Bo Hanus

Der leichte Einstieg in die Elektronik

7., aktualisierte Auflage

Ein leicht verständlicher Grundkurs mit vielen Bauanleitungen

Jokers edition

Bo Hanus

Der leichte Einstieg in die Elektronik

Bo Hanus

Der leichte Einstieg in die Elektronik

7., aktualisierte Auflage

Ein leicht verständlicher Grundkurs mit vielen Bauanleitungen

Jokers edition

Satz: Fotosatz Pfeifer, 82166 Gräfelfing
art & design: www.ideehoch2.de
Druck: Bercker, 47623 Kevelaer
Printed in Germany

ISBN 978-3-7723-**4079-6**

Vorwort

Das C-Control-Pro-System basiert auf dem Atmel Mega 32 RISC-Mikrocontroller. Dieser Mikrocontroller findet seinen Einsatz in Geräten von der Unterhaltungselektronik über Haushaltsmaschinen bis hin zu verschiedenen Industrieapplikationen. Dort übernimmt der Controller wichtige Steuerungsaufgaben. Mit C-Control Pro lassen sich beispielsweise analoge Messwerte und Schalterstellungen erfassen und abhängig von diesen Eingangsbedingungen entsprechende Schaltsignale ausgeben. 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.

Die in diesem Buch vorgestellten Demos und Beispiele wurden sorgfältig und ausführlich getestet. Da es aber eine Vielzahl von Hardware- und Softwarevariationen gibt, kann für ein ordnungsgemäßes Funktionieren dieser Programme keine Garantie übernommen werden.

Inhalt

1 Einleitung 11

2 Mega32 15

3 Application-Board Mega32 16

4 Mega128 18

5 Application-Board Mega128 19

6 Hardware-Einstellung 21
6.1 Application-Board MEGA32 21
6.1.1 Programmierung über USB 21
6.1.2 Programmierung über RS232 22
6.2 Application-Board M128 23
6.2.1 Programmierung über USB 24
6.2.2 Programmierung über RS232 25

7 Software-Installation 28
7.1 Entwicklungsumgebung 28
7.2 USB-Treiber 32

8 Software-Einstellungen 37
8.1 IDE-Update 37
8.2 Compiler 38
8.3 Editor 39

9 Das erste Programm 41
9.1 Programmierung 41
9.2 Fehlersuche 46
9.2.1 Software 48
9.2.2 Hardware 58

10 C und Basic in einem Projekt 60

11 Schutz der Programme (PIN) 63

12 Anschluss externer Komponenten 67
12.1 DCF-Modul 67

12.2 LCD Display 4 × 20 ... 70
12.3 Sensoren ... 74
12.3.1 Digitale Sensoren ... 74
12.3.2 Analoge Sensoren ... 76
12.4 CCI Relais-Modul ... 79
12.5 I²C-Bus-Thermometer-Modul ... 83
12.6 I²C-Bus-Tastatur ... 87

13 Stringverarbeitung ... 91
13.1 Strings in der C-Control-Pro-Umgebung ... 91
13.2 Strings sind Arrays ... 91
13.3 Stringfunktionen in der Bibliothek ... 92
13.4 Stringbearbeitung – selbst gemacht ... 93
13.5 Steuerzeichen ... 98
13.6 Formatierung numerischer Werte ... 99

14 Optimierung von CompactC ... 101
14.1 Optimierung ist Programmierersache ... 101
14.2 Optimierung Schritt für Schritt ... 102
14.3 Switch-Anweisungen sind effizient ... 104
14.4 Arithmetische Ausdrücke vereinfachen ... 105
14.5 Eingliedern von Funktionen ... 106
14.6 Einsparen von Programmcode ... 108
14.7 Projektoptionen prüfen ... 110

15 Optimierung von BASIC ... 113
15.1 Optimierung ist Programmierersache ... 113
15.2 Optimierung Schritt für Schritt ... 113
15.3 Select-Case-Anweisungen sind effizient ... 115
15.4 For-Schleifen benutzen ... 116
15.5 Arithmetische Ausdrücke vereinfachen ... 117
15.6 Eingliedern von Funktionen ... 118
15.7 Einsparen von Programmcode ... 120
15.8 Projektoptionen prüfen ... 122

16 Der Preprozessor ... 124
16.1 Definitionen ... 124
16.2 Bedingte Kompilierung ... 126
16.3 Einfügen von Dateien ... 128
16.4 Preprozessor-Makros ... 129
16.5 Vordefinierte Symbole ... 131
16.6 Compiler-Anweisungen ... 133
16.7 Mischen von BASIC und CompactC ... 133

17 Interruptbehandlung 135
17.1 C-Control-Pro-Interrupts 135
17.2 Externe Interrupts 137
17.3 Interpreter-Interrupts im Detail 139

18 Multithreading 140
18.1 Starten von Threads 140
18.2 Konfiguration des Multithreadings 142
18.3 Warten in Threads 145
18.4 Threads synchronisieren 147
18.5 Multithreading im Detail 149

19 Anwendungen 151
19.1 Voltmeter 151
19.2 Heizungssteuerung mit NTC-Sensoren 154
19.3 Heizungssteuerung mit Raumtemperaturregler 163
19.4 Temperaturschalter mit Sensorüberwachung 168
19.5 Zwei-Kanal-Thermometer 171
19.6 Temperatur-Differenzschalter 174
19.7 Acht-Kanal-Lauflicht 177
19.8 Digital-Timer 181
19.9 Stoppuhr 188
19.10 Gewächshausreglung 193
19.11 3-Kanal-DCF-Zeitschaltuhr 201
19.12 Ein-/Ausschaltverzögerung 211

20 Der Bytecode-Interpreter 215
20.1 Die Speicherbereiche im Interpreter 215
20.2 Die Arbeitsweise des Arithmetik-Stacks 216
20.3 Beispiel: Zuweisung 217
20.4 Beispiel: Funktionsaufruf 219
20.5 Beispiel: if-Anweisung 221
20.6 Beispiel: for-Schleife und Array-Zugriff 222
20.7 Beispiel: switch-Anweisung 224

21 Anhang – Bytecode-Übersicht 226
21.1 Bytecode-Übersicht 226

Sachverzeichnis 241

1 Einleitung

Das C-Control-Pro-System wurde als Nachfolger der beliebten C-Control-I-Serie von Conrad konzipiert. Wie der Name schon sagt, sollen die C-Control-Pro-Module den Ansprüchen der Entwickler genügen, die höhere Anforderungen an die C-Control-Familie stellen. Hatte die C-Control I von Hause aus einen einfachen BASIC-Dialekt, so besitzt die C-Control Pro zwei Programmiersprachen, die sehr nah am Industriestandard sind. Ergänzt wurde das System mit Multithreading und einem Laufzeit-Debugger.

Durch die größere Mächtigkeit des Systems wurde aber auch dessen Komplexität erhöht. Das mitgelieferte Handbuch der C-Control Pro, mit seinen über 200 Seiten keine einfache Lektüre, ist hauptsächlich als Nachschlagewerk für die Entwicklungsumgebung und den Compiler gedacht. Es stößt schnell an seine Grenzen, wenn es entweder um Anfängerfragen oder um anspruchsvolle Details geht. Daher haben viele Anwender nach einem Buch gefragt, das dem Einsteiger hilft, sich mit der C-Control Pro zurechtzufinden, aber auch dem Profi Informationen gibt, um mehr Leistung aus dem System herauszuholen.

Dieses Buch versucht in einer großen Bandbreite, wichtige Aspekte des C-Control-Pro-Systems herauszuarbeiten, und damit Anfängern wie Profis umfassende Informationen zu liefern. Von vielen Anwendern wissen wir, dass sie hauptsächlich in einer Programmiersprache arbeiten. Fast alle Programmbeispiele sind aus diesem Grunde in CompactC und BASIC angegeben. Die Kapitel über die Optimierung von Programmcode sind aus diesem Grunde auch komplett in beiden Sprachen getrennt geschrieben worden. Die Relevanz mancher Kapitel hängt auch von der Ausrichtung des Programmierers ab. So sind das Arbeiten mit nullterminierten Strings und die Benutzung des Preprozessors für den fortgeschrittenen C-Programmierer ein alter Hut, aber der BASIC-Anwender erfährt dort viel Neues. Aber auch die erweiterten Preprozessorfunktionen wie z. B. Makros sind nicht jedem C-Programmierer bekannt.

Dieses Buch gibt in den Kapiteln 2 bis 6 eine ausführliche Beschreibung der Hardware mit vielen Illustrationen. Das Kapitel 7 beschäftigt sich mit den Hardwareeinstellungen, und dabei vornehmlich mit der Konfiguration der Jumper. Gerade ein falsch gesetzter Jumper kann zu erheblichen Seiteneffekten und Fehlern beim Anschluss der eigenen Hardwarekomponenten führen. In Kapitel 8 wird die Installation der Software erklärt; insbesondere der zu installierende USB-Treiber hat bei Anfängern schon zu

Problemen geführt. Das Kapitel 9 beschäftigt sich mit den Einstellungen von IDE und Editor. Für den Anfänger besonders wichtig – in Kapitel 10 wird das erste C-Control-Pro-Programm geschrieben, und wertvolle Tipps zur Fehlersuche und zur Handhabung des Debuggers gegeben. In den Kapiteln 10 bis 12 erfährt der Leser über die Möglichkeiten, CompactC und BASIC zu mischen, sowie die Software-Anwendung mit PIN-Eingabe zu schützen.

Eine der Hauptansprechpunkte im C-Control-Pro-Forum ist die Ansteuerung externer Peripherie. Dieses Thema wird intensiv in Kapitel 12 behandelt. Es gibt dort Beispiele zum Anschluss und zur Programmierung von Komponenten, die man in dieser Form direkt, z. B. bei Conrad, erwerben kann. Die Liste der vorgestellten Peripherie umfasst I^2C-Module wie Thermometer und Bustastaturen, DCF-Module, LCD-Display und Relais. Aber auch die Behandlung von Sensoren, die direkt analog oder digital angeschlossen werden, wird hier demonstriert.

Die Kapitel 13 bis 18 fokussieren stärker die Programmierung des C-Control-Pro-Moduls. In Kapitel 13 wird die Stringverarbeitung mit nullterminierten Zeichenketten im Detail vorgestellt, sowie die Ausnutzung aller Fähigkeiten der Stringfunktionen in der Bibliothek. Ein weiteres Top-Thema aus dem Forum ist die Beschleunigung von CompactC- und BASIC-Programmen. Daher sind die Kapitel 14 und 15 den Optimierungsstrategien gewidmet, die es ermöglichen, das letzte bisschen Geschwindigkeit aus dem Bytecode-Interpreter herauszuholen. Nur wenn man weiß, wie der Compiler manche Konstrukte verarbeitet, und welche Anweisungen besonders effizient sind, kann man die Geschwindigkeit von Programmen steigern.

Der Preprozessor ist den meisten C-Programmierern bekannt, aber sein Potenzial wird oft nicht vollständig genutzt. Insbesondere dem BASIC-Anwender dürfte normalerweise ein Preprozessor gänzlich unbekannt sein. Kapitel 16 beschäftigt sich intensiv mit allen Möglichkeiten, die der mitgelieferte Preprozessor bietet.

Die Kapitel 17 und 18 stellen die Gebiete Interruptbehandlung und Multithreading vor. Diese Themen stellen den Anfänger oft vor Probleme, aber auch Fortgeschrittene kennen meist nicht die Eigenheiten, die die Implementierung in einem Bytecode-Interpreter mit sich bringt. Dem Anfänger werden in diesen Kapiteln in CompactC und BASIC viele Programmbeispiele gegeben, der Profi bekommt eine exakte Erläuterung, wie Interrupts und Multithreading im Detail verarbeitet werden.

Das Kapitel 19 steht wieder ganz im Licht der praktischen Arbeit und bringt zwölf Hardware-Anwendungen mitsamt Programmen in beiden Programmiersprachen. Es werden Probleme und Fallstricke im Zusammenspiel mit den C-Control-Pro-Modulen angesprochen, und es werden auf Eigenheiten zwischen Mega32 und Mega128 hingewiesen. Als praktische Einkaufshilfe existiert zu jeder Anwendung auch eine Bauteilliste mitsamt Conrad-Artikelnummer.

Das letzte Kapitel ist für den Experten gedacht, der mehr über die interne Arbeitsweise des Interpreters erfahren möchte. Im Kapitel 20 werden die Übersetzungen des Compilers und der Bytecode-Interpreter im Detail vorgestellt. Es gibt einen Überblick über das benutzte Speichermodell, die verschiedenen Stack-Arten und wie mit einem stackbasierten Interpreter gearbeitet wird. Für alle wichtigen Sprachkonstrukte von CompactC existieren in Kapitel 21 Beispielprogramme mitsamt ihrer Übersetzung in Bytecodes. Den Abschluss des Buches bildet der Anhang mitsamt einer Auflistung aller bis dato eingesetzten Bytecodes.

Viel Spaß beim Lesen und Freude beim Basteln an der Hardware wünschen Ihnen

Reiner Schirm
und Peter Sprenger

2 Mega32

Im MEGA32 ist der Mikrocontroller ATmega32 verbaut. Er stammt aus der AVR-Familie von ATMEL. Es handelt sich um einen low-power AVR-8-Bit-Mikrocontroller mit Advanced-RISC-Architektur.

Speicher:
Das MEGA32-Modul besitzt 32 kB FLASH, 1 kB EEPROM und 2 kB SRAM.

Reset:
Durch einen Reset wird das Modul in einen definierten Anfangszustand zurückgesetzt. Power-On-Reset und Hardware-Reset sind möglich.

Spannungsmessung:
Das Modul besitzt einen 10-Bit-Analog-Digital-Wandler. Mit diesem können Spannungen erfasst und als ganze Zahl dargestellt werden.

Takterzeugung:
Alle zeitlichen Abläufe sind vom 14,73456-MHz-Quarzoszillator abgeleitet.

Digitalports:
Der MEGA32 besitzt vier digitale Ports mit je acht Pins. Die Ports können als Eingang oder Ausgang pin- und byteweise angesprochen werden.

Timer:
Es stehen ein 8-Bit- und ein 16-Bit-Timer zur Verfügung.

Abb. 2.1: MEGA32

3 Application-Board Mega32

Spannungsversorgung:
Über die eingebaute Niedervoltbuchse wird das Application-Board mit Spannung versorgt.

Ein-/Ausschalter:
Der Schalter befindet sich an der Frontseite des Boards und dient zum Ein- und Ausschalten der Spannungsversorgung.

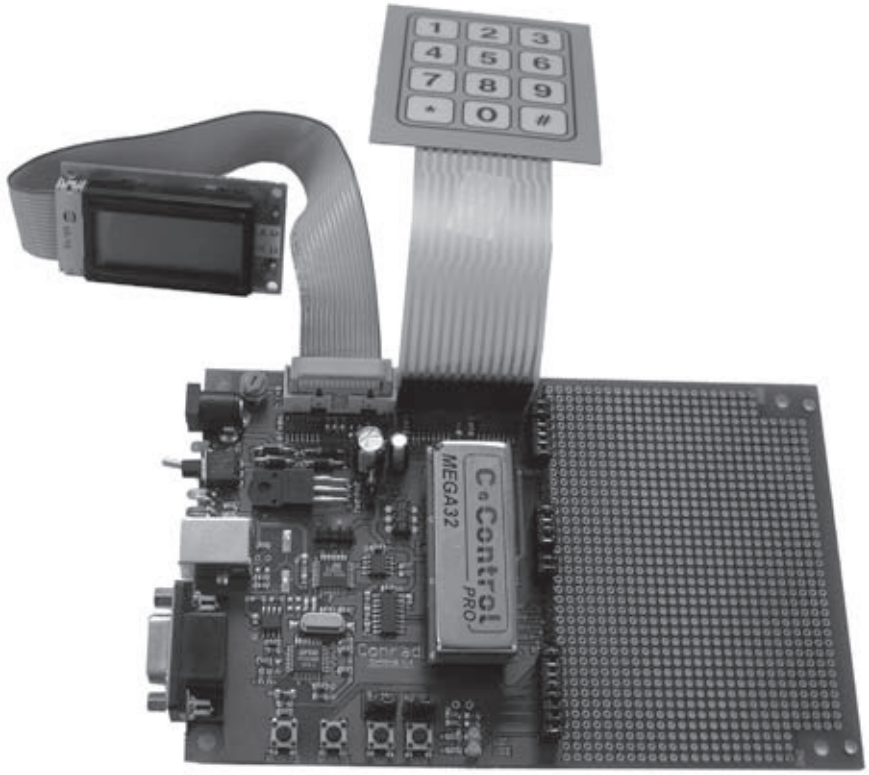


Abb. 3.1: Application-Board MEGA32

Leuchtdioden:
LD3 (grün) befindet sich an der Frontseite unter dem DC-Anschluss und signalisiert die vorhandene Versorgungsspannung.

LD4 (grün) und LD5 (rot) zeigen den Status der USB-Schnittstelle an.

LD1 (grün) und LD2 (grün) befinden sich neben den vier Tastern und stehen dem Anwender zur freien Verfügung.

USB:
Die USB-Schnittstelle dient zur schnellen Übertragung der Programme und zum Debuggen.

Serielle Schnittstelle:
Auf dem Application-Board befindet sich ein hochwertiges Pegelwandler-IC zur Umsetzung der digitalen Bitströme in Non-Return-Zero-Signale nach dem RS232-Standard.

Taster:
Es sind vier Taster vorhanden. SW1 und SW2 stehen dem Anwender zur freien Verfügung. SW3 (RESET1) löst beim MEGA32 einen RESET aus. SW5 (RESET2) löst beim MEGA8, der zur USB-Kommunikation verwendet wird, einen RESET aus.

LCD:
Über den Stecker X14 wird das LCD-Modul an das Board angesteckt.

LCD-Kontrast:
Über den Drehwiderstand PT1 kann der Kontrast des LCD-Displays eingestellt werden.

Tastatur:
Die 12er-Tastatur wird über den Stecker X15 mit dem Application-Board verbunden.

I2C-Schnittstelle:
Über die Port-Pins C.0 und C.1 steht eine I^2C-Schnittstelle zur Verfügung.

Speicher:
Auf dem Application-Board befindet sich ein EEPROM mit einer Speichertiefe von 8 kB. Dieses EEPROM ist über eine I^2C-Schnittstelle ansprechbar.

4 Mega128

Im MEGA128 ist der Mikrocontroller ATmega128 verbaut. Er stammt aus der AVR-Familie von ATMEL. Es handelt sich um einen low-power AVR-8-Bit-Mikrocontroller mit Advanced-RISC-Architektur.

Speicher:
Das MEGA128-Modul besitzt 128 kB FLASH, 4 kB EEPROM und 4 kB SRAM.

Reset:
Durch einen Reset wird das Modul in einen definierten Anfangszustand zurückgesetzt. Power-On-Reset und Hardware-Reset sind möglich.

Spannungsmessung:
Das Modul besitzt einen 10-Bit-Analog-Digital-Wandler. Mit diesem können Spannungen erfasst und als ganze Zahl dargestellt werden.

Takterzeugung:
Alle zeitlichen Abläufe sind vom 14,73456-MHz-Quarzoszillator abgeleitet.

Digitalports:
Der MEGA128 besitzt sieben digitale Ports. Sechs Ports sind mit je 8 Pins ausgestattet. Der siebte Port hat 5 Pins. Die Ports können als Eingang oder Ausgang pin- und byteweise angesprochen werden.

Timer:
Es stehen ein 8-Bit- und zwei 16-Bit-Timer zur Verfügung.

Abb. 4.1: MEGA128

5 Application-Board Mega128

Spannungsversorgung:
Über die eingebaute Niedervoltbuchse wird das Application-Board mit Spannung versorgt.

Ein-/Ausschalter:
Der Schalter befindet sich an der Frontseite des Boards und dient zum Ein- und Ausschalten der Spannungsversorgung.

Leuchtdioden:
LD3 (grün) befindet sich an der Frontseite unter dem DC-Anschluss und signalisiert die vorhandene Versorgungsspannung.

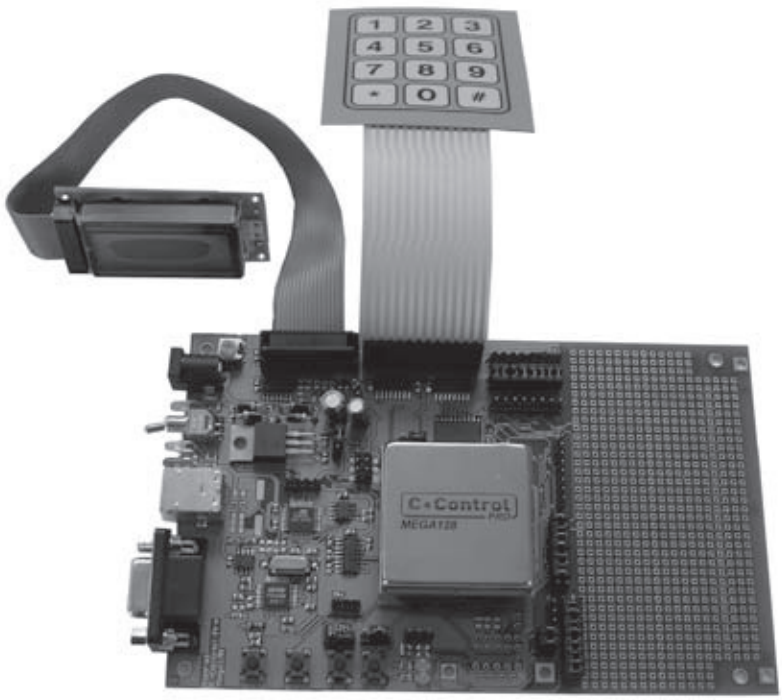


Abb. 5.1: Application-Board MEGA128

LD4 (grün) und LD5 (rot) zeigen den Status der USB-Schnittstelle an.

LD1 (grün) und LD2 (grün) befinden sich neben den vier Tastern und stehen dem Anwender zur freien Verfügung.

USB:
Die USB-Schnittstelle dient zur schnellen Übertragung der Programme und zum Debuggen.

Serielle Schnittstelle:
Für beide Schnittstellen befindet sich auf dem Application-Board ein hochwertiges Pegelwandler-IC zur Umsetzung der digitalen Bitströme in Non-Return-Zero-Signale nach dem RS232-Standard.

Taster:
Es sind vier Taster vorhanden. SW1 und SW2 stehen dem Anwender zur freien Verfügung. SW3 (RESET1) löst beim MEGA128 einen RESET aus. SW5 (RESET2) löst beim MEGA8, der zur USB-Kommunikation verwendet wird, einen RESET aus.

LCD:
Über den Stecker X14 wird das LCD-Modul an das Board angesteckt.

LCD-Kontrast:
Über den Drehwiderstand PT1 kann der Kontrast des LCD-Displays eingestellt werden.

Tastatur:
Die 12er-Tastatur wird über den Stecker X15 mit dem Application-Board verbunden.

I^2C-Schnittstelle:
Über die Port-Pins D.0 und D.1 steht eine I^2C-Schnittstelle zur Verfügung.

Speicher:
Auf dem Application-Board befindet sich ein EEPROM mit 8 kB Speichertiefe und ein SRAM mit 64 kB Speichertiefe. Das EEPROM ist über eine I^2C-Schnittstelle ansprechbar.

6 Hardware-Einstellung

6.1 Application-Board MEGA32

Auf Grund der Bauart des MEGA32 sind einige Pins mehrfach belegt. Bei der Planung Ihres Projektes müssen Sie also bedenken, welche Übertragungsart Sie zur Programmierung verwenden wollen bzw. können. Des Weiteren spielt es eine Rolle, ob Sie im Projekt z. B. Daten über die RS232-Schnittstelle senden oder empfangen wollen.

6.1.1 Programmierung über USB

Wenn Sie die schnelle USB-Kommunikation verwenden wollen, die auch für den Debug-Modus zu empfehlen ist, dürfen die Jumper PA6, PA7, PB4, PB5, PB6 und PB7 auf dem Application Board nicht entfernt werden. Wird dies nicht beachtet, kann Ihr PC keine Verbindung mit dem MEGA32 herstellen.

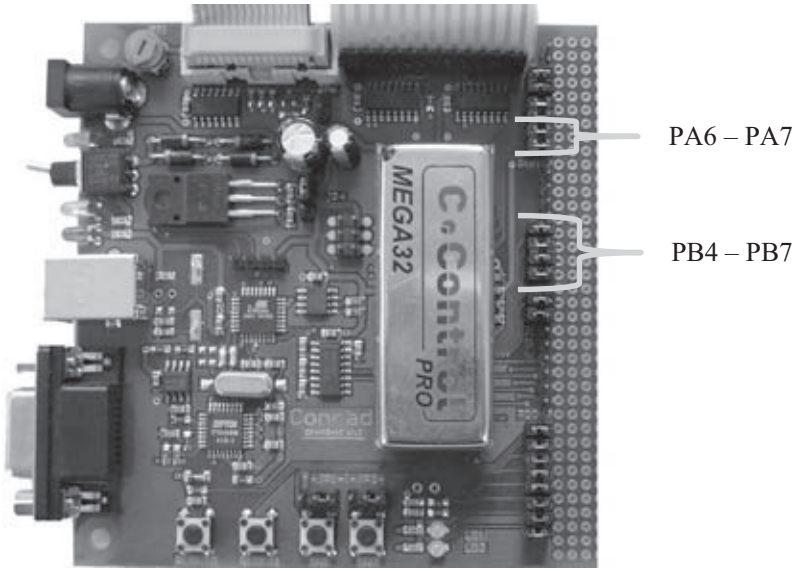


Abb. 6.1: AB – MEGA 32

Die Ports sind wie folgt belegt:

PA6:	TX REQ	(SPI TX REQ)
PA7:	RX BUSY	(SPI RX BUSY)
PB4:	SS	(Slave Select)
PB5:	MOSI	(Master out Slave in)
PB6:	MISO	(Master in Slave out)
PB7:	SCK	(Serial Clock)

Unter bestimmten Bedingungen können Sie z. B. die Ports PB4 bis PB7 aber dennoch verwenden. Vorraussetzung ist, dass Ihr Projekt bereits fehlerfrei funktioniert und Sie aus diesem Grund die USB-Kommunikation nicht mehr benötigen.

Um dies zu erreichen, testen Sie Ihr Projekt unter Verwendung anderer Ports (z. B. PC4 bis PC7). Sobald Ihr Programm fehlerfrei ist, ändern Sie die Ports auf PB4 bis PB7. Übertragen Sie nun das Programm in den MEGA32 und schalten Sie das Board über den Schalter aus. Stecken Sie nun die USB-Kabel am Application-Board ab. Jetzt können Sie die Jumper an den Ports PB4 bis PB7 entfernen und entsprechend Ihren Vorstellungen beschalten. Sie werden feststellen, dass Ihr Projekt automatisch gestartet wird, sobald Sie das Board mit dem Schalter wieder aktivieren. Durch diesen kleinen Trick sind jetzt auch die Ports PB4 bis PB7 verwendbar.

6.1.2 Programmierung über RS232

Wenn Sie zur Programmierung des MEGA32 die RS232-Schnittstelle verwenden wollen, müssen Sie zusätzlich auf einige Dinge achten. Da das Application-Board standardmäßig die USB-Schnittstelle zur Kommunikation mit dem PC verwendet, muss man beim Einschalten signalisieren, dass dies nicht der Fall ist. Schalten Sie also das Application-Board ab. Drücken Sie die Taste SW1 und schalten Sie gleichzeitig das Board wieder ein. Jetzt können Sie den Taster wieder loslassen. Sie werden feststellen, dass die LEDs LD4 und LD5 nun ohne Funktion sind.

Abb. 6.2: Taster SW1

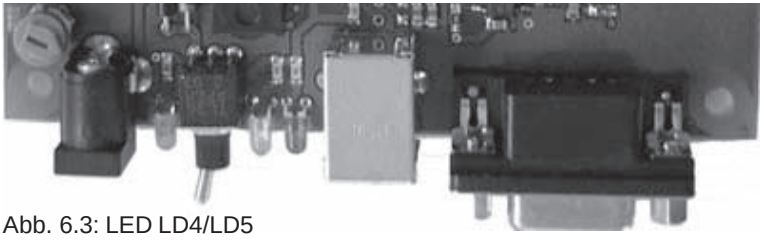


Abb. 6.3: LED LD4/LD5

Wählen Sie nun im Menü „Optionen“ den Punkt „IDE“ aus. Im erscheinenden Fenster klicken Sie bitte auf das Register „Schnittstellen“ und danach auf die Schaltfläche „Schnittstellensuche“. Über ein Fenster wird Ihnen nun angezeigt, dass die C-Control-Hardware gefunden wurde. Bestätigen Sie dies durch Anklicken der Schaltfläche „OK“. Im Feld „Kommunikationsport“ des Registers „Schnittstellen“ können Sie nun die verwendete RS232-Schnittstelle erkennen. (z. B. COM5)

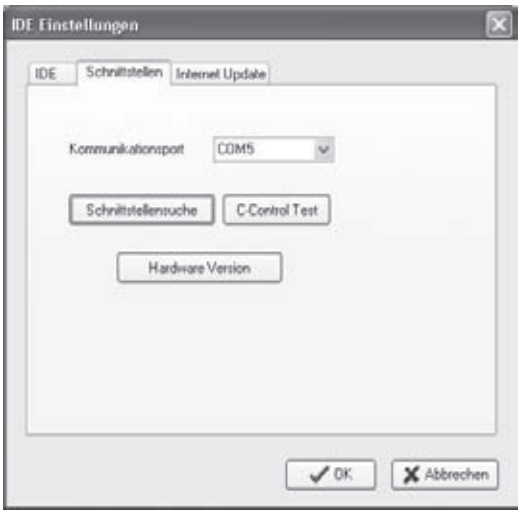


Abb. 6.4: Schnittstellensuche

Abb. 6.5: Hardware gefunden

Voraussetzung für eine funktionierende RS232-Kommunikation ist, dass die Jumper PD0 und PD1 auf dem Application-Board nicht entfernt werden (siehe Abb. 6.6). Wird dies nicht beachtet, kann Ihr PC keine Verbindung mit dem MEGA32 herstellen.

Die Ports sind wie folgt belegt:
PD0: RXD (Empfänger)
PD1: TXD (Sender)

6.2 Application-Board M128

Bei dem MEGA128 sind bauartbedingt einige Pins mehrfach belegt. Bei der Planung Ihres Projektes müssen Sie also bedenken, welche Übertragungsart Sie zur Programmierung verwenden wollen bzw. können. Des Weiteren spielt es eine Rolle, ob Sie im Projekt z. B. Daten über die RS232-Schnittstelle senden oder empfangen wollen. Beim MEGA128 haben Sie den Vorteil, dass zwei RS232-Schnittstellen zur Verfügung stehen.

Abb. 6.6: RS232-Ports – MEGA32

6.2.1 Programmierung über USB

Wenn Sie die schnelle USB-Kommunikation verwenden wollen, die auch für den Debug-Modus zu empfehlen ist, dürfen die Jumper PB0, PB1, PB2, PB3, PB4 und PE5 auf dem Application-Board nicht entfernt werden. Wird dies nicht beachtet, kann Ihr PC keine Verbindung mit dem MEGA128 herstellen.

Die Ports sind wie folgt belegt:

PB0:	SS	(Slave Select)
PB1:	SCK	(Serial Clock)
PB2:	MOSI	(Master out Slave in)
PB3:	MISO	(Master in Slave out)
PB4:	RX BUSY	(SPI RX BUSY)
PE5:	TX REQ	(SPI TX REQ)

Unter bestimmten Bedingungen können Sie z. B. die Ports PB0 bis PB3 aber dennoch verwenden. Voraussetzung ist, dass Ihr Projekt bereits fehlerfrei funktioniert und Sie aus diesem Grund die USB-Kommunikation nicht mehr benötigen.

Um dies zu erreichen, testen Sie Ihr Projekt unter Verwendung anderer Ports (z. B. PC0 bis PC3). Sobald Ihr Programm fehlerfrei ist, ändern Sie die Ports auf PB0 bis PB3.

Übertragen Sie nun das Programm in den MEGA128 und schalten Sie das Board über den Schalter aus. Stecken Sie nun die USB-Kabel am Application-Board ab. Jetzt kön-

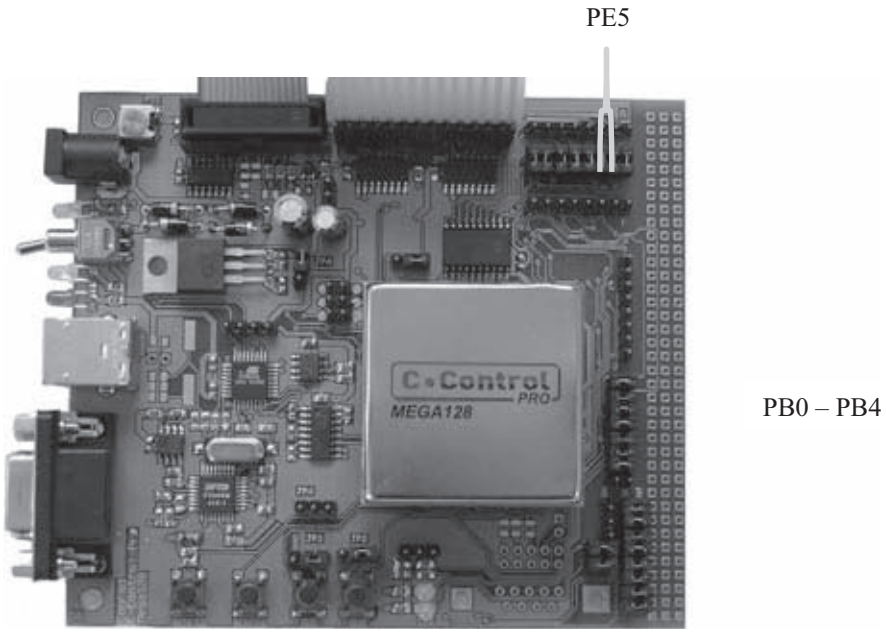


Abb. 6.7: RS232 Ports – MEGA128

nen Sie die Jumper an den Ports PB0 bis PB3 entfernen und entsprechend Ihren Vorstellungen beschalten. Sie werden feststellen, dass Ihr Projekt automatisch gestartet wird sobald Sie das Board mit dem Schalter wieder aktivieren. Durch diesen kleinen Trick sind jetzt auch die Ports PB0 bis PB3 verwendbar.

6.2.2 Programmierung über RS232

Wenn Sie zur Programmierung des MEGA128 die RS232-Schnittstelle verwenden wollen, müssen Sie zusätzlich auf einige Dinge achten. Da das Application-Board standardmäßig die USB-Schnittstelle zur Kommunikation mit dem PC verwendet, muss man beim Einschalten signalisieren, dass dies nicht der Fall ist. Schalten Sie also das Application-Board ab. Drücken Sie die Taste SW1 (siehe Abb. 6.8) und schalten Sie gleichzeitig das Board wieder ein. Jetzt können Sie den Taster wieder loslassen. Sie werden feststellen, dass die LEDs LD4 und LD5 nun ohne Funktion sind.

Wählen Sie nun im Menü „Optionen“ den Punkt „IDE“ aus. Im erscheinenden Fenster klicken Sie bitte auf das Register „Schnittstellen“ und danach auf die Schaltfläche „Schnittstellensuche“. Über ein Fenster wird Ihnen nun angezeigt, dass die C-Control-Hardware gefunden wurde. Bestätigen Sie dies durch Anklicken der Schaltfläche „OK“. Im Feld „Kommunikationsport“ des Registers „Schnittstellen“ können Sie nun die verwendete RS232-Schnittstelle erkennen. (z. B. COM5)

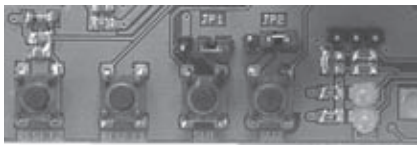


Abb. 6.8: Taster SW1

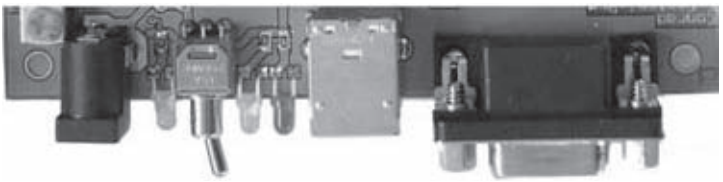


Abb. 6.9: LEDs LD4/LD5

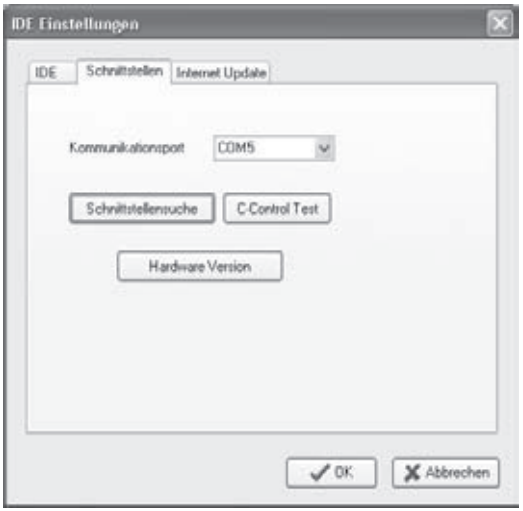


Abb. 6.10: Schnittstellensuche

Abb. 6.11: Hardware gefunden

Voraussetzung für eine funktionierende RS232-Kommunikation ist, dass die Jumper PE0 und PE1 auf dem Application-Board nicht entfernt werden (siehe Abb. 6.12). Wird dies nicht beachtet, kann Ihr PC keine Verbindung mit dem MEGA128 herstellen.

Die Ports sind wie folgt belegt:

PE0:	RXD0	(Empfänger)
PE1:	TXD0	(Sender)

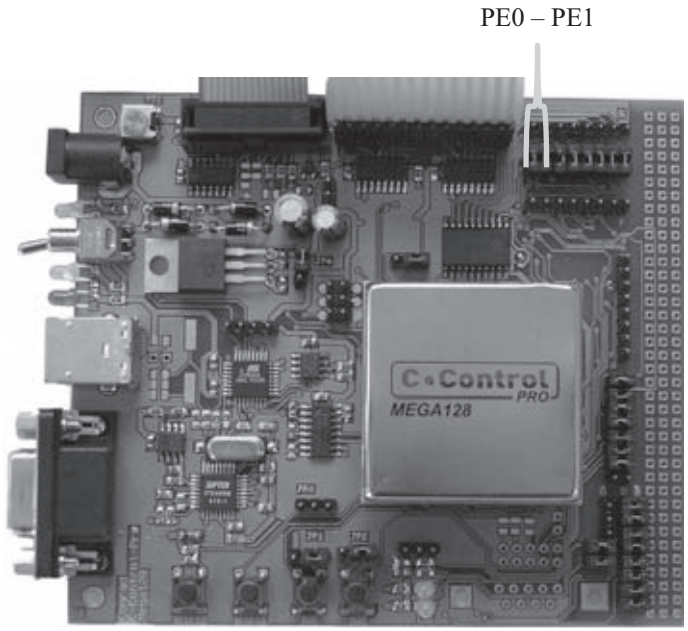


Abb. 6.12: RS232 – MEGA128

7 Software-Installation

7.1 Entwicklungsumgebung

Legen Sie die im Lieferumfang enthaltene CD-ROM in das CD-Laufwerk ein. Öffnen Sie den Explorer und klicken Sie auf den Ordner des CD-Laufwerkes. Nach einem Doppelklick auf die Datei „C-ControlSetup.exe“ beginnt die Installation der Software.

Im ersten Fenster können Sie die Sprache wählen. Diese Auswahl bezieht sich aber nur auf die Installationsdialoge.

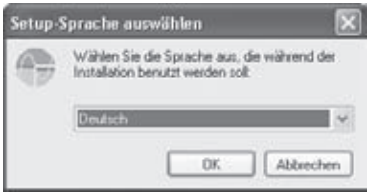


Abb. 7.1: Sprache wählen

Wählen Sie bitte im Listenfeld die gewünschte Sprache aus und bestätigen Sie dies durch einen Klick auf die Schaltfläche „OK“.

Beachten Sie bitte die nun folgenden Fenster, da die angezeigten Informationen für ein erfolgreiches Arbeit mit der C-Control-Pro-Software sehr wichtig sind.

Updates der Software werden in das von Ihnen gewählte C-Control-Pro-Verzeichnis installiert. Da es unter Umständen nötig ist, dass auch die Demos entsprechend aktualisiert werden, wird bei einer Installation auch der Ordner „DemoProgramme“ überschrieben. Bedenken Sie dies bei der Speicherung Ihrer eigenen Programme und Projekte und wählen Sie einen Ordner außerhalb des C-Control-Pro-Verzeichnisses.

Um die Installation fortzuführen, klicken Sie bitte jeweils auf die Schaltfläche „Weiter >“.

Im nächsten Fenster können Sie den Ordner, Pfad und das Verzeichnis wählen in den die C-Control-Pro-Software installiert werden soll.

Sollte der vorgeschlagene Ziel-Ordner nicht Ihren Vorstellungen entsprechen, ändern Sie ihn bitte entsprechend im Eingabefenster ab. Bestätigen Sie Ihre Wahl durch einen Klick auf die Schaltfläche „Weiter >“.

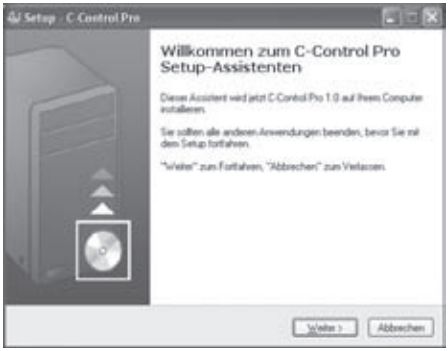


Abb. 7.2: Setup – Willkommen

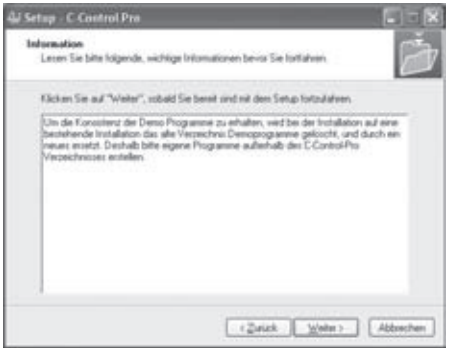


Abb. 7.3: Setup – Konsistenz

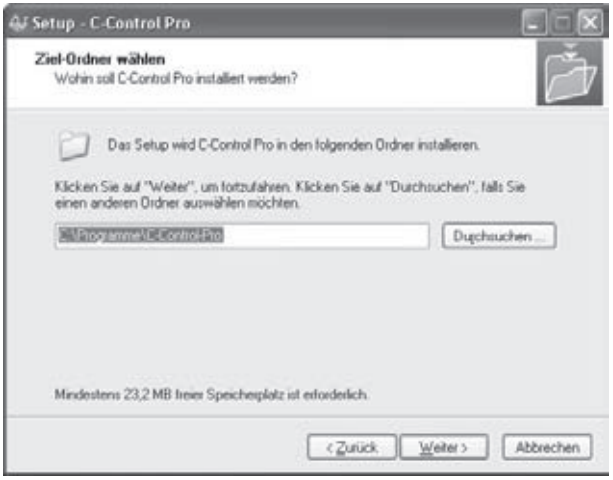


Abb. 7.4: Ziel-Ordner-Auswahl

Bei einer Neuinstallation ist der vorgeschlagene Ziel-Ordner noch nicht auf Ihrem PC vorhanden. Deshalb erfolgt zur Sicherheit nochmals eine Abfrage, ob der gewählte Ordner erstellt werden soll.

Sind die Angaben richtig, bestätigen Sie diese mit einem Klick auf die Schaltfläche „Ja“.

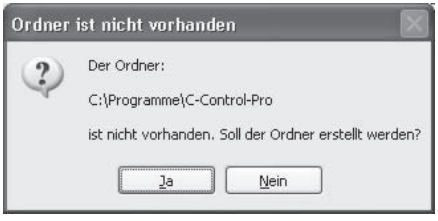


Abb. 7.5: Ziel-Ordner-Abfrage

Sollte Ihnen bei der Eingabe des Ziel-Ordners ein Fehler unterlaufen sein, können Sie durch einen Klick auf die Schaltfläche „Nein“ zur Ziel-Ordner-Auswahl zurückkehren.

Im nächsten Fenster können Sie nun den Ordner wählen, in dem die Programmverknüpfungen hinterlegt werden sollten. Standardmäßig ist dies der Ordner „C-Control Pro“. Über das Eingabefeld können Sie natürlich diesen Namen Ihren eigenen Vorstellungen entsprechend anpassen.

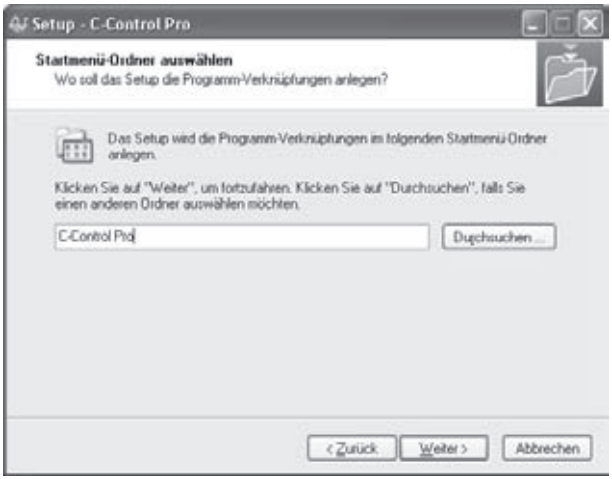


Abb. 7.6: Startmenü-Ordner

Bestätigen Sie Ihre Wahl durch einen Klick auf die Schaltfläche „Weiter >“.

Der nächste Dialog gibt Ihnen die Möglichkeit festzulegen, ob ein Desktop- bzw. ein Schnellstartleistensymbol angelegt werden soll. Ist dies erwünscht, wählen Sie bitte die entsprechenden Kontrollfelder durch einen Klick aus.

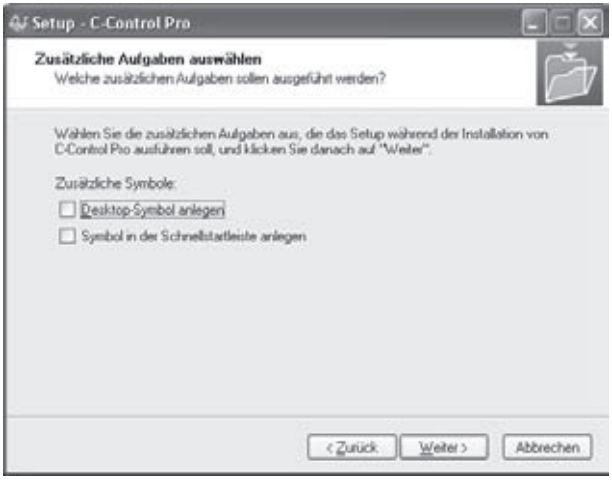


Abb. 7.7: Symbolauswahl

Bestätigen Sie Ihre Wahl durch einen Klick auf die Schaltfläche „Weiter >“.

Zum Schluss werden Ihre Eingaben noch einmal zusammenfassend dargestellt. Sollten Sie feststellen, dass eine oder mehrere Angaben nicht Ihren Wünschen entsprechen, können Sie über einen Klick auf die Schaltfläche „< Zurück“ bis zum entsprechenden Punkt in der Installation zurückgehen und die Einstellungen entsprechend ändern. Sind alle Angaben richtig, klicken Sie bitte zur Bestätigung auf die Schaltfläche „Installieren“.

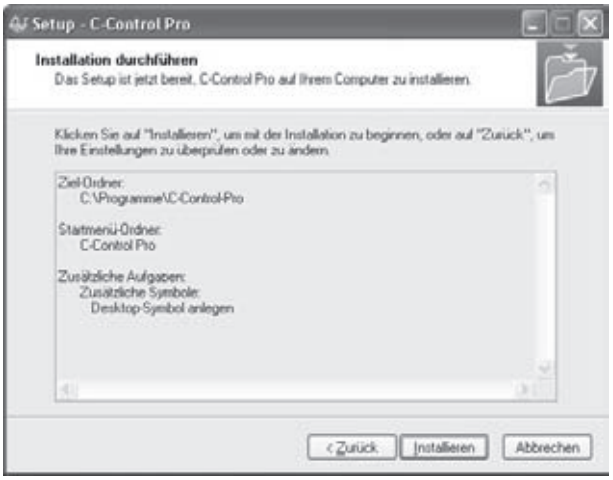


Abb. 7.8: Zusammenfassung

Über den Fortschrittsbalken wird Ihnen der Stand der Installation angezeigt. Neben den erforderlichen Dateien für die IDE werden auch die USB-Treiber, die Demo-Programme und die Anleitung kopiert.

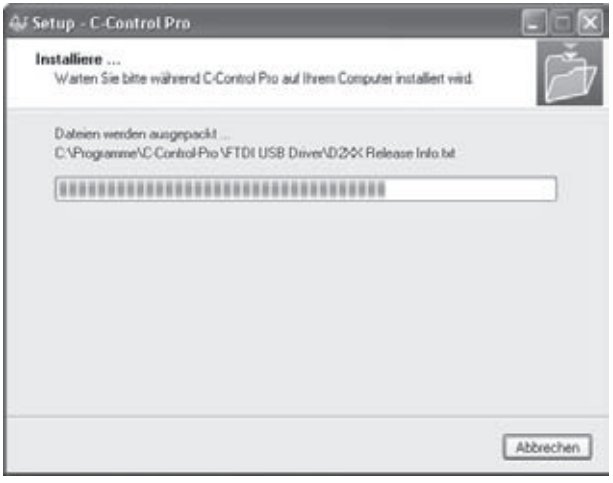


Abb. 7.9: Installation

Die Anleitung kann sowohl als PDF im Ordner „Manual“ gestartet werden als auch über die Hilfe in der IDE. Im Ordner „Datasheets“ auf der CD können Sie noch weitere Informationen zu den Atmel-Chips, dem LCD-Display und den Application-Boards finden. Dieser Ordner wird nicht mit auf die Festplatte kopiert.

Der folgende Dialog ist der Abschluss der Installation. Hier wird abgefragt, wie Sie nun fortfahren wollen.

Abb. 7.10: Installation fertigstellen

Es ist zu empfehlen, die eingestellten Kontrollfelder in dieser Form beizubehalten. In der „ReadMe.txt“-Datei werden Informationen angezeigt, die eventuell aktueller als die in der Anleitung sind und dadurch unter Umständen für einen Programmiererfolg nötig sind. Im Tutorial stehen zwei kurze Filme zur Verfügung, die Sie kurz in die Bedienung der IDE einweisen. Wenn Sie die gezeigten Vorgänge parallel in der Entwicklungsumgebung durchführen, können Sie sehr schnell Ihr erstes Projekt erstellen.

Klicken Sie auf die Schaltfläche „Fertigstellen“, um die Installation abzuschließen und die entsprechenden Einstellungen auszuführen.

7.2 USB-Treiber

Je nachdem, für welche Datenkommunikationsart Sie sich entscheiden (siehe Kapitel 6), ist es nun eventuell erforderlich, den USB-Treiber für das Application-Board zu installieren.

Verbinden Sie hierfür das Board mit dem Netzteil. Zu diesem Zeitpunkt schalten Sie das Application-Board aber noch nicht ein. Stecken Sie das USB-Kabel in die entsprechenden Stecker an Ihrem PC und am Board. Schalten Sie nun das Application-Board ein.

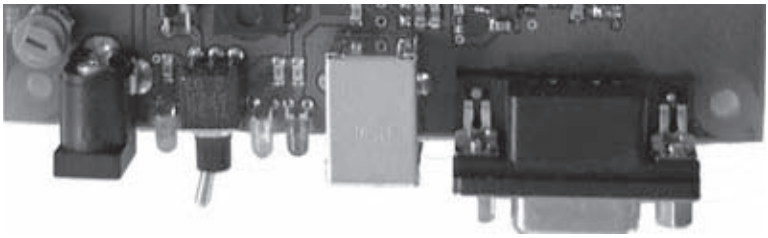


Abb. 7.11: AB-Einschalter

Das Betriebssystem meldet nun, dass es eine neue Hardware gefunden hat und möchte den entsprechenden Treiber installieren.

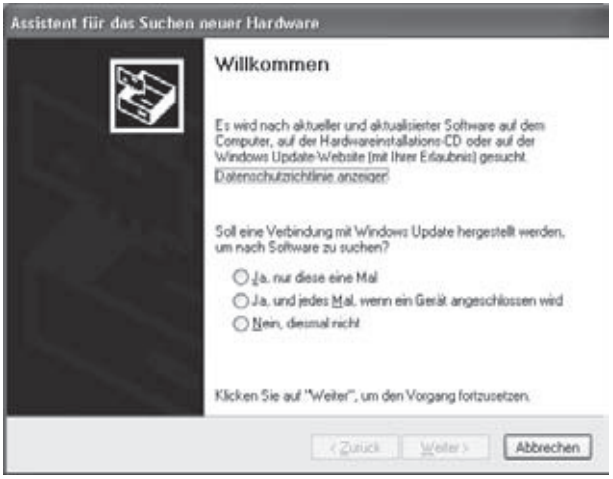


Abb. 7.12: Hardware-Assistent

Klicken Sie bitte auf das Optionsfeld „Nein, diesmal nicht“ und danach auf die Schaltfläche „Weiter >“.

Jetzt ist es erforderlich, den Ort festzulegen wo der Assistent den Treiber für die C-Control Pro findet. Klicken Sie hierfür auf das Optionsfeld „Software von einer Liste oder bestimmten Quelle installieren (für fortgeschrittene Benutzer)“ und danach auf die Schaltfläche „Weiter >“.

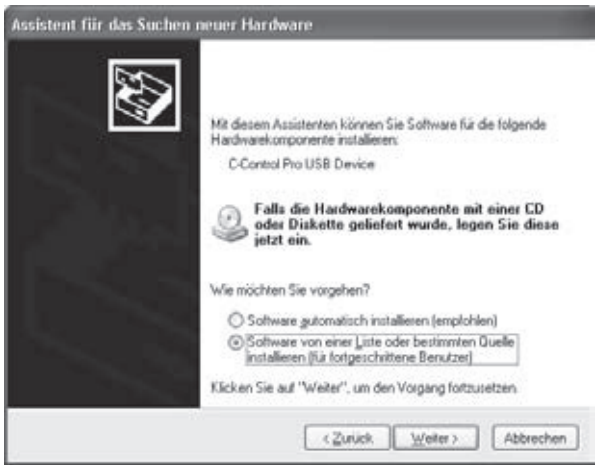


Abb. 7.13: Hardware-Assistent

Im nächsten Fenster wählen Sie bitte das Optionsfeld „Diese Quellen nach dem zutreffendsten Treiber durchsuchen“ und das Kontrollfeld „Folgende Quelle ebenfalls durchsuchen“ durch anklicken aus. In das Eingabefeld geben Sie bitte den Pfad ein in den Sie die C-Control-Pro-Software installiert haben. Im Ordner „FTDI USB Driver“ dieses Verzeichnisses ist der Treiber für die C-Control Pro zu finden. Natürlich können Sie zur Suche des richtigen Pfades auch die Schaltfläche „Durchsuchen“ anklicken und dann entsprechend vorgehen.

Abb. 7.14: Hardware-Assistent

Bestätigen Sie Ihre Eingaben bitte durch einen Klick auf die Schaltfläche „Weiter >“.

Nun erfolgt die eigentliche Installation des Treibers. Wenn angezeigt wird, dass der Treiber den Windows-Logo-Test nicht bestanden hat, klicken Sie bitte auf die Schaltfläche „Installation fortsetzen“.

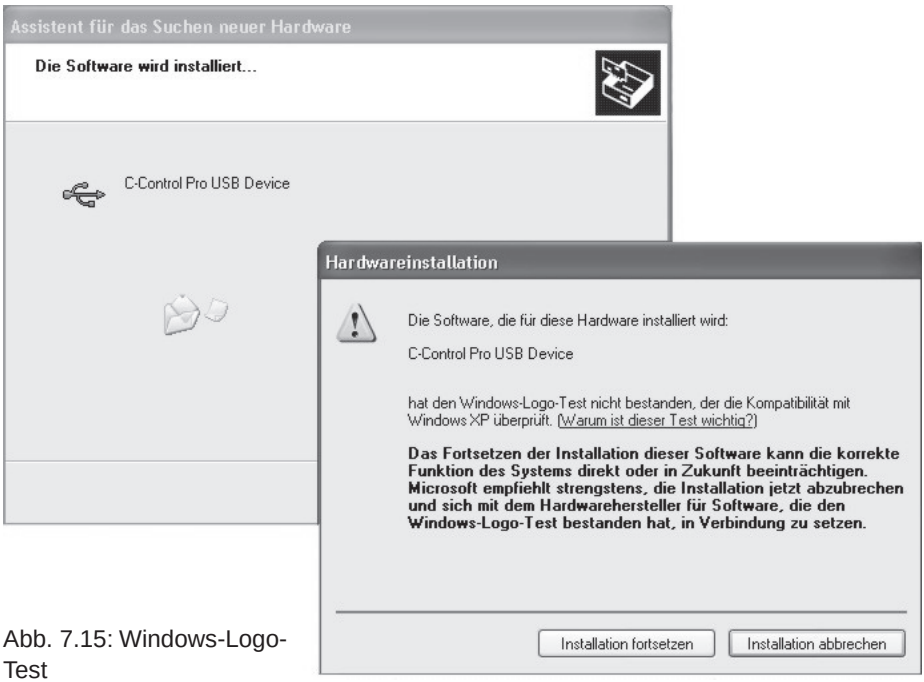


Abb. 7.15: Windows-Logo-Test

Abb. 7.16: Treiberinstallation

Abb. 7.17: Installation abschließen

Um die Treiberinstallation abzuschließen, klicken Sie bitte auf die Schaltfläche „Fertig stellen“. Jetzt steht die USB-Verbindung zur Übertragung der Programme der C-Control Pro zur Verfügung.

Starten Sie nun die IDE durch einen Doppelklick auf das C-Control-Pro-Icon.

Wählen Sie im Menü „Optionen“ den Punkt „IDE“ aus. Im erscheinenden Fenster klicken Sie bitte auf das Register „Schnittstellen“ und danach auf die Schaltfläche

„Schnittstellensuche“. Über ein Fenster wird Ihnen nun angezeigt, dass die C-Control-Hardware gefunden wurde. Bestätigen Sie dies durch Anklicken der Schaltfläche „OK“. Im Feld „Kommunikationsport“ des Registers „Schnittstellen“ können Sie nun die verwendete USB-Schnittstelle erkennen. (z. B. USB0)

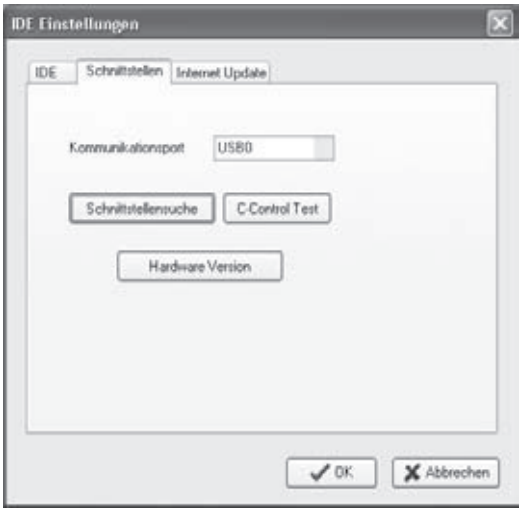


Abb. 7.18: Schnittstellensuche

Abb. 7.19: Hardware gefunden

8 Software-Einstellungen

Nachdem Sie die Installation der IDE (Integrated Development Environment) und Ihrer Hardware abgeschlossen haben, ist es sinnvoll, vor der ersten Programmierung einige Einstellungen durchzuführen.

8.1 IDE-Update

Über diese Funktion wird überprüft, ob seit der Erstellung der im Lieferumfang enthaltenen Software-CD eine neue Version zur Verfügung steht. Normalerweise ist diese Funktion automatisch eingestellt und wird beim ersten Start der IDE bereits ausgeführt. Um diese Funktion einzustellen oder manuell durchzuführen, klicken Sie bitte in der Menüzeile auf „Optionen“ und wählen Sie danach den Punkt IDE.

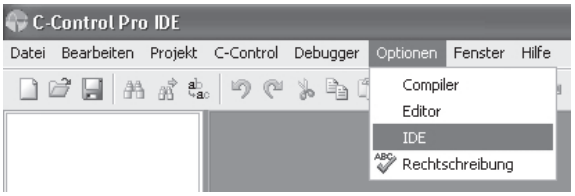


Abb. 8.1: Menüzeile IDE

Im nun erscheinenden Fenster wählen Sie bitte das Register „Internet Update“. Jetzt wird das Dialogfester zur Einstellung der Update-Funktionen angezeigt. Klicken Sie bitte auf die Schaltfläche „Jetzt auf Update prüfen“. Die IDE stellt nun eine Verbindung mit dem C-Control-Pro-Server her und versucht eine neue Version herunterzuladen. Bitte stellen Sie sicher, dass der Verbindungsaufbau nicht durch eine verwendete Firewall blockiert wird. Sollte eine neue Version der C-Control-Pro-IDE zur Verfügung stehen, findet jetzt der entsprechende Download statt. Wenn nicht, wird dies durch das Fenster in Abb. 8.3 angezeigt.

Bitte beachten Sie, dass Updates in das von Ihnen gewählte C-Control-Pro-Verzeichnis installiert werden. Da es unter Umständen wichtig ist, dass auch die Demos entsprechend aktualisiert werden, wird bei einer Installation auch der Ordner „DemoProgramme“ überschrieben. Bedenken Sie dies bei der Speicherung Ihrer eigenen Programme und wählen sie einen Ordner außerhalb des C-Control-Pro-Verzeichnisses.

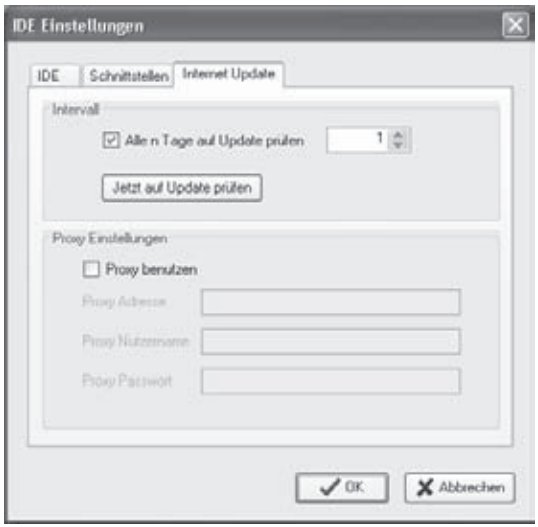


Abb. 8.2: IDE-Einstellungen

Abb. 8.3: Update

8.2 Compiler

Um die Compiler Voreinstellungen zu ändern klicken Sie bitte in der Menüleiste auf „Optionen“ und wählen Sie danach den Punkt „Compiler“.

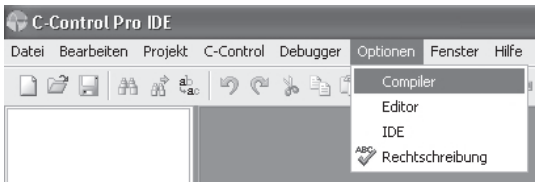


Abb. 8.4: Menüzeile – Computer

Im angezeigten Compiler-Fenster überprüfen Sie bitte, ob im Kontrollfeld „Multithreading“ ein Haken gesetzt ist. Sollte dies nicht der Fall sein, klicken Sie bitte auf das Kästchen und setzten Sie den Haken.

Im unteren Bereich klicken Sie bitte das entsprechende Optionsfeld vor der von Ihnen verwendeten C-Control-Pro-Version an. Für den MEGA32 wählen Sie bitte „C-Control 32“ für den MEGA128 „C-Control 128“.

Zur Kontrolle der Bibliothekseinstellungen klicken Sie bitte auf die Schaltfläche „Bibliothek Konfigurieren“. Hier werden die momentan aktiven Bibliotheken (Libraries) angezeigt. Libraries sind Quelltexte, die zusätzlich zu Ihrem Projekt benötigt werden. Sie enthalten Informationen über Variablen und Funktionen, die zum Ausführen der

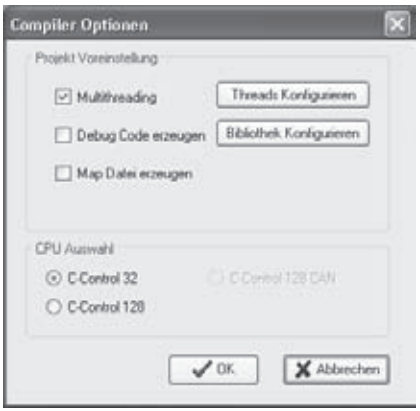


Abb. 8.5: Compiler-Optionen

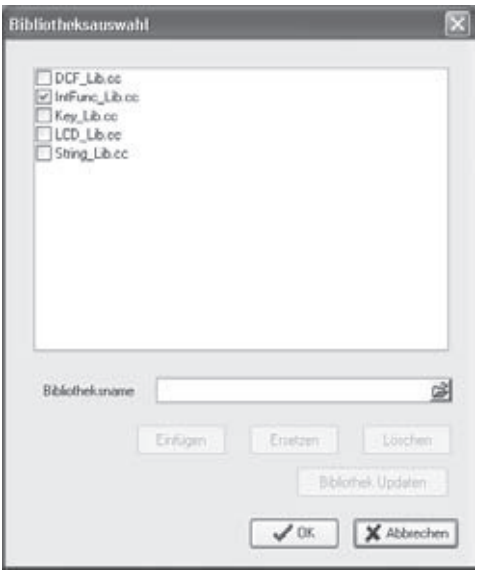


Abb. 8.6: Bibliotheksauswahl

Programme erforderlich sind. Bitte kontrollieren Sie, dass das Kontrollkästchen „IntFunc_Lib.cc“ markiert ist. Bestätigen Sie bitte die Einstellungen durch einen Klick auf die Schaltfläche „OK“.

8.3 Editor

Um die Editor-Funktionen einzustellen, klicken Sie bitte in der Menüleiste auf „Optionen“ und wählen Sie danach den Punkt „Editor“.

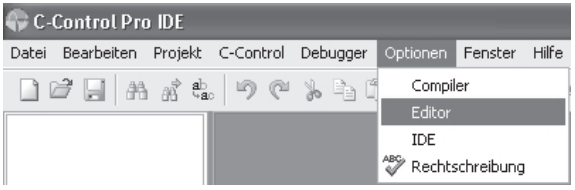


Abb. 8.7: Menüzeile Editor

Um sich das Arbeiten mit diesem Buch zu erleichtern, stellen Sie bitte die Kontrollfelder wie unten Dargestellt ein. Natürlich haben Sie die Möglichkeit, diese Auswahl nach Ihren eigenen Bedürfnissen zu ändern. Für den Anfang sollten Sie aber darauf achten, dass zumindest die Kontrollfelder „Anzeige von Zeilennummern“ und „Zeichne auf Gutter“ markiert sind. Zu finden sind diese über das Register „Anzeige“ (siehe Abb. 8.9).

