C-Control Pro
Mega Series

Table of Contents

Part 7 FAQ   253

Part

1

1 Important Notes

This chapter deals with important information's to warranty, support and operation of the C-Control-Pro hardware and software.

1.1 Introduction

The C-Control Pro Systems are based on the Atmel Mega 32 and the Atmel Mega 128 RISC Microcontrollers, resp.. These Microcontrollers are used in large numbers in a broad variety of devices from entertainment electronics through household appliances to various application facilities in the industries. There the controller takes charge of important control tasks. C-Control Pro offers this highly sophisticated technology to solve your controlling problems. You can acquire analog measuring values and switch positions and provide corresponding switching signals dependent on these input conditions, e. g. for Relais and servo motors. In conjunction with a DCF-77 radio antenna C-Control Pro can receive the time with atomic accuracy and thus take over precise time switch functions. Various hardware interfaces and bus systems allow the cross linking of C-Control Pro with sensors, actors and other control systems. We want to provide a broad user range with our technology. From our former work in C-Control service we know that also customers without any experience in electronics and programming but eager to learn are interested in C-Control. If you happen to belong to this user group please allow us to give you the following advice:

C-Control Pro is only of limited use for the entry into programming of micro computers and electronic circuit technique! We presuppose that you have at least a basic knowledge in a higher programming language such as BASIC, PASCAL, C, C++ or Java. Furthermore we presume that you are used to operating a PC under one of the Microsoft operating systems (98SE/NT/2000/ME/XP). You should further be experienced in working with soldering irons, multimeters and electronic components. We have made every effort to formulate all descriptions as simple as possible. Unfortunately we were not able to do without the use of technical terms and expressions in an operating manual to the themes involved here. If need be please see the appropriate technical literature.

1.2 Reading this operating manual

Please read this operating manual thoroughly prior to putting the C-Control Pro Unit into operation. While several chapters are only of interest for the understanding of the deeper coherence's, others contain important information's whose non-compliance may lead to malfunctions or even damages.

➡ Chapters and paragraphs containing important themes are marked by a symbol.

Please read the entire manual prior to putting the unit into operation since it contains important notes for correct operation. In case of damages to material or personnel caused by improper handling or non-compliance to this operating manual the warranty claim will expire! We will further not take liability for consequential damages.

1.3 Handling

The C-Control Pro Unit contains sensitive components. These can be destroyed by electrostatic discharges! Please observe the general rules on handling electronic components. Please organize your working bench professionally. Ground your body prior to any work being done, e. g. by touching a grounded and conducting object (e. g. heating radiator). Avoid touching the connection

pins of the C-Control Pro Unit.

1.4   Intended use

The C-Control Pro Unit is an electronic device in the sense of an integrated circuit. It serves the programmable controlling of electric and electronic equipment. Construction and operation of this equipment must be in conformance with the valid European licensing principles (CE).

The C-Control Pro must not be galvanically connected to voltages exceeding the directed Extra Low Protective Voltage. Coupling to systems with higher voltages must exclusively be performed by use of components having VDE qualification. In doing so the directed air and leakage paths must be observed as well as sufficient precautions for protection against touching dangerous voltages must be taken.

The PCB of the C-Control Pro Unit carries electronic components with high frequency clock signals and steep pulse slopes. Improper use of the unit may lead to the radiation of electro-magnetic interference signals. The adoption of proper measures (e. g. the use of chokes, limiting resistors, blocking capacitors and shielding's) to ensure the observance of legally directed maximum values lies in the responsibility of the user.

The maximum allowable length of connected wire lines is without additional precautions appr. 0.25 Meters (Exception: Serial Interface). Under influence of strong electro-magnetic alternating fields or interference pulses the function of the C-Control Pro Unit can be detracted. If need be a reset or a restart of the system may become necessary.

During connection of external sub-assemblies the maximum admissible current and voltage values of the particular pins must be observed. The connection of too high a voltage, a voltage of wrong polarity or an excessive current load may lead to immediate damage of the unit. Please keep the C-Control Pro Unit away from spray water or condensation dampness. Observe the safe operating temperature range in Item Technical Data in the attachment.

1.5   Warranty and Liability

For the C-Control Pro Unit Conrad Electronic grants a warranty period of 24 months from the date of billing. Within this time period faulty units will be replaced free of charge if the fault provable originates in faulty production or loss on goods in transit.

The software in the operating system of the Microcontroller as well as the PC software on CD-ROM is shipped in the form as is. Conrad Electronic can not guarantee that the performance features of this software will satisfy individual requirements and that this software will operate free of faults and interruptions. Conrad Electronic can further not be held liable for damages occurring directly by or consequently to the use of the C-Control Pro Unit. The use of the C-Control Pro Unit in systems directly or indirectly serving medical, health or life saving objectives is not authorized.

In case the C-Control Pro Unit incl. software does not satisfy your demands or if you do not agree to our warranty and liability conditions you are to make use of our 14 days money back guarantee. Please return the unit without use marks, in the undamaged original packaging and incl. all accessories within this time-limit to our address for refund or clearing of the value of goods!

1.6 Service

Conrad Electronic provides you with a team of experienced service technicians. If you have any question with regard to our C-Control Pro Unit you can reach our Technical Service by letter, telefax or e-mail.

By letter   Conrad Electronic
Technical Inquiry
Klaus-Conrad-Straße 2
D-92530 Wernberg-Köblitz

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

Please preferably use e-mail communication. If there is a problem possibly provide us with a sketch of your connection diagram in form of an attached picture file (jpg format) as well as the program source code reduced to the part referring to your problem (max. 20 lines). Further information's and current software for download please find on the C-Control homepage under www.c-control.de.

1.7 Open Source

When C-Control Pro was designed also open source software has come into operation:

ANTLR 2.73   http://www.antlr.org
Inno Setup 5.2.3   http://www.jrsoftware.org
GPP (Generic Preprocessor)   http://www.nothingisreal.com/gpp
avra-1.2.3a Assembler   http://avra.sourceforge.net/

In accordance with the rules of "LESSER GPL" (www.gnu.org/copyleft/lesser) during installation of the IDE also the original source code of the avra assembler, the generic pre-processor as well as the source text of the modified version is supplied, which is used with C-Control Pro. Both source texts are found in a ZIP file in the "GNU" sub-directory.

1.8 History

Version 2.01 from 06/27/2009

New Features
• Added Search Function into Editor popupmenu

Error Corrections
• Documentation update
• Error at "unused Code Optimizer" corrected
• Fixed internal handling of data crossing 64kb boundary
• Fixed error when starting programs from Tools menu
• Fixed error when resizing output window
• Corrected translation bugs in Search dialog
• Line offset fixed in Project Search
• Timeout in I2C Routines

- Fixed error message "...tbSetRowCount:new count too small"

Version 2.00 from 05/14/2009

New Features
- Assembler Support
- Enhanced Search Functions in the Editor
- New configurable GUI
- Todo List
- switchable Compiler Warnings
- Program Transfer of Bytecode without Project
- extended Program Info
- Fast Transfer if Interpreter already on Module
- Enhanced Auto-Completion of Keywords and Function Names
- Function Parameter help
- unused Code Optimizer
- Peephole Optimizer
- Support for predefined Arrays in Flash Memory
- Realtime Array Index check
- Optimized Array Access
- better verification of constant array indices
- call functions with string constants
- Enter binary numbers with 0b....
- Addition und Subtraction of Pointers
- Optimized Port OUT, PIN and DDR Access
- Direct Atmel Register Access
- Formatted String Output with Str_Printf()
- convert ASCII strings in numerical values
- ++/-- for BASIC
- Port toggle functions
- RC5 Send and Receive Routines
- Software Clock (Time & Date) with Quartz correction factor
- Servo Routines
- mathematical Round
- Atmel Mega Sleep Function

Error Corrections
- Initialization Timer_T0FRQ corrected
- Initialization Timer_T0PWM corrected
- Initialization Timer_T1FRQ corrected
- Initialization Timer_T1FRQX corrected
- Initialization Timer_T1PWM corrected
- Initialization Timer_T1PWMX corrected
- Initialization Timer_T1PWMXY corrected
- Initialization Timer_T3FRQ corrected
- Refresh for Array Window corrected
- Desktop Reset corrected
- Module Reset corrected
- Bug in Debugfiles >30000 Bytes corrected
- Bug in conditional valuation in CompactC fixed
- Bug in Timer_Disable() fixed

Version 1.72 from 10/22/2008

New Features
- added SPI functions
- RP6 AutoConnect

Error Corrections
- improved quality of serial transfers

Version 1.71 from 06/25/2008

New Features
- new Editor in IDE
- Editor shows all defined function names
- Editor supports code folding
- Simple serial Terminal
- Pulldownmenu to start your own programs (Tool Quickstart)
- Syntaxhighlighting of all standard library functions
- Configuration of Syntaxhighlighting
- Extension of Select .. Case in BASIC
- Automatic case correction for keywords and library function names
- Simple automatic lookup for keywords and library function names
- OneWire Library Functions
- Comments of Blocks in BASIC with /* , */
- New FTDI driver

Error Corrections
- Global For-Loop counter variables in BASIC work now correct
- Char variables work now correct with negative numbers
- "u" after an integer now defines unsigned number
- Project names now can contain special characters
- Thread_Wait() now supports thread parameter
- return command in CompactC without return parameter was working wrong
- Corrected swapped error messages when called functions with pointers
- Corrected error message at assignment, when function had no return parameter
- Nested switch/Select statements are working now
- Very long switch/Select statements are functioning properly now
- Better Error recovery when selected COM Port already in use
- No longer a crash if very huge amounts of faulty data where transferred over USB or COM Port
- "Exit" in BASIC For-Loops is working now
- Compiler error corrected in declaration of array variables

Version 1.63 from 12/21/2007

Error Corrections
- Documentation update

Version 1.62 from 12/08/2007

New Features

- Vista Compatibility

Error Corrections

- Brackets are working correctly
- The compiler is no longer crashing when variable names are not known
- There were sometimes incorrect syntax errors when opening some brace levels and a missing operand
- "Exit" don't worked correctly in BASIC For-Next loops
- The array window could only be opened 16 times, even when some array windows were closed
- Renamed the Text "Compiler" to "Compiler Defaults" in the Options Menu

Version 1.60 from 03/04/2007

New Features

- English language version of IDE - switchable at runtime
- English language Compiler messages
- English language version of help files and manual
- printing of source code from the IDE
- Print preview of source code
- Thread_Wait() extended with thread parameter
- ADC_Set() got a speedup
- DoubleClock mode can be activated in serial functions

Error Corrections

- ExtIntEnable() was only working correct with IRQ 0 and 4
- Serial_Init() und Serial_Init_IRQ() got wrongly a byte as divider instead of a word
- EPROM_WriteFloat und EEPROM_ReadFloat() sometimes worked incorrect
- Thread_Kill() worked erroneous when called from the main thread
- read accesses from globally defined floating point arrays were faulty
- The second serial interface was not working correctly
- EEPROM write accesses that used illegal addresses could overwrite reserved data in EEPROM
- There was a chance with a very low probability that the LCD display content could get corrupted

Version 1.50 from 11/08/2005

New Features

- IDE Support for Mega128
- Improved Cache Algorithm during IDE access to Transit Time Data in the Debugger
- New Library Routines for Timer 3 (Mega128)
- Programs using the extended (>64kb) Address Space (Mega128)
- Supporting the external 64kb SRAM
- Supporting the external Interrupts 3 - 7 (Mega128)
- Routines for the 2. Serial Interface (Mega128)
- Mathematical Functions (Mega128)
- Display of Memory Volume when Interpreter is started
- Internal RAM check for recognition when Global Variables too large for Main Memory
- Interner RAM check for recognition when Thread Configuration too large for Main Memory
- Transit Time Check if Stack Limits have been violated
- Source Files can be moved up and down in the Project Hierarchy
- Warning when Strings too long are assigned

- On demand the Compiler creates a Map File describing the volume of all Program Variables
- New Address model for Global Variables (the same Program runs at different RAM Volumes)
- Interrupt Routines for Serial Interface (up to 256 Byte Receiver Buffer / 256 Byte Transmitter Buffer)
- Fixed wired IRQ Routines to allow Periode Measurement of small time intervals
- Recursions are now usable without limits
- Arrays of any size can now be displayed in a separate Window in the Debugger
- Strings (character arrays) are now shown as Tooltip in the Debugger
- SPI can be switched off in order to use the pins for I/O
- The Serial Interface can be switched off in order to use the pins for I/O
- The Hex value is now additionally shown as Tooltip in the Debugger
- New Function Thread_MemFree()
- Additional EEPROM Routines for Word and Floating Point access
- Time Measurement with Timer_TickCount()
- #pragma Commands to create Errors or Warnings
- Pre-defined Symbol in Pre-Prozessor: __DATE__, __TIME__ __FILE__, __FUNCTION__, __LINE__
- Version Number in Splashscreen
- Extended Documentation
- Interactive Graphics at "Jumper Application Board" in Help File
- New Demo Programs
- Ctrl-F1 starts Context Help

Error Corrections
- An Error is created if the Return Command is missing at the end of a function
- Breakpoint Marks have not always been deleted
- Limits at EEPROM Access can now be checked closer (internal overflow seized)
- In the Debugger a single step can no longer depose the next command too early

Version 1.39 from 06/09/2005

New Features
- BASIC Support
- CompactC and BASIC can be mixed in a project
- Extended Documentation
- Loop Optimizing for For - Next in BASIC
- ThreadInfo Function
- New Demo Programs

Error Corrections
- Compiler does no longer break down at German Umlauts (ä, ö, ü)
- Internal Byte Code of command StoreRel32XT corrected
- Offset in String Table improved

Version 1.28 from 04/26/2005

- Initial Version

Part

2

2   Installation

In this chapter the installation of hardware and software is described.

2.1   Applicationboard

Important Note on Inserting/ Retrieving a Mega Module

For the connection between Module and Application Board high quality connectors have been used in order to ensure intimate contacts. Mounting and dismounting of a Module should only take place during power-down condition (switched off voltage) since otherwise damages may occur to Application Board and/ or Module resp. Because of the high number of contacts (40/ 64 Pins) considerable force may be necessary to insert/ retrieve the Module. When inserting it must be ensured that the Module is pressed into the socket evenly, i. e. not out of line. To do this the Module should be placed onto an even surface. Mount the Module Mega32 in the correct orientation observing the marking for Pin 1. The label inscription will then point towards the control elements on the Application Board

Mounting Orientation of Module MEGA32
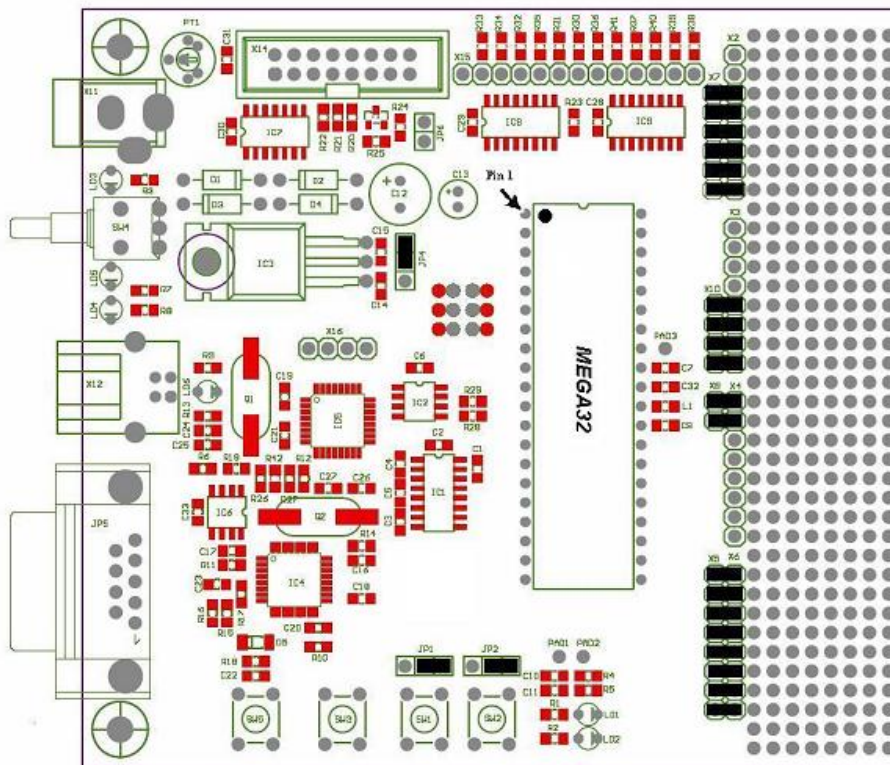
 The connector of Module Mega32 has been designed in such a way that faulty insertion of the Module is not possible. The dismounting of the Module is performed by carefully lifting it from the socket by use of a suitable tool. In order to avoid bending the contacts the lifting of the Module should take place from various sides.

Installation of the USB Drivers

Please connect the Application Board to an appropriate power supply. A Standard 9V/ 250mA Mains Plug-in Power Supply will be sufficient. The polarity does not matter since it is automatically corrected by means of diodes. Depending on additionally used components it may later become necessary to use a power supply with higher output. Establish a connection between the Application Board and your PC by use of a USB cable. Switch on the Application Board.

➡ A Windows Operating System prior to Win98 SE ("*Second Edition*") will supposedly not allow a reliable USB connection between PC and Application Board. From experience Microsoft's USB drivers will only reliably work with all USB devices starting with Win98SE. In such a case it is recommended to either grade up to a more recent Operating System or use only the serial connection to the Application Board.
If the Application Board is connected for the first time then there will be no driver for the FTDI chip. The following window will then be shown under Windows XP.
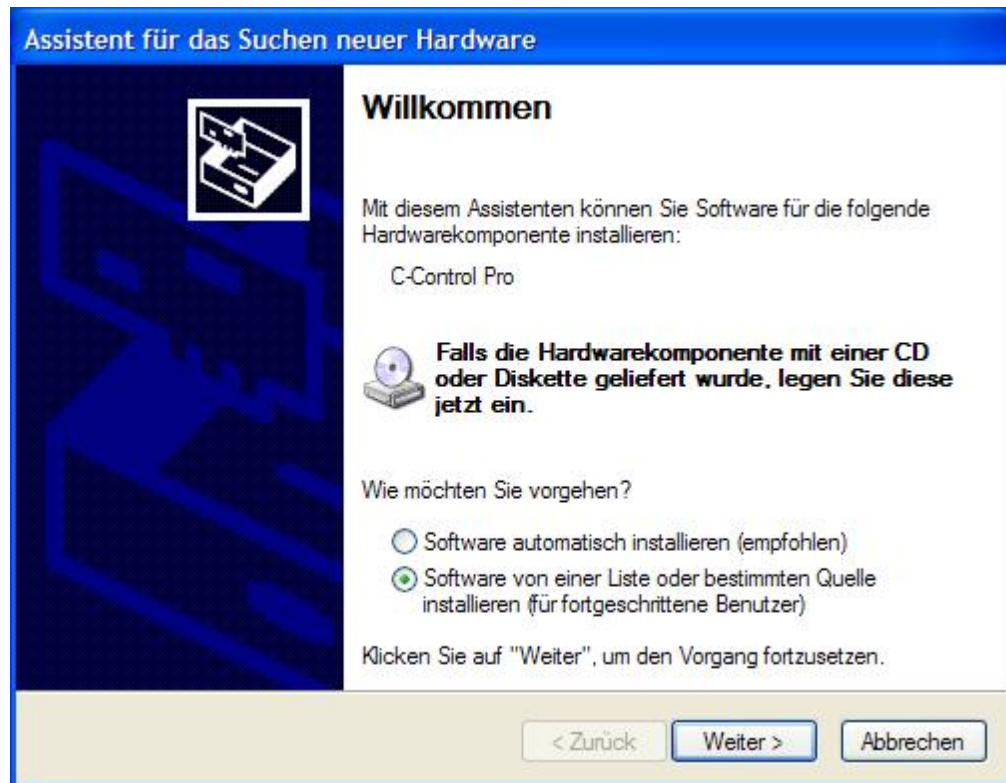
From here select "Install software from a list or other source" and click "Next"..
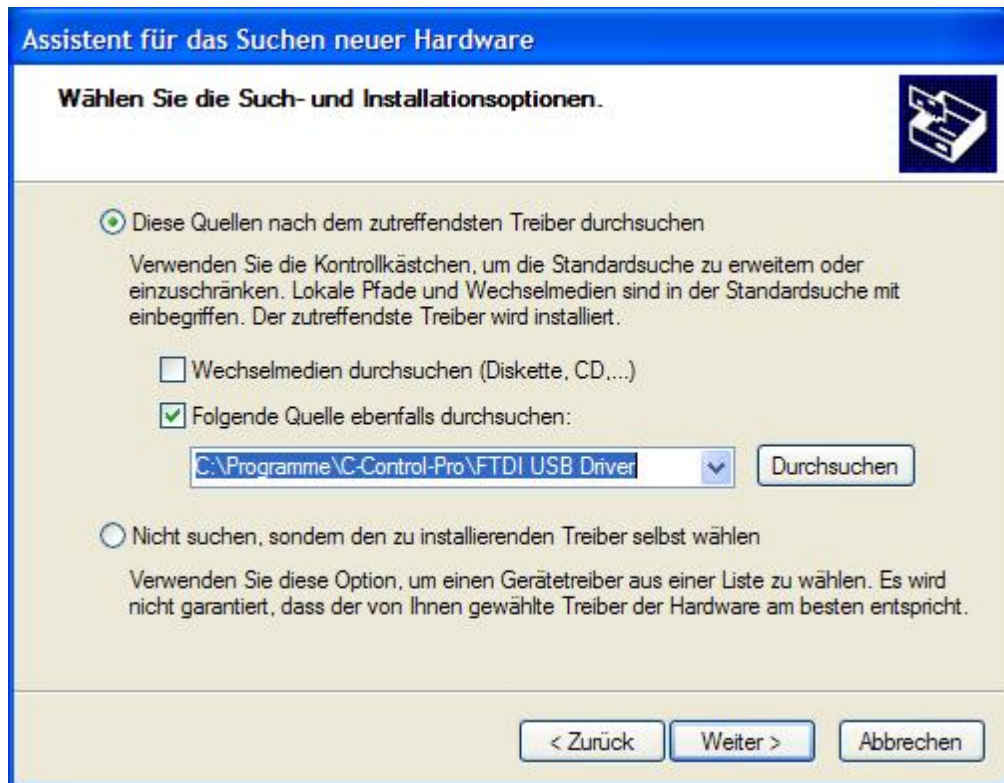
Then type in the path to the driver's directory. If the software has been installed to "C:\Programs" it will be path "C:\Programs\C-Control-Pro\FTDI USB Driver".

The message "C-Control Pro USB Device has not passed the Windows Logo Test ...." will normally appear. This does **not** mean that the driver has failed during the Windows Logo Test. It merely means that the driver has not taken part in the (quite costly) Redmond Test.

Here click "Continue Installation". The USB driver should then be installed after a few seconds.

In the PC software click on IDE in menu Options and select the area Interfaces. Here select the communication port "USB0".

➡ The FTDI driver supports 32 bit and 64 bit operating systems. The specific drivers are located in the "FTDI USB Driver\i386" and "FTDI USB Driver\amd64".

Serial Connection

Due to the slow transmitting speed of the serial interface the USB connection should preferably be used. If however due to hardware grounds the USB interface is not available then the Bootloader can be switched into the Serial Mode.

To do this the key SW1 has to be kept pressed during power-up of the Application Board. After this the Serial Bootloader Mode will be activated.

In the PC software click on IDE in menu Options and select the area Interfaces. Here select the communication port "COMx", which fits to the PC interface connected to the Application Board.

2.2   Software

When the attached CD-ROM is inserted into the computer the Installer should be automatically started in order to install the C-Control Pro software. If this is not the case because e. g. the Autostart Function in Windows is not activated then please manually start the Installer 'C-ControlSetup.exe' in the main directory of your CD-ROM.

➡ For the time of software and USB driver installations the user must be registered as administrator. During normal operation of C-Control Pro this is not necessary.

➡ In order to maintain consistency of the demo program during installation on top of an existing installation the old directory Demo Programs will be deleted and replaced by a new one. It is thus recommended to install other programs outside the C-Control Pro directory.

At the beginning of the installation first select the language in which the installation should take place. After that you can choose whether you want to install C-Control Pro into the standard path or whether you want to specify your own target directory. At the end of the installation process you will be asked if an icon should be created on your desktop.

When the installation process is completed you can choose whether you want to see the "ReadMe" file, have the shortform introduction displayed or directly start the C-Control Pro design platform.

Part 3

3   Hardware

This chapter gives a description of the hardware coming into operation with the C-Control Pro series. The Modules C-Control Pro Mega32 and C-Control Pro Mega128 will be described. Further chapters will comment on construction and function of the accompanying application boards and LCD modules as well as the keyboard.

3.1   Firmware

The operating system of C-Control Pro consists of the following components:

- *Bootloader*
- *Interpreter*

Bootloader

The Bootloader is available at any time. It serves the serial or USB communication with the IDE. By use of command line commands the Interpreter and the user program can be transferred from the PC to the Atmel Risc Chip. If a program is compiled and transferred to the Mega Chip the current Interpreter is also transferred at the same time.

➡ If instead of the USB interface a serial connection should be set up from the IDE to the C-Control Pro module then the push button SW1 (Port **M32**:D.2 and **M128**:E.4 resp. at low level) must be held pressed during power-up of the module. In this mode any communication will be directed through the serial interface. This is useful when the module has already been incorporated into the hardware application and the application board is thus not available. The serial communication however is considerably slower than the USB connection. In serial mode the USB pins are not used and are thus available to the user for other tasks.

➡ Since SW1 initiates the serial Bootloader during module start there should be no signal on Port **M32**:D.2 and **M128**:E.4, resp. during power-up of the application since these ports are also usable as outputs.

SPI Switch Off (only Mega128)

A signal on the SPI interface during switch on can activate USB communication. In order to avoid this PortG.4 (LED 2) can be set LOW during switch on. The SPI interface will then not be activated. The SPI interface can also be manually be switched off by the Interpreter later on using SPI_Disable ().

Interpreter

The Interpreter consists of the following components:

- Bytecode Interpreter
- Multithreading Support
- Interrupt Processing

- User Functions
- RAM and EEPROM Interface

In general the Interpreter processes the bytecode generated by the Compiler. Further most library functions are integrated into it in order to allow access of the bytecode program to e. g. the hardware ports. The RAM and EEPROM Interfaces are used by the IDE's Debugger to get access to the variables when the Debugger is stopped at any Breakpoint.

Autostart

If no USB interface is connected and if SW1 has not been pressed during power-up in order to reach the serial Bootloader mode then the Bytecode (if available) is started in the Interpreter. This means that in case that the module is inserted into a hardware application the mere connection of the operating voltage will suffice to automatically start the user program.

3.2 LCD Matrix

The complete datasheets are on the CD-ROM in the directory "Datasheets".

CHARACTER MODULE FONT TABLE (Standard font)

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

CHARACTER FONT TABLE

LOWER 4 BITS \ UPPER 4 BITS	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)		0	@	P	`	p		─	タ	ミ	α	p
0001	(2)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	β	θ
0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
0100	(5)	$	4	D	T	d	t	、	エ	ト	ャ	μ	Ω
0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
0111	(8)	'	7	G	W	g	w	ア	キ	ヌ	ラ	g	π
1000	(1)	(	8	H	X	h	x	イ	ク	ネ	リ	√	x̄
1001	(2)	)	9	I	Y	i	y	ウ	ケ	ノ	ル	˙	y
1010	(3)	*	:	J	Z	j	z	エ	コ	ハ	レ	j	千
1011	(4)	+	;	K	[	k	{	オ	サ	ヒ	ロ	×	万
1100	(5)	,	<	L	¥	l	|	ヤ	シ	フ	ワ	¢	円
1101	(6)	─	=	M	]	m	}	ユ	ス	ヘ	ン	£	÷
1110	(7)	.	>	N	^	n	→	ヨ	セ	ホ	゛	ñ	
1111	(8)	/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

HANTRONIX Page 47

3.3 Mega32

Mega32 Overview

The Micro Controller ATmega32 originates from the AVR family by ATMEL. It is a low-power Micro

Controller with Advanced RISC Architecture. In the following see a short summary of its hardware resources:

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

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

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

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

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

3.3.1 Module

Module Memory

The C-Control Pro Module provides 32kB FLASH, 1kB EEPROM and 2kB SRAM. A supplementary EEPROM with an 8kB memory depth is mounted on the application board. The latter can be addressed by an I2C interface.

Note: Detailed information can be found in the IC manufacturer's PDF files on the C-Control Pro Software CD-ROM.

ADC-Reference Voltage Generation

The Micro Controller is equipped with an analog-to-digital converter with a 10 Bit resolution. This means that measured voltages can be represented by integral numbers from 0 through 1023. The reference voltage for the lower limit is GND level, i. e. 0V. The reference voltage for the upper limit can be selected by the user:

- 5V Operating Voltage (VCC)
- Internal Reference Voltage of 2.56V
- External Reference Voltage e. g. 4,096V generated by a Reference Voltage IC.

If *x* is a digital measuring value then the corresponding voltage value u is computed as follows:

u = *x* * Reference Voltage / 1024

Clock Generation

Clock generation takes place by a 14.7456MHz Quartz Oscillator. All time dependent actions within the controller are derived from this clock frequency.

Reset

A Reset initiates the return of the Micro Controller system to a defined starting condition. In gerneral the C-Control Pro Module knows two reset sources:

- Power-On-Reset: is automatically executed after switch on of the operating voltage.
- Hardware-Reset: is executed when the Module's RESET (Pin 9) is pulled to "low" and released again by e. g. shortly pressing the connected reset key RESET1 (SW3).

A "Brown-Out-Detection" avoids that the Controller can enter undefined conditions in case of dropping operating voltages.

Digital Ports (PortA, PortB, PortC, PortD)

The C-Control Pro Module provides four digital ports at 8 pins each. To the digital ports it is possible to connect e. g. pushbuttons with pull-up resistors, digital IC's, opto couples or driver circuits for relais. The ports can be addressed either separately, i.e. pin by pin or byte by byte. Each pin can either be input or output.

➡ Never connect two ports directly together which should simultaneously work as outputs!

Digital input pins are high-impedance or wired to internal pull-up resistors and transform an applied voltage signal into a logical value. For this it is required that the voltage signal is within the limits defined for TTL and CMOS IC's high or low levels. During further processing in the program the logical values on the respective input ports are represented as 0 ("low") or 1 ("high"). Pins will take on the values 0 or 1, Bytes from 0 to 255. Output ports are able to give out digital voltage signals by use of an internal driver circuit. Connected circuits can draw (at high level) or feed (at low level) a specific current from or to the ports.

➡ Pay attention to the maximum admissable load current for a single port or for all ports in total. Exceeding the maximal values may lead to destruction of the C-Control Pro Module. After a reset each port is initially configured as input port. By certain commands the direction of data transport can be toggled.

➡ It is important to closely study the pin assignment of M32 and M128 prior to programming since important functions of the program design (e. g. the USB interface of the application board) will apply to specific ports. If these ports are re-programmed or if the matching jumpers on the application board are no longer set then it may happen that the design platform can no longer transfer any programs to the C-Control Pro. Timer inputs and outputs, A/D converter, I2C as well as serial interface are also connected to various port pins.

PLM-Ports

There are two timers available for PLM. These are *Timer_0* with 8 bits and *Timer_1* with 16 bits. They can be used for D/A conversion, to control servo motors in pattern making and to output audio frequencies. A pulse length modulated signal has a period of N so called "Ticks". The duration of one tick is the time base. If the output value of a PLM port is set to X then the port will hold high level for X ticks of one period and will then for the balance of the period drop to low level. For programming of the PLM channels see Timer.
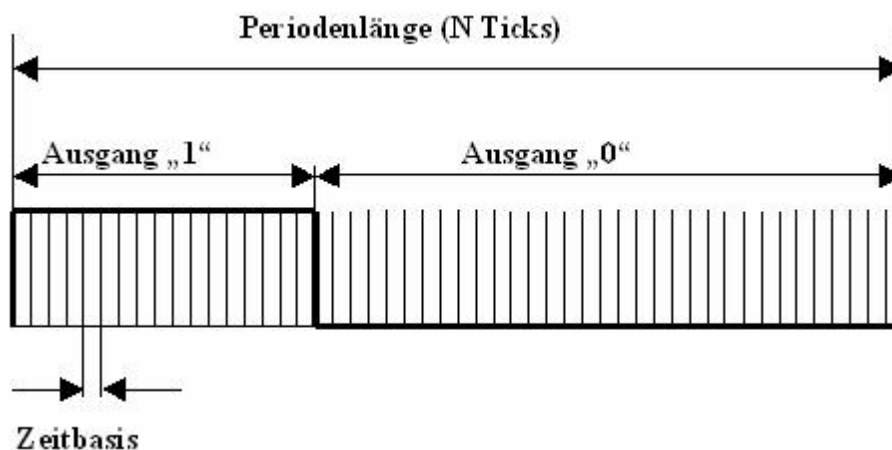
The PLM channels for *Timer_0* and *Timer_1* have independent time base and period length. In applications for pulse width modulated digital to analog conversion the time base and period length are set once and then only the output value is varied. According to their electrical properties the PLM ports are digital ports. Please observe the technical boundary conditions for digital ports (max. current).

Technical Data Module

Note: Detailed information can be found in the IC manufacturer's PDF files on the C-Control Pro Software CD-ROM.

All voltage specifications apply to direct current (DC).

Environmental Conditions	
Range of admissable ambient temperature	0°C … 70°C
Range of admissable ambient relative humidity	20% … 60%
Power Supply	
Range of admissable supply voltage	4,5V … 5,5V
Power reqirement of the module without external loads	appr. 20mA

Clock	
Clock Frequency (Quartz Oscillator)	14.7456MHz
Mechanics	
Overall measurements less pins, appr.	53 mm x 21mm x 8 mm
Weight	appr. 90g
Pin pitch	2.54mm
Number of pins (two rows)	40
Distance between rows	15.24mm

Ports	
Max. adimissable current from digital ports	± 20 mA
Admissable current total on digital ports	200mA
Admissable input voltage on port pins (digital and A/D)	–0.5V ... 5.5V
Internal pull-up resistors (disconnectable)	20 - 50 kOhm

3.3.2   Application Board

USB

The application board provides a USB interface for the program's loading and debugging. Because of the high data rate of this interface data transmission times are considerably shorter compared to the serial interface. Communication takes place through a USB Controller by FTDI and an AVR Mega8 Controller. The Mega8 provides its own Reset push button (SW5). During USB operation the status of the interface is indicated by two light emitting diodes (LD4 red, LD5 green). When only the green LED lights up the USB interface is ready for operation. During data transmission both LED's will light up. This also applies to the Debug mode. Flashing of the red LED indicates an error condition. For USB communication the SPI interface of Mega32 is used (PortB.4 through PortB.7, PortA.6, PortA.7), which must be connected by their respective jumpers.

Note: Detailed information on the Mega32 can be found in the IC manufacturer's PDF files on the C-Control Pro Software CD-ROM.

On-Off Switch

The switch SW4 is located on the front of the application board and serves the power-up (On) or power-down (Off) of the voltage supply.

Light Emitting Diodes (LED)

There are 5 light emitting diodes (LEDs). The LD3 (green) is located on the front below the DC terminals and lights up when supply voltage is applied. LD4 and LD5 indicate the status of the USB interface (see Section USB). The green LEDs LD1 and LD2 are located next to the four push buttons and are freely available to the user. They are connected to VCC through a dropping resistor. By means of jumpers LD1 can be connected to PortD.6 and LD2 to PortD.7. The LEDs will light up when the corresponding pin port is low (GND).

Push Buttons

There are four push buttons provided for. SW3 (RESET1) will initiate a reset with Mega32 while SW3 (RESET2) will do the same with Mega8. The push buttons SW1 and SW2 are freely available to the user. Through jumpers SW1 can be connected to PortD.2 and accordingly SW2 to PortD.3. There is the possibility to connect switches SW1/2 to either GND or VCC. The possibilities to choose from are determined by JP1 and JP2 resp. In order to have a defined level at the input port while the push button is open the corresponding pull-up should be switched on (see Section Digitalports).

➡ Pressing SW1 during power-up of the board will activate the Serial Bootloader Mode.

LCD

An LCD module can be plugged onto the application board. It displays 2 lines at 8 characters each. In general also differently organized displays can be operated through this interface. Each character consists of a monochrome matrix of 5x7 pixels. A flashing cursor below any one of the characters will indicate the current output position. The operating system provides a simple software interface for output on the display. This display is connected to connector X14 (16 poles, double row). By means of a mechanical protection a faulty connection and thus the confusing of poles is avoided.

The LCD module used is of type Hantronix HDM08216L-3. Further information can be found on the Hantronix Webseite http://www.hantronix.com and in the data sheet list on the CD-ROM.

The display is operated in the 4-Bit data mode. Data bits are set to the EXT-Data output, and then clocked into the 74HC164 shift register with triggering EXT-SCK. When LCD-E is set, the 4 Bits are applied to the display.

LCD Contrast (LCD ADJ)

Direct frontal view will allow best readability of the LCD characters. If necessary the contrast must be trivially re-adjusted. The contrast can be adjusted by means of potentiometer PT1.

Keyboard

For user inputs a 12 character keyboard (0..9,*,#) is provided (X15: 13 pole connector). The keyboard is organized 1 out of 12, i. e. there is one line assigned to each key. The keyboard information is read-in serially through a shift register. If no keyboard is used the 12 inputs can be used as additional digital inputs. The keyboard uses a 13 pole terminal (single row) and is plugged to X15 in such a way that the keys will point towards the application board.

With activating the PL (parallel load - KEY-E) input of the 74HC165 all 12 keyboard wires are transferred in the 74HC165 shift register. After that all information bits are latched to Q7 with triggering of CP (clock input - EXT-SCK). There they can be read with the EXT-Data Port. Since one 74HC165 holds only 8 Bit information, Q7 of the 1st 74HC165 is connected with DS of the 2nd 74HC165.

I2C Interface

Through this interface serial data can be transmitted at high speed. To do this only two signal lines are necessary. Data transmission takes place according to the I2C protocol. To effectively use this interface special functions are provided (see Software Description I2C).

I2C SCL	I2C Bus Clock Line	PortC.0
I2C SDA	I2C Bus Data Line	PortC.1

Power Supply (POWER, 5 Volts, GND)

Power is provided to the application board by means of a 9V/ 250mA Mains Plug-in Power Supply. Depending on additionally used components it may later become necessary to use a power supply with higher power rating. A fixed voltage control generates an internally stabilized 5V supply voltage. This voltage is provided to all circuit components on the application board. Due to the power reserve of the Plug-In Power Supply this voltage can also be used to power external ICs.

➡ Please observe the Maximum Drawable Current. Exceeding this current may lead to immediate destruction! Because of the relatively high current consumption of the application board in the vicinity of 125mA it is not recommended for use in devices consistently battery operated. Please see the note on short time breakdowns of the power supply (see Reset Characteristics).

➡ If you turn the application board to a position where the interface connectors (USB and serial) show to the upper side, the leftmost column on the breadboard is GND and the rightmost column is VCC.

Serial Interfaces

The Micro Controller Atmega32 contains in its hardware an asynchronous serial interface according to RS232 standards. The format (Data Bits, Parity Bit, Stop Bit) can be determined during initialization of the interface. The application board contains a high value level conversion IC to transform the digital bit streams to Non Return Zero Signals in accordance with the RS232 standards (positive voltage for low bits, negative voltage for high bits). The level conversion IC makes use of an improved protection against voltage transients. Voltage transients can in electro-magnetically loaded surroundings (e. g. in industrial applications) be induced in the interface cables and thus destroy connected electrical circuits. By means of jumpers the data lines RxD and TxD can be connected to the Controller PortD.0 and PortD.1. During quiescent condition (no active data transmission) a negative voltage of several volts can be measured on Pin TxD against GND. RxD is of high impedance. The 9 pole SUB-D socket of the application board carries RxD on Pin 3 and TxD on Pin 2. Pin 5 is the GND connection. No handshake signals are being used for serial data transmission.

The cable with connection to the NRZ Pins TxD, RxD and RTS may have a length of up to 10 meters. It is recommended to use shielded standard cables. When using longer lines or non-shielded cables interferences may detract correct data transmission. Only use cables of which the pin assignments are known.

➡️ Never connect the serial transmission outputs of two devices directly together! Transmission outputs can usually be identified by their negative output voltage in quiescent condition.

Testing Interfaces

The 4 pole pin strip X16 is to be used for testing purposes only and will not necessarily be armed with components of any kind on every application board. For the user this pin strip is of no importance.

One further testing interface is the 6 pole pin strip (two rows at 3 pins each) at JP4. This pin strip too is only meant for internal use and may likely no longer be fitted with components in future board series.

Technical Data Application Board

Note: Detailed information's can be found in the IC manufacturer's PDF files on the C-Control Pro Software CD-ROM.
All voltage specifications are referring to direct current (DC).

Mechanics	
Overall measurements, appr.	160 mm x 100 mm
Pin pitch wiring field	2.54 mm

Environmental Conditions	
Range of admissible ambient temperature	0°C … 70°C
Range of admissible relative ambient humidity	20% … 60%

Power Supply	
Range of admissibly operating voltage	8V … 24V
Power consumption without external loads	appr. 125mA
Max. admissibly permanent current from a stabilized 5V power supply	200mA

3.3.3 Pin Assignment

PortA through PortD are for direct pin functions (e. g. Port_WriteBit) counted from 0 through 31, see "PortBit".

Pin Assignment for Application Board Mega32

M32 PIN	Port	Port	PortBit	Name	Layout	Remarks
1	PB0	PortB.0	8	T0		Input Timer/Counter0
2	PB1	PortB.1	9	T1		Input Timer/Counter1
3	PB2	PortB.2	10	INT2/AIN0		(+)Analog Comparator, external Interrupt2
4	PB3	PortB.3	11	OT0/AIN1		(-)Analog Comparator, Output Timer0
5	PB4	PortB.4	12		SS	USB-Communication
6	PB5	PortB.5	13		MOSI	USB-Communication
7	PB6	PortB.6	14		MISO	USB-Communication
8	PB7	PortB.7	15		SCK	USB-Communication
9				RESET		
10				VCC		
11				GND		
12				XTAL2		Oscillator : 14,7456MHz
13				XTAL1		Oscillator : 14,7456MHz
14	PD0	PortD.0	24	RXD	EXT-RXD	RS232, serial Interface
15	PD1	PortD.1	25	TXD	EXT-TXD	RS232, serial Interface
16	PD2	PortD.2	26	INT0	EXT-T1	SW1 (Taster1); external Interrupt0
17	PD3	PortD.3	27	INT1	EXT-T2	SW2 (Taster2); external Interrupt1
18	PD4	PortD.4	28	OT1B	EXT-A1	Output B Timer1
19	PD5	PortD.5	29	OT1A	EXT-A2	Output A Timer1
20	PD6	PortD.6	30	ICP	LED1	LED; Input Capture Pin for Pulse/ Period Measurement
21	PD7	PortD.7	31		LED2	LED
22	PC0	PortC.0	16	SCL	EXT-SCL	I2C-Interface

23	PC1	PortC.1	17	SDA	EXT-SDA	I2C-Interface
24	PC2	PortC.2	18			
25	PC3	PortC.3	19			
26	PC4	PortC.4	20			
27	PC5	PortC.5	21			
28	PC6	PortC.6	22			
29	PC7	PortC.7	23			
30				AVCC		
31				GND		
32				AREF		
33	PA7	PortA.7	7	ADC7	RX_BUSY	ADC7 Input; USB-Communication
34	PA6	PortA.6	5	ADC6	TX_REQ	ADC6 Input; USB-Communication
35	PA5	PortA.5	5	ADC5	KEY_EN	ADC5 Input; LCD/Keyboard Interface
36	PA4	PortA.4	4	ADC4	LCD_EN	ADC4 Input; LCD/Keyboard Interface
37	PA3	PortA.3	3	ADC3	EXT_SCK	ADC3 Input; LCD/Keyboard Interface
38	PA2	PortA.2	2	ADC2	EXT_DATA	ADC2 Input; LCD/Keyboard Interface
39	PA1	PortA.1	1	ADC1		ADC1 Input
40	PA0	PortA.0	0	ADC0		ADC0 Input

3.3.4 Jumper Application Board

Jumper

By use of jumpers certain options can be selected. This applies to several ports which are provided with special functions (see Pin Assignment Table for M32). E. g. the serial interface is relized through Pins PortD.0 and PortD.1. If the serial interface is not being used then the corresponding jumpers can be removed and these pins will then be available for other functions. Besides the port jumpers there are additional jumpers which are described in the following.

Ports A through D

The ports available with the Mega32 Module are inscribed in this graph. Here the right side is connected to the module while the left side connects to the components of the application board. If any jumper is pulled then the connection to the application board is suspended. This may lead to obstructions of USB, RS232, etc. on the board.

