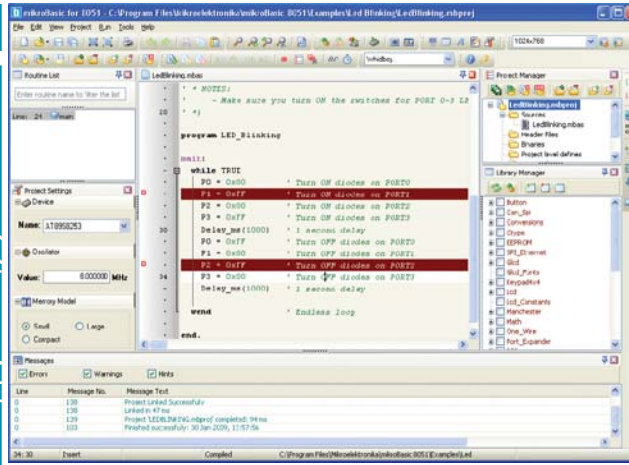
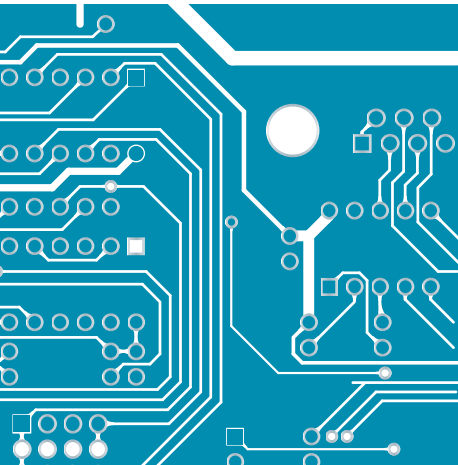


mikroBASIC for 8051



USER'S MANUAL

Develop your applications quickly and easily with the world's most intuitive mikroBASIC for 8051 Microcontrollers.

Highly sophisticated IDE provides the power you need with the simplicity of a Windows based point-and-click environment.

With useful implemented tools, many practical code examples, broad set of built-in routines, and a comprehensive Help, mikroBASIC for 8051 makes a fast and reliable tool, which can satisfy needs of experienced engineers and beginners alike.

December 2008.

Reader's note

DISCLAIMER:

mikroBasic for 8051 and this manual are owned by mikroElektronika and are protected by copyright law and international copyright treaty. Therefore, you should treat this manual like any other copyrighted material (e.g., a book). The manual and the compiler may not be copied, partially or as a whole without the written consent from the mikroElektronika. The PDF-edition of the manual can be printed for private or local use, but not for distribution. Modifying the manual or the compiler is strictly prohibited.

HIGH RISK ACTIVITIES:

The mikroBasic for 8051 compiler is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). mikroElektronika and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

LICENSE AGREEMENT:

By using the mikroBasic for 8051 compiler, you agree to the terms of this agreement. Only one person may use licensed version of mikroBasic for 8051 compiler at a time. Copyright © mikroElektronika 2003 - 2008.

This manual covers mikroBasic for 8051 version 1.1 and the related topics. Newer versions may contain changes without prior notice.

COMPILER BUG REPORTS:

The compiler has been carefully tested and debugged. It is, however, not possible to guarantee a 100 % error free product. If you would like to report a bug, please contact us at the address office@mikroe.com. Please include next information in your bug report:

- Your operating system
- Version of mikroBasic for 8051
- Code sample
- Description of a bug

CONTACT US:

mikroElektronika
Voice: + 381 (11) 36 28 830
Fax: + 381 (11) 36 28 831
Web: www.mikroe.com
E-mail: office@mikroe.com

Windows is a Registered trademark of Microsoft Corp. All other trade and/or services marks are the property of the respective owners.

Table of Contents

CHAPTER 1	Introduction
CHAPTER 2	mikroBasic for 8051 Environment
CHAPTER 3	mikroBasic for 8051 Specifics
CHAPTER 4	8051 Specifics
CHAPTER 5	mikroBasic for 8051 Language Reference
CHAPTER 6	mikroBasic for 8051 Libraries

CHAPTER 1

Features	2
Where to Start	3
mikroElektronika Associates License Statement and Limited Warranty 4	
IMPORTANT - READ CAREFULLY	4
LIMITED WARRANTY	5
HIGH RISK ACTIVITIES	6
GENERAL PROVISIONS	6
Technical Support	7
How to Register	8
Who Gets the License Key	8
How to Get License Key	8
After Receiving the License Key	10

CHAPTER 2

IDE Overview	12
Main Menu Options	13
File Menu Options	14
Edit Menu Options	15
Find Text	16
Replace Text	17
Find In Files	17
Go To Line	18
Replace Text	18
Dialog box for	18
Regular expressions	19
View Menu Options	20
Toolbars	21
File Toolbar	21
Edit Toolbar	21
Edit Toolbar is	21
Advanced Edit Toolbar	22
Find/Replace Toolbar	22

Project Toolbar	23
Build Toolbar	23
Debugger	24
Styles Toolbar	25
Tools Toolbar	25
Project Menu Options	26
Run Menu Options	28
Tools Menu Options	29
Help Menu Options	30
Keyboard Shortcuts	31
IDE Overview	33
Customizing IDE Layout	34
Docking Windows	34
Saving Layout	35
Auto Hide	36
Advanced Code Editor	37
Advanced Editor Features	37
Code Assistant	39
Code Folding	39
Parameter Assistant	40
Code Templates (Auto Complete)	40
Auto Correct	41
Spell Checker	41
Bookmarks	41
Goto Line	41
Comment / Uncomment	41
Code Explorer	42
Routine List	43
Project Manager	44
Project Settings Window	46
Library Manager	47
Error Window	49
Statistics	50
Memory Usage Windows	50
RAM Memory	50

Data Memory	50
XData Memory	51
iData Memory	51
bData Memory	52
PData Memory	52
Displays PData	52
Special Function Registers	53
General Purpose Registers	53
ROM Memory	54
ROM Memory Usage	54
ROM Memory Allocation	54
Procedures Windows	55
Procedures Size Window	55
Procedures Locations Window	56
Integrated Tools	57
USART Terminal	57
ASCII Chart	57
EEPROM Editor	59
7 Segment Display Decoder	60
UDP Terminal	60
Graphic LCD Bitmap Editor	62
LCD Custom Character	63
Options	64
Code editor	64
Tools	64
Output settings	66
Regular Expressions	67
Introduction	67
Simple matches	67
Escape sequences	67
Character classes	68
Metacharacters	68
Metacharacters - Line separators	68
Metacharacters - Predefined classes	69
Metacharacters - Word boundaries	69
Metacharacters - Iterators	70

Metacharacters - Alternatives	71
Metacharacters - Subexpressions	71
Metacharacters - Backreferences	71
mikroBasic for 8051 Command Line Options	72
Projects	73
New Project	73
New Project Wizard Steps	73
Customizing Projects	77
Edit Project	77
Managing Project Group	77
Add/Remove Files from Project	77
Source Files	78
Managing Source Files	78
Creating new source file	78
Opening an existing file	79
Printing an open file	79
Saving file	79
Saving file under a different name	79
Closing file	79
Clean Project Folder	80
Clean Project Folder	80
Compilation	81
Output Files	81
Assembly View	81
Error Messages	82
Compiler Error Messages:	82
Linker Error Messages:	85
Hint Messages:	85
Software Simulator Overview	86
Watch Window	86
Stopwatch Window	88
RAM Window	89
Software Simulator Options	90
Creating New Library	91
Multiple Library Versions	91

CHAPTER 3

Basic Standard Issues	94
Divergence from the Basic Standard	94
Basic Language Exstensions	94
Predefined Globals and Constants	95
SFRs and related constants	95
Math constants	95
Accessing Individual Bits	96
Accessing Individual Bits Of Variables	96
sbit type	96
bit type	97
Interrupts	98
Function Calls from Interrupt	98
Interrupt Priority Level	98
Linker Directives	99
Directive absolute	99
Directive org	99
Built-in Routines	100
Lo	101
Hi	101
Higher	101
Highest	102
Inc	102
Dec	102
Delay_us	103
Delay_ms	103
Vdelay_ms	103
Delay_Cyc	104
Clock_KHz	104
Clock_MHz	104
SetFuncCall	105
Uart_Init	105
Code Optimization	106

Constant folding	106
Constant propagation	106
Copy propagation	106
Value numbering	106
"Dead code" elimination	106
Stack allocation	106
Local vars optimization	106
Better code generation and local optimization	106
Types Efficiency	107

CHAPTER 4

Nested Calls Limitations	108
8051 Memory Organization	108
Program Memory (ROM)	108
Internal Data Memory	109
External Data Memory	110
SFR Memory	110
Memory Models	111
Small model	111
Compact model	111
Large model	112
Memory Type Specifiers	112
code	113
data	113
idata	113
bdata	113
xdata	114
pdata	114

CHAPTER 5

mikroBasic Language Reference	116
Lexical Elements Overview	118
Whitespace	118

Newline Character	118
Whitespace in Strings	119
Comments	119
Tokens	120
Token Extraction Example	120
Literals	121
Integer Literals	121
Floating Point Literals	121
Character Literals	122
String Literals	122
Keywords	123
Identifiers	128
Case Sensitivity	128
Uniqueness and Scope	128
Identifier Examples	129
Punctuators	129
Brackets	129
Parentheses	130
Comma	130
Colon	130
Dot	130
Program Organization	131
Organization of Main Module	131
Organization of Other Modules	132
Scope and Visibility	134
Scope	134
Visibility	134
The visibili	134
Modules	135
Include Clause	135
Main Module	136
Other Modules	136
Interface Section	137
Implementation Section	137
Variables	138

Variables and 8051	138
Constants	139
Labels	140
Symbols	141
Functions and Procedures	142
Functions	142
Calling a function	142
Example	143
Procedures	143
Calling a procedure	143
Example	144
Function Pointers	144
Example:	144
Example:	145
Forward declaration	146
Types	147
Type Categories	147
Simple Types	148
Arrays	149
Array Declaration	149
Constant Arrays	149
Strings	150
Note	150
Pointers	151
@ Operator	151
Structures	152
Structure Member Access	153
Types Conversions	154
Implicit Conversion	154
Promotion	154
Clipping	154
Explicit Conversion	155
Operators	156
Operators Precedence and Associativity	156
Arithmetic Operators	156

Division by Zero	157
Unary Arithmetic Operators	157
Relational Operators	158
Relational Operators in Expressions	158
Bitwise Operators	159
Bitwise Operators Overview	159
Logical Operations on Bit Level	159
The bitwise operators and, or, an	159
Unsigned and Conversions	160
Signed and Conversions	160
Bitwise Shift Operators	161
Boolean Operators	162
Expressions	163
Statements	164
Assignment Statements	164
Conditional Statements	164
If Statement	165
Nested if statements	165
Select Case Statement	166
Nested Case Statements	167
Iteration Statements	168
For Statement	169
Endless Loop	169
While Statement	170
Do Statement	171
Jump Statements	171
asm Statement	172
Directives	173
Compiler Directives	173
Directives #DEFINE and #UNDEFINE	173
Directives #IFDEF, #ELSEIF and #ELSE	173
Predefined Flags	174
Linker Directives	175
Directive absolute	175
Directive org	175

CHAPTER 6

Hardware 8051-specific Libraries	178
Miscellaneous Libraries	178
Library Dependencies	179
CANSPI Library	181
The SPI module i	181
External dependencies of CANSPI Library	182
Library Routines	182
CANSPISetOperationMode	182
CANSPIGetOperationMode	182
CANSPIInitialize	182
CANSPISetBaudRate	182
CANSPISetMask	182
CANSPISetFilter	182
CANSPIread	182
CANSPIwrite	182
The following routines are for an internal use by the lib	182
CANSPISetOperationMode	183
CANSPIGetOperationMode	183
CANSPIInitialize	184
CANSPISetBaudRate	186
CANSPISetMask	187
CANSPISetFilter	188
CANSPIRead	189
CANSPIWrite	190
CANSPI Constants	191
CANSPI_OP_MODE	191
CANSPI_CONFIG_FLAGS	191
CANSPI_TX_MSG_FLAGS	192
CANSPI_RX_MSG_FLAGS	193
CANSPI_MASK	193
CANSPI_FILTER	193
Library Example	194
HW Connection	197

EEPROM Library	198
Library Routines	198
Eeprom_Read	198
Eeprom_Write	199
Eeprom_Write_Block	200
Library Example	201
Graphic LCD Library	202
The mikroBasic for 80	202
of Graphic LCD Library	202
Library Routines	202
Basic routines:	202
Glcd_Init	202
Glcd_Set_Side	202
Glcd_Set_X	202
Glcd_Set_Page	202
Glcd_Read_Data	202
Glcd_Write_Data	202
Glcd_Init	203
Glcd_Set_Side	204
Glcd_Set_X	204
Glcd_Set_Page	205
Glcd_Read_Data	205
Glcd_Write_Data	206
Glcd_Fill	206
Glcd_Dot	207
Glcd_Line	207
Glcd_V_Line	208
Glcd_H_Line	208
Glcd_Rectangle	209
Glcd_Box	210
Glcd_Circle	210
Glcd_Set_Font	211
Glcd_Write_Char	212
Glcd_Write_Text	213
Glcd_Image	214
Library Example	214

The following example	214
HW Connection	216
Keypad Library	217
External dependencies of Keypad Library	217
Library Routines	217
Keypad_Init	217
Keypad_Key_Press	218
Keypad_Key_Click	218
Library Example	218
This is a simple example of using	218
the Keypad Library	218
HW Connection	221
LCD Library	222
External dependencies of LCD Library	222
Library Routines	222
Lcd_Init	223
Lcd_Out	224
Lcd_Out_Cp	224
Lcd_Chr	225
Lcd_Chr_Cp	225
Lcd_Cmd	226
Available LCD Commands	226
Library Example	227
OneWire Library	229
External dependencies of OneWire Library	229
Library Routines	229
Ow_Reset	230
Ow_Read	230
Ow_Write	231
Library Example	231
HW Connection	234
Manchester Code Library	235
Library Routines	235
Man_Receive_Init	236
Man_Receive	236

Man_Send_Init	237
Man_Send	237
Man_Synchro	238
Man_Out	238
Library Example	239
Connection Example	241
Port Expander Library	242
External dependencies of Port Expander Library	242
Library Routines	242
Expander_Init	243
Expander_Read_Byte	244
Expander_Write_Byte	244
Expander_Read_PortA	245
Expander_Read_PortB	245
Expander_Read_PortAB	246
Expander_Write_PortA	247
Expander_Write_PortB	248
Expander_Write_PortAB	249
Expander_Set_DirectionPortA	250
Expander_Set_DirectionPortB	250
Expander_Set_DirectionPortAB	251
Expander_Set_PullUpsPortA	251
Expander_Set_PullUpsPortB	252
Expander_Set_PullUpsPortAB	252
Library Example	253
HW Connection	254
PS/2 Library	255
External dependencies of PS/2 Library	255
Library Routines	255
Ps2_Config	256
Ps2_Key_Read	257
Special Function Keys	258
Library Example	259
HW Connection	260
RS-485 Library	261
External dependencies of RS-485 Library	261

Library Routines	261
RS485master_Init	262
RS485master_Receive	262
RS485master_Send	263
RS485slave_Init	264
RS485slave_Receive	265
RS485slave_Send	266
Library Example	266
This is a simple demonstration o	266
HW Connection	270
Message format and CRC calculations	271
Software I ² C Library	272
External dependencies of Soft_I2C Library	272
Library Routines	272
Soft_I2C_Init	272
Soft_I2C_Init	273
Soft_I2C_Start	273
Soft_I2C_Read	273
Soft_I2C_Write	274
Soft_I2C_Stop	274
Library Example	275
Software SPI Library	278
Library Routines	278
Soft_Spi_Init	279
Soft_Spi_Read	279
Soft_Spi_Write	280
Library Example	280
Software UART Library	282
External dependencies of Software UART Library	282
Library Routines	282
Soft_Uart_Init	283
Soft_Uart_Read	284
Soft_Uart_Write	285
Library Example	286
Sound Library	287

External dependencies of Sound Library	287
Library Routines	287
Sound_Init	287
Sound_Play	287
Sound_Init	287
Sound_Play	288
Library Example	288
The example is a simple demo	288
HW Connection	290
SPI Library	291
Library Routines	291
Spi_Init	291
Spi_Init_Advanced	292
Spi_Read	293
Spi_Write	293
Library Example	294
HW Connection	295
SPI Ethernet Library	296
External dependencies of SPI Ethernet Library	296
Library Routines	297
Spi_Ethernet_Init	298
Spi_Ethernet_Enable	300
Spi_Ethernet_Disable	301
Spi_Ethernet_doPacket	302
Spi_Ethernet_putByte	303
Spi_Ethernet_putBytes	303
Spi_Ethernet_putConstBytes	304
Spi_Ethernet_putString	304
Spi_Ethernet_putConstString	305
Spi_Ethernet_getByte	305
Spi_Ethernet_getBytes	306
Spi_Ethernet_UserTCP	307
Spi_Ethernet_UserUDP	308
Library Example	308
HW Connection	316
SPI Graphic LCD Library	317

External dependencies of SPI Graphic LCD Library	317
Library Routines	317
Spi_Glcd_Init	318
Spi_Glcd_Set_Side	319
Spi_Glcd_Set_Page	319
Spi_Glcd_Set_X	320
Spi_Glcd_Read_Data	320
Spi_Glcd_Write_Data	321
Spi_Glcd_Fill	321
Spi_Glcd_Dot	322
Spi_Glcd_Line	323
Spi_Glcd_V_Line	323
Spi_Glcd_H_Line	324
Spi_Glcd_Rectangle	325
Spi_Glcd_Box	326
Spi_Glcd_Circle	326
Spi_Glcd_Set_Font	327
Spi_Glcd_Write_Char	328
Spi_Glcd_Write_Text	329
Spi_Glcd_Image	330
Library Example	330
HW Connection	332
SPI LCD Library	333
External dependencies of SPI LCD Library	333
Library Routines	333
Spi_Lcd_Config	334
Spi_Lcd_Out	334
Spi_Lcd_Out_Cp	335
Spi_Lcd_Chr	335
Spi_Lcd_Chr_Cp	336
Spi_Lcd_Cmd	336
Available LCD Commands	337
Library Example	338
HW Connection	339
SPI LCD8 (8-bit interface) Library	340
External dependencies of SPI LCD Library	340

Library Routines	340
Spi_Lcd8_Config	341
Spi_Lcd8_Out	341
Spi_Lcd8_Out_Cp	342
Spi_Lcd8_Chr	342
Spi_Lcd8_Chr_Cp	343
Spi_Lcd8_Cmd	343
Available LCD Commands	344
Library Example	345
HW Connection	346
SPI T6963C Graphic LCD Library	347
External dependencies of Spi T6963C Graphic LCD Library	347
Library Routines	348
Spi_T6963C_Config	349
Spi_T6963C_WriteData	350
pi_T6963C_WriteCommand	350
Spi_T6963C_SetPtr	351
Spi_T6963C_WaitReady	351
Spi_T6963C_Fill	351
Spi_T6963C_Dot	352
Spi_T6963C_Write_Char	353
Spi_T6963C_Write_Text	354
Spi_T6963C_Line	355
Spi_T6963C_Rectangle	355
Spi_T6963C_Box	356
Spi_T6963C_Circle	356
Spi_T6963C_Image	357
Spi_T6963C_Sprite	357
Spi_T6963C_Set_Cursor	358
Spi_T6963C_ClearBit	358
Spi_T6963C_SetBit	358
Spi_T6963C_NegBit	359
Spi_T6963C_DisplayGrPanel	359
Spi_T6963C_DisplayTxtPanel	359
Spi_T6963C_SetGrPanel	360
Spi_T6963C_SetTxtPanel	360

Spi_T6963C_PanelFill	361
Spi_T6963C_GrFill	361
Spi_T6963C_TxtFill	361
Spi_T6963C_Cursor_Height	362
Spi_T6963C_Graphics	362
Spi_T6963C_Text	362
Spi_T6963C_Cursor	363
Spi_T6963C_Cursor_Blink	363
Library Example	363
HW Connection	368
T6963C Graphic LCD Library	369
External dependencies of T6963C Graphic LCD Library	370
Library Routines	370
T6963C_Init	372
T6963C_WriteData	373
T6963C_WriteCommand	373
T6963C_SetPtr	374
T6963C_WaitReady	374
T6963C_Fill	374
T6963C_Dot	375
T6963C_Write_Char	376
T6963C_Write_Text	377
T6963C_Line	378
T6963C_Rectangle	378
T6963C_Box	379
T6963C_Circle	379
T6963C_Image	380
T6963C_Sprite	380
T6963C_Set_Cursor	381
T6963C_ClearBit	381
T6963C_SetBit	381
T6963C_NegBit	382
T6963C_DisplayGrPanel	382
T6963C_DisplayTxtPanel	382
T6963C_SetGrPanel	383
T6963C_SetTxtPanel	383

T6963C_PanelFill	384
T6963C_GrFill	384
T6963C_TxtFill	384
T6963C_Cursor_Height	385
T6963C_Graphics	385
T6963C_Text	385
T6963C_Cursor	386
T6963C_Cursor_Blink	386
Library Example	386
HW Connection	391
UART Library	392
Library Routines	392
Uart_Init	392
Uart_Data_Ready	393
Uart_Read	393
Uart_Write	394
Library Example	394
HW Connection	395
Button Library	396
External dependencies of Button Library	396
Library Routines	396
Button	397
Conversions Library	398
Library Routines	398
ByteToStr	399
ShortToStr	399
WordToStr	400
IntToStr	400
LongintToStr	401
LongWordToStr	401
FloatToStr	402
Dec2Bcd	403
Bcd2Dec16	403
Dec2Bcd16	404
Math Library	405

Library Functions	405
acos	406
asin	406
atan	406
atan2	406
ceil	406
cos	406
cosh	406
eval_poly	407
exp	407
fabs	407
floor	407
frexp	407
ldexp	407
log	407
log10	408
modf	408
pow	408
sin	408
sinh	408
sqrt	408
tan	408
tanh	409
String Library	410
Library Functions	410
memchr	410
memcmp	411
memcpy	411
memmove	411
memset	412
strcat	412
strchr	412
strcmp	412
strcpy	413
strcspn	413
strlen	413

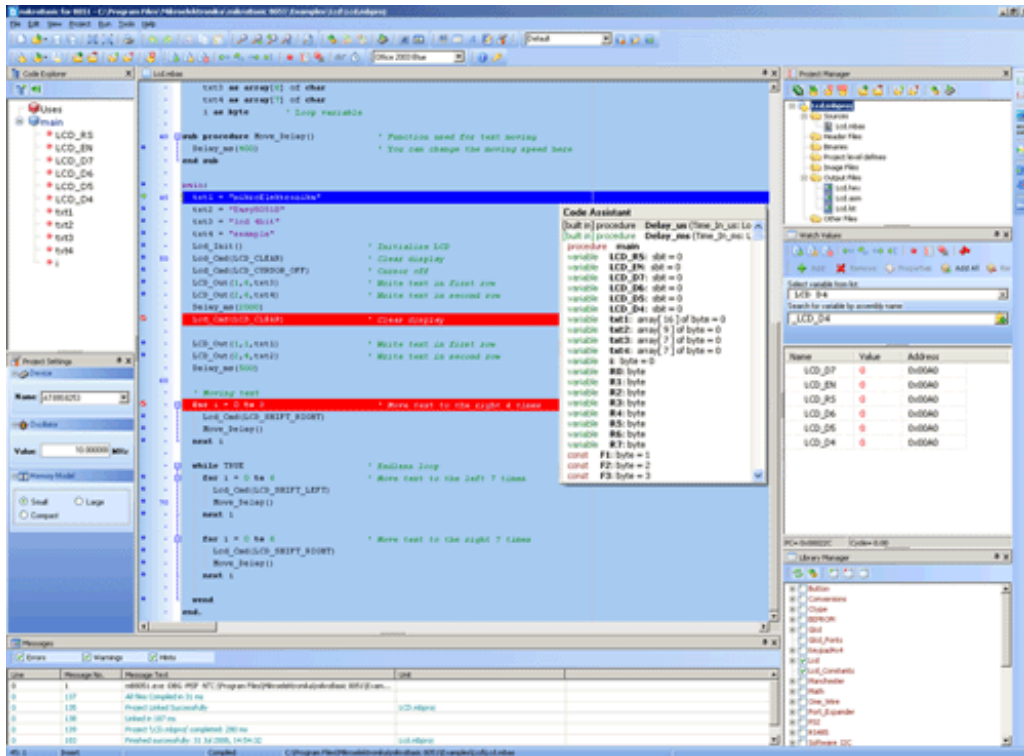
strncat	413
strncmp	413
strncpy	414
strpbrk	414
strchr	414
strspn	414
strstr	414
Time Library	415
Library Routines	415
Time_dateToEpoch	415
Time_epochToDate	416
Time_dateDiff	416
Library Example	417
TimeStruct type definition	418
Trigonometry Library	419
Library Routines	419
sinE3	419
cosE3	420

1

CHAPTER

Introduction to mikroBasic for 8051

The mikroBasic 8051 is a powerful, feature-rich development tool 8051 microcontrollers. It is designed to provide the programmer with the easiest possible solution to developing applications for embedded systems, without compromising performance or control.



Features

mikroBasic for 8051 allows you to quickly develop and deploy complex applications:

- Write your Basic source code using the built-in Code Editor (Code and Parameter Assistants, Code Folding, Syntax Highlighting, Spell Checker, Auto Correct, Code Templates, and more.)
- Use included mikroBasic libraries to dramatically speed up the development: data acquisition, memory, displays, conversions, communication etc.
- Monitor your program structure, variables, and functions in the Code Explorer.
- Generate commented, human-readable assembly, and standard HEX compatible with all programmers.
- Inspect program flow and debug executable logic with the integrated Software Simulator.
- Get detailed reports and graphs: RAM and ROM map, code statistics, assembly listing, calling tree, and more.
- mikroBasic 8051 provides plenty of examples to expand, develop, and use as building bricks in your projects. Copy them entirely if you deem fit – that's why we included them with the compiler.

Where to Start

- In case that you're a beginner in programming 8051 microcontrollers, read carefully the 8051 Specifics chapter. It might give you some useful pointers on 8051 constraints, code portability, and good programming practices.
- If you are experienced in Basic programming, you will probably want to consult mikroBasic Specifics first. For language issues, you can always refer to the comprehensive Language Reference. A complete list of included libraries is available at mikroBasic Libraries.
- If you are not very experienced in Basic programming, don't panic! mikroBasic 8051 provides plenty of examples making it easy for you to go quickly. We suggest that you first consult Projects and Source Files, and then start browsing the examples that you're the most interested in.

MIKROELEKTRONIKA ASSOCIATES LICENSE STATEMENT AND LIMITED WARRANTY

IMPORTANT - READ CAREFULLY

This license statement and limited warranty constitute a legal agreement (“License Agreement”) between you (either as an individual or a single entity) and mikroElektronika (“mikroElektronika Associates”) for software product (“Software”) identified above, including any software, media, and accompanying on-line or printed documentation.

BY INSTALLING, COPYING, OR OTHERWISE USING SOFTWARE, YOU AGREE TO BE BOUND BY ALL TERMS AND CONDITIONS OF THE LICENSE AGREEMENT.

Upon your acceptance of the terms and conditions of the License Agreement, mikroElektronika Associates grants you the right to use Software in a way provided below.

This Software is owned by mikroElektronika Associates and is protected by copyright law and international copyright treaty. Therefore, you must treat this Software like any other copyright material (e.g., a book).

You may transfer Software and documentation on a permanent basis provided. You retain no copies and the recipient agrees to the terms of the License Agreement. Except as provided in the License Agreement, you may not transfer, rent, lease, lend, copy, modify, translate, sublicense, time-share or electronically transmit or receive Software, media or documentation. You acknowledge that Software in the source code form remains a confidential trade secret of mikroElektronika Associates and therefore you agree not to modify Software or attempt to reverse engineer, decompile, or disassemble it, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

If you have purchased an upgrade version of Software, it constitutes a single product with the mikroElektronika Associates software that you upgraded. You may use the upgrade version of Software only in accordance with the License Agreement.

LIMITED WARRANTY

Respectfully excepting the Redistributables, which are provided “as is”, without warranty of any kind, mikroElektronika Associates warrants that Software, once updated and properly used, will perform substantially in accordance with the accompanying documentation, and Software media will be free from defects in materials and workmanship, for a period of ninety (90) days from the date of receipt. Any implied warranties on Software are limited to ninety (90) days.

mikroElektronika Associates' and its suppliers' entire liability and your exclusive remedy shall be, at mikroElektronika Associates' option, either (a) return of the price paid, or (b) repair or replacement of Software that does not meet mikroElektronika Associates' Limited Warranty and which is returned to mikroElektronika Associates with a copy of your receipt. DO NOT RETURN ANY PRODUCT UNTIL YOU HAVE CALLED MIKROELEKTRONIKA ASSOCIATES FIRST AND OBTAINED A RETURN AUTHORIZATION NUMBER. This Limited Warranty is void if failure of Software has resulted from an accident, abuse, or misapplication. Any replacement of Software will be warranted for the rest of the original warranty period or thirty (30) days, whichever is longer.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, MIKROELEKTRONIKA ASSOCIATES AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESSED OR IMPLIED, INCLUDED, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT, WITH REGARD TO SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES.

IN NO EVENT SHALL MIKROELEKTRONIKA ASSOCIATES OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS AND BUSINESS INFORMATION, BUSINESS INTERRUPTION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF MIKROELEKTRONIKA ASSOCIATES HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, MIKROELEKTRONIKA ASSOCIATES' ENTIRE LIABILITY UNDER ANY PROVISION OF THIS LICENSE AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR SOFTWARE PRODUCT PROVIDED, HOWEVER, IF YOU HAVE ENTERED INTO A MIKROELEKTRONIKA ASSOCIATES SUPPORT SERVICES AGREEMENT, MIKROELEKTRONIKA ASSOCIATES' ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT.

HIGH RISK ACTIVITIES

Software is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of Software could lead directly to death, personal injury, or severe physical or environmental damage (“High Risk Activities”). mikroElektronika Associates and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

GENERAL PROVISIONS

This statement may only be modified in writing signed by you and an authorised officer of mikroElektronika Associates. If any provision of this statement is found void or unenforceable, the remainder will remain valid and enforceable according to its terms. If any remedy provided is determined to have failed for its essential purpose, all limitations of liability and exclusions of damages set forth in the Limited Warranty shall remain in effect.

This statement gives you specific legal rights; you may have others, which vary, from country to country. mikroElektronika Associates reserves all rights not specifically granted in this statement.

mikroElektronika

Visegradaska 1A,
11000 Belgrade,
Europe.

Phone: + 381 11 36 28 830

Fax: +381 11 36 28 831

Web: www.mikroe.com

E-mail: office@mikroe.com

TECHNICAL SUPPORT

In case you encounter any problem, you are welcome to our support forums at www.mikroe.com/forum/. Here, you may also find helpful information, hardware tips, and practical code snippets. Your comments and suggestions on future development of the mikroBasic for 8051 are always appreciated — feel free to drop a note or two on our Wishlist.

In our Knowledge Base www.mikroe.com/en/kb/ you can find the answers to Frequently Asked Questions and solutions to known problems. If you can not find the solution to your problem in Knowledge Base then report it to Support Desk www.mikroe.com/en/support/. In this way, we can record and track down bugs more efficiently, which is in our mutual interest. We respond to every bug report and question in a suitable manner, ever improving our technical support.

HOW TO REGISTER


The latest version of the mikroBasic for 8051 is always available for downloading from our website. It is a fully functional software libraries, examples, and comprehensive help included.

The only limitation of the free version is that it cannot generate hex output over 2 KB. Although it might sound restrictive, this margin allows you to develop practical, working applications with no thinking of demo limit. If you intend to develop really complex projects in the mikroBasic for 8051, then you should consider the possibility of purchasing the license key.

Who Gets the License Key

Buyers of the mikroBasic for 8051 are entitled to the license key. After you have completed the payment procedure, you have an option of registering your mikroBasic. In this way you can generate hex output without any limitations.

How to Get License Key

After you have completed the payment procedure, start the program. Select Help › How to Register from the drop-down menu or click the How To Register Icon . Fill out the registration form (figure below), select your distributor, and click the Send button.

How To Register
_ □ ×

Step 1. Fill in the form below. Please, make sure you fill in all required fields.
Step 2. Make sure that you provided a **valid email address** in the "EMAIL" edit box. This email will be used for sending you the activation key.
Step 3. Make sure you select a correct distributor which will make the registration process faster. If your distributor is not on the list then select "Other" and type in distributor's email address in the box below.
Step 4. Press the **SEND** button to send key request. A default email client will open with ready-to-send message.
 Note: If email client does not open, you may copy text of the message and paste it manually into a new email message before sending it to your distributor's email.

NAME*	Marko Medic
ADDRESS	Enter your address
INVOICE	Enter invoice number if available
E-MAIL*	marko.medic@mikroe.com
E-MAIL*	marko.medic@mikroe.com
COMPANY	Enter company name
PRODUCT ID	455A-677169-766564-674C10
DISTRIBUTOR*	mikroElektronika key@mikroe.com

* Required fields

I have made the payment and I wish to request activation key for mikroBasic for 8051-----

Name:
Marko Medic

Address:

Invoice number:

Company:

E-Mail:
marko.medic@mikroe.com

Product key:
455A-677169-766564-674C10

Distributor:
mikroElektronika
key@mikroe.com

Copy to clipboard

SEND

Cancel

This will start your e-mail client with message ready for sending. Review the information you have entered, and add the comment if you deem it necessary. Please, do not modify the subject line.

Upon receiving and verifying your request, we will send the license key to the e-mail address you specified in the form.

After Receiving the License Key

The license key comes as a small autoextracting file – just start it anywhere on your computer in order to activate your copy of compiler and remove the demo limit. You do not need to restart your computer or install any additional components. Also, there is no need to run the mikroBasic for 8051 at the time of activation.

Notes:

- The license key is valid until you format your hard disk. In case you need to format the hard disk, you should request a new activation key.
- Please keep the activation program in a safe place. Every time you upgrade the compiler you should start this program again in order to reactivate the license.

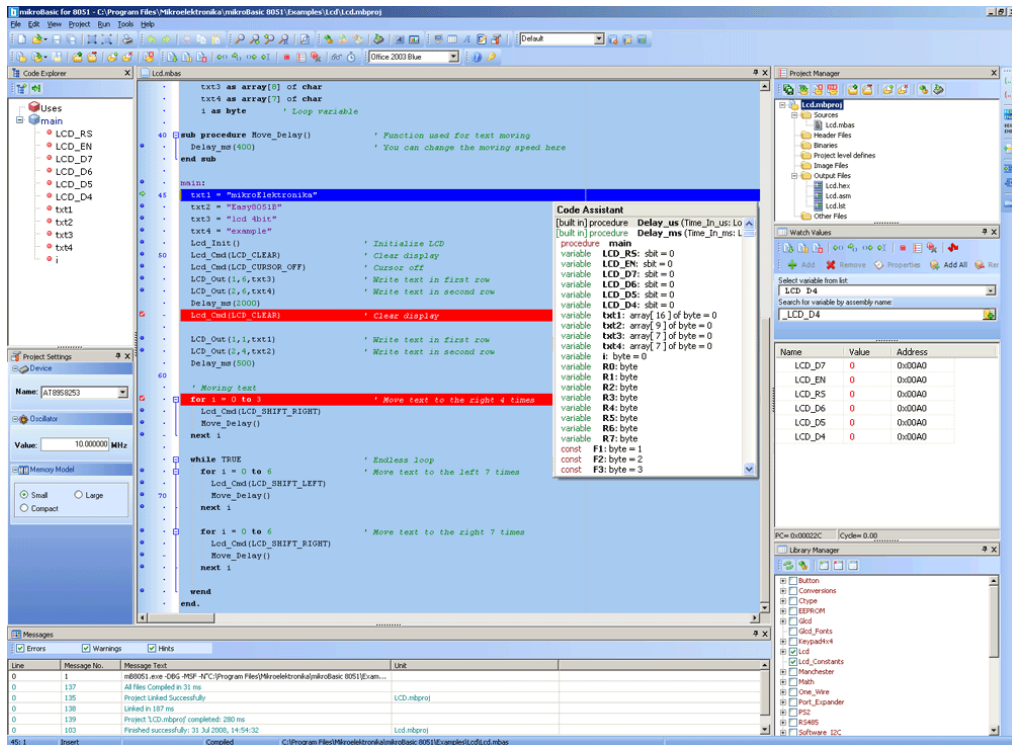
CHAPTER

2

mikroBasic for 8051 Environment

The mikroBasic for 8051 is an user-friendly and intuitive environment:

IDE Overview



- The Code Editor features adjustable Syntax Highlighting, Code Folding, Code Assistant, Parameters Assistant, Spell Checker, Auto Correct for common typos and Code Templates (Auto Complete).
- The Code Explorer (with Keyboard shortcut browser and Quick Help browser) is at your disposal for easier project management.
- The Project Manager allows multiple project management
- General project settings can be made in the Project Settings window
- Library manager enables simple handling libraries being used in a project
- The Error Window displays all errors detected during compiling and linking.
- The source-level Software Simulator lets you debug executable logic step-by-step by watching the program flow.
- The New Project Wizard is a fast, reliable, and easy way to create a project.
- Help files are syntax and context sensitive.
- Like in any modern Windows application, you may customize the layout of mikroBasic for 8051 to suit your needs best.
- Spell checker underlines identifiers which are unknown to the project. In this way it helps the programmer to spot potential problems early, much before the project is compiled.
Spell checker can be disabled by choosing the option in the Preferences dialog (F12).

MAIN MENU OPTIONS

Available Main Menu options are:

File

Edit

View

Project

Run

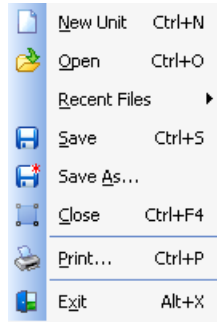
Tools








Help

Related topics: Keyboard shortcuts

FILE MENU OPTIONS

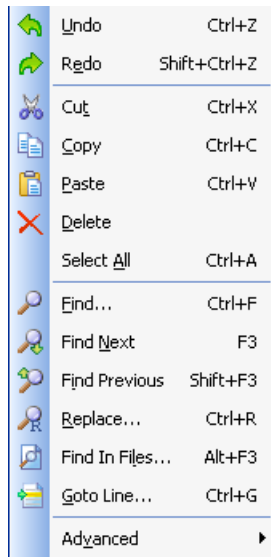
The File menu is the main entry point for manipulation with the source files.










File	Description
 <u>N</u> ew Unit Ctrl+N	Open a new editor window.
 <u>O</u> pen Ctrl+O	Open source file for editing or image file for viewing.
<u>R</u> ecent Files ▶	Reopen recently used file.
 <u>S</u> ave Ctrl+S	Save changes for active editor.
 <u>S</u> ave <u>A</u> s...	Save the active source file with the different name or change the file type.
 <u>C</u> lose Alt+F4	Close active source file.
 <u>P</u> rint... Ctrl+P	Print Preview.
 <u>E</u> xit Alt+X	Exit IDE.

Related topics: Keyboard shortcuts, File Toolbar, Managing Source Files

EDIT MENU OPTIONS

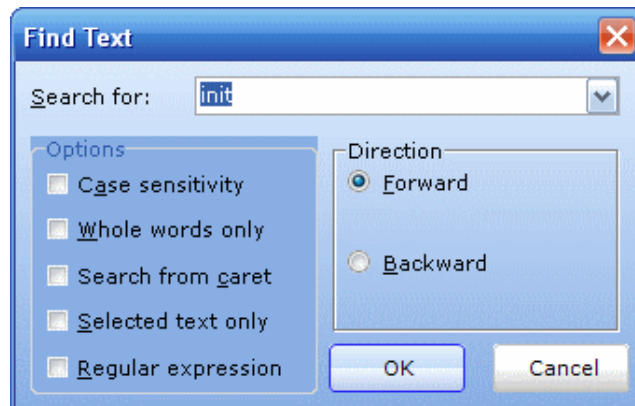


Edit	Description
Undo Ctrl+Z	Undo last change.
Redo Shift+Ctrl+Z	Redo last change.
Cut Ctrl+X	Cut selected text to clipboard.
Copy Ctrl+C	Copy selected text to clipboard.
Paste Ctrl+V	Paste text from clipboard.
Delete	Delete selected text.
Select All Ctrl+A	Select all text in active editor.
Find... Ctrl+F	Find text in active editor.
Find Next F3	Find next occurrence of text in active editor.
Find Previous Shift+F3	Find previous occurrence of text in active editor.
Replace... Ctrl+R	Replace text in active editor.
Find In Files... Alt+F3	Find text in current file, in all opened files, or in files from desired folder.
Goto Line... Ctrl+G	Goto to the desired line in active editor.
Advanced ▶	Advanced Code Editor options.

Advanced>>	Description
 <u>C</u> omment Shift+Ctrl+.	Comment selected code or put single line comment if there is no selection.
 <u>U</u> ncomment Shift+Ctrl+,	Uncomment selected code or remove single line comment if there is no selection.
 <u>I</u> ndent Shift+Ctrl+I	Indent selected code.
 <u>O</u> utdent Shift+Ctrl+U	Outdent selected code.
 <u>L</u> owercase Ctrl+Alt+L	Changes selected text case to lowercase.
 <u>U</u> ppercase Ctrl+Alt+U	Changes selected text case to uppercase.
 <u>T</u> itlecase Ctrl+Alt+T	Changes selected text case to titlecase.

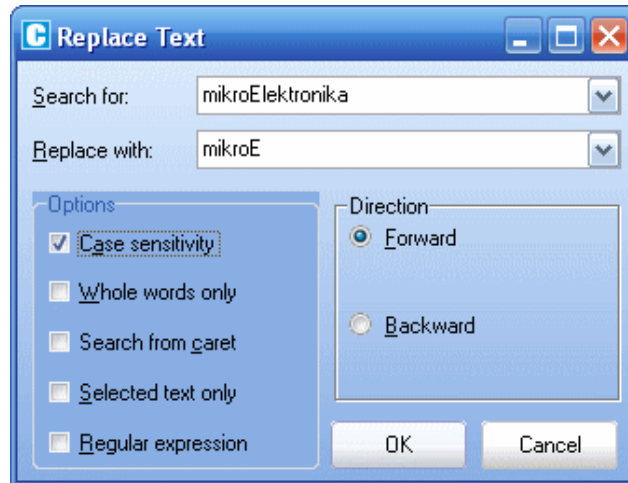
Find Text

Dialog box for searching the document for the specified text. The search is performed in the direction specified. If the string is not found a message is displayed.



Replace Text

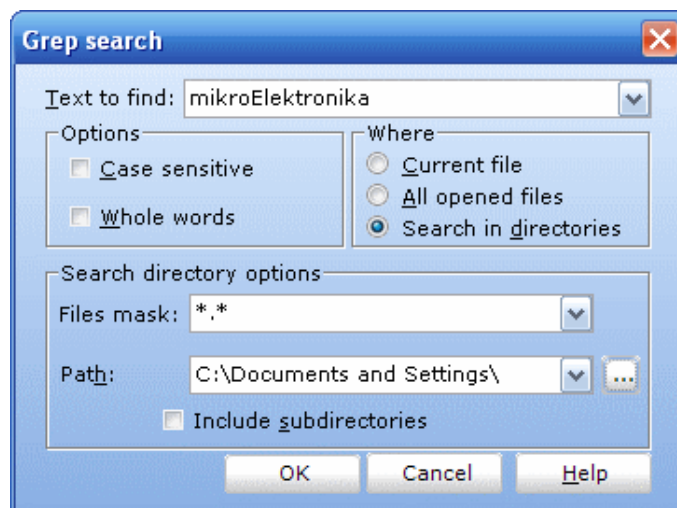
Dialog box for searching for a text string in file and replacing it with another text string.



Find In Files

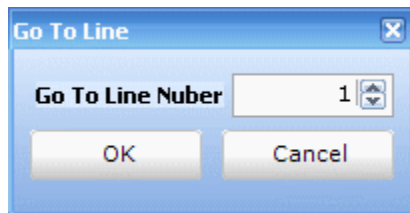
Dialog box for searching for a text string in current file, all opened files, or in files on a disk.

The string to search for is specified in the **Text to find** field. If Search in directories option is selected, The files to search are specified in the **Files mask** and **Path** fields.



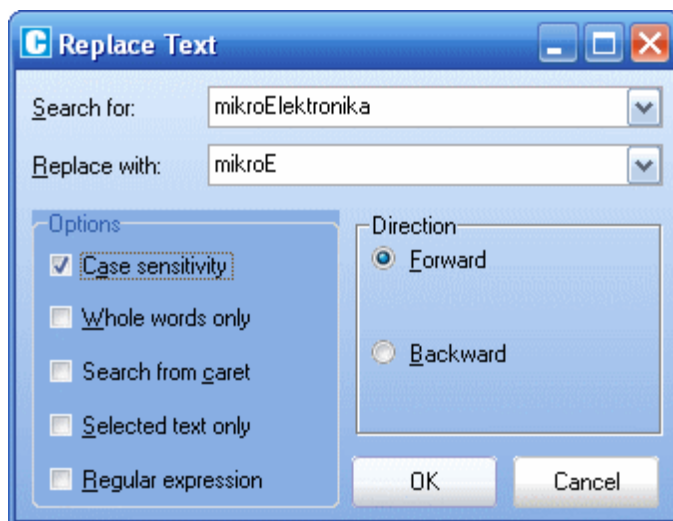
Go To Line

Dialog box that allows the user to specify the line number at which the cursor should be positioned.



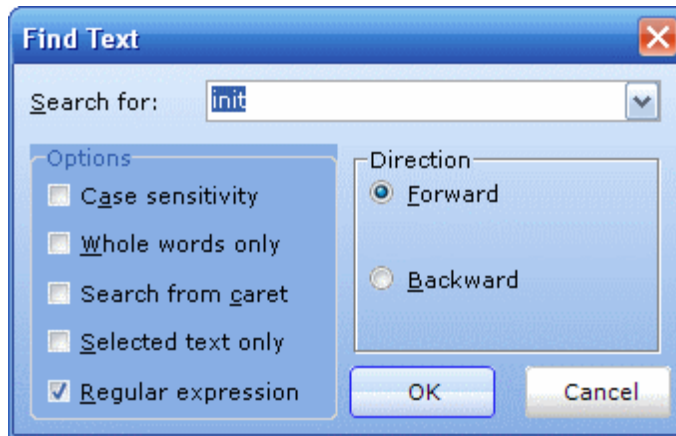
Replace Text

Dialog box for searching for a text string in file and replacing it with another text string.



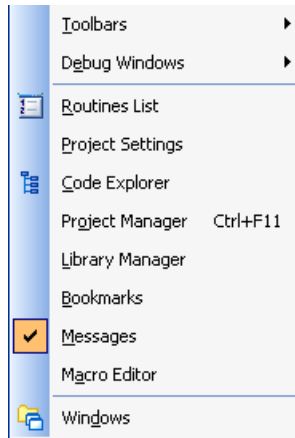
Regular expressions

By checking this box, you will be able to advance your search, through Regular expressions.



Related topics: Keyboard shortcuts, Edit Toolbar, Advanced Edit Toolbar

VIEW MENU OPTIONS










View	Description
Toolbars	Show/Hide toolbars.
Debug Windows	Show/Hide debug windows.
Routines List	Show/Hide Routine List in active editor.
Project Settings	Show/Hide Project Settings window.
Code Explorer	Show/Hide Code Explorer window.
Project Manager Shift+ Ctrl+F11	Show/Hide Project Manager window.
Library Manager	Show/Hide Library Manager window.
Bookmarks	Show/Hide Bookmarks window.
Messages	Show/Hide Error Messages window.
Macro Editor	Show/Hide Macro Editor window.
Windows	Show Window List window.

TOOLBARS

File Toolbar








File Toolbar is a standard toolbar with following options:

Icon	Description
	Opens a new editor window.
	Open source file for editing or image file for viewing.
	Save changes for active window.
	Save changes in all opened windows.
	Close current editor.
	Close all editors.
	Print Preview.

Edit Toolbar



Edit Toolbar is a standard toolbar with following options:

Icon	Description
	Undo last change.
	Redo last change.
	Cut selected text to clipboard.
	Copy selected text to clipboard.
	Paste text from clipboard.

Advanced Edit Toolbar



Advanced Edit Toolbar comes with following options:

Icon	Description
	Comment selected code or put single line comment if there is no selection
	Uncomment selected code or remove single line comment if there is no selection.
	Select text from starting delimiter to ending delimiter.
	Go to ending delimiter.
	Go to line.
	Indent selected code lines.
	Outdent selected code lines.
	Generate HTML code suitable for publishing current source code on the web.

Find/Replace Toolbar



Find/Replace Toolbar is a standard toolbar with following options:

Icon	Description
	Find text in current editor.
	Find next occurrence.
	Find previous occurrence.
	Replace text.
	Find text in files.

Project Toolbar



Project Toolbar comes with following options:

Icon	Description
	Open new project wizard. wizard.
	Open Project
	Save Project
	Add existing project to project group.
	Remove existing project from project group.
	Add File To Project
	Remove File From Project
	Close current project.

Build Toolbar















Build Toolbar comes with following options:

Icon	Description
	Build current project.
	Build all opened projects.
	Build and program active project.
	Start programmer and load current HEX file.
	Open assembly code in editor.
	View statistics for current project.

Debugger



Debugger Toolbar comes with following options:

Icon	Description
	Start Software Simulator.
	Run/Pause debugger.
	Stop debugger.
	Step into.
	Step over.
	Step out.
	Run to cursor.
	Toggle breakpoint.
	Toggle breakpoints.
	Clear breakpoints.
	View watch window
	View stopwatch window

Styles Toolbar







Styles toolbar allows you to easily customize your workspace.

Tools Toolbar



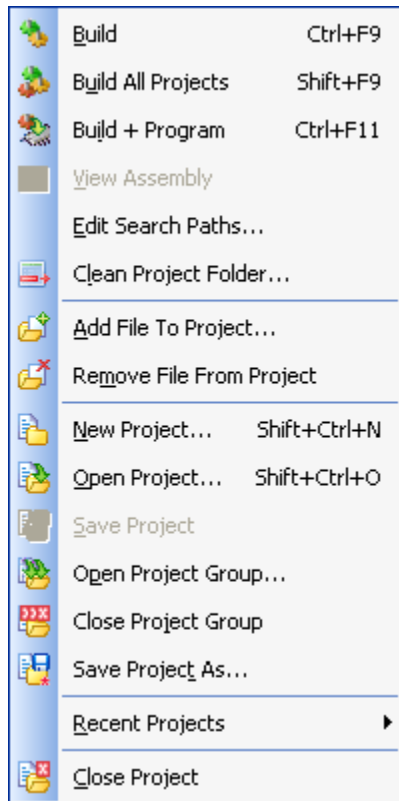
Tools Toolbar comes with following default options:















Icon	Description
	Run USART Terminal
	EEPROM
	ASCII Chart
	Seven segment decoder tool.

The Tools toolbar can easily be customized by adding new tools in Options(F12) window.

Related topics: Keyboard shortcuts, Integrated Tools, Debugger Windows













PROJECT MENU OPTIONS















Project	Description
 Build Ctrl+F	Build active project.
 Build All Shift+F	Build all projects.
 Build + Program Ctrl+F1	Build and program active project.
 View Assembly	View Assembly.
Edit Search Paths...	Edit search paths.
 Clean Project Folder...	Clean Project Folder
 Add File To Project...	Add file to project.
 Remove File From Project	Remove file from project.
 New Project...	Open New Project Wizard
 Open Project... Shift+Ctrl+	Open existing project.
 Save Project	Save current project.
 Open Project Group...	Open project group.
 Close Project Group	Close project group.
 Save Project As...	Save active project file with the different name.
Recent Projects	Open recently used project.
 Close Project	Close active project.

Related topics: Keyboard shortcuts, Project Toolbar, Creating New Project, Project Manager, Project Settings

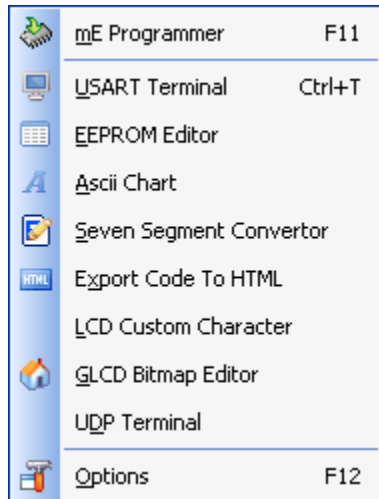
RUN MENU OPTIONS









	Start Debugger	F9
	Stop Debugger	Ctrl+F2
	Pause Debugger	F6
	Step Into	F7
	Step Over	F8
	Step Out	Ctrl+F8
	Jump To Interrupt	F2
	Toggle Breakpoint	F5
	Breakpoints	Shift+F4
	Clear Breakpoints	Shift+Ctrl+F5
	Watch Window	Shift+F5
	View Stopwatch	
	Disassembly mode	Alt+D

Run	Description
 Start Debugger F9	Start Software Simulator.
 Stop Debugger Ctrl+F2	Stop debugger.
 Pause Debugger F6	Pause Debugger.
 Step Into F7	Step Into.
 Step Over F8	Step Over.
 Step Out Ctrl+F8	Step Out.
 Jump To Interrupt F2	Jump to interrupt in current project.
 Toggle Breakpoint F5	Toggle Breakpoint.
 Show/Hide Breakpoints Shift+F4	Breakpoints.
 Clear Breakpoints Shift+Ctrl+F5	Clear Breakpoints.
 Watch Window Shift+F5	Show/Hide Watch Window
 View Stopwatch	Show/Hide Stopwatch Window
Disassembly mode Ctrl+D	Toggle between Basic source and disassembly.

Related topics: Keyboard shortcuts, Debug Toolbar

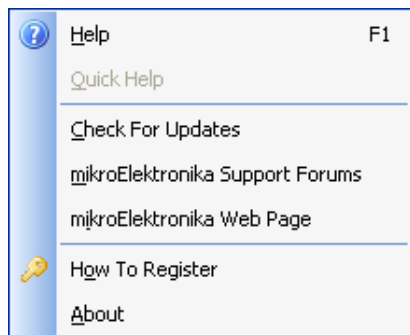
TOOLS MENU OPTIONS



Tools	Description
 PicFlash Programmer F11	Run mikroElektronika Programmer
 USART Terminal Ctrl+T	Run USART Terminal
 EEPROM Editor	Run EEPROM Editor
 Ascii Chart	Run ASCII Chart
 Seven Segment Convertor	Run 7 Segment Display Decoder
 Export Code To HTML	Generate HTML code suitable for publishing source code on the web.
LCD Custom Character	Generate your own custom LCD characters
 GLCD Bitmap Editor	Generate bitmap pictures for GLCD
UDP Terminal	UDP communication terminal.
 Options F12	Open Options window

Related topics: Keyboard shortcuts, Tools Toolbar

HELP MENU OPTIONS



Help	Description
Help F1	Open Help File.
Quick Help	Quick Help.
Check For Updates	Check if new compiler version is available.
mikroElektronika Support Forums	Open mikroElektronika Support Forums in a default browser.
mikroElektronika Web Page	Open mikroElektronika Web Page in a default browser.
How To Register	Information on how to register
About	Open About window.

Related topics: Keyboard shortcuts

KEYBOARD SHORTCUTS

Below is a complete list of keyboard shortcuts available in mikroBasic for 8051 IDE. You can also view keyboard shortcuts in the Code Explorer window, tab Keyboard.

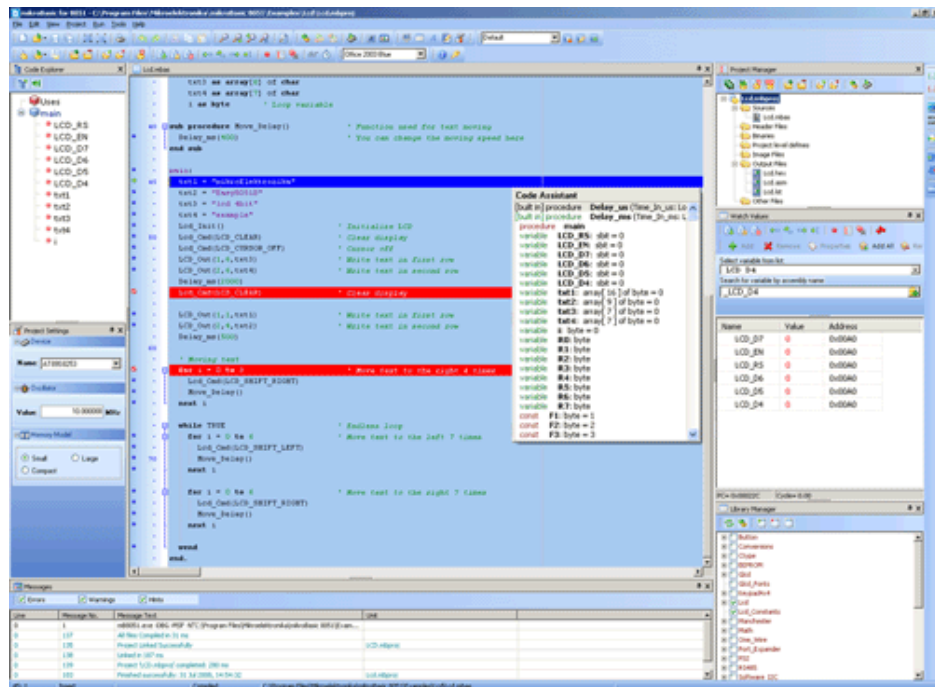
IDE Shortcuts	
F1	Help
Ctrl+N	New Unit
Ctrl+O	Open
Ctrl+Shift+O	Open Project
Ctrl+Shift+N	Open New Project
Ctrl+K	Close Project
Ctrl+F9	Compile
Shift+F9	Compile All
Ctrl+F11	Compile and Program
Shift+F4	View breakpoints
Ctrl+Shift+F5	Clear breakpoints
F11	Start 8051Flash Programmer
F12	Preferences
Basic Editor Shortcuts	
F3	Find, Find Next
Shift+F3	Find Previous
Alt+F3	Grep Search, Find in Files
Ctrl+A	Select All
Ctrl+C	Copy
Ctrl+F	Find
Ctrl+R	Replace
Ctrl+P	Print
Ctrl+S	Save unit
Ctrl+Shift+S	Save All
Ctrl+V	Paste

Ctrl+X	Cut
Ctrl+Y	Delete entire line
Ctrl+Z	Undo
Ctrl+Shift+Z	Redo
Advanced Editor Shortcuts	
Ctrl+Space	Code Assistant
Ctrl+Shift+Space	Parameters Assistant
Ctrl+D	Find declaration
Ctrl+E	Incremental Search
Ctrl+L	Routine List
Ctrl+G	Goto line
Ctrl+J	Insert Code Template
Ctrl+Shift+.	Comment Code
Ctrl+Shift+,	Uncomment Code
Ctrl+number	Goto bookmark
Ctrl+Shift+number	Set bookmark
Ctrl+Shift+I	Indent selection
Ctrl+Shift+U	Unindent selection
TAB	Indent selection
Shift+TAB	Unindent selection
Alt+Select	Select columns
Ctrl+Alt+Select	Select columns
Ctrl+Alt+L	Convert selection to lowercase
Ctrl+Alt+U	Convert selection to uppercase
Ctrl+Alt+T	Convert to Titlecase

Software Simulator Shortcuts	
F2	Jump To Interrupt
F4	Run to Cursor
F5	Toggle Breakpoint
F6	Run/Pause Debugger
F7	Step into
F8	Step over
F9	Debug
Ctrl+F2	Reset
Ctrl+F5	Add to Watch List
Ctrl+F8	Step out
Alt+D	Dissassembly view
Shift+F5	Open Watch Window

IDE OVERVIEW

The mikroBasic for 8051 is an user-friendly and intuitive environment:



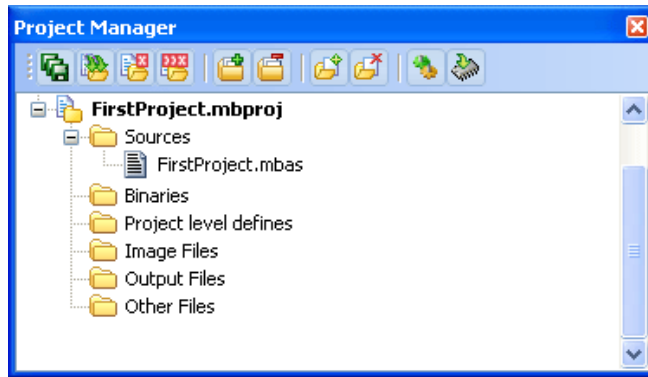
- The Code Editor features adjustable Syntax Highlighting, Code Folding, Code Assistant, Parameters Assistant, Spell Checker, Auto Correct for common typos and Code Templates (Auto Complete).
- The Code Explorer (with Keyboard shortcut browser and Quick Help browser) is at your disposal for easier project management.
- The Project Manager allows multiple project management
- General project settings can be made in the Project Settings window
- Library manager enables simple handling libraries being used in a project
- The Error Window displays all errors detected during compiling and linking.
- The source-level Software Simulator lets you debug executable logic step-by-step by watching the program flow.
- The New Project Wizard is a fast, reliable, and easy way to create a project.
- Help files are syntax and context sensitive.
- Like in any modern Windows application, you may customize the layout of mikroBasic for 8051 to suit your needs best.
- Spell checker underlines identifiers which are unknown to the project. In this way it helps the programmer to spot potential problems early, much before the project is compiled.
- Spell checker can be disabled by choosing the option in the Preferences dialog (F12).

CUSTOMIZING IDE LAYOUT

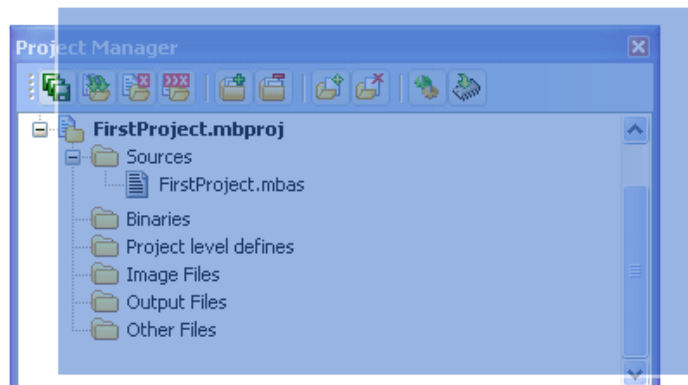
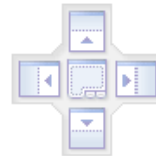
Docking Windows

You can increase the viewing and editing space for code, depending on how you arrange the windows in the IDE.

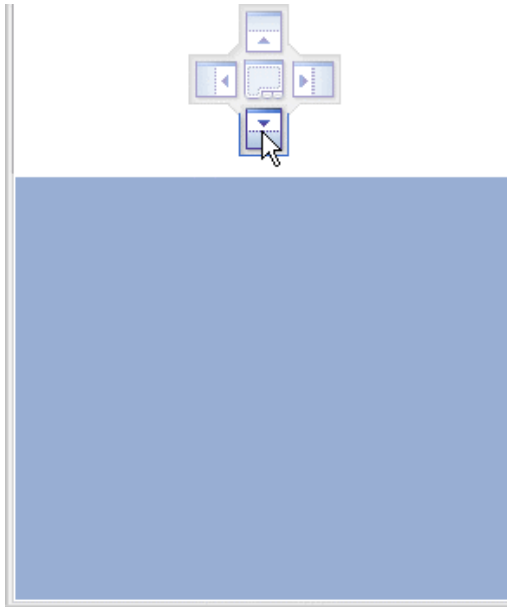
Step 1: Click the window you want to dock, to give it focus.



Step 2: Drag the tool window from its current location. A guide diamond appears. The four arrows of the diamond point towards the four edges of the IDE.




Step 3: Move the pointer over the corresponding portion of the guide diamond. An outline of the window appears in the designated area.





Step 4: To dock the window in the position indicated, release the mouse button.

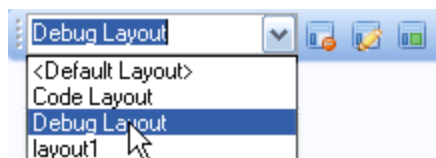
Tip: To move a dockable window without snapping it into place, press CTRL while dragging it.

Saving Layout

Once you have a window layout that you like, you can save the layout by typing the name for the layout and pressing the Save Layout Icon .


To set the layout select the desired layout from the layout drop-down list and click the Set Layout Icon .

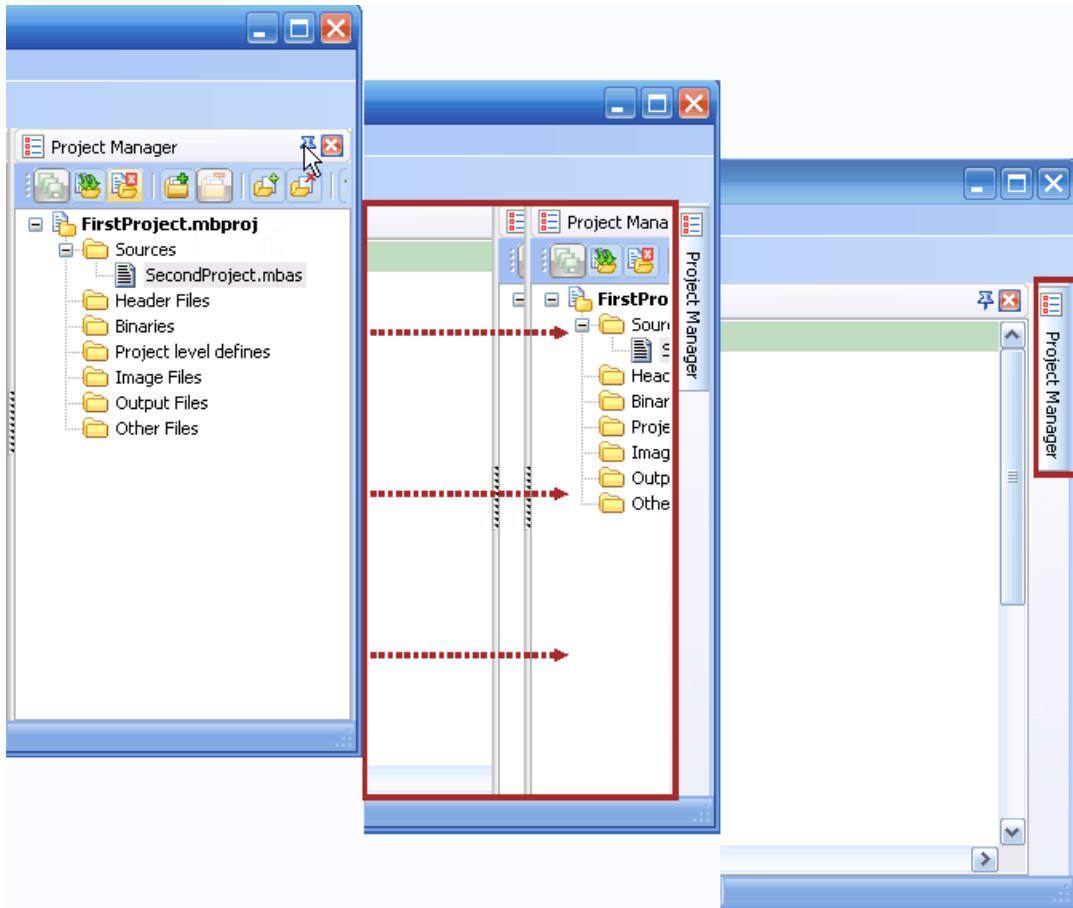
To remove the layout from the drop-down list, select the desired layout from the list and click the Delete Layout Icon .



Auto Hide

Auto Hide enables you to see more of your code at one time by minimizing tool windows along the edges of the IDE when not in use.

- Click the window you want to keep visible to give it focus.
- Click the Pushpin Icon  on the title bar of the window.




When an auto-hidden window loses focus, it automatically slides back to its tab on the edge of the IDE. While a window is auto-hidden, its name and icon are visible on a tab at the edge of the IDE. To display an auto-hidden window, move your pointer over the tab. The window slides back into view and is ready for use.

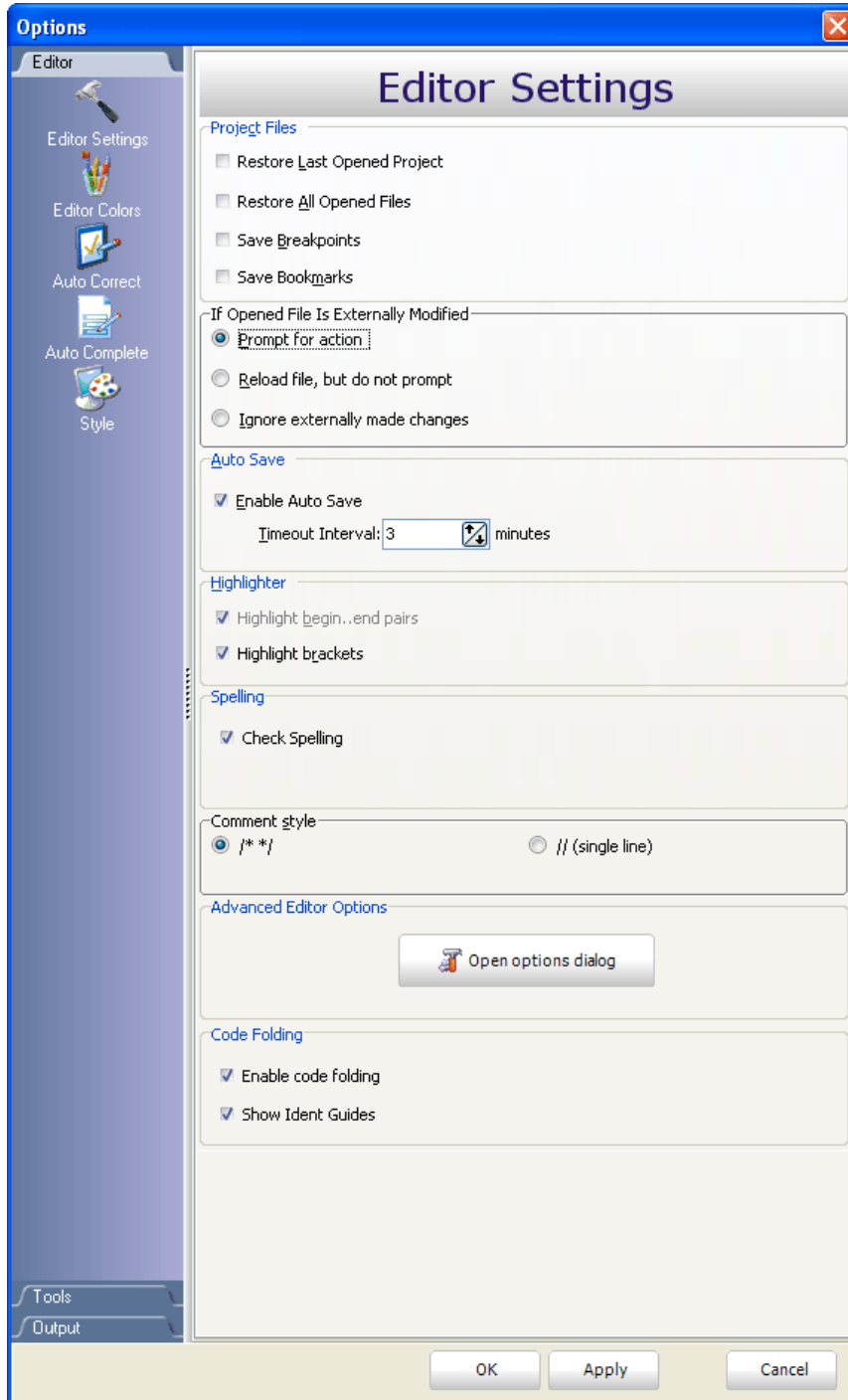
ADVANCED CODE EDITOR

The Code Editor is advanced text editor fashioned to satisfy needs of professionals. General code editing is the same as working with any standard text-editor, including familiar Copy, Paste and Undo actions, common for Windows environment.