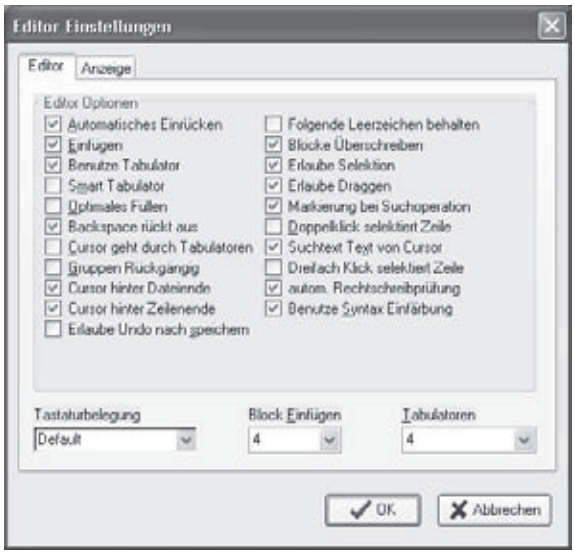


Abb. 8.8: Fenster-Einstellungen

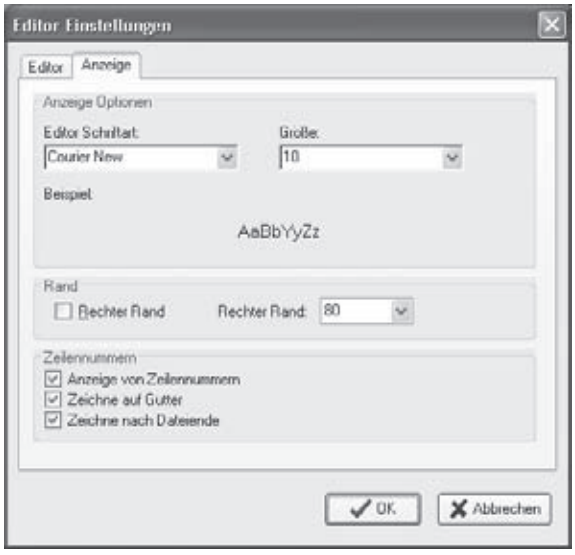


Abb. 8.9: Fenster Editor-Einstellungen

Bestätigen Sie Ihre Einstellungen durch einen Klick auf die Schaltfläche „OK“.

9 Das erste Programm

Programmierung

Bevor Sie mit dem eigentlichen Programmieren beginnen können, müssen Sie ein neues Projekt anlegen. Klicken Sie hierfür in der Menüleiste auf „Projekt“ und danach auf „Neu“.

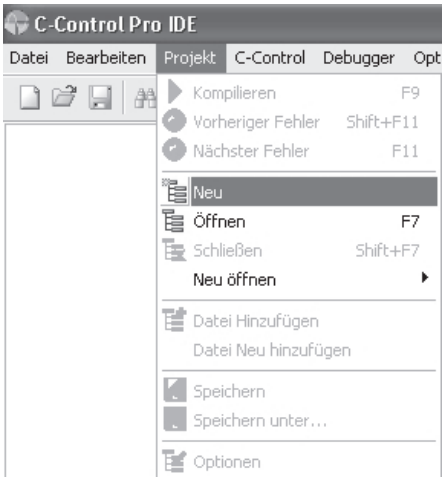


Abb. 9.1: Neues Projekt

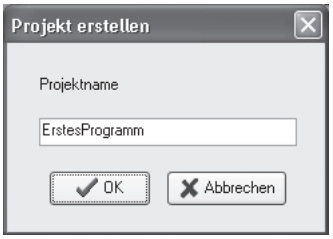


Abb. 9.2: Projektname

Da Sie die Möglichkeit haben, einzelne Programmteile modular anzulegen, d. h. Sie können einzelne Unterprogramme in verschiedenen Projekten verwenden, ist diese Vorgehensweise nötig. In einem Projekt werden alle Programmteile, die zu einer Programmierung gehören, zusammengefasst. Jetzt müssen Sie einen Namen für Ihr neues Projekt vergeben (z. B. „ErstesProgramm“). Dass hierbei keine Leerzeichen akzeptiert werden, ist gewollt, da dies später zu Problemen führen könnte.

Klicken Sie jetzt auf die Schaltfläche „OK“.

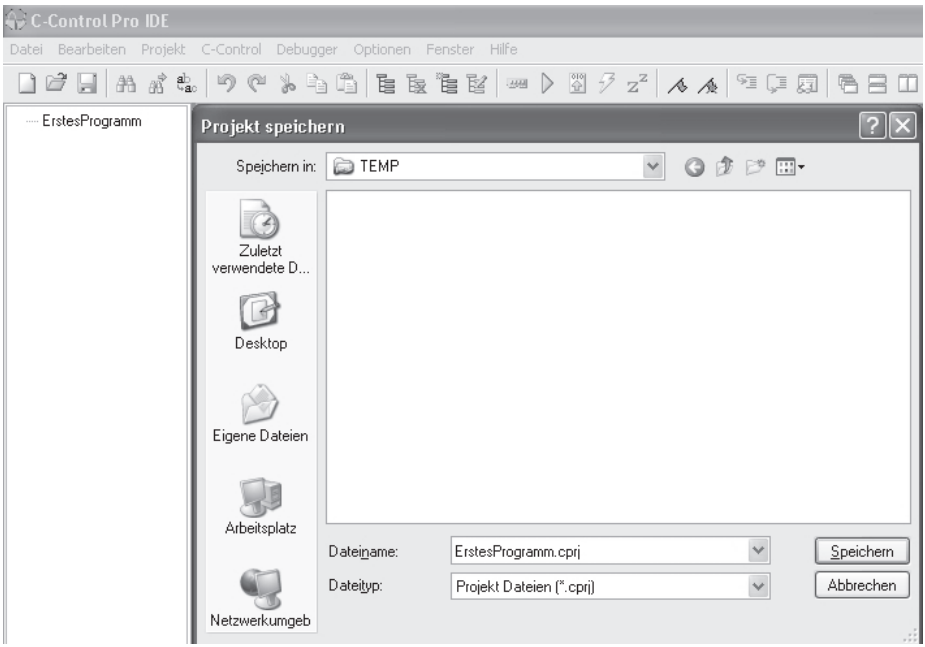


Abb. 9.3: Projekt speichern

Links oben in der Projektübersicht können Sie nun bereits den von Ihnen gewählten Projektnamen erkennen. Geben Sie bitte an, wo das Projekt abgespeichert werden soll. Beachten Sie bitte, dass Updates in das von Ihnen gewählte C-Control-Pro-Verzeichnis installiert werden. Da es unter Umständen nötig ist, dass auch die Demos entsprechend aktualisiert werden, wird bei einer Installation auch der Ordner „DemoProgramme“ überschrieben. Bedenken Sie dies bei der Speicherung Ihrer eigenen Programme und Projekte, und wählen Sie einen Ordner außerhalb des C-Control-Pro-Verzeichnisses.

Wie in Abb. 9.3 zu erkennen ist, wird das neue Projekt in diesem Fall im Ordner „TEMP“ erstellt. Klicken Sie hierfür auf die Schaltfläche „Speichern“.

Klicken Sie nun mit der rechten Maustaste auf „ErstesProgramm“ in der Projektübersicht. Im nun dargestellten Kontextmenü klicken Sie mit der linken Maustaste auf „Datei Neu hinzufügen“. Jetzt erscheint ein Fenster, in dem Sie angeben müssen, wie das Programm heißen soll, und in welcher Sprache Sie es erstellen wollen. Für den Anfang wählen Sie bitte bei Dateityp „Basic Dateien(*.cbas)“. Zur Auswahl klicken Sie auf den Pfeil des Listenfeldes, führen den Mauszeiger über die entsprechende Zeile und klicken danach auf die linke Maustaste.

Abb. 9.4: Datei erstellen

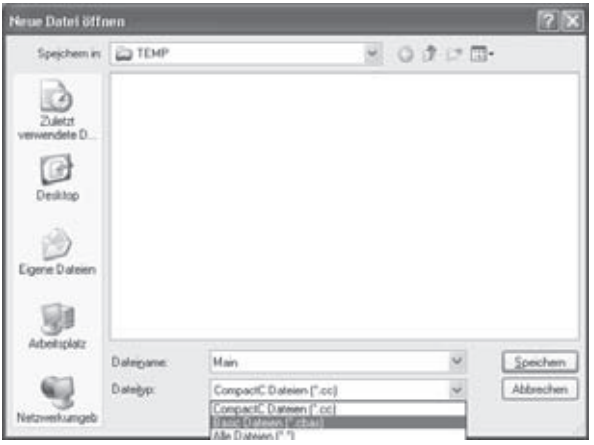


Abb. 9.5: Dateityp wählen

Um das entsprechende Programm zu speichern, müssen Sie nun auf die Schaltfläche „Speichern“ mit der linken Maustaste klicken.

Wiederholen Sie bitte diese Vorgänge auch mit den Programmen „Eingabe“ und „Ausgabe“. Am Schluss sollte die Projektübersicht folgendermaßen aussehen.

Abb. 9.6: Projektübersicht

Bevor Sie nun die ersten Zeilen eingeben, klicken Sie bitte zur Sicherheit nochmals doppelt mit der linken Maustaste auf „Main.cbas“ in der Projektübersicht. Dies ist nur nötig, um sicherzustellen, dass Sie auch das richtige Fenster angezeigt bekommen. Ist dies nicht der Fall, wird das Programm evt. nicht unter dem gewünschten Programmnamen des Projektes abgespeichert. Dies kann aber nur jetzt auftreten, da alle drei Programmteile noch ohne Inhalt sind.

Bei dem nun folgenden Programm ist es nicht erforderlich, dass zwischen MEGA32 und MEGA128 unterschieden wird. Die verwendeten Variablen „PORT_LED1“ und „PORT_SW1“ werden über die vorhandene Bibliotheksdatei „IntFunc_Lib.cc“ (siehe Kapitel 8.2 Compiler) entsprechend der verwendeten Hardware deklariert.

Geben Sie nun folgendes Programm ein. Beachten Sie bitte, dass zwischen Groß- und Kleinschreibung unterschieden wird. Alle Angaben, die hinter einem „’“ stehen, sind

Kommentare. Diese dienen nur zur Erläuterung des Programms und haben mit der Funktion nichts zu tun. Sollte es Ihnen zu viel Arbeit sein, diese Stellen einzugeben, können Sie diese auch ohne Probleme weglassen. Ist ein Programm größer und umfangreicher, ist es aber zu empfehlen, dies auch entsprechend zu kommentieren, da man sonst schnell den Überblick verliert.

```
0  ' Ein- und Ausgabe
1  ' Verwendung der Tasten SW1
2  ' LED1 leuchtet nach dem Drücken von SW1
3  ' erforderliche Libray: IntFunc_Lib.cc
4
5  ' LD1 leuchtet nach dem Drücken von SW1 für 2 Sekunden
6  ' Beide LEDs werden direkt über Einzelpinzugriff angesprochen
7
8  Dim delval As Integer                        ' globale Variablen deklarieen
9  Dim schalter1 As Byte
10
11 ' ------------------------------------------------------------------------
12 ' Hauptprogramm
13 '
14 Sub main()                                   ' Anfang des Hauptprogrammes
15     delval=2000                              ' Verzögerungszeit: 2s
16     Port_DataDirBit(PORT_LED1,PORT_OUT)      ' LED1 auf Ausgabe vorbereiten
17     Port_DataDirBit(PORT_SW1,PORT_IN)        ' SW1  auf Eingabe vorbereiten
18     Port_WriteBit(PORT_LED1,PORT_OFF)        ' LED1 ausschalten
19     Do While True                            ' Endlosschleife wird gestartet
20         SW1()                                ' Funktion SW1 wird abgefragt
21         If schalter1=0 Then                  ' Auswertung der Tasterstellung
22             LED1_On(delval)                  ' Funktion LED_On wird mit
23                                              ' Wertübergabe aufgerufen
24         End If                               ' Ende der Auswertung
25     End While                                ' Ende der Endlosschleife
26 End Sub                                      ' Ende des Hauptprogrammes
```

Abb. 9.7: Hauptprogramm

Als Nächstes geben Sie nun bitte die Programme „Eingabe“ und „Ausgabe“ ein.

Hiefür wechseln Sie durch einen Doppelklick auf den entsprechenden Namen im Projektverzeichnis in das relevante Programm.

```
0  ' ------------------------------------------------------------------------
1  ' Einlesen der Stellung des Tasters 1
2  '
3  Sub SW1()                                    ' Anfang der Funktion SW1
4      schalter1=Port_ReadBit(PORT_SW1)         ' Tasterstellung wird Ausgelesen
5  End Sub                                      ' Ende der Funktion SW1
```

Abb. 9.8: Eingabe

```
0  ' ------------------------------------------------------------------------
1  ' Ein- und Ausschalten der LED 1
2  '
3  Sub LED1_On(delay_val As Integer)            ' Anfang der Funktion LED1_ON
4                                               ' mit Variablendeklaration.
5                                               ' Die Variable delay_val erhält
6                                               ' den Wert der Variablen delay.
7                                               ' Dies erfolgt mit dem Funktions-
8                                               ' aufruf LED1_On(delval)
9      Port_WriteBit(PORT_LED1,PORT_ON)         ' Die LD1 wird eingeschaltet.
10     AbsDelay(delay_val)                      ' Die mit delval festgelegte
11                                              ' Verzögerungszeit wird
12                                              ' abgearbeitet.
13     Port_WriteBit(PORT_LED1,PORT_OFF)        ' Die LD1 wird ausgeschaltet.
14 End Sub                                      ' Ende der Funktion LED1_ON.
```

Abb. 9.9: Ausgabe

Wenn Sie die Eingaben abgeschlossen haben, klicken Sie bitte mit der linken Maustaste im Menü „Datei“ auf den Punkt „Speichern“.

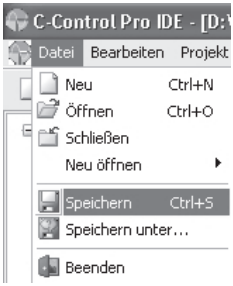


Abb. 9.10: Projekt speichern

Abb. 9.11: Projekt kompilieren

Bevor Sie Ihr Projekt auf die C-Control übertragen können, muss es noch kompiliert werden. Hierfür klicken Sie bitte mit der linken Maustaste im Menü „Projekt“ auf den Punkt „Kompilieren“.

Wenn Sie alle Eingaben korrekt durchgeführt haben, sollten im unteren Bereich der IDE folgende Angaben zu sehen sein.

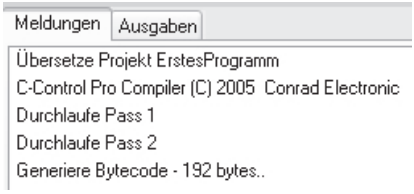


Abb. 9.12: Meldefenster

Sollte dies nicht der Fall sein, werden unter „9.2 Fehlersuche“ Methoden zum Aufspüren und Beseitigen von Fehlern erläutert.

Jetzt muss das kompilierte Projekt auf die C-Control Pro übertragen werden. Hierfür klicken Sie im Menü „C-Control“ auf den Punkt „Übertragen“. Durch ein Fenster mit Fortschrittsbalken wird die Datenübertragung dargestellt.

Abb. 9.13: Projekt Übertragung

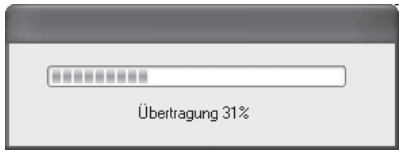


Abb. 9.14: Fortschrittsbalken

Abb. 9.15: Projekt Starten

Sobald die Übertragung abgeschlossen ist, wählen Sie im Menü „C-Control“ den Punkt „Starten“. Alternativ dazu können Sie auch die USB-Verbindung durch ausstecken des USB-Kabels trennen. Durch die Autostartfunktion der C-Control Pro wird dadurch das Projekt ebenfalls gestartet.

Wenn Sie nun auf den Taster SW1 drücken, muss die LD1 für 2 Sekunden leuchten.

Sollte dies nicht der Fall sein, werden unter „9.2 Fehlersuche“ Methoden zum Aufspüren und Beseitigen von Fehlern erläutert.

Um ein laufendes Programm zu beenden, drücken Sie bitte die Taste SW3 (Reset1) auf dem Application-Board. Unter Umständen kann es erforderlich sein, dass Sie auch die Taste SW5 (Reset2) betätigen müssen.

9.2 Fehlersuche

Die bei der Programmierung auftretenden Fehler kann man grundsätzlich in drei Gruppen aufteilen.

Syntaxfehler und Semantikfehler:
Bereits bei der Kompilierung des Programmcodes werden Fehler dieser Art erkannt. Meistens handelt es sich dabei nur um Tippfehler. In Abb. 9.16 wurde zum Beispiel die Variable „PORT_LED1“ falsch geschrieben.

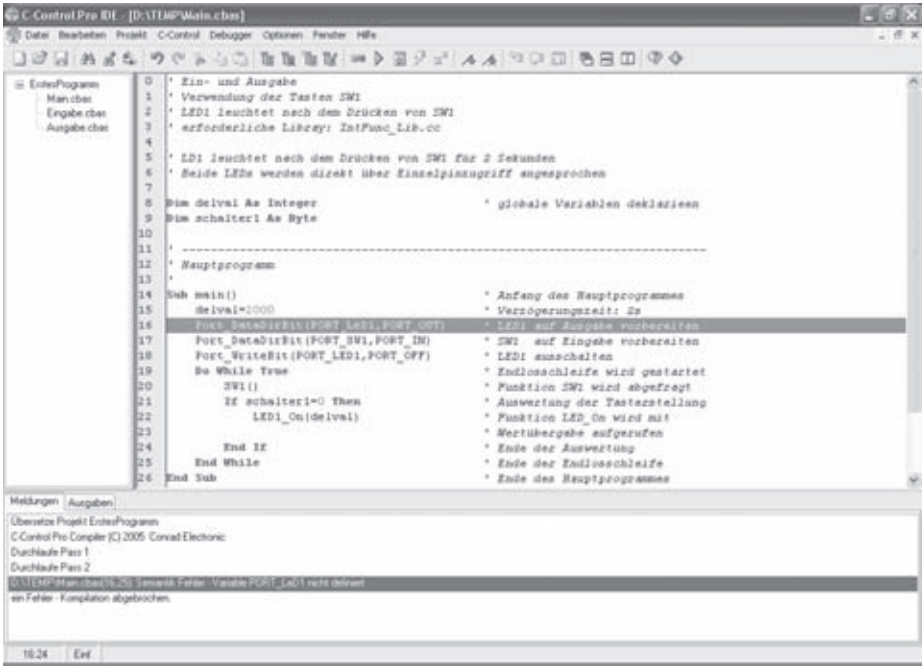


Abb. 9.16: Syntaxfehler

Weitere mögliche Syntaxfehler bzw. Semantikfehler:

- Es werden reservierte Wörter als Bezeichner verwendet (z. B. Dim, Static, Word).
- Bei der C-Programmierung wird vergessen, die Zeilen mit einem Semikolon abzuschließen.
- Reservierte Wörter werden falsch geschrieben. Hierbei ist besonders auf die unterschiedliche Schreibweise in Compact C und Compact Basic zu achten.
- Unzulässige Wertzuweisung; z. B. kann einer Bytevariablen der Wert 100 000 nicht zugewiesen werden.

Laufzeitfehler:
Fehler dieser Art werden beim Kompilieren nicht erkannt. Sie treten erst während der Ausführung des Projektes auf. Das Programm enthält zwar gültige Anweisungen, diese verursachen aber bei ihrer Ausführung Fehler und können zum Programmabbruch führen.

Es wurde darauf Wert gelegt, dass Fehler dieser Art vom Compiler weitestgehend erkannt werden und der Programmierer durch eine Warnmeldung auf einen eventuellen Programmierfehler aufmerksam gemacht wird.

Wird zum Beispiel einem Array eine Zeichenkette zugewiesen, die größer ist als die definierte Größe des Arrays, so wird eine Warnung ausgegeben. Der Programmierer

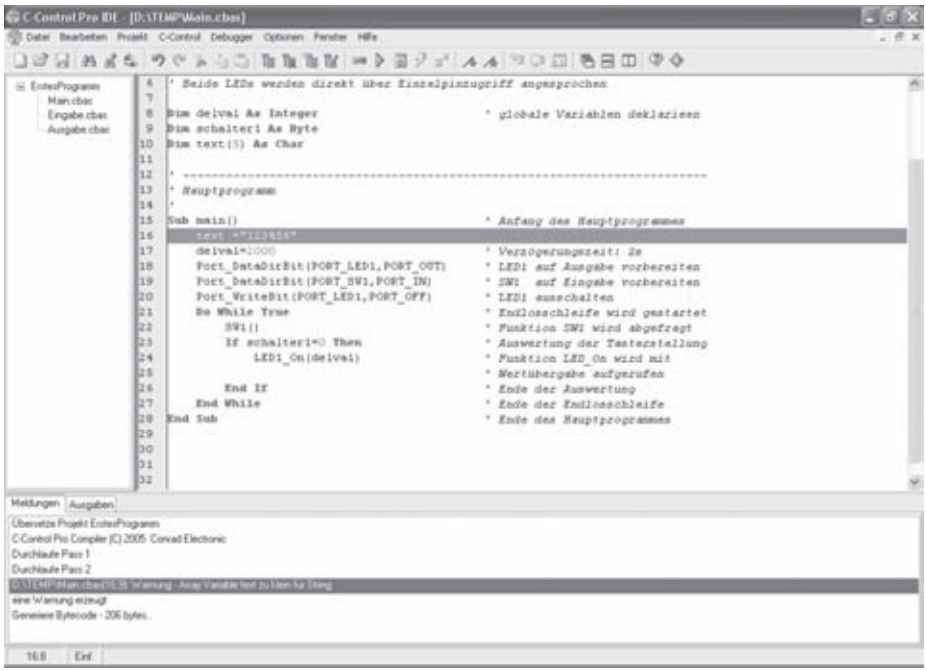


Abb. 9.17: Laufzeitfehler

muss jetzt entscheiden, ob es sich um einen Fehler handelt, oder ob dies so beabsichtigt ist. Würde „123456“ in Abb. 9.17 dem Array zugewiesen, hat dies Auswirkungen auf Variablen, die nach dem Array „text“ deklariert wurden. Die Werte dieser Variablen würden überschrieben werden.

Logische Fehler:
Die wohl unangenehmste Form der Programmierfehler sind logische Fehler. Diese Art wird weder vom Compiler noch durch Warnungen angezeigt. Festzustellen sind logische Fehler zum Beispiel durch fehlerhafte Berechnungen, falsche Portausgaben oder eine nicht erfassbare Tastatureingabe. Um solche Fehler zu lokalisieren, ist der Debugger sehr hilfreich. Die Bedienung wird unter Punkt 9.2.1 beschrieben.

9.2.1 Software

Syntaxfehler

Nach dem Sie Ihr erstes Programm eingegeben haben, und Sie es zum ersten Mal kompiliert haben, sieht das Ergebnis vielleicht wie Abb. 9.18 oder ähnlich aus.

Dies ist aber kein Grund zur Panik. Die IDE ist so programmiert, dass sie den Programmierer so weit wie möglich schon auf den entsprechenden Fehler hinweist. Na-

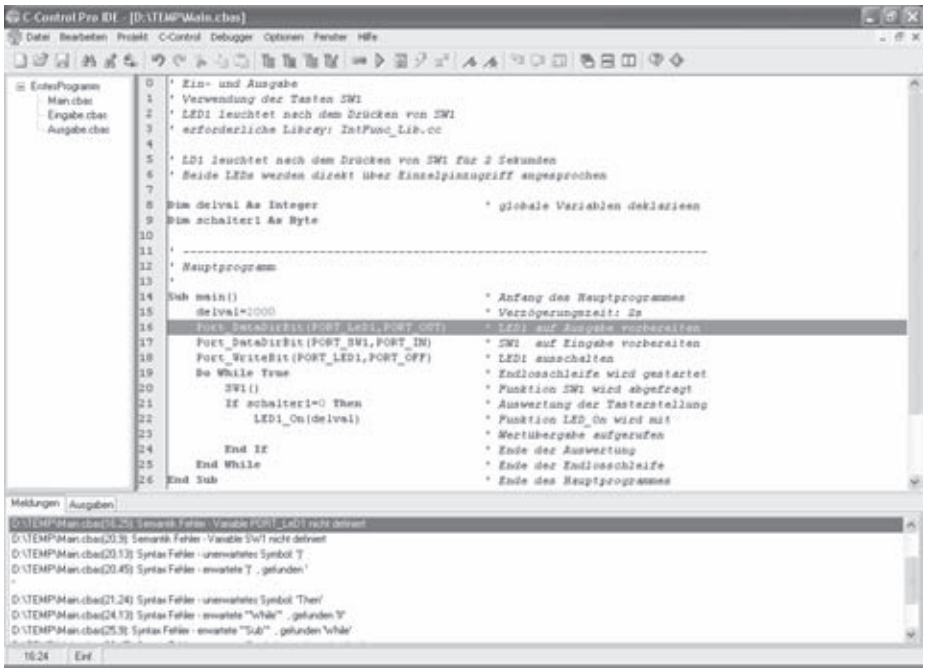


Abb. 9.18: Fehlermeldungen

türlich gibt es Fehler, die erst durch einen früheren Fehler entstehen. Aus diesem Grund ist zu empfehlen, die Fehlerliste von oben nach unten abzuarbeiten.

Von der IDE wird für dieses Vorgehen bereits der erste Fehler im Meldungsfenster und die Zeile, in der der Fehler zu finden ist, blau markiert. In diesem Fall handelt es sich nur um einen Groß-/Kleinschreibfehler. Nachdem Sie den Fehler behoben haben, speichern Sie das Projekt bitte ab und kompilieren es erneut.

Wie Sie sehen können, hat sich das Erscheinungsbild der Meldungen schon etwas geändert.

Abb. 9.19: Fehlermeldungen 2

```
0  ' ------------------------------------------------------------------------
1  ' Einlesen der Stellung des Tasters 1
2  '
3  Sub SW2()                                    ' Anfang der Funktion SW1
4      schalter1=Port_ReadBit(PORT_SW1)         ' Tasterstellung wird Ausgelesen
5  End Sub                                      ' Ende der Funktion SW1
```

Abb. 9.20: Fehlermeldungen 3

Bei dem nun angezeigten Fehler wird es etwas schwieriger. Man muss sich als Erstes überlegen, wo diese Variable „SW1“ (in diesem Fall ist es sogar eine Funktion) eigentlich deklariert wird. Da dies im Programm „Eingabe“ geschieht, wechseln Sie bitte mit einem Doppelklick auf „Eingabe.cbas“ in der Projektübersicht zu diesem Unterprogramm.

Wie Sie sicher erkennen, liegt der Fehler in diesem Fall in Zeile 3. Anstatt „SW1()“ steht hier „SW2()“. Das der Compiler hier ein Problem hat, ist völlig klar. Ändern Sie den Funktionsnamen auf „SW1()“, speichern Sie das Projekt ab und kompilieren Sie es erneut.

Wie Sie jetzt erkennen können, hat sich die Zahl der Fehlermeldungen erheblich reduziert. Dies beruht auf dem oben beschriebenen Effekt, dass es Fehlermeldungen gibt, die erst durch einen früheren Fehler entstehen.

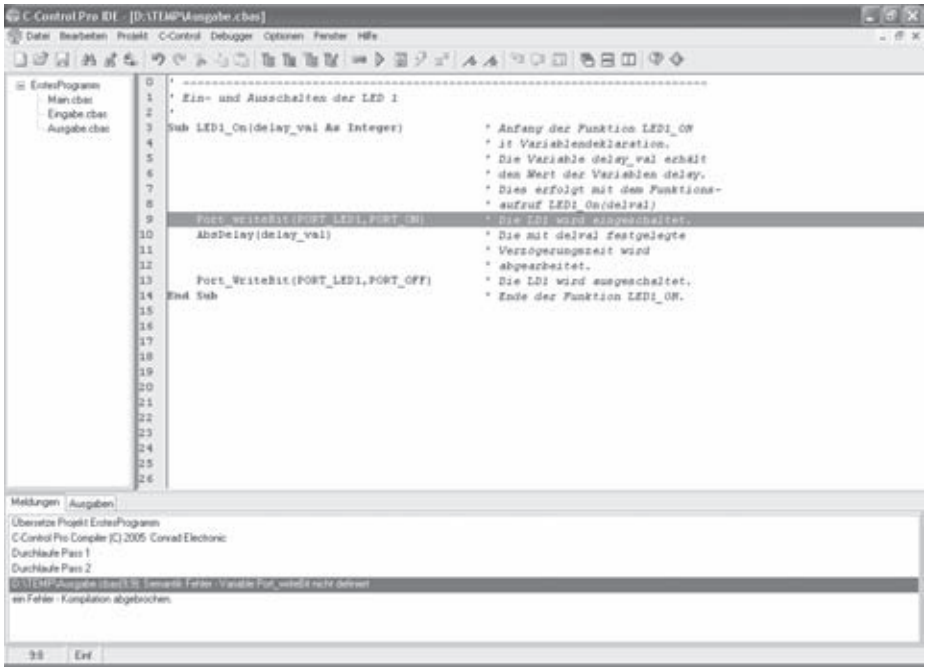


Abb. 9.21: Fehlermeldungen 4

Dieser Fehler ist wiederum nicht im eigentlichen Hauptprogramm „Main.cbas“ zu finden. Da in diesem Fall auch die Ursache der Fehlermeldung nicht im Hauptprogramm zu finden ist, wechselt die IDE automatisch zu dem entsprechenden Unterprogramm. In diesem Fall ist es das Programm „Ausgabe.cbas“. So wie beim ersten Fehler handelt es sich hierbei wieder um einen Tippfehler. Die richtige Syntax lautet: „Port_WriteBit (PORT_LED1,PORT_ON)“. Ändern Sie dies bitte entsprechend ab, speichern Sie das Projekt und kompilieren Sie es erneut.

In diesem Fall sind nun alle Fehler behoben. Da bei jeder Programmierung andere Syntax- bzw. Semantikfehler auftreten, kann dies nur ein Hinweis für die Vorgehensweise sein.

Zusammenfassend sollten Sie bei der Fehlersuche folgende Punkte überprüfen:

1. Ist die Schreibweise der Befehle richtig? (Erkannte Befehle werden grün markiert.)
2. Wurde die Groß-/Kleinschreibung beachtet?
3. Sind alle verwendeten Variablen deklariert?
4. Gibt es zu den verwendeten Funktionsnamen auch entsprechende Funktionen?
5. Wurden erforderliche Sonderzeichen (Klammern, Kommata, Semikolons) richtig verwendet?

Wenn Sie diese Punkte beachten und bei auftretenden Fehlern überprüfen, sollten Sie schnell in der Lage sein, Ihr Programm ohne Syntax- bzw. Semantikfehler zu kompilieren.

Logische Fehler

Logische Fehler sind weitaus schwieriger zu finden als Syntax- bzw. Semantikfehler. Wie der Name schon sagt, ist der Fehler nicht in einem offensichtlich falsch geschriebenen Wort zu finden, sondern beruht auf einem Denkfehler des Programmierers.

Als Erstes sollte man sich überlegen, ob das Projekt überhaupt in der vorliegenden Form rein logisch funktionieren kann. Da der Compiler bei diesen Fehlern keine Hilfestellung geben kann, ist die Fehlersuche unter Umständen sehr zeitraubend.

Die IDE der C-Control Pro bietet hierfür aber auch eine sehr gute Hilfestellung. Der integrierte Debugger hat die Fähigkeit in Echtzeit zu debuggen. Schaltzustände die über die Ports eingelesen werden, können somit auch kontrolliert werden. Drücken Sie zum Beispiel den Taster SW1, so kann diese Zustandsänderung im Debug-Modus als Wertänderung einer Variablen (z. B. „schalter1“) erkannt werden.

Grundsätzlich bietet die IDE folgende Möglichkeiten:

1. Breakpoint
2. Einzelschritt

3. Prozedurschritt
4. Variablen-Überwachung
5. Array-Überwachung
6. Tooltip-Anzeige

Um im Debug-Modus arbeiten zu können, müssen Sie aber erst einen Debug-Code erzeugen lassen.

Klicken Sie hierfür mit der rechten Maustaste auf den Namen Ihres Projektes. Im nun erscheinenden Kontextmenü wählen Sie bitte den Punkt „Optionen“.

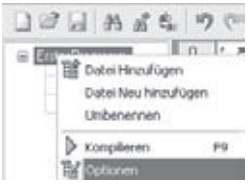


Abb. 9.22: Debug 1

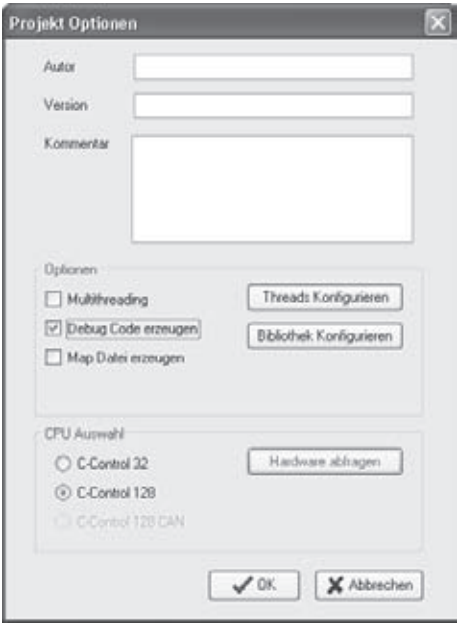


Abb. 9.23: Debug 2

Im nun angezeigten Fenster wählen Sie bitte das Kontrollfeld „Debug Code erzeugen“ durch einen Klick mit der linken Maustaste aus. Bestätigen Sie Ihre Eingabe durch einen Klick mit der linken Maustaste auf die Schaltfläche „OK“.

Kompilieren Sie nun Ihr Projekt. Zur Signalisierung, dass Sie einen Debug-Code für das aktuelle Projekt erstellt haben, wird nun in der Shortcut-Liste die Flagge blau angezeigt.

Abb. 9.24: Debug 3

Übertragen Sie nun Ihr Projekt auf die C-Control Pro und starten Sie es im Debug-Modus. Klicken Sie hierfür mit der linken Maustaste auf das Menü „Debugger“ und wählen Sie anschließend den Punkt „Debug Modus“ aus.

Wie Sie in Abb. 9.25 sehen können, wird nun die erste Zeile Ihres Projektes grün markiert. Gleichzeitig wird auch das Programm an dieser Stelle gestoppt. Dem Debugger muss nun signalisiert werden, wie er weiterarbeiten soll.

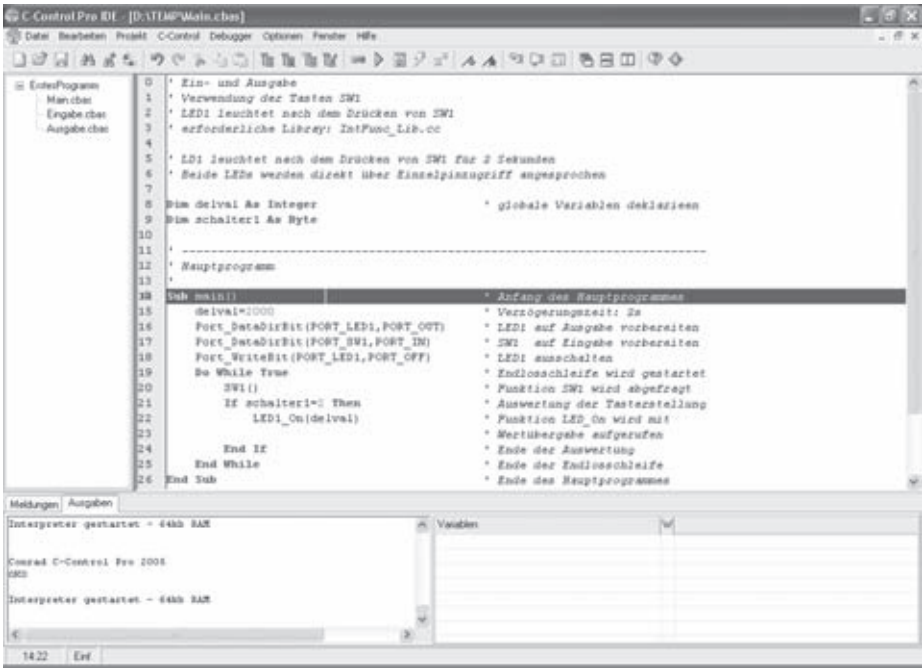


Abb. 9.25: Debug 4

Zur Simulation eines logischen Fehlers wurde in diesem Fall in Zeile 21 der Ausdruck „schalter1=1“ auf „schalter1=2“ geändert.

Zur Überprüfung des momentanen Variablenwertes stellen Sie bitte den Cursor über die gewünschte Variable. Für eine gewisse Zeit wird nun in einem Tooltip der Name, Dezimalwert und der Hexadezimalwert der Variablen angezeigt. Wenn Sie den Variablenwert dauerhaft überwachen wollen, klicken Sie bitte mit der linken Maustaste auf die gewünschte Variable. Danach klicken Sie bitte mit der rechten Maustaste. Im nun erscheinenden Kontextmenü wählen Sie bitte den Punkt „Variable Einfügen“.

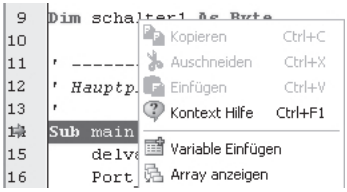


Abb. 9.26: Variablen einfügen

Wiederholen Sie diesen Vorgang bei allen Variablen, die Sie dauerhaft überwachen möchten. Stammt die Variable aus dem Hauptprogramm „Main“ ist dies im Variablenfenster daran zu erkennen, dass nur die Variable ohne Funktionsname aufgeführt ist. Wird die Variable einer Funktion überwacht, die nicht in der „Main“ zu finden ist, wird der Funktionsname und der Variablenname angezeigt. In Abb. 9.27 stammt z. B. die Variable „schalter1“ aus der „Main“ und die Variable „delay_val“ aus der Funktion „LED1_On“.

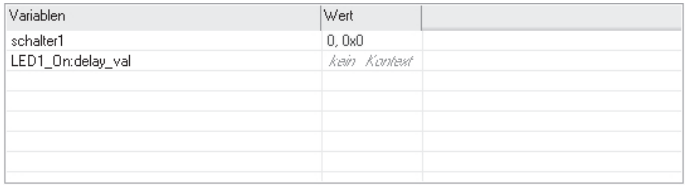


Abb. 9.27: Variablenfenster

Wird in dem Feld „Wert“ „kein Kontext“ angezeigt, bedeutet dies, dass die Variable zu diesem Zeitpunkt noch nicht deklariert worden ist.

Sie können nun das Programm Zeile für Zeile abarbeiten lassen. Wählen Sie hierfür im Menü „Debugger“ den Punkt „Einzelschritt“. Analog können Sie hierfür natürlich auch das Symbol in der Shortcut-Leiste oder die Tastenkombination „Shift+F8“ verwenden.

Abb. 9.28: Einzelschrittmodus

Wie Sie sehen können, springt die grün markierte Zeile bei jeder Betätigung des Einzelschrittes um eine Zeile weiter. Es wird also immer die Zeile markiert, die als nächstes abgearbeitet werden soll. In Zeile 20 wird nun eine Funktion aufgerufen. Wenn Sie hier im Einzelschrittmodus weitermachen, werden Sie feststellen, dass die IDE zum Unterprogramm „Eingabe.cbas“ wechselt. Nachdem das Unterprogramm durch dreimaligen Einzelschritt abgearbeitet wurde, wird wieder das Hauptprogramm angezeigt. Unter Umständen kann es sinnvoll sein, dass die Unterprogramme komplett abgearbeitet werden, ohne dass man jede Zeile einzeln bestätigen muss. Hierfür gibt es den Prozedurschritt. Dieser ist im Menü „Debugger“ und in der Shortcutleiste zu finden. Es kann natürlich auch die Taste „F8“ verwendet werden.

Lassen Sie nun die einzelnen Zeilen im Prozedurschritt abarbeiten. Wie Ihnen sicherlich aufgefallen ist, wird nun nicht in das Unterprogramm „Eingabe.cbas“ gewechselt. Die Funktion wird vielmehr komplett abgearbeitet und danach wird sofort die Zeile 21 markiert.

Wie Sie vielleicht schon gemerkt haben, hat sich nun auch der Wert der Variablen „schalter1“ im Variablenfenster geändert. Der angezeigte Wert entspricht dem momentanen Wert, der beim Abarbeiten der Funktion „SW1()“ von der Hardware ausgelesen wurde. Um dies zu verdeutlichen, halten Sie bitte den Taster „SW1“ auf dem Application-Board gedrückt und führen Sie so viele Prozedurschritte aus, dass die Zeile 20 mindestens einmal abgearbeitet wurde. Wie Sie sehen wird der Variablenwert von „1“ auf „0“ geändert.

Da es sich hierbei um einen Echtzeit-Debugger handelt, können somit alle aktuellen Portzustände direkt überprüft werden.

Bei diesem Beispielfehler können Sie nun feststellen, dass der Wert der Variablen „schalter1“ nur zwischen „0“ und „1“ wechselt. Die in Zeile 21 programmierte If-Anweisung wird also nie erfüllt werden können. Beenden Sie bitte den Debug-Modus, indem Sie im Menü „Debugger“ den Punkt „Debug Modus verlassen“ wählen.

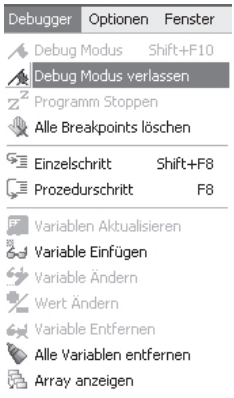


Abb. 9.29: Debug-Modus verlassen

Ändern Sie nun den Ausdruck wieder auf „schalter1=0“, speichern Sie das Projekt ab, kompilieren Sie es, übertragen Sie es und starten Sie den Debug-Modus erneut.

Lassen Sie das Programm wieder Schritt für Schritt im Prozedurmodus abarbeiten. Wie Sie sehen, wird aus der Hardware der Wert „1“ für den nicht gedrückten Schalter ausgelesen. Betätigen Sie nun den Schalter „SW1“ und lassen Sie vom Programm die Zeile 20 abarbeiten. Die Variable „schalter1“ hat nun den Wert 0 angenommen. Den Schalter „SW1“ können Sie nun wieder loslassen. Fahren Sie nun im Einzelschritt-Modus fort. Wie Sie sehen, wird nun die Zeile 22 markiert und die Funktion „LED1_On“ kann nun bearbeitet werden. Im Variablenfenster können Sie sehen, dass sich der Wert der Variablen „delay_val“ mit dem Aufruf der Funktion „LED1_On“ nun auch auf den zugewiesenen Wert „2000“ geändert hat. Durch den Befehl in Zeile 9 wird nun die LD1 eingeschaltet. Die Zeit läuft aber erst ab, wenn Zeile 10 abgearbeitet wird. Mit dem Befehl in Zeile 13 wird die LD1 wieder ausgeschaltet.

Da das Programm nun fehlerfrei funktioniert, können Sie den Debug-Modus wieder verlassen und das Projekt normal starten. Im Variablenfenster werden zwar noch immer die vorher überwachten Variablen angezeigt, eine Aktualisierung erfolgt aber nur im Debug-Modus.

Wenn Sie mit der rechten Maustaste in das Variablenfenster klicken, können Sie mit dem Punkt „Alle Variablen entfernen“ den Inhalt dieses Fensters wieder löschen.

Sollten in Ihrem Programm Arrays zur Anwendung kommen, können Sie diese natürlich in ähnlicher Form überwachen.

Klicken Sie bitte mit der linken Maustaste auf das gewünschte Array. Danach klicken Sie bitte mit der rechten Maustaste. Im nun erscheinenden Kontextmenü wählen Sie bitte den Punkt „Array anzeigen“ (siehe Abb. 9.26).

Wird ein Array z. B. vom Typ „Char“ deklariert, sehen die Inhalte anfangs wie Abb. 9.30 aus. Nachdem Werte zugewiesen wurden, ändert sich die Anzeige z. B. wie in Abb. 9.31. Zur Sicherheit sollten Sie aber jeweils die Schaltfläche „Aktualisieren“ im Array-Fenster betätigen.

Des Weiteren haben Sie die Möglichkeit Breakpoints zu setzen. Dies bedeutet, dass das Programm bis zum ersten Breakpoint abgearbeitet und dann gestoppt wird.

Beachten Sie, dass Breakpoints nur bei Zeilen gesetzt werden können, die Befehle enthalten. Bei Kommentarzeilen ist dies nicht möglich. Setzen Sie z. B. einen Breakpoint in Zeile 13 des Unterprogramms „Ausgabe“. Klicken Sie hierfür mit der linken Maustaste auf die 13. Ein gesetzter Breakpoint wird Ihnen durch einen roten Punkt und durch die rote markierte Zeile signalisiert (siehe Abb. 9.32).

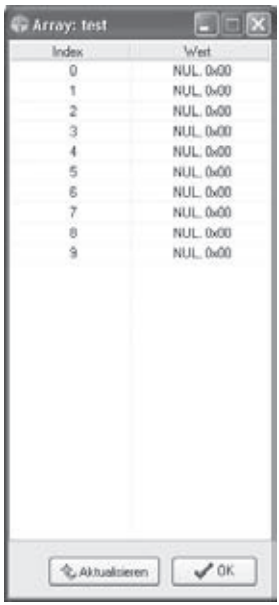


Abb. 9.30: Array deklariert

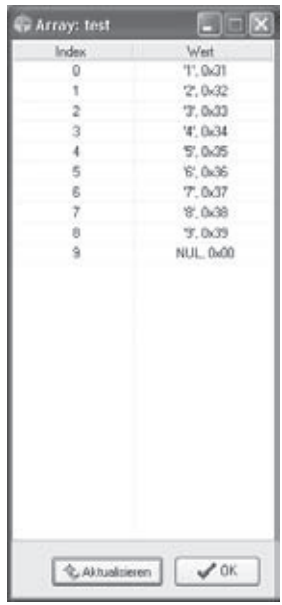


Abb. 9.31: Array mit Werten

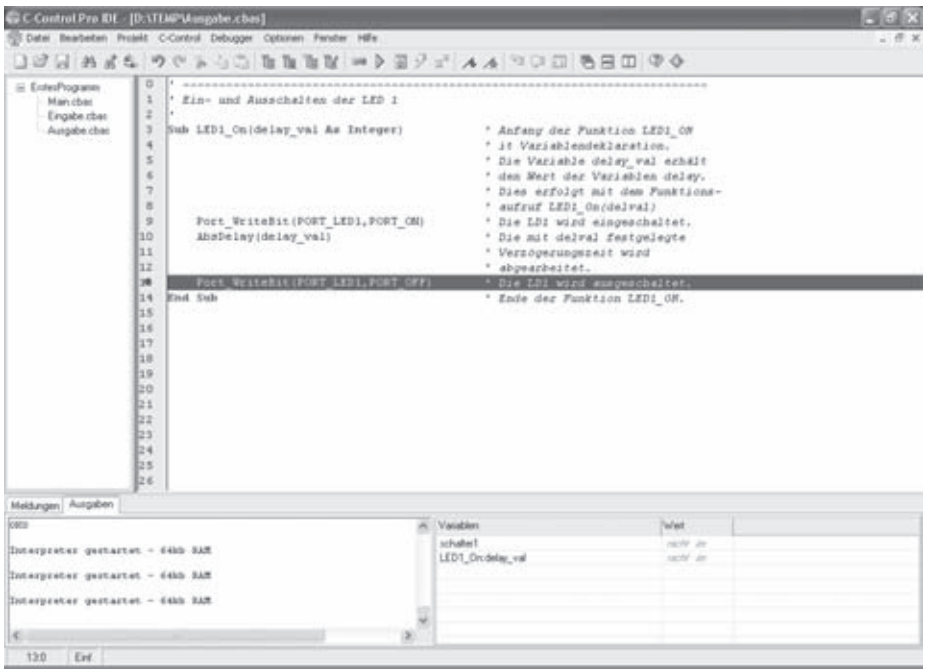


Abb. 9.32: Breakpoint

Speichern Sie nun das Projekt ab, kompilieren Sie es, übertragen Sie es und starten Sie den Debug-Modus. Drücken Sie nun die Taste „SW1“. Das Projekt wird bis zur Zeile 13 des Unterprogramms „Ausgabe“ abgearbeitet und dann gestoppt. Dies können Sie zum einen daran erkennen, dass die LD1 nun leuchtet, zum anderen dass die Zeile 13 nun grün eingefärbt ist.

Abb. 9.33: Breakpoint 2

Sie können nun im Einzelschritt oder Prozedurschritt mit der Überprüfung Ihres Programms fortfahren oder den Debug-Modus beenden. Durch das Setzen von Breakpoints wird das Auffinden von Fehlern vor allem dann erleichtert, wenn man sicher ist, dass bestimmte Programmbereiche bereits fehlerfrei sind. Diese müssen dann nicht umständlich im Einzelschritt oder Prozedurschritt abgearbeitet werden. Durch einen modularen Aufbau der Projekte (siehe „ErstesProgramm“) können somit bereits als funktionierend bewertete Module schnell abgearbeitet werden.

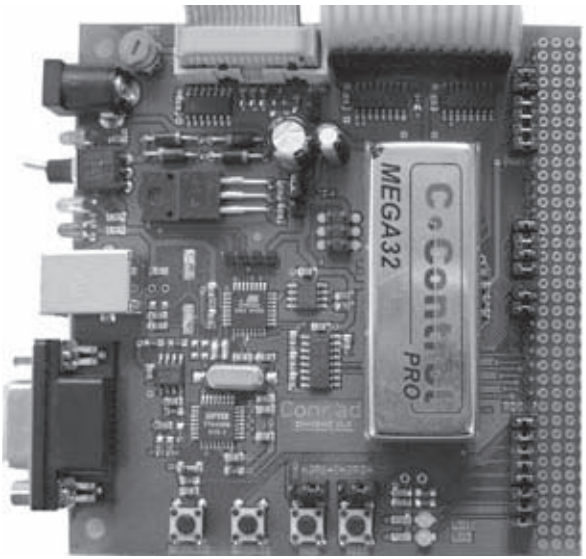
9.2.2 Hardware

Natürlich ist es möglich, dass ein Projekt auch mit einwandfreier Software noch nicht funktioniert. Der Fehler ist in diesem Fall bei der Hardware zu suchen.

Bei jedem Projekt müssen Sie sich im Klaren sein, welche Ausgänge und Eingänge (Ports) Sie verwenden. In der Hilfedatei der IDE finden Sie im Kapitel „Hardware“ unter „Mega32“ und „Mega128“ jeweils die Punkte „Pinzuordnung“ und „Jumper Application“. In diesen Bereichen wird genau beschrieben, welche Jumper für welchen Port, Schalter und LED zuständig sind.

Bei dem oben beschriebenen Programm „ErstesProgramm“ sind z. B. beim MEGA32 folgende Jumper erforderlich (siehe Abb. 9.34).

Bei dem Application-Board des MEGA128 sind folgende Jumper erforderlich (siehe Abb. 9.35).

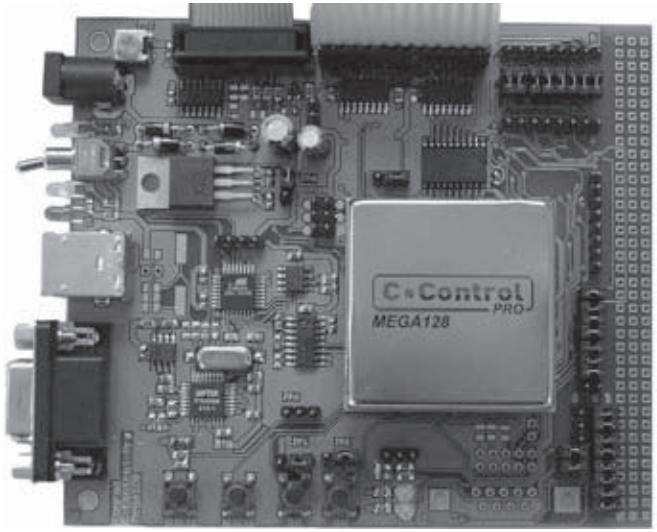


JP1 muss gegen GND schalten, d. h. der Jumper muss rechts gesteckt sein

PD2 muss für den Schalter SW1 gesteckt sein

PD6 muss für die LD1 gesteckt sein

Abb. 9.34: Jumper MEGA32



JP1 muss gegen GND schalten, d. h., der Jumper muss rechts gesteckt sein

PD4 muss für den Schalter SW1 gesteckt sein

PG3 muss für die LD1 gesteckt sein

Abb. 9.35: Jumper MEGA128

10 C und Basic in einem Projekt

Die IDE der C-Control Pro bietet die Möglichkeit, in einem Projekt Programme in C und Basic zu verwenden. Das hat den Vorteil, dass Module der verschiedenen Sprachen ohne zeitraubende Übersetzungsarbeit miteinander kombiniert werden können.

Als Beispiel ist nachfolgend das komplette Beispielprogramm aus „9 Das Erste Programm“ in C dargestellt. Geben Sie dieses bitte wie unter Punkt 9 beschrieben ein.

Der Unterschied bei der Eingabe liegt bei Abb. 9.5. Hier wählen Sie bitte als Dateityp „CompactC Dateien(*.cc)“. Als Projektnamen verwenden Sie bitte „ZweitesProgramm“.

Die Projektübersicht sollte am Schluss aussehen wie in Abb. 10.1.

Abb. 10.1: Zweites Programm

```
0  // Ein- und Ausgabe
1  // Verwendung der Tasten SW1
2  // LED1 leuchtet nach dem Drücken von SW1
3  // erforderliche Library: IntFunc_Lib.cc
4
5  // LD1 leuchtet nach dem Drücken von SW1 für 2 Sekunden
6  // Beide LEDs werden direkt über Einzelpinzugriff angesprochen
7
8  int delval;                              // globale Variablen deklarieen
9  byte schalter1;
10
11 // ---------------------------------------------------------------------
12 // Hauptprogramm
13 //
14 void main(void)
15 {                                        // Anfang des Hauptprogrammes
16     delval=2000;                         // Verzögerungszeit: 2s
17     Port_DataDirBit(PORT_LED1,PORT_OUT); // LED1 auf Ausgabe vorbereiten
18     Port_DataDirBit(PORT_SW1,PORT_IN);   // SW1  auf Eingabe vorbereiten
19     Port_WriteBit(PORT_LED1,PORT_OFF);   // LED1 ausschalten
20     while(1)                             // Endlosschleife wird gestartet
21     {
22         SW1();                           // Funktion SW1 wird abgefragt
23         if (schalter1==0)                // Auswertung der Tasterstellung
24         {
25             LED1_On(delval);             // Funktion LED_On wird mit
26                                          // Wertübergabe aufgerufen
27         }                                // Ende der Auswertung
28     }                                    // Ende der Endlosschleife
29 }                                        // Ende des Hauptprogrammes
```

Abb. 10.2: Zweites Programm (Main)

```
0  // ------------------------------------------------------------------------
1  // Einlesen der Stellung des Tasters 1
2  //
3  void SW1(void)                                  // Anfang der Funktion SW1
4  {
5      schalter1=Port_ReadBit(PORT_SW1);           // Tasterstellung wird Ausgelesen
6  }                                               // Ende der Funktion SW1
```

Abb. 10.3: Zweites Programm (Eingabe)

```
0  // ------------------------------------------------------------------------
1  // Ein- und Ausschalten der LED 1
2  //
3  void LED1_On(int delay_val)                     // Anfang der Funktion LED1_ON
4  {                                               // mit Variablendeklaration.
5                                                  // Die Variable delay_val erhält
6                                                  // den Wert der Variablen delay.
7                                                  // Dies erfolgt mit dem Funktions-
8                                                  // aufruf LED1_On(delval)
9      Port_WriteBit(PORT_LED1,PORT_ON);           // Die LD1 wird eingeschaltet.
10     AbsDelay(delay_val);                        // Die mit delval festgelegte
11                                                 // Verzögerungszeit wird
12                                                 // abgearbeitet.
13     Port_WriteBit(PORT_LED1,PORT_OFF);          // Die LD1 wird ausgeschaltet.
14 }                                               // Ende der Funktion LED1_ON.
```

Abb. 10.4: Zweites Programm (Ausgabe)

Um nun Module aus den Programmen „ErstesProgramm“ und „ZweitesProgramm“ zu mischen, gehen Sie folgendermaßen vor. Schließen sie alle momentan geöffneten Projekte. Erstellen Sie ein neues Projekt. Verwenden Sie z. B. den Namen „Drittes-Programm“. Klicken Sie mit der rechten Maustaste auf den Projektnamen „Drittes-Programm“. Im erscheinenden Kontextmenü wählen Sie bitte den Punkt „Datei Hinzufügen“.

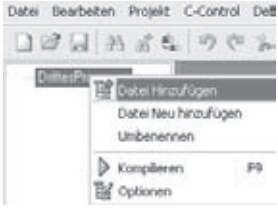


Abb. 10.5: Datei Hinzufügen

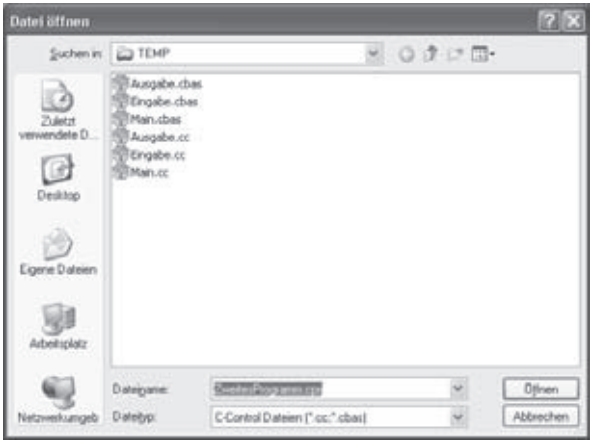


Abb. 10.6: Datei öffnen

Im nächsten Fenster können Sie nun die gewünschten Programmmodule Ihrem Projekt hinzufügen. Wählen Sie zum Beispiel „Main.cc“, „Ausgabe.cbas“ und „Eingabe.cbas“

Wenn Sie nun das Projekt kompilieren und auf die C-Control Pro übertragen, können Sie feststellen, dass die Programme ohne Probleme miteinander funktionieren.

Da die Funktionen der Programme „Main.cc“ und „Main.cbas“, „Ausgabe.cc“ und „Ausgabe.cbas“, „Eingabe.cc“ und „Eingabe.cbas“ in den Projekten „ErstesProgramm“ und „ZweitesProgramm“ jeweils gleich sind, macht es natürlich nur Sinn, Unterprogramme mit verschiedenen Namen zu kombinieren („Main“, „Ausgabe“, „Eingabe“). In den Programmen werden Funktionen mit den gleichen Namen verwendet, dadurch würde die Kombination von z. B. „Main.cc“ und „Main.cbas“ zu Fehlermeldungen führen.

11 Schutz der Programme (PIN)

Der MEGA32 und der MEGA128 bieten die Möglichkeit, die eigenen Projekte vor versehentlichem bzw. unbefugtem Zugriff zu schützen. Den hierfür erforderlichen PIN-Code können Sie folgendermaßen vergeben.

Klicken Sie im Menü „C-Control“ auf den Punkt „PIN Setzen“

Abb. 11.1: PIN setzen

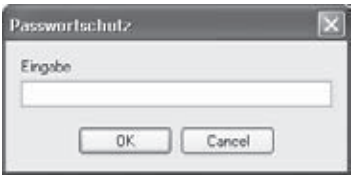


Abb. 11.2: Code eingeben

Im nun erscheinenden Fenster können Sie einen maximal 6-stelligen PIN-Code eingeben.

Um einen PIN-Code vergeben zu können, darf in diesem Moment kein Programm im MEGA32 oder MEGA128 ablaufen. Ist dies der Fall, wird die Meldung Abb. 11.3 ausgegeben.

Abb. 11.3: PIN-Zuweisung nicht erfolgt

Abb. 11.4: PIN-Zuweisung erfolgreich

Beenden Sie das laufende Programm, indem Sie auf den Taster „Reset1“ drücken und weisen Sie den PIN-Code erneut zu. Mit der Meldung Abb. 11.4 wird eine erfolgreiche PIN-Zuweisung bestätigt. Klicken Sie auf die Schaltfläche „OK“, um die Meldung zu bestätigen.

Dass der MEGA32 bzw. MEGA128 PIN-geschützt ist, können Sie daran erkennen, dass das Auslesen der Hardwareversion, das Programmübertragen und das Programmstarten mit folgender Meldung quittiert werden.

Abb. 11.5: Code eingeben

Das momentan im MEGA32 bzw. MEGA128 vorhandene Projekt kann ohne die Eingabe des PIN-Codes nur ohne USB-Verbindung gestartet werden. Wird die C-Control Pro nach dem Programmieren in der eigenen Anwendung eingebaut, startet das Projekt sobald die Versorgungsspannung angeschlossen wird.

Um einen PIN-geschützten C-Control Pro neu programmieren zu können, gibt es zwei Möglichkeiten. Wenn Sie den PIN-Code wissen, klicken Sie bitte im Menü „C-Control“ auf den Punkt „PIN Eingeben“.

Abb. 11.6: PIN eingeben

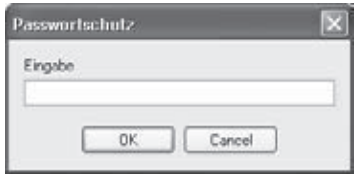


Abb. 11.7: Code eingeben

Im nun erscheinenden Fenster geben Sie bitte den entsprechenden PIN-Code ein.

Beachten Sie hierbei die Groß- und Kleinschreibung. Haben Sie einen falschen PIN-Code eingegeben, wird dies durch Abb. 11.8 angezeigt.

Abb. 11.8: falscher PIN-Code

Dass der MEGA32 bzw. MEGA128 nicht mehr PIN-geschützt ist, können Sie daran erkennen, dass das Auslesen der Hardwareversion, das Programmübertragen und das Programmstarten wieder möglich ist.

Sollte es passieren, dass Sie den PIN-Code der C-Control Pro vergessen haben, besteht die Möglichkeit, das Modul zurückzusetzen. Dies bedeutet aber, dass das momentan im MEGA32 bzw. MEGA128 gespeicherte Projekt ebenfalls gelöscht wird. Wählen Sie hierfür im Menü „C-Control“ den Punkt „Modul zurücksetzen“ aus.

Abb. 11.9: Modul zurücksetzen

Abb. 11.10: Abfrage-Modul zurücksetzen

Wie bereits oben erwähnt, bedeutet das Zurücksetzten des Moduls, dass auch das momentan in der C-Control Pro vorhandene Programm gelöscht wird. Aus diesem Grund erfolgt nochmals eine Abfrage, ob Sie dies wirklich machen wollen. Wenn Sie dies mit einem Klick auf die Schaltfläche „Ja“ bestätigen, wird Ihre Entscheidung mit der Meldung wie in Abb. 11.11 bestätigt.

Abb. 11.11: Modul zurückgesetzt

Jetzt ist die C-Control Pro wieder im Auslieferzustand. Sie können jetzt mit der Übertragung von neuen Projekten beginnen.

Wenn Sie nur den PIN-Code löschen wollen, so ist dies natürlich auch möglich.

Geben Sie hierfür den momentanen PIN-Code ein. Wählen Sie hierfür im Menü „C-Control“ den Punkt „PIN eingeben“.
Danach wählen Sie im Menü „C-Control“ den Punkt „PIN Löschen“. Mit der Meldung Abb. 11.14 wird bestätigt, dass der PIN-Code gelöscht wurde.

Abb. 11.12: PIN eingeben

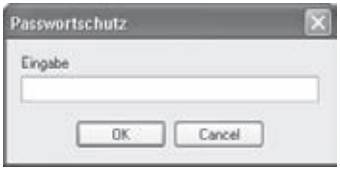


Abb. 11.13: Code eingeben

Abb. 11.14: PIN gelöscht

Jetzt stehen wieder alle Funktion wie vor der PIN-Eingabe zur Verfügung. Beachten Sie, dass die Projekte in der C-Control Pro nun nicht mehr geschützt sind und somit auch versehentlich geändert werden können.

12 Anschluss externer Komponenten

12.1 DCF-Modul

Mit diesem Zusatzmodul verfügt die C-Control Pro über eine so genante DCF-Funktion, d. h. Uhrzeit und Datum werden automatisch durch den Empfang eines Funksignals (DCF77-Signals) durch eine Atomuhr synchronisiert. Dieses Signal wird durch die „Physikalisch Technische Bundesanstalt“ ausgestrahlt. Der Sender für das DCF77-Signal sendet mit einer Frequenz von 77,5 kHz und befindet sich in Mainflingen bei Frankfurt am Main. Das Signal wird kontinuierlich ausgestrahlt und hat eine Reichweite von ca. 1500 km. Befinden Sie sich in diesem Bereich, kann die C-Control Pro mit Hilfe des Demoprogramms „DCF_RTC.cc“ oder „DCF_RTC.cbas“ die Einstellung der Uhrzeit und des Datums selbstständig durchführen. Es steht Ihnen somit eine äußerst genaue Uhrzeitanzeige zur Verfügung.

Die Amplitude des Signals wird jede Sekunde um 25% gesenkt. Die Rückgangsdauer beträgt ca. 50 ms bzw. ca. 150 ms. Abhängig von dieser Dauer kann eine 0 oder eine 1 zugeordnet werden. Die aktuelle Uhrzeit, Datum und einige Prüfbits werden in 59 Bits pro Minute übertragen. Über das Fehlen der Absenkung in der letzten Sekunde wird der Anfang einer Minute signalisiert.

Damit die DCF-Synchronisation ermöglicht werden kann, muss das DCF-Modul mit dem Application-Board verbunden werden. Standardmäßig wird beim MEGA32 Port

Abb. 12.1: DCF-Modul (Best.-Nr. 641138)

D.7 und beim MEGA128 Port F.0 verwendet. Von der C-Control Pro wird das nicht invertierte Signal der DCF-Empfängerplatine verarbeitet. Dies kann an Pin 3 abgegriffen werden.