JP1 and JP2

These jumpers are assigned to push buttons SW1 and SW2. There is the possibility to operate the push button against both GND or VCC. In the basic setting the push buttons are switching to GND.
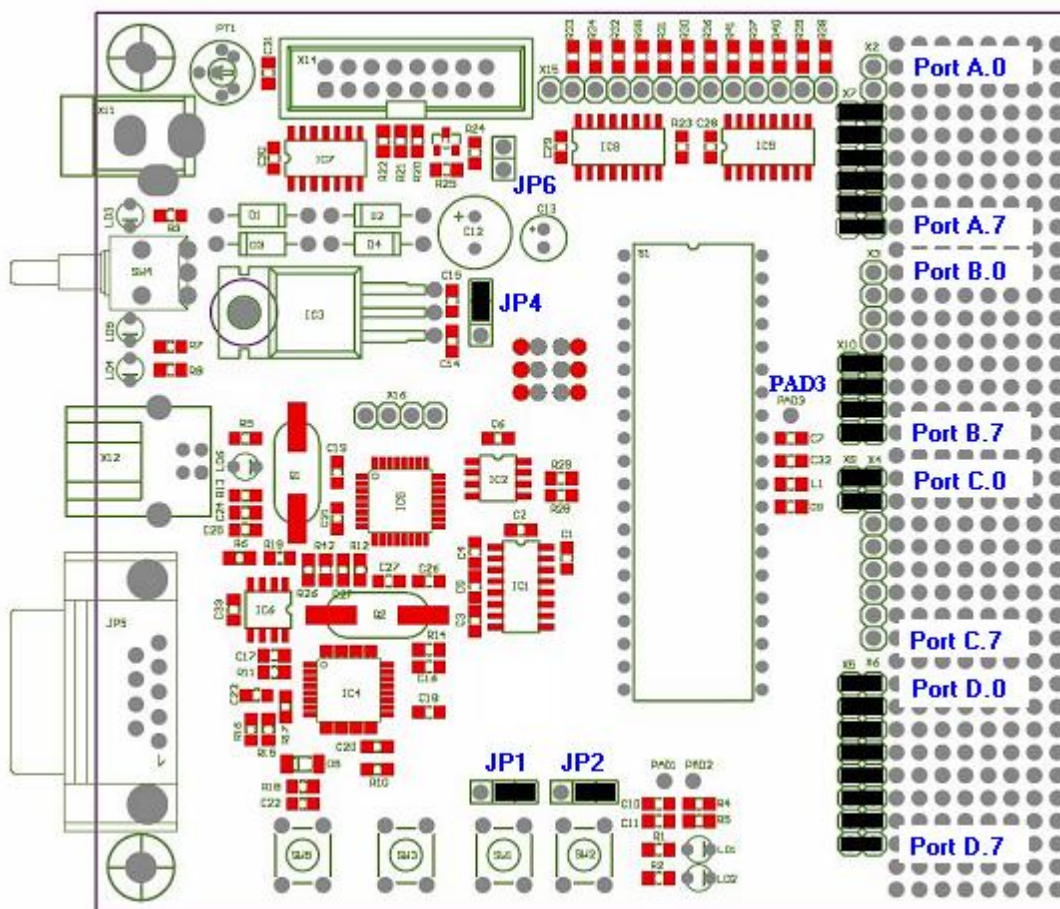
Jumperpositions at delivery

JP4

JP4 serves to toggle the operating voltage (Mains Plug-In Power Supply or USB). The application board should be operated using Plug-In Power Supply and voltage control (Shipping Condition). The maximum current to be drawn from the USB interface is lower than from the Plug-In Power Supply. Exceeding this current can lead to damage on the USB interface of the computer.

JP6

When using the displays the LED back lighting can be switched off by use of JP6.

PAD3

PAD3 (to the right of the module, below the blue inscription) is required as ADC_VREF_EXT for functions ADC_Set and ADC_SetInt.

3.3.5 Connection Diagrams

Additionally the connection diagrams can be found as PDF files on the Installation CD-ROM.

3.3.5.1 Mega 32 Modul

3.3.5.2 Applicationboard
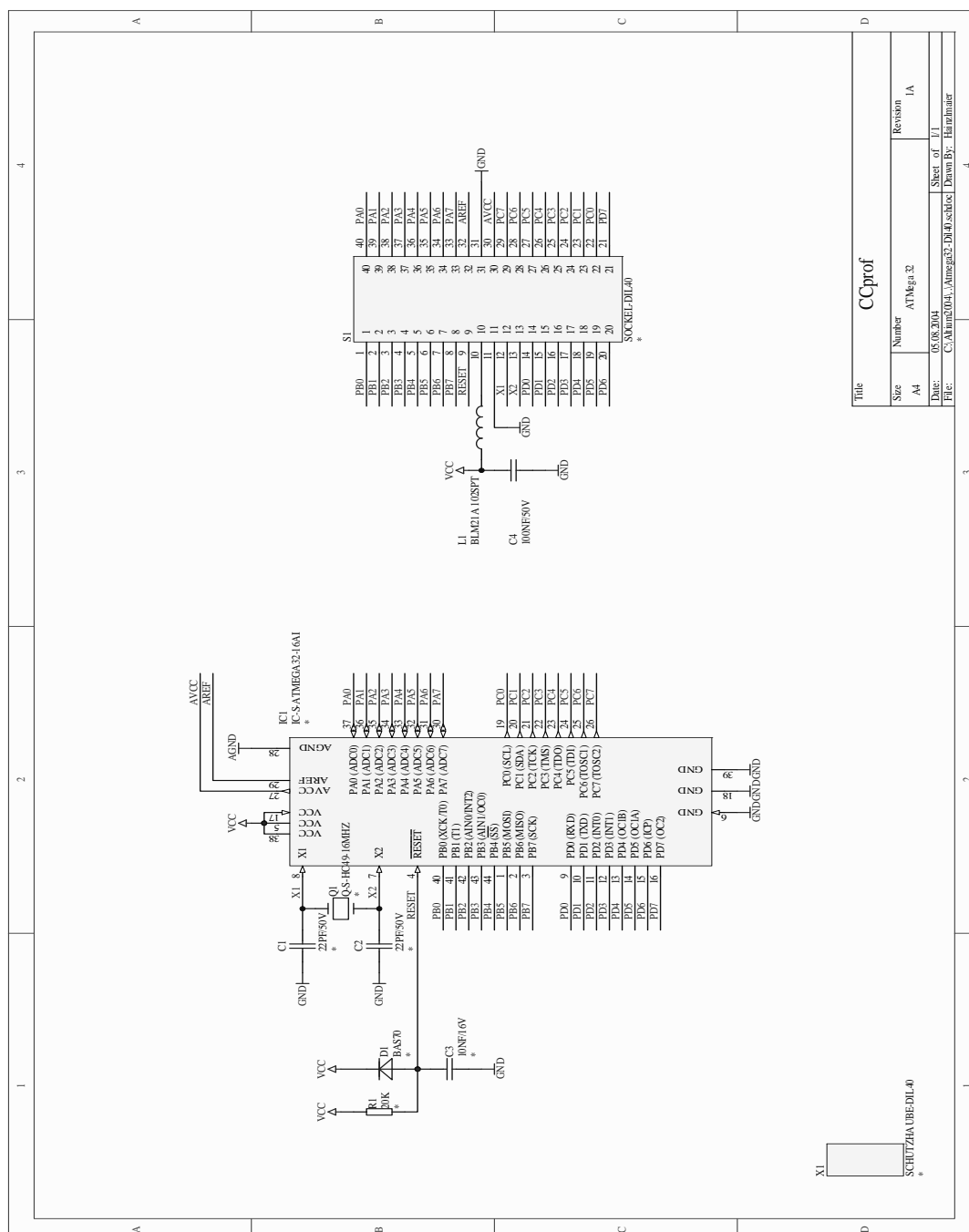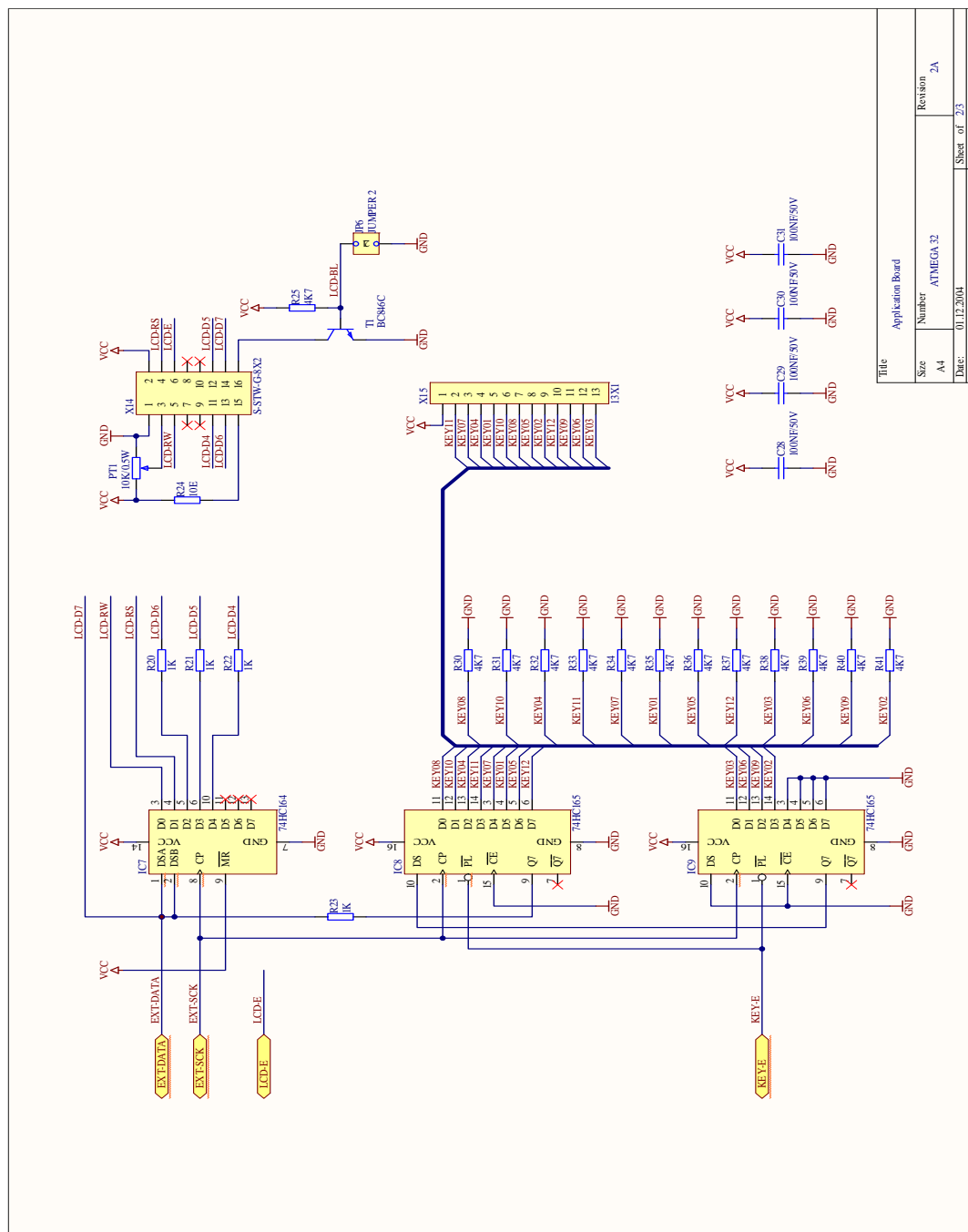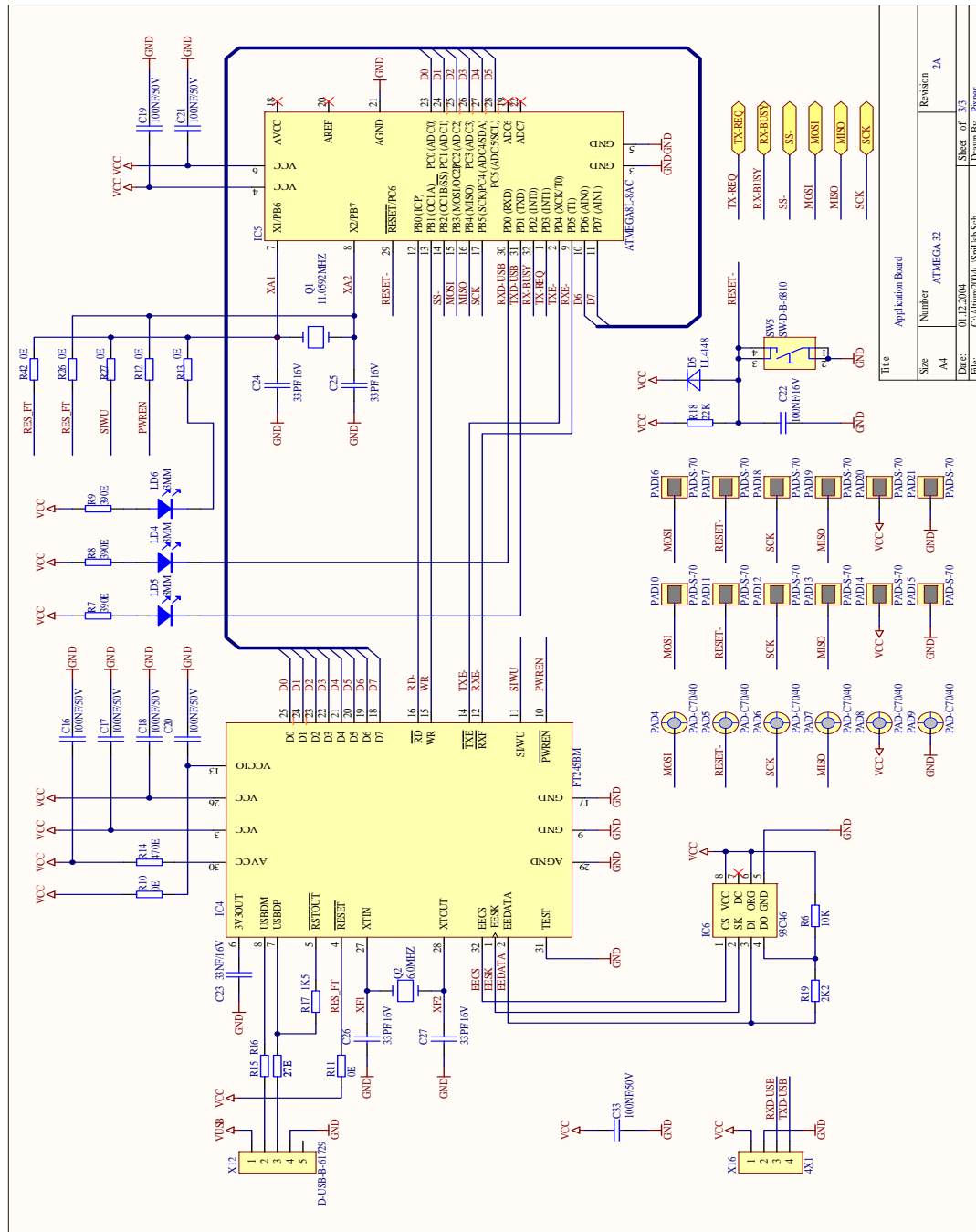
3.3.5.3 Component Parts Plan
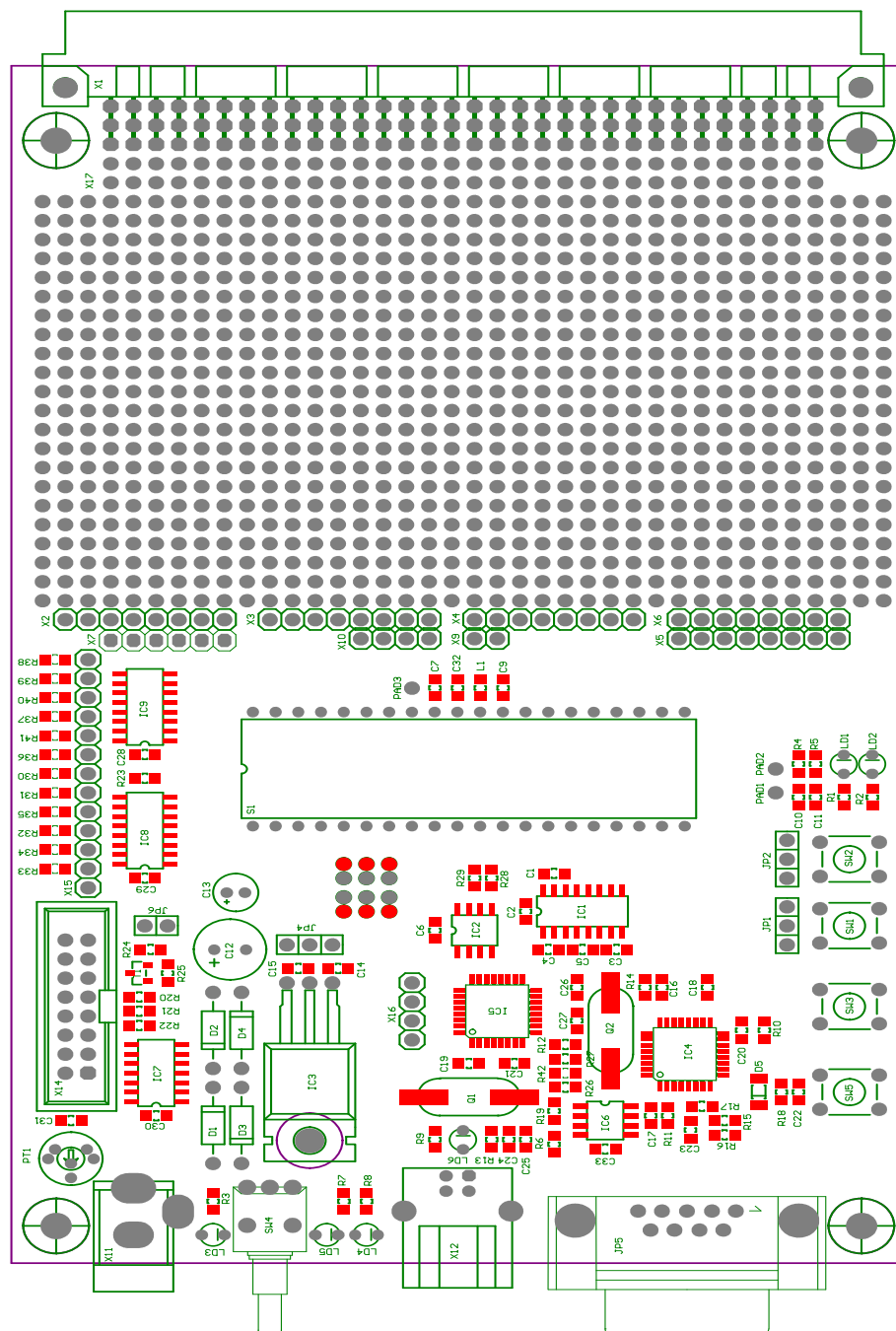
3.4   Mega128

The Micro Controller Atmega128 originates from the AVR family by ATMEL. It is a low-power Micro Controller with Advanced RISC Architecture. In the following see a short summary of its hardware resources:

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

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

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

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

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

- **Special Microcontroller Features**
  Power-on Reset and Programmable Brown-out Detection
  Internal Calibrated RC Oscillator
  External and Internal Interrupt Sources

Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
Software Selectable Clock Frequency
ATmega103 Compatibility Mode Selected by a Fuse
Global Pull-up Disable

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

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

3.4.1 Module

Pin Layout of the Module

The Mega128 Module is shipped on 4 dual row (2x8) square pins. For hardware application the corresponding socket strips must be organized in the following pitch format:
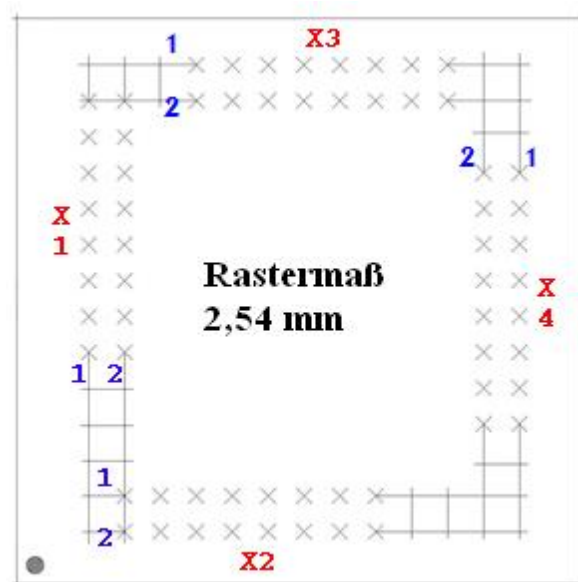
In the graph the socket strip X1-X4 and then the first two pins of the socket strip can be seen. Pin 1 of strip X1 corresponds to terminal X1_1 (see Mega128 Pinzuordnung).

Module Memory

The C-Control Pro 128 Module provides 128kB FLASH, 4kB EEPROM and 4kB SRAM. A supplementary EEPROM with an 8kB memory depth and an SRAM with a 64kB memory depth is mounted on the application board. The EEPROM can be addressed by an I2C interface.

Note: Detailed information can be found in the IC manufacturer's PDF files on the C-Control Pro Software CD-ROM.

ADC Reference Voltage Generation

The Micro Controller is equipped with an analog-to-digital converter with a 10 Bit resolution. This means that measured voltages can be represented by integral numbers from 0 through 1023. The reference voltage for the lower limit is GND level, i. e. 0V. The reference voltage for the upper limit can be selected by the user:

- 5V Operating Voltage (VCC)
- Internal Reference Voltage of 2.56V
- External Reference Voltage e. g. 4.096V generated by a Reference Voltage IC.

If *x* is a digital measuring value then the corresponding voltage value u is computed as follows:

u = *x* * Reference Voltage / 1024

Clock Generation

Clock generation takes place by a 14.7456MHz Quartz Oscillator. All time dependent actions within the controller are derived from this clock frequency.

Reset

A Reset initiates the return of the Micro Controller system to a defined starting condition. In gerneral the C-Control Pro Module knows two reset sources:

- Power-On-Reset: is automatically executed after the operating voltage is switched on.
- Hardware-Reset: is executed when the Module's RESET (X2_3) is pulled to "low" and released again by e. g. shortly pressing the connected Reset push button RESET1 (SW3).

A "Brown-Out-Detection" avoids that the Controller can enter undefined conditions in case of dropping operating voltages.

Digital Ports (PortA, PortB, PortC, PortD, PortE, PortF, PortG)

The C-Control Pro Module provides 6 digital ports at 8 pins each and one digital port with 5 pins. To the digital ports it is possible to connect e. g. push buttons with pull-up resistors, digital IC's, opto couples or driver circuits for relais. The ports can be addressed either separatly, i.e. pin by pin or byte by byte. Each pin can either be input or output.

➡ Note: Never connect two ports directly together which should simultaneously work as outputs!

Digital input pins are high-impedance or wired to internal pull-up resistors and transform an applied voltage signal into a logical value. For this it is required that the voltage signal is within the limits defined for TTL and CMOS ICs high or low levels. During further processing in the program the logical values on the respective input ports are represented as 0 ("low") oder -1 ("high). Pins will take on the values 0 or 1, Bytes from 0 to 255. Output ports are able to give out digital voltage signals by use of

an internal driver circuit. Connected circuits can draw (at high level) or feed (at low level) a specific current from or to the ports

➡ Pay attention to the Maximum Admissible Load Current for a single port or for all ports in total. Exceeding the maximal values may lead to destruction of the C-Control Pro Module. After a reset each port is initially configured as input port. By certain commands the direction of data transport can be toggled.

➡ It is important to closely study the pin assignment of M32 and M128 prior to programming since important functions of the program design (e. g. the USB interface of the application board) will apply to specific ports. If these ports are re-programmed or if the matching jumpers on the application board are no longer set then it may happen that the design platform can no longer transfer any programs to the C-Control Pro. Timer inputs and outputs, A/D converter, I2C as well as serial interface are also connected to various port pins.

PLM Ports

There are three timers available for PLM. These are *Timer_0* with 8 bits and *Timer_1* as well as *Timer_3* with 16 bits each. They can be used for D/A conversion, to control servo motors in pattern making and to output audio frequencies. A pulse length modulated signal has a period of N so called "Ticks". The duration of one tick is the time base. If the output value of a PLM port is set to X then the port will hold high level for X ticks of one period and will then for the balance of the period drop to low level. For programming of the PLM channels see Timer.

The PLM channels for *Timer_0*, *Timer_1* and *Timer_3* have independent time base and period length. In applications for pulse width modulated digital to analog conversion the time base and period length are set once and then only the output value is varied. According to their electrical properties the PLM ports are digital ports. Please observe the technical boundary conditions for digital ports (max. current).

Technical Data Module

Note: Detailed information can be found in the IC manufacturer's PDF files on the C-Control Pro Software CD-ROM.

All voltage specifications apply to direct current (DC).

Environmental Conditions	
Range of admissible ambient temperature	0°C … 70°C
Range of admissible relative ambient humidity	20% … 60%
Power Supply	
Range of admissible operating voltage	4.5V … 5.5V
Power consumption of the module without external loads	appr. 20mA

Clock	
Clock Frequency (Quartz Oscillator)	14.7456MHz
Mechanics	
Overall measurements less pins, appr.	40 mm x 40mm x 8 mm
Weight	appr. 90g
Pin pitch	2.54mm
Number of pins (two rows)	64

Ports	
Max. admissible current from digital ports	± 20 mA
Admissible current total on digital ports	200mA
Admissible input voltage on port pins (digital and A/D)	–0.5V ... 5.5V
Internal pull-up resistors (disconnectable)	20 - 50 kOhm

3.4.2 Application Board

USB

The application board provides a USB interface for the program's loading and debugging. Because of the high data rate of this interface data transmission times are considerably shorter compared to the serial interface. Communication takes place through a USB Controller by FTDI and an AVR Mega8 Controller. The Mega8 provides its own Reset push button (SW5). During USB operation the status of the interface is indicated by two light emitting diodes (LD4 red, LD5 green). When only the green LED lights up the USB interface is ready for operation. During data transmission both LEDs will light up. This also applies to the Debug mode. Flashing of the red LED indicates an error condition. For USB communication the SPI interface of Mega128 is used (PortB.0 through PortB.4, PortE.5), which must be connected by their respective jumpers.

Note: Detailed information on the Mega8 can be found in the IC manufacturer's PDF files on the C-Control Pro Software CD-ROM.

On-Off Switch

The switch SW4 is located on the front of the application board and serves the power-up (On) or power-down (Off) of the voltage supply.

Light Emitting Diodes (LED)

There are 5 light emitting diodes (LEDs). The LD3 (green) is located on the front below the DC terminals and lights up when supply voltage is applied. LD4 and LD5 indicate the status of the USB interface (see Section USB). The green LEDs LD1 and LD2 are located next to the four push buttons and are freely available to the user. They are connected to VCC through a dropping resistor. By means of jumpers LD1 can be connected to PortG.3 and LD2 to PortG.4. The LEDs will light up when the corresponding pin port is low (GND).

Push Buttons

There are four push buttons provided for. SW3 (RESET1) will initiate a reset with Mega128 while SW5 (RESET2) will do the same with Mega8. The push button SW1 and SW2 are freely available to the user. Through jumpers SW1 can be connected to PortE.4 and accordingly SW2 to PortE.6. There is the possibility to connect switches SW1/2 to either GND or VCC. The possibilities to choose from are determined by JP1 and JP2 resp. In order to have a defined level at the input port while the push button is open the corresponding pull-up should be switched on (see Section Digitalports).

➡ Pressing SW1 during power-up of the board will activate the Serial Bootloader Mode.

LCD

An LCD module can be plugged onto the application board. It displays 2 lines at 8 characters each. In general also differently organized displays can be operated through this interface. Each character consists of a monochrome matrix of 5x7 pixels. A flashing cursor below any one of the characters will indicate the current output position. The operating system provides a simple software interface for output on the display. This display is connected to connector X14 (16 poles, double row). By means of a mechanical protection a faulty connection and thus the confusing of poles is avoided.

The LCD module used is of type Hantronix HDM08216L-3. Further information can be found on the Hantronix Webseite http://www.hantronix.com and in the data sheet list on the CD-ROM.

The display is operated in the 4-Bit data mode. Data bits are set to the EXT-Data output, and then clocked into the 74HC164 shift register with triggering EXT-SCK. When LCD-E is set, the 4 Bits are applied to the display.

LCD Contrast (LCD ADJ)

Direct frontal view will allow best readability of the LCD characters. If necessary the contrast must be trivially re-adjusted. The contrast can be adjusted by means of potentiometer PT1.

Keyboard

For user inputs a 12 character keyboard (0..9,*,#) is provided (X15: 13 pole connector). The keyboard is organized 1 out of 12, i. e. there is one line assigned to each key. The keyboard information is read-in serially through a shift register. If no keyboard is used the 12 inputs can be used as additional digital inputs. The keyboard uses a 13 pole terminal (single row) and is plugged to X15 in such a way that the keys will point towards the application board.

With activating the PL (parallel load - KEY-E) input of the 74HC165 all 12 keyboard wires are transferred in the 74HC165 shift register. After that all information bits are latched to Q7 with triggering of CP (clock input - EXT-SCK). There they can be read with the EXT-Data Port. Since one 74HC165 holds only 8 Bit information, Q7 of the 1st 74HC165 is connected with DS of the 2nd 74HC165.

SRAM

The application board holds an SRAM chip (K6X1008C2D) made by Samsung. By using this the available SRAM memory is extended to 64kByte. Mentioned SRAM uses Ports A, C and partly G for triggering. If the SRAM is not used then it can be de-activated by JP7 and then these ports become available to the user.

➡ To deactivate the SRAM the jumper JP7 has to be moved to the left side (orientation: serial interface shows to the left), such that the left pins of JP7 are connected.
➡ Even though the used RAM chip has a capacity of 128kb only 64kb can be used for reason of the memory model.

I2C Interface

Through this interface serial data can be transmitted at high speed. To do this only two signal lines are necessary. Data transmission takes place according to the I2C protocol. To effectively use this interface special functions are provided (see Software Description I2C).

I2C SCL	I2C Bus Clock Line	PortD.0
I2C SDA	I2C Bus Data Line	PortD.1

Power Supply (POWER, 5 Volt, GND)

Power is provided to the application board by means of a 9V/ 250mA Mains Plug-in Power Supply. Depending on additionally used components it may later become necessary to use a power supply with higher power rating. A fixed voltage control generates an internally stabilized 5V supply voltage. This voltage is provided to all circuit components on the application board. Due to the power reserve of the Plug-In Power Supply this voltage can also be used to power external ICs.

➡ Please observe the Maximum Drawable Current. Exceeding this current may lead to immediate destruction! Because of the relatively high current consumption of the application board in the vicinity of 125mA it is not recommended for use in devices consistently battery operated. Please see the note on short time breakdowns of the power supply (see Reset Characteristics).

➡ If you turn the application board to a position where the interface connectors (USB and serial) show to the upper side, the leftmost column on the breadboard is GND and the rightmost column is VCC.

Serial Interfaces

The Micro Controller Atmega128 contains in its hardware two asynchronous serial interfaces according to RS232 standards. The format (Data Bits, Parity Bit, Stop Bit) can be determined during

initialization of the interface. The application board contains a high value level conversion IC for both interfaces to transform the digital bit streams to Non Return Zero Signals in accordance with the RS232 standards (positive voltage for low bits, negative voltage for high bits). The level conversion IC makes use of an improved protection against voltage transients. Voltage transients can in electro-magnetically loaded surroundings (e. g. in industrial applications) be induced in the interface cables and thus destroy connected electrical circuits. By means of jumpers the data lines RxD0 (PortE.0), TxD0 (PortE.1) and RxD1 (PortD.2), TxD1 (PortD.3) can through the Controller be connected to the level converter. During quiescent condition (no active data transmission) a negative voltage of several volts can be measured on Pin TxD against GND. RxD is of high impedance. The 9 pole SUB-D socket of the application board carries RxD0 on Pin 3 and TxD0 on Pin 2. Pin 5 is the GND connection. No handshake signals are being used for serial data transmission. The second serial interface is lead to a 3 pole pin strip. Here RxD1 occupies Pin 2, TxD1 occupies Pin 1 while Pin 3 is GND.

The cable with connection to the NRZ Pins TxD, RxD and RTS may have a length of up to 10 meters. It is recommended to use shielded standard cables. When using longer lines or non-shielded cables interferences may detract correct data transmission. Only use cables of which the pin assignments are known.

➡ Never connect the serial transmission outputs of two devices directly together! Transmission outputs can usually be identified by their negative output voltage in quiescent condition.

Testing Interfaces

The 4 pole pin strip X16 is to be used for testing purposes only and will not necessarily be armed with components of any kind on every application board. For the user this pin strip is of no importance.

One further testing interface is the 6 pole pin strip (two rows at 3 pins each) at the lower right next to JP4. This pin strip too is only meant for internal use and may likely no longer be fitted with components in future board series.

Technical Data Application Board

Note: Detailed information's can be found in the IC manufacturer's PDF files on the C-Control Pro Software CD-ROM.
All voltage specifications are referring to direct current (DC).

Mechanics	
Overall measurements, appr.	160 mm x 100 mm
Pin pitch wiring field	2.54 mm
Environmental Conditions	
Range of admissible ambient temperature	0°C … 70°C
Range of admissible relative ambient humidity	20% … 60%

Power Supply	
Range of admissibly operating voltage	8V … 24V
Power consumption without external loads	appr. 125mA
Max. admissibly permanent current from a stabilized 5V power supply	200mA

3.4.3 Pin Assignment

PortA through PortG are for direct pin functions (e. g. Port_WriteBit) counted from 0 through 52, see "PortBit".

Pin Assignment for Application Board Mega128

Module	M128	Port	Port #	PortBit	Name1	Name2	Internal	Remarks
	1				PEN			prog. Enable
X1_16	2	PE0	4	32	RXD0	PDI	EXT-RXD0	RS232
X1_15	3	PE1	4	33	TXD0	PDO	EXT-TXD0	RS232
X1_14	4	PE2	4	34	AIN0	XCK0		Analog Comparator
X1_13	5	PE3	4	35	AIN1	OC3A		Analog Comparator
X1_12	6	PE4	4	36	INT4	OC3B	EXT-T1	Switch 1
X1_11	7	PE5	4	37	INT5	OC3C	TX-REQ	SPI_TX_REQ
X1_10	8	PE6	4	38	INT6	T3	EXT-T2	Switch 2 / Input Timer 3
X1_9	9	PE7	4	39	INT7	IC3	EXT-DATA	LCD_Interface
X1_8	10	PB0	1	8	SS			SPI
X1_7	11	PB1	1	9	SCK			SPI
X1_6	12	PB2	1	10	MOSI			SPI
X1_5	13	PB3	1	11	MISO			SPI
X1_4	14	PB4	1	12	OC0		RX-BUSY	SPI_RX_BUSY
X1_3	15	PB5	1	13	OC1A		EXT-A1	DAC1
X1_2	16	PB6	1	14	OC1B		EXT-A2	DAC2
X1_1	17	PB7	1	15	OC1C	OC2	EXT-SCK	LCD_Interface
X2_5	18	PG3	6	51	TOSC2		LED1	LED
X2_6	19	PG4	6	52	TOSC1		LED2	LED
X2_3	20				RESET			
X4_10	21				VCC			
X4_12	22				GND			
	23				XTAL2			Oscillator
	24				XTAL1			Oscillator
X2_9	25	PD0	3	24	INT0	SCL	EXT-SCL	I2C
X2_10	26	PD1	3	25	INT1	SDA	EXT-SDA	I2C
X2_11	27	PD2	3	26	INT2	RXD1	EXT-RXD1	RS232
X2_12	28	PD3	3	27	INT3	TXD1	EXT-TXD1	RS232
X2_13	29	PD4	3	28	IC1	A16		IC Timer 1, SRAM bank select
X2_14	30	PD5	3	29	XCK1		LCD-E	LCD_Interface
X2_15	31	PD6	3	30	T1			Input Timer 1
X2_16	32	PD7	3	31	T2		KEY-E	LCD_Interface / Input

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

3.4.4   Jumper Application Board

Jumper

By use of jumpers certain options can be selected. This applies to several ports which are provided with special functions (see Pin Assignment Table for M128). E. g. the serial interface is realized through Pins PortE.0 and PortE.1. If the serial interface is not being used then the corresponding jumpers can be removed and these pins will then be available for other functions. Besides the port jumpers there are additional jumpers which are described in the following.
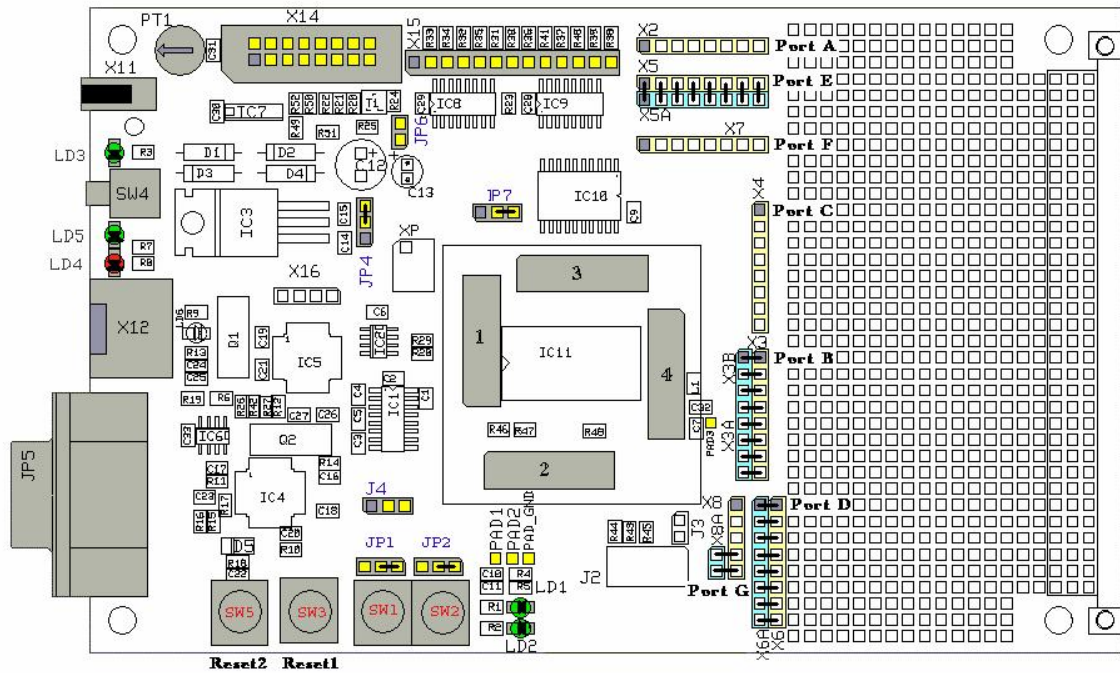
Jumperpositionen im Auslieferzustand

Ports A through G

The ports available with the Mega128 Module are inscribed in this graph. Here the yellow side is connected to the module while the light blue side connects to the components of the application board. If any jumper is pulled then the connection to the application board is suspended. This may lead to obstructions of USB, RS232, etc. on the board. The gray marking indicates the first Pin (Pin 0) of the Port.

JP1 and JP2

These jumpers are assigned to push buttons SW1 and SW2. There is the possibility to operate the push button against both GND or VCC. In the basic setting the push buttons are switching to GND.

JP4

JP4 serves to toggle the operating voltage (Mains Plug-In Power Supply or USB). The application board should be operated using Plug-In Power Supply and voltage control (Shipping Condition). The maximum current to be drawn from the USB interface is lower than from the Plug-In Power Supply. Exceeding this current can lead to damage on the USB interface of the computer.

JP6

When using the displays the LED back lighting can be switched off by use of JP6.

JP7

If the SRAM on the application board is not needed it can be de-activated by use of JP7. These ports will then be available to the user.

➡ To deactivate the SRAM the jumper has to be moved to the left side (orientation: serial interface shows to the left), such that the left pins of JP7 are connected.

J4

To jumper J4 the second serial interface of the Mega128 is connected through a level converter.

Pin 1 (left, gray)	**TxD**
Pin 2 (center)	**RxD**
Pin 3 (right)	**GND**

PAD3

PAD3 (to the right of the module) is required as ADC_VREF_EXT for functions ADC_Set and ADC_SetInt.

3.4.5 Connection Diagrams

Additionally the connection diagrams can be found as PDF files on the Installation CD-ROM.

