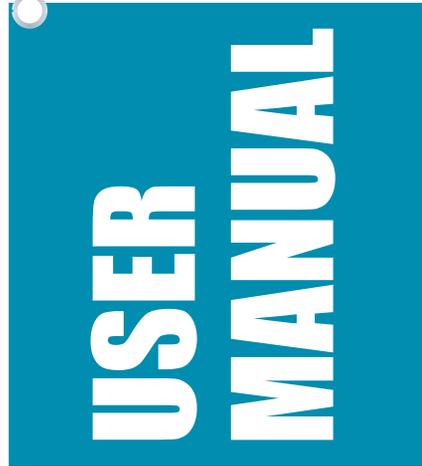
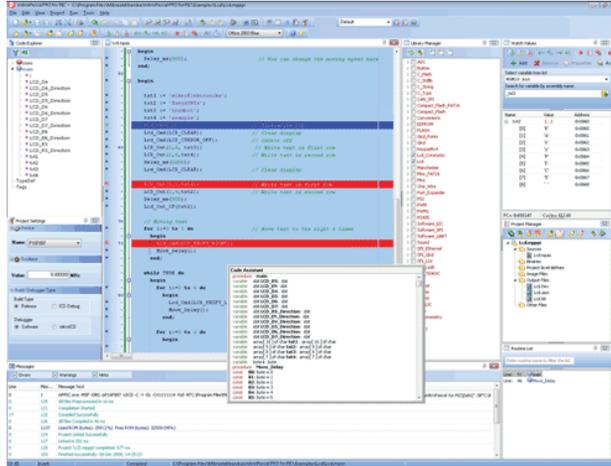
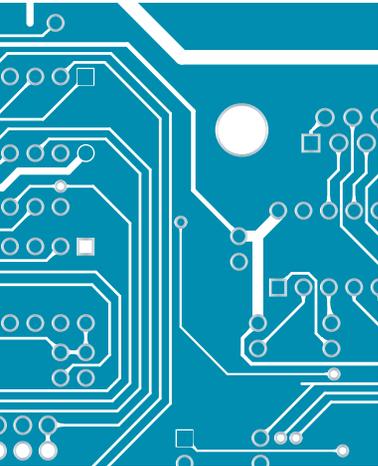


mikroPASCAL PRO for PIC



Develop your applications quickly and easily with the world's most intuitive mikroPascal PRO for PIC 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, mikroPascal PRO for PIC makes a fast and reliable tool, which can satisfy needs of experienced engineers and beginners alike.

April 2009.

Reader's note

DISCLAIMER:

mikroPASCAL PRO for PIC 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 *mikroPASCAL PRO for PIC* 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 *mikroPASCAL PRO for PIC* compiler, you agree to the terms of this agreement. Only one person may use licensed version of *mikroPascal PRO for PIC* compiler at a time. Copyright © mikroElektronika 2003 - 2009.

This manual covers *mikroPASCAL PRO for PIC* version 1.0 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 *mikroPASCAL PRO for PIC*
- 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	<i>mikroPASCAL PRO for PIC Environment</i>
CHAPTER 3	<i>mikroICD (In-Circuit Debugger)</i>
CHAPTER 4	<i>mikroPASCAL PRO for PIC Specifics</i>
CHAPTER 5	PIC Specifics
CHAPTER 6	<i>mikroPASCAL PRO for PIC Language Reference</i>
CHAPTER 7	<i>mikroPASCAL PRO for PIC Libraries</i>

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	7
Who Gets the License Key	7
How to Get License Key	7
After Receiving the License Key	9

CHAPTER 2

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
Regular expressions	18
View Menu Options	19
Toolbars	20
File Toolbar	20
Edit Toolbar	20
Advanced Edit Toolbar	21
Find/Replace Toolbar	21
Project Toolbar	22
Build Toolbar	22
Debugger	23
Styles Toolbar	23
Tools Toolbar	24

Project Menu Options	25
Run Menu Options	27
Tools Menu Options	28
Help Menu Options	29
Keyboard Shortcuts	30
IDE Overview	32
Customizing IDE Layout	33
Docking Windows	33
Saving Layout	34
Auto Hide	35
Advanced Code Editor	36
Advanced Editor Features	36
Code Assistant	37
Code Folding	38
Parameter Assistant	39
Code Templates (Auto Complete)	39
Auto Correct	39
Spell Checker	40
Bookmarks	40
Goto Line	40
Comment / Uncomment	40
Code Explorer	41
Routine List	42
Project Manager	43
Project Settings Window	44
Library Manager	45
Error Window	46
Statistics	47
Memory Usage Windows	47
RAM Memory Usage	47
Used RAM Locations	48
SFR Locations	49
ROM Memory Usage	50
ROM Memory Constants	51
Functions Sorted By Name	52
Functions Sorted By Size	53