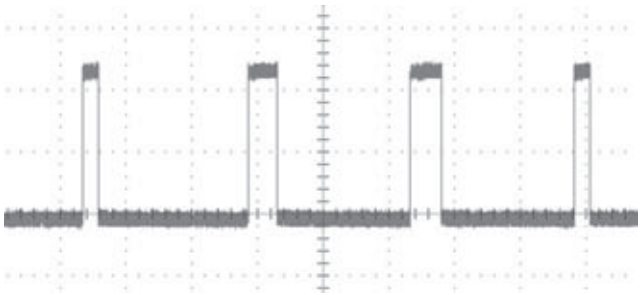


Abb. 12.2: DCF-Signal an Pin 3

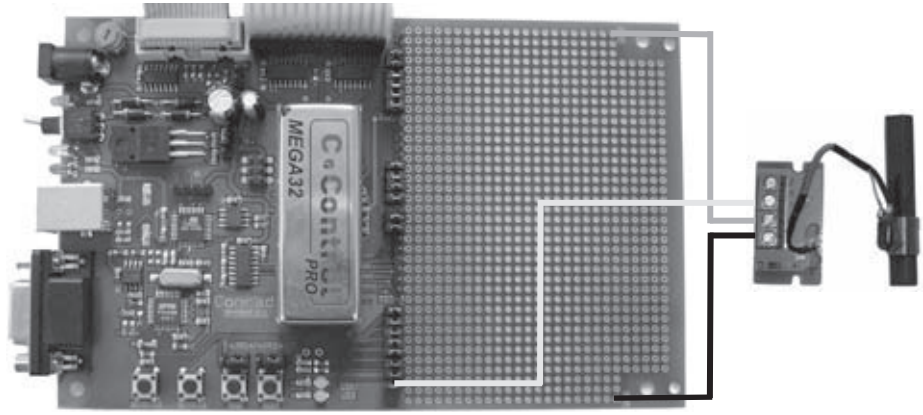


Abb. 12.3: MEGA32 mit DCF-Modul

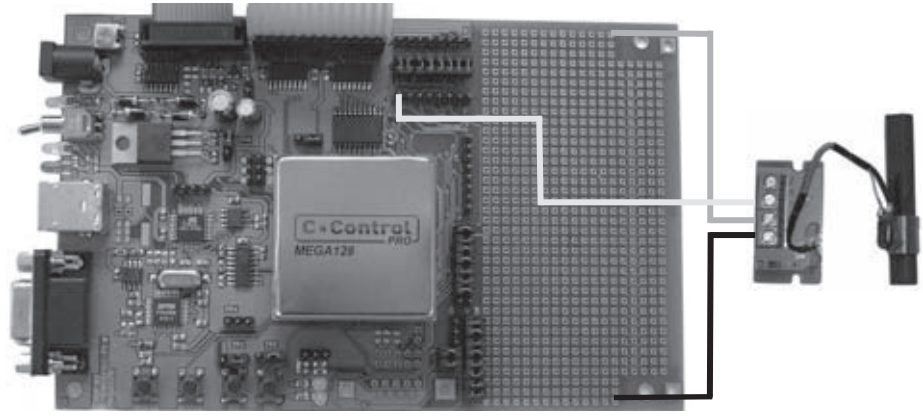


Abb. 12.4: MEGA128 mit DCF-Modul

Sollte auf dem Display nicht die richtige Zeit angezeigt werden, z. B. 35:27:15, überprüfen Sie bitte, ob nicht elektronische Geräte wie Monitore, Computer oder dergleichen den Empfang stören. Bei Synchronisationsproblemen sollten Sie die DCF-Platine weit von den oben angesprochenen Geräten entfernt anbringen. Evt. ist es auch möglich, dass bauliche Voraussetzungen wie Stahlbetonbauten oder metallurgisch behandelte Materialien zu einer Empfangsverschlechterung führen.

Wenn Ihnen die standardmäßigen Eingänge nicht zusagen, da Sie diese vielleicht anders verwenden möchten, ist es natürlich möglich, diese zu ändern.

Öffnen Sie hierfür mit der IDE die Datei „DCF-Lib.cc“. Zu finden ist diese bei der Standardinstallation unter Programm\C-Control Pro\Libraries. Es ist zu empfehlen, dass Sie, bevor Sie Änderungen in dieser Bibliothek vornehmen, eine Sicherheitskopie von dieser Datei anfertigen.

```
0  /*****************************************************************************
1   DCF77 Library
2  *****************************************************************************/
3  #ifdef MEGA32
4  // DCF Input Signal at PortD.7
5  #define DCF_IN 31
6  #endif
7
8  #ifdef MEGA128
9  // DCF Input Signal at PortF.0
10 #define DCF_IN 40
11 #endif
12
```

Abb. 12.5: DCF Lib

In der Zeile 5 wird der Port für den MEGA32 festgelegt und in der Zeile 10 der Port für den MEGA128. Zur Wahl des neuen Ports verwenden Sie bitte die Pinzuordnung in der Hilfe-Datei. Beachten Sie bitte, dass durch die teilweise Mehrfachbelegung der Ports nicht jeder Port verwendet werden kann. Abhängig ist dies von den jeweils benötigten Komponenten des entsprechenden Projektes.

Um eine Änderung durchzuführen, ändern Sie z. B. den Wert 31 in Zeile 5 auf 30. Speichern Sie nun die Änderungen über den Menüpunkt „Datei“, „Speichern“ ab. Wenn Sie nun das DCF-Demo auf dem MEGA32 starten, werden Sie feststellen, dass die Uhrzeit nicht mehr synchronisiert wird. Dies liegt daran, dass der Eingang nun auf Port D.6 liegt. Ändern Sie entsprechend die Verbindungen auf dem Board. Nun wird die Zeit wieder richtig angezeigt.

LCD-Display 4 × 20

Wenn Sie ein anderes Display als das im Lieferumfang enthaltene verwenden wollen, muss es folgende Voraussetzungen erfüllen.

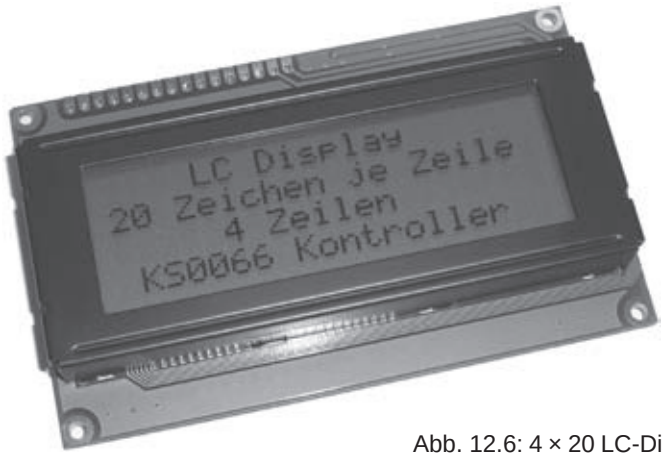


Abb. 12.6: 4 × 20 LC-Display (Best.-Nr. 187275)

PIN	Symbol	Level	Funktion
1	VSS	L	Stromversorgung 0 V (GND)
2	VDD	H	Stromversorgung +5 V
3	VEE	–	Kontrastspannung
4	RS (CS)	H / L	Umschaltung Befehl / Daten
5	R/W (SID)	H / L	H = Lesen, L = Schreiben
6	E (SCLIK)	H	Enable (fallende Flanke)
7	D0 (SOD)	H / L	Display Data, LSB
8	D1	H / L	Display Data
9	D2	H / L	Display Data
10	D3	H / L	Display Data
11	D4	H / L	Display Data
12	D5	H / L	Display Data
13	D6	H / L	Display Data
14	D7	H / L	Display Data, MSB
15	A	–	LED-Beleuchtung + (Vorwiderstand erforderlich)
16	K	–	LED-Beleuchtung –

Des Weiteren ist es wichtig, dass es einen HD44780 bzw. KS0066 Controller besitzt. Ist dies nicht der Fall, kann es sein, dass Sie selbst spezielle Initialisierungssequenzen schreiben müssen. Ist einer der oben genannten Controller verbaut, können Sie die vorhandenen C-Control-Pro-Befehle verwenden.

Durch das folgende kleine Programm wird die erste Zeile des LC-Displays jeweils mit der Zahl eins, die zweite Zeile jeweils mit der Zahl zwei, die dritte mit der Zahl drei und die vierte mit der Zahl vier gefüllt.

Da das verwendete Modul mit einem KS0066-Controller ausgestattet ist, können die Funktionen „LCD_Init()“, „LCD_ClearLCD()“ und „LCD_CursorOff()“ ohne Änderungen in der „LCD_Lib.cc“ verwendet werden.

Erforderliche Bibliotheken: „IntFunc_Lib.cc“, „LCD_Lib.cc“

Wenn Sie das Demo eintippen, vergessen Sie bitte nicht, dass nun zusätzlich zur „IntFunc_Lib.cc“ auch die „LCD_Lib.cc“ erforderlich ist. Wie diese aktiviert wird, können Sie im Kapitel „8.2 Compiler“ nachlesen.

```
0  //--------------------------------------------------------------------------------
1  // Hauptprogramm
2  //
3
4  int i;
5
6  void main(void)
7  {
8      LCD_Init();                          // Display initialisieren
9      LCD_ClearLCD();                      // Display löschen
10     LCD_CursorOff();                     // Display Cursur ausschalten
11
12     for (i=0;i<20;i++)
13     {
14         LCD_CursorPos(0x00+i);           // 1. Zeile
15         LCD_WriteChar(49);
16     }
17     for (i=0;i<20;i++)
18     {
19         LCD_CursorPos(0x40+i);           // 2. Zeile
20         LCD_WriteChar(50);
21     }
22     for (i=0;i<20;i++)
23     {
24         LCD_CursorPos(0x14+i);           // 3. Zeile
25         LCD_WriteChar(51);
26     }
27     for (i=0;i<20;i++)
28     {
29         LCD_CursorPos(0x54+i);           // 4. Zeile
30         LCD_WriteChar(52);
31     }
32 }
```

Abb. 12.7: Display-Demo 1

Sicherlich ist Ihnen aufgefallen, dass zur Cursorposition bei der zweiten Zeile „0x40“, der dritten Zeile „0x14“ und der vierten Zeile „0x54“ addiert werden muss. Diese Werte sind abhängig vom benutzten Modul. Verwenden Sie anstatt des hier beschriebenen 20x4-Moduls z. B. ein 16x4-Modul, so müssten zu den einzelnen Cursorpositionen folgende Werte addiert werden: (Zeile 1: 0x00, Zeile 2: 0x40, Zeile 3: 0x10, Zeile 4: 0x50).

Damit Sie die richtigen Werte verwenden, sollten Sie auf jeden Fall das Datenblatt Ihres Modulherstellers zu Rate ziehen.

Mit dem kleinen Programm Abb. 12.8 wird z. B. der Text „Test“ ab der Position 28 geschrieben.

Erforderliche Bibliotheken: „IntFunc_Lib.cc“, „LCD_Lib.cc“

```
0  //-------------------------------------------------------------------------------
1  // Hauptprogramm
2  //
3
4  char text[21];
5  void main(void)
6  {
7      LCD_Init();                          // Display initialisieren
8      LCD_ClearLCD();                      // Display löschen
9      LCD_CursorOff();                     // Display Cursur ausschalten
10
11     text = "Test";
12     LCD_CursorPos(0x40+8);               // Position 28
13     LCD_WriteText(text);
14 }
```

Abb. 12.8: Display-Demo 2

Mit dem Befehl „LCD_WriteCTRRegister“ ist es möglich, direkt Kommandos an den Display-Controller zu senden.

In dem folgenden Programm (siehe Abb. 12.9) wird gezeigt, mit welchen Befehlen das Display aus- und wieder eingeschaltet werden kann. Des Weiteren sind einige Beispiele für das Verschieben von Texten dargestellt.

Erforderliche Bibliotheken: „IntFunc_Lib.cc“, „LCD_Lib.cc“

In Abb. 12.10 können Sie ein Demo finden, das eine Laufschrift auf dem LCD-Display darstellt. In Zeile 13 wird der gewünschte Text der Variablen „text“ zugewiesen. Bitte beachten Sie, dass bei der Deklaration der Variablen „text“ in Zeile 3 der Wert in eckigen Klammern immer um eins größer sein muss als die tatsächliche Textlänge.

Über die Zeile 25 können Sie die Zeile wählen, auf der der Text ausgegeben werden soll. In Abb. 12.7 wird dies näher beschrieben.

Erforderliche Bibliotheken: „IntFunc_Lib.cc“, „LCD_Lib.cc“

```
//-------------------------------------------------------------------------
// Hauptprogramm
//

int i;
char text[21];
void main(void)
{
    LCD_Init();                             // Display initialisieren
    LCD_ClearLCD();                         // Display löschen
    LCD_CursorOff();                        // Display Cursur ausschalten

    text = "Test";
    LCD_CursorPos(0x40+8);                  // Position 28
    LCD_WriteText(text);
    for (i=0;i<10;i++)
    {
        LCD_WriteCTRRegister(0x08);         // Display ausschalten
        AbsDelay(200);
        LCD_WriteCTRRegister(0x0c);         // Display einschalten
        AbsDelay(200);
    }
    for (i=0;i<8;i++)
    {
        LCD_WriteCTRRegister(16+8);         // Text nach links scrollen
        AbsDelay(200);
    }
    for (i=0;i<16;i++)
    {
        LCD_WriteCTRRegister(16+0+4);       // Text nach rechts scrollen
        AbsDelay(200);
    }
    for (i=0;i<8;i++)
    {
        LCD_WriteCTRRegister(16+8);         // Text nach links scrollen
        AbsDelay(200);
    }
}
```

Abb. 12.9: Display-Demo 3

```
//-------------------------------------------------------------------------
// Hauptprogramm
//
char text[45];                              // Deklaraion des Textarrays
char ausgabe[21];                           // Deklaraion des Ausgabearrays
int pos,j,t;
void main(void)
{
    LCD_Init();                             // Display initialisieren
    LCD_ClearLCD();                         // Display löschen
    LCD_CursorOff();                        // Display Cursur ausschalten
    while (true)
    {
        text = "Dieser Text hat eine Laenge von 44 Zeichen. ";
                                            // Ausgabetext wird zugewiesen.
        for (pos=0; pos<20; pos++)          // Zähler für die aktuelle Cursorposion
        {
            ausgabe[pos] = text[t];         // Ausgabetext wird aufgebaut
            t++;
            if (t >= Str_Len(text))         // 20 Zeichen der Variablen "text"
            {                               // werden der Variablen "ausgabe"
                t=0;                        // zugewiesen. Ist das Letzte
            }                               // Zeichen der Variablen "text"
        }                                   // erreicht wird wieder mit dem
                                            // ersten Zeichen begonnen.
        LCD_CursorPos(0);                   // Ausgabe auf Zeile 1
        LCD_WriteText(ausgabe);
        AbsDelay(200);                      // 200ms Verzögerung
        j++;                                // Zähle für die aktuelle Textposition
        t=j;                                // Neue Textposition wird zugewiesen
        if (t >= Str_Len(text))             // Ist das Letzte Zeichen der Variablen
        {                                   // "text" erreicht wird wieder mit
            j=0;                            // dem ersten Zeichen begonnen.
        }
    }
}
```

Abb. 12.10: Display-Demo 4

Weitere Informationen zum LC-Display finden Sie auf der C-Control-Pro-CD, die im Lieferumfang der C-Control Pro enthalten ist.

Viele der dort aufgeführten Befehle können Sie mit dem Befehl „LCD_WriteCTRRegister(...)“ verwenden. Ob Sie bei der Zuweisung hexadezimale Zahlen oder Dezimalzahlen verwenden ist unerheblich (siehe Abb. 12.8). Das Kommando wird in beiden Fällen erkannt und abgearbeitet.

12.3 Sensoren

Prinzipiell muss man zwischen zwei Arten von Sensoren unterscheiden.

Zum einen die Sensoren, bei denen die Auswertung bereits beim Sensor geschieht und nur noch das Ergebnis an die C-Control Pro weitergeleitet wird. Dies sind zum Beispiel Raumthermostate, die in Abhängigkeit der Temperatur ein Relais ein- und ausschalten. Für diese Signale können die digitalen Ports der C-Control Pro verwendet werden, da die Signale entweder 0 V oder 5 V haben.

Die andere Gruppe der Sensoren liefert einen analogen Wert. Dieser liegt als Spannung zwischen 0 V und 5 V vor und kann somit nur über die ADC-Ports verarbeitet werden.

Bei dem MEGA128 sind dies die Ports PF0 bis PF7 und beim MEGA32 die Ports PA0 bis PA7. Wie bereits in früheren Kapiteln erwähnt, muss man sich auch hier wieder im Klaren sein, welche Ports man in seinem Projekt verwenden will und ob dann bestimmte Ein- und Ausgabemöglichkeiten noch vorhanden sind. Vor allem beim MEGA32 sind durch die Mehrfachbelegung verschiedener Ports die Möglichkeiten zur Verwirklichung eigener Ideen etwas eingeschränkt. Will man zum Beispiel alle acht ADC-Ports verwenden, muss man auf die LCD-Anzeige und die USB-Kommunikation verzichten.

12.3.1 Digitale Sensoren

Für das nächste Beispiel wird das Port C verwendet. In den nächsten Abbildungen ist die Lage der Ports auf dem Application-Board des Mega32 und Mega128 gekennzeichnet.

Beachten Sie bitte, dass beim MEGA128 das externe SRAM mit dem Jumper JP7 ausgeschaltet werden muss.

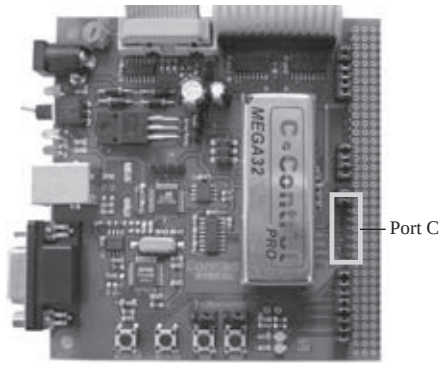


Abb. 12.11: Port C Mega32

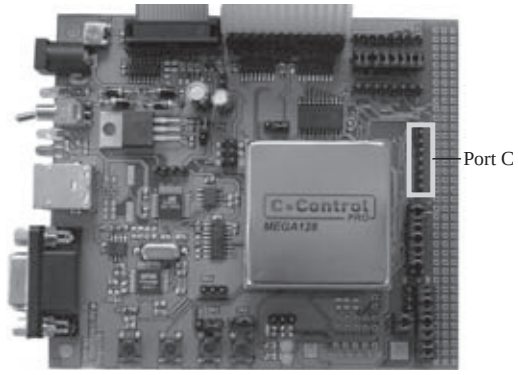


Abb. 12.12: Port C Mega128

In folgendem Programm können Sie sehen wie die Ports PC1 bis PC7 jeweils eingelesen und danach der ermittelte Wert auf dem LC-Display ausgegeben wird.

Erforderliche Bibliotheken: „IntFunc_Lib.cc“, „LCD_Lib.cc“

In den Zeilen 0 und 1 werden die erforderlichen Variablen global deklariert. Die Zeilen 4 bis 6 dienen zur Vorbereitung des Displays. In Zeile 8 werden alle Pins des Portes C für die Eingabe vorbereitet. Der Wert „2“ steht für das Port C und „0“ für Eingabe. Bei dem Befehl „Port_DataDir()“ wird immer der komplette Port angesprochen. Die Auswahl, ob ein Port-Pin für Eingabe oder Ausgabe vorbereitet werden soll, erfolgt über den zweiten Wert.

```
0  int i,wert;                              // Variablendeklartion
1  char ausgabe;
2  void main(void)
3  {
4      LCD_Init();                          // Display initialisieren
5      LCD_ClearLCD();                      // Display löschen
6      LCD_CursorOff();                     // Display Cursur ausschalten
7
8      Port_DataDir(2,0);                   // Datenrichtung: alle Pin's des Ports C
9                                           // werden auf Eingang konfiguriert.
10     while (1)
11     {
12         for (i=1;i<8;i++)                // Es werden 7 Ports nacheinander
13         {                                // ausgelesen.
14             ausgabe = 0x4c;              // Bei 0 V wird "L" ausgegeben
15             wert = Port_ReadBit(16+i);   // Wert des ersten Ports wird gelesen
16             LCD_CursorPos(i-1);          // Cursor Position wird festgelegt
17             if (wert == 1)               // Der eingelesene Wert wird
18             {                            // ausgewertet. Bei 5V wird "H"
19                 ausgabe = 0x48;          // ausgegeben.
20             }
21             LCD_WriteChar(ausgabe);      // Displayausgabe
22         }
23     }
24 }
```

Abb. 12.13: Sensor Demo 1

z. B.	Port_DataDir(2,0)	Alle Pins des Ports C auf Eingang
	Port_DataDir(2,1)	PC0 Ausgang, alle anderen Eingang
	Port_DataDir(2,2)	PC1 Ausgang, alle anderen Eingang
	Port_DataDir(2,3)	PC0 und PC1 Ausgang, alle anderen Eingang
	Port_DataDir(2,4)	PC2 Ausgang, alle anderen Eingang
	Port_DataDir(2,5)	PC0 und PC2 Ausgang, alle anderen Eingang
	Port_DataDir(2,255)	Alle Pins des Ports C auf Ausgang

In den nun folgenden Zeilen werden die eingelesenen Werte ausgewertet. Bei 0 V wird auf dem Display „L“ ausgegeben, bei 5 V wird „H“ ausgegeben. Sind die Eingänge offen, d. h. haben Sie weder 0 V noch 5 V angeschlossen, kann sowohl „L“ als auch „H“ angezeigt werden. Da die Eingänge in diesem Fall nicht definiert sind, ist dieses Verhalten normal.

12.3.2 Analoge Sensoren

Bei einem analogen Sensor muss das Signal erst entsprechend aufbereitet werden. Hierfür stehen beim MEGA32 und MEGA128 jeweils acht ADC-Ports zur Verfügung.

An diesen Ports ist ein Analog-Digital-Wandler mit einer Auflösung von 10 Bit angeschlossen. Dieser wandelt die anliegenden Spannungen in Werte zwischen 0 und 1023 um. Als untere Grenze für die Referenzspannung ist GND (0 V) zu Grunde gelegt. Für den oberen Grenzbereich wird im nachfolgenden Beispiel 5 V (ADC_VREF_VCC) verwendet.

$$Anzeigewert = \frac{1024 \times anliegende\ Spannung}{Referenzspannung}$$

Wollen Sie nun z. B. die Temperatur über einen Sensor ermitteln, müssen Sie sich als Erstes überlegen, welche Art von Sensor sie verwenden wollen. Gehen wir z. B. von einem NTC-Sensor Typ M202 aus. Dieser Sensor hat bei Zimmertemperatur einen Widerstand von ca. 6,7 KOhm. Da es sich um einen NTC-Sensor handelt, sinkt der

Abb. 12.14: NTC-Sensor (Best.-Nr.: 182800)

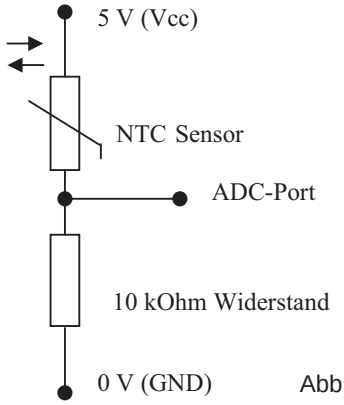


Abb. 12.15: Schaltplan

Widerstand mit steigender Temperatur. Um nun eine auswertbare Spannungsänderung zu erhalten, muss ein Spannungsteiler mit dem NTC-Sensor aufgebaut werden.

In Abb. 12.15 können Sie sehen, dass der Vorwiderstand an GND angeschlossen ist. Steigt nun die Temperatur, sinkt der Widerstand des NTC-Sensors und somit auch der Spannungsabfall in diesem. Gleichzeitig steigt der Spannungsabfall am 10-kOhm-Widerstand. Dies bedeutet, dass mit steigender Temperatur auch die Spannung am Widerstand und somit am ADC-Port steigt.

Um den Sensor nun anzuschließen, müssen Sie als erstes den Spannungsteiler wie in Abb. 12.15 aufbauen. Danach können Sie den Spannungsteiler zum Beispiel an ADC0 anschließen.

Die folgenden Abbildungen zeigen Ihnen die Lage der entsprechenden Ports beim Application-Board des Mega32 und Mega128.

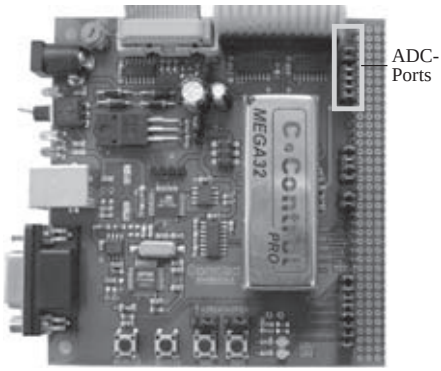


Abb. 12.16: ADC-Ports Mega32

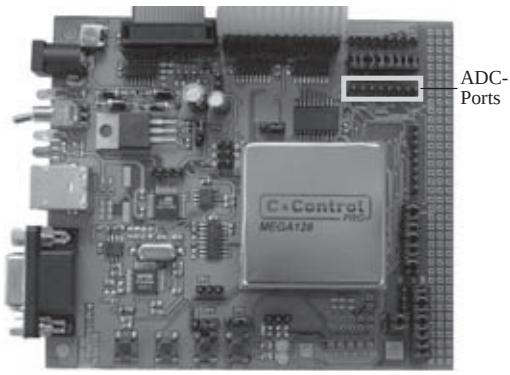


Abb. 12.17: ADC-Ports Mega128

Der Analog-Digital-Wandler ist beim Mega32 am Port A und beim Mega128 am Port F angeschlossen.

Im nun folgenden Programm wird der ADC-Wert erfasst und über das LCD-Display ausgegeben.

Erforderliche Bibliotheken: „IntFunc_Lib.cc“, „LCD_Lib.cc“

In den Zeilen 0 und 1 werden die erforderlichen Variablen global deklariert. Die Zeilen 5 bis 7 dienen zur Vorbereitung des Displays. In Zeile 9 wird die Referenzspannung für den ADC-Port festgelegt. In diesem Fall ist es die Versorgungsspannung Vcc (5 V).

Folgende Referenzspannungen stehen zur Verfügung:

ADC_VREF_BG	2,56 V interne Referenzspannung
ADC_VREF_VCC	Versorgungsspannung (5 V)
ADC_VRAF_EXT	externe Referenzspannung (ist an PAD3 anzuschließen)

```
0  int wert;
1  char ausgabe[8];
2
3  void main(void)
4  {
5      LCD_Init();                          // Display initialisieren
6      LCD_ClearLCD();                      // Display löschen
7      LCD_CursorOff();                     // Display Cursor ausschalten
8
9      ADC_Set(ADC_VREF_VCC,0);             // Referenzspannung: Versorgungsspannung (5V)
10                                          // ADC0: PA0 (Mega32)
11                                          //       PF0 (Mega128)
12     while (true)
13     {
14         LCD_CursorPos(0);                // Cursorposition festlegen
15         wert = ADC_Read();               // ADC-Wert wird gelesen und in
16                                          // der Variblen "wert" gespeichert
17         Str_WriteInt(wert,ausgabe,0);    // Die Integer Variable "wert" wird
18                                          // in einen Text umgewandelt und
19                                          // in im Array "ausgabe" gespeichert
20         LCD_WriteText(ausgabe);          // Das Array "ausgabe" wird ausgegeben
21         AbsDelay(500);                   // Zeitverzögerung: 500 ms
22     }
23 }
```

Abb. 12.18: Demo Analog-Port

Der Wert „0“ steht hier für den Port ADC0. Die Portnummer kann den Wert 0 bis 7 annehmen. Dies entspricht beim Mega32 den Ports A.0 bis A.7 und beim Mega128 den Ports F.0 bis F.7. In Zeile 14 wird der Cursor auf die Stelle „0“ positioniert. Mit dem Befehl „ADC_READ()“ wird nun der zuvor mit der Referenzspannung gesetzte ADC-Port ausgelesen und der entsprechende Wert in der Variablen „wert“ gespeichert. Zeile 17 dient zur Umwandlung der Integer-Variablen „wert“ in das Charakter-Array „ausgabe“. „0“ bedeutet hier, dass kein Offset vorhanden ist und somit der Wert ohne Versatz in das Array geschrieben wird. Der Befehl in Zeile 20 gibt nun den umgewandelten Wert auf dem LCD-Display aus. Durch den AbsDelay-Befehl in Zeile 21 wird die Refreshrate auf 500ms festgelegt.

Im Display sehen Sie nun bei Zimmertemperatur einen Wert von ca. 640. Dieser ist natürlich vom verwendeten Sensor, Vorwiderstand, von Übergangswiderständen usw. abhängig. Da die Widerstandskurve des NTC-Sensors nicht linear ist, ist es nicht ganz einfach, den Messwert z. B. in eine Temperatur umzuwandeln. Wird der Temperatursensor z. B. als Zweipunktschalter verwendet, ist es am einfachsten, wenn Sie den Sensor entsprechend erhitzen und abkühlen und dann die bei den gewünschten Temperaturen angezeigten Werte entsprechend in Ihrem Projekt als Schaltschwellen verwenden.

Neben dem aufgeführten NTC-Sensor können natürlich auch andere Sensoren in gleicher Art und Weise angeschlossen werden.

Abb. 12.19: Betauungsfühler (Best.-Nr.: 156556)

Raumfeuchtefühler (Best.-Nr.: 156584)

Lichtfühler (Best.-Nr.: 156518)

12.4 CCI Relais-Modul

Die Relaisplatine der C-Control-I-Erweiterungsmodule kann natürlich auch für die C-Control Pro verwendet werden. Bei der in Abb. 12.20 dargestellten Platine sind nur sechs Verbindungen zwischen dem Application-Board und der Stiftleiste herzustellen. Es müssen die Versorgungsspannung, GND und die vier Ports für die Ansteuerung der vier Relais verbunden werden.

Die Relaisplatine ist mit einer zweireihigen Steckerleiste ausgestattet. Die zur Verbindung mit dem Application-Board des Mega32 und Mega128 benötigten Kontakte befinden sich alle in der oberen Reihe.

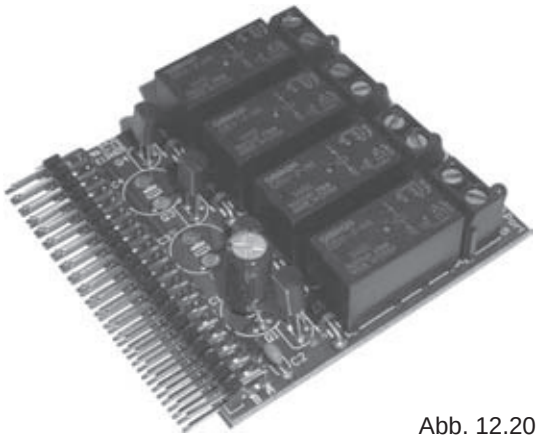


Abb. 12.20: Relaisplatine (Best.-Nr.: 198836)

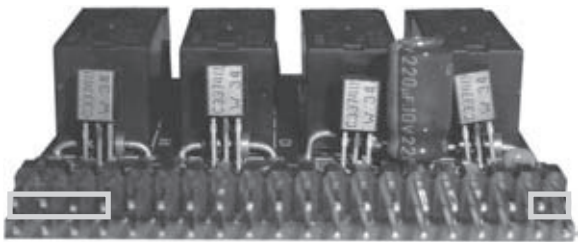


Abb. 12.21: Relaisplatine Steckerleiste

Von links nach rechts sind dies:
Relais K4, Relais K3, Relais K2, Relais K1, Vcc (+5 V), GND

Wenn Sie den Mega32 verwenden, verbinden Sie nun die Platinen folgendermaßen:

Relaisplatine	**Application-Board Mega32**
GND	GND
Vcc	Vcc
Relais K1	Port C.4
Relais K2	Port C.5
Relais K3	Port C.6
Relais K4	Port C.7

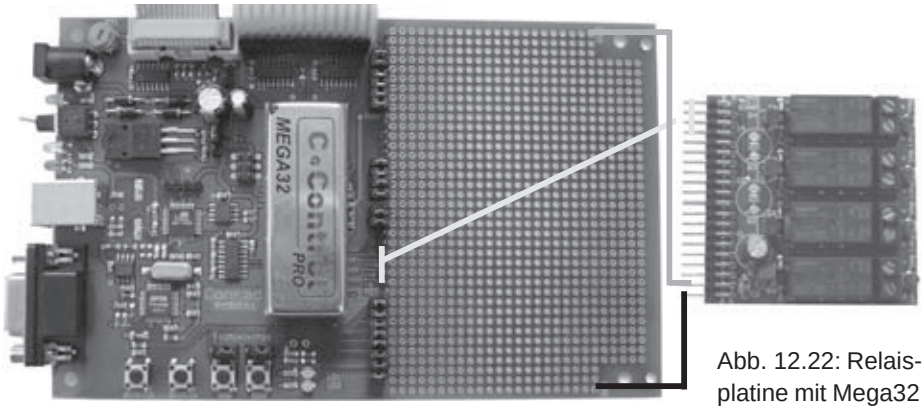


Abb. 12.22: Relaisplatine mit Mega32

Das entsprechende Programm ist in Abb. 12.23 ersichtlich.

Erforderliche Bibliotheken: „IntFunc_Lib.cc“

```
0  int i,k;                                  // globale Variablendeklaration
1
2  void main(void)
3  {
4      Port_DataDir(PortC,240);              // PortC.4 - PortC.7 auf Ausgang
5      Port_DataDirBit(PORT_SW1,PORT_IN);    // Port des Schalters1 auf Eingang
6      while (1)
7      {
8          i = Port_ReadBit(PORT_SW1);       // Zustand des Schalters1 wird
9                                            // eingelesen und in die Variabel "i"
10                                           // gespeichert.
11         if (i == 0)                       // Wenn der Schalter gedrückt ist
12         {                                 // wird folgender Teil abgearbeitet.
13             Port_WriteBit(20+k,PORT_OFF); // Port wird ausgeschaltet, dadurch wird
14             AbsDelay(1000);               // das Relais für 1s eingeschaltet.
15             Port_WriteBit(20+k,PORT_ON);  // Port wird eingeschaltet, dadurch wird
16             AbsDelay(500);                // das Relais ausgeschaltet
17             k++;                          // Port wird um eins erhöht.
18             if (k > 3)                    // Wenn Port C.7 erreicht ist wird
19             {                             // auf Port C.4 zurückgeschaltet.
20                 k = 0;
21             }
22         }
23     }
24 }
```

Abb. 12.23: Steuerung Mega32 für Relaisplatine

Wenn Sie den Mega128 verwenden, verbinden Sie nun die Platinen folgendermaßen:

Relaisplatine	**Application-Board Mega128**
GND	GND
Vcc	Vcc
Relais K1	Port F.4
Relais K2	Port F.5
Relais K3	Port F.6
Relais K4	Port F.7

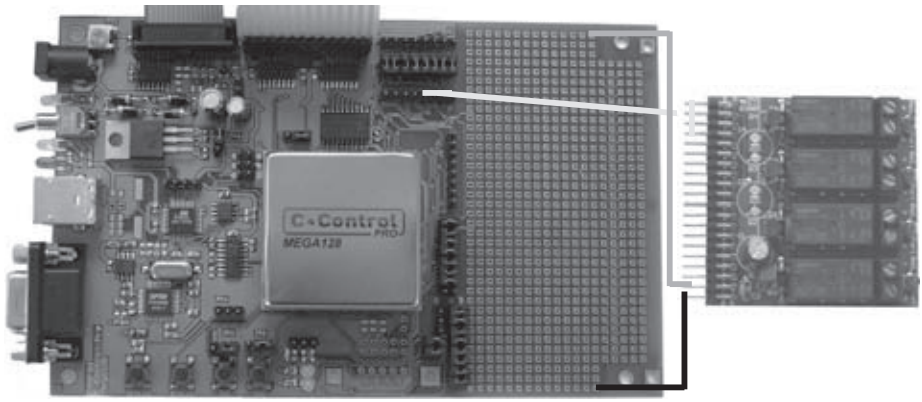


Abb. 12.24: Relaisplatine mit Mega128

Das entsprechende Programm ist in Abb. 12.25 ersichtlich.

Erforderliche Bibliotheken: „IntFunc_Lib.cc“

```
0  int i,k;                                  // globale Variablendeklaration
1
2  void main(void)
3  {
4      Port_DataDir(PortF,240);              // PortF.4 - Portf.7 auf Ausgang
5      Port_DataDirBit(PORT_SW1,PORT_IN);    // Port des Schalters1 auf Eingang
6      while (1)
7      {
8          i = Port_ReadBit(PORT_SW1);       // Zustand des Schalters1 wird
9                                            // eingelesen und in die Variabel "i"
10                                           // gespeichert.
11         if (i == 0)                       // Wenn der Schalter gedrückt ist
12         {                                 // wird folgender Teil abgearbeitet.
13             Port_WriteBit(444k,PORT_OFF); // Port wird ausgeschaltet, dadurch wird
14             AbsDelay(1000);               // das Relais für 1s eingeschaltet.
15             Port_WriteBit(444k,PORT_ON);  // Port wird eingeschaltet, dadurch wird
16             AbsDelay(500);                // das Relais ausgeschaltet
17             k++;                          // Port wird um eins erhöht.
18             if (k > 3)                    // Wenn Port F.7 erreicht ist wird
19             {                             // auf Port F.4 zurückgeschaltet.
20                 k = 0;
21             }
22         }
23     }
24 }
```

Abb. 12.25: Relaisplatine mit Mega128

In der Zeile 0 werden die erforderlichen Variablen global deklariert. Die Zeile 4 dient zur Vorbereitung der Ports. Hier werden aber entgegen früherer Demos über den „Port_DataDir“-Befehl nicht alle, sondern nur die benötigten Ports auf Ausgang eingestellt. Die Zahl „240“ bedeutet, dass nur die Ports C.4, C.5, C.6 und C.7 berücksichtigt werden. Die Zahl setzt sich zusammen aus:

16 (C.4) + 32 (C.5) + 64 (C.6) + 128 (C.7) = 240

Dies gilt natürlich analog beim Mega128 für die Ports F.4, F.5, F.6 und F.7. In der Zeile 5 wird nun der Port des Schalters 1 auf Eingang vorbereitet. Der Befehl in Zeile 6 bildet zusammen mit den Zeilen 7 und 23 eine Endlosschleife. Mit dem Befehl „Port_ReadBit(PORT_SW1)“ wird nun in Zeile 8 der Zustand des Schalterportes eingelesen und in die Variable „i“ gespeichert. Bei den zwei Programmen Abb. 12.23 und Abb. 12.25 wurde zur Festlegung der Schalterports die vordefinierte Variable „PORT_SW1“ verwendet. Natürlich kann dies auch über die entsprechende Zahl des Ports (z. B. Port 26 beim Mega32) erfolgen. Da die Portnummern für die fest eingebauten Schalter und LEDs der Application-Boards des Mega32 und Mega128 voneinander abweichen, wurden entsprechende Variablen hinterlegt, die in Abhängigkeit von der verwendeten C-Control Pro automatisch deklariert werden. Diese Funktion steht auch bei jedem anderen Programm, dass Sie schreiben, zur Verfügung.

Vordefiniert sind z. B. PortA, PortB, PortC, PortD, PortE, PortF, PortG, PORT_LED1, PORT_LED2, PORT_SW1, PORT_SW2, PORT_ON, PORT_OFF, PORT_OUT und PORT_IN.

In Zeile 11 beginnt nun die eigentliche Auswertung der Tastenabfrage. Sobald der Schalter SW1 gedrückt wurde, hat die Variable „i“ den Wert 0. Das heißt, die Zeilen 13 bis 21 werden nun abgearbeitet. Über den „Port_WriteBit“-Befehl wird nun der Port 20 (C.4) bzw. Port 44 (F.4) ausgeschaltet. Aufgrund dieses Vorgangs schaltet jetzt K1 der Relaisplatine und die entsprechende LED leuchtet. Das Relais bleibt nun für eine Sekunde eingeschaltet. Diese Zeit wird über den Befehl in Zeile 14 bestimmt. Danach wird der Port mit dem nächsten Befehl in Zeile 15 wieder ausgeschaltet. Um ein Prellen des Schalters zu vermeiden, wurde eine zusätzliche Verzögerungszeit von 500 ms in Zeile 16 programmiert. Die Zeilen 18 bis 21 erhöhen die Portnummer jeweils um eins. Wie Sie sehen, wird die Variable „k“ jeweils in den Zeilen 13 und 15 zu den Portnummern addiert. Auf diese Weise wird jeder Port immer wieder nacheinander bei jedem Tastendruck durchgeschaltet.

12.5 I²C-Bus-Thermometer-Modul

Das I²C-Bus-Thermometer-Modul der C-Control-I-Erweiterungsmodule kann natürlich auch für die C-Control Pro verwendet werden. Bei der in Abb. 12.26 dargestellten Platine sind nur vier Verbindungen zwischen dem Application-Board und der Stiftleiste herzustellen. Es müssen die Versorgungsspannung, GND, SDA und SCL verbunden werden. In den Abbildungen 12.27 und 12.28 sind die erforderlichen Verbindungen für den Mega32 und Mega128 dargestellt.

Die Jumper an den Ports C.0 und C.1 müssen entfernt werden. An C.0 muss SCL angeschlossen werden und an C.1 SDA.

Die Jumper an den Ports D.0 und D.1 müssen entfernt werden. An D.0 muss SCL angeschlossen werden und D.1 SDA.

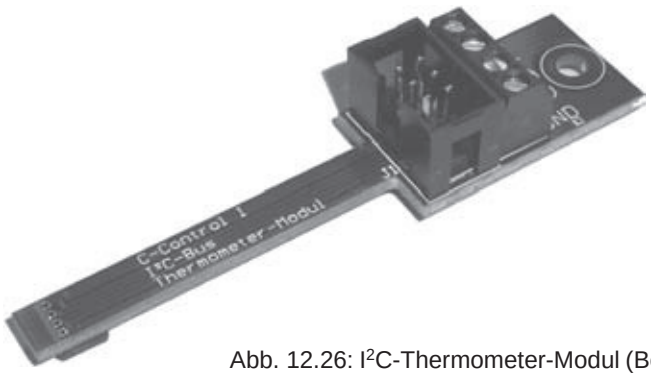


Abb. 12.26: I²C-Thermometer-Modul (Best.-Nr.: 198298)

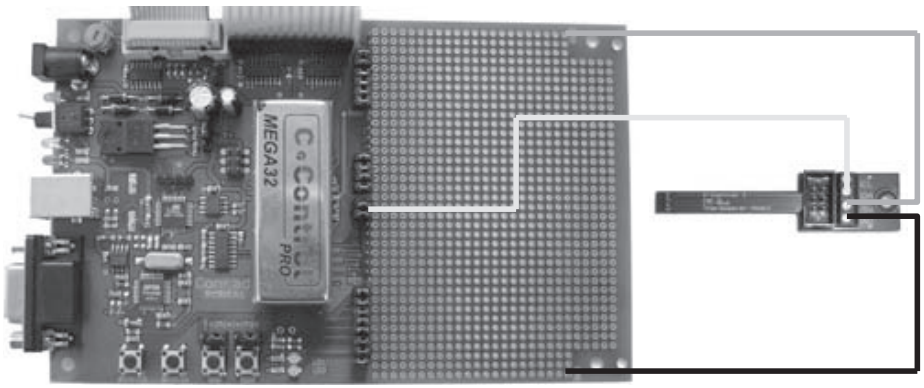


Abb. 12.27: Mega32 mit I²C-Thermometer-Modul

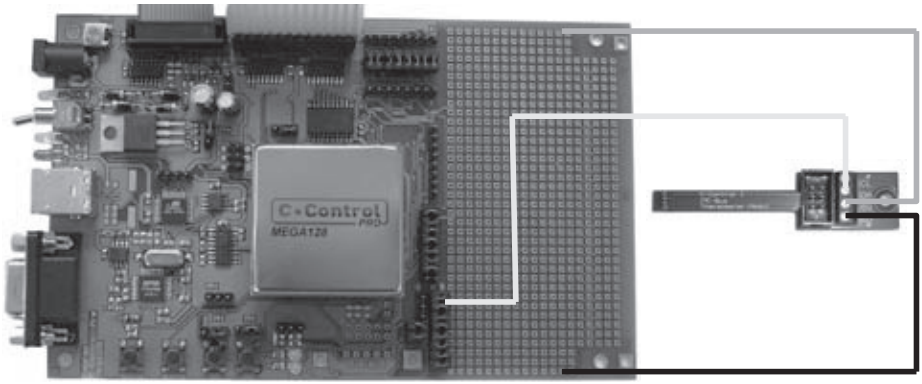


Abb. 12.28: Mega128 mit I²C-Thermometer-Modul

Quellcode CC

Das Programm kann sowohl für den Mega32 als auch für den Mega128 verwendet werden. Es muss nur darauf geachtet werden, dass die I²C-Ports an verschiedenen Stellen der Application-Boards liegen. Die Zuweisung erfolgt automatisch. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

I2CTemp

```
// I2C-Thermometer
// erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

// Über ein I2C-Bus Thermometer Modul wird die aktuelle Temperatur erfasst und
// über die Leitungen SDA und SCL an den MEGA32 bzw. MEGA128 übertragen.
// Der ermittelte Wert wird auf dem Display angezeigt.

// globale Variablendeklaration
int MSDB, LSDB;
char zeile1[9], zeile2[9];

//---------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
      TempInit();            // I2C-Modul wird initialisiert
      DisplayInit();         // LCD-Modul wird initialisiert

      while (true)
      {
      I2C_Start();       // I2C starten
      I2C_Write(0x9E);   // Temperaturmodul mit der Adresse 10011110 aufrufen
      I2C_Write(0xAA);   // Die letzte konvertierte Temperatur wird eingelesen
      I2C_Start();       // I2C erneut starten
      I2C_Write(0x9F);   // Temperaturmodul mit der Adresse 10011111 aufrufen

                            // Mit dem ersten Bit wird das Schreiben und Lesen
                            // unterschieden 1 <=> lesen, 0 <=> schreiben.
      MSDB=I2C_Read_ACK();   // Der MSDB-Wert wird gelesen und mit einem ACK
                            // quittiert.
```

```
        LSDB=I2C_Read_NACK();     // Der LSDB-Wert wird gelesen und mit einem NACK
                                  // quittiert.
        I2C_Stop();               // I2C wird angehalten
        Display();          // Der ermittelte Wert wird auf dem Display ausgegeben.
        AbsDelay(1000);     // Nach einer Sekunde wird die nächste Messung
                                  //durchgeführt.
    }
}
```

I2CTempInit

```
void TempInit(void)
{
      I2C_Init(I2C_100kHz );       // I2C initialisieren

      I2C_Start();          // I2C starten
      I2C_Write(0x9E);      // Temperaturmodul mit der Adresse 10011110 aufrufen
      I2C_Write(0xAC);      // Aufruf des 1-byte Konfigurationsregisters
      I2C_Write(0x02);      // Die Datenübertragung des Moduls wird aktiviert
      I2C_Stop();           // I2C wird angehalten

      I2C_Start();          // I2C starten
      I2C_Write(0x9E);      // Temperaturmodul mit der Adresse 10011110 aufrufen
      I2C_Write(0xA1);      // TH-Register wird aufgerufen
      I2C_Write(0x28);      // MSDB der maximalen Temperatur wird gesendet (+40°C)
      I2C_Write(0x00);      // LSDB der maximalen Temperatur wird gesendet
      I2C_Stop();           // I2C wird angehalten

      I2C_Start();          // I2C starten
      I2C_Write(0x9E);      // Temperaturmodul mit der Adresse 10011110 aufrufen
      I2C_Write(0xA2);      // TL-Register wird aufgerufen
      I2C_Write(0x00);      // MSDB der minimalen Temperatur wird gesendet (+0°C)
      I2C_Write(0x00);      // LSDB der minimalen Temperatur wird gesendet
      I2C_Stop();           // I2C wird angehalten

      I2C_Start();          // I2C starten
      I2C_Write(0x9E);      // Temperaturmodul mit der Adresse 10011110 aufrufen
      I2C_Write(0x51);      // Übertragung wird gestartet
      I2C_Stop();           // I2C wird angehalten
}
```

Display

```
//-------------------------------------------------------------------------------
// Display
//
void DisplayInit(void)
{
      LCD_Init();           // Display initialisieren
      LCD_ClearLCD();       // Display löschen
      LCD_CursorOff();      // Display Cursor ausschalten
      zeile1 = “I2C-Temp“;  // Text der ersten Zeile wird zugewiesen
}


void Display(void)
{
      Str_WriteInt(MSDB, zeile2, 0);
      LCD_CursorPos(0x00);          // LCD Cursor positionieren
      LCD_WriteText(zeile1);        // Die erste Zeile wird ausgegeben.
      LCD_CursorPos(0x42);          // LCD Cursor positionieren
      LCD_WriteText(zeile2);        // Die Temperatur wird ausgegeben.
      LCD_WriteChar(0x20);          // Ein Leerzeichen wird geschrieben.
      LCD_WriteChar(0x43);          // Ein C wird ausgegeben.
}
```

Bevor Sie verwendbare Daten vom Thermometer-Modul erhalten, muss dieses initialisiert werden. Dies geschieht in der Funktion TempInit(). In den nun folgenden Tabellen sind einige Einstellmöglichkeiten zusammengefasst, die während der Initialisierungsphase möglich sind. Für genauere Informationen ziehen Sie bitte das Datenblatt von MAXIM für den DS1631 zu Rate.

Prinzipiell muss man den Schreib- und Lesemodus bei jedem I^2C-Modul unterscheiden. Beim Temperatur-Modul dient der Schreibmodus zur Initialisierung und der Lesemodus zum Auslesen der Temperatur. Im Auslieferzustand sind die Lötbrücken des Temperaturmoduls offen. (d. h. A0, A1 und A2 haben High-Pegel). Dies hat zur Folge, dass die Schreib- und Leseadressen folgende Werte haben.

Wertigkeit:	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Schreiben:	1	0	0	1	1	1	1	0
Lesen:	1	0	0	0	1	1	1	1
	Fix ..... Adresse				A0	A1	A2	R/W

Dies bedeutet, dass zum Schreiben die Adresse 0x9E (158) und zum Lesen die Adresse 0x9F (159) aufgerufen werden muss.

Nach dem Aufruf der Adresse des Moduls müssen die gewünschten Kommandos gesendet werden. Nachfolgend sind die verwendeten Codes aufgeführt.

Access Config [0xAC]
Liest oder schreibt das 1-Byte-Konfigurationsregister.

Access TH [0xA1]
Liest oder schreibt das 2-Byte-T_H-Register.

Access TL [0xA2]
Liest oder schreibt das 2-Byte-T_L-Register.

Start Convert T [0x51]
Initialisiert die Temperaturübertragung.

Read Temperature [0xAA]
Liest den letzten Temperaturwert aus dem 2-Byte-Temperaturregister

Die Temperaturen des T_H- und T_L-Registers setzen sich folgendermaßen zusammen:

	bit 15	bit 14	bit13	bit12	bit11	bit10	bit9	bit8
MS Byte	$S2^6$	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit 0
LS Byte	2^{-1}	2^{-2}	2^{-3}	2^{-4}	0	0	0	0

Temperaturbeziehungen:

Temperatur (°C)	Binärcode	Hexadezimaler Wert
+125	0111 1101 0000 0000	0x7D00
+40	0010 1000 0000 0000	0x2800
+10,125	0000 1010 0010 0000	0x0A20
0	0000 0000 0000 0000	0x0000
-0,5	1111 1111 1000 0000	0xFF80
-10,125	1111 0101 1110 0000	0xF5E0
-55	1100 1001 0000 0000	0xC900

Weitere Codes und die Zusammensetzung des Konfigurationsregisters entnehmen Sie bitte dem Datenblatt des DS1631.

12.6 I²C-Bus-Tastatur

Die I²C-Bus-Tastatur der C-Control-I-Erweiterungsmodule kann natürlich auch für die C-Control Pro verwendet werden. Bei der in Abb. 12.29 dargestellten Platine sind nur vier Verbindungen zwischen dem Application-Board und der Stiftleiste herzustellen. Es müssen die Versorgungsspannung, GND, SDA und SCL verbunden werden. In den Abbildungen 12.30 und 12.31 sind die erforderlichen Verbindungen für den Mega32 und Mega128 dargestellt.

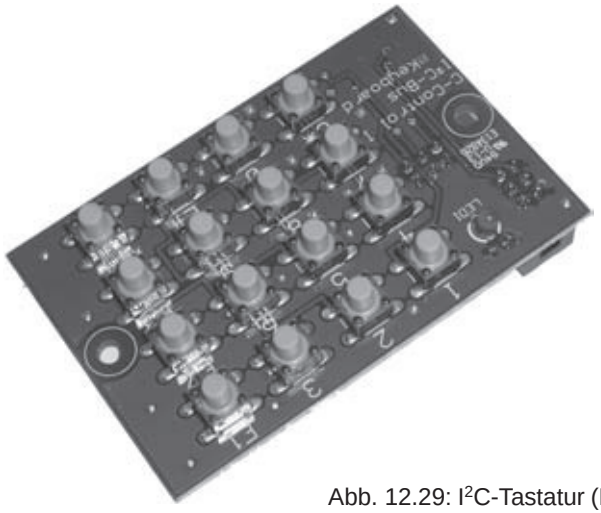


Abb. 12.29: I²C-Tastatur (Best.-Nr.: 198356)

Die Jumper an den Ports C.0 und C.1 müssen entfernt werden. An C.0 muss SCL angeschlossen werden und an C.1 SDA. Der 6-polige Sockel an der I²C-Tastatur hat folgende Belegung: 1: GND, 2: +5 V, 5: SDA, 6: SCL.

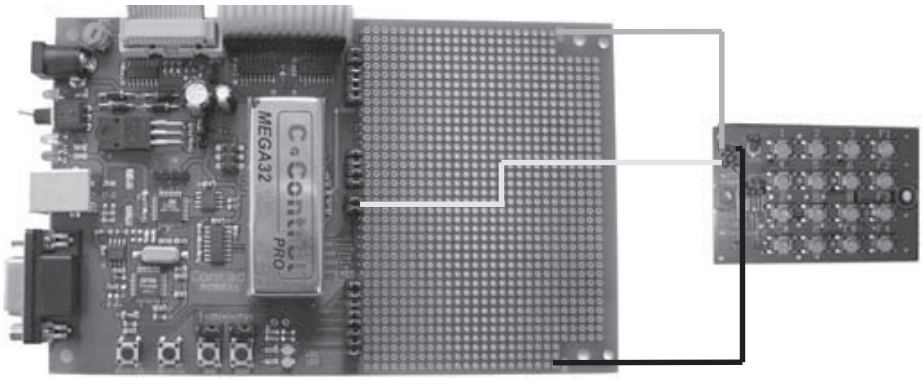


Abb. 12.30: Mega32 mit I²C-Tastatur

Die Jumper an den Ports D.0 und D.1 müssen entfernt werden. An D.0 muss SCL angeschlossen werden und an D.1 SDA. Der 6-polige Sockel an der I²C-Tastatur hat folgende Belegung: 1: GND, 2: +5 V, 5: SDA, 6: SCL.

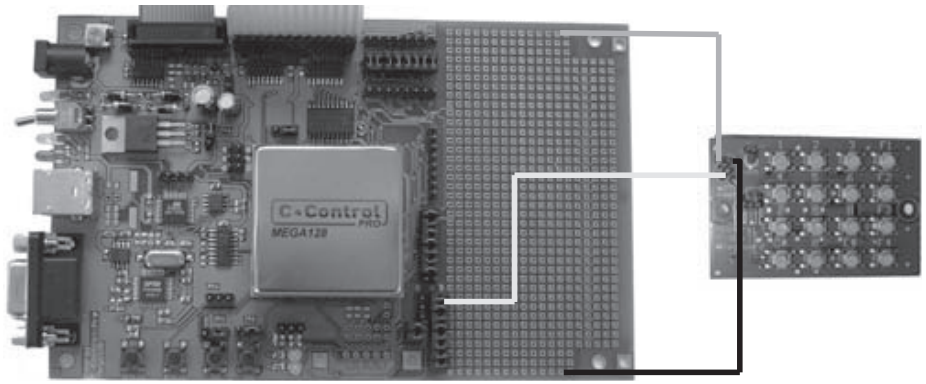


Abb. 12.6.3: Mega128 mit I²C-Tastatur

Quellcode CC

Das Programm kann sowohl für den Mega32 als auch für den Mega128 verwendet werden. Es muss nur darauf geachtet werden, dass die I²C-Ports an verschiedenen Stellen der Application-Boards liegen. Die Zuweisung erfolgt automatisch. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

I2C-Tastatur

```
// I2C-Tastatur
// erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

// Über ein I2C-Bus Tastatur Modul wird der aktuelle Tastendruck erfasst und
// über die Leitungen SDA und SCL an den MEGA32 bzw. MEGA128 übertragen.
// Der der Taste entsprechende Wert wird auf dem Display ausgegeben.

// globale Variablendeklaration
int taste, spalte, zeile;
char zeile1[9], zeile2[9];
char ausgabe, key[17];

//------------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
    DisplayInit();         // LCD-Modul wird initialisiert
    I2C_Init(I2C_100kHz);  // I2C wird initialisiert
    I2C_KeyInit();         // KeyMap wird gelesen
    ausgabe = 0x20;        // default Wert wird gesetzt

    while (true)
    {
        spalte = 0xE0;     // Die erste Spalte der 4x4 Tastaturmatrix wird abgefragt
        TastCode(spalte);  // Funktionsaufruf der Tastaturabfrage
        spalte = 0xD0;     // Die zweite Spalte der 4x4 Tastaturmatrix wird abgefragt
        TastCode(spalte);  // Funktionsaufruf der Tastaturabfrage
        spalte = 0xB0;     // Die dritte Spalte der 4x4 Tastaturmatrix wird abgefragt
        TastCode(spalte);  // Funktionsaufruf der Tastaturabfrage
        spalte = 0x70;     // Die vierte Spalte der 4x4 Tastaturmatrix wird abgefragt
        TastCode(spalte);  // Funktionsaufruf der Tastaturabfrage
        Display();         // Displayausgabe
    }
}
```

I2C-TastCode

```
//------------------------------------------------------------------------------
// Tastatur wird deklariert
//
void I2C_KeyInit(void)
{
    key = “159C260D37*A48#B“;  // KeyMap wird festgelegt
}

void TastCode(int spalt)
{
      switch (spalt)   // Einsprungpunkt in die KeyMap wird
      {                // Spaltenabhängig festgelegt

          case 0x70:
          zeile = 12; break;
          case 0xB0:
          zeile = 8; break;
          case 0xD0:
          zeile = 4; break;
          case 0xE0:
          zeile = 0; break;
      }

      I2C_Start();      // I2C starten
      I2C_Write(0x4E);  // Tastaturmodul mit der Adresse 01011110 aufrufen
```

```
    I2C_Write(spalte + 0x0F);  // Bis auf eine Spalte werden alle Ein-/Aus-
    I2C_Start();               // gänge auf High gesetzt.
    I2C_Write(0x4F);           // Tastaturmodul mit der Adresse 01011111 aufrufen
    taste=I2C_Read_NACK();     // Wert wird gelesen und mit einem NACK quittiert.
    I2C_Stop();                // I2C wird angehalten
    if (taste==spalte + 0x0E) {ausgabe=key[zeile];};    // Abhängig von der
    if (taste==spalte + 0x0D) {ausgabe=key[zeile+1];};  // Spalte und der
    if (taste==spalte + 0x0B) {ausgabe=key[zeile+2];};  // Zeile wird der
    if (taste==spalte + 0x07) {ausgabe=key[zeile+3];};  // entsprechende Wert
}                                                       // aus der KeyMap zugewiesen.
```

Display

```
//---------------------------------------------------------------------------
// Display
//
void DisplayInit(void)
{
    LCD_Init();          // Display initialisieren
    LCD_ClearLCD();      // Display löschen
    LCD_CursorOff();     // Display Cursur ausschalten
    zeile1 = "I2C-Key";  // Text der ersten Zeile wird zugewiesen
}

void Display(void)
{
    LCD_CursorPos(0x00);     // LCD Cursor positionieren
    LCD_WriteText(zeile1);   // Die erste Zeile wird ausgegeben.
    LCD_CursorPos(0x42);     // LCD Cursor positionieren
    LCD_WriteChar(ausgabe);  // Das KeyMapSymbol wird ausgegeben.
}
```

Bevor Sie verwendbare Daten von der Tastatur erhalten, muss diese entsprechend initialisiert werden. Dies geschieht in der Funktion `TastCode()`. Zusammen mit der KeyMap in der Funktion `I2C_KeyInit` wird in dieser zu jedem Taster der 4x4-Matrix der entsprechende Code (Zahl bzw. Buchstabe zugeordnet). Wenn Sie andere Zeichen als die bereits vordefinierten tasterspezifisch ausgeben wollen, müssen Sie diese nur bei der Zuweisung der Variablen `key` in der Funktion `I2C_KeyInit` hinterlegen.

Prinzipiell muss man den Schreib- und Lesemodus bei jedem I^2C-Modul unterscheiden. Bei der Tastatur dient der Schreibmodus zur Vordefinierung der Eingänge und der Lesemodus zum Auslesen der 4x4-Matrix. Im Auslieferzustand sind die Steckbrücken der Tastatur offen. (d. h. A0, A1 und A2 haben High-Pegel). Dies hat zur Folge, dass die Schreib- und Leseadressen folgende Werte haben:

Wertigkeit:	128	64	32	16	8	4	2	1
Schreiben:	0	1	0	1	1	1	1	0
Lesen:	0	1	0	1	1	1	1	1
	Fix			Adresse	A0	A1	A2	R/W

Dies bedeutet, dass zum Schreiben die Adresse 0x4E (78) und zum Lesen die Adresse 0x4F (79) aufgerufen werden muss.

13 Stringverarbeitung

13.1 Strings in der C-Control-Pro-Umgebung

In diesem Kapitel wird der Umgang mit Strings in CompactC und BASIC erklärt. Der fortgeschrittene C-Programmierer wird hier nur wenig Neues entdecken, aber gerade dem Anfänger fehlt meist die Erfahrung im Umgang mit Zeichenketten . Insbesondere bietet die C-Control-Pro-Bibliothek in den Stringfunktionen erweiterte Möglichkeiten, die sich erst dem zweiten Blick erschließen.

13.2 Strings sind Arrays

Zeichenketten, in der englischen Übersetzung „character strings“, werden im C-Control-Pro-System durch Arrays vom Datentyp „Character“ repräsentiert. Der Datentyp „char“, bzw. „Char“ in Basic, definiert eine vorzeichenbehaftete 8-Bit-Zahl. Diese Zahl kann Werte zwischen –128 und 127 annehmen. Welcher Wert welches Zeichen darstellt, wurde 1967 durch die Zeichencodierung ASCII (*American Standard Code for Information Interchange)* genormt.

Eine Zeichenkette bei C-Control Pro ist, traditionell wie bei anderen C/C++ Systemen auch, eine Abfolge von Character-Werten, die durch den Wert Null abgeschlossen sind. Die Länge des Strings ist daher nur von der Array-Größe beschränkt, und Arrays sind nur durch die Größe des RAMs limitiert. Bei der C-Control-Pro Mega 128 mit 64 Kilobyte externem SRAM kann eine Zeichenkette daher über 60 Kilobyte lang werden.

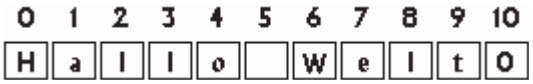


Abb. 13.1: Aufbau einer Zeichenkette im Array

Wichtig: Man muss beim Arbeiten mit Strings immer beachten, dass die maximale Länge eines Strings um eins kleiner ist als die Größe des String-Arrays! Die Null am Ende der Zeichenkette verbraucht ja auch einen Array-Eintrag.

Hier ein Programm für eine simple Stringausgabe in CompactC:

```
void main(void)
{
  char text[30];        // Zeichenkette

  text="Hallo Welt";   // Stringzuweisung
  Msg_WriteText(text); // Stringausgabe
}
```

bzw. in BASIC:

```
Sub main()
  Dim text(30) As Char ' Zeichenkette

  text="Hallo Welt" ' Stringzuweisung
  Msg_WriteText(text) ' Stringausgabe
End Sub
```

13.3 Stringfunktionen in der Bibliothek

Die häufigste Funktion bei der Stringbearbeitung ist das Kopieren, hierfür wird in der Bibliothek die Funktion `Str _Copy` angeboten. Eine Übersicht über alle Stringfunktionen findet man im C-Control-Pro-Benutzerhandbuch im Kapitel „Bibliotheken", Unterverzeichnis „Strings". `Str_Copy()` hat in C und BASIC folgende Syntax:

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

Sub Str_Copy(ByRef destination As Char,ByRef source As
    Char,offset As Word)

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
```

Mit dieser Funktion wird eine Zeichenkette aus dem Array `source` in die Array-Variable `destination` kopiert. Der Parameter `offset` sollte normalerweise Null sein, aber er erweitert die Möglichkeiten, einen bestehenden String zu modifizieren. Setzt man `offset` auf einen Wert größer Null, so wird der `source`-String um `offset` Zeichen versetzt in den `destination`-String kopiert. Benutzt man für `offset` den speziellen Wert 65535 (Hexadezimal $ffff), wird der String `source` an `destination` angehängt. (Dies würde in einem Ansi-C-Compiler der Funktion `strcat()` entsprechen.) Der Preprozessor (siehe Kapitel „Der Preprozessor") kennt die Definition STR _APPEND für den Wert $ffff. Aus Gründen der Verständlichkeit sollte man Preprozessor-Definitionen immer im Quelltext benutzen. Konstanten wie z. B. STR_APPEND sind im C-Control-Pro-System in der Datei „IntFunc_Lib.cc" im Verzeichnis „Libraries" definiert.

Im Folgenden ein Beispiel für die Benutzung von `Str_Copy` in CompactC

```
void main(void)
{
  char source[30];
  char destination[30];

  destination=“Hallo Wort“;               // String zuweisen
  source=“Welt“;
  Str_Copy(destination,source,6);         // “Wort“ überschreiben
  Msg_WriteText(destination);             //“Hallo Welt“ ausgeben
  Msg_WriteChar(13);                      // CR ausgeben
  source=“,neu\r“;
  Str_Copy(destination,source,STR_APPEND);//String anhängen
  Msg_WriteText(destination);             // “Hallo Welt,neu“ ausgeben
}
```

und in BASIC:

```
Sub main()
  Dim source(30) As Char
  Dim destination(30) As Char

  destination=“Hallo Wort“ ’ String zuweisen
  source=“Welt“
  Str_Copy(destination,source,6) ’ “Wort“ überschreiben
  Msg_WriteText(destination) ’ “Hallo Welt“ ausgeben
  Msg_WriteChar(13) ’ C/R ausgeben
  source=“,neu\r“
  Str_Copy(destination,source,STR_APPEND) ’String anhängen
  Msg_WriteText(destination) ’ “Hallo Welt“ ausgeben
End Sub
```

13.4 Stringbearbeitung – selbst gemacht

Da Zeichenketten Arrays sind, steht nichts im Wege, um Routinen wie `Str_Copy` selbst zu schreiben. In der Tat, sind die Funktionen `Str_ Fill()`, `Str_Isalnum()`, `Str_Isalpha()`, `Str_Substr()` in CompactC geschrieben. Man findet sie im Bibliotheksverzeichnis „Libraries“ in der Datei „String_Lib.cc“.

Wie sieht die Arbeitsweise von `Str_Copy()` aus? (Man nimmt hier bewusst eine vereinfachte Form ohne den `offset`-Parameter.) Man kopiert nacheinander in einer Schleife Zeichen aus dem `source`-Array an die gleiche Position in das `destination`-Array. Wird der Endwert Null erreicht, wird als letztes die Null kopiert und dann die Schleife verlassen.

CompactC:

```
void Str_Copy2(char destination[],char source[])
{
  int i;
  i=0;
  while(true)
  {
    destination[i]=source[i]; // ein Zeichen kopieren
    if(destination[i]==0) break; // bei Null abbrechen
```

```
    i=i+1; // Index erhöhen
  }
}

void main(void)
{
  char source[30];
  char destination[30];

  destination="";                // Ziel löschen
  source="Hallo Welt\r";         // source String zuweisen
  Str_Copy2(destination,source); // String kopieren
}
```

BASIC:

```
Sub Str_Copy2(ByRef destination As Char,ByRef source As Char)
  Dim i As Integer

  i=0
  Do While True
    destination(i)=source(i) ' ein Zeichen kopieren
    If destination(i) = 0 Then
      Exit ' bei Null abbrechen
    End If
    i=i+1 ' Index erhöhen
  End While
End Sub

Sub main()
  Dim source(30) As Char
  Dim destination(30) As Char

  destination="" ' Ziel löschen
  source="Hallo Welt\r"
  Str_Copy2(destination,source) ' String kopieren
End Sub
```

Der hier präsentierte Programmcode ist nicht effizient, dafür aber einfacher zu verstehen. Eine optimierte Version, sowie weitere Möglichkeiten zur Effizienzsteigerung werden in den Kapiteln „Optimierung von CompactC“ und „Optimierung von BASIC“ demonstriert.

In der Stringbibliothek des C-Control-Pro-Systems befinden sich keine Funktionen, die aus einer Zeichenkette wieder eine Zahl zu machen. Auch dies kann einfach selbst bewerkstelligt werden. Hat eine Dezimalzahl eine ASCII-Repräsentation, so fängt man mit dem ersten Zeichen des Strings an, verwandelt das Zeichen in einen numerischen Wert, multipliziert mit 10, und addiert den numerischen Wert des nächsten Zeichens. Diesen Vorgang wiederholt man, bis man zum Ende des Strings gekommen ist. Wie wird nun aus einer Ziffer in ASCII-Darstellung der numerische Wert? Die ASCII-Zeichen ‚0‘ bis ‚9‘ liegen als Zeichen in der ASCII-Tabelle hintereinander und haben die Werte 48 (ASCII ‚0‘) bis 57 (ASCII ‚9‘). Man muss vom ASCII-Wert daher nur 48 abziehen, um den numerischen Wert zu erhalten.

Die im folgenden Quellcode benutzte Funktion `Str_Len()` gibt die Länge eines Strings zurück.

Als CompactC:

```
void main(void)
{
  int i,length,num;
  char text[50];

  text="4711";
  num=0;
  length=Str_Len(text);     // Länge ermitteln
  for(i=0;i<length;i++)     // Schleife über alle Zeichen
  {
    num=num*10;             // um eine Zehnerpotenz erhöhen
    num=num+text[i]-48;     // num. Wert der Ziffer addieren
  }
  Msg_WriteInt(num);
}
```

In BASIC:

```
Sub main()
  Dim i,length,num As Integer
  Dim text(50) As Char

  text="4711"
  num=0
  length=Str_Len(text)      // Länge ermitteln
  For i=0 To length-1       // Schleife über alle Zeichen
      num=num*10            // um eine Zehnerpotenz erhöhen
      num=num+text(i)-48    // num. Wert der Ziffer addieren
  Next
  Msg_WriteInt(num)
End Sub
```

Ein weiteres Beispiel, welches hier untersucht werden soll, ist die Funktion `Str_SubStr()` aus der String-Bibliothek. Die Funktion sucht den String `search` in der Zeichenkette `source`. Kommt der String `search` in `source` vor, so wird der Index (also die Position im Array `source`) zurückgegeben, also die Stelle, an der der String `search` das erste Mal im String `source` vorkommt. Sollte `search` gar nicht in `source` vorhanden sein, so erhält man als Rückgabewert –1.

`Str_SubStr()` in CompactC hat folgenden Quellcode:

```
int Str_SubStr(char source[],char search[])
{
    int i,j,len;
    char s,c,v;

    s=search[0];                    // erstes Zeichen von search nehmen
    i=0;                            // Schleifenzähler initialisieren

    do                              // in äußerer Schleife nach erstem Zeichen
    {                               // von search suchen
        c=source[i];
        if(c==s)                    // erstes Zeichen von search gefunden!
        {
            j=1;                    // Schleifenzähler initialisieren
            do                      // in innerer Schleife Reststring vergleichen
            {
                v=search[j];
            } while(v!=0 && v==source[i+j++]);
            if(!v) return(i);       // Wenn v==0 String gefunden
```

```
        }
        i++;

    } while(c); // bei Stringende aufhören

    return(-1); // keine Übereinstimmung gefunden
}
```

Die Arbeitsweise der CompactC-Version von `Str_SubStr()` ist folgende: In der äußeren Schleife wird im String `source` nach dem ersten Zeichen von `search` gesucht, aber beim Stringende von `source` (wenn die Variable `c` zu Null wird) die While-Schleife beendet.