Advanced Editor Features

- Adjustable Syntax Highlighting
- Code Assistant
- Code Folding
- Parameter Assistant
- Code Templates (Auto Complete)
- Auto Correct for common typos
- Spell Checker
- Bookmarks and Goto Line
- Comment / Uncomment

You can configure the Syntax Highlighting, Code Templates and Auto Correct from the Editor Settings dialog. To access the Settings, click **Tools** > **Options** from the drop-down menu, click the Show Options Icon  or press F12 key.





Code Assistant

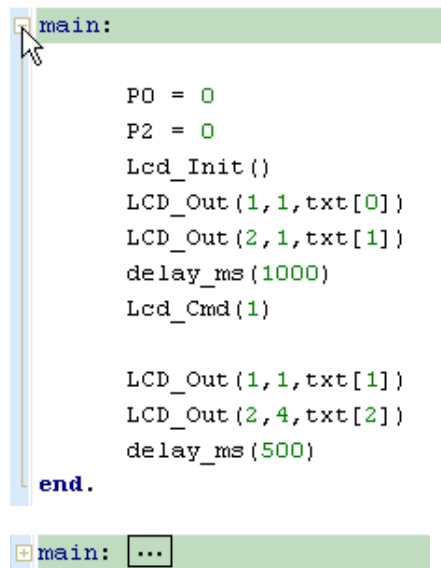
If you type the first few letters of a word and then press Ctrl+Space, all valid identifiers matching the letters you have typed will be prompted in a floating panel (see the image below). Now you can keep typing to narrow the choice, or you can select one from the list using the keyboard arrows and Enter.



Code Folding

Code folding is IDE feature which allows users to selectively hide and display sections of a source file. In this way it is easier to manage large regions of code within one window, while still viewing only those subsections of the code that are relevant during a particular editing session.

While typing, the code folding symbols ( and ) appear automatically. Use the folding symbols to hide/unhide the code subsections.



If you place a mouse cursor over the tooltip box, the collapsed text will be shown in a tooltip style box.

```

main:
main
    PO = 0
    P2 = 0
    Lcd_Init()
    LCD_Out(1,1,txt[0])
    LCD_Out(2,1,txt[1])
    delay_ms(1000)
    Lcd_Cmd(1)

    LCD_Out(1,1,txt[1])
    LCD_Out(2,4,txt[2])
    delay_ms(500)
end.

```

Parameter Assistant

The Parameter Assistant will be automatically invoked when you open parenthesis “(” or press Shift+Ctrl+Space. If the name of a valid function precedes the parenthesis, then the expected parameters will be displayed in a floating panel. As you type the actual parameter, the next expected parameter will become bold.


```

ADC_Read(channel : byte)

```

Code Templates (Auto Complete)

You can insert the Code Template by typing the name of the template (for instance, `whiles`), then press Ctrl+J and the Code Editor will automatically generate a code.


You can add your own templates to the list. Select **Tools > Options** from the drop-down menu, or click the Show Options Icon  and then select the Auto Complete Tab. Here you can enter the appropriate keyword, description and code of your template.

Autocomplete macros can retrieve system and project information:

- %DATE% - current system date
- %TIME% - current system time
- %DEVICE% - device(MCU) name as specified in project settings
- %DEVICE_CLOCK% - clock as specified in project settings
- %COMPILER% - current compiler version

These macros can be used in template code, see template `ptemplate` provided with mikroBasic for 8051 installation.


Auto Correct

The Auto Correct feature corrects common typing mistakes. To access the list of recognized typos, select **Tools > Options** from the drop-down menu, or click the Show Options Icon  and then select the Auto Correct Tab. You can also add your own preferences to the list.

Also, the Code Editor has a feature to comment or uncomment the selected code by simple click of a mouse, using the Comment Icon  and Uncomment Icon  from the Code Toolbar.

Spell Checker

The Spell Checker underlines unknown objects in the code, so it can be easily noticed and corrected before compiling your project.

Select **Tools > Options** from the drop-down menu, or click the Show Options Icon  and then select the Spell Checker Tab.



Bookmarks

Bookmarks make navigation through a large code easier. To set a bookmark, use Ctrl+Shift+number. To jump to a bookmark, use Ctrl+number.

Goto Line

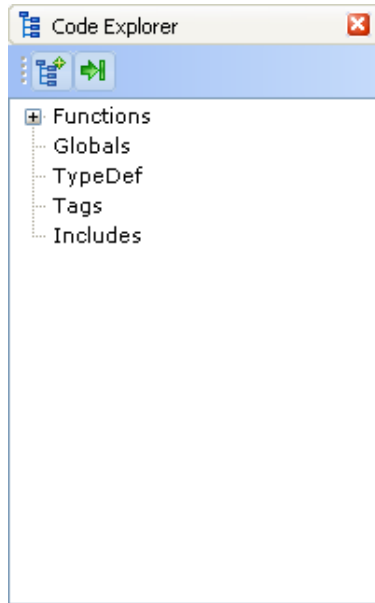
The Goto Line option makes navigation through a large code easier. Use the shortcut Ctrl+G to activate this option.

Comment / Uncomment

Also, the Code Editor has a feature to comment or uncomment the selected code by simple click of a mouse, using the Comment Icon  and Uncomment Icon  from the Code Toolbar.

CODE EXPLORER

The Code Explorer gives clear view of each item declared inside the source code. You can jump to a declaration of any item by right clicking it. Also, besides the list of defined and declared objects, code explorer displays message about first error and it's location in code.



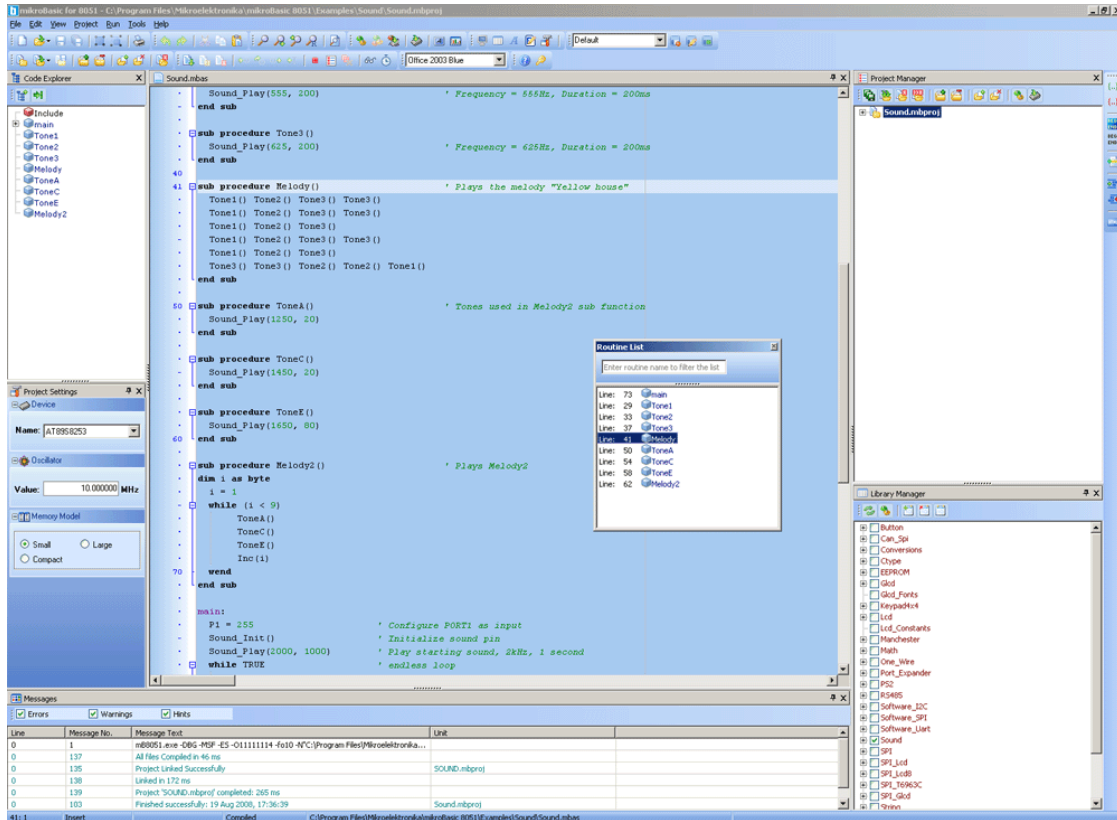
Following options are available in the Code Explorer:

Icon	Description
	Expand/Collapse all nodes in tree.
	Locate declaration in code.

ROUTINE LIST

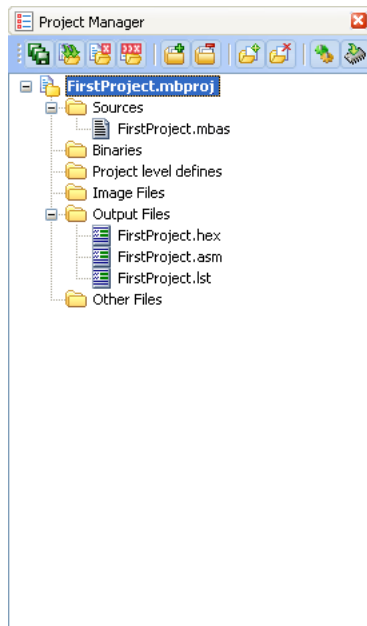
Routine list displays list of routines, and enables filtering routines by name. Routine list window can be accessed by pressing Ctrl+L.

You can jump to a desired routine by double clicking on it.













PROJECT MANAGER

Project Manager is IDE feature which allows users to manage multiple projects. Several projects which together make project group may be open at the same time. Only one of them may be active at the moment. Setting project in active mode is performed by double click on the desired project in the Project Manager.



Following options are available in the Project Manager:

Icon	Description
	Save project Group.
	Open project group.
	Close the active project.
	Close project group.
	Add project to the project group.
	Remove project from the project group.
	Add file to the active project.
	Remove selected file from the project.
	Build the active project.
	Run mikroElektronika's Flash programmer.

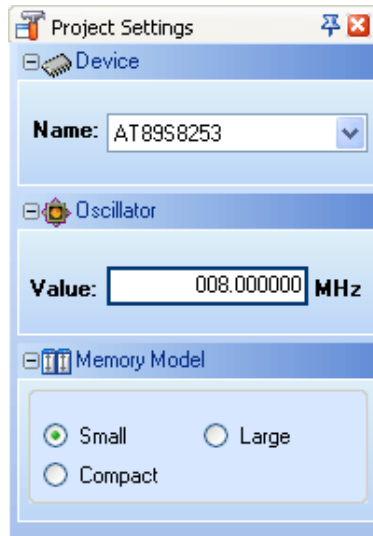
For details about adding and removing files from project see Add/Remove Files from Project.

Related topics: Project Settings, Project Menu Options, File Menu Options, Project Toolbar, Build Toolbar, Add/Remove Files from Project

PROJECT SETTINGS WINDOW

Following options are available in the Project Settings Window:



- Device - select the appropriate device from the device drop-down list.
- Oscillator - enter the oscillator frequency value.
- Memory Model - Select the desired memory model.



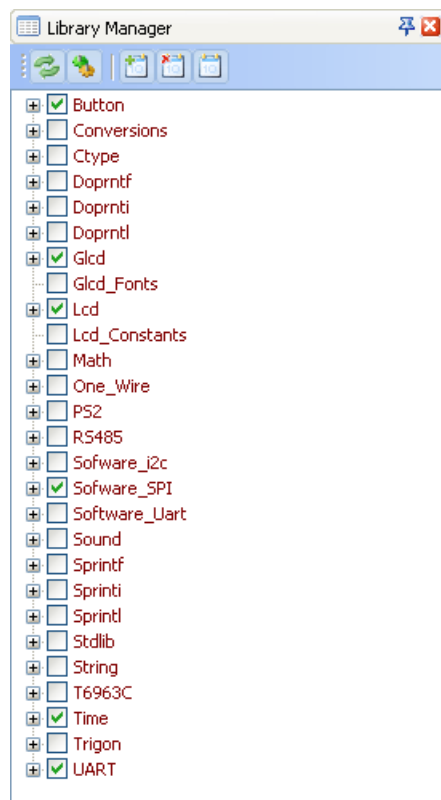
Related topics: Memory Model, Project Manager






LIBRARY MANAGER

Library Manager enables simple handling libraries being used in a project. Library Manager window lists all libraries (extension `.mcl`) which are instantly stored in the compiler Uses folder. The desirable library is added to the project by selecting check box next to the library name.

In order to have all library functions accessible, simply press the button **Check All**  and all libraries will be selected. In case none library is needed in a project, press the button **Clear All**  and all libraries will be cleared from the project.

Only the selected libraries will be linked.



Icon	Description
	Refresh Library by scanning files in "Uses" folder. Useful when new libraries are added by copying files to "Uses" folder.
	Rebuild all available libraries. Useful when library sources are available and need refreshing.
	Include all available libraries in current project.
	No libraries from the list will be included in current project.
	Restore library to the state just before last project saving.

Related topics: mikroBasic for 8051 Libraries, Creating New Library

ERROR WINDOW

In case that errors were encountered during compiling, the compiler will report them and won't generate a hex file. The Error Window will be prompted at the bottom of the main window by default.


The Error Window is located under message tab, and displays location and type of errors the compiler has encountered. The compiler also reports warnings, but these do not affect the output; only errors can interfere with the generation of hex.

Line	Message No.	Message Text	Unit
0	1	mb8051.exe -DBG -MSF -N"C:\Program Files\Mikroelektronika\mikroBasic 8051\Examples\Led Blinking\LedBlinking.mbproj" ...	
31	304	Syntax error: Expected ")" but "P0" found	LedBlinking.mbas
31	304	Syntax error: Expected "wend" but "=" found	LedBlinking.mbas
31	301	"0xFF" is not valid identifier	LedBlinking.mbas
31	304	Syntax error: Expected "end" but "0xFF" found	LedBlinking.mbas
32	304	Syntax error: Expected "." but "P1" found	LedBlinking.mbas
31: 5	Insert	Compiled	C:\Program Files\Mikroelektronika\mikroBasic 8051\Examples\Led Blinking\LedBlinking.mbas

Double click the message line in the Error Window to highlight the line where the error was encountered.

Related topics: Error Messages

STATISTICS

After successful compilation, you can review statistics of your code. Click the Statistics Icon  .

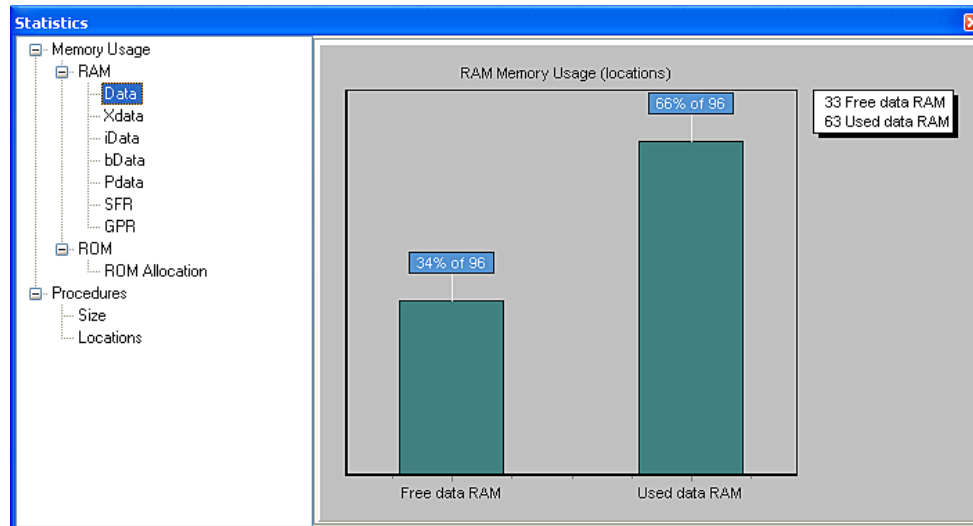
Memory Usage Windows

Provides overview of RAM and ROM usage in the form of histogram.

RAM Memory

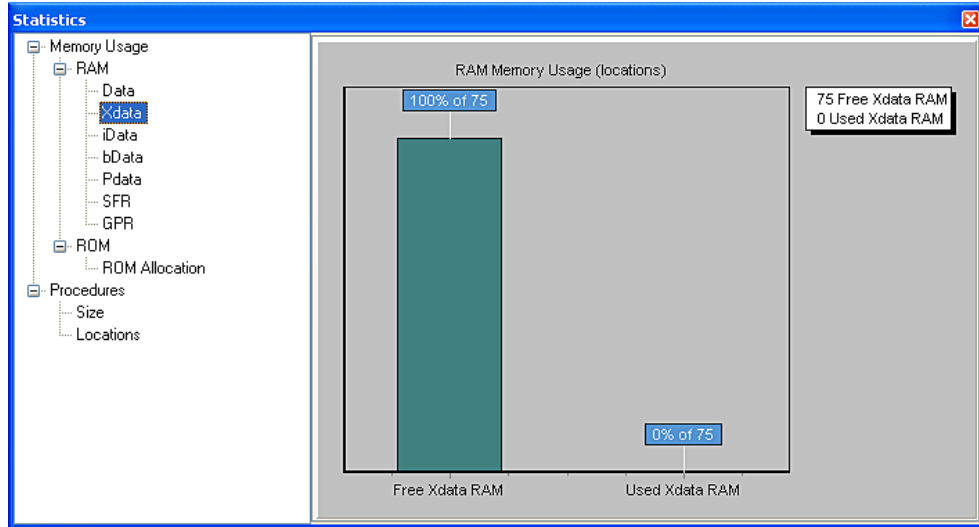
Data Memory

Displays Data memory usage in form of histogram.



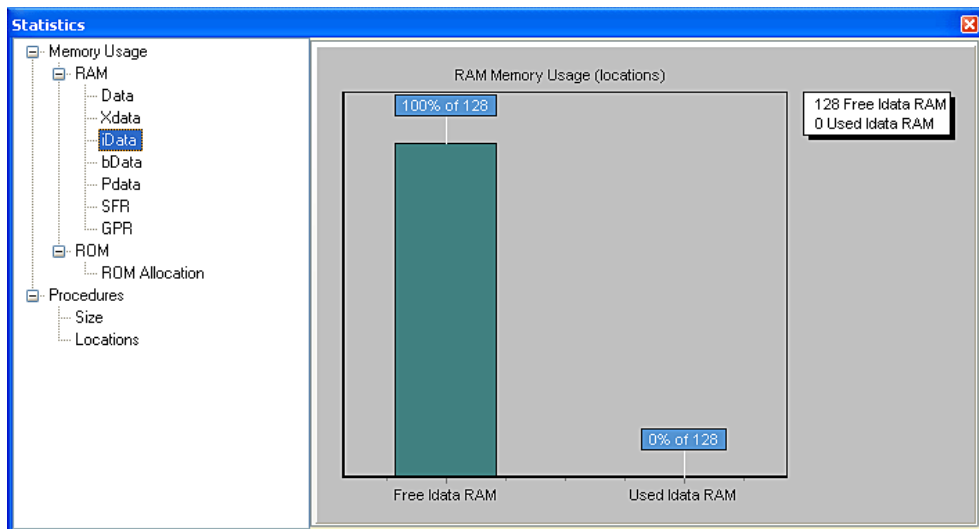
XData Memory

Displays XData memory usage in form of histogram.



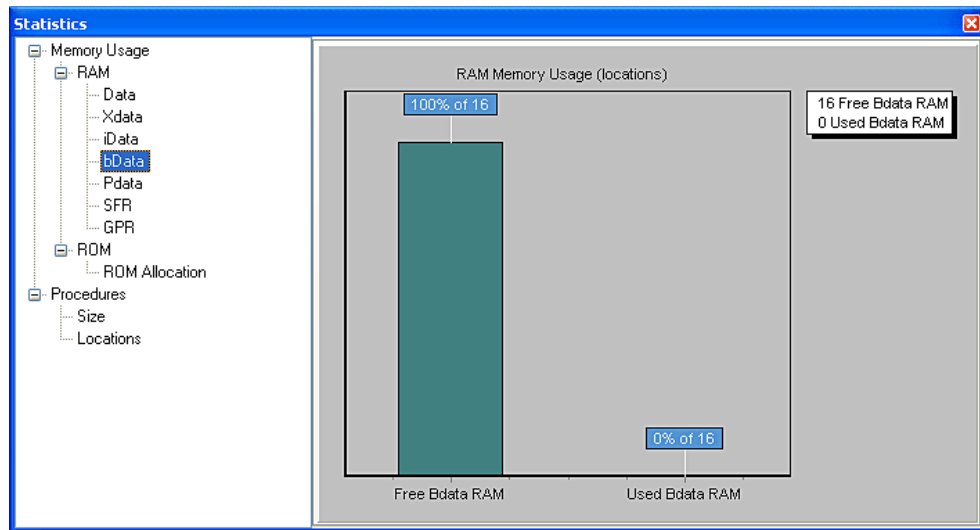
iData Memory

Displays iData memory usage in form of histogram.



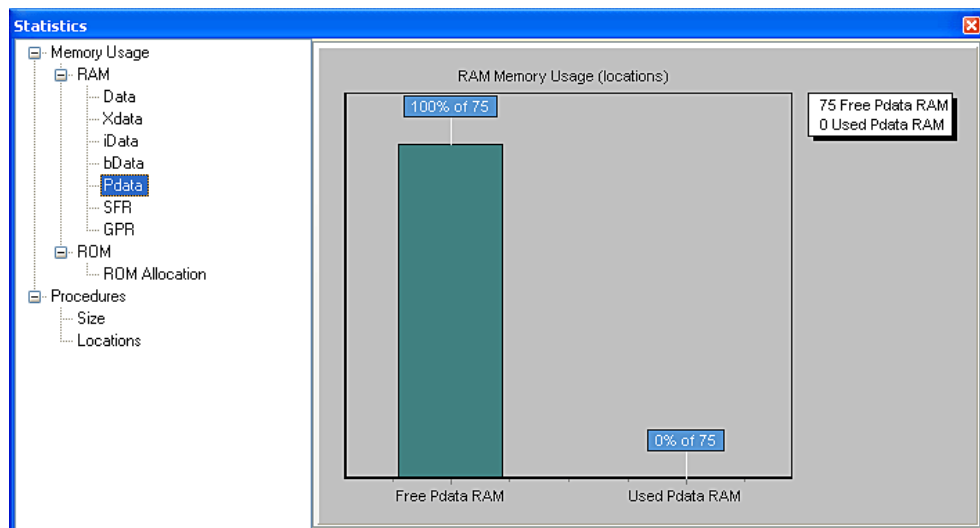
bData Memory

Displays bData memory usage in form of histogram.



PData Memory

Displays PData memory usage in form of histogram.



Special Function Registers

Summarizes all Special Function Registers and their addresses.

Special function registers (SFR)	
Address	Register
0x80	P0
0x81	SP
0x82	DPL
0x82	DP0L
0x83	DPH
0x83	DP0H
0x84	DP1L
0x85	DP1H
0x86	SPDR
0x87	PCON
0x88	TCON
0x89	TMOD
0x8A	TL0
0x8B	TL1
0x8C	TH0

General Purpose Registers

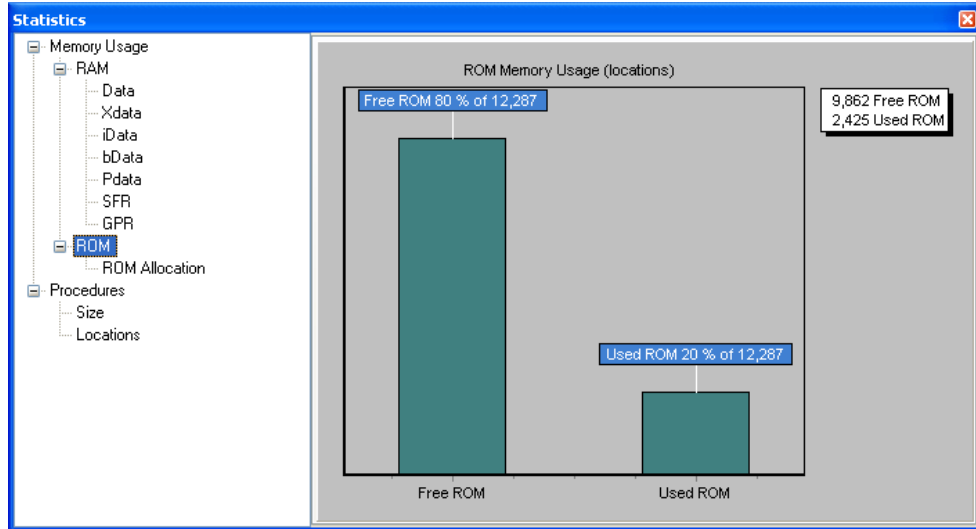
Summarizes all General Purpose Registers and their addresses. Also displays symbolic names of variables and their addresses.

General purpose registers (GPR)	
Address	Register
0x00	R0
0x01	R1
0x02	R2
0x03	R3
0x04	R4
0x05	R5
0x06	R6
0x07	R7
0x09C0	advanced8051_bmp (_advanced8051_bmp)
0xA0	GLCD_CS1 (_GLCD_CS1)
0xA1	GLCD_CS2 (_GLCD_CS2)
0xA2	GLCD_RS (_GLCD_RS)
0xA3	GLCD_RW (_GLCD_RW)
0xA5	GLCD_RST (_GLCD_RST)
0xA4	GLCD_EN (_GLCD_EN)

ROM Memory

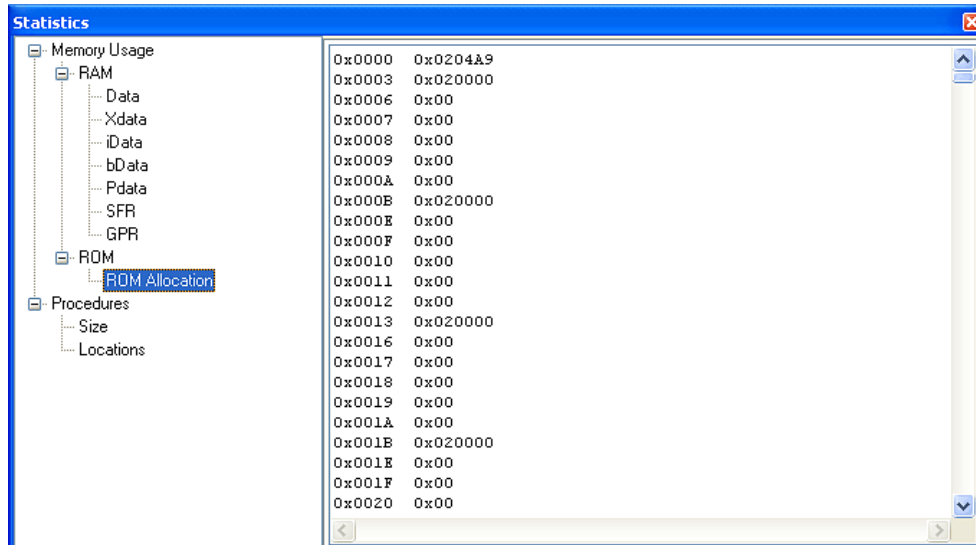
ROM Memory Usage

Displays ROM memory usage in form of histogram.



ROM Memory Allocation

Displays ROM memory allocation.

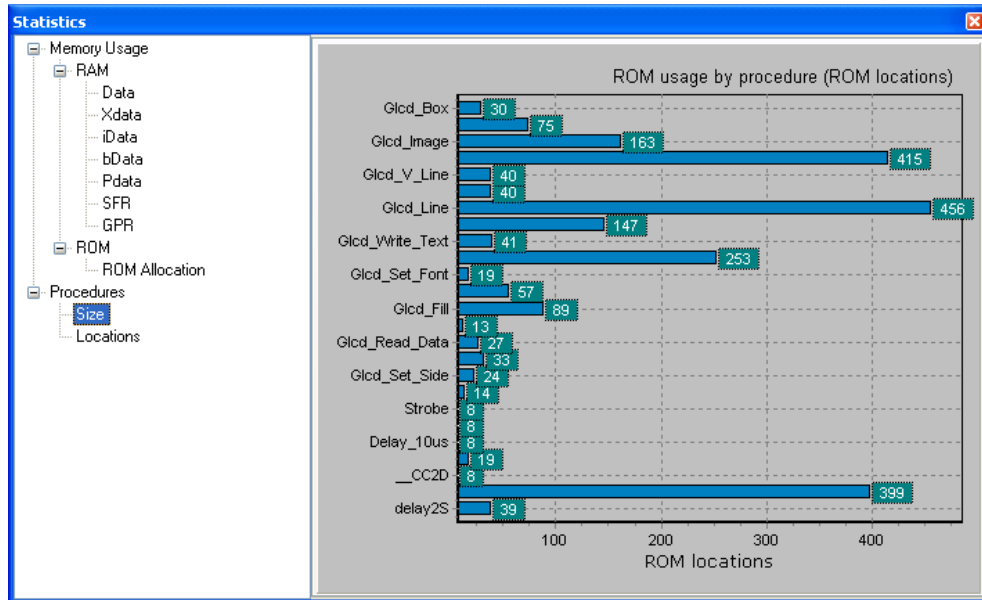


Procedures Windows

Provides overview procedures locations and sizes.

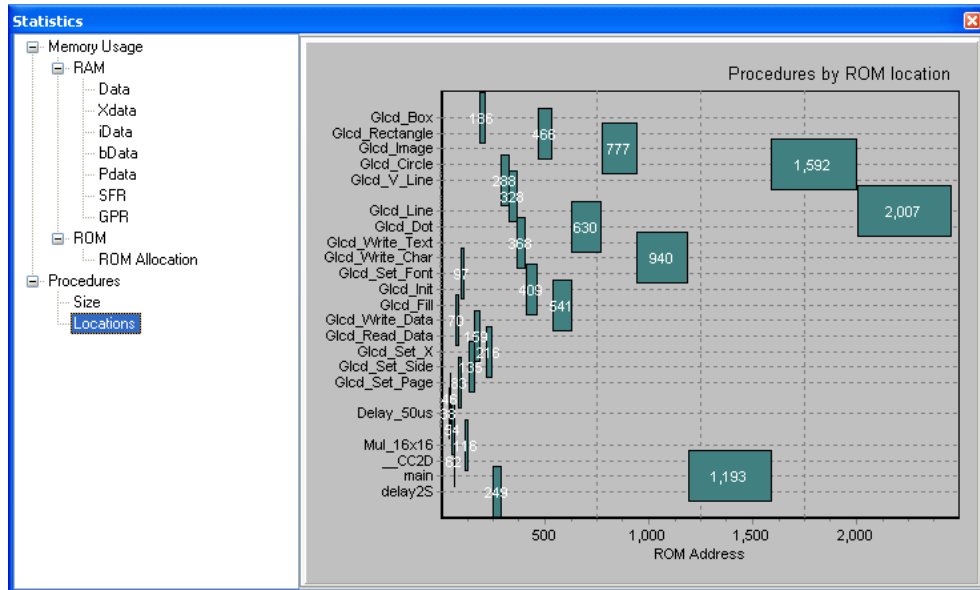
Procedures Size Window

Displays size of each procedure.




Procedures Locations Window

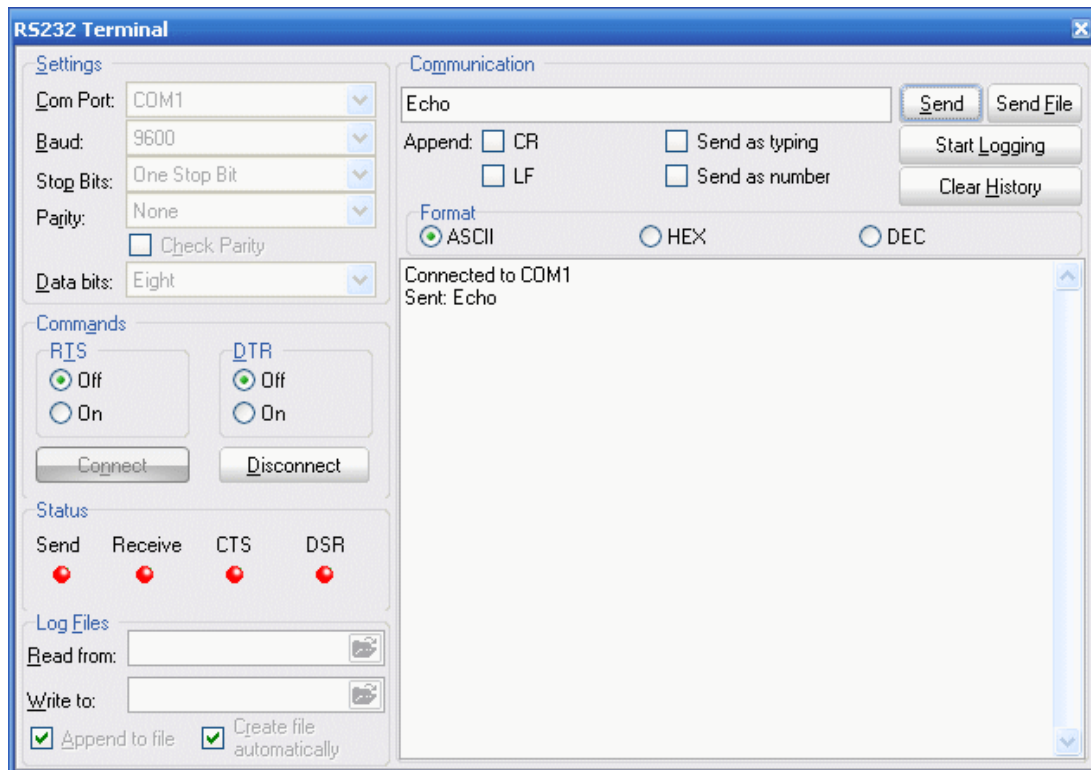
Displays how functions are distributed in microcontroller's memory.




INTEGRATED TOOLS

USART Terminal

The mikroBasic for 8051 includes the USART communication terminal for RS232 communication. You can launch it from the drop-down menu **Tools > USART Terminal** or by clicking the USART Terminal Icon  from Tools toolbar.



ASCII Chart

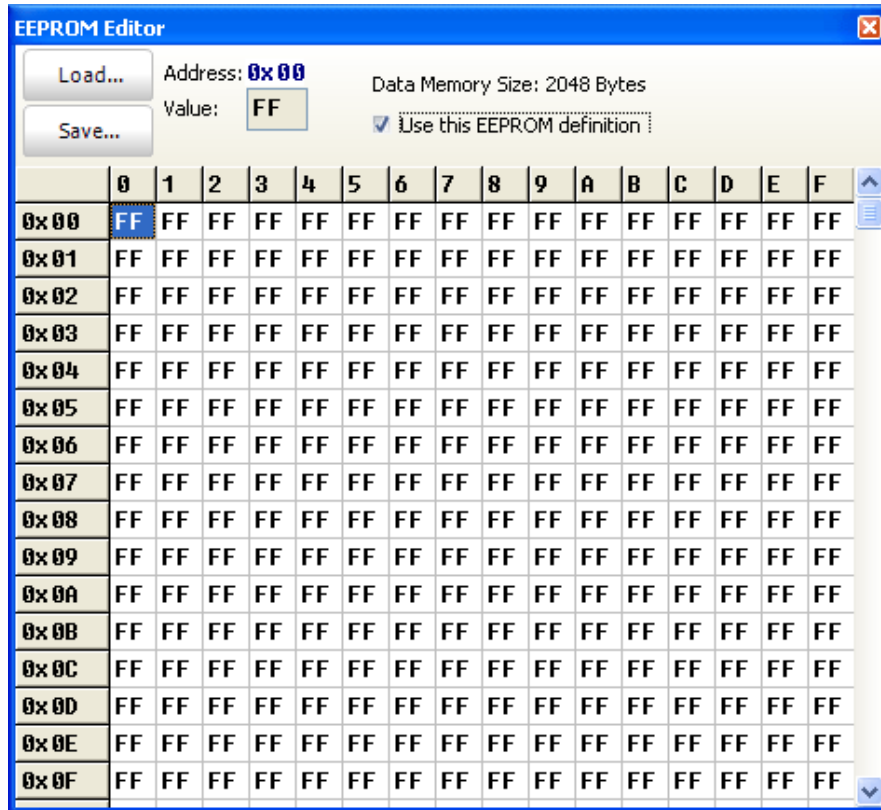
The ASCII Chart is a handy tool, particularly useful when working with LCD display. You can launch it from the drop-down menu **Tools > ASCII chart** or by clicking the View ASCII Chart Icon  from Tools toolbar.

Ascii Chart																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	SPC	!	"	#	\$	%	_	'	()	*	+	,	-	.	/
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	€	□	,	f	„	…	†	‡	^	‰	§	<	œ	□	ž	□
	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	□	'	'	“	”	•	—	—	~	™	š	>	œ	□	ž	Ÿ
	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A		i	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	°	±	²	³	´	µ	¶	·	,	¹	º	»	¼	½	¾	¿
	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ
	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255


EEPROM Editor

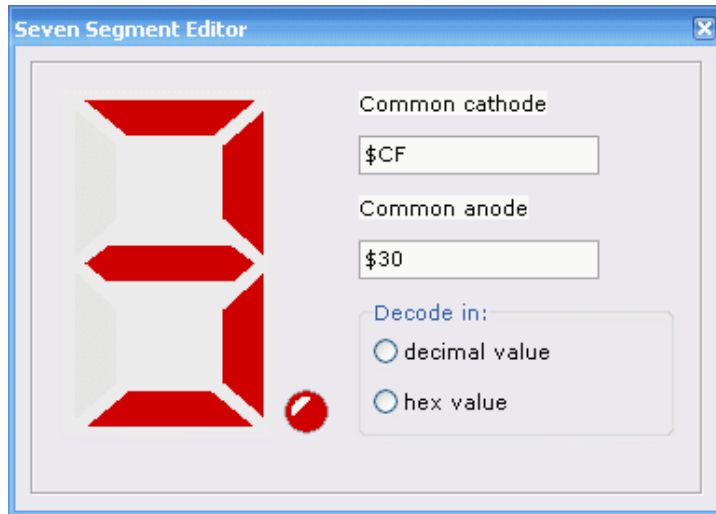
The EEPROM Editor is used for manipulating MCU's EEPROM memory. You can launch it from the drop-down menu **Tools** > **EEPROM Editor**. When Use this EEPROM definition is checked compiler will generate Intel hex file `project_name.ihex` that contains data from EEPROM editor.

When you run mikroElektronika programmer software from mikroBasic for 8051 IDE - `project_name.hex` file will be loaded automatically while `ihex` file must be loaded manually.



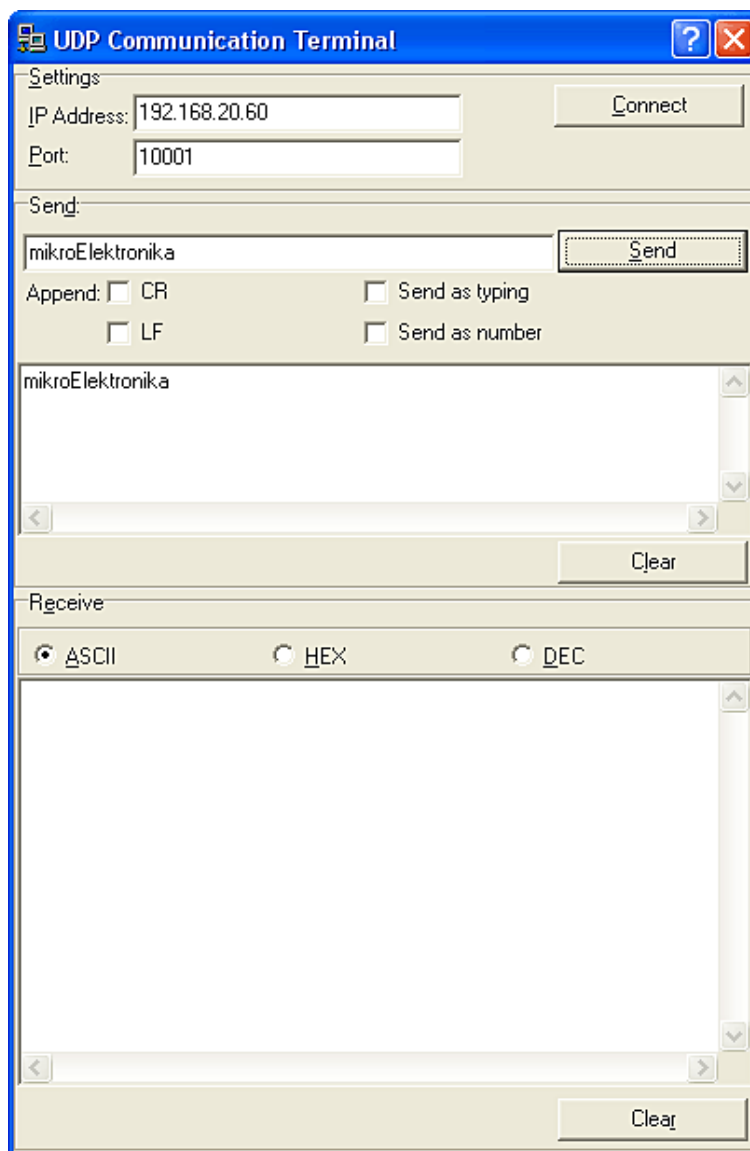
7 Segment Display Decoder

The 7 Segment Display Decoder is a convenient visual panel which returns decimal/hex value for any viable combination you would like to display on 7seg. Click on the parts of 7 segment image to get the requested value in the edit boxes. You can launch it from the drop-down menu **Tools** > **7 Segment Decoder** or by clicking the Seven Segment Icon  from Tools toolbar.



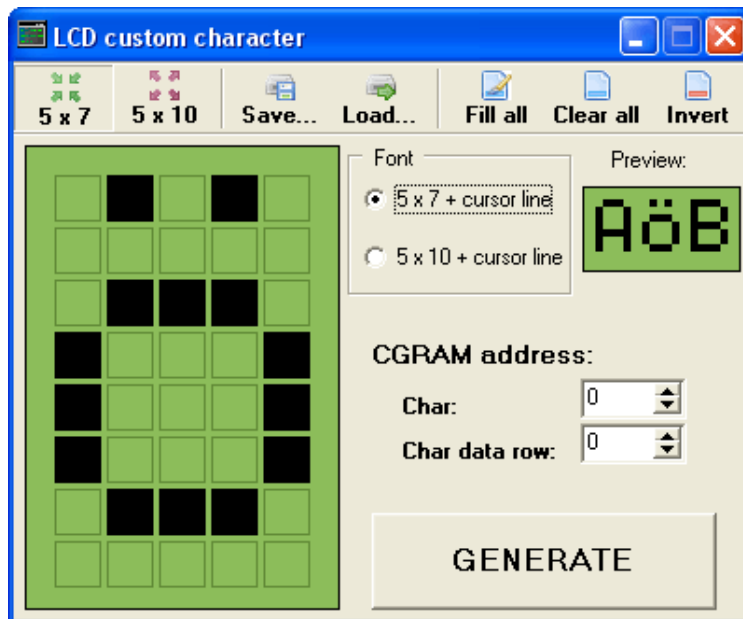
UDP Terminal

The mikroBasic for 8051 includes the UDP Terminal. You can launch it from the drop-down menu **Tools** > **UDP Terminal**.



LCD Custom Character

mikroBasic for 8051 includes the LCD Custom Character. Output is mikroBasic for 8051 compatible code. You can launch it from the drop-down menu **Tools** > **LCD Custom Character**.



OPTIONS

Options menu consists of three tabs: Code Editor, Tools and Output settings

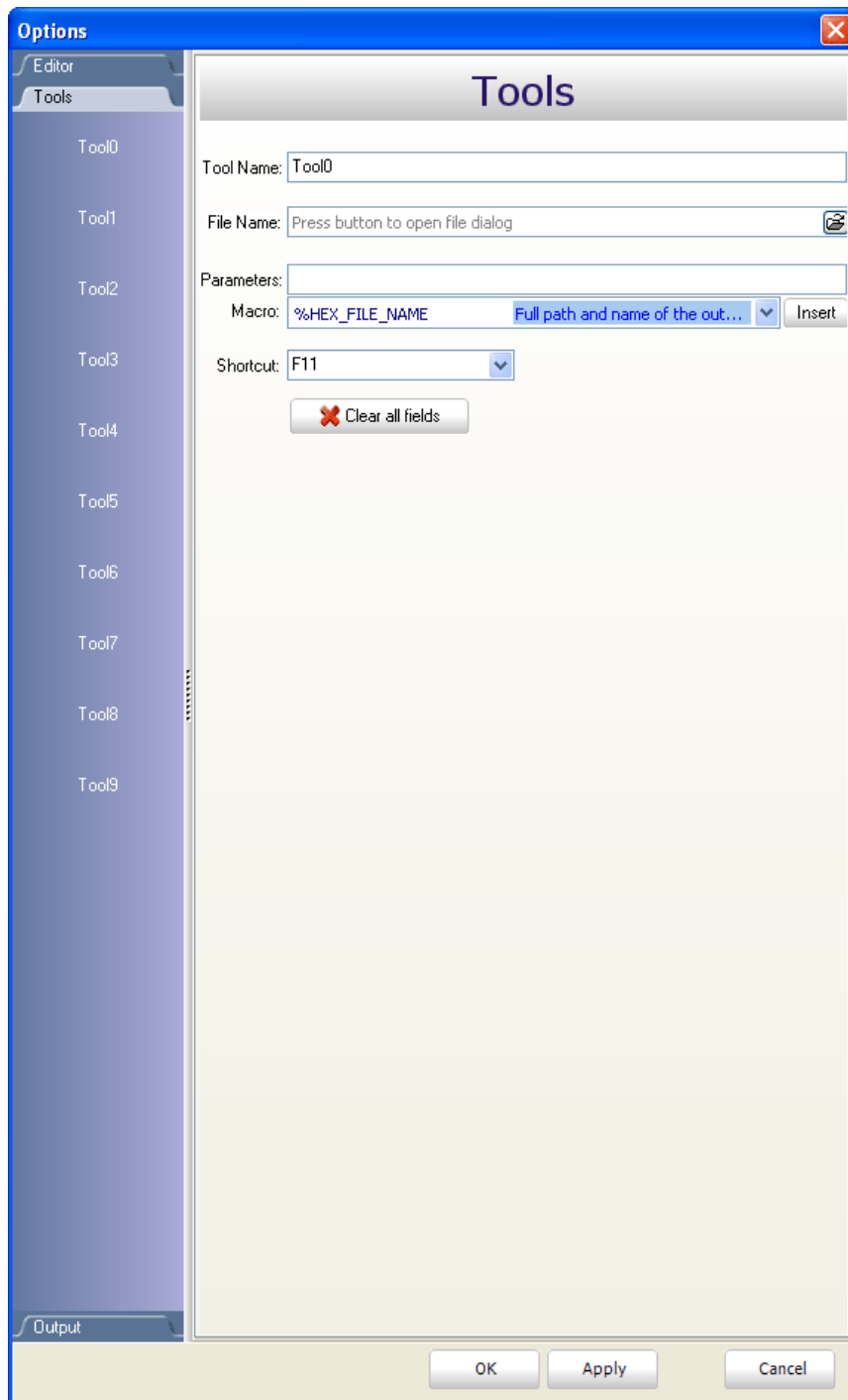
Code editor

The Code Editor is advanced text editor fashioned to satisfy needs of professionals.

Tools

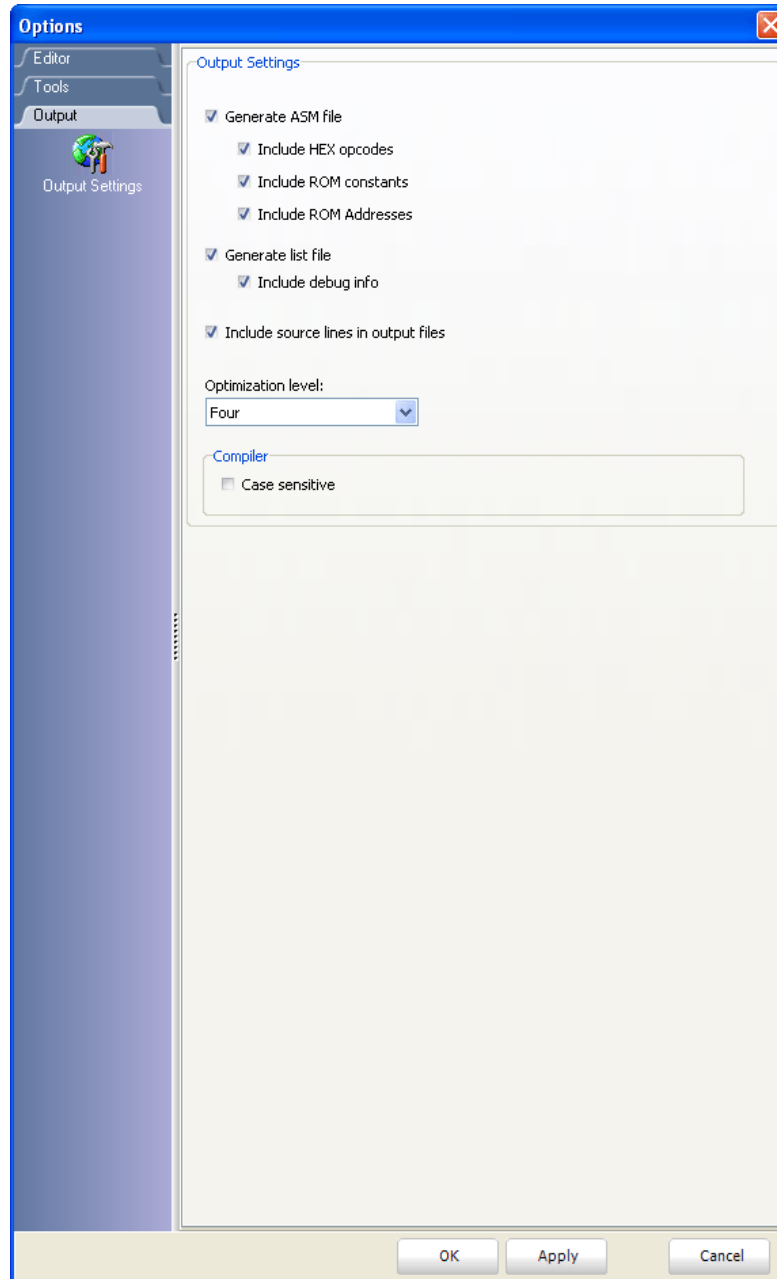
The mikroBasic for 8051 includes the Tools tab, which enables the use of shortcuts to external programs, like Calculator or Notepad.

You can set up to 10 different shortcuts, by editing Tool0 - Tool9.



Output settings

By modifying Output Settings, user can configure the content of the output files. You can enable or disable, for example, generation of ASM and List file.



REGULAR EXPRESSIONS

Introduction

Regular Expressions are a widely-used method of specifying patterns of text to search for. Special metacharacters allow you to specify, for instance, that a particular string you are looking for, occurs at the beginning, or end of a line, or contains n recurrences of a certain character.

Simple matches

Any single character matches itself, unless it is a metacharacter with a special meaning described below. A series of characters matches that series of characters in the target string, so the pattern "short" would match "short" in the target string. You can cause characters that normally function as metacharacters or escape sequences to be interpreted by preceding them with a backslash "\ ". For instance, metacharacter "^" matches beginning of string, but "\^" matches character "^", and "\\ " matches "\ ", etc.

Examples :

```
integer matches string 'integer'
\^integer matches string '^integer'
```

Escape sequences

Characters may be specified using a escape sequences: "\n" matches a newline, "\t" a tab, etc. More generally, "\nn", where nn is a string of hexadecimal digits, matches the character whose ASCII value is nn.

If you need wide(Unicode)character code, you can use "\x{nnnn}", where 'nnnn' - one or more hexadecimal digits.

- \xnn - char with hex code nn
- \x{ nnnn} - char with hex code nnnn (one byte for plain text and two bytes for Unicode)
- \t - tab (HT/TAB), same as \x09
- \n - newline (NL), same as \x0a
- \r - car.return (CR), same as \x0d
- \f - form feed (FF), same as \x0c
- \a - alarm (bell) (BEL), same as \x07
- \e - escape (ESC) , same as \x1b

Examples:

```
sub\x20procedure matches 'sub procedure' (note space in the middle)
\tlongint matches 'longint' (predecessed by tab)
```

Character classes

You can specify a character class, by enclosing a list of characters in [], which will match any of the characters from the list. If the first character after the "[" is "^", the class matches any character not in the list.

Examples:

```
count[aeiou]r finds strings 'countar', 'counter', etc. but not  
'countbr', 'countcr', etc.  
count[^aeiou]r finds strings 'countbr', 'countcr', etc. but not  
'countar', 'counter', etc.
```

Within a list, the "-" character is used to specify a range, so that a-z represents all characters between "a" and "z", inclusive.

If you want "-" itself to be a member of a class, put it at the start or end of the list, or escape it with a backslash.

If you want "]" , you may place it at the start of list or escape it with a backslash.

Examples:

```
[-az] matches 'a', 'z' and '-'  
[az-] matches 'a', 'z' and '-'  
[a\ -z] matches 'a', 'z' and '-'  
[a-z] matches all twenty six small characters from 'a' to 'z'  
[\n-\x0D] matches any of #10,#11,#12,#13.  
[\d-t] matches any digit, '-' or 't'.  
[ ]-a] matches any char from ']'..'a'.
```

Metacharacters

Metacharacters are special characters which are the essence of regular expressions. There are different types of metacharacters, described below.

Metacharacters - Line separators

- ^ - start of line
- \$ - end of line
- \A - start of text
- \Z - end of text
- .

- any character in line

Examples:

```

^PORTA - matches string ' PORTA ' only if it's at the beginning of line
PORTA$ - matches string ' PORTA ' only if it's at the end of line
^PORTA$ - matches string ' PORTA ' only if it's the only string in line
PORT.r - matches strings like 'PORTA', 'PORTB', 'PORT1' and so on

```

The "^" metacharacter by default is only guaranteed to match beginning of the input string/text, and the "\$" metacharacter only at the end. Embedded line separators will not be matched by "^" or "\$".

You may, however, wish to treat a string as a multi-line buffer, such that the "^" will match after any line separator within the string, and "\$" will match before any line separator.

Regular expressions works with line separators as recommended at www.unicode.org (<http://www.unicode.org/unicode/reports/tr18/>):

Metacharacters - Predefined classes

```

\w - an alphanumeric character (including "_")
\W - a nonalphanumeric
\d - a numeric character
\D - a non-numeric
\s - any space (same as [ \t\n\r\f ] )
\S - a non space

```

You may use `\w`, `\d` and `\s` within custom character classes.

Example:

```

routi\de - matches strings like 'routile', 'routi6e' and so on, but not
'routine', 'routime' and so on.

```

Metacharacters - Word boundaries

A word boundary ("`\b`") is a spot between two characters that has a "`\w`" on one side of it and a "`\W`" on the other side of it (in either order), counting the imaginary characters off the beginning and end of the string as matching a "`\W`".

```

\b - match a word boundary)
\B - match a non-(word boundary)

```

Metacharacters - Iterators

Any item of a regular expression may be followed by another type of metacharacters - iterators. Using this metacharacters, you can specify number of occurrences of previous character, metacharacter or subexpression.

- * - zero or more ("greedy"), similar to {0,}
- + - one or more ("greedy"), similar to {1,}
- ? - zero or one ("greedy"), similar to {0,1}
- { n } - exactly n times ("greedy")
- { n, } - at least n times ("greedy")
- { n, m } - at least n but not more than m times ("greedy")
- *? - zero or more ("non-greedy"), similar to {0,}?
- +? - one or more ("non-greedy"), similar to {1,}?
- ?? - zero or one ("non-greedy"), similar to {0,1}?
- { n }? - exactly n times ("non-greedy")
- { n, }? - at least n times ("non-greedy")
- { n, m }? - at least n but not more than m times ("non-greedy")

So, digits in curly brackets of the form, { n, m }, specify the minimum number of times to match the item n and the maximum m. The form { n } is equivalent to { n, n } and matches exactly n times. The form { n, } matches n or more times. There is no limit to the size of n or m, but large numbers will chew up more memory and slow down execution.

If a curly bracket occurs in any other context, it is treated as a regular character.

Examples:

```
count.*r  B- matches strings like 'counter', 'countelkjdf1kj9r' and
'countr'
count.+r  - matches strings like 'counter', 'countelkjdf1kj9r' but
not 'countr'
count.?r  - matches strings like 'counter', 'countar' and 'countr'
but not 'countelkj9r'
counte{ 2 } r - matches string 'counteer'
counte{ 2, } r - matches strings like 'counteer', 'counteeer',
'counteeer' etc.
counte{ 2, 3 } r - matches strings like 'counteer', or 'counteeer' but
not 'counteeer'
```

A little explanation about "greediness". "Greedy" takes as many as possible, "non-greedy" takes as few as possible.

For example, 'b+' and 'b*' applied to string 'abbbbc' return 'bbbb', 'b+?' returns 'b', 'b*?' returns empty string, 'b{ 2, 3 }?' returns 'bb', 'b{ 2, 3 }' returns 'bbb'.

Metacharacters - Alternatives

You can specify a series of alternatives for a pattern using "|" to separate them, so that `bit|bat|bot` will match any of "bit", "bat", or "bot" in the target string (as would `b(i|a|o)t`). The first alternative includes everything from the last pattern delimiter ("(", "[", or the beginning of the pattern) up to the first "|", and the last alternative contains everything from the last "|" to the next pattern delimiter. For this reason, it's common practice to include alternatives in parentheses, to minimize confusion about where they start and end.

Alternatives are tried from left to right, so the first alternative found for which the entire expression matches, is the one that is chosen. This means that alternatives are not necessarily greedy. For example: when matching `rou|rout` against "routine", only the "rou" part will match, as that is the first alternative tried, and it successfully matches the target string (this might not seem important, but it is important when you are capturing matched text using parentheses.) Also remember that "|" is interpreted as a literal within square brackets, so if you write `[bit|bat|bot]`, you're really only matching `[biao]`.

Examples:

```
rou(tine|te) - matches strings 'routine' or 'route'.
```

Metacharacters - Subexpressions

The bracketing construct `(...)` may also be used for define regular subexpressions. Subexpressions are numbered based on the left to right order of their opening parenthesis. First subexpression has number '1'

Examples:

```
(int){8,10} matches strings which contain 8, 9 or 10 instances of the 'int'  
routi([0-9]|a+)e matches 'routi0e', 'routile', 'routine',  
'routinne', 'routinne' etc.
```

Metacharacters - Backreferences

Metacharacters `\1` through `\9` are interpreted as backreferences. `\` matches previously matched subexpression #.

Examples:

```
(.)\1+ matches 'aaaa' and 'cc'.  
(+)\1+ matches 'abab' and '123123'  
(["']?) (\d+)\1 matches "13" (in double quotes), or '4' (in single  
quotes) or 77 (without quotes) etc
```

MIKROBASIC FOR 8051 COMMAND LINE OPTIONS

Usage: mikroBasic8051 [-'opts' ['-opts']] ['infile' ['-opts']] [-'opts']]

Infile can be of *.mbas and *.mcl type.

The following parameters and some more (see manual) are valid:

- P : MCU for which compilation will be done.
- FO : Set oscillator.
- SP : Add directory to the search path list.
- N : Output files generated to file path specified by filename.
- B : Save compiled binary files (*.mcl) to 'directory'.
- O : Miscellaneous output options.
- DBG : Generate debug info.
- E : Set memory model opts (S | C | L (small, compact, large)).
- L : Check and rebuild new libraries.
- C : Turn on case sensitivity.

Example:

```
mikroBasic8051.exe -MSF -DBG -pAT89S8253 -ES -O11111114 -fo10
-N"C:\Lcd\Lcd.mbproj" -SP"C:\Program
Files\Mikroelektronika\mikroBasic 8051\defs\"
-SP"C:\Program Files\Mikroelektronika\mikroBasic 8051\uses\"
-SP"C:\Lcd\" "Lcd.mbas" "System.mcl" "Math.mcl"
"Math_Double.mcl" "Delays.mcl" "__Lib_Lcd.mcl" "__Lib_LcdConsts.mcl"
```

Parameters used in the example:

```
-MSF : Short Message Format; used for internal purposes by IDE.
-DBG : Generate debug info.
-pAT89S8253 : MCU AT89S8253 selected.
-ES : Set small memory model.
-O11111114 : Miscellaneous output options.
-fo10 : Set oscillator frequency [in MHz].
-N"C:\Lcd\Lcd.mbproj" -SP"C:\Program
Files\Mikroelektronika\mikroBasic 8051\defs\" : Output files
generated to file path specified by filename.
-SP"C:\Program Files\Mikroelektronika\mikroBasic 8051\defs\" :
Add directory to the search path list.
-SP"C:\Program Files\Mikroelektronika\mikroBasic 8051\uses\" :
Add directory to the search path list.
-SP"C:\Lcd\" : Add directory to the search path list.
"Lcd.mbas" "System.mcl" "Math.mcl" "Math_Double.mcl"
"Delays.mcl" "__Lib_Lcd.mcl" "__Lib_LcdConsts.mcl" : Specify input files.
```


PROJECTS


The mikroBasic 8051 organizes applications into projects, consisting of a single project file (extension `.mbproj`) and one or more source files (extension `.mbas`). mikroBasic for 8051 IDE allows you to manage multiple projects (see Project Manager). Source files can be compiled only if they are part of a project.

The project file contains the following information:

- project name and optional description,
- target device,
- memory model,
- device flags (config word),
- device clock,
- list of the project source files with paths,
- binary files (*.mcl),
- image files,
- other files.

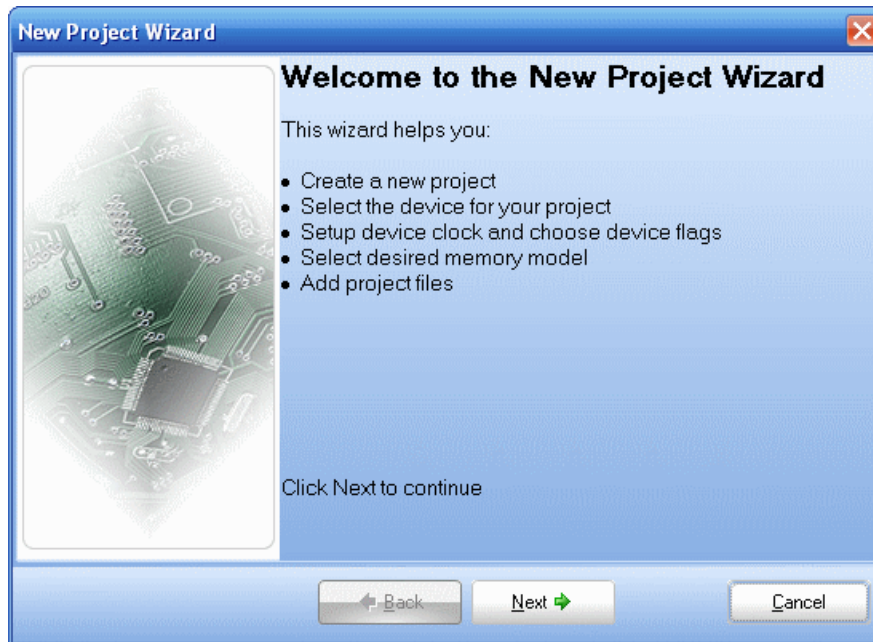
Note that the project does not include files in the same way as preprocessor does, see Add/Remove Files from Project.

New Project

The easiest way to create a project is by means of the New Project Wizard, drop-down menu Project › New Project or by clicking the New Project Icon  from Project Toolbar.

New Project Wizard Steps

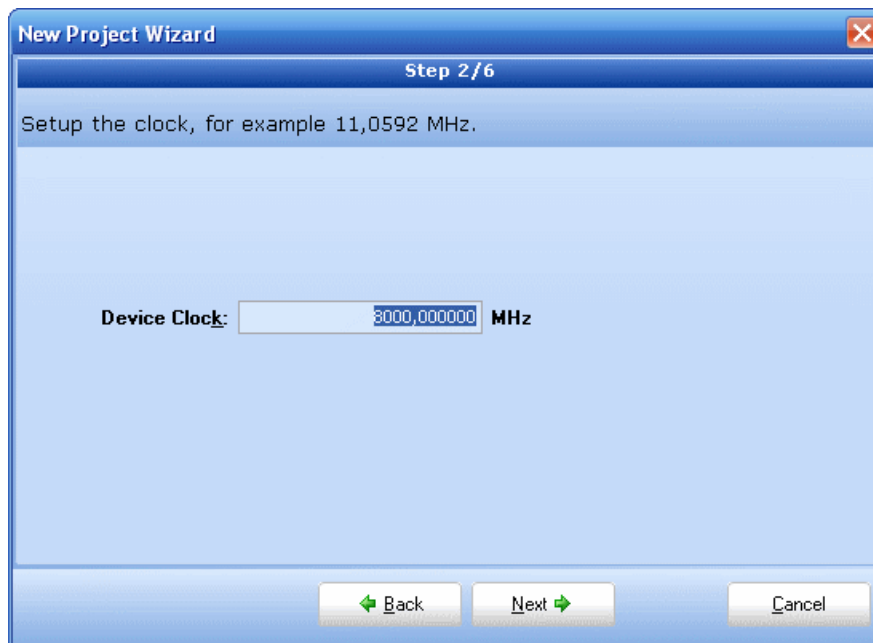
Step One- Provides basic information on settings in the following steps.



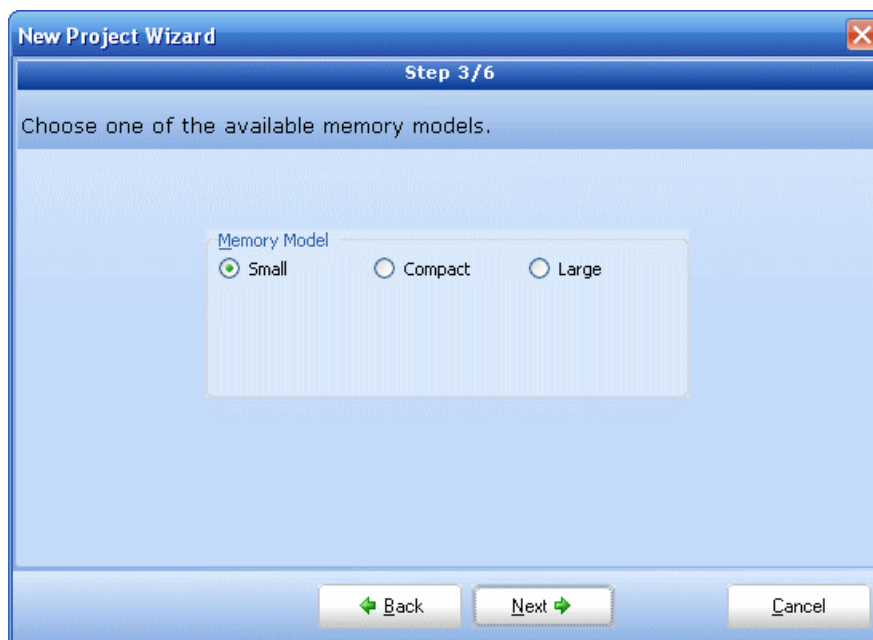
Step Two - Select the device from the device drop-down list.



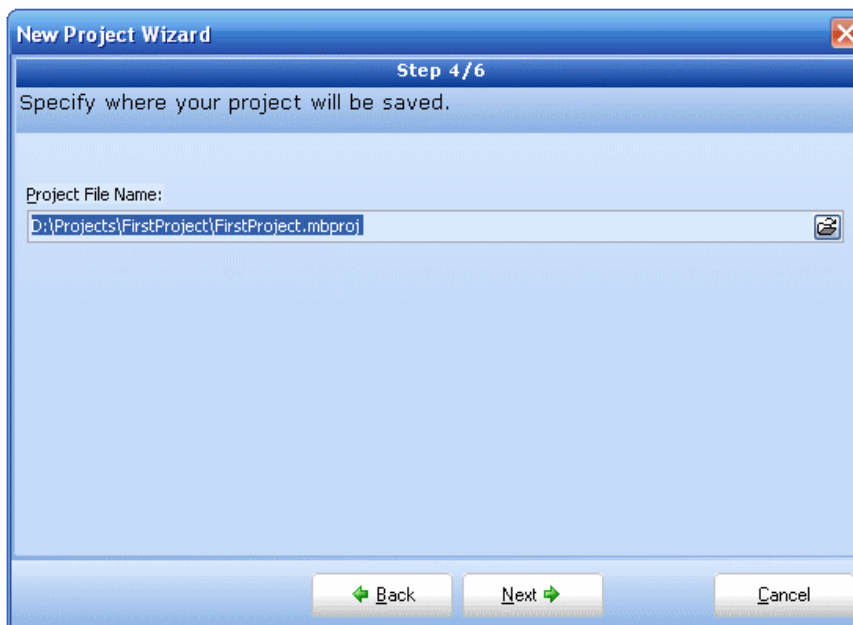
Step Three - enter the oscillator frequency value.



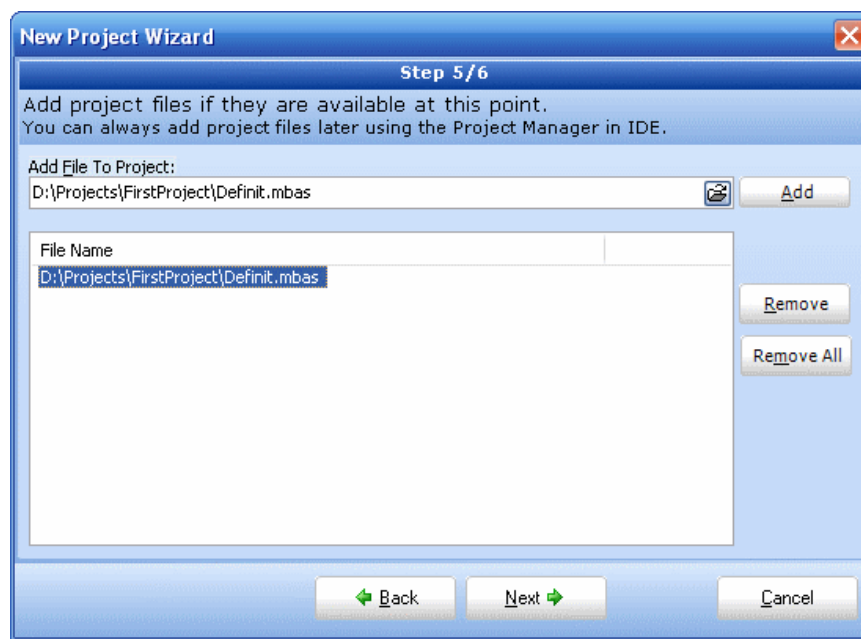
Step Four - Select the desired memory model.