Functions Sorted By Addresses	54
Functions Sorted By Name Chart	55
Functions Sorted By Size Chart	56
Functions Sorted By Addresses Chart	57
Function Tree	58
Memory Summary	59
Integrated Tools	60
USART Terminal	60
EEPROM Editor	61
ASCII Chart	62
Seven Segment Convertor	63
Lcd Custom Character	63
Graphic Lcd Bitmap Editor	64
HID Terminal	65
UDP Terminal	66
mikroBootloader	66
What is a Bootloader	66
How to use mikroBootloader	67
Features	67
Integrating User Code and Boot Code	68
Macro Editor	69
Options	70
Code editor	70
Tools	70
Output settings	71
Regular Expressions	72
Introduction	72
Simple matches	72
Escape sequences	72
Character classes	73
Metacharacters	73
Metacharacters - Line separators	73
Metacharacters - Predefined classes	74
Metacharacters - Word boundaries	74
Metacharacters - Iterators	75
Metacharacters - Alternatives	76

Metacharacters - Subexpressions	76
Metacharacters - Backreferences	76
mikroPascal PRO for PIC Command Line Options	77
Projects	78
New Project	78
New Project Wizard Steps	79
Customizing Projects	82
Managing Project Group	82
Add/Remove Files from Project	82
The project can contain the fol	82
Project Level Defines	83
Source Files	84
Managing Source Files	84
Creating new source file	84
Opening an existing file	84
Printing an open file	84
Saving file	85
Saving file under a different name	85
Closing file	85
Clean Project Folder	86
Compilation	87
Output Files	87
Assembly View	87
Error Messages	88
Compiler Error Messages:	88
Warning Messages:	90
Hint Messages:	90
Software Simulator Overview	91
Breakpoints Window	92
Watch Window	92
View RAM Window	94
Stopwatch Window	95
Software Simulator Options	96
Creating New Library	97
Multiple Library Versions	98

CHAPTER 3

mikroICD Debugger Options	101
mikroICD Debugger Example	102
mikroICD (In-Circuit Debugger) Overview	106
Breakpoints Window	106
Watch Window	106
EEPROM Watch Window	107
Code Watch Window	108
View RAM Window	109
Common Errors	109
mikroICD Advanced Breakpoints	110
mikroICD provides the possibili	110
Program Memory Break	111
File Register Break	111
Emulator Features	111
Event Breakpoints	111
Stopwatch	111

CHAPTER 4

Pascal Standard Issues	114
Divergence from the Pascal Standard	114
Pascal Language Extensions	114
Predefined Globals and Constants	115
SFRs and related constants	115
Math constants	115
Predefined project level defines	115
Accessing Individual Bits	116
Accessing Individual Bits Of Variables	116
sbit type	117
bit type	117
Interrupts	118
P18 priority interrupts	118
Routine Calls from Interrupt	118
Interrupt Examples	119

Linker Directives	120
Directive absolute	120
Directive org	121
Directive orgall	121
Built-in Routines	122
Lo	123
Hi	123
Higher	123
Highest	124
Inc	124
Dec	124
SetBit	125
ClearBit	125
TestBit	125
Delay_us	126
Delay_ms	126
Clock_KHz	126
Clock_MHz	127
Reset	127
ClrWdt	127
DisableContextSaving	128
SetFuncCall	128
GetDateTime	129
GetVersion	129
Code Optimization	130
Constant folding	130
Constant propagation	130
Copy propagation	130
Value numbering	130
"Dead code" elimination	130
Stack allocation	130
Local vars optimization	130
Better code generation and local optimization	130

CHAPTER 5

Types Efficiency	132
Nested Calls Limitations	132
PIC18FxxJxx Specifics	133
Shared Address SFRs	133
PIC16 Specifics	133
Breaking Through Pages	133
Limits of Indirect Approach Through FSR	133
Memory Type Specifiers	134
code	134
data	134
rx	134
sfr	135

CHAPTER 6

Lexical Elements Overview	139
Whitespace	140
Whitespace in Strings	140
Comments	141
Nested comments	141
Tokens	142
Token Extraction Example	142
Literals	143
Integer Literals	143
Floating Point Literals	143
Character Literals	144
String Literals	144
Keywords	145
Identifiers	146
Case Sensitivity	146
Uniqueness and Scope	146
Identifier Examples	146
Punctuators	147
Brackets	147