```
    s=search[0]; // erstes Zeichen von search nehmen
    i=0;
    do // in äußerer Schleife nach erstem Zeichen
    { // von search suchen
        c=source[i];
        if(c==s) // erstes Zeichen von search gefunden!
        {
...
        }
        i++;

    } while(c); // c==0 – Stringende dann aufhören
```

Ist die erste Übereinstimmung von `source` mit `search` gefunden, werden die restlichen Buchstaben von `search` auf ihr Auftreten in `source` überprüft:

```
            j=1; // Schleifenzähler initialisieren
            do // in innerer Schleife Reststring vergleichen
            {
                v=search[j];
            } while(v!=0 && v==source[i+j++]);
            if(!v) return(i); // Wenn v==0 String gefunden
```

Am Ende der inneren Schleife muss dann geprüft werden, ob die innere Schleife abgebrochen wurde, weil der String `search` komplett überprüft wurde (Abbruch wenn `v` gleich Null wird), oder ob der Reststring nicht übereinstimmte (Abbruch wenn `v` ungleich `source[i+j++]`). War demnach `v` gleich Null, ist die Übereinstimmung komplett und der korrekte Index wird zurückgegeben. Ansonsten wird weiter in der äußeren Schleife nach einem Auftreten vom ersten Zeichen von `search` in `source` gesucht.

Die BASIC-Variante von `Str_SubStr()`:

```
Sub Str_SubStr(ByRef source As Char,ByRef search As Char) As Integer
    Dim i,j,len As Integer
    Dim s,c,v As Char

    s=search(0) 'erstes Zeichen von search nehmen
    i=0 'Schleifenzähler initialisieren

    Do 'in äußerer Schleife nach erstem Zeichen
        'von search suchen

        c=source(i)
        If c = s Then 'erstes Zeichen von search gefunden!
            j=0
            Do 'in innerer Schleife Reststring vergleichen
```

```
j=j+1
                v=search(j)
            Loop While v<>0 And v = source(i+j)
            If v=0 Then 'Wenn v==0 String gefunden
                Return i
            End If
        End If
        i=i+1

    Loop While c // bei Stringende aufhören

    Return -1 // keine Übereinstimmung gefunden
End Sub
```

Die Arbeitsweise der BASIC-Variante von `Str_SubStr()` ist die gleiche wie in CompactC: In der äußeren Schleife wird im String `source` nach dem ersten Zeichen von `search` gesucht, aber beim Stringende von `source` (wenn die Variable `c` zu Null wird) die Loop-While-Schleife beendet.

```
    s=search(0) 'erstes Zeichen von search nehmen
    i=0 'Schleifenzähler initialisieren

    Do 'in äußerer Schleife nach erstem Zeichen
        'von search suchen

        c=source(i)
        If c = s Then 'erstes Zeichen von search gefunden!
...
        End If
        i=i+1

    Loop While c // bei Stringende aufhören
```

Ist die erste Übereinstimmung von `source` mit `search` gefunden, werden die restlichen Buchstaben von `search` auf ihr Auftreten in `source` überprüft:

```
            j=0
            Do 'in innerer Schleife Reststring vergleichen
                j=j+1
                v=search(j)
            Loop While v<>0 And v = source(i+j)
            If v=0 Then 'Wenn v==0 String gefunden
                Return i
            End If
```

Am Ende der inneren Schleife muss dann geprüft werden, ob die innere Schleife abgebrochen wurde, weil der String `search` komplett überprüft wurde (Abbruch wenn `v` gleich Null) oder ob der Reststring nicht übereinstimmte (Abbruch wenn `v` ungleich `source(i+j))`. Ist v gleich Null, dann ist die Übereinstimmung komplett und der korrekte Index wird zurückgegeben. Ansonsten wird weiter in der äußeren Schleife nach einem Auftreten vom ersten Zeichen von `search` in `source` gesucht.

Der erfahrene Programmierer, der C und BASIC gut beherrscht, wird bemerken, dass in der BASIC-Variante `j` mit `0` statt mit `1` initialisiert wird. Auch wird die Inkrementierung von `j` (`j=j+1`) an den Anfang der inneren Schleife gestellt. Dies ist nötig, da die BASIC-Syntax den Inkrementierungs-Operator (`++`) nicht beinhaltet.

13.5 Steuerzeichen

Man kann in Stringzuweisungen spezielle Steuerzeichen einbringen, die sonst im Alphabet nicht darstellbar sind. Die Steuerzeichen beginnen immer mit einem ‚\' (Backslash), gefolgt von einem weiteren Zeichen. Möchte man das Zeichen ‚\' in einer Stringzuweisung stehen haben, dann muss man es zweimal schreiben ‚\\'.

Steuerzeichen	num. Wert	Beschreibung
\"	63	Anführungszeichen
\'	93	einzelnes Anführungszeichen
\a	7	Glocke (Bell)
\b	8	Rücktaste (Backspace)
\t	9	Tabulator
\n	10	Zeilenvorschub (Linefeed)
\v	11	Vertikaler Tabulator
\f	12	Form feed
\r	13	Carriage Return
\\	92	Rückstrich (Backslash) selber

Gerade im Umgang mit Ausgabegeräten, seien sie seriell an das C-Control-Pro-Modul angeschlossen, oder im Fall der Debug-Ausgabe des Systems, erleichtern die Steuerzeichen die Textformatierung und andere Steuerfunktionen.

Möchte man solche Steuerzeichen selbst erzeugen, dann kann man natürlich auch mit einem Array-Zugriff ein Zeichen mit dem entsprechenden Wert überschreiben. Man würde dann bei der Zuweisung Füllzeichen verwenden, die später einen neuen Wert bekommen.

CompactC:

```
Source="HalloxWeltx";
Source[5]=13;
Source[10]=13;
```

BASIC

```
Source="HalloxWeltx";
Source(5)=13;
Source(10)=13;
```

In dem Beispiel wurde das Füllzeichen „x" durch ein Carriage-Return ausgetauscht.

13.6 Formatierung numerischer Werte

In der C-Control-Pro-Bibliothek steht für die formatierte Stringausgabe die Funktion `Str_WriteWord` zur Verfügung. Sie hat die 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)

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
```

Mit `Str_WriteWord` kann man nicht nur Zahlen in einen String verwandeln, man kann auch mit dem `base`-Parameter die Zahlenbasis angeben, und mit `minwidth` die minimale Breite bestimmen. Klassische Zahlenbasen sind 10 für die Dezimalzahlen, 16 für Hexadezimale Zahlen, und 2 für die binäre Darstellung. Wird in der Ausgabe von `Str_WriteWord` die minimale Anzahl von Ziffern, die von `minwidth` festgelegt wird, unterschritten, so wird von rechts mit Nullen aufgefüllt.

Angenommen, man möchte folgende Ausgabe erzeugen: Die Zahlen von 0 bis 255 in dezimal, hexadezimal und binär:

```
Dezimal: 000 Hexadezimal: 00 Binär: 00000000
Dezimal: 001 Hexadezimal: 01 Binär: 00000001
...
Dezimal: 254 Hexadezimal: FE Binär: 11111110
Dezimal: 255 Hexadezimal: FF Binär: 11111111
```

So sieht das in CompactC aus:

```
void main(void)
{
  word i; // Schleifenzähler
  char text[60];

  for(i=0;i<256;i++) // Schleife von 0–255
  {
    // Maske zuweisen
    text=„Dezimal: 000 Hexadezimal: 00 Binär: 00000000\r“;
    Str_WriteWord(i,10,text,9,3); // Zahl mit Basis 10
    Str_WriteWord(i,16,text,26,2);// Zahl mit Basis 16
    Str_WriteWord(i,2,text,37,8); // Binärzahl
    text[12]=’ ’; // Null hinter 1. Zahl löschen
    text[28]=’ ’; // Null hinter 2. Zahl löschen
    text[45]=’\r’; // Null hinter 3. Zahl löschen

    Msg_WriteText(text);
  }
}
```

und in BASIC:

```
Sub main()
  Dim i As Integer ' Schleifenzähler
  Dim text(60) As Char

  For i=0 To 255 ' Schleife von 0–255
    ' Maske zuweisen
    text=„Dezimal: 000 Hexadezimal: 00 Binär: 00000000\n“
    Str_WriteWord(i,10,text,9,3) ' Zahl mit Basis 10
    Str_WriteWord(i,16,text,26,2) ' Zahl mit Basis 16
    Str_WriteWord(i,2,text,37,8) ' Binärzahl
    text(12)=32 ' Null hinter 1. Zahl löschen
    text(28)=32 ' Null hinter 2. Zahl löschen
    text(45)=13 ' Null hinter 3. Zahl löschen
    Msg_WriteText(text)
  Next
End Sub
```

Auffällig sind die drei Zuweisungen nach dem letzten `Str_WriteWord()`-Aufruf. Diese sind notwendig, da `Str_WriteWord()` nach dem Erzeugen des Strings auch eine den String terminierende Null in den Zielstring schreibt. Dieses Stringende wird mit den Zuweisungen aufgehoben und durch ein Leerzeichen bzw. ein Carriage-Return ersetzt. Natürlich ist es ein wenig umständlich, so zu arbeiten, aber jeder kann sich ja seine eigene Bibliothek und weitaus mächtigere Versionen von `Str_WriteWord()` selbst erschaffen.

14 Optimierung von CompactC

14.1 Optimierung ist Programmierersache

Der CompactC-Compiler des C-Control-Pro-Systems ist ein Bytecode-Compiler. Er erzeugt ähnlich wie ein Java-Compiler Instruktionen, die dann vom Interpreter auf dem Atmel Mega32 oder Mega128 abgearbeitet werden. Der Compiler hat jedoch keinen „Global Optimizer“, wie es manche teure Entwicklungssysteme haben, die oft mehrere tausend Euro kosten. Solch ein Optimizer braucht in der Entwicklung mehrere Mannjahre, und dies für jede Programmiersprache. Da C-Control Pro CompactC und BASIC unterstützt, wäre hier noch einmal die doppelte Entwicklungszeit nötig gewesen. Deshalb muss beim C-Control-Pro-System der Programmierer selbst optimieren.

Regel: Es gilt hier die Faustregel, je kürzer das Programm (also je weniger Bytecodes der Interpreter abarbeiten muss), umso effizienter läuft das Programm.

Es gibt einige Ausnahmefälle, wie z. B. kleine Schleifen, bei denen es ratsam ist, die Schleife aufzulösen. In diesem Beispiel hat die For-Schleife 3 Durchläufe, sodass die Funktion `fun()` dreimal aufgerufen wird.

```
for(i=0;i<3;i++)
{
  func(i);
}
```

Man kann die Zeit der Schleifenbefehle selbst einsparen, in dem man den Programmcode in der Schleife direkt ausführt. Man ruft die Funktion `func()` direkt dreimal auf. Dies macht nur bei kleinen Schleifen Sinn, und die Programme werden durch diese Optimierung länger:

```
func(0);
func(1);
func(2);
```

14.2 Optimierung Schritt für Schritt

An dieser Stelle wird die Funktion `Str_Copy2()` betrachtet, die schon im Kapitel 13 „Stringverarbeitung“ vorgestellt wurde. Mit ihr kann man einen String aus dem Array `source` in das Array `destination` kopieren. Sie ist von ihrer Funktionalität ähnlich der Funktion `Str_Copy()` aus der Stringbibliothek (siehe C-Control-Benutzerhandbuch Kapitel „Bibliotheken“, Unterkapitel „Strings“). Zur Vereinfachung wird aber der Parameter `offset` von der Original-`Str_Copy()`-Version in diesem Beispiel weggelassen.

Die Funktion `Str_Copy2()` dient an dieser Stelle als Beispiel, um mögliche Optimierungen zu verdeutlichen:

```
void Str_Copy(char destination[],char source[])
{
  int i;

  i=0;
  while(true) // Endlosschleife
  {
    destination[i]=source[i]; //ein Zeichen kopieren
    if(destination[i]==0) break; // bei Null abbrechen
    i=i+1; // Index erhöhen
  }
}
```

Die erste Verbesserung ist die Benutzung des Inkrement-Operators. In CompactC (wie im normalen ANSI-C auch) existieren Inkrement- und Dekrement-Operatoren , die den Inhalt einer Variablen auslesen und gleichzeitig den Inhalt um eins (`++`) erhöhen, oder um eins (`--`) verringern. Steht der Operator vor der Variablen (`++x`), wird der Inhalt erhöht, bevor der Inhalt ausgelesen wird. Steht der Operator hinter der Variablen (`x++`), wird erst der Inhalt gelesen und dann inkrementiert.

Beispiele:

```
a=1; x=++a; // x =2, a=2
a=1; x=a++; // x =1, a=2
a=1; x=--a; // x =0, a=0
a=1; x=a--; // x =1, a=0
```

Gegenüber der Zuweisung `i=i+1`, die vier Bytecodes benötigt, kann man mit `i++` zwei Bytecodes einsparen. Als Zusatznutzen liefert `i++` auch den Wert der Variablen zurück.

Tipp: Im Interpreter existieren hochspezialisierte Bytecodes, die Zeiteinsparungen ermöglichen, z. B. bei den Inkrement- und Dekrement-Operatoren sowie auch bei `switch`-Anweisungen.

Die neue Version von `Str_Copy2()` sieht dann so aus:

```
void Str_Copy2(char destination[],char source[])
{
  int i;

  i=0;
  while(true) // Endlosschleife
  {
    destination[i]=source[i]; // ein Zeichen kopieren
    if(destination[i++]==0) break; // bei Null abbrechen
  }
}
```

Im nächsten Schritt wird keine Endlosschleife benutzt und dann mit einem `break`-Befehl die Schleife verlassen, sondern man verlagert das Abbruchkriterium in die `while`-Bedingung. Dies sieht dann so aus:

```
void Str_Copy2(char destination[],char source[])
{
  int i;

  i=0;
  do
  {
    destination[i]=source[i]; //ein Zeichen kopieren
  } while(destination[i++]!=0);
}
```

Vergleichsoperatoren wie z. B. gleich (`==`), kleiner (`<`) oder größer (`>`), vergleichen 2 Werte und liefern dann als Resultat 0 („falsch“) oder 1 („wahr“) zurück, je nachdem, ob der Vergleich erfüllt wurde oder nicht. Der Vergleich `1==0` liefert z. B. das Ergebnis 0 und `2==2` das Ergebnis 1.

Bedingungen wie `if` oder `while` werden dann ausgeführt, wenn der Ausdruck ungleich 0 ist. Dies ist korrekt, da ein nicht erfüllter Vergleich den Wert 0 ergibt.

In der Zeile

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

liefert der Vergleich `2==2` den Wert 1, also wird das `if` ausgeführt.

```
if(1==0) a++;
```

Hier wird der Vergleich `1==0` zu 0 ausgewertet, das `if` wird nicht ausgeführt.

Man kann daher statt der Zeile

```
if(a!=0) x++;
```

auch

```
if(a) x++;
```

schreiben. Im Gegensatz zum Vergleich `a!=0` spart man hierbei zwei Bytecodes, die nicht ausgeführt werden müssen.

Die endgültige Version von `Str_Copy2()` lautet:

```
void Str_Copy2(char destination[],char source[])
{
  int i;

  i=0;
  do
  {
    destination[i]=source[i]; //ein Zeichen kopieren
  } while(destination[i++]);
}
```

Möchte man diese Optimierung auch in booleschen Verknüpfungen wie *Und* bzw. *Oder* einsetzen, muss man darauf achten, auch die logischen Varianten von *Und* und *Oder* zu benutzen. Das folgende Beispiel ist korrekt:

```
a=2;
if(a && a>0) ...
```

Der `if`-Befehl wird ausgeführt.

Im Falle von

```
a=2;
if(a & a>0) ...
```

liefert der Vergleich `a>0` zwar immer noch den Wert 1, da `&` aber das binäre *Und* ist, ergibt `2 & 1` den Wert 0. Der `if`-Befehl wird nicht ausgeführt.

Tipp: Falls möglich, immer Inkrement- oder Dekrement-Operatoren benutzen.

14.3 Switch-Anweisungen sind effizient

Hat man in einem Programm ein Codefragment wie dieses:

```
if(a==5) x=2;
if(a==8)
{
  x=2;
  y=3;
}
if(a==15)
{
  x=5;
  y=x+y;
}
```

dann lässt es sich viel platzsparender und effizienter wie folgt schreiben:

```
switch(a)
{
  case 5:
    x=2;
  break;
  case 8:
```

```
    x=2;
    y=3;
  break;
  case 15:
    x=5;
    y=x+y;
  break;
}
```

Dies liegt in der Fähigkeit des CompactC-Compilers, eine `switch`-Anweisung in eine Sprungtabelle umzuwandeln. Ein einzelner Bytecode führt den Sprung zum richtigen Code aus, ohne dass mehrere `if`-Befehle ausgeführt werden. In der internen Darstellung sieht der `switch()`-Befehl ungefähr so aus:

```
Wert 5 – label 1;Wert 8 – label 2;Wert 15 – label 3;

switch Bytecode
label 1:
  x=2;
  goto end;
label 2:
  x=2;
  y=3;
  goto end;
label 3:
  x=5;
  y=x+y;
  goto end ;
label end ;
```

Zu jedem Wert, der hinter einem `case`-Befehl vorkommt, in unserem Falle die Werte (5, 8,15), wird in der Sprungtabelle eine Sprungmarke (label) generiert. Der `switch`-Bytecode sucht den passenden Wert und benutzt dann die zugehörige Sprungmarke, um zum richtigen Codefragment zu springen. Der `break`-Befehl springt dann hinter den `switch`-Befehl. Dies ist mit dem Befehl `goto end` symbolisiert.

Tipp: `Switch`-Anweisungen sparen Zeit und Programmcode. Das Einsparungspotenzial mit einer `switch`-Anweisung ist von allen vorgestellten Optimierungen das größte.

14.4 Arithmetische Ausdrücke vereinfachen

Rein numerische Ausdrücke wie z. B.

```
a=5*17–8+7;
```

werden vom CompactC-Compiler zu

```
b=84;
```

vereinfacht.

Sobald aber an einer Stelle Variablen in einem Ausdruck auftauchen,

```
b=5*17-8+a+6*7/3+4*a;
```

generiert der Compiler ab dem Auftauchen der Variablen Programmcode, um den Ausdruck zu berechnen. In unserem Beispiel wird dann intern `5*17-8` berechnet und der restliche Ausdruck dann nicht weiter ausgewertet:

```
b=77+a+6*7/3+4*a;
```

Möchte man Terme nicht selbst ausrechnen (weil man z. B. den Rechenschritt sehen soll), kann man rein numerische Terme klammern. Klammern werden zuerst berechnet, und werden komplett aufgelöst, wenn keine Variablen darin vorkommen:

```
b=(5*17-8)+a+(6*7/3)+4*a;
```

Dieser Ausdruck wird intern dann so ausgedrückt:

```
b=77+a+14+4*a;
```

Am effizientesten ist es natürlich, wenn der Programmierer selbst die Vereinfachung vornimmt und z. B. in diesem Fall die Variablen ausmultipliziert:

```
b=5*a+91;
```

Tipp: Ausdrücke möglichst per Hand optimieren.

Im Gegensatz zu anderen C-Compilern lohnt es sich beim CompactC-Compiler nicht, Multiplikationen oder Divisionen durch Bitschiebe-Befehle zu realisieren.

Die Anweisungen

```
a=4*i;
```

und

```
a=i<<2;
```

haben im Bytecode-Interpreter die gleiche Laufzeit.

14.5 Eingliedern von Funktionen

Nicht immer macht die Definition von Funktionen Sinn. Werden sie nur von wenigen Stellen im Programm aufgerufen, kann es ratsam sein, die Funktionsinhalte an die Stelle des Funktionsaufrufes zu kopieren. Denn die Übergabe von Parametern an Funktionen, sowie der Funktionsaufruf selbst, kosten Zeit.

Dieses einfache Beispiel verdeutlicht eine mögliche Optimierung:

```
int sum(int a, int b)
{
  int c;

  c=a+b;
  return(c);
}

void main(void)
{
  int i,j;

  for(i=0;i<1000;i++)
  {
    for(j=0;j<1000;j++)
    {
      sum(i,j);
    }
  }
}
```

Eine Verbesserung ist dann:

```
void main(void)
{
  int i,j,c;

  for(i=0;i<1000;i++)
  {
    for(j=0;j<1000;j++)
    {
      c=i+j;
    }
  }
}
```

Natürlich sind diese Optimierungen in der Praxis nicht so simpel wie dieses konstruierte Beispiel.

Auch lassen sich ineinander geschachtelte `for`-Schleifen manchmal zu einer Schleife zusammenfassen. Eine Funktion mit zwei `for`-Schleifen:

```
void main(void)
{
  int i,j;
  int a[100][10];

  for(i=0;i<100;i++)
  {
    for(j=0;j<10;j++)
    {
      a[i][j]=i+j;
    }
  }
}
```

Beide Schleifen zu einer `for`-Schleife zusammengefasst:

```
void main(void)
{
  int i,j;
  int a[1000];
```

```
  for(i=0;i<1000;i++)
  {
    a[i]=(i/100)+(i%10);
  }
}
```

In diesem Beispiel wurde eine Abbildung von einem zweidimensionalen Array (100x10) in ein eindimensionales Array (1000) gemacht. Nicht immer lohnt sich solch ein Aufwand. In diesem Beispiel schon, da Array-Zugriffe umso mehr Zeit brauchen, je mehr Dimensionen sie haben.

Ein Array-Zugriff

```
int a[6][8][10];
a[i][j][k]=0;
```

sieht für den Compiler intern so aus:

```
int a[6][8][10];
a[i*80+j*10+k]=0;
```

Tipp: Je mehr Dimensionen ein Array hat, desto länger dauern die Array-Zugriffe.

14.6 Einsparen von Programmcode

Auch das folgende Programm ist wieder konstruiert, um möglichst deutlich die Problematik von unbenutztem Programmcode aufzuzeigen. Es hat daher keinen sinnvollen Nutzen, außer um als Beispiel zu dienen:

```
int func1(int a, int b)
{
  return(a+b);
}

int func2(int a, int b)
{
  return(a*b);
}

int i,c,d,e,f;

void main(void)
{
  int i,j;

  i=0;
  while(true)
  {
    c=func1(i,i+1);
    if(c>0) goto exit1;
    i++;
  }
  e=func1(c,i);

 exit1:
  goto exit2;
```

```
  d=c*7;

 exit2:
}
```

Es ist auf den ersten Blick ersichtlich, dass die Funktion `func2()` und die Variablen `f` und `j` nicht genutzt werden. Trotzdem verbraucht `func2()` Programmcode im Flashspeicher des C-Control-Pro-Moduls, und die Variablen `f` und `j` Platz im RAM. Man sollte die Funktion `func2()` und die nicht genutzten Variablen entfernen:

```
int func1(int a, int b)
{
  return(a+b);
}

int i,c,d,e;

void main(void)
{
  int i;

  i=0;
  while(true)
  {
  c=func1(i,i+1);
    if(c>i+100) goto exit1;
    i++;
  }
  e=func1(c,i);

 exit1:
  goto exit2;
  d=c*7;

 exit2:
}
```

Auf den zweiten Blick erkennt man, dass die Anweisung `goto exit1;` immer zum Ende springt, und die folgende Anweisung `d=c*7;` niemals ausgeführt wird. Die Anweisung `d=c*7` kann deshalb entfernt werden. Im folgenden merkt man, dass bei folgendem Code das `goto exit2;` unnötig ist, da sofort danach die Funktion (bzw. das Hauptprogramm) verlassen wird.

```
  goto exit2;
  exit2:
}
```

Die Variable `i` ist zweimal definiert, einmal global und einmal lokal in der Funktion `main()`. Da `i` nur in `main()` genutzt wird, kann die globale Variable `i` gelöscht werden.

Wenn man sich die Schleife

```
  while(true)
  {
  c=func1(i,i+1);
    if(c>i+100) goto exit1;
    i++;
  }
  e=func1(c,i);
```

genauer ansieht, sieht man eine Eigenschaft in der Programmlogik: Die Schleife kann nur mit einem Sprung zum label `exit1` verlassen werden. Daraus folgt, dass die Anweisung `e=func1(c,i);` niemals ausgeführt werden kann.

Wirft man nun allen Ballast weg, bleibt dann von unserem ursprünglichen Programm der folgende Quelltext übrig:

```
int func1(int a, int b)
{
  return(a+b);
}

int c,d,e;

void main(void)
{
  int i;

  i=0;
  while(true)
  {
  c=func1(i,i+1);
    if(c>0) goto exit1;
    i++;
  }
 exit1:
}
```

Tipp: Der CompactC-Compiler erkennt keinen unbenutzten Programmcode oder unbenutzte Variablen. Der Anwender muss selbst im Auge behalten, ob Programmressourcen wirklich genutzt werden.

14.7 Projektoptionen prüfen

Ist in den Projektoptionen die Erzeugung von Debug-Code eingeschaltet, wird im generierten Bytecode für jede Zeile des Programmtextes, der Anweisungen enthält, ein spezieller Debug-Bytecode generiert. Man kann dies gut im Editor erkennen. Kann man in einer Zeile im Programmtext einen Breakpoint setzen, so ist im Programm für diese Stelle ein Debug-Bytecode erzeugt worden.

Durch Einsatz des Debug-Bytecodes wird erreicht, dass der Debugger nur einmal pro Zeile prüft, ob ein Breakpoint erreicht wurde. Ansonsten müsste diese Überprüfung bei der Abarbeitung jedes Bytecodes durchgeführt werden. Schaltet man die Erzeugung von Debug-Code aus, werden die Programme kleiner und es wird keine Zeit darauf verwandt zu untersuchen, ob an dieser Stelle ein Breakpoint gesetzt war.

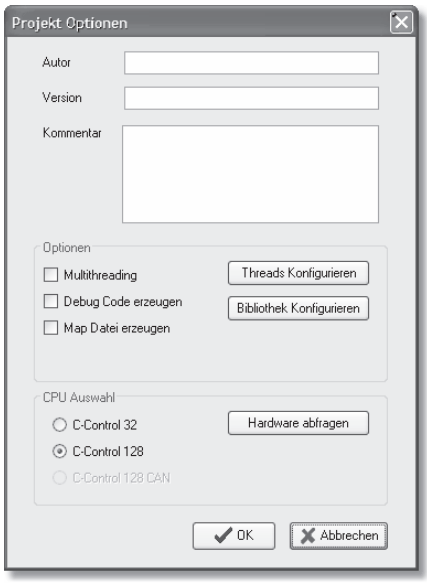


Abb. 14.1: Projektoptionen

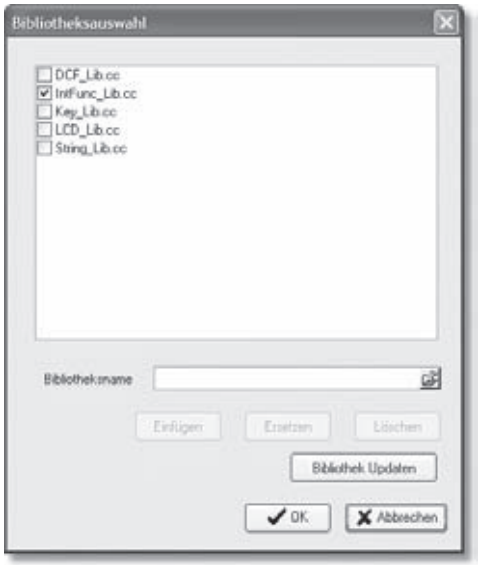


Abb. 14.2: Bibliotheksoptionen

Bei der Auswahl der Bibliotheken ist nur die Bibliothek „IntFunc_Lib.cc“ zwingend notwendig. Alle anderen Bibliotheken sollten nur dann angewählt werden, wenn man den dort benutzten Programmcode wirklich nutzt, da der eingefügte Programmcode sonst Platz im Flashspeicher verbraucht.

Ist der Platz sehr begrenzt, ist es unter Umständen sinnvoll, die Funktionen in den Bibliotheksdateien auf ihre Nutzung hin zu untersuchen. Braucht man z. B. nicht alle Funktionen aus der „String_Lib.cc“, spart es Platz, die nicht verwendeten Funktionen zu entfernen. Vor solch einer Aktion ist es aber ratsam, diese Änderungen an einer Kopie der Bibliothek vorzunehmen und diese Kopie dann als Bibliothek einzubinden.

Ist in den Projektoptionen das Multithreading eingeschaltet, ist die Threadkonfiguration aktiv. Für jeden weiteren Thread, außer dem Hauptthread, wird die dort eingetragene Stackgröße im Speicher reserviert. Gerade bei dem sehr knappen RAM-Speicher des Mega32-Moduls, sollte man darauf achten, dass nicht unnötig an dieser Stelle Speicher verbraucht wird.

Tipp: Man sollte in den Programmoptionen überprüfen, ob die Erzeugung von Debug-Code ausgeschaltet ist, die eingebundenen Bibliotheken genutzt werden, und ob die Optionen für das Multithreading korrekt sind.

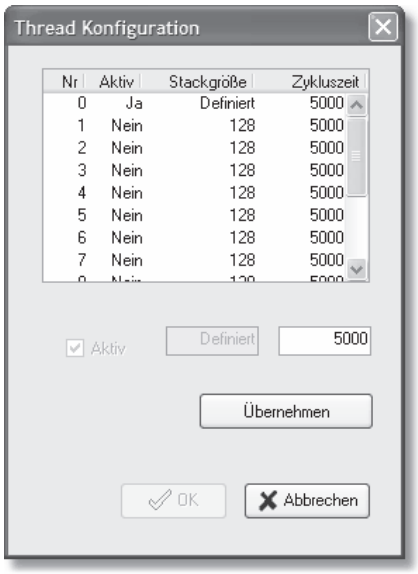


Abb. 14.3: Threadkonfiguration

15 Optimierung von BASIC

15.1 Optimierung ist Programmierersache

Der BASIC-Compiler des C-Control-Pro-Systems ist ein Bytecode-Compiler. Er erzeugt ähnlich wie ein Java-Compiler Instruktionen, die dann vom Interpreter auf dem Atmel Mega32 oder Mega128 abgearbeitet werden. Der Compiler hat jedoch keinen „Global Optimizer“, wie es manche teure Entwicklungssysteme haben, die oft mehrere Tausend Euro kosten. Solch ein Optimizer braucht in der Entwicklung mehrere Mannjahre und dies für jede Programmiersprache. Da C-Control Pro CompactC und BASIC unterstützt, wäre hier noch einmal die doppelte Entwicklungszeit nötig gewesen. Deshalb muss beim C-Control-Pro-System der Programmierer selbst optimieren.

Regel: Es gilt hier die Faustregel, je kürzer das Programm (also je weniger Bytecodes der Interpreter abarbeiten muss), umso effizienter läuft das Programm.

Es gibt davon seltene Ausnahmefälle, wie z. B. kleine Schleifen, wo es ratsam sein kann die Schleife aufzulösen. In diesem Beispiel hat die For-Next-Schleife 3 Durchläufe, so dass die Funktion `fun()` dreimal aufgerufen wird.

```
For i=0 To 2
  func(i)
Next
```

Man kann die Zeit der Schleifenbefehle selbst einsparen, indem man den Programmcode in der Schleife direkt ausführt. Man ruft die Funktion `func()` direkt dreimal auf. Dies macht nur bei kleinen Schleifen Sinn, und die Programme werden durch diese Optimierung länger:

```
func(0)
func(1)
func(2)
```

15.2 Optimierung Schritt für Schritt

An dieser Stelle wird die Funktion `Str_Copy2()` betrachtet, die schon im Kapitel 13 „Stringverarbeitung“ vorgestellt wurde. Mit ihr kann man einen String aus dem Array `source` in das Array `destination` kopieren. Sie ist von ihrer Funktionalität

ähnlich der Funktion `Str_Copy()` aus der Stringbibliothek (siehe C-Control-Benutzerhandbuch Kapitel „Bibliotheken“, Unterkapitel „Strings“). Zur Vereinfachung wird aber der Parameter `offset` von der Original-`Str_Copy()`-Version in diesem Beispiel weggelassen.

Die Funktion `Str_Copy2()` dient an dieser Stelle als Beispiel, um mögliche Optimierungen zu verdeutlichen:

```
Sub Str_Copy2(ByRef destination As Char,ByRef source As Char)
  Dim i As Integer

  i=0
  Do While True
    destination(i)=source(i) ' ein Zeichen kopieren
  If destination(i) = 0 Then
      Exit ' bei Null abbrechen
    End If
    i=i+1 ' Index erhöhen
  End While
End Sub
```

Im ersten Schritt wird keine Endlosschleife benutzt, und dann mit einem `Exit`-Befehl die Schleife verlassen, sondern man verlagert das Abbruchkriterium in die Loop-While-Bedingung. Dies sieht dann so aus:

```
Sub Str_Copy2(ByRef destination As Char,ByRef source As Char)
Dim i As Integer

    i=-1
    Do
    i=i+1 ' Index erhöhen
    destination(i)=source(i) ' ein Zeichen kopieren
    Loop While destination(i)<>0 ' bei Null abbrechen
End Sub
```

Vergleichsoperatoren wie z. B. gleich (`=`), kleiner (`<`) oder größer (`>`), vergleichen 2 Werte und liefern dann als Resultat 0 („falsch“) oder 1 („wahr“) zurück, je nachdem, ob der Vergleich erfüllt wurde oder nicht. Der Vergleich `1 = 0` liefert z. B. das Ergebnis 0 und `2 = 2` das Ergebnis 1.

Bedingungen wie `If` oder `Do-While` werden dann ausgeführt, wenn der Ausdruck ungleich 0 ist. Dies ist korrekt, da ein nicht erfüllter Vergleich den Wert 0 ergibt.

In der nächsten Zeile

```
If 2 = 2 Then ...
```

liefert der Vergleich `2 = 2` den Wert 1, also wird das `If` ausgeführt.

```
If 1 = 0) Then ...
```

Hier wird der Vergleich `1 = 0` zu 0 ausgewertet, das `If` wird nicht ausgeführt.

Man kann daher statt der Zeile

```
If a <> 0 Then ...
```

auch

```
If a Then ...
```

schreiben. Im Gegensatz zum Vergleich `a <> 0` spart man hierbei zwei Bytecodes, die nicht ausgeführt werden müssen.

Die endgültige Version von `Str_Copy2` () lautet:

```
Sub Str_Copy2(ByRef destination As Char,ByRef source As Char)
Dim i As Integer

    i=-1
    Do
i=i+1 ' Index erhöhen
destination(i)=source(i) ' ein Zeichen kopieren
    Loop While destination(i) ' bei Null abbrechen
End Sub
```

Möchte man diese Optimierung auch in booleschen Verknüpfungen wie *Und* bzw. *Oder* einsetzen, muss man darauf achten, dass BASIC im Gegensatz zu CompactC nur die binären Versionen von *Und, Oder* etc. kennt. CompactC besitzt auch die logischen Varianten der booleschen Operationen. Benutzt man die binären Operatoren in einem Berechnungsausdruck, so muss man doch `a <> 0` anstatt nur `a` schreiben:

```
a=2
If a<>0 And a>0 Then ...
```

Korrekt, der `If`-Befehl wird ausgeführt.

Im Falle von

```
a=2
If a And a>0 Then ...
```

liefert der Vergleich `a > 0` zwar immer noch den Wert 1, da `And` aber das binäre *Und* ist, ergibt `2 And 1` den Wert 0. Der `If`-Befehl wird nicht ausgeführt.

15.3 Select-Case-Anweisungen sind effizient

Hat man in einem Programm ein Codefragment wie folgendes:

```
If a = 5 Then
  x=2
End If
If a = 8 Then
  x=2
  y=3
End If
If a = 15 Then
  x=5
  y=x+y
End If
```

dann lässt es sich viel platzsparender und effizienter wie folgt schreiben:

```
Select Case a
  Case 5
    x=2
  Case 8
    x=2
    y=3
  Case 15
    x=5
    y=x+y
}
```

Dies liegt in der Fähigkeit des BASIC-Compilers eine `Select-Case`-Anweisung in eine Sprungtabelle umzuwandeln. Ein einzelner Bytecode führt den Sprung zum richtigen Code aus, ohne das mehrere `If`-Befehle ausgeführt werden. In der internen Darstellung sieht der `Select-Case`-Befehl ungefähr so aus:

```
Wert 5 – label 1;Wert 8 – label 2;Wert 15 – label 3;

switch Bytecode
label 1:
  x=2
  goto end
label 2:
  x=2
  y=3
  goto end
label 3:
  x=5
  y=x+y
  goto end
Lab end
```

Zu jedem Wert, der hinter einem `Case`-Befehl vorkommt, in unserem Falle die Werte (5, 8,15), wird in der Sprungtabelle eine Sprungmarke (label) generiert. Der `switch`-Bytecode sucht den passenden Wert und benutzt dann die zugehörige Sprungmarke, um zum richtigen Codefragment zu springen. Am Ende des Anweisungsblocks (vor dem nächsten `Case`-Befehl) springt man hinter den `Select-Case-`Befehl. Dies ist mit dem Befehl `goto end` symbolisiert.

Tipp: Select-Case-Anweisungen sparen Zeit und Programmcode. Von allen vorgestellten Optimierungen hat die Select-Case-Anweisung das größte Einsparungspotenzial.

15.4 For-Schleifen benutzen

Man kann in BASIC Schleifen, die eine bestimmte Anzahl von Durchläufen ausführen, auf mehrere Weisen realisieren. In diesem Beispiel realisiert als `For-Next-`Schleife und als `Do-While`-Schleife:

```
Sub main()
   Dim i As Integer

   For i=0 To 99
   Next

   i=0
   Do While i<100
     i=i+1
   End While
End Sub
```

Obwohl beide Schleifen das Gleiche machen, ist die `For-Next`-Schleife sehr viel effizienter. Ähnlich wie bei der `Select-Case`-Anweisung existieren spezielle Bytecodes, um die `For-Next`-Schleife zu beschleunigen. Benötigt in unserem Beispiel die `Do-While`-Schleife eine Ausführung von 9 Bytecodes für einen Schleifendurchlauf, wird der gleiche Schleifenzyklus in einer `For-Next`-Schleife in nur einem Bytecode ausgeführt.

Diese Optimierung existiert nur in BASIC. `For`-Schleifen in CompactC sind genauso langsam wie die `Do-While`-Schleife. Dies liegt daran, dass die `for`-Anweisung in CompactC viel mehr Freiheiten erlaubt als die `For-Next`-Befehle in BASIC.

Tipp: Wenn möglich, `For-Next`-Schleifen benutzen. Ein Schleifendurchlauf wird durch Abarbeitung eines Bytecodes ermöglicht.

15.5 Arithmetische Ausdrücke vereinfachen

Rein numerische Ausdrücke wie z. B.

```
b=5*17–8+7
```

werden vom BASIC-Compiler zu

```
b=84
```

vereinfacht.

Sobald aber an einer Stelle Variablen in einem Ausdruck auftauchen,

```
b=5*17–8+a+6*7/3+4*a
```

generiert der Compiler ab dem Auftreten der Variablen Programmcode, um den Ausdruck zu berechnen. In unserem Beispiel wird dann intern 5*17–8 berechnet, und der restliche Ausdruck dann nicht weiter ausgewertet:

```
b=77+a+6*7/3+4*a
```

Möchte man Terme nicht selbst ausrechnen (weil man z. B. den Rechenschritt sehen soll), kann man rein numerische Terme klammern. Klammern werden zuerst berechnet, und werden komplett aufgelöst, wenn keine Variablen darin vorkommen:

```
b=(5*17–8)+a+(6*7/3)+4*a
```

Dieser Ausdruck wird intern dann so ausgedrückt:

```
a=77+a+14+4*a
```

Am effizientesten ist es natürlich, wenn der Programmierer selbst die Vereinfachung vornimmt, und z. B. in diesem Fall die Variablen ausmultipliziert:

```
a=5*a+91
```

Tipp: Ausdrücke möglichst per Hand optimieren.

Im Gegensatz zu anderen Compilern lohnt es sich beim BASIC-Compiler nicht, Multiplikationen oder Divisionen durch Bitschiebe-Befehle zu realisieren.

Die Anweisungen

```
a=4*i
```

und

```
a=i<<2
```

haben im Bytecode-Interpreter die gleiche Laufzeit.

15.6 Eingliedern von Funktionen

Nicht immer macht die Definition von Funktionen Sinn. Werden sie nur von wenigen Stellen im Programm aufgerufen, kann es ratsam sein, die Funktionsinhalte an die Stelle des Funktionsaufrufes zu kopieren. Denn die Übergabe von Parametern an Funktionen, sowie der Funktionsaufruf selbst kosten Zeit.

Dieses einfache Beispiel verdeutlicht eine mögliche Optimierung:

```
Sub sum(a As Integer,b As Integer) As Integer
  Dim c As Integer

  c=a+b
  Return c
End Sub

Sub main()
  Dim i,j As Integer

  For i=0 To 999
    For j=0 To 999
      sum(i,j)
    Next
  Next
End Sub
```

Eine Verbesserung ist dann:

```
Sub main()

  Dim i,j,c As Integer

  For i=0 To 999
    For j=0 To 999
      C=i+j
    Next
  Next
End Sub
```

Natürlich sind diese Optimierungen in der Praxis nicht so simpel wie dieses konstruierte Beispiel.

Auch lassen sich ineinandergeschachtelte `For`-Schleifen manchmal zu einer Schleife zusammenfassen. Eine Funktion mit zwei `for`-Schleifen:

```
Sub main()

  Dim i,j As Integer
  Dim a(100,10) As Integer

  For i=0 To 99
    For j=0 To 9
      a(i,j)=0
    Next
  Next
End Sub
```

Beide Schleifen zu einer `for`-Schleife zusammengefasst:

```
Sub main()

  Dim i,j As Integer
  Dim a(1000) As Integer

  For i=0 To 999
    A(i)=(i/100)+(i Mod 10)
  Next
End Sub
```

In diesem Beispiel wurde eine Abbildung von einem zweidimensionalen Array (100x10) in ein eindimensionales Array (1000) gemacht. Nicht immer lohnt sich solch ein Aufwand. In dem Beispiel schon, da Array-Zugriffe desto mehr Zeit brauchen, je mehr Dimensionen sie haben.

Ein Array-Zugriff

```
Dim a(6)(8)(10) As Integer
A(i)(j)(k)=0
```

sieht für den Compiler intern so aus:

```
Dim a(6)(8)(10) As Integer
a(i*80+j*10+k)=0
```

Tipp: Je mehr Dimensionen ein Array hat, umso länger dauern die Array-Zugriffe.

15.7 Einsparen von Programmcode

Auch das folgende Programm ist wieder konstruiert, um möglichst deutlich die Problematik von unbenutztem Programmcode aufzuzeigen. Es hat daher keinen sinnvollen Nutzen, außer um als Beispiel zu dienen:

```
Sub func1(a As Integer, b As Integer) As Integer
Return a+b
End Sub

Sub func2(a As Integer, b As Integer) As Integer
Return a*b
End Sub

Dim i,c,d,e,f As Integer

Sub main()
  Dim i,j As Integer

  i=0
  Do While True
    c=func1(i,i+1)
    If c>0 Then
      Goto exit1
    End If
    i=i+1
  End While

  e=func1(c,i)

 Lab exit1
  Goto exit2
  d=c*7

 Lab exit2
End Sub
```

Es ist auf den ersten Blick ersichtlich, dass die Funktion `func2()` und die Variablen `f` und `j` nicht genutzt werden. Trotzdem verbraucht `func2()` Programmcode im Flashspeicher des C-Control-Pro-Moduls, und die Variablen `f` und `j` Platz im RAM. Man sollte die Funktion `func2()` und die nicht genutzten Variablen entfernen:

```
Sub func1(a As Integer, b As Integer) As Integer
Return a+b
End Sub

Dim i,c,d,e As Integer

Sub main()
  Dim i As Integer

  i=0
  Do While True
    c=func1(i,i+1)
```

```
    If c>0 Then
      Goto exit1
    End If
    i=i+1
  End While

  e=func1(c,i)

 Lab exit1
  Goto exit2
  d=c*7

 Lab exit2
End Sub
```

Auf den zweiten Blick erkennt man, dass die Anweisung `Goto exit1` immer zum Ende springt, und die folgende Anweisung `d=c*7` niemals ausgeführt wird. Die Anweisung `d=c*7` kann deshalb entfernt werden. Danach sieht man das bei dem Code:

```
  Goto exit2
  Lab exit2
End Sub
```

Die Anweisung `Goto exit2` ist unnötig, da sofort danach die Funktion (bzw. das Hauptprogramm) verlassen wird.

Die Variable `i` ist zweimal definiert. Einmal global und einmal lokal in der Funktion `main()`. Da `i` nur in `main()` genutzt wird, kann die globale Variable `i` gelöscht werden.

Wenn man sich die Schleife

```
  Do While True
    c=func1(i,i+1)
    If c>0 Then
      Goto exit1
    End If
    i=i+1
  End While
  e=func1(c,i)
```

genauer ansieht, sieht man eine Eigenschaft in der Programmlogik: Die Schleife kann nur mit einem Sprung zum Label `exit1` verlassen werden. Daraus folgt, dass die Anweisung `e=func1(c,i)` niemals ausgeführt werden kann.

Wirft man allen Ballast weg, bleibt von unserem ursprünglichen Programm der folgende Quelltext übrig:

```
Sub func1(a As Integer, b As Integer) As Integer
Return a+b
End Sub

Dim c,d,e As Integer

Sub main()
  Dim i As Integer

  i=0
  Do While True
    c=func1(i,i+1)
    If c>0 Then
```

```
      Goto exit1
    End If
    i=i+1
  End While
 Lab exit1
End Sub
```

Tipp: Der BASIC-Compiler erkennt keinen unbenutzten Programmcode oder unbenutzte Variablen. Der Anwender muss selbst im Auge behalten, ob Programmressourcen wirklich genutzt werden.

15.8 Projektoptionen prüfen

Ist in den Projektoptionen die Erzeugung von Debug-Code eingeschaltet, wird im generierten Bytecode für jede Zeile des Programmtexts, der Anweisungen enthält, ein spezieller Debug-Bytecode generiert. Man kann dies gut im Editor erkennen. Kann man in einer Zeile im Programmtext einen Breakpoint setzen, so ist im Programm für diese Stelle ein Debug-Bytecode erzeugt worden.

Durch Einsatz des Debug-Bytecodes wird erreicht, dass der Debugger nur einmal pro Zeile prüft, ob ein Breakpoint erreicht wurde. Ansonsten müsste diese Überprüfung bei der Abarbeitung jedes Bytecodes durchgeführt werden. Schaltet man die Erzeugung von Debug-Code aus, werden die Programme kleiner und es wird keine Zeit darauf verwandt, zu untersuchen, ob an dieser Stelle ein Breakpoint gesetzt war.

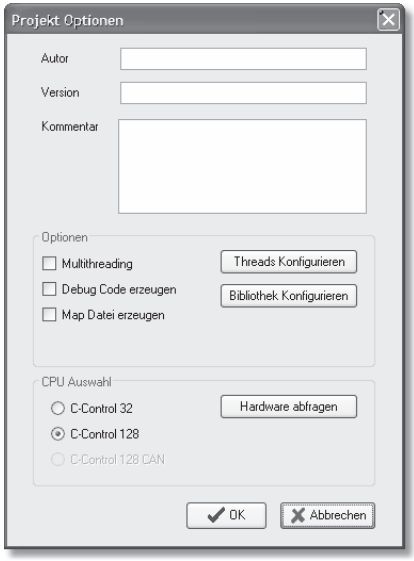


Abb. 15.1: Projektoptionen

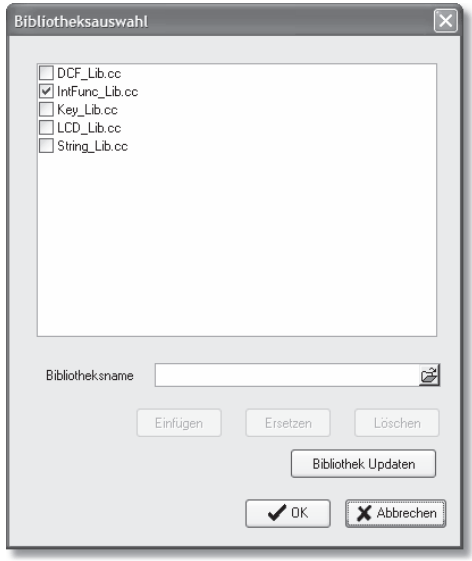


Abb. 15.2: Bibliotheksoptionen

Bei der Auswahl der Bibliotheken ist nur die Bibliothek „IntFunc_Lib.cc“ zwingend notwendig. Alle anderen Bibliotheken sollten nur dann angewählt werden, wenn man den dort benutzten Programmcode wirklich nutzt, da der eingefügte Programmcode sonst Platz im Flashspeicher verbraucht.

Ist der Platz sehr begrenzt, ist es unter Umständen sinnvoll, die Funktionen in den Bibliotheksdateien auf ihre Nutzung hin zu untersuchen. Braucht man z. B. nicht alle Funktionen aus der „String_Lib.cc“, spart es Platz, die nicht verwendeten Funktionen zu entfernen. Vor solch einer Aktion ist es aber ratsam, diese Änderungen an einer Kopie der Bibliothek vorzunehmen, und diese Kopie dann als Bibliothek einzubinden.

Abb. 15.3: Threadkonfiguration

Ist in den Projektoptionen das Multithreading eingeschaltet, ist die Threadkonfiguration aktiv. Für jeden weiteren Thread, außer dem Hauptthread, wird die dort eingetragene Stackgröße im Speicher reserviert. Gerade bei dem sehr knappen RAM-Speicher des Mega32-Moduls sollte man darauf achten, dass nicht unnötig an dieser Stelle Speicher verbraucht wird.

Tipp: Man sollte in den Programmoptionen überprüfen, ob die Erzeugung von Debug-Code ausgeschaltet ist, die eingebundenen Bibliotheken genutzt werden, und ob die Optionen für das Multithreading korrekt sind.

16 Der Preprozessor

Im C-Control-Pro-System ist ein kompletter Preprozessor enthalten. Der Name Preprozessor ergibt sich aus dem Umstand, dass der komplette Quelltext vor (lateinisch „prae“) dem Aufruf des Compilers vom Preprozessor bearbeitet wird.

Der Preprozessor, der hier zum Einsatz kommt, ist der „Gnu Generic Preprocessor“. Er wurde für die C-Control-Pro-IDE erweitert, da die Originalversion die korrekten Zeileninformationen nur fehlerhaft zurückgeliefert hat. Die Original-Quelltexte und auch die Modifikationen werden bei der Installation von C-Control Pro im Verzeichnis „GNU“ angelegt. Der „Gnu Generic Preprocessor“ ist weitaus mächtiger als die Funktionalität, die in diesem Kapitel erläutert wird. Eine ausführliche Beschreibung aller Befehle ist im Internet unter *http://nothingisreal.com/gpp/gpp.html* zu finden. Allerdings sind nur die Preprozessor-Befehle ausführlich getestet, die auch hier beschrieben werden. Eine Nutzung aller anderen Funktionen kann ein nicht vorhersagbares Verhalten zur Folge haben. Man sollte daher bei Experimenten eine Kopie des eigenen Programmtextes erstellt haben.

Auch wenn im weiteren Text hauptsächlich CompactC-Beispiele vorkommen, funktioniert doch der Preprozessor für CompactC und BASIC gleichermaßen.

Tipp: Alle Preprozessor-Anweisungen sind leicht durch das vorangestellte #-Zeichen erkennbar.

16.1 Definitionen

Die Hauptaufgabe des Preprozessors ist die Erstellung von Definitionen. Eine Definition sieht so aus:

```
#define Symbol
```

oder so:

```
#define Symbol Textkonstante
```

Einige Beispiele:

```
#define PI 3.141
#define REGEL_DEF
#define WELT_DEFINITION hallo Welt
```

Der Einsatz des Preprozessors kommt aus der Welt der C-Compiler. Es ist dort Tradition, dass Symbole immer groß geschrieben werden. Dies erleichtert ihre Erkennung im Programm.

Wird ein Symbol wie z. B. `PI` in einem Programm verwendet, wird vom Preprozessor an allen Stellen, wo der Text `PI` steht, `PI` durch den Text `3.141` ersetzt. Man muss sich im Klaren darüber sein, dass keinerlei syntaktische Tests stattfinden; es ist ein reines Austauschen von Zeichen im Quelltext. Symbole in Zeichenketten, also Quelltext, der in doppelten Anführungszeichen steht, werden nicht ausgetauscht.

Wichtig: Man muss bei der Erstellung von Definitionen Sorgfalt walten lassen; man kann mit gefährlichen Definitionen schnell schwer durchschaubare Phänomene erzeugen: Eine Definition wie `#define int float` führt beim späteren Lesen zum Unverständnis des Programms.

Es ist wichtig zu wissen, dass sich die Textkonstante bei Definitionen bis zum Ende der Zeile erstreckt. Im folgenden Beispiel gehört zum Symbol `MEINE_DEF` die Textkonstante `hallo Welt // meine Definition`.

```
#define MEINE_DEF hallo Welt // meine Definition
```

Man kann ein Symbol ohne Textkonstante definieren, wie z. B:

```
#define REGEL_DEF
```

Diese Konstante wird im Quelltext durch eine leere Zeichenkette ersetzt, d. h. alle auftretenden Symbole `REGEL_DEF` werden aus dem Programmtext entfernt.

Besteht ein Projekt aus mehreren Quelldateien, ist eine Definition für alle Quelldateien existent ab der Datei, in der die Konstante definiert wurde. Daher ist es möglich, die Reihenfolge der Quelldateien in einem Projekt zu ändern. Klickt man mit der rechten Maustaste auf eine Projektdatei, kann man in den Optionen mit „Nach oben“ und „Nach unten“ die Reihenfolge der Projektdateien beeinflussen. Die Dateien werden in der Reihenfolge von oben nach unten kompiliert.

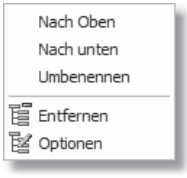


Abb. 16.1: Projekt-Optionsmenü mit rechter Maustaste

Benötigt man in einer in der Projekthierarchie weiter unten stehenden Datei eine Definition, die aber erst in einer späteren Datei definiert wird, sollte man mit der Option „Nach unten“ die aktuelle Datei hinter die Datei mit der Definition schieben.

Mit der Anweisung `#undef` wird eine Definition wieder entfernt. Auch hier gilt wieder, die Definition wird erst ab der Stelle im Programmtext, wo die `#undef`-Anweisung auftaucht, ungültig.

```
#undef PI
```

16.2 Bedingte Kompilierung

Der Preprozessor bietet die Möglichkeit, anhand von Definitionen zu entscheiden, ob bestimmte Programmzeilen im Quelltext verbleiben sollen oder gelöscht werden. Bei den Zeilen

```
#ifdef FUNKTIONSAUFRUF
  func1(a,b);
  func2(a,b);
#endif
```

werden die Zeilen `func1(a,b);` und `func2(a,b);` nur dann ausgeführt, wenn `FUNKTIONSAUFRUF` als Definition vorher gesetzt wurde.

Nach einem Befehl zur bedingten Kompilierung (im letzten Beispiel `#ifdef FUNKTIONSAUFRUF`) muss immer ein abschließendes `#endif` kommen, um den Block der Anweisungen abzuschließen.

Es existieren folgende Anweisungen zur bedingten Kompilierung im Preprozessor:

`#ifdef`	Anweisungen werden ausgeführt, wenn Symbol definiert
`#ifndef`	Anweisungen werden ausgeführt, wenn Symbol nicht definiert
`#else`	Alternativer Anweisungsblock
`#if`	Anweisungen werden ausgeführt, wenn der folgende arithmetische Ausdruck wahr ist
`#elif`	Alternativer `#if`-Anweisungsblock

Es ist möglich, bedingte Anweisungsblöcke zu definieren, die nur dann kompiliert werden, wenn ein Symbol nicht definiert ist.

```
#ifdef FUNKTIONSAUFRUF
  func1(a,b);
  func2(a,b);
#endif
#ifndef FUNKTIONSAUFRUF
  func3(a,b);
  func4(a,b);
#endif
```

Hier werden `func1()` und `func2()` aufgerufen, wenn `FUNKTIONSAUFRUF` definiert wurde, alternativ werden die Anweisungen `func3()` und `func4()` ausgeführt, wenn das Symbol `FUNKTIONSAUFRUF` nicht definiert ist.

Das letzte Beispiel lässt sich mit der `#else`-Anweisung vereinfachen:

```
#ifdef FUNKTIONSAUFRUF
  func1(a,b);
  func2(a,b);
#else
  func3(a,b);
  func4(a,b);
#endif
```

Die Programmanweisungen hinter `#else` werden dann kompiliert, wenn die Anweisungen vor `#else` nicht vom Compiler ausgeführt werden.

Definiert man die Symbole zu numerischen Werten, besteht die Möglichkeit, abhängig von arithmetischen Ausdrücken zu kompilieren. Ein Anwendungsfall wären z. B. Abhängigkeiten von Versionsnummern.

```
#define VERSION 5

#if VERSION >= 4
  func1(x); // Funktionsaufruf ab Version 4
#endif
```

Man kann auch rechnen und boolesche Operationen durchführen:

```
#define Zaehler 2
#define DEBUG 1

#if Zaehler + 2 >= 8 && DEBUG == 1
  func4(Zaehler); // wenn Zaehler gross genug und kein Debug
#endif
```

Um tiefe Schachtelungen zu vermeiden existiert die `#elif`-Anweisung. Das Wort `elif` ist als Abkürzung für „else if“ zu verstehen. Der Befehl `#elif` ist ein `#if`-Befehl im alternativen (`#else`) Anweisungsteil. Das folgende Beispiel verdeutlicht dies:

```
#define Version 5

#if VERSION > 1
  func1(a);
#else
#if VERSION > 2
  func2(a);
#else
#if VERSION > 3
  func3(a);
#else
  func(4);
#endif
#endif
#endif
```

Dies ist nur schwer verständlich und lässt sich mit `#elif` eleganter schreiben.

```
#define Version 5

#if VERSION > 1
  func1(a);
#elif VERSION > 2
```

```
  func2(a);
#elif VERSION > 3
  func3(a);
#else
  func(4);
#endif
```

Folgende Operatoren werden in arithmetischen Ausdrücken von `#if` und `#elif` verstanden:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo
&	binäres Und
|	binäres Oder
^	Binäres exclusives Oder

&&	logisches Und
||	logisches Oder
>	größer
>=	größer gleich
<	kleiner
<=	kleiner gleich
==	gleich
!=	ungleich

16.3 Einfügen von Dateien

Der Preprozessor-Befehl `#include name` fügt Textdateien an der Stelle ein, wo die `#include`-Anweisung im Quelltext steht.

Datei „func.cc“

```
void func1(int a, int b)
{
return(a+b);
}
```

Hauptprogramm

```
#include “func.cc“

void main(void)
{
  int x;

  x=func1(1,2);
}
```

Das Einfügen von Dateien wird da angewendet, wo mehrere Projekte die gleichen Definitionen oder Funktionen besitzen. Eleganter ist es eigentlich, die gemeinsame Datei in das Verzeichnis „Libraries“ zu kopieren und sie in den Bibliotheksoptionen zu aktivieren.

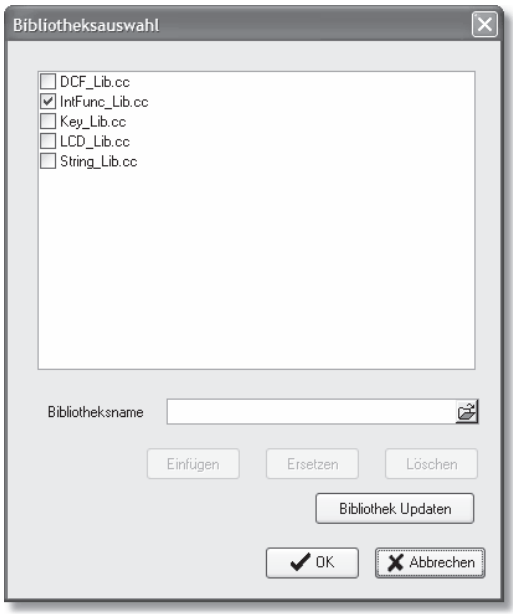


Abb. 16.2: Bibliotheksoptionen

16.4 Preprozessor-Makros

Der Preprozessor besitzt neben der einfachen Variante der Textersetzung auch einen komplexeren Mechanismus, dem man Textargumente übergeben kann. Das Makro

```
#define add(a,b,c) i= a b c;

void main(void)
{
  add(2,+,5);
}
```

führt eine Textersetzung aus. Der Parameter `a` wird durch `2` ersetzt, Parameter `b` durch `+`, und Parameter `c` wird mit `5` ausgetauscht. Der Quelltext, den dann der Compiler sieht, ist folgender:

```
void main(void)
{
  i= 5 + 2;
}
```

Die Mächtigkeit der Makros erkennt man daran, dass man komplette Funktionen als Makros definieren kann:

```
#define myfunc(name,op) int name(int a, int b)\
{\
  return(a op b);\
}
```

```
// hier werden die Funktionen func1(), func2() und func3()
// generiert
myfunc(func1,+)
myfunc(func2,*)
myfunc(func3,/)

void main(void)
{
  Msg_WriteInt(func1(5,7));
  Msg_WriteInt(func2(5,7));
  Msg_WriteInt(func3(15,3));
}
```

Die umgedrehten Schrägstriche („Backslashes“) am Ende der Zeile zeigen dem Preprozessor, dass die Makrodefinition bis zur nächsten Zeile geht. Dadurch lassen sich auch längere Textpassagen, die über mehrere Zeilen gehen, als Makro definieren. In diesem Beispiel wird die Funktion `func2()` mit dem Makroaufruf `myfunc(func2,+)` im Quelltext erstellt.

Tipp: Mit Makros lassen sich generische Funktionen programmieren, die dann z. B. für verschiedene Variablentypen definiert werden. Dies hat Ähnlichkeit zu den Templates in C++, es findet aber keine syntaktische Prüfung statt.

Als letztes Beispiel zu den Preprozessor-Makros eine Funktion, die in einem Array nach einem Wert sucht, aber nicht auf den Variablentyp des Arrays festgelegt ist:

```
#define ARRAY_LEN 99
#define search(typ) int array_search(typ v,typ a[])\
{\
  int i;\
  for(i=0;i<ARRAY_LEN;i++)\
  {\
    if(v==a[i])\
    {\
      return(i);\
    }\
  }\
  return(-1);\
}

int mlist[99];

// hier wird die Funktion array_search generiert
search(int) // erst hier wird der Typ “int“ definiert

void main(void)
{
  int indx;

  indx=array_search(22,mlist);
}
```

16.5 Vordefinierte Symbole

Durch die C-Control-Pro-IDE werden bestimmte Preprozessor-Symbole vorbelegt.