3.4.5.1 Mega 128 Module

➡ The shown connection diagram shows the planned C-Control Pro Module with CAN Bus interface. This Module has not been built. Inside the C-Control Pro 128 Module is working a Mega 128 processor, and not a AT90CAN128 like shown in this diagram. Therefore there is also no ATA6660 CAN-Bus Transceiver inside the C-Control Module.

schäffel electronic gmbh

Project: MEGA128V2

PCB-Design: MEGA128 MODUL 2/3

10mm   Sheet 1   of 1

3.4.5.2 Applicationboard

schäffel electronic gmbh

Project: mega128app_v2

PCB-Design: MEGA Appl.-Board

10mm | Sheet 3 of 4

schäffel electronic gmbh

Project: mega128app_v2

PCB-Design: MEGA Appl.-Board

10mm | Sheet 4 of 4
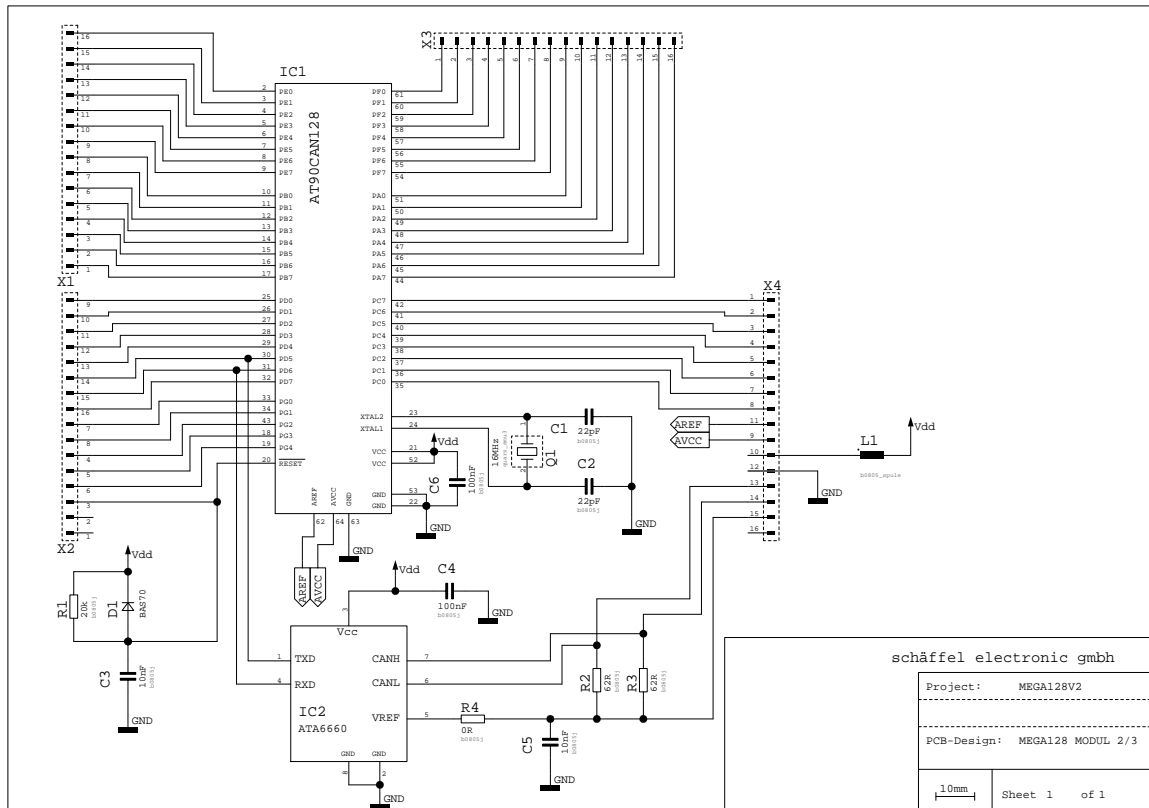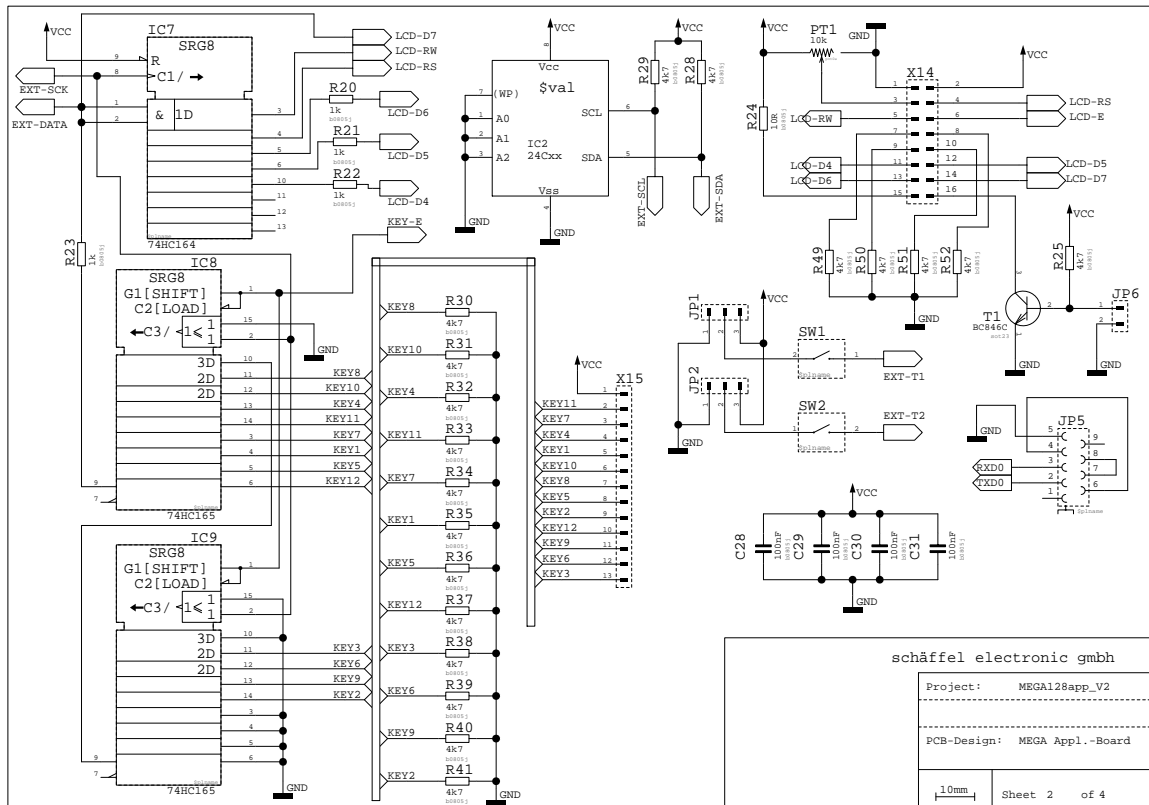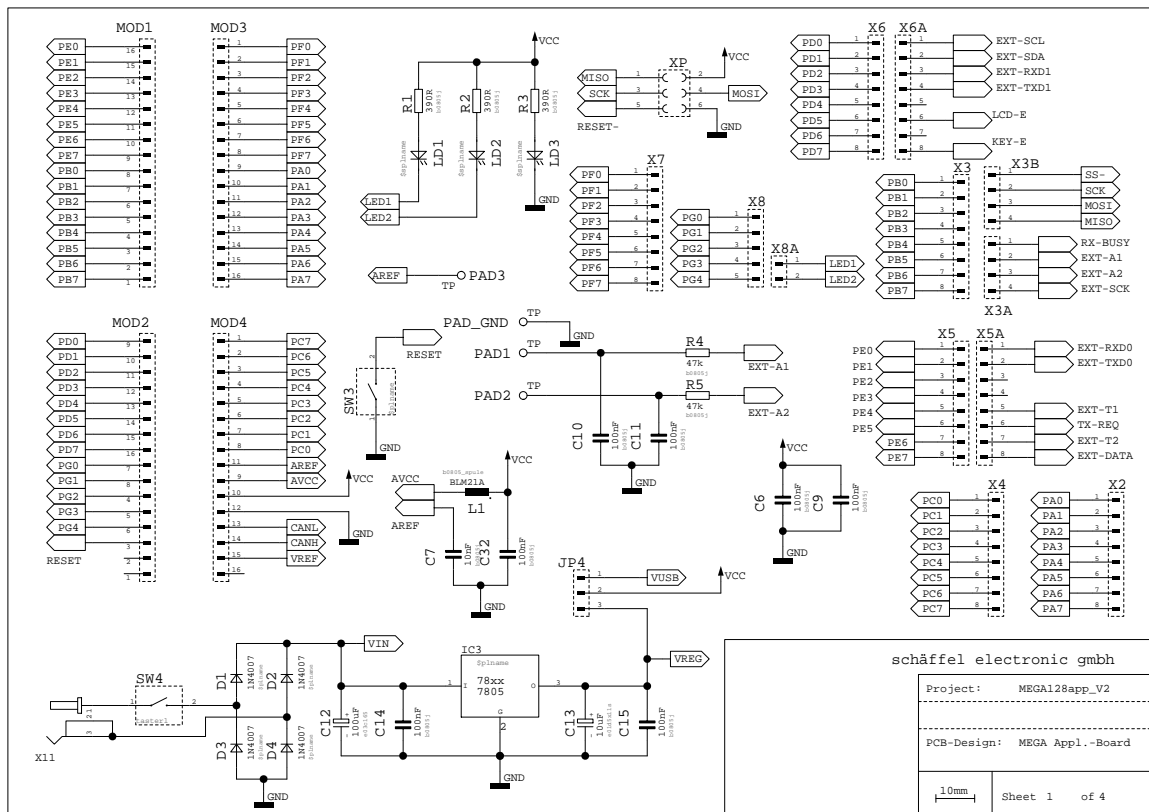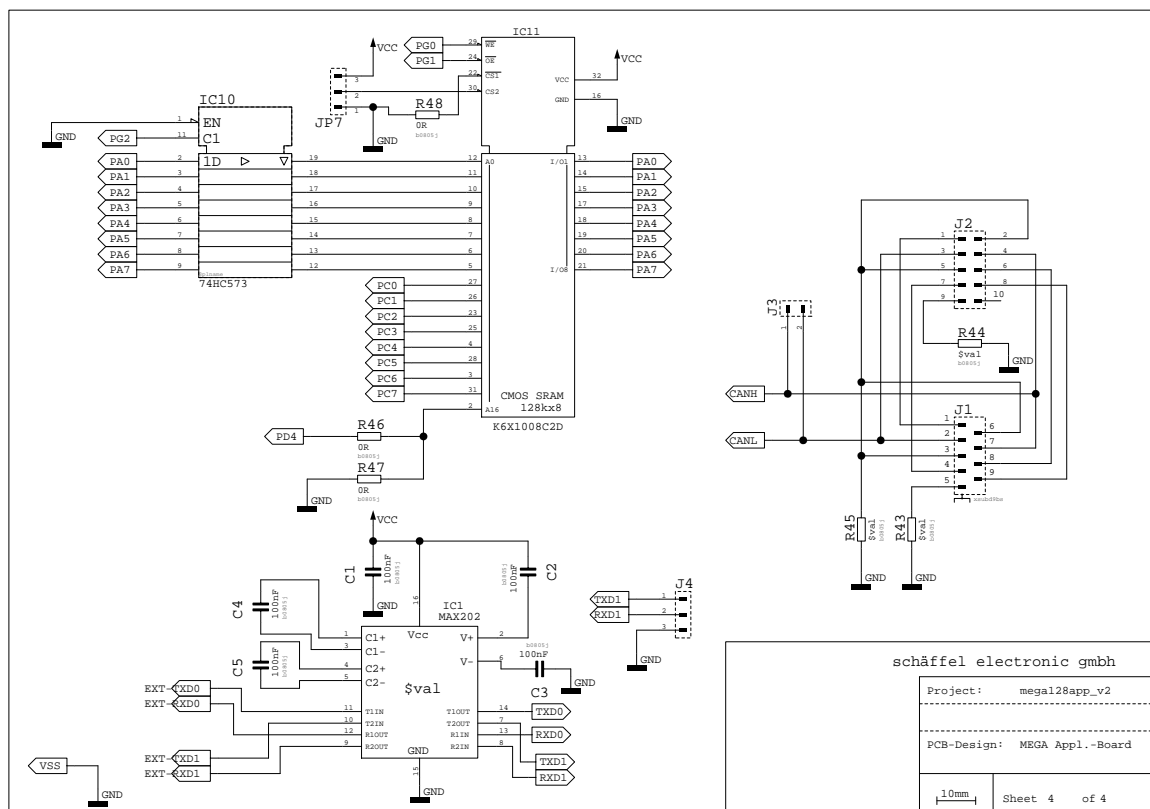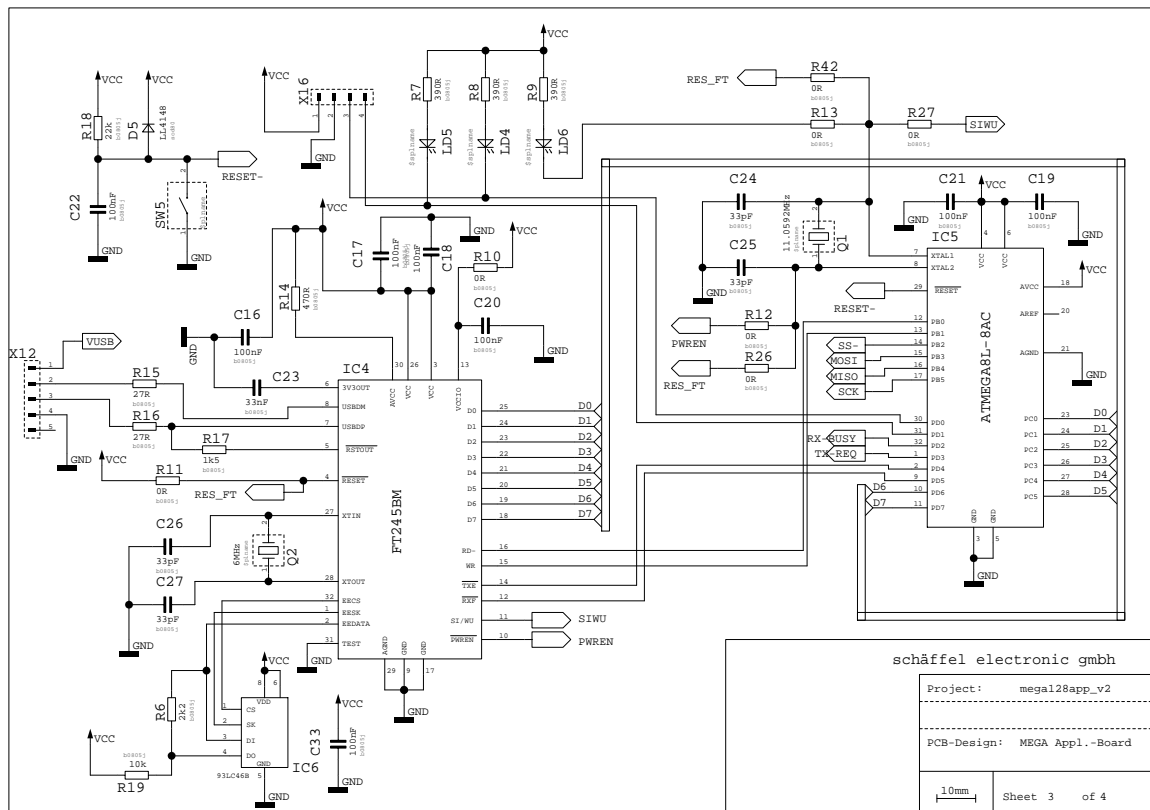
3.4.5.3 Component Parts Plan

Part

4
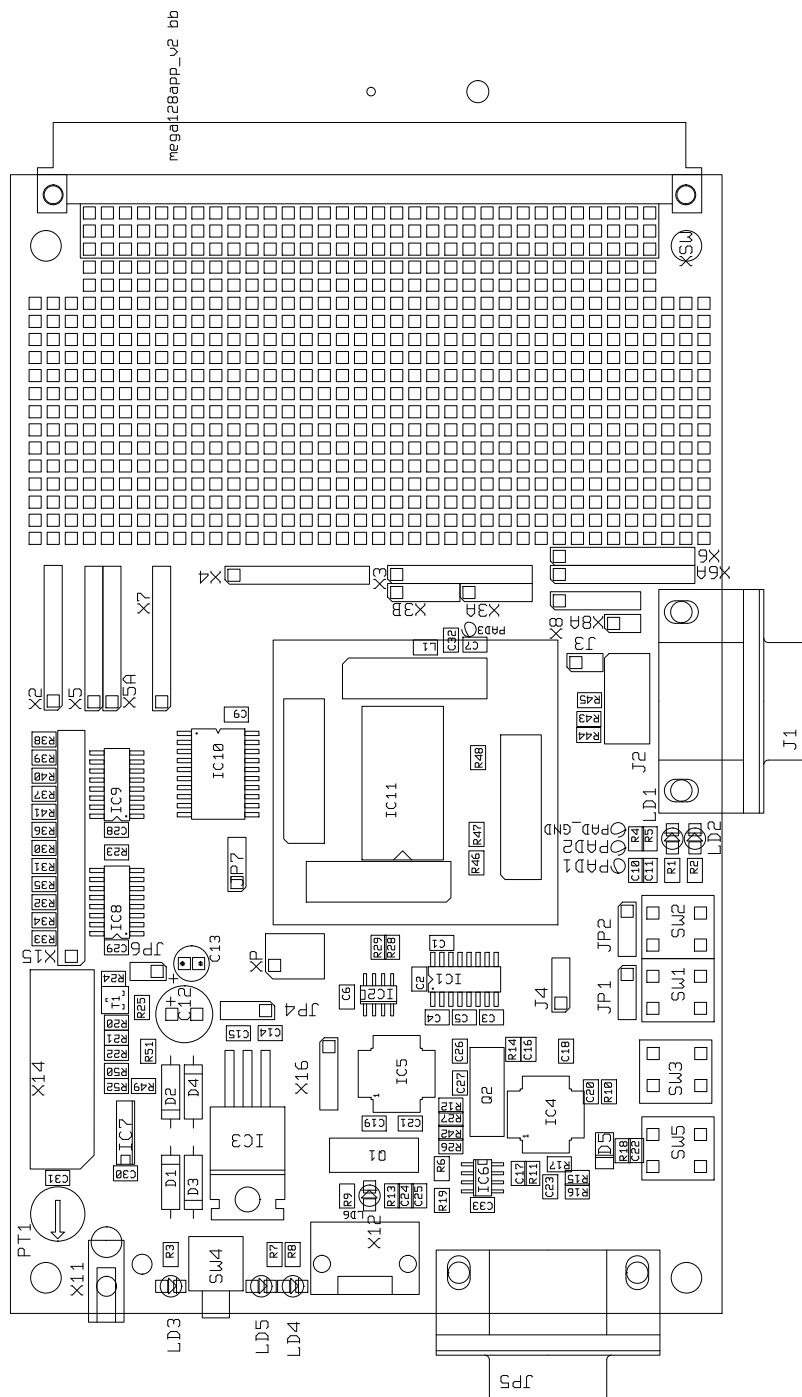
4   IDE

The C-Control Pro User Interface (IDE) consists of the following main elements:

Sidebar for Project Files	Here several files can be filed to form a project
Editor Window	In order to edit files as many editor windows as necessary can be opened.
Compiler Messages	Here error messages and general compiler informations are displayed
C-Control Outputs	Distribution of the CompactC program's debug messages
Variables Window	Here monitored variables are displayed
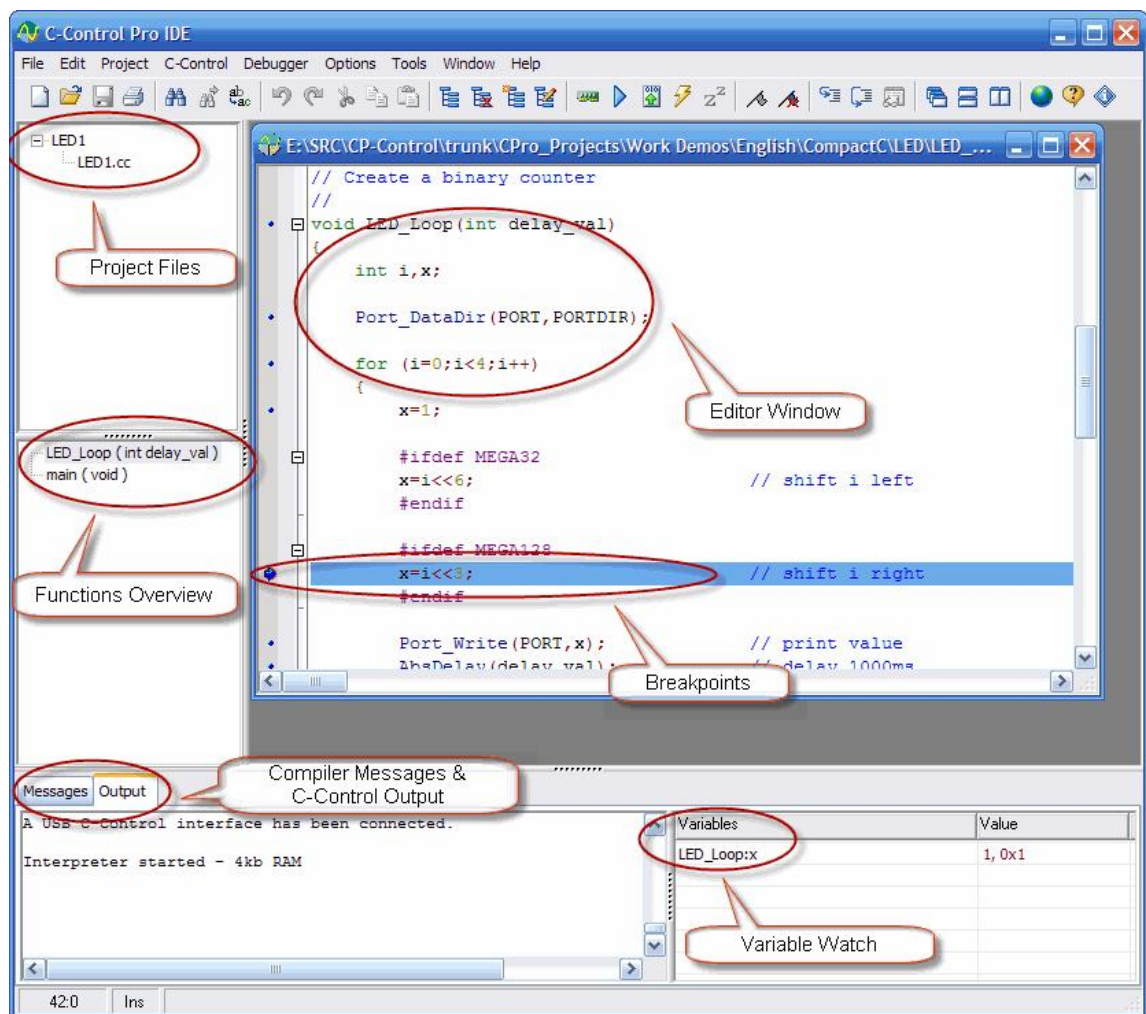
4.1 Projects

Every program for the C-Control Pro Module is configured through a project. The project states which source files and libraries are being utilized. Also the settings of the Compiler are noted. A project consists of the project file with the extension ".cprj" and the appropriate source files.

4.1.1 Create Projects

In the menu Project the dialog box Create Project can be opened by use of item New. Here a project name is issued for the project. Then the project is created in the sidebar.

➡️ It is not necessary to decide in advance whether a CompactC or a BASIC project will be created. In a project CompactC or BASIC files can be arranged combined as project files in order to create a program. The source text files in a project will determine which programming language will be used. Files with the extension "*.cc" will run in a CompactC context while files with the extension "*.cbas" are translated into BASIC.
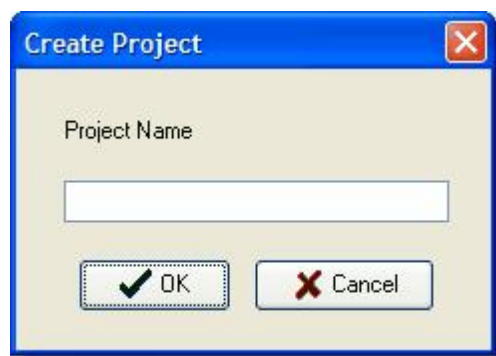
4.1.2 Compile Projects
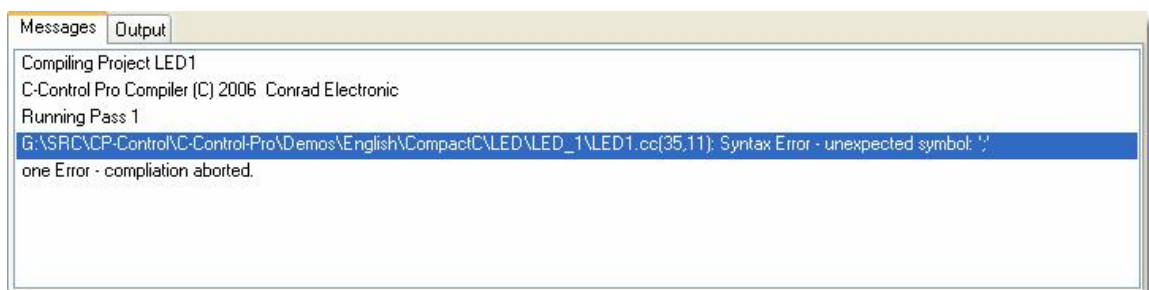
In menu Project the current project can be translated by the Compiler by use of Compile (F9). The Compiler messages are displayed in a separate window section. If errors arise during compilation then one error will be described per line. The form is:

```
File Name(Line,Column): Error Description
```

The error positions can be found in the source text by use of commands Next Error (F11) or Previous Error (Shift-F11). Both commands are found in menu item Project. Alternative the cursor can in the Editor be placed onto the error position by use of a double mouse click on the Compiler's error message.

After successful compilation the Byte Code will be filed in the project list as file with the extension "*.bc".

By a right mouse click in the area of the compiler messages the following actions can be initiated:

- delete – will delete the list of compiler messages
- copy to clipboard – will copy all text messages onto the clipboard

4.1.3 Project Management

A right mouse click on the newly created project in the sidebar will open a pop-up menu with the following options:

- Newly Add – A new file will be set up and simultaneously an editor window will be opened.
- Add – An existing file will be attached to the project.
- Rename – The name of the project will be changed (This is not necessarily the name of the project file).
- Compile – The compiler for the project is started.
- Options – The project options can be changed.

Adding of Project Files

When clicking Add project file the file Open Dialog will appear. Here the files to be added to the project can be selected. Any number of files can be selected.

Alternative by use of Drag&Drop files from the Windows Explorer can be transferred into the project management.

Project Files

When files have been added to the project these can be opened by a double mouse click onto the file name. By use of a right click further options will appear:

Up
Down
Rename
Remove
Options
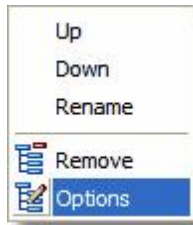
- Up – The project file will move up the list (also with Ctrl - Arrow up).
- Down – The project file will move down (also with Ctrl - Arrow down).
- Rename – The name of the project file will be changed.
- Delete – The file will be deleted from the project.
- Options – The project options can be changed.

4.1.4   Thread Options

In order to activate a Thread to runtime it must be activated in the selection box and the parameters *Stack Size* and *Cycle Time* must be set.
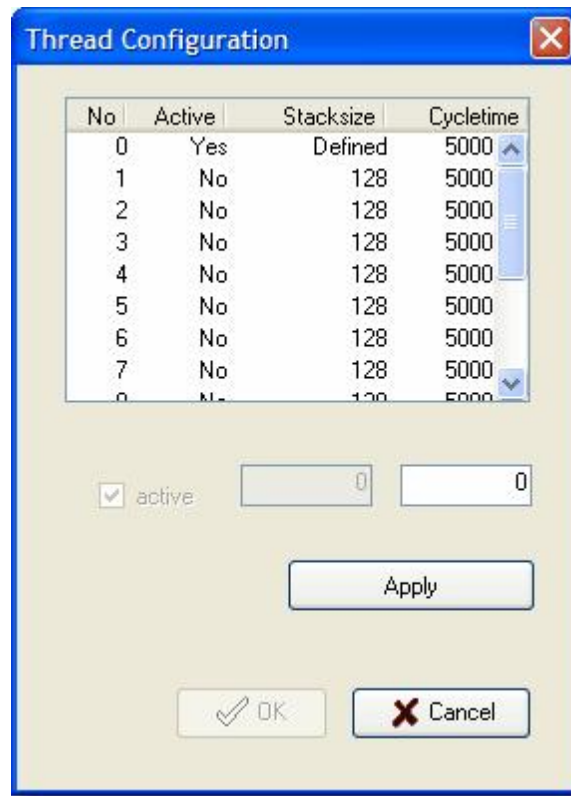
To each additional Thread, besides the main program, an area on the stack will be assigned which must not be exceeded.

➡ If a Thread uses more space than assigned to then the memory space of the other Threads is affected and a program breakdown may occur.

Cycle time is the number of cycles (Bytecode Operations) which a Thread can process before a change to another Thread takes place. By use of the number of Threads up to the Thread change also the priority of Threads is controlled. See also Threads.

4.1.5 Project Options

.

For each project the compiler settings can be changed separately.

The items *Author*, *Version*, *Commentary* can be freely inscribed. They serve as memory support in order to better remember the project details at a later date.

In "Select CPU" the target platform of the project is determined. A mouse click on "*Scan Hardware*" will scan the connected C-Control Pro Module and select the correct CPU.

In "Options" Multi Threading is configured and it is further determined if a Debug Code should be generated.

➡ If a Debug Code is compiled the Byte Code becomes insignificantly longer. For each line in the source text which contains executable commands the Byte Code will be one Byte longer.

➡ In case Multi Threading should be used the selection box in the project options must be selected. Further the parameters for each separate Thread must be set under "Configure Threads".

In the options can also be selected if a Map File should be generated.

4.1.6 Library Management

In the Library Management the source text libraries can be chosen that will be compiled in addition to the project files.
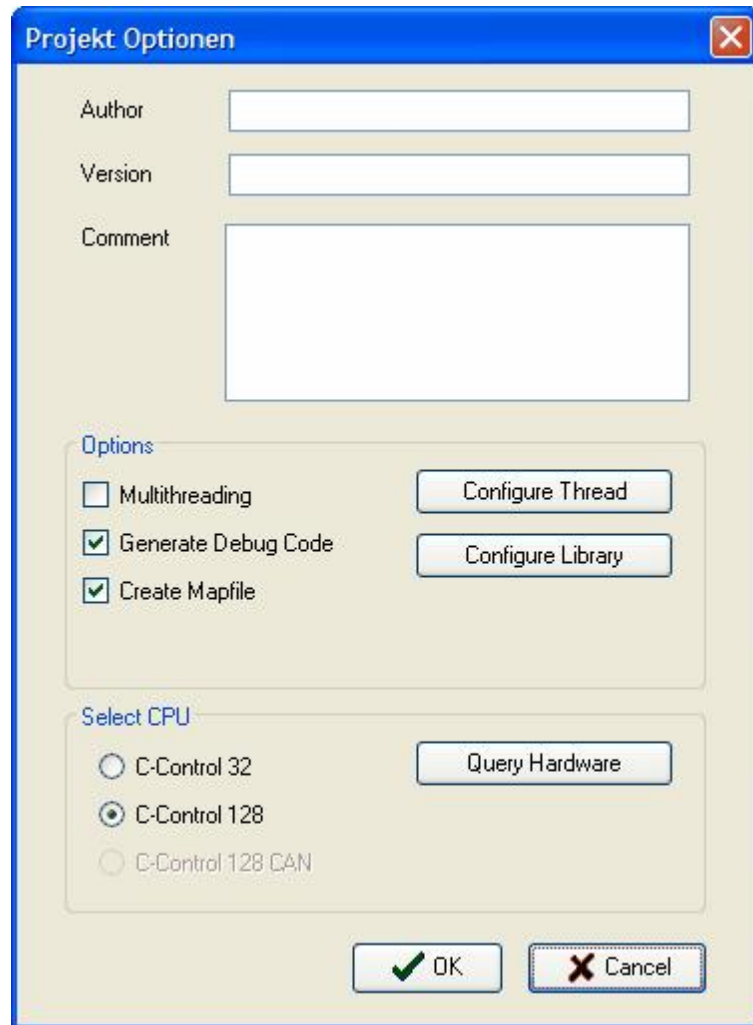
Only those files will be used for compilation whose CheckBox has been selected.

The list can be altered by use of the path text input field "Library Name" and the buttons in the dialog:

- Add – The path will be added to the list.
- Replace – The selected entry in the list is replaced by the path name.
- Delete – The selected list entry is deleted.
- Update Library – Files present in the Compiler Presetting but not in this list will be added.

4.2   Editor

Several windows can be opened in the C-Control Pro Interface. Each window can be altered in size and displayed text detail. A double mouse click on the title line will maximize the window.
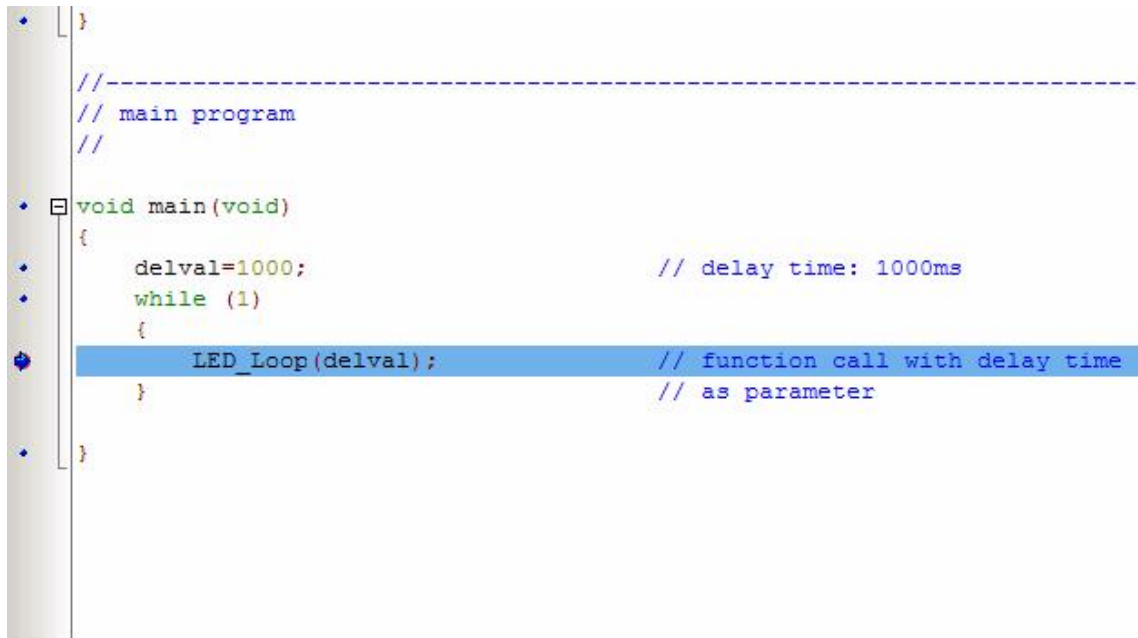
A mouse click in the area to the left of the text will there set a Breakpoint. Prior to this the source text must be compiled error free with "*Debug Info*" and in the corresponding line really executable program text must be placed (i. e. no commentary line o. e.).

Functions Overview

On the left side is an overview of all syntactically correct defined functions. The function names with parameters are expressed in this view. The function where the cursor in this moment resides is drawn with a grey bar in the background. After a double click on the function name the cursor jumps to the beginning of that function in the editor.
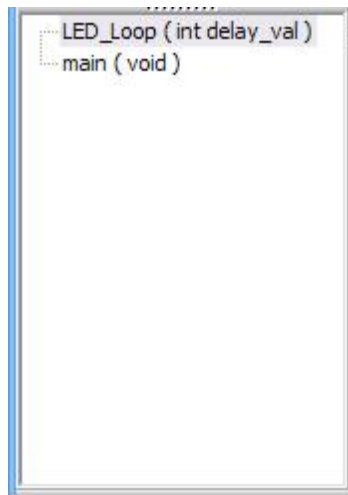
Code Folding

To maintain a good overview over the source code, the code can be folded. After the syntactical analyzer, that is built into the editor, recognizes a defined function, beams are drawn on the left side along the range of the function. A click on the minus sign in the small box folds the text, so that only the first line of the function can be seen. Another click on the small plus sign, and the code unfolds again.
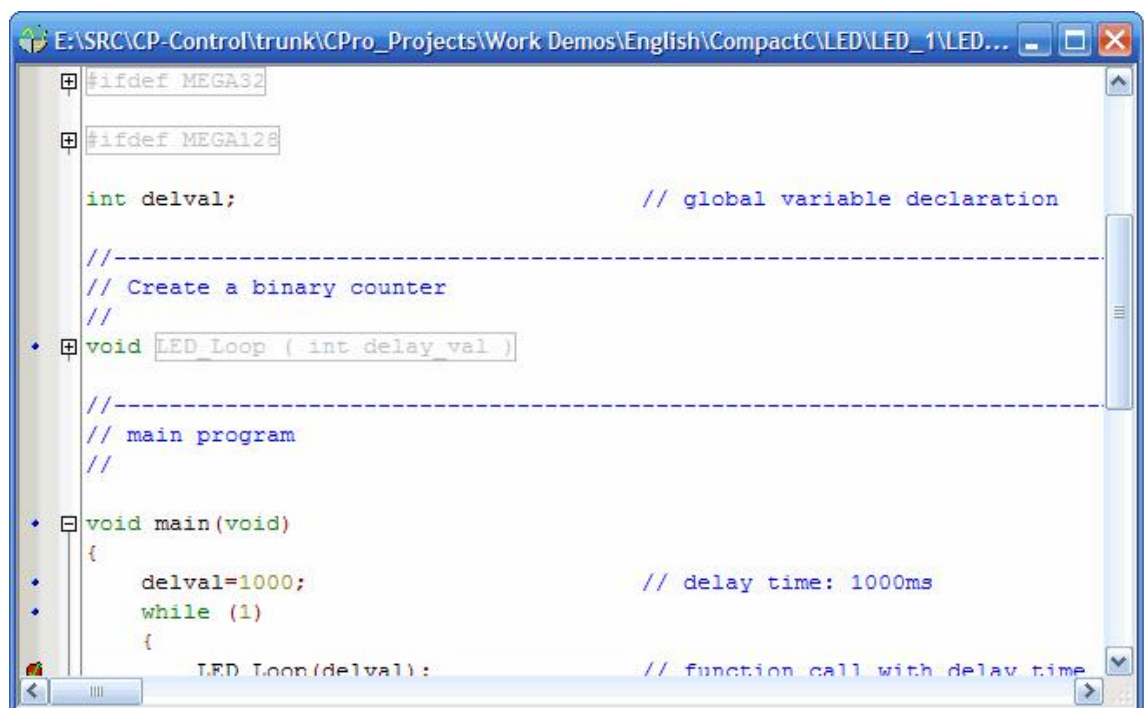
To fold or unfold all functions in an editor file, the options Full Collapse and Full Expand are selectable in the right click editor pull-up menu.

Syntactical Input Help

The editor now has a syntactical input help. When the beginning of a reserved word or a function name from the standard library is typed into the editor, the input help can be activated with Ctrl-Space. In dependency from the already entered characters, a popup select box opens, that shows the words that can be inserted into the source code.
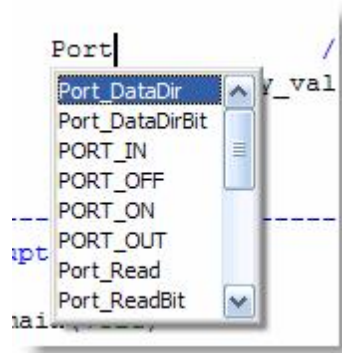
Refresh Editor View

Should the syntactic analyzer fail and cannot recognize the defined function blocks (can seldom happen in find - replace operations), the syntactic analysis can repeated if the command Refresh is selected from the Edit pull-down menu.

4.2.1 Editor Functions

Under menu item Edit the most important editor functions can be found:

- Undo (Ctrl-Z) – will execute an Undo operation. The possible number of Undo steps depends on the settings in Undo Groups.
- Restore (Ctrl-Y) – will restore the editor condition that has been changed by previous use of the Undo command.
- Cut (Ctrl-X) – will cut out selected text and will copy it to the clipboard.
- Copy (Ctrl-C) – will copy selected text to the clipboard.
- Insert (Ctrl-V) – will copy the contents of the clipboard to the cursor position.
- Select All (Ctrl-A) – will select the entire text.
- Search (Ctrl-F) – will open the Search dialog.
- Continue Search (F3) – will continue the search using the set search criteria.
- Replace (Ctrl-R) – will open the Replace dialog.
- Go To (Alt-G) – will allow to jump to a definite line.

Search/Restore Dialog
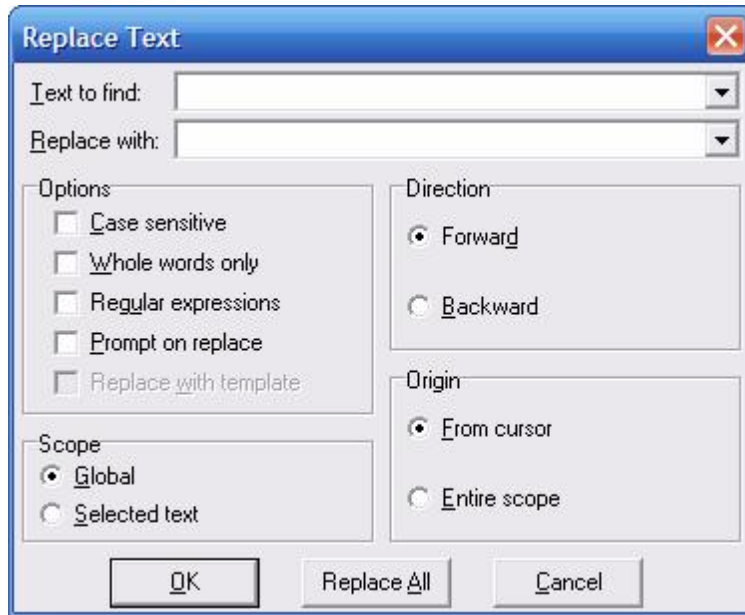
- Text to find – Input field for the text to be searched for.
- Replace with – Text that will replace the text found.
- Case Sensitive – makes the distinction between upper and lower case writing.
- Whole words only – will find only whole words rather than part character chains.
- Regular expressions – activates the input of Regular Expressions in the search mask.
- Prompt on replace – prior to replacing the user will be asked for approval.

Furthermore it can be pre-determined whether the entire text or a selected text area only should be scoured and what search direction should be used.

4.2.2   Print Preview

To deliver the source code as Hard Copy or for archiving purposes, the C-Control Pro IDE has built in printer functions. The following options can be selected from the File Pull-Down Menu:

Print: Prints the indicated pages
Print Preview: Shows a print preview
Printer Setup: Choose the printer, paper size and orientation
Page Setup: Header and Footer lines, line numbers and other parameters can be selected

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

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

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


#ifdef MEGA32
#define PORT 3
#define PORTDIR 0xC0                    // LED1=PortD.6, LED2=PortD.7
#endif

#ifdef MEGA128
#define PORT 6                          // Application Board 2.Version PortG
#define PORTDIR 0x18                    // LED1=PortG.3, LED2=PortG.4
#endif

int delval;                            // global variable declaration

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

    Port_DataDir(PORT,PORTDIR);

