byte</b>	Nein	0 ... 255	8
<b>int</b>	Ja	-32768 ... +32767	16
<b>unsigned int</b>	Nein	0 ... 65535	16
<b>word</b>	Nein	0 ... 65535	16
<b>float</b>	Ja	$\pm 1.175e-38$ to $\pm 3.402e38$	32

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

### Strings

Es existiert kein expliziter "String" Datentyp. Ein String basiert auf einem character array. Man muß die Größe des arrays so wählen, 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.

## 5.2.4 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. 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.

Beispiele:

**word** a;

**int** i, j;

```
a=0x3ff;  
a=50000u;  
i=15;  
j=-22;
```

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".

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

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 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.

## 5.2.5 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 beeinflussen, in dem man Klammern setzt. Klammern haben die größte Priorität. Möchte man das letzte Beispiel strikt von links nach rechts auswerten:

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

Eine Aufstellung der Prioritäten findet sich in der [Präzedenz Tabelle](#).

### 5.2.5.1 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
+	<b>Addition</b>	2+1	3
		3.2 + 4	7.2
-	<b>Subtraktion</b>	2 - 3	-1
		22 - 1.1e1	11
*	<b>Multiplikation</b>	5 * 4	20
/	<b>Division</b>	7 / 2	3
		7.0 / 2	3.5
%	<b>Modulo</b>	15 % 4	3
		17 % 2	1
-	<b>negatives Vorzeichen</b>	-(2+2)	-4

### 5.2.5.2 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

### 5.2.5.3 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

### 5.2.5.4 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

### 5.2.5.5 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

### 5.2.5.6 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

### 5.2.6 Kontrollstrukturen

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



### 5.2.6.1 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++;
```

### 5.2.6.2 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
```

### 5.2.6.3 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;
}
```

### 5.2.6.4 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;
}
```

### 5.2.6.5 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++;  
}
```

#### 5.2.6.6 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.

### 5.2.6.7 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
}
```

## 5.2.7 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 Klammer nach dem Funktionsnamen die Parameter für die Funktion durch Komma getrennt übergeben. Man gibt ähnlich wie in der Variablendeklaration erst den Datentyp und danach den Parameternamen an. Will man keinen Parameter übergeben, schreibt man **void** in die runden Klammern. Ein Beispiel:

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

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

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

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

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

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

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

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

## Rückgabeparameter

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

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

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

## Referenzen

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

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

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

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

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

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

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



## 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");
}
```

## 5.2.8 Tabellen

### 5.2.8.1 Operator Präzedenz

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

### 5.2.8.2 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

### 5.2.8.3 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

<b>for</b>	<b>goto</b>	<b>if</b>	<b>int</b>	<b>return</b>
<b>signed</b>	<b>static</b>	<b>switch</b>	<b>true</b>	<b>unsigned</b>
<b>void</b>	<b>while</b>	<b>word</b>		

## 5.3 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 "Demoprogramme" 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.

### 5.3.1 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.

### 5.3.2 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. Deklaration von Array [Variablen](#).

## 5.3.3 Datentypen

Werte haben immer einen bestimmten Datentyp. Die Integerwerte (ganzzahlige Werte) haben in BASIC einen 8 oder 16 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
Word	Nein	0 ... 65535	16
Single	Ja	±1.175e-38 to ±3.402e38	32

## 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.

### 5.3.4 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. 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, dass 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.

Beispiele:

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

```
a=&H3ff
a=50000u
i=15
j=-22
a=0x3ff
```

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 Beispielformatdefinition:

```
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 **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".

## 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 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"
```

```
Str_StrCopy(str_array,Single_str,40) // kopiert Single_str in den zweiten String v
```

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 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.

## 5.3.5 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 beeinflussen, in dem man Klammern setzt. Klammern haben die größte Priorität. Möchte man das letzte Beispiel strikt von links nach rechts auswerten:

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

Eine Aufstellung der Prioritäten findet sich in der [Präzedenz Tabelle](#).

### 5.3.5.1 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
		3.2 + 4	7.2
-	Subtraktion	2 - 3	-1
		22 - 1.1e1	11
*	Multiplikation	5 * 4	20
/	Division	7 / 2	3

		7.0 / 2	3.5
Mod	<b>Modulo</b>	15 Mod 4 17 Mod 2	3 1
-	<b>negatives Vorzeichen</b>	-(2+2)	-4

### 5.3.5.2 Bitoperatoren

Bitoperatoren sind nur für Integer Datentypen erlaubt.

Operator	Erklärung	Beispiel	Ergebnis
<b>And</b>	<b>Und</b>	&H0f And 3 &Hf0 And &H0f	3 0
<b>Or</b>	<b>Oder</b>	1 Or 3 &Hf0 Or &H0f	3 &Hff
<b>Xor</b>	<b>exclusives Oder</b>	&Hff Xor &H0f &Hf0 Xor &H0f	&Hf0 &Hff
<b>Not</b>	<b>Bitinvertierung</b>	Not &Hff Not &Hf0	0 &H0f

### 5.3.5.3 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
<b>&lt;&lt;</b>	<b>um ein Bit nach links schieben</b>	1 << 2 3 << 3	4 24
<b>&gt;&gt;</b>	<b>um ein Bit nach rechts schieben</b>	&Hff >> 6 16 >> 2	3 4

### 5.3.5.4 In- Dekrement Operatoren\_2

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.

### 5.3.5.5 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

### 5.3.6 Kontrollstrukturen

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

#### 5.3.6.1 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

**5.3.6.2 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
```

### 5.3.6.3 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
End For
```

```

    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

```

#### 5.3.6.4 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

```



### 5.3.6.5 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
```

### 5.3.6.6 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 <b>gleich</b> Konstante
< Konstante	Ausdruck <b>kleiner</b> Konstante
<= Konstante	Ausdruck <b>kleiner gleich</b> Konstante
> Konstante	Ausdruck <b>größer</b> Konstante
>= Konstante	Ausdruck <b>größer gleich</b> Konstante
<> Konstante	Ausdruck <b>ungleich</b> 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.

### 5.3.7 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!

## 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

```

## 5.3.8 Tabellen

### 5.3.8.1 Operator Präzedenz

Rang	Operator
10	( )
9	- (negatives Vorzeichen)
8	* /

7	Mod
6	+ -
5	<< >>
4	= <> < <= > >=
3	Not
2	And
1	Or Xor

### 5.3.8.2 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

### 5.3.8.3 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			

## 5.4 Assembler

Mit IDE Version 2.0 wurde die Möglichkeit eröffnet auch Assembler Routinen in ein Projekt einzubinden. Als Assembler wird der Open Source Assembler AVRA eingesetzt. Die Quellen des Assemblers sind im Installationsverzeichnis "GNU" zu finden. Man kann von CompactC und Basic Assembler Routinen 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.

### 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.

### 5.4.1 Ein Beispiel

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

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

    MOVW R28, R10    ; move parameter stack pointer into Y
    LDD R26, Y+5     ; load parameter "a" into X (R26)
    LDD R27, Y+6

    LDD R30, Y+0     ; load byte parameter "c" into Z (R30)
    CLR R31          ; hi byte zero because parameter is byte

    ADD R26, R30     ; add Z to X
    ADC R27, R31

    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 X on stack
    ST Z, R27

    ret
#endif
```

## 5.4.2 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 Y (R28,R29) kopiert, und dann indirekt über Y die beiden Parameter a und c geladen.

```
; example for accessing and returning parameter
; we have int proc2(int a, float b, byte c);
MOVW R28, R10 ; move parameter stack pointer into Y
LDD R26, Y+5 ; load parameter "a" into X (R26)
LDD R27, Y+6

LDD R30, Y+0 ; load byte parameter "c" into Z (R30)
CLR R31 ; hi byte zero because parameter is byte
```

Man hat jetzt die beiden Parameter a und c in den Registerpaaren X und Z. Nun kann man die Zahlen addieren.

```
ADD R26, R30 ; add Z to X
ADC R27, R31
```

## 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.