Symbol	Bedeutung
MEGA32	Kompilierung für Mega32
MEGA128	Kompilierung für Mega128
__DEBUG__	Debug Code wird erzeugt
__MAPFILE__	Eine Speicherlayout-Datei wird geschrieben

Die Symbole MEGA32 und MEGA128 werden dann definiert, wenn in den Projektoptionen das Mega32-Modul oder das Mega128-Modul als Zielplattform eingestellt wurde. Es ist dadurch möglich, Programme zu schreiben, die auf dem Mega32 und dem Mega128 gleichermaßen kompilierbar sind. Der folgende Quelltext stammt aus der Datei „IntFunc_Lib.cc“ die im „Libraries“-Verzeichnis liegt:

```
#ifdef MEGA32
#define PORT_LED1 30 // LED1 Portnr bei Mega32
#define PORT_LED2 31 // LED1 Portnr bei Mega32
#endif

#ifdef MEGA128
// LED1=PortG.3, LED2=PortG.4
#define PORT_LED1 51 // LED1 Portnr bei Mega128
#define PORT_LED2 52 // LED2 Portnr bei Mega128
#endif
```

Die Funktion LED_Loop() läuft unabhängig davon, ob der Mega32-Chip oder der Mega128-Chip als Plattform ausgesucht wird.

```
// Ein- und Ausschalten der LED1
//
void LED_Loop(int delay_val)
{
    Port_WriteBit(PORT_LED1,PORT_ON); // LED1 einschalten
    AbsDelay(delay_val); // variable Verzögerung
    Port_WriteBit(PORT_LED1,PORT_OFF); // LED1 ausschalten
    AbsDelay(delay_val); // variable Verzögerung
}
```

Auch ist Programmtext denkbar, der daran erinnert, den Debug-Code auszuschalten.

```
#ifdef __DEBUG__
  Text=“Debug Code ausschalten“;
  Msg_WriteText(text);
#endif
```

Weitere Symbole sind als Konstanten vordefiniert:

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

In diesen Konstanten werden Datum und Uhrzeit der Kompilierung festgehalten. Auch die Zeilennummer, der Dateiname und der Funktionsname vom Ort des entsprechenden Symbols sind gespeichert.

```
Void main(void)
{
  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
}
```

Hat eine Funktion mehrere Funktionsaufrufe von verschiedenen Orten, lässt sich der Aufrufsort mit Hilfe der Zeilennummer bestimmen:

```
int error;
char linenr[10];

void func(void)
{
  char text[60];

  if(error)
  {
  text="Fehler in Funktions func – Aufruf von Zeile:";
  Str_Copy(text, linenr,STR_APPEND);
  Msg_WriteText(text);
  }
}

void main(void)
{
  error=1; // dies ist nur eine Demo
  linenr=__LINE__; // setzen der Zeilennummer
  func();
}
```

16.6 Compiler-Anweisungen

Es existieren im CompactC- und BASIC-Compiler zusätzliche `#pragma`-Befehle, um Fehler und Warnungen auszugeben.

Befehl	Bedeutung
`#pragma Error "Text..."`	Textausgabe mit Syntax-Error
`#pragma Warning "Text..."`	Textausgabe mit Warnung
`#pragma Message "Text..."`	Nur Textausgabe

In Kombination mit vordefinierten Preprozessor-Symbolen gibt man Warnungen aus, wenn z. B. noch Debug-Code für das Programm erzeugt wird, oder wenn der Quelltext für einen speziellen Prozessortyp nicht geeignet ist.

Zwei Beispiele:

```
#ifndef MEGA128
#pragma Error "Das Programm funktioniert nur mit Mega128"
#endif

#ifdef __DEBUG__
#pragma Warning "nicht vergessen,den Debug Code ausschalten!"
#endif
```

In den Demoprogrammen, die mit Timer 3 arbeiten, wird diese Technik angewendet, da Timer 3 nur auf dem Mega128 vorhanden ist.

16.7 Mischen von BASIC und CompactC

Mit Hilfe von bisher undokumentierten Compiler-Befehlen ist es möglich, zwischen CompactC und BASIC in einer Datei umzuschalten. Die Anweisung `#pragma Language "CompactC" "null"` schaltet die Codeerzeugung auf CompactC, mit der Anweisung `#pragma Language "CCBasic" "null"` erkennt der Compiler die BASIC-Syntax.

Der folgende Sourcecode illustriert die Umschaltung:

```
#pragma Language "CompactC" "null"

// CompactC Code
int func2(int a, int b)
{
  return(a*b);
}

#pragma Language "CCBasic" "null"

' BASIC Code
Sub func1(a As Integer, b As Integer) As Integer
```

```
  Return a+b
End Sub


#pragma Language “CompactC“ “null“

// CompactC Code
int global;

#pragma Language “CCBasic“ “null“

’ BASIC Code
Sub main()
  Dim c As Integer

  c=func1(2,4)
  c=func2(2,4)
End Sub
```

Wichtig: Das Mischen von BASIC und CompactC in einer Textdatei kann zu Konfusion bei der Ausgabe von Zeilennummern führen. Syntaxfehler werden dann an der falschen Zeile angezeigt.

17 Interruptbehandlung

17.1 C-Control-Pro-Interrupts

Der Atmel Mega32- oder Mega128-Prozessor kann für verschiedene Ereignisse Hardware-Interrupts generieren. Die möglichen Interruptquellen sind:

1. Externer Interrupt
2. Analog-Comparator
3. Analog-Digital-Wandler (ADC)
4. Timer-Interrupt
5. Serielle Interrupts
6. I2C, SPI, EEPROM Interrupts

Eine genaue Beschreibung aller Interrupts und Ursachen, die einen Interrupt auslösen können, würde den Rahmen dieses Kapitels sprengen. Bitte lesen Sie dafür die offiziellen zum Mega32 und Mega128 veröffentlichten PDF-Dateien von Atmel. Die Homepage für Atmel AVR-CPUs ist unter *http://www.atmel.com/products/avr/* zu finden.

Die folgende Tabelle gibt eine Übersicht, welche Interrupts vom C-Control-Pro-System an den Anwender weitergegeben werden. Der Interruptname ist ein Preprozessor-Symbol, das in der Datei „IntFunc_Lib.cc“ definiert ist.

Nr.	**Interrupt Name**	**Beschreibung**
0	`INT_0`	externer Interrupt0
1	`INT_1`	externer Interrupt1
2	`INT_2`	externer Interrupt2
3	`INT_TIM1CAPT`	Timer1 Capture
4	`INT_TIM1CMPA`	Timer1 CompareA
5	`INT_TIM1CMPB`	Timer1 CompareB
6	`INT_TIM1OVF`	Timer1 Overflow
7	`INT_TIM0COMP`	Timer0 Compare
8	`INT_TIM0OVF`	Timer0 Overflow
9	`INT_ANA_COMP`	Analog Comparator

Nr.	Interrupt Name	Beschreibung
10	INT_ADC	ADC
11	INT_TIM2COMP	Timer2 Compare
12	INT_TIM2OVF	Timer2 Overflow
13	INT_3	externer Interrupt3 (nur Mega128)
14	INT_4	externer Interrupt4 (nur Mega128)
15	INT_5	externer Interrupt5 (nur Mega128)
16	INT_6	externer Interrupt6 (nur Mega128)
17	INT_7	externer Interrupt7 (nur Mega128)
18	INT_TIM3CAPT	Timer3 Capture (nur Mega128)
19	INT_TIM3CMPA	Timer3 CompareA (nur Mega128)
20	INT_TIM3CMPB	Timer3 CompareB (nur Mega128)
21	INT_TIM3CMPC	Timer3 CompareC (nur Mega128)
22	INT_TIM3OVF	Timer3 Overflow (nur Mega128)

Damit ein Interrupt eine Aktion auslösen kann, muss für ihn eine Interrupt-Service Routine definiert werden. Jede beliebige BASIC- oder CompactC-Funktion kann als Interruptfunktion dienen. Um dies zu spezifizieren, wird der Funktion `Irq_SetVect()` die Interruptnummer und der Name der aufzurufenden Interrupt-Service-Routine übergeben.

```
Syntax

void Irq_SetVect(byte irqnr,float vect); // CompactC
Sub Irq_SetVect(irqnr As Byte,vect As Single) ' BASIC

irqnr spezifiziert die Nummer des Interrupts
vect ist der Name der aufzurufenden Interrupt-Funktion
```

Am Ende der Service-Funktion muss signalisiert werden, dass der Interrupt bearbeitet wurde. Diese Signalisierung findet durch den Aufruf von `Irq_GetCount()` statt. `Irq_GetCount()` gibt als Rückgabewert an, wie oft der entsprechende Interrupt vor dem Aufruf von `Irq_GetCount()` getriggert wurde.

```
Syntax

byte Irq_GetCount(byte irqnr); // CompactC
Sub Irq_GetCount(irqnr As Byte) As Byte ' BASIC

irqnr ist die Nummer des zu quittierenden Interrupts
```

Für die Timerfunktionen, die einen Interrupt auslösen können, ist dies im offiziellen C-Control-Pro-Handbuch oder in der Hilfedatei gekennzeichnet. So löst die Funktion `Timer_T0Time()` nach einer bestimmten Zeit den Timer-0-Compare-Interrupt `INT_TIM0COMP` aus.

Im CompactC-Beispielprogramm wird nach 6,94 ms (100 x 69,44 µs) Port A.0 eingeschaltet. Vorteiler PS0_1024 steht für einen Wert von 69,44 µs.

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

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

void main(void)
{
  Port_DataDirBit(0,0);               // PortA.0 Ausgang
  Port_WriteBit(0,0);                 // PortA.0 Ausgang=0

  // Interrupt Service Routine definieren
  Irq_SetVect(INT_TIM0COMP,Timer0_ISR);
  Timer_T0Time(100,PS0_1024); //Zeit festlegen,Timer0 starten
}
```

Das Beispiel in BASIC:

```
Sub Timer0_ISR()

  Dim irqcnt As Integer

  Port_WriteBit(0,1)
  Timer_T0Stop() ' Timer0 anhalten
  Irq_GetCount(INT_TIM0COMP)
End Sub

Sub main()

  Port_DataDirBit(0,0) ' PortA.0 Ausgang
  Port_WriteBit(0,0) ' PortA.0 Ausgang=0

  ' Interrupt Service Routine definieren
  Irq_SetVect(INT_TIM0COMP,Timer0_ISR)
  Timer_T0Time(100,PS0_1024) ' Zeit festlegen,Timer0 starten
End Sub
```

17.2 Externe Interrupts

Der Atmel Mega32 verfügt über 3 externe Interrupts, bei dem Mega128-Modul sind es 8 mögliche externe Interruptquellen. Um zu erkennen, auf welchen Ports die externen Interrupts geschaltet sind, bitte im C-Control-Pro-Benutzerhandbuch die Pinzuordnungstabellen für den Mega32 und den Mega128 ansehen. Die Tabellen sind im Handbuch unter „Hardware → Mega32 → Pinzuordnung“, bzw. „Hardware → Mega128 → Pinzuordnung“ zu finden.

Um externe Interrupts verarbeiten zu können, müssen sie erst einmal freigeschaltet werden. Die Funktion `Ext_IntEnable()` schaltet den gewünschten Interrupt frei und wählt über den Mode-Parameter die Art und Weise, wenn getriggert werden soll.

```
Syntax

void Ext_IntEnable(byte IRQ,byte Mode); // CompactC
Sub Ext_IntEnable(IRQ As Byte,Mode As Byte) ' BASIC

IRQ ist die Nummer des 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

Mit dem Aufruf von Ext_IntDisable() werden externe Interrupts wieder gesperrt.

Syntax

void Ext_IntDisable(byte IRQ); // CompactC
Sub Ext_IntDisable(IRQ As Byte) ' BASIC

IRQ ist die Nummer des Interrupts Mega32 (0–2) bzw. Mega128 (0–7)
```

Wichtig: Bei den Funktionen `Ext_IntEnable()` und `Ext_IntDisable()` ist der Parameter `IRQ` anders nummeriert als in den Funktionen `Irq_SetVect()` und `Irq_GetCount()`! Gibt man bei `Irq_SetVect()` den Wert `INT_3 (13)` an, bekommt die Funktion `Ext_IntEnable()` den Wert `3` als IRQ-Parameter.

Hier ein Demoprogramm in CompactC, das den externen Interrupteingang 0 einschaltet, 10 Sekunden lang Interrupt-Requests zählt, und danach den externen Interrupt 0 wieder deaktiviert:

```
int irqcnt;

void Ext_ISR(void)
{
  irqcnt++; // Interrupts zählen
  Irq_GetCount(INT_0); // Interrupt Acknowledge
}

void main(void)
{
  // Interrupt Service Routine definieren
  Irq_SetVect(INT_0,Ext_ISR);
  Ext_IntEnable(0,2); // Interrupt aktivieren

  AbsDelay (10000); // 10 Sek. warten
  Ext_IntDisable(0); // Interrupt deaktivieren
}
```

Externe Interrupts in BASIC:

```
Dim irqcnt As Integer

Sub Ext_ISR()

  irqcnt=irqcnt+1 ' Interrupts zählen
  Irq_GetCount(INT_0) ' Interrupt Acknowledge
End Sub

Sub main()
```

```
  ' Interrupt Service Routine definieren
  Irq_SetVect(INT_0,Ext_ISR)
  Ext_IntEnable(0,2) ' Interrupt aktivieren
  AbsDelay (10000) ' 10 Sek. warten
  Ext_IntDisable(0) ' Interrupt deaktivieren
End Sub
```

Wichtig: INT0 ist auch der Eingang von SW1. Bei gedrücktem SW1 wird der serielle Bootloader aktiviert. Daher führt eine Verbindung von Timer0 mit INT0 zu dem Verhalten, dass der automatische Start der Applikation bei Einschalten der Betriebsspannung nicht funktioniert.

17.3 Interpreter-Interrupts im Detail

Die Verarbeitung von Interrupts läuft im Interpreter anders ab, als man es von Interruptroutinen in Assembler kennt. Daher kann es manchmal wichtig sein, den genauen Ablauf der Interruptverarbeitung im Bytecode-Interpreter zu verstehen:

- Schritt 1: Ein Hardware-Interrupt wird ausgelöst und springt in die entsprechende Assembler-Interrupt-Service-Routine. Dort wird der Zähler für diesen Interrupt um eins inkrementiert.
- Schritt 2: Der Interpreter arbeitet weiter den Bytecode oder die Bibliotheksfunktion ab, währenddessen der Interrupt ausgelöst wurde. Manche Bibliotheksfunktionen, wie z. B. die Sinusfunktion oder die serielle Eingabe, können sehr lange dauern.
- Schritt 3: Vor Ausführung des nächsten Bytecodes wird überprüft, ob ein Interrupt getriggert wurde. Dafür werden die Zähler der Interrupts als Indikator genommen. Die Überprüfung der Zähler findet in numerischer Reihenfolge statt, angefangen mit Interruptnummer Null.
- Schritt 4: Ist ein Zähler ungleich Null, wird als nächster Bytecode die definierte Interrupt-Service-Routine ausgeführt. Sollte keine Service-Routine definiert sein, wird der Interrupt ignoriert.
- Schritt 5: Der Aufruf von `Irq_GetCount()` in der Interrupt-Service-Routine setzt den Interruptzähler wieder auf Null.
- Schritt 6: Alle weiteren Interruptzähler werden geprüft, ob ein Interrupt stattgefunden hat.
- Schritt 7: Der Original-Programmablauf wird restauriert, und es wird hinter dem Bytecode weitergearbeitet, der ausgeführt wurde, als der Interrupt getriggert wurde.

18 Multithreading

Das C-Control-Pro-System unterstützt Multithreading, also die quasi-gleichzeitige Abarbeitung von mehreren Programmabläufen. Es werden der Hauptthread (das Hauptprogramm, das in der Funktion `main()` gestartet wird) und bis zu 13 weitere Threads unterschieden. Der Hauptthread wird automatisch bei Programmbeginn gestartet und erhält all den Speicher, der nicht für die zusätzlichen Threads konfiguriert wurde.

18.1 Starten von Threads

Ähnlich wie eine Interrupt-Service-Routine (siehe Kapitel 17: „Interruptbehandlung“), kann jede Funktion als Startfunktion für einen neuen Thread dienen.

Neue Threads werden mit einem Aufruf von `Thread_Start()` generiert.

```
Syntax

void Thread_Start(byte thread,float func); // CompactC
 Sub Thread_Start(Byte thread As Byte,func As Single) 'BASIC

Parameter
thread (1–13) Nummer des Threads, der gestartet werden soll
func   Funktionsname, in welcher der neue Thread beginnt
```

Mit `Thread_Kill()` werden laufende Threads wieder gestoppt.

```
Syntax

void Thread_Kill(byte thread); // CompactC
Sub Thread_Kill(thread As Byte) 'BASIC

Parameter
thread (1–13) Nummer des Threads, der gestartet werden soll
```

Im folgenden Programm werden für das Betätigen von SW1 und SW2 die Strings „Taster 1“ bzw. „Taster 2“ als Debugmeldung ausgegeben. Der Taster 2 wird in einer Endlosschleife im Hauptprogramm überwacht (Thread 0), den Taster 1 überprüft ein eigener Thread, der in Funktion `thread1` startet.

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

void thread1(void)
{
```

```
  while(true) // Endlosschleife
  {
    // wurde SW1 gedrückt wird “Taster 1“ ausgegeben
    if(!Port_ReadBit(PORT_SW1)) Msg_WriteText(str1);
  }
}

void main(void)
{
  str1=“Taster 1“; // Variablendeklaration
  str2=“Taster 2“; // Variablendeklaration

  Port_DataDirBit(PORT_SW1,PORT_IN); // SW1 auf Eingang
  Port_DataDirBit(PORT_SW2,PORT_IN); // SW2 auf Eingang
  Port_WriteBit(PORT_SW1,1); // PullUp für Eingang
  Port_WriteBit(PORT_SW2,1); // PullUp für Eingang
  Thread_Start(1,thread1); // Thread starten

  while(true) // Endlosschleife
  {
    // wurde SW2 gedrückt wird “Taster 2“ ausgegeben
    if(!Port_ReadBit(PORT_SW2)) Msg_WriteText(str2);
  }
}
```

Die BASIC-Variante:

```
Dim str1(12),str2(12) As Char ’ globale Variablendeklaration

Sub thread1()

    Do While True ’ Endlosschleife
        ’ wurde SW1 gedrückt wird “Taster 1“ ausgegeben
       If Not Port_ReadBit(PORT_SW1) Then
         Msg_WriteText(str1)
       End If
    End While
End Sub

Sub main()

    str1=“Taster 1“ ’ Variablendeklaration
    str2=“Taster 2“ ’ Variablendeklaration

    Port_DataDirBit(PORT_SW1,PORT_IN) ’ SW1 auf Eingang
    Port_DataDirBit(PORT_SW2,PORT_IN) ’ SW2 auf Eingang
    Port_WriteBit(PORT_SW1,1) ’ PullUp für Eingang
    Port_WriteBit(PORT_SW2,1) ’ PullUp für Eingang
    Thread_Start(1,thread1) ’ Thread starten

    Do While True ’ Endlosschleife
     ’ wurde SW2 gedrückt wird “Taster 2“ ausgegeben
      If Not Port_ReadBit(PORT_SW2) Then
        Msg_WriteText(str2)
      End If
    End While
End Sub
```

18.2 Konfiguration des Multithreadings

In der Dialogbox zur Threadkonfiguration lässt sich einstellen, ob ein Thread aktiv ist, wie viel Speicher er verbrauchen darf, und welche Zykluszeit er besitzt. Thread Nr. 0 ist der Hauptthread, der immer aktiv ist und den freien Restspeicher erhält.

Welche Stack-Größe benötigt ein Thread? Ein Thread verbraucht nur Platz für lokale Variablen und übergebene Parameter, in den Funktionen, die ein Thread durchläuft. Das anschließende Programm verdeutlicht dies:

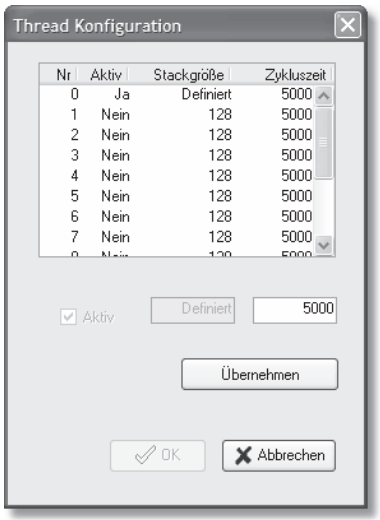


Abb. 18.1: Threadkonfiguration

```
int glob_a, glob_b; // 2 globale integer

void func1(void)
{
  int a,b,c; // 3 Integer Variablen
  float fa,fb; // 2 Float Variablen

  func2(5);
}

void func2(int x) // 1 Integer Variable
{
  char a,b,c; // 3 character Variablen
}

void func3(void)
{
  int a,b,c; // 3 Integer Variablen
}

void main(void)
{
  Thread_Start(1,func1); // Thread starten
  while(true); // Endlosschleife
}
```

Das Programm in BASIC:

```
Dim glob_a, glob_b As Integer ' 2 globale integer

Sub func1()
  Dim a,b,c As Integer ' 3 Integer Variablen
  Dim fa,fb As Single ' 2 Float Variablen

  func2(5)
End Sub

Sub func2(x As Integer) ' 1 Integer Variable
  Dim a,b,c As Char ' 3 Character Variablen
End Sub

Sub func3()
  Dim a,b,c As Integer ' 3 Integer Variablen
End Sub

Sub main()
  Thread_Start(1,func1) ' Thread starten
  Do While True ' Endlosschleife
  End While
End Sub
```

Möchte man den Speicherverbrauch des Thread 1 berechnen, sieht man, dass der Thread in Funktion `func1()` gestartet wird, und danach `func2()` aufruft. Es zählen nur die lokalen Variablen und Parameterübergaben. In `func1()` sind dies drei Integer-Variablen (= 3*2 Bytes) plus zwei Floating-Point-Variablen (= 2*4 Bytes). Dies sind dann 6+8 = 14 Bytes in der Funktion `func1()`. In `func2()` sind 3 Character-Variablen definiert, und es wird ein Integer übergeben. Dies entspricht einer Nutzung von 5 Bytes. In diesem Beispielprogramm muss daher der Speicherverbrauch von Thread 1 auf mindestens 19 gesetzt werden (in der C-Control-Pro-IDE erzwingt die Dialog-Eingabe aber einen minimalen Wert von 32 für jeden Thread).

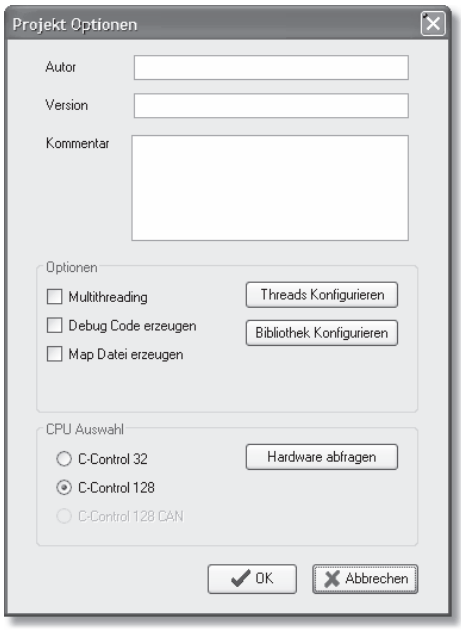


Abb. 18.2: Projekt-Optionen

Um bei komplexeren Programmen einfacher die Nutzung lokaler Variablen bestimmen zu können, existiert in den Projekt-Optionen die Möglichkeit, eine Map-Datei zu erzeugen.

Die Map-Datei zeigt dann direkt die Namen und Länge der lokalen Variablen einer Funktion sowie deren Gesamtlänge an.

```
Globale Variablen Laenge Position (RAM Anfang)
-------------------------------------------------------------
glob_a 2 2
glob_b 2 4
Gesamtlaenge: 4 bytes

Lokale Variablen Laenge Position (Stackrelativ)
-------------------------------------------------------------
Funktion func1()
a 2 12
b 2 10
c 2 8
fa 4 4
fb 4 0
Gesamtlaenge: 14 bytes

Funktion func2()
x 2 3
a 1 2
b 1 1
c 1 0
Gesamtlaenge: 5 bytes

Funktion func3()
a 2 4
b 2 2
c 2 0
Gesamtlaenge: 6 bytes

Funktion main()
Gesamtlaenge: 0 bytes
```

Tipp: Vorsicht bei Threads in rekursiven Funktionen! Deren Stack-Verbrauch ist schwierig abzuschätzen und direkt abhängig von der Anzahl der Rekursions-Ebenen.

Für die Überwachung des Speicherverbrauchs lässt sich auch die Funktion `Thread_MemFree()` einsetzen. Sie gibt den freien Speicher für den eigenen Thread zurück. Im folgenden Programm ruft sich eine rekursive Funktion 10-mal selbst auf. Bei der Ausgabe des freien Speicherplatzes sieht man, dass nach jedem Verlassen der rekursiven Funktion der von ihr genutzte Speicher zurückgegeben wird.

```
void recursion(int a)
{
  if(--a) recursion(a); // a dekrementieren und rekursion

  Msg_WriteWord(Thread_MemFree());
  Msg_WriteChar(13);
}

void thread(void)
{
  recursion(10);
}

void main(void)
{
  Thread_Start(1,thread); // Thread starten
  while(true);
}
```

Die Rekursion in BASIC:

```
Sub recursion(a As Integer)

  a=a-1
  If a>0 Then
    recursion(a) ' a dekrementieren und rekursion
  End If

  Msg_WriteWord(Thread_MemFree())
  Msg_WriteChar(13)
End Sub

Sub thread()
  recursion(10)
End Sub

Sub main()
  Thread_Start(1,thread) ' Thread starten
  Do While True
  End While
End Sub
```

Zusätzlich zum Speicherverbrauch kann die Zykluszeit eines Threads konfiguriert werden. Der Begriff Zykluszeit ist etwas missverständlich, die „Zykluszeit“ ist ein Zähler, der angibt, nach wie vielen Instruktionen (Zyklen) zum nächsten Thread gewechselt wird. Die Priorität eines Threads kann beeinflusst werden, in dem man ändert, wie viele Bytecodes ein Thread bis zum nächsten Threadwechsel ausführen darf. Je kleiner die Anzahl der Zyklen bis zum Wechsel, desto geringer die Priorität des Threads. Die Ausführungszeit eines Bytecode ist im Mittel 7–9 µsec. Bei manchen Bytecode-Befehlen dauert es jedoch länger, z. B. bei Floatingpoint-Operationen.

Wenn z. B. Thread 0 die Zykluszeit 5000 besitzt, und Thread 1 bekommt eine Zykluszeit von 2500, hat Thread 0 im Durchschnitt die doppelte Rechenzeit wie Thread 1 zur Verfügung.

Wichtig: Wenn man Prioritäten von Threads richtig abschätzen möchte, sollte man berücksichtigen, dass einzelne Bytecodes oder Interpreterfunktionen länger dauern können als andere. Da z. B. `Serial_Read()` wartet, bis ein Zeichen von der seriellen Schnittstelle ankommt, kann in Ausnahmefällen ein Zyklus sehr lange dauern.

18.3 Warten in Threads

Wenn ein Thread eine Zeitspanne warten soll, kann er dazu nicht die Funktion `AbsDelay()` einsetzen. Die Zeitspanne, die `AbsDelay()` wartet, ist sehr genau, aber die Genauigkeit wird dadurch erreicht, dass die CPU keine andere Tätigkeit durchführt.

```
Syntax

void AbsDelay(word ms); // CompactC
Sub AbsDelay(ms As Word) 'BASIC

Parameter
ms   Wartezeit in ms
```

Damit, während ein Thread wartet, der nächste Thread arbeiten kann, wurde die Funktion `Thread_Delay()` in die Bibliothek aufgenommen. Nach einem Aufruf von `Thread_Delay()` wird der aufrufende Thread aus der Liste der ausführbaren Threads herausgenommen. Zusätzlich wird ein Zähler gesetzt, der bestimmt, nach wie viel 10-ms-Schritten der Thread wieder aufgenommen werden darf.

```
Syntax

void Thread_Delay(word delay);
Sub Thread_Delay(delay As Word)

Parameter
delay   Anzahl von 10ms Ticks, die gewartet werden sollen
```

Ist der Zähler abgelaufen, kann der wartende Thread bei dem nächsten regulären Thread-Wechsel wieder weiterarbeiten.

Das Demoprogramm in CompactC gibt im Hauptprogramm und in einem Thread „gleichzeitig“ die Strings „Thread1“ und „Thread2“ aus. Da der Thread mittels `Thread_Delay()` doppelt so lange wartet (2000 ms) wie das Hauptprogramm (1000 ms), werden doppelt so viele Strings vom Hauptprogramm ausgegeben.

```
void thread1(void)
{
  while(true) // Endlosschleife
  {
    // "Thread2" wird ausgegeben
    Msg_WriteText(str2);
    // Danach ist der Thread für 2000ms "schlafend"
    Thread_Delay(200);
  }
}

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

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

  Thread_Start(1,thread1); // Thread Start

  while(true)              // Endlosschleife
  {
     // Der Thread ist für 1000ms "schlafend"
     Thread_Delay(100);
     // "Thread1" wird ausgegeben
     Msg_WriteText(str1);
   }
}
```

`Thread_Delay()` in BASIC:

```
Sub thread1()
  Do While True ' Endlosschleife
    ' "Thread2" wird ausgegeben
    Msg_WriteText(str2)
    ' Danach ist der Thread für 2000ms "schlafend"
    Thread_Delay(200)
  End While
End Sub

Dim str1(12),str2(12) As Char ' globale Variablendeklaration

Sub main()

  str1="Thread1" ' Variablendeklaration
  str2="Thread2" ' Variablendeklaration

  Thread_Start(1,thread1) ' Thread Start

  Do While True ' Endlosschleife
     ' Der Thread ist für 1000ms "schlafend"
     Thread_Delay(100)
     ' "Thread1" wird ausgegeben
     Msg_WriteText(str1)
   End While
End Sub
```

An dieser Stelle der Hinweis, dass die Zeitangabe für `Thread_Delay()` in Einheiten von 10 ms erfolgt. Ein Wert von 100 entspricht daher einer Zeit von 1000 ms.

Wichtig: Der Zähler, den `Thread_Delay()` benutzt, wird durch den 10-ms-Interrupt von Timer 2 getriggert. Sollte Timer 2 vom Anwender für andere Belange umprogrammiert werden, ist die Funktionalität von `Thread_Delay()` nicht gegeben!

18.4 Threads synchronisieren

Manchmal ist es wichtig, dass Threads nicht zur gleichen Zeit auf die gleiche Ressource zugreifen sollen (diese Ressource kann eine bestimmte Hardware sein, die über einen Port angesteuert wird) oder bestimmte Speicherbereiche dürfen nur von einem Thread zur selben Zeit angesprochen werden. Diese Art der Synchronisation wird über die Thread-Funktionen `Thread_Wait()`, `Thread_Signal()` und `Thread_Resume()` gesteuert.

```
Syntax

void Thread_Wait(byte signal); // CompactC
Sub Thread_Wait(signal As Byte) 'BASIC

void Thread_Signal(byte signal);  // CompactC
Sub Thread_Signal(signal As Byte) 'BASIC

Parameter
signal   Wert des Signals
```

Durch den Aufruf von `Thread_Wait()` wartet ein Thread auf ein Signal (Zahl zwischen 0 und 255). Wird mittels `Thread_Signal()` das Signal gegeben, wird der Thread fortgesetzt. Es können mehrere Threads auf dasselbe Signal warten.

```
Syntax

void Thread_Resume(byte thread); // CompactC
Sub Thread_Resume(thread As Byte) 'BASIC

Parameter
thread   (0–13) Nummer des Threads
```

Alternativ kann ein wartender Thread durch die Funktion `Thread_Resume()` aktiviert werden. Der Parameter von `Thread_Resume()` ist kein Signal, sondern die Nummer des zu erweckenden Threads.

Ein CompactC-Programm zur Thread-Synchronisation:

```
int wait_flag;

void thread1(void)
{
  if(wait_flag) Thread_Wait(7); // Auf Signal warten
  Msg_WriteText(str1); // "Thread 1" ausgeben
}

void thread2(void)
{
  if(wait_flag) Thread_Wait(8); // Auf Signal warten
  Msg_WriteText(str2); // "Thread 2" ausgeben
}

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

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

  wait_flag=true;

  Thread_Start(1,thread1); // Thread 1 starten
  Thread_Start(2,thread2); // Thread 2 starten

  Thread_Delay(200); // 2 Sek. warten

  Thread_Signal(7); // Signal Thread 1
  Thread_Delay(100); // 1 Sek. warten
  Thread_Resume(2); // Resume Thread 2
  Thread_Delay(50); // 0.5 Sek. warten
}
```

Das Programm in BASIC:

```
Dim wait_flag As Integer

Sub thread1()
  If wait_flag Then
    Thread_Wait(7) ' Auf Signal warten
  End If
  Msg_WriteText(str1) ' "Thread 1" ausgeben
End Sub
```

```
Sub thread2()
  If wait_flag Then
    Thread_Wait(8) ' Auf Signal warten
  End If
  Msg_WriteText(str2) ' "Thread 2" ausgeben
End Sub

Dim str1(12),str2(12) As Char ' globale Variablendeklaration

Sub main()
  str1="Thread 1"
  str2="Thread 2"

    wait_flag=True

    Thread_Start(1,thread1) ' Thread 1 starten
    Thread_Start(2,thread2) ' Thread 2 starten

    Thread_Delay(200) '2 Sek. warten

    Thread_Signal(7) ' Signal Thread 1
    Thread_Delay(100) ' 1 Sek. warten
    Thread_Resume(2) ' Resume Thread 2
    Thread_Delay(50) ' 0.5 Sek. warten
End Sub
```

Eine alternative Möglichkeit zu verhindern, dass mehr als ein Thread auf eine Ressource zugreift, ist die Möglichkeit den Threadwechsel komplett zu unterbinden. Schaltet man mit `Thread_Lock()` den Mechanismus zum Threadwechsel aus, läuft der gleiche Thread solange weiter, bis ein erneuter Aufruf von `Thread_Lock()` den Threadwechsel wieder reaktiviert.

```
Syntax

void Thread_Lock(byte lock);  // CompactC
Sub Thread_Lock(lock As Byte) 'BASIC

Parameter
lock   bei 1 Threadwechsel unterbunden, 0 wieder zugelassen
```

18.5 Multithreading im Detail

Wie läuft das Multithreading auf dem C-Control-Pro-System genau ab? In welchen einzelnen Schritten werden Threads abgearbeitet?

Ein Thread kann fünf verschiedene Zustände haben:

Zustand	Bedeutung
aktiv	Der Thread wird momentan abgearbeitet.
inaktiv	Der Thread kann nach einem Threadwechsel wieder aktiv werden.
schlafend	Ein `Thread_Delay()` wird ausgeführt.
wartend	Der Thread wartet auf ein Signal.
gesichert	Der Threadwechsel wurde mit `Thread_Lock()` deaktiviert.

Im Zustand „aktiv“ wird ein Thread in diesem Moment abgearbeitet. Es kann immer nur ein Thread gleichzeitig „aktiv“ sein. Findet ein Threadwechsel statt, sucht der Interpreter nach dem nächsten Thread aus der Liste, der den Zustand „inaktiv“ hat.

Die Arbeitsweise des Multithreadings im Detail:

- Schritt 1: Ist einer der Interruptzähler ungleich Null? Wenn ja, gehe zu Interruptverarbeitung. Siehe Kapitel 17: „Interruptbehandlung“.
- Schritt 2: Ist der Thread im Zustand „gesichert“? Wenn ja, dann führe den nächsten Bytecode aus.
- Schritt 3: Ist der Zyklenzähler („Zykluszeit“) abgelaufen? Wenn ja, dann setze den Thread auf den Zustand „inaktiv“. Wenn nein, dann führe den nächsten Bytecode aus.
- Schritt 4: Gehe die Thread-Liste durch zum nächsten Thread mit dem Zustand „schlafend“ oder „inaktiv“.
- Schritt 5: Ist der Zustand „schlafend“, dann überprüfe, ob inzwischen die Anzahl der 10-ms-Ticks abgelaufen ist, die der Thread warten sollte. Ist die Wartezeit abgelaufen, dann setze den Zustand auf „aktiv“ und mache mit Schritt 3 weiter.
- Schritt 6: Ist der Zustand „inaktiv“, dann setze den Thread auf „aktiv“ und mache bei Schritt 3 weiter.

19 Anwendungen

19.1 Voltmeter

Der Mega32 und der Mega128 sind mit einem Analog-Digital-Wandler ausgestattet. Über diesen können analoge Spannungen in digitale Signale umgewandelt werden.

$$Anzeigewert = \frac{1024 \times anliegende\ Spannung}{Referenzspannung}$$

Mit Hilfe dieses Wandlers kann man z. B. ein Voltmeter programmieren. Im nachfolgenden Programm wurden hierfür zwei ADC-Kanäle verwendet. Auf diese Weise ist es möglich zwei Spannungen gleichzeitig zu überwachen. Ohne weitere externe Beschaltungen können Spannungen von 0 bis 5 V gemessen werden. Voraussetzung ist hierfür, dass die Referenzspannung (ADC_VREF_VCC) verwendet wird.

Stückliste:

Menge	Art.-Nr.	Artikel	oder Art.-Nr.	oder Artikel
1	198206	Mega32	198219	Mega128
1	198245	Board Mega32	198258	Board Mega128

Voltmeter – Quellcode CBasic

Das Programm kann sowohl für den Mega32 als auch für den Mega128 verwendet werden. Es muss nur darauf geachtet werden, dass die ADC-Ports an verschiedenen Stellen der Application-Boards liegen. Die Zuweisung erfolgt automatisch.

```
' 2 Kanal Voltmeter
' Eingang ADC0 und ADC1
' Mega32: PA0 (PortA.0), PA1 (PortA.1)
' Mega128: PF0 (PortF.0), PF1 (PortF.1
' erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

' Die an den ADC-Ports 0 und 1 anliegende Spannung wird in eine digitale Zahl
' zwischen 0 und 1023 umgewandelt. Der diesem Wert entsprechende Spannungswert
' wird auf dem LCD Display angezeigt. Bei einem Überlauf wird " HHHH V" ange-
' zeigt.

Dim wert1, wert2 As Single         ' Deklaration der globalen
Dim zeile1(10) As Char             ' Variablen
Dim zeile2(10) As Char
```

```
'-------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
   LCD_Init()                     ' Display wird initialisiert
   LCD_ClearLCD()                 ' Display wird gelöscht
   LCD_CursorOff()                ' Cursor wird ausgeschaltet

   zeile1=" 0.00"                 ' default Wert für Zeile1
   zeile2=" 0.00"                 ' default Wert für Zeile2
   Do While (True)                ' Endlosschleife wird gestartet
      ADC_Set(ADC_VREF_VCC,0)     ' Port 0 des A/D Wandlers wird initialisiert
                                  ' Vcc (5V) ist Referenzspannung
      wert1 = ADC_Read()*5.0/1024.0' Der analoge Zahlenwert des ADC Portes 0
                                  ' wird in die Variable "wert1" gespeichert
                                  ' "5.0/1024.0" dient zur Umrechnung in den
                                  ' entsprechenden Spannungswert
      ADC_Set(ADC_VREF_VCC,1)     ' Port 1 des A/D Wandlers wird initialisiert
                                  ' Vcc (5V) ist Referenzspannung
      wert2 = ADC_Read()*5.0/1024.0' Der analoge Zahlenwert des ADC Portes 1
                                  ' wird in die Variable "wert1" gespeichert
                                  ' "5.0/1024.0" dient zur Umrechnung in den
                                  ' entsprechenden Spannungswert
      Ausgabe()                   ' Durch diesen Funktionsaufruf werden die
                                  ' Messwerte auf dem Display ausgegeben.
      AbsDelay(500)               ' 500ms Verzögerung bis zur nächsten Messung
   End While
End Sub
```

```
' Displayausgabe des 2 Kanal Voltmeters
' erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

'-------------------------------------------------------------------------------
' Displayausgabe
'
Sub Ausgabe()
Dim einheit(3) As Char            ' Deklaration der globalen
  einheit = " V"                  ' Festlegung der angezeigten Einheit
      If wert1>4.99 Then          ' Bei zu großen Messwerten an ADC0 wird
            zeile1=" HHHH"        ' " HHHH" in Zeile1 angezeigt.
      Else
            Str_WriteFloat(wert1,2,zeile1,1)    ' Die Floatvariable "wert1" wird in
      End If                                    ' einen Text mit 2 Dezimalstellen
                                                ' und einem Offset von 1 umgewandelt
      If wert2>4.99 Then          ' Bei zu großen Messwerten an ADC1 wird
            zeile2=" HHHH"        ' " HHHH" in Zeile2 angezeigt.
      Else
            Str_WriteFloat(wert2,2,zeile2,1)    ' Die Floatvariable "wert2" wird in
      End If                                    ' einen Text mit 2 Dezimalstellen
                                                ' und einem Offset von 1 umgewandelt
      LCD_CursorPos(0x00)         ' Der Cursor wird an den Anfang der ersten
                                  ' Zeile gesetzt.
      LCD_WriteText(zeile1)       ' Variable "zeile1" wird ausgegeben
      LCD_WriteText(einheit)      ' Einheit wird ausgegeben
      LCD_CursorPos(0x40)         ' Der Cursor wird an den Anfang der zweiten
                                  ' Zeile gesetzt.
      LCD_WriteText(zeile2)       ' Variable "zeile1" wird ausgegeben
      LCD_WriteText(einheit)      ' Einheit wird ausgegeben
End Sub
```

Voltmeter – Quellcode CC

Das Programm kann sowohl für den Mega32 als auch für den Mega128 verwendet werden. Es muss nur darauf geachtet werden, dass die ADC-Ports an verschiedenen Stellen der Application-Boards liegen. Die Zuweisung erfolgt automatisch.

```
// 2 Kanal Voltmeter
// Eingang ADC0 und ADC1
// Mega32: PA0 (PortA.0), PA1 (PortA.1)
// Mega128: PF0 (PortF.0), PF1 (PortF.1)
// erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

// Die an den ADC-Ports 0 und 1 anliegende Spannung wird in eine digitale Zahl
// zwischen 0 und 1023 umgewandelt. Der diesem Wert entsprechende Spannungswert
// wird auf dem LCD Display angezeigt. Bei einem Überlauf wird " HHHH V" ange-
// zeigt.

float wert1, wert2;                 // Deklaration der globalen
char zeile1[10];                    // Variablen
char zeile2[10];

//-------------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
   LCD_Init();              // Display wird initialisiert
   LCD_ClearLCD();          // Display wird gelöscht
   LCD_CursorOff();         // Cursor wird ausgeschaltet

   zeile1=" 0.00";          // default Wert für Zeile1
   zeile2=" 0.00";          // default Wert für Zeile2
   while (true)             // Endlosschleife wird gestartet
   {
      ADC_Set(ADC_VREF_VCC,0);     // Port 0 des A/D Wandlers wird initialisiert
                                   // Vcc (5V) ist Referenzspannung
      wert1 = ADC_Read()*5.0/1024.0;      // Der analoge Zahlenwert des ADC Portes 0
                                          // wird in die Variable "wert1" gespeichert
                                          // "5.0/1024.0" dient zur Umrechnung in den
                                          // entsprechenden Spannungswert
      ADC_Set(ADC_VREF_VCC,1);     // Port 1 des A/D Wandlers wird initialisiert
                                          // Vcc (5V) ist Referenzspannung
      wert2 = ADC_Read()*5.0/1024.0;      // Der analoge Zahlenwert des ADC Portes 1
                                          // wird in die Variable "wert1" gespeichert
                                          // "5.0/1024.0" dient zur Umrechnung in den
                                          // entsprechenden Spannungswert
      Ausgabe();            // Durch diesen Funktionsaufruf werden die
                            // Messwerte auf dem Display ausgegeben.
      AbsDelay(500);        // 500ms Verzögerung bis zur nächsten Messung
   }
}
```

```
// Displayausgabe des 2 Kanal Voltmeters
// erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

//-------------------------------------------------------------------------------
// Displayausgabe
//
void Ausgabe(void)
{
  char einheit[3];          // Deklaration der globalen
  einheit = " V";           // Festlegung der angezeigten Einheit
       if (wert1>4.99)      // Bei zu großen Messwerten an ADC0 wird
      {
```

```
            zeile1=" HHHH";             // " HHHH" in Zeile1 angezeigt.
      }

      else
      {
            Str_WriteFloat(wert1,2,zeile1,1);   // Die Floatvariable "wert1" wird in
                                                // einen Text mit 2 Dezimalstellen
      }                                   // und einem Offset von 1 umgewandelt
       if (wert2>4.99)                    // Bei zu großen Messwerten an ADC1 wird
      {
            zeile2=" HHHH";             // " HHHH" in Zeile2 angezeigt.
      }
      else
      {
            Str_WriteFloat(wert2,2,zeile2,1);   // Die Floatvariable "wert2" wird in
                                                // einen Text mit 2 Dezimalstellen
      }                             // und einem Offset von 1 umgewandelt
      LCD_CursorPos(0x00);          // Der Cursor wird an den Anfang der ersten
                                    // Zeile gesetzt.
      LCD_WriteText(zeile1);        // Variable "zeile1" wird ausgegeben
      LCD_WriteText(einheit);       // Einheit wird ausgegeben
      LCD_CursorPos(0x40);          // Der Cursor wird an den Anfang der zweiten
                                    // Zeile gesetzt.
      LCD_WriteText(zeile2);        // Variable "zeile1" wird ausgegeben
      LCD_WriteText(einheit);       // Einheit wird ausgegeben
}
```

19.2 Heizungssteuerung mit NTC-Sensoren

Der Mega128 ist mit einem Analog-Digital-Wandler ausgestattet. Über diesen können analoge Spannungen in digitale Signale umgewandelt werden. Wie bereits in Kapitel 12.3.2 beschrieben, besteht die Möglichkeit, auf diese Weise Temperaturen zu messen. Bei dem nun folgenden Beispiel wurden NTC-Sensoren als Temperaturfühler in den zu heizenden Räumen installiert. Als Beschaltung wurde der unter Abb. 12.15 beschriebene Spannungsteiler verwendet. Diese Messwerte werden jeweils über einen ADC-Port verarbeitet. Des Weiteren wird das unter Kapitel 12.1 beschriebe DCF-Modul zur Ermittlung der genauen Zeit verwendet, die als Zeitbasis für die Nachtabsenkung der Temperatur benötigt wird. Auf dem im Kapitel 12.2 beschriebenen 4x20-LCD-Display werden die Schaltzustände und die aktuelle Uhrzeit angezeigt. Das in Kapitel 12.4 beschriebene Relais-Modul dient zur Ausgabe der Schaltzustände. Die Aktualisierung der Uhrzeit erfolgt sekundengenau. Die Schaltzustände auf dem Display und bei den Relais werden jede Minute aktualisiert.

Verbinden Sie bitte das DCF-Modul mit dem Port F.0 und die Sensoren mit den Ports F.1, F.2 und F.3 (siehe Abb. 12.15). Das Relais-Modul muss mit den Ports A.0, A.1, A.2 und A.7 verbunden werden.

Dieses Beispiel ist für drei Sensoren mit drei Ventilausgängen und einem Pumpenausgang ausgelegt. Eine Erweiterung ist nur durch die Anzahl der ADC-Ports beschränkt. Eine Steuerung mit sieben NTC-Sensoren ist also durchaus realisierbar.

Stückliste:

Menge	Art.-Nr.	Artikel:
1	198219	Mega128
1	198258	Board Mega128
1	187275	LCD-Display 4x20
1	641138	DCF-Modul
3	182800	NTC-Sensor
3	408280	10-kOhm-Widerstand
1	198836	Relais-Modul

Heizungssteuerung 1 – Quellcode CBasic

Das Programm kann nur für den Mega128 verwendet werden. Es muss zusätzlich darauf geachtet werden, dass das SRAM des Application-Boards mit Jumper J7 abgeschaltet wird. Des Weiteren ist ein 4x20-Display erforderlich. Es ist empfehlenswert die einzelnen Programmteile modular aufzubauen. Dadurch wird das Projekt übersichtlicher und einzelne Teile können leichter exportiert werden. So können die einzelnen Teile z. B. folgendermaßen lauten: Heizungssteuerung, DCF-Uhr, Display, Temperaturabfrage, Relaissteuerung.

```
' Heizungssteuerung
' Eingang: DCF PortF.0, ADC1 PortF.1, ADC2 PortF.2, ADC3 PortF.3
' Ausgang: PortA.0, PortA.1, PortA.2, PortA.7
' Das SRAM muss mit JP7 deaktiviert werden.
' erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc, DCF_Lib.cc

' Über die DCF Antenne wird die aktuelle Uhrzeit festgestellt. Diese dient als
' Zeitbasis für die Heizungssteuerung. Die Temperaturen werden in den einzelnen
' Räumen über NTC Fühler erfaßt und ausgewertet. Je nach eingestellter Soll-
' Temperatur wird jeder einzelne Raum auf die entsprechende Temperatur aufge-
' heizt. Während der Nachtabsenkung wird die Temperatur auf die Nachttemperatur
' erwärmt.

' globale Variablendeklaration
Dim zeile1(21), zeile2(21), zeile3(21) As Char    ' Arrays für die einzelnen Zeilen
Dim temp1, temp2, temp3 As Integer ' Variablen der Isttemperaturen
Dim solltemp1, solltemp2, solltemp3 As Integer    ' Variablen der Solltemperaturen
Dim relais1, relais2, relais3 As Integer          ' Variablen der Relaisausgänge
Dim nachtabs, nacht As Integer                    ' Variablen für die Nachtabsenkung
Dim nachtein, nachtaus As Integer                 ' Variablen für die Nachtabsenkung

'--------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
  solltemp1 = 645    ' Solltemperatur von Raum 1 (670 <=> ca. 21°C)
  solltemp2 = 645    ' Solltemperatur von Raum 2
  solltemp3 = 645    ' Solltemperatur von Raum 3
  nachtabs = 580     ' Nachttemperatur der Räume (580 <=> ca.17 °C)
  nachtein = 22      ' Die Nachtabsenkung beginnt um 22:00 Uhr und
  nachtaus = 6       ' endet um 6:00 Uhr.
```

```
      LCD_Init()            ' Display initialisieren
      LCD_ClearLCD()        ' Display löschen
      LCD_CursorOff()       ' Display Cursur ausschalten

      Port_DataDir(PortA,&HFF)     ' alle Ausgänge von PortA auf Ausgang
      DCFUhr()              ' Funktionsaufruf zur DCF Synchronisation und
                            ' der weiteren Display- und Relaisaktualisierung
      Temperatur()                ' Funktionsaufruf zur ersten Temperaturmessung
      Raumdisplay()         ' Funktionsaufruf zur ersten Displayausgabe
      Displaytext()         ' Funktionsaufruf zur Ausgabe der Displaymaske
      Relais()              ' Funktionsaufruf zur ersten Relaisausgabe
      Do While (True)       ' Endlosschleife
      End While
End Sub
```

DCF-Uhr

```
'-------------------------------------------------------------------------------
' Festlegung des Synchronisationszeitpunktes
'
Sub INT_10ms()
      Dim irqcnt As Integer
      RTC(&H01,&H15)                    ' DCF Update um 01:15
      DCF_PULS()                        ' DCF_MODE=1 Puls suchen
      DCF_SYNC()                        ' DCF_MODE=2 Synchronisation
      DCF_FRAME()                       ' DCF_MODE=3 Datenaufnahme
      irqcnt=Irq_GetCount(INT_TIM2COMP)   ' Interrupt Request Counter
End Sub

'-------------------------------------------------------------------------------
' Festlegung der Startzeit im unsynchronisiertem Zustand
'
Sub Time_Init()
      cnt1=0        ' cnt1 zählt im 10ms Takt
      Sekunde=0     ' Startzeit: 0 Sekunden
      Minute=0      ' Startzeit: 0 Minuten
      Stunde=12     ' Startzeit: 12 Stunden
End Sub

'-------------------------------------------------------------------------------
' Zeitabgleich
'
Sub RTC(U_Stunde As Byte, U_Minute As Byte)
      cnt1=cnt1+1                   ' 10ms Zähler um Eins erhöhen
      If cnt1=100 Then              ' Wenn diese Schleife 100 mal durchlaufen
                                    ' wurde ist eine Sekunde vergangen.
      DCF_Set()                     ' Display Buffer wird aktualisiert.
      DCF_Write()                   ' Jede Sekunde wird die Zeit ausgeben.
      Sekunde=Sekunde+1             ' Sekunden Zähler um Eins erhöhen
      If Sekunde=60 Then            ' Wenn diese Schleife 60 mal durchlaufen
                                    ' wurde ist eine Sekunde vergangen.
            Temperatur()            ' Jede Minute wird eine Temperaturmessung
            Raumdisplay()           ' durchgeführt und das Display und die
            Relais()                ' Relaisausgänge aktualisiert.
            Sekunde=0               ' Der Sekundenzähler wird auf 0 gesetzt.
            If Minute=U_Minute And Stunde=U_Stunde Then
                                    ' Zeitabgleich bei eingestellter Uhrzeit
                DCF_START()         ' DCF-Erfassung wird gestartet.
            End If
            Minute=Minute+1         ' Der Minutenzähler wird um Eins erhöht.
            If Minute=60 Then       ' Wenn diese Schleife 60 mal durchlaufen
                                    ' wurde ist eine Minute vergangen.
                  Minute=0          ' Der Minutenzähler wird auf 0 gesetzt.
                  Stunde=Stunde+1         ' Der Stundenzähler wird um Eins erhöht.
                  If Stunde=24 Then
                  Stunde=0
```

```
                  End If
             End If         ' Ist die Stunde 24 erreicht wird der
      End If                ' Stundenzähler auf 0 zurückgesetzt.
      cnt1=0                ' 10ms Zähler wird auf 0 gesetzt.
      End If
End Sub

'-------------------------------------------------------------------------------
' DCFUhr
'
Sub DCFUhr()
      Time_Init()                         ' Funktionsaufruf zur Festlegung
                                          ' der Startzeit.
      DCF_INIT()                          ' Initialisierung des DCF Betriebes.
      DCF_START()                         ' DCF-Erfassung wird gestartet.
      Irq_SetVect(INT_TIM2COMP,INT_10ms)  ' Interrupt Service Routine definieren
                                          ' Timer2 erzeugt einen 10ms Interrupt.
End Sub
```

Display

```
'-------------------------------------------------------------------------------
' Uhrzeit in den Display Buffer schreiben
'
Sub DCF_Set()
      Dim sep(2) As Char                  ' Array für den Doppelpunkt
      sep=":"                             ' Wertzuweisung zur Variable sep
      Str_WriteWord(Stunde,10,zeile2,0,2) ' Die Variablen Stunde, Minute und
      Str_Copy(zeile2,sep,STR_APPEND)     ' Sekunde werden zu einem String
      Str_WriteWord(Minute,10,zeile2,STR_APPEND,2)       ' zusammengestellt.
      Str_Copy(zeile2,sep,STR_APPEND)
      Str_WriteWord(Sekunde,10,zeile2,STR_APPEND,2)
End Sub

'-------------------------------------------------------------------------------
' Displayausgabe
'
Sub Displaytext()
      zeile1 ="Heizungssteuerung V1"      ' Der Text für die erste und für
      zeile3 ="R-1 R-2 R-3 Pumpe"  ' die dritte Zeile wird festgelegt
      LCD_CursorPos(&H00)          ' LCD Cursor positionieren
      LCD_WriteText(zeile1)        ' Die erste Zeile wird ausgegeben.
      LCD_CursorPos(&H14)          ' LCD Cursor positionieren
      LCD_WriteText(zeile3)        ' Die dritte Zeile wird ausgegeben.
End Sub

Sub DCF_Write()
      LCD_CursorPos(&H4c)          ' LCD Cursor positionieren
      LCD_WriteText(zeile2)        ' Der Uhrzeit-String wird in Zeile
End Sub                            ' zwei ausgegeben.

Sub Raumdisplay()
  Dim absenkung(10)As Char         ' Das Array für den Absenkungstext
                                   ' wird deklariert.
  LCD_CursorPos(&H55)              ' LCD Cursor positionieren
  If relais1=1 Then
                                   ' Wenn das Relais1 eingeschaltet
      LCD_WriteChar(&H45)          ' ist wird auf dem Display E bei
                                   ' R-1 angezeigt.

  Else
      LCD_WriteChar(&H41)          ' Wenn nicht wird A angezeigt.
  End If
  LCD_CursorPos(&H59)              ' LCD Cursor positionieren
  If relais2=1 Then
```

```
                                        ' Wenn das Relais2 eingeschaltet
    LCD_WriteChar(&H45)                 ' ist wird auf dem Display E bei
                                        ' R-2 angezeigt.
  Else
    LCD_WriteChar(&H41)                 ' Wenn nicht wird A angezeigt.
  End If
LCD_CursorPos(&H5d)                     ' LCD Cursor positionieren
If relais3=1 Then
                                        ' Wenn das Relais3 eingeschaltet
    LCD_WriteChar(&H45)                 ' ist wird auf dem Display E bei
                                        ' R-2 angezeigt.
Else
    LCD_WriteChar(&H41)                 ' Wenn nicht wird A angezeigt.
End If
LCD_CursorPos(&H65)                     ' LCD Cursor positionieren
If relais1 Or relais2 Or relais3 Then   ' Ist Relais1, Relais2 oder Relais3
                                        ' eingeschaltet wird im Display
    LCD_WriteChar(&H45)                 ' E bei Pumpe angezeigt.

Else
    LCD_WriteChar(&H41)                 ' Wenn nicht wird A angezeigt.
End If
If Stunde < nachtaus Or Stunde >= nachtein Then
                                        ' Liegt die aktuelle Stunde inner-
            LCD_CursorPos(&H40)         ' halb des Nachtabsenkungszeitraums
            absenkung =“ABSENKUNG“      ' wird ABSENKUNG ausgegeben.
                  LCD_WriteText(absenkung)
Else                                ' Liegt die aktuelle Stunde außer-
            LCD_CursorPos(&H40)         ' halb des Nachtabsenkungszeitraums
            Absenkung =“ “              ' wird nichts ausgegeben.
                  LCD_WriteText(absenkung)
End If
End Sub
```

Temperaturabfrage

```
'-------------------------------------------------------------------------------
' Temperatur der NTC’s wird ausgelesen
'
Sub Temperatur()
      If Stunde < nachtaus Or Stunde >= nachtein Then
                                        ' Liegt die aktuelle Stunde innerhalb des
            nacht = solltemp1-nachtabs  ' Nachabsenkungsbereichs wird der Absen-
                                        ' kungswert errechnet.
      Else
            nacht = 0                   ' Liegt sie außerhalb wird der Absenkungs-
      End If                            ' wert auf 0 gesetzt.

      ADC_Set(ADC_VREF_VCC,1)      ' ADC-Port1 wird zur A/D-Wandlung vorbereitet
      temp1 = ADC_Read()           ' Der ermittelte Wert wird in die Variable
      If temp1<solltemp1-nacht Then       ' temp1 geschrieben. Ist der Wert von temp1
                                          ' kleiner als der Sollwert abzüglich des
            relais1 = 1           ' Absenkungswertes, wird das Relais1 ein-
                                  ' geschaltet.
      Else

            relais1 = 0           ' Wenn nicht, wird das Relais1 ausgeschaltet.
      End If
      ADC_Set(ADC_VREF_VCC,2)      ' ADC-Port2 wird zur A/D-Wandlung vorbereitet
      temp2 = ADC_Read()           ' Der ermittelte Wert wird in die Variable
      If temp2<solltemp2-nacht Then       ' temp2 geschrieben. Ist der Wert von temp2
                                          ' kleiner als der Sollwert abzüglich des
            relais2 = 1                   ' Absenkungswertes, wird das Relais2 ein-
                                          ' geschaltet.
      Else
      relais2 = 0                 ' Wenn nicht, wird das Relais2 ausgeschaltet.
```

```
      End If
      ADC_Set(ADC_VREF_VCC,3)      ' ADC-Port3 wird zur A/D-Wandlung vorbereitet
      temp3 = ADC_Read()           ' Der ermittelte Wert wird in die Variable
      If temp3<solltemp3-nacht Then       ' temp3 geschrieben. Ist der Wert von temp3
                                          ' kleiner als der Sollwert abzüglich des
            relais3 = 1                   ' Absenkungswertes, wird das Relais3 ein-
                                          ' geschaltet.
      Else
            relais3 = 0           ' Wenn nicht, wird das Relais3 ausgeschaltet.
      End If
End Sub
```

Relaissteuerung

```
'-------------------------------------------------------------------------------
' Ansteuerung der Relais
'
Sub Relais()
      Port_WriteBit(0,relais1)     ' Port A.0 wird in Abhängigkeit von der
                                   ' Variablen relais1 ein- oder ausgeschaltet.
      Port_WriteBit(1,relais2)     ' Port A.1 wird in Abhängigkeit von der
                                   ' Variablen relais2 ein- oder ausgeschaltet.
      Port_WriteBit(2,relais3)     ' Port A.2 wird in Abhängigkeit von der
                                   ' Variablen relais3 ein- oder ausgeschaltet.
      If relais1 Or relais2 Or relais3 Then
                                   ' Sobald mindestens eines der Relais 1 bis
            Port_WriteBit(7,1)     ' 3 eingeschaltet ist wird auch Relais 8
                                   ' (Ausgang PortA.7) eingeschaltet.
      Else                         ' Ist kein Relais eingeschaltet wird auch
            Port_WriteBit(7,0)     ' Relais 8 ausgeschaltet.
      End If
End Sub
```

Heizungssteuerung 1 – Quellcode CC

Das Programm kann nur für den Mega128 verwendet werden. Es muss zusätzlich darauf geachtet werden, dass das SRAM des Application-Boards mit Jumper J7 abgeschaltet wird. Des Weiteren ist ein 4x20-Display erforderlich. Es ist empfehlenswert, die einzelnen Programmteile modular aufzubauen. Dadurch wird das Projekt übersichtlicher und einzelne Teile können leichter exportiert werden. So können die einzelnen Teile z. B. folgendermaßen lauten: Heizungssteuerung, DCF-Uhr, Display, Temperaturabfrage, Relaissteuerung.

```
// Heizungssteuerung
// Eingang: DCF PortF.0, ADC1 PortF.1, ADC2 PortF.2, ADC3 PortF.3
// Ausgang: PortA.0, PortA.1, PortA.2, PortA.7
// Das SRAM muss mit JP7 deaktiviert werden.
// erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc, DCF_Lib.cc

// Über die DCF Antenne wird die aktuelle Uhrzeit festgestellt. Diese dient als
// Zeitbasis für die Heizungssteuerung. Die Temperaturen werden in den einzelnen
// Räumen über NTC Fühler erfaßt und ausgewertet. Je nach eingestellter Soll-
// Temperatur wird jeder einzelne Raum auf die entsprechende Temperatur aufge-
// heizt. Während der Nachtabsenkung wird die Temperatur auf die Nachttemperatur
// erwärmt.

// globale Variablendeklaration
char zeile1[21], zeile2[21], zeile3[21];  // Arrays für die einzelnen Zeilen
int temp1, temp2, temp3;                  // Variablen der Isttemperaturen
int solltemp1, solltemp2, solltemp3;      // Variablen der Solltemperaturen
```

```
int relais1, relais2, relais3;     // Variablen der Relaisausgänge
int nachtabs, nacht;               // Variablen für die Nachtabsenkung
int nachtein, nachtaus;            // Variablen für die Nachtabsenkung

//---------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
  solltemp1 = 645;                 // Solltemperatur von Raum 1 (670 <=> ca. 21°C)
  solltemp2 = 645;                 // Solltemperatur von Raum 2
  solltemp3 = 645;                 // Solltemperatur von Raum 3
  nachtabs = 580;                  // Nachttemperatur der Räume (580 <=> ca.17 °C)
  nachtein = 22;                   // Die Nachtabsenkung beginnt um 22:00 Uhr und
  nachtaus = 6;                    // endet um 6:00 Uhr.

  LCD_Init();                      // Display initialisieren
  LCD_ClearLCD();                  // Display löschen
  LCD_CursorOff();                 // Display Cursur ausschalten

  Port_DataDir(PortA,0xFF);        // alle Ausgänge von PortA auf Ausgang
  DCFUhr();                        // Funktionsaufruf zur DCF Synchronisation und
                                   // der weiteren Display- und Relaisaktualisierung
  Temperatur();                    // Funktionsaufruf zur ersten Temperaturmessung
  Raumdisplay();                   // Funktionsaufruf zur ersten Displayausgabe
  Displaytext();                   // Funktionsaufruf zur Ausgabe der Displaymaske
  Relais();                        // Funktionsaufruf zur ersten Relaisausgabe
  while (true);                    // Endlosschleife
}
```

DCF-Uhr

```
//---------------------------------------------------------------------------
// Festlegung des Synchronisationszeitpunktes
//
void INT_10ms(void)
{
      int irqcnt;
      RTC(0x01,0x15);       // DCF Update um 01:15
      DCF_PULS();           // DCF_MODE=1 Puls suchen
      DCF_SYNC();           // DCF_MODE=2 Synchronisation
      DCF_FRAME();          // DCF_MODE=3 Datenaufnahme

      irqcnt=Irq_GetCount(INT_TIM2COMP);  // Interrupt Request Counter
}

//---------------------------------------------------------------------------
// Festlegung der Startzeit im unsynchronisiertem Zustand
//
void Time_Init(void)
{
      cnt1=0;       // cnt1 zählt im 10ms Takt
      Sekunde=0;    // Startzeit: 0 Sekunden
      Minute=0;     // Startzeit: 0 Minuten
      Stunde=12;    // Startzeit: 12 Stunden
}

//---------------------------------------------------------------------------
// Zeitabgleich
//
void RTC(byte U_Stunde, byte U_Minute)
{
      cnt1++;                     // 10ms Zähler um Eins erhöhen
      if (cnt1==100)              // Wenn diese Schleife 100 mal durchlaufen
      {                           // wurde ist eine Sekunde vergangen.
            DCF_Set();            // Display Buffer wird aktualisiert.
```

```
            DCF_Write();          // Jede Sekunde wird die Zeit ausgeben.
            Sekunde++;            // Sekunden Zähler um Eins erhöhen
            if (Sekunde==60)      // Wenn diese Schleife 60 mal durchlaufen
            {                     // wurde ist eine Sekunde vergangen.
                  Temperatur();   // Jede Minute wird eine Temperaturmessung
                  Raumdisplay(); // durchgeführt und das Display und die
                  Relais();       // Relaisausgänge aktualisiert.
                  Sekunde=0;      // Der Sekundenzähler wird auf 0 gesetzt.
                  if (Minute==U_Minute && Stunde==U_Stunde)
                  {               // Zeitabgleich bei eingestellter Uhrzeit
                        DCF_START();   // DCF-Erfassung wird gestartet.
                  }
                  Minute++;       // Der Minutenzähler wird um Eins erhöht.
                  if (Minute==60)       // Wenn diese Schleife 60 mal durchlaufen
                  {      // wurde ist eine Minute vergangen.
                  Minute=0;       // Der Minutenzähler wird auf 0 gesetzt.
                  Stunde++;       // Der Stundenzähler wird um Eins erhöht.
                  if (Stunde==24) Stunde=0;
                  }               // Ist die Stunde 24 erreicht wird der
            }                     // Stundenzähler auf 0 zurückgesetzt.
            cnt1=0;               // 10ms Zähler wird auf 0 gesetzt.
      }
}

//---------------------------------------------------------------------------
// DCFUhr
//
void DCFUhr(void)
{
      Time_Init();                        // Funktionsaufruf zur Festlegung
                                          // der Startzeit.
      DCF_INIT();                         // Initialisierung des DCF Betriebes.
      DCF_START();                        // DCF-Erfassung wird gestartet.
      Irq_SetVect(INT_TIM2COMP,INT_10ms); // Interrupt Service Routine definieren
                                          // Timer2 erzeugt einen 10ms Interrupt.
}


//---------------------------------------------------------------------------
// Uhrzeit in den Display Buffer schreiben
//
void DCF_Set(void)
{
      char sep[2];                              // Array für den Doppelpunkt
      sep=":";                                  // Wertzuweisung zur Variable sep
      Str_WriteWord(Stunde,10,zeile2,0,2);      // Die Variablen Stunde, Minute und
      Str_Copy(zeile2,sep,STR_APPEND);          // Sekunde werden zu einem String
      Str_WriteWord(Minute,10,zeile2,STR_APPEND,2);     // zusammengestellt.
      Str_Copy(zeile2,sep,STR_APPEND);
      Str_WriteWord(Sekunde,10,zeile2,STR_APPEND,2);
}
```

Display

```
//---------------------------------------------------------------------------
// Displayausgabe
//
void Displaytext(void)
{
  zeile1 ="Heizungssteuerung V1";  // Der Text für die erste und für
  zeile3 ="R-1 R-2 R-3 Pumpe";     // die dritte Zeile wird festgelegt
  LCD_CursorPos(0x00);             // LCD Cursor positionieren
  LCD_WriteText(zeile1);           // Die erste Zeile wird ausgegeben.
  LCD_CursorPos(0x14);             // LCD Cursor positionieren
  LCD_WriteText(zeile3);           // Die dritte Zeile wird ausgegeben.
}
```

```
void DCF_Write(void)
{
      LCD_CursorPos(0x4c);                // LCD Cursor positionieren
      LCD_WriteText(zeile2);              // Der Uhrzeit-String wird in Zeile
}                                         // zwei ausgegeben.

void Raumdisplay(void)
{
  char absenkung[10];                     // Das Array für den Absenkungstext
                                          // wird deklariert.
  LCD_CursorPos(0x55);                    // LCD Cursor positionieren
  if (relais1==1)
  {                                       // Wenn das Relais1 eingeschaltet
        LCD_WriteChar(0x45);              // ist wird auf dem Display E bei
  }                                       // R-1 angezeigt.
  else LCD_WriteChar(0x41);               // Wenn nicht wird A angezeigt.

  LCD_CursorPos(0x59);                    // LCD Cursor positionieren
  if (relais2==1)
  {                                       // Wenn das Relais2 eingeschaltet
        LCD_WriteChar(0x45);              // ist wird auf dem Display E bei
  }                                       // R-2 angezeigt.
  else LCD_WriteChar(0x41);               // Wenn nicht wird A angezeigt.

  LCD_CursorPos(0x5d);                    // LCD Cursor positionieren
  if (relais3==1)
  {                                       // Wenn das Relais3 eingeschaltet
        LCD_WriteChar(0x45);              // ist wird auf dem Display E bei
  }                                       // R-2 angezeigt.
  else LCD_WriteChar(0x41);               // Wenn nicht wird A angezeigt.


  LCD_CursorPos(0x65);                    // LCD Cursor positionieren
  if (relais1|relais2|relais3)            // Ist Relais1, Relais2 oder Relais3
  {                                       // eingeschaltet wird im Display
        LCD_WriteChar(0x45);              // E bei Pumpe angezeigt.
  }
  else LCD_WriteChar(0x41);               // Wenn nicht wird A angezeigt.

  if (Stunde < nachtaus | Stunde >= nachtein)
  {                                       // Liegt die aktuelle Stunde inner-
      LCD_CursorPos(0x40);                // halb des Nachtabsenkungszeitraums
      absenkung =“ABSENKUNG“;             // wird ABSENKUNG ausgegeben.
      LCD_WriteText(absenkung);
  }
  else
  {                                       // Liegt die aktuelle Stunde außer-
      LCD_CursorPos(0x40);                // halb des Nachtabsenkungszeitraums
      absenkung =“ “;                     // wird nichts ausgegeben.
            LCD_WriteText(absenkung);
  }
}
```

Temperaturabfrage

```
//---------------------------------------------------------------------------
// Temperatur der NTC’s wird ausgelesen
//
void Temperatur(void)
{
      if (Stunde < nachtaus | Stunde >= nachtein)
      {                                   // Liegt die aktuelle Stunde innerhalb des
            nacht = solltemp1-nachtabs;   // Nachabsenkungsbereichs wird der Absen-
      }                                   // kungswert errechnet.
      else nacht = 0;                     // Liegt sie außerhalb wird der Absenkungs-
                                          // wert auf 0 gesetzt.
```

```
  ADC_Set(ADC_VREF_VCC,1);           // ADC-Port1 wird zur A/D-Wandlung vorbereitet
      temp1 = ADC_Read();            // Der ermittelte Wert wird in die Variable
      if (temp1<solltemp1-nacht)     // temp1 geschrieben. Ist der Wert von temp1
      {                              // kleiner als der Sollwert abzüglich des
            relais1 = 1;             // Absenkungswertes, wird das Relais1 ein-
      }                              // geschaltet.
      else relais1 = 0;              // Wenn nicht, wird das Relais1 ausgeschaltet.

  ADC_Set(ADC_VREF_VCC,2);           // ADC-Port2 wird zur A/D-Wandlung vorbereitet
      temp2 = ADC_Read();            // Der ermittelte Wert wird in die Variable
      if (temp2<solltemp2-nacht)     // temp2 geschrieben. Ist der Wert von temp2
      {                              // kleiner als der Sollwert abzüglich des
            relais2 = 1;             // Absenkungswertes, wird das Relais2 ein-
      }                              // geschaltet.
      else relais2 = 0;              // Wenn nicht, wird das Relais2 ausgeschaltet.

      ADC_Set(ADC_VREF_VCC,3);       // ADC-Port3 wird zur A/D-Wandlung vorbereitet
      temp3 = ADC_Read();            // Der ermittelte Wert wird in die Variable
      if (temp3<solltemp3-nacht)     // temp3 geschrieben. Ist der Wert von temp3
      {                              // kleiner als der Sollwert abzüglich des
            relais3 = 1;             // Absenkungswertes, wird das Relais3 ein-
      }                              // geschaltet.
      else relais3 = 0;              // Wenn nicht, wird das Relais3 ausgeschaltet.
}
```

Relaissteuerung

```
//---------------------------------------------------------------------------
// Ansteuerung der Relais
//
void Relais(void)
{
  Port_WriteBit(0,relais1);          // Port A.0 wird in Abhängigkeit von der
                                     // Variablen relais1 ein- oder ausgeschaltet.
  Port_WriteBit(1,relais2);          // Port A.1 wird in Abhängigkeit von der
                                     // Variablen relais2 ein- oder ausgeschaltet.
  Port_WriteBit(2,relais3);          // Port A.2 wird in Abhängigkeit von der
                                     // Variablen relais3 ein- oder ausgeschaltet.
  if (relais1|relais2|relais3)       // Sobald mindestens eines der Relais 1 bis
  {                                  // 3 eingeschaltet ist wird auch Relais 8
    Port_WriteBit(7,1);              // (Ausgang PortA.7) eingeschaltet.
  }                                  // Ist kein Relais eingeschaltet wird auch
  else Port_WriteBit(7,0);           // Relais 8 ausgeschaltet.
}
```

19.3 Heizungssteuerung mit Raumtemperaturregler

Durch die vorhandenen Digitalports können logische Signale leicht erfasst, verarbeitet und wieder ausgegeben werden. In dem nun folgenden Programm werden die digitalen Schaltzustände der Raumtemperaturregler verarbeitet und an eine entsprechend angeschlossene Relaiskarte ausgegeben. In dem Beispiel werden drei Temperaturregler verwendet, die drei Ventilantriebe und eine Heizungspumpe ansteuern. Die Schaltzustände werden zusätzlich auf dem LC-Display angezeigt. Natürlich können Sie dieses Programm entsprechend Ihren Bedürfnissen erweitern.