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

        #ifdef MEGA32
        x=i<<6;                         // shift i left
        #endif
```

4.2.3 Keyboard Shortcuts

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

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

Esc	Reset selection
Ctrl + digit (0-9)	Go to Bookmark digit (0-9)
Shift + Ctrl + (0-9)	Set Bookmark digit (0-9)
Ctrl + Space	Auto completion popup

4.2.4 Regular Expressions

The search function in the editor supports Regular Expressions. With this function character chains can highly flexible be searched for and replaced.

^	A Circumflex at the beginning of the word finds the word at the beginning of a line
$	A Dollar Sign represents the end of a line
.	A Dot symbolizes an arbitrary character
*	A Star stands for the repeated appearance of a pattern. The number of repetitions may also be Zero.
+	A Plus stands for the multiple or at least solitary appearance of a pattern
[ ]	Characters in square brackets represent the appearance of one of the characters
[^]	A Circumflex in square brackets negates the selection
[-]	A Minus in square brackets symbolizes a character range
{ }	Tailed braces will group separate expressions. Up to ten levels may be nested
\	A Back Slash will take the special meaning from the following character

Examples

Example	will find
^void	the word "void" only at the beginning of a line
;$	the Semicolon only at the end of a line
^void$	Only "void" may stand in this line
vo.*d	e. g. "vod","void","vqqd"
vo.+d	e. g. "void","vqqd" but not "vod"
[qs]	the letters 'q' or 's'
[qs]port	"qport" or "sport"
[^qs]	all letters other than 'q' or 's'
[a-g]	all letters from 'a' through 'g' (including)
{tg}+	e. g. "tg", "tgtg", "tgtgtg" asf.
\$	'$'

4.3 C-Control Hardware

Under menu item C-Control all hardware relevant functions can be executed. These include transfer and start of the program on the hardware as well as password functions.

4.3.1 Start Program

Program Transfer

After a project has been translated free of errors the Bytecode must first be transferred onto Mega32 or Mega 128 before it can be executed. This is done by use of the command Transfer (Shift-F9) in menu C-Control.

➡ Not only the Bytecode is transferred to the Mega Module. At the same time the latest interpreter version is sent to the C-Control Module.

Start

By Start (F10) the execution of the Bytecode is brought about on Mega 32 or Mega128.

Stop

During normal operation a program will be stopped by pressing the RESET1 button. For performance reasons the program execution on the Module is during normal operation not being stopped by use of software. This can however be performed with the IDE function Stop Program when the program runs in Debug Mode.

➡ In rare cases the system can get jammed during USB operation when the RESET1 button is pressed. To overcome this please also press RESET2 in order to issue a Reset pulse to the Mega8, too. The Mega8 is on the Application Board responsible for the USB interface.

Auto Start

If no USB interface is connected and SW1 has not been pressed during power-up in order to reach the Serial Bootloader Mode the Bytecode (if available) is started in the Interpreter. I.e. if the Module is built into any hardware application the mere connection of the operating voltage is sufficient to automatically start the user program.

➡ A signal on INT_0 during switch-on of the C-Control Pro Module can interfere with the auto start behaviour. According to the pin assignments of M32 and M128 INT_0 is connected to the same pin as switch SW1. When SW1 is pressed during power-up of the Module this will activate the Serial Bootloader Mode and the program will not automatically be started.

4.3.2 Outputs

For display of Debug messages there is an "Outputs" window section.
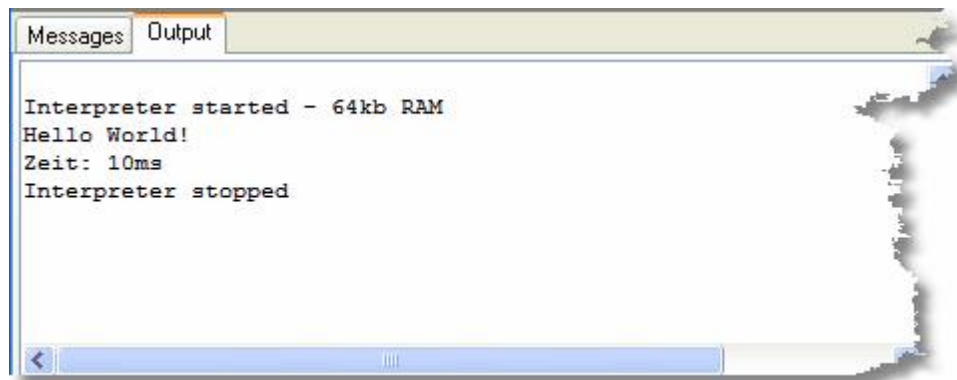
Here is shown when the Bytecode Interpreter has been started and terminated and for how long (in milliseconds) the Interpreter was in operation. The operation time however is not very useful if the Interpreter has been stopped during Debug Mode.

The Outputs window can also be used to display the user's own Debug messages. For this there are several Debug Functions.

With a right mouse click in the Debug Outputs section the following commands can be selected:

- Delete – will delete the list of Debug outputs
- Copy to Clipboard – will copy all text messages onto the clipboard

4.3.3 PIN Functions

Some solitary functions of the Interpreter can be protected by use of an alpha-numeric PIN. If an Interpreter is protected by a PIN normal operations are prohibited. By means of a new transfer the Interpreter can be overwritten, the PIN will however stay preserved. Also a normal start other than the Autostart behaviour is no longer allowed. Furthermore the scans of hardware and firmware version numbers are locked.

If access to a forbidden function is tried a dialod with the following text will be displayed: "C-Control is Password protected. Operation not allowed!".

Through inscription of the PIN with Enter PIN in the C-Control Menu all operations can again be released.

In order to enter a new PIN or to delete a set PIN there are the commands Set PIN and Delete PIN in the C-Control Menu. If there is an old PIN in exitence then the Module must of course first be unlocked by entering the old PIN. The PIN can have a length of up to 6 alpha-numeric characters.

➡ In case the password has been lost there is an emergency function which can be used to reset the Module to its initial state. In C-Control there is the option Reset Module which can be used to delete PIN, Interpreter and Program.

4.3.4 Version Check

Since the C-Control Pro Mega Series supports various hardware platforms it is important to closely monitor the current version numbers of Bootloader, Interpreter and Hardware. This is possible by use of item Hardware Version in the C-Control menu.
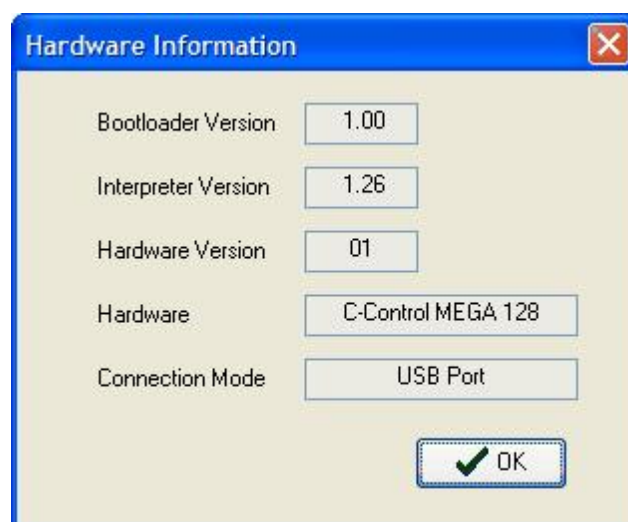
4.4 Debugger

In order to activate the Debugger the project must first be compiled in Debug Code free of errors and then transferred to the Module. The file holding the Debug Code (*.dbg) must be present in the project list.

In the Debugger menu all Debugger commands can be found. The Debugger ist started with Debug Mode (Shift-F10).   If at this point of time no Breakpoint is set then the Debugger will stop at the first executable instruction.

If in Debug Mode, the next Breakpoint will be reached by use of Start (F10). If no Breakpoint is set then the programm will be executed in its normal way. There is the exception however that the program flow can be stopped by use of Stop Program. This only works providing that the program has been started from the Debug Mode.

If the Debugger has stopped in the program (a green bar is displayed) then the program can be executed in single steps. The instructions Single Step (Shift-F8) and Procedure Step (F8) respectivelly will execute the program code up to the next code line and will then stop again. Opposing to Single Step the function Procedure Step will not jump into the function calls but will overpass them.

➡ If a loop contains only one code line then one single step will execute the entire loop since only after this branching out to a new code line will take place.

With the instruction Leave Debug Mode the Debug Mode will be terminated.

➡ During active Debug Mode the program text can not be altered. This is because line numbers holding set Breakpoints must not be moved out of place. Otherwise the Debugger would not be able to synchronize with the Bytecode onto the C-Control Module.

4.4.1 Breakpoints

The editor allows to set up to 16 Breakpoints. A Breakpoint is entered by a mouse click to the left of the beginning of a line (see IDE or Editor Window).
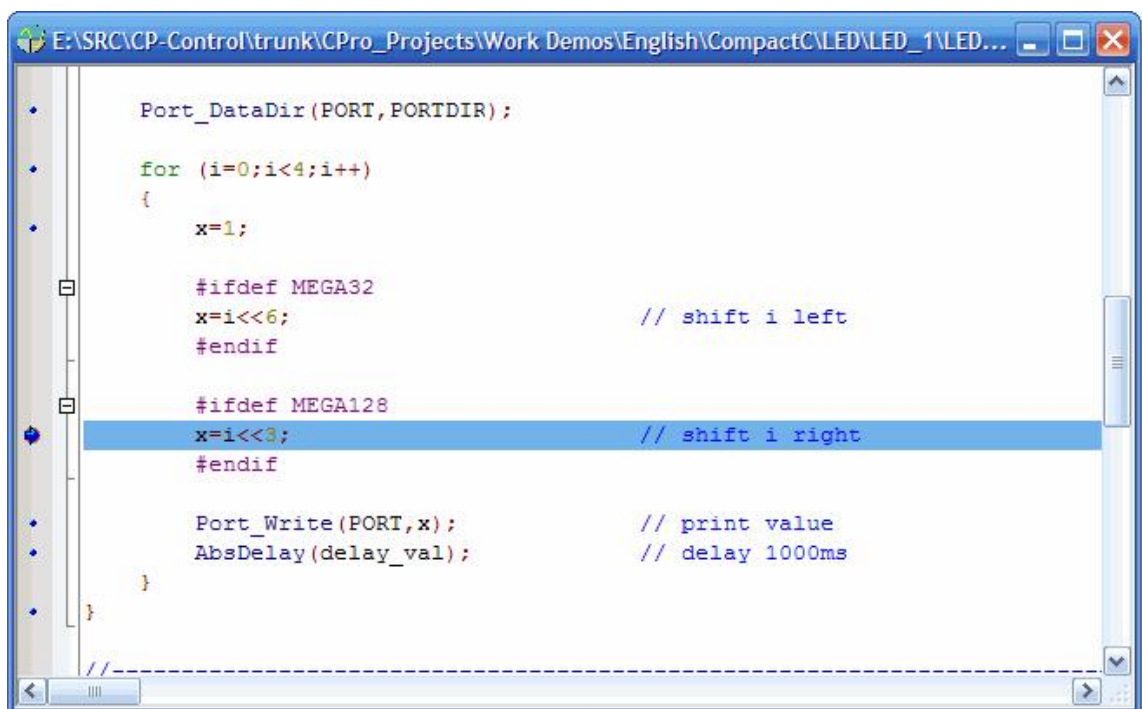
➡ The number of Breakpoints is limited to 13 because this information is carried along in RAM

during operation of the Bytecode Interpreter. Other Debuggers on the Market will set Breakpoints directly into the program code. In our case this is not desirable since it would drastically reduce the life time of the flash memory (appr. 10,000 writing accesses).

4.4.2   Array Window

In order to monitor the contents of Array Variables it is possible to call up a window with the array contents. To do this the pointer is placed over the the variable and Show Array is selected by a right mouse click.
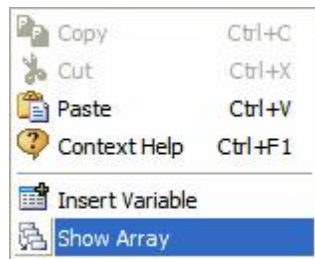
On the left side the Array indices are shown while the contents are displayed on the right side. It should be noted that with multi-dimensional arrays the indices on the right will gain at the faster pace.
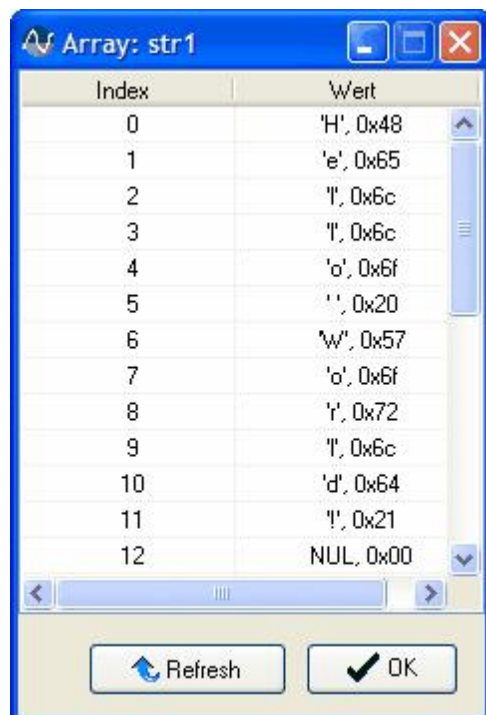
The contents of an array window may at every stop of the Debugger or at every single step no longer be actual. If with each single step in the Debugger several array windows are newly brought up-to-date then delays may occur since the data must always be loaded from the Module. For this reason there are three operating modes:
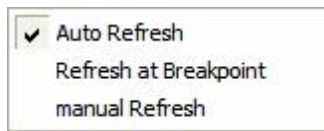
Auto Actualize	Actualize at Single Step and Breakpoint
Actualize at Breakpoint	Actualize only at Breakpoint
Manually Actualize	Only by clicking switch "Actualize"

4.4.3   Variable Watch Window

The contents of variables can be displayed within the Debugger. To do this the mouse pointer is placed over the variable. Within approximately 2 seconds the content of the variable is displayed in form of a Tooltip. The variable is first displayed in accordance to its data type and then, separated by a comma, as Hex number with a preceeding "0x".

If several variables need to be monitored then the variables can be comprised in a list.

In order to enter a variable into the list of monitored variables there are two possibilities. For one the cursor can be placed in the text editor at the beginning of a variable and then Insert Variable can be selected by a right mouse click.
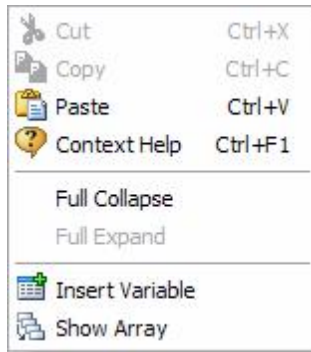
The other possibility is by use of the context menu in the variables list which can also be activated by a right mouse click.

When Insert Variable is selected then the variable to be monitored can as text be entered into the list. In case of a local variable the function name with a preceeding colon (**Function Name** : **Variable Name**) is entered. With Change Variable the text entry in the list can be altered and with Delete Variable the variable can be entirely erased from the list. Prior to this the line holding the variable to be deleted must be selected. The command Delete All Variables will delete all entries from the list.
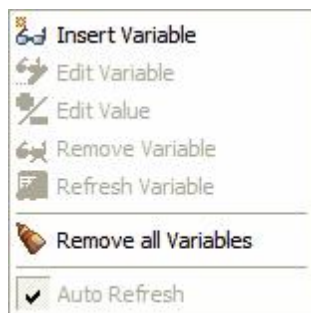
Under certain circumstances an error message is shown instead of a value in the list:

no Debug Code	No Debug Code has been generated
wrong Syntax	During text entry invalid characters have been entered for a variable
Function unknown	The Function Name is not known
Variable unknown	The Variable Name is not known
not in Debug Mode	The Debug Mode has not been activated
no Context	Local variables can only be displayed while within this function
not actual	The content of the variable has not been updated

If a high number of variables is entered in the monitor list it may during single step operation take quite some time until all variable contents from the module have been scanned. For this reason the Option Auto Actualize can be switched off for individual variables. The contents of these variables will then only be displayed after the command Actualize Variable is executed. This way the Debugger can quickly be operated in single steps and the contents are only actualized on demand.

➡ Variables of the Character type are displayed as single ASCII characters.

4.5 Tools

Terminal Window

In the Tools pulldown menu a simple terminal program can be started.
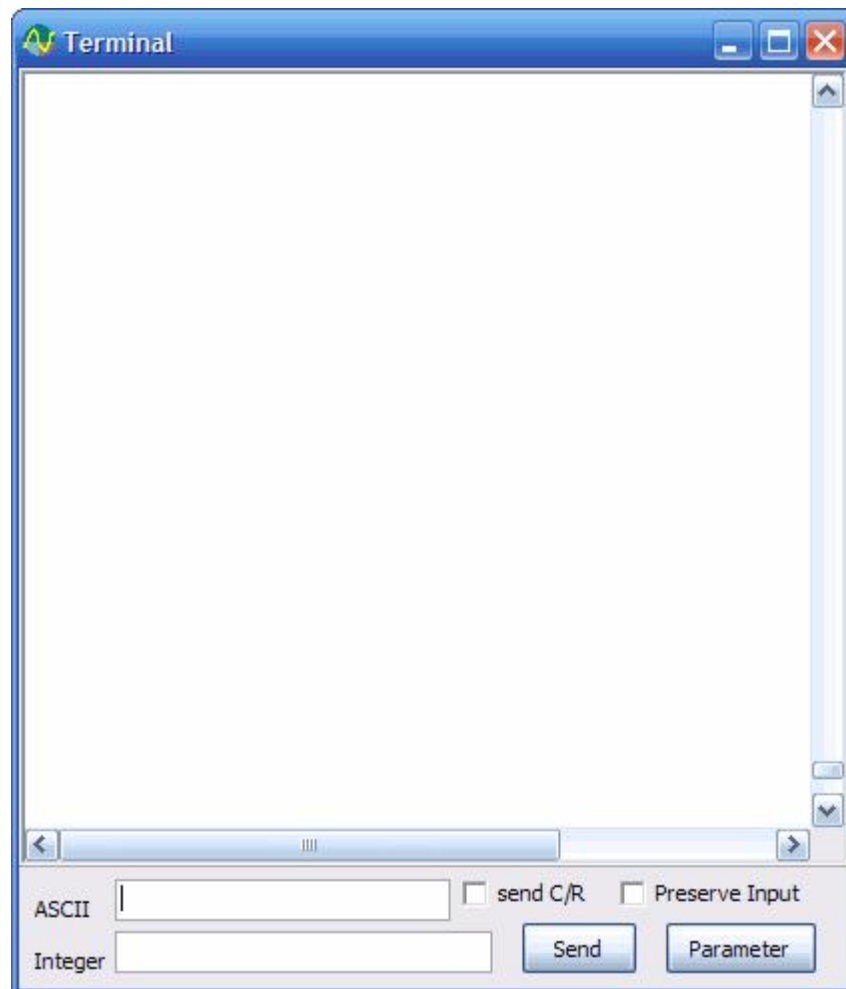
Received characters are directly shown in the terminal window. Characters can be send in two different ways. On the one hand the user can click into the terminal window and directly type the characters from the keyboard, on the other hand the text can be entered in to the ASCII input line and send with the Send button. Instead of ASCII the characters can be defined as integer values in the Integer input line. Is send C/R selected, a Carriage Return (13) is sent at the end of the line. Enable Preserve Input to prevent that the input lines are cleared after pressing the Send button. The Parameter button opens the Terminal settings dialog from the IDE settings.

4.6 Options

In Menu Options all IDE settings and Compiler pre-settings can be found.

4.6.1 Editor Settings
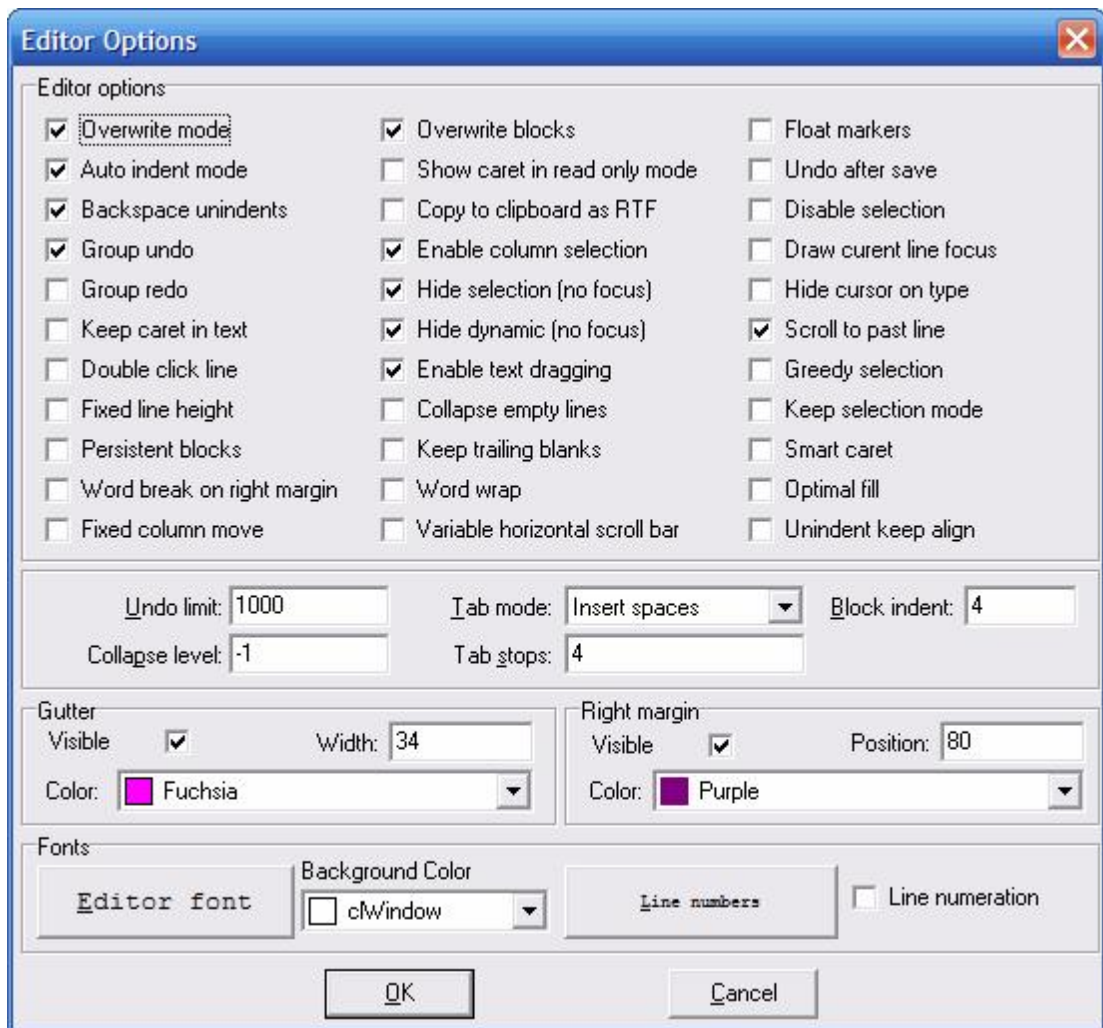
- Overwrite mode – Inserts text at the cursor overwriting existing text.
- Auto indent mode - Positions the cursor under the first non blank character of the preceding non blank line when you press Enter.
- Backspace unindents - Aligns the insertion point to the previous indentation level (outdents it) when you press Backspace, if the cursor is on the first non blank character of a line.
- Group undo - Undo operation will not be performed in small steps but in blocks.

Group redo - If it is set Redo will involve group of changes.

Keep caret in text - Allows move caret only into text like in Memo.

Double click line - Highlights the line when you double-click any character in the line. If disabled, only the selected word is highlighted.
Fixed line height - Prevents line height calculation. Line height will be calculated by means of Default Style.
Persistent blocks - Keeps marked blocks selected even when the cursor is moved using the arrow keys, until a new block is selected.
Overwrite blocks - Replaces a marked block of text with whatever is typed next. If Persistent Blocks is also selected, text you enter is appended following the currently selected block.
Show caret in read only mode - Shows caret in read only mode.
Copy to clipboard as RTF - Copies selected text also in RTF format.
Enable column selection - Enabled column selection mode.
Hide selection - Hides selection when editor loses focus.
Hide dynamic - Hides dynamic highlighting when editor loses focus.
Enable text dragging - Enables drag & drop operation for text movement.
Collapse empty lines - Collapse empty lines after text range when this rang have been collapsed.
Keep trailing blanks - Keeps any blanks you might have at the end of a line.
Float markers - If it is set markers are linked to text, so they will move with text during editing. Otherwise they are linked to caret position, and stay unchanged during editing. Also markers save scroll position.
Undo after save - Stay undo buffer unchanged after save with SaveToFile method.
Disable selection - Disables any selection.
Draw current line focus - Draws focus rectangle around current line when editor have focus.
Hide cursor on type - Hides mouse cursor when user type text and mouse cursor within client area.
Scroll to last line - When it is true you may scroll to last line of text, otherwise you can scroll to last page. When this option is off and total text height less then client height vertical scroll bar will be hidden.
Greedy selection - If this option is set selection will contain extra column/line during column/line selection modes.
Keep selection mode - Selection enabled for caret movement commands (like in BRIEF).
Smart caret - Acts on the caret movement (up, down, line start, line end). Caret is moved to the nearest position on the screen.
Word wrap - Determines whether the editor wraps text at the right side of text area.
Word break on right margin - Determines whether text wraps (word-wrap mode) on the right margin instead of right side of client area.
Optimal fill - Begins every auto indented line with the minimum number of characters possible, using tabs and spaces as necessary.
Fixed column move - Keeps X position of caret before editing text, this position is used when moving up/down caret.
Variable horizontal scroll bar - Sets range of horizontal scroll bar to the maximal width of only visible lines. Hides horizontal scroll bar if visible lines fit client width.
Unindent keep align - Restricts unindent operation when at least one of lines can not be unindented.

At **Block indent** the number of blanks is inscribed by which a selected block can be indented or backed by use of the Tabulator key.
The input field **Tab stops** determines the width of the tabulator by numbers of characters.

4.6.2   Syntax Highlighting

In this Dialog the user can change the specific Syntax Highlighting for CompactC and BASIC. The chosen language for the setting is CompactC or BASIC in dependency on what language is used in the actual selected editor window.
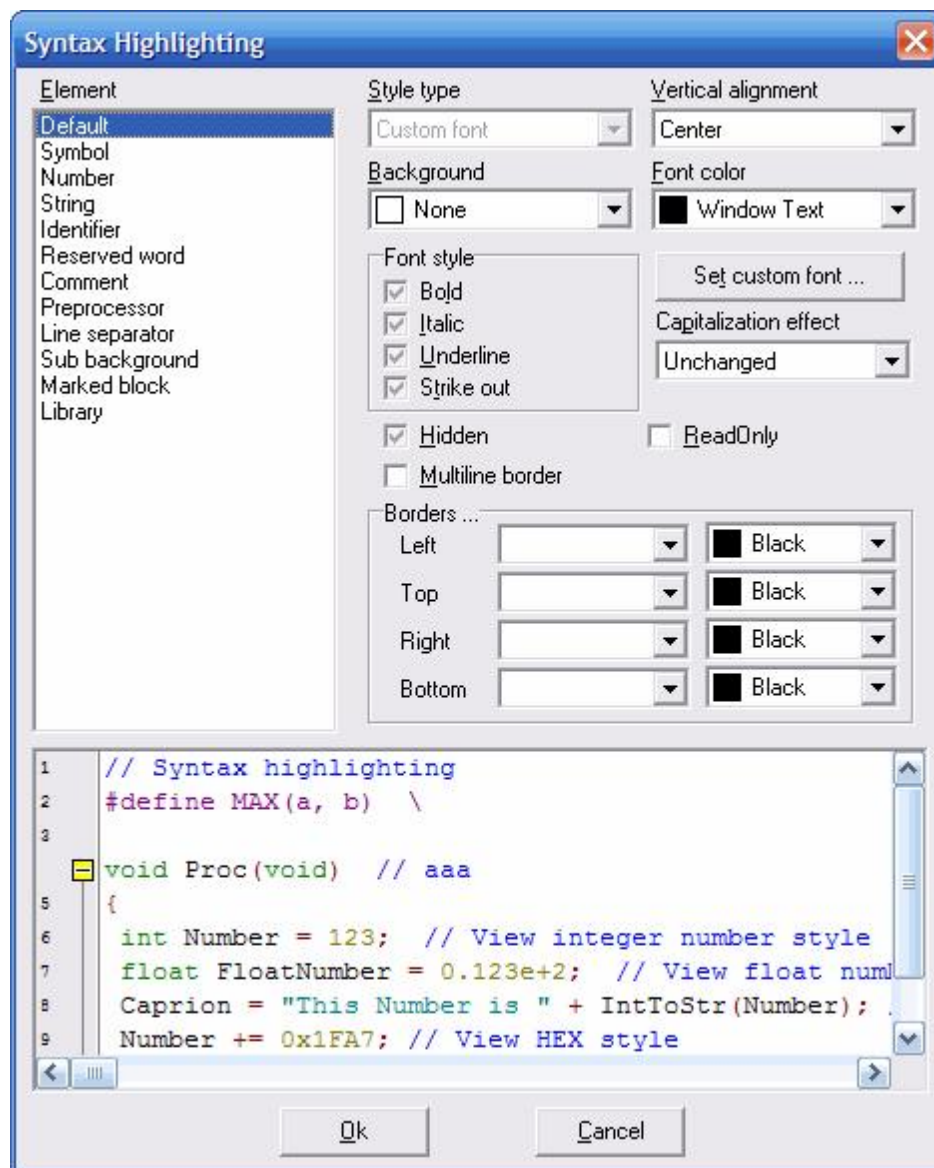
You can change the attributes of the font, and the foreground- and background color. With Multiline border a colored border can be drawn around the highlighted strings. Also case changes can be made with the option Capitalization Effect. The selectable Elements have the following meaning:

Symbol: all non alpha-numeric characters
Number: all numeric characters
String: all characters that are recognized as strings
Identifier: all names that are not reserved words or part of the library
Reserved Word: alle reserved words of the destination language
Comment: comments
Preprocessor: preprocessor statements
Marked Block: marked editor blocks

Library: function names of the standard library

Default, Line separator and Sub background are not used.

4.6.3   Compiler Presetting

In the Compiler Presetting the standard values can be configured which will be stored during creation of a new project. Presetting can be reached under Compiler in the Options menu.
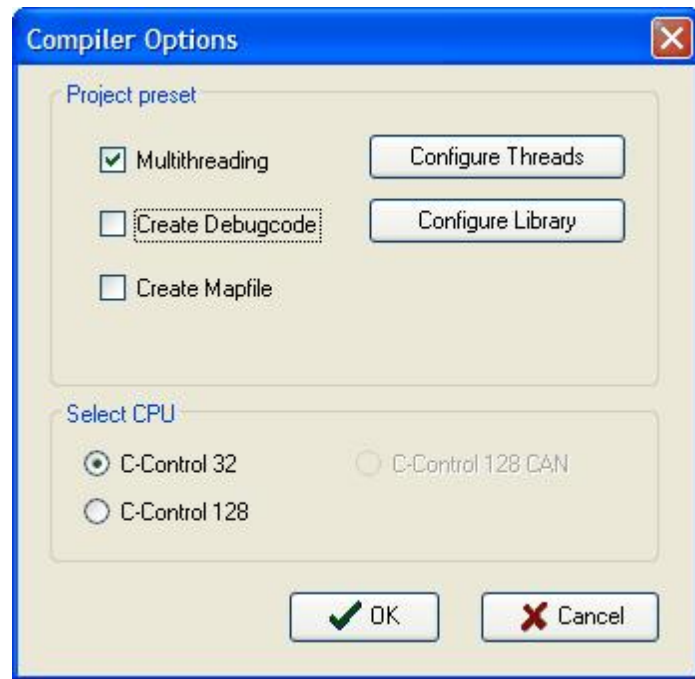
A description of the options can be found under Project Options. The selection boxes "Configure Threads" and "Configure Library" are identical to the descriptions in chapter Projects.

4.6.4 IDE Settings

Separate aspects of the IDE can be configured.
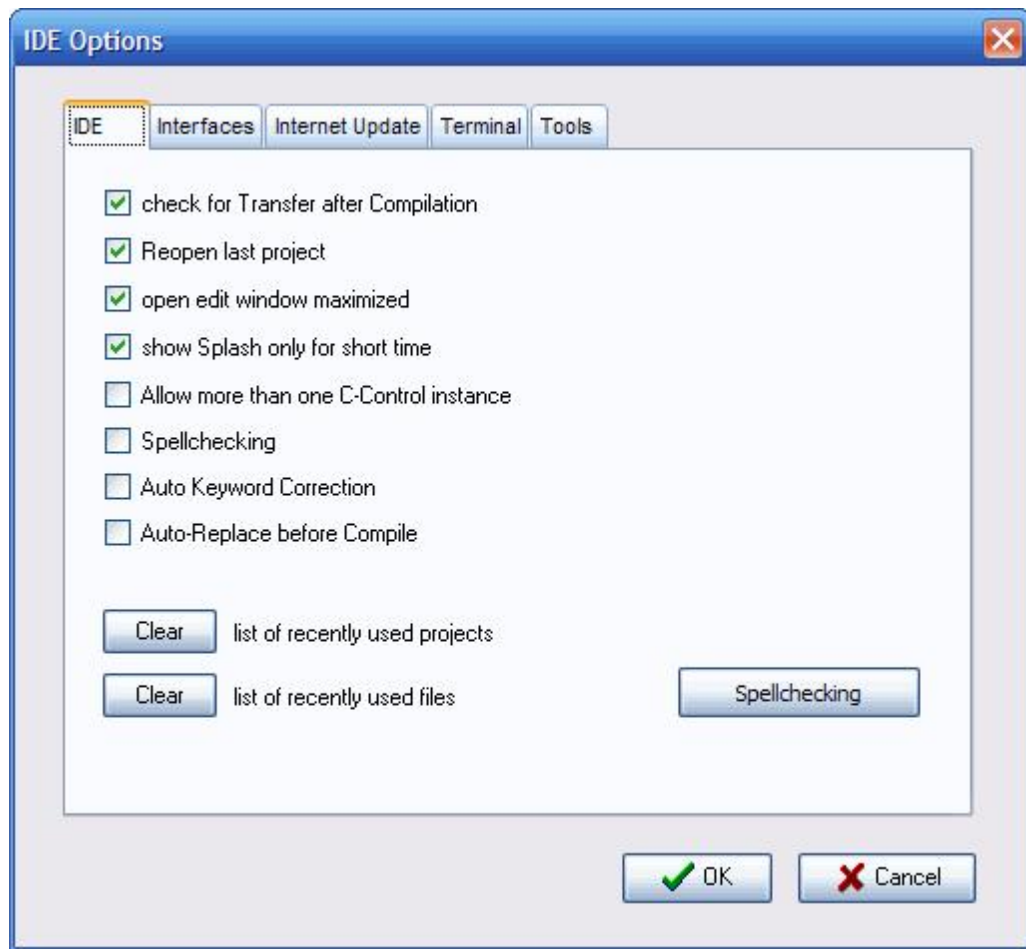
- Transfer After Compiling Callup – After a program has been compiled but not transferred to the C-Control Module then the user will be questioned whether or not the program should be started.
- Open Last Project – The last open project will be re-opened when the C-Control Pro IDE is started.
- Open Maximized Editor Window – When a file is opened the editor window will automatically be switched to maximum size.
- Splashscreen Short Display - The Splashscreen is only displayed until the main window is opened.
- Allow Multiple Instances Of C-Control Pro – When the C-Control Pro interface is started several times it may create conflicts with regard to the USB interface.

Also here the lists of the "last opened projects" as well as the "last opened files" can be deleted.

4.6.4.1 Interfaces

Through a selection box the connection to the application board can be set. USB connections will start with the prefix "USB" and will then be successively numbered: USB0, USB1, ... Serial interfaces will be handled equally. They will start with the prefix "COM": COM0, COM1, ..., aso.
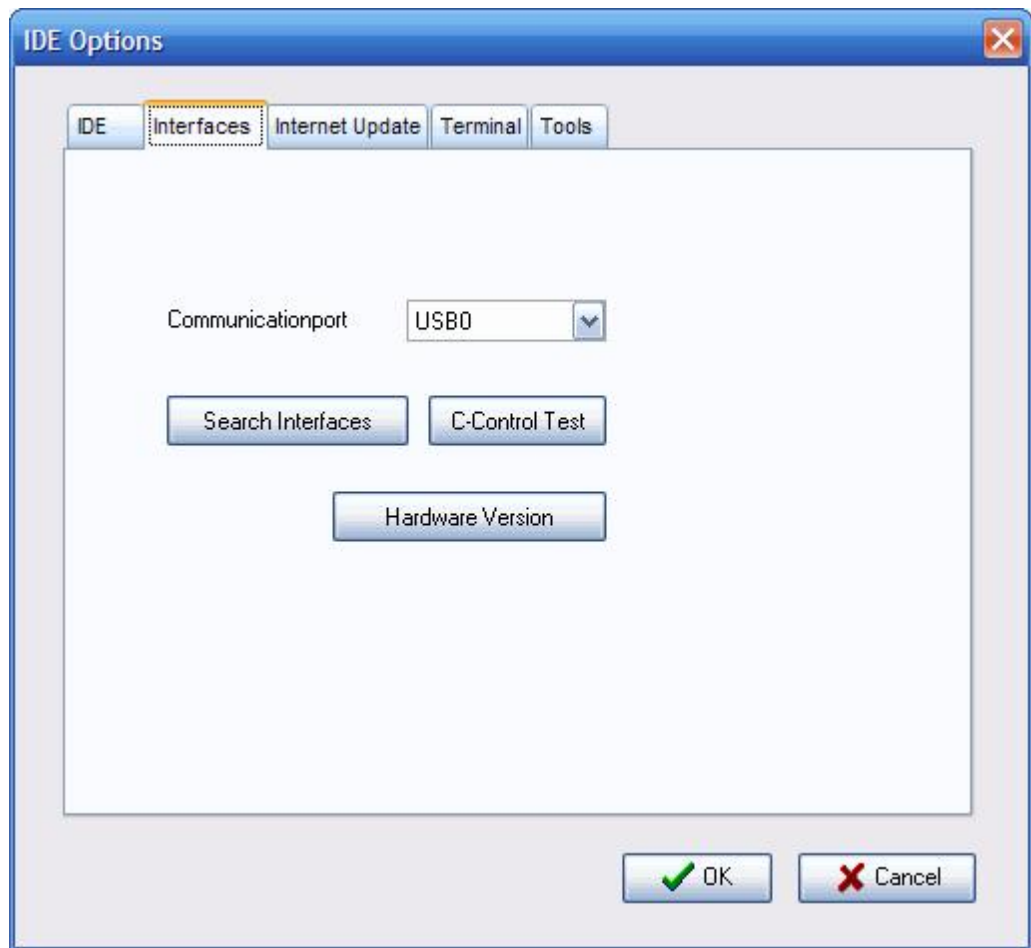
By use of the button "Search Interface" all interfaces will be evaluated until the command line interface of C-Control Pro will react. In order to recognize an application board power must be supplied and the firmware must not have stalled. It is recommended to switch the power off and on again prior to the searching action.

The buttons "C-Control Test" and "Hardware Version" allow to immediately see whether or not the selected interface can sensibly communicate with the C-Control Pro Module.

4.6.4.2   Internet Update

In order to check if any improvements or error corrections have been issued by Conrad Electronic the Internet Update can be activated. When the selection box "Update Check Every n Days" is selected then an update will be searched for in the Internet at an interval of n days at every start of the IDE. The parameter n can be set in the input field on the right.

The button "Update Check Now" will immediately activate an update search.

➡ In order to have the Internet update function correctly the MS Internet Explorer must not be in "Offline" Mode.
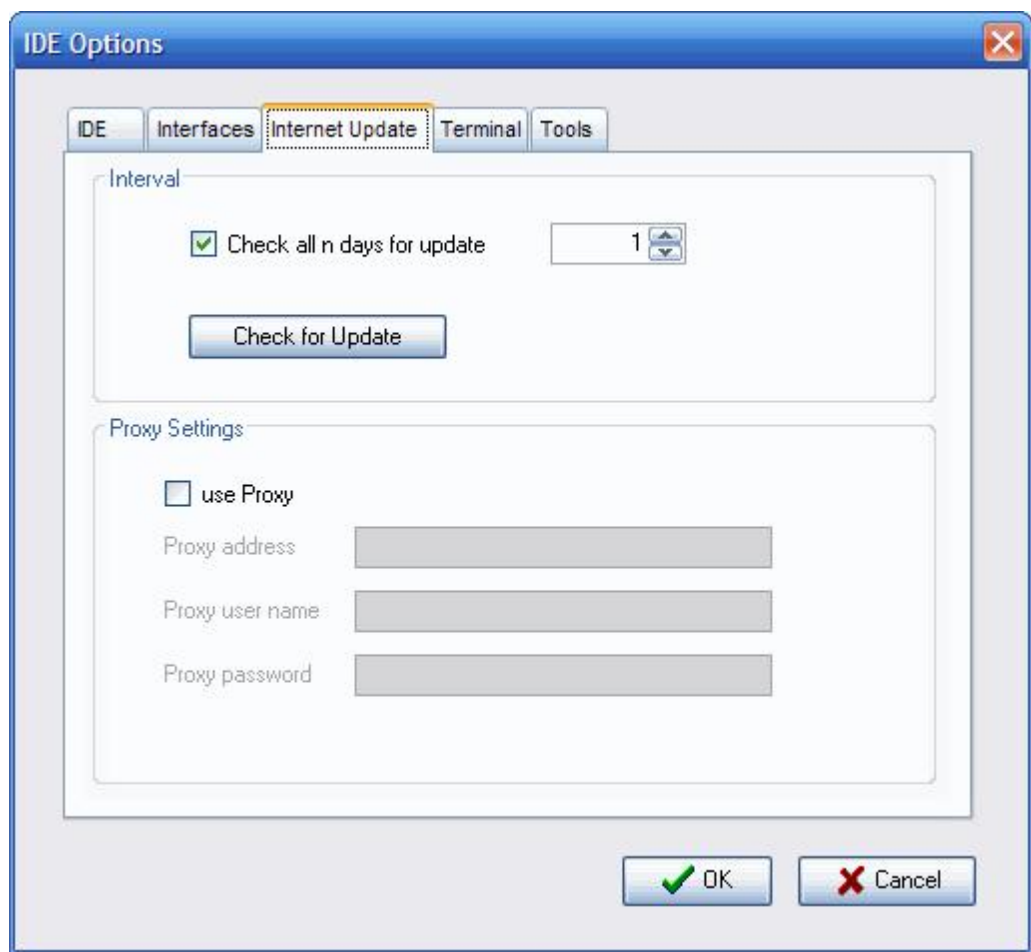
If e. g. the Internet access is restricted by a Proxy due to a firewall then the Proxy settings such as address, user name and password can be entered in this dialog.

➡ If there are Proxy data set in the MS Internet Explorer then they will be of higher priority and will thus overwrite the settings in this dialog.

4.6.4.3 Terminal

Here you can set the serial parameter for the built in terminal program. For the Port entry an available serial COM Port can be chosen from. Further the standard baudrates, the number of Data Bits and Stop Bits, and the Flow Control is selectable.
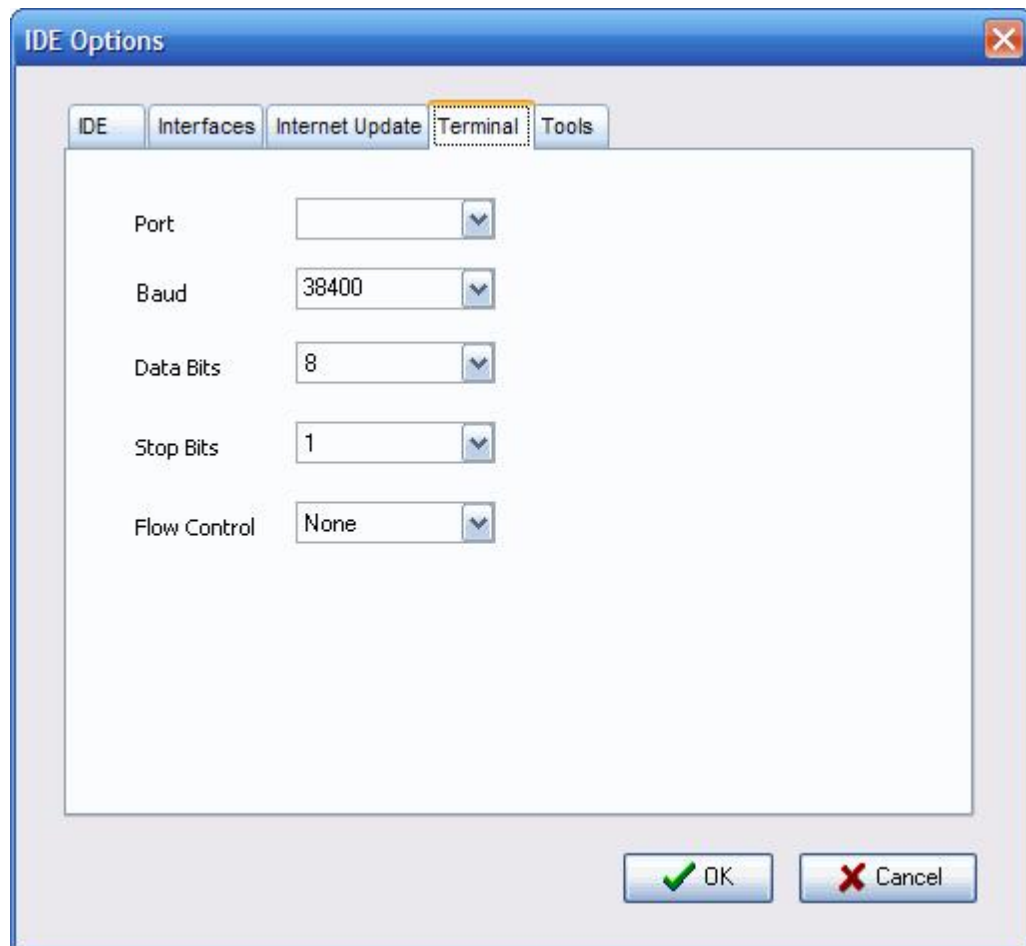
4.6.4.4 Tools

In the Tool settings the user can insert, delete and edit entries that defines external programs that can be executed fast and simple from the IDE. The names of the programs can be found in the Tools pulldown menu and can be started with a single click.
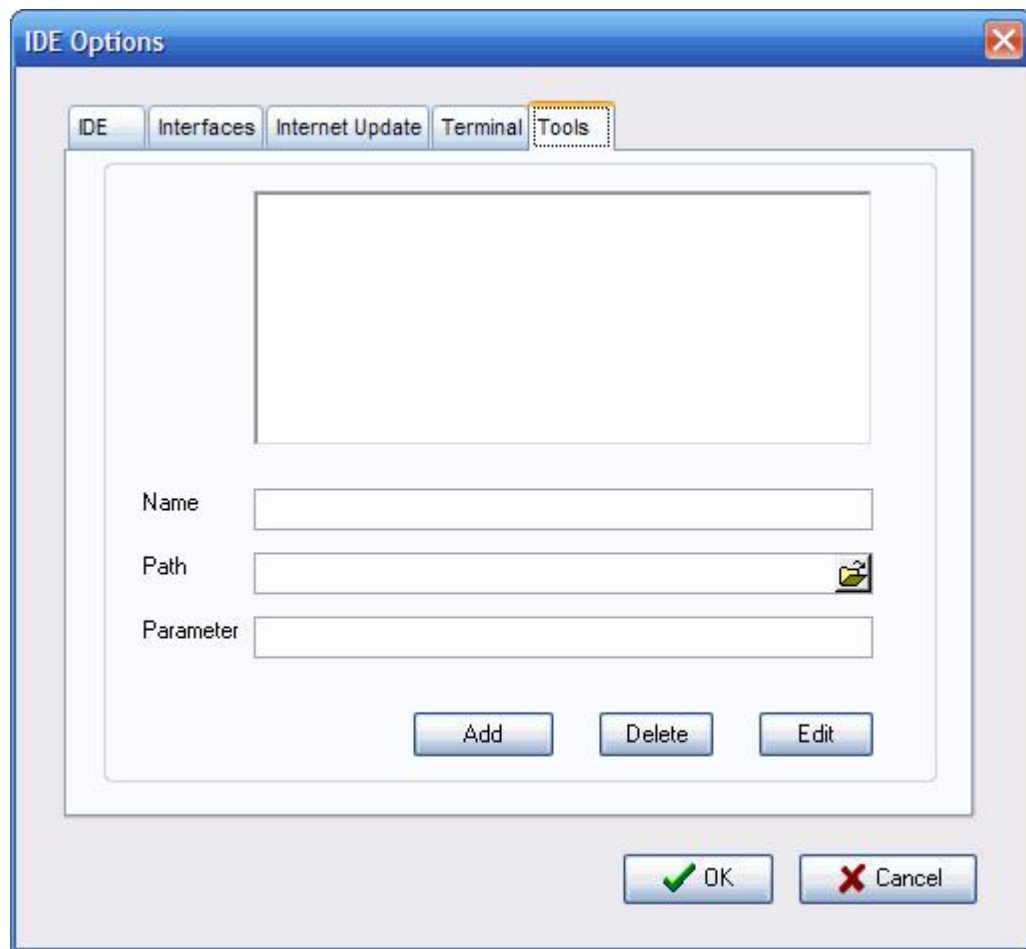
For each program that is inserted, the user can choose the name, the execution path and the parameters that are submitted.

4.7 Windows

When there are several windows opened within the editor area they can automatically be arranged by use of commands in the Window Menu.

- Overlap – The windows will be arranged on top of each other with each successive window placed fractionally lower and more to the right than the preceeding one.
- Beneath – The windows are placed vertically beneath each other.
- Side By Side – Will arrange the windows next to each other from left to right.
- Minimize All – Will minimize all windows to symbol size.
- Close – Will close all active windows.

4.8 Help

Under menu item Help the Help file can be opened by use of Contents (Key F1).

Menu item Program Version will open the window "Version Information" and will at the same time copy the contents onto the clipboard.
These informations are important if a Support E-Mail needs to be sent to Conrad Electronic. Since these informations are automatically placed onto the clipboard when Program Version is called up the data can easily be added to the end of an E-Mail.

```
Version Information

C-Control IDE Version:1.60.0.0
Compact-C Compiler Version:1.26.0.0
Bootloader Version: 1.00   Interpreter Version: 1.26
Hardware:C-Control MEGA 128   Hardware Rev:01
Connection Type:USB Port
Total Mem: 1610076160  Free mem: 867225600
Microsoft Windows XP Professional Service Pack 2
Build: 2600 - WinDir: C:\WINDOWS
CPU: AMD Athlon(tm) XP 2600+
Model: 8 Family: 6 Stepping: 1
IDString: AuthenticAMD
Screen Resolution: 1680x1050  Colors: 16777216

                                    ✓ OK
```

If the user needs to find a certain search term in the Help file the Context Help may be of advantage. If e. g. in the Editor the cursor stands over the word "AbsDelay" and the correct parameters are searched for then Context Help should be selected. This function will automatically use the word under the cursor for a search term and will consequently show the results in the Help File.

```
Copy            Ctrl+C
Cut             Ctrl+X
Paste           Ctrl+V
Context Help    Ctrl+F1

Insert Variable
Show Array
```
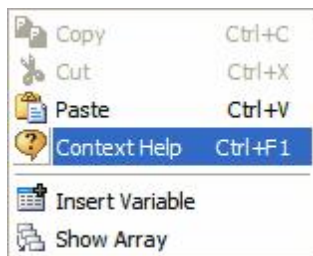
The command Context Help is also available in the editor window after a right mouse click.

Part

5

5   Compiler

5.1   General Features

This domain provides information on the Compiler's properties and features which are independent of the programming language used.

5.1.1   External RAM

The Application Board of **Mega128** carries external RAM. This RAM is automatically recognized by the Interpreter and used for the program to be carried out. For this reason a program memory of appr. 63848 Bytes rather than appr. 2665 Bytes is available. For this it is not necessary to newly compile the program.

➡ If the SRAM is not needed it can be deactivated by JP7 and the ports will be free for other uses. To deactivate the SRAM the jumper JP7 has to be moved to the left side (orientation: serial interface shows to the left), such that the left pins of JP7 are connected.

5.1.2   Preprocessor

➡ The Gnu Generic Preprocessor used here provides some additional functions which are documented under http://nothingisreal.com/gpp/gpp.html. Only the functions described here however have also together with the C-Control Pro Compiler been thoroughly tested. Using the here undocumented functions will thus be at your own risk!

The C-Control development system contains a complete C-Preprozessor. The Preprocessor processes the source text prior to Compiler start. The following commands are supported:

Definitions

By the command "#define" text constants are defined.

```
#define symbol text constant
```

Since the Preprocessor runs ahead of the Compiler at each appearance of symbol in the source text the symbol will be replaced by text constant.

Example

```
#define PI 3.141
```

➡ If a project consists of several sources then #define is a constant for all source files existing following the file, in which the constant has been defined. It is thus possible to change the order of source files in a project.

Conditional Compiling

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

It is possible to monitor which parts of the source texts are really being compiled. After a #ifdef symbol instruction the following text is only compiled when symbol has also been defined by #define symbol.

If there is an optional #else instruction then the source text will be processed after #else if the symbol has not been defined.

Insertion of Text

```
#include path\file name
```

By this instruction a text file can be inserted into the source code.

➡ Because of some restrictions in the Preprocessor a path within a #include instruction must not contain any blank characters!

5.1.2.1   Predefined Symbols

In order to ease the work with different versions of the C-Control Pro series there are a number of definitions which are set depending on target system and Compiler project options. These constants can be called up by #ifdef, #ifndef or #if.

Symbol	Meaning
MEGA32	Configuration for Mega 32
MEGA128	Configuration for Mega 128
MEGA128CAN	Configuration for Mega 128 CAN Bus
__DEBUG__	Debug Data will be created
__MAPFILE__	A Memory Layout will be computed

The following constants contain a string. It is sensible to use them in conjunction with text outputs.

Symbol	Meaning
__DATE__	Current Date
__TIME__	Time of Compiling
__LINE__	Current Line in Sourcecode
__FILE__	Name of Current Source File
__FUNCTION__	Current Function Name

Example

Line number, file name and function name will be issued. Since file names may become quite long it is recommended to dimension character arrays somewhat generous.