```
; 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 X on stack
ST    Z, R27

```

### 5.4.3 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.
- 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 sichernden 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.

## 5.5 ASCII Tabelle

ASCII Tabelle				
CHA	DEC	HEX	BIN	Description
R				

<b>NUL</b>	000	000	00000000	Null Character
<b>SOH</b>	001	001	00000001	Start of Header
<b>STX</b>	002	002	00000010	Start of Text
<b>ETX</b>	003	003	00000011	End of Text
<b>EOT</b>	004	004	00000100	End of Transmission
<b>ENQ</b>	005	005	00000101	Enquiry
<b>ACK</b>	006	006	00000110	Acknowledgment
<b>BEL</b>	007	007	00000111	Bell
<b>BS</b>	008	008	00001000	Backspace
<b>HAT</b>	009	009	00001001	Horizontal TAB
<b>LF</b>	010	00A	00001010	Line Feed
<b>VT</b>	011	00B	00001011	Vertical TAB
<b>FF</b>	012	00C	00001100	Form Feed
<b>CR</b>	013	00D	00001101	Carriage Return
<b>SO</b>	014	00E	00001110	Shift Out
<b>SI</b>	015	00F	00001111	Shift In
<b>DLE</b>	016	010	00010000	Data Link Escape
<b>DC1</b>	017	011	00010001	Device Control 1
<b>DC2</b>	018	012	00010010	Device Control 2
<b>DC3</b>	019	013	00010011	Device Control 3
<b>DC4</b>	020	014	00010100	Device Control 4
<b>NAK</b>	021	015	00010101	Negative Acknowledgment
<b>SYN</b>	022	016	00010110	Synchronous Idle
<b>ETB</b>	023	017	00010111	End of Transmission Block
<b>CAN</b>	024	018	00011000	Cancel
<b>EM</b>	025	019	00011001	End of Medium
<b>SUB</b>	026	01A	00011010	Substitute
<b>ESC</b>	027	01B	00011011	Escape
<b>FS</b>	028	01C	00011100	File Separator
<b>GS</b>	029	01D	00011101	Group Separator
<b>RS</b>	030	01E	00011110	Request to Send, Record Separator

<b>US</b>	031	01F	00011111	Unit Separator
<b>SP</b>	032	020	00100000	Space
<b>!</b>	033	021	00100001	Exclamation Mark
<b>“</b>	034	022	00100010	Double Quote
<b>#</b>	035	023	00100011	Number Sign
<b>\$</b>	036	024	00100100	Dollar Sign
<b>%</b>	037	025	00100101	Percent
<b>&amp;</b>	038	026	00100110	Ampersand
<b>‘</b>	039	027	00100111	Single Quote
<b>(</b>	040	028	00101000	Left Opening Parenthesis
<b>)</b>	041	029	00101001	Right Closing Parenthesis
<b>*</b>	042	02A	00101010	Asterisk
<b>+</b>	043	02B	00101011	Plus
<b>,</b>	044	02C	00101100	Comma
<b>-</b>	045	02D	00101101	Minus or Dash
<b>.</b>	046	02E	00101110	Dot

<b>CHA R</b>	<b>DEC</b>	<b>HEX</b>	<b>BIN</b>	<b>Description</b>
<b>/</b>	047	02F	00101111	Forward Slash
<b>0</b>	048	030	00110000	
<b>1</b>	049	031	00110001	
<b>2</b>	050	032	00110010	
<b>3</b>	051	033	00110011	
<b>4</b>	052	034	00110100	
<b>5</b>	053	035	00110101	
<b>6</b>	054	036	00110110	
<b>7</b>	055	037	00110111	
<b>8</b>	056	038	00111000	
<b>9</b>	057	039	00111001	
<b>:</b>	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	

<b>Z</b>	090	05A	01011010	
<b>[</b>	091	05B	01011011	Left Opening Bracket
<b>\</b>	092	05C	01011100	Back Slash
<b>]</b>	093	05D	01011101	Right Closing Bracket
<b>^</b>	094	05E	01011110	Caret

<b>CHAR</b>	<b>DEC</b>	<b>HEX</b>	<b>BIN</b>	<b>Description</b>
<b>_</b>	095	05F	01011111	Underscore
<b>`</b>	096	060	01100000	
<b>a</b>	097	061	01100001	
<b>b</b>	098	062	01100010	
<b>c</b>	099	063	01100011	
<b>d</b>	100	064	01100100	
<b>e</b>	101	065	01100101	
<b>f</b>	102	066	01100110	
<b>g</b>	103	067	01100111	
<b>h</b>	104	068	01101000	
<b>i</b>	105	069	01101001	
<b>j</b>	106	06A	01101010	
<b>k</b>	107	06B	01101011	
<b>l</b>	108	06C	01101100	
<b>m</b>	109	06D	01101101	
<b>n</b>	110	06E	01101110	
<b>o</b>	111	06F	01101111	
<b>p</b>	112	070	01110000	
<b>q</b>	113	071	01110001	
<b>r</b>	114	072	01110010	
<b>s</b>	115	073	01110011	
<b>t</b>	116	074	01110100	
<b>u</b>	117	075	01110101	

<b>v</b>	118	076	01110110	
<b>w</b>	119	077	01110111	
<b>x</b>	120	078	01111000	
<b>y</b>	121	079	01111001	
<b>z</b>	122	07A	01111010	
<b>{</b>	123	07B	01111011	Left Opening Brace
<b> </b>	124	07C	01111100	Vertical Bar
<b>}</b>	125	07D	01111101	Right Closing Brace
<b>~</b>	126	07E	01111110	Tilde
<b>DEL</b>	127	07F	01111111	Delete



# **Kapitel**



## 6 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.

### 6.1 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!

### 6.2 Allgemein

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

#### 6.2.1 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.

➔ 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

## 6.2.2 Sleep

### 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

## 6.3 Analog-Comparator

Der Analog-Comparator ermöglicht, zwei analoge Signale zu vergleichen. Das Ergebnis dieses Vergleichs wird entweder als „0“ oder „1“ zurückgegeben.

### 6.3.1 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:

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

### 6.3.2 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 werden verwendet
// 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);
    }
}

```

## 6.4 Analog-Digital-Wandler

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

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

### Analogeingänge ADC0 ... ADC7, ADC\_BG, ADC\_GND

Als Eingänge für den ADC stehen die Eingänge ADC0 ... ADC7 (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:

$$u = x * \text{Referenzspannung} / 1024$$

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

<b>ADC22x10</b>	Differenzeingänge ADC2, ADC2, Verstärkung 10	; Offsetmessung
<b>ADC23x10</b>	Differenzeingänge ADC2, ADC3, Verstärkung 10	
<b>ADC22x200</b>	Differenzeingänge ADC2, ADC2, Verstärkung 200	; Offsetmessung
<b>ADC23x200</b>	Differenzeingänge ADC2, ADC3, Verstärkung 200	
<b>ADC20x1</b>	Differenzeingänge ADC2, ADC0, Verstärkung 1	
<b>ADC21x1</b>	Differenzeingänge ADC2, ADC1, Verstärkung 1	
<b>ADC22x1</b>	Differenzeingänge ADC2, ADC2, Verstärkung 1	; Offsetmessung
<b>ADC23x1</b>	Differenzeingänge ADC2, ADC3, Verstärkung 1	
<b>ADC24x1</b>	Differenzeingänge ADC2, ADC4, Verstärkung 1	
<b>ADC25x1</b>	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:

$$u = x * \text{Referenzspannung} / 512 / V$$

### 6.4.1 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

### 6.4.2 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

## 6.4.3 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

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

Für den Standort von PAD3 siehe Jumper Application Board [M32](#) oder [M128](#).

## 6.4.5 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	0xC0	2,56V interne Referenzspannung
ADC_VREF_VCC	0x40	Versorgungsspannung (5V)
ADC_VREF_EXT	0x00	externe Referenzspannung an PAD3

Für den Standort von PAD3 siehe Jumper Application Board [M32](#) oder [M128](#).

### 6.4.6 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 initialisiert wurde. Liegt das Messergebnis bereit, wird ein ADC\_Interrupt ausgelöst.

#### Parameter

Keine

## 6.5 Clock

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.

### 6.5.1 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

## 6.5.2 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

## 6.5.3 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](#).

### Parameter

<u>hour</u>	Stunde
<u>min</u>	Minute
<u>sec</u>	Sekunde
<u>corr</u>	Korrekturfaktor

## 6.6 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) 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 wird die RTC mit der neuen Zeit gesetzt, und läuft dann selbständig weiter. Die RTC als auch die DCF77-Dekodierung ist über einen 10ms Interrupt gesteuert. Diese Zeitbasis ist von der Quarzfrequenz des Controllers abgeleitet. DCF\_Mode steuert den Ablauf für die DCF77-Zeitaufnahme.

#### Tabelle DCF-Modi

DCF_Mode	Beschreibung
0	kein DCF77-Betrieb
1	Puls suchen
2	Synchronisation auf Frameanfang

### 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.

## 6.6.1 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

## 6.6.2 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

### 6.6.3 DCF\_PULS

**DCF Funktionen**

---

**Syntax**

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

```
sub DCF_PULS( )
```

**Beschreibung**

[DCF\\_Mode](#) auf 1 schalten ("Puls suchen").

**Parameter**

Keine

### 6.6.4 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

## 6.6.5 DCF\_SYNC

### DCF Funktionen

---

#### Syntax

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

```
Sub DCF_SYNC( )
```

#### Beschreibung

[DCF\\_Mode](#) auf 2 schalten ("Synchronisation auf Frameanfang").

#### Parameter

Keine

## 6.7 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.

### 6.7.1 Msg\_WriteChar

#### Debug Message Funktionen

---

#### Syntax

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

```
Sub Msg_WriteChar(c As Char);
```

#### Beschreibung

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

#### Parameter

`c` das auszugebende Zeichen

### 6.7.2 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

### 6.7.3 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

### 6.7.4 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

## 6.7.5 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

## 6.7.6 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

## 6.8 Direct Access

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!

### 6.8.1 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

### 6.8.2 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

## 6.9 EEPROM

Auf dem C-Control Pro Modul sind **M32:1kB M128:4kB** EEPROM integriert. Diese Bibliotheksfunktionen ermöglichen den Zugriff auf das EEPROM vom Interpreter. 32 Byte des EEPROM Bereichs werden für interne Zwecke benutzt, und sind daher nicht zugreifbar.