Step Five - Specify the location where your project will be saved.



Step Six - Add project file to the project if they are available at this point. You can always add project files later using Project Manager



Related topics: Project Manager, Project Settings, Memory Model

CUSTOMIZING PROJECTS

Edit Project

You can change basic project settings in the Project Settings window. You can change chip, oscillator frequency, and memory model. Any change in the Project Setting Window affects currently active project only, so in case more than one project is open, you have to ensure that exactly the desired project is set as active one in the Project Manager.

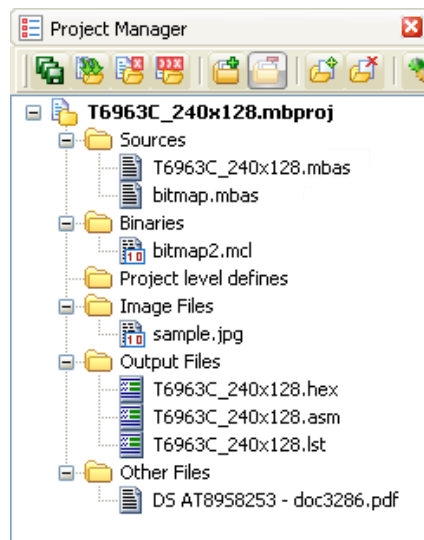
Managing Project Group

mikroBasic for 8051 IDE provides convenient option which enables several projects to be open simultaneously. If you have several projects being connected in some way, you can create a project group.

The project group may be saved by clicking the Save Project Group Icon from the Project Manager window. The project group may be reopened by clicking the Open Project Group Icon. All relevant data about the project group is stored in the project group file (extension .mpg)


ADD/REMOVE FILES FROM PROJECT


The project can contain the following file types:



- .mbas source files
- .mcl binary files
- .pld project level defines files (future upgrade)
- image files
- .hex, .asm and .lst files, see output files. These files can not be added or removed from project.
- other files

The list of relevant source files is stored in the project file (extension .mbproj).

To add source file to the project, click the Add File to Project Icon . Each added source file must be self-contained, i.e. it must have all necessary definitions after preprocessing.

To remove file(s) from the project, click the Remove File from Project Icon .

Note: For inclusion of the module files, use the `include` clause. See File Inclusion for more information.

Related topics: Project Manager, Project Settings, Memory Model

SOURCE FILES



Source files containing Basic code should have the extension `.mbas`. The list of source files relevant to the application is stored in project file with extension `.mbproj`, along with other project information. You can compile source files only if they are part of the project.

Use the preprocessor directive `#include` to include header files with the extension `.h`. Do not rely on the preprocessor to include source files other than headers — see Add/Remove Files from Project for more information.

Managing Source Files


Creating new source file

To create a new source file, do the following:

1. Select **File** › **New Unit** from the drop-down menu, or press `Ctrl+N`, or click the New File Icon  from the File Toolbar.
2. A new tab will be opened. This is a new source file. Select **File** › **Save** from the drop-down menu, or press `Ctrl+S`, or click the Save File Icon  from the File Toolbar and name it as you want.

If you use the New Project Wizard, an empty source file, named after the project with extension `.mbas`, will be created automatically. The mikroBasic 8051 does not require you to have a source file named the same as the project, it's just a matter of convenience.


Opening an existing file

1. Select **File** › **Open** from the drop-down menu, or press Ctrl+O, or click the Open File Icon  from the File Toolbar. In Open Dialog browse to the location of the file that you want to open, select it and click the Open button.
2. The selected file is displayed in its own tab. If the selected file is already open, its current Editor tab will become active.

Printing an open file

1. Make sure that the window containing the file that you want to print is the active window.
2. Select **File** › **Print** from the drop-down menu, or press Ctrl+P.
3. In the Print Preview Window, set a desired layout of the document and click the OK button. The file will be printed on the selected printer.

Saving file

1. Make sure that the window containing the file that you want to save is the active window.
2. Select **File** › **Save** from the drop-down menu, or press Ctrl+S, or click the Save File Icon  from the File Toolbar.

Saving file under a different name

1. Make sure that the window containing the file that you want to save is the active window.
2. Select **File** › **Save As** from the drop-down menu. The New File Name dialog will be displayed.
3. In the dialog, browse to the folder where you want to save the file.
4. In the File Name field, modify the name of the file you want to save.
5. Click the Save button.

Closing file

1. Make sure that the tab containing the file that you want to close is the active tab.
2. Select **File** › **Close** from the drop-down menu, or right click the tab of the file that you want to close and select **Close** option from the context menu.
3. If the file has been changed since it was last saved, you will be prompted to save your changes.

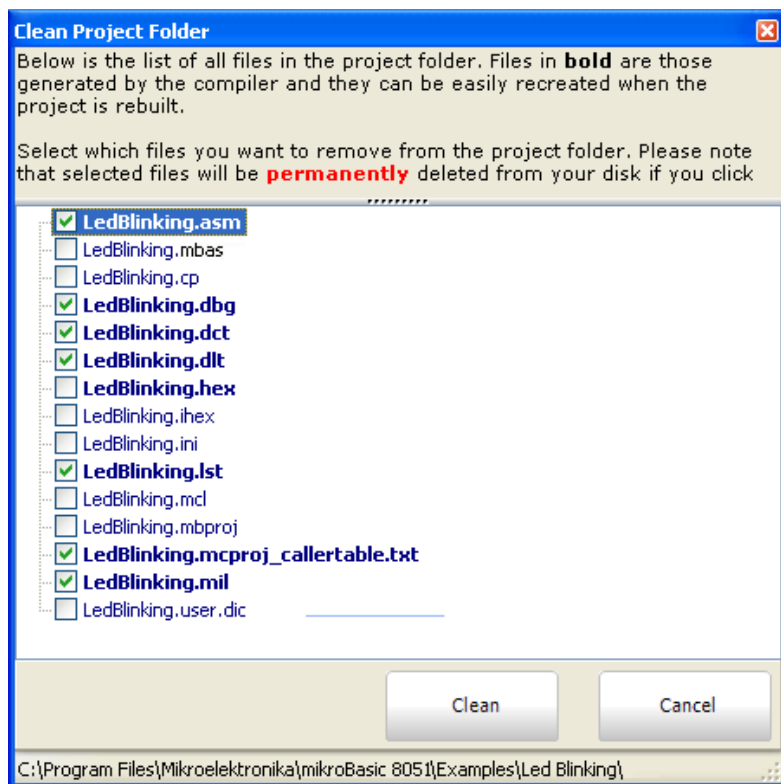
Related topics:File Menu, File Toolbar, Project Manager, Project Settings,

CLEAN PROJECT FOLDER



Clean Project Folder

This menu gives you option to choose which files from your current project you want to delete.

Files marked in **bold** can be easily recreated by building a project. Other files should be marked for deletion only with a great care, because IDE cannot recover them.



COMPILATION

When you have created the project and written the source code, it's time to compile it. Select **Project** > **Build** from the drop-down menu, or click the Build Icon  from the Project Toolbar. If more more than one project is open you can compile all open projects by selecting **Project** > **Build All** from the drop-down menu, or click the Build All Icon  from the Project Toolbar.


Progress bar will appear to inform you about the status of compiling. If there are some errors, you will be notified in the Error Window. If no errors are encountered, the mikroBasic for 8051 will generate output files.

OUTPUT FILES

Upon successful compilation, the mikroBasic for 8051 will generate output files in the project folder (folder which contains the project file `.mbproj`). Output files are summarized in the table below:

Format	Description	File Type
Intel HEX	Intel style hex records. Use this file to program 8051 MCU.	<code>.hex</code>
Binary	mikro Compiled Library. Binary distribution of application that can be included in other projects.	<code>.mcl</code>
List File	Overview of 8051 memory allotment: instruction addresses, registers, routines and labels.	<code>.lst</code>
Assembler File	Human readable assembly with symbolic names, extracted from the List File.	<code>.asm</code>

ASSEMBLY VIEW

After compiling the program in the mikroBasic for 8051, you can click the View Assembly icon  or select **Project** > **View Assembly** from the drop-down menu to review the generated assembly code (`.asm` file) in a new tab window. Assembly is human-readable with symbolic names.

Related topics:Project Menu, Project Toolbar, Error Window, Project Manager, Project Settings

ERROR MESSAGES

Compiler Error Messages:

- "%s" is not valid identifier.
- Unknown type "%s".
- Identifier "%s" was not declared.
- Syntax error: Expected "%s" but "%s" found.
- Argument is out of range "%s".
- Syntax error in additive expression.
- File "%s" not found.
- Invalid command "%s".
- Not enough parameters.
- Too many parameters.
- Too many characters.
- Actual and formal parameters must be identical.
- Invalid ASM instruction: "%s".
- Identifier "%s" has been already declared in "%s".
- Syntax error in multiplicative expression.
- Definition file for "%s" is corrupted.
- ORG directive is currently supported for interrupts only.
- Not enough ROM.
- Not enough RAM.
- External procedure "%s" used in "%s" was not found.
- Internal error: "%s".
- Unit cannot recursively use itself.
- "%s" cannot be used out of loop.
- Supplied and formal parameters do not match ("%s" to "%s").
- Constant cannot be assigned to.
- Constant array must be declared as global.
- Incompatible types ("%s" to "%s").
- Too many characters ("%s").
- Soft_Uart cannot be initialized with selected baud rate/device clock.
- Main label cannot be used in modules.
- Break/Continue cannot be used out of loop.
- Preprocessor Error: "%s".
- Expression is too complicated.
- Duplicated label "%s".
- Complex type cannot be declared here.
- Record is empty.
- Unknown type "%s".
- File not found "%s".
- Constant argument cannot be passed by reference.
- Pointer argument cannot be passed by reference.

- Operator "%s" not applicable to these operands "%s".
- Exit cannot be called from the main block.
- Array parameter must be passed by reference.
- Error occurred while compiling "%s".
- Recursive types are not allowed.
- Adding strings is not allowed, use "strcat" procedure instead.
- Cannot declare pointer to array, use pointer to structure which has array field.
- Return value of the function "%s" is not defined.
- Assignment to for loop variable is not allowed.
- "%s" is allowed only in the main program.
- Start address of "%s" has already been defined.
- Simple constant cannot have a fixed address.
- Invalid date/time format.
- Invalid operator "%s".
- File "%s" is not accessible.
- Forward routine "%s" is missing implementation.
- ";" is not allowed before "else".
- Not enough elements: expected "%s", but "%s" elements found.
- Too many elements: expected "%s" elements.
- "external" is allowed for global declarations only.
- Integer const expected.
- Recursion in definition.
- Array corrupted.
- Arguments cannot have explicit memory specifier.
- Bad storage class.
- Pointer to function required.
- Function required.
- Pointer required.
- Illegal pointer conversion to double.
- Integer type needed.
- Members can not have memory specifier.
- Members can not be of bit or sbit type.
- Too many initializers.
- Too many initializers of subaggregate.
- Already used [%s] .
- Address must be greater than 0.
- [%s] Identifier redefined.
- User abort.
- Expression must be greater than 0.
- Invalid declarator expected '(' or identifier.
- Typdef name redefined: [%s] .
- Declarator error.
- Specifier/qualifier list expected.

- [%s] already used.
- ILevel can be used only with interrupt service routines.
- ';' expected but [%s] found.
- Expected '{ ' .
- [%s] Identifier redefined.
- '(' expected but [%s] found.
- ')' expected but [%s] found.
- 'case' out of switch.
- ':' expected but [%s] found.
- 'default' label out of switch.
- Switch expression must evaluate to integral type.
- While expected but [%s] found.
- 'continue' outside of loop.
- Unreachable code.
- Label redefined.
- Too many chars.
- Unresolved type.
- Arrays of objects containing zero-size arrays are illegal.
- Invalid enumerator.
- ILevel can be used only with interrupt service routines.
- ILevel value must be integral constant.
- ILevel out of range [0..4].
- ')' expected but [%s] found.
- '(' expected but [%s] found.
- 'break' outside of loop or switch.
- Empty char.
- Nonexistent field [%s] .
- Illegal char representation: [%s] .
- Initializer syntax error: multidimension array missing subscript.
- Too many initializers of subaggregate.
- At least one Search Path must be specified.
- Not enough RAM for call satck.
- Parameter [%s] must not be of bit or sbit type.
- Function must not have return value of bit or sbit type.
- Redefinition of [%s] already defined in [%s] .
- Main function is not defined.
- System routine not found for initialization of: [%s] .
- Bad aggregate definition [%s] .
- Unresolved extern [%s] .
- Bad function absolute address [%s] .
- Not enough RAM [%s] .
- Compilation Started.
- Compiled Successfully.
- Finished (with errors): 01 Mar 2008, 14:22:26
- Project Linked Successfully.
- All files Preprocessed in [%s] ms.
- All files Compiled in [%s] ms.
- Linked in [%s] ms.
- Project [%s] completed: [%s] ms.

LINKER ERROR MESSAGES:


- Linker error: "%s" "%s".
- Warning: Variable "%s" is not initialized.
- Warning: Return value of the function "%s" is not defined.
- Hint: Constant "%s" has been declared, but not used.
- Warning: Identifier "%s" overrides declaration in unit "%s".
- Constant "%s" was not found.
- Address of the routine has already been defined.
- Duplicated label "%s".
- File "%s" not found.

HINT MESSAGES:

- Hint: Variable "%s" has been declared, but not used.
- Warning: Variable "%s" is not initialized.
- Warning: Return value of the function "%s" is not defined.
- Hint: Constant "%s" has been declared, but not used.
- Warning: Identifier "%s" overrides declaration in unit "%s".
- Warning: Generated baud rate is "%s" bps (error = "%s" percent).
- Warning: Result size may exceed destination array size.
- Warning: Infinite loop.
- Warning: Implicit typecast performed from "%s" to "%s".
- Hint: Unit "%s" has been recompiled.
- Hint: Variable "%s" has been eliminated by optimizer.
- Warning: Implicit typecast of integral value to pointer
- Warning: Library "%s" was not found in search path.
- Warning: Interrupt context saving has been turned off.
- Hint: Compiling unit "%s".

SOFTWARE SIMULATOR OVERVIEW

The Source-level Software Simulator is an integral component of the mikroBasic for 8051 environment. It is designed to simulate operations of the 8051 MCUs and assist the users in debugging Basic code written for these devices.

After you have successfully compiled your project, you can run the Software Simulator by selecting **Run > Start Debugger** from the drop-down menu, or by clicking the Start Debugger Icon  from the Debugger Toolbar. Starting the Software Simulator makes more options available: Step Into, Step Over, Step Out, Run to Cursor, etc. Line that is to be executed is color highlighted (blue by default).

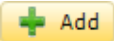
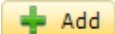
Note: The Software Simulator simulates the program flow and execution of instruction lines, but it cannot fully emulate 8051 device behavior, i.e. it doesn't update timers, interrupt flags, etc.

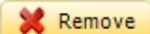
Watch Window

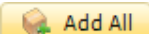
The Software Simulator Watch Window is the main Software Simulator window which allows you to monitor program items while simulating your program. To show the Watch Window, select **View > Debug Windows > Watch** from the drop-down menu.

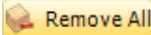
The Watch Window displays variables and registers of the MCU, along with their addresses and values.

There are two ways of adding variable/register to the watch list:

- by its real name (variable's name in "Basic" code). Just select desired variable/register from **Select variable from list** drop-down menu and click the Add Button  .
- by its name ID (assembly variable name). Simply type name ID of the variable/register you want to display into **Search the variable by assembly name** box and click the Add Button  .

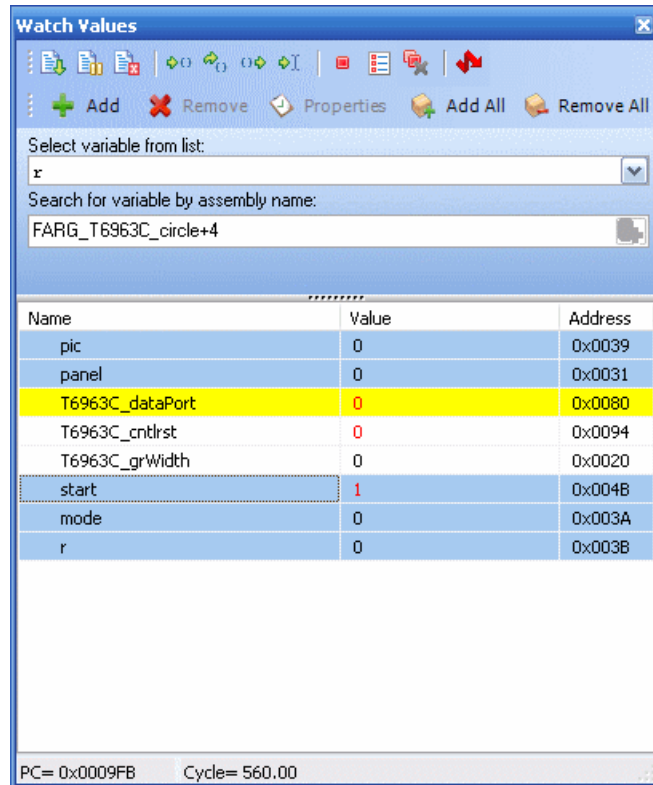
Variables can also be removed from the Watch window, just select the variable that you want to remove and then click the Remove Button  .

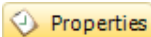
Add All Button  adds all variables.

Remove All Button  removes all variables.

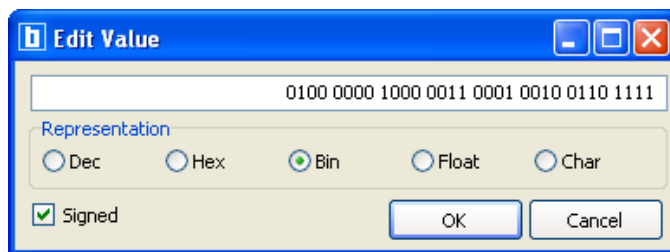
You can also expand/collapse complex variables, i.e. struct type variables, strings...

Values are updated as you go through the simulation. Recently changed items are colored red.



Double clicking a variable or clicking the Properties Button  opens the Edit Value window in which you can assign a new value to the selected variable/register. Also, you can choose the format of variable/register representation between decimal, hexadecimal, binary, float or character. All representations except float are unsigned by default. For signed representation click the check box next to the **Signed** label.

An item's value can be also changed by double clicking item's value field and typing the new value directly.

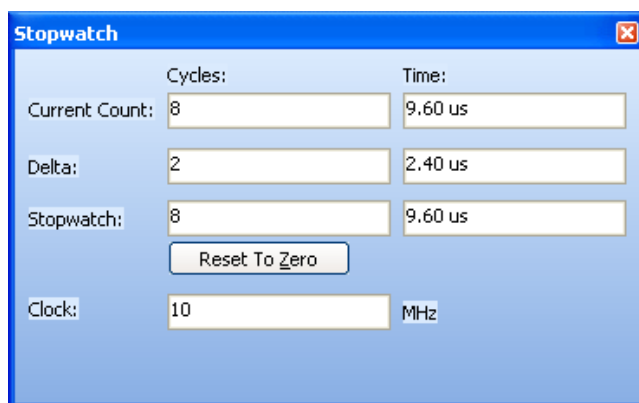


Stopwatch Window

The Software Simulator Stopwatch Window is available from the drop-down menu, **View > Debug Windows > Stopwatch**.

The Stopwatch Window displays a current count of cycles/time since the last Software Simulator action. Stopwatch measures the execution time (number of cycles) from the moment Software Simulator has started and can be reset at any time. Delta represents the number of cycles between the lines where Software Simulator action has started and ended.

Note: The user can change the clock in the Stopwatch Window, which will recalculate values for the latest specified frequency. Changing the clock in the Stopwatch Window does not affect actual project settings – it only provides a simulation.



RAM Window

The Software Simulator RAM Window is available from the drop-down menu, **View › Debug Windows › RAM**.

The RAM Window displays a map of MCU's RAM, with recently changed items colored red. You can change value of any field by double-clicking it.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	BC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	...
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00	00	00	00	01	00	00	00	00	...
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0080	00	BC	55	0E	00	00	00	00	00	00	00	00	00	00	00	00	...
0090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

SOFTWARE SIMULATOR OPTIONS

Name	Description:
Start Debugger	Start Software Simulator.
Run/Pause Debugger	Run or pause Software Simulator.
Stop Debugger	Stop Software Simulator.
Toggle Breakpoints	Toggle breakpoint at the current cursor position. To view all breakpoints, select Run > View Breakpoints from the drop-down menu. Double clicking an item in the Breakpoints Window List locates the breakpoint.
Run to cursor	Execute all instructions between the current instruction and cursor position.
Step Into	Execute the current Basic (single or multi-cycle) instruction, then halt. If the instruction is a routine call, enter the routine and halt at the first instruction following the call.
Step Over	Execute the current Basic (single or multi-cycle) instruction, then halt.
Step Out	Execute all remaining instructions in the current routine, return and then halt.

Related topics: Run Menu, Debug Toolbar

CREATING NEW LIBRARY

mikroBasic for 8051 allows you to create your own libraries. In order to create a library in mikroBasic for 8051 follow the steps bellow:

1. Create a new Basic source file, see Managing Source Files
2. Save the file in the compiler's Uses folder:
`DriveName:\Program Files\Mikroelektronika\mikroBasic
8051\Uses__Lib_Example.mbas`
3. Write a code for your library and save it.
4. Add `__Lib_Example.mbas` file in some project, see Project Manager.
Recompile the project.
5. Compiled file `__Lib_Example.mcl` should appear in `...\mikroBasic
8051\Uses\` folder.
6. Open the definition file for the MCU that you want to use. This file is placed in the compiler's Defs folder:
`DriveName:\Program Files\Mikroelektronika\mikroBasic
8051\Defs\`
 and it is named `MCU_NAME.mlk`, for example `AT89S8253.mlk`
7. Add the `Library_Alias` and `Library_Name` at the end of the definition file, for example `#pragma SetLib([Example_Library, __Lib_Example])`
8. Add Library to `mlk` file for each MCU that you want to use with your library.
9. Click Refresh button in Library Manager

Multiple Library Versions

Library Alias represents unique name that is linked to corresponding Library `.mcl` file. For example UART library for AT89S8253 is different from UART library for AT89S4051 MCU. Therefore, two different UART Library versions were made, see `mlk` files for these two MCUs. Note that these two libraries have the same Library Alias (UART) in both `mlk` files. This approach enables you to have identical representation of UART library for both MCUs in Library Manager.

Related topics: Library Manager, Project Manager, Managing Source Files

CHAPTER

3

mikroBasic for 8051 Specific

The following topics cover the specifics of mikroBasic compiler:

- Basic Standard Issues
- Predefined Globals and Constants
- Accessing Individual Bits
- Interrupts
- 8051 Pointers
- Linker Directives
- Built-in Routines
- Code Optimization

BASIC STANDARD ISSUES

Divergence from the Basic Standard

Function recursion is not supported because of no easily-usable stack and limited memory 8051 Specific

Basic Language Exstensions

mikroBasic for 8051 has additional set of keywords that do not belong to the standard Basic language keywords:

- code
- data
- idata
- bdata
- xdata
- pdata
- small
- compact
- large
- at
- sbit
- bit
- sfr
- ilevel

Related topics: Keywords, 8051 Specific

PREDEFINED GLOBALS AND CONSTANTS

In order to facilitate 8051 programming, mikroBasic for 8051 implements a number of predefined globals and constants.

SFRs and related constants

All 8051 SFRs are implicitly declared as global variables of `volatile word` type. These identifiers have an external linkage, and are visible in the entire project. When creating a project, the mikroBasic for 8051 will include an appropriate (*.mbas) file from defs folder, containing declarations of available SFRs and constants (such as PORTB, ADPCFG, etc). All identifiers are in upper case, identical to nomenclature in the Microchip datasheets.

For a complete set of predefined globals and constants, look for “Defs” in the mikroBasic for 8051 installation folder, or probe the Code Assistant for specific letters (Ctrl+Space in the Code Editor).

Math constants

In addition, several commonly used math constants are predefined in mikroBasic for 8051:

```
PI           = 3.1415926
PI_HALF     = 1.5707963
TWO_PI      = 6.2831853
E           = 2.7182818
```

ACCESSING INDIVIDUAL BITS

The mikroBasic for 8051 allows you to access individual bits of 8-bit variables. It also supports sbit and bit data types

Accessing Individual Bits Of Variables

Simply use the direct member selector (.) with a variable, followed by one of identifiers 0, 1, ... , 15 with 15 being the most significant bit.

There is no need of any special declarations. This kind of selective access is an intrinsic feature of mikroBasic for 8051 and can be used anywhere in the code. Identifiers 0-15 are not case sensitive and have a specific namespace. You may override them with your own members 0-15 within any given structure.

If you are familiar with a particular MCU, you can also access bits by name:

```
' Clear Bit 3 on Port0  
P0.B3 = 0
```

See Predefined Globals and Constants for more information on register/bit names.

sbit type

The mikroBasic Compiler have sbit data type which provides access to bit-addressable SFRs. For example:

```
dim LEDA as sbit at P0.0  
dim Name as sbit at sfr-name.<Bbit-position>
```

The previously declared SFR (sfr-name) is the base address for the sbit. It must be evenly divisible by 8. The bit-position (which must be a number from 0-7) follows the dot symbol ('.') and specifies the bit position to access. For example:

```
dim OV as sbit at PSW.2  
dim CY as sbit at PSW.7
```


bit type

The mikroBasic Compiler provides a `bit` data type that may be used for variable declarations. It can not be used for argument lists, and function-return values.

```
dim bf as bit ' bit variable
```

All bit variables are stored in a bit addressable portion 0x20-0x2F segment located in the internal memory area of the 8051. Because this area is only 16 bytes long, a maximum of 128 bit variables may be declared within any one scope.

There are no pointers to bit variables:

```
dim ptr as ^bit ' invalid
```

An array of type bit is not valid:

```
dim arr as array[5] of bit ' invalid
```

Bit variables can not be initialized nor they can be members of structures and unions.

Related topics: Predefined globals and constants

INTERRUPTS

8051 derivates acknowledges an interrupt request by executing a hardware generated LCALL to the appropriate servicing routine ISRs. ISRs are organized in IVT. ISR is defined as a standard function but with the org directive afterwards which connects the function with specific interrupt vector. For example org 0x000B is IVT address of Timer 0 Overflow interrupt source of the AT89S8253.

For more information on interrupts and IVT refer to the specific data sheet.

Function Calls from Interrupt

Calling functions from within the interrupt routine is allowed. The compiler takes care about the registers being used, both in "interrupt" and in "main" thread, and performs "smart" context-switching between them two, saving only the registers that have been used in both threads. It is not recommended to use function call from interrupt. In case of doing that take care of stack depth.

Interrupt Priority Level

8051 MCUs has possibilty to assign different priority level trough setting appropriate values to coresponding SFRs. You should also assign ISR same priority level by `ilevel` keyword followed by interrupt priority number.

Available interrupt priority levels are: 0 (default), 1, 2 and 3.

```
sub procedure Timer0ISR() org 0x000B ilevel 2
    //set Timer0ISR to be ISR for Timer 0 Overflow priority level 2.
end sub
```

Related topics: Basic standard issues

LINKER DIRECTIVES

mikroBasic for 8051 uses internal algorithm to distribute objects within memory. If you need to have a variable or routine at the specific predefined address, use the linker directives `absolute` and `org`.

Note: You must specify an even address when using the linker directives.

Directive `absolute`

The directive `absolute` specifies the starting address in RAM for a variable. If the variable spans more than 1 word (16-bit), higher words will be stored at the consecutive locations.

The `absolute` directive is appended to the declaration of a variable:

```
dim x as word absolute 0x32
' Variable x will occupy 1 word (16 bits) at address 0x32

dim y as longint absolute 0x34
' Variable y will occupy 2 words at addresses 0x34 and 0x36
```

Be careful when using `absolute` directive, as you may overlap two variables by accident. For example:

```
dim i as word absolute 0x42
' Variable i will occupy 1 word at address 0x42;

dim jj as longint absolute 0x40
' Variable will occupy 2 words at 0x40 and 0x42; thus,
' changing i changes jj at the same time and vice versa
```

Note: You must specify an even address when using the directive `absolute`.

Directive `org`

The directive `org` specifies the starting address of a routine in ROM. It is appended to the declaration of routine. For example:

```
sub procedure proc(dim par as word) org 0x200
' Procedure will start at the address 0x200;
...
end sub
```

Note: You must specify an even address when using the directive `org`.

BUILT-IN ROUTINES

The mikroBasic for 8051 compiler provides a set of useful built-in utility functions.

The `Lo`, `Hi`, `Higher`, `Highest` routines are implemented as macros. If you want to use these functions you must include `built_in.h` header file (located in the `include` folder of the compiler) into your project.

The `Delay_us` and `Delay_ms` routines are implemented as “inline”; i.e. code is generated in the place of a call, so the call doesn’t count against the nested call limit.

The `Vdelay_ms`, `Delay_Cyc` and `Get_Fosc_kHz` are actual Basic routines. Their sources can be found in `Delays.mbas` file located in the `uses` folder of the compiler.

- `Lo`
- `Hi`
- `Higher`
- `Highest`

- `Inc`
- `Dec`

- `Delay_us`
- `Delay_ms`
- `Vdelay_ms`
- `Delay_Cyc`

- `Clock_Khz`
- `Clock_Mhz`

- `SetFuncCall`
- `Uart_Init`

Lo

Prototype	<code>sub function Lo(number as longint) as byte</code>
Returns	Lowest 8 bits (byte) of <code>number</code> , bits 7..0.
Description	Function returns the lowest byte of <code>number</code> . Function does not interpret bit patterns of <code>number</code> – it merely returns 8 bits as found in register. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Requires	Arguments must be variable of scalar type (i.e. Arithmetic Types and Pointers).
Example	<pre>d = 0x1AC30F4 tmp = Lo(d) ' Equals 0xF4</pre>

Hi

Prototype	<code>sub function Hi(number as longint) as byte</code>
Returns	Returns next to the lowest byte of <code>number</code> , bits 8..15.
Description	Function returns next to the lowest byte of <code>number</code> . Function does not interpret bit patterns of <code>number</code> – it merely returns 8 bits as found in register. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Requires	Arguments must be variable of scalar type (i.e. Arithmetic Types and Pointers).
Example	<pre>d = 0x1AC30F4 tmp = Hi(d) ' Equals 0x30</pre>

Higher

Prototype	<code>sub function Higher(number as longint) as byte</code>
Returns	Returns next to the highest byte of <code>number</code> , bits 16..23.
Description	Function returns next to the highest byte of <code>number</code> . Function does not interpret bit patterns of <code>number</code> – it merely returns 8 bits as found in register. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Requires	Arguments must be variable of scalar type (i.e. Arithmetic Types and Pointers).
Example	<pre>d = 0x1AC30F4 tmp = Higher(d) ' Equals 0xAC</pre>

Highest

Prototype	<code>sub function Highest(number as longint) as byte</code>
Returns	Returns the highest byte of number, bits 24..31.
Description	Function returns the highest byte of <code>number</code> . Function does not interpret bit patterns of <code>number</code> – it merely returns 8 bits as found in register. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Requires	Arguments must be variable of scalar type (i.e. Arithmetic Types and Pointers).
Example	<pre>d = 0x1AC30F4 tmp = Highest(d) ' Equals 0x01</pre>

Inc

Prototype	<code>sub procedure Inc(dim byref par as longint)</code>
Returns	Nothing.
Description	Increases parameter <code>par</code> by 1.
Requires	Nothing.
Example	<pre>p = 4 Inc(p) ' p is now 5</pre>

Dec

Prototype	<code>sub procedure Dec(dim byref par as longint)</code>
Returns	Nothing.
Description	Decreases parameter <code>par</code> by 1.
Requires	Nothing.
Example	<pre>p = 4 Dec(p) ' p is now 3</pre>

Delay_us

Prototype	<code>sub procedure Delay_us(const time_in_us as longword)</code>
Returns	Nothing.
Description	Creates a software delay in duration of <code>time_in_us</code> microseconds (a constant). Range of applicable constants depends on the oscillator frequency. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Requires	Nothing.
Example	<code>Delay_us(1000) ' One millisecond pause</code>

Delay_ms

Prototype	<code>sub procedure Delay_ms(const time_in_ms as longword)</code>
Returns	Nothing.
Description	Creates a software delay in duration of <code>time_in_ms</code> milliseconds (a constant). Range of applicable constants depends on the oscillator frequency. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Requires	Nothing.
Example	<code>Delay_ms(1000) ' One second pause</code>

Vdelay_ms

Prototype	<code>sub procedure Vdelay_ms(time_in_ms as word)</code>
Returns	Nothing.
Description	Creates a software delay in duration of <code>time_in_ms</code> milliseconds (a variable). Generated delay is not as precise as the delay created by <code>Delay_ms</code> . Note that <code>Vdelay_ms</code> is library function rather than a built-in routine; it is presented in this topic for the sake of convenience.
Requires	Nothing.
Example	<code>pause = 1000 ' ... Vdelay_ms(pause) ' ~ one second pause</code>

Delay_Cyc

Prototype	<code>sub procedure Delay_Cyc(Cycles_div_by_10 as byte)</code>
Returns	Nothing.
Description	Creates a delay based on MCU clock. Delay lasts for 10 times the input parameter in MCU cycles. Note that <code>Delay_Cyc</code> is library function rather than a built-in routine; it is presented in this topic for the sake of convenience. There are limitations for <code>Cycles_div_by_10</code> value. Value <code>Cycles_div_by_10</code> must be between 2 and 257.
Requires	Nothing.
Example	<code>Delay_Cyc(10) ' Hundred MCU cycles pause</code>

Clock_KHz

Prototype	<code>sub function Clock_KHz() as word</code>
Returns	Device clock in KHz, rounded to the nearest integer.
Description	Function returns device clock in KHz, rounded to the nearest integer. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Requires	Nothing.
Example	<code>clk = Clock_kHz()</code>

Clock_MHz

Prototype	<code>sub function Clock_MHz() as byte</code>
Returns	Device clock in MHz, rounded to the nearest integer.
Description	Function returns device clock in MHz, rounded to the nearest integer. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Requires	Nothing.
Example	<code>clk = Clock_Mhz()</code>

SetFuncCall

Prototype	<code>sub procedure SetFuncCall(FuncName as string)</code>
Returns	Nothing.
Description	<p>Function informs the linker about a specific routine being called. SetFuncCall has to be called in a routine which accesses another routine via a pointer.</p> <p>Function prepares the caller tree, and informs linker about the procedure usage, making it possible to link the called routine.</p>
Requires	Nothing.
Example	<pre>sub procedure first(p, q as byte) ... SetFuncCall(second) ' let linker know that we will call the routine 'second' ... end sub</pre>

Uart_Init

Prototype	<code>sub procedure Uart_Init(baud_rate as longword)</code>
Returns	Nothing.
Description	<p>Configures and initializes the UART module.</p> <p>The internal UART module module is set to:</p> <ul style="list-style-type: none"> - 8-bit data, no parity - 1 STOP bit - disabled automatic address recognition - timer1 as baudrate source (mod2 = autoreload 8bit timer) <p>Parameters :</p> <ul style="list-style-type: none"> - <code>baud_rate</code>: requested baud rate <p>Refer to the device data sheet for baud rates allowed for specific Fosc.</p>
Requires	MCU with the UART module and TIMER1 to be used as baudrate source.
Example	<pre>// Initialize hardware UART and establish communication at 2400 bps Uart_Init(2400)</pre>

CODE OPTIMIZATION

Optimizer has been added to extend the compiler usability, cut down the amount of code generated and speed-up its execution. The main features are:

Constant folding

All expressions that can be evaluated in the compile time (i.e. are constant) are being replaced by their results. (3 + 5 -> 8);

Constant propagation

When a constant value is being assigned to a certain variable, the compiler recognizes this and replaces the use of the variable by constant in the code that follows, as long as the value of a variable remains unchanged.

Copy propagation

The compiler recognizes that two variables have the same value and eliminates one of them further in the code.

Value numbering

The compiler "recognizes" if two expressions yield the same result and can therefore eliminate the entire computation for one of them.

"Dead code" elimination

The code snippets that are not being used elsewhere in the programme do not affect the final result of the application. They are automatically removed.

Stack allocation

Temporary registers ("Stacks") are being used more rationally, allowing VERY complex expressions to be evaluated with a minimum stack consumption.

Local vars optimization

No local variables are being used if their result does not affect some of the global or volatile variables.

Better code generation and local optimization

Code generation is more consistent and more attention is paid to implement specific solutions for the code "building bricks" that further reduce output code size.

CHAPTER

4

8051 Specifics

Types Efficiency

First of all, you should know that 8051 ALU, which performs arithmetic operations, is optimized for working with bytes. Although mikroBasic is capable of handling very complex data types, 8051 may choke on them, especially if you are working on some of the older models. This can dramatically increase the time needed for performing even simple operations. Universal advice is to use the smallest possible type in every situation. It applies to all programming in general, and doubly so with microcontrollers. Types efficiency is determined by the part of RAM memory that is used to store a variable/constant. See the example.

Nested Calls Limitations

There are no Nested Calls Limitations, except by RAM size. A Nested call represents a function call to another function within the function body. With each function call, the stack increases for the size of the returned address. Number of nested calls is equal to the capacity of RAM which is left out after allocation of all variables.

Note: There are many different types of derivatives, so it is necessary to be familiar with characteristics and special features of the microcontroller in you are using.

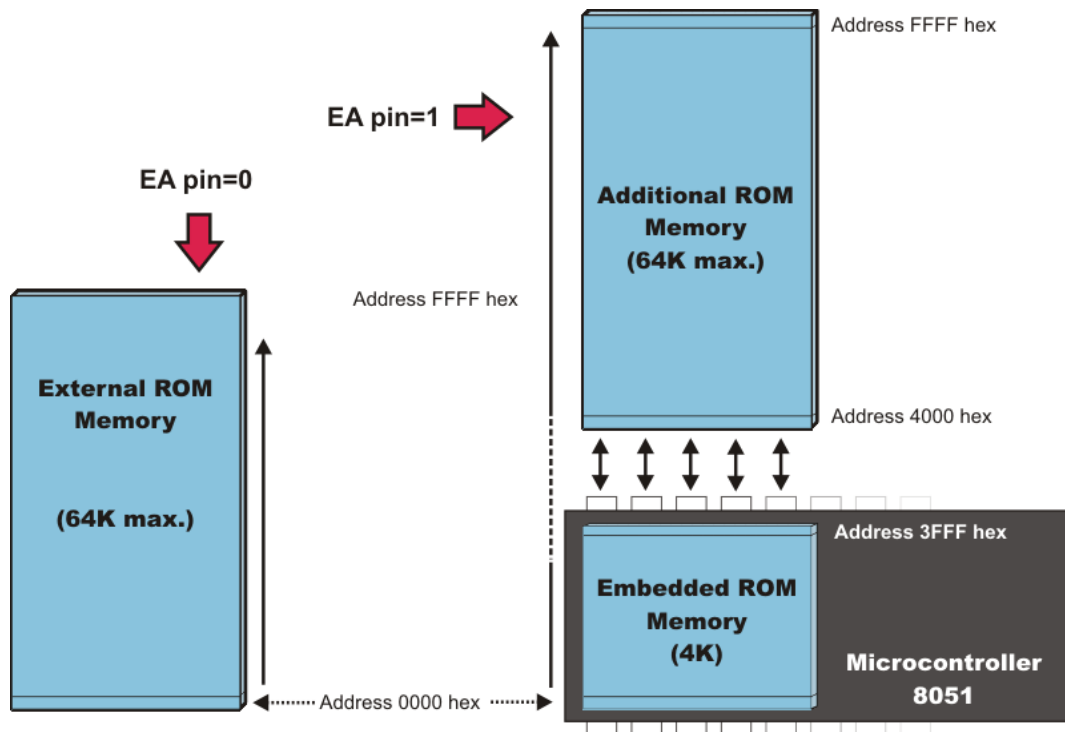
8051 MEMORY ORGANIZATION

The 8051 microcontroller's memory is divided into Program Memory and Data Memory. Program Memory (ROM) is used for permanent saving program being executed, while Data Memory (RAM) is used for temporarily storing and keeping intermediate results and variables.

Program Memory (ROM)

Program Memory (ROM) is used for permanent saving program (CODE) being executed. The memory is read only. Depending on the settings made in compiler, program memory may also used to store a constant variables. The 8051 executes programs stored in program memory only. `code` memory type specifier is used to refer to program memory.

8051 memory organization allows external program memory to be added. How does the microcontroller handle external memory depends on the pin EA logical state.



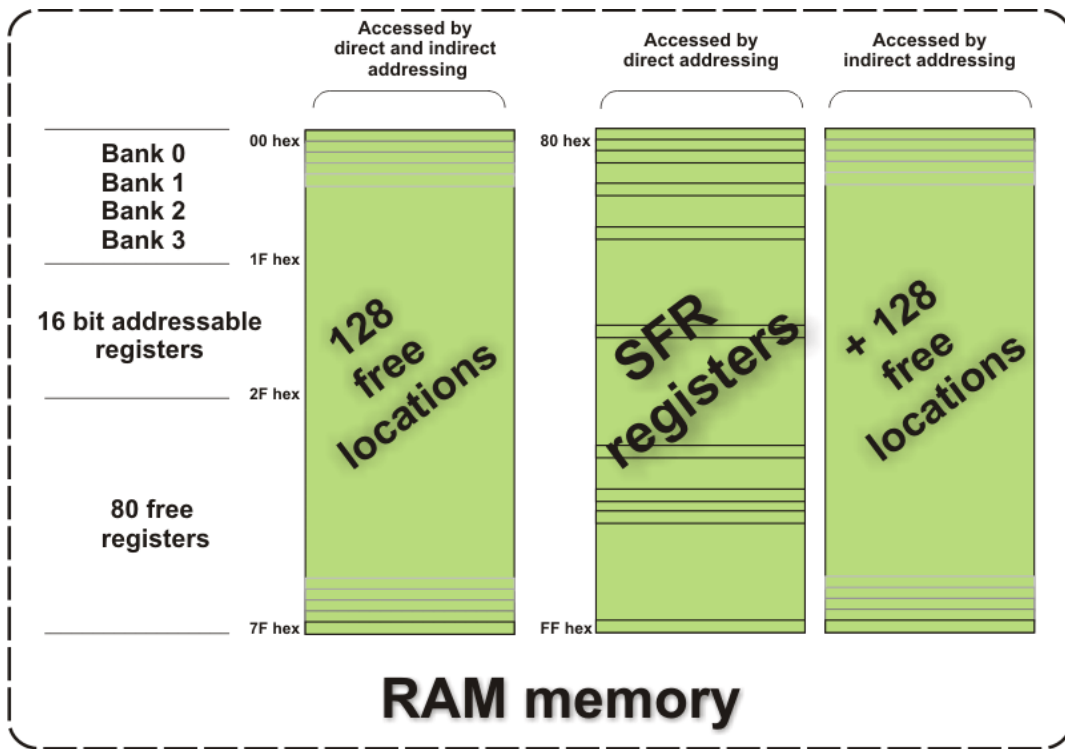
Internal Data Memory

Up to 256 bytes of internal data memory are available depending on the 8051 derivative. Locations available to the user occupy addressing space from 0 to 7Fh, i.e. first 128 registers and this part of RAM is divided in several blocks. The first 128 bytes of internal data memory are both directly and indirectly addressable. The upper 128 bytes of data memory (from 0x80 to 0xFF) can be addressed only indirectly.

Since internal data memory is used for CALL stack also and there is only 256 bytes splitted over few different memory areas fine utilizing of this memory is crucial for fast and compact code. See types efficiency also.

Memory block in the range of 20h to 2Fh is bit-addressable, which means that each bit being there has its own address from 0 to 7Fh. Since there are 16 such registers, this block contains in total of 128 bits with separate addresses (Bit 0 of byte 20h has the bit address 0, and bit 7 of byte 2Fh has the bit address 7Fh).

Three memory type specifiers can be used to refer to the internal data memory: data, idata, and bdata.



External Data Memory

Access to external memory is slower than access to internal data memory. There may be up to 64K Bytes of external data memory. Several 8051 devices provide on-chip XRAM space that is accessed with the same instructions as the traditional external data space. This XRAM space is typically enabled via proper setting of SFR register and overlaps the external memory space. Setting of that register must be manually done in code, before any access to external memory or XRAM space is made.

The mikroBasic for 8051 has two memory type specifiers that refers to external memory space: `xdata` and `pdata`.

SFR Memory

The 8051 provides 128 bytes of memory for Special Function Registers (SFRs). SFRs are bit, byte, or word-sized registers that are used to control timers, counters, serial I/O, port I/O, and peripherals.

Refer to Special Function Registers for more information. See `sbit` also.

Related topics: Accessing individual bits, SFRs, Memory type specifiers, Memory models

MEMORY MODELS

The memory model determines the default memory type to use for function arguments, automatic variables, and declarations that include no explicit memory type. The mikroBasic for 8051 provides three memory models:

- Small
- Compact
- Large

You may also specify the memory model on a function-by-function basis by adding the memory model to the function declaration.

Small memory model generates the fastest, most efficient code. This is default memory model. You may override the default memory type imposed by the memory model by explicitly declaring a variable with a memory type specifier.

Small model

In this model, all variables, by default, reside in the internal data memory of the 8051 system—as if they were declared explicitly using the `data` memory type specifier. In this memory model, variable access is very efficient. However, all objects (that are not explicitly located in another memory area) and the call stack must fit into the internal RAM.

Call Stack size is critical because the stack space used depends on the nesting depth of the various functions.

Compact model

Using the compact model, by default, all variables are allocated in a single page 256 bytes of external data memory of the 8051 system—as if they were explicitly declared using the `pdata` memory type specifier. This memory model can accommodate a maximum of 256 bytes of variables. The limitation is due to the addressing scheme used which is indirect through registers R0 and R1 (`@R0`, `@R1`). This memory model is not as efficient as the small model and variable access is not as fast. However, the compact model is faster than the large model. mikroBasic for 8051 uses the `@R0` and `@R1` operands to access external memory with instructions that use 8 bit wide pointers and provide only the low-order byte of the address. The high-order address byte (or page) is provided by Port 2 on most 8051 derivatives (see data sheet for details).

Large model

In the large model all variables reside in external data memory (which may be up to 64K Bytes). This is the same as if they were explicitly declared using the `xdata` memory type specifier. The `DPTR` is used to address external memory. Instruction set is not optimized for this memory model (access to external memory) so it needs more code than the small or compact model to manipulate with the variables.

```
sub function xadd(dim a1 as byte, dim a2 as byte) as byte; large
'allocate parameters and local variables in xdata space
    return xadd = a1+a2
end sub
```

Related topics: Memory type specifiers, 8051 Memory Organization, Accessing individual bits, SFRs, Project Settings

MEMORY TYPE SPECIFIERS

The mikroBasic for 8051 supports usage of all memory areas. Each variable may be explicitly assigned to a specific memory space by including a memory type specifier in the declaration, or implicitly assigned (based on a memory model).

The following memory type specifiers can be used:

- `code`
- `data`
- `idata`
- `bdata`
- `xdata`
- `pdata`

Memory type specifiers can be included in svariable declaration.
For example:

```
data data_buffer as byte           ' puts data_buffer in data ram
xdata x_data  as array[100] of byte ' puts array in external mem-
ory
idata ibuffer as real              ' puts ibuffer in idata ramm
```

If no memory type is specified for a variable, the compiler locates the variable in the memory space determined by the memory model: Small, Compact, or Large.

code

Description	Program memory (64 KBytes); accessed by opcode <code>MOVC @A+DPTR</code> . The <code>code</code> memory type may be used for constants and functions. This memory is accessed using 16-bit addresses and may be on-chip or external.
Example	<pre>' puts txt in program memory code const txt as string[11] = 'Enter text:'</pre>

data

Description	Directly addressable internal data memory; fastest access to variables (128 bytes). This memory is directly accessed using 8-bit addresses and is the on-chip RAM of the 8051. It has the shortest (fastest) access time but the amount of data is limited in size (to 128 bytes or less).
Example	<pre>' puts x in data ram data x as byte</pre>

idata

Description	Indirectly addressable internal data memory; accessed across the full internal address space (256 bytes). This memory is indirectly accessed using 8-bit addresses and is the on-chip RAM of the 8051. The amount of <code>idata</code> is limited in size (to 128 bytes or less) it is upper 128 addresses of RAM
Example	<pre>' puts x in idata ram idata x as byte</pre>

bdata

Description	Bit-addressable internal data memory; supports mixed bit and byte access (16 bytes). This memory is directly accessed using 8-bit addresses and is the on-chip bit-addressable RAM of the 8051. Variables declared with the <code>bdata</code> type are bit-addressable and may be read and written using bit instructions. For more information about the <code>bdata</code> type refer to the Accessing Individual Bits.
Example	<pre>' puts x in bdata bdata x as byte</pre>

xdata

Description	External data memory (64 KBytes); accessed by opcode MOVX @DPTR. This memory is indirectly accessed using 16-bit addresses and is the external data RAM of the 8051. The amount of xdata is limited in size (to 64K or less).
Example	<pre>' puts x in xdata xdata x as byte</pre>

pdata

Description	Paged (256 bytes) external data memory; accessed by opcode MOVX @Rn. This memory is indirectly accessed using 8-bit addresses and is one 256-byte page of external data RAM of the 8051. The amount of pdata is limited in size (to 256 bytes).
Example	<pre>' puts x in pdata pdata x as byte</pre>

Related topics: 8051 Memory Organization, Memory models, Accessing individual bits, SFRs, Constants, Functions

CHAPTER

5

mikroBasic **Language Reference**

The mikroBasic for 8051 Language Reference describes the syntax, semantics and implementation of the mikroBasic for 8051 language.

The aim of this reference guide is to provide a more understandable description of the mikroBasic for 8051 language to the user.

MIKROBASIC LANGUAGE REFERENCE

Lexical Elements

- Whitespace
- Comments
- Tokens

- Literals
- Keywords
- Identifiers
- Punctuators

Program Organization

- Program Organization
- Scope and Visibility
- Modules

Variables

Constants

Labels

Symbols

Functions and Procedures

- Functions
- Procedures

Types

- Simple Types
- Arrays
- Strings
- Pointers
- Structures
- Types Conversions
 - Implicit Conversion
 - Explicit Conversion

Operators

- Introduction to Operators
- Operators Precedence and Associativity
- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Boolean Operators

Expressions

- Expressions

Statements

- Introduction to Statements
- Assignment Statements
- Conditional Statements

 - If Statement
 - Select Case Statement

- Iteration Statements (Loops)

 - For Statement
 - While Statement
 - Do Statement

- Jump Statements

 - Break and Continue Statements
 - Exit Statement
 - Goto Statement
 - Gosub Statement

- asm Statement

Directives

- Compiler Directives
- Linker Directives

LEXICAL ELEMENTS OVERVIEW

These topics provide a formal definition of the mikroBasic for 8051 lexical elements. They describe different categories of word-like units (tokens) recognized by the language.

In tokenizing phase of compilation, the source code file is parsed (that is, broken down) into tokens and whitespace. The tokens in mikroBasic are derived from a series of operations performed on your programs by the compiler.

A mikroBasic program starts as a sequence of ASCII characters representing the source code, created by keystrokes using a suitable text editor (such as the mikroBasic Code Editor). The basic program unit in mikroBasic is a file. This usually corresponds to a named file located in RAM or on disk, having the extension `.mbas.`

WHITESPACE

Whitespace is a collective name given to spaces (blanks), horizontal and vertical tabs, and comments. Whitespace serves to indicate where tokens start and end, but beyond this function, any surplus whitespace is discarded.

For example, the two sequences

```
dim tmp as byte
dim j as word
```

and

```
dim tmp as byte
dim j as word
```

are lexically equivalent and parse identically.

Newline Character

Newline character (CR/LF) is not a whitespace in BASIC, and serves as a statement terminator/separator. In mikroBasic for 8051, however, you may use newline to break long statements into several lines. Parser will first try to get the longest possible expression (across lines if necessary), and then check for statement terminators.

Whitespace in Strings

The ASCII characters representing whitespace can occur within string literals, where they are protected from the normal parsing process (they remain as a part of the string). For example, statement

```
some_string = "mikro foo"
```

parses to four tokens, including a single string literal token:

```
some_string  
=  
"mikro foo"  
newline character
```

COMMENTS

Comments are pieces of text used to annotate a program, and are technically another form of whitespace. Comments are for the programmer's use only; they are stripped from the source text before parsing.

Use the apostrophe to create a comment:

```
' Any text between an apostrophe and the end of the  
' line constitutes a comment. May span one line only.
```

There are no multi-line comments in mikroBasic for 8051

TOKENS

Token is the smallest element of a mikroBasic for 8051 program, meaningful to the compiler. The parser separates tokens from the input stream by creating the longest token possible using the input characters in a left-to-right scan.

mikroBasic for 8051 recognizes the following kinds of tokens:

- keywords
- identifiers
- constants
- operators
- punctuators (also known as separators)

Token Extraction Example

Here is an example of token extraction. See the following code sequence:

```
end_flag = 0
```

The compiler would parse it into four tokens:

```
end_flag  ' variable identifier
=         ' assignment operator
0         ' literal
newline  ' statement terminator
```

Note that `end_flag` would be parsed as a single identifier, rather than the keyword `end` followed by the identifier `_flag`.

LITERALS

Literals are tokens representing fixed numeric or character values.

The data type of a constant is deduced by the compiler using such clues as numeric value and format used in the source code.

Integer Literals

Integral values can be represented in decimal, hexadecimal or binary notation.

In decimal notation, numerals are represented as a sequence of digits (without commas, spaces or dots), with optional prefix `+` or `-` operator to indicate the sign. Values default to positive (`6258` is equivalent to `+6258`).

The dollar-sign prefix (`$`) or the prefix `0x` indicates a hexadecimal numeral (for example, `$8F` or `0x8F`).

The percent-sign prefix (`%`) indicates a binary numeral (for example, `%0101`).

Here are some examples:

```
11      ' decimal literal
$11     ' hex literal, equals decimal 17
0x11    ' hex literal, equals decimal 17
%11     ' binary literal, equals decimal 3
```

The allowed range of values is imposed by the largest data type in mikroBasic for 8051 – `longword`. The compiler will report an error if the literal exceeds `4294967295` (`$FFFFFFFF`).

Floating Point Literals

A floating-point value consists of:

- Decimal integer
- Decimal point
- Decimal fraction
- e or E and a signed integer exponent (optional)

You can omit either decimal integer or decimal fraction (but not both).

Negative floating constants are taken as positive constants with the unary operator minus (`-`) prefixed.

mikroBasic limits floating-point constants to the range of $\pm 1.17549435082 * 10^{-38} .. \pm 6.80564774407 * 10^{38}$.

Here are some examples:

```
0.          ' = 0.0
-1.23       ' = -1.23
23.45e6     ' = 23.45 * 10^6
2e-5        ' = 2.0 * 10^-5
3E+10       ' = 3.0 * 10^10
.09E34      ' = 0.09 * 10^34
```

Character Literals

Character literal is one character from the extended ASCII character set, enclosed with quotes (for example, "A"). Character literal can be assigned to variables of byte and char type (variable of byte will be assigned the ASCII value of the character). Also, you can assign character literal to a string variable.

String Literals

String literal is a sequence of characters from the extended ASCII character set, enclosed with quotes. Whitespace is preserved in string literals, i.e. parser does not "go into" strings but treats them as single tokens.

Length of string literal is a number of characters it consists of. String is stored internally as the given sequence of characters plus a final `null` character. This `null` character is introduced to terminate the string, it does not count against the string's total length.

String literal with nothing in between the quotes (null string) is stored as a single `null` character.

You can assign string literal to a string variable or to an array of char.

Here are several string literals:

```
"Hello world!"           ' message, 12 chars long
"Temperature is stable"  ' message, 21 chars long
"  "                    ' two spaces, 2 chars long
"C"                      ' letter, 1 char long
""                        ' null string, 0 chars long
```

The quote itself cannot be a part of the string literal, i.e. there is no escape sequence. You could use the built-in function `Chr` to print a quote: `Chr(34)`. Also, see String Splicing.

KEYWORDS

Keywords are special-purpose words which cannot be used as normal identifier names.

Beside standard BASIC keywords, all relevant SFR are defined as global variables and represent reserved words that cannot be redefined (for example: `P0`, `TMR1`, `T1CON`, etc). Probe Code Assistant for specific letters (Ctrl+Space in Editor) or refer to Predefined Globals and Constants.

Here is the alphabetical listing of keywords in mikroBasic for 8051:

- abs
- and
- appactivate
- array
- as
- asc
- asm
- atn
- attribute
- base
- bdata
- beep
- begin
- bit
- boolean
- byte
- call
- case
- cbool
- cbyte
- ccur
- cdate
- cdate
- cdbl
- char
- chdir
- chdrive
- chr
- cint
- circle
- class
- clear
- clng
- close
- code
- command

- compact
- compare
- const
- cos
- createobject
- csng
- cstr
- curdir
- currency
- cvar
- cverr
- date
- dateadd
- datediff
- datepart
- dateserial
- datevalue
- ddb
- deftype
- dim
- dir
- div
- do
- doevents
- double
- each
- empty
- end
- end
- environ
- eof
- eqv
- erase
- err
- error
- exit
- exp
- explicit
- explicit
- fileattr
- fileattr
- filecopy
- filedatetime
- filelen
- fix
- float
- for
- form
- format

- forward
- freefile
- fv
- get
- getattr
- getobject
- gosub
- goto
- hex
- hour
- idata
- if
- iif
- ilevel
- imp
- include
- input
- instr
- int
- integer
- ipmt
- irr
- is
- isarray
- isdate
- isempty
- iserror
- ismissing
- isnull
- isnumeric
- isobject
- kill
- large
- lbound
- lcase
- left
- len
- let
- line
- loc
- lock
- lof
- log
- long
- longint
- loop
- lset
- ltrim
- me
- mid

- minute
- mirr
- mkdir
- mod
- module
- month
- msgbox
- name
- new
- next
- not
- not
- nothing
- now
- nper
- npv
- object
- oct
- on
- open
- option
- option
- option
- or
- org
- pdata
- pmt
- ppmt
- print
- private
- property
- pset
- public
- put
- pv
- qbcolor
- raise
- randomize
- rate
- real
- redim
- rem
- reset
- resume
- return
- rgb
- right
- rmdir
- rnd
- rset

- rtrim
- sbit
- second
- seek
- select
- sendkeys
- set
- setattr
- sfr
- sgn
- shell
- short
- sin
- single
- sln
- small
- space
- spc
- sqr
- static
- step
- stop
- str
- strcmp
- strconv
- string
- sub
- switch
- syd
- system
- tab
- tan
- time
- timer
- timeserial
- timevalue
- to
- trim
- typedef
- typename
- ubound
- ucase
- unlock
- until
- val
- variant
- vartype
- version
- volatile

- weekday
- wend
- while
- width
- with
- word
- write
- xdata
- xor

Also, mikroBasic includes a number of predefined identifiers used in libraries. You could replace them by your own definitions, if you plan to develop your own libraries. For more information, see mikroBasic Libraries.

IDENTIFIERS

Identifiers are arbitrary names of any length given to functions, variables, symbolic constants, user-defined data types and labels. All these program elements will be referred to as objects throughout the help (don't be confused with the meaning of object in object-oriented programming).

Identifiers can contain letters from a to z and A to Z, the underscore character “_” and digits from 0 to 9. First character must be a letter or an underscore, i.e. identifier cannot begin with a numeral.

Case Sensitivity

mikroBasic for 8051 is not case sensitive, so `Sum`, `sum`, and `suM` are equivalent identifiers.

Uniqueness and Scope

Although identifier names are arbitrary (within the rules stated), errors result if the same name is used for more than one identifier within the same scope. Simply, duplicate names are illegal within the same scope. For more information, refer to Scope and Visibility.

Identifier Examples

Here are some valid identifiers:

```
temperature_V1
Pressure
no_hit
dat2string
SUM3
_vtext
```

... and here are some invalid identifiers:

```
7temp          ' NO -- cannot begin with a numeral
%higher        ' NO -- cannot contain special characters
xor            ' NO -- cannot match reserved word
j23.07.04      ' NO -- cannot contain special characters (dot)
```

PUNCTUATORS

The mikroBasic punctuators (also known as separators) are:

- [] – Brackets
- () – Parentheses
- , – Comma
- : – Colon
- . – Dot

Brackets

Brackets [] indicate single and multidimensional array subscripts:

```
dim alphabet as byte[ 30]
' ...
alphabet[ 2] = "c"
```

For more information, refer to Arrays.

Parentheses

Parentheses () are used to group expressions, isolate conditional expressions and indicate function calls and function declarations:

```
d = c * (a + b)           ' Override normal precedence
if (d = z) then ...      ' Useful with conditional statements
func()                   ' Function call, no arguments
sub function func2(dim n as word) ' Function declaration w/ parameters
```

For more information, refer to Operators Precedence and Associativity, Expressions, or Functions and Procedures.

Comma

Comma (,) separates the arguments in function calls:

```
Lcd_Out(1, 1, txt)
```

Furthermore, the comma separates identifiers in declarations:

```
dim i, j, k as word
```

The comma also separates elements in initialization lists of constant arrays:

```
const MONTHS as byte[ 12] = (31,28,31,30,31,30,31,31,30,31,30,31)
```

Colon

Colon (:) is used to indicate a labeled statement:

```
start:  nop
        '...
goto start
```

For more information, refer to Labels.

Dot

Dot (.) indicates access to a structure member. For example:

```
person.surname = "Smith"
```

For more information, refer to Structures.

Dot is a necessary part of floating point literals. Also, dot can be used for accessing individual bits of registers in mikroBasic.

PROGRAM ORGANIZATION

mikroBasic for 8051 imposes strict program organization. Below you can find models for writing legible and organized source files. For more information on file inclusion and scope, refer to Modules and to Scope and Visibility.

Organization of Main Module

Basically, a main source file has two sections: declaration and program body. Declarations should be in their proper place in the code, organized in an orderly manner. Otherwise, the compiler may not be able to comprehend the program correctly.

When writing code, follow the model presented below. The main module should look like this:

```

program <program name>
include <include other modules>

'*****
'* Declarations (globals):
'*****

' symbols declarations
symbol ...

' constants declarations
const ...

' structures declarations
structure ...

' variables declarations
dim Name[, Name2...] as [ ^ ] type [ absolute 0x123] [ external]
[ volatile] [ register] [ sfr]

' procedures declarations
sub procedure procedure_name(...)
    <local declarations>
    ...
end sub

' functions declarations
sub function function_name(...) as return_type
    <local declarations>
    ...
end sub

'*****
'* Program body:
'*****

main:
    ' write your code here
end.

```

Organization of Other Modules

Modules other than main start with the keyword `module`. Implementation section starts with the keyword `implements`. Follow the model presented below:

```
module <module name>
include <include other modules>

*****
'* Interface (globals):
*****

' symbols declarations
symbol ...

' constants declarations
const ...

' structures declarations
structure ...

' variables declarations
dim Name[, Name2...] as [^]type [absolute 0x123] [external]
[volatile] [register] [sfr]

' procedures prototypes
sub procedure sub_procedure_name([dim byref] [const] ParamName as
[^]type, [dim byref] [const] ParamName2, ParamName3 as [^]type)

' functions prototypes
sub function sub_function_name([dim byref] [const] ParamName as
[^]type, [dim byref] [const] ParamName2, ParamName3 as [^]type) as
[^]type

*****
'* Implementation:
*****

implements

' constants declarations
const ...

' variables declarations
dim ...

' procedures declarations
sub procedure sub_procedure_name([dim byref] [const] ParamName as
[^]type, [dim byref] [const] ParamName2, ParamName3 as [^]type);
[ilevel 0x123] [overload] [forward]
    <local declarations>
    ...
end sub
```

```
' functions declarations
sub function sub_function_name([ dim byref] [ const] ParamName as
[ ^]type, [ dim byref] [ const] ParamName2, ParamName3 as [ ^]type) as
[ ^]type [ ilevel 0x123] [ overload] [ forward]
  <local declarations>
  ...
end sub

end.
```

Note: Sub functions and sub procedures must have the same declarations in the interface and implementation section. Otherwise, compiler will report an error.

SCOPE AND VISIBILITY

Scope

The scope of identifier is a part of the program in which the identifier can be used to access its object. There are different categories of scope, depending on how and where identifiers are declared:

Place of declaration	Scope
Identifier is declared in the declaration section of the main module, out of any function or procedure	Scope extends from the point where it is declared to the end of the current file, including all routines enclosed within that scope. These identifiers have a file scope and are referred to as globals.
Identifier is declared in the function or procedure	Scope extends from the point where it is declared to the end of the current routine. These identifiers are referred to as locals.
Identifier is declared in the interface section of the module	Scope extends the interface section of a module from the point where it is declared to the end of the module, and to any other module or program that uses that module. The only exception are symbols which have a scope limited to the file in which they are declared.
Identifier is declared in the implementation section of the module, but not within any function or procedure	Scope extends from the point where it is declared to the end of the module. The identifier is available to any function or procedure in the module.

Visibility

The visibility of an identifier is a region of the program source code from where a legal access to the identifier's associated object can be made .

Scope and visibility usually coincide, though there are circumstances under which an object becomes temporarily hidden by the appearance of a duplicate identifier: the object still exists but the original identifier cannot be used to access it until the scope of the duplicate identifier is ended.

Technically, visibility cannot exceed scope, but scope can exceed visibility.

MODULES

In mikroBasic for 8051, each project consists of a single project file and one or more module files. The project file, with extension `.mbproj` contains information on the project, while modules, with extension `.mbas`, contain the actual source code. See Program Organization for a detailed look at module arrangement.

Modules allow you to:

- break large programs into encapsulated modules that can be edited separately,
- create libraries that can be used in different projects,
- distribute libraries to other developers without disclosing the source code.

Each module is stored in its own file and compiled separately; compiled modules are linked to create an application. To build a project, the compiler needs either a source file or a compiled module file for each module.

Include Clause

mikroBasic includes modules by means of the `include` clause. It consists of the reserved word `include`, followed by a quoted module name. Extension of the file should not be included.

You can include one file per `include` clause. There can be any number of the `include` clauses in each source file, but they all must be stated immediately after the program (or module) name.

Here's an example:

```
program MyProgram

include "utils"
include "strings"
include "MyUnit"
```

...

For the given module name, the compiler will check for the presence of `.mcl` and `.mbasj` files, in order specified by search paths.

- If both `.mbasj` and `.mcl` files are found, the compiler will check their dates and include the newer one in the project. If the `.mbasj` file is newer than the `.mcl`, then `.mbasj` file will be recompiled and new `.mcl` will be created, overwriting the old `.mcl`.
- If only the `.mbasj` file is found, the compiler will create the `.mcl` file and include it in the project;
- If only the `.mcl` file is present, i.e. no source code is available, the compiler will include it as found;
- If none of the files found, the compiler will issue a "File not found" warning.

Main Module

Every project in mikroBasic requires a single main module file. The main module is identified by the keyword `program` at the beginning. It instructs the compiler where to "start".

After you have successfully created an empty project with Project Wizard, Code Editor will display a new main module. It contains the bare-bones of the program:

```
program MyProject

' main procedure
main:
' Place program code here
end.
```

Other than comments, nothing should precede the keyword `program`. After the program name, you can optionally place the `include` clauses.

Place all global declarations (constants, variables, labels, routines, structures) before the label `main`.

Other Modules

Modules other than main start with the keyword `module`. Newly created blank module contains the bare-bones:

```
module MyModule

implements

end.
```

Other than comments, nothing should precede the keyword `module`. After the module name, you can optionally place the `include` clauses.

Interface Section

Part of the module above the keyword `implements` is referred to as interface section. Here, you can place global declarations (constants, variables, labels, routines, structures) for the project.

Do not define routines in the interface section. Instead, state the prototypes of routines (from implementation section) that you want to be visible outside the module. Prototypes must exactly match the declarations.

Implementation Section

Implementation section hides all the irrelevant innards from other modules, allowing encapsulation of code.

Everything declared below the keyword `implements` is private, i.e. has its scope limited to the file. When you declare an identifier in the implementation section of a module, you cannot use it outside the module, but you can use it in any block or routine defined within the module.

By placing the prototype in the interface section of the module (above the `implements`) you can make the routine public, i.e. visible outside of module. Prototypes must exactly match the declarations.

VARIABLES

Variable is an object whose value can be changed during the runtime. Every variable is declared under unique name which must be a valid identifier. This name is used for accessing the memory location occupied by the variable.

Variables are declared in the declaration part of the file or routine — each variable needs to be declared before it is used. Global variables (those that do not belong to any enclosing block) are declared below the `include` statements, above the label `main`.

Specifying a data type for each variable is mandatory. mikroBasic syntax for variable declaration is:

```
dim identifier_list as type
```

Here, `identifier_list` is a comma-delimited list of valid identifiers, and `type` can be any data type.

For more details refer to Types and Types Conversions. For more information on variables' scope refer to the chapter Scope and Visibility.

Here are a few examples:

```
dim i, j, k as byte
dim counter, temp as word
dim samples as longint[100]
```

Variables and 8051

Every declared variable consumes part of RAM memory. Data type of variable determines not only the allowed range of values, but also the space a variable occupies in RAM memory. Bear in mind that operations using different types of variables take different time to be completed. mikroBasic for 8051 recycles local variable memory space – local variables declared in different functions and procedures share the same memory space, if possible.

There is no need to declare SFR explicitly, as mikroBasic for 8051 automatically declares relevant registers as global variables of word. For example: `W0`, `TMR1`, etc.

CONSTANTS

Constant is a data whose value cannot be changed during the runtime. Using a constant in a program consumes no RAM memory. Constants can be used in any expression, but cannot be assigned a new value.

Constants are declared in the declaration part of the program or routine, with the following syntax:

```
const constant_name [ as type] = value
```

Every constant is declared under unique `constant_name` which must be a valid identifier. It is a tradition to write constant names in uppercase. Constant requires you to specify `value`, which is a literal appropriate for the given type. `type` is optional and in the absence of it, the compiler assumes the “smallest” type that can accommodate `value`.

Note: You cannot omit type if declaring a constant array.

Here are a few examples:

```
const MAX as longint = 10000
const MIN = 1000      ' compiler will assume word type
const SWITCH = "n"    ' compiler will assume char type
const MSG = "Hello"   ' compiler will assume string type
const MONTHS as byte[ 12] = (31,28,31,30,31,30,31,31,30,31,30,31)
```

LABELS

Labels serve as targets for the `goto` and `gosub` statements. Mark the desired statement with label and colon like this:

```
label_identifier : statement
```

No special declaration of label is necessary in mikroBasic for 8051.

Name of the label needs to be a valid identifier. The labeled statement and `goto/gosub` statement must belong to the same block. Hence it is not possible to jump into or out of routine. Do not mark more than one statement in a block with the same label.

Note: The label `main` marks the entry point of a program and must be present in the main module of every project. See Program Organization for more information.

Here is an example of an infinite loop that calls the procedure `Beep` repeatedly:

```
loop:  
    Beep  
goto loop
```

SYMBOLS

mikroBasic symbols allow you to create simple macros without parameters. You can replace any line of code with a single identifier alias. Symbols, when properly used, can increase code legibility and reusability.

Symbols need to be declared at the very beginning of the module, right after the module name and (optional) `include` clauses. Check Program Organization for more details. Scope of a symbol is always limited to the file in which it has been declared.