Das Programm ist im vorliegenden Zustand für den Mega32 ausgelegt. Verbinden Sie hiefür die drei Raumtemperaturregler mit den Ports B.0, B.1, B.2. Die Regler müssen gegen 0 V schalten. Das Relais-Modul muss mit den Ports C.2, C.3, C.4 und C.5 verbunden werden. Wenn Sie den Mega128 verwenden wollen, müssen Sie nur die Ports entsprechend ändern, da die momentan benutzten beim Mega128 teilweise für wichtige Übertragungsfunktionen verwendet werden. Unter Umständen ist es erforderlich, dass Sie mit dem Anschluss der Stromversorgung der Relaisplatine bis nach dem Programmübertragen warten müssen. Durch die Autostartfunktion des Mega32 wird das Programm dann automatisch gestartet.

Stückliste:

Menge	Art.-Nr.	Artikel
1	198206	Mega32
1	198245	Board Mega32
3	615900	Raumtemperaturregler
1	198836	Relais-Modul

Heizungssteuerung 2 – Quellcode CBasic

Das Programm ist für den Mega32 ausgelegt. Wenn Sie den Mega128 verwenden wollen, müssen Sie die verwendeten Ports entsprechend ändern. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
' Heizungssteuerung
' Eingang: PortB.0, PortB.1, PortB.2
' Ausgang: PortC.2, PortC.3, PortC.4, PortC.5
' erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

' Die Temperatur wird in den einzelnen Räumen über Rautemperaturregler er-
' mittelt. Die jeweiligen Schaltzustandsänderungen bewirkten eine Pegeländerung
' an den Eingängen. Über die Ausgänge werden mit Hilfe einer Relaisplatine
' Ventile angesteuert die den Heißwasserfluss regeln. Sobald ein Ventil an-
' gesprochen wurde wird gleichzeitig die Heizungspumpe eingeschaltet. Zur
' optischen Kontrolle werden alle Schaltzustände auf dem LC-Display angezeigt.

' globale Variablendeklaration
Dim eingang(4) As Byte

'--------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
      LCD_Init()            ' Display initialisieren
      LCD_ClearLCD()        ' Display löschen
      LCD_CursorOff()       ' Display Cursur ausschalten

      Port_DataDir(PortC,&H3C)     ' C.2, C.3, C.4 und C.5 als Ausgang deklariert
      Port_DataDir(PortB,&H00)     ' Port B komplett als Eingang deklariet
      Port_Write(PortB,&H07)       ' Interne Pullup Widerstände der Ports B.0, B.1,
                                   ' und B.2 aktiviert.
```

```
      Do While (True)
                  Eingaenge()           ' Funktionsaufruf zur Zustandsfeststellung
                                        ' der Eingänge.
                  Display()             ' Funktionsaufruf zur Displayausgabe.
                  Ausgaenge()           ' Funktionsaufruf zur Relaisausgabe
      End While
End Sub
```

Eingänge

```
'-------------------------------------------------------------------------------
' Auslesen der digitalen Eingänge
'
Sub Eingaenge()
      Dim i As Integer
      For i=1 To 3                      ' Schleife zur Abfrage der einzelnen
                                        ' Eingangs-Ports.
            eingang(i)=Port_ReadBit(7+i) ' Der an den Eingängen vorhandene
      Next                              ' Schaltungszustand wird in die ent-
End Sub                                 ' sprechende Variable eingang gespeichert.
```

Display

```
'-------------------------------------------------------------------------------
' Displayausgaben erzeugen
'
Sub Display()
      Dim i As Integer
      Dim zeile1(9) As Char            ' Deklaration des Arrays für die erste Zeile.
      zeile1 ="VVV P"                   ' Der Text für Zeile 1 wird zugewiesen.
      LCD_CursorPos(&H00)               ' LCD Cursor positionieren
      LCD_WriteText(zeile1)             ' Die erste Zeile wird ausgegeben.
      For i=1 To 3                     ' Schleife zur Ausgabe der einzelnen Zustände
                                        ' auf dem Display.
            LCD_CursorPos(&H3F+i)       ' LCD Cursor positionieren
            If eingang(i)=1 Then
                                        ' Liegt am Eingang ein logische 1 an (5V)
                  LCD_WriteChar(&H45)   ' wird auf dem Display E ausgegeben.
            Else
                  LCD_WriteChar(&H41)   ' Wenn nicht wird A angezeigt.
            End If
      Next
      LCD_CursorPos(&H47)               ' LCD Cursor positionieren
      If eingang(1)=1 Or eingang(2)=1 Or eingang(3)=1 Then
                                        ' Ist eines der Ventil-Ports auf E
            LCD_WriteChar(&H45)         ' wird auch das Pumpen-Port auf E gesetzt.
      Else
            LCD_WriteChar(&H41)         ' Wenn nicht, wird es ausgeschaltet.
      End If
End Sub
```

Ausgänge

```
'-------------------------------------------------------------------------------
' Aktualisierung der Relais
'
Sub Ausgaenge()
      Dim i As Integer
      For i=1 To 3                      ' Schleife zur Ausgabe der einzelnen
                                        ' Portzustände in Abhängigkeit von
          Port_WriteBit(17+i,eingang(i))  ' der entsprechenden Variable
      Next                              ' eingang.
      If eingang(1) Or eingang(2) Or eingang(3)=1 Then
                                        ' Ist eines der Ventil-Ports aktiv
```

```
            Port_WriteBit(21,1)   ' wird auch das Pumpen-Port ein-
      Else                        ' geschaltet.
            Port_WriteBit(21,0)   ' Wenn nicht, wird es ausgeschaltet.
      End If
End Sub
```

Heizungssteuerung 2 – Quellcode CC

Das Programm ist für den Mega32 ausgelegt. Wenn Sie den Mega128 verwenden wollen, müssen Sie die verwendeten Ports entsprechend ändern. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
// Heizungssteuerung
// Eingang: PortB.0, PortB.1, PortB.2
// Ausgang: PortC.2, PortC.3, PortC.4, PortC.5
// erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

// Die Temperatur wird in den einzelnen Räumen über Rautemperaturregler er-
// mittelt. Die jeweiligen Schaltzustandsänderungen bewirkten eine Pegeländerung
// an den Eingängen. Über die Ausgänge werden mit Hilfe einer Relaisplatine
// Ventile angesteuert die den Heißwasserfluss regeln. Sobald ein Ventil an-
// gesprochen wurde wird gleichzeitig die Heizungspumpe eingeschaltet. Zur
// optischen Kontrolle werden alle Schaltzustände auf dem LC-Display angezeigt.

// globale Variablendeklaration
byte eingang[4];

//--------------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
  LCD_Init();                     // Display initialisieren
  LCD_ClearLCD();                 // Display löschen
  LCD_CursorOff();                // Display Cursur ausschalten

  Port_DataDir(PortC,0x3C);       // C.2, C.3, C.4 und C.5 als Ausgang deklariert
  Port_DataDir(PortB,0x00);       // Port B komplett als Eingang deklariet
  Port_Write(PortB,0x07);         // Interne Pullup Widerstände der Ports B.0, B.1,
                                  // und B.2 aktiviert.
  while (true)
  {
      Eingaenge();                // Funktionsaufruf zur Zustandsfeststellung
                                  // der Eingänge.
      Display();                  // Funktionsaufruf zur Displayausgabe.
      Ausgaenge();                // Funktionsaufruf zur Relaisausgabe
  }
}
```

Eingänge

```
//---------------------------------------------------------------------------
// Auslesen der digitalen Eingänge
//
void Eingaenge(void)
{
  int i;
  for (i=1;i<4;i++)               // Schleife zur Abfrage der einzelnen
  {                               // Eingangs-Ports.
     eingang[i]=Port_ReadBit(7+i); // Der an den Eingängen vorhandene
  }                               // Schaltungszustand wird in die ent-
}                                 // sprechende Variable eingang gespeichert.
```

Display

```
//---------------------------------------------------------------------------
// Displayausgaben erzeugen
//
void Display(void)
{
  char zeile1[9];                 // Deklaration des Arrays für die erste Zeile.
  zeile1 =“VVV P“;                // Der Text für Zeile 1 wird zugewiesen.
  LCD_CursorPos(0x00);            // LCD Cursor positionieren
  LCD_WriteText(zeile1);          // Die erste Zeile wird ausgegeben.
  int i;
  for (i=1;i<4;i++)               // Schleife zur Ausgabe der einzelnen Zustände
  {                               // auf dem Display.
        LCD_CursorPos(0x3F+i);    // LCD Cursor positionieren
        if (eingang[i]==1)
        {                         // Liegt am Eingang ein logische 1 an (5V)
             LCD_WriteChar(0x45); // wird auf dem Display E ausgegeben.
        }
        else LCD_WriteChar(0x41); // Wenn nicht wird A angezeigt.
  }

  LCD_CursorPos(0x47);            // LCD Cursor positionieren
  if (eingang[1]==1|eingang[2]==1|eingang[3]==1)
  {                               // Ist eines der Ventil-Ports auf E
        LCD_WriteChar(0x45);      // wird auch das Pumpen-Port auf E gesetzt.
  }
  else LCD_WriteChar(0x41);       // Wenn nicht, wird es ausgeschaltet.
}
```

Ausgänge

```
//---------------------------------------------------------------------------
// Aktualisierung der Relais
//
void Ausgaenge(void)
{
  int i;
  for (i=1;i<4;i++)                     // Schleife zur Ausgabe der einzelnen
  {                                     // Portzustände in Abhängigkeit von
        Port_WriteBit(17+i,eingang[i]); // der entsprechenden Variable
  }                                     // eingang.
  if (eingang[1]|eingang[2]|eingang[3]==1)// Ist eines der Ventil-Ports aktiv
  {                                     // wird auch das Pumpen-Port ein-
        Port_WriteBit(21,1);            // geschaltet.
  }
  else Port_WriteBit(21,0);             // Wenn nicht, wird es ausgeschaltet.
}
```

19.4 Temperaturschalter mit Sensorüberwachung

Beim Unterschreiten einer von Ihnen eingestellten Temperatur wird das am Ausgang angeschlossene Relais eingeschaltet. Die Schalttemperatur stellen Sie über ein Potentiometer stufenlos ein. Die LEDs LD1 und LD2 signalisieren Ihnen die Schaltzustände. Das Programm ist sowohl für den Mega32 als auch für den Mega128 unverändert verwendbar. Beim Mega32 muss das Potentiometer an Port A.0, der Sensor an Port A.1 und das Relais an Port B.2 angeschlossen werden. Beim Mega128 sind dies das Port F.0 für das Potentiometer, Port F.1 für den Sensor und Port F.2 für das Relais.

LD2 blinkt, wenn der Sensor unterbrochen oder kurzgeschlossen ist. LD1 leuchtet, wenn der Relais-Ausgang eingeschaltet ist.

Stückliste:

Menge	Art.-Nr.	Artikel	oder Art.-Nr.	oder Artikel
1	198206	Mega32	198219	Mega128
1	198245	Board Mega32	198258	Board Mega128
1	182800	NTC-Sensor		
1	408280	10-kOhm-Widerstand		
1	198836	Relais-Modul		
1	445673	22-kOhm-Poti lin		

Wie Sie den NTC-Sensor anschließen müssen, können Sie in Abb.12.15 erkennen. Das Potentiometer muss ähnlich angeschlossen werden: Ein Ende des Widerstandes auf 5 V, das andere Ende auf GND (0 V). Der Schleifer muss nun noch mit dem ADC-Port A.0 bzw. F.0 verbunden werden. Die Relaisplatine verbinden Sie bitte mit dem Port B.2 bzw. F.2 (siehe Kapitel 12.4).

Temperaturschalter – Quellcode CBasic
Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden.

```
' Temperaturschalter mit Sensorüberwachung
' Mega32
' Eingang: PortA.0, PortA.1
' Ausgang: Port B.2
' Mega128
' Eingang: PortF.0, PortF.1
' Ausgang: Port F.2
' erforderliche Library: IntFunc_Lib.cc

' An dem PortA.0 bzw. PortF.0 muss das Potentiometer angeschlossen werden.
' Über dieses kann die gewünschte Schaltschwelle stufenlos eingestellt werden.
```

```
' Der NTC-Sensor wird an dem PortA.1 bzw. PortF.1 angeschlossen. Dieser dient
' zur Ermittlung der aktuellen Temperatur. Über den Ausgang PortB.2 bzw. F.2
' wird ein entsprechend angeschlossenes Relais geschaltet. LD1 leuchtet wenn
' das Relais geschaltet hat und LD2 blinkt wenn der Sensor unterbrochen oder
' kurzgeschlossen ist.


#ifdef MEGA32                           ' Wird ein Mega32 verwendet wird der
#define aPort 10                        ' Ausgabeport auf diese Weise deklariert.
#endif

#ifdef MEGA128                          ' Wird ein Mega128 verwendet wird der
#define aPort 42                        ' Ausgabeport auf diese Weise deklariert.
#endif

' globale Variablendeklaration
Dim poti, sensor, sensor2 As Integer

'-------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
      Port_DataDirBit(aPort,PORT_OUT)     ' Port wird auf Ausgabe vorbereitet.
      Port_DataDirBit(PORT_LED1,PORT_OUT) ' LED1 Port wird auf Ausgabe gesetzt.
      Port_DataDirBit(PORT_LED2,PORT_OUT) ' LED2 Port wird auf Ausgabe gesetzt.
      Port_WriteBit(PORT_LED1,PORT_OFF)   ' LED1 wird ausgeschalet.
      Port_WriteBit(PORT_LED2,PORT_OFF)   ' LED2 wird ausgeschaltet.

      Do While (True)
            ADC_Set(ADC_VREF_VCC,ADC0)    ' Die Referenzspannung und der Messport
                                          ' werden ausgewählt.
            poti = ADC_Read()             ' Die der anliegenden Spannung ent-
                                          ' sprechenden Zahl wird in die Variable
                                          ' poti gespeichert.
            ADC_Set(ADC_VREF_VCC,ADC1)    ' Die Referenzspannung und der Messport
                                          ' werden ausgewählt.
            sensor2 = sensor              ' Alter Sensorwert wird zwischengespeichert.
            sensor = ADC_Read()           ' Die der anliegenden Spannung ent-
                                          ' sprechenden Zahl wird in die Variable
                                          ' sensor gespeichert.
            ' Liegt eine Unterbrechung des Sensors vor oder ist er kurzgeschlossen
            ' blinkt LED2.
            If sensor<400 Or sensor>900 Or sensor>sensor2+2 Or sensor<sensor2-2 Then
                Port_WriteBit(PORT_LED2,PORT_ON)        ' LED2 wird eingeschaltet.
                AbsDelay(100)                           ' 100ms leuchtet die LED
                Port_WriteBit(PORT_LED2,PORT_OFF)       ' LED2 wird ausgeschaltet.
                AbsDelay(100)                   ' 100ms ist die LED ausgeschaltet
                Port_WriteBit(aPort,PORT_ON)            ' LED2 wird ausgeschaltet.
                Port_WriteBit(PORT_LED1,PORT_OFF)       ' Relais wird ausgeschaltet.
            Else
                If sensor < poti Then                   ' Ist der Messwert kleiner
                                                        ' als der eigestellte Wert
                   Port_WriteBit(aPort,PORT_OFF)        ' wird das Relais und die
                   Port_WriteBit(PORT_LED1,PORT_ON)     ' LED1 eingeschaltet
                End If
                If sensor > poti+5 Then                 ' ist der Messwert kleiner
                                                        ' als der eigestellte Wert+5
                   Port_WriteBit(aPort,PORT_ON)         ' wird das Relais und LED1
                   Port_WriteBit(PORT_LED1,PORT_OFF)    ' ausgeschaltet. Je größer
                End If                  ' der addierte Wert ist, desto
            End If                      ' größer wird die Hysterese.
      End While
End Sub
```

Temperaturschalter – Quellcode CC

Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden.

```
// Temperaturschalter mit Sensorüberwachung
// Mega32
// Eingang: PortA.0, PortA.1
// Ausgang: Port B.2
// Mega128
// Eingang: PortF.0, PortF.1
// Ausgang: Port F.2
// erforderliche Library: IntFunc_Lib.cc

// An dem PortA.0 bzw. PortF.0 muss das Potentiometer angeschlossen werden.
// Über dieses kann die gewünschte Schaltschwelle stufenlos eingestellt werden.
// Der NTC-Sensor wird an dem PortA.1 bzw. PortF.1 angeschlossen. Dieser dient
// zur Ermittlung der aktuellen Temperatur. Über den Ausgang PortB.2 bzw. F.2
// wird ein entsprechend angeschlossenes Relais geschaltet. LD1 leuchtet, wenn
// das Relais geschaltet hat und LD2 blinkt, wenn der Sensor unterbrochen oder
// kurzgeschlossen ist.

#ifdef MEGA32                           // Wird ein Mega32 verwendet wird der
#define aPort 10                        // Ausgabeport auf diese Weise deklariert.
#endif

#ifdef MEGA128                          // Wird ein Mega128 verwendet wird der
#define aPort 42                        // Ausgabeport auf diese Weise deklariert.
#endif

// globale Variablendeklaration
int poti, sensor, sensor2;

//---------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
      Port_DataDirBit(aPort,PORT_OUT);    // Port wird auf Ausgabe vorbereitet.
      Port_DataDirBit(PORT_LED1,PORT_OUT);// LED1 Port wird auf Ausgabe gesetzt.
      Port_DataDirBit(PORT_LED2,PORT_OUT);// LED2 Port wird auf Ausgabe gesetzt.
      Port_WriteBit(PORT_LED1,PORT_OFF);  // LED1 wird ausgeschalet.
      Port_WriteBit(PORT_LED2,PORT_OFF);  // LED2 wird ausgeschaltet.

      while (true)
      {
            ADC_Set(ADC_VREF_VCC,ADC0);  // Die Referenzspannung und der Messport
                                         // werden ausgewählt.
            poti = ADC_Read();           // Die der anliegenden Spannung ent-
                                         // sprechenden Zahl wird in die Variable
                                         // poti gespeichert.
            ADC_Set(ADC_VREF_VCC,ADC1);  // Die Referenzspannung und der Messport
                                         // werden ausgewählt.
            sensor2 = sensor;
            sensor = ADC_Read();         // Die der anliegenden Spannung ent-
                                         // sprechenden Zahl wird in die Variable
                                         // sensor gespeichert.


            // Liegt eine Unterbrechung des Sensors vor oder ist er kurzgeschlossen
                                         // blinkt LED2.
            if (sensor<400 | sensor>900 | sensor>sensor2+2 | sensor<sensor2-2)
            {
                  Port_WriteBit(PORT_LED2,PORT_ON);   // LED2 wird eingeschaltet.
                  AbsDelay(100);                      // 100ms leuchtet die LED
```

```
                    Port_WriteBit(PORT_LED2,PORT_OFF);  // LED2 wird ausgeschaltet.
                    AbsDelay(100);                 // 100ms ist die LED ausgeschaltet
                    Port_WriteBit(aPort,PORT_ON);       // LED2 wird ausgeschaltet.
                    Port_WriteBit(PORT_LED1,PORT_OFF);  // Relais wird ausgeschaltet.
             }
             else
             {
                    if (sensor < poti)                  // Ist der Messwert kleiner
                    {                                   // als der eigestellte Wert
                           Port_WriteBit(aPort,PORT_OFF); // wird das Relais und die
                           Port_WriteBit(PORT_LED1,PORT_ON); // LED1 eingeschaltet
                    }
                    if (sensor > poti+5)                // ist der Messwert kleiner
                    {                                   // als der eigestellte Wert+5
                           Port_WriteBit(aPort,PORT_ON); // wird das Relais und LED1
                           Port_WriteBit(PORT_LED1,PORT_OFF); // ausgeschaltet. Je
                    }                      // größer der addierte Wert ist,
             }                             // desto größer wird die Hysterese.
      }
}
```

19.5 Zwei-Kanal-Thermometer

Über zwei NTC-Sensoren wird die aktuelle Temperatur ermittelt. Durch den Anschluss an je ein A/D-Port des Mega128 wird der vorhandene Analogwert in einen entsprechenden digitalen Wert umgewandelt. Mit Hilfe der beim Mega128 implementierten mathematischen Funktionen wird anschließend der digitale Wert in eine entsprechende Temperatur umgerechnet und anschließend auf dem Display angezeigt. Da aus Speicherplatzgründen die mathematischen Funktionen nur beim Mega128 vorhanden sind, ist dieses Programm auch nur für diesen geeignet.

Stückliste:

Menge	Art.-Nr.:	Artikel
1	198219	Mega128
1	198258	Board Mega128
2	182800	NTC-Sensor
2	408280	10-kOhm-Widerstand

Wie Sie den NTC-Sensor anschließen müssen, können Sie in Abb.12.15 erkennen. Als Eingänge werden die Ports F.0 und F.1 verwendet. Da sowohl bei den NTC-Sensoren als auch bei den Widerständen des Spannungsteilers Bauteiltoleranzen vorhanden sind, kann die verwendete Formel nur als Anhaltspunkt dienen.

temp1 = (1.0/(1.0/1980.0*ln(tnom/sensor1)+(1.0/298.0)))-273.0

Treten bei Ihrer Anzeige des Temperaturwertes extreme Abweichungen auf, so können Sie natürlich die Formeln entsprechend abändern. Hilfreich sind hierfür die Da-

tenblätter Ihrer verwendeten NTC-Sensors. Um den A/D-Wert bei 25°C (`tnom`) zu ermitteln, erwärmen Sie den NTC-Sensor auf diese Temperatur. Starten Sie nun das Programm im Debug-Modus und lesen Sie den Wert der entsprechenden Sensorvariablen `sensor1` oder `sensor2` aus, indem Sie den Cursor nach Abarbeitung der entsprechenden Zeile `sensor1 = ADC-Read()` oder `sensor2 = ADC-Read()` auf die Variable `sensor1` oder `sensor2` stellen. Im nun erscheinenden Tooltip ist der momentane Wert der Variablen zu erkennen. Weisen Sie diese Zahl im Programm der Variablen `tnom` zu. (z. B. 700.0)

Zwei-Kanal-Thermometer – Quellcode CBasic

Das Programm kann wegen der verwendeten mathematischen Funktionen nur zusammen mit dem Mega128 verwendet werden.

```
' Zwei-Kanal-Thermometer
' Mega128
' Eingang: PortF.0, PortF.1
' erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

' An den Ports F.0, F.1 wird jeweils ein NTC-Fühler angeschlossen.
' Der ermittelte Wert wird in einen °C-Wert umgerechnet und auf dem Display
' angezeigt.

' globale Variablendeklaration
Dim sensor1, sensor2 As Single     ' Variablen für die Sensorwerte.
Dim temp1, temp2 As Single         ' Variablen für die errechneten Temperaturwerte.
Dim tnom As Single                 ' A/D-Wert bei 25°C.
Dim T1(9), T2(9) As Char           ' Textausgabe Arrays.

'--------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
      LCD_Init()                          ' Display initialisieren
      LCD_ClearLCD()                      ' Display löschen
      LCD_CursorOff()                     ' Display Cursur ausschalten

      tnom = 700.0                        ' Zuweisung des A/D-Wertes bei 25°C
      Do While (True)                     ' Endlosschleife wird gestartet.

            ADC_Set(ADC_VREF_VCC,ADC0)    ' Die Referenzspannung und der Messport
                                          ' werden ausgewählt.
            sensor1 = ADC_Read()          ' Die der anliegenden Spannung ent-
                                          ' sprechenden Zahl wird in die Variable
                                          ' sensor1 gespeichert.
            ADC_Set(ADC_VREF_VCC,ADC1)    ' Die Referenzspannung und der Messport
                                          ' werden ausgewählt.
            sensor2 = ADC_Read()          ' Die der anliegenden Spannung ent-
                                          ' sprechenden Zahl wird in die Variable
                                          ' sensor2 gespeichert.
            ' Mit diesen Formeln werden die A/D-Werte in °C-Werte umgerechnet
            temp1 = (1.0/(1.0/1980.0*ln(tnom/sensor1)+(1.0/298.0)))-273.0
            temp2 = (1.0/(1.0/1980.0*ln(tnom/sensor2)+(1.0/298.0)))-273.0


            Str_WriteFloat(temp1,1,T1,0) ' Die Floatvariablen werden nun in
            Str_WriteFloat(temp2,1,T2,0) ' Strings mit einer Dezimalstelle
                                          ' umgewandelt.
```

```
                    LCD_CursorPos(0x00)   ' LCD Cursor positionieren
                    LCD_WriteText(T1)     ' Die erste Zeile wird ausgegeben.
                    LCD_WriteChar(0x20)   ' Ein Leerzeichen wird geschrieben.
                    LCD_WriteChar(0x43)   ' Ein C wird ausgegeben.
                    LCD_CursorPos(0x40)   ' LCD Cursor positionieren
                    LCD_WriteText(T2)     ' Die zweite Zeile wird ausgegeben.
                    LCD_WriteChar(0x20)   ' Ein Leerzeichen wird geschrieben.
                    LCD_WriteChar(0x43)   ' Ein C wird ausgegeben.
                    AbsDelay(1000)        ' Jede Sekunde erfolgt eine Messung.
      End While
End Sub
```

Zwei-Kanal-Thermometer – Quellcode CC

Das Programm kann wegen der verwendeten mathematischen Funktionen nur zusammen mit dem Mega128 verwendet werden.

```
// Zwei-Kanal-Thermometer
// Mega128
// Eingang: PortF.0, PortF.1
// erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

// An den Ports F.0, F.1 wird jeweils ein NTC-Fühler angeschlossen.
// Der ermittelte Wert wird in einen °C-Wert umgerechnet und auf dem Display
// angezeigt.

// globale Variablendeklaration
float sensor1, sensor2;            // Variablen für die Sensorwerte.
float temp1, temp2;                // Variablen für die errechneten Temperaturwerte.
float tnom;                        // A/D-Wert bei 25°C.
char T1[9], T2[9];                 // Textausgabe Arrays.

//--------------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
      LCD_Init();                          // Display initialisieren
      LCD_ClearLCD();                      // Display löschen
      LCD_CursorOff();                     // Display Cursur ausschalten

      tnom = 700.0;                        // Zuweisung des A/D-Wertes bei 25°C
      while (true)                         // Endlosschleife wird gestartet.
      {
            ADC_Set(ADC_VREF_VCC,ADC0);  // Die Referenzspannung und der Messport
                                         // werden ausgewählt.
            sensor1 = ADC_Read();        // Die der anliegenden Spannung ent-
                                         // sprechenden Zahl wird in die Variable
                                         // sensor1 gespeichert.
            ADC_Set(ADC_VREF_VCC,ADC1);  // Die Referenzspannung und der Messport
                                         // werden ausgewählt.
            sensor2 = ADC_Read();        // Die der anliegenden Spannung ent-

                                         // sprechenden Zahl wird in die Variable
                                         // sensor2 gespeichert.

            // Mit diesen Formeln werden die A/D-Werte in °C-Werte umgerechnet
            temp1 = (1.0/(1.0/1980.0*ln(tnom/sensor1)+(1.0/298.0)))-273.0;
            temp2 = (1.0/(1.0/1980.0*ln(tnom/sensor2)+(1.0/298.0)))-273.0;

            Str_WriteFloat(temp1,1,T1,0);// Die Floatvariablen werden nun in
            Str_WriteFloat(temp2,1,T2,0);// Strings mit einer Dezimalstelle
                                         // umgewandelt.
```

```
            LCD_CursorPos(0x00);        // LCD Cursor positionieren
            LCD_WriteText(T1);          // Die erste Zeile wird ausgegeben.
            LCD_WriteChar(0x20);        // Ein Leerzeichen wird geschrieben.
            LCD_WriteChar(0x43);        // Ein C wird ausgegeben.
            LCD_CursorPos(0x40);        // LCD Cursor positionieren
            LCD_WriteText(T2);          // Die zweite Zeile wird ausgegeben.
            LCD_WriteChar(0x20);        // Ein Leerzeichen wird geschrieben.
            LCD_WriteChar(0x43);        // Ein C wird ausgegeben.
            AbsDelay(1000);             // Jede Sekunde erfolgt eine Messung.
      }
}
```

19.6 Temperaturdifferenzschalter

Es können zwei Temperaturen an verschiedenen Orten gemessen und miteinander verglichen werden. Das Relais wird in Abhängigkeit von der über das Potentiometer einstellbaren Temperaturdifferenz eingeschaltet. Ist die Temperatur am Sensor1 (Vorlauf) um die Temperatur des Sensors2 (Rücklauf) plus die Temperaturdifferenz (Poti) größer, wird das Relais für die Umwälzpumpe aktiviert. Als Einschaltkontrolle wird parallel zum Relais auch die LED1 eingeschaltet. Nach dem Unterschreiten der geforderten Temperatur wird das Relais und die LED wieder ausgeschaltet.

Stückliste:

Menge	Art.-Nr.	Artikel	oder Art.-Nr.	oder Artikel
1	198206	Mega32	198219	Mega128
1	198245	Board Mega32	198258	Board Mega128
2	182800	NTC-Sensor		
2	408280	10-kOhm-Widerstand		
1	198836	Relais-Modul		
1	445673	22-kOhm-Poti lin		

Wie Sie den NTC-Sensor anschließen müssen, können Sie in Abb.12.15 erkennen. Das Potentiometer muss ähnlich angeschlossen werden: Ein Ende des Widerstandes auf 5 V, das andere Ende auf GND (0 V). Der Schleifer muss nun noch mit dem ADC-Port A.3 bzw. F.3 verbunden werden. Die Relaisplatine verbinden Sie bitte mit dem Port B.2 bzw. F.3 (siehe Kapitel 12.4).

Temperaturdifferenzschalter – Quellcode CBasic
Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden.

```
' Temperaturdifferenzschalter
' Mega32
' Eingang: PortA.0, PortA.1, PortA.3
' Ausgang: Port B.2
' Mega128
' Eingang: PortF.0, PortF.1, PortF.2
' Ausgang: Port F.3
' erforderliche Library: IntFunc_Lib.cc

' Das Programm kann für den Mega32 und Mega128 verwendet werden.
' Über die A/D-Ports A.0 und A.1 bzw. F.0 und F.1 werden die zu vergleichenden
' Temperaturen eingelesen. Am Port A.3 bzw. F.3 wird ein Potentiometer ange-
' schlossen. Über diese kann die Temperaturdifferenz zwischen den beiden
' Fühlern eingestellt werden. Die Ausgabe des Schaltvorganges erfolgt über den
' Port B.2 bzw. F.3. Zusätzlich leuchtet die LED1 bei aktiviertem Ausgang.

#ifdef MEGA32                           ' Wird ein Mega32 verwendet wird der
#define aPort 10                        ' Ausgabeport auf diese Weise deklariert.
#endif
#ifdef MEGA128                          ' Wird ein Mega128 verwendet wird der
#define aPort 43                        ' Ausgabeport auf diese Weise deklariert.
#endif

' globale Variablendeklaration
Dim poti, sensor1, sensor2, hys As Integer
'-------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
      Port_DataDirBit(aPort,PORT_OUT)     ' Port wird auf Ausgabe vorbereitet.
      Port_DataDirBit(PORT_LED1,PORT_OUT) ' LED1 Port wird auf Ausgabe gesetzt.
      Port_WriteBit(PORT_LED1,PORT_OFF)   ' LED1 wird ausgeschalet.
      Do While (True)
            hys=5                         ' Hysterese für die Schaltschwellen.
            ADC_Set(ADC_VREF_VCC,ADC0)    ' Die Referenzspannung und der Messport
                                          ' werden ausgewählt.
            sensor1 = ADC_Read()          ' Die der anliegenden Spannung ent-
                                          ' sprechenden Zahl wird in die Variable
                                          ' sensor1 gespeichert.
            ADC_Set(ADC_VREF_VCC,ADC1)    ' Die Referenzspannung und der Messport
                                          ' werden ausgewählt.
            sensor2 = ADC_Read()          ' Die der anliegenden Spannung ent-
                                          ' sprechenden Zahl wird in die Variable
                                          ' sensor2 gespeichert.
            ADC_Set(ADC_VREF_VCC,ADC2)    ' Die Referenzspannung und der Messport
                                          ' werden ausgewählt.
            poti = ADC_Read()             ' Die der anliegenden Spannung ent-
                                          ' sprechenden Zahl wird in die Variable
                                          ' poti gespeichert.
            If sensor1 > sensor2+poti/2+hys Then
                  Port_WriteBit(PORT_LED1,PORT_ON)    ' LED2 wird eingeschaltet.
                  Port_WriteBit(aPort,PORT_OFF)       ' Relais wird eingeschaltet.
            End If
            If sensor1 < sensor2+poti/2 Then
                  Port_WriteBit(PORT_LED1,PORT_OFF)   ' LED2 wird eingeschaltet.
                  Port_WriteBit(aPort,PORT_ON)        ' Relais wird eingeschaltet.
            End If
      End While
End Sub
```

Temperaturdifferenzschalter – Quellcode CC

Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden.

```
// Temperaturdifferenzschalter
// Mega32
// Eingang: PortA.0, PortA.1, PortA.3
// Ausgang: Port B.2
// Mega128
// Eingang: PortF.0, PortF.1, PortF.2
// Ausgang: Port F.3
// erforderliche Library: IntFunc_Lib.cc

// Das Programm kann für den Mega32 und Mega128 verwendet werden.
// Über die A/D-Ports A.0 und A.1 bzw. F.0 und F.1 werden die zu vergleichenden
// Temperaturen eingelesen. Am Port A.3 bzw. F.3 wird ein Potentiometer ange-
// schlossen. Über diese kann die Temperaturdifferenz zwischen den beiden
// Fühlern eingestellt werden. Die Ausgabe des Schaltvorganges erfolgt über den
// Port B.2 bzw. F.3. Zusätzlich leuchtet die LED1 bei aktiviertem Ausgang.

#ifdef MEGA32                           // Wird ein Mega32 verwendet wird der
#define aPort 10                        // Ausgabeport auf diese Weise deklariert.
#endif
#ifdef MEGA128                          // Wird ein Mega128 verwendet wird der
#define aPort 43                        // Ausgabeport auf diese Weise deklariert.
#endif

// globale Variablendeklaration
int poti, sensor1, sensor2, hys;
//------------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
      Port_DataDirBit(aPort,PORT_OUT);    // Port wird auf Ausgabe vorbereitet.
      Port_DataDirBit(PORT_LED1,PORT_OUT);// LED1 Port wird auf Ausgabe gesetzt.
      Port_WriteBit(PORT_LED1,PORT_OFF);  // LED1 wird ausgeschalet.

      while (true)
      {
            hys=5;                        // Hysterese für die Schaltschwellen.
            ADC_Set(ADC_VREF_VCC,ADC0);   // Die Referenzspannung und der Messport
                                          // werden ausgewählt.
            sensor1 = ADC_Read();         // Die der anliegenden Spannung ent-
                                          // sprechenden Zahl wird in die Variable
                                          // sensor1 gespeichert.
            ADC_Set(ADC_VREF_VCC,ADC1);   // Die Referenzspannung und der Messport
                                          // werden ausgewählt.
            sensor2 = ADC_Read();         // Die der anliegenden Spannung ent-
                                          // sprechenden Zahl wird in die Variable
                                          // sensor2 gespeichert.
            ADC_Set(ADC_VREF_VCC,ADC2);   // Die Referenzspannung und der Messport
                                          // werden ausgewählt.
            poti = ADC_Read();            // Die der anliegenden Spannung ent-
                                          // sprechenden Zahl wird in die Variable
                                          // poti gespeichert.
            if (sensor1 > sensor2+poti/2+hys)
            {

                  Port_WriteBit(PORT_LED1,PORT_ON);   // LED2 wird eingeschaltet.
                  Port_WriteBit(aPort,PORT_OFF);      // Relais wird eingeschaltet.
            }
            if (sensor1 < sensor2+poti/2)
```

```
            {
                  Port_WriteBit(PORT_LED1,PORT_OFF); // LED2 wird eingeschaltet.
                  Port_WriteBit(aPort,PORT_ON);      // Relais wird eingeschaltet.
            }
      }
}
```

19.7 Acht-Kanal-Lauflicht

Es stehen Ihnen 5 Lauflichtvarianten zur Verfügung. Mit dem Schalter S1 auf dem Application-Board können Sie die einzelnen Typen durchschalten. Hierfür müssen Sie den Taster jeweils solange drücken, bis die neue Sequenz gestartet wird. Über das Potentiometer, das an den Port A.0 bzw. F.0 angeschlossen wird, können Sie die Geschwindigkeit stufenlos einstellen. Die Aktivierungszeit pro Kanal kann von 0 bis 1 Sekunde geregelt werden. Über Port C bzw. Port A werden die Schaltzustände an zwei Relaisplatinen weitergegeben.

Stückliste:

Menge	Art.-Nr.	Artikel	oder Art.-Nr.	oder Artikel
1	198206	Mega32	198219	Mega128
1	198245	Board Mega32	198258	Board Mega128
2	198836	Relais-Modul		
1	445673	22-kOhm-Poti lin		

Ein Ende des Potentiometerwiderstandes muss mit 5 V, das andere Ende mit GND (0 V) verbunden werden. Der Schleifer muss nun an das ADC-Port A.0 bzw. F.0 angeschlossen werden. Die Relaisplatinen verbinden Sie bitte mit dem Port C bzw. Port A. (siehe Kapitel 12.4). Für eine sichere Funktion muss des Weiteren das SDRAM des Mega128-Application-Boards über JP7 ausgeschaltet werden.

Lauflichtsteuerung – Quellcode CBasic
Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden.

```
' 8-Kanal-Lauflicht
' Mega32
' Eingang: PortA.0, PortD.2
' Ausgang: PortC.0 bis PortC.7
' Mega128
' Eingang: PortF.0, PortE.4
' Ausgang: PortA.0 bis PortA.7
' erforderliche Library: IntFunc_Lib.cc

' Das Programm kann für den Mega32 und Mega128 verwendet werden.
```

```
' Über das A/D-Ports A.0 bzw. F.0 wird die Lauflichtgeschwindigkeit über ein
' angeschlossenes Potentiometer eingestellt. Mit dem Taster S1 können die
' verschiedenen Lauflichtvarianten ausgewählt werden. Das Lauflicht wird über
' PortC.0 bis C.7 bzw. PortA.0 bis A.7 ausgegeben.

#ifdef MEGA32                           ' Wird ein Mega32 verwendet wird der
#define aPort PortC                     ' Ausgabeport auf diese Weise deklariert.
#define bit 16                          ' Erste Portbitadresse wird zugewiesen
#endif

#ifdef MEGA128                          ' Wird ein Mega128 verwendet wird der
#define aPort PortA                     ' Ausgabeport auf diese Weise deklariert.
#define bit 0                           ' Erste Portbitadresse wird zugewiesen
#endif

' globale Variablendeklaration
Dim zeit, schalter1 As Integer
Dim lauflicht As Integer
Dim i As Integer

'-------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
   Port_DataDir(aPort,0xFF)             ' Ports auf Ausgabe vorbereiten.
   Port_DataDirBit(PORT_SW1,PORT_IN)    ' Ports der Schalter werden auf
   Port_DataDirBit(PORT_SW2,PORT_IN)    ' Eingang vorbereitet.
   lauflicht=1                          ' Default-Wert wird zugewiesen.
   schalter1=1                          ' Default-Wert wird zugewiesen.

   Do While (True)                      ' Endlosschleife wird gestartet.
      ADC_Set(ADC_VREF_VCC,ADC0)        ' Die Referenzspannung und der Messport
                                        ' werden ausgewählt.
      zeit = ADC_Read()                 ' Die der anliegenden Spannung ent-
                                        ' sprechenden Zahl wird in die Variable
                                        ' zeit gespeichert.
      If Port_ReadBit(PORT_SW1)=PORT_ON Then
                                        ' Ist Schalter 1 gedrückt wird der
            schalter1 = schalter1+1     ' Zähler schalter1 um Eins erhöht.
                  If schalter1>5 Then         ' Die nächste Lauflichtvariante
                        schalter1=1           ' wird angewählt. Ist die letzte er-
                  End If                ' reicht wird auf die erste gewechselt.
      End If
      Select Case schalter1             ' Auswahl der Varianten abhängig vom
                                        ' Zählerstand schalter1.



      Case 1                            ' Erste Variante:
                  i = 0
                  Do While i <8         ' Zähler von 0 bis 7 wird gestartet.
                     Port_WriteBit(bit+i,PORT_OFF)    ' Relais wird eingeschaltet.
                     AbsDelay(zeit)                   ' Poti Zeitverzögerung.
                     Port_WriteBit(bit+i,PORT_ON)     ' Relais wird eingeschaltet.
                     i=i+1
                  End While
      Case 2                                          ' Zweite Variante:
                  i = 7
                  Do While i>-1         ' Zähler von 7 bis 0 wird gestartet.
                     Port_WriteBit(bit+i,PORT_OFF)    ' Relais wird eingeschaltet.
                     AbsDelay(zeit)                   ' Poti Zeitverzögerung.
                     Port_WriteBit(bit+i,PORT_ON)     ' Relais wird eingeschaltet.
                     i=i-1
                  End While
      Case 3                                          ' Dritte Variante:
                  i = 0
```

```
                  Do While i <8               ' Zähler von 0 bis 7 wird gestartet.
                     Port_WriteBit(bit+i,PORT_OFF)    ' Relais wird eingeschaltet.
                     AbsDelay(zeit)                   ' Poti Zeitverzögerung.
                     Port_WriteBit(bit+i,PORT_ON)     ' Relais wird eingeschaltet.
                     i=i+1
                  End While
                  i = 6
                  Do While i>0                ' Zähler von 6 bis 1 wird gestartet.
                     Port_WriteBit(bit+i,PORT_OFF)    ' Relais wird eingeschaltet.
                     AbsDelay(zeit)                   ' Poti Zeitverzögerung.
                     Port_WriteBit(bit+i,PORT_ON)     ' Relais wird eingeschaltet.
                     i=i-1
                  End While
            Case 4                                    ' Vierte Variante:
                  i = 0
                  Do While i<4                ' Zähler von 0 bis 3 wird gestartet.
                     Port_WriteBit(bit+i,PORT_OFF)    ' Relais wird eingeschaltet.
                     Port_WriteBit(bit+7-i,PORT_OFF)  ' Relais wird eingeschaltet.
                     AbsDelay(zeit)                   ' Poti Zeitverzögerung.
                     Port_WriteBit(bit+i,PORT_ON)     ' Relais wird eingeschaltet.
                     Port_WriteBit(bit+7-i,PORT_ON)   ' Relais wird eingeschaltet.
                     i=i+1
                  End While
            Case 5                                    ' Fünfte Variante:
                  i = 3
                  Do While i>-1               ' Zähler von 3 bis 0 wird gestartet.
                     Port_WriteBit(bit+i,PORT_OFF)    ' Relais wird eingeschaltet.
                     Port_WriteBit(bit+7-i,PORT_OFF)  ' Relais wird eingeschaltet.
                     AbsDelay(zeit)                   ' Poti Zeitverzögerung.
                     Port_WriteBit(bit+i,PORT_ON)     ' Relais wird eingeschaltet.
                     Port_WriteBit(bit+7-i,PORT_ON)   ' Relais wird eingeschaltet.
                  i=i-1
                  End While
            End Case
      End While
End Sub
```

Lauflichtsteuerung – Quellcode CC

Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden.

```
// 8-Kanal-Lauflicht
// Mega32
// Eingang: PortA.0, PortD.2
// Ausgang: PortC.0 bis PortC.7
// Mega128
// Eingang: PortF.0, PortE.4
// Ausgang: PortA.0 bis PortA.7
// erforderliche Library: IntFunc_Lib.cc

// Das Programm kann für den Mega32 und Mega128 verwendet werden.
// Über das A/D-Ports A.0 bzw. F.0 wird die Lauflichtgeschwindigkeit über ein
// angeschlossenes Potentiometer eingestellt. Mit dem Taster S1 können die
// verschiedenen Lauflichtvarianten ausgewählt werden. Das Lauflicht wird über
// PortC.0 bis C.7 bzw. PortA.0 bis A.7 ausgegeben.

#ifdef MEGA32                          // Wird ein Mega32 verwendet wird der
#define aPort PortC                    // Ausgabeport auf diese Weise deklariert.
#define bit 16                         // Erste Portbitadresse wird zugewiesen
#endif

#ifdef MEGA128                         // Wird ein Mega128 verwendet wird der
```

```
#define aPort PortA                        // Ausgabeport auf diese Weise deklariert.
#define bit 0                              // Erste Portbitadresse wird zugewiesen
#endif

// globale Variablendeklaration
int zeit, schalter1;
int lauflicht;
int i;

//---------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
   Port_DataDir(aPort,0xFF);               // Ports auf Ausgabe vorbereiten.
   Port_DataDirBit(PORT_SW1,PORT_IN);      // Ports der Schalter werden auf
   Port_DataDirBit(PORT_SW2,PORT_IN);      // Eingang vorbereitet.
   lauflicht=1;                            // Default-Wert wird zugewiesen.
   schalter1=1;                            // Default-Wert wird zugewiesen.

   while (true)                            // Endlosschleife wird gestartet.
   {
       ADC_Set(ADC_VREF_VCC,ADC0);         // Die Referenzspannung und der Messport
                                           // werden ausgewählt.
       zeit = ADC_Read();                  // Die der anliegenden Spannung ent-
                                           // sprechenden Zahl wird in die Variable
                                           // zeit gespeichert.
       if (Port_ReadBit(PORT_SW1)==PORT_ON)
       {                                   // Ist Schalter 1 gedrückt wird der
           schalter1++;                    // Zähler schalter1 um Eins erhöht.
           if (schalter1>5) schalter1=1;   // Die nächste Lauflichtvariante
       }                                   // wird angewählt. Ist die letzte er-

                                           // reicht wird auf die erste gewechselt.
       switch(schalter1)                   // Auswahl der Varianten abhängig vom
       {                                   // Zählerstand schalter1.
           case 1:                         // Erste Variante:
               for (i=0;i<8;i++)           // Zähler von 0 bis 7 wird gestartet.
               {
                   Port_WriteBit(bit+i,PORT_OFF); // Relais wird eingeschaltet.
                   AbsDelay(zeit);                // Poti Zeitverzögerung.
                   Port_WriteBit(bit+i,PORT_ON);  // Relais wird eingeschaltet.
               }
           break;
           case 2:                                // Zweite Variante:
               for (i=7;i>-1;i--)                 // Zähler von 7 bis 0 wird gestartet.
               {
                   Port_WriteBit(bit+i,PORT_OFF); // Relais wird eingeschaltet.
                   AbsDelay(zeit);                // Poti Zeitverzögerung.
                   Port_WriteBit(bit+i,PORT_ON);  // Relais wird eingeschaltet.
               }
           break;
           case 3:                                // Dritte Variante.
               for (i=0;i<8;i++)                  // Zähler von 0 bis 7 wird gestartet.
               {
                   Port_WriteBit(bit+i,PORT_OFF); // Relais wird eingeschaltet.
                   AbsDelay(zeit);                // Poti Zeitverzögerung.
                   Port_WriteBit(bit+i,PORT_ON);  // Relais wird eingeschaltet.
               }
               for (i=6;i>0;i--)                  // Zähler von 6 bis 1 wird gestartet.
               {
                   Port_WriteBit(bit+i,PORT_OFF); // Relais wird eingeschaltet.
                   AbsDelay(zeit);                // Poti Zeitverzögerung.
                   Port_WriteBit(bit+i,PORT_ON);  // Relais wird eingeschaltet.
               }
           break;
```

```
          case 4:                                  // Vierte Variante.
              for (i=0;i<4;i++)                    // Zähler von 0 bis 3 wird gestartet.
              {
                  Port_WriteBit(bit+i,PORT_OFF);   // Relais wird eingeschaltet.
                  Port_WriteBit(bit+7-i,PORT_OFF); // Relais wird eingeschaltet.
                  AbsDelay(zeit);                  // Poti Zeitverzögerung.
                  Port_WriteBit(bit+i,PORT_ON);    // Relais wird eingeschaltet.
                  Port_WriteBit(bit+7-i,PORT_ON);  // Relais wird eingeschaltet.
              }
          break;
          case 5:                                  // Fünfte Variante.
              for (i=3;i>-1;i--)                   // Zähler von 3 bis 0 wird gestartet.
              {
                  Port_WriteBit(bit+i,PORT_OFF);   // Relais wird eingeschaltet.
                  Port_WriteBit(bit+7-i,PORT_OFF); // Relais wird eingeschaltet.
                  AbsDelay(zeit);                  // Poti Zeitverzögerung.
                  Port_WriteBit(bit+i,PORT_ON);    // Relais wird eingeschaltet.
                  Port_WriteBit(bit+7-i,PORT_ON);  // Relais wird eingeschaltet.
              }
          break;
      }
  }
}
```

19.8 Digital-Timer

Über diesen Countdown-Timer können Sie Zeiten bis 99 Stunden, 99 Minuten und 99 Sekunden einstellen. Die Eingabe erfolgt über die Folientastatur. Durch Drücken der Taste * wird der Countdown gestartet. Wird die Taste * bei laufendem Timer gedrückt, wird dieser gestoppt und auf den alten eingestellten Wert zurückgesetzt. Nun kann eine neue Zeit eingegeben werden oder die alte von neuem gestartet werden. Ist der Countdown abgelaufen, wird über den Port A.0 ein Signal ausgegeben, das z. B. zur Ansteuerung eines Summers geeignet ist.

Stückliste:

Menge	Art.-Nr.	Artikel	oder Art.-Nr.	oder Artikel
1	198206	Mega32	198219	Mega128
1	198245	Board Mega32	198258	Board Mega128
1	710070	Miniatursummer		

Um den Summer ansteuern zu können, verbinden Sie bitte das rote Kabel mit 5 V und das schwarze Kabel mit dem Port A.0.

Timer – Quellcode CBasic

Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
' CountDown Timer
' Mega32, Mega128
' Ausgang: PortA.0
' erforderliche Library: IntFunc_Lib.cc, Key_Lib.cc, LCD_Lib.cc

' Das Programm kann für den Mega32 und Mega128 verwendet werden.
' Über die Folientastatur kann ein Timerwert in der Form Stunden, Minuten,
' Sekunden eingegeben werden. Der maximale Wert beträgt 99:99:99. Wird die
' Taste * auf der Folientastatur gedrück beginnt der CountDown. Wird die
' Taste * erneut gedrückt wird der CountDown gestoppt und der Timer wird auf
' den zuvor eingestellten Wert zurückgesetzt. Nach Ablauf des CountDowns wird
' der Ausgang A.0 aktiviert. Dieser wird ebenfalls nach dem Drücken der Taste
' * deaktiviert.

' globale Variablendeklaration
Dim zeile1(10) As Char                      ' Variabel für Displayausgabe
Dim sekunde, minute, stunde, cnt1 As Word   ' Timer Variablen
Dim sez, see, mz, me, sz, se As Word        ' Variablen zur Berechnung der Timerzeit
Dim key_in As Word                          ' Tastatur Eingabevariable
Dim key_ch, key_ch1 As Byte                 ' Tastatur verarbeitungsvariablen
Dim pos, start As Byte
'-------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
   Port_DataDirBit(0,PORT_OUT)              ' Port A.0 wird auf Ausgabe vorbereitet.
   Port_WriteBit(0,PORT_OFF)                ' Port A.0 wird ausgeschaltet

   LCD_Init()                               ' Display wird initialisiert.
   LCD_ClearLCD()                           ' Display wird gelöscht.
   Key_Init()                               ' Tastatur wird initialisert.
   Display_Buffer_Set()                     ' Displaypuffer wird aktualisiert.
   ausgabe()                                ' Display wird geschrieben

   Irq_SetVect(INT_TIM2COMP,INT_10ms)       ' Interrupt Service Routine definieren
                                            ' Timer2 erzeugt einen 10ms interrupt
   Do While (True)                          ' Endlosschleife
   End While
End Sub
```

Interrupt

```
'-------------------------------------------------------------------------------
' Timer
'
Sub INT_10ms()
   Dim irqcnt As Integer
   If start =1 Then                   ' Nur wenn die Taste * gedrückt wurde, wird
                                      ' dieser Programmteil abgearbeitet.
       cnt1 = cnt1+1                  ' 10ms Zähler wird um Eins erhöht.
       If cnt1=100 Then               ' Wenn 1 Sekunde vergangen ist, wird diese
                                      ' Schleife abgearbeitet.
           If sekunde >0 Then         ' Nur wenn der aktuelle Wert der Sekunden
                                      ' größer als 0 ist, wird der Sekundenwert
               sekunde = sekunde-1    ' um Eins verringert.

           Else
               If minute >0 Then      ' Ist dies nicht der Fall wird geprüft ob
                                      ' der Minutenwert größer als 0 ist. Wenn ja,
                   minute = minute-1  ' wird dieser um Eins verringert und die
                   sekunde = 59       ' Sekunden auf 59 gesetzt.