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

##### Parameter

pos Position im EEPROM

##### Rückgabewert

der Wert des byte an Position pos im EEPROM

### 6.9.2 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. Die ersten 32 byte sind für das C-Control Pro OS reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu. Der Wert von pos ist eine Byte Position im EEPROM. Dies sollte bei word oder Fließkommazugriffen beachtet werden.

**Parameter**

pos Byte Position im EEPROM

**Rückgabewert**

der Wert des word an Position pos im EEPROM

### 6.9.3 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. Die ersten 32 byte sind für das C-Control Pro OS reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu. Der Wert von pos ist eine Byte Position im EEPROM. Dies sollte bei word oder Fließkommazugriffen beachtet werden.

**Parameter**

pos Byte Position im EEPROM

**Rückgabewert**

der Fließkommawert an Position pos im EEPROM

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

**Parameter**

pos Position im EEPROM

val der ins EEPROM zu schreibende Wert

## 6.9.5 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. Die ersten 32 byte sind für das C-Control Pro OS reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu. Der Wert von pos ist eine Byte Position im EEPROM. Dies sollte bei word oder Fließkommazugriffen beachtet werden.

**Parameter**

pos Byte Position im EEPROM

val der ins EEPROM zu schreibende Wert

## 6.9.6 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. Die ersten 32 byte sind für das C-Control Pro OS reserviert. Ein Wert für pos von 0 und größer greift deshalb auf byte 32 und aufwärts im EEPROM zu. Der Wert von pos ist eine Byte Position im EEPROM. Dies sollte bei word oder Fließkommazugriffen beachtet werden.

**Parameter**

pos Byte Position im EEPROM  
val der ins EEPROM zu schreibende Wert

## 6.10 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.

### 6.10.1 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
```

### 6.10.2 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

### 6.10.3 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

### 6.10.4 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

### 6.10.5 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

## 6.10.6 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

## 6.10.7 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

### 6.10.8 I2C Status Tabelle

Tabelle: **Status Codes Master Transmitter Mode**

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

Tabelle: **Status Codes Master Receiver Mode**

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

### 6.10.9 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);
}

```

## 6.11 Interrupt

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

**Tabelle Interrupts**

Interrupt Name	Beschreibung
INT_0	externer Interrupt0
INT_1	externer Interrupt1
INT_2	externer Interrupt2
INT_3	externer Interrupt3 (nur Mega128)
INT_4	externer Interrupt4 (nur Mega128)
INT_5	externer Interrupt5 (nur Mega128)
INT_6	externer Interrupt6 (nur Mega128)
INT_7	externer Interrupt7 (nur Mega128)
INT_TIM1CAPT	Timer1 Capture
INT_TIM1CMPA	Timer1 CompareA
INT_TIM1CMPB	Timer1 CompareB
INT_TIM1OVF	Timer1 Overflow
INT_TIM0COMP	Timer0 Compare
INT_TIM0OVF	Timer0 Overflow

INT_ANA_COMP	Analog Comparator
INT_ADC	ADC
INT_TIM2COMP	Timer2 Compare
INT_TIM2OVF	Timer2 Overflow
INT_TIM3CAPT	Timer3 Capture (nur Mega128)
INT_TIM3CMPA	Timer3 CompareA (nur Mega128)
INT_TIM3CMPB	Timer3 CompareB (nur Mega128)
INT_TIM3CMPC	Timer3 CompareC (nur Mega128)
INT_TIM3OVF	Timer3 Overflow (nur Mega128)

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

➔ Ein Signal auf INT\_0 beim Einschalten des C-Control Pro Moduls kann das [Autostartverhalten](#) stören. Nach der Pinzuordnung von [M32](#) und [M128](#) liegt der INT\_0 auf dem gleichen Pin wie der SW1. Wird der SW1 beim Einschalten des Moduls gedrückt, führt dies zur Aktivierung des seriellen Bootloader Modus, und das Programm wird nicht automatisch gestartet.

### 6.11.1 Ext\_IntEnable

#### Interrupt 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. Ein Signal auf **Mega32**:IRQ 0 **Mega128**:IRQ 4 kann zu [Autostart](#) Problemen führen.

➔ 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\(\)](#).

➔ Der IRQ2 des **Mega32** kann nur flankengesteuert arbeiten. Siehe der andere Mode Parameter.

#### Parameter

IRQ Nummer des freizuschaltenden Interrupts **Mega32** (0-2) bzw. **Mega128** (0-7)

Mode Parameter:

- 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

Mode Parameter für **Mega32** und IRQ2:

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

### 6.11.2 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) bzw. **Mega128** (0-7)

### 6.11.3 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.

### 6.11.4 Irq\_SetVect

Interrupt Funktionen [Beispiel](#)

#### Syntax

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

```
Sub Irq_SetVect(irqnr As Byte,vect As Single)
```

#### Beschreibung

Setzt die aufzurufende Interrupt Funktion für einen bestimmten Interrupt. Am Ende der Interruptroutine muß die Funktion [Irq\\_GetCount\(\)](#) aufgerufen werden, ansonsten wird ununterbrochen in die Interrupt Funktion gesprungen.

#### Parameter

irqnr spezifiziert den Typ des Interrupts (siehe [Tabelle](#))

vect ist der Name der aufzurufenden Interrupt Funktion

#### Anmerkung

Der Fließkomma Datentyp mag unpassend erscheinen, aber er wird intern als 4 Byte Wert behandelt. Ein Funktionszeiger muß seit der Mega128 Unterstützung mehr als 16 Bit lang sein.

### 6.11.5 IRQ Beispiel

#### Beispiel: Verwendung von Interrupt Routinen

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

```
int cnt;
```

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

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

```
void main(void)
{
    cnt=0;

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

}

## 6.12 Keyboard

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

### 6.12.1 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

### 6.12.2 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

### 6.12.3 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

## 6.13 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.

### 6.13.1 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

### 6.13.2 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

### 6.13.3 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

### 6.13.4 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 <u>pos</u>	Position im Display
0x00-0x07	0-7 in der 1. Zeile
0x40-0x47	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 <u>pos</u>	Position im Display
0x00-0x1f	0-31 in der 1. Zeile
0x40-0x5f	0-31 in der 2. Zeile
0x20-0x3f	0-31 in der 3. Zeile
0x60-0x6f	0-31 in der 4. Zeile

## 6.13.5 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

## 6.13.6 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

## 6.13.7 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

## 6.13.8 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

### 6.13.9 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

### 6.13.10 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

### 6.13.11 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

## 6.13.12 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

## 6.13.13 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

### 6.13.14 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

### 6.13.15 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

## 6.14 Mathematik

Im folgenden sind die mathematischen Funktionen aufgeführt, die der C-Control Pro 128 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.

### 6.14.1 acos

#### Mathematische 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.

### 6.14.2 asin

#### Mathematische 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.

### 6.14.3 atan

#### Mathematische 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.

### 6.14.4 ceil

#### Mathematische 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

### 6.14.5 cos

#### Mathematische 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)

### 6.14.6 exp

#### Mathematische Funktionen ---

##### Syntax

```
float exp(float val);
```

```
Sub exp(val As Single) As Single
```

##### Beschreibung

Die Funktion  $e^{\text{val}}$  wird berechnet

##### Parameter

val Exponent

##### Rückgabewert

Ergebnis der Funktion

### 6.14.7 fabs

#### Mathematische 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

## 6.14.8 floor

### Mathematische 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

## 6.14.9 ldexp

### Mathematische 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

**6.14.10 ln****Mathematische 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

**6.14.11 log****Mathematische 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

### 6.14.12 pow

#### Mathematische 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

### 6.14.13 round

#### Mathematische 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

### 6.14.14 sin

#### Mathematische 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)

## 6.14.15 sqrt

### Mathematische 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

## 6.14.16 tan

### Mathematische 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.

## 6.15 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.

### 6.15.1 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

### 6.15.2 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 Tabelle)

**Portbits Tabelle**

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
ab hier nur Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

**6.15.3 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

## 6.15.4 Onewire Beispiel

### CompactC