Symbol is declared as:

```
symbol alias = code
```

Here, `alias` must be a valid identifier which you will use throughout the code. This identifier has a file scope. The `code` can be any line of code (literals, assignments, function calls, etc).

Using a symbol in the program consumes no RAM – the compiler will simply replace each instance of a symbol with the appropriate line of code from the declaration.

Here is an example:

```
symbol MAXALLOWED = 216           ' Symbol as alias for numeric value
symbol PORT = P0                   ' Symbol as alias for SFR
symbol MYDELAY = Delay_ms(1000)   ' Symbol as alias for procedure call

dim cnt as byte ' Some variable

' ...
main:

if cnt > MAXALLOWED then
    cnt = 0
    PORT.1 = 0
    MYDELAY
end if
```

Note: Symbols do not support macro expansion in a way the C preprocessor does.

FUNCTIONS AND PROCEDURES

Functions and procedures, collectively referred to as routines, are subprograms (self-contained statement blocks) which perform a certain task based on a number of input parameters. When executed, a function returns value while procedure does not.

FUNCTIONS

Function is declared like this:

```
sub function function_name(parameter_list) as return_type
  [ local declarations ]
  function body
end sub
```

`function_name` represents a function's name and can be any valid identifier. `return_type` is a type of return value and can be any simple type. Within parentheses, `parameter_list` is a formal parameter list similar to variable declaration. In mikroBasic for 8051, parameters are always passed to a function by value. To pass an argument by address, add the keyword `byref` ahead of identifier.

`Local declarations` are optional declarations of variables and/or constants, local for the given function. `Function body` is a sequence of statements to be executed upon calling the function.

Calling a function

A function is called by its name, with actual arguments placed in the same sequence as their matching formal parameters. The compiler is able to coerce mismatching arguments to the proper type according to implicit conversion rules. Upon a function call, all formal parameters are created as local objects initialized by values of actual arguments. Upon return from a function, a temporary object is created in the place of the call and it is initialized by the value of the function result. This means that function call as an operand in complex expression is treated as the function result.

In standard Basic, a `function_name` is automatically created local variable that can be used for returning a value of a function. mikroBasic for 8051 also allows you to use the automatically created local variable `result` to assign the return value of a function if you find function name to be too ponderous. If the return value of a function is not defined the compiler will report an error.

Function calls are considered to be primary expressions and can be used in situations where expression is expected. A function call can also be a self-contained statement and in that case the return value is discarded.

Example

Here's a simple function which calculates x^n based on input parameters `x` and `n` (`n > 0`):

```
sub function power(dim x, n as byte) as longint
dim i as byte
  result = 1
  if n > 0 then
    for i = 1 to n
      result = result*x
    next i
  end if
end sub
```

Now we could call it to calculate, say, 312:

```
tmp = power(3, 12)
```

PROCEDURES

Procedure is declared like this:

```
sub procedure procedure_name(parameter_list)
  [ local declarations ]
  procedure body
end sub
```

`procedure_name` represents a procedure's name and can be any valid identifier. Within parentheses, `parameter_list` is a formal parameter list similar to variable declaration. In mikroBasic for 8051, parameters are always passed to procedure by value; to pass argument by address, add the keyword `byref` ahead of identifier.

`Local declarations` are optional declaration of variables and/or constants, local for the given procedure. `Procedure body` is a sequence of statements to be executed upon calling the procedure.

Calling a procedure

A procedure is called by its name, with actual arguments placed in the same sequence as their matching formal parameters. The compiler is able to coerce mismatching arguments to the proper type according to implicit conversion rules. Upon procedure call, all formal parameters are created as local objects initialized by values of actual arguments.

Procedure call is a self-contained statement.

Example

Here's an example procedure which transforms its input time parameters, preparing them for output on LCD:

```
sub procedure time_prep(dim byref sec, min, hr as byte)
    sec = ((sec and $F0) >> 4)*10 + (sec and $0F)
    min = ((min and $F0) >> 4)*10 + (min and $0F)
    hr = ((hr and $F0) >> 4)*10 + (hr and $0F)
end sub
```

Function Pointers

Function pointers are allowed in mikroBasic for 8051. The example shows how to define and use a function pointer:

Example:

Example demonstrates the usage of function pointers. It is shown how to declare a procedural type, a pointer to function and finally how to call a function via pointer.

```
program Example;
```

```
typedef TMyFunctionType = function (dim param1, param2 as byte, dim
param3 as word) as word ' First, define the procedural type
```

```
dim MyPtr as ^TMyFunctionType ' This is a pointer to previously
defined type
dim sample as word
```

```
sub function Func1(dim p1, p2 as byte, dim p3 as word) as word ' Now,
define few functions which will be pointed to. Make sure that param-
eters match the type definition
    result = p1 and p2 or p3
end sub
```

```
sub function Func2(dim abc, def as byte, dim ghi as word) as word '
Another function of the same kind. Make sure that parameters match
the type definition
    result = abc * def + ghi
end sub
```

```
sub function Func3(dim first, yellow as byte, dim monday as word) as
word ' Yet another function. Make sure that parameters match the
type definition
    result = monday - yellow - first
end sub
```

```
' main program:
```


Forward declaration

A function can be declared without having it followed by its implementation, by having it followed by the forward procedure. The effective implementation of that function must follow later in the module. The function can be used after a forward declaration as if it had been implemented already. The following is an example of a forward declaration:

```
program Volume

dim Volume as word

sub function First(a as word, b as word) as word forward

sub function Second(c as word) as word
dim tmp as word
  tmp = First(2, 3)
  result = tmp * c
end sub

sub function First(a, b as word) as word
  result = a * b
end sub

main:
  Volume = Second(4)
end.
```

TYPES

Basic is strictly typed language, which means that every variable and constant need to have a strictly defined type, known at the time of compilation.

The type serves:

- to determine correct memory allocation required,
- to interpret the bit patterns found in the object during subsequent accesses,
- in many type-checking situations, to ensure that illegal assignments are trapped.

mikroBasic supports many standard (predefined) and user-defined data types, including signed and unsigned integers of various sizes, arrays, strings, pointers and structures.

Type Categories

Types can be divided into:

- simple types
- arrays
- strings
- pointers
- structures

SIMPLE TYPES

Simple types represent types that cannot be divided into more basic elements and are the model for representing elementary data on machine level. Basic memory unit in mikroBasic for 8051 has 16 bits.

Here is an overview of simple types in mikroBasic for 8051:

Type	Size	Range
<code>byte, char</code>	8-bit	0 .. 255
<code>short</code>	8-bit	-127 .. 128
<code>word</code>	16-bit	0 .. 65535
<code>integer</code>	16-bit	-32768 .. 32767
<code>longword</code>	32-bit	0 .. 4294967295
<code>longint</code>	32-bit	-2147483648 .. 2147483647
<code>float</code>	32-bit	$\pm 1.17549435082 * 10^{-38}$.. $\pm 6.80564774407 * 10^38$
<code>bit</code>	1-bit	0 or 1
<code>sbit</code>	1-bit	0 or 1

You can assign signed to unsigned or vice versa only using the explicit conversion. Refer to Types Conversions for more information.

ARRAYS

An array represents an indexed collection of elements of the same type (called the base type). Since each element has a unique index, arrays, unlike sets, can meaningfully contain the same value more than once.

Array Declaration

Array types are denoted by constructions in the following form:

```
type[ array_length]
```

Each of elements of an array is numbered from 0 through `array_length - 1`. Every element of an array is of `type` and can be accessed by specifying array name followed by element's index within brackets.

Here are a few examples of array declaration:

```
dim weekdays as byte[ 7]
dim samples  as word[ 50]

main:
  ' Now we can access elements of array variables, for example:
  samples[ 0] = 1
  if samples[ 37] = 0 then
    ' ...
```

Constant Arrays

Constant array is initialized by assigning it a comma-delimited sequence of values within parentheses. For example:

```
' Declare a constant array which holds number of days in each month:
const MONTHS as byte[ 12] = (31,28,31,30,31,30,31,31,30,31,30,31)
```

Note that indexing is zero based; in the previous example, number of days in January is `MONTHS[0]` and number of days in December is `MONTHS[11]`.

The number of assigned values must not exceed the specified length. Vice versa is possible, when the trailing “excess” elements will be assigned zeroes.

For more information on arrays of `char`, refer to Strings.

STRINGS

A string represents a sequence of characters equivalent to an array of `char`. It is declared like this:

```
string[ string_length]
```

The specifier `string_length` is a number of characters a string consists of. The string is stored internally as the given sequence of characters plus a final `null` character (zero). This appended “stamp” does not count against string’s total length.

A null string (“”) is stored as a single `null` character.

You can assign string literals or other strings to string variables. The string on the right side of an assignment operator has to be shorter than another one, or of equal length. For example:

```
dim msg1 as string[ 20]
dim msg2 as string[ 19]

main:
    msg1 = "This is some message"
    msg2 = "Yet another message"

    msg1 = msg2 ' this is ok, but vice versa would be illegal
```

Alternately, you can handle strings element–by–element. For example:

```
dim s as string[ 5]
' ...
s = "mik"
' s[ 0] is char literal "m"
' s[ 1] is char literal "i"
' s[ 2] is char literal "k"
' s[ 3] is zero
' s[ 4] is undefined
' s[ 5] is undefined
```

Be careful when handling strings in this way, since overwriting the end of a string will cause an unpredictable behavior.

Note

mikroBasic for 8051 includes String Library which automatizes string related tasks.

POINTERS

A pointer is a data type which holds a memory address. While a variable accesses that memory address directly, a pointer can be thought of as a reference to that memory address.

To declare a pointer data type, add a caret prefix (^) before type. For example, if you are creating a pointer to an `integer`, you would write:

```
^integer
```

To access the data at the pointer's memory location, you add a caret after the variable name. For example, let's declare variable `p` which points to `word`, and then assign the pointed memory location value 5:

```
dim p as ^word
'...
p^ = 5
```

A pointer can be assigned to another pointer. However, note that only address, not value, is copied. Once you modify the data located at one pointer, the other pointer, when dereferenced, also yields modified data.

@ Operator

The @ operator returns the address of a variable or routine, i.e. @ constructs a pointer to its operand. The following rules are applied to @:

- If `X` is a variable, `@X` returns the address of `X`.
- If `F` is a routine (a function or procedure), `@F` returns `F`'s entry point (the result is of `longint`).

STRUCTURES

A structure represents a heterogeneous set of elements. Each element is called a member; the declaration of a structure type specifies a name and type for each member. The syntax of a structure type declaration is

```
structure structname
  dim member1 as type1
  '...
  dim membern as typen
end structure
```

where structname is a valid identifier, each type denotes a type, and each member is a valid identifier. The scope of a member identifier is limited to the structure in which it occurs, so you don't have to worry about naming conflicts between member identifiers and other variables.

For example, the following declaration creates a structure type called Dot:

```
structure Dot
  dim x as float
  dim y as float
end structure
```

Each Dot contains two members: x and y coordinates; memory is allocated when you instantiate the structure, like this:

```
dim m, n as Dot
```

This variable declaration creates two instances of Dot, called m and n.

A member can be of the previously defined structure type. For example:

```
' Structure defining a circle:
structure Circle
  dim radius as float
  dim center as Dot
end structure
```


Structure Member Access

You can access the members of a structure by means of dot (.) as a direct member selector. If we had declared the variables `circle1` and `circle2` of the previously defined type `Circle`:

```
dim circle1, circle2 as Circle
```

we could access their individual members like this:

```
circle1.radius = 3.7  
circle1.center.x = 0  
circle1.center.y = 0
```

You can also commit assignments between complex variables, if they are of the same type:

```
circle2 = circle1 ' This will copy values of all members
```

TYPES CONVERSIONS

Conversion of variable of one type to variable of another type is typecasting. mikroBasic for 8051 supports both implicit and explicit conversions for built-in types.

Implicit Conversion

Compiler will provide an automatic implicit conversion in the following situations:

- statement requires an expression of particular type (according to language definition) and we use an expression of different type,
- operator requires an operand of particular type and we use an operand of different type,
- function requires a formal parameter of particular type and we pass it an object of different type,
- `result` does not match the declared function return type.

Promotion

When operands are of different types, implicit conversion promotes the less complex to the more complex type taking the following steps:

```
byte/char ? word
short     ? integer
short     ? longint
integer   ? longint
integral  ? float
```

Higher bytes of extended unsigned operand are filled with zeroes. Higher bytes of extended signed operand are filled with bit sign (if number is negative, fill higher bytes with one, otherwise with zeroes). For example:

```
dim a as byte
dim b as word
'...
a = $FF
b = a ' a is promoted to word, b becomes $00FF
```

Clipping

In assignments and statements that require an expression of particular type, destination will store the correct value only if it can properly represent the result of expression, i.e. if the result fits in destination range.

If expression evaluates to more complex type than expected excess data will be simply clipped (the higher bytes are lost).

```

dim i as byte
dim j as word
'...
j = $FF0F
i = j ' i becomes $0F, higher byte $FF is lost

```

Explicit Conversion

Explicit conversion can be executed at any point by inserting type keyword (*byte*, *word*, *short*, *integer*, *longint*, or *float*) ahead of the expression to be converted. The expression must be enclosed in parentheses. Explicit conversion can be performed only on the operand left of the assignment operator.

Special case is the conversion between signed and unsigned types. Explicit conversion between signed and unsigned data does not change binary representation of data — it merely allows copying of source to destination.

For example:

```

dim a as byte
dim b as short
'...
b = -1
a = byte(b) ' a is 255, not 1

' This is because binary representation remains
' 11111111; it's just interpreted differently now

```

You cannot execute explicit conversion on the operand left of the assignment operator:

```

word(b) = a ' Compiler will report an error

```

OPERATORS

Operators are tokens that trigger some computation when being applied to variables and other objects in an expression.

There are four types of operators in mikroBasic for 8051:

- Arithmetic Operators
- Bitwise Operators
- Boolean Operators
- Relational Operators

OPERATORS PRECEDENCE AND ASSOCIATIVITY

There are 4 precedence categories in mikroBasic for 8051. Operators in the same category have equal precedence with each other.

Each category has an associativity rule: left-to-right (\rightarrow), or right-to-left (\leftarrow). In the absence of parentheses, these rules resolve the grouping of expressions with operators of equal precedence.

Precedence	Operands	Operators	Associativity
4	1	@ not + -	\leftarrow
3	2	* / div mod and << >>	\rightarrow
2	2	+ - or xor	\rightarrow
1	2	= <> < > <= >=	\rightarrow

ARITHMETIC OPERATORS

Arithmetic operators are used to perform mathematical computations. They have numerical operands and return numerical results. Since the `char` operators are technically `bytes`, they can be also used as unsigned operands in arithmetic operations.

All arithmetic operators associate from left to right.

Operator	Operation	Operands	Result
+	addition	byte, short, word, integer, longint, longword, float	byte, short, word, integer, longint, longword, float
-	subtraction	byte, short, word, integer, longint, longword, float	byte, short, word, integer, longint, longword, float
*	multiplication	byte, short, word, integer, longint, longword, float	word, integer, longint, longword, float
/	division, floating-point	byte, short, word, integer, longint, longword, float	float
div	division, rounds down to nearest integer	byte, short, word, integer, longint, longword	byte, short, word, integer, longint, longword
mod	modulus, returns the remainder of integer division (cannot be used with floating points)	byte, short, word, integer, longint, longword	byte, short, word, integer, longint, longword

Division by Zero

If 0 (zero) is used explicitly as the second operand (i.e. `x div 0`), the compiler will report an error and will not generate code.

But in case of implicit division by zero: `x div y`, where `y` is 0 (zero), the result will be the maximum integer (i.e. 255, if the result is `byte` type; 65536, if the result is `word` type, etc.).

Unary Arithmetic Operators

Operator `-` can be used as a prefix unary operator to change sign of a signed value. Unary prefix operator `+` can be used, but it doesn't affect data.

For example:

```
b = -a
```

Relational Operators

Use relational operators to test equality or inequality of expressions. All relational operators return `TRUE` or `FALSE`.

Operator	Operation
=	equal
<>	not equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal

All relational operators associate from left to right.

Relational Operators in Expressions

The equal sign (=) can also be an assignment operator, depending on context.

Precedence of arithmetic and relational operators was designated in such a way to allow complex expressions without parentheses to have expected meaning:

```

if aa + 5 >= bb - 1.0 / cc then      ' same as: if (aa + 5) >= (bb -
(1.0 / cc) then
    dd = My_Function()
end if

```

BITWISE OPERATORS

Use the bitwise operators to modify the individual bits of numerical operands.

Bitwise operators associate from left to right. The only exception is the bitwise complement operator `not` which associates from right to left.

Bitwise Operators Overview

Operator	Operation
<code>and</code>	bitwise AND; compares pairs of bits and generates a 1 result if both bits are 1, otherwise it returns 0
<code>or</code>	bitwise (inclusive) OR; compares pairs of bits and generates a 1 result if either or both bits are 1, otherwise it returns 0
<code>xor</code>	bitwise exclusive OR (XOR); compares pairs of bits and generates a 1 result if the bits are complementary, otherwise it returns 0
<code>not</code>	bitwise complement (unary); inverts each bit
<code><<</code>	bitwise shift left; moves the bits to the left, it discards the far left bit and assigns 0 to the right most bit.
<code>>></code>	bitwise shift right; moves the bits to the right, discards the far right bit and if unsigned assigns 0 to the left most bit, otherwise sign extends

Logical Operations on Bit Level

<code>and</code>	0	1
0	0	0
1	0	1

<code>or</code>	0	1
0	0	1
1	1	1

<code>xor</code>	0	1
0	0	1
1	1	0

<code>not</code>	0	1
	1	0

The bitwise operators `and`, `or`, and `xor` perform logical operations on the appropriate pairs of bits of their operands. The operator `not` complements each bit of its operand. For example:

```

$1234 and $5678          ' equals $1230

' because ..

' $1234 : 0001 0010 0011 0100
' $5678 : 0101 0110 0111 1000
' -----
'   and : 0001 0010 0011 0000

' .. that is, $1230

' Similarly:

$1234 or  $5678          ' equals $567C
$1234 xor $5678          ' equals $444C
not $1234                 ' equals $EDCB

```

Unsigned and Conversions

If number is converted from less complex to more complex data type, the upper bytes are filled with zeroes. If number is converted from more complex to less complex data type, the data is simply truncated (upper bytes are lost).

For example:

```

dim a as byte
dim b as word
' ...
a = $AA
b = $F0F0
b = b and a
' a is extended with zeroes; b becomes $00A0

```

Signed and Conversions

If number is converted from less complex to more complex data type, the upper bytes are filled with ones if sign bit is 1 (number is negative); the upper bytes are filled with zeroes if sign bit is 0 (number is positive). If number is converted from more complex to less complex data type, the data is simply truncated (the upper bytes are lost).

For example:


```
dim a as byte
dim b as word
' ...
a = -12
b = $70FF
b = b and a

' a is sign extended, upper byte is $FF;
' b becomes $70F4
```

Bitwise Shift Operators

The binary operators `<<` and `>>` move the bits of the left operand by a number of positions specified by the right operand, to the left or right, respectively. Right operand has to be positive and less than 255.

With shift left (`<<`), left most bits are discarded, and “new” bits on the right are assigned zeroes. Thus, shifting unsigned operand to the left by n positions is equivalent to multiplying it by 2^n if all discarded bits are zero. This is also true for signed operands if all discarded bits are equal to the sign bit.

With shift right (`>>`), right most bits are discarded, and the “freed” bits on the left are assigned zeroes (in case of unsigned operand) or the value of the sign bit (in case of signed operand). Shifting operand to the right by n positions is equivalent to dividing it by 2^n .

BOOLEAN OPERATORS

Although mikroBasic for 8051 does not support `boolean` type, you have Boolean operators at your disposal for building complex conditional expressions. These operators conform to standard Boolean logic and return either `TRUE` (all ones) or `FALSE` (zero):

Operator	Operation
<code>and</code>	logical AND
<code>or</code>	logical OR
<code>xor</code>	logical exclusive OR (XOR)
<code>not</code>	logical negation

Boolean operators associate from left to right. Negation operator `not` associates from right to left.

EXPRESSIONS

An expression is a sequence of operators, operands, and punctuators that returns a value.

The primary expressions include: literals, constants, variables and function calls. From them, using operators, more complex expressions can be created. Formally, expressions are defined recursively: subexpressions can be nested up to the limits of memory.

Expressions are evaluated according to certain conversion, grouping, associativity and precedence rules that depend on the operators used, presence of parentheses, and data types of the operands. The precedence and associativity of the operators are summarized in Operator Precedence and Associativity. The way operands and subexpressions are grouped does not necessarily specify the actual order in which they are evaluated by mikroBasic for 8051.

STATEMENTS

Statements define algorithmic actions within a program. Each statement needs to be terminated with a semicolon (;). In the absence of specific jump and selection statements, statements are executed sequentially in the order of appearance in the source code.

The most simple statements are assignments, procedure calls and jump statements. These can be combined to form loops, branches and other structured statements.

Refer to:

- Assignment Statements
- Conditional Statements
- Iteration Statements (Loops)
- Jump Statements

- asm Statement

ASSIGNMENT STATEMENTS

Assignment statements have the following form:

```
variable = expression
```

The statement evaluates `expression` and assigns its value to `variable`. All rules of implicit conversion are applied. `Variable` can be any declared variable or array element, and `expression` can be any expression.

Do not confuse the assignment with relational operator = which tests for equality. mikroBasic for 8051 will interpret the meaning of the character = from the context.

Conditional Statements

Conditional or selection statements select from alternative courses of action by testing certain values. There are two types of selection statements:

- if
- select case

IF STATEMENT

Use the keyword `if` to implement a conditional statement. The syntax of the `if` statement has the following form:

```
if expression then
    statements
[ else
    other statements]
end if
```

When `expression` evaluates to true, `statements` execute. If `expression` is false, `other statements` execute. The `expression` must convert to a boolean type; otherwise, the condition is ill-formed. The `else` keyword with an alternate block of statements (other statements) is optional.

Nested if statements

Nested if statements require additional attention. A general rule is that the nested conditionals are parsed starting from the innermost conditional, with each `else` bound to the nearest available `if` on its left:

```
if expression1 then
if expression2 then
statement1
else
statement2
end if
end if
```

The compiler treats the construction in this way:

```
if expression1 then
    if expression2 then
        statement1
    else
        statement2
    end if
end if
```

In order to force the compiler to interpret our example the other way around, we have to write it explicitly:

```
if expression1 then
    if expression2 then
        statement1
    end if
else
    statement2
end if
```

SELECT CASE STATEMENT

Use the `select case` statement to pass control to a specific program branch, based on a certain condition. The `select case` statement consists of selector expression (condition) and list of possible values. The syntax of the `select case` statement is:

```
select case selector
  case value_1
    statements_1
  ...
  case value_n
    statements_n
  [ case else
    default_statements]
end select
```

`selector` is an expression which should evaluate as integral value. values can be literals, constants or expressions and `statements` can be any statements. The `case else` clause is optional.

First, the `selector` expression (condition) is evaluated. The `select case` statement then compares it against all available `values`. If the match is found, the `statements` following the match evaluate, and the `select case` statement terminates. In case there are multiple matches, the first matching statement will be executed. If none of the `values` matches the `selector`, then `default_statements` in the `case else` clause (if there is one) are executed.

Here is a simple example of the `select case` statement:

```
select case operator
  case "*"
    res = n1 * n2
  case "/"
    res = n1 / n2
  case "+"
    res = n1 + n2
  case "-"
    res = n1 - n2
  case else
    res = 0
    cnt = cnt + 1
end select
```

Also, you can group values together for a match. Simply separate the items by commas:

```
select case reg
  case 0
    opmode = 0
  case 1,2,3,4
    opmode = 1
  case 5,6,7
    opmode = 2
end select
```

Nested Case Statements

Note that the `select case` statements can be nested – `values` are then assigned to the innermost enclosing `select case` statement.

ITERATION STATEMENTS

Iteration statements let you loop a set of statements. There are three forms of iteration statements in mikroBasic for 8051:

- for
- while
- repeat

You can use the statements `break` and `continue` to control the flow of a loop statement. `break` terminates the statement in which it occurs, while `continue` begins executing the next iteration of the sequence.

FOR STATEMENT

The for statement implements an iterative loop and requires you to specify the number of iterations. The syntax of the for statement is:

```
for counter = initial_value to final_value [ step step_value]
    statements
next counter
```

`counter` is a variable being increased by `step_value` with each iteration of the loop. The parameter `step_value` is an optional integral value, and defaults to 1 if omitted. Before the first iteration, `counter` is set to `initial_value` and will be incremented until it reaches (or exceeds) the `final_value`. With each iteration, `statements` will be executed.

`initial_value` and `final_value` should be expressions compatible with `counter`; `statements` can be any statements that do not change the value of `counter`.

Note that the parameter `step_value` may be negative, allowing you to create a countdown.

Here is an example of calculating scalar product of two vectors, `a` and `b`, of length `n`, using the `for` statement:

```
s = 0
for i = 0 to n-1
    s = s + a[i] * b[i]
next i
```

Endless Loop

The `for` statement results in an endless loop if `final_value` equals or exceeds the range of the `counter`'s type.

WHILE STATEMENT

Use the `while` keyword to conditionally iterate a statement. The syntax of the `while` statement is:

```
while expression
  statements
wend
```

`statements` are executed repeatedly as long as `expression` evaluates true. The test takes place before statements are executed. Thus, if `expression` evaluates false on the first pass, the loop does not execute.

Here is an example of calculating scalar product of two vectors, using the `while` statement:

```
s = 0
i = 0
while i < n
  s = s + a[ i ] * b[ i ]
  i = i + 1
wend
```

Probably the easiest way to create an endless loop is to use the statement:

```
while TRUE
  ...
wend
```

DO STATEMENT

The do statement executes until the condition becomes true. The syntax of the do statement is:

```
do
    statements
loop until expression
```

`statements` are executed repeatedly until `expression` evaluates true. `expression` is evaluated after each iteration, so the loop will execute `statements` at least once.

Here is an example of calculating scalar product of two vectors, using the `do` statement:

```
s = 0
i = 0
do
    s = s + a[ i ] * b[ i ]
    i = i + 1
loop until i = n
```

JUMP STATEMENTS

A jump statement, when executed, transfers control unconditionally. There are five such statements in mikroBasic for 8051:

- break
- continue
- exit
- goto
- gosub

ASM STATEMENT

mikroBasic for 8051 allows embedding assembly in the source code by means of the `asm` statement. Note that you cannot use numerals as absolute addresses for register variables in assembly instructions. You may use symbolic names instead (listing will display these names as well as addresses).

You can group assembly instructions with the `asm` keyword:

```
asm
    block of assembly instructions
end asm
```

mikroBasic comments are not allowed in embedded assembly code. Instead, you may use one-line assembly comments starting with semicolon.

If you plan to use a certain mikroBasic variable in embedded assembly only, be sure to at least initialize it (assign it initial value) in mikroBasic code; otherwise, the linker will issue an error. This is not applied to predefined globals such as `P0`.

For example, the following code will not be compiled because the linker won't be able to recognize the variable `myvar`:

```
program test

dim myvar as word

main:
    asm
        MOV #10, W0
        MOV W0, _myvar
    end asm
end.
```

Adding the following line (or similar) above the `asm` block would let linker know that variable is used:

```
myvar = 20
```

DIRECTIVES

Directives are words of special significance which provide additional functionality regarding compilation and output.

The following directives are at your disposal:

- Compiler directives for conditional compilation,
- Linker directives for object distribution in memory.

COMPILER DIRECTIVES

Any line in source code with leading # is taken as a compiler directive. The initial # can be preceded or followed by whitespace (excluding new lines). The compiler directives are not case sensitive.

You can use conditional compilation to select particular sections of code to compile while excluding other sections. All compiler directives must be completed in the source file in which they begun.

Directives #DEFINE and #UNDEFINE

Use directive #DEFINE to define a conditional compiler constant (“flag”). You can use any identifier for a flag, with no limitations. No conflicts with program identifiers are possible because the flags have a separate name space. Only one flag can be set per directive.

For example:

```
#DEFINE extended_format
```

Use #UNDEFINE to undefine (“clear”) previously defined flag.

Directives #IFDEF, #ELSEIF and #ELSE

Conditional compilation is carried out by the #IFDEF directive. #IFDEF tests whether a flag is currently defined or not; i.e. whether the previous #DEFINE directive has been processed for that flag and is still in force.

The directive #IFDEF is terminated by the #ENDIF directive and can have any number of the #ELSEIF clauses and an optional #ELSE clause:

```
#IFDEF flag THEN
    block of code
[ #ELSEIF flag_1 THEN
    block of code 1
...
#ELSEIF flag_n THEN
    block of code n ]
[ #ELSE
    alternate block of code ]
#ENDIF
```

First, `#IFDEF` checks if `flag` is set by means of `#DEFINE`. If so, only block of code will be compiled. Otherwise, the compiler will check flags `flag_1 .. flag_n` and execute the appropriate block of code `i`. Eventually, if none of the flags is set, alternate block of code in `#ELSE` (if any) will be compiled.

`#ENDIF` ends the conditional sequence. The result of the preceding scenario is that only one section of code (possibly empty) is passed on for further processing. The processed section can contain further conditional clauses, nested to any depth; each `#IFDEF` must be matched with a closing `#ENDIF`.

Here is an example:

```
' Uncomment the appropriate flag for your application:
'#DEFINE resolution8
'#DEFINE resolution10
'#DEFINE resolution12

#IFDEF resolution8 THEN
    ... ' code specific to 8-bit resolution
#ELSEIF resolution10 THEN
    ... ' code specific to 10-bit resolution
#ELSEIF resolution12 THEN
    ... ' code specific to 12-bit resolution
#ELSE
    ... ' default code
#ENDIF
```

Predefined Flags

The compiler sets directives upon completion of project settings, so the user doesn't need to define certain flags.

Here is an example:

```
#IFDEF AT89S8253 ' If AT89S8253 MCU is selected
#IFDEF AT89      ' AT89S8253 and AT89 flags will be automatically
defined
```

LINKER DIRECTIVES

mikroBasic for 8051 uses internal algorithm to distribute objects within memory. If you need to have a variable or routine at the specific predefined address, use the linker directives `absolute` and `org`.

Note: You must specify an even address when using the linker directives.

Directive `absolute`

The directive `absolute` specifies the starting address in RAM for a variable. If the variable spans more than 1 word (16-bit), higher words will be stored at the consecutive locations.

The `absolute` directive is appended to the declaration of a variable:

```
dim x as word absolute 0x32
' Variable x will occupy 1 word (16 bits) at address 0x32

dim y as longint absolute 0x34
' Variable y will occupy 2 words at addresses 0x34 and 0x36
```

Be careful when using `absolute` directive, as you may overlap two variables by accident. For example:

```
dim i as word absolute 0x42
' Variable i will occupy 1 word at address 0x42;

dim jj as longint absolute 0x40
' Variable will occupy 2 words at 0x40 and 0x42; thus,
' changing i changes jj at the same time and vice versa
```

Note: You must specify an even address when using the directive `absolute`.

Directive `org`

The directive `org` specifies the starting address of a routine in ROM. It is appended to the declaration of routine. For example:

```
sub procedure proc(dim par as word) org 0x200
' Procedure will start at the address 0x200;
...
end sub
```

Note: You must specify an even address when using the directive `org`.

CHAPTER



mikroBasic for 8051 Libraries

mikroBasic for 8051 provides a set of libraries which simplify the initialization and use of 8051 compliant MCUs and their modules:

Use Library manager to include mikroBasic for 8051 Libraries in you project.

Hardware 8051-specific Libraries

- CANSPI Library
- EEPROM Library
- Graphic LCD Library
- Keypad Library
- LCD Library
- Manchester Code Library
- OneWire Library
- Port Expander Library
- PS/2 Library
- RS-485 Library
- Software I2C Library
- Software SPI Library
- Software UART Library
- Sound Library
- SPI Library
- SPI Ethernet Library
- SPI Graphic LCD Library
- SPI LCD Library
- SPI LCD8 Library
- SPI T6963C Graphic LCD Library
- T6963C Graphic LCD Library
- UART Library

Miscellaneous Libraries

- Button Library
- Conversions Library
- Math Library
- String Library
- Time Library
- Trigonometry Library

See also Built-in Routines.

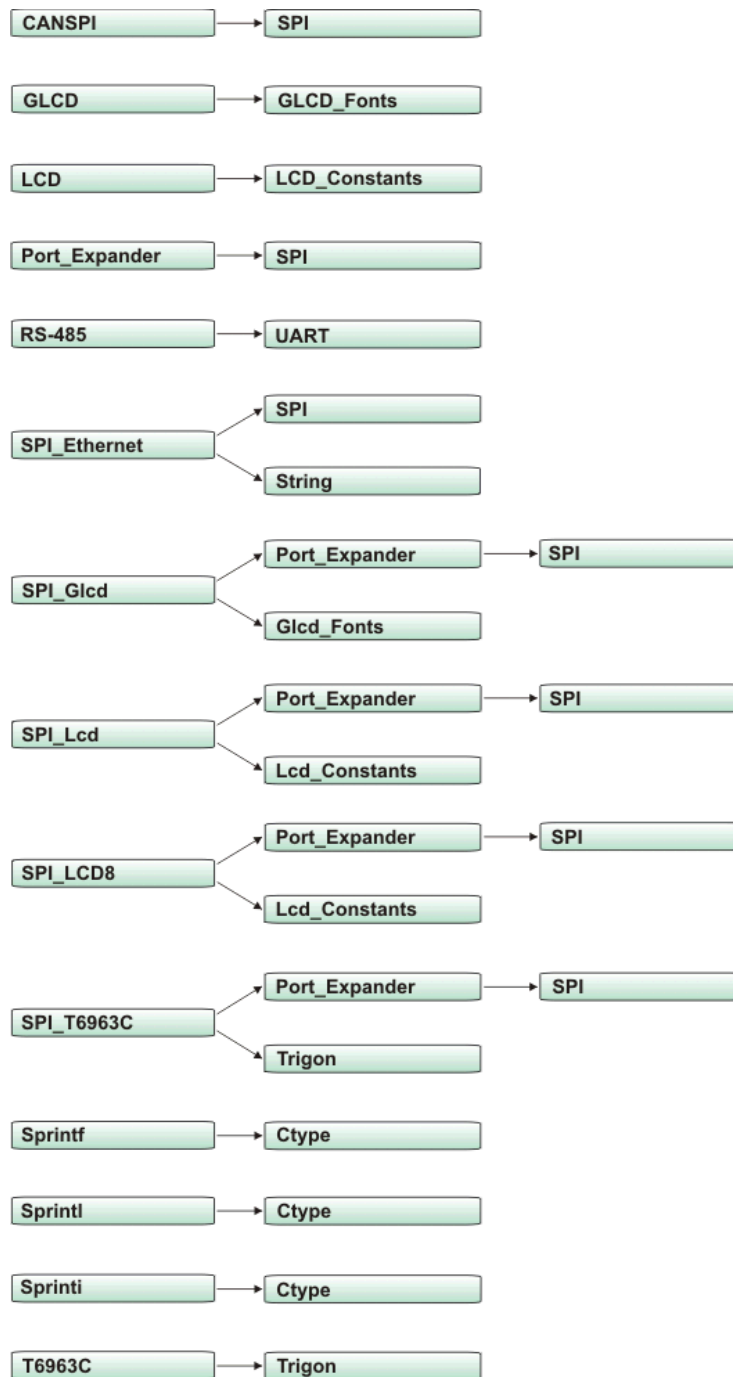
LIBRARY DEPENDENCIES

Certain libraries use (depend on) function and/or variables, constants defined in other libraries.

Image below shows clear representation about these dependencies.

For example, SPI_Glcd uses Glcd_Fonts and Port_Expander library which uses SPI library.

This means that if you check SPI_Glcd library in Library manager, all libraries on which it depends will be checked too.



Related topics: Library manager, 8051 Libraries

CANSPI LIBRARY

The SPI module is available with a number of the 8051 compliant MCUs. The mikroBasic for 8051 provides a library (driver) for working with mikroElektronika's CANSPI Add-on boards (with MCP2515 or MCP2510) via SPI interface.

The CAN is a very robust protocol that has error detection and signalization, self-checking and fault confinement. Faulty CAN data and remote frames are re-transmitted automatically, similar to the Ethernet.

Data transfer rates depend on distance. For example, 1 Mbit/s can be achieved at network lengths below 40m while 250 Kbit/s can be achieved at network lengths below 250m. The greater distance the lower maximum bitrate that can be achieved. The lowest bitrate defined by the standard is 200Kbit/s. Cables used are shielded twisted pairs.

CAN supports two message formats:

- Standard format, with 11 identifier bits and
- Extended format, with 29 identifier bits

Note:

- Consult the CAN standard about CAN bus termination resistance.
- An effective CANSPI communication speed depends on SPI and certainly is slower than "real" CAN.
- CANSPI module refers to mikroElektronika's CANSPI Add-on board connected to SPI module of MCU.

External dependencies of CANSPI Library

The following variables must be defined in all projects using CANSPI Library:	Description:	Example :
<code>dim CanSpi_CS as sbit external</code>	Chip Select line.	<code>dim CanSpi_CS as sbit at P1.B0</code>
<code>dim CanSpi_RST as sbit external</code>	Reset line.	<code>dim CanSpi_Rst as sbit at P1.B2</code>

Library Routines

- CANSPISetOperationMode
- CANSPIGetOperationMode
- CANSPIInitialize
- CANSPISetBaudRate
- CANSPISetMask
- CANSPISetFilter
- CANSPIread
- CANSPIWrite

The following routines are for an internal use by the library only:

- RegsToCANSPIID
- CANSPIIDToRegs

Be sure to check CANSPI constants necessary for using some of the sub functions.

CANSPISetOperationMode

Prototype	<code>sub procedure CANSPISetOperationMode(dim mode as byte, dim WAIT as byte)</code>
Returns	Nothing.
Description	<p>Sets the CANSPI module to requested mode.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>mode</code>: CANSPI module operation mode. Valid values: <code>CANSPI_OP_MODE</code> constants (see CANSPI constants). - <code>WAIT</code>: CANSPI mode switching verification request. If <code>WAIT = 0</code>, the call is non-blocking. The sub function does not verify if the CANSPI module is switched to requested mode or not. Caller must use <code>CANSPIGetOperationMode</code> to verify correct operation mode before performing mode specific operation. If <code>WAIT != 0</code>, the call is blocking – the sub function won't "return" until the requested mode is set.
Requires	<p>The CANSPI routines are supported only by MCUs with the SPI module.</p> <p>MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.</p>
Example	<pre>' set the CANSPI module into configuration mode (wait inside CANSPISetOperationMode until this mode is set) CANSPISetOperationMode(CANSPI_MODE_CONFIG, 0xFF)</pre>

CANSPIGetOperationMode

Prototype	<code>sub function CANSPIGetOperationMode() as byte</code>
Returns	Current operation mode.
Description	The sub function returns current operation mode of the CANSPI module. Check <code>CANSPI_OP_MODE</code> constants (see CANSPI constants) or device datasheet for operation mode codes.
Requires	<p>The CANSPI routines are supported only by MCUs with the SPI module.</p> <p>MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.</p>
Example	<pre>' check whether the CANSPI module is in Normal mode and if it is do something. if (CANSPIGetOperationMode() = CANSPI_MODE_NORMAL) then ... end if</pre>

CANSPIInitialize

Prototype	<code>sub procedure CANSPIInitialize(dim SJW as byte, dim BRP as byte, dim PHSEG1 as byte, dim PHSEG2 as byte, dim PROPSEG as byte, dim CAN_CONFIG_FLAGS as byte)</code>
Returns	Nothing.
Description	<p>Initializes the CANSPI module.</p> <p>Stand-Alone CAN controller in the CANSPI module is set to:</p> <ul style="list-style-type: none"> - Disable CAN capture - Continue CAN operation in Idle mode - Do not abort pending transmissions - Fcan clock: 4*Tcy (Fosc) - Baud rate is set according to given parameters - CAN mode: Normal - Filter and mask registers IDs are set to zero - Filter and mask message frame type is set according to <code>CAN_CONFIG_FLAGS</code> value <p><code>SAM</code>, <code>SEG2PHTS</code>, <code>WAKFIL</code> and <code>DBEN</code> bits are set according to <code>CAN_CONFIG_FLAGS</code> value.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - SJW as defined in CAN controller's datasheet - BRP as defined in CAN controller's datasheet - PHSEG1 as defined in CAN controller's datasheet - PHSEG2 as defined in CAN controller's datasheet - PROPSEG as defined in CAN controller's datasheet - <code>CAN_CONFIG_FLAGS</code> is formed from predefined constants (see CANSPI constants)
Requires	<p><code>CanSpi_CS</code> and <code>CanSpi_Rst</code> variables must be defined before using this sub function.</p> <p>The CANSPI routines are supported only by MCUs with the SPI module.</p> <p>The SPI module needs to be initialized. See the <code>Spi_Init</code> and <code>Spi_Init_Advanced</code> routines.</p> <p>MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.</p>

Example

```
' initialize the CANSPI module with the appropriate baud rate and
message acceptance flags along with the sampling rules
dim Can_Init_Flags as byte
...
Can_Init_Flags = CAN_CONFIG_SAMPLE_THRICE and ' form value to
be used
CAN_CONFIG_PHSEG2_PRG_ON and ' with
CANSPIInitialize
CAN_CONFIG_XTD_MSG and
CAN_CONFIG_DBL_BUFFER_ON and
CAN_CONFIG_VALID_XTD_MSG
...
Spi_Init() ' initialize SPI
module
CANSPIInitialize(1,3,3,3,1,Can_Init_Flags) ' initialize exter-
nal CANSPI module
```

CANSPISetBaudRate

Prototype	<code>sub procedure CANSPISetBaudRate(dim SJW as byte, dim BRP as byte, dim PHSEG1 as byte, dim PHSEG2 as byte, dim PROPSEG as byte, dim CAN_CONFIG_FLAGS as byte)</code>
Returns	Nothing.
Description	<p>Sets the CANSPI module baud rate. Due to complexity of the CAN protocol, you can not simply force a bps value. Instead, use this sub function when the CANSPI module is in Config mode.</p> <p><code>SAM</code>, <code>SEG2PHTS</code> and <code>WAKFIL</code> bits are set according to <code>CAN_CONFIG_FLAGS</code> value. Refer to datasheet for details.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>SJW</code> as defined in CAN controller's datasheet - <code>BRP</code> as defined in CAN controller's datasheet - <code>PHSEG1</code> as defined in CAN controller's datasheet - <code>PHSEG2</code> as defined in CAN controller's datasheet - <code>PROPSEG</code> as defined in CAN controller's datasheet - <code>CAN_CONFIG_FLAGS</code> is formed from predefined constants (see CANSPI constants)
Requires	<p>The CANSPI module must be in Config mode, otherwise the sub function will be ignored. See <code>CANSPISetOperationMode</code>.</p> <p>The CANSPI routines are supported only by MCUs with the SPI module.</p> <p>MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.</p>
Example	<pre>' set required baud rate and sampling rules dim can_config_flags as byte ... CANSPISetOperationMode(CANSPI_MODE_CONFIG, 0xFF) set CONFIGURATION mode (CANSPI module must be in config mode for baud rate settings) can_config_flags = CANSPI_CONFIG_SAMPLE_THRICE and CANSPI_CONFIG_PHSEG2_PRG_ON and CANSPI_CONFIG_STD_MSG and CANSPI_CONFIG_DBL_BUFFER_ON and CANSPI_CONFIG_VALID_XTD_MSG and CANSPI_CONFIG_LINE_FILTER_OFF CANSPISetBaudRate(1, 1, 3, 3, 1, can_config_flags)</pre>

CANSPISetMask

Prototype	<code>sub procedure CANSPISetMask(dim CAN_MASK as byte, dim val as longint, dim CAN_CONFIG_FLAGS as byte)</code>
Returns	Nothing.
Description	<p>Configures mask for advanced filtering of messages. The parameter <code>value</code> is bit-adjusted to the appropriate mask registers.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>CAN_MASK</code>: CANSPI module mask number. Valid values: <code>CANSPI_MASK</code> constants (see CANSPI constants) - <code>val</code>: mask register value - <code>CAN_CONFIG_FLAGS</code>: selects type of message to filter. Valid values: <ul style="list-style-type: none"> <code>CANSPI_CONFIG_ALL_VALID_MSG,</code> <code>CANSPI_CONFIG_MATCH_MSG_TYPE</code> and <code>CANSPI_CONFIG_STD_MSG,</code> <code>CANSPI_CONFIG_MATCH_MSG_TYPE</code> and <code>CANSPI_CONFIG_XTD_MSG.</code> <p>(see CANSPI constants)</p>
Requires	<p>The CANSPI module must be in Config mode, otherwise the sub function will be ignored. See <code>CANSPISetOperationMode</code>.</p> <p>The CANSPI routines are supported only by MCUs with the SPI module.</p> <p>MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.</p>
Example	<pre>' set the appropriate filter mask and message type value CANSPISetOperationMode(CANSPI_MODE_CONFIG,0xFF) set CONFIGURATION mode (CANSPI module must be in config mode for mask settings) ' Set all B1 mask bits to 1 (all filtered bits are relevant): ' Note that -1 is just a cheaper way to write 0xFFFFFFFF. ' Complement will do the trick and fill it up with ones. CANSPISetMask(CANSPI_MASK_B1, -1, CANSPI_CONFIG_MATCH_MSG_TYPE and CANSPI_CONFIG_XTD_MSG)</pre>

CANSPISetFilter

Prototype	<code>sub procedure CANSPISetFilter(dim CAN_FILTER as byte, dim val as longint, dim CAN_CONFIG_FLAGS as byte)</code>
Returns	Nothing.
Description	<p>Configures message filter. The parameter <code>value</code> is bit-adjusted to the appropriate filter registers.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>CAN_FILTER</code>: CANSPI module filter number. Valid values: <code>CANSPI_FILTER</code> constants (see CANSPI constants) - <code>val</code>: filter register value - <code>CAN_CONFIG_FLAGS</code>: selects type of message to filter. Valid values: <ul style="list-style-type: none"> <code>CANSPI_CONFIG_ALL_VALID_MSG,</code> <code>CANSPI_CONFIG_MATCH_MSG_TYPE</code> and <code>CANSPI_CONFIG_STD_MSG,</code> <code>CANSPI_CONFIG_MATCH_MSG_TYPE</code> and <code>CANSPI_CONFIG_XTD_MSG.</code> <p>(see CANSPI constants)</p>
Requires	<p>The CANSPI module must be in Config mode, otherwise the sub function will be ignored. See <code>CANSPISetOperationMode</code>.</p> <p>The CANSPI routines are supported only by MCUs with the SPI module.</p> <p>MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.</p>
Example	<pre>' set the appropriate filter value and message type CANSPISetOperationMode(CANSPI_MODE_CONFIG,0xFF) ' set CONFIGURATION mode (CANSPI module must be in config mode for filter settings) ' Set id of filter B1_F1 to 3: CANSPISetFilter(CANSPI_FILTER_B1_F1, 3, CANSPI_CONFIG_XTD_MSG)</pre>

CANSPIRead

Prototype	<pre>sub function CANSPIRead(dim byref id as longint, dim byref rd_data as array[20] of byte, dim data_len as byte, dim CAN_RX_MSG_FLAGS as byte) as byte</pre>
Returns	<ul style="list-style-type: none"> - 0 if nothing is received - 0xFF if one of the Receive Buffers is full (message received)
Description	<p>If at least one full Receive Buffer is found, it will be processed in the following way:</p> <ul style="list-style-type: none"> - Message ID is retrieved and stored to location provided by the <code>id</code> parameter - Message data is retrieved and stored to a buffer provided by the <code>rd_data</code> parameter - Message length is retrieved and stored to location provided by the <code>data_len</code> parameter - Message flags are retrieved and stored to location provided by the <code>CAN_RX_MSG_FLAGS</code> parameter <p>Parameters:</p> <ul style="list-style-type: none"> - <code>id</code>: message identifier storage address - <code>rd_data</code>: data buffer (an array of bytes up to 8 bytes in length) - <code>data_len</code>: data length storage address. - <code>CAN_RX_MSG_FLAGS</code>: message flags storage address
Requires	<p>The CANSPI module must be in a mode in which receiving is possible. See CANSPISetOperationMode.</p> <p>The CANSPI routines are supported only by MCUs with the SPI module.</p> <p>MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.</p>
Example	<pre>' check the CANSPI module for received messages. If any was received do something. dim msg_rcvd, rx_flags, data_len as byte rd_data as array[8] of byte msg_id as longint ... CANSPISetOperationMode(CANSPI_MODE_NORMAL,0xFF) ' set NORMAL mode (CANSPI module must be in mode in which receive is possible) ... rx_flags = 0 clear message flags if (msg_rcvd = CANSPIRead(msg_id, rd_data, data_len, rx_flags) ... end if</pre>

CANSPIWrite

Prototype	<code>sub function CANSPIWrite(dim id as longint, dim byref wr_data as array[20] of byte, dim data_len as byte, dim CAN_TX_MSG_FLAGS as byte) as byte</code>
Returns	<ul style="list-style-type: none"> - 0 if all Transmit Buffers are busy - 0xFF if at least one Transmit Buffer is available
Description	<p>If at least one empty Transmit Buffer is found, the sub function sends message in the queue for transmission.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>id</code>: CAN message identifier. Valid values: 11 or 29 bit values, depending on message type (standard or extended) - <code>wr_data</code>: data to be sent (an array of bytes up to 8 bytes in length) - <code>data_len</code>: data length. Valid values: 1 to 8 - <code>CAN_RX_MSG_FLAGS</code>: message flags
Requires	<p>The CANSPI module must be in mode in which transmission is possible. See CANSPISetOperationMode.</p> <p>The CANSPI routines are supported only by MCUs with the SPI module.</p> <p>MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.</p>
Example	<pre>' send message extended CAN message with the appropriate ID and data dim tx_flags as byte rd_data as array[8] of byte msg_id as longint ... CANSPISetOperationMode(CAN_MODE_NORMAL, 0xFF) set NORMAL mode (CANSPI must be in mode in which transmission is possible) tx_flags = CANSPI_TX_PRIORITY_0 ands CANSPI_TX_XTD_FRAME ' set message flags CANSPIWrite(msg_id, rd_data, 2, tx_flags)</pre>

CANSPI Constants

There is a number of constants predefined in the CANSPI library. You need to be familiar with them in order to be able to use the library effectively. Check the example at the end of the chapter.

CANSPI_OP_MODE

The CANSPI_OP_MODE constants define CANSPI operation mode. sub function CANSPISetOperationMode expects one of these as it's argument:

```
const
    CANSPI_MODE_BITS      = 0xE0      ' Use this to access opmode bits
    CANSPI_MODE_NORMAL    = 0x00
    CANSPI_MODE_SLEEP     = 0x20
    CANSPI_MODE_LOOP      = 0x40
    CANSPI_MODE_LISTEN    = 0x60
    CANSPI_MODE_CONFIG    = 0x80
```

CANSPI_CONFIG_FLAGS

The CANSPI_CONFIG_FLAGS constants define flags related to the CANSPI module configuration. The sub functions CANSPIInitialize, CANSPISetBaudRate, CANSPISetMask and CANSPISetFilter expect one of these (or a bitwise combination) as their argument:

```
const
    CANSPI_CONFIG_DEFAULT      = 0xFF      ' 11111111

    CANSPI_CONFIG_PHSEG2_PRG_BIT = 0x01
    CANSPI_CONFIG_PHSEG2_PRG_ON  = 0xFF      ' XXXXXXXX1
    CANSPI_CONFIG_PHSEG2_PRG_OFF = 0xFE      ' XXXXXXXX0

    CANSPI_CONFIG_LINE_FILTER_BIT = 0x02
    CANSPI_CONFIG_LINE_FILTER_ON  = 0xFF      ' XXXXXX1X
    CANSPI_CONFIG_LINE_FILTER_OFF = 0xFD      ' XXXXXX0X

    CANSPI_CONFIG_SAMPLE_BIT      = 0x04
    CANSPI_CONFIG_SAMPLE_ONCE     = 0xFF      ' XXXXX1XX
    CANSPI_CONFIG_SAMPLE_THRICE    = 0xFB      ' XXXXX0XX

    CANSPI_CONFIG_MSG_TYPE_BIT    = 0x08
    CANSPI_CONFIG_STD_MSG         = 0xFF      ' XXXX1XXX
    CANSPI_CONFIG_XTD_MSG         = 0xF7      ' XXXX0XXX
```

```

CANSPI_CONFIG_MSG_BITS      = 0x60
CANSPI_CONFIG_ALL_MSG      = 0xFF   ' X11XXXXX
CANSPI_CONFIG_VALID_XTD_MSG = 0xDF   ' X10XXXXX
CANSPI_CONFIG_VALID_STD_MSG = 0xBF   ' X01XXXXX
CANSPI_CONFIG_ALL_VALID_MSG = 0x9F   ' X00XXXXX

```

You may use bitwise `and` to form config byte out of these values. For example:

```

init = CANSPI_CONFIG_SAMPLE_THRICE   and
       CANSPI_CONFIG_PHSEG2_PRG_ON   and
       CANSPI_CONFIG_STD_MSG         and
       CANSPI_CONFIG_DBL_BUFFER_ON   and
       CANSPI_CONFIG_VALID_XTD_MSG   and
       CANSPI_CONFIG_LINE_FILTER_OFF
...
CANSPIInitialize(1, 1, 3, 3, 1, init)  ' initialize CANSPI

```

CANSPI_TX_MSG_FLAGS

CANSPI_TX_MSG_FLAGS are flags related to transmission of a CAN message:

```

const
  CANSPI_TX_PRIORITY_BITS = 0x03
  CANSPI_TX_PRIORITY_0    = 0xFC   ' XXXXXX00
  CANSPI_TX_PRIORITY_1    = 0xFD   ' XXXXXX01
  CANSPI_TX_PRIORITY_2    = 0xFE   ' XXXXXX10
  CANSPI_TX_PRIORITY_3    = 0xFF   ' XXXXXX11

  CANSPI_TX_FRAME_BIT     = 0x08
  CANSPI_TX_STD_FRAME     = 0xFF   ' XXXXX1XX
  CANSPI_TX_XTD_FRAME     = 0xF7   ' XXXXX0XX

  CANSPI_TX_RTR_BIT       = 0x40
  CANSPI_TX_NO_RTR_FRAME  = 0xFF   ' X1XXXXXX
  CANSPI_TX_RTR_FRAME     = 0xBF   ' X0XXXXXX

```

You may use bitwise `and` to adjust the appropriate flags. For example:

```

' form value to be used as sending message flag:
send_config = CANSPI_TX_PRIORITY_0   and
              CANSPI_TX_XTD_FRAME   and
              CANSPI_TX_NO_RTR_FRAME
...
CANSPIWrite(id, data, 1, send_config)

```


CANSPI_RX_MSG_FLAGS

CANSPI_RX_MSG_FLAGS are flags related to reception of CAN message. If a particular bit is set then corresponding meaning is TRUE or else it will be FALSE.

```
const
    CANSPI_RX_FILTER_BITS = 0x07    ' Use this to access filter bits
    CANSPI_RX_FILTER_1   = 0x00
    CANSPI_RX_FILTER_2   = 0x01
    CANSPI_RX_FILTER_3   = 0x02
    CANSPI_RX_FILTER_4   = 0x03
    CANSPI_RX_FILTER_5   = 0x04
    CANSPI_RX_FILTER_6   = 0x05

    CANSPI_RX_OVERFLOW   = 0x08    ' Set if Overflowed else cleared
    CANSPI_RX_INVALID_MSG = 0x10    ' Set if invalid else cleared
    CANSPI_RX_XTD_FRAME   = 0x20    ' Set if XTD message else cleared
    CANSPI_RX_RTR_FRAME   = 0x40    ' Set if RTR message else cleared
    CANSPI_RX_DBL_BUFFERED = 0x80    ' Set if this message was hard-
ware double-buffered
```

You may use bitwise `and` to adjust the appropriate flags. For example:

```
if (MsgFlag and CANSPI_RX_OVERFLOW <> 0) then
    ...
    ' Receiver overflow has occurred.
    ' We have lost our previous message.
end if
```

CANSPI_MASK

The CANSPI_MASK constants define mask codes. sub function CANSPISetMask expects one of these as it's argument:

```
const
    CANSPI_MASK_B1 = 0
    CANSPI_MASK_B2 = 1
```

CANSPI_FILTER

The CANSPI_FILTER constants define filter codes. sub functions CANSPISetFilter expects one of these as it's argument:

```
const
    CANSPI_FILTER_B1_F1 = 0
    CANSPI_FILTER_B1_F2 = 1
    CANSPI_FILTER_B2_F1 = 2
    CANSPI_FILTER_B2_F2 = 3
    CANSPI_FILTER_B2_F3 = 4
    CANSPI_FILTER_B2_F4 = 5
```

Library Example

This is a simple demonstration of CANSPI Library routines usage. First node initiates the communication with the second node by sending some data to its address. The second node responds by sending back the data incremented by 1. First node then does the same and sends incremented data back to second node, etc.

Code for the first CANSPI node:

```
program CanSpilst

dim Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as byte      ' CAN
flags
    Rx_Data_Len as byte                ' Received data length in bytes
    RxTx_Data   as byte[ 8]            ' CAN rx/tx data buffer
    Msg_Rcvd    as byte                ' Reception flag
    Tx_ID, Rx_ID as longint           ' CAN rx and tx ID

' CANSPI module connections
dim CanSpi_CS as sbit at P1.B0
dim CanSpi_Rst as sbit at P1.B2
' End CANSPI module connections

main:
    Can_Init_Flags = 0                '
    Can_Send_Flags = 0                ' Clear flags
    Can_Rcv_Flags = 0                '

    Can_Send_Flags = CAN_TX_PRIORITY_0 and      ' Form value to be used
                    CAN_TX_XTD_FRAME and      ' with CANSPIWrite
                    CAN_TX_NO_RTR_FRAME

    Can_Init_Flags = CAN_CONFIG_SAMPLE_THRICE and ' Form value to be
used
                    CAN_CONFIG_PHSEG2_PRG_ON and      ' with CANSPIInit
                    CAN_CONFIG_XTD_MSG and
                    CAN_CONFIG_DBL_BUFFER_ON and
                    CAN_CONFIG_VALID_XTD_MSG

    Spi_Init()                          ' Initialize SPI module
    CANSPIInitialize(1,3,3,3,1,Can_Init_Flags) ' Initialize external
CANSPI module

    CANSPISetOperationMode(CAN_MODE_CONFIG,0xFF) ' Set CONFIGURATION
mode
```

```

CANSPISetMask(CAN_MASK_B1,-1,CAN_CONFIG_XTD_MSG)      ' Set all
mask1 bits to ones
  CANSPISetMask(CAN_MASK_B2,-1,CAN_CONFIG_XTD_MSG)    ' Set
all mask2 bits to ones
  CANSPISetFilter(CAN_FILTER_B2_F4,3,CAN_CONFIG_XTD_MSG)  ' Set id
of filter B2_F4 to 3

  CANSPISetOperationMode(CAN_MODE_NORMAL,0xFF)        ' Set NORMAL mode

  RxTx_Data[ 0] = 9                                     ' Set initial data to be sent

  Tx_ID = 12111                                       ' Set transmit ID

  CANSPIWrite(Tx_ID, RxTx_Data, 1, Can_Send_Flags)    ' Send
initial message

  while TRUE                                          '
Endless loop
  Msg_Rcvd = CANSPIRead( Rx_ID , RxTx_Data , Rx_Data_Len,
Can_Rcv_Flags)  ' Receive message
    if (Rx_ID = 3) and (Msg_Rcvd <> 0) then
' If message received check id
                                P0 = RxTx_Data[ 0]
' ID correct, output data at PORT0
                                Inc(RxTx_Data[ 0])
' Increment received data
  Delay_ms(10)
  CANSPIWrite(Tx_ID, RxTx_Data, 1, Can_Send_Flags)
' Send incremented data back
    end if
  wend
end.

```

Code for the second CANSPI node:

```

program CanSpi2nd

dim Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as byte      ' CAN
flags
  Rx_Data_Len as byte                                          ' Received data length in bytes
  RxTx_Data as byte[ 8]                                        ' CAN rx/tx data buffer
  Msg_Rcvd as byte                                           ' Reception flag
  Tx_ID, Rx_ID as longint                                     ' CAN rx and tx ID

' CANSPI module connections
dim CanSpi_CS as sbit at P1.B0
dim CanSpi_Rst as sbit at P1.B2
' End CANSPI module connections

```

```

main:
    Can_Init_Flags = 0
    Can_Send_Flags = 0
    Can_Rcv_Flags = 0

    Can_Send_Flags = CAN_TX_PRIORITY_0 and
                    CAN_TX_XTD_FRAME and
with CANSPIWrite
                    CAN_TX_NO_RTR_FRAME

    Can_Init_Flags = CAN_CONFIG_SAMPLE_THRICE and
value to be used
                    CAN_CONFIG_PHSEG2_PRG_ON and
with CANSPIInit
                    CAN_CONFIG_XTD_MSG and
                    CAN_CONFIG_DBL_BUFFER_ON and
                    CAN_CONFIG_VALID_XTD_MSG and
                    CAN_CONFIG_LINE_FILTER_OFF

    Spi_Init()
    CANSPIInitialize(1,3,3,3,1,Can_Init_Flags)
CAN-SPI module

    CANSPISetOperationMode(CAN_MODE_CONFIG,0xFF)
CONFIGURATION mode

    CANSPISetMask(CAN_MASK_B1,-1,CAN_CONFIG_XTD_MSG)
bits to ones
    CANSPISetMask(CAN_MASK_B2,-1,CAN_CONFIG_XTD_MSG)
bits to ones
    CANSPISetFilter(CAN_FILTER_B2_F3,12111,CAN_CONFIG_XTD_MSG)
id of filter B2_F3 to 12111

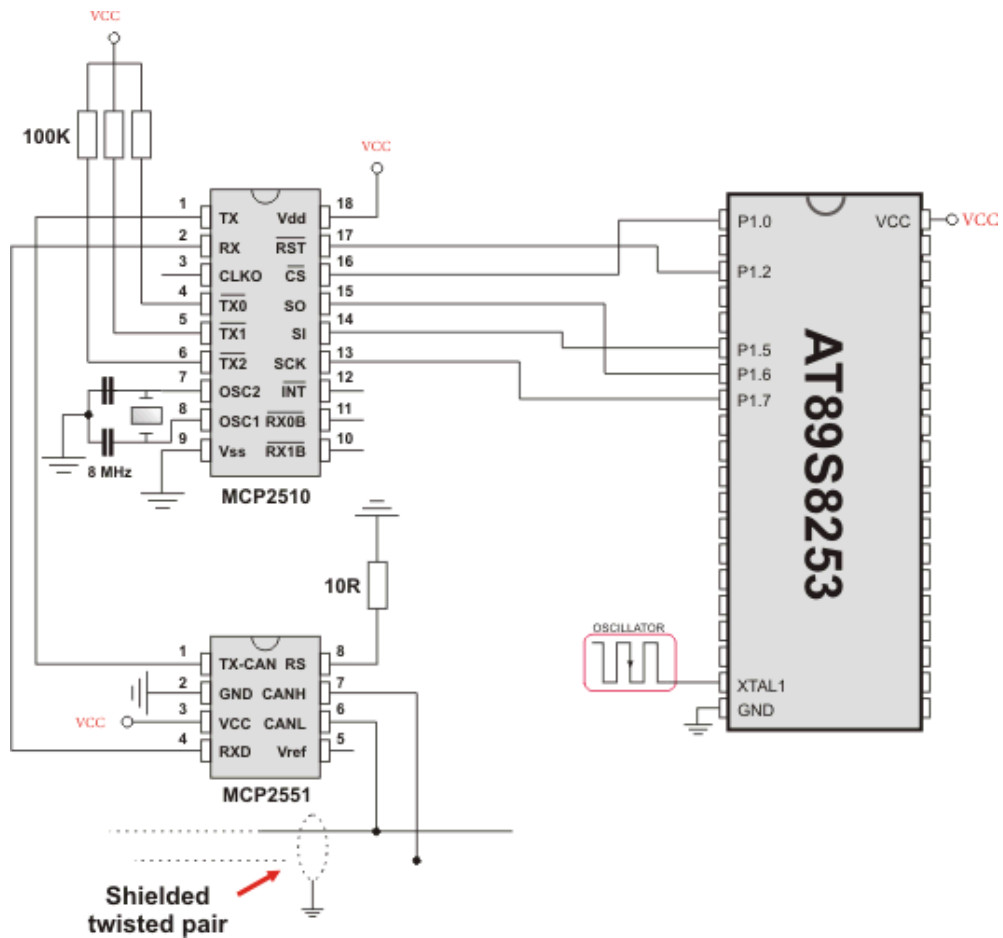
    CANSPISetOperationMode(CAN_MODE_NORMAL,0xFF)
Set NORMAL mode

    Tx_ID = 3
Set tx ID

    while TRUE
        Msg_Rcvd = CANSPIRead( Rx_ID , RxTx_Data , Rx_Data_Len,
Can_Rcv_Flags)
Receive message
        if Rx_ID = 12111 then
            if Msg_Rcvd <> 0 then
                P0 = RxTx_Data[ 0]
                Inc(RxTx_Data[ 0] )
                CANSPIWrite(Tx_ID, RxTx_Data, 1, Can_Send_Flags)
            end if
        end if
    wend
end.

```

HW Connection



Example of interfacing CAN transceiver MCP2510 with MCU via SPI interface

EEPROM LIBRARY

EEPROM data memory is available with a number of 8051 family. The mikroBasic for 8051 includes a library for comfortable work with MCU's internal EEPROM.

Note: EEPROM Library functions implementation is MCU dependent, consult the appropriate MCU datasheet for details about available EEPROM size and address range.

Library Routines

- Eeprom_Read
- Eeprom_Write
- Eeprom_Write_Block

Eeprom_Read

Prototype	<code>sub function Eeprom_Read(dim address as word) as byte</code>
Returns	Byte from the specified address.
Description	<p>Reads data from specified <code>address</code>.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>address</code>: address of the EEPROM memory location to be read.
Requires	Nothing.
Example	<pre>dim eeAddr as word temp as byte ... eeAddr = 2 temp = Eeprom_Read(eeAddr)</pre>

Eeprom_Write

Prototype	<code>sub function Eeprom_Write (dim address as word, dim wrdata as byte) as byte</code>
Returns	<ul style="list-style-type: none"> - 0 writing was successful - 1 if error occurred
Description	<p>Writes wrdata to specified address.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>address</code>: address of the EEPROM memory location to be written. - <code>wrdata</code>: data to be written. <p>Note: Specified memory location will be erased before writing starts.</p>
Requires	Nothing.
Example	<pre>dim eeWrite as byte = 0x55 wrAddr as word = 0x732 ... eeWrite = 0x55 wrAddr = 0x732 Eeprom_Write(wrAddr, eeWrite)</pre>

Eeprom_Write_Block

Prototype	<code>sub function Eeprom_Write_Block(dim address as word, dim byref ptrdata as byte) as byte</code>
Returns	<ul style="list-style-type: none"> - 0 writing was successful - 1 if error occurred
Description	<p>Writes one EEPROM row (32 bytes block) of data.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>address</code>: starting address of the EEPROM memory block to be written. - <code>ptrdata</code>: data block to be written. <p>Note: Specified memory block will be erased before writing starts.</p>
Requires	<p>EEPROM module must support block write operations.</p> <p>It is the user's responsibility to maintain proper address alignment. In this case, address has to be a multiply of 32, which is the size (in bytes) of one row of MCU's EEPROM memory.</p>
Example	<pre>dim wrAddr as word iArr as string[16] ... wrAddr = 0x0100 iArr = 'mikroElektronika' Eeprom_Write_Block(wrAddr, iArr)</pre>

Library Example

This example demonstrates using the EEPROM Library with AT89S8253 MCU.

First, some data is written to EEPROM in byte and block mode; then the data is read from the same locations and displayed on P0, P1 and P2.

```

program Eeprom

dim dat as byte[ 32]           ' Data buffer, loop variable
      i as byte

main:
  for i = 0 to 31
    dat[i] = i                   ' Fill data buffer
  next i

  Eeprom_Write(2,0xAA)          ' Write some data at address 2
  Eeprom_Write(0x732,0x55)      ' Write some data at address 0x732
  Eeprom_Write_Block(0x100,dat) ' Write 32 bytes block at
address 0x100

  Delay_ms(1000)                ' Blink P0 and P1 diodes
  P0 = 0xFF                     '   to indicate reading start
  P1 = 0xFF
  Delay_ms(1000)
  P0 = 0x00
  P1 = 0x00
  Delay_ms(1000)

  P0 = Eeprom_Read(2)           ' Read data from address 2
and display it on PORT0
  P1 = Eeprom_Read(0x732)      ' Read data from address 0x732
and display it on PORT1
  Delay_ms(1000)

  for i = 0 to 31              ' Read 32 bytes block from
address 0x100
    P2 = Eeprom_Read(0x100+i)  '   and display data on PORT2
    Delay_ms(100)
  next i
end.

```

Graphic LCD Library

The mikroBasic for 8051 provides a library for operating Graphic LCD 128x64 (with commonly used Samsung KS108/KS107 controller).

For creating a custom set of GLCD images use GLCD Bitmap Editor Tool.