               Else
                   If stunde >0 Then   ' Ist dies nicht der Fall wird geprüft ob
                                       ' der Stundenwert größer als 0 ist. Wenn ja,
```

```
                    stunde = stunde-1' wird dieser um Eins verringert und die
                    minute = 59      ' Minuten und die Sekunden werden auf 59
                    sekunde = 59     ' gesetzt.
                Else
                                     ' Ist dies nicht der Fall ist der Timer
                    summer()         ' abgelaufen und der Summer wird aktiviert.
                    sekunde = 0      ' Die Sekunden werden auf 0 gesetzt.
                End If
            End If
        End If
        cnt1=0                       ' Der 10ms Zähler wird auf 0 gesetzt.
        Display_Buffer_Set()         ' Display Buffer wird aktualisiert
        ausgabe()                    ' Jede Sekunde wird die Zeit ausgegeben.
    End If
End If
tastatur()                           ' Tastaturabfrage wird aufgerufen
irqcnt=Irq_GetCount(INT_TIM2COMP)    ' Interrupt Request Counter
End Sub
```

Tastatur

```
'-------------------------------------------------------------------------------
' Tastaturabfrage der Folientastatur mit Zeichenumwandlung
'
Sub tastatur()
   key_in=Key_Scan()                      ' Die Tastatureingabe wird gelesen
   If key_in<>0 Then                      ' und in die Variable key_in geschrieben
       key_ch=Key_TranslateKey(key_in)    ' Die Eingabe wird in ASCII Zeichen
       If key_ch<>key_ch1 Then            ' umgewendelt. Handelt es sich um
                                          ' eine neue Eingabe wird diese
           key_ch1=key_ch                 ' weiterbearbeitet.
           If key_ch=35 Or key_ch=42 Then ' Bei einem * oder # wird kein
               pos =pos-1                 ' Zeichen auf dem Display ausgegeben.
               If key_ch=42 And start=0 Then   ' Mit dem Zeichen * kann der Timer
                   start =1               ' gestartet werden.
                   LCD_CursorOff()        ' Der Cursor wird nach dem Start
               Else                       ' ausgeschaltet.
                   start =0               ' Bei laufendem Timer kann dieser
                   LCD_CursorOn()         ' mit der Taste * gestoppt werden.
                   zeitberechnung()       ' Der Cursor wird wieder aktiviert
                   Display_Buffer_Set()   ' und der alte Timerwert wird wieder
                   ausgabe()              ' auf das Display zurückgeschrieben.
               End If
           Else
               LCD_WriteChar(key_ch)
               Select Case pos            ' Handelt es sich bei der Eingabe
                   Case 0                 ' um eine Zahl wird dieser auf
                       sz =(key_ch-48)*10 ' dem Display ausgegeben.
                   Case 1                 ' Je nach Position werden die
                       se =key_ch-48      ' Werte der Variablen Sekunde,
                   Case 3                 ' Minute und Stunde verändert.
                       mz =(key_ch-48)*10 ' Zur Bestimmung der aktuellen
                   Case 4                 ' Einstellungen werden hier die
                       me =key_ch-48      ' Variablen in Zehner- und Einer-
                   Case 6:                ' Stellen zerlegt und entsprechend
                       sez =(key_ch-48)*10' aufbereitet.
                   Case 7:
                       see =key_ch-48
               End Case
           End If
           pos =pos+1                     ' Cursorposition wird um Eins erhöht.
           If pos=2 Then                  ' Der Doppelpunkt zwischen Stunden und
               pos = 3                    ' Minuten wird übersprungen.
           End If
```

```
            If pos=5 Then                   ' Der Doppelpunkt zwischen Minuten und
                pos = 6                     ' Sekunden wird übersprungen.
            End If
            If pos>7 Then                   ' Vom rechten Ende des Displays wird
                pos = 0                     ' auf die linke Seite gesprungen.
            End If
            LCD_CursorPos(pos)              ' Cursor wird positioniert.
            zeitberechnung()                ' h, m, s werden berechnet
        End If
    Else
        key_ch1=-1
    End If
End Sub
```

Ausgabe

```
'-------------------------------------------------------------------------------
' Displayausgabe
'
Sub ausgabe()
   LCD_CursorPos(&H00)                          ' Cursor wird positioniert.
   LCD_WriteText(zeile1)                        ' Zeile1 wird geschrieben
   LCD_CursorPos(&H00)                          ' Cursor wird positioniert.
End Sub

'-------------------------------------------------------------------------------
' Summer
'
Sub summer()
   Port_WriteBit(0,PORT_ON)                     ' PortA.0 wird eingeschaltet.
   AbsDelay(500)                                ' 500ms Verzögerung.
   Port_WriteBit(0,PORT_OFF)                    ' PortA.0 wird ausgeschaltet.
End Sub

'-------------------------------------------------------------------------------
' Eingabewert in den Displaypuffer schreiben
'
Sub Display_Buffer_Set()
   Dim sep(2) As Char                         ' Array für Trennzeichen deklarieren.
   sep=„:“                                    ' Wert wird dem Array zugewiesen.
   Str_WriteWord(stunde,10,zeile1,0,2)        ' Stunden, Minuten, und Sekunden
   Str_Copy(zeile1,sep,STR_APPEND)            ' werden zu einem String zusammen-
   Str_WriteWord(minute,10,zeile1,STR_APPEND,2) ' gefasst.
   Str_Copy(zeile1,sep,STR_APPEND)
   Str_WriteWord(sekunde,10,zeile1,STR_APPEND,2)
End Sub

'-------------------------------------------------------------------------------
' h, m, s werden aus den Eingabewerten berechnet
'
Sub zeitberechnung()
   stunde = sz+se           ' Stunden werden aus den Variablen sz und se gebildet
   minute = mz+me           ' Minuten werden aus den Variablen mz und me gebildet
   sekunde = sez+see        ' Sekunden werden aus den Variablen sez und see gebildet
End Sub
```

Timer – Quellcode CC

Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
// CountDown Timer
// Mega32, Mega128
// Ausgang: PortA.0
// erforderliche Library: IntFunc_Lib.cc, Key_Lib.cc, LCD_Lib.cc

// Das Programm kann für den Mega32 und Mega128 verwendet werden.
// Über die Folientastatur kann ein Timerwert in der Form Stunden, Minuten,
// Sekunden eingegeben werden. Der maximale Wert beträgt 99:99:99. Wird die

// Taste * auf der Folientastatur gedrück beginnt der CountDown. Wird die
// Taste * erneut gedrückt wird der CountDown gestoppt und der Timer wird auf
// den zuvor eingestellten Wert zurückgesetzt. Nach Ablauf des CountDowns wird
// der Ausgang A.0 aktiviert. Dieser wird ebenfalls nach dem Drücken der Taste
// * deaktiviert.

// globale Variablendeklaration
char zeile1[10];                          // Variabel für Displayausgabe
word sekunde, minute, stunde, cnt1;       // Timer Variablen
word sez, see, mz, me, sz, se;            // Variablen zur Berechnung der Timerzeit
word key_in;                              // Tastatur Eingabevariable
byte key_ch, key_ch1;                     // Tastatur verarbeitungsvariablen
byte pos, start;
//---------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
   Port_DataDirBit(0,PORT_OUT);           // Port A.0 wird auf Ausgabe vorbereitet.
   Port_WriteBit(0,PORT_OFF);             // Port A.0 wird ausgeschaltet

   LCD_Init();                            // Display wird initialisiert.
   LCD_ClearLCD();                        // Display wird gelöscht.
   Key_Init();                            // Tastatur wird initialisert.
   Display_Buffer_Set();                  // Displaypuffer wird aktualisiert.
   ausgabe();                             // Display wird geschrieben

   Irq_SetVect(INT_TIM2COMP,INT_10ms);    // Interrupt Service Routine definieren
                                          // Timer2 erzeugt einen 10ms interrupt
   while (true);                          // Endlosschleife
}
```

Interrupt

```
//---------------------------------------------------------------------------
// Timer
//
void INT_10ms(void)
{
   int irqcnt;
   if (start ==1)                 // Nur wenn die Taste * gedrückt wurde, wird
   {                              // dieser Programmteil abgearbeitet.
       cnt1++;                    // 10ms Zähler wird um Eins erhöht.
       if (cnt1==100)             // Wenn 1 Sekunde vergangen ist, wird diese
       {                          // Schleife abgearbeitet.
           if (sekunde >0)        // Nur wenn der aktuelle Wert der Sekunden
           {                      // größer als 0 ist, wird der Sekundenwert
               sekunde--;         // um Eins verringert.
           }
           else if (minute >0)    // Ist dies nicht der Fall wird geprüft ob
               {                  // der Minutenwert größer als 0 ist. Wenn ja,
                   minute--;      // wird dieser um Eins verringert und die
                   sekunde = 59;  // Sekunden auf 59 gesetzt.
               }
```

```
               else if (stunde >0)     // Ist dies nicht der Fall wird geprüft ob
               {                       // der Stundenwert größer als 0 ist. Wenn ja,
                   stunde--;           // wird dieser um Eins verringert und die
                   minute = 59;        // Minuten und die Sekunden werden auf 59
                   sekunde = 59;       // gesetzt.
               }
               else

{                                      // Ist dies nicht der Fall ist der Timer
                   summer();           // abgelaufen und der Summer wird aktiviert.
                   sekunde = 0;        // Die Sekunden werden auf 0 gesetzt.
               }
           cnt1=0;                     // Der 10ms Zähler wird auf 0 gesetzt.
           Display_Buffer_Set();       // Display Buffer wird aktualisiert
           ausgabe();                  // Jede Sekunde wird die Zeit ausgegeben.
       }
   }
   tastatur();                         // Tastaturabfrage wird aufgerufen
   irqcnt=Irq_GetCount(INT_TIM2COMP);  // Interrupt Request Counter
}
```

Tastatur

```
//-------------------------------------------------------------------------------
// Tastaturabfrage der Folientastatur mit Zeichenumwandlung
//
void tastatur(void)
{
   key_in=Key_Scan();                     // Die Tastatureingabe wird gelesen
   if (key_in!=0)                         // und in die Variable key_in geschrieben
   {
       key_ch=Key_TranslateKey(key_in);   // Die Eingabe wird in ASCII Zeichen
       if (key_ch!=key_ch1)               // umgewendelt. Handelt es sich um
       {                                  // eine neue Eingabe wird diese
           key_ch1=key_ch;                // weiterbearbeitet.
           if (key_ch==35 | key_ch==42)   // Bei einem * oder # wird kein
           { pos --;                      // Zeichen auf dem Display ausgegeben.
               if (key_ch ==42&& start==0)// Mit dem Zeichen * kann der Timer
               { start =1;                // gestartet werden.
                   LCD_CursorOff();       // Der Cursor wird nach dem Start
               }                          // ausgeschaltet.
               else
               { start =0;                // Bei laufendem Timer kann dieser
                    LCD_CursorOn();       // mit der Taste * gestoppt werden.
                    zeitberechnung();     // Der Cursor wird wieder aktiviert
                    Display_Buffer_Set(); // und der alte Timerwert wird wieder
                    ausgabe();            // auf das Display zurückgeschrieben.
               }
           }
           else
           {
               LCD_WriteChar(key_ch);
               switch (pos)
               {
                   case 0:
                       sz =(key_ch-48)*10;
                   break;                 // Handelt es sich bei der Eingabe
                   case 1:                // um eine Zahl wird dieser auf
                       se =key_ch-48;     // dem Display ausgegeben.
                   break;                 // Je nach Position werden die
                   case 3:                // Werte der Variablen Sekunde,
                       mz =(key_ch-48)*10;// Minute und Stunde verändert.
                   break;                 // Zur Bestimmung der aktuellen
                   case 4:                // Einstellungen werden hier die
                       me =key_ch-48;     // Variablen in Zehner- und Einer-
```

```
                    break;                  // Stellen zerlegt und entsprechend
                    case 6:                 // aufbereitet.

                    sez =(key_ch-48)*10;
                    break;
                    case 7:
                        see =key_ch-48;
                   break;
                }
            }
            pos++;                          // Cursorposition wird um Eins erhöht.
            if (pos==2)                     // Der Doppelpunkt zwischen Stunden und
            { pos = 3;}                     // Minuten wird übersprungen.
            if (pos==5)                     // Der Doppelpunkt zwischen Minuten und
            { pos = 6;}                     // Sekunden wird übersprungen.
            if (pos>7)                      // Vom rechten Ende des Displays wird
            { pos = 0;}                     // auf die linke Seite gesprungen.
            LCD_CursorPos(pos);             // Cursor wird positioniert.
            zeitberechnung();               // h, m, s werden berechnet
        }
    }
    else
    {
        key_ch1=-1;
    }
}
```

Ausgabe

```
//-------------------------------------------------------------------------------
// Displayausgabe
//
void ausgabe(void)
{
   LCD_CursorPos(0x00);                     // Cursor wird positioniert.
   LCD_WriteText(zeile1);                   // Zeile1 wird geschrieben
   LCD_CursorPos(0x00);                     // Cursor wird positioniert.
}

//-------------------------------------------------------------------------------
// Summer
//
void summer(void)
{
   Port_WriteBit(0,PORT_ON);                // PortA.0 wird eingeschaltet.
   AbsDelay(500);                           // 500ms Verzögerung.
   Port_WriteBit(0,PORT_OFF);               // PortA.0 wird ausgeschaltet.
}

//-------------------------------------------------------------------------------
// Eingabewert in den Displaypuffer schreiben
//
void Display_Buffer_Set(void)
{
   char sep[2];                                 // Array für Trennzeichen deklarieren.
   sep=":";                                     // Wert wird dem Array zugewiesen.
   Str_WriteWord(stunde,10,zeile1,0,2);         // Stunden, Minuten, und Sekunden
   Str_Copy(zeile1,sep,STR_APPEND);             // werden zu einem String zusammen-
   Str_WriteWord(minute,10,zeile1,STR_APPEND,2);// gefasst.
   Str_Copy(zeile1,sep,STR_APPEND);
   Str_WriteWord(sekunde,10,zeile1,STR_APPEND,2);
}

//-------------------------------------------------------------------------------
// h, m, s werden aus den Eingabewerten berechnet
```

```
//
void zeitberechnung(void)
{
   stunde = sz+se;          // Stunden werden aus den Variablen sz und se gebildet
   minute = mz+me;          // Minuten werden aus den Variablen mz und me gebildet
   sekunde = sez+see;       // Sekunden werden aus den Variablen sez und see gebildet
}
```

19.9 Stoppuhr

Mit dieser Stoppuhr können Sie Zeiten im 10-ms-Bereich messen und auf dem Display anzeigen lassen. Über den Taster SW1 wird die Zeit gestartet und angehalten. Da der Schalter an einem ganz normalen Port angeschlossen ist, können Sie das Schaltsignal natürlich auch über einen externen Geber auf diesen Port legen. Wird bei laufender Zeit die Taste SW2 gedrückt, so wird in der zweiten Zeile die Zwischenzeit angezeigt. Haben Sie die Stoppuhr mit der Taste SW1 gestoppt, so können Sie mit der Taste SW2 auf 0 zurücksetzen.

Stückliste:

Menge	Art.-Nr.	Artikel	oder Art.-Nr.	oder Artikel
1	198206	Mega32	198219	Mega128
1	198245	Board Mega32	198258	Board Mega128

Stoppuhr – Quellcode CC

Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden. Es ist zu empfehlen die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
// Stoppuhr
// Mega32, Mega128
// erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

// Das Programm kann für den Mega32 und Mega128 verwendet werden.
// Über den Taster SW1 wird die Zeit gestartet und wieder angehalten. Bei
// laufender Zeit erhält man durch Drücken der Taste SW2 die Zwischenzeit in
// Ist die Zeit gestoppt, können Sie mit SW2 das Display auf 0 zurücksetzen.

// globale Variablendeklaration
char zeile1[10];                        // Variabel für Displayausgabe
word sekunde, minute, stunde, cnt1;     // Timer Variablen
byte pos, start, stopp, prell;
//---------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
```

```
   Port_DataDirBit(PORT_SW1,PORT_IN);     // SW1-Port wird auf Ausgabe vorbereitet.

   Port_DataDirBit(PORT_SW2,PORT_IN);     // SW2-Port wird auf Ausgabe vorbereitet.
   Port_WriteBit(PORT_SW1,PORT_OFF);      // Pullup-Widerstände von SW1 und SW2
   Port_WriteBit(PORT_SW2,PORT_OFF);      // werden eingeschaltet.

   LCD_Init();                            // Display wird initialisiert.
   LCD_ClearLCD();                        // Display wird gelöscht.
   LCD_CursorOff();                       // Cursor wird ausgeschaltet.
   Display_Buffer_Set();                  // Displaypuffer wird aktualisiert.
   ausgabe();                             // Display wird geschrieben

   Irq_SetVect(INT_TIM2COMP,INT_10ms);    // Interrupt Service Routine definieren
                                          // Timer2 erzeugt einen 10ms interrupt
   while (true);                          // Endlosschleife
}
```

Interrupt

```
//---------------------------------------------------------------------------
// Zeit
//
void INT_10ms(void)
{
   int irqcnt;
   if (start ==1)                         // Nur wenn die Taste SW1 gedrückt wurde,
   {                                      // wird dieser Programmteil abgearbeitet.
       cnt1++;                            // 10ms Zähler wird um Eins erhöht.
       if (cnt1==100)                     // Wenn 1 Sekunde vergangen ist, wird diese
       {                                  // Schleife abgearbeitet.
           sekunde++;                     // Sekundenzähler wird um Eins erhöht.
           if (sekunde==60)               // Wenn 60 Sekunden vergangen sind, wird
           {                              // diese Schleife abgearbeitet.
               minute++;                  // Minutenzähler wird um Eins erhöht.
               if (minute==60)            // Wenn 60 Minuten vergangen sind, wird
               {                          // das Display auf 00:00:00 zurückgesetzt.
                   minute=0;              // Der Minutenzähler wird auf 0 gesetzt.
               }
               sekunde=0;                 // Der Sekundenzähler wird auf 0 gesetzt.
           }
           cnt1=0;                        // Der 10ms Zähler wird auf 0 gesetzt.
       }
       Display_Buffer_Set();              // Display Buffer wird aktualisiert
       ausgabe();                         // Jede Sekunde wird die Zeit ausgegeben.
   }
   tasten();                              // Aufruf der Funktion zur Tastenabfrage.
   irqcnt=Irq_GetCount(INT_TIM2COMP);     // Interrupt Request Counter
}
```

Ausgabe

```
//---------------------------------------------------------------------------
// Displayausgabe
//
void ausgabe(void)
{

   LCD_CursorPos(0x00);                   // Cursor wird positioniert.
   LCD_WriteText(zeile1);                 // Zeile1 wird geschrieben
}

void ausgabe2(void)
```

```
{
   LCD_CursorPos(0x40);                          // Cursor wird positioniert.
   LCD_WriteText(zeile1);                        // Zeile2 wird geschrieben
}

//-------------------------------------------------------------------------------
// Eingabewert in den Displaypuffer schreiben
//
void Display_Buffer_Set(void)
{
   char sep[2];                                  // Array für Trennzeichen deklarieren.
   sep=":";                                      // Wert wird dem Array zugewiesen.
   Str_WriteWord(minute,10,zeile1,0,2);          // Minuten, Sekunden und Zehntel
   Str_Copy(zeile1,sep,STR_APPEND);              // werden zu einem String zusammen-
   Str_WriteWord(sekunde,10,zeile1,STR_APPEND,2); // gefasst.
   Str_Copy(zeile1,sep,STR_APPEND);
   Str_WriteWord(cnt1,10,zeile1,STR_APPEND,2);
}
```

Tasten

```
//-------------------------------------------------------------------------------
// Tastenabfrage
//
void tasten(void)
{
   prell++;                          // Diese Variable eleminiert das Tasterprellen.
   if (Port_ReadBit(PORT_SW1)==0 && start==0 && prell>20)
   {                                 // Wird der Taster SW1 gedrückt und die Zeit läuft
       prell=0;                      // noch nicht läuft, so wird die Zeit gestartet.
       start=1;
   }
   if (Port_ReadBit(PORT_SW2)==0 && start==1 && prell>20)
   {                                 // Wird der Taster SW2 gedrückt und die Zeit läuft
       prell=0;                      // bereits, wird die Zwischenzeit in der zweiten
       ausgabe2();                   // Zeile ausgegeben.
   }
   if (Port_ReadBit(PORT_SW1)==0 && start==1 && prell>20)
   {                                 // Wird der Taster SW1 gedürckt und die Zeit läuft
       prell=0;                      // bereits, so wird die Zeit gestoppt.
       start=0;
   }
   if (Port_ReadBit(PORT_SW2)==0 && start==0 && prell>20)
   {                                 // Wird der Taster SW2 gedrückt und die Zeit läuft
       prell=0;                      // nicht, so wird das Display zurückgesetzt.
       LCD_ClearLCD();               // Display wird gelöscht.
       LCD_CursorOff();              // Cursor wird ausgeschaltet.
       cnt1 = 0; sekunde = 0; minute = 0;  // Variablen werden zurückgesetzt.
       Display_Buffer_Set();         // Displaypuffer wird aktualisiert.
       ausgabe();                    // Display wird geschrieben
   }
}
```

Stoppuhr – Quellcode CBasic

Das Programm kann sowohl für den Mega32 als auch für den Mega128 unverändert verwendet werden. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
' Stoppuhr
' Mega32, Mega128
' erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc

' Das Programm kann für den Mega32 und Mega128 verwendet werden.
' Über den Taster SW1 wird die Zeit gestartet und wieder angehalten. Bei
' laufender Zeit erhält man durch Drücken der Taste SW2 die Zwischenzeit in
' Ist die Zeit gestoppt, können Sie mit SW2 das Display auf 0 zurücksetzen.

' globale Variablendeklaration
Dim zeile1(10) As Char                    ' Variabel für Displayausgabe
Dim sekunde, minute, stunde, cnt1 As Word ' Timer Variablen
Dim pos, start, stopp, prell As Byte
'-------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
   Port_DataDirBit(PORT_SW1,PORT_IN)      ' SW1-Port wird auf Ausgabe vorbereitet.
   Port_DataDirBit(PORT_SW2,PORT_IN)      ' SW2-Port wird auf Ausgabe vorbereitet.
   Port_WriteBit(PORT_SW1,PORT_OFF)       ' Pullup-Widerstände von SW1 und SW2
   Port_WriteBit(PORT_SW2,PORT_OFF)       ' werden eingeschaltet.

   LCD_Init()                             ' Display wird initialisiert.
   LCD_ClearLCD()                         ' Display wird gelöscht.
   LCD_CursorOff()                        ' Cursor wird ausgeschaltet.
   Display_Buffer_Set()                   ' Displaypuffer wird aktualisiert.
   ausgabe()                              ' Display wird geschrieben

   Irq_SetVect(INT_TIM2COMP,INT_10ms)     ' Interrupt Service Routine definieren
                                          ' Timer2 erzeugt einen 10ms interrupt
   Do While (True)                        ' Endlosschleife
   End While
End Sub
```

Interrupt

```
'-------------------------------------------------------------------------------
' Zeit
'
Sub INT_10ms()
   Dim irqcnt As Integer
   If start =1 Then                       ' Nur wenn die Taste SW1 gedrückt wurde,
                                          ' wird dieser Programmteil abgearbeitet.
       cnt1 =cnt1+1                       ' 10ms Zähler wird um Eins erhöht.
       If cnt1 =100 Then                  ' Wenn 1 Sekunde vergangen ist, wird diese
                                          ' Schleife abgearbeitet.
           sekunde =sekunde+1             ' Sekundenzähler wird um Eins erhöht.
           If sekunde =60 Then            ' Wenn 60 Sekunden vergangen sind, wird
                                          ' diese Schleife abgearbeitet.
               minute =minute+1           ' Minutenzähler wird um Eins erhöht.

               If minute =60 Then         ' Wenn 60 Minuten vergangen sind, wird
                                          ' das Display auf 00:00:00 zurückgesetzt.
                   minute =0              ' Der Minutenzähler wird auf 0 gesetzt.
               End If
               sekunde=0                  ' Der Sekundenzähler wird auf 0 gesetzt.
           End If
           cnt1=0                         ' Der 10ms Zähler wird auf 0 gesetzt.
       End If
       Display_Buffer_Set()               ' Display Buffer wird aktualisiert
       ausgabe()                          ' Jede Sekunde wird die Zeit ausgegeben.
   End If
   tasten()                               ' Aufruf der Funktion zur Tastenabfrage.
   irqcnt=Irq_GetCount(INT_TIM2COMP)      ' Interrupt Request Counter
End Sub
```

Ausgabe

```
'-------------------------------------------------------------------------------
' Displayausgabe
'
Sub ausgabe()
   LCD_CursorPos(0x00)                    ' Cursor wird positioniert.
   LCD_WriteText(zeile1)                  ' Zeile1 wird geschrieben
End Sub

Sub ausgabe2()
   LCD_CursorPos(0x40)                    ' Cursor wird positioniert.
   LCD_WriteText(zeile1)                  ' Zeile2 wird geschrieben
End Sub

'-------------------------------------------------------------------------------
' Eingabewert in den Displaypuffer schreiben
'
Sub Display_Buffer_Set()
   Dim sep(2) As Char                             ' Array für Trennzeichen deklarieren.
   sep=":"                                        ' Wert wird dem Array zugewiesen.
   Str_WriteWord(minute,10,zeile1,0,2)            ' Minuten, Sekunden und Zehntel
   Str_Copy(zeile1,sep,STR_APPEND)                ' werden zu einem String zusammen-
   Str_WriteWord(sekunde,10,zeile1,STR_APPEND,2)  ' gefasst.
   Str_Copy(zeile1,sep,STR_APPEND)
   Str_WriteWord(cnt1,10,zeile1,STR_APPEND,2)
End Sub
```

Tasten

```
'-------------------------------------------------------------------------------
' Tastenabfrage
'
Sub tasten()
   prell =prell+1                  ' Diese Variable eleminiert das Tasterprellen.
   If Port_ReadBit(PORT_SW1)=0 And start=0 And prell>20 Then
       prell=0                     ' Wird der Taster SW1 gedrückt und die Zeit läuft
       start=1                     ' noch nicht läuft, so wird die Zeit gestartet.
   End If
   If Port_ReadBit(PORT_SW2)=0 And start=1 And prell>20 Then
                                   ' Wird der Taster SW2 gedrückt und die Zeit läuft
       prell=0                     ' bereits, wird die Zwischenzeit in der zweiten
       ausgabe2()                  ' Zeile ausgegeben.

   End If
   If Port_ReadBit(PORT_SW1)=0 And start=1 And prell>20 Then
       prell=0                     ' Wird der Taster SW1 gedürckt und die Zeit läuft
       start=0                     ' bereits, so wird die Zeit gestoppt.
   End If
   If Port_ReadBit(PORT_SW2)=0 And start=0 And prell>20 Then
                                   ' Wird der Taster SW2 gedrückt und die Zeit läuft
       prell=0                     ' nicht, so wird das Display zurückgesetzt.
       LCD_ClearLCD()              ' Display wird gelöscht.
       LCD_CursorOff()             ' Cursor wird ausgeschaltet.
       cnt1 = 0
       sekunde = 0
       minute = 0                  ' Variablen werden zurückgesetzt.
       Display_Buffer_Set()        ' Displaypuffer wird aktualisiert.
       ausgabe()                   ' Display wird geschrieben
   End If
End Sub
```

19.10 Gewächshausreglung

Über dieses Projekt können Sie Ihr Gewächshaus komplett regeln und überwachen. Über drei NTC-Sensoren wird die Temperatur überwacht. Überschreiten die Messwerte aller drei Sensoren bestimmte Werte, wird die Jalousie geschlossen. Ist die Sonneneinstrahlung zu hoch, wird die Jalousie ebenfalls geschlossen. Steigt die Luftfeuchtigkeit über einen bestimmten Wert, wird ein Lüfter eingeschaltet. Steigt die Luftfeuchtigkeit dennoch weiter an, wird ein zweiter Lüfter dazugeschaltet. Ist die Bodenfeuchte zu gering, wird eine Bewässerungspumpe eingeschaltet. Des Weiteren wird überwacht, ob die Temperatur an einem der drei NTC-Sensoren nicht unter einen bestimmten Wert fällt. Ist dies der Fall, wird die Heizung eingeschaltet. Über die RS232-Schnittstelle werden alle Schaltzustände ausgegeben. Über ein Terminal-Programm (z. B. HyperTerminal) können Sie auf diese Weise die aktuellen Schaltzustände auf Ihrem PC kontrollieren.

Stückliste:

Menge	Art.-Nr.	Artikel	oder Art.-Nr.	oder Artikel
1	198206	Mega32	198219	Mega128
1	198245	Board Mega32	198258	Board Mega128
3	182800	NTC-Sensor		
2	156584	Raumfeuchtefühler		
1	156518	Lichtfühler		
6	408280	10-kOhm-Widerstand		
2	198836	Relais-Modul		

Die Sensoren verbinden Sie bitte über einen Spannungsteiler mit den entsprechenden Eingängen (siehe Abb.12.15) Die Relaisplatinen verbinden Sie bitte wie in Kapitel 12.4 beschrieben mit den im Programm angegebenen Ports. Über ein RS232-Kabel können Sie die C-Control mit Ihrem PC verbinden. Das Terminalprogramm muss hierfür auf die Übertragungsrate 9600-8-N-1 eingestellt werden.
Mit dem HyperTerminal erhalten Sie folgende Anzeige:

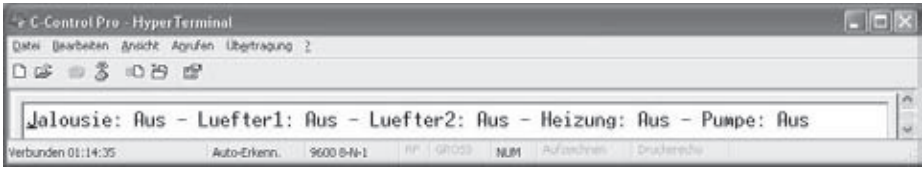


Abb. 19.1: HyperTerminal

Gewächshaus – Quellcode CC

Das Programm kann für den Mega32 und Mega128 verwendet werden. Für den Mega128 müssen Sie aber die Ein- und Ausgabeports im Programm noch entsprechend anpassen. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
// Regelung eines Gewächshauses
// Mega32
// erforderliche Library: IntFunc_Lib.cc,

// Analoger Eingang PA0: Temperatur1
// Analoger Eingang PA1: Temperatur2
// Analoger Eingang PA2: Temperatur3
// Analoger Eingang PA3: Helligkeitsmessung
// Analoger Eingang PA4: Luftfeuchtigkeitsmessung
// Analoger Eingang PA5: Bodenfeuchtigkeitsmessung

// Digitalausgang PC2: Jalousie
// Digitalausgang PC3: Lüfter1
// Digitalausgang PC4: Lüfter2
// Digitalausgang PC5: Heizung
// Digitalausgang PC6: Wasserpumpe

// Ist die Temperatur1, Temperatur2 und Temperatur3 oder die Helligkeit
// über einem bestimmten Wert wird die Jalousie geschlossen.
// Ist die Luftfeuchtigkeit über einem bestimmten Wert wird Lüfter1 eingeschaltet.
// Ist die Luftfeuchtigkeit über einem noch höheren Wert wird der Lüfter2 zuge-
// schaltet. Ist die Bodenfeuchtigkeit unter einem bestimmten Wert wird die
// Wasserpumpe eingeschaltet. Alle Schaltvorgänge werden als Textnachricht über
// die RS232 Schnittstelle gesendet. LD1 wird als Kontrollleuchte für die Heizung
// und LD2 als Kontrolleuchte für die Pumpe verwendet.

//-------------------------------------------------------------------------------
// Deklaration der globalen Variablen
//
int temp1;                                              // Variable Temperaturfühler 1
int temp2;                                              // Variable Temperaturfühler 2
int temp3;                                              // Variable Temperaturfühler 3
int lux;                                                // Variable Helligkeitssensor
int lhyd;                                               // Variable Luftfeuchtigkeit
int bhyd;                                               // Variable Bodenfeuchtigkeit
byte jalo, luft1, luft2, heiz, wasser;                  // Variablen für Textausgabe
char ausgabetext[100], text1[10], text2[10], text3[20];  // Text Arrays


//-------------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
   outputAktoren();                       // Aktoren vorbereiten
   rs232int();                            // RS232 Schnittstelle initialisieren
   while(true)                            // Endlosschleife wird gestartet.
   {
       inputSensor();                     // Sensorwerte werden eingelesen

      // Jalousie Regelung
      // Überschreitet die Temperatur an allen drei Sensoren einen bestimmten
      // Wert oder ist die Helligkeit zu groß wir diese Schleife abgearbeitet.
         if ((temp1>670 && temp2>670 && temp3>670) | lux>180)
```

```
{                       // Diese Wert müssen Sie natürlich an
                        // Ihre Sensoren und Gegebenheiten anpassen.
    Port_WriteBit(18,1);// Jalousie wird ausgefahren
    jalo=1;             // Variable für Textausgabe wird gesetzt.
}
else
{
    Port_WriteBit(18,0);// Jalousie wird einfahren
    jalo=0;             // Variable für Textausgabe wird gesetzt.
}

// Lüfter Regelung
// Überschreitet die Luftfeuchtigkeit einen bestimmten Wert wird der
// erste Lüfter eingeschaltet.
if (lhyd>250)
{                       // Diesen Wert müssen Sie natürlich an
                        // Ihren Sensor und Gegebenheiten anpassen.
    Port_WriteBit(19,1);// Lüfter1 wird einschalten.
    luft1=1;            // Variable für Textausgabe wird gesetzt.
}
else
{
    Port_WriteBit(19,0);// Lüfter1 wird ausschalten.
    luft1=0;            // Variable für Textausgabe wird gesetzt.
}
// Überschreitet die Luftfeuchtigkeit einen bestimmten Wert wird der
// zweite Lüfter zugeschaltet.
if (lhyd>350)
{                       // Diesen Wert müssen Sie natürlich an
                        // Ihren Sensor und Gegebenheiten anpassen.
    Port_WriteBit(20,1);// Lüfter2 wird einschalten.
    luft2=1;            // Variable für Textausgabe wird gesetzt.
}
else
{
    Port_WriteBit(20,0);// Lüfter2 wird ausschalten
    luft2=0;            // Variable für Textausgabe wird gesetzt.
}

// Heizung Regelung
// Unterschreitet die Temperatur an einem der drei Sensoren einen be-
// stimmten Wirt wird die Heizung eingeschaltet.
if (temp1<655 | temp2<655 | temp3<655)
{                       // Diese Wert müssen Sie natürlich an
                        // Ihre Sensoren und Gegebenheiten anpassen.
    Port_WriteBit(21,1);// Heizung wird eingeschaltet.

  Port_WriteBit(30,0);  // Kontroll LED1 wird eingeschaltet.
  heiz=1;               // Variable für Textausgabe wird gesetzt.
}
else
{
    Port_WriteBit(21,0);// Heizung wird ausgeschaltet.
    Port_WriteBit(30,1);// Kontroll LED1 wird ausgeschaltet.
    heiz=0;             // Variable für Textausgabe wird gesetzt.
}

// Bodenfeuchte Regelung
// Unterschreitet die Bodenfeuchte einen bestimmten Wert, wird die Wasser-
// pumpe eingeschaltet.
if (bhyd<177)
{                       // Diesen Wert müssen Sie natürlich an
                        // Ihren Sensor und Gegebenheiten anpassen.
    Port_WriteBit(22,1);// Pumpe wird eingeschaltet.
    Port_WriteBit(31,0);// Kontroll LED2 wird eingeschaltet
    wasser=1;           // Variable für Textausgabe wird gesetzt.
```

```
          }
          else
          {
              Port_WriteBit(22,0);// Pumpe wird ausgeschaltet.
              Port_WriteBit(31,1);// Kontroll LED2 wird ausgeschaltet.
              wasser=0;           // Variable für Textausgabe wird gesetzt.
          }
      AbsDelay(2000);             // Über diese Variable können Sie die
                                  // Aktualisierungzeit festlegen.
                                  // Momentan liegt diese bei 2 Sekunden.
      ausgabe();                  // Ausgabefunktion wird aufgerufen.
   }
}
```

Sensoren

```
//-------------------------------------------------------------------------------
// Initialisierung und Auslesen der A/D Eingänge
//
void inputSensor(void)
{
   ADC_Set(ADC_VREF_VCC,ADC0);            //Festlegung der Referenzspannung für ADC0
   temp 1=ADC_Read();                     //Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC1);            //Festlegung der Referenzspannung für ADC1
   temp2=ADC_Read();                      //Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC2);            //Festlegung der Referenzspannung für ADC2
   temp3=ADC_Read();                      //Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC3);            //Festlegung der Referenzspannung für ADC3
   lux=ADC_Read();                        //Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC4);            //Festlegung der Referenzspannung für ADC4
   lhyd=ADC_Read();                       //Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC5);            //Festlegung der Referenzspannung für ADC5
   bhyd=ADC_Read();                       //Auslesen des anliegenden Wertes
}
```

Ausgabe

```
//-------------------------------------------------------------------------------
// Initialisierung der Digitalausgänge
//
void outputAktoren(void)

{
   Port_DataDirBit(18,1);                 //PC-2 auf Ausgang (Jalousie)
   Port_DataDirBit(19,1);                 //PC-3 auf Ausgang (Lüfter1)
   Port_DataDirBit(20,1);                 //PC-4 auf Ausgang (Lüfter2)
   Port_DataDirBit(21,1);                 //PC-5 auf Ausgang (Heizung)
   Port_DataDirBit(22,1);                 //PC-6 auf Ausgang (Pumpe)
   Port_DataDirBit(30,1);                 //PD-6 auf Ausgang (LED1)
   Port_DataDirBit(31,1);                 //PD-7 auf Ausgang (LED2)
   Port_WriteBit(30,1);                   //LED1 aus
   Port_WriteBit(31,1);                   //LED2 aus
}

//-------------------------------------------------------------------------------
// Initialisierung der RS232 Schnittstelle
//
void rs232int(void)
{
   Serial_Init(0,SR_8BIT|SR_1STOP|SR_NO_PAR,SR_BD9600);
}

//-------------------------------------------------------------------------------
// Ausgabe der Textnachrichten
```

```
//
void ausgabe(void)
{
   ausgabetext = “Jalousie: “;                    // Ausgabetext wird zugewiesen.
   text1 = “Ein“;                                 // Schaltzustandstexte werden
   text2 = “Aus“;                                 // zugewiesen.
   if (jalo ==1)                                  // Ist die Varible gesetzt,
   {                                              // wird der Text Ein an den
      Str_Copy(ausgabetext,text1,STR_APPEND);     / String ausgabetext angehängt.
   }
   else                                           // Wenn nicht, wird aus an den
   {                                              // String ausgabetext angehängt.
      Str_Copy(ausgabetext,text2,STR_APPEND);
   }
   text3 = “ – Luefter1: “;                       // Ausgabetext wird zugewiesen
   Str_Copy(ausgabetext,text3,STR_APPEND);        // und den String ausgabetext
   if (luft1 ==1)                                 // angehängt.
   {                                              // Ist die Varible gesetzt,
      Str_Copy(ausgabetext,text1,STR_APPEND);     // wird der Text Ein an den
   }                                              // String ausgabetext angehängt.
   else                                           // Wenn nicht, wird aus an den
   {                                              // String ausgabetext angehängt.
      Str_Copy(ausgabetext,text2,STR_APPEND);
   }
   text3 = “ – Luefter2: “;                       // Ausgabetext wird zugewiesen
   Str_Copy(ausgabetext,text3,STR_APPEND);        // und den String ausgabetext
   if (luft2 ==1)                                 // angehängt.
   {                                              // Ist die Varible gesetzt,
      Str_Copy(ausgabetext,text1,STR_APPEND);     // wird der Text Ein an den
   }                                              // String ausgabetext angehängt.
   else                                           // Wenn nicht, wird aus an den
   {                                              // String ausgabetext angehängt.
      Str_Copy(ausgabetext,text2,STR_APPEND);
   }
   text3 = “ – Heizung: “;                        // Ausgabetext wird zugewiesen
   Str_Copy(ausgabetext,text3,STR_APPEND);        // und den String ausgabetext
   if (heiz ==1)                                  // angehängt.
   {                                              // Ist die Varible gesetzt,
      Str_Copy(ausgabetext,text1,STR_APPEND);     // wird der Text Ein an den
   }                                              // String ausgabetext angehängt.

   else                                           // Wenn nicht, wird aus an den
   {                                              // String ausgabetext angehängt.
      Str_Copy(ausgabetext,text2,STR_APPEND);
   }
   text3 = “ – Pumpe: “;                          // Ausgabetext wird zugewiesen
   Str_Copy(ausgabetext,text3,STR_APPEND);        // und den String ausgabetext
   if (wasser ==1)                                // angehängt.
   {                                              // Ist die Varible gesetzt,
      Str_Copy(ausgabetext,text1,STR_APPEND);     // wird der Text Ein an den
   }                                              // String ausgabetext angehängt.
   else                                           // Wenn nicht, wird aus an den
   {                                              // String ausgabetext angehängt.
      Str_Copy(ausgabetext,text2,STR_APPEND);
   }

   Serial_WriteText(0,ausgabetext);               // Der Text wird über die RS232-
   Serial_Write(0,0x0d);                          // Schnittstelle gesendet.
}
```

Gewächshaus – Quellcode CBasic

Das Programm kann für den Mega32 und Mega128 verwendet werden. Für den Mega128 müssen Sie aber die Ein- und Ausgabeports im Programm noch entsprechend anpassen. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
' Regelung eines Gewächshauses
' Mega32
' erforderliche Library: IntFunc_Lib.cc,

' Analoger Eingang PA0: Temperatur1
' Analoger Eingang PA1: Temperatur2
' Analoger Eingang PA2: Temperatur3
' Analoger Eingang PA3: Helligkeitsmessung
' Analoger Eingang PA4: Luftfeuchtigkeitsmessung
' Analoger Eingang PA5: Bodenfeuchtigkeitsmessung

' Digitalausgang PC2: Jalousie
' Digitalausgang PC3: Lüfter1
' Digitalausgang PC4: Lüfter2
' Digitalausgang PC5: Heizung
' Digitalausgang PC6: Wasserpumpe

' Ist die Temperatur1, Temperatur2 und Temperatur3 oder die Helligkeit
' über einem bestimmten Wert wird die Jalousie geschlossen.
' Ist die Luftfeuchtigkeit über einem bestimmten Wert wird Lüfter1 eingeschaltet.
' Ist die Luftfeuchtigkeit über einem noch höheren Wert wird der Lüfter2 zuge-
' schaltet. Ist die Bodenfeuchtigkeit unter einem bestimmten Wert wird die
' Wasserpumpe eingeschaltet. Alle Schaltvorgänge werden als Textnachricht über
' die RS232 Schnittstelle gesendet. LD1 wird als Kontrollleuchte für die Heizung
' und LD2 als Kontrolleuchte für die Pumpe verwendet.

'-------------------------------------------------------------------------------
' Deklaration der globalen Variablen
'
Dim temp1 As Integer                                    ' Variable Temperaturfühler 1
Dim temp2 As Integer                                    ' Variable Temperaturfühler 2
Dim temp3 As Integer                                    ' Variable Temperaturfühler 3
Dim lux As Integer                                      ' Variable Helligkeitssensor
Dim lhyd As Integer                                     ' Variable Luftfeuchtigkeit
Dim bhyd As Integer                                     ' Variable Bodenfeuchtigkeit
Dim jalo, luft1, luft2, heiz, wasser As Byte            ' Variablen für Textausgabe
Dim ausgabetext(100), text1(10), text2(10), text3(20) As Char   ' Text Arrays

'-------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
   outputAktoren()          ' Aktoren vorbereiten
   rs232int()               ' RS232 Schnittstelle initialisieren

   Do While(True)           ' Endlosschleife wird gestartet.
       inputSensor()        ' Sensorwerte werden eingelesen

       ' Jalousie Regelung
       ' Überschreitet die Temperatur an allen drei Sensoren einen bestimmten
       ' Wert oder ist die Helligkeit zu groß wir diese Schleife abgearbeitet.
       If (temp1>670 And temp2>670 And temp3>670) Or lux>180 Then
       ' Diese Wert müssen Sie natürlich an
       ' Ihre Sensoren und Gegebenheiten anpassen.
       Port_WriteBit(18,1)          ' Jalousie wird ausgefahren
       jalo=1                       ' Variable für Textausgabe wird gesetzt.
       Else
```

```
            Port_WriteBit(18,0) ' Jalousie wird einfahren
            jalo=0              ' Variable für Textausgabe wird gesetzt.
        End If

        ' Lüfter Regelung
        ' Überschreitet die Luftfeuchtigkeit einen bestimmten Wert wird der
        ' erste Lüfter eingeschaltet.
        If lhyd>250 Then
                                ' Diesen Wert müssen Sie natürlich an
                                ' Ihren Sensor und Gegebenheiten anpassen.
            Port_WriteBit(19,1) ' Lüfter1 wird einschalten.
            luft1=1             ' Variable für Textausgabe wird gesetzt.
        Else
            Port_WriteBit(19,0) ' Lüfter1 wird ausschalten.
            luft1=0             ' Variable für Textausgabe wird gesetzt.
        End If
        ' Überschreitet die Luftfeuchtigkeit einen bestimmten Wert wird der
        ' zweite Lüfter zugeschaltet.
        If lhyd>350 Then
                                ' Diesen Wert müssen Sie natürlich an
                                ' Ihren Sensor und Gegebenheiten anpassen.
            Port_WriteBit(20,1) ' Lüfter2 wird einschalten.
            luft2=1             ' Variable für Textausgabe wird gesetzt.
        Else
            Port_WriteBit(20,0) ' Lüfter2 wird ausschalten
            luft2=0             ' Variable für Textausgabe wird gesetzt.
        End If

        ' Heizung Regelung
        ' Unterschreitet die Temperatur an einem der drei Sensoren einen be-
        ' stimmten Wirt wird die Heizung eingeschaltet.
        If temp1<655 Or temp2<655 Or temp3<655 Then
                                ' Diese Wert müssen Sie natürlich an
                                ' Ihre Sensoren und Gegebenheiten
                                ' anpassen.
            Port_WriteBit(21,1) ' Heizung wird eingeschaltet.

            Port_WriteBit(30,0) ' Kontroll LED1 wird eingeschaltet.
            heiz=1              ' Variable für Textausgabe wird gesetzt.
        Else
            Port_WriteBit(21,0) ' Heizung wird ausgeschaltet.
            Port_WriteBit(30,1) ' Kontroll LED1 wird ausgeschaltet.
            heiz=0              ' Variable für Textausgabe wird gesetzt.
        End If

        ' Bodenfeuchte Regelung
        ' Unterschreitet die Bodenfeuchte einen bestimmten Wert, wird die Wasser-
        ' pumpe eingeschaltet.
        If bhyd<177 Then
                                ' Diesen Wert müssen Sie natürlich an
                                ' Ihren Sensor und Gegebenheiten anpassen.
            Port_WriteBit(22,1) ' Pumpe wird eingeschaltet.
            Port_WriteBit(31,0) ' Kontroll LED2 wird eingeschaltet
            wasser=1            ' Variable für Textausgabe wird gesetzt.
        Else
            Port_WriteBit(22,0) ' Pumpe wird ausgeschaltet.
            Port_WriteBit(31,1) ' Kontroll LED2 wird ausgeschaltet.
            wasser=0            ' Variable für Textausgabe wird gesetzt.
        End If
      AbsDelay(2000)            ' Über diese Variable können Sie die
                                ' Aktualisierungzeit festlegen.
                                ' Momentan liegt diese bei 2 Sekunden.
      ausgabe()                 ' Ausgabefunktion wird aufgerufen.
   End While
End Sub
```

Sensoren

```
'-------------------------------------------------------------------------------
' Initialisierung und Auslesen der A/D Eingänge
'
Sub inputSensor()
   ADC_Set(ADC_VREF_VCC,ADC0)             'Festlegung der Referenzspannung für ADC0
   temp1=ADC_Read()                       'Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC1)             'Festlegung der Referenzspannung für ADC1
   temp2=ADC_Read()                       'Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC2)             'Festlegung der Referenzspannung für ADC2
   temp3=ADC_Read()                       'Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC3)             'Festlegung der Referenzspannung für ADC3
   lux=ADC_Read()                         'Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC4)             'Festlegung der Referenzspannung für ADC4
   lhyd=ADC_Read()                        'Auslesen des anliegenden Wertes
   ADC_Set(ADC_VREF_VCC,ADC5)             'Festlegung der Referenzspannung für ADC5
   bhyd=ADC_Read()                        'Auslesen des anliegenden Wertes
End Sub
```

Ausgabe

```
'-------------------------------------------------------------------------------
' Initialisierung der Digitalausgänge
'
Sub outputAktoren()
   Port_DataDirBit(18,1)                  'PC-2 auf Ausgang (Jalousie)
   Port_DataDirBit(19,1)                  'PC-3 auf Ausgang (Lüfter1)
   Port_DataDirBit(20,1)                  'PC-4 auf Ausgang (Lüfter2)
   Port_DataDirBit(21,1)                  'PC-5 auf Ausgang (Heizung)
   Port_DataDirBit(22,1)                  'PC-6 auf Ausgang (Pumpe)
   Port_DataDirBit(30,1)                  'PD-6 auf Ausgang (LED1)
   Port_DataDirBit(31,1)                  'PD-7 auf Ausgang (LED2)
   Port_WriteBit(30,1)                    'LED1 aus
   Port_WriteBit(31,1)                    'LED2 aus
End Sub
'-------------------------------------------------------------------------------
' Initialisierung der RS232 Schnittstelle
'
Sub rs232int()
   Serial_Init(0,SR_8BIT Or SR_1STOP Or SR_NO_PAR,SR_BD9600)
End Sub
'-------------------------------------------------------------------------------
' Ausgabe der Textnachrichten
'
Sub ausgabe()
   ausgabetext = "Jalousie: "                ' Ausgabetext wird zugewiesen.
   text1 = "Ein"                             ' Schaltzustandstexte werden
   text2 = "Aus"                             ' zugewiesen.
   If jalo =1 Then                           ' Ist die Varible gesetzt,
      Str_Copy(ausgabetext,text1,STR_APPEND) ' wird der Text Ein an den
   Else                                      ' String ausgabetext angehängt.
      Str_Copy(ausgabetext,text2,STR_APPEND) ' Wenn nicht, wird aus an den
   End If                                    ' String ausgabetext angehängt.
   text3 = " – Luefter1: "                   ' Ausgabetext wird zugewiesen
   Str_Copy(ausgabetext,text3,STR_APPEND)    ' und den String ausgabetext
   If luft1 =1 Then                          ' angehängt.
                                             ' Ist die Varible gesetzt,
      Str_Copy(ausgabetext,text1,STR_APPEND) ' wird der Text Ein an den
   Else                                      ' String ausgabetext angehängt.
      Str_Copy(ausgabetext,text2,STR_APPEND) ' Wenn nicht, wird aus an den
   End If                                    ' String ausgabetext angehängt.
   text3 = " – Luefter2: "                   ' Ausgabetext wird zugewiesen
   Str_Copy(ausgabetext,text3,STR_APPEND)    ' und den String ausgabetext
   If luft2 =1 Then                          ' angehängt.
                                             ' Ist die Varible gesetzt,
```

```
      Str_Copy(ausgabetext,text1,STR_APPEND)      ' wird der Text Ein an den
   Else                                           ' String ausgabetext angehängt.
      Str_Copy(ausgabetext,text2,STR_APPEND)      ' Wenn nicht, wird aus an den
   End If                                         ' String ausgabetext angehängt.
   text3 = " - Heizung: "                         ' Ausgabetext wird zugewiesen
   Str_Copy(ausgabetext,text3,STR_APPEND)         ' und den String ausgabetext
   If heiz =1 Then                                ' angehängt.
                                                  ' Ist die Varible gesetzt,
      Str_Copy(ausgabetext,text1,STR_APPEND)      ' wird der Text Ein an den
   Else                                           ' String ausgabetext angehängt.
      Str_Copy(ausgabetext,text2,STR_APPEND)      ' Wenn nicht, wird aus an den
   End If                                         ' String ausgabetext angehängt.
   text3 = " - Pumpe: "                           ' Ausgabetext wird zugewiesen
   Str_Copy(ausgabetext,text3,STR_APPEND)         ' und den String ausgabetext
   If wasser =1 Then                              ' angehängt.
                                                  ' Ist die Varible gesetzt,
      Str_Copy(ausgabetext,text1,STR_APPEND)      ' wird der Text Ein an den
                                                  ' String ausgabetext angehängt.
   Else                                           ' Wenn nicht, wird aus an den
      Str_Copy(ausgabetext,text2,STR_APPEND)      ' String ausgabetext angehängt.
   End If

   Serial_WriteText(0,ausgabetext)                ' Der Text wird über die RS232-
   Serial_Write(0,0x0d)                           ' Schnittstelle gesendet.
End Sub
```

19.11 3-Kanal-DCF-Zeitschaltuhr

Das nachfolgende Programm stellt eine 3-Kanal-DCF-Zeitschaltuhr zur Verfügung. Die Uhrzeit und die Schaltzeiten werden auf dem 4x20-Display angezeigt. Über die angeschlossene DCF-Antenne wird die Uhrzeit synchronisiert. Ist mindestens eine der beiden Schaltbedingungen pro Relais erfüllt, wird dieses durch Aktivierung des entsprechenden Kanals eingeschaltet. Im Programm besteht die Möglichkeit, die Schaltzeiten bereits vorzubelegen. Des Weiteren kann mit Hilfe des Tasters SW1 die gewünschte Schaltzeit angewählt werden und mit dem Taster SW2 entsprechend verändert werden. Die aktuell änderbare Zeit wird mit dem Cursor angezeigt. Es ist zu empfehlen, dass nach dem Start des Programms erst die DCF-Synchronisation abgewartet wird. Dies erkennen Sie daran, dass die Uhrzeit von 12:00 auf die aktuelle Zeit umgestellt wird. Sollten Sie bereits während der Synchronisation die Taster SW1 und SW2 betätigen, kann dies zu Fehlern bei der Synchronisation führen, bzw. sie wird sich entsprechend verlängern.

Stückliste:

Menge	Art.-Nr.	Artikel	oder Art.-Nr.	oder Artikel
1	198206	Mega32	198219	Mega128
1	198245	Board Mega32	198258	Board Mega128
1	187275	LCD Display 4x20		
1	641138	DCF-Modul		
2	198836	Relais-Modul		

Das DCF-Modul verbinden Sie bitte wie in Kapitel 12.1 beschrieben mit dem Board des Mega32 bzw. Mega128. In Kapitel 12.2 ist beschrieben, welche LCD-Module geeignet sind und wie diese angeschlossen werden müssen. Das Relais-Modul verbinden Sie bitte mit den Ports B.0 bis B.2 bzw. A.0 bis A.2. Die genaue Vorgehensweise entnehmen Sie bitte Kapitel 12.4.

Zeitschaltuhr – Quellcode CC

Das Programm kann für den Mega32 und Mega128 unverändert verwendet werden. Die Portzuweisungen werden im Programm entsprechend der verwendeten Module automatisch vorgenommen. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
// 3-Kanal DCF Zeitschaltuhr
// Mega32:
// Eingang: DCF PortD.7
// Ausgang: PortB.0, PortB.1, PortB.2
// Mega128:
// Eingang: DCF PortF.0
// Ausgang: PortA.0, PortA.1, PortA.2
// Das SRAM muss mit JP7 deaktiviert werden.
// erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc, DCF_Lib.cc

// Über die DCF Antenne wird die aktuelle Uhrzeit festgestellt. Diese dient als
// Zeitbasis für die Zeitschaltuhr. Mit dem Taster SW1 können Sie die einzelnen
// Schaltzeiten anwählen. Wird danach der Taster SW2 gedrückt ändert sich die
// entsprechende Uhrzeit. Für jeden Kanal stehen zwei Schaltzeiten zur Verfügung.


#ifdef MEGA32                           // Wird ein Mega32 verwendet wird der
#define aPort PortB                     // Ausgabeport auf diese Weise deklariert.
#define bit 8                           // Erste Portbitadresse wird zugewiesen
#endif

#ifdef MEGA128                          // Wird ein Mega128 verwendet wird der
#define aPort PortA                     // Ausgabeport auf diese Weise deklariert.
#define bit 0                           // Erste Portbitadresse wird zugewiesen
#endif

// globale Variablendeklaration
char zeile1[21], zeile2[21], zeile3[21], zeile4[21];
word zeit[15];                  // Arrays für die Zeichenausgabe werden deklariert
int schalter1, schalter2, j;

//---------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
   zeit[1]=12; zeit[2]=13;         // Variablen zur Vorbelegung der Schaltzeiten
   zeit[3]=15; zeit[4]=18;         // des Relais eins.
   zeit[5]=0; zeit[6]=10;          // Variablen zur Vorbelegung der Schaltzeiten
   zeit[7]=11; zeit[8]=12;         // des Relais zwei.
   zeit[9]=23; zeit[10]=08;        // Variablen zur Vorbelegung der Schaltzeiten
   zeit[11]=10; zeit[12]=12;       // des Relais drei.
   schalter1=1;                    // default Wert wird festgelegt

   LCD_Init();                     // Display initialisieren
```

```
   LCD_ClearLCD();                     // Display löschen
   Port_DataDir(aPort,0xFF);           // alle Ausgänge des Ausgabeports auf Ausgang
   Port_DataDirBit(PORT_SW1,PORT_IN);  // Ports der Schalter werden auf
   Port_DataDirBit(PORT_SW2,PORT_IN);  // Eingang vorbereitet.
   Port_WriteBit(PORT_SW1,PORT_OFF);   // Pullup Widerstand eingeschaltet
   Port_WriteBit(PORT_SW2,PORT_OFF);   // Pullup Widerstand eingeschaltet

   Time_Init();                        // Funktionsaufruf zur Festlegung
                                       // der Startzeit.
   DCF_INIT();                         // Initialisierung des DCF Betriebes.
   DCF_START();                        // DCF-Erfassung wird gestartet.
   Irq_SetVect(INT_TIM2COMP,INT_10ms); // Interrupt Service Routine definieren
                                       // Timer2 erzeugt einen 10ms Interrupt.
   Schaltzeiten();                     // Schaltzeiten werden aktualisiert
   Write2();                           // Schaltzeiten werden ausgegeben.
   Displaytext();                      // Funktionsaufruf zur Ausgabe der Displaymaske
   while (true);                       // Endlosschleife
}
```

DCF-Uhr

```
//-------------------------------------------------------------------------------
// Festlegung des Synchronisationszeitpunktes
//
void INT_10ms(void)
{
   int irqcnt;
   RTC(0x01,0x15);                        // DCF Update um 01:15
   DCF_PULS();                            // DCF_MODE=1 Puls suchen
   DCF_SYNC();                            // DCF_MODE=2 Synchronisation
   DCF_FRAME();                           // DCF_MODE=3 Datenaufnahme
   irqcnt=Irq_GetCount(INT_TIM2COMP);     // Interrupt Request Counter
}

//-------------------------------------------------------------------------------
// Festlegung der Startzeit im unsynchronisiertem Zustand
//
void Time_Init(void)
{
   cnt1=0;                                // cnt1 zählt im 10ms Takt
   Sekunde=0;                             // Startzeit: 0 Sekunden
   Minute=0;                              // Startzeit: 0 Minuten
   Stunde=12;                             // Startzeit: 12 Stunden
}

//-------------------------------------------------------------------------------
// Zeitabgleich
//
void RTC(byte U_Stunde, byte U_Minute)
{
   cnt1++;                                // 10ms Zähler um Eins erhöhen
   if (cnt1==100)                         // Wenn diese Schleife 100 mal durchlaufen
   {                                      // wurde ist eine Sekunde vergangen.
       Relais();                          // Relaisausgänge aktualisiert.
       Taster();                          // Tasten werden abgefragt
       Set();                             // Display Buffer wird aktualisiert.
       Write();                           // Jede Sekunde wird das Display ausgeben.
       Cursor();                          // Cursorposition wird aktualisiert.
       Sekunde++;                         // Sekunden Zähler um Eins erhöhen
       if (Sekunde==60)                   // Wenn diese Schleife 60 mal durchlaufen
       {                                  // wurde ist eine Sekunde vergangen.
           Sekunde=0;                     // Der Sekundenzähler wird auf 0 gesetzt.
           if (Minute==U_Minute && Stunde==U_Stunde)
           {                              // Zeitabgleich bei eingestellter Uhrzeit
               DCF_START();               // DCF-Erfassung wird gestartet.
           }
```

```
            Minute++;                       // Der Minutenzähler wird um Eins erhöht.
            if (Minute==60)                 // Wenn diese Schleife 60 mal durchlaufen
            {                               // wurde ist eine Minute vergangen.
                Minute=0;                   // Der Minutenzähler wird auf 0 gesetzt.
                Stunde++;                   // Der Stundenzähler wird um Eins erhöht.
                if (Stunde==24) Stunde=0;
            }                               // Ist die Stunde 24 erreicht wird der
        }                                   // Stundenzähler auf 0 zurückgesetzt.
        cnt1=0;                             // 10ms Zähler wird auf 0 gesetzt.
    }
}
```

Display

```
//-------------------------------------------------------------------------------
// Uhrzeit in den Display Buffer schreiben
//
void Set(void)
{
   char sep[16];                                  // Array für die Trennzeichen
   sep=":";                                       // Wertzuweisung
   Str_WriteWord(Stunde,10,zeile1,0,2);           // Die Variablen Stunde, Minute und
   Str_Copy(zeile1,sep,STR_APPEND);               // Sekunde werden zu einem String
   Str_WriteWord(Minute,10,zeile1,STR_APPEND,2);  // zusammengestellt.

   sep=" Zeitschaltuhr";                          // Die erste Zeile wird ver-
   Str_Copy(zeile2,sep,STR_APPEND);               // vollständigt.
}

//-------------------------------------------------------------------------------
// Schaltzeiten in den Display Buffer schreiben
//
void Schaltzeiten(void)
{
   char sep2[3];                                  // Array für die Trennzeichen
   sep2=" ";                                      // Wertzuweisung
   char sep[3];                                   // Array für die Trennzeichen
   sep="-";                                       // Wertzuweisung
   Str_WriteWord(zeit[1],10,zeile3,0,2);          // Die Zeilen drei und vier
   Str_Copy(zeile3,sep,STR_APPEND);               // werden zusammengestellt.
   Str_WriteWord(zeit[2],10,zeile3,STR_APPEND,2);
   Str_Copy(zeile3,sep2,STR_APPEND);
   Str_WriteWord(zeit[5],10,zeile3,STR_APPEND,2);
   Str_Copy(zeile3,sep,STR_APPEND);
   Str_WriteWord(zeit[6],10,zeile3,STR_APPEND,2);
   Str_Copy(zeile3,sep2,STR_APPEND);
   Str_WriteWord(zeit[9],10,zeile3,STR_APPEND,2);
   Str_Copy(zeile3,sep,STR_APPEND);
   Str_WriteWord(zeit[10],10,zeile3,STR_APPEND,2);
   Str_WriteWord(zeit[3],10,zeile4,0,2);
   Str_Copy(zeile4,sep,STR_APPEND);
   Str_WriteWord(zeit[4],10,zeile4,STR_APPEND,2);
   Str_Copy(zeile4,sep2,STR_APPEND);
   Str_WriteWord(zeit[7],10,zeile4,STR_APPEND,2);
   Str_Copy(zeile4,sep,STR_APPEND);
   Str_WriteWord(zeit[8],10,zeile4,STR_APPEND,2);
   Str_Copy(zeile4,sep2,STR_APPEND);
   Str_WriteWord(zeit[11],10,zeile4,STR_APPEND,2);
   Str_Copy(zeile4,sep,STR_APPEND);
   Str_WriteWord(zeit[12],10,zeile4,STR_APPEND,2);
}

//-------------------------------------------------------------------------------
// Displayausgabe
//
void Displaytext(void)
```

```
{
   zeile2 =“OUT-1 OUT-2 OUT-3“;           // Der Text für die erste und für
   LCD_CursorPos(0x40);                   // LCD Cursor positionieren
   LCD_WriteText(zeile2);                 // Die erste Zeile wird ausgegeben.
}

void Write(void)
{
   LCD_CursorPos(0x00);                   // LCD Cursor positionieren
   LCD_WriteText(zeile1);                 // Zeile eins wird ausgegeben
}

void Write2(void)
{
   LCD_CursorPos(0x14);                   // LCD Cursor positionieren
   LCD_WriteText(zeile3);                 // Zeile drei wird ausgegeben
   LCD_CursorPos(0x54);                   // LCD Cursor positionieren
   LCD_WriteText(zeile4);                 // Zeile vier wird ausgegeben
}

void Cursor(void)
{
   switch(schalter1)                      // Auswahl der Varianten
   {                                      // abhängig vom Zählerstand
       case 1: LCD_CursorPos(0x15); break;// schalter1. Die verschiedenen
       case 2: LCD_CursorPos(0x18); break;// Cursorpositionen werden in
       case 3: LCD_CursorPos(0x55); break;// Abhängigkeit von der
       case 4: LCD_CursorPos(0x58); break;// Betätigungsanzahl des
       case 5: LCD_CursorPos(0x1c); break;// Schalters SW1 zugewiesen.
       case 6: LCD_CursorPos(0x1f); break;
       case 7: LCD_CursorPos(0x5c); break;
       case 8: LCD_CursorPos(0x5f); break;
       case 9: LCD_CursorPos(0x23); break;
       case 10: LCD_CursorPos(0x26); break;
       case 11: LCD_CursorPos(0x63); break;
       case 12: LCD_CursorPos(0x66); break;
   }
}
```

Taster

```
//---------------------------------------------------------------------------
// Abfrage der Taster
//
void Taster(void)
{
   j++;
   if (j>1)
   {
       if (Port_ReadBit(PORT_SW1)==PORT_ON)
       {                                  // Ist Schalter 1 gedrückt wird der
           schalter1++;                   // Zähler schalter1 um Eins erhöht.
           if (schalter1>12) schalter1=1;
       }                                  // Ist die letzte Position erreicht,
                                          // wird auf die erste gewechselt.
       if (Port_ReadBit(PORT_SW2)==PORT_ON)
       {                                  // Ist Schalter 1 gedrückt wird der
           zeit[schalter1]++;             // Uhrzeit um eine Stunde erhöht.
           if (zeit[schalter1]>23) zeit[schalter1]=0;
           Schaltzeiten();                // Displaypuffer wird aktualisiert
           Write2();                      // Display wird ausgegeben
                                          // Ist die Stunde 24 errecht, wird auf
       }                                  // Null gewechselt.
       j=0;
   }
}
```

Relais-Steuerung

```
//-------------------------------------------------------------------------------
// Ansteuerung der Relais
//
void Relais(void)
{
   int k,i,p;                     // Variablendeklaration
   int out[5];                    // Array für den Schaltzustand
   p= 0;                          // Portzähler
   for (i=0; i<9; i=i+4)          // Schleife zur Aktualisierung jedes Relais
   {
       for (k=1; k<4; k=k+2)      // Schleife zur Abfrage der zwei Schaltzeiten
       {                          // jedes einzelnen Relais.

       if (zeit[k+i]> zeit[k+i+1]) // Ist die Einschaltzeit größer als die
           {                      // Ausschaltzeit, wird diese Schleife
               if (Stunde >= zeit[k+i]|Stunde < zeit[k+i+1])    // abgearbeitet.
               {                  // Ist die aktuelle Zeit größer oder gleich
                   out[k]=1;      // der Einschaltzeit oder kleiner als die
               }                  // Ausschaltzeit wird das Relais eingeschaltet.
               else
               {
                   out[k]=0;      // Wenn nicht, wird es ausgeschaltet.
               }
           }
           else
           {
               if (Stunde >= zeit[k+i]&&Stunde < zeit[k+i+1])
               {                  // Ist die aktuelle Zeit größer oder gleich
                   out[k]=1;      // der Einschaltzeit und kleiner als die
               }                  // Ausschaltzeit wird das Relais eingeschaltet.
               else
               {
                   out[k]=0;      // Wenn nicht, wird es ausgeschaltet.
               }
           }
       }
       if (out[1]==1|out[3]==1)   // Ist Schaltbedingung eins oder zwei
       {                          // erfüllt, wird das Relais eingeschaltet
           Port_WriteBit(bit+p,1);
       }
       else
       {
           Port_WriteBit(bit+p,0); // Wenn nicht, wird es ausgeschaltet.
       }
       p++;                       // Portzähler wird um Eins erhöht.
   }
}
```

Zeitschaltuhr – Quellcode CBasic

Das Programm kann für den Mega32 und Mega128 unverändert verwendet werden. Die Portzuweisungen werden im Programm entsprechend der verwendeten Module automatisch vorgenommen. Es ist zu empfehlen, die einzelnen Programmteile modular aufzubauen, da dadurch das Projekt übersichtlicher wird und einzelne Teile leichter exportiert werden können.

```
' 3-Kanal DCF Zeitschaltuhr
' Mega32:
' Eingang: DCF PortD.7
' Ausgang: PortB.0, PortB.1, PortB.2
' Mega128:
' Eingang: DCF PortF.0
' Ausgang: PortA.0, PortA.1, PortA.2
' Das SRAM muss mit JP7 deaktiviert werden.
' erforderliche Library: IntFunc_Lib.cc, LCD_Lib.cc, DCF_Lib.cc

' Über die DCF Antenne wird die aktuelle Uhrzeit festgestellt. Diese dient als
' Zeitbasis für die Zeitschaltuhr. Mit dem Taster SW1 können Sie die einzelnen
' Schaltzeiten anwählen. Wird danach der Taster SW2 gedrückt ändert sich die
' entsprechende Uhrzeit. Für jeden Kanal stehen zwei Schaltzeiten zur Verfügung.