```
// Beispielprogramm um den DS18S20 Temperatur Sensor von Dallas Maxim zu lesen
void main(void)
{
    char text[40];
    int ret, i;
    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 den DS18S20 Temperatur 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
```

## 6.16 Port

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-/Ausgabeports 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. Eine Tabelle der Pinzuordnung von [M32](#) und [M128](#) ist in der Dokumentation..



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

➔ 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.

### 6.16.1 Port\_DataDir

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 = 0x02`, dann wird der Pin 2 des Atmel Mega (gleich PortB.1 - siehe Pinzuordnung von [M32](#) und [M128](#)) auf Ausgang konfiguriert.

#### Parameter

port Portnummer (siehe Tabelle)

val Ausgabe byte

#### Portnummern Tabelle

Definition	Wert
PortA	0
PortB	1
PortC	2
PortD	3
PortE ( <a href="#">Mega128</a> )	4
PortF ( <a href="#">Mega128</a> )	5
PortG ( <a href="#">Mega128</a> )	6

## 6.16.2 Port\_DataDirBit

### 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.

### Parameter

portbit Bitnummer des Ports (siehe Tabelle)

val 0=Eingang, 1= Ausgang

### Portbits Tabelle

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
ab hier nur Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

### 6.16.3 Port\_Read

#### 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. (Für die Abbildung zwischen Portbits und den Pins des Atmel Mega Chips siehe Pinzuordnung von [M32](#) und [M128](#)).

#### Parameter

port Portnummer (siehe Tabelle)

#### Rückgabewert

Wert des Ports

#### Portnummern Tabelle

Definition	Wert
PortA	0
PortB	1
PortC	2
PortD	3
PortE ( <a href="#">Mega128</a> )	4
PortF ( <a href="#">Mega128</a> )	5
PortG ( <a href="#">Mega128</a> )	6

### 6.16.4 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 [M32](#) und [M128](#).)

➔ 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.

#### Parameter

portbit Bitnummer des Ports (siehe Tabelle)

#### Rückgabewert

Bitwert des Ports (0 oder 1)

#### Portbits Tabelle

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
ab hier nur Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

### 6.16.5 Port\_Toggle

#### 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 Tabelle)

### Portnummern Tabelle

Definition	Wert
PortA	0
PortB	1
PortC	2
PortD	3
PortE ( <a href="#">Mega128</a> )	4
PortF ( <a href="#">Mega128</a> )	5
PortG ( <a href="#">Mega128</a> )	6

## 6.16.6 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. Ist ein Pin auf Eingang geschaltet, so wird der interne Pullup Widerstand umgeschaltet. (Für die Abbildung zwischen Portbits und den Pins des Atmel Mega Chips siehe Pinzuordnung von [M32](#) und [M128](#)).

➡ 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 Tabelle)

**Portbits Tabelle**

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
ab hier nur Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

**6.16.7 Port\_Write**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](#)).

**Parameter**

port Portnummer (siehe Tabelle)

val Ausgabe byte

**Portnummern Tabelle**

Definition	Wert
PortA	0
PortB	1
PortC	2
PortD	3
PortE (Mega128)	4
PortF (Mega128)	5
PortG (Mega128)	6

### 6.16.8 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 [M32](#) und [M128](#)).

➔ 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.

#### Parameter

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

#### Portbits Tabelle

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...

PortC.7	23
PortD.0	24
...	...
PortD.7	31
ab hier nur Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

### 6.16.9 Port Beispiel

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

```
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);
    }
}
```

## 6.17 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

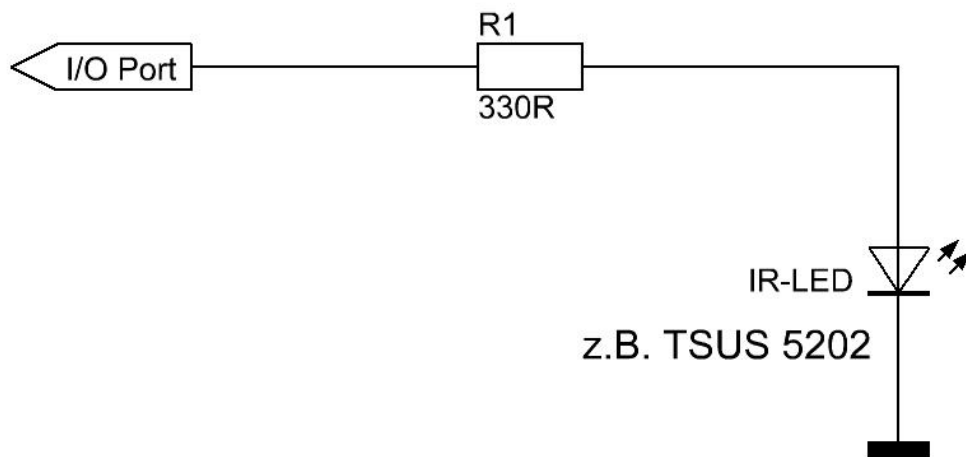


Die Startbits dienen dem Infrarotempfänger zur Synchronisation mit der Übertragung, sowie dazu, seine Verstärkungsregelung auf das Signal einzustellen. Das Togglebit ändert seinen Wert bei jedem Tastendruck. Dadurch kann man das lange Drücken einer Taste (und damit das wiederholte Senden eines Befehls) 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?

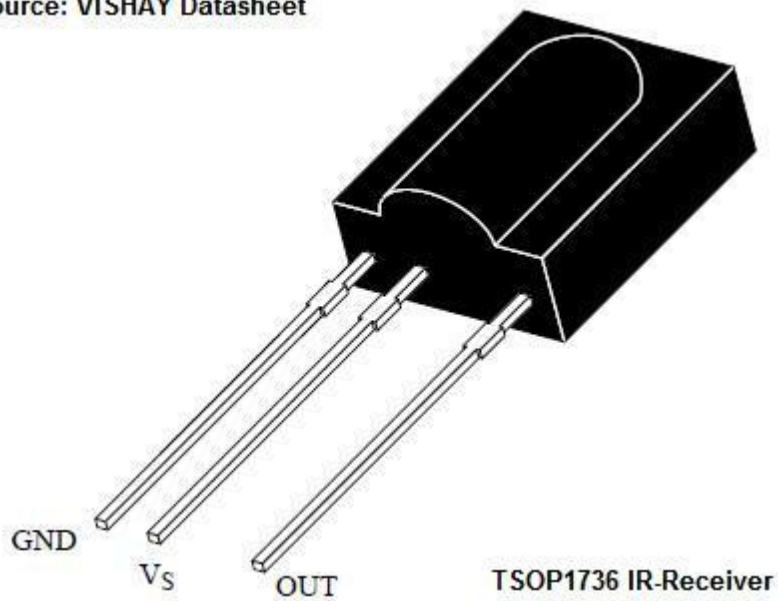
In der C-Control Pro werden an den Pin an den die IR-Diode angeschlossen und konfiguriert wurde, 36 KHz als Trägerfrequenz erzeugt. Die Sendepulse sind  $6,9444 \mu\text{s}$  lang. Zwischen den Sendepulsen ist jeweils eine Pause von  $20,8332 \mu\text{s}$ . Für ein Bit mit dem Wert "1" wird der Sendergenerator für  $889 \mu\text{s}$  ausgeschaltet und anschließend für  $889 \mu\text{s}$  eingeschaltet das entspricht 32 IR-Impulse. Ein "0"-Bit beginnt dagegen mit  $889 \mu\text{s}$  Sendezeit (32 IR-Impulse), gefolgt von  $889 \mu\text{s}$  Pause. Folglich dauert ein Bit  $1,778 \text{ ms}$  und die Übertragung eines kompletten 14-Bit Datenworts  $24,889 \text{ ms}$ . Falls man die Taste auf der Fernbedienung gedrückt hält, wird das Datenwort alle  $113,778 \text{ ms}$  wiederholt dies entspricht der Dauer von 64 Bit.

### Anschluß an C-Control Pro (Sender Diode)

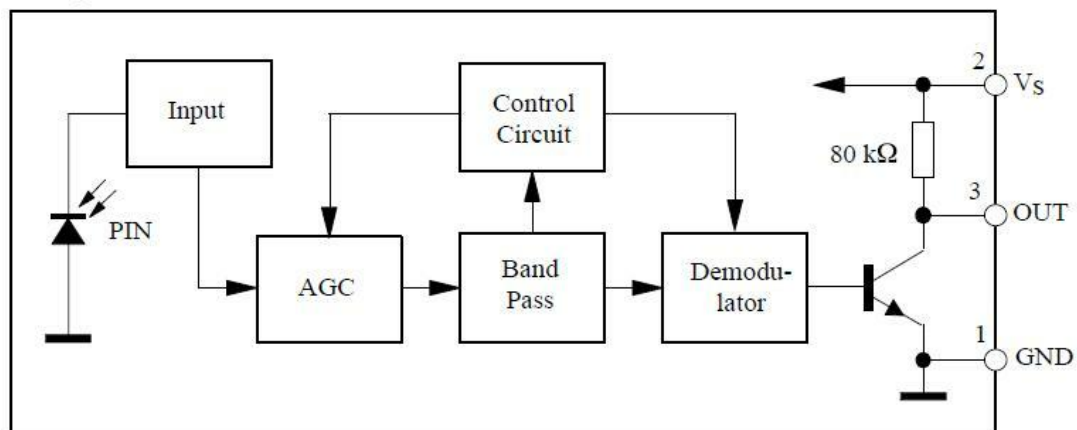


Anschluß an C-Control Pro (Empfänger)

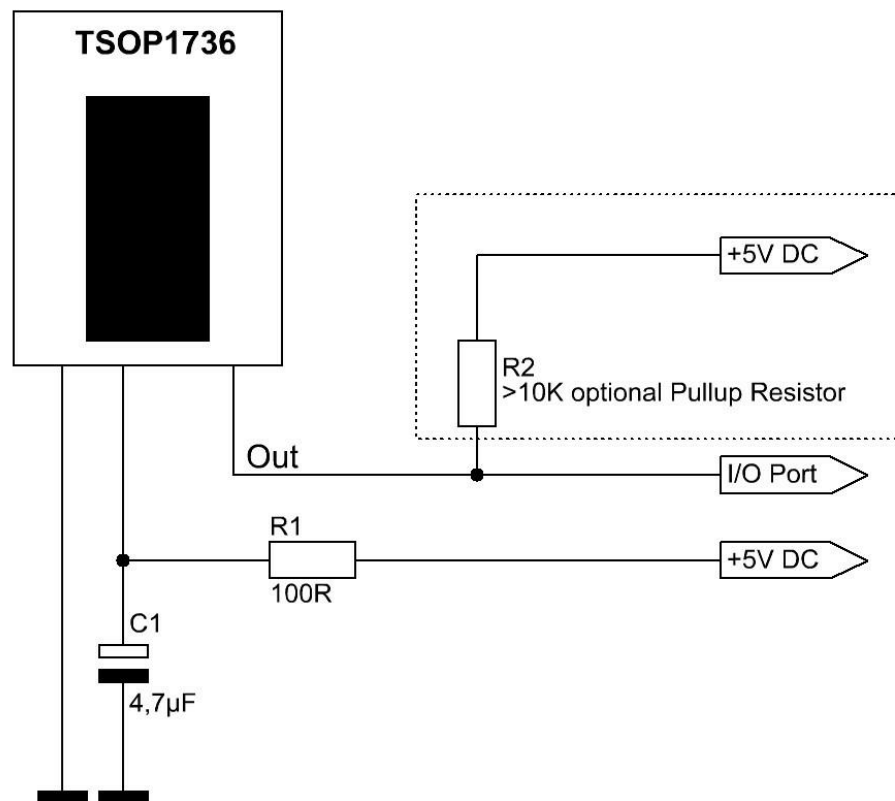
Source: VISHAY Datasheet



Pinbelegung des TSOP1736 IR-Empfängers



Technologischer Aufbau des Empfängers



Externe Beschaltung des Empfängers zum Anschluss an die C-Control Pro

### 6.17.1 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 Tabelle)

**Portbits Tabelle**

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
ab hier nur Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

**6.17.2 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

### 6.17.3 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.

#### Parameter

data     auszugebendes RC5 Datenwort

## 6.18 RS232

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

### 6.18.1 Divider

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.

Laut dem Atmel Prozessor Handbuch wird folgende Formel angewendet um einen divider für eine Baudrate zu ermitteln:

$$\text{divider} = (14745600 / \text{Baudrate} / 16) - 1$$

**Beispiel:**  $15 = (14745600 / 57600 / 16) - 1$

### DoubleClick Modus

Wenn man das High-Bit setzt, wird der DoubleClock Modus eingeschaltet. Man muß dann den doppelten Wert als divider eintragen. Für 57600 Baud kann man z.B. statt 0x0f (dezimal 15) auch 0x801e benutzen. Für MIDI (31250 Baud) bekäme man einen divider =  $(14745600 / 31250 / 16) - 1 = 28,49$ . Setzt man nun den DoubleClock Modus, so kommt man viel genauer an den richtigen Wert: 0x8039

**Tabelle divider Definitionen:**

<b>divider</b>	<b>Definition</b>	<b>Baudrate</b>
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
0x8039	SR_BDMIDI	31250bps
23	SR_BD38400	38400bps
15	SR_BD57600	57600bps
11	SR_BD76800	76800bps
7	SR_BD115200	115200bps
3	SR_BD230400	230400bps

## 6.18.2 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..)

### 6.18.3 Serial\_Init

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.

➡ 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 (=0x803a) liegt jetzt sehr nahe an 31250baud. Beispiel für 19200baud: Der Teiler für 19200baud ist 0x002f. Für den DoubleClock Modus verdoppelt man nun den Teiler (= 0x005e). Nun das Hi-Bit setzen, und man kann anstatt 0x2f auch 0x805e 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

### 6.18.4 Serial\_Init\_IRQ

Serielle Funktionen [Beispiel](#)

#### Syntax