External dependencies of Graphic LCD Library

The following variables must be defined in all projects using Graphic LCD Library:	Description:	Example :
<code>dim GLCD_DataPort as byte external volatile sfr</code>	LCD Data Port.	<code>dim GLCD_DataPort as byte at P0 sfr</code>
<code>dim GLCD_CS1 as sbit external</code>	Chip Select 1 line.	<code>dim GLCD_CS1 as sbit at P2.B0</code>
<code>dim GLCD_CS2 as sbit external</code>	Chip Select 2 line.	<code>dim GLCD_CS2 as sbit at P2.B0</code>
<code>dim GLCD_RS as sbit external</code>	Register select line.	<code>dim GLCD_RS as sbit at P2.B0</code>
<code>dim GLCD_RW as sbit external</code>	Read/Write line.	<code>dim GLCD_RW as sbit at P2.B0</code>
<code>dim GLCD_RST as sbit external</code>	Reset line.	<code>dim GLCD_RST as sbit at P2.B0</code>
<code>dim GLCD_EN as sbit external</code>	Enable line.	<code>dim GLCD_EN as sbit at P2.B0</code>

Library Routines

Basic routines:

- Glcd_Init
- Glcd_Set_Side
- Glcd_Set_X
- Glcd_Set_Page
- Glcd_Read_Data
- Glcd_Write_Data

Advanced routines:

- Glcd_Fill
- Glcd_Dot
- Glcd_Line
- Glcd_V_Line
- Glcd_H_Line
- Glcd_Rectangle
- Glcd_Box
- Glcd_Circle
- Glcd_Set_Font
- Glcd_Write_Char
- Glcd_Write_Text
- Glcd_Image

Glcd_Init

Prototype	<code>sub procedure Glcd_Init()</code>
Returns	Nothing.
Description	Initializes the GLCD module. Each of the control lines is both port and pin configurable, while data lines must be on a single port (pins <0:7>).
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - GLCD_CS1 : chip select 1 signal pin - GLCD_CS2 : chip select 2 signal pin - GLCD_RS : register select signal pin - GLCD_RW : read/write signal pin - GLCD_EN : enable signal pin - GLCD_RST : reset signal pin - GLCD_DataPort : data port <p>must be defined before using this function.</p>
Example	<pre>' glcd pinout settings dim GLCD_DataPort as byte at P0 sfr dim GLCD_CS1 as sbit at P2.B0 dim GLCD_CS2 as sbit at P2.B1 dim GLCD_RS as sbit at P2.B2 dim GLCD_RW as sbit at P2.B3 dim GLCD_RST as sbit at P2.B5 dim GLCD_EN as sbit at P2.B4 ... Glcd_Init()</pre>

Glcd_Set_Side

Prototype	<code>sub procedure Glcd_Set_Side(dim x_pos as byte)</code>
Returns	Nothing.
Description	<p>Selects GLCD side. Refer to the GLCD datasheet for detailed explanation.</p> <p>Parameters :</p> <p>- <code>x_pos</code>: position on x-axis. Valid values: 0..127</p> <p>The parameter <code>x_pos</code> specifies the GLCD side: values from 0 to 63 specify the left side, values from 64 to 127 specify the right side.</p> <p>Note: For side, x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<p>The following two lines are equivalent, and both of them select the left side of GLCD:</p> <pre>Glcd_Select_Side(0) Glcd_Select_Side(10)</pre>

Glcd_Set_X

Prototype	<code>sub procedure Glcd_Set_X(dim x_pos as byte)</code>
Returns	Nothing.
Description	<p>Sets x-axis position to <code>x_pos</code> dots from the left border of GLCD within the selected side.</p> <p>Parameters :</p> <p>- <code>x_pos</code>: position on x-axis. Valid values: 0..63</p> <p>Note: For side, x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<pre>Glcd_Set_X(25)</pre>

Glcd_Set_Page

Prototype	<code>sub procedure Glcd_Set_Page(dim page as byte)</code>
Returns	Nothing.
Description	<p>Selects page of the GLCD.</p> <p>Parameters :</p> <p>- <code>page</code>: page number. Valid values: 0..7</p> <p>Note: For side, x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<code>Glcd_Set_Page(5)</code>

Glcd_Read_Data

Prototype	<code>sub function Glcd_Read_Data() as byte</code>
Returns	One byte from GLCD memory.
Description	Reads data from from the current location of GLCD memory and moves to the next location.
Requires	<p>GLCD needs to be initialized, see <code>Glcd_Init</code> routine.</p> <p>GLCD side, x-axis position and page should be set first. See functions <code>Glcd_Set_Side</code>, <code>Glcd_Set_X</code>, and <code>Glcd_Set_Page</code>.</p>
Example	<pre>dim data as byte ... data = Glcd_Read_Data()</pre>

Glcd_Write_Data

Prototype	<code>sub procedure Glcd_Write_Data(dim ddata as byte)</code>
Returns	Nothing.
Description	Writes one byte to the current location in GLCD memory and moves to the next location. Parameters : - <code>ddata</code> : data to be written
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine. GLCD side, x-axis position and page should be set first. See functions <code>Glcd_Set_Side</code> , <code>Glcd_Set_X</code> , and <code>Glcd_Set_Page</code> .
Example	<pre>dim data as byte ... Glcd_Write_Data(data)</pre>

Glcd_Fill

Prototype	<code>sub procedure Glcd_Fill(dim pattern as byte)</code>
Returns	Nothing.
Description	Fills GLCD memory with the byte <code>pattern</code> . Parameters : - <code>pattern</code> : byte to fill GLCD memory with To clear the GLCD screen, use <code>Glcd_Fill(0)</code> . To fill the screen completely, use <code>Glcd_Fill(0xFF)</code> .
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<pre>' Clear screen Glcd_Fill(0)</pre>

Glcd_Dot

Prototype	<code>sub procedure Glcd_Dot(dim x_pos as byte, dim y_pos as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a dot on GLCD at coordinates (x_pos, y_pos).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_pos</code>: x position. Valid values: 0..127 - <code>y_pos</code>: y position. Valid values: 0..63 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines a dot state: 0 clears dot, 1 puts a dot, and 2 inverts dot state.</p> <p>Note: For x and y axis layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<pre>' Invert the dot in the upper left corner Glcd_Dot(0, 0, 2)</pre>

Glcd_Line

Prototype	<code>sub procedure Glcd_Line(dim x_start as integer, dim y_start as integer, dim x_end as integer, dim y_end as integer, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a line on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_start</code>: x coordinate of the line start. Valid values: 0..127 - <code>y_start</code>: y coordinate of the line start. Valid values: 0..63 - <code>x_end</code>: x coordinate of the line end. Valid values: 0..127 - <code>y_end</code>: y coordinate of the line end. Valid values: 0..63 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the line color: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<pre>' Draw a line between dots (0,0) and (20,30) Glcd_Line(0, 0, 20, 30, 1)</pre>

Glcd_V_Line

Prototype	<code>sub procedure Glcd_V_Line(dim y_start as byte, dim y_end as byte, dim x_pos as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a vertical line on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>y_start</code>: y coordinate of the line start. Valid values: 0..63 - <code>y_end</code>: y coordinate of the line end. Valid values: 0..63 - <code>x_pos</code>: x coordinate of vertical line. Valid values: 0..127 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the line color: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<code>' Draw a vertical line between dots (10,5) and (10,25)</code> <code>Glcd_V_Line(5, 25, 10, 1)</code>

Glcd_H_Line

Prototype	<code>sub procedure Glcd_V_Line(dim x_start as byte, dim x_end as byte, dim y_pos as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a horizontal line on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_start</code>: x coordinate of the line start. Valid values: 0..127 - <code>x_end</code>: x coordinate of the line end. Valid values: 0..127 - <code>y_pos</code>: y coordinate of horizontal line. Valid values: 0..63 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the line color: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<code>' Draw a horizontal line between dots (10,20) and (50,20)</code> <code>Glcd_H_Line(10, 50, 20, 1)</code>

Glcd_Rectangle

Prototype	<code>sub procedure Glcd_Rectangle(dim x_upper_left as byte, dim y_upper_left as byte, dim x_bottom_right as byte, dim y_bottom_right as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a rectangle on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_upper_left</code>: x coordinate of the upper left rectangle corner. Valid values: 0..127 - <code>y_upper_left</code>: y coordinate of the upper left rectangle corner. Valid values: 0..63 - <code>x_bottom_right</code>: x coordinate of the lower right rectangle corner. Valid values: 0..127 - <code>y_bottom_right</code>: y coordinate of the lower right rectangle corner. Valid values: 0..63 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the rectangle border: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<code>' Draw a rectangle between dots (5,5) and (40,40) Glcd_Rectangle(5, 5, 40, 40, 1)</code>

Glcd_Box

Prototype	<code>sub procedure Glcd_Box(dim x_upper_left as byte, dim y_upper_left as byte, dim x_bottom_right as byte, dim y_bottom_right as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a box on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_upper_left</code>: x coordinate of the upper left box corner. Valid values: 0..127 - <code>y_upper_left</code>: y coordinate of the upper left box corner. Valid values: 0..63 - <code>x_bottom_right</code>: x coordinate of the lower right box corner. Valid values: 0..127 - <code>y_bottom_right</code>: y coordinate of the lower right box corner. Valid values: 0..63 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the box fill: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<code>' Draw a box between dots (5,15) and (20,40) Glcd_Box(5, 15, 20, 40, 1)</code>

Glcd_Circle

Prototype	<code>sub procedure Glcd_Circle(dim x_center as integer, dim y_center as integer, dim radius as integer, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a circle on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_center</code>: x coordinate of the circle center. Valid values: 0..127 - <code>y_center</code>: y coordinate of the circle center. Valid values: 0..63 - <code>radius</code>: radius size - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the circle line: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<code>' Draw a circle with center in (50,50) and radius=10 Glcd_Circle(50, 50, 10, 1)</code>

Glcd_Set_Font

Prototype	<code>sub procedure Glcd_Set_Font(dim byref const ActiveFont as ^byte, dim FontWidth as byte, dim FontHeight as byte, dim FontOffs as word)</code>
Returns	Nothing.
Description	<p>Sets font that will be used with Glcd_Write_Char and Glcd_Write_Text routines.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>activeFont</code>: font to be set. Needs to be formatted as an array of char - <code>aFontWidth</code>: width of the font characters in dots. - <code>aFontHeight</code>: height of the font characters in dots. - <code>aFontOffs</code>: number that represents difference between the mikroBasic for 8051 character set and regular ASCII set (eg. if 'A' is 65 in ASCII character, and 'A' is 45 in the mikroBasic for 8051 character set, aFontOffs is 20). Demo fonts supplied with the library have an offset of 32, which means that they start with space. <p>The user can use fonts given in the file “__Lib_GLCDFonts.mpas” file located in the Uses folder or create his own fonts.</p>
Requires	GLCD needs to be initialized, see Glcd_Init routine.
Example	<code>' Use the custom 5x7 font "myfont" which starts with space (32): Glcd_Set_Font(myfont, 5, 7, 32)</code>

Glcd_Write_Char

Prototype	<code>sub procedure Glcd_Write_Char(dim chr as byte, dim x_pos as byte, dim page_num as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Prints character on the GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none">- <code>chr</code>: character to be written- <code>x_pos</code>: character starting position on x-axis. Valid values: 0..(127-FontWidth)- <code>page_num</code>: the number of the page on which character will be written. Valid values: 0..7- <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the character: 0 white, 1 black, and 2 inverts each dot.</p> <p>Note: For x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine. Use <code>Glcd_Set_Font</code> to specify the font for display; if no font is specified, then default 5x8 font supplied with the library will be used.
Example	<pre>' Write character 'C' on the position 10 inside the page 2: Glcd_Write_Char('C', 10, 2, 1)</pre>

Glcd_Write_Text

Prototype	<code>sub procedure Glcd_Write_Text(dim byref text as string[20], dim x_pos as byte, dim page_num as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Prints text on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>text</code>: text to be written - <code>x_pos</code>: text starting position on x-axis. - <code>page_num</code>: the number of the page on which text will be written. Valid values: 0..7 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the text: 0 white, 1 black, and 2 inverts each dot.</p> <p>Note: For x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine. Use <code>Glcd_Set_Font</code> to specify the font for display; if no font is specified, then default 5x8 font supplied with the library will be used.
Example	<code>' Write text "Hello world!" on the position 10 inside the page 2: Glcd_Write_Text("Hello world!", 10, 2, 1)</code>

Glcd_Image

Prototype	<code>sub procedure Glcd_Image(dim byref const image as ^byte)</code>
Returns	Nothing.
Description	<p>Displays bitmap on GLCD.</p> <p>Parameters :</p> <p>- <code>image</code>: image to be displayed. Bitmap array must be located in code memory.</p> <p>Use the mikroBasic for 8051 integrated GLCD Bitmap Editor to convert image to a constant array suitable for displaying on GLCD.</p>
Requires	GLCD needs to be initialized, see <code>Glcd_Init</code> routine.
Example	<pre>' Draw image my_image on GLCD Glcd_Image(my_image)</pre>

Library Example

The following example demonstrates routines of the GLCD library: initialization, clear(pattern fill), image displaying, drawing lines, circles, boxes and rectangles, text displaying and handling.

```
program Glcd

' Declarations
include bitmap
' end Declarations

' Glcd module connections
dim GLCD_DataPort as byte at P0 sfr          ' GLCD data port

dim GLCD_CS1 as sbit at P2.B0              ' GLCD chip select 1 signal
dim GLCD_CS2 as sbit at P2.B1              ' GLCD chip select 2 signal
dim GLCD_RS  as sbit at P2.B2              ' GLCD register select signal
dim GLCD_RW  as sbit at P2.B3              ' GLCD read/write signal
dim GLCD_RST as sbit at P2.B5              ' GLCD reset signal
dim GLCD_EN  as sbit at P2.B4              ' GLCD enable signal
' End Glcd module connections

sub procedure delay2S()                    ' 2 seconds delay sub function
    Delay_ms(2000)
end sub

dim conter as word
    someText as char[ 17]
```

```

main:
  Glcd_Init()           ' Initialize GLCD
  Glcd_Fill(0x00)      ' Clear GLCD

  while TRUE
    Glcd_Image(@advanced8051_bmp) ' Draw image
    Delay2S()

    Glcd_Fill(0x00)    ' Clear GLCD

    Glcd_Box(62,40,124,56,1) ' Draw box
    Glcd_Rectangle(5,5,84,35,1) ' Draw rectangle
    Glcd_Line(0, 63, 127, 0,1) ' Draw line

    delay2S()

    for conter = 5 to 59           ' Draw horizontal and
vertical lines
      Delay_ms(50)
      Glcd_V_Line(2, 54, conter, 2)
      Glcd_H_Line(2, 120, conter, 2)
    next conter

    Delay2S()

    Glcd_Fill(0x00)

    Glcd_Set_Font(@Character8x8, 8, 8, 32) ' Choose font
    Glcd_Write_Text("mikroE", 5, 7, 2) ' Write string

    for conter = 1 to 10           ' Draw circles
      Glcd_Circle(63,32, 3*conter, 1)
    next conter
    Delay2S()

    Glcd_Box(12,20, 70,63, 2)    ' Draw box
    Delay2S()

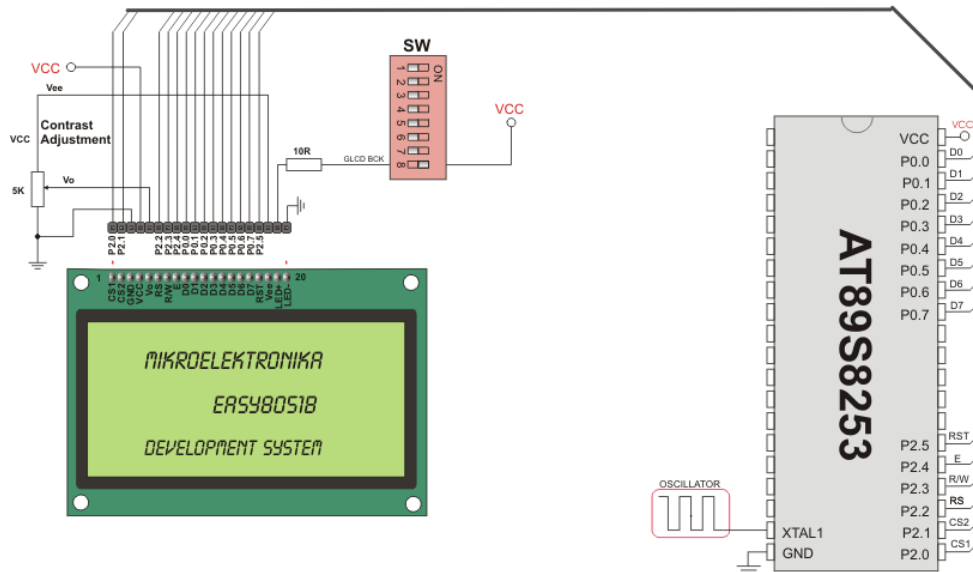
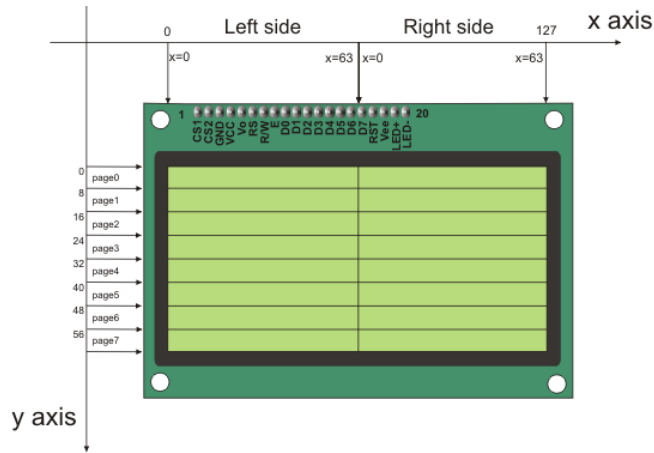
    Glcd_Set_Font(@FontSystem5x8, 5, 8, 32) ' Change font
    someText = "Font 5x8"
    Glcd_Write_Text(someText, 5, 3, 1) ' Write string
    Delay2S()

    Glcd_Set_Font(@System3x6, 3, 6, 32) ' Change font
    someText = "SMALL:FONT:3X6"
    Glcd_Write_Text(someText, 20, 7, 0) ' Write string
    Delay2S()
  wend

end.

```

HW Connection



GLCD HW connection

KEYPAD LIBRARY

The mikroBasic for 8051 provides a library for working with 4x4 keypad. The library routines can also be used with 4x1, 4x2, or 4x3 keypad. For connections explanation see schematic at the bottom of this page.

Note: Since sampling lines for 8051 MCUs are activated by logical zero Keypad Library can not be used with hardwares that have protective diodes connected with anode to MCU side, such as mikroElektronika's Keypad extra board HW.Rev v1.20

External dependencies of Keypad Library

The following variable must be defined in all projects using Keypad Library:	Description:	Example :
<code>dim keypadPort as byte external sfr</code>	Keypad Port.	<code>dim keypadPort as byte at P0 sfr</code>

Library Routines

- Keypad_Init
- Keypad_Key_Press
- Keypad_Key_Click

Keypad_Init

Prototype	<code>sub procedure Keypad_Init()</code>
Returns	Nothing.
Description	Initializes port for working with keypad.
Requires	<code>keypadPort</code> variable must be defined before using this function.
Example	<pre>' Initialize P0 for communication with keypad dim keypadPort as byte at P0 sfr ... Keypad_Init()</pre>

Keypad_Key_Press

Prototype	<code>sub function Keypad_Key_Press() as byte</code>
Returns	The code of a pressed key (1..16). If no key is pressed, returns 0.
Description	Reads the key from keypad when key gets pressed.
Requires	Port needs to be initialized for working with the Keypad library, see Keypad_Init.
Example	<pre>dim kp as byte ... kp = Keypad_Key_Press()</pre>

Keypad_Key_Click

Prototype	<code>sub function Keypad_Key_Click() as byte</code>
Returns	The code of a clicked key (1..16). If no key is clicked, returns 0.
Description	Call to <code>Keypad_Key_Click</code> is a blocking call: the function waits until some key is pressed and released. When released, the function returns 1 to 16, depending on the key. If more than one key is pressed simultaneously the function will wait until all pressed keys are released. After that the function will return the code of the first pressed key.
Requires	Port needs to be initialized for working with the Keypad library, see Keypad_Init.
Example	<pre>dim kp as byte ... kp = Keypad_Key_Click()</pre>

Library Example

This is a simple example of using the Keypad Library. It supports keypads with 1..4 rows and 1..4 columns. The code being returned by `Keypad_Key_Click()` function is in range from 1..16. In this example, the code returned is transformed into ASCII codes [0..9,A..F] and displayed on LCD. In addition, a small single-byte counter displays in the second LCD row number of key presses.

```

program Keypad

dim kp, cnt, oldstate as byte
    txt as byte[ 5]

' Keypad module connections
dim keypadPort as byte at P0 sfr
' End Keypad module connections

' Lcd pinout definition
dim LCD_RS as sbit at P2.B0
dim LCD_EN as sbit at P2.B1

dim LCD_D7 as sbit at P2.B5
dim LCD_D6 as sbit at P2.B4
dim LCD_D5 as sbit at P2.B3
dim LCD_D4 as sbit at P2.B2
' end Lcd definitions

main:
    oldstate = 0
    cnt = 0
    Keypad_Init()
    Lcd_Init()
    Lcd_Cmd(LCD_CLEAR)
    Lcd_Cmd(LCD_CURSOR_OFF)
    Lcd_Out(1, 1, "Key  :")
on LCD
    Lcd_Out(2, 1, "Times:")

    while TRUE

        kp = 0
        pressed and released
            while ( kp = 0 )
                kp = Keypad_Key_Click()
            wend
            output, transform key to it's ASCII value
                select case kp
                    'case 10: kp = 42   ' "*"           ' Uncomment this block
for keypad4x3
                    'case 11: kp = 48   ' "0"
                    'case 12: kp = 35   ' "#"
                    'default: kp += 48

' Reset counter
' Initialize Keypad
' Initialize LCD
' Clear display
' Cursor off
' Write message text

' Reset key code variable
' Wait for key to be
' Store key code in kp
' Prepare value for

```

```

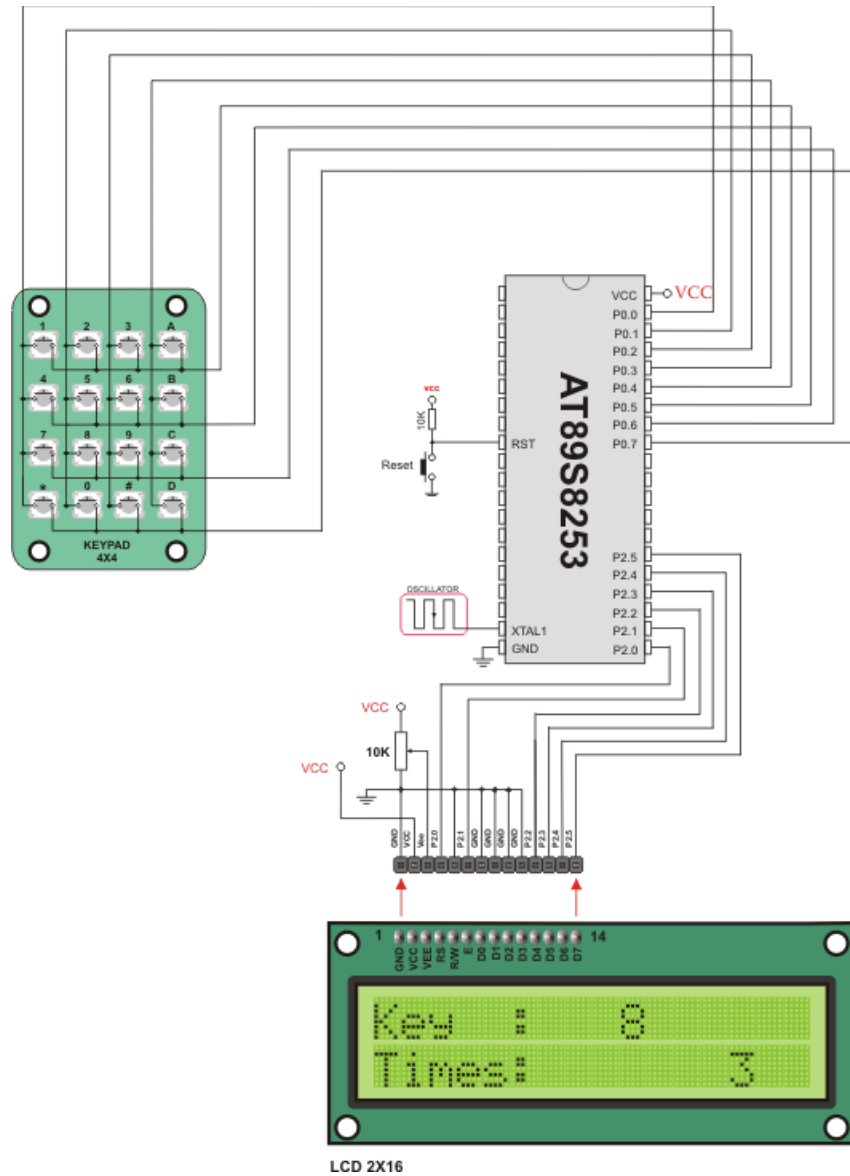
        case 1
            kp = 49 ' 1          ' Uncomment this block for keypad4x4
        case 2
            kp = 50 ' 2
        case 3
            kp = 51 ' 3
        case 4
            kp = 65 ' A
        case 5
            kp = 52 ' 4
        case 6
            kp = 53 ' 5
        case 7
            kp = 54 ' 6
        case 8
            kp = 66 ' B
        case 9
            kp = 55 ' 7
        case 10
            kp = 56 ' 8
        case 11
            kp = 57 ' 9
        case 12
            kp = 67 ' C
        case 13
            kp = 42 ' *
        case 14
            kp = 48 ' 0
        case 15
            kp = 35 ' #
        case 16
            kp = 68 ' D
    end select ' end case

    if (kp <> oldstate) then          ' Pressed key differs
from previous
        cnt = 1
        oldstate = kp
    else                               ' Pressed key is same as previous
        Inc(cnt)
    end if
    Lcd_Chr(1, 10, kp)                 ' Print key ASCII value on LCD

    if (cnt = 255) then                ' If counter variable overflow
        cnt = 0
        Lcd_Out(2, 10, " ")
    end if

        WordToStr(cnt, txt)           ' Transform counter
value to string
        Lcd_Out(2, 10, txt)           ' Display counter value
on LCD
    wend
end.
```

HW Connection



4x4 Keypad connection scheme

LCD LIBRARY

The mikroBasic for 8051 provides a library for communication with LCDs (with HD44780 compliant controllers) through the 4-bit interface. An example of LCD connections is given on the schematic at the bottom of this page.

For creating a set of custom LCD characters use LCD Custom Character Tool.

External dependencies of LCD Library

The following variables must be defined in all projects using LCD Library:	Description:	Example :
<code>dim LCD_RS as sbit external</code>	Register Select line.	<code>dim LCD_RS as sbit at P2.B0</code>
<code>dim LCD_EN as sbit external</code>	Enable line.	<code>dim LCD_EN as sbit at P2.B1</code>
<code>dim LCD_D7 as sbit external</code>	Data 7 line.	<code>dim LCD_D7 as sbit at P2.B5</code>
<code>dim LCD_D6 as sbit external</code>	Data 6 line.	<code>dim LCD_D6 as sbit at P2.B4</code>
<code>dim LCD_D5 as sbit external</code>	Data 5 line.	<code>dim LCD_D5 as sbit at P2.B3</code>
<code>dim LCD_D4 as sbit external</code>	Data 4 line.	<code>dim LCD_D4 as sbit at P2.B2</code>

Library Routines

- Lcd_Init
- Lcd_Out
- Lcd_Out_Cp
- Lcd_Chr
- Lcd_Chr_Cp
- Lcd_Cmd

Lcd_Init

Prototype	<code>sub procedure Lcd_Init()</code>
Returns	Nothing.
Description	Initializes LCD module.
Requires	<p>Global variables:</p> <ul style="list-style-type: none"> - <code>LCD_D7</code>: data bit 7 - <code>LCD_D6</code>: data bit 6 - <code>LCD_D5</code>: data bit 5 - <code>LCD_D4</code>: data bit 4 - <code>RS</code>: register select (data/instruction) signal pin - <code>EN</code>: enable signal pin <p>must be defined before using this function.</p>
Example	<pre>' lcd pinout settings dim LCD_RS as sbit at P2.B0 LCD_EN as sbit at P2.B1 LCD_D7 as sbit at P2.B5 LCD_D6 as sbit at P2.B4 LCD_D5 as sbit at P2.B3 LCD_D4 as sbit at P2.B2 ... Lcd_Init()</pre>

Lcd_Out

Prototype	<code>sub procedure Lcd_Out(dim row as byte, dim column as byte, dim byref text as string[20])</code>
Returns	Nothing.
Description	Prints text on LCD starting from specified position. Both string variables and literals can be passed as a text. Parameters : - <code>row</code> : starting position row number - <code>column</code> : starting position column number - <code>text</code> : text to be written
Requires	The LCD module needs to be initialized. See Lcd_Init routine.
Example	' Write text "Hello!" on LCD starting from row 1, column 3: <code>Lcd_Out(1, 3, "Hello!")</code>

Lcd_Out_Cp

Prototype	<code>sub procedure Lcd_Out_Cp(dim byref text as string[19])</code>
Returns	Nothing.
Description	Prints text on LCD at current cursor position. Both string variables and literals can be passed as a text. Parameters : - <code>text</code> : text to be written
Requires	The LCD module needs to be initialized. See Lcd_Init routine.
Example	' Write text "Here!" at current cursor position: <code>Lcd_Out_Cp("Here!")</code>

Lcd_Chr

Prototype	<code>sub procedure Lcd_Chr(dim row as byte, dim column as byte, dim out_char as byte)</code>
Returns	Nothing.
Description	Prints character on LCD at specified position. Both variables and literals can be passed as a character. Parameters : - <code>row</code> : writing position row number - <code>column</code> : writing position column number - <code>out_char</code> : character to be written
Requires	The LCD module needs to be initialized. See Lcd_Init routine.
Example	<code>' Write character "i" at row 2, column 3: Lcd_Chr(2, 3, 'i')</code>

Lcd_Chr_Cp

Prototype	<code>sub procedure Lcd_Chr_Cp(dim out_char as byte)</code>
Returns	Nothing.
Description	Prints character on LCD at current cursor position. Both variables and literals can be passed as a character. Parameters : - <code>out_char</code> : character to be written
Requires	The LCD module needs to be initialized. See Lcd_Init routine.
Example	<code>' Write character "e" at current cursor position: Lcd_Chr_Cp('e')</code>

Lcd_Cmd

Prototype	<code>sub procedure Lcd_Cmd(dim out_char as byte)</code>
Returns	Nothing.
Description	<p>Sends command to LCD.</p> <p>Parameters :</p> <p>- <code>out_char</code>: command to be sent</p> <p>Note: Predefined constants can be passed to the function, see Available LCD Commands.</p>
Requires	The LCD module needs to be initialized. See Lcd_Init table.
Example	<pre>' Clear LCD display: Lcd_Cmd(LCD_CLEAR)</pre>

Available LCD Commands

Lcd Command	Purpose
<code>LCD_FIRST_ROW</code>	Move cursor to the 1st row
<code>LCD_SECOND_ROW</code>	Move cursor to the 2nd row
<code>LCD_THIRD_ROW</code>	Move cursor to the 3rd row
<code>LCD_FOURTH_ROW</code>	Move cursor to the 4th row
<code>LCD_CLEAR</code>	Clear display
<code>LCD_RETURN_HOME</code>	Return cursor to home position, returns a shifted display to its original position. Display data RAM is unaffected.
<code>LCD_CURSOR_OFF</code>	Turn off cursor
<code>LCD_UNDERLINE_ON</code>	Underline cursor on
<code>LCD_BLINK_CURSOR_ON</code>	Blink cursor on
<code>LCD_MOVE_CURSOR_LEFT</code>	Move cursor left without changing display data RAM
<code>LCD_MOVE_CURSOR_RIGHT</code>	Move cursor right without changing display data RAM
<code>LCD_TURN_ON</code>	Turn LCD display on
<code>LCD_TURN_OFF</code>	Turn LCD display off
<code>LCD_SHIFT_LEFT</code>	Shift display left without changing display data RAM
<code>LCD_SHIFT_RIGHT</code>	Shift display right without changing display data RAM

Library Example

The following code demonstrates usage of the LCD Library routines:

```

program Lcd

' LCD module connections
dim LCD_RS as sbit at P2.0
dim LCD_EN as sbit at P2.1

dim LCD_D7 as sbit at P2.5
dim LCD_D6 as sbit at P2.4
dim LCD_D5 as sbit at P2.3
dim LCD_D4 as sbit at P2.2
' End LCD module connections

dim txt1 as char[ 16]
    txt2 as char[ 9]
    txt3 as char[ 8]
    txt4 as char[ 7]
    i as byte                ' Loop variable

sub procedure Move_Delay()    ' Function used for text moving
    Delay_ms(400)            ' You can change the moving speed here
end sub

main:
    txt1 = "mikroElektronika"
    txt2 = "Easy8051B"
    txt3 = "lcd 4bit"
    txt4 = "example"
    Lcd_Init()                ' Initialize LCD
    Lcd_Cmd(LCD_CLEAR)        ' Clear display
    Lcd_Cmd(LCD_CURSOR_OFF)  ' Cursor off
    LCD_Out(1,6,txt3)         ' Write text in first row
    LCD_Out(2,6,txt4)         ' Write text in second row
    Delay_ms(2000)
    Lcd_Cmd(LCD_CLEAR)        ' Clear display

    LCD_Out(1,1,txt1)         ' Write text in first row
    LCD_Out(2,4,txt2)         ' Write text in second row
    Delay_ms(500)

' Moving texts
for i = 0 to 3                ' Move text to the right 4 times
    Lcd_Cmd(LCD_SHIFT_RIGHT)
    Move_Delay()
next i

```

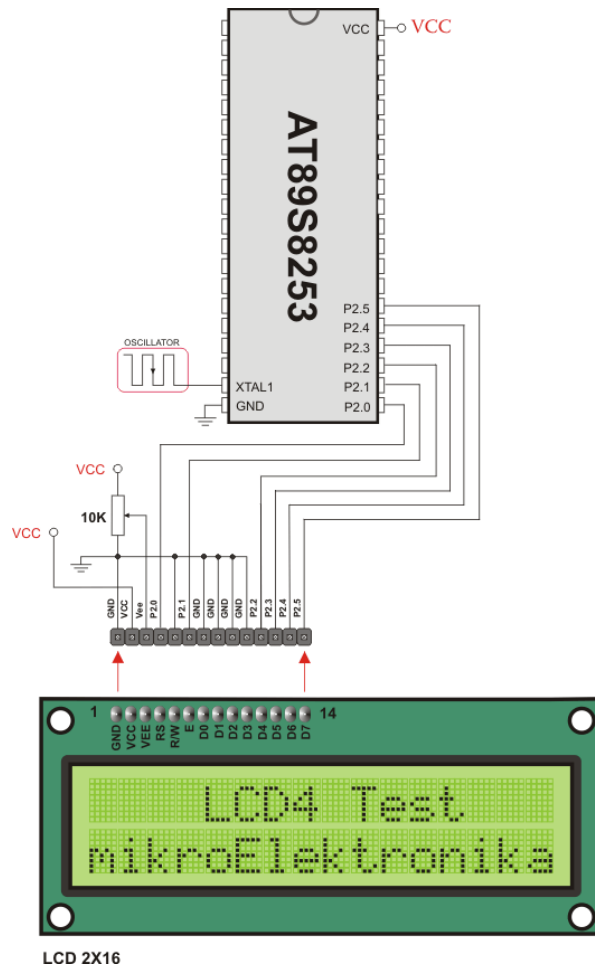
```

while TRUE                                     ' Endless loop
  for i = 0 to 6                               ' Move text to the left 7 times
    Lcd_Cmd(LCD_SHIFT_LEFT)
    Move_Delay()
  next i

  for i = 0 to 6                               ' Move text to the right 7 times
    Lcd_Cmd(LCD_SHIFT_RIGHT)
    Move_Delay()
  next i

wend
end.

```



LCD HW connection

ONEWIRE LIBRARY

The OneWire library provides routines for communication via the Dallas OneWire protocol, e.g. with DS18x20 digital thermometer. OneWire is a Master/Slave protocol, and all communication cabling required is a single wire. OneWire enabled devices should have open collector drivers (with single pull-up resistor) on the shared data line.

Slave devices on the OneWire bus can even get their power supply from data line. For detailed schematic see device datasheet.

Some basic characteristics of this protocol are:

- single master system,
- low cost,
- low transfer rates (up to 16 kbps),
- fairly long distances (up to 300 meters),
- small data transfer packages.

Each OneWire device has also a unique 64-bit registration number (8-bit device type, 48-bit serial number and 8-bit CRC), so multiple slaves can co-exist on the same bus.

Note: Oscillator frequency F_{osc} needs to be at least 8MHz in order to use the routines with Dallas digital thermometers.

External dependencies of OneWire Library

This variable must be defined in any project that is using OneWire Library:	Description:	Example :
<code>dim OW_Bit as sbit external</code>	OneWire line.	<code>dim OW_Bit as sbit at P2.B7</code>

Library Routines

- Ow_Reset
- Ow_Read
- Ow_Write

Ow_Reset

Prototype	<code>sub function Ow_Reset() as word</code>
Returns	<ul style="list-style-type: none"> - 0 if the device is present - 1 if the device is not present
Description	<p>Issues OneWire reset signal for DS18x20.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - None.
Requires	<p>Devices compliant with the Dallas OneWire protocol.</p> <p>Global variable <code>OW_Bit</code> must be defined before using this function.</p>
Example	<pre>' Issue Reset signal on One-Wire Bus Ow_Reset()</pre>

Ow_Read

Prototype	<code>sub function Ow_Read() as byte</code>
Returns	Data read from an external device over the OneWire bus.
Description	Reads one byte of data via the OneWire bus.
Requires	<p>Devices compliant with the Dallas OneWire protocol.</p> <p>Global variable <code>OW_Bit</code> must be defined before using this function.</p>
Example	<pre>' Read a byte from the One-Wire Bus dim read_data as byte ... read_data = Ow_Read()</pre>

Ow_Write

Prototype	<code>sub procedure Ow_Write(dim par as byte)</code>
Returns	Nothing.
Description	Writes one byte of data via the OneWire bus. Parameters : - <code>par</code> : data to be written
Requires	Devices compliant with the Dallas OneWire protocol. Global variable <code>OW_Bit</code> must be defined before using this function.
Example	<code>' Send a byte to the One-Wire Bus Ow_Write(0xCC)</code>

Library Example

This example reads the temperature using DS18x20 connected to pin P1.2. After reset, MCU obtains temperature from the sensor and prints it on the LCD. Make sure to pull-up P1.2 line and to turn off the P1 leds.

```
program OneWire

' Lcd pinout definition
dim LCD_RS as sbit at P2.0
dim LCD_EN as sbit at P2.1

dim LCD_D7 as sbit at P2.5
dim LCD_D6 as sbit at P2.4
dim LCD_D5 as sbit at P2.3
dim LCD_D4 as sbit at P2.2
' end Lcd definition

' OneWire pinout
dim OW_Bit as sbit at P1.B2
' end OneWire definition

' Set TEMP_RESOLUTION to the corresponding resolution of used DS18x20 sensor:
' 18S20: 9 (default setting can be 9,10,11,or 12)
' 18B20: 12
const TEMP_RESOLUTION as byte = 9

dim text as char[ 8]
temp as word
```

```

sub procedure Display_Temperature(dim temp2write as word)

const RES_SHIFT as byte = TEMP_RESOLUTION - 8
dim temp_whole as byte
    temp_fraction as word

text = "000.0000"
' check if temperature is negative
if (temp2write and 0x8000) then
    text[0] = "-"
    temp2write = not temp2write + 1
end if

' extract temp_whole
temp_whole = temp2write >> RES_SHIFT

' convert temp_whole to characters
if (temp_whole div 100 ) then
    text[0] = temp_whole div 100 + 48
end if

text[1] = (temp_whole div 10) mod 10 + 48      ' Extract
tens digit
text[2] = temp_whole mod 10 + 48             ' Extract
ones digit

' extract temp_fraction and convert it to unsigned int
temp_fraction = temp2write << (4-RES_SHIFT)
temp_fraction = temp_fraction and 0x000F
temp_fraction = temp_fraction * 625

' convert temp_fraction to characters

text[4] = temp_fraction div 1000 + 48        ' Extract
thousands digit
text[5] = (temp_fraction div 100) mod 10 + 48 ' Extract
hundreds digit
text[6] = (temp_fraction div 10) mod 10 + 48 ' Extract
tens digit
text[7] = temp_fraction mod 10 + 48         ' Extract
ones digit

' print temperature on LCD
Lcd_Out(2, 5, text)
end sub

```



```
main:
  Lcd_Init()           ' Initialize LCD
  Lcd_Cmd(LCD_CLEAR)  ' Clear LCD
  Lcd_Cmd(LCD_CURSOR_OFF) ' Turn cursor off
  Lcd_Out(1, 1, " Temperature: ")
  ' Print degree character, "C" for Centigrades
  Lcd_Chr(2,13,223)
  Lcd_Chr(2,14,"C")

  ' main loop
  while TRUE
    ' Perform temperature reading
    Ow_Reset()         ' Onewire reset
signal
    Ow_Write(0xCC)     ' Issue command
SKIP_ROM
    Ow_Write(0x44)     ' Issue command
CONVERT_T
    Delay_us(120)

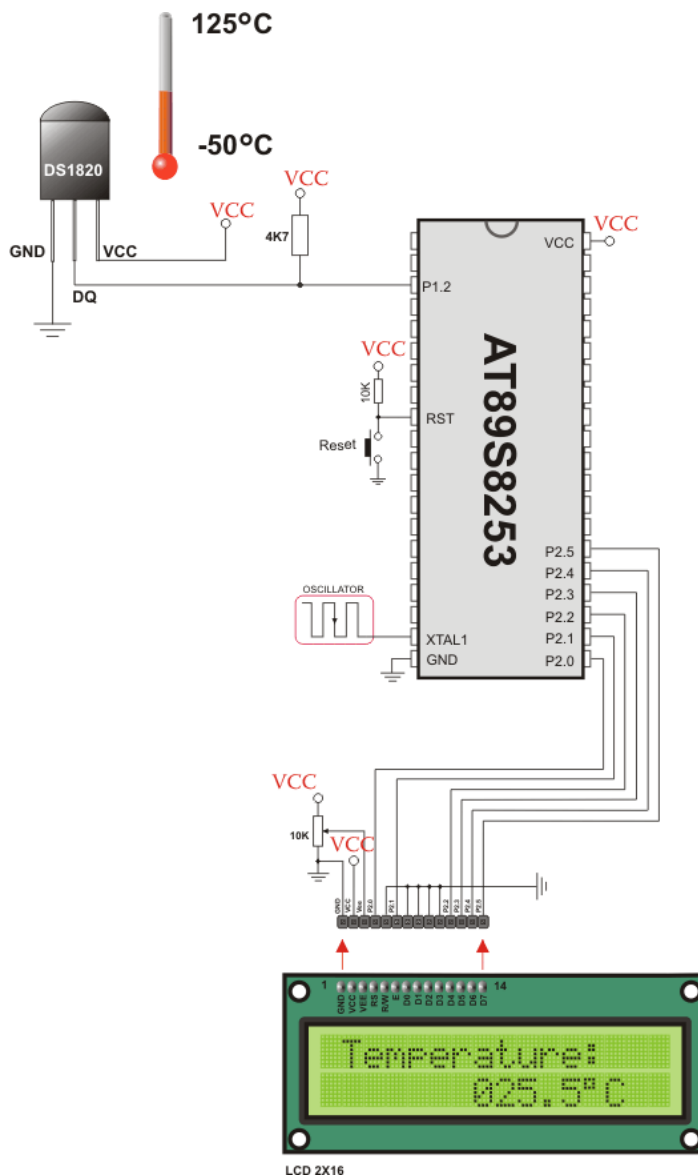
    Ow_Reset()
    Ow_Write(0xCC)     ' Issue command
SKIP_ROM
    Ow_Write(0xBE)     ' Issue command
READ_SCRATCHPAD

    temp = Ow_Read()
    temp = (Ow_Read() << 8) or temp

    ' Format and display result on Lcd
    Display_Temperature(temp)

    Delay_ms(500)
  wend
end.
```

HW Connection

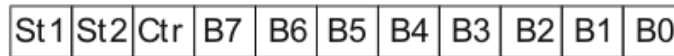


Example of DS1820 connection

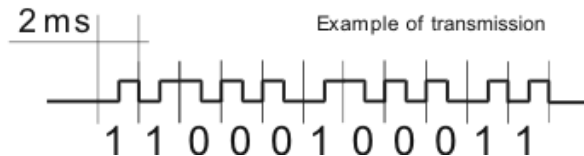
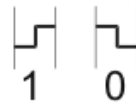
MANCHESTER CODE LIBRARY

The mikroBasic for 8051 provides a library for handling Manchester coded signals. The Manchester code is a code in which data and clock signals are combined to form a single self-synchronizing data stream; each encoded bit contains a transition at the midpoint of a bit period, the direction of transition determines whether the bit is 0 or 1; the second half is the true bit value and the first half is the complement of the true bit value (as shown in the figure below).

Manchester RF_Send_Byte format



Bi-phase coding



Notes: The Manchester receive routines are blocking calls (`Man_Receive_Init` and `Man_Synchro`). This means that MCU will wait until the task has been performed (e.g. byte is received, synchronization achieved, etc).

The following variables must be defined in all projects using Manchester Code Library:	Description:	Example :
<code>dim MANRXPIN as sbit external</code>	Receive line.	<code>dim MANRXPIN as sbit at P0.B0</code>
<code>dim MANTXPIN as sbit external</code>	Transmit line.	<code>dim MANTXPIN as sbit at P1.B1</code>

Library Routines

- Man_Receive_Init
- Man_Receive
- Man_Send_Init
- Man_Send
- Man_Synchro
- Man_Out

The following routines are for the internal use by compiler only:

- Manchester_0
- Manchester_1
- Manchester_Out

Man_Receive_Init

Prototype	<code>sub function Man_Receive_Init() as word</code>
Returns	<ul style="list-style-type: none"> - 0 - if initialization and synchronization were successful. - 1 - upon unsuccessful synchronization.
Description	<p>The function configures Receiver pin and performs synchronization procedure in order to retrieve baud rate out of the incoming signal.</p> <p>Note: In case of multiple persistent errors on reception, the user should call this routine once again or Man_Synchro routine to enable synchronization.</p>
Requires	MANRXPIN variable must be defined before using this function.
Example	<pre>' Initialize Receiver dim MANRXPIN as sbit at P0.B0 ... Man_Receive_Init()</pre>

Man_Receive

Prototype	<code>sub function Man_Receive(dim byreferror as byte) as byte</code>
Returns	A byte read from the incoming signal.
Description	<p>The function extracts one byte from incoming signal.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>error</code>: error flag. If signal format does not match the expected, the <code>error</code> flag will be set to non-zero.
Requires	To use this function, the user must prepare the MCU for receiving. See Man_Receive_Init.
Example	<pre>dim data, error as byte ... data = 0 error = 0 data = Man_Receive(&error) if (error <> 0) then ' error handling end if</pre>

Man_Send_Init

Prototype	<code>sub procedure Man_Send_Init()</code>
Returns	Nothing.
Description	The function configures Transmitter pin.
Requires	<code>MANTXPIN</code> variable must be defined before using this function.
Example	<pre>' Initialize Transmitter: dim MANTXPIN as sbit at P1.B1 ... Man_Send_Init()</pre>

Man_Send

Prototype	<code>sub procedure Man_Send(tr_data as byte)</code>
Returns	Nothing.
Description	<p>Sends one byte.</p> <p>Parameters :</p> <p>- <code>tr_data</code>: data to be sent</p> <p>Note: Baud rate used is 500 bps.</p>
Requires	To use this function, the user must prepare the MCU for sending. See <code>Man_Send_Init</code> .
Example	<pre>dim msg as byte ... Man_Send(msg)</pre>

Man_Synchro

Prototype	<code>sub function Man_Synchro() as word</code>
Returns	<ul style="list-style-type: none"> - 0 - if synchronization was not successful. - Half of the manchester bit length, given in multiples of 10us - upon successful synchronization.
Description	Measures half of the manchester bit length with 10us resolution.
Requires	To use this function, you must first prepare the MCU for receiving. See Man_Receive_Init.
Example	<pre>dim man_half_bit_len as word ... man_half_bit_len = Man_Synchro()</pre>

Man_Out

Prototype	<code>sub procedure Man_Out(BitValue as byte)</code>
Returns	Nothing.
Description	<p>Sends one byte in Manchester format.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>BitValue</code>: data to be sent
Requires	To use this function, the user must prepare the MCU for sending. See Man_Send_Init.
Example	<pre>dim BitValue as byte ... Man_Out(BitValue)</pre>

Library Example

The following code is code for the Manchester receiver, it shows how to use the Manchester Library for receiving data:

```

program Manchester_Receiver

' LCD module connections
dim LCD_RS as sbit at P2.0
dim LCD_EN as sbit at P2.1

dim LCD_D7 as sbit at P2.5
dim LCD_D6 as sbit at P2.4
dim LCD_D5 as sbit at P2.3
dim LCD_D4 as sbit at P2.2
' End LCD module connections

' Manchester module connections
dim MANRXPIN as sbit at P0.B0
dim MANTXPIN as sbit at P1.B1
' End Manchester module connections

dim error_, ErrorCount, temp as byte

main:
    ErrorCount = 0

    Lcd_Init()                ' Initialize LCD
    Lcd_Cmd(LCD_CLEAR)       ' Clear LCD display

    Man_Receive_Init()       ' Initialize Receiver

    while TRUE                ' Endless loop
        Lcd_Cmd(LCD_FIRST_ROW) ' Move cursor to the
1st row
        while TRUE           ' Wait for the "start"
byte
            temp = Man_Receive(error_) ' Attempt byte receive
            if (temp = 0x0B) then    ' "Start" byte, see
Transmitter example
                break                ' We got the start-
ing sequence
            end if
            if (error_ <> 0) then    ' Exit so we do not
loop forever
                break
            end if
            ' Inc(P1)
        wend

```

```

while ( temp <> 0x0E )
    temp = Man_Receive(error_)
    if (error_ <> 0) then
        Lcd_Chr_CP("?")
        Inc(ErrorCount)
        if (ErrorCount > 20) then
            temp = Man_Synchro()
            'Man_Receive_Init()
Initialize Receiver again
            ErrorCount = 0
        end if
    else
        if (temp <> 0x0E) then
received(see Transmitter example)
            Lcd_Chr_CP(temp)
        end if
    end if
    Delay_ms(25)
wend
wend
end.

```

The following code is code for the Manchester transmitter, it shows how to use the Manchester Library for transmitting data:

```

program Manchester_Transmitter

' Manchester module connections
dim MANRXPIN as sbit at P0.B0
dim MANTXPIN as sbit at P1.B1
' End Manchester module connections

dim index, character as byte
s1 as char[17]

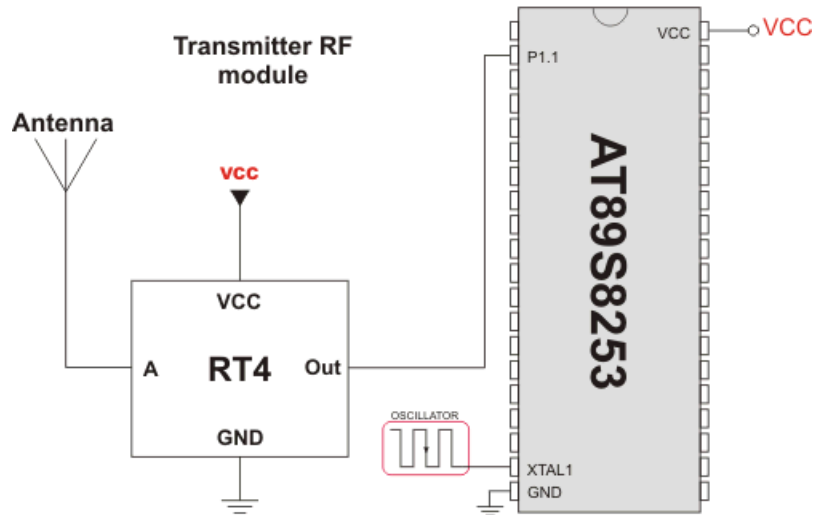
main:
s1 = "mikroElektronika"
Man_Send_Init()

while TRUE
    Man_Send(0x0B)
    Delay_ms(100)

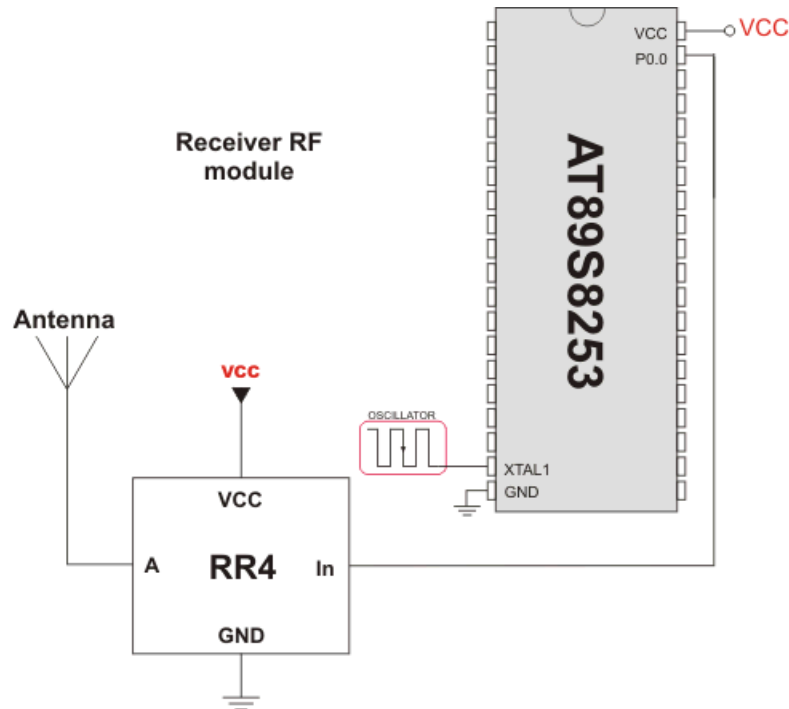
    character = s1[0]
    index = 0
    while (character <> 0)
        Man_Send(character)
        Delay_ms(90)
        Inc(index)
        character = s1[index]
    wend
    Man_Send(0x0E)
    Delay_ms(1000)
wend
end.

```


Connection Example



Simple Transmitter connection



Simple Receiver connection

PORT EXPANDER LIBRARY

The mikroBasic for 8051 provides a library for communication with the Microchip's Port Expander MCP23S17 via SPI interface. Connections of the 8051 compliant MCU and MCP23S17 is given on the schematic at the bottom of this page.

Note: Library uses the SPI module for communication. The user must initialize SPI module before using the Port Expander Library.

Note: Library does not use Port Expander interrupts.

External dependencies of Port Expander Library

The following variables must be defined in all projects using Port Expander Library:	Description:	Example :
<code>dim SPExpanderCS as sbit external</code>	Chip Select line.	<code>dim SPExpanderCS as sbit at P1.B1</code>
<code>dim SPExpanderRST as sbit external</code>	Reset line.	<code>dim SPExpanderRST as sbit at P1.B0</code>

Library Routines

- Expander_Init
- Expander_Read_Byte
- Expander_Write_Byte
- Expander_Read_PortA
- Expander_Read_PortB
- Expander_Read_PortAB
- Expander_Write_PortA
- Expander_Write_PortB
- Expander_Write_PortAB
- Expander_Set_DirectionPortA
- Expander_Set_DirectionPortB
- Expander_Set_DirectionPortAB
- Expander_Set_PullUpsPortA
- Expander_Set_PullUpsPortB
- Expander_Set_PullUpsPortAB

Expander_Init

Prototype	<code>sub procedure Expander_Init(dim ModuleAddress as byte)</code>
Returns	Nothing.
Description	<p>Initializes Port Expander using SPI communication.</p> <p>Port Expander module settings :</p> <ul style="list-style-type: none"> - hardware addressing enabled - automatic address pointer incrementing disabled (byte mode) - BANK_0 register addressing - slew rate enabled <p>Parameters :</p> <ul style="list-style-type: none"> - <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page
Requires	<p><code>SPExpanderCS</code> and <code>SPExpanderRST</code> variables must be defined before using this function.</p> <p>SPI module needs to be initialized. See <code>Spi_Init</code> and <code>Spi_Init_Advanced</code> routines.</p>
Example	<pre>' port expander pinout definition dim SPExpanderCS as sbit at P1.B1 SPExpanderRST as sbit at P1.B0 ... Spi_Init() ' initialize SPI module Expander_Init(0) ' initialize port expander</pre>

Expander_Read_Byte

Prototype	<code>sub function Expander_Read_Byte(dim ModuleAddress as byte, dim RegAddress as byte) as byte</code>
Returns	Byte read.
Description	<p>The function reads byte from Port Expander.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page - <code>RegAddress</code>: Port Expander's internal register address
Requires	Port Expander must be initialized. See <code>Expander_Init</code> .
Example	<pre>' Read a byte from Port Expander's register dim read_data as byte ... read_data = Expander_Read_Byte(0,1)</pre>

Expander_Write_Byte

Prototype	<code>sub procedure Expander_Write_Byte(dim ModuleAddress as byte, dim RegAddress as byte, dim Data_ as byte)</code>
Returns	Nothing.
Description	<p>Routine writes a byte to Port Expander.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page - <code>RegAddress</code>: Port Expander's internal register address - <code>Data_</code>: data to be written
Requires	Port Expander must be initialized. See <code>Expander_Init</code> .
Example	<pre>' Write a byte to the Port Expander's register Expander_Write_Byte(0,1,0xFF)</pre>

Expander_Read_PortA

Prototype	<code>sub function Expander_Read_PortA(dim ModuleAddress as byte) as byte</code>
Returns	Byte read.
Description	The function reads byte from Port Expander's PortA. Parameters : - <code>ModuleAddress</code> : Port Expander hardware address, see schematic at the bottom of this page
Requires	Port Expander must be initialized. See <code>Expander_Init</code> . Port Expander's PortA should be configured as input. See <code>Expander_Set_DirectionPortA</code> and <code>Expander_Set_DirectionPortAB</code> routines.
Example	<pre>' Read a byte from Port Expander's PORTA dim read_data as byte ... Expander_Set_DirectionPortA(0,0xFF) ' set expander's porta to be input ... read_data = Expander_Read_PortA(0)</pre>

Expander_Read_PortB

Prototype	<code>sub function Expander_Read_PortB(dim ModuleAddress as byte) as byte</code>
Returns	Byte read.
Description	The function reads byte from Port Expander's PortB. Parameters : - <code>ModuleAddress</code> : Port Expander hardware address, see schematic at the bottom of this page
Requires	Port Expander must be initialized. See <code>Expander_Init</code> . Port Expander's PortB should be configured as input. See <code>Expander_Set_DirectionPortB</code> and <code>Expander_Set_DirectionPortAB</code> routines.
Example	<pre>' Read a byte from Port Expander's PORTB dim read_data as byte ... Expander_Set_DirectionPortB(0,0xFF) ' set expander's portb to be input ... read_data = Expander_Read_PortB(0)</pre>

Expander_Read_PortAB

Prototype	<code>sub function Expander_Read_PortAB(dim ModuleAddress as byte) as word</code>
Returns	Word read.
Description	<p>The function reads word from Port Expander's ports. PortA readings are in the higher byte of the result. PortB readings are in the lower byte of the result.</p> <p>Parameters :</p> <p>- <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page</p>
Requires	<p>Port Expander must be initialized. See <code>Expander_Init</code>.</p> <p>Port Expander's PortA and PortB should be configured as inputs. See <code>Expander_Set_DirectionPortA</code>, <code>Expander_Set_DirectionPortB</code> and <code>Expander_Set_DirectionPortAB</code> routines.</p>
Example	<pre>' Read a byte from Port Expander's PORTA and PORTB dim read_data as word ... Expander_Set_DirectionPortAB(0,0xFFFF) ' set expander's porta and portb to be input ... read_data = Expander_Read_PortAB(0)</pre>

Expander_Write_PortA

Prototype	<code>sub procedure Expander_Write_PortA(dim ModuleAddress as byte, dim Data_ as byte)</code>
Returns	Nothing.
Description	<p>The function writes byte to Port Expander's PortA.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page - <code>Data_</code>: data to be written
Requires	<p>Port Expander must be initialized. See <code>Expander_Init</code>.</p> <p>Port Expander's PortA should be configured as output. See <code>Expander_Set_DirectionPortA</code> and <code>Expander_Set_DirectionPortAB</code> routines.</p>
Example	<pre>' Write a byte to Port Expander's PORTA ... Expander_Set_DirectionPortA(0,0x00) ' set expander's porta to be output ... Expander_Write_PortA(0, 0xAA)</pre>

Expander_Write_PortB

Prototype	<code>sub procedure Expander_Write_PortB(dim ModuleAddress as byte, dim Data_ as byte)</code>
Returns	Nothing.
Description	<p>The function writes byte to Port Expander's PortB.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page - <code>Data_</code>: data to be written
Requires	<p>Port Expander must be initialized. See <code>Expander_Init</code>.</p> <p>Port Expander's PortB should be configured as output. See <code>Expander_Set_DirectionPortB</code> and <code>Expander_Set_DirectionPortAB</code> routines.</p>
Example	<pre>' Write a byte to Port Expander's PORTB ... Expander_Set_DirectionPortB(0,0x00) ' set expander's portb to be output ... Expander_Write_PortB(0, 0x55)</pre>

Expander_Write_PortAB

Prototype	<code>sub procedure Expander_Write_PortAB(dim ModuleAddress as byte, dim Data_ as word)</code>
Returns	Nothing.
Description	<p>The function writes word to Port Expander's ports.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page - <code>Data_</code>: data to be written. Data to be written to PortA are passed in Data's higher byte. Data to be written to PortB are passed in Data's lower byte
Requires	<p>Port Expander must be initialized. See <code>Expander_Init</code>.</p> <p>Port Expander's PortA and PortB should be configured as outputs. See <code>Expander_Set_DirectionPortA</code>, <code>Expander_Set_DirectionPortB</code> and <code>Expander_Set_DirectionPortAB</code> routines.</p>
Example	<pre>' Write a byte to Port Expander's PORTA and PORTB ... Expander_Set_DirectionPortAB(0,0x0000) ' set expander's porta and portb to be output ... Expander_Write_PortAB(0, 0xAA55)</pre>

Expander_Set_DirectionPortA

Prototype	<code>sub procedure Expander_Set_DirectionPortA(dim ModuleAddress as byte, dim Data_ as byte)</code>
Returns	Nothing.
Description	<p>The function sets Port Expander's PortA direction.</p> <p>Parameters :</p> <ul style="list-style-type: none">- <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page- <code>Data_</code>: data to be written to the PortA direction register. Each bit corresponds to the appropriate pin of the PortA register. Set bit designates corresponding pin as input. Cleared bit designates corresponding pin as output.
Requires	Port Expander must be initialized. See <code>Expander_Init</code> .
Example	<pre>' Set Port Expander's PORTA to be output Expander_Set_DirectionPortA(0,0x00)</pre>

Expander_Set_DirectionPortB

Prototype	<code>sub procedure Expander_Set_DirectionPortB(dim ModuleAddress as byte, dim Data_ as byte)</code>
Returns	Nothing.
Description	<p>The function sets Port Expander's PortB direction.</p> <p>Parameters :</p> <ul style="list-style-type: none">- <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page- <code>Data_</code>: data to be written to the PortB direction register. Each bit corresponds to the appropriate pin of the PortB register. Set bit designates corresponding pin as input. Cleared bit designates corresponding pin as output.
Requires	Port Expander must be initialized. See <code>Expander_Init</code> .
Example	<pre>' Set Port Expander's PORTB to be input Expander_Set_DirectionPortB(0,0xFF)</pre>

Expander_Set_DirectionPortAB

Prototype	<code>sub procedure Expander_Set_DirectionPortAB(dim ModuleAddress as byte, dim Direction as word)</code>
Returns	Nothing.
Description	<p>The function sets Port Expander's PortA and PortB direction.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page - <code>Direction</code>: data to be written to direction registers. Data to be written to the PortA direction register are passed in <code>Direction</code>'s higher byte. Data to be written to the PortB direction register are passed in <code>Direction</code>'s lower byte. Each bit corresponds to the appropriate pin of the PortA/PortB register. Set bit designates corresponding pin as input. Cleared bit designates corresponding pin as output.
Requires	Port Expander must be initialized. See <code>Expander_Init</code> .
Example	<code>' Set Port Expander's PORTA to be output and PORTB to be input Expander_Set_DirectionPortAB(0,0x00FF)</code>

Expander_Set_PullUpsPortA

Prototype	<code>sub procedure Expander_Set_PullUpsPortA(dim ModuleAddress as byte, dim Data_ as byte)</code>
Returns	Nothing.
Description	<p>The function sets Port Expander's PortA pull up/down resistors.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page - <code>Data_</code> : data for choosing pull up/down resistors configuration. Each bit corresponds to the appropriate pin of the PortA register. Set bit enables pull-up for corresponding pin.
Requires	Port Expander must be initialized. See <code>Expander_Init</code> .
Example	<code>' Set Port Expander's PORTA pull-up resistors Expander_Set_PullUpsPortA(0, 0xFF)</code>

Expander_Set_PullUpsPortB

Prototype	<code>sub procedure Expander_Set_PullUpsPortB(dim ModuleAddress as byte, dim Data_ as byte)</code>
Returns	Nothing.
Description	<p>The function sets Port Expander's PortB pull up/down resistors.</p> <p>Parameters :</p> <ul style="list-style-type: none">- <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page- <code>Data_</code> : data for choosing pull up/down resistors configuration. Each bit corresponds to the appropriate pin of the PortB register. Set bit enables pull-up for corresponding pin.
Requires	Port Expander must be initialized. See <code>Expander_Init</code> .
Example	<pre>' Set Port Expander's PORTB pull-up resistors Expander_Set_PullUpsPortB(0, 0xFF)</pre>

Expander_Set_PullUpsPortAB

Prototype	<code>sub procedure Expander_Set_PullUpsPortAB(dim ModuleAddress as byte, dim PullUps as word)</code>
Returns	Nothing.
Description	<p>The function sets Port Expander's PortA and PortB pull up/down resistors.</p> <p>Parameters :</p> <ul style="list-style-type: none">- <code>ModuleAddress</code>: Port Expander hardware address, see schematic at the bottom of this page- <code>PullUps</code>: data for choosing pull up/down resistors configuration. PortA pull up/down resistors configuration is passed in <code>PullUps</code>'s higher byte. PortB pull up/down resistors configuration is passed in <code>PullUps</code>'s lower byte. Each bit corresponds to the appropriate pin of the PortA/PortB register. Set bit enables pull-up for corresponding pin.
Requires	Port Expander must be initialized. See <code>Expander_Init</code> .
Example	<pre>' Set Port Expander's PORTA and PORTB pull-up resistors Expander_Set_PullUpsPortAB(0, 0xFFFF)</pre>

Library Example

The example demonstrates how to communicate with Port Expander MCP23S17.

Note that Port Expander pins A2 A1 A0 are connected to GND so Port Expander Hardware Address is 0.

```
program PortExpander

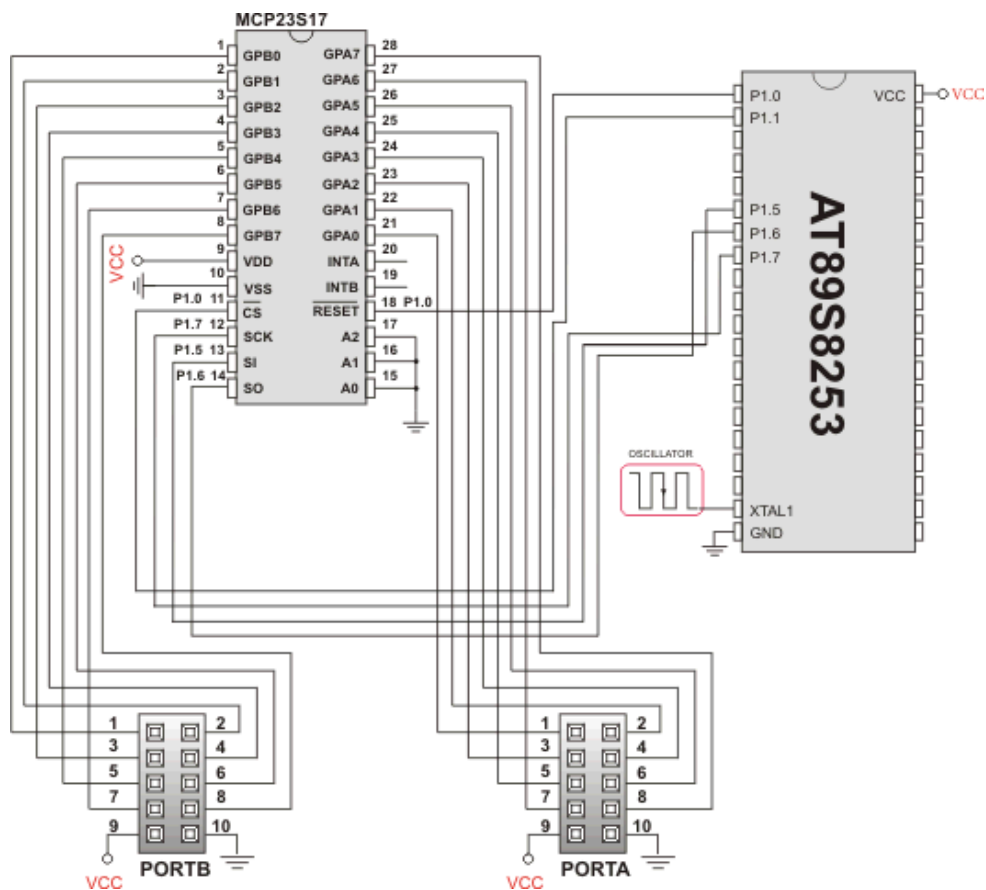
dim i as byte

' Port Expander module connections
dim SPExpanderRST as sbit at P1.B0
dim SPExpanderCS as sbit at P1.B1
' End Port Expander module connections

main:
    i = 0
    Spi_Init()                ' Initialize SPI module
    Expander_Init(0)         ' Initialize Port Expander
    Expander_Set_DirectionPortA(0, 0x00) ' Set Expander"s PORTA
    to be output
    Expander_Set_DirectionPortB(0,0xFF) ' Set Expander"s PORTB
    to be input
    Expander_Set_PullUpsPortB(0,0xFF)   ' Set pull-ups to all
    of the Expander"s PORTB pins

    while TRUE                ' Endless loop
        Expander_Write_PortA(0, i)     ' Write i to expander"s
    PORTA
        Inc(i)
        P0 = Expander_Read_PortB(0)    ' Read expander"s PORTB
    and write it to PORT0
        Delay_ms(100)
    wend
end.
```

HW Connection



Port Expander HW connection

PS/2 LIBRARY

The mikroBasic for 8051 provides a library for communication with the common PS/2 keyboard.

Note: The library does not utilize interrupts for data retrieval, and requires the oscillator clock to be at least 6MHz.

Note: The pins to which a PS/2 keyboard is attached should be connected to the pull-up resistors.

Note: Although PS/2 is a two-way communication bus, this library does not provide MCU-to-keyboard communication; e.g. pressing the Caps Lock key will not turn on the Caps Lock LED.