#ifdef MEGA32                           ' Wird ein Mega32 verwendet wird der
#define aPort PortB                     ' Ausgabeport auf diese Weise deklariert.
#define bit 8                           ' Erste Portbitadresse wird zugewiesen
#endif

#ifdef MEGA128                          ' Wird ein Mega128 verwendet wird der
#define aPort PortA                     ' Ausgabeport auf diese Weise deklariert.
#define bit 0                           ' Erste Portbitadresse wird zugewiesen
#endif

' globale Variablendeklaration
Dim zeile1(21), zeile2(21), zeile3(21), zeile4(21) As Char
Dim zeit(15) As Word              ' Arrays für die Zeichenausgabe werden deklariert
Dim schalter1, schalter2, j As Integer

'-------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
   zeit(1)=12 : zeit(2)=13            ' Variablen zur Vorbelegung der Schaltzeiten
   zeit(3)=15 : zeit(4)=18            ' des Relais eins.
   zeit(5)=0 : zeit(6)=10             ' Variablen zur Vorbelegung der Schaltzeiten
   zeit(7)=11 : zeit(8)=12            ' des Relais zwei.
   zeit(9)=23 : zeit(10)=08           ' Variablen zur Vorbelegung der Schaltzeiten
   zeit(11)=10: zeit(12)=12           ' des Relais drei.
   schalter1=1                        ' default Wert wird festgelegt

   LCD_Init()                         ' Display initialisieren
   LCD_ClearLCD()                     ' Display löschen
   Port_DataDir(aPort,0xFF)           ' alle Ausgänge des Ausgabeports auf Ausgang
   Port_DataDirBit(PORT_SW1,PORT_IN)  ' Ports der Schalter werden auf
   Port_DataDirBit(PORT_SW2,PORT_IN)  ' Eingang vorbereitet.
   Port_WriteBit(PORT_SW1,PORT_OFF)   ' Pullup Widerstand eingeschaltet
   Port_WriteBit(PORT_SW2,PORT_OFF)   ' Pullup Widerstand eingeschaltet

   Time_Init()                        ' Funktionsaufruf zur Festlegung
                                      ' der Startzeit.
   DCF_INIT()                         ' Initialisierung des DCF Betriebes.
   DCF_START()                        ' DCF-Erfassung wird gestartet.
   Irq_SetVect(INT_TIM2COMP,INT_10ms) ' Interrupt Service Routine definieren
                                      ' Timer2 erzeugt einen 10ms Interrupt.
   Schaltzeiten()                     ' Schaltzeiten werden aktualisiert
   Write2()                           ' Schaltzeiten werden ausgegeben
   Displaytext()                      ' Funktionsaufruf zur Ausgabe der Displaymaske
   Do While (True)                    ' Endlosschleife
   End While
End Sub
```

DCF-Uhr

```
'-------------------------------------------------------------------------------
' Festlegung des Synchronisationszeitpunktes
'
Sub INT_10ms()
   Dim irqcnt As Integer
   RTC(0x01,0x15)                      ' DCF Update um 01:15
   DCF_PULS()                          ' DCF_MODE=1 Puls suchen
   DCF_SYNC()                          ' DCF_MODE=2 Synchronisation
   DCF_FRAME()                         ' DCF_MODE=3 Datenaufnahme

 irqcnt=Irq_GetCount(INT_TIM2COMP)     ' Interrupt Request Counter
End Sub

'-------------------------------------------------------------------------------
' Festlegung der Startzeit im unsynchronisiertem Zustand
'
Sub Time_Init()
   cnt1=0                              ' cnt1 zählt im 10ms Takt
   Sekunde=0                           ' Startzeit: 0 Sekunden
   Minute=0                            ' Startzeit: 0 Minuten
   Stunde=12                           ' Startzeit: 12 Stunden
End Sub

'-------------------------------------------------------------------------------
' Zeitabgleich
'
Sub RTC(U_Stunde As Byte, U_Minute As Byte)
   cnt1 = cnt1+1                       ' 10ms Zähler um Eins erhöhen
   If cnt1=100 Then                    ' Wenn diese Schleife 100 mal durchlaufen
                                       ' wurde ist eine Sekunde vergangen.
       Relais()                        ' Relaisausgänge aktualisiert.
       Taster()                        ' Tasten werden abgefragt
       Set()                           ' Display Buffer wird aktualisiert.
       Write()                         ' Jede Sekunde wird das Display ausgeben.
       Cursor()                        ' Cursorposition wird aktualisiert.
       Sekunde =Sekunde+1              ' Sekunden Zähler um Eins erhöhen
       If Sekunde=60 Then              ' Wenn diese Schleife 60 mal durchlaufen
                                       ' wurde ist eine Sekunde vergangen.
           Sekunde=0                   ' Der Sekundenzähler wird auf 0 gesetzt.
           If Minute=U_Minute And Stunde=U_Stunde Then
               DCF_START()             ' Zeitabgleich bei eingestellter Uhrzeit
           End If                      ' DCF-Erfassung wird gestartet.
           Minute =Minute+1            ' Der Minutenzähler wird um Eins erhöht.
           If Minute=60 Then           ' Wenn diese Schleife 60 mal durchlaufen
                                       ' wurde ist eine Minute vergangen.
               Minute=0                ' Der Minutenzähler wird auf 0 gesetzt.
               Stunde =Stunde+1        ' Der Stundenzähler wird um Eins erhöht.
               If Stunde=24 Then       ' Ist die Stunde 24 erreicht wird der
                   Stunde=0            ' Stundenzähler auf 0 zurückgesetzt.
               End If
           End If
       End If
       cnt1=0                          ' 10ms Zähler wird auf 0 gesetzt.
   End If
End Sub
```

Display

```
'-------------------------------------------------------------------------------
' Uhrzeit in den Display Buffer schreiben
'
Sub Set()
   Dim sep(16) As Char                 ' Array für die Trennzeichen
   sep=„:“                             ' Wertzuweisung
```

```
   Str_WriteWord(Stunde,10,zeile1,0,2) ' Die Variablen Stunde, Minute und
   Str_Copy(zeile1,sep,STR_APPEND)     ' Sekunde werden zu einem String
   Str_WriteWord(Minute,10,zeile1,STR_APPEND,2)      ' zusammengestellt.
   sep=" Zeitschaltuhr"                ' Die erste Zeile wird ver-
   Str_Copy(zeile2,sep,STR_APPEND)     ' vollständigt.
End Sub

'-------------------------------------------------------------------------------
' Schaltzeiten in den Display Buffer schreiben
'
Sub Schaltzeiten()
   Dim sep2(3) As Char                 ' Array für die Trennzeichen
   Dim sep(3) As Char                  ' Array für die Trennzeichen
   sep2=" "                            ' Wertzuweisung
   sep="-"                             ' Wertzuweisung
   Str_WriteWord(zeit(1),10,zeile3,0,2)' Die Zeilen drei und vier
   Str_Copy(zeile3,sep,STR_APPEND)     ' werden zusammengestellt.
   Str_WriteWord(zeit(2),10,zeile3,STR_APPEND,2)
   Str_Copy(zeile3,sep2,STR_APPEND)
   Str_WriteWord(zeit(5),10,zeile3,STR_APPEND,2)
   Str_Copy(zeile3,sep,STR_APPEND)
   Str_WriteWord(zeit(6),10,zeile3,STR_APPEND,2)
   Str_Copy(zeile3,sep2,STR_APPEND)
   Str_WriteWord(zeit(9),10,zeile3,STR_APPEND,2)
   Str_Copy(zeile3,sep,STR_APPEND)
   Str_WriteWord(zeit(10),10,zeile3,STR_APPEND,2)
   Str_WriteWord(zeit(3),10,zeile4,0,2)
   Str_Copy(zeile4,sep,STR_APPEND)
   Str_WriteWord(zeit(4),10,zeile4,STR_APPEND,2)
   Str_Copy(zeile4,sep2,STR_APPEND)
   Str_WriteWord(zeit(7),10,zeile4,STR_APPEND,2)
   Str_Copy(zeile4,sep,STR_APPEND)
   Str_WriteWord(zeit(8),10,zeile4,STR_APPEND,2)
   Str_Copy(zeile4,sep2,STR_APPEND)
   Str_WriteWord(zeit(11),10,zeile4,STR_APPEND,2)
   Str_Copy(zeile4,sep,STR_APPEND)
   Str_WriteWord(zeit(12),10,zeile4,STR_APPEND,2)
End Sub

'-------------------------------------------------------------------------------
' Displayausgabe
'
Sub Displaytext()
   zeile2 ="OUT-1 OUT-2 OUT-3"         ' Der Text für die erste und für
   LCD_CursorPos(0x40)                 ' LCD Cursor positionieren
   LCD_WriteText(zeile2)               ' Die erste Zeile wird ausgegeben.
End Sub

Sub Write()
   LCD_CursorPos(0x00)                 ' LCD Cursor positionieren
   LCD_WriteText(zeile1)               ' Zeile eins wird ausgegeben
End Sub

Sub Write2()
   LCD_CursorPos(0x14)                 ' LCD Cursor positionieren
   LCD_WriteText(zeile3)               ' Zeile drei wird ausgegeben
   LCD_CursorPos(0x54)                 ' LCD Cursor positionieren
   LCD_WriteText(zeile4)               ' Zeile vier wird ausgegeben
End Sub

Sub Cursor()
   Select Case schalter1               ' Auswahl der Varianten
       Case 1: LCD_CursorPos(0x15)     ' abhängig vom Zählerstand
       Case 2: LCD_CursorPos(0x18)     ' schalter1. Die verschiedenen
       Case 3: LCD_CursorPos(0x55)     ' Cursorpositionen werden in
       Case 4: LCD_CursorPos(0x58)     ' Abhängigkeit von der
       Case 5: LCD_CursorPos(0x1c)     ' Betätigungsanzahl des
```

```
       Case 6: LCD_CursorPos(0x1f)     ' Schalters SW1 zugewiesen.
       Case 7: LCD_CursorPos(0x5c)
       Case 8: LCD_CursorPos(0x5f)
       Case 9: LCD_CursorPos(0x23)
       Case 10: LCD_CursorPos(0x26)
       Case 11: LCD_CursorPos(0x63)
       Case 12: LCD_CursorPos(0x66)
   End Case
End Sub
```

Taster

```
'-------------------------------------------------------------------------------
' Abfrage der Taster
'
Sub Taster()
   j =j+1
   If j>1 Then
       If Port_ReadBit(PORT_SW1)=PORT_ON Then
                                          ' Ist Schalter 1 gedrückt wird der
           schalter1 =schalter1+1         ' Zähler schalter1 um Eins erhöht.
           If schalter1>12 Then
               schalter1=1                ' Ist die letzte Position erreicht,
           End If                         ' wird auf die erste gewechselt.
       End If
       If Port_ReadBit(PORT_SW2)=PORT_ON Then
                                          ' Ist Schalter 1 gedrückt wird der
           zeit(schalter1) =zeit(schalter1)+1 ' Uhrzeit um eine Stunde erhöht.
           If zeit(schalter1)>23 Then
               zeit(schalter1)=0
           End If
           Schaltzeiten()                 ' Displaypuffer wird aktualisiert
           Write2()                       ' Display wird ausgegeben
                                          ' Ist die Stunde 24 errecht, wird auf
       End If                             ' Null gewechselt.
      j=0
   End If
End Sub
```

Relais-Steuerung

```
'-------------------------------------------------------------------------------
' Ansteuerung der Relais
'
Sub Relais()
   Dim k,i,p As Integer               ' Variablendeklaration
   Dim out(5) As Integer              ' Array für den Schaltzustand
   p =0                               ' Portzähler
   For i=0 To 8 Step 4                ' Schleife zur Aktualisierung jedes Relais
       For k=1 To 3 Step 2            ' Schleife zur Abfrage der zwei Schaltzeiten
                                      ' jedes einzelnen Relais.
           If zeit(k+i)> zeit(k+i+1) Then     ' Ist die Einschaltzeit größer als die
                                      ' Ausschaltzeit, wird diese Schleife
               If Stunde >= zeit(k+i)Or Stunde < zeit(k+i+1) Then   ' abgearbeitet.
                                      ' Ist die aktuelle Zeit größer oder gleich
                   out(k)=1           ' der Einschaltzeit oder kleiner als die
                                      ' Ausschaltzeit wird das Relais eingeschaltet.
               Else
                   out(k)=0           ' Wenn nicht, wird es ausgeschaltet.
               End If

               Else
               If Stunde >= zeit(k+i)And Stunde < zeit(k+i+1) Then
                                      ' Ist die aktuelle Zeit größer oder gleich
                   out(k)=1           ' der Einschaltzeit und kleiner als die
```

```
                                    ' Ausschaltzeit wird das Relais eingeschaltet.
            Else
                out(k)=0            ' Wenn nicht, wird es ausgeschaltet.
            End If
        End If
    Next
    If out(1)=1 Or out(3)=1 Then    ' Ist Schaltbedingung eins oder zwei
                                    ' erfüllt, wird das Relais eingeschaltet
        Port_WriteBit(bit+p,1)
    Else
        Port_WriteBit(bit+p,0)      ' Wenn nicht, wird es ausgeschaltet.
    End If
    p =p+1                          ' Portzähler wird um Eins erhöht.
  Next
End Sub
```

19.12 Ein-/Ausschaltverzögerung

Mit diesem Programm wird eine Ein-/Ausschaltverzögerung realisiert. Über den Port A.1 bzw. F.1 wird bestimmt, ob das Programm als Ein- oder Ausschaltverzögerung arbeiten soll. Liegen 5 V an diesem Port, erfolgt eine Einschaltverzögerung, liegen 0 V an diesem Port, eine Ausschaltverzögerung. Über das an dem Port A.0 bzw. F.0 angeschlossene Potentiometer kann die Verzögerungszeit eingestellt werden. Sobald nun ein Aktivierungssignal über das Port A.2 bzw. F.2 erfasst wird, läuft die eingestellte Verzögerungszeit ab. Das am Ausgang B.0 bzw. A.0 wird in Abhängigkeit von der Verzögerungszeit und des eingestellten Schaltverhaltens ein- bzw. ausgeschaltet.

Stückliste:

Menge	Art.-Nr.	Artikel	oder Art.-Nr.	oder Artikel
1	198206	Mega32	198219	Mega128
1	198245	Board Mega32	198258	Board Mega128
2	198836	Relais-Modul		
1	445673	22-kOhm-Poti lin		

Ein Ende des Potentiometerwiderstandes muss mit 5 V, das andere Ende mit GND (0 V) verbunden werden. Der Schleifer muss nun an das ADC-Port A.0 bzw. F.0 angeschlossen werden. Die Relaisplatinen verbinden Sie bitte mit dem Port B bzw. Port A. (siehe Kapitel 12.4). Für eine sichere Funktion muss des Weiteren das SDRAM des Mega128-Application-Boards über JP7 ausgeschaltet werden.

Ein-/Ausschaltverzögerung – Quellcode CBasic
Das Programm kann für den Mega32 und Mega128 unverändert verwendet werden. Die Portzuweisungen werden im Programm entsprechend der verwendeten Module automatisch vorgenommen.

```
' Ein-, Ausschaltverzögerung
' Mega32
' Eingang: PortA.0, PortA.1, PortA.2
' Ausgang: Port B.0
' Mega128
' Eingang: PortF.0, PortF.1, PortF.2
' Ausgang: Port A.0
' erforderliche Library: IntFunc_Lib.cc

' Das Programm kann für den Mega32 und Mega128 verwendet werden.
' Über das A/D-Ports A.0 bzw. F.0 wird die Zeit über ein angeschlossenes Poti
' eigestellt. Liegt an A.1 bzw. F.1 eine Spannung von 5 V an, so arbeitet das
' Programm als Einschaltverzögerung. Wird dieses Port auf 0 V gelegt, wird eine
' Ausschaltverzögerung durchgeführt. Über das Port B.0 bzw. A.0 erfolgt die
' Ausgabe des Schaltzustandes. Die Verzögerung wird durch 5 V an A.2 bzw. F.2
' aktiviert.
' Beim Mega 128 muss das SDRAM über JP7 deaktiviert werden.

#ifdef MEGA32                      ' Wird ein Mega32 verwendet werden die Ports
#define ePort 1                    ' auf diese Weise deklariert
#define sPort 2
#define aPort 8
#endif

#ifdef MEGA128                     ' Wird ein Mega128 verwendet werden die Ports
#define ePort 41                   ' auf diese Weise deklariert
#define sPort 42
#define aPort 0
#endif

' globale Variablendeklaration
Dim poti, signal, faktor As Integer

'--------------------------------------------------------------------------------
' Hauptprogramm
'
Sub main()
   faktor =10                                ' Über diesen Faktor kann die maximale
                                             ' Verzögerung bestimmt werden.
                                             ' 10 bedeutet 10 Sekunden
   Port_DataDirBit(aPort,PORT_OUT)           ' Port wird auf Ausgabe vorbereitet.
   Port_DataDirBit(ePort,PORT_IN)            ' Port wird auf Eingabe vorbereitet.
   Port_DataDirBit(sPort,PORT_IN)            ' Port wird auf Eingabe vorbereitet.

   Do While (True)
       ADC_Set(ADC_VREF_VCC,ADC0)            ' Die Referenzspannung und der Messport
                                             ' werden ausgewählt.
       poti = ADC_Read()                     ' Die der anliegenden Spannung ent-
                                             ' sprechenden Zahl wird in die Variable
                                             ' poti gespeichert.
       If Port_ReadBit(ePort) Then           ' Liegt an dem ePort eine Spannung von
                                             ' 5V arbeitet das Programm als Einschalt-
                                             ' verzögerung.
           If Port_ReadBit(sPort) Then       ' Liegen 5V an dem sPort startet die
                                             ' Einschaltverzögerung in Abhängigkeit
               AbsDelay(poti*faktor)         ' zur Potistellung.
               Port_WriteBit(aPort,PORT_OFF) ' Relais wird eingeschaltet.
           Else
                                             ' Liegt am sPort keine Spannung mehr an,
               Port_WriteBit(aPort,PORT_ON)  ' wird das Relais ausgeschaltet.
           End If
       Else                                  ' Liegt an dem ePort eine Spannung von
                                             ' 0V arbeitet das Programm als Ausschalt-
                                             ' verzögerung.
```

```
          If Port_ReadBit(sPort) Then        ' Liegen 5V an dem sPort startet die
                                             ' Ausschaltverzögerung in Abhängigkeit
              Port_WriteBit(aPort,PORT_OFF)  ' zur Potistellung.
              AbsDelay(poti*faktor)          ' Ausschaltzeit läuft ab.
          Else
                                             ' Liegt am sPort keine Spannung an,
              Port_WriteBit(aPort,PORT_ON)   ' wird das Relais ausgeschaltet.
          End If
       End If
   End While
End Sub
```

Ein-/Ausschaltverzögerung – Quellcode CC

Das Programm kann für den Mega32 und Mega128 unverändert verwendet werden. Die Portzuweisungen werden im Programm entsprechend der verwendeten Module automatisch vorgenommen.

```
// Ein-, Ausschaltverzögerung
// Mega32
// Eingang: PortA.0, PortA.1, PortA.2
// Ausgang: Port B.0
// Mega128
// Eingang: PortF.0, PortF.1, PortF.2
// Ausgang: Port A.0
// erforderliche Library: IntFunc_Lib.cc

// Das Programm kann für den Mega32 und Mega128 verwendet werden.
// Über das A/D-Ports A.0 bzw. F.0 wird die Zeit über ein angeschlossenes Poti
// eigestellt. Liegt an A.1 bzw. F.1 eine Spannung von 5 V an, so arbeitet das
// Programm als Einschaltverzögerung. Wird dieses Port auf 0 V gelegt, wird eine
// Ausschaltverzögerung durchgeführt. Über das Port B.0 bzw. A.0 erfolgt die
// Ausgabe des Schaltzustandes. Die Verzögerung wird durch 5 V an A.2 bzw. F.2
// aktiviert.
// Beim Mega 128 muss das SDRAM über JP7 deaktiviert werden.

#ifdef MEGA32                          // Wird ein Mega32 verwendet werden die Ports
#define ePort 1                        // auf diese Weise deklariert
#define sPort 2
#define aPort 8
#endif

#ifdef MEGA128                         // Wird ein Mega128 verwendet werden die Ports
#define ePort 41                       // auf diese Weise deklariert
#define sPort 42
#define aPort 0

#endif

// globale Variablendeklaration
int poti, signal, faktor;

//---------------------------------------------------------------------------
// Hauptprogramm
//
void main(void)
{
   faktor =10;                         // Über diesen Faktor kann die maximale
                                       // Verzögerung bestimmt werden.
                                       // 10 bedeutet 10 Sekunden
   Port_DataDirBit(aPort,PORT_OUT);    // Port wird auf Ausgabe vorbereitet.
   Port_DataDirBit(ePort,PORT_IN);     // Port wird auf Eingabe vorbereitet.
   Port_DataDirBit(sPort,PORT_IN);           // Port wird auf Eingabe vorbereitet.
```

```
   while (true)
   {
       ADC_Set(ADC_VREF_VCC,ADC0);            // Die Referenzspannung und der Messport
                                              // werden ausgewählt.
       poti = ADC_Read();                     // Die der anliegenden Spannung ent-
                                              // sprechenden Zahl wird in die Variable
                                              // poti gespeichert.
       if (Port_ReadBit(ePort))               // Liegt an dem ePort eine Spannung von
       {                                      // 5V arbeitet das Programm als Ein-
                                              // schaltverzögerung.
           if (Port_ReadBit(sPort))           // Liegen 5V an dem sPort startet die
           {                                  // Einschaltverzögerung in Abhängigkeit
               AbsDelay(poti*faktor);         // zur Potistellung.
              Port_WriteBit(aPort,PORT_OFF);  // Relais wird eingeschaltet.
           }
           else
           {                                  // Liegt am sPort keine Spannung mehr an,
               Port_WriteBit(aPort,PORT_ON);  // wird das Relais ausgeschaltet.
           }
       }
       else                                   // Liegt an dem ePort eine Spannung von
       {                                      // 0V arbeitet das Programm als Aus-
                                              // schaltverzögerung.
           if (Port_ReadBit(sPort))           // Liegen 5V an dem sPort startet die
           {                                  // Ausschaltverzögerung in Abhängigkeit
               Port_WriteBit(aPort,PORT_OFF); // zur Potistellung.
               AbsDelay(poti*faktor);         // Ausschaltzeit läuft ab.
           }
           else
           {                                  // Liegt am sPort keine Spannung an,
               Port_WriteBit(aPort,PORT_ON);  // wird das Relais ausgeschaltet.
           }
       }
   }
}
```

20 Der Bytecode-Interpreter

Dieses Kapitel erklärt, welche Bytecode-Instruktionen zur Verfügung stehen und wie sie arbeiten. An vielen Beispielen wird erklärt, wie Variablenzugriffe, Kontrollstrukturen und andere Sprachelemente in Bytecode umgesetzt werden. Im Anhang des Buches gibt es dann eine vollständige Auflistung aller Bytecodes mit ihrer Funktionsweise. Natürlich kann diese Aufzählung nur den aktuellen Stand wiedergeben und spätere Erweiterungen nicht berücksichtigen.

Wichtig: Dieses Kapitel ist für den Experten gedacht, der mehr über die interne Arbeitsweise des Interpreters erfahren möchte. Um diesen Bereich zu verstehen, muss man absolut sattelfest in CompactC oder BASIC sein. Zusätzlich ist es hilfreich, wenn man schon Erfahrung mit stack-basierten Interpretern gesammelt hat.

20.1 Die Speicherbereiche im Interpreter

Der Interpreter benutzt bei der Abarbeitung der Bytecodes drei verschiedene Speicherbereiche:

- Der Bereich für globale Variablen.
- Der Programm-Stack (für jeden Thread einmal vorhanden)
- Der Arithmetik-Stack (für jeden Thread einmal vorhanden)

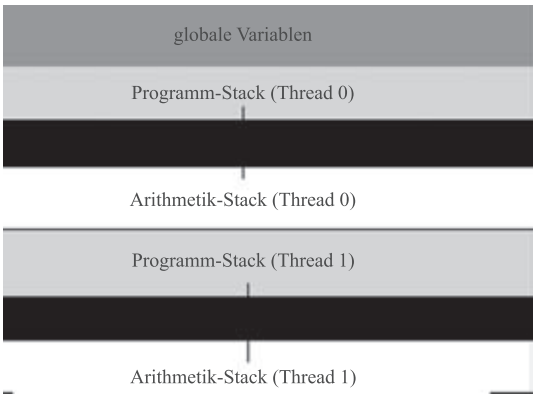


Abb. 20.1: Interpreter-Speicher

Im Programm-Stack werden lokale Variablen und Parameter von Funktionsaufrufen gespeichert. Er beginnt am oberen Ende des für den Thread reservierten Speichers, und beginnt nach unten zu wachsen. Die Position, ab der der Speicher zur Verfügung steht, ist der Stackpointer des Programm-Stacks. Wird Platz auf dem Programm-Stack benötigt, wird der Stackpointer um die Anzahl der benötigten Bytes dekrementiert. Bei der Freigabe von Speicher auf dem Programm-Stack, erhöht sich der Stackpointer wieder um den freien Bereich.

In jeder Funktion wird am Anfang Platz für lokale Variablen auf dem Programm-Stack angelegt und dabei der freie Stack-Speicher verkleinert. Am Ende der Funktion erfolgt die Freigabe des Speichers für die lokalen Variablen.

Im Falle eines Funktionsaufrufes werden die übergebenen Parameter auf dem Programm-Stack abgelegt, sodass die aufgerufene Funktion auf diese Werte zugreifen kann. Danach wird ein Sprung zur Funktion ausgeführt und die Rücksprungadresse auf dem Programm-Stack gesichert.

Bei der Rückkehr aus der Zielfunktion laufen die beschriebenen Aktionen rückwärts ab: Die Return-Anweisung holt sich die Rücksprungadresse vom Stack und springt zurück in die aufrufende Funktion. Dort wird der Stackpointer um den Betrag erhöht, den die Aufrufparameter auf dem Programm-Stack an Platz benötigen.

Der zweite Stack-Bereich, den jeder Thread besitzt, ist der Arithmetik-Stack. Dieser Stack wird benutzt um arithmetische Berechnungen durchzuführen. Die Funktionsweise ist ähnlich wie bei einem Taschenrechner mit UPN („umgekehrte polnische Notation“).

20.2 Die Arbeitsweise des Arithmetik-Stacks

Alle Einträge auf dem Arithmetik-Stack sind immer 4 Bytes lang. D. h., ein Stack-Eintrag kann alle möglichen Datentypen, vom Character bis zur Floating-Point-Zahl, aufnehmen. Auf den Stack können einzelne Werte geladen und davon wieder heruntergeholt werden. Aber das Wichtigste sind die arithmetischen Operationen, die auf dem letzten Stack-Eintrag oder den letzten beiden Stack-Einträgen arbeiten. Um bei dem Bild der UPN-Rechnungen zu bleiben, wird hier der letzte Stack-Eintrag als X und der vorletzte Eintrag als Y bezeichnet.

Möchte man z. B. ~(5+7*13) berechnen („~“ ist die Bitinvertierung aus CompactC), so sieht das auf dem Arithmetik-Stack so aus:

Man lädt zuerst die Werte 5,13,7 (in dieser Reihenfolge) auf den Stack.

X	7
Y	13
Z	5

Danach folgt die Multiplikation, die zwei Operatoren benötigt. Solche binären Operationen arbeiten immer mit den Einträgen X und Y. Der Stack wird nach unten verschoben und das Ergebnis wird nach X geschrieben.

X	91
Y	5

Man beachte: Vor dem Verschieben des Stacks war X das Y-Register. Diese Verschiebung, auch manchmal „Stacklift“ genannt, hat zur Folge, dass aus Z der Y-Eintrag wurde, und aus Y der Eintrag X.

Jetzt wird die Addition (auch eine binäre Operation auf X und Y) ausgeführt:

X	96

Zum Schluss noch die Bitinvertierung mit 8 Bit. Operationen wie die Bitinvertierung, die nur auf einem Eintrag arbeiten, heißen unär und beeinflussen nur X.

X	159

20.3 Beispiel: Zuweisung

Für alle wichtigen Variablenzugriffe, Funktionsaufrufe und Kontrollstrukturen folgen nun Übersetzungen von CompactC nach Bytecode.

```
    | byte a;

    | void main(void)
    | {
05: DecSP 2
    | int b;
    | a=5;
08: LoadImm8 5
10: StoreTopAdr8 1
    | b=~(a+7*13);
13: LoadTopAdr8 1
16: LoadImm8 91
18: Add16Sig
19: Inv16
20: StoreStRel16 2
    | }
23: IncSP_8Bit 2
```

In diesem Beispiel beginnen alle Original-CompactC-Zeilen mit einem „|“, und alle Bytecodes haben am Anfang eine Adresse (Zahl + „:“).

Man sieht an Adresse 05, dass mit dem Befehl `DecSP 2` zwei Byte Platz auf dem Programm-Stack geschaffen werden. Dieser Bereich ist für die lokale Variable `int b`. Der Befehl `DecSP` steht für „Decrement Stack Pointer“.

Am Ende der Funktion (Adresse 23) wird dieser Platz wieder freigegeben. Das Kommando `IncSP_8Bit` bedeutet „Increment Stack Pointer 8Bit“. Dieser Befehl bekommt ein 8-Bit-Argument, um wie viel der Stackpointer zu erhöhen ist. Es existiert auch die Variante `IncSP_16Bit`, die nach dem Kommando-Bytecode ein 16-Bit-Argument erhält.

```
05: DecSP 2
...
23: IncSP_8Bit 2
```

Der Bytecode `LoadImm8` („Load Immediate 8Bit“) lädt einen direkten Wert auf den Arithmetik-Stack. Dieser 8-Bit-Wert steht direkt hinter dem Bytecode. Der Wert 5 muss jetzt in der globalen 8-Bit-Variable gespeichert werden. Dies geschieht durch `StoreTopAdr8`. Diese Anweisung speichert ein Byte vom Arithmetik-Stack in den globalen Speicherbereich. Das Argument 1 ist eine Adresse im globalen Speicherbereich. Gleichzeitig entfernt `StoreTopAdr8` den Wert 5 vom Arithmetik-Stack.

```
    | a=5;
08: LoadImm8 5
10: StoreTopAdr8 1
```

Der Befehl `LoadTopAdr8` ist das Gegenstück zu `StoreTopAdr8`, und lädt einen 8-Bit-Wert (die Variable a) aus dem globalen Speicher auf den Arithmetik-Stack. Der Compiler hat schon 7*13=91 berechnet, und lädt den Wert als zweites auf den Arithmetik-Stack. Die Operationen `Add16Sig` („Addition 16 Bit Signed“) und `Inv16` („Invert 16 Bit“) vollenden die Berechnung. Warum werden an dieser Stelle auch 16-Bit-Befehle benutzt? Dies liegt an der Beteiligung von `int b`, was eine vorzeichenbehaftete („signed“) 16-Bit-Variable ist. Wenn an einer Berechnung verschiedene Datentypen beteiligt sind, dann hat immer die Variable mit der größeren Genauigkeit Vorrang. In diesem Fall bedeutet dies, dass ab einem gewissen Punkt mit 16 Bit und Vorzeichen gerechnet wird. An Adresse 20 wird das Ergebnis vom Arithmetik-Stack in die lokale Variable b gespeichert. Der Bytecode `StoreStRel16` („Store Stack Relative 16 Bit“) speichert einen 16-Bit-Wert auf dem Programm-Stack ab.

```
    | b=~(a+7*13);
13: LoadTopAdr8 1
16: LoadImm8 91
18: Add16Sig
19: Inv16
20: StoreStRel16 2
```

20.4 Beispiel: Funktionsaufruf

Im nächsten Beispiel ein Programm mit Funktionsaufruf:

```
    | char func1(char a, int b, float c)
    | {
    | AbsDelay(b);
05: LoadStRel16 7
08: WriteSP2
09: GenFunc 0
11: IncSP_8Bit 2
    | return(a);
13: LoadStRel8 9
16: Ret
    | }

    | void main(void)
    | {
    | func1(1,2,3);
17: LoadImm8 1
19: WriteSP1
20: LoadImm8 2
22: WriteSP2
23: LoadImm8 3
25: ToFloat 2
27: WriteSP4
28: Call 5
31: IncSP_8Bit 7
33: Drop 1
    | }
```

Die Funktion `func1()` hat selbst keine lokalen Variablen. Deshalb wird am Anfang auch kein Speicher auf dem Programm-Stack alloziert. Der Befehl `LoadStRel16` („Load Stack Relative 16 Bit“) lädt einen 16-Bit-Datenwert aus dem Programmstack in den Arithmetik-Stack. Dies ist der Parameter `b` der von `main()` übergeben wurde. Da jetzt ein Funktionsaufruf erfolgt, wird `b` gleich wieder auf den Programmstack geschrieben. Das Kommando `WriteSP2` schreibt den 16-Bit-Wert aus dem Arithmetik-Stack in den Programmstack. Gleichzeitig wird der Wert aus dem Arithmetik-Stack entfernt.

```
    | AbsDelay(b);
05: LoadStRel16 7
08: WriteSP2
09: GenFunc 0
11: IncSP_8Bit 2
```

An Adresse 19 erfolgt mit dem Befehl `GenFunc 0` der Aufruf von `AbsDelay()`. Da 2 Byte für die Parameterübergabe reserviert wurden, müssen diese auch wieder mit `IncSP_8Bit 2` freigegeben werden.

Alle internen Funktionen des Interpreters haben eine Funktionsnummer. Diese Funktionsnummer wird in der Datei „IntFunc_Lib.cc“ mit dem Funktionsnamen in Verbindung gebracht. Einträge in der „IntFunc_Lib.cc“ sehen folgendermaßen aus:

```
void AbsDelay $opc (0x00)(word val);
void Port_DataDir $opc(0x01)(byte port,byte val);
void Port_Write $opc(0x02)(byte port,byte val);
```

Ein Aufruf von `Port_DataDir()` erzeugt demnach den Bytecode `GenFunc 1`.

Die Funktion `func1()` gibt den Wert von `a` wieder an den Aufrufer zurück. Rückgabewerte werden auf den Arithmetik-Stack zurückgegeben. Es muss also nur der Wert von `a` mit `LoadStRel8` aus dem Programm-Stack geladen werden. An Adresse 16 erfolgt mit `Ret` der Rücksprung zur aufrufenden Funktion. Der Bytecode `Ret` nimmt dabei die Rücksprungadresse vom Programm-Stack und springt zum Aufrufer.

```
    | return(a);
13: LoadStRel8 9
16: Ret
```

Beim Aufruf von `func1()` in `main()` werden jeweils die Parameter `1,2,3` mit `LoadImm8` auf den Arithmetik-Stack geladen, und danach als Übergabeparameter auf den Programm-Stack gespeichert. Der zweite Aufrufparameter ist in `func1()` als `int b` deklariert. Hätte man hier einen Wert gewählt, der mit `LoadImm8` nicht mehr geladen werden kann (z. B. 1000), dann hätte an Adresse 20 der Befehl `LoadImm16` gestanden.

Etwas Besonderes, ist auch bei dem dritten Parameter zu erkennen. Die Zahlenerkennung im Compiler arbeitet selbstständig und hat beim Wert 3 im Funktionsaufruf `func1(1,2,3)` eine Integerzahl erkannt. Hätte als dritter Parameter 3.0 gestanden, und nicht 3, wäre direkt an Adresse 23 mit `LoadImm32` eine 32-Bit-Floating-Point-Zahl erzeugt worden. Nun aber wird bei der weiteren Analyse offenbar, dass `func1()` als dritten Parameter einen Floating-Point-Wert benötigt. Daher wird an Adresse 25 die Integerzahl in eine Fließkommazahl verwandelt.

```
    | func1(1,2,3);
17: LoadImm8 1
19: WriteSP1
20: LoadImm8 2
22: WriteSP2
23: LoadImm8 3
25: ToFloat 2
27: WriteSP4
```

In Adresse 28 erfolgt der Sprung in den Bytecode von Funktion `func1()`. Der Befehl `Call` springt an eine bestimmte Adresse, legt aber vorher die aktuelle Adresse als Rücksprungadresse auf den Programm-Stack. Da zur Übergabe 7 Byte als Parameter auf den Programm-Stack gespeichert wurden, wird der Speicher an Adresse 31 wieder freigegeben. Das Kommando `Drop` entfernt Einträge aus dem Arithmetik-Stack. Da `func1()` einen Wert als Rückgabeparameter auf dem Arithmetik-Stack angelegt hat, und `main()` diesen Wert gar nicht benutzt, muss er vom Arithmetik-Stack entfernt werden.

```
28: Call 5
31: IncSP_8Bit 7
33: Drop 1
```

20.5 Beispiel: if-Anweisung

Wie sehen nun Kontrollstrukturen wie die CompactC-`if`-Bedingung aus?

```
    | void main(void)
    | {
    | int a,b;
05: DecSP 4
    | a=5;
08: LoadImm8 5
10: StoreStRel16 2
    | if(a>10)
13: LoadStRel16 2
16: LoadImm8 10
18: Gr16BitSig
19: BranchFalse 30
    | b=1;
22: LoadImm8 1
24: StoreStRel16 0
27: Goto 35
    | else b=2;
30: LoadImm8 2
32: StoreStRel16 0
    | }
35: IncSP_8Bit 4
```

Es wird Platz auf dem Programm-Stack für die lokalen Variablen geschaffen (4 Byte) und der Wert 5 der Variablen `a` zugewiesen. Dies ist schon aus den anderen Beispielen bekannt.

```
    | int a,b;
05: DecSP 4
    | a=5;
08: LoadImm8 5
10: StoreStRel16 2
```

Um die if-Bedingung auszuwerten, werden die Variable a und der Wert 10 auf den Arithmetik-Stack geladen. Das Kommando `Gr16BitSig` („Greater than 16 Bit Signed“) vergleicht die beiden Operanden auf dem Arithmetik-Stack auf die „größer“ Bedingung, und speichert das Ergebnis „True“ (Wert 1) oder „False“ (Wert 0) in X. Ist die Bedingung nicht erfüllt (auf dem Arithmetik-Stack steht in X eine Null), wird mit `BranchFalse` in den `else`-Zweig der CompactC-`if`-Anweisung gesprungen.

```
    | if(a>10)
13: LoadStRel16 2
16: LoadImm8 10
18: Gr16BitSig
19: BranchFalse 30
```

Kommt man an diese Stelle, wurde der `BranchFalse` nicht ausgeführt, die Bedingung war demnach True. Nachdem `b=1` durchgeführt wurde, wird mit `Goto` hinter die `if`-Anweisung gesprungen. Springt man nicht, würde zusätzlich auch der `else`-Zweig der `if`-Anweisung (ab Adresse 30) abgearbeitet.

```
    | b=1;
22: LoadImm8 1
24: StoreStRel16 0
27: Goto 35
```

Dies ist der `else`-Zweig der `if`-Anweisung:

```
    | else b=2;
30: LoadImm8 2
32: StoreStRel16 0
```

Am Ende des Programms wieder die Freigabe des Speichers für die lokalen Variablen:

```
    | }
35: IncSP_8Bit 4
```

20.6 Beispiel: For-Schleife und Array-Zugriff

Als nächstes Übersetzungsbeispiel ein CompactC-Programm mit `for`-Schleife und mehrdimensionalem Array-Zugriff.

```
    | int a[5][5];

    | void main(void)
    | {
    |int i;
05: DecSp 2
    | for(i=0;i<5;i++)
08: LoadImm8 0
10: StoreStRel16 0
13: LoadStRel16 0
16: LoadImm8 5
18: Lt16BitSig
19: BranchFalse 52
    | {
    | a[i][3]=i;
22: LoadStRel16 0
25: LoadImm8 10
27: Mul16Unsig
28: LoadImm8 3
30: LoadImm8 2
32: Mul16Unsig
33: Add16Sig
34: LoadAdrTopRel 50
37: Add16Sig
38: LoadStRel16 0
41: StoreAdr16Xt
    | }
42: LoadAdrStRel 0
45: AddAdr16SigPst 1
47: Drop 1
49: Goto 13
    | }
52: IncSP_8Bit 2
```

Am Anfang, wie in den anderen Beispielen auch, die Reservierung des Speichers für die lokalen Variablen:

```
    |int i;
05: DecSp 2
```

Nun folgt die Abarbeitung der `for`-Schleife. Als Erstes wird in den Adressen 8 bis 10, die Variable `i` nach Null initialisiert. Es folgt die Abfrage der Bedingung der `for`-

Schleife. Die Variable `i` und der Wert 5 werden auf den Arithmetik-Stack geladen (Adresse 13 bis 16), und mit dem Operator `Lt16BitSig` („Less than 16Bit Signed“) verglichen. Ist die Bedingung nicht erfüllt, wird direkt mit `BranchFalse` die `for`-Schleife verlassen.

```
    | for(i=0;i<5;i++)
08: LoadImm8 0
10: StoreStRel16 0
13: LoadStRel16 0
16: LoadImm8 5
18: Lt16BitSig
19: BranchFalse 52
```

Um den mehrdimensionalen Array-Zugriff zu verstehen, muss man die Arithmetik hinter dem Zugriff erkennen. Das Integer-Array `a` hat 25 Felder. Es ist bei C-Compilern üblich (wie auch bei CompactC), dass bei im Speicher aufeinander folgenden Feldern der letzte Index sich am schnellsten verändert. D. h., im Speicher folgt nach `a[0][0]` das Feld `a[0][1]`, bzw. nach dem Feld `a[0][4]` das Feld `a[1][0]`. Möchte man die Adresse von `a[i][j]` ausrechnen, nimmt man die Anfangsadresse und addiert i*5+j. Man multipliziert zusätzlich alles mit 2, da alle Felder vom Typ Integer (= 2 Byte) sind. Diese Berechnung findet sich auch im Bytecode wieder. Die Variable `i` wird durch `LoadStRel16` geladen und mit 10 multipliziert (Adressen 22–27). Danach lädt man den Index 3 und multipliziert ihn mit 2 (= Anzahl Bytes pro Feld). Die beiden Ergebnisse werden addiert (Adresse 33). Um die korrekte Adresse des gewünschten Feldes zu erhalten, muss jetzt noch die Anfangsadresse des Arrays addiert werden. Der Befehl `LoadAdrTopRel` lädt die absolute Adresse auf den Arithmetik-Stack, die relativ zum Anfang des globalen Speicherbereichs ist. In Adresse 38 wird nochmals die Variable `i` geladen, und mit `StoreAdr16XT`, an die berechnete Feldadresse gespeichert.

```
    | a[i][3]=i;
22: LoadStRel16 0
25: LoadImm8 10
27: Mul16Unsig
28: LoadImm8 3
30: LoadImm8 2
32: Mul16Unsig
33: Add16Sig
34: LoadAdrTopRel 50
37: Add16Sig
38: LoadStRel16 0
41: StoreAdr16Xt
```

Am Ende der `for`-Schleife wird die Anweisung `i++` ausgeführt. Wie im Kapitel über die Optimierung von CompactC schon dargelegt, existieren spezielle Bytecode-Befehle, um die Abarbeitung von Inkrement- und Dekrement-Operatoren zu beschleunigen. Es wird die Adresse der Variablen `i` mit Hilfe von `LoadAdrStRel` auf den Arithmetik-Stack geladen, und der Befehl `AddAdr16SigPst 1` lädt den Inhalt der Adresse auf den Arithmetik-Stack und erhöht danach den Inhalt der Variablen um eins. Da die Anweisung `i++` nicht nur eine Variable erhöht, sondern auch deren Inhalt

zurückgibt (wie z. B. bei `x=i++`), steht der Inhalt von `i` auf dem Arithmetik-Stack. Da dieser Wert nicht benutzt wird, entfernt der Bytecode `Drop 1`, den letzten Eintrag vom Arithmetik-Stack. Es folgt noch der Sprung an Adresse 13, wo erneut die Bedingung (`i<5`) der `for`-Schleife getestet wird.

```
    | }
42: LoadAdrStRel 0
45: AddAdr16SigPst 1
47: Drop 1
49: Goto 13
```

20.7 Beispiel: Switch-Anweisung

Der Code, der für `switch`-Anweisungen generiert wird, ist sehr effizient, da für diesen Zweck extra spezielle Bytecodes implementiert wurden (siehe Kapitel 15: „Optimierung von CompactC“).

```
    | void main(void)
    | {
    |int b;
05: DecSp 2
    | switch(5)
08: LoadImm8 5
10: Switch 46
    | {
    | case 0:
    | b=0;
13: LoadImm8 0
15: StoreStRel 0
    | break;
18: Goto 42
    | case 3:
    | b=4;
21: LoadImm8 4
23: StoreStRel 0
    | break;
26: Goto 42
    | case 5:
    | b=5;
29: LoadImm8 5
31: StoreStRel 0
    | break;
34: Goto 42
    | default:
    | b=1;
37: LoadImm8 0
39: StoreStRel 0
    | }
    | }
42: IncSP_8Bit 2
```

Der Wert (5) für die `switch`-Anweisung wird auf den Arithmetik-Stack geladen und danach der `switch`-Bytecode ausgeführt. Das Argument 46 bezieht sich bei diesem Bytecode nicht auf eine RAM-Adresse, sondern auf eine Sprungtabelle im Flashspeicher. Die Sprungtabelle wird vom Compiler generiert und direkt mit dem Programm

abgespeichert. Der `switch`-Bytecode nimmt das X aus dem Arithmetik-Stack und sucht in der Sprungtabelle nach diesem Wert. Wird der Wert gefunden, ist eine Programm-Adresse zugeordnet, die der Interpreter dann direkt anspringt.

Ist der Wert nicht in der Tabelle, dann erfolgt der Sprung zum Bytecode hinter der `default:`-Anweisung.

```
    | switch(5)
08: LoadImm8 5
10: Switch 46
```

Die Befehle in jedem `case`-Fall sind hier einfach gehalten und speichern einen konstanten Wert in der Variable `b`. Der `break`-Befehl am Ende jedes `case`-Falles wird vom Compiler als `Goto`-Bytecode generiert, der an das Ende der `switch`-Anweisung springt.

```
    | case 0:
    | b=0;
13: LoadImm8 0
15: StoreStRel 0
    | break;
18: Goto 42
```

21 Anhang – Bytecode-Übersicht

In diesem Anhang findet sich eine Liste und Beschreibung aller Bytecodes, die im aktuellen Compiler verwendet werden. Ein Bytecode besteht immer aus einer 8-Bit-Bytecodenummer und einem nachfolgenden Parameter. Die Anzahl der Bytes für den Parameter ist variabel und hängt von dem verwendeten Bytecode ab.

21.1 Bytecode-Übersicht

Bytecode-Name	**Nr.**	**Parameter-Länge**	**Beschreibung**
Ignore	0	0	Interne Verwendung im Compiler
Label	1	0	Interne Verwendung im Compiler
Stop	2	0	Programmabbruch Bytecode-Interpreter
NOP	3	0	Keine Operation (No OPeration)
Line	4	0	Benutzt der Debugger zur Überprüfung, ob an dieser Adresse ein Breakpoint gesetzt wurde
Goto	5	2	Springe zu Adresse; die Adresse ist als 16-Bit-Parameter gegeben
Call	6	2	Speichere aktuelle Adresse +3 auf dem Programm-Stack, dann springe zu Adresse; die Adresse ist als 16-Bit-Parameter gegeben
Ret	7	0	Nehme Adresse vom Programm-Stack und springe dorthin
DecSP	8	2	Dekrementiere Stackpointer (Programm-Stack); die Anzahl der Bytes, um die dekrementiert wird, ist direkt als 16-Bit-Parameter gegeben
IncSP_16Bit	9	2	Inkrementiere Stackpointer (Programm-Stack); die Anzahl der Bytes, um die dekrementiert wird, ist direkt als 16-Bit-Parameter gegeben

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
LoadTopAdr8	10	2	Lade 8-Bit-Operand auf den Arithmetik-Stack; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum oberen Ende des globalen Speichers gegeben
LoadTopAdr16	11	2	Lade 16-Bit-Operand auf den Arithmetik-Stack; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum oberen Ende des globalen Speichers gegeben
LoadTopAdr32	12	2	Lade 32-Bit-Operand auf den Arithmetik-Stack; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum oberen Ende des globalen Speichers gegeben
LoadImm8	13	1	Lade 8-Bit-Operand auf den Arithmetik-Stack; der Operand selbst ist als Parameter direkt gegeben
LoadImm16	14	2	Lade 16-Bit-Operand auf den Arithmetik-Stack; der Operand selbst ist als Parameter direkt gegeben
LoadImm32	15	4	Lade 8-Bit-Operand auf den Arithmetik-Stack; der Operand selbst ist als Parameter direkt gegeben
LoadStRel8	16	2	Lade 8-Bit-Operand auf den Arithmetik-Stack; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
LoadStRel16	17	2	Lade 16-Bit-Operand auf den Arithmetik-Stack; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
LoadStRel32	18	2	Lade 32-Bit-Operand auf den Arithmetik-Stack; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
StoreTopAdr8	19	2	Nimm 8-Bit-Operand vom Arithmetik-Stack und speichere ihn; die Zieladresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum oberen Ende des globalen Speichers gegeben

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
StoreTopAdr16	20	2	Nimm 16-Bit-Operand vom Arithmetik-Stack und speichere ihn; die Zieladresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum oberen Ende des globalen Speichers gegeben
StoreTopAdr32	21	2	Nimm 32-Bit-Operand vom Arithmetik-Stack und speichere ihn; die Zieladresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum oberen Ende des globalen Speichers gegeben
StoreStRel8	22	2	Nimm 8-Bit-Operand vom Arithmetik-Stack und speichere ihn; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
StoreStRel16	23	2	Nimm 8-Bit-Operand vom Arithmetik-Stack und speichere ihn; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
StoreStRel32	24	2	Nimm 32-Bit-Operand vom Arithmetik-Stack und speichere ihn; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
Add16Sig	25	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe eine 16-Bit-Addition mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y + X)
Sub16Sig	26	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe eine 16-Bit-Subtraktion mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y – X)
Mul16Sig	27	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe eine 16-Bit-Multiplikation mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y * X)
Mul16Unsig	28	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe eine 16-Bit-Multiplikation ohne Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y * X)

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
Div16Sig	29	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe eine 16-Bit-Division mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y / X)
Div16Unsig	30	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe eine 16-Bit-Division ohne Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y / X)
BranchFalse	31	2	Nimm 16-Bit-Operand vom Arithmetik-Stack, wenn Operand gleich Null springe zu Adresse; die Adresse ist als 16-Bit-Parameter gegeben
BranchTrue	32	2	Nimm 16-Bit-Operand vom Arithmetik-Stack, wenn Operand ungleich Null springe zu Adresse; die Adresse ist als 16-Bit-Parameter gegeben
BitAnd16	33	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe ein 16-Bit binäres *Und* aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y & X)
BoolAnd16	34	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe ein 16-Bit logisches *Und* aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y && X)
BitOr16	35	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe ein 16-Bit binäres *Oder* aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y | X)
BoolOr16	36	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe ein 16-Bit logisches *Oder* aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y || X)
BitXor16	37	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe ein 16-Bit binäres *Exklusiv-Oder* aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y ^ X)
ShLeft16	38	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, und führe ein 16-Bit-Bitschieben von X um Y Bits nach links aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y << X)

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
ShRight16	49	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, und führe ein 16-Bit-Bitschieben von X um Y Bits nach rechts aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y >> X)
Lt16Bit	40	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-„kleiner“-Vergleich ohne Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y < X)
Le16Bit	41	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-„kleiner oder gleich“-Vergleich ohne Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y <= X)
Gr16Bit	42	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-„größer“-Vergleich ohne Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y > X)
Ge16Bit	43	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-„größer oder gleich“-Vergleich ohne Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y >= X)
Swap16	44	0	Vertausche die beiden 16-Bit-Operanden X und Y auf dem Arithmetik-Stack (X <> Y)
GenFunc	45	1	Eine interne Interpreter-Funktion wird ausgeführt. Die Adresse der Funktion ist in einer Sprungtabelle gegeben, der Index in die Tabelle ist der 8-Bit-Parameter
Drop	46	1	Entfernt die Anzahl von Einträgen aus dem Arithmetik-Stack, die dem 8-Bit-Parameter entspricht
AddFloat	47	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe eine Fließkomma-Addition mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y + X)
SubFloat	48	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe eine Fließkomma-Subtraktion mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y – X)

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
MultFloat	49	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe eine Fließkomma-Multiplikation mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y * X)
DivFloat	50	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe eine Fließkomma-Division mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y / X)
WSP1	51	0	Nimm 8-Bit-Operanden vom Arithmetik-Stack, speichere Operanden auf Programm-Stack
WSP2	52	0	Nimm 16-Bit-Operanden vom Arithmetik-Stack, speichere Operanden auf Programm-Stack
WSP4	53	0	Nimm 32-Bit-Operanden vom Arithmetik-Stack, speichere Operanden auf Programm-Stack
StoreStRel8Sig	54	2	Nimm 8-Bit-Operand vom Arithmetik-Stack und speichere ihn; die Adresse des Operanden ist als 16-Bit-Parameter mit Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
Store-StRel16Sig	55	2	Nimm 8-Bit-Operand vom Arithmetik-Stack und speichere ihn; die Adresse des Operanden ist als 16-Bit-Parameter mit Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
Store-StRel32Sig	56	2	Nimm 32-Bit-Operand vom Arithmetik-Stack und speichere ihn; die Adresse des Operanden ist als 16-Bit-Parameter mit Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
Eq16Bit	57	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-Vergleich auf Gleichheit aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y == X)
Neq16Bit	58	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-Vergleich auf Ungleichheit aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y != X)

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
LoadAdr8XT	59	0	Nimm 16-Bit-Operand X vom Arithmetik-Stack, verwende X als Speicheradresse, um einen 8-Bit-Operanden auf den Arithmetik-Stack zu laden
LoadAdr16XT	60	0	Nimm 16-Bit-Operand X vom Arithmetik-Stack, verwende X als Speicheradresse, um einen 16-Bit-Operanden auf den Arithmetik-Stack zu laden
LoadAdr32XT	61	0	Nimm 16-Bit-Operand X vom Arithmetik-Stack, verwende X als Speicheradresse, um einen 8-Bit-Operanden auf den Arithmetik-Stack zu laden
LoadStRel8XT	62	0	Nimm 16-Bit-Operand X vom Arithmetik-Stack, addiere X zum Stackpointer und verwende die Summe als Speicheradresse, um einen 8-Bit-Operanden auf den Arithmetik-Stack zu laden
Load-StRel16XT	63	0	Nimm 16-Bit-Operand X vom Arithmetik-Stack, addiere X zum Stackpointer und verwende die Summe als Speicheradresse, um einen 16-Bit-Operanden auf den Arithmetik-Stack zu laden
Load-StRel32XT	64	0	Nimm 16-Bit-Operand X vom Arithmetik-Stack, addiere X zum Stackpointer und verwende die Summe als Speicheradresse, um einen 32-Bit-Operanden auf den Arithmetik-Stack zu laden
StoreAdr8XT	65	0	Nimm den 8-Bit-Operanden X und den 16-Bit-Operanden Y vom Arithmetik-Stack und speichere X; die Zieladresse entspricht dem Operanden Y
StoreAdr16XT	66	0	Nimm den 16-Bit-Operanden X und den 16-Bit-Operanden Y vom Arithmetik-Stack und speichere X; die Zieladresse entspricht dem Operanden Y
StoreAdr32XT	67	0	Nimm den 32-Bit-Operanden X und den 16-Bit-Operanden Y vom Arithmetik-Stack und speichere X; die Zieladresse entspricht dem Operanden Y
StoreStRel8XT	68	0	Nimm den 8-Bit-Operanden X und den 16-Bit-Operanden Y vom Arithmetik-Stack und speichere X; die Zieladresse entspricht der Summe aus dem Operanden Y und dem Stackpointer des Programm-Stacks

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
StoreStRel16XT	69	0	Nimm den 16-Bit-Operanden X und den 16-Bit-Operanden Y vom Arithmetik-Stack und speichere X; die Zieladresse entspricht der Summe aus dem Operanden Y und dem Stackpointer des Programm-Stacks
StoreStRel32XT	70	0	Nimm den 32-Bit-Operanden X und den 16-Bit-Operanden Y vom Arithmetik-Stack und speichere X; die Zieladresse entspricht der Summe aus dem Operanden Y und dem Stackpointer des Programm-Stacks
LoadFuncPtr	71	0	Wird nur intern vom Compiler verwendet
Dup	72	0	Lade den Inhalt von X neu auf den Arithmetik-Stack (dupliziert X)
QuickAddSig8	73	1	Addiere den Inhalt des vorzeichenbehafteten 8-Bit-Parameters auf den Inhalt von X
AddAdr16SigPre	74	1	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack und verwende ihn als Speicheradresse, addiere den vorzeichenbehafteten 8-Bit-Parameter auf den Inhalt der Speicheradresse mit einer 16-Bit-Addition, lade den Inhalt der Speicheradresse als 16-Bit-Operanden auf den Arithmetik-Stack
AddAdr8SigPre	75	1	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack und verwende ihn als Speicheradresse, addiere den vorzeichenbehafteten 8-Bit-Parameter auf den Inhalt der Speicheradresse mit einer 8-Bit-Addition, lade den Inhalt der Speicheradresse als 8-Bit-Operanden auf den Arithmetik-Stack
Not16	76	0	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack, führe ein logisches Negieren auf X aus und lade das Ergebnis auf den Arithmetik-Stack (X = !X)
Inv16	77	0	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack, führe ein binäres Invertieren auf X aus und lade das Ergebnis auf den Arithmetik-Stack (X = ^X)

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
Min16	78	0	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack, führe ein arithmetisches Negieren auf X aus und lade das Ergebnis auf den Arithmetik-Stack (X = –X)
Mod16	79	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe eine 16-Bit Y Modulo X aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y % X)
LoadAdrStRel	80	2	Addiere den Inhalt des 16-Bit-Parameters auf Stackpointer des Programm-Stacks und lade das Ergebnis als 16-Bit-Operand auf den Arithmetik-Stack
AddAdrStRel	81	0	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack, addiere den Stackpointer des Programm-Stacks auf X und lade das Ergebnis auf den Arithmetik-Stack
AddAdr16SigPst	82	1	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack und verwende ihn als Speicheradresse, lade den Inhalt der Speicheradresse als 16-Bit-Operanden auf den Arithmetik-Stack, addiere den vorzeichenbehafteten 8-Bit-Parameter auf den Inhalt der Speicheradresse mit einer 16-Bit-Addition
AddAdr8SigPst	83	1	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack und verwende ihn als Speicheradresse, lade den Inhalt der Speicheradresse als 8-Bit-Operanden auf den Arithmetik-Stack, addiere den vorzeichenbehafteten 8-Bit-Parameter auf den Inhalt der Speicheradresse mit einer 8-Bit-Addition
StorageLabel	84	0	Wird nur intern vom Compiler verwendet
StorageCopy	85	2	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack und verwende ihn als Zieladresse; der 16-Bit-Parameter zeigt auf Speicher im Flash, das erste Byte im Flashspeicher zeigt die Länge des Kopiervorgangs an, ist dieses Byte Null, bilden die beiden nächsten Bytes eine 16-Bit-Länge; kopiere die folgenden Bytes aus dem Flash an die Zieladresse (X)

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
Init_DB	86	0	Wird nur intern vom Compiler verwendet
Switch	87	2	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack und vergleiche den Wert von X mit den Werten in der Sprungtabelle, deren Anfang im 16-Bit-Parameter steht; wird ein richtiger Wert gefunden, dann springe an die Adresse nach dem Wert, ansonsten springe an die „default“-Adresse
Swap32	88	0	Vertausche die beiden 32-Bit-Operanden X und Y auf dem Arithmetik-Stack (X <> Y)
LtFloat	89	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe einen Fließkomma-„kleiner“-Vergleich aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y < X)
LeFloat	90	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe einen Fließkomma-„kleiner oder gleich“-Vergleich aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y <= X)
GrFloat	91	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe einen Fließkomma-„größer“-Vergleich aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y > X)
GeFloat	92	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe einen Fließkomma-„größer oder gleich“-Vergleich aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y >= X)
EqFloat	93	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe einen Fließkomma-Vergleich auf Gleichheit aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y == X)
NeqFloat	94	0	Nimm die beiden Fließkomma-Operanden X und Y vom Arithmetik-Stack, führe einen Fließkomma-Vergleich auf Ungleichheit aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y != X)

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
ToFloat	95	1	Konvertiere abhängig vom 8-Bit-Parameter den Operanden X oder den Operanden Y vom Arithmetik-Stack in eine Fließkommazahl; wenn der 8-Bit-Parameter gleich 1: konvertiere 16-Bit-X nach Fließkomma ohne Vorzeichen 2: konvertiere 16-Bit-Y nach Fließkomma ohne Vorzeichen 3: konvertiere 16-Bit-X nach Fließkomma mit Vorzeichen 4: konvertiere 16-Bit-Y nach Fließkomma mit Vorzeichen
ToInt	96	0	Konvertiere den Fließkomma-Operanden X in 16-Bit-Operanden
MinFloat	97	0	Nimm den Fließkomma-Operanden X vom Arithmetik-Stack, führe ein arithmetisches Negieren auf X aus und lade das Ergebnis auf den Arithmetik-Stack (X = -X)
Lt16BitSig	98	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-„kleiner“-Vergleich mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y < X)
Le16BitSig	99	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-„kleiner oder gleich“-Vergleich mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y <= X)
Gt16BitSig	100	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-„größer“-Vergleich mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y > X)
Ge16BitSig	101	0	Nimm die beiden 16-Bit-Operanden X und Y vom Arithmetik-Stack, führe einen 16-Bit-„größer oder gleich“-Vergleich mit Vorzeichen aus, lade den Arithmetik-Stack mit dem Ergebnis (X=Y >= X)

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
ForCheck	102	2 + 2	Der zweite 16-Bit-Parameter, addiert zum Stackpointer des Programmstacks, zeigt auf Speicherstruktur: 1. Word – For-Schleifen-Variable 2. Word – Zielwert 3. Word – Schrittweite Überprüfe, ob Variable schon Zielwert überschritten ist, wenn ja, springe zur Adresse, die durch den ersten 16-Bit-Parameter spezifiziert ist.
ForJump	103	2 + 2	Der zweite 16-Bit-Parameter, addiert zum Stackpointer des Programm-Stacks, zeigt auf Speicherstruktur: 1. Word – For-Schleifen-Variable 2. Word – Zielwert 3. Word – Schrittweite Addiere Schrittweite auf Variable, dann überprüfe, ob Variable schon Zielwert überschritten hat, wenn ja, springe zur Adresse, die durch den ersten 16-Bit-Parameter spezifiziert ist.
LoadAdrTopRel	104	2	Subtrahiere den Inhalt des 16-Bit-Parameters von der oberen Speichergrenze des Speichers für globale Variablen und lade das Ergebnis als 16-Bit-Operand auf den Arithmetik-Stack
Goto_24	105	3	Springe zu Adresse; die Adresse ist als 16-Bit-Parameter gegeben; setze das RAMPZ-Register des Mega128 auf den 8-Bit-Parameter nach der Adresse
Call_24	106	3	Speichere aktuelle Adresse +3 auf dem Programm-Stack, dann springe zu Adresse, die Adresse ist als 16-Bit-Parameter gegeben, setze das RAMPZ-Register des Mega128 auf den 8-Bit-Parameter nach der Adresse
BranchFalse_24	107	3	Nimm 16-Bit-Operand vom Arithmetik-Stack; wenn Operand gleich Null, springe zu Adresse; die Adresse ist als 16-Bit-Parameter gegeben; setze das RAMPZ-Register des Mega128 auf den 8-Bit-Parameter nach der Adresse

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
BranchTrue_24	108	3	Nimm 16-Bit-Operand vom Arithmetik-Stack; wenn Operand ungleich Null, springe zu Adresse; die Adresse ist als 16-Bit-Parameter gegeben; setze das RAMPZ-Register des Mega128 auf den 8-Bit-Parameter nach der Adresse
StorageCopy_24	109	3	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack und verwende ihn als Zieladresse; der 24-Bit-Parameter (16-Bit-Adresse + 8-Bit-RAMPZ-Mega128) zeigt auf Speicher im Flash, das erste Byte im Flashspeicher zeigt die Länge des Kopiervorgangs an, ist dieses Byte Null, bilden die beiden nächsten Bytes eine 16-Bit-Länge; kopiere die folgenden Bytes aus dem Flash an die Zieladresse (X)
Switch_24	110	3	Nimm den 16-Bit-Operanden X vom Arithmetik-Stack und vergleiche den Wert von X mit den Werten in der Sprungtabelle, deren Anfang im 24-Bit-Parameter (16-Bit-Adresse + 8-Bit-RAMPZ-Mega128) steht; wird ein richtiger Wert gefunden, dann springe an die Adresse nach dem Wert, ansonsten springe an die „default“-Adresse
ForCheck_24	111	3 + 2	Der zweite 16-Bit-Parameter addiert zum Stackpointer des Programm-Stacks zeigt auf Speicherstruktur: 4. Word – For-Schleifen-Variable 5. Word – Zielwert 6. Word – Schrittweite Überprüfe, ob Variable schon Zielwert überschritten hat, wenn ja, springe zur Adresse, die durch den ersten 24-Bit-Parameter (16-Bit-Adresse + 8-Bit-RAMPZ-Mega128) spezifiziert ist.

Bytecode-Name	Nr.	Parameter-Länge	Beschreibung
ForJump_24	112	3 + 2	Der zweite 16-Bit-Parameter, addiert zum Stackpointer des Programm-Stacks zeigt auf Speicherstruktur: 7. Word – For-Schleifen-Variable 8. Word – Zielwert 9. Word – Schrittweite Addiere Schrittweite auf Variable, dann überprüfe, ob Variable schon Zielwert überschritten hat, wenn ja, springe zur Adresse, die durch den ersten 24-Bit-Parameter (16-Bit-Adresse + 8-Bit-RAMPZ-Mega128) spezifiziert ist.
Addr_Trans	113	0	Springe vom ersten 64-kbyte-Flashsegment in das zweite Segment (RAMPZ = 1)
IncSP_8Bit	114	1	Inkrementiere Stackpointer (Programm-Stack), die Anzahl der Bytes um die dekrementiert wird, ist direkt als 8-Bit-Parameter gegeben
LoadTopAdr8SExt	115	2	Lade 8-Bit-Operand auf den Arithmetik-Stack; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum oberen Ende des globalen Speichers gegeben
LoadStRel8SExt	116	2	Lade vorzeichenbehafteten 8-Bit-Operand auf den Arithmetik-Stack; die Adresse des Operanden ist als 16-Bit-Parameter ohne Vorzeichen relativ zum Stackpointer des Programm-Stacks gegeben
LoadAdr8XTSExt	117	0	Nimm 16-Bit-Operand X vom Arithmetik-Stack, verwende X als Speicheradresse, um einen vorzeichenbehafteten 8-Bit-Operanden auf den Arithmetik-Stack zu laden
LoadStRel8XTSExt	118	0	Nimm 16-Bit-Operand X vom Arithmetik-Stack, addiere X zum Stackpointer und verwende die Summe als Speicheradresse, um einen vorzeichenbehafteten 8-Bit-Operanden auf den Arithmetik-Stack zu laden

Sachverzeichnis

A
AbsDelay 145
Add16Sig 217
aktiv 149
Anführungszeichen 98
Arithmetik-Stack 216
arithmetische Ausdrücke 105, 117
Array-Zugriff 222
ASCII 91
Ausführungszeit 145

B
Bedingte Kompilierung 126
Bibliotheksoptionen 122

C
Call 220
Carriage Return 98
C-Control-ProInterrupts 135
Character 216
Copy2 115

D
DATE 132
Datentypen 216
DEBUG 131
DecS 217
#define 124
Dekrement Operator 102
Dezimalzahlen 99
Drop 220
#elif 126
#else 126
#endif 126

E
ExterneInterrupts 137
Ext_IntDisable 138
Ext_IntEnable 137

F
FILE 132
Floating-Point 216
Formfeed 98
For-Schleifen 116
FUNCTION 132
Funktionsaufruf 219

G
GenFunc 219
gesichert 149
Glocke 98
Gnu Generic Preprocessor 124
Goto 221
Gr16BitSig 221

H
Hardware Interrupts 135
Hauptthread 140
Hexadezimale 99
hochspezialisierte Bytecodes 102

I
#if 126
#ifdef 126
#ifndef 126
inaktiv 149
#include 128
IncSP_8Bit 217

Inkrement-Operator 102
INT_0 135
INT_1 135
INT_2 135
INT_ADC 136
INT_ANA_COMP 135
Interpreter-Speicher 215
INT_TIM0COMP 135
INT_TIM1CAPT 135
INT_TIM1CMPA 135
INT_TIM1CMPB 135
INT_TIM1OVF 135
Inv16 217
Irq_GetCount 136
Irq_SetVect 136

L
LINE 132
LoadAdrTopRel 223
LoadImm8 217
LoadTopAdr8 217

M
Makros 129
Map-Datei 143
MAPFILE 131
MEGA128 131
MEGA32 131
Multithreading 140

O
$opc 219
Optimierung 101
#pragma Error 133
#pragma Language 133
#pragma Message 133
#pragma Warning 133
#pragma 133

P
Preprozessor Makros 129
Preprozessor 124
Prioritätenvon Threads 145
Programm-Stack 216
Projektoptionen 110
Projektoptionen 122

R
rekursive Funktionen 144
Rückstrich 98
Rücktaste 98

S
schlafend 149
Select-Case-Anweisungen 115
Signal 148
Speicherbereiche 215
Speicherverbrauch 143
Stackgröße 142
Stacklift 217
Stackpointer 216
Steuerzeichen 98
StoreTopAdr8 217
STR 92
Str 92
Str 94
Str 95
Strings 91
switch-Anweisungen 104

T
Tabulator 98
Thread_Delay 146
Thread_Kill 140
Threadkonfiguration 123, 142
Thread_Lock 149
Thread_MemFree 144
Thread_Resume 147
Thread_Signal 147
Threads synchronisieren 147
Thread_Start 140
Thread_Wait 147
TIME 132

U
#undef 126
UPN 216

V
Vertikaler Tabulator 98
Vordefinierte Symbole 131

W
wartend 149
Write Word 99
WriteSP2 219

Z
Zahlenbasis 99
Zeichenketten 91
Zeilenvorschub 98
Zykluszeit 145

Der leichte Einstieg in die

Elektronik

Dieses ultimative Einsteigerbuch ermöglicht es Ihnen, sich ohne Vorkenntnisse schnell und leicht in der modernen Elektronik zurechtzufinden.

Es hat den Anschein, dass sich die Elektronik im Computerzeitalter zu einer geheimnisvollen Branche entwickelt, die viele Outsider abschreckt, sich eingehend damit zu befassen. Viele Interessenten haben schon aufgrund der verwirrenden Fachwörter Angst vor der nur scheinbar komplizierten Materie. Dabei ist die Elektronik gar nicht so schwierig zu verstehen. Im Gegenteil!

Viele interessante Schaltungen weisen Ihnen den Weg zum Verständnis und führen geradewegs zum befreienden Aha-Erlebnis. Der Autor hat bewusst auf theoretische Abhandlungen verzichtet, um Sie direkt zum Erfolg zu führen. Maßgeschneiderte Bauanleitungen bieten dabei nicht nur entsprechende Erfolgserlebnisse, sondern auch praktische Nutzanwendungen im Alltag.

Aus dem Inhalt:

- Das nötige Grundlagenwissen
- Wozu eignen sich die nötigen Bausteine der Elektronik?
- Wie können einfache, preiswerte und nützliche Schaltungen im Selbstbau erstellt werden?
- Was, wie und womit wird in der Elektronik gemessen?
- Entwickeln Sie eigene Schaltungen
- Wie können Schaltungen über den PC angesteuert werden?

Jokers edition

ISBN 978-3-7723-4079-6