```
void Serial_Init_IRQ(byte serport,byte ramaddr[],byte recvlen,byte sendlen,byte par)

Sub Serial_Init_IRQ(serport As Byte,ByRef ramaddr As Byte,recvlen As Byte,sendlen As Byte,
par As Byte,div As Byte)
```

#### 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. Die Größe des Puffers muß die **Größe des Empfangspuffers plus die Größe des Sendepuffers plus 6** sein (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.

➔ 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 (=0x803a) liegt jetzt sehr nahe an 31250baud. Beispiel für 19200baud: Der Teiler für 19200baud ist 0x002f. Für den DoubleClock Modus verdoppelt man nun den Teiler (= 0x005e). Nun das Hi-Bit setzen, und man kann anstatt 0x2f auch 0x805e für 19200baud benutzen .

➔ Wenn im seriellen Interrupt Modus gearbeitet wird, immer [Serial\\_ReadEx\(\)](#) benutzen. Serial\_Read()



funktioniert nur im normalen (polled) Modus.

#### Parameter

<u>serport</u>	Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)
<u>ramaddr</u>	Adresse des Puffers
<u>recvlen</u>	Größe des Empfangspuffers
<u>sendlen</u>	Größe des Sendepuffers
<u>par</u>	Schnittstellenparameter (siehe Tabelle)
<u>div</u>	Baudrateninitialisierung mittels Teiler (siehe <a href="#">Tabelle</a> )

#### 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

## 6.18.5 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

## 6.18.6 Serial\_Read

**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.

#### Parameter

serport   Schnittstellennummer (0 = 1.serielle, 1 = 2.serielle etc..)

#### Rückgabewert

empfangenes byte aus der seriellen Schnittstelle

## 6.18.7 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 (0x100)** 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 (0x100) kein Zeichen in der Schnittstelle

### 6.18.8 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

### 6.18.9 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

### 6.18.10 Serial Beispiel

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

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

    while(str[i]) Serial_Write(0,str[i++]); // Gib den String aus
}
```

### 6.18.11 Serial Beispiel (IRQ)

```
// 35 byte Sende + Empfangspuffer + 6 byte interne FIFO Verwaltung
byte buffer[41]; // Array deklarier

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

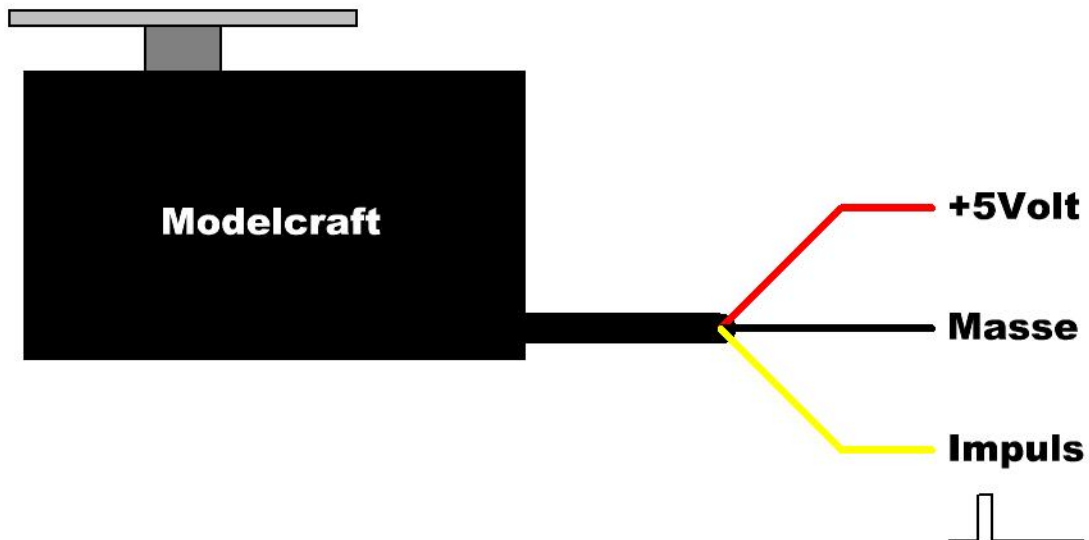
    str="test";
    i=0;
    // Initialisiere Schnittstelle mit 19200baud, 8 Bit, 1 Stop Bit, keine Parität
    // 20 byte Empfangspuffer - 15 byte Sendepuffer
    Serial_Init_IRQ(0,buffer,20,15,SR_8BIT|SR_1STOP|SR_NO_PAR,SR_BD19200);

    while(str[i]) Serial_Write(0,str[i++]); // Gib den String aus
}
```

## 6.19 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.

### 6.19.1 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, ti
```

#### 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. Die Größe beträgt `servo_cnt * 3`. Möchte man also 10 Servos betreiben, so sollte man ein **byte** Array von 30 Bytes reservieren.

➔ Für die Servoansteuerung wird ein 16-Bit Timer benötigt. Dies ist entweder Timer 1 oder Timer 3 (bei Mega128). 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 (0=Timer 1, 1=Timer 3 **nur Mega128**)

## 6.19.2 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 Atmel Mega Chips siehe Pinzuordnung von [M32](#) und [M128](#)).

➔ 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 Tabelle)  
pos      Pulslänge zur Steuerung des Servoarms in µsec (500 - 2500)

### Portbits Tabelle

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
<b>ab hier nur Mega128</b>	
PortE.0	32
...	...
PortE.7	39

PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

### 6.19.3 Servo Beispiel

```
// Ansteuerung von 3 Servos und beenden nach 10 Sek.
void main(void)
{
    byte servo_var[30]; // Servo interne Variablen