```
char txt[60];

txt=__LINE__;
Msg_WriteText(txt);    // Issue Line Number
Msg_WriteChar(13);     // LF
txt=__FILE__;
Msg_WriteText(txt);    // Issue File Number
Msg_WriteChar(13);     // LF
txt=__FUNCTION__;
Msg_WriteText(txt);    // Issue Function Name
Msg_WriteChar(13);     // LF
```

5.1.3   Pragma Instructions

By use of the #pragma instruction output and flow of the Compiler can be controlled. The following commands are authorized:

#pragma Error "xyz..."	An error is created and text "xyz..." is issued
#pragma Warning "xyz..."	A warning is created and text "xyz..." is issued
#pragma Message "xyz..."	The text "xyz..." is issued by the Compiler

Example

These #pragma instructions are often used in conjunction with Preprocessor commands and Predefined Constants. A classical example is the creation of an error message in case specific hardware criteria are not met.

```
#ifdef MEGA128
#pragma Error "Counter Functions not with Timer0 and Mega128"
#endif
```

5.1.4   Map File

If during compilation a Map File has been generated then the memory size of the used variable can there be ascertained.

Example

The project CNT0.cprj generates the following Map File during compilation:

```
Global Variable              Length     Position (RAM Start)
-------------------------------------------------------------
Total Length: 0 bytes

Local Variable               Length     Position (Stack relative)
-------------------------------------------------------------
Function Pulse()
count                          2          4
i                              2          0
Total Length: 4 bytes

Function main()
count                          2          2
n                              2          0
Total Length: 4 bytes
```

From this list can be seen that no global variables are being used. There are further the two functions "Pulse()" and "main()". Each one of these functions consumes a memory space of 4 Bytes on local variables.

5.2   CompactC

One possibility to program the C-Control Pro Mega 32 or Mega 128 is offered by the programming language CompactC. The Compiler translates the language CompactC into a Bytecode which is then processed by the Interpreter of the C-Control Pro. The language volume of CompactC does essentially correspond with ANSI-C. It is however reduced to some extent since the firmware had to be implemented in a resource saving way. The following language constructs are missing:

- **structs** / **unions**
- **typedef**
- **enum**
- constants (**const** instruction)
- pointer Arithmetic

Detailed program examples can be found in directory "Demo Programs" which was installed along with the design interface. There example solutions can be found for almost every field of purpose.

The following chapter contains a systematic introduction into syntax and semantics of CompactC.

5.2.1   Program

A program consists of a number of instructions (such as "a=5;") which are distributed among various Functions. The starting function, which must be present in every program, is the function "main()". The following is a minimalistic program able to print a number into the output window:

```
void main(void)
{
```

```
        Msg_WriteInt(42);   // the answer to anything
}
```

Projects

A program can be separated into several files which are combined in a project (see Project Management). In addition to these project files Libraries can be added to the project which are able to offer functions used by the program.

5.2.2   Instructions

Instruction

An instruction consists of several reserved command words, identifiers and operators and is at the end terminated by a semicolon (';'). In order to separate various elements of an instruction there are spaces in between the instruction elements which are called "*Whitespaces*". By "spaces" space characters, tabulators and line feeds ("C/R and LF") are meant. It is of no consequence whether a space is built by one or several "*Whitespaces*".

Simple Instruction:

```
a=  5;
```

➡ An instruction does not necessarily have to completely stand in one line. Since line feeds do also belong to the space category it is legitimate to separate an instruction across several lines.

```
if(a==5)    // instruction across 2 lines
a=a+10;
```

Instruction Block

Several instructions can be grouped into a block. Here the block is opened by a left tailed bracket ("{"), followed by the instructions and closed at the end by a right tailed bracket ("}"). A block does not necessariliy have to be terminated by a semicolon. I. e., if a block builds the end of an instruction then the last character in the instruction will be the right tailed bracket.

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

Comments

There are two types of commentaries, which are the single line and the multi line commentaries. The text within commentaries is ignored by the Compiler.

- Single line commentaries start with "//" and end up at the line's end.

- Multi line commentaries start with "/*" and end up with "*/".

```
/* a
multi line
commentary */

// a single line commentary
```

Identifier

Identifier are the names of Functions or Variables.

- Valid characters are letters (A-Z,a-z), numbers (0-9) and the low dash ('_')
- An identifier always starts with a letter
- Upper and lower case writings are differentiated
- Reserved Words are not allowed as identifier
- The length of identifiers is unlimited

Arithmetic Expressions

An arithmetic expression is a quantity of values connected by Operators. In this case quantities can either be Figures, Variables and Functions.

A simple example:

```
2 + 3
```

Here the numerical values 2 and 3 are connected by the Operator "+". An arithmetic value again represents a value. In this case the value is 5.

Further examples:

```
a - 3

b + f(5)

2 + 3 * 6
```

Following the rule "Dot before Line" here 3 times 6 is calculated first and then 2 is added. This priority is in case of operators called precedence. A list of priorities can be found in the Precedence Table.

➡ Comparisons too are arithmetic expressions. The comparison operators return a truth value of "1" or "0", depening on whether the comparison was true or not. The expression "3 < 5" yields the value "1" (true).

Constant Expressions

An expression or portions of an expression can be constant. Portions of an expression can already

be calculated during Compiler runtime.

So e. g. the expression

```
12 + 123 - 15
```

is combined by the Compiler to

```
120.
```

In some cases expressions must be constant in order to be valid. E. g. also see Declaration of Array [Variables](#).

5.2.3   Data Types

Values always are of a certain data type. Integer values (integral values; whole numbered values) in CompactC are of the 8 Bit or 16 Bit wide data type, floating point values are always 4 byte long.

Data Type	Sign	Range	Bit
char	Yes	-128 … +127	8
unsigned char	No	0 … 255 8	8
byte	No	0 … 255 8	8
int	Yes	-32768 … +32767	16
unsigned int	No	0 … 65535	16
word	No	0 … 65535	16
float	Yes	±1.175e-38 to ±3.402e38	32

As one can see the data types "**unsigned char**" and **byte** as well as "**unsigned int**" and **word** are identical.

Strings

There is no explicit "String" data type. A string is based on a character array. The size of the array must be chosen in such a way that all characters of the string fit into the character array. Additionally some space is needed for a terminating character (decimal Zero) in order to indicate the end of the character string.

Type Conversion

In arithmetic expressions it is very often the case that individual values are not of the same type. So the data types of the following expression are combined (a is of type integer variable).

```
a + 5.5
```

In this case a is first converted into the data type **float** and then 5.5 is added. The data type of the result is also **float**. For data type conversion there are the following rules:

- If in a linkage of 8 Bit or 16 Bit integer values one of the two data types is sign afflicted ("**signed**") then the result of the expression is also sign afflicted. I. e. the operation is executed "**signed**".

- If one of the operands is of the **float** type then the result is also of the **float** type. If one of the two operands happens to be of the 8 Bit or 16 Bit data type then it will be converted into a **float** data type prior to the operation.

5.2.4 Variables

Variables can take on various values depending on the [Data Type](#) by which they have been defined. A variable definition appears as follows:

Type Variable Name;

When several variables of the same type need to be defined then these variables can be stated separated by commas:

Typ Name1, Name2, Name3, ...;

As types are allowed: **char**, **unsigned char**, **byte**, **int**, **unsigned int**, **word**, **float**

Examples:

int a;

int i,j;

float xyz;

Integer variables may have decimal figure values or Hex values assigned to. With Hex values the characters "**0x**" will be placed ahead of the figure. With variables of the sign afflicted data type negative decimal figures can be assigned to by putting a minus sign ahead of the figure.

➡ Numbers without period or exponent are normally of type signed integer. To explicitly define an unsigned integer write an "u" direct after the number.

Examples:

word a;
int i,j;

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

Floating Point Figures (data type **float**) may contain a decimal point and an exponent.

float x,y;

```
x=5.70;
y=2.3e+2;
x=-5.33e-1;
```

sizeof Operator

By the operator **sizeof**() the number of Bytes a variable takes up in memory can be determined.

Examples:

```
int s;
float f:

s=sizeof(f);   // the value of s is 4
```

➡ With arrays only the Byte length of the basic data type is returned. On Order to calculate the memory consumption of the array the value must be multiplied by the number of elements.

Array Variables

If behind the name, which in case of a variable definition is set in brackets, a figure value is written then an array has been defined. An array will arrange the space for a defined variable manifold in memory. With the following example definition

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

a tenfold memory space has been arranged for variable x. The first memory space can be addressed by `x[0]`, the second by `x[1]`, the third by `x[2]`, ... up to `x[9]`. When defining of course other index dimensions can also be chosen. The memory space of C-Control Pro is the only limit.

Multi dimensional arrays can also be declared by attaching further brackets during variable definition:

```
int x[3][4];      // array with 3*4 entries
int y[2][2][2];   // array with 2*2*2 entries
```

➡ Arrays may in CompactC have up to 16 indices (dimensions). The maximum value for an index is 65535. The indices of arrays are in any case zero based, i .e. each index will start with a 0.

➡ Only if the compiler option "Check Array Index Limits" is set, there will be a verification whether or not the defined index limits of an array have been exceeded. Otherwise, if an index becomes too large during program execution the access to alien variables will be tried which in turn may create a good chance for a program breakdown.

Table support by predefined Arrays

Since version 2.0 of the IDE arrays can be predefined with values:

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

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

    xloc= fglob;
}
```

Because there is more flash memory than RAM available, it is possible with the **flash** keyword to define data that are written in the flash memory only. These data can be copied to a RAM array with same dimensions with an assignment operation. In this example this is done through "xloc= fglob". This kind of assignment is not available in normal "C".

Strings

There is no explicit "String" data type. A string is based on a character array. The size of the array must be chosen in such a way that all characters of the string fit into the character array. Additionally some space is needed for a terminating character (decimal Zero) inorder to indicate the end of the character string.

Example for a character string with a 20 character maximum:

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

As an exception **char** arrays may have character strings assigned to. Here the character string is placed between quotation marks.

```
str1="hallo world!";
```

➡ Strings cannot be assigned to multi dimensional **Char** arrays. There are however tricks for advanced users:

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

single_str="A String";
Str_StrCopy(str_array,single_str,40);  // will copy single_str in the second string
```

This will work because with a gap of 40 characters after the first string there will in str_array be room for the second string.

Visibility of Variables

When variables are declared outside of functions then they will have global visibility. I. e. they can be addressed from every function. Variable declarations within functions produce local variables. Local variables can only be reached within the function. An example:

```
int a,b;

void func1(void)
```

```
{
    int a,x,y;
    // global b is accessible
    // global a is not accessible since concealed by local a
    // local x,y are accessible
    // u is not accessible since local to function main
}

void main(void)
{
    int u;
    // globale a,b are accessible
    // local u is accessible
    // x,y not accessible since local to function func1
}
```

Global variables have a defined memory space which is available throughout the entire program run.

➡ At program start the global variables will be initialized by zero.

Local variables will during calculation of a function be arranged on the stack. I. e. local variables exist in memory only during the time period in which the function is executed.

If with local variables the same name is selected as with a global variable then the local variable will conceal the global variable. While the program is working in the function where the identically named variable has been defined the global variable cannot be addressed.

Static Variables

With local variables the property **static** can be placed for the data type.

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

In opposition to normal local variables will static variables still keep their value even if the function is left. At a further call-up of the function the static variable will have the same contents as when leaving the function. In order to have the contents of a **static** variable defined at first access the static variables will equally to global variables at program start also be initialized by zero.

5.2.5   Operators

Priorities of Operators

Operators separate arithmetic expressions into partial expressions. The operators are then evaluated in the succession of their priorities (precedence). Expressions with operators of identical precedence will be calculated from left to right.
Example:

```
i= 2+3*4-5;  // result 9 => first 3*4, then +2, finally -5
```

The succession of the execution can be influenced by setting of parenthesis. Parenthesis have the highest priority.
If the last example should strictly be calculated from left to right, then:

```
i= (2+3)*4-5;  // result 15 => first 2+3, then *4, finally -5
```

A list of priorities can be found in Precedence Table.

5.2.5.1   Arithmetic Operators

All arithmetic operators with the exception of Modulo are defined for Integer and Floating Point data types. Modulo is restricted to data type Integer only.
➡ It must be observed that in an expression the figure 7 will have an Integer data type assigned to it. If a figure of data type **float** should be explicitly created then a decimal point has to be added: 7.0

Operator	Description	Example	Result
+	**Addition**	2+1	3
		3.2 + 4	7.2
-	**Subtraction**	2 - 3	-1
		22 - 1.1e1	11
*	**Multiplication**	5 * 4	20
/	**Division**	7 / 2	3
		7.0 / 2	3.5
%	**Modulo**	15 % 4	3
		17 % 2	1
-	**Negative Sign**	-(2+2)	-4

5.2.5.2   Bit Operators

Bit operators are only allowed for Integer data types

Operator	Description	Example	Result
&	**And**	0x0f & 3	3
		0xf0 & 0x0f	0
		Or	1	3	3
		0xf0	0x0f	0xff
^	**exclusive Or**	0xff ^ 0x0f	0xf0
		0xf0 ^ 0x0f	0xff
~	**Bit inversion**	~0xff	0
		~0xf0	0x0f

5.2.5.3   Bit-Shift Operators

Bit-Shift operators are only allowed for Integer data types. With a Bit-Shift operation a 0 will always be moved into one end.

Operator	Description	Example	Result
<<	shift to left	1 << 2	4
		3 << 3	24
>>	shift to right	0xff >> 6	3
		16 >> 2	4

5.2.5.4   In- /Decrement Operators

Incremental and decremental operators are only allowed for variables with Integer data types.

Operator	Description	Example	Result
variable++	first variable value, after access variable gets incremented by one (postincrement)	a++	a
variable--	first variable value, after access variable gets decremented by one (postdecrement)	a--	a
++variable	value of the variable gets incremented by one before access (preincrement)	++a	a+1
--variable	value of the variable gets decremented by one before access (predecrement)	--a	a-1

5.2.5.5   Comparison Operators

Comparison operators are allowed for **float** and Integer data types.

Operator	Description	Example	Result
<	smaller	1 < 2	1
		2 < 1	0
		2 < 2	0
>	greater	-3 > 2	0
		3 > 2	1
<=	smaller or equal	2 <= 2	1
		3 <= 2	0
>=	greater or equal	2 >= 3	0
		3 >= 2	1
==	equal	5 == 5	1
		1 == 2	0
!=	not equal	2 != 2	0

		2 != 5	1

5.2.5.6   Logical Operators

Logical operators are only allowed for Integer data types. Any value unequal **null** is meant to be a logical 1. Only **null** is valid as logical 0.

Operator	Description	Example	Result
&&	logical And	1 && 1	1
		5 && 0	0
||	logical Or	0 || 0	0
		1 || 0	1
!	logical Not	!2	0
		!0	1

5.2.6   Control Structures

Control structures allow to change the program completion depending on expressions, variables or external influences.

5.2.6.1   Conditional Valuation

With a conditional valuation expressions can be generated which will be conditionally calculated. The form is:

```
( Expression1 )  ? Expression2 : Expression3
```

The result of this expression is expression2, if expression1 had been calculated as unequal 0, otherwise the result is expression 3.

Examples:

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

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

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

5.2.6.2   do .. while

With a **do** .. **while** construct the instructions can depending on a condition be repeated in a loop:

```
do Instruction while( Expression );
```

The instruction or the Instruction Block is being executed. At the end the *Expression* is evaluated. If the result is unequal 0 then the execution of the expression will be repeated. The entire procedure will constantly be repeated until the *Expression* takes on the value 0.

Example:

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

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

➡ The essential difference between the **do .. while** loop and the normal **while** loop is the fact that in a **do .. while** loop the instruction is executed at least once.

break Instruction

A **break** instruction will leave the loop and the program execution will start with the next instruction after the **do .. while** loop.

continue Instruction

When executing **continue** within a loop there will immediately be a new calculation of the *Expression*. Depending on the result the loop will be repeated at unequal 0. At a result of 0 the loop will be terminated.

Example:

```
do
{
    a++;
    if(a>10) break; // will terminate loop
} while(1);  // endless loop
```

5.2.6.3 for

A **for** loop is normally used to program a definite number of loop runs.

```
for(Instruction1; Expression; Instruction2) Instruction3;
```

At first Instruction1 will be executed which normally contains an initialization. Following the evaluation of the *Expression* takes place. If the *Expression* is unequal 0 Instruction2 and Instruction3 will be executed and the loop will repeat itself. When *Expression* reaches the value 0 the loop will be

terminated. As with other loop types at Instruction3 an [Instruction Block](#) can be used instead of a single instruction.

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

➡️ It must be observed that variable i will within the loop run through values 0 through 9 rather than 1 through 10!

If a loop needs to be programmed with a different step width Instruction2 needs to be modified accordingly:

```c
for(i=0;i<100;i=i+3)  // variable i does now increment in steps to 3
{
    a=5*i;
}
```

break Instruction

A **break** instruction will leave the loop and the program execution starts with the next instruction after the **for** loop.

continue Instruction

continue will immediately initialize a new calculation of the *Expression*. Depending on the result Instruction2 will be executed at unequal 0 and the loop will repeat itself. A result of 0 will terminate the loop.

Example:

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

5.2.6.4   goto

Even though it should be avoided within structured programming languages, it is possible with **goto** to jump to a label within a procedure:

```c
// for loop with realized with goto
void main(void)
{
    int a;
```

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

5.2.6.5   if .. else

An **if** instruction does have the following syntax:

```
if( Expression ) Instruction1;
else Instruction2;
```

After the **if** instruction an Arithmetic Expression will follow in parenthesis. If this *Expression* is evaluated as unequal 0 then Instruction1 will be executed. By use of the command word **else** an alternative Instruction2 can be defined which will be executed when the *Expression* has been calculated as 0. The addition of an **else** instruction is optional and is not necessary.

Examples:

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

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

An Instruction Block can be defined instead of a single instruction.

Examples:

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

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

5.2.6.6   switch

If depending on the value of an expression various commands should be executed a **switch** instruction is an elegant solution:

```
switch( Expression )
{
    case constant_1:
        Instruction_1;
    break;

    case constant_2:
        Instruction_2;
    break;
    .
    .
    case constant_n:
        Instruction_n;
    break;
    default:    // default is optional
        Instruction_0;
};
```

The value of the *Expression* is calculated. Then the program execution will jump to the constant corresponding to the value of the *Expression* and will continue the program from there. If no constant corresponds to the value of the expression the **switch** construct will be left.

If a **default** is defined within a **switch** instruction then the instructions after **default** will be executed if no constant corresponding to the value of the instruction has been found.

Example:

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

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

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

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

➡ The execution of a **switch** statement is highly optimized. All values are stored inside a jumptable. Therefore exists a constraint that the calculated Expression is of type signed 16 Bit

Integer (-32768 .. 32767). For this reason a e.g. "**case** > 32767" is rather senseless.

break Instruction

A **break** will leave the **switch** instruction. If **break** is left out ahead of **case** then the instruction will be executed even when a jump to the preceeding **case** does take place:

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

    case 2:
        a++;  // is also executed at a value of a==1

    case 3:
        a++; // is also executed at a value of a==1 or a==2
}
```

In this example all three "a++" instructions are executed if a equals 1.

5.2.6.7   while

With a **while** instruction the instructions can depending on a condition be repeated in a loop.

```
while( Expression ) Instruction;
```

At first the *Expression* is evaluated. If the result is unequal 0 then the *Expression* is executed. After that the *Expression* is again calculated and the entire procedure will constantly be repeated until the *Expression* takes on the value 0. An [Instruction Block](#) can be defined instead of a single instruction.

Example:

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

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

break Instruction

If a **break** is executed within the loop then the loop will be left and the program execution starts with the next instruction after the **while** loop.

continue Instruction

An execution of **continue** within a loop will <u>immediately</u> initialize a new calculation of the *Expression*. Depending on the result the loop will be repeated at unequal 0. A result of 0 will terminate the loop.

Example:

```
while(1)  // endless loop
{
    a++;
    if(a>10) break; // will terminate the loop
}
```

5.2.7   Functions

In order to structure a larger program it is separated into several sub-functions. This not only improves the readability but allows to combine all program instructions repeatedly appearing in functions. A program does in any case contain the function "main", which is started in first place.   After that other functions can be called up.
A simple example:

```
void func1(void)
{
    // instructions in function func1
    .
    .
}

void main(void)
{
    // function func1 will be called up twice
    func1();
    func1();
}
```

Parameter Passing

In order to enable functions to be flexibly used they can be set up parametric. To do this the parameters for the function are separated by commas and passed in parenthesis after the function name. Similar to the variables declaration first the data type and then the parameter name are stated. If no parameter is passed then **void** has to be set into the parenthesis.
An example:

```
void func1(word param1, float param2)
{
    Msg_WriteHex(param1);  // first parameter output
    Msg_WriteFloat(param2);  // second parameter output
}
```

➡ Similar to local variables passed parameters are only visible within the function itself.

In order to call up function func1 by use of the parameters the parameters for call up should be written in the same succession as they have been defined in func1. If the function does not get parameters the parenthesis will stay empty.

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

    func1(128,12.0);  // you can passs numerical constants
    a=100;
    f=12.0;
    func1(a+28,f); // or yet variables too and even numerical expressions
}
```

➡ When calling up a function all parameters must always be stated. The following call up is inadmissible:

```
func1();        // func1 gets 2 parameters!
func1(128);     // func1 gets 2 parameters!
```

Return Parameters

It is not only possible to pass parameters. A function can also offer a return value. The data type of this value is during function definition entered ahead of the function name. If no value needs to be returned the data type used will be **void**.

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

The return value is within the function stated as instruction "**return** *Expression*". If there is a function of the **void** type then the **return** instruction can be used without parameters in order to leave the function.

References

Since it is not possible to pass on arrays as parameters the access to parameters is possible through references. For this a pair of brackets is written after the parameter names in the parameter declaration of a function.

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

    i=0;
```

```
    while(str[i]) i++;  // repeat character as long as unequal zero
    return(i);
}

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

    text="hello world";
    len=StringLength(text);
}
```

In **main** the reference of text is presented as parameters to the function StringLength. If in a function a normal parameter is changed then the change is not visible outside this function. With references this is different. Through parameter *str* in StringLength the contents of *text* can be changed since *str* is only the reference (pointer) to the array variable *text*.

➡ Presently arrays can only be passed "by Reference"!

Strings as Parameter

Since Version 2.0 of the IDE it is possible to call functions with a string as parameter. The called function gets the string as reference. Since references are RAM based and predefined strings are stored in the flash memory, the compiler creates internally an anonymous variable, and copies the data from flash into memory.

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

void main(void)
{
    int len;

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

5.2.8   Tabellen

5.2.8.1   Operator Precedence

Rang	Operator
13	( )
12	++ -- ! ~ - (negatives Vorzeichen)
11	* / %
10	+ -

9	<< >>
8	< <= > >=
7	== !=
6	&
5	^
4	|
3	&&
2	||
1	? :

5.2.8.2 Operators

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

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

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

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

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

	Bitoperatoren
&	**Und**
|	**Oder**
^	**exclusives Oder**
~	**Bitinvertierung**

5.2.8.3 Reserved Words

The following words are **reserved** and cannot be used as identifier:

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

5.3 BASIC

The second programming language for the C-Control Pro Mega Module is BASIC. The Compiler translates the BASIC commands into a Bytecode which is then processed by the C-Control Pro Interpreter. The language volume of the BASIC dialect used here corresponds to a large extent to the industry standard of the large software suppliers.

The following language constructs are missing:

- Object oriented programming
- Structures
- Constants

Detailed program examples can be found in directory "Demo Programs" which was installed along with the design interface. There example solutions can be found for almost every field of purpose of the C-Control Pro Module.

The following chapters offer a systematical introduction to syntax and semantics of C-Control Pro BASIC.

5.3.1 Program

A program consists of a number of instructions (such as e. g. "a=5;") which are distributed among various Functions. The starting function, which must be present in every program, is the function "main()". The following is a simplistic program able to print a number into the output window:

```
Sub main()
    Msg_WriteInt(42)  // the answer to anything
End Sub
```

Projects

A program can be separated into several files which are combined in a project (see Project Management). In addition to these project files Libraries can be added to the project which are able to offer functions used by the program.

5.3.2 Instructions

Instruction

An instruction consists of several reserved command words, identifiers and operators and is at the end terminated by the end of the line. In order to separate various elements of an instruction there are spaces in between the instruction elements which are called "*Whitespaces*". By "spaces" space characters, tabulators and line feeds ("C/R and LF") are meant. It is of no consequence whether a space is built by one or several "*Whitespaces*".

Simple Instruction:

```
a=   5
```

➡ An instruction does not necessarily have to completely stand in one line. By use of the "_" character (low dash) it is possible to extend the instruction into the next line.

```
If a=5 _ ' instruction across two lines
a=a+10
```

➡ It is also possible to place more than one instruction into the same line. The ":" character (colon) will then separate the individual instructions. For reason of better readability however this option should rather seldom be used.

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

Comments

Comments

There are two types of commentaries, which are the single line and the multi line commentaries. The text within commentaries is ignored by the Compiler.

- Single line commentaries start with a single quotation mark and end up at the line's end.
- Multi line commentaries start with "/*" and end up with "*/".

```
/* a
multi line
commentary */

' a single line commentary
```

Identifier

Identifiers are the names of <u>Functions</u> or <u>Variables</u>.

- Valid characters are letters (A-Z,a-z), numbers (0-9) and the low dash ('_')
- An identifier always starts with a letter
- Upper and lower case writings are differentiated
- <u>Reserved Words</u> are not allowed as identifiers

- The length of an identifier is unlimited

Arithmetic Expressions

An arithmetic expression is a quantity of values connected by Operators. In this case quantities can either be Figures, Variables or Functions.

A simple example:

```
2 + 3
```

Here the numerical values 2 and 3 are connected by the Operator "+". An arithmetic value again represents a value. In this case the value is 5.

Further examples:

```
a - 3

b + f(5)

2 + 3 * 6
```

Following the rule "Dot before Line" here 3 times 6 is calculated first and then 2 is added. This priority is in case of operators called precedence. A list of priorities can be found in the Precedence Table.

➡ Comparisons too are arithmetic expressions. The comparison operators return a truth value of "1" or "0", depending on whether the comparison was true or not. The expression "3 < 5" yields the value "1" (true).

Constant Expressions

An expression or portions of an expression can be constant. Portions of an expression can already be calculated during Compiler runtime.

So e. g. the expression

```
12 + 123 - 15
```

is combined by the Compiler to

```
120.
```

In some cases expressions must be constant in order to be valid. E. g. also see Declaration of Array Variables.

5.3.3   Data Types

Values always are of a certain data type. Integer values (integral values; whole numbered values) in BASIC are of the 8 Bit or 16 Bit wide data type, floating point values are always 4 byte long.

Data Type	Sign	Range	Bit
Char	Yes	-128 … +127	8
Byte	No	0 … 255 8	8
Integer	Yes	-32768 … +32767	16
Word	No	0 … 65535	16
Single	Yes	±1.175e-38 to ±3.402e38	32

Strings

There is no explicit "String" data type. A string is based on a character array. The size of the array must be chosen in such a way that all characters of the string fit into the character array. Additionally some space is needed for a terminating character (decimal Zero) in order to indicate the end of the character string.

Type Conversion

In arithmetic expressions it is very often the case that individual values are not of the same type. So the data types

```
a + 5.5
```

In this case a is first converted into the **Single** data type and then 5.5 is added.
The data type of the result is also **Single**. For data type conversion there are the following rules:

- If in a linkage of 8 Bit or 16 Bit integer values one of the two data types is sign afflicted then the result of the expression is also sign afflicted.

- If one of the operands is of the **Single** type then the result is also of the **Single** type. If one of the two operands happens to be of the 8 Bit or 16 Bit data type then it will be converted into a **Single** data type prior to the operation.

5.3.4   Variables

Variables can take on various values depending on the Data Type by which they have been defined. A variable definition appears as follows:

Dim Variable Name **As** Type

When several variables of the same type need to be defined then these variables can be stated separated by commas:

Dim Name1, Name2, Name3 **As Integer**

As types are allowed: **Char**, **Byte**, **Integer**, **Word**, **Single**

Examples:

```
Dim a As Integer

Dim i,j As Integer

Dim xyz As Single
```

Integer variables may have decimal figure values or Hex values assigned to. With Hex values the characters "**&H**" will be placed ahead of the figure. Just as with CompactC it is also allowed to place the prefix "**0x**" ahead of the Hex values. With variables of the sign afflicted data type negative decimal figures can be assigned to by putting a minus sign ahead of the figure.

➡ Numbers without period or exponent are normally of type signed integer. To explicitly define an unsigned integer write an "u" direct after the number.

Examples:

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

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

Floating Point Figures (data type **Single**) may contain a decimal point and an exponent.

```
Dim x,y As Single

x=5.70
y=2.3e+2
x=-5.33e-1
```

SizeOf Operator

By the operator **SizeOf**() the number of Bytes a variable takes up in memory can be determined.

Examples:

```
Dim s As Integer
Dim f As Single

s=SizeOf(f)   ' the value of s is 4
```

➡ With arrays only the Byte length of the basic data type is returned. On order to calculate the memory consumption of the array the value must be multiplied by the number of elements.

Array Variables

If behind the name, which in case of a variable definition is set in parenthesis, a figure value is written then an array has been defined. An array will arrange the space for a defined variable manifold in memory. With the following example definition

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

a tenfold memory space has been arranged for variable x. The first memory space can be addressed by x[0], the second by x[1], the third by x[2], ... up to x[9]. When defining of course other index dimensions can also be chosen. The memory space of C-Control Pro is the only limit.

Multi dimensional arrays can also be declared by attaching further indices during variable definition, which have to be separated by commas,:

```
Dim x(3,4) As Integer  ' array with 3*4 entries
Dim y(2,2,2) As Integer ' array with 2*2*2 entries
```

➡ Arrays may in BASIC have up to 16 indices (dimensions). The maximum value for an index is 65535. The indices of arrays are in any case zero based, i .e. each index will start with a 0.

➡ Only if the compiler option "Check Array Index Limits" is set, there will be a verification whether or not the defined index limits of an array have been exceeded. Otherwise, if an index becomes too large during program execution the access to alien variables will be tried which in turn may create a good chance for a program breakdown.

Table support by predefined Arrays

Since version 2.0 of the IDE arrays can be predefined with values:

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

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

    xloc= fglob
End Sub
```

Because there is more flash memory than RAM available, it is possible with the **flash** keyword to define data that are written in the flash memory only. These data can be copied to a RAM array with same dimensions with an assignment operation. In this example this is done through "xloc= fglob".

Strings

There is no explicit "String" data type. A string is based on an array of data type **Char**. The size of the array must be chosen in such a way that all characters of the string fit into the character array. Additionally some space is needed for a terminating character (decimal Zero) inorder to indicate the end of the character string.

Example for a character string with a 20 character maximum:

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

As an exception **Char** arrays may have character strings assigned to. Here the character string is placed between quotation marks.

```
str1="hallo world!"
```

➡ Strings cannot be assigned to multi dimensional **Char** arrays. There are however tricks for advanced users:

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

Single_str="A String"
Str_StrCopy(str_array,Single_str,40)  // will copy Single_str in the second string
```

This will work because with a gap of 40 characters after the first string there will in str_array be room for the second string.

Visibility of Variables

When variables are declared outside of functions then they will have global visibility. I. e. they can be addressed from every function. Variable declarations within functions produce local variables. Local variables can only be reached within the function. An example:

```
Dim a,b As Integer

Sub func1()
Dim a,x,y As Integer
    // global b is accessible
    // global a is not accessible since concealed by local a
    // local x,y is accessible
    // u is not accessible since local to function main
End Sub

Sub main()
    Dim u As Integer
    // global a,b is accessible
    // local u is accessible
    // x,y u is not accessible since local to function main
End Sub
```

Global variables have a defined memory space which is available throughout the entire program run.

➡ At program start the global variables will be initialized by zero.

Local variables will during calculation of a function be arranged on the stack. I. e. local variables exist in memory only during the time period in which the function is executed.

If with local variables the same name is selected as with a global variable then the local variable will conceal the global variable. While the program is working in the function where the identically named variable has been defined the global variable cannot be addressed.

Static Variables

With local variables the property **Static** can be placed for the data type.

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

In opposition to normal local variables will static variables still keep their value even if the function is left. At a further call-up of the function the static variable will have the same contents as when leaving the function. In order to have the contents of a **Static** variable defined at first access the static variables will equally to global variables at program start also be initialized by zero.

5.3.5   Operators

Priorities of Operators

Operators separate arithmetic expressions into partial expressions. The operators are then evaluated in the succession of their priorities (precedence). Expressions with operators of identical precedence will be calculated from left to right.
Example:

```
i= 2+3*4-5  ' result 9 => first 3*4, then +2, finally -5
```

The succession of the execution can be influenced by setting of parenthesis. Parenthesis have the highest priority.
If the last example should strictly be calculated from left to right, then:

```
i= (2+3)*4-5  ' result 15 => first 2+3, then *4, finally -5
```

A list of priorities can be found in Precedence Table.

5.3.5.1   Arithmetic Operators

All arithmetic operators with the exception of Modulo are defined for Integer and Floating Point data types. Modulo is restricted to data type Integer only.
➡️ It must be observed that in an expression the figure 7 will have an Integer data type assigned to it. If a figure of data type **Single** should be explicitly created then a decimal point has to be added: 7.0

Operator	Description	Example	Result

+	**Addition**	2+1	3
		3.2 + 4	7.2
-	**Subtraction**	2 - 3	-1
		22 - 1.1e1	11
*	**Multiplication**	5 * 4	20
/	**Division**	7 / 2	3
		7.0 / 2	3.5
%	**Modulo**	15 % 4	3
		17 % 2	1
-	**Negative Sign**	-(2+2)	-4

5.3.5.2 Bitoperators

Bit operators are only allowed for Integer data types

Operator	**Description**	**Example**	**Result**
And	**And**	&H0f And 3	3
		&Hf0 And &H0f	0
Or	**Or**	1 Or 3	3
		&Hf0 Or &H0f	&Hff
Xor	**exclusive Or**	&Hff Xor &H0f	&Hf0
		&Hf0 Xor &H0f	&Hff
Not	**Bit inversion**	Not &Hff	0
		Not &Hf0	&H0f

5.3.5.3 Bit-Shift Operators

Bit-Shift operators are only allowed for Integer data types. With a Bit-Shift operation a 0 will always be moved into one end.

Operator	**Description**	**Example**	**Result**
<<	**shift to left**	1 << 2	4
		3 << 3	24
>>	**shift to right**	&Hff >> 6	3
		16 >> 2	4

5.3.5.4 In-/Decrement Operators_2

Incremental and decremental operators are only allowed for variables with Integer data types.

Operator	Description	Example	Result
variable++	first variable value, after access variable gets incremented by one (postincrement)	a++	a
variable--	first variable value, after access variable gets decremented by one (postdecrement)	a--	a
++variable	value of the variable gets incremented by one before access (preincrement)	++a	a+1
--variable	value of the variable gets decremented by one before access (predecrement)	--a	a-1

➡ These operators are normally not a part of a Basic dialect and have their origin in the world of C inspired languages.

5.3.5.5   Comparison Operators

Comparison operators are allowed for **Single** and Integer data types.

Operator	Description	Example	Result
<	smaller	1 < 2 2 < 1 2 < 2	1 0 0
>	greater	-3 > 2 3 > 2	0 1
<=	smaller or equal	2 <= 2 3 <= 2	1 0
>=	greater or equal	2 >= 3 3 >= 2	0 1
=	equal	5 = 5 1 = 2	1 0
<>	not equal	2 <> 2 2 <> 5	0 1

5.3.6   Control Structures

Control structures allow to change the program completion depending on expressions, variables or external influences.

5.3.6.1   Do Loop While

With a **Do ... Loop While** construct the instructions can depending on a condition be repeated in a loop:

```
Do
    Instructions
Loop While Expression
```

The instructions are being executed. At the end the *Expression* is evaluated. If the result is unequal 0 then the execution of the expression will be repeated. The entire procedure will constantly be repeated until the *Expression* takes on the value 0.

Examples:

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

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

➡ The essential difference between the **Do Loop While** loop and the normal **Do While** loop is the fact that in a **Do Loop While** loop the instruction is executed at least once.

Exit Instruction

An **Exit** instruction will leave the loop and the program execution starts with the next instruction after the **Do Loop While** loop.

Example:

```
Do
    a=a+1
    If a>10 Then
        Exit ' Will terminate loop
    End If
Loop While 1  ' Endless loop
```

5.3.6.2   Do While

With a **while** instruction the instructions can depending on a condition be repeated in a loop:

```
Do While Expression
    Instructions
End While
```

At first the *Expression* is evaluated. If the result is unequal 0 then the expression is executed. After that the *Expression* is again calculated and the entire procedure will constantly be repeated until the *Expression* takes on the value 0.

Examples:

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

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

Exit Instruction

If an **Exit** instruction is executed within a loop than the loop will be left and the program execution starts with the next instruction after the **While** loop.

Example:

```
Do While 1   ' Endless loop
    a=a+1
    If a>10 Then
        Exit ' Will terminate loop
    End If
End While
```

5.3.6.3   For Next

A **For Next** loop is normally used to program a definite number of loop runs.

```
For Counter Variable=Startvalue To Endvalue Step Stepwidth
    Instructions
Next
```

The Counter Variable is set to a Start Value. Then the instructions are repeated until the End Value is reached. With each loop run the value of the Counter Variable is increased by one step width which may also be negative. The stating of the step width after the End Value is optional. If no Step Width is stated it has the value 1.

➡ Since the **For Next** loop will be used to especially optimized the counter variable must be of the Integer type.

Example:

```
For i=1 To 10
    If i>a Then
        a=i
    End If
    a=a-1
Next
```

```
For i=1 To 10 Step 3   'Increment i in steps of 3
    If i>3 Then
        a=i
    End If
    a=a-1
Next
```

➡ In this location please note again that arrays are in any case zero based. A For Next loop should thus rather run from 0 through 9.

Exit Instruction

An **Exit** instruction will leave the loop and the program execution starts with the next instruction after the **For** loop.

Example:

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

5.3.6.4   Goto

Even though it should be avoided within structured programming languages, it is still possible with **goto** to jump to a label within a procedure. In order to mark a label the command word **Lab** is set in front of the label name.

```
' For loop with goto will realize

Sub main()
    Dim a As Integer

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

5.3.6.5   If .. Else

An **If** instruction does have the following syntax:

```
If Expression1 Then
    Instructions1
ElseIf Expression2 Then
```

```
        Instructions2
Else
        Instructions3
End If
```

After the **if** instruction an [Arithmetic Expression](#) will follow. If this *Expression* is evaluated as unequal 0 then Instruction1 will be executed. By use of the command word **else** an alternative Instruction2 can be defined which will be executed when the *Expression* has been calculated as 0. The addition of an **else** instruction is optional and not really necessary.

If directly in an **Else** branch an **If** instruction needs again to be placed then it is possible to initialize an **If** again direcly by use of an **ElseIf**. Thus the new **If** does not need to be interlocked into an **Else** block and the source text remains more clearly.

Examples:

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

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

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

5.3.6.6   Select Case

If depending on the value of an expression various commands should be executed then a **Select Case** instruction seems to be an elegant solution:

```
Select Case Expression
    Case constant_comparison1
        Instructions_1
    Case constant_comparison2
        Instructions_2
    .
    .
    Case constant_comparison_x
        Instructions_x
    Else    ' Else is optional
        Instructions
```

```
End Case
```

The value of the *Expression* is calculated. Then the program execution will jump to the first constant comparison that can be evaluated as true and will continue the program from there. If no constant comparison can be fulfilled the **Select Case** construct will be left.

For constant comparisons special comparisons and ranges can be defined . Here examples for all possibilities:

Comparison	Execute on
Constant, **=** Constant	Expression **equal** Constant
< Constant	Expression **smaller** Constant
<= Constant	Expression **smaller equal** Constant
> Constant	Expression **greater** Constant
>= Constant	Expression **greater equal** Constant
<> Constant	Expression **unequal** Constant
Constant1 To Constant2	Constant1 **<=** Expression **<=** Constant2

➡ The new features that allow to use comparisons are introduced for Select Case statements with IDE version 1.71. This extension is not available for CompactC switch statements.

➡ The execution of a **Select Case** statement is highly optimized. All values are stored inside a jumptable. Therefore exists a constraint that the calculated Expression is of type signed 16 Bit Integer (-32768 .. 32767). For this reason a e.g. "**Case** > 32767" is rather senseless.

Exit Instruction

An **Exit** will leave the **Select Case** instruction.

If an **Else** is defined within a **Select Case** instruction then the instructions after **Else** will be executed if no constant comparison could be fulfilled.

Example:

```
Select Case a+2
    Case 1
        b=b*2
    Case = 5*5
        b=b+2
    Case 100 And &Hf
        b=b/c
    Case < 10
        b=10
    Case <= 10
        b=11
    Case 20 To 30
        b=12
    Case > 100
        b=13
```

```
    Case >= 100
         b=14
    Case <> 25
         b=15
    Else
         b=b+2
End Case
```

➡ In CompactC the instructions will be continued after a **Case** instruction until a **break** comes up or the **switch** instruction is left. With BASIC this is different: Here the execution of the commands will break off after a **Case**, if the next **Case** instruction is reached.

5.3.7   Functions

In order to structure a larger program it is separated into several sub-functions. This not only improves the readability but allows to combine all program instructions repeatedly appearing in functions. A program does in any case contain the function "main", which is started in first place. After that other functions can be called up from main. A simple example:

```
Sub func1()
    ' Instructions in function func1
    .
    .
End Sub

Sub main()
    ' Function func1 will be called up twice
    func1()
    func1()
End Sub
```

Parameter Passing

In order to enable functions to be flexibly used they can be set up parametric. To do this the parameters for the function are separated by commas and passed in parenthesis after the function name. Similar to the variables declaration first the parameter name and then the data type is stated. If no parameter is passed then the parenthesis will stay empty.
An example:

```
Sub func1(param1 As Word, param2 As Single)
    Msg_WriteHex(param1)  ' first parameter output
    Msg_WriteFloat(param2)  ' second parameter output
End Sub
```

➡ Similar to local variables passed parameters are only visible within the function itself.

In order to call up function func1 by use of the parameters the parameters for call up should be written in the same succession as they have been defined in func1. If the function does not get parameters the parenthesis will stay empty.

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

    func1(128,12.0)  ' you can pass Numerical constants
    a=100
    f=12.0
    func1(a+28,f) ' or yet variables too and even numerical expressions
End Sub
```

➡️ When calling up a function all parameters must always be stated. The following call up is inadmissible:

```
func1()         ' func1 gets 2 parameters!
func1(128)      ' func1 gets 2 parameters!
```

Return Parameters

It is not only possible to pass parameters. A function can also offer a return value. The data type of this value is during function definition entered after the parameter list of the function.

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

The return value is within the function stated as instruction "**return** *Expression*". If there is a function without return value then the **return** instruction can be used without parameters in order to leave the function.

References

Since it is not possible to pass on arrays as parameters the access to parameters is possible through references. For this the attribute "**ByRef**" is written ahead of the parameter name in the parameter declaration of a function.

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

    i=0
    Do While str(i)
        i=i+1  ' Repeat character as long as unequal zero

    End While
    Return i
End Sub

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

```
        Text="hello world"
        Len=StringLength(Text)
End Sub
```

In **main** the reference of text is presented as parameters to the function StringLength. If in a function a normal parameter is changed then the change is not visible outside this function. With references this is different. Through parameter *str* can in StringLength the contents of *text* be changed since *str* is only the reference (pointer) to the array variable *text*.

➡ Presently arrays can only be presented "by Reference"!

Strings as Parameter

Since Version 2.0 of the IDE it is possible to call functions with a string as parameter. The called function gets the string as reference. Since references are RAM based and predefined strings are stored in the flash memory, the compiler creates internally an anonymous variable, and copies the data from flash into memory.

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

Sub main()
    Dim Len As Integer

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

5.3.8   Tables

5.3.8.1   Operator Precedence

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

5.3.8.2 Operators

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

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

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

	Bitoperatoren
And	Und
Or	Oder
Xor	exclusives Oder
Not	Bitinvertierung

5.3.8.3 Reserved Words

The following words are **reserved** and cannot be used as identifiers:

And	As	ByRef	Byte	Case
Char	Dim	Do	Else	ElseIf
End	Exit	False	For	Goto
If	Integer	Lab	Loop	Mod
Next	Not	Opc	Or	Return
Select	Single	SizeOf	Static	Step
Sub	Then	To	True	While
Word	Xor			

5.4 Assembler

With IDE Version 2.0 it is possible to integrate Assembler routines into a project. The used Assembler is the GNU Open Source Assembler AVRA. The sources of the AVRA Assembler can be found in the installation directory "GNU". Assembler routines that are called from CompactC and Basic run in full CPU speed, in contrary to the Bytecode Interpreter. It is possible to pass paraneters to Assembler procedures and get their return values. Also global CompactC and Basic variables can be accessed. The compiler recognizes assembler files with their ".asm" ending. Assembler sources are added to a project like CompactC or Basic files.

➡ The programming in assembly language is only recommended for the advanced user of the system. The programming is very complex and error prone, and should only be used by these people that have a very good knowledge of the system.

Literature

You can find manifold literature about assembly language programming on the internet and in the book trade. Important are the "AVR Instruction Reference Manual" that can be found on the Atmel website and in the "Manual" directory of the C-Control Pro installation, and the "AVR Assembler User Guide" from the Atmel website.

5.4.1 An Example

The structure of assembly routines is explained in the following example (also included in the demo programs). In the project the CompactC source code file must have the ending ".cc", the assembler sourcefiles have to end with ".asm".

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

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

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

The procedures *proc1* and *proc2* must first be declared, before they can be called. This happens with the keyword **$asm**. The declaration in Basic looks similar:

```
' Basic delaration of assembler routines
$Asm("tag1") proc1()
$Asm("tag2") proc2(a As Integer, b As Single, c As Byte) As Integer
```

The strings "tag1" and "tag" are visible in the declaration. These strings are defined in a ".def" file, if the Assembler routines are really called from the CompactC and Basic source. In this case the ".def" file looks like:´

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

When all the routines in the Assembler sources are placed in ".ifdef ..." directions, only the routines are assembled that are really called. This save space at the code generation. Additionally the position of the global variables are stored in the definition file. The ".def" file is automatically included in the translation of the assembler files, it needed not to be manually included.

Here follows the assembler source of procedure *proc1*. In this source the global variable *glob1* is set to the value 42.