External dependencies of PS/2 Library

The following variables must be defined in all projects using PS/2 Library:	Description:	Example :
<code>dim PS2_DATA as sbit external</code>	PS/2 Data line.	<code>dim PS2_DATA as sbit at P0.B0</code>
<code>dim PS2_CLOCK as sbit external</code>	PS/2 Clock line.	<code>dim PS2_CLOCK as sbit at P0.B1</code>

Library Routines

- Ps2_Config
- Ps2_Key_Read

Ps2_Config

Prototype	<code>sub procedure Ps2_Config()</code>
Returns	Nothing.
Description	Initializes the MCU for work with the PS/2 keyboard.
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>PS2_DATA</code>: Data signal pin - <code>PS2_CLOCK</code>: Clock signal pin <p>must be defined before using this function.</p>
Example	<pre>' PS2 pinout definition dim PS2_DATA as sbit at P0.B0 PS2_CLOCK as sbit at P0.B1 ... Ps2_Config() ' Init PS/2 Keyboard</pre>

Ps2_Key_Read

Prototype	<code>sub function Ps2_Key_Read(dim byref value as byte, dim byref special as byte, dim byref pressed as byte) as byte</code>
Returns	<ul style="list-style-type: none"> - 1 if reading of a key from the keyboard was successful - 0 if no key was pressed
Description	<p>The function retrieves information on key pressed.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>value</code>: holds the value of the key pressed. For characters, numerals, punctuation marks, and space <code>value</code> will store the appropriate ASCII code. Routine “recognizes” the function of Shift and Caps Lock, and behaves appropriately. For special function keys see Special Function Keys Table. - <code>special</code>: is a flag for special function keys (F1, Enter, Esc, etc). If key pressed is one of these, <code>special</code> will be set to 1, otherwise 0. - <code>pressed</code>: is set to 1 if the key is pressed, and 0 if it is released.
Requires	PS/2 keyboard needs to be initialized. See Ps2_Config routine.
Example	<pre> dim value, special, pressed as byte ... do { if (Ps2_Key_Read(value, special, pressed)) then if ((value = 13) and (special = 1)) then break end if end if loop until (0=1) </pre>

Special Function Keys

Key	Value returned
F1	1
F2	2
F3	3
F4	4
F5	5
F6	6
F7	7
F8	8
F9	9
F10	10
F11	11
F12	12
Enter	13
Page Up	14
Page Down	15
Backspace	16
Insert	17
Delete	18
Windows	19
Ctrl	20
Shift	21
Alt	22
Print Screen	23
Pause	24
Caps Lock	25
End	26
Home	27

Scroll Lock	28
Num Lock	29
Left Arrow	30
Right Arrow	31
Up Arrow	32
Down Arrow	33
Escape	34
Tab	35

Library Example

This simple example reads values of the pressed keys on the PS/2 keyboard and sends them via UART.

```

program PS2

dim keydata, special, down as byte

' PS2 module connections
dim PS2_DATA as sbit at P0.B0
    PS2_CLOCK as sbit at P0.B1
' End PS2 module connections

main:
    Uart_Init(4800)           ' Initialize UART module at 4800 bps
    Ps2_Config()            ' Initialize PS/2 Keyboard
    Delay_ms(100)           ' Wait for keyboard to finish
    keydata = 0 special = 0 down = 0

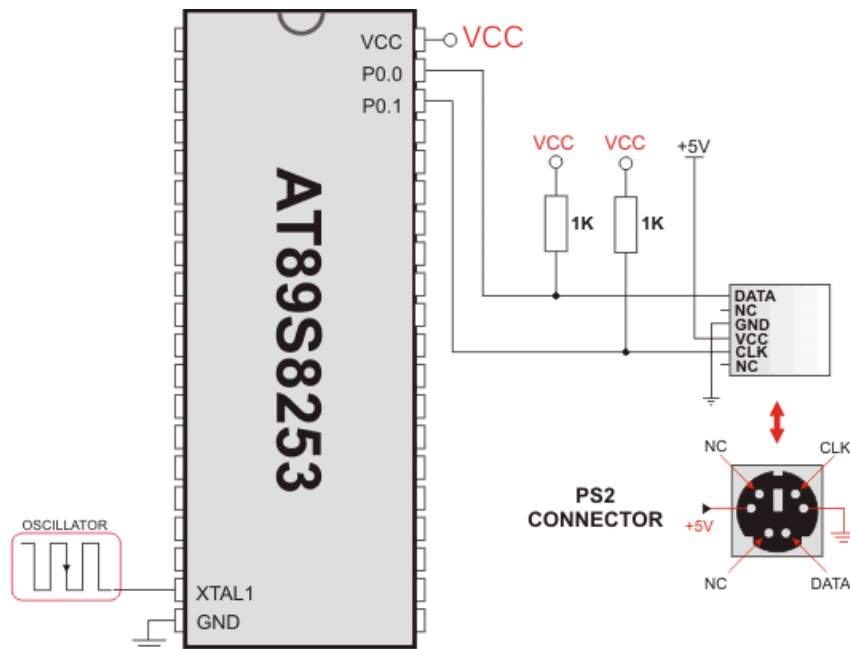
while true

    if (Ps2_Key_Read(keydata, special, down)) <> 0 then      ' If data
was read from PS/2
        if ((down = 1) and (keydata = 16)) <> 0 then          '
Backspace read
            Uart_Write(0x08)                                   ' Send
Backspace to usart terminal

        else if ((down = 1) and (keydata = 13)) <> 0 then    ' Enter
read
            Uart_Write(10)  ' Send carriage return to usart terminal
            Uart_Write(13)  ' Uncomment this line if usart terminal
also expects line feed
            '
            for new line transition
                else if ((down = 1) and (special = 0) and (keydata <> 0))
then      ' Common key read
                    Uart_Write(keydata)
                Send key to usart terminal
                    end if
            end if
        end if
        Delay_ms(10)           ' Debounce period
    wend
end.

```

HW Connection



Example of PS2 keyboard connection

RS-485 LIBRARY

RS-485 is a multipoint communication which allows multiple devices to be connected to a single bus. The mikroBasic for 8051 provides a set of library routines for comfortable work with RS485 system using Master/Slave architecture. Master and Slave devices interchange packets of information. Each of these packets contains synchronization bytes, CRC byte, address byte and the data. Each Slave has unique address and receives only packets addressed to it. The Slave can never initiate communication.

It is the user's responsibility to ensure that only one device transmits via 485 bus at a time.

The RS-485 routines require the UART module. Pins of UART need to be attached to RS-485 interface transceiver, such as LTC485 or similar (see schematic at the bottom of this page).

Library constants:

- START byte value = 150
- STOP byte value = 169
- Address 50 is the broadcast address for all Slaves (packets containing address 50 will be received by all Slaves except the Slaves with addresses 150 and 169).

External dependencies of RS-485 Library

The following variable must be defined in all projects using RS-485 Library:	Description:	Example :
<pre>dim rs485_transceive as sbit external</pre>	Control RS-485 Transmit/Receive operation mode	<pre>dim rs485_transceive as sbit at P3.B2</pre>

Library Routines

- RS485master_Init
- RS485master_Receive
- RS485master_Send
- RS485slave_Init
- RS485slave_Receive
- RS485slave_Send

RS485master_Init

Prototype	<code>sub procedure Rs485master_Init()</code>
Returns	Nothing.
Description	Initializes MCU as a Master for RS-485 communication.
Requires	<p><code>rs485_transceive</code> variable must be defined before using this function. This pin is connected to RE/DE input of RS-485 transceiver(see schematic at the bottom of this page). RE/DE signal controls RS-485 transceiver operation mode. Valid values: 1 (for transmitting) and 0 (for receiving)</p> <p>UART HW module needs to be initialized. See <code>Uart_Init</code>.</p>
Example	<pre>' rs485 module pinout dim rs485_transceive as sbit at P3.B2 ' transmit/receive control set to port3.bit2 ... Uart_Init(9600) ' initialize usart module Rs485master_Init() ' intialize mcu as a Master for RS-485 communication</pre>

RS485master_Receive

Prototype	<code>sub procedure Rs485master_Receive(dim byref data_buffer as array[20] of byte)</code>
Returns	Nothing.
Description	<p>Receives messages from Slaves. Messages are multi-byte, so this routine must be called for each byte received.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>data_buffer</code>: 7 byte buffer for storing received data, in the following manner: - <code>data[0..2]</code> : message content - <code>data[3]</code> : number of message bytes received, 1–3 - <code>data[4]</code> : is set to 255 when message is received - <code>data[5]</code> : is set to 255 if error has occurred - <code>data[6]</code> : address of the Slave which sent the message <p>The function automatically adjusts <code>data[4]</code> and <code>data[5]</code> upon every received message. These flags need to be cleared by software.</p>
Requires	MCU must be initialized as a Master for RS-485 communication. See <code>RS485master_Init</code> .
Example	<pre>dim msg as array[20] of byte ... RS485master_Receive(msg)</pre>

RS485master_Send

Prototype	<code>sub procedure Rs485master_Send(dim byref data_buffer as array[20] of byte, dim datalen as byte, dim slave_address as byte)</code>
Returns	Nothing.
Description	<p>Sends message to Slave(s). Message format can be found at the bottom of this page.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>data_buffer</code>: data to be sent - <code>datalen</code>: number of bytes for transmission. Valid values: 0...3. - <code>slave_address</code>: Slave(s) address
Requires	<p>MCU must be initialized as a Master for RS-485 communication. See <code>RS485master_Init</code>.</p> <p>It is the user's responsibility to ensure (by protocol) that only one device sends data via 485 bus at a time.</p>
Example	<pre>dim msg as array[20] of byte ... ' send 3 bytes of data to slave with address 0x12 RS485master_Send(msg, 3, 0x12)</pre>

RS485slave_Init

Prototype	<code>sub procedure Rs485slave_Init(dim slave_address as byte)</code>
Returns	Nothing.
Description	<p>Initializes MCU as a Slave for RS-485 communication.</p> <p>Parameters :</p> <p>- <code>slave_address</code>: Slave address</p>
Requires	<p><code>rs485_transceive</code> variable must be defined before using this function. This pin is connected to RE/DE input of RS-485 transceiver(see schematic at the bottom of this page). RE/DE signal controls RS-485 transceiver operation mode. Valid values: 1 (for transmitting) and 0 (for receiving)</p> <p>UART HW module needs to be initialized. See <code>Uart_Init</code>.</p>
Example	<pre>' rs485 module pinout dim rs485_transceive as sbit at P3.B2 ' transmit/receive control set to port3.bit2 ... Uart_Init(9600) ' initialize usart module Rs485slave_Init(160) ' intialize mcu as a Slave for RS-485 communication with address 160</pre>

RS485slave_Receive

Prototype	<code>sub procedure RS485slave_Receive(dim byref data_buffer as array[20] of byte)</code>
Returns	Nothing.
Description	<p>Receives messages from Master. If Slave address and Message address field don't match then the message will be discarded. Messages are multi-byte, so this routine must be called for each byte received.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>data_buffer</code>: 6 byte buffer for storing received data, in the following manner: - <code>data[0..2]</code> : message content - <code>data[3]</code> : number of message bytes received, 1–3 - <code>data[4]</code> : is set to 255 when message is received - <code>data[5]</code> : is set to 255 if error has occurred <p>The function automatically adjusts <code>data[4]</code> and <code>data[5]</code> upon every received message. These flags need to be cleared by software.</p>
Requires	MCU must be initialized as a Slave for RS-485 communication. See <code>RS485slave_Init</code> .
Example	<pre>dim msg as array[5] of byte ... RS485slave_Read(msg)</pre>

RS485slave_Send

Prototype	<code>sub procedure Rs485slave_Send(dim byref data_buffer as array[20] of byte, dim datalen as byte)</code>
Returns	Nothing.
Description	Sends message to Master. Message format can be found at the bottom of this page. Parameters : - <code>data_buffer</code> : data to be sent - <code>datalen</code> : number of bytes for transmission. Valid values: 0...3.
Requires	MCU must be initialized as a Slave for RS-485 communication. See <code>RS485slave_Init</code> . It is the user's responsibility to ensure (by protocol) that only one device sends data via 485 bus at a time.
Example	<pre>dim msg as array[8] of byte ... ' send 2 bytes of data to the master RS485slave_Send(msg, 2)</pre>

Library Example

This is a simple demonstration of RS485 Library routines usage.

Master sends message to Slave with address 160 and waits for a response. The Slave accepts data, increments it and sends it back to the Master. Master then does the same and sends incremented data back to Slave, etc.

Master displays received data on P0, while error on receive (0xAA) and number of consecutive unsuccessful retries are displayed on P1. Slave displays received data on P0, while error on receive (0xAA) is displayed on P1. Hardware configurations in this example are made for the Easy8051B board and AT89S8253.

RS485 Master code:

```
program RS485_Master
dim dat as byte[ 20]           ' Buffer for receiving/sending messages
  counter, j as byte
  count as longint

' RS485 module connections
dim rs485_transceive as sbit at P3.B2 ' Transmit/Receive control set to P3.2
' End RS485 module connections
```

```

' Interrupt routine
sub procedure UartRxHandler() ORG 0x23
    EA = 0                                ' Clear global interrupt enable flag
    if ( RI <> 0 ) then                  ' Test UART receive interrupt flag
        Rs485master_Receive(dat)        ' UART receive interrupt detected,
                                        ' receive data using RS485 communication
        RI = 0                          ' Clear UART interrupt flag
    end if
    EA = 1                                ' Set global interrupt enable flag
end sub

main:
    count = 0
    P0 = 0                                ' Clear ports
    P1 = 0

    Uart_Init(9600)                       ' Initialize UART module at 9600 bps
    Delay_ms(100)

    Rs485master_Init()                   ' Intialize MCU as RS485 master
    dat[ 0 ] = 0x55                       ' Fill buffer
    dat[ 1 ] = 0x00
    dat[ 2 ] = 0x00
    dat[ 4 ] = 0                          ' Ensure that message received flag is 0
    dat[ 5 ] = 0                          ' Ensure that error flag
is 0
    dat[ 6 ] = 0
    Rs485master_Send(dat,1,160)           ' Send message to slave with
address 160                               ' message data is stored
in dat                                     ' message is 1 byte long

    ES = 1                                ' Enable UART interrupt
    RI = 0                                ' Clear UART RX interrupt
flag
    EA = 1                                ' Enable interrupts

    while TRUE                            ' Endless loop
message receiving                          ' Upon completed valid
        Inc(count)                        ' data[ 4 ] is set to 255
er                                         ' Increment loop pass count-

        if (dat[ 5 ] <> 0) then          ' If error detected, sig-
signal it by                               ' setting PORT1 to 0xAA
            P1 = 0xAA
        end if

```

```

if (dat[ 4] <> 0) then                                ' If message received successfully

    count = 0                                         ' Reset loop pass counter
    dat[ 4] = 0                                       ' Clear message received flag
    j = dat[ 3]                                       ' Read number of message received bytes
    for counter = 1 to j
        P0 = dat[ counter-1]                          ' Show received data on PORT0
    next counter

    Inc(dat[ 0] )                                     ' Increment first received byte dat[ 0]

    Delay_ms(10)
    Rs485master_Send(dat,1,160)                       ' And send it back to Slave
end if

    if ( count > 10000 ) then                         ' If loop is passed 100000
times with
        ' no message received
        Inc(P1)                                       ' Signal receive message failure on PORT1
        count = 0                                     ' Reset loop pass counter
        Rs485master_Send(dat,1,160)                 ' Retry send message
        if (P1 > 10) then                             ' If sending failed 10 times
            P1 = 0                                     ' Clear PORT1
            Rs485master_Send(dat,1,50)              ' Send message on broadcast
address
        end if
    end if
wend
end.

```

RS485 Slave code:

```

program RS485_Slave

dim dat as byte[ 20]                                ' Buffer for receiving/sending
messages
    counter, j as byte

' RS485 module connections
dim rs485_transceive as sbit at P3.B2              ' Transmit/Receive control
set to P3.2
' End RS485 module connections

' Interrupt routine
sub procedure UartRxHandler() ORG 0x23
    EA = 0                                           ' Clear global interrupt enable flag
    if( RI <> 0) then                                 ' Test UART receive interrupt flag
        Rs485slave_Receive(dat)                     ' UART receive interrupt detected,

```

```

        RI = 0                ' receive data using RS485 communication
    end if                  ' Clear UART interrupt flag
    EA = 1                  ' Set global interrupt enable flag
end sub

main:
    P0 = 0                  ' Clear ports
    P1 = 0

    Uart_Init(9600)        ' Initialize UART module at 9600 bps
    Delay_ms(100)
    Rs485slave_Init(160)   ' Intialize MCU as slave, address 160
    dat[ 4] = 0            ' ensure that message received flag is 0
    dat[ 5] = 0            ' ensure that error flag is 0

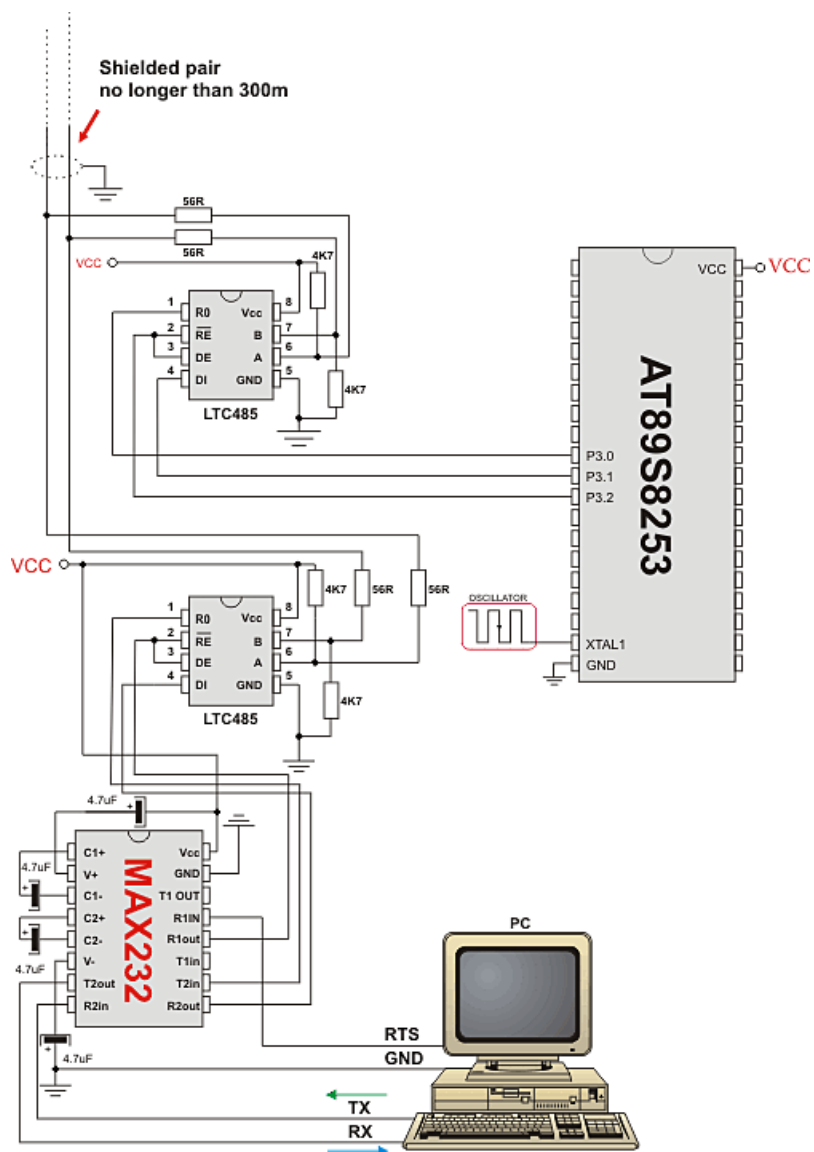
    ES = 1                  ' Enable UART interrupt
    RI = 0                  ' Clear UART RX interrupt flag
    EA = 1                  ' Enable interrupts

    while TRUE              ' Endless loop
                                ' Upon completed valid
message receiving
                                ' data[ 4] is set to 255
                                ' If error detected, sig-
                                '
                                ' setting PORT1 to 0xAA
        if (dat[ 5] <> 0) then
            P1 = 0xAA
        end if

        if (dat[ 4] <> 0) then ' If message received suc-
            cessfully
                dat[ 4] = 0    ' Clear message received
            flag
                j = dat[ 3]    ' Read number of message
            received bytes
                for counter = 1 to j
                    P0 = dat[counter-1] ' Show received data on
            PORT0
                next counter
                dat[ 0] = dat[ 0] + 1 ' Increment received dat[ 0]
                Delay_ms(10)
                Rs485slave_Send(dat,1) ' And send back to Master
            end if
        wend
    end.

```

HW Connection



Example of interfacing PC to 8051 MCU via RS485 bus with LTC485 as RS-485 transceiver

Message format and CRC calculations

Q: How is CRC checksum calculated on RS485 master side?

```

START_BYTE = 0x96; ' 10010110
STOP_BYTE  = 0xA9; ' 10101001

```

```

PACKAGE:
-----

```

```

START_BYTE 0x96
ADDRESS
DATALEN
[ DATA1]           ' if exists
[ DATA2]           ' if exists
[ DATA3]           ' if exists
CRC
STOP_BYTE  0xA9

```

```

DATALEN bits
-----

```

```

bit7 = 1  MASTER SENDS
        0  SLAVE  SENDS
bit6 = 1  ADDRESS WAS XORed with 1, IT WAS EQUAL TO START_BYTE or
STOP_BYTE
        0  ADDRESS UNCHANGED
bit5 = 0  FIXED
bit4 = 1  DATA3 (if exists) WAS XORed with 1, IT WAS EQUAL TO
START_BYTE or STOP_BYTE
        0  DATA3 (if exists) UNCHANGED
bit3 = 1  DATA2 (if exists) WAS XORed with 1, IT WAS EQUAL TO
START_BYTE or STOP_BYTE
        0  DATA2 (if exists) UNCHANGED
bit2 = 1  DATA1 (if exists) WAS XORed with 1, IT WAS EQUAL TO
START_BYTE or STOP_BYTE
        0  DATA1 (if exists) UNCHANGED
bit1bit0 = 0 to 3 NUMBER OF DATA BYTES SEND

```

```

CRC generation :
-----

```

```

crc_send = datalen ^ address;
crc_send ^= data[ 0];      ' if exists
crc_send ^= data[ 1];      ' if exists
crc_send ^= data[ 2];      ' if exists
crc_send = ~crc_send;
if ((crc_send == START_BYTE) || (crc_send == STOP_BYTE))
    crc_send++;

```

```

NOTE:  DATALEN<4..0>  can not take the START_BYTE<4..0>  or
STOP_BYTE<4..0>  values.

```

SOFTWARE I²C LIBRARY

The mikroBasic for 8051 provides routines for implementing Software I₂C communication. These routines are hardware independent and can be used with any MCU. The Software I₂C library enables you to use MCU as Master in I₂C communication. Multi-master mode is not supported.

Note: This library implements time-based activities, so interrupts need to be disabled when using Software I₂C.

Note: All I₂C Library functions are blocking-call functions (they are waiting for I₂C clock line to become logical one).

Note: The pins used for I₂C communication should be connected to the pull-up resistors. Turning off the LEDs connected to these pins may also be required.

External dependencies of Soft_I2C Library

The following variables must be defined in all projects using Soft_I2C Library:	Description:	Example :
<code>dim Soft_I2C_Scl as sbit external</code>	Soft I ₂ C Clock line.	<code>dim Soft_I2C_Scl as sbit at P1.B3</code>
<code>dim Soft_I2C_Sda as sbit external</code>	Soft I ₂ C Data line.	<code>dim Soft_I2C_Sda as sbit at P1.B4</code>

Library Routines

- Soft_I2C_Init
- Soft_I2C_Start
- Soft_I2C_Read
- Soft_I2C_Write
- Soft_I2C_Stop

Soft_I2C_Init

Prototype	<code>sub procedure Soft_I2C_Init()</code>
Returns	Nothing.
Description	Configures the software I ₂ C module.
Requires	<code>Soft_I2C_Scl</code> and <code>Soft_I2C_Sda</code> variables must be defined before using this function.
Example	<pre>' soft_i2c pinout definition dim Soft_I2C_Scl as sbit at P1.B3 Soft_I2C_Sda as sbit at P1.B4 ... Soft_I2C_Init()</pre>

Soft_I2C_Start

Prototype	<code>sub procedure Soft_I2C_Start()</code>
Returns	Nothing.
Description	Determines if the I ₂ C bus is free and issues START signal.
Requires	Software I ₂ C must be configured before using this function. See <code>Soft_I2C_Init</code> routine.
Example	<pre>' Issue START signal Soft_I2C_Start()</pre>

Soft_I2C_Read

Prototype	<code>sub function Soft_I2C_Read(dim ack as word) as byte</code>
Returns	One byte from the Slave.
Description	<p>Reads one byte from the slave.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>ack</code>: acknowledge signal parameter. If the <code>ack==0</code> not acknowledge signal will be sent after reading, otherwise the acknowledge signal will be sent.
Requires	<p>Soft I₂C must be configured before using this function. See <code>Soft_I2C_Init</code> routine.</p> <p>Also, START signal needs to be issued in order to use this function. See <code>Soft_I2C_Start</code> routine.</p>
Example	<pre>dim take as word ... ' Read data and send the not_acknowledge signal take = Soft_I2C_Read(0)</pre>

Soft_I2C_Write

Prototype	<code>sub function Soft_I2C_Write(dim _Data as byte) as byte</code>
Returns	<ul style="list-style-type: none"> - 0 if there were no errors. - 1 if write collision was detected on the I₂C bus.
Description	<p>Sends data byte via the I₂C bus.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>_Data</code>: data to be sent
Requires	<p>Soft I₂C must be configured before using this function. See <code>Soft_I2C_Init</code> routine.</p> <p>Also, START signal needs to be issued in order to use this function. See <code>Soft_I2C_Start</code> routine.</p>
Example	<pre>dim _data, error as byte ... error = Soft_I2C_Write(data) error = Soft_I2C_Write(0xA3)</pre>

Soft_I2C_Stop

Prototype	<code>sub procedure Soft_I2C_Stop()</code>
Returns	Nothing.
Description	Issues STOP signal.
Requires	Soft I ₂ C must be configured before using this function. See <code>Soft_I2C_Init</code> routine
Example	<pre>' Issue STOP signal Soft_I2C_Stop()</pre>

Library Example

The example demonstrates Software I₂C Library routines usage. The 8051 MCU is connected (SCL, SDA pins) to PCF8583 RTC (real-time clock). Program reads date and time are read from the RTC and prints it on LCD.

```
program Soft_I2C

dim seconds, minutes, hours, day, month_, year as byte      ' Global
date/time variables

' Software I2C connections
dim Soft_I2C_Scl as sbit at P1.B3
dim Soft_I2C_Sda as sbit at P1.B4
' End Software I2C connections

' LCD module connections
dim LCD_RS as sbit at P2.0
dim LCD_EN as sbit at P2.1

dim LCD_D7 as sbit at P2.5
dim LCD_D6 as sbit at P2.4
dim LCD_D5 as sbit at P2.3
dim LCD_D4 as sbit at P2.2
' End LCD module connections

'----- Reads time and date information from RTC
(PCF8583)
sub procedure Read_Time()

    Soft_I2C_Start()                ' Issue start signal
    Soft_I2C_Write(0xA0)           ' Address PCF8583, see PCF8583
datasheet
    Soft_I2C_Write(2)              ' Start from address 2
    Soft_I2C_Start()              ' Issue repeated start signal
    Soft_I2C_Write(0xA1)          ' Address PCF8583 for reading
R/W=1
    seconds = Soft_I2C_Read(1)     ' Read seconds byte
    minutes = Soft_I2C_Read(1)    ' Read minutes byte
    hours = Soft_I2C_Read(1)      ' Read hours byte
    day = Soft_I2C_Read(1)        ' Read year/day byte
    month_ = Soft_I2C_Read(0)     ' Read weekday/month byte
    Soft_I2C_Stop()              ' Issue stop signal
end sub
```

```
'----- Formats date and time
sub procedure Transform_Time()
    seconds = ((seconds and 0xF0) >> 4)*10 + (seconds and 0x0F) '
Transform seconds
    minutes = ((minutes and 0xF0) >> 4)*10 + (minutes and 0x0F) '
Transform months
    hours = ((hours and 0xF0) >> 4)*10 + (hours and 0x0F) '
Transform hours
    year = (day and 0xC0) >> 6 '
Transform year
    day = ((day and 0x30) >> 4)*10 + (day and 0x0F) '
Transform day
    month_ = ((month_ and 0x10) >> 4)*10 + (month_ and 0x0F) '
Transform month
end sub

'----- Output values to LCD
sub procedure Display_Time()

    Lcd_Chr(1, 6, (day / 10) + 48) ' Print tens digit of day
variable
    Lcd_Chr(1, 7, (day mod 10) + 48) ' Print oness digit of day
variable
    Lcd_Chr(1, 9, (month_ / 10) + 48)
    Lcd_Chr(1,10, (month_ mod 10) + 48)
    Lcd_Chr(1,15, year + 56) ' Print year vaiable + 8
(start from year 2008)

    Lcd_Chr(2, 6, (hours / 10) + 48)
    Lcd_Chr(2, 7, (hours mod 10) + 48)
    Lcd_Chr(2, 9, (minutes / 10) + 48)
    Lcd_Chr(2,10, (minutes mod 10) + 48)
    Lcd_Chr(2,12, (seconds / 10) + 48)
    Lcd_Chr(2,13, (seconds mod 10) + 48)
end sub

'----- Performs project-wide init
sub procedure Init_Main()

    Soft_I2C_Init() ' Initialize Soft I2C communication

    Lcd_Init() ' Initialize LCD
    Lcd_Cmd(LCD_CLEAR) ' Clear LCD display
    Lcd_Cmd(LCD_CURSOR_OFF) ' Turn cursor off
```

```
LCD_Out(1,1,"Date:")      ' Prepare and output static text on LCD
  LCD_Chr(1,8,":")
  LCD_Chr(1,11,":")
  LCD_Out(2,1,"Time:")
  LCD_Chr(2,8,":")
  LCD_Chr(2,11,":")
  LCD_Out(1,12,"200")
end sub

'----- Main sub procedure
main:
  Init_Main()            ' Perform initialization

  while TRUE            ' Endless loop
    Read_Time()         ' Read time from RTC(PCF8583)
    Transform_Time()    ' Format date and time
    Display_Time()      ' Prepare and display on LCD
  wend
end.
```

SOFTWARE SPI LIBRARY

The mikroBasic for 8051 provides routines for implementing Software SPI communication. These routines are hardware independent and can be used with any MCU. The Software SPI Library provides easy communication with other devices via SPI: A/D converters, D/A converters, MAX7219, LTC1290, etc.

Library configuration:

- SPI to Master mode
- Clock value = 20 kHz.
- Data sampled at the middle of interval.
- Clock idle state low.
- Data sampled at the middle of interval.
- Data transmitted at low to high edge.

Note: The Software SPI library implements time-based activities, so interrupts need to be disabled when using it.

The following variables must be defined in all projects using Software SPI Library:	Description:	Example :
<code>dim SoftSpi_SDI as sbit external</code>	Data In line.	<code>dim SoftSpi_SDI as sbit at P0.B4</code>
<code>dim SoftSpi_SDO as sbit external</code>	Data Out line.	<code>dim SoftSpi_SDO as sbit at P0.B5</code>
<code>dim SoftSpi_CLK as sbit external</code>	Clock line.	<code>dim SoftSpi_CLK as sbit at P0.B3</code>

Library Routines

- Soft_Spi_Init
- Soft_Spi_Read
- Soft_Spi_Write

Soft_Spi_Init

Prototype	<code>sub procedure Soft_SPI_Init()</code>
Returns	Nothing.
Description	Configures and initializes the software SPI module.
Requires	SoftSpi_CLK, SoftSpi_SDI and SoftSpi_SDO variables must be defined before using this function.
Example	<pre>' soft_spi pinout definition dim SoftSpi_SDI as sbit at P0.B4 SoftSpi_SDO as sbit at P0.B5 SoftSpi_CLK as sbit at P0.B3 ... Soft_SPI_Init() ' Init Soft_SPI</pre>

Soft_Spi_Read

Prototype	<code>sub function Soft_Spi_Read(dim sdata as byte) as byte</code>
Returns	Byte received via the SPI bus.
Description	<p>This routine performs 3 operations simultaneously. It provides clock for the Software SPI bus, reads a byte and sends a byte.</p> <p>Parameters :</p> <p>- <code>sdata</code>: data to be sent.</p>
Requires	Soft SPI must be initialized before using this function. See <code>Soft_Spi_Init</code> routine.
Example	<pre>dim data_read as byte data_send as byte ... ' Read a byte and assign it to data_read variable ' (data_send byte will be sent via SPI during the Read operation) data_read = Soft_Spi_Read(data_send)</pre>

Soft_Spi_Write

Prototype	<code>sub procedure Soft_Spi_Write(dim sdata as byte)</code>
Returns	Nothing.
Description	This routine sends one byte via the Software SPI bus. Parameters : - <code>sdata</code> : data to be sent.
Requires	Soft SPI must be initialized before using this function. See <code>Soft_Spi_Init</code> routine.
Example	<pre>' Write a byte to the Soft SPI bus Soft_Spi_Write(0xAA)</pre>

Library Example

This code demonstrates using library routines for `Soft_SPI` communication. Also, this example demonstrates working with Microchip's MCP4921 12-bit D/A converter.

```
program Soft_SPI

' DAC module connections
dim Chip_Select as sbit at P3.B4
    SoftSpi_CLK as sbit at P1.B7
    SoftSpi_SDI as sbit at P1.B6
    SoftSpi_SDO as sbit at P1.B5
' End DAC module connections

dim value as word

sub procedure InitMain()
    P0 = 255                ' Set PORT0 as input
    Soft_SPI_Init()        ' Initialize Soft_SPI
end sub

' DAC increments (0..4095) --> output voltage (0..Vref)
sub procedure DAC_Output(dim valueDAC as word)
dim temp as byte

    Chip_Select = 0        ' Select DAC chip

    ' Send High Byte
    temp = word(valueDAC >> 8)    ' Store valueDAC[ 11..8] to temp[ 3..0]
    temp = (temp and 0x0F) or 0x30    ' Define DAC setting, see MCP4921
datasheet
    Soft_SPI_Write(temp)        ' Send high byte via Soft SPI
```



```

' Send Low Byte
temp = valueDAC           ' Store valueDAC[ 7..0] to temp[ 7..0]
Soft_SPI_Write(temp)     ' Send low byte via Soft SPI

Chip_Select = 1           ' Deselect DAC chip
end sub

main:
InitMain()               ' Perform main initialization

value = 2048              ' When program starts, DAC gives
                          ' the output in the mid-range
while TRUE                ' Endless loop
  if ((P0_0 = 0) and (value < 4095)) then ' If P0.0 is con-
    nected to GND
    Inc(value)             ' increment value
  else
    if (( P0_1 = 0 ) and (value > 0)) then ' If P0.1 is con-
    nected to GND
    Dec(value)             ' decrement value
    end if
  end if

  DAC_Output(value)       ' Perform output
  Delay_ms(10)            ' Slow down key repeat pace
wend
end.

```

SOFTWARE UART LIBRARY

The mikroBasic for 8051 provides routines for implementing Software UART communication. These routines are hardware independent and can be used with any MCU. The Software UART Library provides easy communication with other devices via the RS232 protocol.

Note: The Software UART library implements time-based activities, so interrupts need to be disabled when using it.

External dependencies of Software UART Library

The following variable must be defined in all projects using RS-485 Library:	Description:	Example :
<code>dim Soft_Uart_RX as sbit external</code>	Receive line.	<code>dim Soft_Uart_RX as sbit at P3.B0</code>
<code>dim Soft_Uart_TX as sbit external</code>	Transmit line.	<code>dim Soft_Uart_TX as sbit at P3.B1</code>

Library Routines

- Soft_Uart_Init
- Soft_Uart_Read
- Soft_Uart_Write

Soft_Uart_Init

Prototype	<code>sub function Soft_Uart_Init(dim baud_rate as longword, dim inverted as byte) as word</code>
Returns	Nothing.
Description	<p>Configures and initializes the software UART module.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>baud_rate</code>: baud rate to be set. Maximum baud rate depends on the MCU's clock and working conditions. - <code>inverted</code>: inverted output flag. When set to a non-zero value, inverted logic on output is used.
Requires	<p>Global variables:</p> <ul style="list-style-type: none"> - <code>Soft_Uart_RX</code> receiver pin - <code>Soft_Uart_TX</code> transmitter pin <p>must be defined before using this function.</p>
Example	<pre>' Initialize Software UART communication on pins Rx, Tx, at 9600 bps Soft_Uart_Init(9600, 0)</pre>

Soft_Uart_Read

Prototype	<code>sub function Soft_Uart_Read(dim byref error as byte) as byte</code>
Returns	Byte received via UART.
Description	<p>The function receives a byte via software UART. This is a blocking function call (waits for start bit).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>error</code>: Error flag. Error code is returned through this variable. Upon successful transfer this flag will be set to zero. A non zero value indicates communication error.
Requires	Software UART must be initialized before using this function. See the <code>Soft_Uart_Init</code> routine.
Example	<pre> dim data as byte error as byte ... ' wait until data is received do data = Soft_Uart_Read(error) loop until (error = 0) ' Now we can work with data: if (data) then ... end if </pre>

Soft_Uart_Write

Prototype	<code>sub procedure Soft_Uart_Write(dim udata as byte)</code>
Returns	Nothing.
Description	This routine sends one byte via the Software UART bus. Parameters : - <code>udata</code> : data to be sent.
Requires	Software UART must be initialized before using this function. See the <code>Soft_Uart_Init</code> routine. Be aware that during transmission, software UART is incapable of receiving data – data transfer protocol must be set in such a way to prevent loss of information.
Example	<pre>dim some_byte as byte ... ' Write a byte via Soft Uart some_byte = 0x0A Soft_Uart_Write(some_byte)</pre>

Library Example

This example demonstrates simple data exchange via software UART. If MCU is connected to the PC, you can test the example from the mikroBasic for 8051 USART Terminal Tool.

```
program Soft_UART

' Soft UART connections
dim Soft_Uart_RX as sbit at P3.B0
dim Soft_Uart_TX as sbit at P3.B1
' End Soft UART connections

dim i, error_, byte_read as byte           ' Auxiliary variables

main:
    Soft_Uart_Init(4800, 0)                 ' Initialize Soft UART
    at 4800 bps
    for i = "A" to "z" step -1             ' Send bytes from "A"
    downto "z"
        Soft_Uart_Write(i)
        Delay_ms(100)
    next i

    while TRUE                             ' Endless loop
        byte_read = Soft_Uart_Read ( error_ ) ' Read byte, then test
        error flag
        if (error_ <> 0) then              ' If error was detect-
        ed
            P0 = 0xAA                       ' signal it on
            PORT0
        else
            Soft_Uart_Write(byte_read)      ' If error was not
            detected, return byte read
        end if
    wend
end.
```

SOUND LIBRARY

The mikroBasic for 8051 provides a Sound Library to supply users with routines necessary for sound signalization in their applications. Sound generation needs additional hardware, such as piezo-speaker (example of piezo-speaker interface is given on the schematic at the bottom of this page).

External dependencies of Sound Library

The following variables must be defined in all projects using Sound Library:	Description:	Example :
<code>dim Sound_Play_Pin as sbit external</code>	Sound output pin.	<code>dim Sound_Play_Pin as sbit at P0.B3</code>

Library Routines

- Sound_Init
- Sound_Play

Sound_Init

Prototype	<code>sub procedure Sound_Init()</code>
Returns	Nothing.
Description	Configures the appropriate MCU pin for sound generation.
Requires	<code>Sound_Play_Pin</code> variable must be defined before using this function.
Example	<pre>' Initialize the pin P0.3 for playing sound dim Sound_Play_Pin as sbit at P0.B3 ... Sound_Init()</pre>

Sound_Play

Prototype	<code>sub procedure Sound_Play(dim byref freq_in_Hz as word, dim byref duration_ms as word)</code>
Returns	Nothing.
Description	Generates the square wave signal on the appropriate pin. Parameters : - <code>freq_in_Hz</code> : signal frequency in Hertz (Hz) - <code>duration_ms</code> : signal duration in milliseconds (ms)
Requires	In order to hear the sound, you need a piezo speaker (or other hardware) on designated port. Also, you must call <code>Sound_Init</code> to prepare hardware for output before using this function.
Example	<code>' Play sound of 1KHz in duration of 100ms Sound_Play(1000, 100)</code>

Library Example

The example is a simple demonstration of how to use the Sound Library for playing tones on a piezo speaker.

`program Sound`

```
' Sound connections
dim Sound_Play_Pin as sbit at P0.B3
' End Sound connections
```

```
sub procedure Tone1()
    Sound_Play(500, 200)           ' Frequency = 500Hz, Duration =
200ms
end sub
```

```
sub procedure Tone2()
    Sound_Play(555, 200)         ' Frequency = 555Hz, Duration =
200ms
end sub
```

```
sub procedure Tone3()
    Sound_Play(625, 200)         ' Frequency = 625Hz, Duration =
200ms
end sub
```



```

sub procedure Melody()                                ' Plays the melody
"Yellow house"
    Tone1() Tone2() Tone3() Tone3()
    Tone1() Tone2() Tone3() Tone3()
    Tone1() Tone2() Tone3()
    Tone1() Tone2() Tone3() Tone3()
    Tone1() Tone2() Tone3()
    Tone3() Tone3() Tone2() Tone2() Tone1()
end sub

sub procedure ToneA()                                ' Tones used in Melody2
sub function
    Sound_Play(1250, 20)
end sub

sub procedure ToneC()
    Sound_Play(1450, 20)
end sub

sub procedure ToneE()
    Sound_Play(1650, 80)
end sub

sub procedure Melody2()                              ' Plays Melody2
dim i as byte
    i = 1
    while (i < 9)
        ToneA()
        ToneC()
        ToneE()
        Inc(i)
    wend
end sub

main:
    P1 = 255                                          ' Configure PORT1 as input
    Sound_Init()                                     ' Initialize sound pin
    Sound_Play(2000, 1000)                            ' Play starting sound, 2kHz, 1 sec-
ond
    while TRUE                                       ' endless loop
        if (P1_7 = 0) then                            ' If P1.7 is pressed play Tone1
            Tone1()
            while ( P1_7 = 0)                          ' Wait for button to be released
                nop
            wend
        end if

```

```

if ( P1_6 = 0) then                                ' If P1.6 is pressed play Tone2
    Tone2()
    while ( P1_6 = 0)                                ' Wait for button to be released
        nop
    wend
end if

if ( P1_5 = 0) then                                ' If P1.5 is pressed play Tone3
    Tone3()
    while ( P1_5 = 0)                                ' Wait for button to be released
        nop
    wend
end if

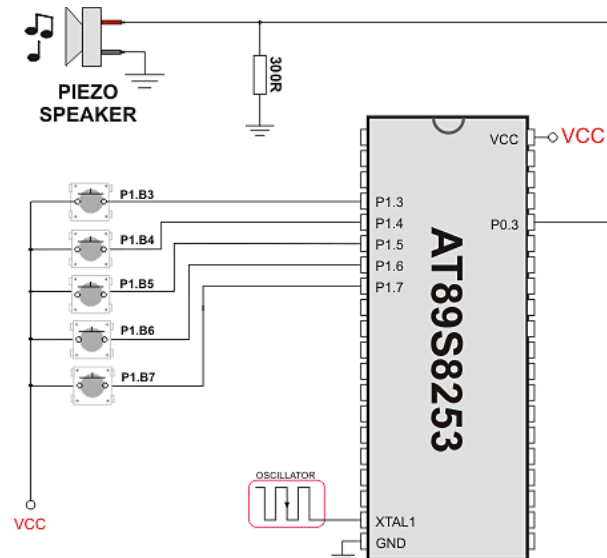
if ( P1_4 = 0 ) then                               ' If P1.4 is pressed play Melody2
    Melody2()
    while ( P1_4 = 0 )                               ' Wait for button to be released
        nop
    wend
end if

if ( P1_3 = 0) then                               ' If P1.3 is pressed play Melody
    Melody()
    while ( P1_3 = 0 )                               ' Wait for button to be released
        nop
    wend
end if
wend
end.

```

HW Connection

Example of Sound Library
sonnection



SPI LIBRARY

mikroBasic for 8051 provides a library for comfortable with SPI work in Master mode. The 8051 MCU can easily communicate with other devices via SPI: A/D converters, D/A converters, MAX7219, LTC1290, etc.

Library Routines

- Spi_Init
- Spi_Init_Advanced
- Spi_Read
- Spi_Write

Spi_Init

Prototype	<code>sub procedure Spi_Init()</code>
Returns	Nothing.
Description	This routine configures and enables SPI module with the following settings: <ul style="list-style-type: none">- master mode- clock idle low- 8 bit data transfer- most significant bit sent first- serial output data changes on idle to active transition of clock state- serial clock = fosc/128 (fosc/64 in x2 mode)
Requires	MCU must have SPI module.
Example	<pre>' Initialize the SPI module with default settings Spi_Init()</pre>

Spi_Init_Advanced

Prototype	<code>sub procedure Spi_Init_Advanced(dim adv_setting as byte)</code>			
Returns	Nothing.			
Description	This routine configures and enables the SPI module with the user defined settings.			
	Parameters :			
	- <code>adv_setting</code> : SPI module configuration flags. Predefined library constants (see the table below) can be ORed to form appropriate configuration value.			
		Bit	Mask	Description
				Predefined library const
		Master/slave [4] and clock rate select [1:0] bits		
	4, 1, 0	0x10	Sck = Fosc/4 (Fosc/2 in x2 mode), Master mode	MASTER_OSC_DIV4
		0x11	Sck = Fosc/16 (f/8 in x2 mode), Master mode	MASTER_OSC_DIV16
		0x12	Sck = Fosc/64 (f/32 in x2 mode), Master mode	MASTER_OSC_DIV64
		0x13	Sck = Fosc/128 (f/64 in x2 mode), Master mode	MASTER_OSC_DIV128
		SPI clock phase		
	2	0x00	Data changes on idle to active transition of the clock	IDLE_2_ACTIVE
	0x04	Data changes on active to idle transition of the clock	ACTIVE_2_IDLE	
	SPI clock polarity			
3	0x00	Clock idle level is low	CLK_IDLE_LOW	
	0x08	Clock idle level is high	CLK_IDLE_HIGH	
	Data order			
	0x00	Most significant bit sent first	DATA_ORDER_MSB	
	0x20	Least significant bit sent first	DATA_ORDER_LSB	
Requires	MCU must have SPI module.			
Example	<pre>' Set SPI to the Master Mode, clock = Fosc/4 , clock IDLE state low and data transmitted at low to high clock edge: Spi_Init_Advanced(MASTER_OSC_DIV4 or DATA_ORDER_MSB or CLK_IDLE_LOW or IDLE_2_ACTIVE)</pre>			

Spi_Read

Prototype	<code>sub function Spi_Read(dim buffer as byte) as byte</code>
Returns	Received data.
Description	<p>Reads one byte from the SPI bus.</p> <p>Parameters :</p> <p>- <code>buffer</code>: dummy data for clock generation (see device Datasheet for SPI modules implementation details)</p>
Requires	SPI module must be initialized before using this function. See <code>Spi_Init</code> and <code>Spi_Init_Advanced</code> routines.
Example	<pre>' read a byte from the SPI bus dim take, dummy1 as byte ... take = Spi_Read(dummy1)</pre>

Spi_Write

Prototype	<code>sub procedure Spi_Write(dim wrdata as byte)</code>
Returns	Nothing.
Description	<p>Writes byte via the SPI bus.</p> <p>Parameters :</p> <p>- <code>wrdata</code>: data to be sent</p>
Requires	SPI module must be initialized before using this function. See <code>Spi_Init</code> and <code>Spi_Init_Advanced</code> routines.
Example	<pre>' write a byte to the SPI bus dim buffer as byte ... Spi_Write(buffer)</pre>

Library Example

The code demonstrates how to use SPI library functions for communication between SPI module of the MCU and MAX7219 chip. MAX7219 controls eight 7 segment displays.

```
program SPI

' Serial 7-seg Display connections
dim CHIP_SEL as sbit at P1.B0      ' Chip Select pin definition
' End Serial 7-seg Display connections

sub procedure Select_max()          ' Function for selecting MAX7219
    CHIP_SEL = 0
    Delay_us(1)
end sub

sub procedure Deselect_max()       ' Function for deselecting MAX7219
    Delay_us(1)
    CHIP_SEL = 1
end sub

sub procedure Max7219_init()      ' Initializing MAX7219
    Select_max()
    Spi_Write(0x09)                ' BCD mode for digit decoding
    Spi_Write(0xFF)
    Deselect_max()

    Select_max()
    Spi_Write(0x0A)
    Spi_Write(0x0F)                ' Segment luminosity intensity
    Deselect_max()

    Select_max()
    Spi_Write(0x0B)
    Spi_Write(0x07)                ' Display refresh
    Deselect_max()

    Select_max()
    Spi_Write(0x0C)
    Spi_Write(0x01)                ' Turn on the display
    Deselect_max()

    Select_max()
    Spi_Write(0x00)
    Spi_Write(0xFF)                ' No test
    Deselect_max()
end sub
```

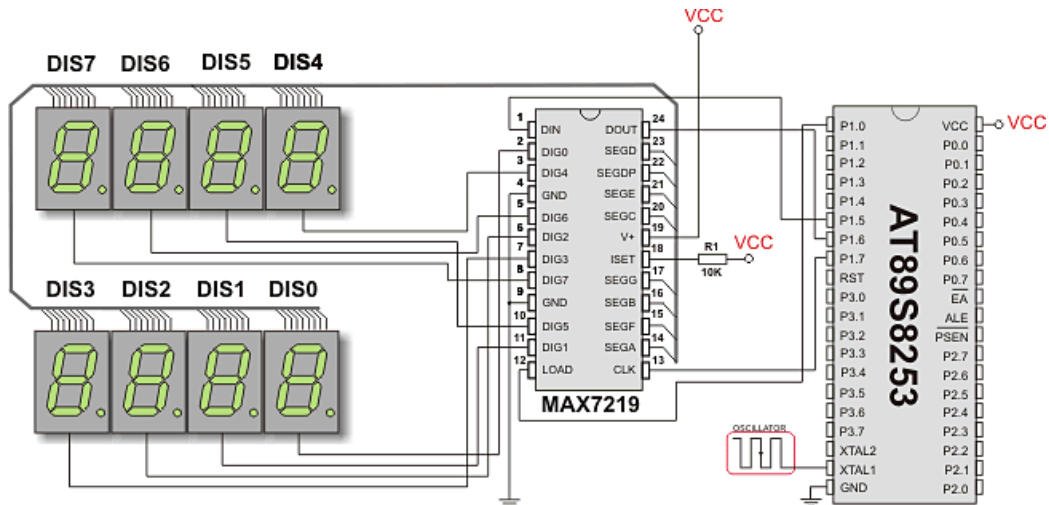
```

dim digit_position, digit_value as byte

main:
  Spi_Init()      ' Initialize SPI module, standard configuration
                  ' Instead of SPI_init, you can use SPI_init_Advanced
as shown below
                  ' Spi_Init_Advanced(MASTER_OSC_DIV4 | DATA_ORDER_MSB
| CLK_IDLE_LOW | IDLE_2_ACTIVE)
  Max7219_init()  ' Initialize max7219
  while TRUE      ' Endless loop
    for digit_value = 0 to 9
      for digit_position = 8 to 1 step -1
        Select_max()          ' Select max7219
        Spi_Write(digit_position) ' Send digit position
        Spi_Write(digit_value)  ' Send digit value
        Deselect_max()          ' Deselect MAX7219
        Delay_ms(300)
      next digit_position
    next digit_value
  wend
end.

```

HW Connection



SPI HW connection

SPI ETHERNET LIBRARY

The [ENC28J60](#) is a stand-alone Ethernet controller with an industry standard Serial Peripheral Interface (SPI™). It is designed to serve as an Ethernet network interface for any controller equipped with SPI.

The [ENC28J60](#) meets all of the IEEE 802.3 specifications. It incorporates a number of packet filtering schemes to limit incoming packets. It also provides an internal DMA module for fast data throughput and hardware assisted IP checksum calculations. Communication with the host controller is implemented via two interrupt pins and the SPI, with data rates of up to 10 Mb/s. Two dedicated pins are used for LED link and network activity indication.

This library is designed to simplify handling of the underlying hardware ([ENC28J60](#)). It works with any 8051 MCU with integrated SPI and more than 4 Kb ROM memory.

SPI Ethernet library supports:

- IPv4 protocol.
- ARP requests.
- ICMP echo requests.
- UDP requests.
- TCP requests (no stack, no packet reconstruction).
- packet fragmentation is **NOT** supported.

Note: The appropriate hardware SPI module must be initialized before using any of the SPI Ethernet library routines. Refer to Spi Library.

External dependencies of SPI Ethernet Library

The following variables must be defined in all projects using SPI Ethernet Library:	Description:	Example :
<code>dim Spi_Ethernet_CS as sbit external sfr</code>	ENC28J60 chip select pin.	<code>dim Spi_Ethernet_CS as sbit at P1.B1 sfr</code>
<code>dim Spi_Ethernet_RST as sbit external sfr</code>	ENC28J60 reset pin.	<code>dim Spi_Ethernet_RST as sbit at P1.B0 sfr</code>

The following routines must be defined in all project using SPI Ethernet Library:	Description:	Example :
<pre> sub function Spi_Ethernet_UserTCP(dim remoteHost as ^byte, dim remotePort as word, dim localPort as word, dim reqLength as word) as word </pre>	<p>TCP request handler.</p>	<p>Refer to the library example at the bottom of this page for code implementation.</p>
<pre> sub function Spi_Ethernet_UserUDP(dim remoteHost as ^byte, dim remotePort as word, dim destPort as word, dim reqLength as word) as word </pre>	<p>UDP request handler.</p>	<p>Refer to the library example at the bottom of this page for code implementation.</p>

Library Routines

- Spi_Ethernet_Init
- Spi_Ethernet_Enable
- Spi_Ethernet_Disable
- Spi_Ethernet_doPacket
- Spi_Ethernet_putByte
- Spi_Ethernet_putBytes
- Spi_Ethernet_putString
- Spi_Ethernet_putConstString
- Spi_Ethernet_putConstBytes
- Spi_Ethernet_getByte
- Spi_Ethernet_getBytes
- Spi_Ethernet_UserTCP
- Spi_Ethernet_UserUDP

Spi_Ethernet_Init

Prototype	<code>sub procedure Spi_Ethernet_Init(dim mac as ^byte, dim ip as ^byte, dim fullDuplex as byte)</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It initializes ENC28J60 controller. This function is internally splitted into 2 parts to help linker when coming short of memory.</p> <p>ENC28J60 controller settings (parameters not mentioned here are set to default):</p> <ul style="list-style-type: none"> - receive buffer start address : 0x0000. - receive buffer end address : 0x19AD. - transmit buffer start address: 0x19AE. - transmit buffer end address : 0x1FFF. - RAM buffer read/write pointers in auto-increment mode. - receive filters set to default: CRC + MAC Unicast + MAC Broadcast in OR mode. - flow control with TX and RX pause frames in full duplex mode. - frames are padded to 60 bytes + CRC. - maximum packet size is set to 1518. - Back-to-Back Inter-Packet Gap: 0x15 in full duplex mode; 0x12 in half duplex mode. - Non-Back-to-Back Inter-Packet Gap: 0x0012 in full duplex mode; 0x0C12 in half duplex mode. - Collision window is set to 63 in half duplex mode to accomodate some ENC28J60 revisions silicon bugs. - CLKOUT output is disabled to reduce EMI generation. - half duplex loopback disabled. - LED configuration: default (LEDA-link status, LEDB-link activity). <p>Parameters:</p> <ul style="list-style-type: none"> - <code>mac</code>: RAM buffer containing valid MAC address. - <code>ip</code>: RAM buffer containing valid IP address. - <code>fullDuplex</code>: ethernet duplex mode switch. Valid values: 0 (half duplex mode) and 1 (full duplex mode).
Requires	The appropriate hardware SPI module must be previously initialized.
Example	<pre>const Spi_Ethernet_HALFDUPLEX = 0 const Spi_Ethernet_FULLDUPLEX = 1 myMacAddr as byte[6] ' my MAC address myIpAddr as byte[4] ' my IP addr ... myMacAddr[0] = 0x00 myMacAddr[1] = 0x14</pre>

Example

```
myMacAddr[ 2] = 0xA5
myMacAddr[ 3] = 0x76
myMacAddr[ 4] = 0x19
myMacAddr[ 5] = 0x3F

myIpAddr[ 0]  = 192
myIpAddr[ 1]  = 168
myIpAddr[ 2]  = 20
myIpAddr[ 3]  = 60

Spi_Init()
Spi_Ethernet_Init(PORTC, 0, PORTC, 1, myMacAddr, myIpAddr,
Spi_Ethernet_FULLDUPLEX)
```

Spi_Ethernet_Enable

Prototype	<code>sub procedure Spi_Ethernet_Enable(dim enFlt as byte)</code>		
Returns	Nothing.		
Description	<p>This is MAC module routine. This routine enables appropriate network traffic on the ENC28J60 module by the means of it's receive filters (unicast, multicast, broadcast, crc). Specific type of network traffic will be enabled if a corresponding bit of this routine's input parameter is set. Therefore, more than one type of network traffic can be enabled at the same time. For this purpose, predefined library constants (see the table below) can be ORed to form appropriate input value.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>enFlt</code>: network traffic/receive filter flags. Each bit corresponds to the appropriate network traffic/receive filter: 		
	Bit	Mask	Description
	0	0x01	MAC Broadcast traffic/receive filter flag. When set, MAC broadcast traffic will be enabled.
	1	0x02	MAC Multicast traffic/receive filter flag. When set, MAC multicast traffic will be enabled.
	2	0x04	not used
	3	0x08	not used
	4	0x10	not used
	5	0x20	CRC check flag. When set, packets with invalid CRC field will be discarded.
	6	0x40	not used
	7	0x80	MAC Unicast traffic/receive filter flag. When set, MAC unicast traffic will be enabled.
		Predefined library const	
		Spi_Ethernet_BROADCAST	
		Spi_Ethernet_MULTICAST	
		none	
		none	
		none	
		Spi_Ethernet_CRC	
		none	
		Spi_Ethernet_UNICAST	
		<p>Note: Advance filtering available in the ENC28J60 module such as Pattern Match, Magic Packet and Hash Table can not be enabled by this routine. Additionally, all filters, except CRC, enabled with this routine will work in OR mode, which means that packet will be received if any of the enabled filters accepts it.</p> <p>Note: This routine will change receive filter configuration on-the-fly. It will not, in any way, mess with enabling/disabling receive/transmit logic or any other part of the ENC28J60 module. The ENC28J60 module should be properly cofigured by the means of Spi_Ethernet_Init routine.</p>	

Requires	Ethernet module has to be initialized. See Spi_Ethernet_Init.
Example	<code>Spi_Ethernet_Enable(Spi_Ethernet_CRC or Spi_Ethernet_UNICAST) ' enable CRC checking and Unicast traffic</code>

Spi_Ethernet_Disable

Prototype	<code>sub procedure Spi_Ethernet_Disable(dim disFlt as byte)</code>		
Returns	Nothing.		
Description	<p>This is MAC module routine. This routine disables appropriate network traffic on the ENC28J60 module by the means of it's receive filters (unicast, multicast, broadcast, crc). Specific type of network traffic will be disabled if a corresponding bit of this routine's input parameter is set. Therefore, more than one type of network traffic can be disabled at the same time. For this purpose, predefined library constants (see the table below) can be ORed to form appropriate input value.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>disFlt</code>: network traffic/receive filter flags. Each bit corresponds to the appropriate network traffic/receive filter: 		
	Bit	Mask	Description
	0	0x01	MAC Broadcast traffic/receive filter flag. When set, MAC broadcast traffic will be disabled.
	1	0x02	MAC Multicast traffic/receive filter flag. When set, MAC multicast traffic will be disabled.
	2	0x04	not used
	3	0x08	not used
	4	0x10	not used
	5	0x20	CRC check flag. When set, CRC check will be disabled and packets with invalid CRC field will be accepted.
	6	0x40	not used
	7	0x80	MAC Unicast traffic/receive filter flag. When set, MAC unicast traffic will be disabled.
		Predefined library const	
		<code>Spi_Ethernet_BROADCAST</code>	
		<code>Spi_Ethernet_MULTICAST</code>	
		<code>none</code>	
		<code>none</code>	
		<code>none</code>	
		<code>Spi_Ethernet_CRC</code>	
		<code>none</code>	
		<code>Spi_Ethernet_UNICAST</code>	

Description	<p>Note: Advance filtering available in the <code>ENC28J60</code> module such as <code>Pattern Match</code>, <code>Magic Packet</code> and <code>Hash Table</code> can not be disabled by this routine.</p> <p>Note: This routine will change receive filter configuration on-the-fly. It will not, in any way, mess with enabling/disabling receive/transmit logic or any other part of the <code>ENC28J60</code> module. The <code>ENC28J60</code> module should be properly configured by the means of <code>Spi_Ethernet_Init</code> routine.</p>
Requires	Ethernet module has to be initialized. See <code>Spi_Ethernet_Init</code> .
Example	<code>Spi_Ethernet_Disable(Spi_Ethernet_CRC or Spi_Ethernet_UNICAST) ' disable CRC checking and Unicast traffic</code>

Spi_Ethernet_doPacket

Prototype	<code>sub function Spi_Ethernet_doPacket() as byte</code>
Returns	<ul style="list-style-type: none"> - 0 - upon successful packet processing (zero packets received or received packet processed successfully). - 1 - upon reception error or receive buffer corruption. <code>ENC28J60</code> controller needs to be restarted. - 2 - received packet was not sent to us (not our IP, nor IP broadcast address). - 3 - received IP packet was not IPv4. - 4 - received packet was of type unknown to the library.
Description	<p>This is MAC module routine. It processes next received packet if such exists. Packets are processed in the following manner:</p> <ul style="list-style-type: none"> - ARP & ICMP requests are replied automatically. - upon TCP request the <code>Spi_Ethernet_UserTCP</code> function is called for further processing. - upon UDP request the <code>Spi_Ethernet_UserUDP</code> function is called for further processing. <p>Note: <code>Spi_Ethernet_doPacket</code> must be called as often as possible in user's code.</p>
Requires	Ethernet module has to be initialized. See <code>Spi_Ethernet_Init</code> .
Example	<pre>while TRUE ... Spi_Ethernet_doPacket() ' process received packets ... wend</pre>

Spi_Ethernet_putByte

Prototype	<code>sub procedure Spi_Ethernet_putByte(dim v as byte)</code>
Returns	Nothing.
Description	This is MAC module routine. It stores one byte to address pointed by the current <code>ENC28J60</code> write pointer (<code>EWRPT</code>). Parameters: - <code>v</code> : value to store
Requires	Ethernet module has to be initialized. See <code>Spi_Ethernet_Init</code> .
Example	<pre>dim data as byte ... Spi_Ethernet_putByte(data) ' put an byte into ENC28J60 buffer</pre>

Spi_Ethernet_putBytes

Prototype	<code>sub procedure Spi_Ethernet_putBytes(dim ptr as ^byte, dim n as byte)</code>
Returns	Nothing.
Description	This is MAC module routine. It stores requested number of bytes into <code>ENC28J60</code> RAM starting from current <code>ENC28J60</code> write pointer (<code>EWRPT</code>) location. Parameters: - <code>ptr</code> : RAM buffer containing bytes to be written into <code>ENC28J60</code> RAM. - <code>n</code> : number of bytes to be written.
Requires	Ethernet module has to be initialized. See <code>Spi_Ethernet_Init</code> .
Example	<pre>dim buffer as byte[17] ... buffer = "mikroElektronika" ... Spi_Ethernet_putBytes(buffer, 16) ' put an RAM array into ENC28J60 buffer</pre>

Spi_Ethernet_putConstBytes

Prototype	<code>sub procedure Spi_Ethernet_putConstBytes(const ptr as ^byte, dim n as byte)</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It stores requested number of const bytes into ENC28J60 RAM starting from current ENC28J60 write pointer (EWRPT) location.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - ptr: const buffer containing bytes to be written into ENC28J60 RAM. - n: number of bytes to be written.
Requires	Ethernet module has to be initialized. See Spi_Ethernet_Init.
Example	<pre>const buffer as byte[17] ... buffer = "mikroElektronika" ... Spi_Ethernet_putConstBytes(buffer, 16) ' put a const array into ENC28J60 buffer</pre>

Spi_Ethernet_putString

Prototype	<code>sub function Spi_Ethernet_putString(dim ptr as ^byte) as word</code>
Returns	Number of bytes written into ENC28J60 RAM.
Description	<p>This is MAC module routine. It stores whole string (excluding null termination) into ENC28J60 RAM starting from current ENC28J60 write pointer (EWRPT) location.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - ptr: string to be written into ENC28J60 RAM.
Requires	Ethernet module has to be initialized. See Spi_Ethernet_Init.
Example	<pre>dim buffer as string[16] ... buffer = "mikroElektronika" ... Spi_Ethernet_putString(buffer) ' put a RAM string into ENC28J60 buffer</pre>

Spi_Ethernet_putConstString

Prototype	<code>sub function Spi_Ethernet_putConstString(const ptr as ^byte) as word</code>
Returns	Number of bytes written into ENC28J60 RAM.
Description	This is MAC module routine. It stores whole const string (excluding null termination) into ENC28J60 RAM starting from current ENC28J60 write pointer (EWRPT) location. Parameters: - <code>ptr</code> : const string to be written into ENC28J60 RAM.
Requires	Ethernet module has to be initialized. See Spi_Ethernet_Init.
Example	<pre>const buffer as string[16] ... buffer = "mikroElektronika" ... Spi_Ethernet_putConstString(buffer) ' put a const string into ENC28J60 buffer</pre>

Spi_Ethernet_getByte

Prototype	<code>sub function Spi_Ethernet_getByte() as byte</code>
Returns	Byte read from ENC28J60 RAM.
Description	This is MAC module routine. It fetches a byte from address pointed to by current ENC28J60 read pointer (ERDPT).
Requires	Ethernet module has to be initialized. See Spi_Ethernet_Init.
Example	<pre>dim buffer as byte<> ... buffer = Spi_Ethernet_getByte() ' read a byte from ENC28J60 buffer</pre>

Spi_Ethernet_getBytes

Prototype	<code>sub procedure Spi_Ethernet_getBytes(dim ptr as ^byte, dim addr as word, dim n as byte)</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It fetches requested number of bytes from ENC28J60 RAM starting from given address. If value of 0xFFFF is passed as the address parameter, the reading will start from current ENC28J60 read pointer (ERDPT) location.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - ptr: buffer for storing bytes read from ENC28J60 RAM. - addr: ENC28J60 RAM start address. Valid values: 0..8192. - n: number of bytes to be read.
Requires	Ethernet module has to be initialized. See Spi_Ethernet_Init.
Example	<pre>dim buffer as byte[16] ... Spi_Ethernet_getBytes(buffer, 0x100, 16) ' read 16 bytes, starting from address 0x100</pre>

Spi_Ethernet_UserTCP

Prototype	<code>sub function Spi_Ethernet_UserTCP(dim remoteHost as ^byte, dim remotePort as word, dim localPort as word, dim reqLength as word) as word</code>
Returns	<ul style="list-style-type: none"> - 0 - there should not be a reply to the request. - Length of TCP/HTTP reply data field - otherwise.
Description	<p>This is TCP module routine. It is internally called by the library. The user accesses to the TCP/HTTP request by using some of the Spi_Ethernet_get routines. The user puts data in the transmit buffer by using some of the Spi_Ethernet_put routines. The function must return the length in bytes of the TCP/HTTP reply, or 0 if there is nothing to transmit. If there is no need to reply to the TCP/HTTP requests, just define this function with return(0) as a single statement.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>remoteHost</code>: client's IP address. - <code>remotePort</code>: client's TCP port. - <code>localPort</code>: port to which the request is sent. - <code>reqLength</code>: TCP/HTTP request data field length. <p>Note: The function source code is provided with appropriate example projects. The code should be adjusted by the user to achieve desired reply.</p>
Requires	Ethernet module has to be initialized. See Spi_Ethernet_Init.
Example	This function is internally called by the library and should not be called by the user's code.

Spi_Ethernet_UserUDP

Prototype	<code>sub function Spi_Ethernet_UserUDP(dim remoteHost as ^byte, dim remotePort as word, dim destPort as word, dim reqLength as word) as word</code>
Returns	<ul style="list-style-type: none"> - 0 - there should not be a reply to the request. - Length of UDP reply data field - otherwise.
Description	<p>This is UDP module routine. It is internally called by the library. The user accesses to the UDP request by using some of the Spi_Ethernet_get routines. The user puts data in the transmit buffer by using some of the Spi_Ethernet_put routines. The function must return the length in bytes of the UDP reply, or 0 if nothing to transmit. If you don't need to reply to the UDP requests, just define this function with a return(0) as single statement.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>remoteHost</code>: client's IP address. - <code>remotePort</code>: client's port. - <code>destPort</code>: port to which the request is sent. - <code>reqLength</code>: UDP request data field length. <p>Note: The function source code is provided with appropriate example projects. The code should be adjusted by the user to achieve desired reply.</p>
Requires	Ethernet module has to be initialized. See Spi_Ethernet_Init.
Example	This function is internally called by the library and should not be called by the user's code.