    // 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);
}
```

## 6.20 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. Wird mit dem Application Board gearbeitet, wird der SPI Bus genutzt um Daten mit dem Mega8 Controller auszutauschen. In eigenen Anwendungen kann der Benutzer die SPI Funktionen zum Datentransfer nutzen.

### 6.20.1 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 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

## 6.20.2 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	$f_{Osc} / 128$



### 6.20.3 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

### 6.20.4 SPI\_ReadBuf

#### SPI Funktionen

---

##### Syntax

```
void SPI_ReadBuf(byte buf[], byte length);
```

```
Sub SPI_ReadBuf(ByRef buf As Byte, length As Byte)
```

##### 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

### 6.20.5 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

### 6.20.6 SPI\_WriteBuf

#### SPI Funktionen

---

##### Syntax

```
void SPI_WriteBuf(byte buf[], byte length);  
  
Sub SPI_WriteBuf(ByRef buf As Byte, length As Byte)
```

##### Beschreibung

Ein Anzahl von Bytes wird auf die SPI Schnittstelle geschrieben.

#### Parameter

buf        Zeiger auf byte array  
length    Anzahl der auszugebenden bytes

## 6.21 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.

### 6.21.1 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

## 6.21.2 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** (0xffff), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird der Source String hinter den Destination String kopiert.

## 6.21.3 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

## 6.21.4 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

## 6.21.5 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

## 6.21.6 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).

## 6.21.7 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 % 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.

**%[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](#).

Formatierung	Beschreibung
%[width]d	Integerzahl
%[width][.prec]u	vorzeichenloser Integer
%[width]x	Hexzahl
%[width][.prec]f	Fließkommazahl
%[width]s	String

#### Parameter

str      Zeiger auf Zielzeichenkette  
format   Zeiger auf Format String

## 6.21.8 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.

## 6.21.9 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.

## 6.21.10 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.

## 6.21.11 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

## 6.21.12 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

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

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

## 6.21.13 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** (0xffff), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird die Integer Zahl an den Text String angehängt.

### 6.21.14 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 (0xffff), so wird als offset die Länge des Zielstrings angenommen. In diesem Fall wird die Integer Zahl an den Text String angehängt.

## 6.21.15 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
```

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

Die Priorität eines Threads kann beeinflusst werden, in dem man ändert, wie viele Bytecodes ein Thread bis zum nächsten Threadwechsel ausführen darf (siehe [Threadoptionen](#)). Je kleiner die

Anzahl der Zyklen bis zum Wechsel, desto geringer die Priorität des Threads. Die Ausführungszeit eines Bytecodes ist im Mittel 7-9 µsec. Bei einzelnen Bytecode Befehlen dauert es jedoch länger, z. B. Floatingpoint Operationen.

➔ Auch interne Interpreterfunktionen gelten als ein Zyklus. Da z.B. [Serial\\_Read](#) wartet, bis ein Zeichen von der seriellen Schnittstelle ankommt, kann in Ausnahmefällen ein Zyklus sehr lange dauern.

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

➔ Damit zusätzliche Threads gestartet werden können muß "Multithreading" in den [Projektoptionen](#) eingeschaltet werden, und die Parameter für die weiteren Threads in den [Threadoptionen](#) auf korrekte Wert gesetzt werden.

➔ 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.

## 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
<i>aktiv</i>	Der Thread wird momentan abgearbeitet

<i>inaktiv</i>	Kann nach einem Threadwechsel wieder aktiviert werden
<i>schlafend</i>	Wird nach einer Anzahl von Ticks wieder auf "inaktiv" gesetzt
<i>wartend</i>	Der Thread wartet auf ein Signal

### 6.22.1 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\_Cycles() 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

### 6.22.2 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

### 6.22.3 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:

<b>TI_THREADNUM</b>	Nummer des aufrufenden Threads
<b>TI_STACKSIZE</b>	Definierte Stackgröße
<b>TI_CYCLES</b>	Anzahl der auszuführenden Cycles vor einem Threadwechsel

**Rückgabewert**

angeforderter Parameter

### 6.22.4 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

## 6.22.5 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

## 6.22.6 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

## 6.22.7 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

## 6.22.8 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

## 6.22.9 Thread\_Start

### Thread Funktionen

[Beispiel](#)

---

#### Syntax

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

```
Sub Thread_Start(thread As Byte, func As Single)
```

## 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

### Anmerkung

Der Fließkomma Datentyp mag unpassend erscheinen, aber er wird intern als 4 Byte Wert behandelt. Ein Funktionszeiger muß seit der Mega128 Unterstützung mehr als 16 Bit lang sein.

## 6.22.10 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

## 6.22.11 Thread Beispiel

```
// Demoprogramm zum Multithreading  
// Das Programm ist nicht entprellt, ein kurzes Tasten führt daher zu  
// mehrfacher Ausgabe des Strings  
  
void thread1(void)  
{  
    while(true) // Endlosschleife
```



```

    {
        if(!Port_ReadBit(PORT_SW2)) Msg_WriteText(str2); // SW2 gedrückt
    }
}

char str1[12],str2[12];

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

    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

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

    while(true) // Endlosschleife
    {
        if(!Port_ReadBit(PORT_SW1)) Msg_WriteText(str1); // SW1 gedrückt
    }
}

```

## 6.22.12 Thread Beispiel 2

```

// multithread2: Multithreading mit Thread_Delay
// erforderliche Library: IntFunc_Lib.cc

void thread1(void)
{
    while(true)
    {
        Msg_WriteText(str2); Thread_Delay(200);
    }
    // "Thread2" wird ausgegeben.
    // Danach ist der Thread
    // für 200ms "schlafend".

char str1[12],str2[12]; // globale Variablendeklaration

//-----
// Hauptprogramm
//
void main(void)
{
    str1="Thread1"; // Variablendeklaration
    str2="Thread2"; // Variablendeklaration

    Thread_Start(1,thread1); // Funktionsaufruf mit Angabe der
    // Threadnummer.

```

```

while(true)                                // Endlosschleife
{
    Thread_Delay(100); Msg_WriteText(str1);
}                                           // Der Thread ist für 100ms "schlafend".
                                           // Danach wird "Thread1" ausgegeben.

```

## 6.23 Timer

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](#)

### 6.23.1 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();

```

### 6.23.2 Frequenzerzeugung

Zur Frequenzerzeugung können *Timer\_0*, *Timer\_1* und *Timer\_3* folgendermaßen eingesetzt werden:

#### Timer0 (8 Bit)

1. Beispiel:

```
// Rechtecksignal mit  $10 \cdot 1,085 \mu\text{s} = 10,85 \mu\text{s}$  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= $138,9 \mu\text{s} \cdot 100 = 13,9 \text{ms}$ , 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  $125 \cdot 4,34 \mu\text{s} = 1085 \mu\text{s}$  Periode

```
Timer_T1FRQ(125, PS_64);
```

### Timer3 (16 Bit) (nur Mega128)

Beispiel: Frequenzerzeugung mit  $10 \cdot 1,085 \mu\text{s} = 10,85 \mu\text{s}$  Periode und  $2 \cdot 1,085 \mu\text{s} = 2,17 \mu\text{s}$  Phasenverschiebung

```
Timer_T3FRQX(10, 2, PS_8);
```

## 6.23.3 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.

### 6.23.4 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(10,20,10,PS_1024); // Periode: 100*69,44 µs = 6,94 ms
                                   // PulsA: 20*69,44 µs = 1,389 ms
                                   // PulsB: 10*69,44 µs = 694,4 µs
```

### 6.23.5 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 Signalfanke gemessen. Die Periode wird zwischen zwei steigenden Signalfanken 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_TOPWM(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_TlPM(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.

## 6.23.6 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...  
}
```

## 6.23.7 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)

## 6.23.8 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

## 6.23.9 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:

#### Mega32

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
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

#### Mega128

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS0_1 (1)	135,6 ns
PS0_8 (2)	1,085 µs
PS0_32 (3)	4,340 µs
PS0_64 (4)	8,681 µs
PS0_128 (5)	17,36 µs
PS0_256 (6)	34,72 µs
PS0_1024 (7)	138,9 µs

### 6.23.10 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 **0xFF** übergeben.

➔ Beim **Mega128** ist aus Hardwaregründen die Benutzung von *Timer\_0* als Zähler nicht möglich!

##### Rückgabewert

der gemessene Zählerwert

### 6.23.11 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.

##### Parameter

PW Pulsweite

### 6.23.12 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.

#### Parameter

PW Pulsweite

PS Vorteiler

#### Tabelle prescaler:

##### Mega32

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
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

##### Mega128

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

## 6.23.13 Timer\_T0Start

#### Timer Funktionen

#### Syntax

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

```
Sub Timer_T0Start(prescaler As Byte)
```

#### Beschreibung

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

#### Parameter

prescaler Vorteiler (Tabelle [prescaler](#))

### 6.23.14 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

### 6.23.15 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:

#### Mega32

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)

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

**Mega128**

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

**6.23.16 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

**6.23.17 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 Signalfanke 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

## 6.23.18 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:

##### Mega32

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
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

##### Mega128

Vorteiler (prescaler)	Zeitbasis (Dauer eines Ticks)
PS0_1 (1)	135,6 ns
PS0_8 (2)	1,085 µs
PS0_32 (3)	4,340 µs
PS0_64 (4)	8,681 µs
PS0_128 (5)	17,36 µs
PS0_256 (6)	34,72 µs

PS0\_1024 (7)

138,9  $\mu$ s

### 6.23.19 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 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](#))

### 6.23.20 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 **0xFFFF** übergeben.

#### Rückgabewert

der gemessene Zählerwert

### 6.23.21 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](#)).

## 6.23.22 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.

### Parameter

PW0 Pulsweite

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

### 6.23.24 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.

#### Parameter

PW1 Pulsweite

### 6.23.25 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) und PortD.5 (PIN19). **Mega128**: PortB.5 (X1\_3) und PortB.6 (X1\_2).