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

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

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

  ret
.endif
```

In the second part of the assembler sources the passed parameters "a" and "c" are added as integers, and then the sum is returned.

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

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

    LDD R30, Y+0  ; load byte parsmeter "c" into Z (R30)
```

```
    CLR R31          ; hi byte zero because parameter is byte


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

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

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

    ret
.endif
```

5.4.2   Data Access

Global Variables

In the Bytecode Interpreter in the register R8 and R9 lies the 16-Bit pointer to the end of the global variable memory. If a global variable that is defined in the ".def" file should be accessed, the address of the variable can be calculated when the variable position is subtracted from the R8, R9 16-Bit pointer. This looks like:

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

When the address of the global variable is in the X register pair (R26,R27), the desired value (in our example 42) can be written there:

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

Parameter Passing

Parameters are passed on the stack of the Bytecode Interpreter. The stackpointer (SP) lies in the register pair R10,R11. Are parameters passed, they are written one after another onto the stack. Since the stack grows to the bottom, in our example (integer a, floating point b, byte c) the memory layout looks like this:

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

If the variables a and c should be accessed, a will be found at SP+5 and c at SP. In the following Assembler code the stack pointer SP (R10,R11) will be copied in the register pair Y (R28,R29), and the parameters a and c are loaded indirect via Y.

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

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

The parameter a and c are now in the register pairs X and Z. Now they can be added:

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

Return Parameters

In the routine *proc2* the sum is returned. Return parameters are written on the Parameter Stack (PSP) of the Bytecode Interpreter. The pointer to the PSP lies in the register pair R6,R7. To return a parameter the PSP pointer must be increased by 4 before the parameter can be written. In opposite to the normal parameter passing the type of the return parameter is not important. All parameter on the Parameter Stack have the same length of 4 bytes.

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

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

5.4.3 Guideline

The most important topics on how to program in Assembler for C-Control Pro are explained here:

- Assembler calls are atomar. An Assembler call cannot be interrupted by Multithreading or an Bytecode Interruptroutine. This is similar to Library calls. An interrupt is recorded immediately by the internal interrupt structure, but the corresponding Bytecode interrupt routine is called after the assembler procedure has been ended.

- The register R0, R1, R22, R23, R24, R25, R26, R27, R30 und R31 can be used in Assembler routines without backup. If other register are used, the contents must be saved first. Normally these values are stored on the stack. E.g.

```
at begin: PUSH R5
          PUSH R6
...
at end:   POP R6
          POP R5
```

- An Assembler routine is left with a "RET" instruction. At this point the CPU stack must be in the same state as before the call. The contents of the register that need to be backuped must be restored.

- Debugging only works in the Bytecode Interpreter, it is not possible to debug in Assembler.

- The Bytecode Interpreter has a fixed memory layout. In **no** case use Assembler directives like . **byte**, **.db**, **.dw**, .dseg or similar. In an access to the data segment this would cause the Assembler to overwrite memory that is used by the Bytecode Interpreter. If global variables are needed, they should be declared in CompactC and Basic, and then can be accessed like described in the chapter Data Access.

- **Do not** set the address of an Assembler routine with .org. The IDE generates itself a .org directive when starting the AVRA Assembler.

5.5   ASCII Table

ASCII Table				
CHAR	**DEC**	**HEX**	**BIN**	**Description**
NUL	000	000	00000000	Null Character
SOH	001	001	00000001	Start of Header
STX	002	002	00000010	Start of Text
ETX	003	003	00000011	End of Text
EOT	004	004	00000100	End of Transmission
ENQ	005	005	00000101	Enquiry
ACK	006	006	00000110	Acknowledgment
BEL	007	007	00000111	Bell
BS	008	008	00001000	Backspace
HAT	009	009	00001001	Horizontal TAB
LF	010	00A	00001010	Line Feed
VT	011	00B	00001011	Vertical TAB

FF	012	00C	00001100	Form Feed
CR	013	00D	00001101	Carriage Return
SO	014	00E	00001110	Shift Out
SI	015	00F	00001111	Shift In
DLE	016	010	00010000	Data Link Escape
DC1	017	011	00010001	Device Control 1
DC2	018	012	00010010	Device Control 2
DC3	019	013	00010011	Device Control 3
DC4	020	014	00010100	Device Control 4
NAK	021	015	00010101	Negative Acknowledgment
SYN	022	016	00010110	Synchronous Idle
ETB	023	017	00010111	End of Transmission Block
CAN	024	018	00011000	Cancel
EM	025	019	00011001	End of Medium
SUB	026	01A	00011010	Substitute
ESC	027	01B	00011011	Escape
FS	028	01C	00011100	File Separator
GS	029	01D	00011101	Group Separator
RS	030	01E	00011110	Request to Send, Record Separator
US	031	01F	00011111	Unit Separator
SP	032	020	00100000	Space
!	033	021	00100001	Exclamation Mark
"	034	022	00100010	Double Quote
#	035	023	00100011	Number Sign
$	036	024	00100100	Dollar Sign
%	037	025	00100101	Percent
&	038	026	00100110	Ampersand
'	039	027	00100111	Single Quote
(	040	028	00101000	Left Opening Parenthesis
)	041	029	00101001	Right Closing Parenthesis
*	042	02A	00101010	Asterisk

+	043	02B	00101011	Plus
,	044	02C	00101100	Comma
-	045	02D	00101101	Minus or Dash
.	046	02E	00101110	Dot

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

G	071	047	01000111	
H	072	048	01001000	
I	073	049	01001001	
J	074	04A	01001010	
K	075	04B	01001011	
L	076	04C	01001100	
M	077	04D	01001101	
N	078	04E	01001110	
O	079	04F	01001111	
P	080	050	01010000	
Q	081	051	01010001	
R	082	052	01010010	
S	083	053	01010011	
T	084	054	01010100	
U	085	055	01010101	
V	086	056	01010110	
W	087	057	01010111	
X	088	058	01011000	
Y	089	059	01011001	
Z	090	05A	01011010	
[	091	05B	01011011	Left Opening Bracket
\	092	05C	01011100	Back Slash
]	093	05D	01011101	Right Closing Bracket
^	094	05E	01011110	Caret

CHAR	DEC	HEX	BIN	Description
_	095	05F	01011111	Underscore
`	096	060	01100000	
a	097	061	01100001	
b	098	062	01100010	

c	099	063	01100011	
d	100	064	01100100	
e	101	065	01100101	
f	102	066	01100110	
g	103	067	01100111	
h	104	068	01101000	
i	105	069	01101001	
j	106	06A	01101010	
k	107	06B	01101011	
l	108	06C	01101100	
m	109	06D	01101101	
n	110	06E	01101110	
o	111	06F	01101111	
p	112	070	01110000	
q	113	071	01110001	
r	114	072	01110010	
s	115	073	01110011	
t	116	074	01110100	
u	117	075	01110101	
v	118	076	01110110	
w	119	077	01110111	
x	120	078	01111000	
y	121	079	01111001	
z	122	07A	01111010	
{	123	07B	01111011	Left Opening Brace
|	124	07C	01111100	Vertical Bar
}	125	07D	01111101	Right Closing Brace
~	126	07E	01111110	Tilde
DEL	127	07F	01111111	Delete

Part

6

6 Libraries

In this part of the documentation all attached Help functions are described which allow the user to comfortably gain access to the hardware. At the beginning of each function the syntax for CompactC and BASIC is shown. After that the description of functions and involved parameters will follow.

6.1 Internal Functions

To allow the Compiler to recognise the internal functions present in the Interpreter these functions must be defined in library "IntFunc_Lib.cc". If this library is not tied in no outputs can be performed by the program. The following would e. g. be a typical entry in "IntFunc_Lib.cc":

```
void Msg_WriteHex$Opc(0x23)(Word val);
```

This definition states that the function ("Msg_WriteHex") in the Interpreter is called up by a jump vector of 0x23 and a word has to be transferred to the stack as a parameter.

➡ Changes in the library "IntFunc_Lib.cc" may cause that the functions declared there can no longer be executed correctly.

6.2 General

In this chapter all single functions are collected that cannot be categorized to other chapters in the library.

6.2.1 AbsDelay

General Functions

Syntax

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

```
Sub AbsDelay(ms As Word);
```

Description

The function Absdelay() waits for a specified number of milliseconds.

➡ This function works in a very accurate manner, but suspends the bytecode interpreter. A thread change will not happen during this time. Interrupts are recognized, but will not be processed since the interpreter is necessary for this operations.

➡ Please use Thread_Delay instead of AbsDelay if you work with threads. If you call an AbsDelay(1000) in an endless loop nevertheless, the following will happen: Since the thread is changing after 5000 cycles (default value) to the next thread, the next thread will begin after after about 5000 * 1000ms. This happens because an AbsDelay() call will be treated like on cycle.

Parameter

```
ms    wait duration in milliseconds
```

6.2.2 Sleep

General Functions

Syntax

```
void Sleep(byte ctrl);

Sub Sleep(ctrl As Byte)
```

Description

Using this function the Atmel CPU is set in one of the 6 different sleep modes. The exact functionality is provided in the Atmel Mega Reference Manual in the chapter "Power Management and Sleep Modes". The value of ctrl is written into the bits *SM0* and *SM2*. The sleep enable bit (SE in **MCUCR**) is set and a assembler *sleep* instruction is executed.

Parameter

ctrl   Initialization (*SM0* to *SM2*)

Sleep Modes

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

6.3 Analog-Comparator

The Analog Comparator allows to compare two analog signals. The result of this comparison is returned as either "0" or "1".

6.3.1 AComp

AComp Functions **Example**

Syntax

```
void AComp(byte mode);

Sub AComp(mode As Byte);
```

Description

The Analog Comparator allows to compare two analog signals. The result of this comparison is returned as either "0" or "1". (Comparator Output). The negative input is **Mega32**: AIN1 (PortB.3), **Mega128**: AIN1 (PortE.3). The positive input can either be **Mega32**: AIN0 (PortB.2), **Mega128**: AIN0 (PortE.2) , or an internal reference voltage of 1,22V.

Parameter

mode   working mode

Mode Values:

0x00	external inputs (+)AIN0 and (-)AIN1 are used
0x40	external Input (-)AIN1and internal reference voltage are used
0x80	Analog-Comparator gets disabled

6.3.2   AComp Example

Example: Usage of Analog-Comparators

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

// The function AComp returns the value of the comparator.
// If the voltage on input PB2/PE2 is greater than the input PB3/PE3 the
// function AComp returns the value 1.
// Mode:
// 0x00  external inputs (+)AIN0 and (-)AIN1 are used
// 0x40  external input (-)AIN1 and the internal reference voltage are used
// 0x80  the Analog-Comparator is disabled
// In this example you can call AComp with parameter 0 (both inputs are used)
// or with 0x40 (internal reference voltage on (+) input, external Input PB3/PE3)


//----------------------------------------------------------------------
// main program
//
void main(void)
{
    while (true)
```

```
    {
        if (AComp(0x40)==1)          // Input (+) band gap reference 1,22V
        {
            Msg_WriteChar('1');      // Output: 1
        }
        else
        {
            Msg_WriteChar('0');      // Output: 0
        }
        // the comparator value is read all 500ms
        AbsDelay(500);
    }
}
```

6.4   Analog-Digital-Converter

The Micro Controller has an Analog Digital Converter with a resolution of 10 Bit. I. e. measured voltages can be displayed as integral numbers from 0 through 1023. Reference voltage for the lower limit is GND level (0V). The reference voltage for the upper limit can be selected at will.

- External Reference Voltage
- AVCC with capacitor on AREF
- Internal Reference Voltage 2.56V with capacitor on AREF

Analog Inputs ADC0 ... ADC7, ADC_BG, ADC_GND

For the ADC the Inputs ADC0 ... ADC7 (Port A.0 to A.7 with **Mega32**, Port F.0 to F.7 with **Mega128)**, an internal Band Gap (1.22V) or GND (0V) are available. ADC_BG and ADC_GND can be used for review of the ADC.

If x is a digital measuring value then the corresponding voltage value u is calculated as follows:

*u = x * Reference Voltage / 1024*

If the external reference voltage e. g. produced by a reference voltage IC is 4.096V, then the difference of one bit of the digitized measuring value corresponds to a voltage difference of 4mV, or:

*u = x * 0,004V*

➡ The result of an A/D conversion can be influenced, if any Port Bit (configured for output) on the same Port as the A/D channel, is changed during the measurement.

Differential Inputs

ADC22x10	Differential Inputs ADC2, ADC2, Gain 10	; Offset Measurement
ADC23x10	Differential Inputs ADC2, ADC3, Gain 10	
ADC22x200	Differential Inputs ADC2, ADC2, Gain 200	; Offset Measurement
ADC23x200	Differential Inputs ADC2, ADC3, Gain 200	
ADC20x1	Differential Inputs ADC2, ADC0, Gain 1	
ADC21x1	Differential Inputs ADC2, ADC1, Gain 1	
ADC22x1	Differential Inputs ADC2, ADC2, Gain 1	; Offset Measurement
ADC23x1	Differential Inputs ADC2, ADC3, Gain 1	
ADC24x1	Differential Inputs ADC2, ADC4, Gain 1	
ADC25x1	Differential Inputs ADC2, ADC5, Gain 1	

ADC2 is the negative input.

The ADC can also perform differential measurements. The result can either be positive or negative. The resolution during differential operation amounts to +/- 9 bit and is displayed in Two's Complement format. For differential operation an amplifier with gains of V: x1, x10, x200 is available. If x is a digital measuring value then the corresponding voltage value u is calculated as follows:

u = x * Reference Voltage / 512 / V

6.4.1   ADC_Disable

ADC Functions

Syntax

```
void ADC_Disable(void);

Sub ADC_Disable()
```

Description

This function disables to the A/D-Converter to reduce power consumption.

Parameter

```
None
```

6.4.2   ADC_Read

ADC Functions

Syntax

```
word ADC_Read(void);

Sub ADC_Read() As Word
```

Description

The function ADC_Read delivers the digitized measured value from one of the 8 ADC ports. The port number (0..7) has been given as a parameter in the call of ADC_Set(). The result is in the range from 0 to 1023 according to the 10bit resolution of the A/D-Converter. The analog inputs ADC0 to ADC7 can be measured against ground, or differentiation measurement with gain factor of 1/10/100 can be made.

Return Parameter

measured value at the ADC-Port

6.4.3   ADC_ReadInt

ADC Functions

Syntax

word ADC_ReadInt(**void**);

Sub ADC_ReadInt() **As Word**

Description

This function is used to read the measurement value after a successful ADC-Interrupt. The ADC-Interrupt gets triggered after the AD conversion is completed and a new measurement value is available. See ADC_SetInt and ADC_StartInt. The function ADC_Read delivers the digitized measured value from one of the 8 ADC ports. The port number (0..7) has been given as a parameter in the call of ADC_Set(). The result is in the range from 0 to 1023 according to the 10bit resolution of the A/D-Converter. The analog inputs ADC0 to ADC7 can be measured against ground, or differentiation measurement with gain factor of 1/10/100 can be made.

Return Parameter

measured value of ADC-Port

6.4.4   ADC_Set

ADC Functions

Syntax

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

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

Description

The function ADC_Set initializes the Analog-Digital converter. The reference voltage and the measurement channel number is selected and the A/D converter is prepared for usage. After the measurement the value is read with ADC_Read().

➡ The result of an A/D conversion can be influenced, if any Port Bit (configured for output) on the same Port as the A/D channel, is changed during the measurement.

Parameter

channel   port number (0..7)of ADC (Port A.0 to A.7 at Mega32, Port F.0 to F.7 at Mega128)
v_ref   reference voltage (see table)

Name	Value	Description
ADC_VREF_BG	0xC0	2,56V internal reference voltage
ADC_VREF_VCC	0x40	supply voltage (5V)
ADC_VREF_EXT	0x00	external reference voltage on PAD3

For the location of PAD3 see Jumper Application Board M32 or M128.

6.4.5 ADC_SetInt

ADC Functions

Syntax

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

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

Description

The function ADC_SetInt initializes the Analog-Digital converter for interrupt usage. The reference voltage and the measurement channel number is selected and the A/D converter is prepared for the measurement. An interrupt service routine must be defined. After successful interrupt the value can be read with ADC_ReadInt().

➡ The result of an A/D conversion can be influenced, if any Port Bit (configured for output) on the same Port as the A/D channel, is changed during the measurement.

Parameter

channel   port number (0..7)of ADC (Port A.0 to A.7 at Mega32, Port F.0 to F.7 at Mega128)
v_ref   reference voltage (see table)

Name	Value	Description
ADC_VREF_BG	0xC0	2,56V internal reference voltage
ADC_VREF_VCC	0x40	supply voltage (5V)
ADC_VREF_EXT	0x00	external reference voltage on PAD3

For the location of PAD3 see Jumper Application Board M32 or M128.

6.4.6 ADC_StartInt

ADC Functions

Syntax

void ADC_StartInt(**void**);

Sub ADC_StartInt()

Description

The measurement is started if the A/D converter has previously been initialized to interrupt service with a call to ADC_SetInt(). After the measurement is ready, the interrupt gets triggered.

Parameter

None

6.5 Clock

The internal software clock is clocked by the 10ms interrupt of Timer2. Time and date can be set and then continue to run independently. Leap years are taken into account. Depending on the Quartz inaccuracy the error is between 4-6 seconds per day. A correction factor in 10ms ticks can be applied, that is added every hour to the internal counter.

Example: If you have a deviation of 9.5 sec for 2 days, then you have to correct a deviation of 9.5 / (2 * 24) = 0.197 sec. This corresponds to a correction factor of 20, if the software clock goes in advance, or -20 else.

➡️ When Timer 2 off, or used for other purposes, the internal software clock is not functional.

6.5.1 Clock_GetVal

Clock Functions

Syntax

byte Clock_GetVal(**byte** indx);

Sub Clock_GetVal(indx **As Byte**) **As Byte**

Description

All Date and Time values of the internal software clock can be read.

➡️ The values of day and month are zero based, a one should be added when printing.

Parameter

<u>indx</u>   index of date or time parameter

#define	Index	Meaning
CLOCK_SEC	0	Second
CLOCK_MIN	1	Minute
CLOCK_HOUR	2	Hour
CLOCK_DAY	3	Day
CLOCK_MON	4	Month
CLOCK_YEAR	5	Year

6.5.2   Clock_SetDate

Clock Functions

Syntax

void Clock_SetDate(**byte** <u>day</u>, **byte** <u>mon</u>, **byte** <u>year</u>);

Sub Clock_SetDate(<u>day</u> **As Byte**, <u>mon</u> **As Byte**, <u>year</u> **As Byte**)

Description

Sets the date of the internal software clock.

➡ The values of day and month are zero based.

Parameter

<u>day</u>   Day
<u>mon</u>   Month
<u>year</u>   Year

6.5.3   Clock_SetTime

Clock Functions

Syntax

void Clock_SetTime(**byte** <u>hour</u>, **byte** <u>min</u>, **byte** <u>sec</u>, **char** <u>corr</u>);

Sub Clock_SetTime(<u>hour</u> **As Byte**, <u>min</u> **As Byte**, <u>sec</u> **As Byte**, <u>corr</u> **As Char**)

Description

Sets the time of the internal software clock. For a description of the correction factor refer to chapter Clock.

Parameter

hour   Hour
min   Minute
sec   Second
corr   Correction Factor

6.6   DCF 77

All DCF routines are realized in library "LCD_Lib.cc". For use of this function the library "DCF_Lib. cc" has to be tied into the project.

RTC with DCF 77 Time Synchronization

The DCF 77 Time Signal

The logical informations (time informations) are transmitted in addition to the normal frequency (carrier frequency of the transmitter, i. e. 77.5 kHz). This is performed by negative modulation of the signal (decrease of carrier amplitude to 25%). The start of the decrease lies at the respective beginning of the seconds 0 … 58 within a minute. In second 59 there is no decrease, so the following second mark can indicate the beginning of a minute and the receiver can be synchronized. The sign duration yields the logical value of the signs: 100 ms are "0", 200 ms are "1". Because of this there are 59 bits for informations available within one minute. From these the second marks 1 through 14 are used for operation informations which are not meant for DCF 77 users. The second marks 15 through 19 indicate the transmitter antenna, the time zone and will give notice of coming time changes.

From second 20 through 58 the time information for the respective following minute will be transmitted serially in from of BCD numbers, whereby in any case the least significant bit will be the start bit.

Bits	Meaning
20	Start bit (in any case "1")
21 - 27	Minute
28	Parity Minute
29 - 34	Hour
35	Parity Hour
36 - 41	Day of the Month
42 - 44	Weekday
45 - 49	Month
50 - 57	Year
58	Parity Date

This signifies that reception must be in progress for at least one full minute before time information can be provided. The information decoded during this minute is only secured by three parity bits. So two incorrectly received bits will already lead to a transmission error that can not be recognised in this way. For higher demands additional checking mechanisms can be used, such as plausibility check (is the received time within the admissible limits) or multiple reading of the DCF 77 time

information with data comparison. Another possibility would be to compare the DCF time with the current RTC time and only allow a specific deviation. This method does not work right after program start since the RTC has to be set first.

Description of the example program "DCF_RTC.cc"

The program DCF_RTC.cc represents a clock which is synchronized by use of DCF 77. Time and date are displayed on an LCD. Synchronization takes place after program start and then daily at a time determined in the program (Update_Hour, Update_Minute). There are two libraries used: DCF_Lib.cc and LCD_Lib.cc.

For the radio reception of the time signal a DCF 77 receiver is necessary. The output of the DCF receiver is connected to the input port (**Mega32**: PortD.7 - **M128**: PortF.0). At first the beginning of a time information has to be found. It will be synchronized onto the pulse gap (bit 59). Following the bit will be received in seconds time. There will be a parity check after the minute and hour information and also at the end of the transmission. The result of the parity check will be stored in DCF_ARRAY [6]. For transfer of the time information DCF_ARRAY[0..6] will be used. After reception of a valid time information the RTC will be set with this new time and will then run independently. RTC as well as DCF 77 decoding is controlled by a 10ms interrupt. This time base is derived from the quartz frequency of the Controller. DCF_Mode will control the completion of the DCF 77 time reception.

Table DCF Modes

DCF_Mode	Description
0	No DCF 77 operation
1	Find pulse
2	Synchronization on frame start
3	Decode and store data. Parity check

RTC (Real Time Clock)

The RTC is controlled by a 10ms interrupt and runs in the background independent of the user program. The display on the LCD is updated every second. The display format is in the first line: Hour : Minute : Second, in the second line: Date of Day : Month : Year.

LED1 flashes once per second.
After program start the RTC begins with the set time. The date is set to zero and thus indicates that no DCF time adjustment has yet taken place. After reception of the DCF time the RTC is updated with the current data. The RTC is not backed up by a battery, i. e. the clock time will not be updated if there is no power applied to the Controller.

6.6.1   DCF_FRAME

DCF Functions

Syntax

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

```
Sub DCF_FRAME()
```

Description

Set DCF_Mode to 3 ("data decode and save, parity check").

Parameter

None

6.6.2 DCF_INIT

DCF Functions

Syntax

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

```
Sub DCF_INIT()
```

Description

DCF_INIT initializes DCF usage. The input of the DCF signal is adjusted. DCF_Mode is set to 0.

Parameter

None

6.6.3 DCF_PULS

DCF Functions

Syntax

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

```
Sub DCF_PULS()
```

Description

Set DCF_Mode to 1 ("look for pulse").

Parameter

None

6.6.4 DCF_START

DCF Functions

___

Syntax

void DCF_START(**void**);

Sub DCF_START()

Description

DCF_START initializes all variables and sets [DCF_Mode](#) to 1. From now on DCF time recording is working automatically.

Parameter

None

6.6.5 DCF_SYNC

DCF Functions

___

Syntax

void DCF_SYNC(**void**);

Sub DCF_SYNC()

Description

Set [DCF_Mode](#) to 2 ("synchronize for frame beginning").

Parameter

None

6.7 Debug

The Debug Message Functions allow to send formatted text to the output window of the IDE. These functions are interrupt driven with a buffer of up to 128 Byte. I. e. 128 Byte can be transferred through the debug interface without the Mega 32 or Mega 128 Module having to wait for completion of the output. The transmission of the individual characters takes place in the background. If it is tried to send more than 128 Byte then the Mega RISC CPU will have to wait until all characters not fitting into the buffer anymore have been transferred.

6.7.1 Msg_WriteChar

Debug Message Functions

Syntax

void Msg_WriteChar(**char** c);

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

Description

One character is written to the output window. A C/R (Carriage Return - Value:13) generates a jump to the next line (linefeed).

Parameter

c   output character

6.7.2 Msg_WriteFloat

Debug Message Functions

Syntax

void Msg_WriteFloat(**float** val);

Sub Msg_WriteFloat(val **As Single**)

Description

The passed floating point number is displayed with a preceding decimal sign.

Parameter

val   float value

6.7.3   Msg_WriteHex

Debug Message Functions

Syntax

void Msg_WriteHex(**word** <u>val</u>);

Sub Msg_WriteHex(<u>val</u> **As Word**)

Description

The 16bit value is displayed in the output window. The Output is formatted as a hexadecimal value with 4 digits. Leading zeros are displayed.

Parameter

<u>val</u>   16bit integer value

6.7.4   Msg_WriteInt

Debug Message Functions

Syntax

void Msg_WriteInt(**int** <u>val</u>);

Sub Msg_WriteInt(<u>val</u> **As Integer**)

Description

The passed 16bit value is display in the output window. Negative values are displayed with a preceding minus sign.

Parameter

<u>val</u>   16bit integer value

6.7.5   Msg_WriteText

Debug Message Functions

Syntax

void Msg_WriteText(**char** <u>text</u>[]);

Sub Msg_WriteText(**ByRef** <u>text</u> **As Char**)

Description

All characters of a character array up to the terminating null are sent to the output window.

Parameter

text   pointer to char array

6.7.6   Msg_WriteWord

Debug Message Functions

Syntax

void Msg_WriteWord(**word** val);

Sub Msg_WriteWord(val **As Word**)

Description

The parameter val is written to the output windows as an unsigned decimal number.

Parameter

val   16bit unsigned integer value

6.8   Direct Access

The Direct Access functions allow direct access to all registers of the Atmel processor. The Register numbers of the Atmel MEGA32 and Mega128 processors can be found in the Reference manual in the chapter "**Register Summary**".

➡ **Caution!** A careless reading or writing access to a register can strongly affect the functionality of all library functions. Only someone who knows what he does, should use the Direct Access functions!

6.8.1   DirAcc_Read

Direct Access Functions

Syntax

byte DirAcc_Read(**byte** register);

Sub DirAcc_Read(register **As Byte**) **As Byte**

Description

A Byte is read from a Register of the Atmel CPU.

Parameter

<u>register</u> Register number (refer to chapter "**Register Summary**" in the Atmel Reference Manual)

Return Parameter

Value of Register

6.8.2 DirAcc_Write

Direct Access Functions

Syntax

void DirAcc_Write(**byte** <u>register</u>, **byte** <u>val</u>);

Sub DirAcc_Write(<u>register</u> **As Byte**, <u>val</u> **As Byte**)

Description

A Byte value is written into a Register of the Atmel CPU.

Parameter

<u>register</u> Register number (refer to chapter "**Register Summary**" in the Atmel Reference Manual)
<u>val</u>   Byte value

6.9   EEPROM

The C-Control Pro Modules integrate **M32**:1kB resp. **M128**:4kB EEPROM. These library functions allow access to the EEPROM of the Interpreter. 32 Bytes of the EEPROM area are used for internal tasks and are thus not accessible.

6.9.1 EEPROM_Read

EEPROM Functions

Syntax

byte EEPROM_Read(**word** <u>pos</u>);

Sub EEPROM_Read(<u>pos</u> **As Word**) **As Byte**

Description

Reads one byte out of the EEPROM at position pos. The first 32 byte are reserved for the OS of C-Control Pro. Therefore a pos value of 0 and higher accesses the EEPROM memory at position 32 and upwards.

Parameter

pos   byte position in EEPROM

Return Parameter

EEPROM value

6.9.2 EEPROM_ReadWord

EEPROM Functions

Syntax

```
word EEPROM_ReadWord(word pos);
```

```
Sub EEPROM_ReadWord(pos As Word) As Word
```

Description

Reads one word out of the EEPROM at position pos. The first 32 byte are reserved for the OS of C-Control Pro. Therefore a pos value of 0 and higher accesses the EEPROM memory at position 32 and upwards. The value of pos describes a byte position in the EEPROM. This should be taken care of when using word or floating point accesses.

Parameter

pos   byte position in EEPROM

Return Parameter

EEPROM value

6.9.3 EEPROM_ReadFloat

EEPROM Functions

Syntax

```
float EEPROM_ReadFloat(word pos);
```

```
Sub EEPROM_ReadFloat(pos As Word) As Single
```

Description

Reads a floating point value out of the EEPROM at position <u>pos</u>. The first 32 byte are reserved for the OS of C-Control Pro. Therefore a pos value of 0 and higher accesses the EEPROM memory at position 32 and upwards. The value of <u>pos</u> describes a byte position in the EEPROM. This should be taken care of when using word or floating point accesses.

Parameter

<u>pos</u>   byte position in EEPROM

Return Parameter

EEPROM value

6.9.4   EEPROM_Write

EEPROM Functions

Syntax

void EEPROM_Write(**word** <u>pos</u>,**byte** <u>val</u>);

Sub EEPROM_Write(<u>pos</u> **As Word**,<u>val</u> **As Byte**)

Description

Writes one byte into the EEPROM at position <u>pos</u>. The first 32 byte are reserved for the OS of C-Control Pro. Therefore a pos value of 0 and higher accesses the EEPROM memory at position 32 and upwards.

Parameter

<u>pos</u>   byte position in EEPROM
<u>val</u>   new EEPROM value

6.9.5   EEPROM_WriteWord

EEPROM Functions

Syntax

void EEPROM_WriteWord(**word** <u>pos</u>,**word** <u>val</u>);

Sub EEPROM_WriteWord(<u>pos</u> **As Word**,<u>val</u> **As Word**)

Description

Writes one word into the EEPROM at position <u>pos</u>. The first 32 byte are reserved for the OS of C-Control Pro. Therefore a pos value of 0 and higher accesses the EEPROM memory at position 32 and upwards. The value of <u>pos</u> describes a byte position in the EEPROM. This should be taken care of when using word

or floating point accesses.

Parameter
pos   byte position in EEPROM
val   new EEPROM value

6.9.6 EEPROM_WriteFloat

EEPROM Functions

Syntax

void EEPROM_WriteFloat(**word** pos,**float** val);

Sub EEPROM_WriteFloat(pos **As Word**,val **As Single**)

Description

Writes a floating point value into the EEPROM at position pos. The first 32 byte are reserved for the OS of C-Control Pro. Therefore a pos value of 0 and higher accesses the EEPROM memory at position 32 and upwards. The value of pos describes a byte position in the EEPROM. This should be taken care of when using word or floating point accesses.

Parameter

pos   byte position in EEPROM
val   new EEPROM value

6.10 I2C

The Controller provides an I2C Logic which allows effective communication. The Controller operates as an I2C Master (single master system). A slave operating mode is possible but not yet implemented in the current version.

6.10.1 I2C_Init

I2C Functions   **Example**

Syntax

void I2C_Init(**byte** I2C_BR);

Sub I2C_Init(I2C_BR **As Byte**)

Description

This function initializes the I2C interface.

Parameter

I2C_BR   describes the baud rate. The following values are predefined:

```
I2C_100kHz
I2C_400kHz
```

6.10.2   I2C_Read_ACK

I2C Functions

Syntax

byte I2C_Read_ACK(**void**);

Sub I2C_Read_ACK() **As Byte**

Description

This function receives a byte and acknowledges with ACK. Afterwards the status of the interface can be returned with I2C_Status().

Return Parameter

value read from the I2C bus

6.10.3   I2C_Read_NACK

I2C Functions   **Example**

Syntax

byte I2C_Read_NACK(**void**);

Sub I2C_Read_NACK() **As Byte**

Description

This function receives a byte and acknowledges with NACK. Afterwards the status of the interface can be returned with I2C_Status().

Return Parameter

value read from the I2C bus

6.10.4 I2C_Start

I2C Functions   **Example**

Syntax

```
void I2C_Start(void);

Sub I2C_Start()
```

Description

This function introduces communication with a starting sequence. Afterwards the status of the interface can be returned with I2C_Status().

Parameter

None

6.10.5 I2C_Status

I2C Functions

Syntax

```
byte I2C_Status(void);

Sub I2C_Status()
```

Description

With I2C_Status the status of the I2C interface can be accessed. For the meaning of the return value please look inside the I2C status code table.

Return Parameter

current I2C Status

6.10.6 I2C_Stop

I2C Functions   **Example**

Syntax

```
void I2C_Stop(void);

Sub I2C_Stop()
```

Description

This function ceases the I2C communication with a stop sequence. Afterwards the status of the interface can be returned with I2C_Status().

Parameter

None

6.10.7 I2C_Write

I2C Functions   **Example**

Syntax

```
void I2C_Write(byte data);

Sub I2C_Write(data As Byte)
```

Description

I2C_Write() sends a byte to the I2C bus. Afterwards the status of the interface can be returned with I2C_Status().

Parameter

data data byte

6.10.8 I2C Status Table

Table: **Status Codes Master Transmitter Mode**

Status Code	Description
0x08	a START sequence has been sent
0x10	a "repeated" START sequence has been sent
0x18	SLA+W has been sent, ACK has been received
0x20	SLA+W has been sent, NACK has been received
0x28	Data byte has been sent, ACK has been received
0x30	Data byte has been sent, NACK has been received
0x38	conflict with SLA+W or data bytes

Table: **Status Codes Master Receiver Mode**

Status Code	Description
0x08	a START sequence has been sent
0x10	a "repeated" START sequence has been sent
0x38	conflict with SLA+R or data bytes
0x40	SLA+R has been sent, ACK has been received
0x48	SLA+R has been sent, NACK has been received
0x50	Data byte has been sent, ACK has been received
0x58	Data byte has been sent, NACK has been received

6.10.9 I2C Example

Example: read EEPROM 24C64 and write without I2C_Status check

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

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

    address=0x20;
    data=0x42;

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

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

6.11 Interrupt

The Controller provides a multitude of interrupts. Some of them are used for system functions and are thus not available to the user. The following interrupts can be utilized by the user.

Table: Interrupts

Interrupt Name	Description
INT_0	external Interrupt0
INT_1	external Interrupt1
INT_2	external Interrupt2
INT_3	external Interrupt3 (only Mega128)
INT_4	external Interrupt4 (only Mega128)
INT_5	external Interrupt5 (only Mega128)
INT_6	external Interrupt6 (only Mega128)
INT_7	external Interrupt7 (only Mega128)
INT_TIM1CAPT	Timer1 Capture
INT_TIM1CMPA	Timer1 CompareA
INT_TIM1CMPB	Timer1 CompareB
INT_TIM1OVF	Timer1 Overflow
INT_TIM0COMP	Timer0 Compare
INT_TIM0OVF	Timer0 Overflow
INT_ANA_COMP	Analog Comparator
INT_ADC	ADC
INT_TIM2COMP	Timer2 Compare
INT_TIM2OVF	Timer2 Overflow
INT_TIM3CAPT	Timer3 Capture (only Mega128)
INT_TIM3CMPA	Timer3 CompareA (only Mega128)
INT_TIM3CMPB	Timer3 CompareB (only Mega128)
INT_TIM3CMPC	Timer3 CompareC (only Mega128)
INT_TIM3OVF	Timer3 Overflow (only Mega128)

The corresponding interrupt has to receive the corresponding instructions in an Interrupt Service Routine (ISR) and also the interrupt has to be enabled. See **Example**. During execution of the interrupt routine the Multi Threading is suspended.

➡ A signal on INT_0 can interfere with the Autostart Behaviour when the C-Control Pro Module is switched on. According to the pin assignment of M32 and M128 INT_0 shares the same pin with SW1. If SW1 is pressed during power up of the Module then the Bootloader Mode will be activated and the program will not be automatically started.

6.11.1 Ext_IntEnable

Interrupt Functions

Syntax

void Ext_IntEnable(**byte** IRQ,**byte** Mode);

Sub Ext_IntEnable(IRQ **As Byte**,Mode **As Byte**)

Description

This function enables the external Interrupt IRQ. The Mode parameter defines when to trigger the interrupt. Caution: A signal on INT_0 at power up time can lead to Autostart problems.

➡ The IRQ parameter is defined between 0 and 2 for the **Mega32** and between 0 and 7 for the **Mega128**. Please do not mistake with the irqnr parameter of Irq_SetVect().

➡ The IRQ2 of Mega32 can only work edge triggered. See the different Mode parameter.

Parameter

IRQ   number of the interrupt to be enabled
Mode parameter:

0:   a low level triggers the interrupt
1:   every changing edge triggers the interrupt
2:   a falling edge triggers the interrupt
3:   a rising edge triggers the interrupt

Mode   parameter for **Mega32** and IRQ2:

0:   a falling edge triggers the interrupt
1:   a rising edge triggers the interrupt

6.11.2 Ext_IntDisable

Interrupt Functions

Syntax

void Ext_IntDisable(**byte** IRQ);

```
Sub Ext_IntDisable(IRQ As Byte)
```

Description

The external Interrupt <u>IRQ</u> gets disabled.

Parameter

<u>IRQ</u> number of the interrupt to disable

6.11.3 Irq_GetCount

Interrupt Functions   **Example**

Syntax

```
byte Irq_GetCount(byte irqnr);

Sub Irq_GetCount(irqnr As Byte) As Byte
```

Description

 Acknowledges the interrupt. If the function is not called at the end of a interrupt service routine, the interrupt service routine gets called continuously.

Parameter

<u>irqnr</u> specifies the interrupt type (see <u>table</u>)

Return Parameter

The return value expresses how often a interrupt got triggered until the function Irq_GetCount() has been called. A value greater 1 shows that the interrupts
are triggered more rapidly than the interrupt service routine is processed.

6.11.4 Irq_SetVect

Interrupt Functions   **Example**

Syntax

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

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

Description

Defines an interrupt service routine for a specified interrupt. At the end of the interrupt service routine the function <u>Irq_GetCount</u>() has to be called, otherwise the interrupt service routine gets called continuously.

Parameter

<u>irqnr</u> specifies the interrupt type (see <u>table</u>)
<u>vect</u> is the name of the interrupt function to be called

Remark

The floating point data type seems not suitable for an interrupt vector, but a 4 byte data type is needed to specify a function in the Mega128 environment.

6.11.5 IRQ Example

Example: Usage of Interrupt Routines

```
// normally Timer 2 is called every 10ms. In this example the variable
// cnt gets increased by one every 10ms

int cnt;

void ISR(void)
{
    int irqcnt;

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

void main(void)
{
    cnt=0;

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

6.12 Keyboard

One part of these keyboard routines is implemented in the Interpreter, another can be called up after appending library "LCD_Lib.cc". Since the functions in
 "LCD_Lib.cc" are realized through Bytecode they are slower when executed. Library functions however have the advantage that they can be taken from the project by omitting the library in case they are not needed. Direct Interpreter functions are always present, will however take up flash memory.

6.12.1 Key_Init

Keyboard Functions   (Library "*Key_Lib.cc*")

Syntax

```
void Key_Init(void);

Sub Key_Init()
```

Description

The global keymap array gets initialized with the ASCII values of the keyboard.

Parameter

None

6.12.2 Key_Scan

Keyboard Functions

Syntax

```
word Key_Scan(void);

Sub Key_Scan() As Word
```

Description

Key_Scan scans sequentially the input pins of the connected keyboard and returns the result as a bit field with 16 bits. Bits that are set represent keys that have been pressed during the scan.

Return Parameter

16 bits that represent the input lines of the keyboard

6.12.3 Key_TranslateKey

Keyboard Functions   (Library "*Key_Lib.cc*")

Syntax

```
char Key_TranslateKey(word keys);

Sub Key_TranslateKey(keys As Word) As Char
```

Description

This help function looks for the first "1" in the bit field, and returns the ASCII value of the corresponding key.

Parameter

keys   bit field value that has been retuned from <u>Key_Scan</u>()

Return Parameter

ASCII value of recognized keys
-1 if no key is pressed

6.13   LCD

A part of these routines is implemented in the Interpreter, another part can be called up by appending library "LCD_Lib.cc". Since the functions in "LCD_Lib.cc" are realized through Bytecode they are slower when executed. Library functions however have the advantage that they can be taken from the project by omitting the library in case they are not needed. Direct Interpreter functions are always present, will however take up flash memory.

6.13.1   LCD_ClearLCD

LCD Functions   (Library "*LCD_Lib.cc*")

Syntax

void LCD_ClearLCD(**void**);

Sub LCD_ClearLCD()

Description

Clears the display and enables the Cursor.

Parameter

None

6.13.2   LCD_CursorOff

LCD Functions   (Library "*LCD_Lib.cc*")

Syntax

void LCD_CursorOff(**void**);

Sub LCD_CursorOff()

Description

Turns the cursor off on the display.

Parameter

None

6.13.3 LCD_CursorOn

LCD Functions   (Library "*LCD_Lib.cc*")

Syntax

void LCD_CursorOn(**void**);

Sub LCD_CursorOn()

Description

Turns the cursor in the display on.

Parameter

None

6.13.4 LCD_CursorPos

LCD Functions   (Library "*LCD_Lib.cc*")

Syntax

void LCD_CursorPos(**byte** pos);

Sub LCD_CursorPos(pos **As Byte**)

Description

Moves the cursor to position pos.

Parameter

pos cursorposition

Value of pos	**Position on Display**
0x00-0x07	0-7 on 1st line

0x40-0x47	0-7 on 2nd line

The following table is valid for displays with more than 2 lines and up to 32 chars per line:

Value of <u>pos</u>	**Position on Display**
0x00-0x1f	0-31 on line 1
0x40-0x5f	0-31 on line 2
0x20-0x3f	0-31 on line 3
0x60-0x6f	0-31 on line 4

6.13.5 LCD_Init

LCD Functions   (Library "*LCD_Lib.cc*")

Syntax

void LCD_Init(**void**);

Sub LCD_Init()

Description

High level intialization of the LCD display. Calls LCD_InitDisplay() as first.

Parameter

None

6.13.6 LCD_Locate

LCD Functions

Syntax

void LCD_Locate(**int** <u>row</u>, **int** <u>column</u>);

Sub LCD_Locate(<u>row</u> **As Integer**, <u>column</u> **As Integer**)

Description

Sets the cursor of the LCD display to given row and column.

Parameter

<u>row</u>
<u>column</u>

6.13.7 LCD_SubInit

LCD Functions

Syntax

void LCD_SubInit(**void**);

Sub LCD_SubInit()

Description

Initializes the display ports on assembler level. Must be called before all other LCD output functions. This function will be used as first command from LCD_Init().

Parameter

None

6.13.8 LCD_TestBusy

LCD Functions

Syntax

void LCD_TestBusy(**void**);

Sub LCD_TestBusy()

Description

This function waits for a non-busy of the display controller. If the controller is accessed in his busy period the output data will be corrupted.

Parameter

None

6.13.9 LCD_WriteChar

LCD Functions   (Library "*LCD_Lib.cc*")

Syntax

void LCD_WriteChar(**char** c̲);

Sub LCD_WriteChar(<u>c</u> **As Char**)

Description

Displays one character at the cursor position on the LCD display.

Parameter

<u>c</u>   ASCII value of output character

6.13.10 LCD_WriteCTRRegister

LCD Functions   (Library "*LCD_Lib.cc*")

Syntax

void LCD_WriteCTRRegister(**byte** <u>cmd</u>);

Sub LCD_WriteCTRRegister(<u>cmd</u> **As Byte**)

Description

Sends a command to the display controller.

Parameter

<u>cmd</u>   byte command

6.13.11 LCD_WriteDataRegister

LCD Functions   (Library "*LCD_Lib.cc*")

Syntax

void LCD_WriteDataRegister(**char** <u>x</u>);

Sub LCD_WriteDataRegister(<u>x</u> **As Char**)

Description

Sends a data byte to the display controller.

Parameter

<u>x</u>   data byte

6.13.12 LCD_WriteFloat

LCD Functions

Syntax

void LCD_WriteFloat(**float** <u>value</u>, **byte** <u>length</u>);

Sub LCD_WriteFloat(<u>value</u> **As Single**, <u>length</u> **As Byte**)

Description

Writes a floating point value with given length to LCD display.

Parameter

<u>value</u>   floating point value
<u>length</u>   output length

6.13.13 LCD_WriteRegister

LCD Functions

Syntax

void LCD_WriteRegister(**byte** <u>y</u>,**byte** <u>x</u>);

Sub LCD_WriteRegister(<u>y</u> **As Byte**,<u>x</u> **As Byte**)

Description

LCD_WriteRegister divides the data byte <u>y</u> in 2 nibbles (4bit values) and
sends the nibbles to the display controller.

<u>y</u> data byte
<u>x</u> command nibble

6.13.14 LCD_WriteText

LCD Functions   (Library "*LCD_Lib.cc*")

Syntax

void LCD_WriteText(**char** <u>text[]</u>);

Sub LCD_WriteText(**ByRef** <u>Text</u> **As Char**)

Description

All characters of the char array up to the terminating zero are displayed.

Parameter

text   char array

6.13.15 LCD_WriteWord

LCD Functions

Syntax

void LCD_WriteWord(**word** value, **byte** length);

Sub LCD_WriteWord(value **As Word**, length **As Byte**)

Description

Writes an unsigned integer (word) with given length to the LCD display. If the resulting LCD output is smaller than the given length, the output filled with zeros "0" at the beginning.

Parameter

value   word value
length output length

6.14   Math

In the following the mathematical functions are listed which the C-Control Pro 128 is able to master with single floating point accuracy (32 bit). These functions are not contained in the C-Control Pro 32 since it would then not offer enough memory for user programs.

6.14.1   acos

Mathematical Functions

Syntax

float acos(**float** val);

Sub acos(val **As Single**) **As Single**

Description

The mathematical arc cosine (inverse cosine) is calculated.