Library Example

This code shows how to use the 8051 mini Ethernet library :

- the board will reply to ARP & ICMP echo requests
- the board will reply to UDP requests on any port :

returns the request in upper char with a header made of remote host IP & port number

- the board will reply to HTTP requests on port 80, GET method with pathnames :

/ will return the HTML main page
/s will return board status as text string
/t0 ... /t7 will toggle P3.b0 to P3.b7 bit and return HTML main page
all other requests return also HTML main page.

```

// duplex config flags
#define Spi_Ethernet_HALFDUPLEX      0x00 // half duplex
#define Spi_Ethernet_FULLDUPLEX     0x01 // full duplex

// mE ethernet NIC pinout
sfr sbit Spi_Ethernet_RST at P1.B0;
sfr sbit Spi_Ethernet_CS at P1.B1;
// end ethernet NIC definitions

/*****
 * ROM constant strings
 */
const code byte httpHeader[] = "HTTP/1.1 200 OK\nContent-type: " ;
// HTTP header
const code byte httpMimeTypeHTML[] = "text/html\n\n" ; //
HTML MIME type
const code byte httpMimeTypeScript[] = "text/plain\n\n" ; //
TEXT MIME type
idata byte httpMethod[] = "GET /";
/*
 * web page, splited into 2 parts :
 * when coming short of ROM, fragmented data is handled more effi-
 * ciently by linker
 *
 * this HTML page calls the boards to get its status, and builds
 * itself with javascript
 */
const code char *indexPage = // Change the IP
address of the page to be refreshed
"<meta http-equiv=\"refresh\"
content=\"3;url=http://192.168.1.60\">\
<HTML><HEAD></HEAD><BODY>\
<h1>8051 + ENC28J60 Mini Web Server</h1>\
<a href=/>Reload</a>\
<script src=/s></script>\
<table><tr><td><table border=1 style=\"font-size:20px ;font-family:
terminal ;\">\
<tr><th colspan=2>P0</th></tr>\
<script>\
var str,i;\
str=\"\";\
for(i=0;i<8;i++)\
{ str+=\"<tr><td bgcolor=pink>BUTTON #\"+i+\"</td>\";\
if(P0&(1<<i)){ str+=\"<td bgcolor=red>ON\";}\
else { str+=\"<td bgcolor=#cccccc>OFF\";}\
str+=\"</td></tr>\";\
document.write(str) ;\
</script>\
" ;

```

```
const char *indexPage2 = "</table></td><td>\n\
<table border=1 style=\"font-size:20px ;font-family: terminal ;\">\n\
<tr><th colspan=3>P3</th></tr>\n\
<script>\n\
var str,i;\n\
str=\"\";\n\
for(i=0;i<8;i++)\n\
{ str+=\"<tr><td bgcolor=yellow>LED #\"+i+\"</td>\";\n\
if(P3&(1<<i)){ str+=\"<td bgcolor=red>ON\";\n\
else { str+=\"<td bgcolor=#cccccc>OFF\";\n\
str+=\"</td><td><a href=/t\"+i+\">Toggle</a></td></tr>\";\n\
document.write(str) ;\n\
</script>\n\
</table></td></tr></table>\n\
This is HTTP request\n\
#<script>document.write(REQ)</script></BODY></HTML>\n\
\" ;\n\
\n\
/*****\n\
 * RAM variables\n\
 */\n\
idata byte myMacAddr[ 6] = { 0x00, 0x14, 0xA5, 0x76, 0x19, 0x3f} ;\n\
// my MAC address\n\
idata byte myIpAddr[ 4] = { 192, 168, 1, 60} ; //\n\
my IP address\n\
idata byte getRequest[ 15] ; //\n\
HTTP request buffer\n\
idata byte dyna[ 29] ; //\n\
buffer for dynamic response\n\
idata unsigned long httpCounter = 0 ;\n\
// counter of HTTP requests\n\
\n\
/*****\n\
 * functions\n\
 */\n\
\n\
/*\n\
 * put the constant string pointed to by s to the ENC transmit buffer.\n\
 */\n\
/*unsigned int putConstString(const code char *s)\n\
{\n\
    unsigned int ctr = 0 ;\n\
\n\
    while(*s)\n\
    {\n\
        Spi_Ethernet_putByte(*s++) ;\n\
        ctr++ ;\n\
    }\n\
    return(ctr) ;\n\
}*/
```

```
/*
 * it will be much faster to use library Spi_Ethernet_putConstString
 routine
 * instead of putConstString routine above. However, the code will
 be a little
 * bit bigger. User should choose between size and speed and pick the
 implementation that
 * suites him best. If you choose to go with the putConstString def-
 inition above
 * the #define line below should be commented out.
 *
 */
#define putConstString Spi_Ethernet_putConstString

/*
 * put the string pointed to by s to the ENC transmit buffer
 */
/*unsigned int    putString(char *s)
 {
     unsigned int ctr = 0 ;

     while(*s)
     {
         Spi_Ethernet_putByte(*s++) ;

         ctr++ ;
     }
     return(ctr) ;
 }*/

/*
 * it will be much faster to use library Spi_Ethernet_putString rou-
 tine
 * instead of putString routine above. However, the code will be a
 little
 * bit bigger. User should choose between size and speed and pick the
 implementation that
 * suites him best. If you choose to go with the putString defini-
 tion above
 * the #define line below should be commented out.
 *
 */
#define putString Spi_Ethernet_putString

/*
 * this function is called by the library
 * the user accesses to the HTTP request by successive calls to
 Spi_Ethernet_getByte()
 * the user puts data in the transmit buffer by successive calls to
 Spi_Ethernet_putByte()
 * the function must return the length in bytes of the HTTP reply,
 or 0 if nothing to transmit
 *
 * if you don't need to reply to HTTP requests,
 * just define this function with a return(0) as single statement
 *
 */
```

```
unsigned int Spi_Ethernet_UserTCP(byte *remoteHost, unsigned int
remotePort, unsigned int localPort, unsigned int reqLength)
{
    idata unsigned int len; // my reply length

    if(localPort != 80) // I listen
only to web request on port 80
    {
        return(0) ;
    }

    // get 10 first bytes only of the request, the rest does not
matter here
    for(len = 0 ; len < 10 ; len++)
    {
        getRequest[ len] = Spi_Ethernet_getByte() ;
    }
    getRequest[ len] = 0 ;

    len = 0;

    if(memcmp(getRequest, httpMethod, 5)) // only GET
method is supported here
    {
        return(0) ;
    }

    httpCounter++ ; // one more
request done

    if(getRequest[ 5] == 's') // if request
path name starts with s, store dynamic data in transmit buffer
    {
        // the text string replied by this request can be
interpreted as javascript statements
        // by browsers

        len = putConstString(httpHeader) ; //
HTTP header
        len += putConstString(httpMimeTypeScript) ; //
with text MIME type

        // add P3 value (buttons) to reply
        len += putConstString("var P3=") ;
        WordToStr(P3, dyna) ;
        len += putString(dyna) ;
        len += putConstString(";") ;
    }
}
```



```

        // add P0 value (LEDs) to reply
        len += putConstString("var P0=") ;
        WordToStr(P0, dyna) ;
        len += putString(dyna) ;
        len += putConstString(";") ;

        // add HTTP requests counter to reply
        WordToStr(httpCounter, dyna) ;
        len += putConstString("var REQ=") ;
        len += putString(dyna) ;
        len += putConstString(";") ;
    }
    else if(getRequest[ 5] == 't') //
if request path name starts with t, toggle P3 (LED) bit number that
comes after
    {
        byte    bitMask = 0 ; // for bit
mask
        if(isdigit(getRequest[ 6] )) //
if 0 <= bit number <= 9, bits 8 & 9 does not exist but does not mat-
ter
            {
                bitMask = getRequest[ 6] - '0' ; //
convert ASCII to integer
                bitMask = 1 << bitMask ; //
create bit mask
                P3 ^= bitMask ; //
toggle P3 with xor operator
            }
    }

    if(len == 0) //
what do to by default
    {
        len = putConstString(httpHeader) ; //
HTTP header
        len += putConstString(httpMimeTypeHTML) ; //
with HTML MIME type
        len += putConstString(indexPage) ; //
HTML page first part
        len += putConstString(indexPage2) ; //
HTML page second part
    }

    return(len) ; //
return to the library with the number of bytes to transmit
    }

```

```
/*
 * this function is called by the library
 * the user accesses to the UDP request by successive calls to
Spi_Ethernet_getByte()
 * the user puts data in the transmit buffer by successive calls to
Spi_Ethernet_putByte()
 * the function must return the length in bytes of the UDP reply, or
0 if nothing to transmit
 *
 * if you don't need to reply to UDP requests,
 * just define this function with a return(0) as single statement
 *
 */
unsigned int Spi_Ethernet_UserUDP(byte *remoteHost, unsigned int
remotePort, unsigned int destPort, unsigned int reqLength)
{
    idata unsigned int len ; // my reply
length
    idata byte * ptr ; // pointer to the
dynamic buffer

    // reply is made of the remote host IP address in human read-
able format
    ByteToStr(remoteHost[ 0], dyna) ; // first IP
address byte
    dyna[ 3] = '.' ;
    ByteToStr(remoteHost[ 1], dyna + 4) ; // second
    dyna[ 7] = '.' ;
    ByteToStr(remoteHost[ 2], dyna + 8) ; // third
    dyna[ 11] = '.' ;
    ByteToStr(remoteHost[ 3], dyna + 12) ; // fourth

    dyna[ 15] = ':' ; // add
separator

    // then remote host port number
    WordToStr(remotePort, dyna + 16) ;
    dyna[ 21] = '[' ;
    WordToStr(destPort, dyna + 22) ;
    dyna[ 27] = ']' ;
    dyna[ 28] = 0 ;

    // the total length of the request is the length of the
dynamic string plus the text of the request
    len = 28 + reqLength;

    // puts the dynamic string into the transmit buffer
    Spi_Ethernet_putBytes(dyna, 28) ;
}
```

```
// then puts the request string converted into upper char into the
transmit buffer
    while(reqLength-->0)
    {
        Spi_Ethernet_putByte(toupper(Spi_Ethernet_getByte()))
    }
;

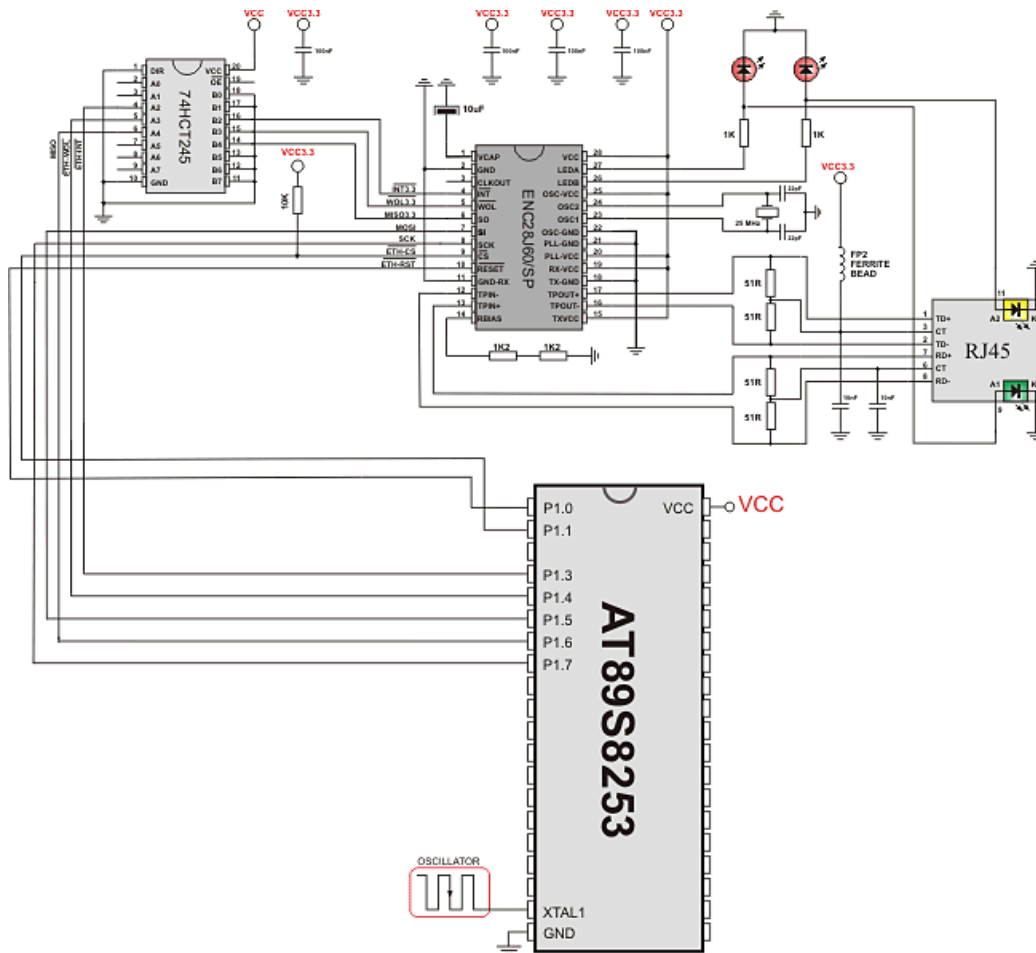
    return(len) ; // back to the library with the
length of the UDP reply
}

/*
 * main entry
 */
procedure main()
{
    /*
    * starts ENC28J60 with :
    * reset bit on P1_0
    * CS bit on P1_1
    * my MAC & IP address
    * full duplex
    */
    Spi_Init_Advanced(MASTER_OSC_DIV16 | CLK_IDLE_LOW |
IDLE_2_ACTIVE | DATA_ORDER_MSB);
    Spi_Ethernet_Init(myMacAddr, myIpAddr, Spi_Ethernet_FULLDU-
PLEX) ; // full duplex, CRC + MAC Unicast + MAC Broadcast filtering

    while(1) // do forever
    {
        /*
        * if necessary, test the return value to get error
code
        */
        Spi_Ethernet_doPacket() ; // process incoming
Ethernet packets

        /*
        * add your stuff here if needed
        * Spi_Ethernet_doPacket() must be called as often
as possible
        * otherwise packets could be lost
        */
    }
}
```

HW Connection



SPI GRAPHIC LCD LIBRARY

The mikroBasic for 8051 provides a library for operating Graphic LCD 128x64 (with commonly used Samsung KS108/KS107 controller) via SPI interface.

For creating a custom set of GLCD images use GLCD Bitmap Editor Tool.

Note: The library uses the SPI module for communication. User must initialize SPI module before using the SPI Graphic LCD Library.

Note: This Library is designed to work with the mikroElektronika's Serial LCD/GLCD Adapter Board pinout, see schematic at the bottom of this page for details.

External dependencies of SPI Graphic LCD Library

The implementation of SPI Graphic LCD Library routines is based on Port Expander Library routines.

External dependencies are the same as Port Expander Library external dependencies.

Library Routines

Basic routines:

- Spi_Glcd_Init
- Spi_Glcd_Set_Side
- Spi_Glcd_Set_Page
- Spi_Glcd_Set_X
- Spi_Glcd_Read_Data
- Spi_Glcd_Write_Data

Advanced routines:

- Spi_Glcd_Fill
- Spi_Glcd_Dot
- Spi_Glcd_Line
- Spi_Glcd_V_Line
- Spi_Glcd_H_Line
- Spi_Glcd_Rectangle
- Spi_Glcd_Box
- Spi_Glcd_Circle
- Spi_Glcd_Set_Font
- Spi_Glcd_Write_Char
- Spi_Glcd_Write_Text
- Spi_Glcd_Image

Spi_Glcd_Init

Prototype	<code>sub procedure Spi_Glcd_Init(dim DeviceAddress as byte)</code>
Returns	Nothing.
Description	<p>Initializes the GLCD module via SPI interface.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>DeviceAddress</code>: spi expander hardware address, see schematic at the bottom of this page
Requires	<p><code>SPExpanderCS</code> and <code>SPExpanderRST</code> variables must be defined before using this function.</p> <p>The SPI module needs to be initialized. See <code>Spi_Init</code> and <code>Spi_Init_Advanced</code> routines.</p>
Example	<pre>' port expander pinout definition dim SPExpanderRST as sbit at P1.B0 SPExpanderCS as sbit at P1.B1 ... Spi_Init_Advanced(MASTER_OSC_DIV4 or CLK_IDLE_LOW or IDLE_2_ACTIVE or DATA_ORDER_MSB) Spi_Glcd_Init(0)</pre>

Spi_Glcd_Set_Side

Prototype	<code>sub procedure SPI_Glcd_Set_Side(dim x_pos as byte)</code>
Returns	Nothing.
Description	<p>Selects GLCD side. Refer to the GLCD datasheet for detail explanation.</p> <p>Parameters :</p> <p>- <code>x_pos</code>: position on x-axis. Valid values: 0..127</p> <p>The parameter <code>x_pos</code> specifies the GLCD side: values from 0 to 63 specify the left side, values from 64 to 127 specify the right side.</p> <p>Note: For side, x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.
Example	<p>The following two lines are equivalent, and both of them select the left side of GLCD:</p> <pre>SPI_Glcd_Set_Side(0) SPI_Glcd_Set_Side(10)</pre>

Spi_Glcd_Set_Page

Prototype	<code>sub procedure Spi_Glcd_Set_Page(dim page as byte)</code>
Returns	Nothing.
Description	<p>Selects page of GLCD.</p> <p>Parameters :</p> <p>- <code>page</code>: page number. Valid values: 0..7</p> <p>Note: For side, x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.
Example	<code>Spi_Glcd_Set_Page(5)</code>

Spi_Glcd_Set_X

Prototype	<code>sub procedure Spi_Glcd_Set_X(dim x_pos as byte)</code>
Returns	Nothing.
Description	<p>Sets x-axis position to <code>x_pos</code> dots from the left border of GLCD within the selected side.</p> <p>Parameters :</p> <p>- <code>x_pos</code>: position on x-axis. Valid values: 0..63</p> <p>Note: For side, x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.
Example	<code>Spi_Glcd_Set_X(25)</code>

Spi_Glcd_Read_Data

Prototype	<code>sub function Spi_Glcd_Read_Data() as byte</code>
Returns	One byte from GLCD memory.
Description	Reads data from the current location of GLCD memory and moves to the next location.
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines. GLCD side, x-axis position and page should be set first. See the functions <code>Spi_Glcd_Set_Side</code> , <code>Spi_Glcd_Set_X</code> , and <code>Spi_Glcd_Set_Page</code> .
Example	<pre>dim data as byte ... data = Spi_Glcd_Read_Data()</pre>

Spi_Glcd_Write_Data

Prototype	<code>sub procedure Spi_Glcd_Write_Data(dim Ddata as byte)</code>
Returns	Nothing.
Description	Writes one byte to the current location in GLCD memory and moves to the next location. Parameters : - <code>Ddata</code> : data to be written
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines. GLCD side, x-axis position and page should be set first. See the functions <code>Spi_Glcd_Set_Side</code> , <code>Spi_Glcd_Set_X</code> , and <code>Spi_Glcd_Set_Page</code> .
Example	<pre>dim ddata as byte ... Spi_Glcd_Write_Data(ddata)</pre>

Spi_Glcd_Fill

Prototype	<code>sub procedure Spi_Glcd_Fill(dim pattern as byte)</code>
Returns	Nothing.
Description	Fills GLCD memory with byte pattern. Parameters : - <code>pattern</code> : byte to fill GLCD memory with To clear the GLCD screen, use <code>Spi_Glcd_Fill(0)</code> . To fill the screen completely, use <code>Spi_Glcd_Fill(0xFF)</code> .
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.
Example	<pre>' Clear screen Spi_Glcd_Fill(0)</pre>

Spi_Glcd_Dot

Prototype	<code>sub procedure Spi_Glcd_Dot (dim x_pos as byte, dim y_pos as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a dot on GLCD at coordinates (x_pos, y_pos).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - x_pos: x position. Valid values: 0..127 - y_pos: y position. Valid values: 0..63 - color: colx_pos as byte; page_num as byte; color as byte) or parameter. Valid values: 0..2 <p>The parameter color determines the dot state: 0 clears dot, 1 puts a dot, and 2 inverts dot state.</p> <p>Note: For x and y axis layout explanation see schematic at the bottom of this page.</p>
Requires	GLCD needs to be initialized for SPI communication, see Spi_Glcd_Init routines.
Example	<code>' Invert the dot in the upper left corner Spi_Glcd_Dot(0, 0, 2)</code>

Spi_Glcd_Line

Prototype	<code>sub procedure SPI_Glcd_Line(dim x_start as integer, dim y_start as integer, dim x_end as integer, dim y_end as integer, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a line on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_start</code>: x coordinate of the line start. Valid values: 0..127 - <code>y_start</code>: y coordinate of the line start. Valid values: 0..63 - <code>x_end</code>: x coordinate of the line end. Valid values: 0..127 - <code>y_end</code>: y coordinate of the line end. Valid values: 0..63 - <code>color</code>: color parameter. Valid values: 0..2 <p>Parameter <code>color</code> determines the line color: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.
Example	<code>' Draw a line between dots (0,0) and (20,30)</code> <code>Spi_Glcd_Line(0, 0, 20, 30, 1)</code>

Spi_Glcd_V_Line

Prototype	<code>sub procedure Spi_Glcd_V_Line(dim y_start as byte, dim y_end as byte, dim x_pos as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a vertical line on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>y_start</code>: y coordinate of the line start. Valid values: 0..63 - <code>y_end</code>: y coordinate of the line end. Valid values: 0..63 - <code>x_pos</code>: x coordinate of vertical line. Valid values: 0..127 - <code>color</code>: color parameter. Valid values: 0..2 <p>Parameter <code>color</code> determines the line color: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.
Example	<code>' Draw a vertical line between dots (10,5) and (10,25)</code> <code>Spi_Glcd_V_Line(5, 25, 10, 1)</code>

Spi_Glcd_H_Line

Prototype	<code>sub procedure Spi_Glcd_V_Line(dim x_start as byte, dim x_end as byte, dim y_pos as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a horizontal line on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_start</code>: x coordinate of the line start. Valid values: 0..127 - <code>x_end</code>: x coordinate of the line end. Valid values: 0..127 - <code>y_pos</code>: y coordinate of horizontal line. Valid values: 0..63 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the line color: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.
Example	<code>' Draw a horizontal line between dots (10,20) and (50,20)</code> <code>Spi_Glcd_H_Line(10, 50, 20, 1)</code>

Spi_Glcd_Rectangle

Prototype	<code>sub procedure Spi_Glcd_Rectangle(dim x_upper_left as byte, dim y_upper_left as byte, dim x_bottom_right as byte, dim y_bottom_right as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a rectangle on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_upper_left</code>: x coordinate of the upper left rectangle corner. Valid values: 0..127 - <code>y_upper_left</code>: y coordinate of the upper left rectangle corner. Valid values: 0..63 - <code>x_bottom_right</code>: x coordinate of the lower right rectangle corner. Valid values: 0..127 - <code>y_bottom_right</code>: y coordinate of the lower right rectangle corner. Valid values: 0..63 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the rectangle border: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.
Example	<code>' Draw a rectangle between dots (5,5) and (40,40)</code> <code>Spi_Glcd_Rectangle(5, 5, 40, 40, 1)</code>

Spi_Glcd_Box

Prototype	<code>sub procedure Spi_Glcd_Box(dim x_upper_left as byte, dim y_upper_left as byte, dim x_bottom_right as byte, dim y_bottom_right as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a box on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_upper_left</code>: x coordinate of the upper left box corner. Valid values: 0..127 - <code>y_upper_left</code>: y coordinate of the upper left box corner. Valid values: 0..63 - <code>x_bottom_right</code>: x coordinate of the lower right box corner. Valid values: 0..127 - <code>y_bottom_right</code>: y coordinate of the lower right box corner. Valid values: 0..63 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the box fill: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.
Example	<code>' Draw a box between dots (5,15) and (20,40)</code> <code>Spi_Glcd_Box(5, 15, 20, 40, 1)</code>

Spi_Glcd_Circle

Prototype	<code>sub procedure Spi_Glcd_Circle(dim x_center as integer, dim y_center as integer, dim radius as integer, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a circle on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x_center</code>: x coordinate of the circle center. Valid values: 0..127 - <code>y_center</code>: y coordinate of the circle center. Valid values: 0..63 - <code>radius</code>: radius size - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the circle line: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routine.
Example	<code>' Draw a circle with center in (50,50) and radius=10</code> <code>Spi_Glcd_Circle(50, 50, 10, 1)</code>

Spi_Glcd_Set_Font

Prototype	<code>sub procedure SPI_Glcd_Set_Font(dim const activeFont as ^byte, dim aFontWidth as byte, dim aFontHeight as byte, dim aFontOffs as word)</code>
Returns	Nothing.
Description	<p>Sets font that will be used with Spi_Glcd_Write_Char and Spi_Glcd_Write_Text routines.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>activeFont</code>: font to be set. Needs to be formatted as an array of char - <code>aFontWidth</code>: width of the font characters in dots. - <code>aFontHeight</code>: height of the font characters in dots. - <code>aFontOffs</code>: number that represents difference between the mikroBasic character set and regular ASCII set (eg. if 'A' is 65 in ASCII character, and 'A' is 45 in the mikroBasic character set, aFontOffs is 20). Demo fonts supplied with the library have an offset of 32, which means that they start with space. <p>The user can use fonts given in the file “__Lib_GLCD_fonts.mpas” file located in the Uses folder or create his own fonts.</p>
Requires	GLCD needs to be initialized for SPI communication, see Spi_Glcd_Init routines.
Example	<code>' Use the custom 5x7 font "myfont" which starts with space (32): Spi_Glcd_Set_Font(myfont, 5, 7, 32)</code>

Spi_Glcd_Write_Char

Prototype	<code>sub procedure SPI_Glcd_Write_Char(dim chr1 as byte, dim x_pos as byte, dim page_num as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Prints character on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none">- <code>chr1</code>: character to be written- <code>x_pos</code>: character starting position on x-axis. Valid values: 0..(127-FontWidth)- <code>page_num</code>: the number of the page on which character will be written. Valid values: 0..7- <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the character: 0 white, 1 black, and 2 inverts each dot.</p> <p>Note: For x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	<p>GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.</p> <p>Use the <code>Spi_Glcd_Set_Font</code> to specify the font for display; if no font is specified, then the default 5x8 font supplied with the library will be used.</p>
Example	<pre>' Write character 'C' on the position 10 inside the page 2: Spi_Glcd_Write_Char("C", 10, 2, 1)</pre>

Spi_Glcd_Write_Text

Prototype	<code>sub procedure SPI_Glcd_Write_Text(dim byref text as string[20] , dim x_pos as byte, dim page_num as byte, dim color as byte)</code>
Returns	Nothing.
Description	<p>Prints text on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>text</code>: text to be written - <code>x_pos</code>: text starting position on x-axis. - <code>page_num</code>: the number of the page on which text will be written. Valid values: 0..7 - <code>color</code>: color parameter. Valid values: 0..2 <p>The parameter <code>color</code> determines the color of the text: 0 white, 1 black, and 2 inverts each dot.</p> <p>Note: For x axis and page layout explanation see schematic at the bottom of this page.</p>
Requires	<p>GLCD needs to be initialized for SPI communication, see <code>Spi_Glcd_Init</code> routines.</p> <p>Use the <code>Spi_Glcd_Set_Font</code> to specify the font for display; if no font is specified, then the default 5x8 font supplied with the library will be used.</p>
Example	<code>' Write text "Hello world!" on the position 10 inside the page 2: Spi_Glcd_Write_Text("Hello world!", 10, 2, 1)</code>

Spi_Glcd_Image

Prototype	<code>sub procedure Spi_Glcd_Image(dim const image as ^byte)</code>
Returns	Nothing.
Description	<p>Displays bitmap on GLCD.</p> <p>Parameters :</p> <p>- <code>image</code>: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic for 8051 pointer to const and pointer to RAM equivalency).</p> <p>Use the mikroBasic's integrated GLCD Bitmap Editor (menu option Tools > GLCD Bitmap Editor) to convert image to a constant array suitable for displaying on GLCD.</p>
Requires	GLCD needs to be initialized for SPI communication, see Spi_Glcd_Init routines.
Example	<pre>' Draw image my_image on GLCD Spi_Glcd_Image(my_image)</pre>

Library Example

The example demonstrates how to communicate to KS0108 GLCD via the SPI module, using serial to parallel convertor MCP23S17.

```
program SPI_Glcd

include bitmap

' Port Expander module connections
dim SPExpanderRST as sbit at P1.B0
dim SPExpanderCS as sbit at P1.B1
' End Port Expander module connections

dim counter, counter2 as byte
    jj as word
    someText as string[ 20]

sub procedure delay2S
    Delay_ms(2000)
end sub

main:
    Spi_Init_Advanced(MASTER_OSC_DIV4 or CLK_IDLE_LOW or IDLE_2_ACTIVE or
DATA_ORDER_MSB)
    Spi_Glcd_Init(0) ' Initialize GLCD via SPI
    Spi_Glcd_Fill(0x00) ' Clear GLCD
```

```
while TRUE
  Spi_Glcd_Image(@truck_bmp)           ' Draw image
  Delay2S() Delay2S()

  Spi_Glcd_Fill(0x0)
  Delay2S
  Spi_Glcd_Box(62,40,124,56,1)         ' Draw box
  Spi_Glcd_Rectangle(5,5,84,35,1)     ' Draw rectangle
  Spi_Glcd_Line(0, 63, 127, 0,1)      ' Draw line

  Delay2S()
  counter = 5                           ' Draw horizontal
and vertical line
  while counter < 60
    Delay_ms(250)
    Spi_Glcd_V_Line(2, 54, counter, 1)
    Spi_Glcd_H_Line(2, 120, counter, 1)
    counter = counter + 5
  wend

  Delay2S()

  Spi_Glcd_Fill(0x00)

  Spi_Glcd_Set_Font(@Character8x8, 8, 8, 32) ' Choose font
  Spi_Glcd_Write_Text("mikroE", 5, 7, 2)   ' Write string

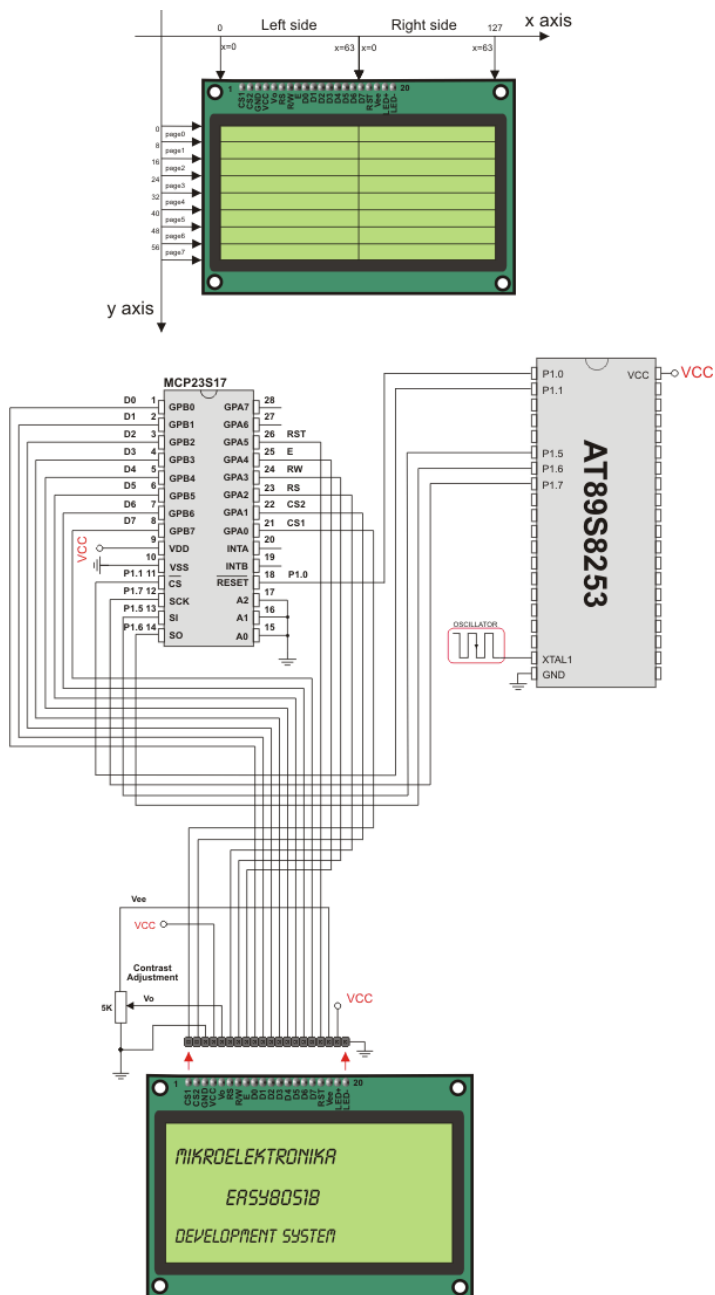
  for counter2 = 1 to 10                 ' Draw circles
    Spi_Glcd_Circle(63,32, 3*counter2, 1)
  next counter2
  Delay2S()

  Spi_Glcd_Box(12,20, 70,63, 2)         ' Draw box
  Delay2S()

  Spi_Glcd_Set_Font(@FontSystem5x8, 5, 8, 32) ' Change font
  someText = "Hello 8051"
  Spi_Glcd_Write_Text(someText, 5, 3, 0) ' Write string
  Delay2S()

  Spi_Glcd_Set_Font(@System3x6, 3, 6, 32)
  someText = "SMALL:NOT:SMALLER"
  Spi_Glcd_Write_Text(someText, 20,5, 1) ' Write string
  Delay2S()
wend
end.
```

HW Connection



SPI GLCD HW connection

SPI LCD LIBRARY

The mikroBasic for 8051 provides a library for communication with LCD (with HD44780 compliant controllers) in 4-bit mode via SPI interface.

For creating a custom set of LCD characters use LCD Custom Character Tool.

Note: The library uses the SPI module for communication. The user must initialize the SPI module before using the SPI LCD Library.

Note: This Library is designed to work with the mikroElektronika's Serial LCD Adapter Board pinout. See schematic at the bottom of this page for details.

External dependencies of SPI LCD Library

The implementation of SPI LCD Library routines is based on Port Expander Library routines.

External dependencies are the same as Port Expander Library external dependencies.

Library Routines

- Spi_Lcd_Config
- Spi_Lcd_Out
- Spi_Lcd_Out_Cp
- Spi_Lcd_Chr
- Spi_Lcd_Chr_Cp
- Spi_Lcd_Cmd

Spi_Lcd_Config

Prototype	<code>sub procedure Spi_Lcd_Config(dim DeviceAddress as byte)</code>
Returns	Nothing.
Description	<p>Initializes the LCD module via SPI interface.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>DeviceAddress</code>: spi expander hardware address, see schematic at the bottom of this page
Requires	<p><code>SPExpanderCS</code> and <code>SPExpanderRST</code> variables must be defined before using this function.</p> <p>The SPI module needs to be initialized. See <code>Spi_Init</code> and <code>Spi_Init_Advanced</code> routines.</p>
Example	<pre>' port expander pinout definition dim SPExpanderCS as sbit at P1.B1 SPExpanderRST as sbit at P1.B0 ... Spi_Init() ' initialize spi Spi_Lcd_Config(0) ' initialize lcd over spi interface</pre>

Spi_Lcd_Out

Prototype	<code>sub procedure Spi_Lcd_Out(dim row as byte, dim column as byte, dim byref text as string[19])</code>
Returns	Nothing.
Description	<p>Prints text on the LCD starting from specified position. Both string variables and literals can be passed as a text.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>row</code>: starting position row number - <code>column</code>: starting position column number - <code>text</code>: text to be written
Requires	LCD needs to be initialized for SPI communication, see <code>Spi_Lcd_Config</code> routines.
Example	<pre>' Write text "Hello!" on LCD starting from row 1, column 3: Spi_Lcd_Out(1, 3, "Hello!")</pre>

Spi_Lcd_Out_Cp

Prototype	<code>sub procedure Spi_Lcd_Out_CP(dim text as string[19])</code>
Returns	Nothing.
Description	Prints text on the LCD at current cursor position. Both string variables and literals can be passed as a text. Parameters : - <code>text</code> : text to be written
Requires	LCD needs to be initialized for SPI communication, see Spi_Lcd_Config routines.
Example	' Write text "Here!" at current cursor position: <code>Spi_Lcd_Out_CP("Here!")</code>

Spi_Lcd_Chr

Prototype	<code>sub procedure Spi_Lcd_Chr(dim Row as byte, dim Column as byte, dim Out_Char as byte)</code>
Returns	Nothing.
Description	Prints character on LCD at specified position. Both variables and literals can be passed as character. Parameters : - <code>Row</code> : writing position row number - <code>Column</code> : writing position column number - <code>Out_Char</code> : character to be written
Requires	LCD needs to be initialized for SPI communication, see Spi_Lcd_Config routines.
Example	' Write character "i" at row 2, column 3: <code>Spi_Lcd_Chr(2, 3, 'i')</code>

Spi_Lcd_Chrcp

Prototype	<code>sub procedure Spi_Lcd_Chrcp(dim Out_Char as byte)</code>
Returns	Nothing.
Description	Prints character on LCD at current cursor position. Both variables and literals can be passed as character. Parameters : - <code>Out_Char</code> : character to be written
Requires	LCD needs to be initialized for SPI communication, see <code>Spi_Lcd_Config</code> routines.
Example	<code>' Write character "e" at current cursor position: Spi_Lcd_Chrcp('e')</code>

Spi_Lcd_Cmd

Prototype	<code>sub procedure Spi_Lcd_Cmd(dim out_char as byte)</code>
Returns	Nothing.
Description	Sends command to LCD. Parameters : - <code>out_char</code> : command to be sent Note: Predefined constants can be passed to the function, see Available LCD Commands.
Requires	LCD needs to be initialized for SPI communication, see <code>Spi_Lcd_Config</code> routines.
Example	<code>' Clear LCD display: Spi_Lcd_Cmd(LCD_CLEAR)</code>

Available LCD Commands

Lcd Command	Purpose
LCD_FIRST_ROW	Move cursor to the 1st row
LCD_SECOND_ROW	Move cursor to the 2nd row
LCD_THIRD_ROW	Move cursor to the 3rd row
LCD_FOURTH_ROW	Move cursor to the 4th row
LCD_CLEAR	Clear display
LCD_RETURN_HOME	Return cursor to home position, returns a shifted display to its original position. Display data RAM is unaffected.
LCD_CURSOR_OFF	Turn off cursor
LCD_UNDERLINE_ON	Underline cursor on
LCD_BLINK_CURSOR_ON	Blink cursor on
LCD_MOVE_CURSOR_LEFT	Move cursor left without changing display data RAM
LCD_MOVE_CURSOR_RIGHT	Move cursor right without changing display data RAM
LCD_TURN_ON	Turn LCD display on
LCD_TURN_OFF	Turn LCD display off
LCD_SHIFT_LEFT	Shift display left without changing display data RAM
LCD_SHIFT_RIGHT	Shift display right without changing display data RAM

Library Example

This example demonstrates how to communicate LCD via the SPI module, using serial to parallel convertor MCP23S17.

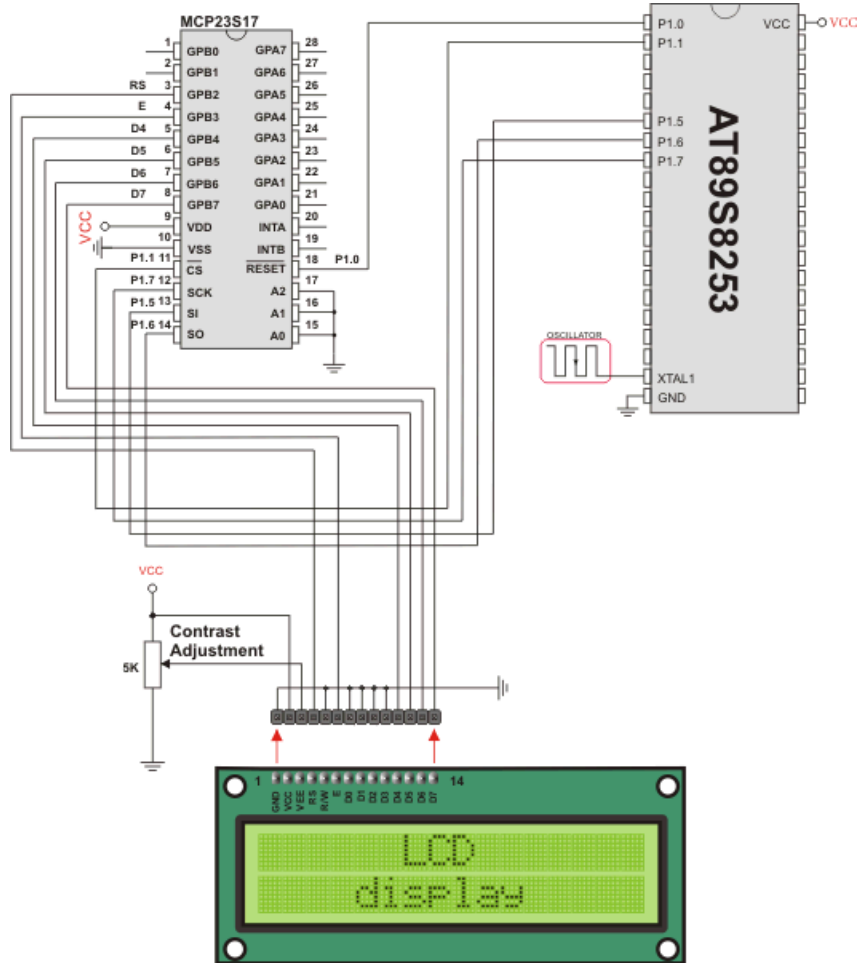
```
program Spi_LCD

dim text as char[17]

' Port Expander module connections
dim SPExpanderRST as sbit at P1.B0
dim SPExpanderCS as sbit at P1.B1
' End Port Expander module connections

main:
    text = "mikroElektronika"
    Spi_Init()                ' Initialize SPI
    Spi_Lcd_Config(0)         ' Initialize LCD over SPI interface
    Spi_Lcd_Cmd(LCD_CLEAR)   ' Clear display
    Spi_Lcd_Cmd(LCD_CURSOR_OFF) ' Turn cursor off
    Spi_Lcd_Out(1,6, "mikroE") ' Print text to LCD, 1st row, 6th column
    Spi_Lcd_Chr_CP("!")      ' Append "!"
    Spi_Lcd_Out(2,1, text)    ' Print text to LCD, 2nd row, 1st column
    Spi_Lcd_Out(3,1,"mikroE") ' For LCD with more than two rows
    Spi_Lcd_Out(4,15,"mikroE") ' For LCD with more than two rows
end.
```

HW Connection



SPI LCD HW connection

SPI LCD8 (8-BIT INTERFACE) LIBRARY

The mikroBasic for 8051 provides a library for communication with LCD (with HD44780 compliant controllers) in 8-bit mode via SPI interface.

For creating a custom set of LCD characters use LCD Custom Character Tool.

Note: Library uses the SPI module for communication. The user must initialize the SPI module before using the SPI LCD Library.

Note: This Library is designed to work with mikroElektronika's Serial LCD/GLCD Adapter Board pinout, see schematic at the bottom of this page for details.

External dependencies of SPI LCD Library

The implementation of SPI LCD Library routines is based on Port Expander Library routines.

External dependencies are the same as Port Expander Library external dependencies.

Library Routines

- Spi_Lcd8_Config
- Spi_Lcd8_Out
- Spi_Lcd8_Out_Cp
- Spi_Lcd8_Chr
- Spi_Lcd8_Chr_Cp
- Spi_Lcd8_Cmd

Spi_Lcd8_Config

Prototype	<code>sub procedure Spi_Lcd8_Config(dim DeviceAddress as byte)</code>
Returns	Nothing.
Description	<p>Initializes the LCD module via SPI interface.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>DeviceAddress</code>: spi expander hardware address, see schematic at the bottom of this page
Requires	<p>SPExpanderCS and SPExpanderRST variables must be defined before using this function.</p> <p>The SPI module needs to be initialized. See Spi_Init and Spi_Init_Advanced routines.</p>
Example	<pre>' port expander pinout definition dim SPExpanderCS as sbit at P1.B1 SPExpanderRST as sbit at P1.B0 ... Spi_Init() ' initialize spi interface Spi_Lcd8_Config(0) ' intialize lcd in 8bit mode via spi</pre>

Spi_Lcd8_Out

Prototype	<code>sub procedure Spi_Lcd8_Out(dim row as byte, dim column as byte, dim byref text as string[19])</code>
Returns	Nothing.
Description	<p>Prints text on LCD starting from specified position. Both string variables and literals can be passed as a text.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>row</code>: starting position row number - <code>column</code>: starting position column number - <code>text</code>: text to be written
Requires	LCD needs to be initialized for SPI communication, see Spi_Lcd8_Config routines.
Example	<pre>' Write text "Hello!" on LCD starting from row 1, column 3: Spi_Lcd8_Out(1, 3, "Hello!")</pre>

Spi_Lcd8_Out_Cp

Prototype	<code>sub procedure Spi_Lcd8_Out_CP(dim text as string[19])</code>
Returns	Nothing.
Description	Prints text on LCD at current cursor position. Both string variables and literals can be passed as a text. Parameters : - <code>text</code> : text to be written
Requires	LCD needs to be initialized for SPI communication, see Spi_Lcd8_Config routines.
Example	' Write text "Here!" at current cursor position: <code>Spi_Lcd8_Out_CP("Here!")</code>

Spi_Lcd8_Chr

Prototype	<code>sub procedure Spi_Lcd8_Chr(dim Row as byte, dim Column as byte, dim Out_Char as byte)</code>
Returns	Nothing.
Description	Prints character on LCD at specified position. Both variables and literals can be passed as character. Parameters : - <code>row</code> : writing position row number - <code>column</code> : writing position column number - <code>out_char</code> : character to be written
Requires	LCD needs to be initialized for SPI communication, see Spi_Lcd8_Config routines.
Example	' Write character "i" at row 2, column 3: <code>Spi_Lcd8_Chr(2, 3, 'i')</code>

Spi_Lcd8_Chrcp

Prototype	<code>sub procedure Spi_Lcd8_Chrcp(dim Out_Char as byte)</code>
Returns	Nothing.
Description	Prints character on LCD at current cursor position. Both variables and literals can be passed as character. Parameters : - <code>out_char</code> : character to be written
Requires	LCD needs to be initialized for SPI communication, see Spi_Lcd8_Config routines.
Example	Print "e" at current cursor position: <code>' Write character "e" at current cursor position: Spi_Lcd8_Chrcp('e')</code>

Spi_Lcd8_Cmd

Prototype	<code>sub procedure Spi_Lcd8_Cmd(dim out_char as byte)</code>
Returns	Nothing.
Description	Sends command to LCD. Parameters : - <code>out_char</code> : command to be sent Note: Predefined constants can be passed to the function, see Available LCD Commands.
Requires	LCD needs to be initialized for SPI communication, see Spi_Lcd8_Config routines.
Example	<code>' Clear LCD display: Spi_Lcd8_Cmd(LCD_CLEAR)</code>

Available LCD Commands

Lcd Command	Purpose
LCD_FIRST_ROW	Move cursor to the 1st row
LCD_SECOND_ROW	Move cursor to the 2nd row
LCD_THIRD_ROW	Move cursor to the 3rd row
LCD_FOURTH_ROW	Move cursor to the 4th row
LCD_CLEAR	Clear display
LCD_RETURN_HOME	Return cursor to home position, returns a shifted display to its original position. Display data RAM is unaffected.
LCD_CURSOR_OFF	Turn off cursor
LCD_UNDERLINE_ON	Underline cursor on
LCD_BLINK_CURSOR_ON	Blink cursor on
LCD_MOVE_CURSOR_LEFT	Move cursor left without changing display data RAM
LCD_MOVE_CURSOR_RIGHT	Move cursor right without changing display data RAM
LCD_TURN_ON	Turn LCD display on
LCD_TURN_OFF	Turn LCD display off
LCD_SHIFT_LEFT	Shift display left without changing display data RAM
LCD_SHIFT_RIGHT	Shift display right without changing display data RAM

Library Example

This example demonstrates how to communicate LCD in 8-bit mode via the SPI module, using serial to parallel convertor MCP23S17.

```

program SPI_Lcd8

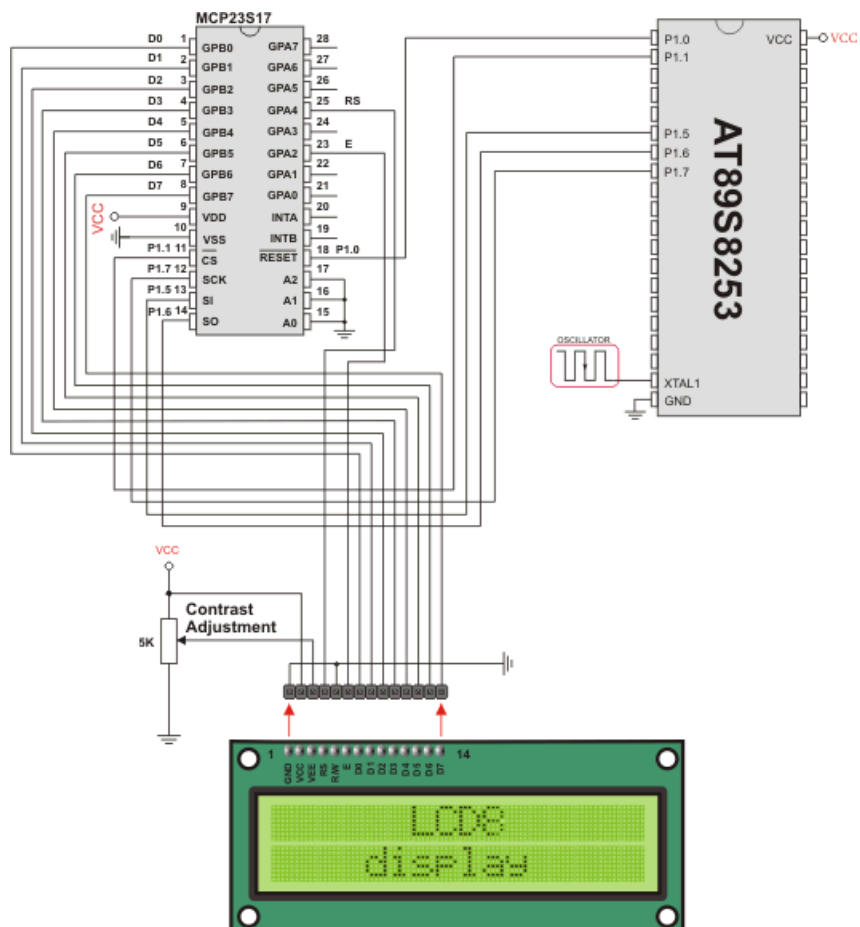
dim text as char[16]

' Port Expander module connections
dim SPExpanderRST as sbit at P1.B0
dim SPExpanderCS as sbit at P1.B1
' End Port Expander module connections

main:
    text = "mikroE"
    Spi_Init()                                ' Initialize SPI inter-
face
    Spi_Lcd8_Config(0)                        ' Intialize LCD in
8bit mode via SPI
    Spi_Lcd8_Cmd(LCD_CLEAR)                  ' Clear display
    Spi_Lcd8_Cmd(LCD_CURSOR_OFF)           ' Turn cursor off
    Spi_Lcd8_Out(1,6, text)                  ' Print text to LCD,
1st row, 6th column...
    Spi_Lcd8_Chr_CP("!")                    ' Append "!"
    Spi_Lcd8_Out(2,1, "mikroelektronika")  ' Print text to LCD,
2nd row, 1st column...
    Spi_Lcd8_Out(3,1, text)                  ' For LCD modules with
more than two rows
    Spi_Lcd8_Out(4,15, text)                ' For LCD modules with
more than two rows
end.

```

HW Connection



SPI LCD8 HW connection

SPI T6963C GRAPHIC LCD LIBRARY

The mikroBasic for 8051 provides a library for working with GLCDs based on TOSHIBA T6963C controller via SPI interface. The Toshiba T6963C is a very popular LCD controller for the use in small graphics modules. It is capable of controlling displays with a resolution up to 240x128. Because of its low power and small outline it is most suitable for mobile applications such as PDAs, MP3 players or mobile measurement equipment. Although this controller is small, it has a capability of displaying and merging text and graphics and it manages all interfacing signals to the displays Row and Column drivers.

For creating a custom set of GLCD images use GLCD Bitmap Editor Tool.

Note: The library uses the SPI module for communication. The user must initialize SPI module before using the Spi T6963C GLCD Library.

Note: This Library is designed to work with mikroElektronika's Serial GLCD 240x128 and 240x64 Adapter Boards pinout, see schematic at the bottom of this page for details.

Note: Some mikroElektronika's adapter boards have pinout different from T6369C datasheets. Appropriate relations between these labels are given in the table below:

Adapter Board	T6369C datasheet
RS	C/D
R/W	/RD
E	/WR

External dependencies of Spi T6963C Graphic LCD Library

The implementation of Spi T6963C Graphic LCD Library routines is based on Port Expander Library routines.

External dependencies are the same as Port Expander Library external dependencies.

Library Routines

- Spi_T6963C_Config
- Spi_T6963C_WriteData
- Spi_T6963C_WriteCommand
- Spi_T6963C_SetPtr
- Spi_T6963C_WaitReady
- Spi_T6963C_Fill
- Spi_T6963C_Dot
- Spi_T6963C_Write_Char
- Spi_T6963C_Write_Text
- Spi_T6963C_Line
- Spi_T6963C_Rectangle
- Spi_T6963C_Box
- Spi_T6963C_Circle
- Spi_T6963C_Image
- Spi_T6963C_Sprite
- Spi_T6963C_Set_Cursor

Note: The following low level library routines are implemented as macros. These macros can be found in the [Spi_T6963C.h](#) header file which is located in the SPI T6963C example projects folders.

- Spi_T6963C_ClearBit
- Spi_T6963C_SetBit
- Spi_T6963C_NegBit
- Spi_T6963C_DisplayGrPanel
- Spi_T6963C_DisplayTxtPanel
- Spi_T6963C_SetGrPanel
- Spi_T6963C_SetTxtPanel
- Spi_T6963C_PanelFill
- Spi_T6963C_GrFill
- Spi_T6963C_TxtFill
- Spi_T6963C_Cursor_Height
- Spi_T6963C_Graphics
- Spi_T6963C_Text
- Spi_T6963C_Cursor
- Spi_T6963C_Cursor_Blink

Spi_T6963C_Config

Prototype	<code>sub procedure Spi_T6963C_Config(dim width as word, dim height as byte, dim fntW as byte, dim DeviceAddress as byte, dim wr as byte, dim rd as byte, dim cd as byte, dim rst as byte)</code>
Returns	Nothing.
Description	<p>Initializes the Graphic Lcd controller.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>width</code>: width of the GLCD panel - <code>height</code>: height of the GLCD panel - <code>fntW</code>: font width - <code>DeviceAddress</code>: SPI expander hardware address, see schematic at the bottom of this page - <code>wr</code>: write signal pin on GLCD control port - <code>rd</code>: read signal pin on GLCD control port - <code>cd</code>: command/data signal pin on GLCD control port - <code>rst</code>: reset signal pin on GLCD control port <p>Display RAM organization: The library cuts RAM into panels : a complete panel is one graphics panel followed by a text panel (see schematic below).</p> <pre>schematic: +-----+ /\ + GRAPHICS PANEL #0 + + + + + + + +-----+ PANEL 0 + TEXT PANEL #0 + + + \ +-----+ /\ + GRAPHICS PANEL #1 + + + + + + + +-----+ PANEL 1 + TEXT PANEL #2 + + + +-----+ \</pre>

Requires	<p><code>SPExpanderCS</code> and <code>SPExpanderRST</code> variables must be defined before using this function.</p> <p>The SPI module needs to be initialized. See the <code>Spi_Init</code> and <code>Spi_Init_Advanced</code> routines.</p>
Example	<pre>' port expander pinout definition dim SPExpanderRST as sbit at P1.B0 dim SPExpanderCS sbit at P1.B1 ... Spi_Init_Advanced(MASTER_OSC_DIV4 OR CLK_IDLE_LOW OR IDLE_2_ACTIVE OR DATA_ORDER_MSB) Spi_T6963C_Config(240, 64, 8, 0, 0, 1, 3, 4)</pre>

Spi_T6963C_WriteData

Prototype	<code>sub procedure Spi_T6963C_WriteData(dim Ddata as byte)</code>
Returns	Nothing.
Description	<p>Writes data to T6963C controller via SPI interface.</p> <p>Parameters :</p> <p>- <code>Ddata</code>: data to be written</p>
Requires	Toshiba GLCD module needs to be initialized. See <code>Spi_T6963C_Config</code> routine.
Example	<code>Spi_T6963C_WriteData(AddrL)</code>

Spi_T6963C_WriteCommand

Prototype	<code>sub procedure Spi_T6963C_WriteCommand(dim Ddata as byte)</code>
Returns	Nothing.
Description	<p>Writes command to T6963C controller via SPI interface.</p> <p>Parameters :</p> <p>- <code>Ddata</code>: command to be written</p>
Requires	Toshiba GLCD module needs to be initialized. See <code>Spi_T6963C_Config</code> routine.
Example	<code>Spi_T6963C_WriteCommand(Spi_T6963C_CURSOR_POINTER_SET)</code>

Spi_T6963C_SetPtr

Prototype	<code>sub procedure Spi_T6963C_SetPtr(dim p as word, dim c as byte)</code>
Returns	Nothing.
Description	Sets the memory pointer p for command c. Parameters : - p: address where command should be written - c: command to be written
Requires	SToshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_SetPtr(T6963C_grHomeAddr + start, T6963C_ADDRESS_POINTER_SET)</code>

Spi_T6963C_WaitReady

Prototype	<code>sub procedure Spi_T6963C_WaitReady()</code>
Returns	Nothing.
Description	Pools the status byte, and loops until Toshiba GLCD module is ready.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_WaitReady()</code>

Spi_T6963C_Fill

Prototype	<code>sub procedure Spi_T6963C_Fill(dim v as byte, dim start as word, dim len as word)</code>
Returns	Nothing.
Description	Fills controller memory block with given byte. Parameters : - v: byte to be written - start: starting address of the memory block - len: length of the memory block in bytes
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_Fill(0x33, 0x00FF, 0x000F)</code>

Spi_T6963C_Dot

Prototype	<code>sub procedure Spi_T6963C_Dot(dim x as integer, dim y as integer, dim color as byte)</code>
Returns	Nothing.
Description	<p>Draws a dot in the current graphic panel of GLCD at coordinates (x, y).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x</code>: dot position on x-axis - <code>y</code>: dot position on y-axis - <code>color</code>: color parameter. Valid values: Spi_T6963C_BLACK and Spi_T6963C_WHITE
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_Dot(x0, y0, pcolor)</code>

Spi_T6963C_Write_Char

Prototype	<code>sub procedure Spi_T6963C_Write_Char(dim c as byte, dim x as byte, dim y as byte, dim mode as byte)</code>
Returns	Nothing.
Description	<p>Writes a char in the current text panel of GLCD at coordinates (x, y).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>c</code>: char to be written - <code>x</code>: char position on x-axis - <code>y</code>: char position on y-axis - <code>mode</code>: mode parameter. Valid values: Spi_T6963C_ROM_MODE_OR, Spi_T6963C_ROM_MODE_XOR, Spi_T6963C_ROM_MODE_AND and Spi_T6963C_ROM_MODE_TEXT <p>Mode parameter explanation:</p> <ul style="list-style-type: none"> - OR Mode: In the OR-Mode, text and graphics can be displayed and the data is logically "OR-ed". This is the most common way of combining text and graphics for example labels on buttons. - XOR-Mode: In this mode, the text and graphics data are combined via the logical "exclusive OR". This can be useful to display text in negative mode, i.e. white text on black background. - AND-Mode: The text and graphic data shown on display are combined via the logical "AND function". - TEXT-Mode: This option is only available when displaying just a text. The Text Attribute values are stored in the graphic area of display memory. <p>For more details see the T6963C datasheet.</p>
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_Write_Char("A",22,23,AND)</code>

Spi_T6963C_Write_Text

Prototype	<code>sub procedure Spi_T6963C_Write_Text(dim str as ^byte, dim x as byte, dim y as byte, dim mode as byte)</code>
Returns	Nothing.
Description	<p>Writes text in the current text panel of GLCD at coordinates (x, y).</p> <p>Parameters :</p> <ul style="list-style-type: none">- <code>str</code>: text to be written- <code>x</code>: text position on x-axis- <code>y</code>: text position on y-axis- <code>mode</code>: mode parameter. Valid values: Spi_T6963C_ROM_MODE_OR, Spi_T6963C_ROM_MODE_XOR, Spi_T6963C_ROM_MODE_AND and Spi_T6963C_ROM_MODE_TEXT <p>Mode parameter explanation:</p> <ul style="list-style-type: none">- OR Mode: In the OR-Mode, text and graphics can be displayed and the data is logically "OR-ed". This is the most common way of combining text and graphics for example labels on buttons.- XOR-Mode: In this mode, the text and graphics data are combined via the logical "exclusive OR". This can be useful to display text in negative mode, i.e. white text on black background.- AND-Mode: The text and graphic data shown on the display are combined via the logical "AND function".- TEXT-Mode: This option is only available when displaying just a text. The Text Attribute values are stored in the graphic area of display memory. <p>For more details see the T6963C datasheet.</p>
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_Write_Text("GLCD LIBRARY DEMO, WELCOME !", 0, 0, T6963C_ROM_MODE_EXOR)</code>

Spi_T6963C_Line

Prototype	<code>sub procedure Spi_T6963C_Line(dim x0 as integer, dim y0 as integer, dim x1 as integer, dim y1 as integer, dim pcolor as byte)</code>
Returns	Nothing.
Description	<p>Draws a line from (x0, y0) to (x1, y1).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x0</code>: x coordinate of the line start - <code>y0</code>: y coordinate of the line end - <code>x1</code>: x coordinate of the line start - <code>y1</code>: y coordinate of the line end - <code>pcolor</code>: color parameter. Valid values: <code>Spi_T6963C_BLACK</code> and <code>Spi_T6963C_WHITE</code>
Requires	Toshiba GLCD module needs to be initialized. See <code>Spi_T6963C_Config</code> routine.
Example	<code>Spi_T6963C_Line(0, 0, 239, 127, T6963C_WHITE)</code>

Spi_T6963C_Rectangle

Prototype	<code>sub procedure Spi_T6963C_Rectangle(dim x0 as integer, dim y0 as integer, dim x1 as integer, dim y1 as integer, dim pcolor as byte)</code>
Returns	Nothing.
Description	<p>Draws a rectangle on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x0</code>: x coordinate of the upper left rectangle corner - <code>y0</code>: y coordinate of the upper left rectangle corner - <code>x1</code>: x coordinate of the lower right rectangle corner - <code>y1</code>: y coordinate of the lower right rectangle corner - <code>pcolor</code>: color parameter. Valid values: <code>Spi_T6963C_BLACK</code> and <code>Spi_T6963C_WHITE</code>
Requires	Toshiba GLCD module needs to be initialized. See <code>Spi_T6963C_Config</code> routine.
Example	<code>Spi_T6963C_Rectangle(20, 20, 219, 107, T6963C_WHITE)</code>

Spi_T6963C_Box

Prototype	<code>sub procedure Spi_T6963C_Box(dim x0 as integer, dim y0 as integer, dim x1 as integer, dim y1 as integer, dim pcolor as byte)</code>
Returns	Nothing.
Description	<p>Draws a box on the GLCD</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x0</code>: x coordinate of the upper left box corner - <code>y0</code>: y coordinate of the upper left box corner - <code>x1</code>: x coordinate of the lower right box corner - <code>y1</code>: y coordinate of the lower right box corner - <code>pcolor</code>: color parameter. Valid values: <code>Spi_T6963C_BLACK</code> and <code>Spi_T6963C_WHITE</code>
Requires	Toshiba GLCD module needs to be initialized. See <code>Spi_T6963C_Config</code> routine.
Example	<code>Spi_T6963C_Box(0, 119, 239, 127, T6963C_WHITE)</code>

Spi_T6963C_Circle

Prototype	<code>sub procedure Spi_T6963C_Circle(dim x as integer, dim y as integer, dim r as longint, dim pcolor as byte)</code>
Returns	Nothing.
Description	<p>Draws a circle on the GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x</code>: x coordinate of the circle center - <code>y</code>: y coordinate of the circle center - <code>r</code>: radius size - <code>pcolor</code>: color parameter. Valid values: <code>Spi_T6963C_BLACK</code> and <code>Spi_T6963C_WHITE</code>
Requires	Toshiba GLCD module needs to be initialized. See <code>Spi_T6963C_Config</code> routine.
Example	<code>Spi_T6963C_Circle(120, 64, 110, T6963C_WHITE)</code>

Spi_T6963C_Image

Prototype	<code>sub procedure Spi_T6963C_image(const pic as ^byte)</code>
Returns	Nothing.
Description	<p>Displays bitmap on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>pic</code>: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic for 8051 pointer to const and pointer to RAM equivalency). <p>Use the mikroBasic's integrated GLCD Bitmap Editor (menu option Tools › GLCD Bitmap Editor) to convert image to a constant array suitable for displaying on GLCD.</p>
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_Image(my_image)</code>

Spi_T6963C_Sprite

Prototype	<code>sub procedure Spi_T6963C_sprite(dim px, py, sx, sy as byte, const pic as ^byte)</code>
Returns	Nothing.
Description	<p>Fills graphic rectangle area (px, py) to (px+sx, py+sy) with custom size picture.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>px</code>: x coordinate of the upper left picture corner. Valid values: multiples of the font width - <code>py</code>: y coordinate of the upper left picture corner - <code>pic</code>: picture to be displayed - <code>sx</code>: picture width. Valid values: multiples of the font width - <code>sy</code>: picture height <p>Note: If <code>px</code> and <code>sx</code> parameters are not multiples of the font width they will be scaled to the nearest lower number that is a multiple of the font width.</p>
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_Sprite(76, 4, einstein, 88, 119) ' draw a sprite</code>

Spi_T6963C_Set_Cursor

Prototype	<code>sub procedure Spi_T6963C_set_cursor(dim x, y as byte)</code>
Returns	Nothing.
Description	<p>Sets cursor to row x and column y.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - x: cursor position row number - y: cursor position column number
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_Set_Cursor(cposx, cposy)</code>

Spi_T6963C_ClearBit

Prototype	<code>sub procedure Spi_T6963C_clearBit(dim b as byte)</code>
Returns	Nothing.
Description	<p>Clears control port bit(s).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - b: bit mask. The function will clear bit x on control port if bit x in bit mask is set to 1.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>' clear bits 0 and 1 on control port Spi_T6963C_ClearBit(0x03)</code>

Spi_T6963C_SetBit

Prototype	<code>sub procedure Spi_T6963C_setBit(dim b as byte)</code>
Returns	Nothing.
Description	<p>Sets control port bit(s).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - b: bit mask. The function will set bit x on control port if bit x in bit mask is set to 1.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>' set bits 0 and 1 on control port Spi_T6963C_SetBit(0x03)</code>

Spi_T6963C_NegBit

Prototype	<code>sub procedure Spi_T6963C_negBit(dim b as byte)</code>
Returns	Nothing.
Description	Negates control port bit(s). Parameters : - b : bit mask. The function will negate bit x on control port if bit x in bit mask is set to 1.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<pre>' negate bits 0 and 1 on control port Spi_T6963C_NegBit(0x03)</pre>

Spi_T6963C_DisplayGrPanel

Prototype	<code>sub procedure Spi_T6963C_DisplayGrPanel(dim n as byte)</code>
Returns	Nothing.
Description	Display selected graphic panel. Parameters : - n : graphic panel number. Valid values: 0 and 1.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<pre>' display graphic panel 1 Spi_T6963C_DisplayGrPanel(1)</pre>

Spi_T6963C_DisplayTxtPanel

Prototype	<code>sub procedure Spi_T6963C_DisplayTxtPanel(dim n as byte)</code>
Returns	Nothing.
Description	Display selected text panel. Parameters : - n : text panel number. Valid values: 0 and 1.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<pre>' display text panel 1 Spi_T6963C_DisplayTxtPanel(1)</pre>

Spi_T6963C_SetGrPanel

Prototype	<code>sub procedure Spi_T6963C_SetGrPanel(dim n as byte)</code>
Returns	Nothing.
Description	Compute start address for selected graphic panel and set appropriate internal pointers. All subsequent graphic operations will be preformed at this graphic panel. Parameters : - <i>n</i> : graphic panel number. Valid values: 0 and 1.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<pre>' set graphic panel 1 as current graphic panel. Spi_T6963C_SetGrPanel(1)</pre>

Spi_T6963C_SetTxtPanel

Prototype	<code>sub procedure Spi_T6963C_SetTxtPanel(dim n as byte)</code>
Returns	Nothing.
Description	Compute start address for selected text panel and set appropriate internal pointers. All subsequent text operations will be preformed at this text panel. Parameters : - <i>n</i> : text panel number. Valid values: 0 and 1.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<pre>' set text panel 1 as current text panel. Spi_T6963C_SetTxtPanel(1)</pre>

Spi_T6963C_PanelFill

Prototype	<code>sub procedure Spi_T6963C_PanelFill(dim v as byte)</code>
Returns	Nothing.
Description	Fill current panel in full (graphic+text) with appropriate value (0 to clear). Parameters : - <code>v</code> : value to fill panel with.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<pre>clear current panel Spi_T6963C_PanelFill(0)</pre>

Spi_T6963C_GrFill

Prototype	<code>sub procedure Spi_T6963C_GrFill(dim v as byte)</code>
Returns	Nothing.
Description	Fill current graphic panel with appropriate value (0 to clear). Parameters : - <code>v</code> : value to fill graphic panel with.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<pre>' clear current graphic panel Spi_T6963C_GrFill(0)</pre>

Spi_T6963C_TxtFill

Prototype	<code>sub procedure Spi_T6963C_TxtFill(dim v as byte)</code>
Returns	Nothing.
Description	Fill current text panel with appropriate value (0 to clear). Parameters : - <code>v</code> : this value increased by 32 will be used to fill text panel.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<pre>' clear current text panel Spi_T6963C_TxtFill(0)</pre>

Spi_T6963C_Cursor_Height

Prototype	<code>sub procedure Spi_T6963C_Cursor_Height(dim n as byte)</code>
Returns	Nothing.
Description	Set cursor size. Parameters : - <code>n</code> : cursor height. Valid values: 0..7.
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>Spi_T6963C_Cursor_Height(7)</code>

Spi_T6963C_Graphics

Prototype	<code>sub procedure Spi_T6963C_Graphics(dim n as byte)</code>
Returns	Nothing.
Description	Enable/disable graphic displaying. Parameters : - <code>n</code> : graphic enable/disable parameter. Valid values: 0 (disable graphic displaying) and 1 (enable graphic displaying).
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>enable graphic displaying</code> <code>Spi_T6963C_Graphics(1)</code>

Spi_T6963C_Text

Prototype	<code>sub procedure Spi_T6963C_Text(dim n as byte)</code>
Returns	Nothing.
Description	Enable/disable text displaying. Parameters : - <code>n</code> : text enable/disable parameter. Valid values: 0 (disable text displaying) and 1 (enable text displaying).
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>' enable text displaying</code> <code>Spi_T6963C_Text(1)</code>

Spi_T6963C_Cursor

Prototype	<code>sub procedure Spi_T6963C_Cursor(dim n as byte)</code>
Returns	Nothing.
Description	Set cursor on/off. Parameters : - <i>n</i> : on/off parameter. Valid values: 0 (set cursor off) and 1 (set cursor on).
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>' set cursor on Spi_T6963C_Cursor(1)</code>

Spi_T6963C_Cursor_Blink

Prototype	<code>sub procedure Spi_T6963C_Cursor_Blink(dim n as byte)</code>
Returns	Nothing.
Description	Enable/disable cursor blinking. Parameters : - <i>n</i> : cursor blinking enable/disable parameter. Valid values: 0 (disable cursor blinking) and 1 (enable cursor blinking).
Requires	Toshiba GLCD module needs to be initialized. See Spi_T6963C_Config routine.
Example	<code>' enable cursor blinking Spi_T6963C_Cursor_Blink(1)</code>

Library Example

The following drawing demo tests advanced routines of the Spi T6963C GLCD library. Hardware configurations in this example are made for the T6963C 240x128 display, Easy8051B board and AT89S8253.

```
program SPI_T6963C_240x128

include __Lib_T6963C_Consts
include bitmap
include bitmap2

' Port Expander module connections
dim SPExpanderRST as sbit at P1.B0
dim SPExpanderCS  as sbit at P1.B1
' End Port Expander module connections
```

```

dim   panel as byte           ' current panel
      i as word               ' general purpose register
      curs as byte           ' cursor visibility
      cposx, cposy as word    ' cursor x-y position
      txtcols as byte        ' number of text coloms
      txt, txt1 as string[ 29] idata

txt1 = " EINSTEIN WOULD HAVE LIKED mC"
txt  = " GLCD LIBRARY DEMO, WELCOME !"

P0 = 255 ' all inputs

' * init display for 240 pixel width and 128 pixel height
' * 8 bits character width
' * data bus on MCP23S17 portB
' * control bus on MCP23S17 portA
' * bit 2 is !WR
' * bit 1 is !RD
' * bit 0 is !CD
' * bit 4 is RST
' *
' * chip enable, reverse on, 8x8 font internally set in library

' Initialize SPI module
      Spi_Init_Advanced(MASTER_OSC_DIV4 or CLK_IDLE_LOW or
IDLE_2_ACTIVE or DATA_ORDER_MSB)
' Initialize SPI Toshiba 240x128
Spi_T6963C_Config(240, 128, 8, 0, 2, 1, 0, 4)
Delay_ms(1000)

' *
' * Enable both graphics and text display at the same time
' *

Spi_T6963C_graphics(1)
Spi_T6963C_text(1)

panel = 0
i = 0
curs = 0
cpoxy = 0
cposx = 0

' *
' * Text messages
' *

Spi_T6963C_write_text(txt, 0, 0, T6963C_ROM_MODE_XOR)
Spi_T6963C_write_text(txt1, 0, 15, T6963C_ROM_MODE_XOR)

```

```

'   *
'   * Cursor
'   *

Spi_T6963C_cursor_height(8)           ' 8 pixel height
Spi_T6963C_set_cursor(0, 0)           ' Move cursor to top left
Spi_T6963C_cursor(0)                   ' Cursor off

'   *
'   * Draw rectangles
'   *

Spi_T6963C_rectangle( 0,  0, 239, 127, T6963C_WHITE)
Spi_T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE)
Spi_T6963C_rectangle(40, 40, 199,  87, T6963C_WHITE)
Spi_T6963C_rectangle(60, 60, 179,  67, T6963C_WHITE)

'   *
'   * Draw a cross
'   *

Spi_T6963C_line(0, 0, 239, 127, T6963C_WHITE)
Spi_T6963C_line(0, 127, 239, 0, T6963C_WHITE)

'   *
'   * Draw solid boxes
'   *

Spi_T6963C_box(0, 0, 239, 8, T6963C_WHITE)
Spi_T6963C_box(0, 119, 239, 127, T6963C_WHITE)

'   *
'   * Draw circles
'   *

Spi_T6963C_circle(120, 64, 10, T6963C_WHITE)
Spi_T6963C_circle(120, 64, 30, T6963C_WHITE)
Spi_T6963C_circle(120, 64, 50, T6963C_WHITE)
Spi_T6963C_circle(120, 64, 70, T6963C_WHITE)
Spi_T6963C_circle(120, 64, 90, T6963C_WHITE)
Spi_T6963C_circle(120, 64, 110, T6963C_WHITE)
Spi_T6963C_circle(120, 64, 130, T6963C_WHITE)

Spi_T6963C_sprite(76, 4, @einstein, 88, 119)  ' Draw a sprite

Spi_T6963C_setGrPanel(1)                     ' Select other graphic panel

Spi_T6963C_image(@banner_bmp)                 ' Fill the graphic
screen with a picture

```

```

while TRUE                                     ' Endless loop

'      *
'      * If P0_0 is pressed, toggle the display between graphic
panel 0 and graphic 1
'      *

    if ( P0_0 = 0) then
        Inc(panel)
        panel = panel and 1
        Spi_T6963C_displayGrPanel(panel)
        Delay_ms(300)

'      *
'      * If P0_1 is pressed, display only graphic panel
'      *

    else
    if ( P0_1 = 0) then
        Spi_T6963C_graphics(1)
        Spi_T6963C_text(0)
        Delay_ms(300)

'      *
'      * If P0_2 is pressed, display only text panel
'      *

    else
    if ( P0_2 = 0) then
        Spi_T6963C_graphics(0)
        Spi_T6963C_text(1)
        Delay_ms(300)

'      *
'      * If P0_3 is pressed, display text and graphic panels
'      *

    else
    if( P0_3 = 0) then
        Spi_T6963C_graphics(1)
        Spi_T6963C_text(1)
        Delay_ms(300)

'      *
'      * If P0_4 is pressed, change cursor
'      *

```

```

else
    if ( P0_4 = 0) then
        Inc(curs)
        if (curs = 3) then
            curs = 0
        end if

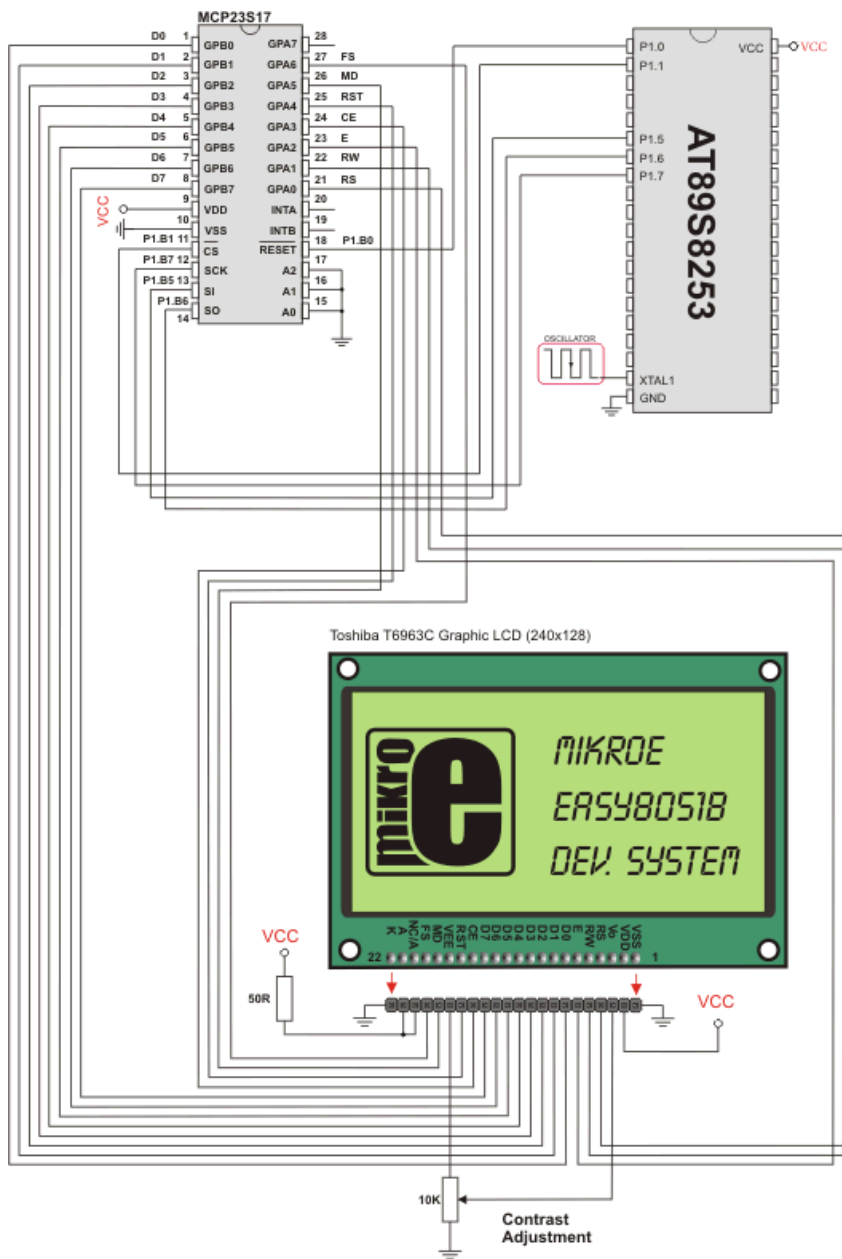
        select case curs
            case 0
                ' No cursor
                Spi_T6963C_cursor(0)
            case 1
                ' Blinking cursor
                Spi_T6963C_cursor(1)
                Spi_T6963C_cursor_blink(1)
            case 2
                ' Non blinking cursor
                Spi_T6963C_cursor(1)
                Spi_T6963C_cursor_blink(0)
        end select 'end case
    end if
    Delay_ms(300)
end if
end if
end if
end if

'      *
'      * Move cursor, even if not visible
'      *

Inc(cposx)
if (cpox = txtcols) then
    cposx = 0
    cposy = cposy + 1
    if (cpoxy = (128 div T6963C_CHARACTER_HEIGHT)) then
        cposy = 0
    end if
    Spi_T6963C_set_cursor(cposx, cposy)
    Delay_ms(100)
end if
wend
end.