### Parameter

period Periodendauer  
PW0 Pulsweite  
PS Vorteiler

### Tabelle prescaler:

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

## 6.23.26 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).

### Parameter

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

## 6.23.27 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).

### Parameter

<u>period</u>	Periodendauer
<u>PW0</u>	Pulsweite Kanal A
<u>PW1</u>	Pulsweite Kanal B
<u>PW2</u>	Pulsweite Kanal C
<u>PS</u>	Vorteiler (Tabelle <a href="#">prescaler</a> )

## 6.23.28 Timer\_T1Start

### Timer Funktionen

---

## Syntax

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

```
Sub Timer_T1Start(prescaler As Byte)
```

## Beschreibung

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

### Parameter

<u>prescaler</u>	Vorteiler (Tabelle <a href="#">prescaler</a> )
------------------	--

## 6.23.29 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

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

## 6.23.31 Timer\_T3CNT

### Timer Funktionen

### Syntax

```
void Timer_T3CNT(void);
```

```
Sub Timer_T3CNT()
```

## Beschreibung

Diese Funktion initialisiert den Counter3. Der Counter3 wird bei einer positiven Signalfanke an Eingang PortE.6 (X1\_10) inkrementiert.

### Parameter

Keine

## 6.23.32 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 Signalfanke 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

## 6.23.33 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)	Zeitbasis (Dauer eines Ticks)
PS0_1 (1)	135,6 ns
PS0_8 (2)	1,085 µs
PS0_32 (3)	4,340 µs
PS0_64 (4)	8,681 µs
PS0_128 (5)	17,36 µs
PS0_256 (6)	34,72 µs
PS0_1024 (7)	138,9 µs

### 6.23.34 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](#))

### 6.23.35 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 **0xFFFF** übergeben.

### Rückgabewert

der gemessene Zählerwert

## 6.23.36 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](#)).

## 6.23.37 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.

### Parameter

PW0 Pulsweite

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

### 6.23.39 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.

**Parameter**PW1 Pulsweite**6.23.40 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).

**Parameter**period PeriodendauerPW0 PulsweitePS Vorteiler**Tabelle prescaler:**

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

**6.23.41 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).

**Parameter**

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

**6.23.42 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) .

**Parameter**

period    Periodendauer  
PW0      Pulsweite Kanal A  
PW1      Pulsweite Kanal B  
PW2      Pulsweite Kanal C  
PS        Vorteiler (Tabelle [prescaler](#))

**6.23.43 Timer\_T3Start****Timer Funktionen**

---

**Syntax**

```
void Timer_T3Start(byte prescaler);
```

```
Sub Timer_T3Start(prescaler As Byte)
```

**Beschreibung**

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

**Parameter**

prescaler    Vorteiler (Tabelle [prescaler](#))



### 6.23.44 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

### 6.23.45 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)	Zeitbasis (Dauer eines Ticks)
PS_1 (1)	67,8 ns
PS_8 (2)	542,5 ns
PS_64 (3)	4,34 µs