Parameter

<u>val</u> input value between -1 and 1

Return Parameter

arc cosine of the input value in the range [0..Pi], expressed in radians

6.14.2 asin

Mathematical Functions

Syntax

```
float asin(float val);

Sub asin(val As Single) As Single
```

Description

The mathematical arc sine (inverse sine) is calculated.

Parameter

<u>val</u> input value between -1 and 1

Return Parameter

arc sine of the input value in the range [-Pi/2..Pi/2], expressed in radians

6.14.3 atan

Mathematical Functions

Syntax

```
float atan(float val);

Sub atan(val As Single) As Single
```

Description

The mathematical arc tangent (inverse tangent) is calculated.

Parameter

<u>val</u> input value

Return Parameter

arc tangent of the input value in the range [-Pi/2..Pi/2], expressed in radians

6.14.4 ceil

Mathematical Functions

Syntax

float ceil(**float** val);

Sub ceil(<u>val</u> **As Single**) **As Single**

Description

The largest integer value of the floating point number x is calculated.

Parameter

<u>val</u> input value

Return Parameter

result

6.14.5 cos

Mathematical Functions

Syntax

float cos(**float** val);

Sub cos(<u>val</u> **As Single**) **As Single**

Description

The mathematical cosine is calculated.

Parameter

<u>val</u> input angle expressed in radians

Return Parameter

cosine of the input value between -1 and 1

6.14.6 exp

Mathematical Functions

Syntax

```
float exp(float val);

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

Description

The exponential function e ^ val is calculated.

Parameter

val exponent

Return Parameter

result

6.14.7 fabs

Mathematical Functions

Syntax

```
float fabs(float val);

Sub fabs(val As Single) As Single
```

Description

The absolute value of the floating point number val is calculated.

Parameter

val input value

Return Parameter

result

6.14.8 floor

Mathematical Functions

Syntax

```
float floor(float val);
```

```
Sub floor(val As Single) As Single
```

Description

The smallest integer value of the floating point number x is calculated.

Parameter

val   input value

Return Parameter

result

6.14.9 ldexp

Mathematical Functions

Syntax

```
float ldexp(float val,int expn);
```

```
Sub ldexp(val As Single,expn As Integer) As Single
```

Description

The function val * 2 ^ expn is calculated (also used as internal help function for other mathematical functions).

Parameter

val   multiplier
expn   exponent

Return Parameter

result

6.14.10 ln

Mathematical Functions

Syntax

```
float ln(float val);
```

```
Sub ln(val As Single) As Single
```

Description

The natural logarithm is calculated.

Parameter

<u>val</u> input value

Return Parameter

result

6.14.11 log

Mathematical Functions

Syntax

float log(**float** val);

Sub log(<u>val</u> **As Single**) **As Single**

Description

The logarithm base 10 is calculated.

Parameter

<u>val</u> input value

Return Parameter

result

6.14.12 pow

Mathematical Functions

Syntax

float pow(**float** <u>x</u>,**float** <u>y</u>);

Sub pow(<u>x</u> **As Single**,<u>y</u> **As Single**) **As Single**

Description

The power function <u>x</u> ^ <u>y</u> is calculated.

Parameter

<u>x</u> base
<u>y</u> exponent

Return Parameter

result

6.14.13 round

Mathematical Functions

Syntax

```
float round(float val);

Sub round(val As Single) As Single
```

Description

Rounding function. The floating point value is rounded up or down to a number without decimal places.

Parameter

val   input value

Return Parameter

result of the function

6.14.14 sin

Mathematical Functions

Syntax

```
float sin(float val);

Sub sin(val As Single) As Single
```

Description

The mathematical sine is calculated.

Parameter

val   input angle expressed in radians

Return Parameter

sine of the input value between -1 and 1

6.14.15 sqrt

Mathematical Functions

Syntax

float sqrt(**float** val);

Sub sqrt(<u>val</u> **As Single**) **As Single**

Description

The square root of a positive floating point number is calculated.

Parameter

<u>val</u> input value

Return Parameter

result

6.14.16 tan

Mathematical Functions

Syntax

float tan(**float** val);

Sub tan(<u>val</u> **As Single**) **As Single**

Description

The mathematical tangent is calculated.

Parameter

<u>val</u> input angle expressed in radians

Return Parameter

tangent of the input value

6.15   OneWire

1-Wire or One-Wire is a serial interface that needs only one wire for signaling and power. The data is transferred asynchronously (without clock signal) in groups of 64 bit. Data can either be sent or received, but not at the same time (half-duplex).

The special about 1-Wire devices is the parasitically power supply, that is made over the signal wire:

When there is no communication, the signal wire has a +5V level and charges a capacitor. During low-pulse communication the slave device is powered from his capacitor. Dependent on the charge of the capacitor, low-time gaps up to 960 µs can be bridged.

6.15.1 Onewire_Read

1-Wire Functions

Syntax

byte Onewire_Read(**void**);

Sub Onewire_Read() **As Byte**

Description

A Byte is read from the One-Wire Bus.

Return Parameter

value read from One-Wire Bus

6.15.2 Onewire_Reset

1-Wire Functions

Syntax

void Onewire_Reset(**byte** portbit);

Sub Onewire_Reset(portbit **As Byte**)

Description

A reset is made on the One-Wire Bus. The port bit number for the One-Wire Bus communication is specified.

Parameter

portbit   port bit number (see table)

Portbits Table

Definition	Portbit
PortA.0	0
…	…
PortA.7	7

PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
from here only Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

6.15.3 Onewire_Write

1-Wire Functions

Syntax

```
void Onewire_Write(byte data);

Sub Onewire_Write(data As Byte)
```

Description

A byte is written to the One-Wire Bus.

Parameter

data   data byte

6.15.4 Onewire Example

<u>CompactC</u>

```
// Sample Code to read DS18S20 temperature sensor from Dallas Maxim
void main(void)
{
    char text[40];
    int ret, i;
    byte rom_code[8];
    byte scratch_pad[9];

    ret= OneWire_Reset(7); // PortA.7
```

```
if(ret == 0)
{
    text= "no device found";
    Msg_WriteText(text);
    goto end;
}

OneWire_Write(0xcc); // skip ROM cmd
OneWire_Write(0x44); // start temperature measure cmd

AbsDelay(3000);

OneWire_Reset(7);    // PortA.7
OneWire_Write(0xcc); // skip ROM cmd
OneWire_Write(0xbe); // read scratch_pad cmd
for(i=0;i<9;i++)     // read whole scratchpad
{
    scratch_pad[i]= OneWire_Read();
    Msg_WriteHex(scratch_pad[i]);
}
Msg_WriteChar('\r');

text= "Temperature: ";
Msg_WriteText(text);

temp= scratch_pad[1]*256 + scratch_pad[0];
Msg_WriteFloat(temp* 0.5);
Msg_WriteChar('C');
Msg_WriteChar('\r');

    end:
}
```

BASIC

```
' Sample Code to read DS18S20 temperature sensor from Dallas Maxim
Dim Text(40) As Char
Dim ret,i As Integer
Dim temp As Integer
Dim rom_code(8) As Byte
Dim scratch_pad(9) As Byte

Sub main()

    ret = OneWire_Reset(7) ' PortA.7

    If ret = 0 Then
        Text= "no device found"
        Msg_WriteText(Text)
        GoTo Ende
    End If
```

```
OneWire_Write(0xcc)    ' skip ROM cmd
OneWire_Write(0x44)    ' start temperature measure cmd

AbsDelay(3000)

OneWire_Reset(7)       ' PortA.7
OneWire_Write(0xcc)    ' skip ROM cmd
OneWire_Write(0xbe)    ' read scratch_pad cmd

For i = 0 To 9         ' read whole scratchpad
    scratch_pad(i)= OneWire_Read()
    Msg_WriteHex(scratch_pad(i))
Next
Msg_WriteChar(13)

Text = "Temperature: "
Msg_WriteText(Text)

temp = scratch_pad(1) * 256 + scratch_pad(0)
Msg_WriteFloat(temp * 0.5)
Msg_WriteChar(99)
Msg_WriteChar(13)

    Lab Ende
End Sub
```

6.16 Port

The Atmel Mega 32 provides 4 input/output ports at 8 bits each. The Atmel Mega 128 provides 6 input/output ports at 8 bits each and one input/output port at 5 bits. Each bit of the individual ports can be configured as input or output. Since however the number of pins in the Mega 32 Risc CPU is limited, additional functions are assigned to individual ports. A pin assignment table for M32 and M128 can be found in the documentation.

➡ It is important to study the pin assignment prior to programming since important functions of the program design (e. g. the USB Interface of the Application Board) are assigned to specific ports. If these ports are programmed differently or the corresponding jumpers on the Application Board are no longer set it may happen that the design interface is no longer able to transfer programs to the C-Control Pro.

➡ The direction of data flow (input/output) can be determined with function Port_DataDir or Port_DataDirBit. If a pin is configured as input then this pin can either be operated high resistive ("floating") or with an internal pull-up resistor. If with Port_Write or Port_WriteBit a "1" is written to an input then the pull-up resistor (Reference Level VCC) is activated and the input is defined.

6.16.1 Port_DataDir

Port Functions **Example**

Syntax

void Port_DataDir(**byte** <u>port</u>,**byte** <u>val</u>);

Sub Port_DataDir(<u>port</u> **As Byte**,<u>val</u> **As Byte**)

Description

The function Port_DataDir configures the port for input or output direction. Is a bit set, then the Pin corresponding to the bit position is switched to output. Example: Is <u>port</u> = PortB and <u>val</u> = 0x02, then PortB.1 is configured for output, all other ports on PortB are set to input (see Pin Assignment of M32 and M128).

Parameter

<u>port</u> port number (see table)
<u>val</u> output byte

port number table

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

6.16.2 Port_DataDirBit

Port Functions

Syntax

void Port_DataDirBit(**byte** <u>portbit</u>,**byte** <u>val</u>);

Sub Port_DataDirBit(<u>portbit</u> **As Byte**,<u>val</u> **As Byte**)

Description

The function Port_DataDirBit configures one bit (Pin) of a port for input or output direction. Is a bit set, then the Pin corresponding to the bit position is switched to output. Example: Is <u>portbit</u> = 10 and <u>val</u> = 0, then PortB.2 is configured for input. All other ports on PortB stay the same (see Pin Assignment of M32

and [M128](#)).

➡️ Port Bit access is always significant slower than the normal Port access that transfers 8 Bit. If the desired values of all Port Bits are known, 8-Bit Port access is always preferable.

Parameter

portbit   port bit number (see table)
val   0=Input, 1= Output

Portbits Table

Definition	Portbit
PortA.0	0
…	…
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
from here only Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

6.16.3 Port_Read

Port Functions

Syntax

```
byte Port_Read(byte port);

Sub Port_Read(port As Byte) As Byte
```

Description

Reads a byte from the specified port. Only the Pins of port that are configured for input return a valid value on their bit position (see Pin Assignment of M32 and M128).

Parameter

port   port number (see table)

Return Parameter

port byte value

port number table

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

6.16.4 Port_ReadBit

Port Functions

Syntax

byte Port_ReadBit(**byte** port);

Sub Port_ReadBit(port **As Byte**) **As Byte**

Description

The function Port_ReadBit reads the value of a Pin that is configured for input. (See Pin Assignment of M32 and M128).

➡ Port Bit access is always significant slower than the normal Port access that transfers 8 Bit. If the desired values of all Port Bits are known, 8-Bit Port access is always preferable.

Parameter

portbit   bit number of port (see table)

Return Parameter

bit value (0 or 1)

Portbits Table

Definition	Portbit
PortA.0	0
…	…
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
from here only Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

6.16.5 Port_Toggle

Port Functions

Syntax

```
void Port_Toggle(byte port);

Sub Port_Toggle(port As Byte)
```

Description

Inverts all Bits on the specified port. Only the Pins of port that are configured for output will show their value as port output on their bit position (see Pin Assignment of M32 and M128). Is a Pin configured as input, this will set an internal pull-up resistor on (bit = 1) or off (bit = 0). See Pin Assignment of M32 and M128.

Parameter

port   port number (see table)

port number table

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

6.16.6 Port_ToggleBit

Port Functions

Syntax

void Port_ToggleBit(**byte** portbit);

Sub Port_ToggleBit(portbit **As Byte**)

Description

The function Port_WriteBit inverts the value of a Pin that is configured for output. Is a Pin configured as input, this will set an internal pull-up resistor on (bit = 1) or off (bit = 0). See Pin Assignment of M32 and M128.

➡ Port Bit access is always significant slower than the normal Port access that transfers 8 Bit. If the desired values of all Port Bits are known, 8-Bit Port access is always preferable.

Parameter

portbit   bit number of port (see table)

Portbits Table

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24

...	...
PortD.7	31
from here only Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...
PortF.7	47
PortG.0	48
...	...
PortG.4	52

6.16.7 Port_Write

Port Functions   **Example**

Syntax

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

Sub Port_**Write**(port **As Byte**,val **As Byte**)

Description

Writes a byte to the specified port. Only the Pins of port that are configured for output will show their value as port output on their bit position (see Pin Assignment of M32 and M128). Is a Pin configured as input, this will set an internal pull-up resistor on (bit = 1) or off (bit = 0). See Pin Assignment of M32 and M128.

Parameter

port   port number (see table)
val   output byte

port number table

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

6.16.8 Port_WriteBit

Port Functions

Syntax

void Port_WriteBit(**byte** <u>portbit</u>,**byte** <u>val</u>);

Sub Port_WriteBit(<u>portbit</u> **As Byte**,<u>val</u> **As Byte**)

Description

The function Port_WriteBit sets the value of a Pin that is configured for output. Is a Pin configured as input, a Port_WriteBit() will set an internal pull-up resistor on (bit = 1) or off (bit = 0). See Pin Assignment of M32 and M128.

Parameter

<u>portbit</u>   bit number of port (see table)
<u>val</u>   bit value (0 or 1)

Portbits Table

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

6.16.9 Port Example

```
// Program toggles the LED's on the applicationboard
// alternately every second


void main(void)
{
    Port_DataDirBit(PORT_LED1,PORT_OUT);
    Port_DataDirBit(PORT_LED2,PORT_OUT);

    while(true)  // endless loop
    {
        Port_WriteBit(PORT_LED1,PORT_ON);
        Port_WriteBit(PORT_LED2,PORT_OFF);
        AbsDelay(1000);
        Port_WriteBit(PORT_LED1,PORT_OFF);
        Port_WriteBit(PORT_LED2,PORT_ON);
        AbsDelay(1000);
    }
}
```

6.17 RC5

A common used standard protocol for infrared data communication is the RC5 code, originally developed by Phillips. This code has an instruction set of 2048 different instructions and is divided into 32 address of each 64 instructions. Every kind of equipment use his own address, so this makes it possible to change the volume of the TV without change the volume of the hifi. The transmitted code is a dataword wich consists of 14 bits.

Original protocol:

- 2 start bits for the automatic gain control in the infrared receiver
- 1 toggle bit (changes every time a new button is pressed on the IR transmitter)
- 5 address bits for the system address
- 6 instruction bits for the pressed key

The start bits help the IR receiver to synchronize and to adjust the automatic gain control of the signal. The toggle bit changes its value with every keypress. Therefore it is possible to distinguish the long press of a key with repeated presses of the same key. After a while there was a need to extend the number of possible instructions from 64 to 128. To maintain compatibility the second start bit was used for this purpose. If the second start bit is "1", the first 64 instructions can be addressed, if the 2nd start bit is "0" the next 64 instructions can be selected.

How are the individual bits transferred?

The C-Control Pro generates a carrier frequency of 36Khz on the configured pin, that is connected to the IR-Diode. All transmission pulses are 6,9444 long. There is a delay of 20,8332 µs between two pulses. For a "1" value, the frequency generation of the transmission is turned of for 889µs, and then turned on for 889µs (this equals to 32 IR impulses). A value of "0" is created with a pause of 889µs, followed from a frequency generation of 889µs. The time to transfer a whole bit is 1,778ms (2 * 889µs) and to transfer a complete 14 bit dataword is 24,889ms. If akey on remote control is pressed

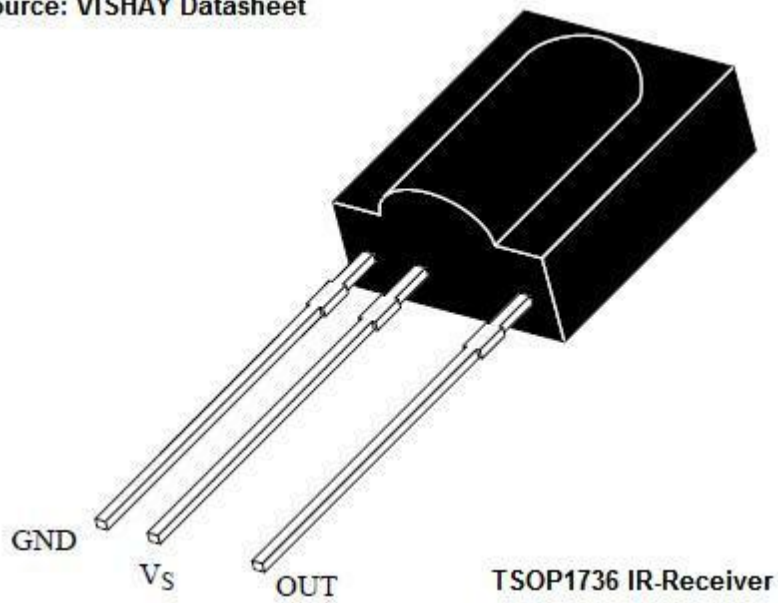for a longer duration, the corresponding dataword is repeated every 113m778ms.
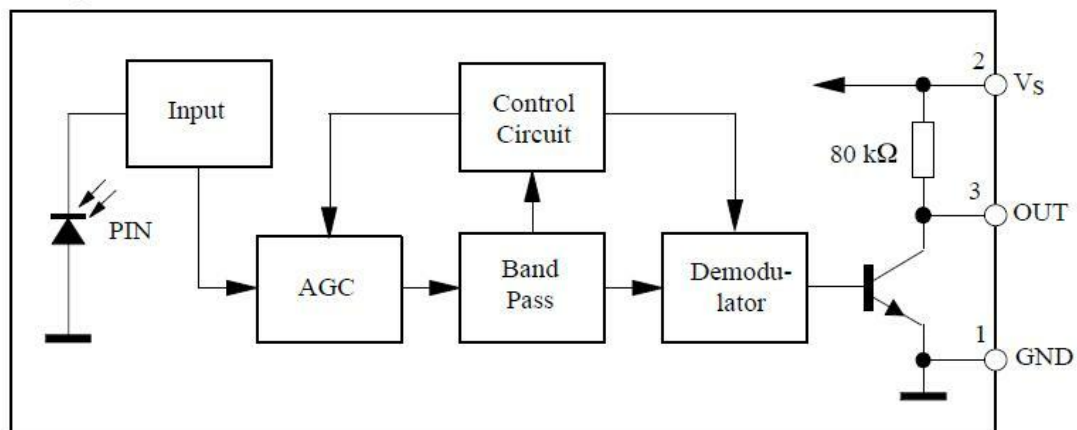
Connection to C-Control Pro (Sender diode)

I/O Port ——— R1 330R ——— IR-LED z.B. TSUS 5202
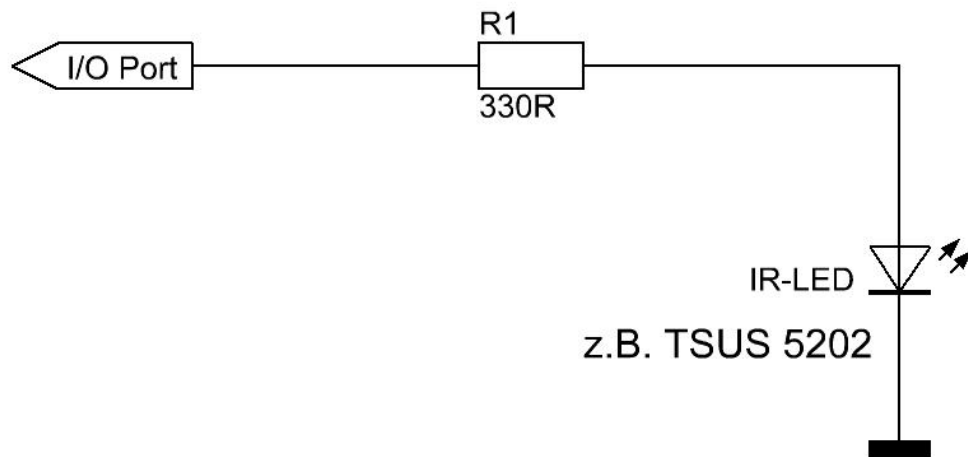
Connection to C-Control Pro (Receiver)

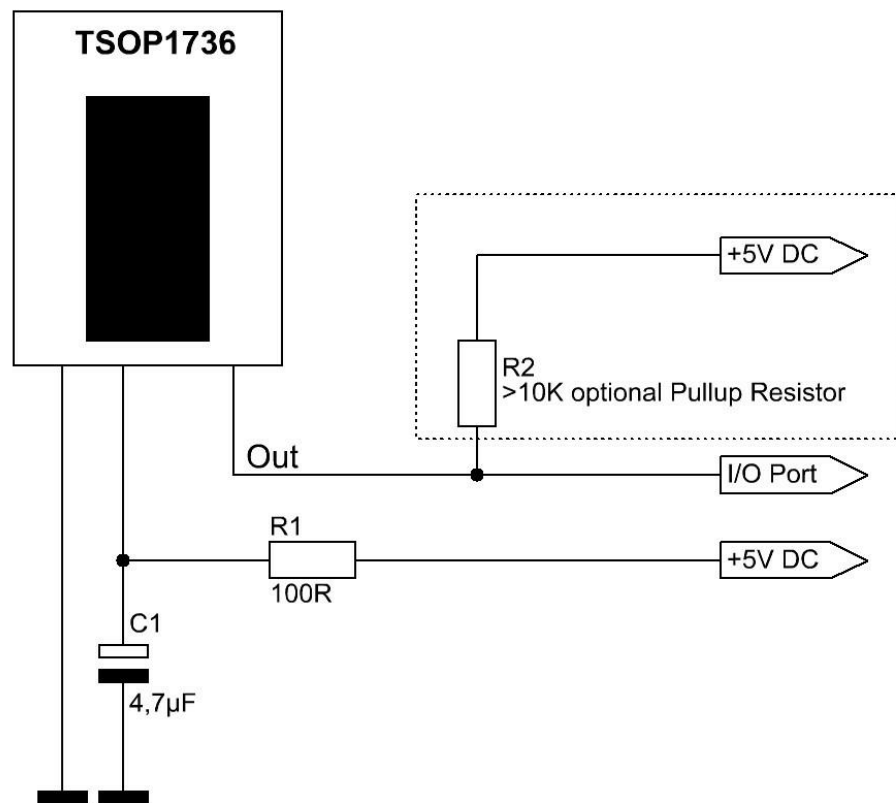Source: VISHAY Datasheet

GND   V$_S$ OUT   **TSOP1736 IR-Receiver**

Pin assignment of TSOP1736 IR-Receiver

Internal struture of receiver

External circuit of receiver for connection to C-Control Pro

6.17.1 RC5_Init

RC5 Functions

Syntax

void RC5_Init(**byte** pin);

Sub RC5_Init(pin **As Byte**)

Description

The port pin is defined, that is connected to RC5 sender or receiver.

Parameter

pin   bit number of port (see table)

Portbits Table

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

6.17.2 RC5_Read

RC5 Functions

Syntax

word RC5_Read(**void**);

Sub RC5_Read() **As Word**

Description

Recognized RC5 datawords are received from the defined port pin. If there is no signal, the receive routine waits up to 130ms. This is because there is a 113ms gap between two repeated RC5 datawords. A return value of 0 means that no RC5 signal could be detected.

➡ This function will not recognize if a different format than RC-5 is used. In case of doubt it will return wrong values.

Return Parameter

14 Bit of the received RC-5 commands

6.17.3 RC5_Write

RC5 Functions

Syntax

void RC5_Write(**word** <u>data</u>);

Sub RC5_**Write**(<u>data</u> **As Word**)

Description

The 14 bit of a RC5 dataword are send to the defined port pin.

Parameter

<u>data</u>   recognized RC5 dataword

6.18 RS232

The serial interface can be operated at speeds of up to 230.4 kilo baud. With the functions for the serial interface the first parameter will indicate the port number (0 or 1). Mega32 does only provide one serial interface (0), while Mega128 does provide two (0, 1).

6.18.1 Divider

The functions <u>Serial_Init</u>() and <u>Serial_Init_IRQ</u> get a <u>divider</u> value as baudrate parameter. The baudrate is derived from the processor clock (14,7456 MHz).

According to the Atmel processor handbook the following formula is used to calculate the divider for a specified baudrate:

divider = (14745600 / baudrate / 16) -1

Example: 15 = (14745600 / 57600 / 16) -1

DoubleClock Mode

If the High-Bit of the divider is set, the DoubleClock Mode is enabled. In this mode the <u>divider</u> value must be doubled. E.g. for 57600 baud a divider value of 0x0f (decimal 15) or 0x801e can be used. For the MIDI baudrate (31250 baud) a divider of (14745600 / 31250 / 16) -1 = 28.49 had to be used. If DoubleClock is enabled, the divider value can be specified more accurate: 0x8039

Table <u>divider</u> definition:

divider	definition	baudrate
767	SR_BD1200	1200bps
383	SR_BD2400	2400bps
191	SR_BD4800	4800bps
95	SR_BD9600	9600bps
63	SR_BD14400	14400bps
47	SR_BD19200	19200bps
31	SR_BD28800	28800bps
0x8039	SR_BDMIDI	31250bps
23	SR_BD38400	38400bps
15	SR_BD57600	57600bps
11	SR_BD76800	76800bps
7	SR_BD115200	115200bps
3	SR_BD230400	230400bps

6.18.2 Serial_Disable

Serial Functions

Syntax

void Serial_Disable(**byte** <u>serport</u>);

Sub Serial_Disable(<u>serport</u> **As Byte**)

Description

The serial interface gets switched off and the corresponding ports can be used otherwise.

Parameter

<u>serport</u>   interface number (0 = 1st serial port, 1 = 2nd serial port, ...)

6.18.3 Serial_Init

Serial Functions **Example**

Syntax

void Serial_Init(**byte** serport,**byte** par,**byte** divider);

Sub Serial_Init(serport **As Byte**,par **As Byte**,divider **As Byte**)

Description

The serial interface gets initialized. The parameter par is defined through successive or-ing of predefined bit values. The values of *character length*, *stop bits* and *parity* are or'd together. E.g. "SR_7BIT | SR_2STOP | SR_EVEN_PAR" means 7 bit character length, 2 stop bits and even parity (see Example). An example in BASIC Syntax: "SR_7BIT Or SR_2STOP Or SR_EVEN_PAR". The baud rate is defined as a divider value (see divider table).

➡ It is possible to activate the DoubleClock Mode of the Atmel AVR. This happens if the Hi-bit of the divider is set. In DoubleClock mode the normal value from the divider table must be doubled to get the same baudrate. This has the advantage that baudrates, that have no exact divider value can be represented. E.g. MIDI: The new value SB_MIDI (=0x803a) lies much nearer at the correct value of 31250baud. An example for 19200 baud: The normal divider value for 19200 baud is 0x002f. If DoubleClock Mode is used, the divider must be doubled (=0x005e). Then set the Hi-bit, and the alternative divider value for 19200 baud is 0x805e.

Parameter

serport   interface number (0 = 1st serial port, 1 = 2nd serial port, ...)
par   interface parameter (see par table)
divider   baud rate initialization (see table)

table par definitions:

Definition	Function
SR_5BIT	5 Bit char length
SR_6BIT	6 Bit char length
SR_7BIT	7 Bit char length
SR_8BIT	8 Bit char length
SR_1STOP	1 stop bit
SR_2STOP	2 stop bit
SR_NO_PAR	no parity
SR_EVEN_PAR	even parity
SR_ODD_PAR	odd parity

6.18.4 Serial_Init_IRQ

Syntax

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

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

Description

The serial interface gets initialized for usage in interrupt mode. The user has to provide a global variable as a serial buffer. This buffer services as a storage for the data that is sent to the serial interface and is received from it. The size of the buffer must be **length of the send buffer plus the length of the receive buffer plus 6 bytes** (see Example).

The maximum value for the size of the send and the receive buffer is 255 bytes each. The parameter par is defined through successive or-ing of predefined bit values. The values of *character length*, *stop bits* and *parity* are or'd together. E.g. "SR_7BIT | SR_2STOP | SR_EVEN_PAR" means 7 bit character length, 2 stop bits and even parity (see Example). An example in BASIC Syntax: "SR_7BIT Or SR_2STOP Or SR_EVEN_PAR". The baud rate is defined as a divider value (see divider table).

➡ It is possible to activate the DoubleClock Mode of the Atmel AVR. This happens if the Hi-bit of the divider is set. In DoubleClock mode the normal value from the divider table must be doubled to get the same baudrate. This has the advantage that baudrates, that have no exact divider value can be represented. E.g. MIDI: The new value SB_MIDI (=0x803a) lies much nearer at the correct value of 31250baud. An example for 19200 baud: The normal divider value for 19200 baud is 0x002f. If DoubleClock Mode is used, the divider must be doubled (=0x005e). Then set the Hi-bit, and the alternative divider value for 19200 baud is 0x805e.

➡ Please use Serial_ReadExt() if you work in serial IRQ mode. Serial_Read() only supports polled mode.

Parameter

serport interface number (0 = 1st serial port, 1 = 2nd serial port, ...)
ramaddr address of the buffer
recvlen size of receive buffer
sendlen size of send buffer
par   interface parameter (see par table)
divider baud rate initialization (see table)

table par definitions:

Definition	Function
SR_5BIT	5 Bit char length
SR_6BIT	6 Bit char length
SR_7BIT	7 Bit char length

SR_8BIT	8 Bit char length
SR_1STOP	1 stop bit
SR_2STOP	2 stop bit
SR_NO_PAR	no parity
SR_EVEN_PAR	even parity
SR_ODD_PAR	odd parity

6.18.5 Serial_IRQ_Info

Serial Functions

Syntax

byte Serial_IRQ_Info(**byte** serport, **byte** info);

Sub Serial_IRQ_Info(serport **As Byte,** info **As Byte**) **As Byte**

Description

In dependency of the info parameter the function returns how many bytes have been received or a written to the send buffer.

Parameter

serport   interface number (0 = 1st serial port, 1 = 2nd serial port)

info values**:**

RS232_FIFO_RECV (0)   number of bytes received
RS232_FIFO_SEND(1)   number of bytes written to he send buffer

Return Parameter

result in bytes

6.18.6 Serial_Read

Serial Functions

Syntax

byte Serial_Read(**byte** serport);

Sub Serial_Read(serport **As Byte**) **As Byte**

Description

Reads one byte from the serial interface. If is there is no byte available in the serial interface, the function

waits until a byte has been received.

➡️ Please use Serial_ReadExt() if you work in serial IRQ mode. Serial_Read() only supports polled mode.

Parameter

serport interface number (0 = 1st serial port, 1 = 2nd serial port)

Return Parameter

received byte from the serial interface

6.18.7 Serial_ReadExt

Serial Functions

Syntax

```
word Serial_ReadExt(byte serport);

Sub Serial_ReadExt(serport As Byte) As Word
```

Description

Reads one byte from the serial interface. In opposite to Serial_Read() Serial_ReadExt() returns immediately even if there is no byte available in the serial port. In this case 256 (0x100) is returned.

➡️ Please use Serial_ReadExt() if you work in serial IRQ mode. Serial_Read() only supports polled mode.

Parameter

serport interface number (0 = 1st serial port, 1 = 2nd serial port)

Return Parameter

received byte from the serial interface
256 (0x100) if there was no byte available

6.18.8 Serial_Write

Serial Functions Example

Syntax

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

Sub Serial_Write(serport As Byte, val As Byte)
```

Description

One byte is send to the serial interface.

Parameter

serport interface number (0 = 1st serial port, 1 = 2nd serial port)
val output byte value

6.18.9 Serial_WriteText

Serial Functions

Syntax

```
void Serial_WriteText(byte serport,char text[]);

Sub Serial_WriteText(serport As Byte,ByRef Text As Char)
```

Description

All characters of the char array up to the terminating zero are send to the serial interface.

Parameter

serport interface number (0 = 1st serial port, 1 = 2nd serial port)
text char array

6.18.10 Serial Example

```
// string output on the serial interface
void main(void)
{
    int i;
    char str[10];

    str="test";
    i=0;
    // initialize serial port with 19200baud, 8 bit, 1 stop bit, no parity
    Serial_Init(0,SR_8BIT|SR_1STOP|SR_NO_PAR,SR_BD19200);

    while(str[i]) Serial_Write(0,str[i++]);  // output string to serial port
}
```

6.18.11 Serial Example (IRQ)

```
// 35 byte send + receive buffer + 6 byte internal FIFO organization
byte buffer[41];                        // array declaration

// string output to serial interface
void main(void)
{
```

```
    int i;
    char str[10];

    str="test";
    i=0;
    // initialize serial port with 19200baud, 8 bit, 1 stop bit, no parity
    // 20 byte receive buffer - 15 byte send buffer
    Serial_Init_IRQ(0,buffer,20,15,SR_8BIT|SR_1STOP|SR_NO_PAR,SR_BD19200);

    while(str[i]) Serial_Write(0,str[i++]);  // display string
}
```

6.19 Servo

RC servos are composed of a DC motor mechanically linked to a potentiometer. Pulse-width modulation (PWM) signals sent to the servo are translated into position commands by electronics inside the servo. When the servo is commanded to rotate, the DC motor is powered until the potentiometer reaches the value corresponding to the commanded position. The servo is controlled by three wires: ground (usually black/orange), power (red) and control (brown/other colour). The servo will move based on the pulses sent over the control wire, which set the angle of the actuator arm. The servo expects a pulse every 20 ms in order to gain correct information about the angle. The width of the servo pulse dictates the range of the servo's angular motion. A servo pulse of 1.5 ms width will set the servo to its "neutral" position, or 90°. For example a servo pulse of 1.25 ms could set the servo to 0° and a pulse of 1.75 ms could set the servo to 180°. The physical limits and timings of the servo hardware varies between brands and models, but a general servo's angular motion will travel somewhere in the range of 180° - 210° and the neutral position is almost always at 1.5 ms.

Connection to C-Control Pro
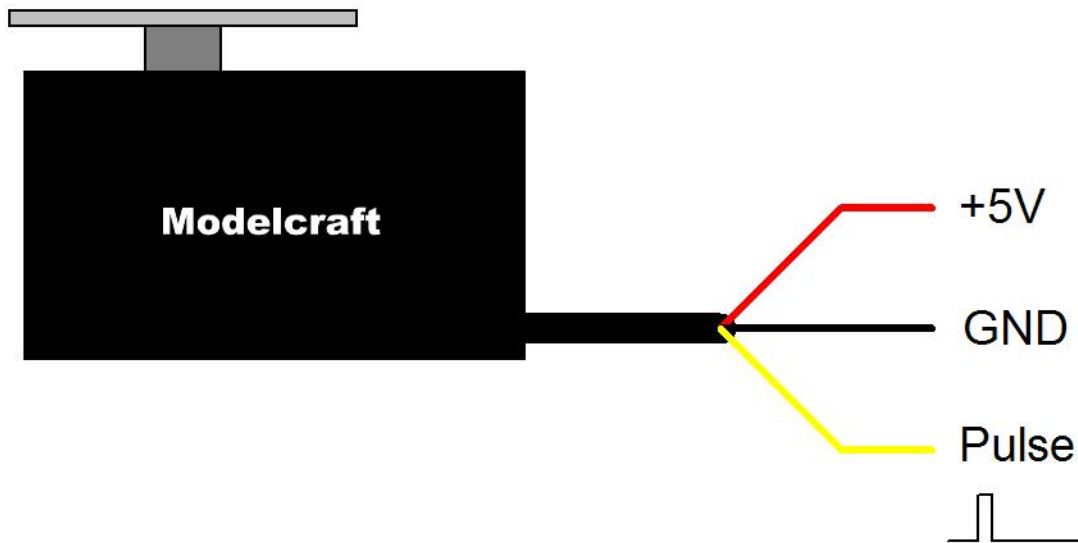
+5Volt ist the supply voltage of the servo, it must provide enough current to drive the servo. The ground of the servo and the ground of the C-Control Pro unit must be the same. The pulse for the servo is generated by the PWM signal of the C-Control unit.

6.19.1   Servo_Init

Servo Functions   **Example**

Syntax

void Servo_Init(**byte** servo_cnt, **byte** servo_interval, **byte** ramaddr[], **byte** timer);

Sub Servo_Init(servo_cnt **As Byte**, servo_interval **As Byte**, **ByRef** ramaddr **As Byte**, ti

Description

Intializes the internal servo routines. The servo_cnt parameter controls how many servos can be driven at the same time. The servo_interval parameter describes the period length (10 or 20ms), with timer the used 16-Bit timer can be chosen. Timer 3 is only available on the Mega128. The user must supply ram space to operate the servos. The required size is servo_cnt * 3. E.g., if the user wants to operate 10 servos, at **byte** array of 30 bytes is needed.

➡️ A 16-bit Timer is needed for the servo steering routines. This has to be Timer 1 or Timer 3 (Mega128). Is the timer turned off, or is used for other purposes the servo routines will not work.

Parameter

servo_cnt   number of possible servos (maximum 20)

servo_interval periodic length (0=10ms, 1=20ms)
ramaddr   address of memory block
timer   16-Bit Timer used for servo steering (0=Timer 1, 1=Timer 3 only Mega128)

6.19.2 Servo_Set

Servo Functions Example

Syntax

```
void Servo_Set(byte portbit, word pos);

Sub Serial_Init(portbit As Byte, pos As Word)
```

Description

Sets the pulse length to steer the actuator arm. The output port is set with the portbit parameter (See Pin Assignment of M32 and M128).

➡ The sum of all user set pulse lengths should not exceed the period length (see servo_interval parameter), otherwise an erratic behaviour could happen. E.g. with 20ms period length, a total of 8 servos can each be set to a pulse length of 2500µs. To have some safety margin, the sum of the pulse lengths should be **less** than the period length for a small amount.

Parameter

portbit   bit number of port (see table)
pos   pulse length for servo in µsec (500 - 2500)

Portbits Table

Definition	Portbit
PortA.0	0
...	...
PortA.7	7
PortB.0	8
...	...
PortB.7	15
PortC.0	16
...	...
PortC.7	23
PortD.0	24
...	...
PortD.7	31
from here only Mega128	
PortE.0	32
...	...
PortE.7	39
PortF.0	40
...	...

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

6.19.3 Servo Example

```
// Activation of 3 Servos and stop after 10 seconds
void main(void)
{
    byte servo_var[30]; // Servo internal variables

    // Max. 10 Servos, 20ms interval, Timer 3
    Servo_Init(10, 1, servo_var, 1);

    Servo_Set(7, 2000);  // Servo Portbit 7   2000µs
    Servo_Set(6, 1800);  // Servo Portbit 6   1800µs
    Servo_Set(5, 1600);  // Servo Portbit 5   1600µs

    AbsDelay(5000);

    Servo_Set(7, 1000);  // Servo Portbit 7   1000µs

    AbsDelay(5000);

    Servo_Set(7, 0);     // all Servos off
    Servo_Set(6, 0);
    Servo_Set(5, 0);
}
```

6.20 SPI

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard named by Motorola that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines.

6.20.1 SPI_Disable

SPI Functions

Syntax

```
void SPI_Disable(void);

Sub SPI_Disable()
```

Description

The SPI will be disabled and the corresponding ports can be used otherwise.

➡ Disabling the SPI interface will prevent usage of the USB interface on the application board. On the other hand, if you don't use the USB interface, SPI_Disable() will allow to use these ports for other purposes.

Parameter

None

6.20.2 SPI_Enable

SPI Functions

Syntax

void SPI_Enable(**byte** ctrl);

Sub SPI_Enable(ctrl **As Byte**)

Description

The SPI interface is initialized with the value of ctrl (see **SPCR** register in Atmel Mega Reference Manual).

Parameter

ctrl   initialization parameter (Mega SPCR Register)

Bit 7 - SPI Interrupt Enable (do not enable, cannot be used from C-Control Pro now)
Bit 6 - SPI Enable (must be set)
Bit 5 - Data Order (1 = LSB first, 0 = MSB first)
Bit 4 - Master/Slave Select (1 = Master, 0 = Slave)
Bit 3 - Clock polarity (1 = leading edge falling, 0 = leading edge rising)
Bit 2 - Clock Phase (1 = sample on trailing edge, 0 = sample on leading edge)

Bit 1	Bit 0	SCK Frequency
0	0	f_{Osc} / 4
0	1	f_{Osc} / 16
1	0	f_{Osc} / 64
1	1	f_{Osc} / 128

6.20.3 SPI_Read

SPI Functions

Syntax

byte SPI_Read();

Sub SPI_**Read**() **As Byte**

Description

A byte is read from the SPI interface.

Return Parameter

received byte from the SPI interface

6.20.4 SPI_ReadBuf

SPI Functions

Syntax

void SPI_ReadBuf(**byte** <u>buf</u>[], **byte** <u>length</u>);

Sub SPI_ReadBuf(**ByRef** <u>buf</u> **As Byte**, <u>length</u> **As Byte**)

Description

A number of bytes are read from the SPI interface into an array.

Parameter

<u>buf</u>   pointer to byte array
<u>length</u>   number of bytes to read

6.20.5 SPI_Write

SPI Functions

Syntax

void SPI_Write(**byte** <u>data</u>);

Sub SPI_**Write**(<u>data</u> **As Byte**)

Description

One byte is send to the serial interface.

Parameter

<u>data</u>   output byte value

6.20.6 SPI_WriteBuf

SPI Functions

Syntax

void SPI_WriteBuf(**byte** <u>buf</u>[], **byte** <u>length</u>);

Sub SPI_WriteBuf(**ByRef** <u>buf</u> **As Byte**, <u>length</u> **As Byte**)

Description

A number of bytes are sent to the SPI interface.

Parameter

<u>buf</u>   pointer to byte array
<u>length</u>   number of bytes to be transferred

6.21 Strings

One part of these string routines is implemented in the Interpreter, another can be called up after appending library "String_Lib.cc". Since the functions in "String_Lib.cc" are realized through Bytecode they are slower when executed. Library functions however have the advantage that they can be taken from the project by omitting the library in case they are not needed. Direct Interpreter functions are always present, will however take up flash memory.
 There is no explicit "String" data type. A string is based on a character array. The size of the array must be chosen in such a way that all characters of the string fit into the character array. Additionally some space is needed for a terminating character (decimal Zero) in order to indicate the end of the character string.

6.21.1 Str_Comp

String Functions

Syntax

char Str_Comp(**char** <u>str1</u>[],**char** <u>str2</u>[]);

Sub Str_Comp(**ByRef** <u>str1</u> **As Char**,**ByRef** <u>str2</u> **As Char**) **As Char**

Description

Two strings are compared.

Parameter

<u>str1</u> pointer to char array 1
<u>str2</u> pointer to char array 2

Return Parameter

0   both strings are equal
<0   if the first string is smaller than the second
>0   if the first string is greater than the second

Remark

The attribute smaller or greater is specified for the character difference at the first point of difference between both strings.

6.21.2 Str_Copy

String Functions

Syntax

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

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

Description

The source string (source) is copied to the destination string (destination). During copying also the string termination character of the source character string is copied.

Parameter

destination   pointer to destination string
source   pointer to source string
offset   Number of characters by which the source string is offset when copied to the destination string..

If offset has the value **STR_APPEND** (0xffff) then the length of the destination string is assumed as offset. In this case the source string is copied behind the destination string.