```

HW Connection



Spi T6963C GLCD HW connection

T6963C GRAPHIC LCD LIBRARY

The mikroBasic for 8051 provides a library for working with GLCDs based on TOSHIBA T6963C controller. The Toshiba T6963C is a very popular LCD controller for the use in small graphics modules. It is capable of controlling displays with a resolution up to 240x128. Because of its low power and small outline it is most suitable for mobile applications such as PDAs, MP3 players or mobile measurement equipment. Although small, this controller has a capability of displaying and merging text and graphics and it manages all the interfacing signals to the displays Row and Column drivers.

For creating a custom set of GLCD images use GLCD Bitmap Editor Tool.

Note: ChipEnable(CE), FontSelect(FS) and Reverse(MD) have to be set to appropriate levels by the user outside of the T6963C_Init function. See the Library Example code at the bottom of this page.

Note: Some mikroElektronika's adapter boards have pinout different from T6369C datasheets. Appropriate relations between these labels are given in the table below:

Adapter Board	T6369C datasheet
RS	C/D
R/W	/RD
E	/WR

External dependencies of T6963C Graphic LCD Library

The following variables must be defined in all projects using T6963C Graphic LCD library:	Description:	Example :
<code>dim T6963C_dataPort as byte external sfr</code>	T6963C Data Port.	<code>dim T6963C_dataPort as byte at P0 sfr</code>
<code>dim T6963C_ctrlPort as byte external sfr</code>	T6963C Control Port.	<code>dim T6963C_ctrlPort as byte at P1 sfr</code>
<code>dim T6963C_ctrlwr as sbit external</code>	Write signal.	<code>dim T6963C_ctrlwr as sbit at P1.B2</code>
<code>dim T6963C_ctrlrd as sbit external</code>	Read signal.	<code>dim T6963C_ctrlrd as sbit at P1.B1</code>
<code>dim T6963C_ctrlcd as sbit external</code>	Command/Data signal.	<code>dim T6963C_ctrlcd as sbit at P1.B0</code>
<code>dim T6963C_ctrlrst as sbit external</code>	Reset signal.	<code>dim T6963C_ctrlrst as sbit at P1.B4</code>

Library Routines

- T6963C_Init
- T6963C_WriteData
- T6963C_WriteCommand
- T6963C_SetPtr
- T6963C_WaitReady
- T6963C_Fill
- T6963C_Dot
- T6963C_Write_Char
- T6963C_Write_Text
- T6963C_Line
- T6963C_Rectangle
- T6963C_Box
- T6963C_Circle
- T6963C_Image
- T6963C_Sprite
- T6963C_Set_Cursor

Note: The following low level library routines are implemented as macros. These macros can be found in the `T6963C.h` header file which is located in the T6963C example projects folders.

- T6963C_ClearBit
- T6963C_SetBit
- T6963C_NegBit
- T6963C_DisplayGrPanel
- T6963C_DisplayTxtPanel
- T6963C_SetGrPanel
- T6963C_SetTxtPanel
- T6963C_PanelFill
- T6963C_GrFill
- T6963C_TxtFill
- T6963C_Cursor_Height
- T6963C_Graphics
- T6963C_Text
- T6963C_Cursor
- T6963C_Cursor_Blink

T6963C_Init

Prototype	<code>sub procedure T6963C_init(dim width as word, dim height, fntW as byte)</code>
Returns	Nothing.
Description	<p>Initializes the Graphic Lcd controller.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>width</code>: width of the GLCD panel - <code>height</code>: height of the GLCD panel - <code>fntW</code>: font width <p>Display RAM organization: The library cuts the RAM into panels : a complete panel is one graphics panel followed by a text panel (see schematic below).</p> <p>schematic:</p> <pre> +-----+ \ + GRAPHICS PANEL #0 + + + + + + + +-----+ PANEL 0 + TEXT PANEL #0 + + + \ +-----+ \ + GRAPHICS PANEL #1 + + + + + + + +-----+ PANEL 1 + TEXT PANEL #2 + + + +-----+ \ </pre>
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>T6963C_dataPort</code> : Data Port - <code>T6963C_ctrlPort</code> : Control Port - <code>T6963C_ctrlwr</code> : write signal pin - <code>T6963C_ctrlrd</code> : read signal pin - <code>T6963C_ctrlcd</code> : command/data signal pin - <code>T6963C_ctrlrst</code> : reset signal pin <p>must be defined before using this function.</p>

Example	<pre>' T6963CGLCD pinout definition dim T6963C_dataPort as byte at P0 sfr ' pointer to DATA BUS port dim T6963C_ctrlPort as byte at P1 sfr ' pointer to CONTROL BUS port dim T6963C_ctrlwr as sbit at P1.B2 ' WR write signal dim T6963C_ctrlrd as sbit at P1.B1 ' RD read signal dim T6963C_ctrlcd as sbit at P1.B0 ' CD command/data signal dim T6963C_ctrlrst as sbit at P1.B4 ' RST reset signal ... ' init display for 240 pixel width, 128 pixel height and 8 bits character width T6963C_init(240, 128, 8)</pre>
----------------	--

T6963C_WriteData

Prototype	<code>sub procedure T6963C_WriteData(dim mydata as byte)</code>
Returns	Nothing.
Description	Writes data to T6963C controller. Parameters : - <code>mydata</code> : data to be written
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_WriteData(AddrL)</code>

T6963C_WriteCommand

Prototype	<code>sub procedure T6963C_WriteCommand(dim mydata as byte)</code>
Returns	Nothing.
Description	Writes command to T6963C controller. Parameters : - <code>mydata</code> : command to be written
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_WriteCommand(T6963C_CURSOR_POINTER_SET)</code>

T6963C_SetPtr

Prototype	<code>sub procedure T6963C_SetPtr(dim p as word, dim c as byte)</code>
Returns	Nothing.
Description	Sets the memory pointer p for command c. Parameters : - p: address where command should be written - c: command to be written
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_SetPtr(T6963C_grHomeAddr + start, T6963C_ADDRESS_POINTER_SET)</code>

T6963C_WaitReady

Prototype	<code>sub procedure T6963C_WaitReady()</code>
Returns	Nothing.
Description	Pools the status byte, and loops until Toshiba GLCD module is ready.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_WaitReady()</code>

T6963C_Fill

Prototype	<code>sub procedure T6963C_Fill(dim v as byte, dim start, len as word)</code>
Returns	Nothing.
Description	Fills controller memory block with given byte. Parameters : - v: byte to be written - start: starting address of the memory block - len: length of the memory block in bytes
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Fill(0x33,0x00FF,0x000F)</code>

T6963C_Dot

Prototype	<code>sub procedure T6963C_Dot(dim x, y as integer, dim color as byte)</code>
Returns	Nothing.
Description	Draws a dot in the current graphic panel of GLCD at coordinates (x, y). Parameters : - <code>x</code> : dot position on x-axis - <code>y</code> : dot position on y-axis - <code>color</code> : color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Dot(x0, y0, pcolor)</code>

T6963C_Write_Char

Prototype	<code>sub procedure T6963C_Write_Char(dim c, x, y, mode as byte)</code>
Returns	Nothing.
Description	<p>Writes a char in the current text panel of GLCD at coordinates (x, y).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>c</code>: char to be written - <code>x</code>: char position on x-axis - <code>y</code>: char position on y-axis - <code>mode</code>: mode parameter. Valid values: T6963C_ROM_MODE_OR, T6963C_ROM_MODE_XOR, T6963C_ROM_MODE_AND and T6963C_ROM_MODE_TEXT <p>Mode parameter explanation:</p> <ul style="list-style-type: none"> - OR Mode: In the OR-Mode, text and graphics can be displayed and the data is logically "OR-ed". This is the most common way of combining text and graphics for example labels on buttons. - XOR-Mode: In this mode, the text and graphics data are combined via the logical "exclusive OR". This can be useful to display text in the negative mode, i.e. white text on black background. - AND-Mode: The text and graphic data shown on display are combined via the logical "AND function". - TEXT-Mode: This option is only available when displaying just a text. The Text Attribute values are stored in the graphic area of display memory. <p>For more details see the T6963C datasheet.</p>
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Write_Char('A',22,23,AND)</code>

T6963C_Write_Text

Prototype	<code>sub procedure T6963C_Write_Text(dim str as ^byte, dim x, y, mode as byte)</code>
Returns	Nothing.
Description	<p>Writes text in the current text panel of GLCD at coordinates (x, y).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>str</code>: text to be written - <code>x</code>: text position on x-axis - <code>y</code>: text position on y-axis - <code>mode</code>: mode parameter. Valid values: T6963C_ROM_MODE_OR, T6963C_ROM_MODE_XOR, T6963C_ROM_MODE_AND and T6963C_ROM_MODE_TEXT <p>Mode parameter explanation:</p> <ul style="list-style-type: none"> - OR Mode: In the OR-Mode, text and graphics can be displayed and the data is logically "OR-ed". This is the most common way of combining text and graphics for example labels on buttons. - XOR-Mode: In this mode, the text and graphics data are combined via the logical "exclusive OR". This can be useful to display text in the negative mode, i.e. white text on black background. - AND-Mode: The text and graphic data shown on display are combined via the logical "AND function". - TEXT-Mode: This option is only available when displaying just a text. The Text Attribute values are stored in the graphic area of display memory. <p>For more details see the T6963C datasheet.</p>
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Write_Text(" GLCD LIBRARY DEMO, WELCOME !", 0, 0, T6963C_ROM_MODE_XOR)</code>

T6963C_Line

Prototype	<code>sub procedure T6963C_Line(dim x0, y0, x1, y1 as integer, dim pcolor as byte)</code>
Returns	Nothing.
Description	Draws a line from (x0, y0) to (x1, y1). Parameters : <ul style="list-style-type: none">- x0: x coordinate of the line start- y0: y coordinate of the line end- x1: x coordinate of the line start- y1: y coordinate of the line end- pcolor: color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Line(0, 0, 239, 127, T6963C_WHITE)</code>

T6963C_Rectangle

Prototype	<code>sub procedure T6963C_Rectangle(dim x0, y0, x1, y1 as integer, dim pcolor as byte)</code>
Returns	Nothing.
Description	Draws a rectangle on GLCD. Parameters : <ul style="list-style-type: none">- x0: x coordinate of the upper left rectangle corner- y0: y coordinate of the upper left rectangle corner- x1: x coordinate of the lower right rectangle corner- y1: y coordinate of the lower right rectangle corner- pcolor: color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Rectangle(20, 20, 219, 107, T6963C_WHITE)</code>

T6963C_Box

Prototype	<code>sub procedure T6963C_Box(dim x0, y0, x1, y1 as integer, dim pcolor as byte)</code>
Returns	Nothing.
Description	<p>Draws a box on GLCD</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x0</code>: x coordinate of the upper left box corner - <code>y0</code>: y coordinate of the upper left box corner - <code>x1</code>: x coordinate of the lower right box corner - <code>y1</code>: y coordinate of the lower right box corner - <code>pcolor</code>: color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Box(0, 119, 239, 127, T6963C_WHITE)</code>

T6963C_Circle

Prototype	<code>sub procedure T6963C_Circle(dim x, y as integer, dim r as longint, dim pcolor as byte)</code>
Returns	Nothing.
Description	<p>Draws a circle on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>x</code>: x coordinate of the circle center - <code>y</code>: y coordinate of the circle center - <code>r</code>: radius size - <code>pcolor</code>: color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Circle(120, 64, 110, T6963C_WHITE)</code>

T6963C_Image

Prototype	<code>sub procedure T6963C_Image(const pic as ^byte)</code>
Returns	Nothing.
Description	<p>Displays bitmap on GLCD.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>pic</code>: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic for 8051 pointer to const and pointer to RAM equivalency). <p>Use the mikroBasic's integrated GLCD Bitmap Editor (menu option Tools > GLCD Bitmap Editor) to convert image to a constant array suitable for displaying on GLCD.</p>
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Image(mc)</code>

T6963C_Sprite

Prototype	<code>sub procedure T6963C_Sprite(dim px, py, sx, sy as byte, const pic as ^byte)</code>
Returns	Nothing.
Description	<p>Fills graphic rectangle area (px, py) to (px+sx, py+sy) with custom size picture.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>px</code>: x coordinate of the upper left picture corner. Valid values: multiples of the font width - <code>py</code>: y coordinate of the upper left picture corner - <code>pic</code>: picture to be displayed - <code>sx</code>: picture width. Valid values: multiples of the font width - <code>sy</code>: picture height <p>Note: If px and sx parameters are not multiples of the font width they will be scaled to the nearest lower number that is a multiple of the font width.</p>
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Sprite(76, 4, einstein, 88, 119) ' draw a sprite</code>

T6963C_Set_Cursor

Prototype	<code>sub procedure T6963C_Set_Cursor(dim x, y as byte)</code>
Returns	Nothing.
Description	Sets cursor to row x and column y. Parameters : - <code>x</code> : cursor position row number - <code>y</code> : cursor position column number
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Set_Cursor(cposx, cposy)</code>

T6963C_ClearBit

Prototype	<code>sub procedure T6963C_ClearBit(dim b as byte)</code>
Returns	Nothing.
Description	Clears control port bit(s). Parameters : - <code>b</code> : bit mask. The function will clear bit x on control port if bit x in bit mask is set to 1.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>' clear bits 0 and 1 on control port T6963C_ClearBit(0x03)</code>

T6963C_SetBit

Prototype	<code>sub procedure T6963C_SetBit(dim b as byte)</code>
Returns	Nothing.
Description	Sets control port bit(s). Parameters : - <code>b</code> : bit mask. The function will set bit x on control port if bit x in bit mask is set to 1.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>' set bits 0 and 1 on control port T6963C_SetBit(0x03)</code>

T6963C_NegBit

Prototype	<code>sub procedure T6963C_NegBit(dim b as byte)</code>
Returns	Nothing.
Description	Negates control port bit(s). Parameters : - b : bit mask. The function will negate bit x on control port if bit x in bit mask is set to 1.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<pre>' negate bits 0 and 1 on control port T6963C_NegBit(0x03)</pre>

T6963C_DisplayGrPanel

Prototype	<code>sub procedure T6963C_DisplayGrPanel(dim n as byte)</code>
Returns	Nothing.
Description	Display selected graphic panel. Parameters : - n : graphic panel number. Valid values: 0 and 1.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<pre>' display graphic panel 1 T6963C_DisplayGrPanel(1)</pre>

T6963C_DisplayTxtPanel

Prototype	<code>sub procedure T6963C_DisplayTxtPanel(dim n as byte)</code>
Returns	Nothing.
Description	Display selected text panel. Parameters : - n : text panel number. Valid values: 0 and 1.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<pre>' display text panel 1 T6963C_DisplayTxtPanel(1)</pre>

T6963C_SetGrPanel

Prototype	<code>sub procedure T6963C_SetGrPanel(dim n as byte)</code>
Returns	Nothing.
Description	<p>Compute start address for selected graphic panel and set appropriate internal pointers. All subsequent graphic operations will be preformed at this graphic panel.</p> <p>Parameters :</p> <p>- <code>n</code>: graphic panel number. Valid values: 0 and 1.</p>
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>' set graphic panel 1 as current graphic panel. T6963C_SetGrPanel(1)</code>

T6963C_SetTxtPanel

Prototype	<code>sub procedure T6963C_SetTxtPanel(dim n as byte)</code>
Returns	Nothing.
Description	<p>Compute start address for selected text panel and set appropriate internal pointers. All subsequent text operations will be preformed at this text panel.</p> <p>Parameters :</p> <p>- <code>n</code>: text panel number. Valid values: 0 and 1.</p>
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>' set text panel 1 as current text panel. T6963C_SetTxtPanel(1)</code>

T6963C_PanelFill

Prototype	<code>sub procedure T6963C_PanelFill(dim v as byte)</code>
Returns	Nothing.
Description	Fill current panel in full (graphic+text) with appropriate value (0 to clear). Parameters : - <code>v</code> : value to fill panel with.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<pre>clear current panel T6963C_PanelFill(0)</pre>

T6963C_GrFill

Prototype	<code>sub procedure T6963C_GrFill(dim v as byte)</code>
Returns	Nothing.
Description	Fill current graphic panel with appropriate value (0 to clear). Parameters : - <code>v</code> : value to fill graphic panel with.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<pre>' clear current graphic panel T6963C_GrFill(0)</pre>

T6963C_TxtFill

Prototype	<code>sub procedure T6963C_TxtFill(dim v as byte)</code>
Returns	Nothing.
Description	Fill current text panel with appropriate value (0 to clear). Parameters : - <code>v</code> : this value increased by 32 will be used to fill text panel.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<pre>' clear current text panel T6963C_TxtFill(0)</pre>

T6963C_Cursor_Height

Prototype	<code>sub procedure T6963C_Cursor_Height(dim n as byte)</code>
Returns	Nothing.
Description	Set cursor size. Parameters : - <code>n</code> : cursor height. Valid values: 0..7.
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Cursor_Height(7)</code>

T6963C_Graphics

Prototype	<code>sub procedure T6963C_Graphics(dim n as byte)</code>
Returns	Nothing.
Description	Enable/disable graphic displaying. Parameters : - <code>n</code> : on/off parameter. Valid values: 0 (disable graphic displaying) and 1 (enable graphic displaying).
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>' enable graphic displaying T6963C_Graphics(1)</code>

T6963C_Text

Prototype	<code>sub procedure T6963C_Text(dim n as byte)</code>
Returns	Nothing.
Description	Enable/disable text displaying. Parameters : - <code>n</code> : on/off parameter. Valid values: 0 (disable text displaying) and 1 (enable text displaying).
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<code>' enable text displaying T6963C_Text(1)</code>

T6963C_Cursor

Prototype	<code>sub procedure T6963C_Cursor(dim n as byte)</code>
Returns	Nothing.
Description	Set cursor on/off. Parameters : - <code>n</code> : on/off parameter. Valid values: 0 (set cursor off) and 1 (set cursor on).
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<pre>' set cursor on T6963C_Cursor(1)</pre>

T6963C_Cursor_Blink

Prototype	<code>sub procedure T6963C_Cursor_Blink(dim n as byte)</code>
Returns	Nothing.
Description	Enable/disable cursor blinking. Parameters : - <code>n</code> : on/off parameter. Valid values: 0 (disable cursor blinking) and 1 (enable cursor blinking).
Requires	Toshiba GLCD module needs to be initialized. See the T6963C_Init routine.
Example	<pre>' enable cursor blinking T6963C_Cursor_Blink(1)</pre>

Library Example

The following drawing demo tests advanced routines of the T6963C GLCD library. Hardware configurations in this example are made for the T6963C 240x128 display, Easy8051B board and AT89S8253.

```
program T6963C_240x128

include __Lib_T6963C_Consts
include bitmap
include bitmap2

' T6963C module connections
dim T6963C_dataPort as byte at P0 sfr      ' DATA port
dim T6963C_cntlPort as byte at P1 sfr     ' CONTROL port
```

```

dim T6963C_cntlwr  as sbit at P1.B2      ' WR write signal
dim T6963C_cntlrd  as sbit at P1.B1      ' RD read signal
dim T6963C_cntlcd  as sbit at P1.B0      ' CD command/data signal
dim T6963C_cntlrst as sbit at P1.B4      ' RST reset signal
' End T6963C module connections

dim  panel as byte      ' current panel
     i  as word         ' general purpose register
curs  as byte          ' cursor visibility
cposx, cposy as word    ' cursor x-y position
txtcols as byte        ' number of text coloms
txt, txt1 as string[ 29] idata

txt1 = " EINSTEIN WOULD HAVE LIKED mC"
txt  = " GLCD LIBRARY DEMO, WELCOME !"

P2 = 255 ' all inputs
' Clear T6963C ports
P1 = 0   ' control bus
P0 = 0   ' data bus

'
' * init display for 240 pixel width and 128 pixel height
' * 8 bits character width
' * data bus on P0
' * control bus on P1
' * bit 2 is !WR
' * bit 1 is !RD
' * bit 0 is !CD
' * bit 4 is RST

T6963C_init(240, 128, 8)

'
' *
' * enable both graphics and text display at the same time
' *

T6963C_graphics(1)
T6963C_text(1)

panel = 0
i      = 0
curs   = 0
cposx  = 0
cposy  = 0
txtcols = 240 div 8      ' calculate number of text colomns
                        ' (grafic display width divided by
font width)

```

```

'      *
'      * text messages
'      *

T6963C_write_text(txt, 0, 0, T6963C_ROM_MODE_XOR)
T6963C_write_text(txt1, 0, 15, T6963C_ROM_MODE_XOR)

'      *
'      * cursor
'      *

T6963C_cursor_height(8)           ' 8 pixel height
T6963C_set_cursor(0, 0)           ' move cursor to top left
T6963C_cursor(0)                  ' cursor off

'      *
'      * draw rectangles
'      *

T6963C_rectangle(0, 0, 239, 127, T6963C_BLACK)
T6963C_rectangle(20, 20, 219, 107, T6963C_BLACK)
T6963C_rectangle(40, 40, 199, 87, T6963C_BLACK)
T6963C_rectangle(60, 60, 179, 67, T6963C_BLACK)

'      *
'      * draw a cross
'      *

T6963C_line(0, 0, 239, 127, T6963C_BLACK)
T6963C_line(0, 127, 239, 0, T6963C_BLACK)

'      *
'      * draw solid boxes
'      *

T6963C_box(0, 0, 239, 8, T6963C_BLACK)
T6963C_box(0, 119, 239, 127, T6963C_BLACK)

'      *
'      * draw circles
'      *

T6963C_circle(120, 64, 10, T6963C_BLACK)
T6963C_circle(120, 64, 30, T6963C_BLACK)

T6963C_sprite(76, 4, @einstein, 88, 119) ' draw a sprite

T6963C_setGrPanel(1)                ' select other graphic panel

T6963C_Image(@banner_bmp)

```

```

while TRUE

'      *
'      * if P2_0 is pressed, toggle the display between graphic
panel 0 and graphic 1
'      *

    if ( P2_0 = 0 ) then
        Inc(panel)
        panel = panel and 1
        T6963C_displayGrPanel(panel)
        Delay_ms(300)

'      *
'      * if P2_1 is pressed, display only graphic panel
'      *

    else
    if ( P2_1 = 0 ) then
        T6963C_graphics(1)
        T6963C_text(0)
        Delay_ms(300)

'      *
'      * if P2_2 is pressed, display only text panel
'      *

    else
    if (P2_2 = 0) then

        T6963C_graphics(0)
        T6963C_text(1)
        Delay_ms(300)

'      *
'      * if P2_3 is pressed, display text and graphic panels
'      *

    else
    if(P2_3 = 0) then
        T6963C_graphics(1)
        T6963C_text(1)
        Delay_ms(300)

'      *
'      * if P2_4 is pressed, change cursor
'      *

```

```

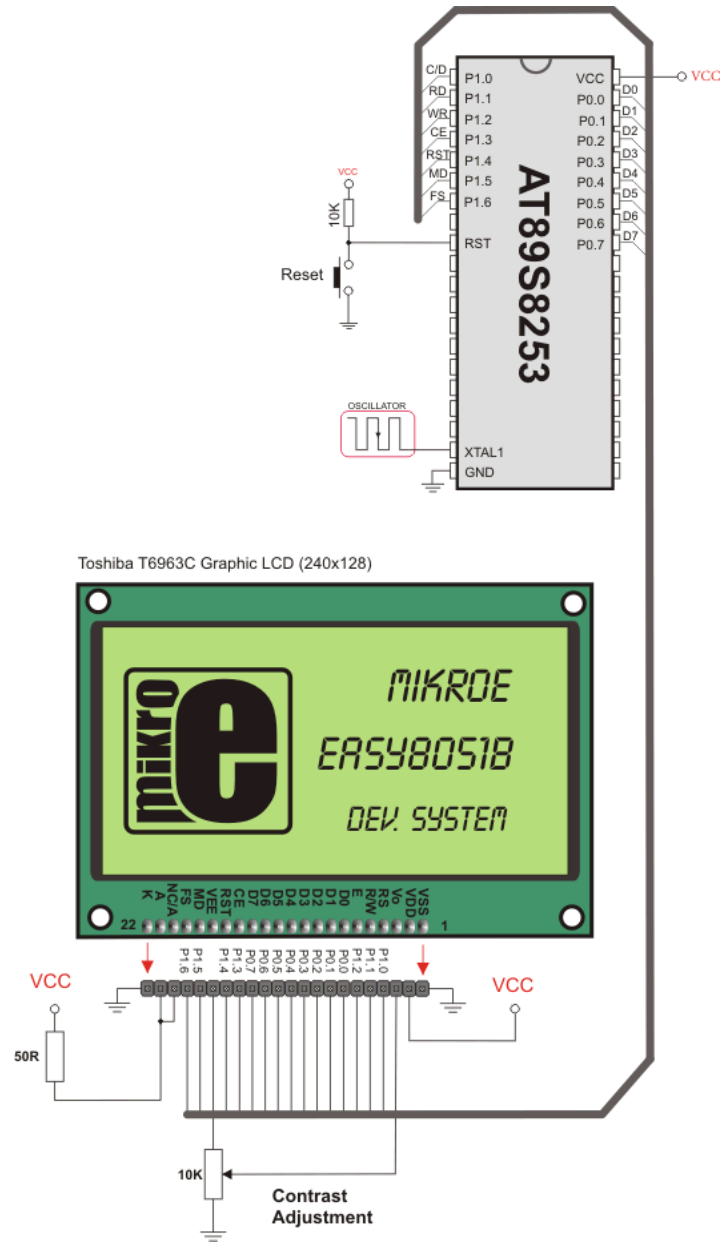
else
    if(P2_4 = 0) then
        curs = curs + 1
        if (curs = 3) then
            curs = 0
        end if
        select case curs
            case 0
                T6963C_cursor(0)
            case 1
                T6963C_cursor(1)
                T6963C_cursor_blink(1)
            case 2
                T6963C_cursor(1)
                T6963C_cursor_blink(0)
        end select ' end case
        Delay_ms(300)
    end if
end if
end if
end if

'      *
'      * move cursor, even if not visible
'      *

Inc(cposx)
if(cposx = txtcols) then
    cposx = 0
    cposy = cposy + 1
    if (cposy = (128 div T6963C_CHARACTER_HEIGHT)) then
        '
if y end
        cposy = 0
        '
grafic height (128) div character height
    end if
end if
T6963C_set_cursor(cposx, cposy)
Delay_ms(100)
wend
end.

```

HW Connection



T6963C GLCD HW connection

UART LIBRARY

The UART hardware module is available with a number of 8051 compliant MCUs. The mikroBasic for 8051 UART Library provides comfortable work with the Asynchronous (full duplex) mode.

Library Routines

- Uart_Init
- Uart_Data_Ready
- Uart_Read
- Uart_Write

Uart_Init

Prototype	<code>sub procedure Uart_Init(dim baud_rate as longint)</code>
Returns	Nothing.
Description	<p>Configures and initializes the UART module.</p> <p>The internal UART module module is set to:</p> <ul style="list-style-type: none"> - 8-bit data, no parity - 1 STOP bit - disabled automatic address recognition - timer1 as baudrate source (mod2 = autoreload 8bit timer) <p>Parameters :</p> <ul style="list-style-type: none"> - <code>baud_rate</code>: requested baud rate <p>Refer to the device data sheet for baud rates allowed for specific Fosc.</p>
Requires	MCU with the UART module and TIMER1 to be used as baudrate source.
Example	<pre>' Initialize hardware UART and establish communication at 2400 bps Uart_Init(2400)</pre>

Uart_Data_Ready

Prototype	<code>sub function Uart_Data_Ready() as byte</code>
Returns	- 1 if data is ready for reading - 0 if there is no data in the receive register
Description	The function tests if data in receive buffer is ready for reading.
Requires	MCU with the UART module. The UART module must be initialized before using this routine. See the Uart_Init routine.
Example	<pre>dim receive as byte ... ' read data if ready if (Uart_Data_Ready)=1 then receive = Uart_Read()</pre>

Uart_Read

Prototype	<code>sub function Uart_Read() as byte</code>
Returns	Received byte.
Description	The function receives a byte via UART. Use the Uart_Data_Ready function to test if data is ready first.
Requires	MCU with the UART module. The UART module must be initialized before using this routine. See Uart_Init routine.
Example	<pre>dim receive as byte ... ' read data if ready if (Uart_Data_Ready)=1 then receive = Uart_Read()</pre>

Uart_Write

Prototype	<code>sub procedure Uart_Write(dim TxData as byte)</code>
Returns	Nothing.
Description	The function transmits a byte via the UART module. Parameters : - <code>TxData</code> : data to be sent
Requires	MCU with the UART module. The UART module must be initialized before using this routine. See <code>Uart_Init</code> routine.
Example	<code>dim data as byte</code> <code>...</code> <code>data = 0x1E</code> <code>Uart_Write(data)</code>

Library Example

This example demonstrates simple data exchange via UART. If MCU is connected to the PC, you can test the example from the mikroBasic for 8051 USART Terminal.

```

program UART

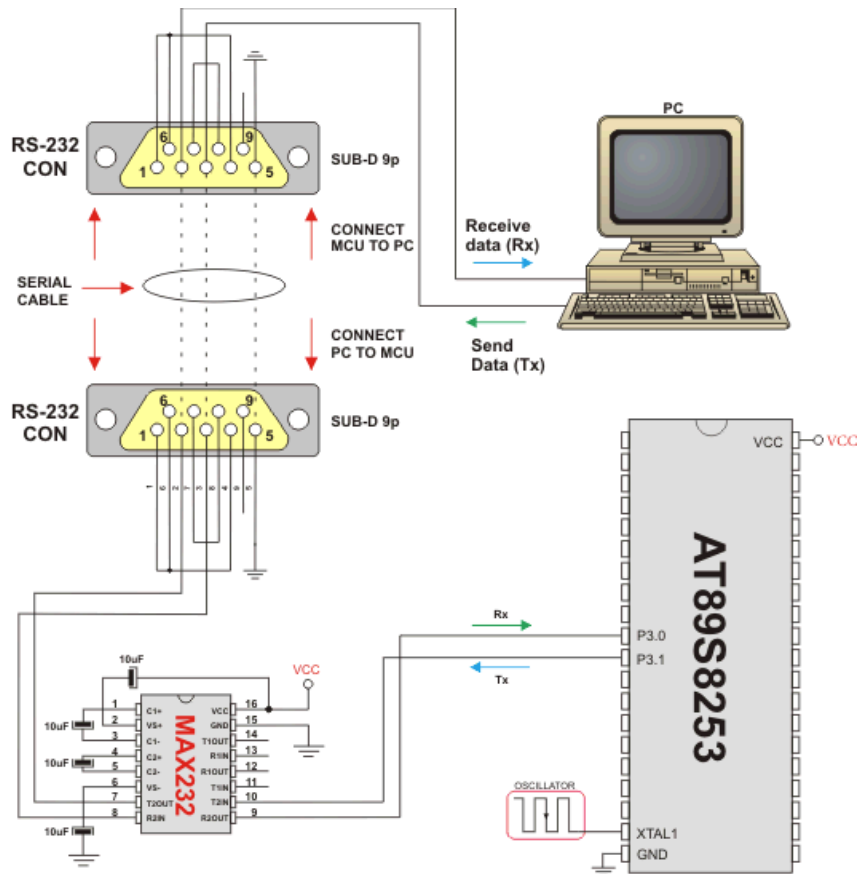
dim uart_rd as byte

main:
    Uart_Init(4800)           ' Initialize UART module at 4800 bps
    Delay_ms(100)            ' Wait for UART module to stabilize

    while TRUE              ' Endless loop
        if (Uart_Data_Ready() <> 0) then ' Check if UART module has received
data
            uart_rd = Uart_Read()        ' Read data
            Uart_Write(uart_rd)         ' Send the same data back
        end if
    wend
end.

```

HW Connection



UART HW connection

BUTTON LIBRARY

The Button library contains miscellaneous routines useful for a project development.

External dependencies of Button Library

The following variable must be defined in all projects using Button library:	Description:	Example :
<pre>dim Button_Pin as sbit external</pre>	Declares Button_Pin, which will be used by Button Library.	<pre>dim Button_Pin as sbit at P0_0</pre>

Library Routines

- Button

Button

Prototype	<code>sub function Button(dim time_ms as byte, dim active_state as byte) as byte</code>
Returns	<ul style="list-style-type: none"> - 255 if the pin was in the active state for given period. - 0 otherwise
Description	<p>The function eliminates the influence of contact flickering upon pressing a button (debouncing). The Button pin is tested just after the function call and then again after the debouncing period has expired. If the pin was in the active state in both cases then the function returns 255 (true).</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>time_ms</code>: debouncing period in milliseconds - <code>active_state</code>: determines what is considered as active state. Valid values: 0 (logical zero) and 1 (logical one)
Requires	<p><code>Button_Pin</code> variable must be defined before using this function.</p> <p>Button pin must be configured as input.</p>
Example	<p>P2 is inverted on every P0.B0 one-to-zero transition :</p> <pre> program Button ' Button connections dim Button_Pin as sbit at P0.B0; ' Declare Button_Pin. It will be used by Button Library. ' End Button connections bit oldstate; ' Old state flag main: P0 = 255 ' Configure PORT0 as input P2 = 0xAA ' Initial PORT2 value do { if (Button(1, 1)) ' Detect logical one oldstate = 1; ' Update flag if (oldstate && Button(1, 0)) { ' Detect one-to-zero transi- tion P2 = ~P2; ' Invert PORT2 oldstate = 0; ' Update flag } } while(1); ' Endless loop } '~! </pre>

CONVERSIONS LIBRARY

mikroBasic for 8051 Conversions Library provides routines for numerals to strings and BCD/decimal conversions.

Library Routines

You can get text representation of numerical value by passing it to one of the following routines:

- ByteToStr
- ShortToStr
- WordToStr
- IntToStr
- LongintToStr
- LongWordToStr
- FloatToStr

The following sub functions convert decimal values to BCD and vice versa:

- Dec2Bcd
- Bcd2Dec16
- Dec2Bcd16

ByteToStr

Prototype	<code>sub procedure ByteToStr(dim input as word, dim byref output as string[2])</code>
Returns	Nothing.
Description	<p>Converts input byte to a string. The output string is right justified and remaining positions on the left (if any) are filled with blanks.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>input</code>: byte to be converted - <code>output</code>: destination string
Requires	Nothing.
Example	<pre>dim t as word txt as string[2] ... t = 24 ByteToStr(t, txt) ' txt is " 24" (one blank here)</pre>

ShortToStr

Prototype	<code>sub procedure ShortToStr(dim input as short, dim byref output as string[3])</code>
Returns	Nothing.
Description	<p>Converts input short (signed byte) number to a string. The output string is right justified and remaining positions on the left (if any) are filled with blanks.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>input</code>: short number to be converted - <code>output</code>: destination string
Requires	Nothing.
Example	<pre>dim t as short txt as string[3] ... t = -24 ByteToStr(t, txt) ' txt is " -24" (one blank here)</pre>

WordToStr

Prototype	<code>sub procedure WordToStr(dim input as word, dim byref output as string[4])</code>
Returns	Nothing.
Description	<p>Converts input word to a string. The output string is right justified and the remaining positions on the left (if any) are filled with blanks.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>input</code>: word to be converted - <code>output</code>: destination string
Requires	Nothing.
Example	<pre>dim t as word txt as string[4] ... t = 437 WordToStr(t, txt) ' txt is " 437" (two blanks here)</pre>

IntToStr

Prototype	<code>sub procedure IntToStr(dim input as integer, dim byref output as string[5])</code>
Returns	Nothing.
Description	<p>Converts input integer number to a string. The output string is right justified and the remaining positions on the left (if any) are filled with blanks.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>input</code>: integer number to be converted - <code>output</code>: destination string
Requires	Nothing.
Example	<pre>dim input as integer txt as string[5] '... input = -4220 IntToStr(input, txt) ' txt is ' -4220'</pre>

LongintToStr

Prototype	<code>sub procedure LongintToStr(dim input as longint, dim byref output as string[10])</code>
Returns	Nothing.
Description	<p>Converts input longint number to a string. The output string is right justified and the remaining positions on the left (if any) are filled with blanks.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>input</code>: longint number to be converted - <code>output</code>: destination string
Requires	Nothing.
Example	<pre>dim input as longint txt as string[10] '...' input = -12345678 IntToStr(input, txt) ' txt is ' -12345678'</pre>

LongWordToStr

Prototype	<code>sub procedure LongWordToStr(dim input as longword, dim byref output as string[9])</code>
Returns	Nothing.
Description	<p>Converts input double word number to a string. The output string is right justified and the remaining positions on the left (if any) are filled with blanks.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>input</code>: double word number to be converted - <code>output</code>: destination string
Requires	Nothing.
Example	<pre>dim input as longint txt as string[9] '...' input = 12345678 IntToStr(input, txt) ' txt is ' 12345678'</pre>

FloatToStr

Prototype	<code>sub function FloatToStr(dim input as real, dim byref output as string[22])</code>
Returns	<ul style="list-style-type: none"> - 3 if input number is NaN - 2 if input number is -INF - 1 if input number is +INF - 0 if conversion was successful
Description	<p>Converts a floating point number to a string.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>input</code>: floating point number to be converted - <code>output</code>: destination string <p>The output string is left justified and null terminated after the last digit.</p> <p>Note: Given floating point number will be truncated to 7 most significant digits before conversion.</p>
Requires	Nothing.
Example	<pre>dim ff1, ff2, ff3 as real txt as string[22] ... ff1 = -374.2 ff2 = 123.456789 ff3 = 0.000001234 FloatToStr(ff1, txt) ' txt is "-374.2" FloatToStr(ff2, txt) ' txt is "123.4567" FloatToStr(ff3, txt) ' txt is "1.234e-6"</pre>

Dec2Bcd

Prototype	<code>sub function Dec2Bcd(dim decnum as byte) as byte</code>
Returns	Converted BCD value.
Description	Converts input number to its appropriate BCD representation. Parameters : - <code>decnum</code> : number to be converted
Requires	Nothing.
Example	<code>dim a, b as byte</code> <code>...</code> <code>a = 22</code> <code>b = Dec2Bcd(a) ' b equals 34</code>

Bcd2Dec16

Prototype	<code>sub function Bcd2Dec16(dim bcdnum as word) as word</code>
Returns	Converted decimal value.
Description	Converts 16-bit BCD numeral to its decimal equivalent. Parameters : - <code>bcdnum</code> : 16-bit BCD numeral to be converted
Requires	Nothing.
Example	<code>dim a, b as word</code> <code>...</code> <code>a = 0x1234 ' a equals 4660</code> <code>b = Bcd2Dec16(a) ' b equals 1234</code>

Dec2Bcd16

Prototype	<code>sub function Dec2Bcd16 (dim decnum as word) as word</code>
Returns	Converted BCD value.
Description	<p>Converts decimal value to its BCD equivalent.</p> <p>Parameters :</p> <p>- <code>decnum</code> decimal number to be converted</p>
Requires	Nothing.
Example	<pre> dim a, b as word ... a = 2345 b = Dec2Bcd16(a) ' b equals 9029 </pre>

MATH LIBRARY

The mikroBasic for 8051 provides a set of library functions for floating point math handling. See also Predefined Globals and Constants for the list of predefined math constants.

Library Functions

- acos
- asin
- atan
- atan2
- ceil
- cos
- cosh
- eval_poly
- exp
- fabs
- floor
- frexp
- ldexp
- log
- log10
- modf
- pow
- sin
- sinh
- sqrt
- tan
- tanh

acos

Prototype	<code>sub function acos(dim x as real) as real</code>
Description	The function returns the arc cosine of parameter <code>x</code> ; that is, the value whose cosine is <code>x</code> . The input parameter <code>x</code> must be between -1 and 1 (inclusive). The return value is in radians, between 0 and π (inclusive).

asin

Prototype	<code>sub function asin(dim x as real) as real</code>
Description	The function returns the arc sine of parameter <code>x</code> ; that is, the value whose sine is <code>x</code> . The input parameter <code>x</code> must be between -1 and 1 (inclusive). The return value is in radians, between $-\pi/2$ and $\pi/2$ (inclusive).

atan

Prototype	<code>sub function atan(dim arg as real) as real</code>
Description	The function computes the arc tangent of parameter <code>arg</code> ; that is, the value whose tangent is <code>arg</code> . The return value is in radians, between $-\pi/2$ and $\pi/2$ (inclusive).

atan2

Prototype	<code>sub function atan2(dim y as real, dim x as real) as real</code>
Description	This is the two-argument arc tangent function. It is similar to computing the arc tangent of <code>y/x</code> , except that the signs of both arguments are used to determine the quadrant of the result and <code>x</code> is permitted to be zero. The return value is in radians, between $-\pi$ and π (inclusive).

ceil

Prototype	<code>sub function ceil(dim x as real) as real</code>
Description	The function returns value of parameter <code>x</code> rounded up to the next whole number.

cos

Prototype	<code>sub function cos(dim arg as real) as real</code>
Description	The function returns the cosine of <code>arg</code> in radians. The return value is from -1 to 1.

cosh

Prototype	<code>sub function cosh(dim x as real) as real</code>
Description	The function returns the hyperbolic cosine of <code>x</code> , defined mathematically as $(e^x + e^{-x}) / 2$. If the value of <code>x</code> is too large (if overflow occurs), the function fails.

eval_poly

Prototype	<code>sub function eval_poly(dim x as real, dim byref d as array[10] of real, dim n as integer) as real</code>
Description	Function Calculates polynom for number <code>x</code> , with coefficients stored in <code>d[]</code> , for degree <code>n</code> .

exp

Prototype	<code>sub function exp(dim x as real) as real</code>
Description	The function returns the value of e — the base of natural logarithms — raised to the power <code>x</code> (i.e. e^x).

fabs

Prototype	<code>sub function fabs(dim d as real) as real</code>
Description	The function returns the absolute (i.e. positive) value of <code>d</code> .

floor

Prototype	<code>sub function floor(dim x as real) as real</code>
Description	The function returns the value of parameter <code>x</code> rounded down to the nearest integer.

frexp

Prototype	<code>sub function frexp(dim value as real, dim byref eptr as integer) as real</code>
Description	The function splits a floating-point value <code>value</code> into a normalized fraction and an integral power of 2. The return value is a normalized fraction and the integer exponent is stored in the object pointed to by <code>eptr</code> .

ldexp

Prototype	<code>sub function ldexp(dim value as real, dim newexp as integer) as real</code>
Description	The function returns the result of multiplying the floating-point number <code>value</code> by 2 raised to the power <code>newexp</code> (i.e. returns <code>value * 2^{newexp}</code>).

log

Prototype	<code>sub function log(dim x as real) as real</code>
Description	The function returns the natural logarithm of <code>x</code> (i.e. $\log_e(x)$).

log10

Prototype	<code>sub function log10(dim x as real) as real</code>
Description	The function returns the base-10 logarithm of <code>x</code> (i.e. $\log_{10}(x)$).

modf

Prototype	<code>sub function modf(dim val as real, dim byref iptr as real) as real</code>
Description	The function returns the signed fractional component of <code>val</code> , placing its whole number component into the variable pointed to by <code>iptr</code> .

pow

Prototype	<code>sub function pow(dim x as real, dim y as real) as real</code>
Description	The function returns the value of <code>x</code> raised to the power <code>y</code> (i.e. x^y). If <code>x</code> is negative, the function will automatically cast <code>y</code> into <code>longint</code> .

sin

Prototype	<code>sub function sin(dim arg as real) as real</code>
Description	The function returns the sine of <code>arg</code> in radians. The return value is from -1 to 1.

sinh

Prototype	<code>sub function sinh(dim x as real) as real</code>
Description	The function returns the hyperbolic sine of <code>x</code> , defined mathematically as $(e^x - e^{-x}) / 2$. If the value of <code>x</code> is too large (if overflow occurs), the function fails.

sqrt

Prototype	<code>sub function sqrt(dim x as real) as real</code>
Description	The function returns the non negative square root of <code>x</code> .

tan

Prototype	<code>sub function tan(dim x as real) as real</code>
Description	The function returns the tangent of <code>x</code> in radians. The return value spans the allowed range of floating point in mikroBasic for 8051.

tanh

Prototype	<code>sub function tanh(dim x as real) as real</code>
Description	The function returns the hyperbolic tangent of x , defined mathematically as $\sinh(x)/\cosh(x)$.

STRING LIBRARY

The mikroBasic for 8051 includes a library which automatizes string related tasks.

Library Functions

- memchr
- memcmp
- memcpy
- memmove
- memset
- strcat
- strchr
- strcmp
- strcpy
- strlen
- strncat
- strncpy
- strspn
- strcspn
- strncmp
- strpbrk
- strrchr
- strstr

memchr

Prototype	<code>sub function memchr(dim p as word, dim ch as byte, dim n as word) as word</code>
Description	<p>The function locates the first occurrence of the word <code>ch</code> in the initial <code>n</code> words of memory area starting at the address <code>p</code>. The function returns the offset of this occurrence from the memory address <code>p</code> or <code>0xFFFF</code> if <code>ch</code> was not found.</p> <p>For the parameter <code>p</code> you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example <code>@mystring</code> or <code>@PORTB</code>.</p>

memcmp

Prototype	<code>sub function memcmp(p1, p2, n as word) as integer</code>								
Description	<p>The function returns a positive, negative, or zero value indicating the relationship of first <code>n</code> words of memory areas starting at addresses <code>p1</code> and <code>p2</code>.</p> <p>This function compares two memory areas starting at addresses <code>p1</code> and <code>p2</code> for <code>n</code> words and returns a value indicating their relationship as follows:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>< 0</code></td> <td><code>p1 "less than" p2</code></td> </tr> <tr> <td><code>= 0</code></td> <td><code>p1 "equal to" p2</code></td> </tr> <tr> <td><code>> 0</code></td> <td><code>p1 "greater than" p2</code></td> </tr> </tbody> </table> <p>The value returned by the function is determined by the difference between the values of the first pair of words that differ in the strings being compared.</p> <p>For parameters <code>p1</code> and <code>p2</code> you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example <code>@mystring</code> or <code>@PORTB</code>.</p>	Value	Meaning	<code>< 0</code>	<code>p1 "less than" p2</code>	<code>= 0</code>	<code>p1 "equal to" p2</code>	<code>> 0</code>	<code>p1 "greater than" p2</code>
Value	Meaning								
<code>< 0</code>	<code>p1 "less than" p2</code>								
<code>= 0</code>	<code>p1 "equal to" p2</code>								
<code>> 0</code>	<code>p1 "greater than" p2</code>								

memcpy

Prototype	<code>sub procedure memcpy(p1, p2, nn as word)</code>
Description	<p>The function copies <code>nn</code> words from the memory area starting at the address <code>p2</code> to the memory area starting at <code>p1</code>. If these memory buffers overlap, the <code>memcpy</code> function cannot guarantee that words are copied before being overwritten. If these buffers do overlap, use the <code>memmove</code> function.</p> <p>For parameters <code>p1</code> and <code>p2</code> you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example <code>@mystring</code> or <code>@PORTB</code>.</p>

memmove

Prototype	<code>sub procedure memmove(p1, p2, nn as word)</code>
Description	<p>The function copies <code>nn</code> words from the memory area starting at the address <code>p2</code> to the memory area starting at <code>p1</code>. If these memory buffers overlap, the <code>Memmove</code> function ensures that the words in <code>p2</code> are copied to <code>p1</code> before being overwritten.</p> <p>For parameters <code>p1</code> and <code>p2</code> you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example <code>@mystring</code> or <code>@PORTB</code>.</p>

memset

Prototype	<code>sub procedure memset(dim p as word, dim character as byte, dim n as word)</code>
Description	The function fills the first n words in the memory area starting at the address p with the value of word character. For parameter p you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example @mystring or @PORTB.

strcat

Prototype	<code>sub procedure strcat(dim byref s1, s2 as string[100])</code>
Description	The function appends the value of string s2 to string s1 and terminates s1 with a null character.

strchr

Prototype	<code>sub function strchr(dim byref s as string[100] , dim ch as byte) as word</code>
Description	The function searches the string s for the first occurrence of the character ch. The null character terminating s is not included in the search. The function returns the position (index) of the first character ch found in s; if no matching character was found, the function returns 0xFFFF.

strcmp

Prototype	<code>sub function strcmp(dim byref s1, s2 as string[100]) as integer</code>								
Description	The function lexicographically compares the contents of the strings s1 and s2 and returns a value indicating their relationship: <table border="0"> <tr> <td>Value</td> <td>Meaning</td> </tr> <tr> <td>< 0</td> <td>s1 "less than" s2</td> </tr> <tr> <td>= 0</td> <td>s1 "equal to" s2</td> </tr> <tr> <td>> 0</td> <td>s1 "greater than" s2</td> </tr> </table> The value returned by the function is determined by the difference between the values of the first pair of words that differ in the strings being compared.	Value	Meaning	< 0	s1 "less than" s2	= 0	s1 "equal to" s2	> 0	s1 "greater than" s2
Value	Meaning								
< 0	s1 "less than" s2								
= 0	s1 "equal to" s2								
> 0	s1 "greater than" s2								

strcpy

Prototype	<code>sub procedure strcpy(dim byref s1, s2 as string[100])</code>
Description	The function copies the value of the string <code>s2</code> to the string <code>s1</code> and appends a null character to the end of <code>s1</code> .

strcspn

Prototype	<code>sub function strcspn(dim byref s1, s2 as string[100]) as word</code>
Description	The function searches the string <code>s1</code> for any of the characters in the string <code>s2</code> . The function returns the index of the first character located in <code>s1</code> that matches any character in <code>s2</code> . If the first character in <code>s1</code> matches a character in <code>s2</code> , a value of 0 is returned. If there are no matching characters in <code>s1</code> , the length of the string is returned (not including the terminating null character).

strlen

Prototype	<code>sub function strlen(dim byref s as string[100]) as word</code>
Description	The function returns the length, in words, of the string <code>s</code> . The length does not include the null terminating character.

strncat

Prototype	<code>sub procedure strncat(dim byref s1, s2 as string[100] , dim size as byte)</code>
Description	The function appends at most <code>size</code> characters from the string <code>s2</code> to the string <code>s1</code> and terminates <code>s1</code> with a null character. If <code>s2</code> is shorter than the <code>size</code> characters, <code>s2</code> is copied up to and including the null terminating character.

strncmp

Prototype	<code>sub function strncmp(dim byref s1, s2 as string[100] , dim len as byte) as integer</code>								
Description	The function lexicographically compares the first <code>len</code> words of the strings <code>s1</code> and <code>s2</code> and returns a value indicating their relationship: <table border="0"> <tr> <td>Value</td> <td>Meaning</td> </tr> <tr> <td>< 0</td> <td>s1 "less than" s2</td> </tr> <tr> <td>= 0</td> <td>s1 "equal to" s2</td> </tr> <tr> <td>> 0</td> <td>s1 "greater than" s2</td> </tr> </table> The value returned by the function is determined by the difference between the values of the first pair of words that differ in the strings being compared (within first <code>len</code> words).	Value	Meaning	< 0	s1 "less than" s2	= 0	s1 "equal to" s2	> 0	s1 "greater than" s2
Value	Meaning								
< 0	s1 "less than" s2								
= 0	s1 "equal to" s2								
> 0	s1 "greater than" s2								

strncpy

Prototype	<code>sub procedure strncpy(dim byref s1, s2 as string[100], dim size as byte)</code>
Description	The function copies at most <code>size</code> characters from the string <code>s2</code> to the string <code>s1</code> . If <code>s2</code> contains fewer characters than <code>size</code> , <code>s1</code> is padded out with null characters up to the total length of the <code>size</code> characters.

strpbrk

Prototype	<code>sub function strpbrk(dim byref s1, s2 as string[100]) as word</code>
Description	The function searches <code>s1</code> for the first occurrence of any character from the string <code>s2</code> . The null terminator is not included in the search. The function returns an index of the matching character in <code>s1</code> . If <code>s1</code> contains no characters from <code>s2</code> , the function returns <code>0xFFFF</code> .

strrchr

Prototype	<code>sub function strrchr(dim byref s as string[100], dim ch as byte) as word</code>
Description	The function searches the string <code>s</code> for the last occurrence of the character <code>ch</code> . The null character terminating <code>s</code> is not included in the search. The function returns an index of the last <code>ch</code> found in <code>s</code> ; if no matching character was found, the function returns <code>0xFFFF</code> .

strspn

Prototype	<code>sub function strspn(dim byref s1, s2 as string[100]) as word</code>
Description	The function searches the string <code>s1</code> for characters not found in the <code>s2</code> string. The function returns the index of first character located in <code>s1</code> that does not match a character in <code>s2</code> . If the first character in <code>s1</code> does not match a character in <code>s2</code> , a value of 0 is returned. If all characters in <code>s1</code> are found in <code>s2</code> , the length of <code>s1</code> is returned (not including the terminating null character).

strstr

Prototype	<code>sub function strstr(dim byref s1, s2 as string[100]) as word</code>
Description	The function locates the first occurrence of the string <code>s2</code> in the string <code>s1</code> (excluding the terminating null character). The function returns a number indicating the position of the first occurrence of <code>s2</code> in <code>s1</code> ; if no string was found, the function returns <code>0xFFFF</code> . If <code>s2</code> is a null string, the function returns 0.

TIME LIBRARY

The Time Library contains functions and type definitions for time calculations in the UNIX time format which counts the number of seconds since the "epoch". This is very convenient for programs that work with time intervals: the difference between two UNIX time values is a real-time difference measured in seconds.

What is the epoch?

Originally it was defined as the beginning of 1970 GMT. (January 1, 1970 Julian day) GMT, Greenwich Mean Time, is a traditional term for the time zone in England.

The TimeStruct type is a structure type suitable for time and date storage.

Library Routines

- Time_dateToEpoch
- Time_epochToDate
- Time_datediff

Time_dateToEpoch

Prototype	<code>sub function Time_dateToEpoch(dim byref ts as TimeStruct) as longint</code>
Returns	Number of seconds since January 1, 1970 0h00mn00s.
Description	This function returns the UNIX time : number of seconds since January 1, 1970 0h00mn00s. Parameters : - <code>ts</code> : time and date value for calculating UNIX time.
Requires	Nothing.
Example	<pre>dm ts1 as TimeStruct Epoch as longint ... ' what is the epoch of the date in ts ? epoch = Time_dateToEpoch(ts1)</pre>

Time_epochToDate

Prototype	<code>sub procedure Time_epochToDate(dim e as longint, dim byref ts as TimeStruct)</code>
Returns	Nothing.
Description	<p>Converts the UNIX time to time and date.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - e: UNIX time (seconds since UNIX epoch) - ts: time and date structure for storing conversion output
Requires	Nothing.
Example	<pre>dim ts2 as TimeStruct epoch as longint ... ' what date is epoch 1234567890 ? epoch = 1234567890 Time_epochToDate(epoch, ts2)</pre>

Time_dateDiff

Prototype	<code>sub function Time_dateDiff(dim t1 as ^TimeStruct, dim t2 as ^TimeStruct) as longint</code>
Returns	Time difference in seconds as a signed long.
Description	<p>This function compares two dates and returns time difference in seconds as a signed long. The result is positive if t1 is before t2, null if t1 is the same as t2 and negative if t1 is after t2.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - t1: time and date structure (the first comparison parameter) - t2: time and date structure (the second comparison parameter)
Requires	Nothing.
Example	<pre>dim ts1, ts2 as TimeStruct diff as longint ... ' how many seconds between these two dates contained in ts1 and ts2 buffers? diff = Time_dateDiff(ts1, ts2)</pre>

Library Example

Demonstration of Time library routines usage for time calculations in UNIX time format.

```

program Time_Demo;
  { *
    * simple time structure
    * }
  type TimeStruct = record
    ss as byte ;      ' seconds
    mn as byte ;      ' minutes
    hh as byte ;      ' hours
    md as byte ;      ' day in month, from 1 to 31
    wd as byte ;      ' day in week, monday=0, tuesday=1, .... sun-
day=6
    mo as byte ;      ' month number, from 1 to 12 (and not from 0
to 11 as with UNIX C time !)
    yy as word ;      ' year Y2K compliant, from 1892 to 2038
  end;

  var ts1, ts2      : TimeStruct ;
      buf            as array[ 256] of byte ;
      epoch, diff   : longint ;

      ts1.ss := 0 ;
      ts1.mn := 7 ;
      ts1.hh := 17 ;
      ts1.md := 23 ;
      ts1.mo := 5 ;
      ts1.yy := 2006 ;

  { *
    * what is the epoch of the date in ts ?
    * }
  epoch := Time_dateToEpoch(ts1) ;      ' epoch = 1148404020

  { *
    * what date is epoch 1234567890 ?
    * }

  epoch := 1234567890 ;
  Time_epochToDate(epoch, ts2) ;      ' ts2.ss := 30 ;
                                       ' ts2.mn := 31 ;
                                       ' ts2.hh := 23 ;
                                       ' ts2.md := 13 ;
                                       ' ts2.wd := 4 ;
                                       ' ts2.mo := 2 ;
                                       ' ts2.yy := 2009 ;

```

```
{ *  
 * how much seconds between this two dates ?  
 *}  
diff := Time_dateDiff(ts1, ts2) ; ' diff = 86163870
```

end.

TimeStruct type definition

```
type TimeStruct = record  
  ss as byte ; ' seconds  
  mn as byte ; ' minutes  
  hh as byte ; ' hours  
  md as byte ; ' day in month, from 1 to 31  
  wd as byte ; ' day in week, monday=0, tuesday=1, .... sun-  
day=6  
  mo as byte ; ' month number, from 1 to 12 (and not from 0  
to 11 as with UNIX C time !)  
  yy as word ; ' year Y2K compliant, from 1892 to 2038  
end;
```

TRIGONOMETRY LIBRARY

The mikroBasic for 8051 implements fundamental trigonometry functions. These functions are implemented as look-up tables. Trigonometry functions are implemented in integer format in order to save memory.

Library Routines

- sinE3
- cosE3

sinE3

Prototype	<code>sub function sinE3(dim angle_deg as word) as integer</code>
Returns	The function returns the sine of input parameter.
Description	<p>The function calculates sine multiplied by 1000 and rounded to the nearest integer:</p> <pre>result = round(sin(angle_deg)*1000)</pre> <p>Parameters:</p> <p><code>angle_deg</code>: input angle in degrees</p> <p>Note: Return value range: <code>-1000..1000</code>.</p>
Requires	Nothing.
Example	<pre>dim res as integer ... res = sinE3(45) ' result is 707</pre>

cosE3

Prototype	<code>sub function cosE3(dim angle_deg as word) as integer</code>
Returns	The function returns the cosine of input parameter.
Description	<p>The function calculates cosine multiplied by 1000 and rounded to the nearest integer:</p> <pre>result = round(cos(angle_deg)*1000)</pre> <p>Parameters:</p> <pre>angle_deg: input angle in degrees</pre> <p>Note: Return value range: -1000..1000.</p>
Requires	Nothing.
Example	<pre>dim res as integer ... res = cosE3(196) ' result is -193</pre>



MikroElektronika

SOFTWARE AND HARDWARE SOLUTIONS

FOR EMBEDDED WORLD

...making it simple

If you have any other question, comment or a business proposal, please contact us:

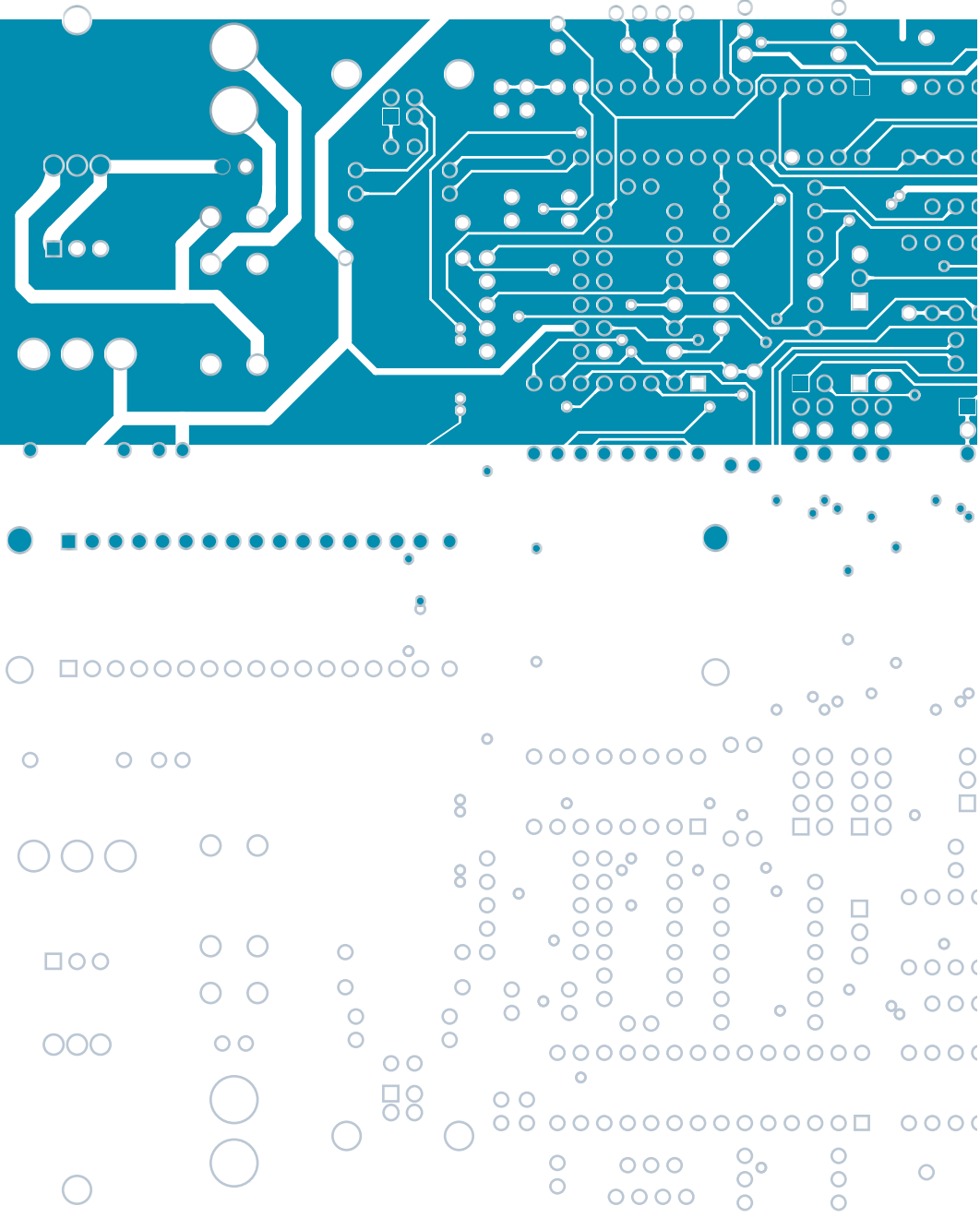
web: www.mikroe.com

e-mail: office@mikroe.com

If you are experiencing problems with any of our products
or you just want additional information, please let us know.

www.mikroe.com/en/support

TECHNICAL SUPPORT:



X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for [Development Software](#) category:

Click to view products by [MikroElektronika](#) manufacturer:

Other Similar products are found below :

[RAPPID-560XBSW](#) [RAPPID-567XFSW](#) [DG-ACC-NET-CD](#) [SRP004001-01](#) [SW006021-1NH](#) [SW163052](#) [SYSWINEV21](#) [Core429-SA](#)
[SW500006-HPA](#) [CWP-BASIC-FL](#) [W128E13](#) [CWP-PRO-FL](#) [SYSMACSE210L](#) [SYSMACSE203L](#) [AD-CCES-NODE-1](#) [NT-ZJCAT1-EV4](#)
[CWA-BASIC-FL](#) [RAPPID-567XKSW](#) [CWA-STANDARD-R](#) [SW89CN0-ZCC](#) [CWA-LS-DVLPR-NL](#) [VDSP-21XX-PCFLOAT](#) [RAPPID-](#)
[563XMSW](#) [IPS-EMBEDDED](#) [SWR-DRD-L-01](#) [SDAWIR-4532-01](#) [SYSMAC-SE201L](#) [MPROG-PRO535E](#) [AFLCF-08-LX-CE060-R21](#)
[WS02-CFSC1-EV3-UP](#) [SYSMAC-STUDIO-EIPCPLR](#) [LIB-PL-PC-N-1YR-DISKID](#) [SYSMACSE2XXL](#) [LS1043A-SWSP-PRM](#) [1120270005](#)
[1120270006](#) [MIKROBASIC PRO FOR FT90X \(USB DONGLE\)](#) [MIKROC PRO FOR AVR \(USB DONGLE LICENSE\)](#) [MIKROC PRO FOR](#)
[FT90X \(USB DONGLE\)](#) [MIKROBASIC PRO FOR AVR \(USB DONGLE LICEN](#) [MIKROBASIC PRO FOR FT90X](#) [MIKROC PRO FOR](#)
[DSPIC30/33 \(USB DONGLE LI](#) [MIKROC PRO FOR FT90X](#) [MIKROC PRO FOR PIC32 \(USB DONGLE LICENSE](#) [52202-588](#)
[MIKROPASCAL PRO FOR ARM \(USB DONGLE LICE](#) [MIKROPASCAL PRO FOR FT90X](#) [MIKROPASCAL PRO FOR FT90X \(USB](#)
[DONGLE\)](#) [MIKROPASCAL PRO FOR PIC32 \(USB DONGLE LI](#) [SW006021-2H](#)