PS_256 (4)	17,36 µs
PS_1024 (5)	69,44 µs

## 6.23.46 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
}
```

# **Kapitel**



## 7 FAQ

### Probleme

1. Es existiert keine USB Verbindung zum Application Board.
  - Ist der FTDI USB Treiber auf dem PC geladen? Oder erscheint vielleicht beim Einstecken des USB Steckers ein "unbekanntes Gerät" im Hardware Manager?
  - Ist in Optionen->IDE->Schnittstellen der richtige Kommunikationsport eingestellt?
  - Wird eine Windowsversion vor Windows 98 SE ("*Second Edition*") benutzt? Die USB Treiber von Microsoft funktionieren erst ab Win98SE zuverlässig mit USB Geräten.
  - Werden die Ports **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".
2. Die serielle Schnittstelle gibt keine Zeichen aus oder empfängt keine Zeichen.
  - Werden die Ports D.0-D.1 versehentlich in der Software benutzt? (siehe Pinzuordnung von [M32](#) und [M128](#)). Sind die Jumper auf dem Application Board bei diesen Ports auch gesetzt?
3. Das Application Board reagiert nicht auf Kommandos, wenn es seriell angeschlossen ist.
  - Um den Bootloader in den seriellen Modus zu bekommen, muß beim Einschalten des Application Boards der Taster SW1 gedrückt werden. (Jumper für SW1 beachten). Für den seriellen Mode kann **M32**:PortD.2 **M128**:PortE.4 (SW1) auch fest auf GND gelegt werden.
4. 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.
5. Es wurde die Tastenbelegung des Editors "**xyz**" eingestellt, aber manche Tastaturbefehle funktionieren nicht.
  - Die Möglichkeit, die Tastenbelegung eines bestimmten Editors in der IDE einzuschalten ist nur eine Näherung. Manchmal ist es zu aufwendig, die entsprechenden Funktionen des "fremden" Editors zu unterstützen, ein anderes Mal können Tastaturbefehle mit den Keyboard Shortcuts in der IDE kollidieren.
6. 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.

7. 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.

8. 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 <u>pos</u>	Position im Display
0x00-0x1f	0-31 in der 1. Zeile
0x40-0x5f	0-31 in der 2. Zeile
0x20-0x3f	0-31 in der 3. Zeile
0x60-0x6f	0-31 in der 4. Zeile

9. 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.

10. Wo ist auf dem Mega128 Application Board die 2. serielle Schnittstelle?

- Siehe J4 im Kapitel [Jumper Application Board](#) M128.

11. 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.

12. 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.

13. 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.

14. 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.

15. 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 "\Dokumente und Einstellungen\Alle Benutzer\Gemeinsame Dokumente" (XP oder früher) bzw. "\Benutzer\Öffentlich\Öffentliche Dokumente" (Vista) zu finden.

16. 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 zu programmieren.

17. Kann man das C-Control Pro Modul auch mit anderen Compilern programmieren?

- Es existieren mehrere Entwicklungssysteme für die Atmel Mega CPU. Teilweise sind diese Compiler kommerziell oder frei. Ein Beispiel einer freien Entwicklungsumgebung ist der GNU C-Compiler. Mit Hilfe eines AVR ISP Programmers kann man dann mit dem GNU C-Compiler geschriebene Programme auf den Atmel Mega 32 oder 128 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.

# Sachverzeichnis

- - -

-- 102, 123

- # -

#define 90

#endif 90

#ifdef 90

#include 90

- + -

++ 102, 123

- A -

AbsDelay 145

AComp 147

acos 181

ADC\_Disable 149

ADC\_Read 149

ADC\_ReadInt 150

ADC\_Set 150

ADC\_SetInt 151

ADC\_StartInt 152

Addition 101, 122

Analog-Comparator 147

Anweisungen 94, 115

Anweisungsblock 94

ApplicationBoard 23, 32, 40, 50

Arithmetische Operatoren 101, 122

Array 97, 118

Array Fenster 75

ASCII 138

asin 181

Assembler 134

Assembler Beispiel 134

Assembler Datenzugriff 136

Assembler Leitfaden 138

atan 182

Atmel Register 161

Ausdrücke 94, 115

Ausgaben 70

Auto Aktualisieren 73

Autostart 17, 69

- B -

Baudrate 85

bedingte Bewertung 104

Bestückungsplan 35, 52

Bezeichner 94, 115

Bibliotheksverwaltung 60

Bitinvertierung 102, 123

Bitoperatoren 102, 123

Bitschiebe Operatoren 102, 123

Bootloader 17

break 104, 105, 107, 109

Breakpoints 72

byte 96, 117

- C -

Case 107, 128

ceil 182

char 96, 117

Clock\_GetVal 152

Clock\_SetDate 153

Clock\_SetTime 153

Code-Falten 61

COM Port 85

Compilervoreinstellung 81

Conrad 4

continue 104, 105, 109

cos 183

Cosinus 183

CPU Auswahl 59

- D -

Datenbits 85

Datentypen 96, 117

DCF\_FRAME 156

DCF\_INIT 156

DCF\_Lib.cc 154

DCF\_PULS 157

DCF\_RTC.cc 154  
 DCF\_START 157  
 DCF\_SYNC 158  
 DCF77 154  
 Debugger 72  
 default 107  
 DirAcc\_Read 161  
 DirAcc\_Write 161  
 Direct\_Access 161  
 Divider 204  
 Division 101, 122  
 Do 124, 125  
 do while 104  
 Druckvorschau 65

## - E -

Editor Ansicht Erneuern 61  
 Editoreinstellungen 78  
 EEPROM 162, 163, 164  
 EEPROM\_Read 162  
 EEPROM\_ReadFloat 163  
 EEPROM\_ReadWord 162  
 EEPROM\_Write 163  
 EEPROM\_WriteFloat 164  
 EEPROM\_WriteWord 164  
 Einleitung 2  
 else 106, 128  
 email 4  
 Ereigniszähler 233  
 Ersetzen 64  
 exclusives Oder 102, 123  
 Exit 124, 125, 126  
 exp 183  
 Ext 169  
 Ext\_IntDisable 171  
 Ext\_IntEnable 170  
 externes RAM 42, 90

## - F -

fabs 183  
 FAQ 259  
 Fax 4  
 Fehlerkorrekturen 5  
 Fenster 87

Firewall 84  
 Firmware 17  
 float 96, 117  
 floor 184  
 For 105, 126  
 formattierte Zeichenausgabe 220  
 Frequenzerzeugung 233  
 Frequenzmessung 234  
 Funktionen 109, 130  
 Funktionsübersicht 61

## - G -

gleich 103, 124  
 Goto 106, 127  
 GPP 4  
 größer 103, 124  
 größer gleich 103, 124

## - H -

Haltepunkte 72  
 Handhabung 3  
 Hardware Version 71  
 Hilfe 88  
 Historie 5

## - I -

I2C Status Codes 168  
 I2C\_Init 165  
 I2C\_Read\_ACK 165  
 I2C\_Read\_NACK 166  
 I2C\_Start 166  
 I2C\_Status 166  
 I2C\_Stop 167  
 I2C\_Write 167  
 IDE Einstellungen 82  
 if 106, 128  
 Installation 11, 15  
 int 96, 117  
 Interne Funktionen 145  
 Internet Explorer 84  
 Internet Update 84  
 IntFunc\_Lib.cc 145  
 IRQ 169



IRQ Beispiel 172  
Irq\_GetCount 171  
Irq\_SetVect 172

## - J -

Jumper 28, 46

## - K -

Key\_Init 173  
Key\_Scan 173  
Key\_TranslateKey 174  
kleiner 103, 124  
kleiner gleich 103, 124  
Kommentare 94, 115  
Kompilieren 56  
Kontexthilfe 88

## - L -

LCD Matrix 19  
LCD\_ClearLCD 174  
LCD\_CursorOff 175  
LCD\_CursorOn 175  
LCD\_CursorPos 175  
LCD\_Init 176  
LCD\_Locate 176  
LCD\_SubInit 177  
LCD\_TestBusy 177  
LCD\_WriteChar 178  
LCD\_WriteCTRRegister 178  
LCD\_WriteDataRegister 178  
LCD\_WriteFloat 179  
LCD\_WriteRegister 179  
LCD\_WriteText 180  
LCD\_WriteWord 180  
ldexp 184  
links schieben 102, 123  
ln 185  
log 185  
Logische Operatoren 103  
logisches Nicht 103  
logisches Oder 103  
logisches Und 103  
Loop While 124

## - M -

Map Datei 92  
Meldungen 56  
Modul 31, 48  
Modul Mega128 37  
Modul Mega32 20  
Modulo 101, 122  
Msg\_WriteChar 158  
Msg\_WriteFloat 159  
Msg\_WriteHex 159  
Msg\_WriteInt 159  
Msg\_WriteText 160  
Msg\_WriteWord 160  
Multiplikation 101, 122  
Muster 68

## - N -

Nächster Fehler 56  
Nebeneinander 87  
neue Features 5  
Next 126

## - O -

Oder 102, 123  
Onewire Beispiel 190  
Onewire\_Read 188  
Onewire\_Reset 188  
Onewire\_Write 189  
Open Source 4  
Operatoren 101, 122  
Operatoren Tabelle 112, 133

## - P -

Periodenmessung 235  
PIN 70  
Pinzuordnung 27, 45  
Port\_DataDir 192  
Port\_DataDirBit 193  
Port\_Read 194  
Port\_ReadBit 194  
Port\_Toggle 195

Port\_ToggleBit 196  
 Port\_Write 197  
 Port\_WriteBit 198  
 pow 186  
 Präzedenz 112, 132  
 Preprozessor 90  
 Programm 93, 115  
 Programm starten 69  
 Programmversion 88  
 Projekt 56  
 Projektdateien 57  
 Projekte Kompilieren 56  
 Projektname 56  
 Projektoptionen 59  
 Proxy 84  
 Pulsmessung 235  
 Pulsweitenmodulation 235

## - Q -

Quadratwurzel 187

## - R -

RC5 199  
 RC5\_Init 202  
 RC5\_Read 203  
 RC5\_Write 204  
 rechts schieben 102, 123  
 Rechtschreibprüfung 82  
 Referenzspannung 150, 151  
 Reguläre Ausdrücke 68  
 reserviert 113, 134  
 reservierte Worte 113, 134  
 round 186

## - S -

Schaltplan 31, 32, 48, 50  
 Schnittstelle 83  
 Schnittstellensuche 83  
 Select 128  
 Serial Beispiel 211  
 Serial Beispiel (IRQ) 211  
 Serial\_Disable 205  
 Serial\_Init 206

Serial\_Init\_IRQ 207  
 Serial\_IRQ\_Info 208  
 Serial\_Read 209  
 Serial\_ReadExt 209  
 Serial\_Write 210  
 Serial\_WriteText 210  
 serieller Bootloader 17  
 Service 4  
 Servo 211  
 Servo Beispiel 214  
 Servo\_Init 212  
 Servo\_Set 213  
 Sichtbarkeit von Variablen 97, 118  
 sin 186  
 SInus 186  
 sizeof 97, 118  
 Sleep 146  
 Smart Tabulator 78  
 SPI Abschaltung 17  
 SPI\_Disable 214  
 SPI\_Enable 215  
 SPI\_Read 216  
 SPI\_ReadBuf 216  
 SPI\_Write 216  
 SPI\_WriteBuf 217  
 Splashscreen 82  
 sqrt 187  
 SRAM 42, 90  
 Starten 69  
 static 97, 118  
 Stopbits 85  
 Str\_Comp 217  
 Str\_Copy 218  
 Str\_Fill 218  
 Str\_Isalnum 219  
 Str\_Isalpha 219  
 Str\_Len 220  
 Str\_Printf 220  
 Str\_Printf Beispiel 225  
 Str\_ReadFloat 221  
 Str\_ReadInt 221  
 Str\_Substr 222  
 Str\_WriteFloat 223  
 Str\_WriteInt 223  
 Str\_WriteWord 224  
 Strings 96, 97, 117, 118, 217

Subtraktion 101, 122  
Suchen 64  
switch 107  
Syntaktische Eingabehilfe 61  
Syntaxhervorhebung 79

## - T -

Tabellen 97, 118  
tan 187  
Tangens 187  
Tastaturbelegung 78  
Tastaturkürzel 66  
Terminal 76  
Terminal Einstellungen 85  
Thread\_Cycles 227  
Thread\_Delay 227  
Thread\_Info 228  
Thread\_Kill 228  
Thread\_Lock 229  
Thread\_MemFree 229  
Thread\_Resume 230  
Thread\_Signal 230  
Thread\_Start 230  
Thread\_Wait 231  
Threadoptionen 60  
Threads 225  
Timer 233  
Timer\_T0CNT 237  
Timer\_T0Disable 237  
Timer\_T0FRQ 238  
Timer\_T0GetCNT 239  
Timer\_T0PW 239  
Timer\_T0PWM 239  
Timer\_T0Start 240  
Timer\_T0Stop 241  
Timer\_T0Time 241  
Timer\_T1CNT 242  
Timer\_T1CNT\_Int 242  
Timer\_T1FRQ 243  
Timer\_T1FRQX 244  
Timer\_T1GetCNT 244  
Timer\_T1GetPM 244  
Timer\_T1PM 245  
Timer\_T1PWA 245  
Timer\_T1PWB 246  
Timer\_T1PWM 246  
Timer\_T1PWMX 247  
Timer\_T1PWMY 247  
Timer\_T1Start 248  
Timer\_T1Stop 248  
Timer\_T1Time 249  
Timer\_T3CNT 249  
Timer\_T3CNT\_Int 250  
Timer\_T3FRQ 250  
Timer\_T3FRQX 251  
Timer\_T3GetCNT 251  
Timer\_T3GetPM 252  
Timer\_T3PM 253  
Timer\_T3PWA 252  
Timer\_T3PWB 253  
Timer\_T3PWM 254  
Timer\_T3PWMX 254  
Timer\_T3PWMY 255  
Timer\_T3Start 255  
Timer\_T3Stop 256  
Timer\_T3Time 256  
Timer\_TickCount 257  
Timerfunktionen 236  
Typkonvertierung 96, 117

## - U -

Überlappend 87  
Übertragen 69  
Umbenennen 57  
Und 102, 123  
ungleich 103, 124  
unsigned char 96, 117  
unsigned int 96, 117  
Untereinander 87  
USB 11, 83

## - V -

Variable Ändern 73  
Variable Einfügen 73  
Variablen 97, 118  
Variablen Aktualisieren 73  
Variablen Fenster 73  
Vergleichsoperatoren 103, 124  
Versionsüberprüfung 71

Verwendung 3  
void 109, 130  
vordefinierte Arrays 97, 118  
Vorheriger Fehler 56  
Vorzeichen 101, 122

## - W -

Werkzeug Einstellungen 86  
Werkzeuge 76  
While 109, 125  
word 96, 117

## - Z -

Zeiger 109, 130