6.21.3 Str_Fill

String Functions (Library "*String_Lib.cc*")

Syntax

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

Sub Str_Fill(ByRef dest As Char,c As Char,len As Word)
```

Description

The string dest is filled with character c.

Parameter

<u>dest</u> pointer to destination string
<u>c</u>   character that is written into the string
<u>len</u> count, how often <u>c</u> is written into the string

6.21.4 Str_Isalnum

String Functions (Library "*String_Lib.cc*")

Syntax

byte Str_Isalnum(**char** <u>c</u>);

Sub Str_Isalnum(<u>c</u> **As Char**) **As Byte**

Description

A character is tested if it is alphabetically or a digit.

Parameter

<u>c</u> tested character

Return Parameter

1 if the character is alphabetically or a digit (upper- or lowercase)
0 else

6.21.5 Str_Isalpha

String Functions (Library "*String_Lib.cc*")

Syntax

byte Str_Isalpha(**char** <u>c</u>);

Sub Str_Isalpha(<u>c</u> **As Char**) **As Byte**

Description

A character is tested if it is alphabetically.

Parameter

<u>c</u> tested character

Return Parameter

1 if the character is alphabetically (upper- or lowercase)

0 else

6.21.6 Str_Len

String Functions

Syntax

word Str_Len(**char** <u>str</u>[]);

Sub Str_Len(**ByRef** <u>str</u> **As Char**) **As Word**

Description

The length of the string (character array) is returned.

Parameter

<u>str</u> pointer to string

Return Parameter

length of the string (without terminating zero)

6.21.7 Str_Printf

String Functions   **Example**

Syntax

void Str_Printf(**char** <u>str</u>[], **char** <u>format</u>[], ...);

Sub Str_Printf(**ByRef** <u>str</u> **As Char, ByRef** <u>format</u> **As Char, ...**)

Description

This function creates a formatted string into <u>str</u>. The format string is similar to the formatting of printf() in C. The format always begins with **%** and ends with a **type** (**d**,**x**,**s**,**f**). In the following table all type parameters are explained. Between **%** and **type** an optional **width** and **precision** can be used.

%[*width*][.*prec*]**Typ**   (the brackets describes the optional part)

The **width** is the minimal space for the output of the number. If the number is smaller than **width**, the number is padded to the left with spaces. If the **width** begins with "0" the left is padded width "0" instead of spaces. A period "." describes an optional **precision** parameter, that defines the number of decimal places, when floating point numbers (**%f**) are used, or the base of the number when using unsigned integer (**%u**). See Str_Printf Example.

Format	Description

%[width**]d**	integer
%[width**][**.prec**]u**	unsigned integer
%[width**]x**	hexadecimal
%[width**][**.prec**]f**	floating point
%[width**]s**	string

Parameter

str   pointer to string
format pointer to format string

6.21.8 Str_ReadFloat

String Functions

Syntax

```
float Str_ReadFloat(char str[]);

Sub Str_ReadFloat(ByRef str As Char) As Single
```

Description

The value of a string representing a floating point number is returned. The number is recognized, even if there or other characters after the number.

Parameter

str pointer to string

Return Parameter

floating point value of string

6.21.9 Str_ReadInt

String Functions

Syntax

```
int Str_ReadInt(char str[]);

Sub Str_ReadInt(ByRef str As Char) As Integer
```

Description

The value of a string representing an integer number is returned. The number is recognized, even if there or other characters after the number.

Parameter

<u>str</u> pointer to string

Return Parameter

integer value of string

6.21.10 Str_ReadNum

String Functions

Syntax

word Str_ReadNum(**char** <u>str</u>[], **byte** <u>base</u>);

Sub Str_ReadNum(**ByRef** <u>str</u> **As Char,** <u>base</u> **As Byte**) **As Word**

Description

The value of a string representing an unsigned number is returned. The number is recognized, even if there or other characters after the number. The <u>base</u> parameter is the base of the numeric value. E. g. to read a hexadecimal number, a base of 16 is to apply.

Parameter

<u>str</u> pointer to string
<u>base</u> base of converted number

Return Parameter

numeric value of string

6.21.11 Str_Substr

String Functions (Library "*String_Lib.cc*")

Syntax

int Str_SubStr(**char** <u>source</u>[],**char** <u>search</u>[]);

Sub Str_SubStr(**ByRef** <u>source</u> **As Char**,**ByRef** <u>search</u> **As Char**) **As Integer**

Description

A substring <u>search</u> is searched inside string <u>source</u>. If the substring is found, the position of the substring is returned.

Parameter

<u>source</u> string that is searched
<u>search</u> substring that is looked for

Return Parameter

position of the found substring
-1   else

6.21.12 Str_WriteFloat

String Functions

Syntax

void Str_WriteFloat(**float** <u>n</u>, **byte** <u>decimal</u>, **char** <u>text</u>[], **word** <u>offset</u>);

Sub Str_WriteFloat(<u>n</u> **As Single**,<u>decimal</u> **As Byte**,**ByRef** <u>text</u> **As Char**,<u>offset</u> **As Word**)

Description

The floating point number n is converted to an ASCII string with <u>decimal</u> number of decimal digits after the period. The result is stored in the string <u>text</u> with an offset of <u>offset</u>. The offset parameter is used to change a string after a specified number (<u>offset</u>) of characters and leave the beginning of the string intact.

Parameter

<u>n</u>   float number
<u>decimal</u> number of decimal digit after the period
<u>text</u>   pointer to destination string
<u>offset</u>   offset that is applied to the position where the string is copied

If <u>offset</u> has the value **STR_APPEND** (0xffff) then the length of the destination string is assumed as offset. In this case the source string is copied behind the destination string.

6.21.13 Str_WriteInt

String Functions

Syntax

void Str_WriteInt(**int** <u>n</u>, **char** <u>text</u>[], **word** <u>offset</u>);

Sub Str_WriteInt(<u>n</u> **As Integer**,**ByRef** <u>text</u> **As Char**,<u>offset</u> **As Word**)

Description

The integer number <u>n</u> is converted to a signed ASCII string. The result is stored in the string <u>text</u> with an offset of <u>offset</u>. The offset parameter is used to change a string after a specified number (<u>offset</u>) of characters and leave the beginning of the string intact.

Parameter

<u>n</u>   integer number

text   pointer to destination string
offset offset that is applied to the position where the string is copied

If <u>offset</u> has the value **STR_APPEND** (0xffff) then the length of the destination string is assumed as offset. In this case the source string is copied behind the destination string.

6.21.14 Str_WriteWord

String Functions

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)
```

Description

The word n is converted to an ASCII string. The result is stored in the string <u>text</u> with an offset of <u>offset</u>. The offset parameter is used to change a string after a specified number (<u>offset</u>) of characters and leave the beginning of the string intact. If the resulting string is smaller than <u>minwidth</u> the beginning of the string gets filled with zeros "0".

The base of the numbering system can be given in the <u>base</u> parameter. If you set <u>base</u> to 2, you will get a string with binary digits. A base of 8 produces an octal string, and a base of 16 a hexadecimal string. If the base is set to a number greater than 16, more characters of the alphabet are used. E.g. a base of 18 produces a string with the digits '0'-'9' and 'A'-'H'.

Parameter

<u>n</u>   16 bit word
<u>base</u>   base of the number system
<u>text</u>   pointer to destination string
<u>offset</u>   offset that is applied to the position where the string is copied
<u>minwidth</u> minimal width of the string

If <u>offset</u> has the value **STR_APPEND** (0xffff) then the length of the destination string is assumed as offset. In this case the source string is copied behind the destination string.

6.21.15 Str_Printf Example

```
// CompactC
void main(void)
{
    char str[80];

    // Integer
```

```
      Str_Printf(str, "arg1: %d\r", 1234);
      Msg_WriteText(str);

      // Ouput of integer, floating point, string und hex number
      Str_Printf(str, "arg1: %8d arg2:%10.3f arg3:%20s arg4: %x\r",
          1234, 2.34567, "hello world", 256);
      Msg_WriteText(str);
      Str_Printf(str, "arg1: %u arg2: %.2u\r", 65000, 0xff);
      Msg_WriteText(str);}
}


' Basic
Sub main()
    Dim str(80) As Char

    Str_Printf(str, "arg1: %08d arg2:%10.3f arg3:%20s arg4: %x\r",
      1234, 2.34567, "hello world", 256)
    Msg_WriteText(str)
    Str_Printf(str, "arg1: %u arg2: %.2u\r", 65000, &Hff)
    Msg_WriteText(str)
End Sub
```

6.22 Threads

Multi Threading

Multi Threading is a so to speak parallel execution of several tasks in a program. One of these tasks is called "Thread". When Multi Threading it will rather rapidly be toggled between the various threads so the impression of simultaneousness is created.

The C-Control Pro firmware supports besides the main program (Thread "0") up to 13 additional threads. When multi threading after a certain number of processed Byte instructions the current thread will be set "*inactive*" and the next executable thread is searched for. After that the execution of the new thread will be started. The new thread may again be the same as before depending on how many threads had been activated or are ready for processing. The main program counts as first thread. Therefore thread "0" is active at all times even if no threads have explicitly been started.

The priority of threads can be influenced by changing the number of Bytecodes which one thread is allowed to execute until the next thread change takes place (see Thread Options). The smaller the number of cycles until the change takes place, the lower the priority of the thread. The execution time of one Bytecode is appr. 7 – 9 microseconds. With certain Bytecode instructions however this will take longer (e. g. Floating Point Operations).

➡ Also internal Interpreter functions count as cycles. Since e. g. Serial_Read will wait until a character arrives from the serial interface a cycle can in exceptional cases become quite long.

A thread will receive as much space for its local variables as has been assigned to in the Thread Options of the project. The exception is thread "0" (the main program). This thread will receive the entire memory space that has been left over by the other threads. Therefore it should be planned in advance how much memory space may be needed by each additional thread.

➡ In order to start additional threads "*Multi Threading*" must be activated in [Project Options](#) and for the additional threads the correct values must be set in [Thread Options](#).

➡ When working with threads [Thread_Delay](#) rather than [AbsDelay](#) must always be used. If nevertheless e. g. an AbsDelay(1000) is used the following effect may occur: Since the thread will only change to the next thread after 5,000 cycles (default value), the thread would run for 5,000 * 1,000ms = 5,000 seconds before the next thread would be able to start working.

Thread Synchronisation

Sometimes it is necessary for a thread to wait for another thread. This may e. g. be a common hardware resource which can only execute one thread. Sometimes also critical program areas may be defined which may only be entered by one thread. This functions are being realized through instructions [Thread_Wait](#) and [Thread_Signal](#).

A thread bound to wait will execute instruction Thread_Wait with a signal number. The condition of the thread is set on *waiting*. This means that the thread may be ignored at a possible thread change. If the other thread has completed its critical work it will send the command Thread_Signal with the same signal number the first thread had used for its Thread_Wait. The thread condition of the waiting thread will change from *waiting* to *inactive* and will then be considered again at a possible thread change.

Deadlocks

When all active threads set out for a waiting condition with [Thread_Wait](#) then there will be no more threads which can release the other threads from their waiting conditon. Therefore these constellations should be avoided when programming.

Table Thread Conditions

Condition	Meaning
active	The thread is presently executed
inactive	Can be activated again after a thread change
sleeping	Will after a number of ticks be set to "inactive" again
waiting	The thread awaits a signal

6.22.1 Thread_Cycles

Thread Functions

Syntax

void Thread_Cycles(**byte** <u>thread</u>,**word** <u>cycles</u>);

Sub Thread_Cycles(<u>thread</u> **As Byte**,<u>cycles</u> **As Word**)

Description

Sets the number of executed bytecode instructions before thread change to the parameter <u>cycles</u>.

➡️ If a thread is freshly started, it will get the cycle count that was defined in the project options. It only makes sense to call Thread_Cylces() **after** a thread has been started.

Parameter

<u>thread</u> (0-13) number of the thread
<u>cycles</u> cycle count until thread change

6.22.2 Thread_Delay

Thread Functions   **Example**

Syntax

void Thread_Delay(**word** <u>delay</u>);

Sub Thread_Delay(<u>delay</u> **As Word**)

Description

With this function a thread will set to "sleep" for a specified time. After this time the thread is again ready for execution. The waiting period is given in ticks that are created by Timer 2. If Timer 2 is set off or used for other purposes, the mode of operation of Thread_Delay() is not defined.

➡️ Even if Thread_Delay() looks like any other wait function, you have to keep in mind that the thread is not automatically executed after the waiting period. The thread is then ready for execution, but it will not started until the next thread change.

Parameter

<u>delay</u> number of 10ms ticks that should be waited

6.22.3 Thread_Info

Thread Functions

Syntax

word Thread_Info(**byte** <u>info</u>);

Sub Thread_Info(<u>info</u> **As Byte**) **As Word**

Description

The function returns information about the calling thread. The <u>info</u> parameter defines what kind of information is returned.

Parameter

<u>info</u> values:

TI_THREADNUM   number of the calling thread
TI_STACKSIZE   defined stack size
TI_CYCLES   number of cycles before thread change

Return Parameter

info result

6.22.4 Thread_Kill

Thread Functions

Syntax

void Thread_Kill(**byte** <u>thread</u>);

Sub Thread_Kill(<u>thread</u> **As Byte**)

Description

Terminates a thread. If 0 is given as thread number, the whole program will be terminated.

Parameter

<u>thread</u> (0-13) thread number

6.22.5 Thread_Lock

Thread Functions

Syntax

void Thread_Lock(**byte** <u>lock</u>);

Sub Thread_Lock(<u>lock</u> **As Byte**)

Description

With this function you can inhibit thread changes. This is reasonable if you have a series of port operations or other hardware actions that should not timely be separated in a thread change.

➡️ If you forget to remove the thread lock, the multithreading is not working.

Parameter

lock if set to 1 thread changes are inhibited, 0 means thread changes are allowed

6.22.6 Thread_MemFree

Thread Functions

Syntax

word Thread_MemFree(**void**);

Sub Thread_MemFree() **As Word**

Description

Returns the free memory that is available for the calling thread.

Parameter

None

Return Parameter

free memory in bytes

6.22.7 Thread_Resume

Thread Functions

Syntax

void Thread_Resume(**byte** thread);

Sub Thread_Resume(thread **As Byte**)

Description

If a thread has the state "waiting" it can be set to "inactive" with this function call. "Inactive" means that a thread is ready for activation at a thread change.

Parameter

thread (0-13) thread number

6.22.8 Thread_Signal

Thread Functions

Syntax

void Thread_Signal(**byte** signal);

Sub Thread_Signal(signal **As Byte**)

Description

Has a thread been set to state "waiting" with a call to Thread_Wait() it can be set to "inactive" with a call to Thread_Signal(). The signal parameter must have the same value as the value that has been used in the call to Thread_Wait().

Parameter

signal signal value

6.22.9 Thread_Start

Thread Functions   **Example**

Syntax

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

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

Description

A new thread gets started. Every function in the program can be used as starting function for the thread.

➡ If the thread is started inside a function that has parameters defined in the function header, the value of these parameters is undefined!

Parameter

thread   (0-13) thread number
func   function name of the function where the thread will be started

Remark

The floating point data type seems not suitable for an function vector, but a 4 byte data type is needed to specify a function in the Mega128 environment.

6.22.10 Thread_Wait

Thread Functions

Syntax

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

Sub Thread_Wait(thread As Byte,signal As Byte)
```

Description

The thread gets the state "waiting". The state can be changed back to "inactive" with calls to Thread_Resume() or Thread_Signal().

Parameter

thread   (0-13) thread number
signal   signal value

6.22.11 Thread Example

```
// demo program of multithreading
// this program makes no debouncing, therefore a short trigger of the switch
// can lead to more than one string outputs

void thread1(void)
{
    while(true)  // endless loop
    {
        if(!Port_ReadBit(PORT_SW2)) Msg_WriteText(str2);  // SW2 is pressed
    }
}

char str1[12],str2[12];

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

    Port_DataDirBit(PORT_SW1, PORT_IN);  // set Pin to input
    Port_DataDirBit(PORT_SW2, PORT_IN);  // set Pin to input
    Port_WriteBit(PORT_SW1, 1);  // set pull-up
    Port_WriteBit(PORT_SW1, 1);  // set pull-up

    Thread_Start(1,thread1); // start new Thread

    while(true)     // endless loop
    {
        if(!Port_ReadBit(PORT_SW1)) Msg_WriteText(str1); // SW1 is pressed
    }
```

```
}
```

6.22.12 Thread Example 2

```
// multithread2: multithreading with Thread_Delay()
// necessary library: IntFunc_Lib.cc

void thread1(void)
{
    while(true)
    {
        Msg_WriteText(str2); Thread_Delay(200);
    }                                       // "Thread2" is displayed
}                                           // after that the thread
                                            // sleeps for 200ms


char str1[12],str2[12];                     // global variable declaration

//--------------------------------------------------------------------
// main program
//
void main(void)
{
    str1="Thread1";                         // variable declaration
    str2="Thread2";                         // variable declaration

    Thread_Start(1,thread1);                // start new thread

    while(true)                             // endless loop
    {
        Thread_Delay(100); Msg_WriteText(str1);
    }                                       // the thread sleeps for 100ms
}                                           // after that "Thread1" is displayed
```

6.23 Timer

In C-Control Pro Mega 32 there are two, in Mega128 are three independent timers available. These are *Timer_0* with 8 bit and *Timer_1* with 16 bit (*Timer_3* with 16 bit for Mega128 only). *Timer_2* is used by the firmware as an internal time base and is set firm to a 10ms interrupt. These internal timers can be utilized for a multitude of tasks:

- Event Counter
- Frequency Generation
- Pulse Width Modulation
- Timer Functions
- Pulse & Period Measurement
- Frequency Measurement

6.23.1 Event Counter

Here are two examples for how a Timer can be used for an Event Counter:

Timer0 (8 Bit)

```
//  Example: Pulse Counting with CNT0
Timer_T0CNT();
pulse(n);            //  generate n Pulses
count=Timer_T0GetCNT();
```

➡ With **Mega128** for reasons of the hardware the use of *Timer_0* as counter is not possible!

Timer1 (16 Bit)

```
//  Example: Pulse Counting with CNT1
Timer_T1CNT();
pulse(n);            //  generate n Pulses
count=Timer_T1GetCNT();
```

6.23.2 Frequency Generation

To generate frequencies *Timer_0, Timer_1* and *Timer_3* can be utilized as follows:

Timer0 (8 Bit)

1. Example:

```
// Square Wave Signal with 10*1,085 µs = 10,85 µs Period Duration
Timer_T0FRQ(10, PS0_8)
```

2. Example: Pulsed Frequency Blocks (Project FRQ0)

```
void main(void)
{
    int delval;          // Variable for the On/Off Time
    delval=200;          // Value Assignment for Variable delval

 // Frequency: Period=138,9 µs*100=13,9 ms,Frequency=72Hz
    Timer_T0FRQ(100,PS0_1024); // Timer is set to Frequency

    while (1)
    {
        AbsDelay(delval);        // Time Delay by 200ms
        Timer_T0Stop();          // Timer is stopped
        AbsDelay(delval);        // Time Delay by 200ms
        Timer_T0Start(PS0_1024); // Timer will be switched on with
                                 // Timer Prescaler PS0_1024.
    }
```

}

➡️ The program will on **Mega128** not work in USB mode since output PB4 is in conjunction with the USB interface used on the Application Board.

Timer1 (16 Bit)

Example: Frequency Generation with 125 * 4,34 µs = 1085µs Period

Timer_T1FRQ(125,PS_64);

Timer3 (16 Bit) (only Mega128)

Example: Frequency Generation with 10*1,085 µs =10,85 µs Period and 2*1,085µs =2,17 µs Phase Shift

Timer_T3FRQX(10,2,PS_8);

6.23.3 Frequency Measurement

Timer_1 (16Bit) and *Timer_3* (16Bit) (only Mega128) can be used for direct measurement of a frequency. The pulses per second are being counted, the result is then delivered in Hertz units. The maximum frequency is 64kHz and is yielded by the 16 bit counter. An example for this kind of frequency measurement can be found under "Demo Programs/FreqMeasurement". By shortening the measuring time also higher frequencies can be measured. The result has then to be re-calculated accordingly.

6.23.4 Pulse Width Modulation

There are two independent timers available for pulse width modulation. These are *Timer_0* with 8 bit and *Timer_1* with 16 bit. By use of a pulse width modulation Digital-Analog-Converters can be realized very easily. On the Mega128 *Timer_3* can be used additionally.

Timer0 (8 Bit)

Example: Pulse Width Modulation with 138,9 µs Period and 5,42 µs Pulse Width, changed to 10,84 µs Pulse Width

```
//  Pulse: 10*542,5 ns = 5,42 µs, Period: 256*542,5 ns = 138,9 µs
Timer_T0PWM(10,PS0_8);

Timer_T0PW(20);        //  Pulse: 20*542,5 ns = 10,84 µs
```

Timer1 (16 Bit)

Example: Pulse Width Modulation with 6,4 ms Period and 1,28 ms Pulse Width Channel A and 640 µs Pulse Width Channel B

```
Timer_T1PWMX(10,20,10,PS_1024); // Period: 100*69,44 µs = 6,94 ms
                                 // PulseA: 20*69,44 µs = 1,389 ms
                                 // PulseB: 10*69,44 µs = 694,4 µs
```

6.23.5 Pulse & Period Measurement

By use of *Timer_1* or *Timer_3* (only Mega128) pulse widths and signal periods can be measured. Here by use of the Input Capture Function (specific register of the Controller) the time between two signal slopes is measured. This function utilizes the Capture-Interrupt (INT_TIM1CAPT). A pulse is measured between a rising and the next falling signal edge. A period is measured between two rising signal edges. Because of the Input Capture Function program delay times will not as an inaccuracy be entered into the measuring result. With a programmable prescaler the resolution of *Timer_1* can be set. Prescaler see **Table**.

Example: Activate Pulse Width Measurement (Project PMeasurement) 434 µs (100 x 4,34 µs, see Table)

```
word PM_Value;

void Timer1_ISR(void)
{
    int irqcnt;

    PM_Value=Timer_T1GetPM();
    irqcnt=Irq_GetCount(INT_TIM1CAPT);
}

void main(void)
{
    byte n;

    // Define Interrupt Service Routine
    Irq_SetVect(INT_TIM1CAPT,Timer1_ISR);

    Timer_T0PWM(100,PS0_64);   // Start Pulse Generator Timer 0

 // Measurement starts here
 // Output Timer0 OC0(PortB.3) connect to ICP(input capture pin, PortD.6)

    PM_Value=0;
 // Set mode to Pulse Width Measurement and determine prescaler
    Timer_T1PM(0,PS_64);

    while(PM_Value==0);     // Measure Pulse Width or Period

    Msg_WriteHex(PM_Value); // Output Measuring Value
}
```

➡ For reason of better survey only a simplified version is shown here. Because of a collision on Pin B.4 *Timer_0* is used for pulse generation with Mega128. The entire program can be found in directory PW_Measurement.

6.23.6 Timer Functions

In C-Control Pro Mega 32 there are two, in Mega128 three independent Timer available. These are *Timer_0* with 8 bit and *Timer_1* with 16 bit (*Timer_3* with 16 bit for Mega128 only). The timer have a programmable prescaler (see **Table**). After the defined period the timer will trigger an interrupt. An interrupt routine can then be used to execute specific actions.

Timer_T0Time (8 Bit)

Example: Timer0: Switch output on with a delay of 6,94 ms (100x 69,44 µs, see Table)

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

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

void main(void)
{
    Port_DataDirBit(0,0);               // PortA.0 Output
    Port_WriteBit(0,0);                 // PortA.0 Output=0
    Irq_SetVect(INT_TIM0COMP,Timer0_ISR);// define Interrupt Service Routine
    Timer_T0Time(100,PS0_1024);         // set time and start Timer0
    //  other program code....
}
```

6.23.7 Timer_Disable

Timer Functions

Syntax

```
void Timer_Disable(byte timer);

Sub Timer_Disable(timer As Byte)
```

Description

This function disables the specified timer. Timer functions occupy I/O ports. If a timer is not needed and the corresponding I/O ports are used otherwise, the timer must be disabled.

Parameter

0 = *Timer_0*
1 = *Timer_1*
3 = *Timer_3* (only Mega128)

6.23.8 Timer_T0CNT

Timer Functions

Syntax

void Timer_T0CNT(**void**);

Sub Timer_T0CNT()

Description

These function initializes Counter0. Counter0 gets incremented at every positive signal edge at Input **Mega32**:T0 (PIN1).

➡ Due to hardware reasons it is not possible to use *Timer_0* as a counter in the **Mega128!**

Parameter

None

6.23.9 Timer_T0FRQ

Timer Functions

Syntax

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

Sub Timer_T0FRQ(period **As Byte**,PS **As Byte**)

Description

This function initializes Timer0 for frequency generation. Parameters are period duration and prescaler, see table. The output signal is generated at **Mega32**: PortB.3 (PIN4), **Mega128**: PortB.4 (X1_4). The frequency generation is started automatically. There is a extended prescaler definition for the Mega128, see table.

Parameter

period   period duration
PS   prescaler

Table prescaler:

Mega32

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	135,6 ns

PS0_8 (2)	1,085 µs
PS0_64 (3)	8,681 µs
PS0_256 (4)	34,72 µs
PS0_1024 (5)	138,9 µs

Mega128

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	135,6 ns
PS0_8 (2)	1,085 µs
PS0_32 (3)	4,340 µs
PS0_64 (4)	8,681 µs
PS0_128 (5)	17,36 µs
PS0_256 (6)	34,72 µs
PS0_1024 (7)	138,9 µs

6.23.10 Timer_T0GetCNT

Timer Functions

Syntax

byte Timer_T0GetCNT(**void**);

Sub Timer_T0GetCNT() **As Byte**

Description

The value of Counter0 is read. If there was an overflow a value of 0xff is returned.

➡ Due to hardware reasons it is not possible to use *Timer_0* as a counter in the **Mega128**!

Return Parameter

counter value

6.23.11 Timer_T0PW

Timer Functions

Syntax

void Timer_T0PW(**byte** PW);

Sub Timer_T0PW(PW **As Byte**)

Description

This function sets a new pulse width for Timer0 without changing the prescaler.

Parameter

PW pulse width

6.23.12 Timer_T0PWM

Timer Functions

Syntax

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

Sub Timer_T0PWM(PW **As Byte,**PS **As Byte**)

Description

This function initializes Timer0 with given prescaler and pulse width, see table. The output signal is generated at **Mega32**: PortB.3 (PIN4), **Mega128**: PortB.4 (X1_4). There is an extended prescaler definition for the Mega128, see table.

Parameter

PW pulse width
PS prescaler

Table prescaler:

Mega32

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_64 (3)	4,34 µs
PS0_256 (4)	17,36 µs
PS0_1024 (5)	69,44 µs

Mega128

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

6.23.13 Timer_T0Start

Timer Functions

Syntax

void Timer_T0Start(**byte** prescaler);

Sub Timer_T0Start(prescaler **As Byte**)

Description

The frequency generation is started with the already set parameters. The prescaler must be given again.

Parameter

prescaler   prescaler (see table)

6.23.14 Timer_T0Stop

Timer Functions

Syntax

void Timer_T0Stop(**void**);

Sub Timer_T0Stop()

Description

The frequency generation gets stopped. The output signal can be 0 or 1, dependent on the last state. Only the clock generation is stopped, all other settings stay the same.

Parameter

None

6.23.15 Timer_T0Time

Timer Functions

Syntax

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

Sub Timer_T0Time(Time **As Byte**,PS **As Byte**)

Description

This function initializes Timer_0 with a prescaler and a timer interval value, see table. After the timing interval is expired The Timer_0 Interrupt (INT_TIM0COMP) is triggered. There is an extended prescaler definition for the Mega128, see table.

Parameter

Time   time period after that the interrupt is triggered
PS   prescaler

Table prescaler:

Mega32

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_64 (3)	4,34 µs
PS0_256 (4)	17,36 µs
PS0_1024 (5)	69,44 µs

Mega128

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

6.23.16 Timer_T1CNT

Timer Functions

Syntax

```
void Timer_T1CNT(void);

Sub Timer_T1CNT()
```

Description

These function initializes Counter1. Counter1 gets incremented at every positive signal edge at Input **Mega32**: PortB.1 (PIN2) **Mega128**: PortD.6 (X2_15).

Parameter

None

6.23.17 Timer_T1CNT_Int

Timer Functions

Syntax

void Timer_T1CNT_Int(**word** limit);

Sub Timer_T1CNT_Int(limit **As Word**)

Description

These function initializes Counter1. Counter1 gets incremented at every positive signal edge at Input **Mega32**: PortB.1 (PIN2) **Mega128**: PortD.6 (X2_15). After the limit is reached an interrupt ("Timer1 CompareA" - define: INT_TIM1CMPA) is triggered. An appropriate Interrupt Service Routine must be specified.

Parameter

limit

6.23.18 Timer_T1FRQ

Timer Functions

Syntax

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

Sub Timer_T1FRQ(period **As Word**,PS **As Byte**)

Description

This function initializes Timer1 for frequency generation. Parameters are period duration and prescaler, see table. The output signal is generated at **Mega32**: PortD.5 (PIN19). **Mega128**: PortB.5 (X1_3). The frequency generation is started automatically. There is an extended prescaler definition for the Mega128, see table.

Parameter

period period duration
PS   prescaler

Table prescaler:

Mega32

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	135,6 ns
PS0_8 (2)	1,085 µs
PS0_64 (3)	8,681 µs
PS0_256 (4)	34,72 µs
PS0_1024 (5)	138,9 µs

Mega128

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	135,6 ns
PS0_8 (2)	1,085 µs
PS0_32 (3)	4,340 µs
PS0_64 (4)	8,681 µs
PS0_128 (5)	17,36 µs
PS0_256 (6)	34,72 µs
PS0_1024 (7)	138,9 µs

6.23.19 Timer_T1FRQX

Timer Functions

Syntax

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

Sub Timer_T1FRQX(period As Word,skew As Word,PS As Byte)
```

Description

This function initializes Timer1 for frequency generation. Parameters are period duration, prescaler and phase shift,see table. The output signal is generated at **Mega32**: PortD.5 (PIN19). **Mega128**: PortB.5 (X1_3). The frequency generation is started automatically. There is an extended prescaler definition for the Mega128, see table. The phase shift must be smaller than half the period.

Parameter

period   period duration
skew   phase shift
PS   prescaler (table prescaler)

6.23.20 Timer_T1GetCNT

Timer Functions

Syntax

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

Sub Timer_T1GetCNT() **As Word**

Description

The value of Counter1 is read. If there was an overflow a value of 0xffff is returned.

Return Parameter

counter value

6.23.21 Timer_T1GetPM

Timer Functions

Syntax

word Timer_T1GetPM(void);

Sub Timer_T1GetPM() **As Word**

Description

Returns the result of the measurement.

Parameter

None

Return Parameter

result of measurement

➡ To calculate the correct value, the 16bit result is multiplied with the entry of the prescaler Table that was passed in the call to Timer_T1PM.

6.23.22 Timer_T1PWA

Timer Functions

Syntax

void Timer_T1PWA(**word** PW0);

Sub Timer_T1PWA(PW0 **As Word**)

Description

This function sets a new pulse width (Channel A) for Timer1 without changing the prescaler.

Parameter

<u>PW0</u> pulse width

6.23.23 Timer_T1PM

Timer Functions

Syntax

void Timer_T1PM(**byte** <u>Mode</u>,**byte** <u>PS</u>);

void Timer_T1PM(<u>Mode</u> **As Byte,**<u>PS</u> **As Byte**)

Description

This function defines if pulse width measurement or period measurement should be done. Then it initializes *Timer_1* and sets the prescaler.

Parameter

<u>Mode</u>   0 = pulse width measurement, 1 = period measurement
<u>PS</u>   prescaler

Table <u>prescaler</u>:

Mega32

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_64 (3)	4,34 µs
PS0_256 (4)	17,36 µs
PS0_1024 (5)	69,44 µs

Mega128

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

6.23.24 Timer_T1PWB

Timer Functions

Syntax

void Timer_T1PWB(**word** PW1);

Sub Timer_T1PWB(PW1 **As Word**)

Description

This function sets a new pulse width (Channel B) for Timer1 without changing the prescaler.

Parameter

PW1   pulse width

6.23.25 Timer_T1PWM

Timer Functions

Syntax

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

Sub Timer_T1PWM(period **As Word,**PW0 **As Word,**PS **As Byte**)

Description

This function initializes *Timer_1* with given period duration, pulse width and prescaler, see table. The output signal is generated at **Mega32**: PortD.5 (PIN19), **Mega128**: PortB.5 (X1_3). There is an extended prescaler definition for the Mega128, see table.

Parameter

period   period duration
PW0   pulse width
PS   prescaler

Table prescaler:

Mega32

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_64 (3)	4,34 µs
PS0_256 (4)	17,36 µs

PS0_1024 (5)	69,44 µs

Mega128

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

6.23.26 Timer_T1PWMX

Timer Functions

Syntax

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

Sub Timer_T1PWMX(period **As Word,**PW0 **As Word,**PW1 **As Word,**PS **As Byte**)

Description

This function initializes *Timer_1* with given period duration, prescaler, pulse width for channel A and B. The output signal is generated at
Mega32: PortD.4 (PIN18) and PortD.5 (PIN19). **Mega128**: PortB.5 (X1_3) and PortB.6 (X1_2).

Parameter

period   period duration
PW0   pulse width channel A
PW1   pulse width channel B
PS   prescaler (see table prescaler)

6.23.27 Timer_T1PWMY

Timer Functions

Syntax

void Timer_T1PWMY(**word** period,**word** PW0,**word** PW1,**word** PW2,**byte** PS);

Sub Timer_T1PWMY(period **As Word,**PW0 **As Word,**PW1 **As Word,**PW2 **As Word,**PS **As Byte**)

Description

This function initializes *Timer_1* with given period duration, prescaler, pulse width for channel A, B and C. The output signal is generated at
PortB.5 (X1_3) , PortB.6 (X1_2) and PortB.7 (X1_1).

Parameter

period   period duration
PW0   pulse width channel A
PW1   pulse width channel B
PW2   pulse width channel C
PS   prescaler (see table prescaler)

6.23.28 Timer_T1Start

Timer Functions

Syntax

void Timer_T1Start(**byte** prescaler);

Sub Timer_T1Start(prescaler **As Byte**)

Description

The frequency generation is started with the already set parameters. The prescaler must be given again.

Parameter

prescaler   prescaler (see table)

6.23.29 Timer_T1Stop

Timer Functions

Syntax

void Timer_T1Stop(**void**);

Sub Timer_T1Stop()

Description

The frequency generation gets stopped. The output signal can be 0 or 1, dependent on the last state. Only the clock generation is stopped, all other settings stay the same.

Parameter

None

6.23.30 Timer_T1Time

Timer Functions

Syntax

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

Sub Timer_T1Time(Time **As Word**,PS **As Byte**)

Description

This function initializes *Timer_1* with a prescaler and a timer interval value (16bit), see table. After the timing interval is expired *Timer_1* Interrupt (INT_TIM1CMPA) is triggered. There is an extended prescaler definition for the Mega128, see table.

Parameter

Time   time period after that the interrupt is triggered
PS   prescaler

Table prescaler:

Mega32

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_64 (3)	4,34 µs
PS0_256 (4)	17,36 µs
PS0_1024 (5)	69,44 µs

Mega128

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

6.23.31 Timer_T3CNT

Timer Functions

Syntax

void Timer_T3CNT(**void**);

Sub Timer_T3CNT()

Description

These function initializes Counter3. Counter3 gets incremented at every positive signal edge at Input PortE.6 (X1_10)

Parameter

None

6.23.32 Timer_T3CNT_Int

Timer Functions

Syntax

void Timer_T3CNT_Int(**word** limit);

Sub Timer_T3CNT_Int(limit **As Word**)

Description

These function initializes *Counter_3*. *Counter_3* gets incremented at every positive signal edge at Input PortE.6 (X1_10). After the limit is reached an interrupt ("Timer3 CompareA" - define: INT_TIM3CMPA) is triggered. An appropriate Interrupt Service Routine must be specified.

Parameter

limit

6.23.33 Timer_T3FRQ

Timer Functions

Syntax

void Timer_T3FRQ(**word** period,**byte** PS);

Sub Timer_T3FRQ(period **As Word**,PS **As Byte**)

Description

This function initializes Timer3 for frequency generation. Parameters are period duration and prescaler, see table. The output signal is generated at PortE.3 (X1_13). The frequency generation is started automatically..

Parameter

period   period duration
PS   prescaler

Table prescaler:

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	135,6 ns
PS0_8 (2)	1,085 µs
PS0_32 (3)	4,340 µs
PS0_64 (4)	8,681 µs
PS0_128 (5)	17,36 µs
PS0_256 (6)	34,72 µs
PS0_1024 (7)	138,9 µs

6.23.34 Timer_T3FRQX

Timer Functions

Syntax

void Timer_T3FRQX(**word** period**,word** skew**,byte** PS);

Sub Timer_T3FRQX(period **As Word**,skew **As Word**,PS **As Byte**)

Description

This function initializes Timer3 for frequency generation. Parameters are period duration, prescaler and phase shift,see table. The output signal is generated at PortE.3 (X1_13) und PortE.4 (X1_12). The frequency generation is started automatically. There is an extended prescaler definition for the Mega128, see table. The phase shift must be smaller than half the period.

Parameter

period   period duration
skew   phase shift
PS   prescaler (table prescaler)

6.23.35 Timer_T3GetCNT

Timer Functions

Syntax

word Timer_T3GetCNT(**void**);

Sub Timer_T3GetCNT() **As Word**

Description

The value of Counter1 is read. If there was an overflow a value of 0xffff is returned.

Return Parameter

counter value

6.23.36 Timer_T3GetPM

Timer Functions

Syntax

word Timer_T3GetPM(void);

Sub Timer_T3GetPM() **As Word**

Description

Returns the result of the measurement.

Parameter

None

Return Parameter

result of measurement

➡️ To calculate the correct value, the 16bit result is multiplied with the entry of the prescaler Table that was passed in the call to Timer_T3PM.

6.23.37 Timer_T3PWA

Timer Functions

Syntax

void Timer_T3PWA(**word** PW0);

Sub Timer_T3PWA(PW0 **As Word**)

Description

This function sets a new pulse width (Channel A) for Timer3 without changing the prescaler.

Parameter

<u>PW0</u>   pulse width

6.23.38 Timer_T3PM

Timer Functions

Syntax

void Timer_T3PM(**byte** <u>Mode</u>,**byte** <u>PS</u>);

void Timer_T3PM(<u>Mode</u> **As Byte,**<u>PS</u> **As Byte**)

Description

This function defines if pulse width measurement or period measurement should be done. Then it initializes *Timer_3* and sets the prescaler.

Parameter

<u>Mode</u>   0 = pulse width measurement, 1 = period measurement
<u>PS</u>   prescaler

Table <u>prescaler</u>:

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

6.23.39 Timer_T3PWB

Timer Functions

Syntax

void Timer_T3PWB(**word** PW1);

Sub Timer_T3PWB(PW1 **As Word**)

Description

This function sets a new pulse width (Channel B) for Timer3 without changing the prescaler.

Parameter

PW1 pulse width

6.23.40 Timer_T3PWM

Timer Functions

Syntax

void Timer_T3PWM(**word** period,**word** PW0,**byte** PS);

Sub Timer_T3PWM(period **As Word,**PW0 **As Word,**PS **As Byte**)

Description

This function initializes *Timer_3* with given period duration, pulse width and prescaler, see table. The output signal is generated at PortE.3 (X1_13).

Parameter

period period duration
PW0 pulse width
PS prescaler

Table prescaler:

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

6.23.41 Timer_T3PWMX

Timer Functions

Syntax

void Timer_T3PWMX(**word** period,**word** PW0,**word** PW1,**byte** PS);

Sub Timer_T3PWMX(period **As Word,**PW0 **As Word,**PW1 **As Word,**PS **As Byte**)

Description

This function initializes *Timer_3* with given period duration, prescaler, pulse width for channel A and B. The output signal is generated at
 PortE.3 (X1_13) and PortE.4 (X1_12).

Parameter

period	period duration
PW0	pulse width channel A
PW1	pulse width channel B
PS	prescaler (see table prescaler)

6.23.42 Timer_T3PWMY

Timer Functions

Syntax

void Timer_T3PWMY(**word** period,**word** PW0,**word** PW1,**word** PW2,**byte** PS);

Sub Timer_T3PWMY(period **As Word,**PW0 **As Word,**PW1 **As Word,**PW2 **As Word,**PS **As Byte**)

Description

This function initializes *Timer_3* with given period duration, prescaler, pulse width for channel A, B and C. The output signal is generated at
PortE.3 (X1_13), PortE.4 (X1_12) and PortE.5 (X1_11).

Parameter

period	period duration
PW0	pulse width channel A
PW1	pulse width channel B
PW2	pulse width channel C
PS	prescaler (see table prescaler)

6.23.43 Timer_T3Start

Timer Functions

Syntax

void Timer_T3Start(**byte** prescaler);

Sub Timer_T3Start(prescaler **As Byte**)

Description

The frequency generation is started with the already set parameters. The prescaler must be given again.

Parameter

prescaler   prescaler (see table)

6.23.44 Timer_T3Stop

Timer Functions

Syntax

void Timer_T3Stop(**void**);

Sub Timer_T3Stop()

Description

The frequency generation gets stopped. The output signal can be 0 or 1, dependent on the last state. Only the clock generation is stopped, all other settings stay the same.

Parameter

None

6.23.45 Timer_T3Time

Timer Functions

Syntax

void Timer_T3Time(**word** Time,**byte** PS);

Sub Timer_T3Time(Time **As Word**,PS **As Byte**)

Description

This function initializes *Timer_3* with a prescaler and a timer interval value (16bit), see table. After the timing interval is expired *Timer_3* Interrupt ([INT_TIM3CMPA](#)) is triggered.

Parameter

Time   time period after that the interrupt is triggered
PS   prescaler

Table prescaler:

Prescaler	Time Base (duration of a tick)
PS0_1 (1)	67,8 ns
PS0_8 (2)	542,5 ns
PS0_32 (3)	2,17 µs
PS0_64 (4)	4,34 µs
PS0_128 (5)	8,68 µs
PS0_256 (6)	17,36 µs
PS0_1024 (7)	69,44 µs

6.23.46 Timer_TickCount

Timer Functions

Syntax

```
word Timer_TickCount(void);

Sub Timer_TickCount() As Word
```

Description

Measures the number of 10ms ticks between two calls of Timer_TickCount(). Ignore the return value of the first call to Timer_TickCount(). If the delay between the two calls is greater than 655.36 seconds, the result is undefined.

Parameter

None

Return Parameter

time interval expressed in 10ms ticks

Example

```
void main(void)
```

```
{
    word time;
    Timer_TickCount();
    AbsDelay(500);  // wait 500 ms
    time=Timer_TickCount();  // the value should be 50
}
```

Part

7

7 FAQ

<u>Problems</u>

1. No USB connection existing to the Application Board.

- Has the FTDI USB driver been loaded onto the PC? Or does "Unknown Device" appear in the Hardware Manager, when the USB connector is plugged in?
- Has the correct communication port been set in Options->IDE->Interfaces?
- Is a Windows version prior to Windows 98 SE ("*Second Edition*") being used? The Microsoft USB drivers do not reliably work with USB devices prior to Win98SE.
- Are the ports **M32**:B.4-B.7,A.6-A.7 resp. **M128**:B.0-B.4,E.5 erroneously being used in the software (see pin assignment of M32 and M128)? Are the jumpers on the Application Board set to these ports?
- A signal on **M32**:PortD.2 resp. **M128**:PortE.4 (SW1) during startup will activate the serial Bootloader.
- (Mega128 only) Is Port.G4 (LED2) on Low during Reset? See SPI Switch Off in chapter "Firmware".

2. The serial interface does not issue any characters or does not receive any characters.

- Are the Ports D.0-D.1 erroneously used in the software (see pin assignment of M32 and M128)? Are the jumpers on the Application Board set to these ports?

3. The Application Board does not react to any commands when serially connected.

- In order to get the Bootloader into the serial mode the button SW1 must be pressed during startup of the Application Board (observe jumper for SW1). For the serial mode **M32**:PortD.2 resp. **M128**: PortE.4 (SW1) can also be fixed to GND level.

4. The Hardware Application does not start by itself (Autostart Behaviour).

- A signal on the SPI interface during startup may activate USB communication.
- A signal on **M32**:PortD.2 resp. **M128**:PortE.4 (SW1) during startup may activate the serial Bootloader.

5. The key assignment of the editor "**xyz**" has been set but some keyboard commands do not function.

- The possibility to switch on the key assignment of a specific editor in the IDE is only an approximation. Sometimes it is too expensive to support the corresponding functions in a "foreign" editor, some other time the keyboard commands can collide with the keyboard shortcuts in the IDE.

6. The spelling check does not function.

- Is the spelling check switched on in Options->Editor?
- The spelling check does only display spelling errors in the commentaries. The check of any other

area would not make sense.

7. Where can be determined whether the new project is a BASIC or C project?

- There is no difference in project type. The source text files in a project determine which programming language is being used. Files with the extension *.cc will run in a CompactC context, Files with the extension *.cbas will be translated into BASIC. Also C and BASIC can be combined in a project.

8. I am using an LCD other than the one shipped with the product, but am using the same Controller. The cursor positions do not work correctly.

- The Controller can display 4 lines at 32 characters each. The beginnings of the lines are stored transposed in memory following the scheme below:

Value of <u>pos</u>	**Position in the display**
0x00-0x1f	0-31 in the line 1
0x40-0x5f	0-31 in the line 2
0x20-0x3f	0-31 in the line 3
0x60-0x6f	0-31 in the line 4

9. How much RAM do I have for my programs?

- There are 930 bytes left for own programs on the Mega32, on the Mega128 remain 2494 bytes. Interpreter and Debugger are using buffer for interrupt driven I/O, and 256 bytes for the data stack. Beside this resources, there are some internal tables, that are needed for interrupt handling and multitasking. Additionally some RAM Variables are used from library functions.

10. Where is the second serial interface on the Mega128 Application Board?

- See J4 chapter <u>Jumper Application Board</u> M128.

11. I need no USB connection to the application board, how can I reclaim the reserved ports for USB?

- The USB interface is wired to the C-Control module over the SPI interface. The SPI interface can be disabled with <u>SPI_Disable</u>(). Do not forget to remove the jumper that connects the SPI with the Mega8 (USB interface) on the application board.

12. Where do I have the supply voltage on the breadboard?

- If you turn the application board to a position where the interface connectors (USB and serial) show to the upper side, the leftmost column on the breadboard is GND and the rightmost column is VCC. You can see it clearly, when you take a look of the backside of the board.

13. I need more ports for my hardware application. Many ports are used by other functions.

- Take a look at the Pin Assignment of M32 and M128. You can use all ports that have no special functionality (SPI, RS232, LCD, Keyboard etc.) that is needed for your application. Do not forget to remove the jumper that connects the port pins to the application board. Otherwise the behaviour can be undetermined.

14. How can I switch on the Pull-Up resistor of a port?

- First switch the port to input with PortDataDir() (or PortDataDirBit()), then use PortWrite() (or PortWriteBit()) to write a "1" into the port.

15. Where are the demo programs located?

- Due to Vista Compatibility the demo programs are installed to "\Documents and Settings\All Users\Documents" (XP and earlier) or to "\Users\Public\Public Documents" (Vista) directory. When replacing an old installation, the old "Demos" directory is deleted. Therefore please create own programs outside of the "C-Control Pro Demos" directory.

16. Can I program the C-Control Pro Module in Linux?

- There is no native IDE for Linux, but customer had sucessfully started the IDE under Wine und programmed the module in serial mode.

17. Is it possible to develop for C-Control Pro with other Compilers?

- There are many developing systems for the Atmel Mega CPU. Some of these Compilers are commercial, others a free. A good example of a free development system is the GNU C-Compiler. You can transfer programs, that you wrote with the GNU C-Compiler, to the Atmel Mega CPU with a AVR ISP programmer. But once you overwrote the installed bootloader, there is no way back, you cannot longer use the C-Control Pro software.

Index

- - -

- # -

- + -

- A -

- B -

- C -

Transfer   65
Type Conversion   91, 112

- U -

unsigned char   91
unsigned int   91
USB   10
USB Interface   78

- V -

Variables   92, 112
Variables Window   70
Version Check   67
Visibility of Variables   92, 112
void   104

- W -

Warranty   3
While   103, 119
Window   81
Word   91, 112