Parentheses	147
Comma	147
Semicolon	148
Colon	148
Dot	148
Program Organization	149
Organization of Main Unit	149
Organization of Other Units	150
Scope and Visibility	152
Scope	152
Visibility	152
Units	153
Uses Clause	153
Main Unit	154
Other Units	154
Interface Section	154
Implementation Section	155
Variables	156
External Modifier	156
Variables and PIC	156
Constants	157
Labels	158
Functions and Procedures	159
Functions	159
Calling a function	159
Example	160
Procedures	160
Calling a procedure	161
Example	161
Example:	161
Forward declaration	162
Functions reentrancy	162
Types	163
Type Categories	163
Simple Types	164
Arrays	165

Array Declaration	165
Constant Arrays	165
Multi-dimensional Arrays	166
Strings	166
String Concatenating	167
Note	167
Pointers	168
Function Pointers	168
Example:	168
@ Operator	170
Records	171
Accessing Fields	172
Types Conversions	173
Implicit Conversion	173
Promotion	173
Clipping	173
Explicit Conversion	174
Conversions Examples	174
Operators	175
Operators Precedence and Associativity	175
Arithmetic Operators	176
Division by Zero	176
Unary Arithmetic Operators	176
Relational Operators	177
Relational Operators in Expressions	177
Bitwise Operators	178
Bitwise Operators Overview	178
Logical Operations on Bit Level	178
Bitwise operators and, or, and xor perform logical operation	178
Unsigned and Conversions	179
Signed and Conversions	179
Bitwise Shift Operators	180
Boolean Operators	180
Expressions	181
Statements	182
Assignment Statements	182

Compound Statements (Blocks)	183
Conditional Statements	183
If Statement	184
Nested if statements	184
Case statement	185
Nested Case statement	186
Iteration Statements	187
For Statement	187
Endless Loop	187
While Statement	188
Repeat Statement	188
Jump Statements	189
Break and Continue Statements	189
Break Statement	189
Continue Statement	190
Exit Statement	190
Goto Statement	191
asm Statement	192
Directives	192
Compiler Directives	193
Directives \$DEFINE and \$UNDEFINE	193
Directives \$IFDEF, \$IFNDEF, \$ELSE	193
Include Directive \$I	194
Predefined Flags	194
Linker Directives	195
Directive absolute	195
Directive org	195
Directive orgall	196

CHAPTER 7

Hardware PIC-specific Libraries	198
Miscellaneous Libraries	198
Library Dependencies	199
Hardware Libraries	200
ADC Library	201

ADC_Read	201
Library Example	202
HW Connection	202
CAN Library	203
Library Routines	203
CANSetOperationMode	204
CANGetOperationMode	204
CANInitialize	205
CANSetBaudRate	206
CANSetMask	207
CANSetFilter	208
CANRead	209
CANWrite	210
CAN Constants	210
CAN_CONFIG_FLAGS	211
CAN_TX_MSG_FLAGS	212
CAN_RX_MSG_FLAGS	212
CAN_MASK	213
CAN_FILTER	213
Library Example	213
HW Connection	216
CANSPI Library	217
External dependencies of CANSPI Library	218
Library Routines	218
CANSPISetOperationMode	219
CANSPIGetOperationMode	220
CANSPIInitialize	220
CANSPISetBaudRate	222
CANSPISetMask	223
CANSPISetFilter	224
CANSPIRead	225
CANSPIWrite	226
CANSPI Constants	226
CANSPI_OP_MODE	227
CANSPI_CONFIG_FLAGS	227
CANSPI_TX_MSG_FLAGS	228

CANSPI_RX_MSG_FLAGS	228
CANSPI_MASK	229
CANSPI_FILTER	229
Library Example	230
HW Connection	233
Compact Flash Library	234
External dependencies of Compact Flash Library	235
Library Routines	236
Cf_Init	237
Cf_Detect	238
Cf_Enable	238
Cf_Disable	238
Cf_Read_Init	239
Cf_Read_Byte	239
Cf_Write_Init	240
Cf_Write_Byte	240
Cf_Read_Sector	241
Cf_Write_Sector	241
Cf_Fat_Init	242
Cf_Fat_QuickFormat	242
Cf_Fat_Assign	243
Cf_Fat_Reset	244
Cf_Fat_Read	244
Cf_Fat_Rewrite	245
Cf_Fat_Append	245
Cf_Fat_Delete	245
Cf_Fat_Write	246
Cf_Fat_Set_File_Date	246
Cf_Fat_Get_File_Date	247
Cf_Fat_Get_File_Size	247
Cf_Fat_Get_Swap_File	248
Library Example	250
HW Connection	255
EEPROM Library	256
Library Routines	256
EEPROM_Read	256

EEPROM_Write	256
Library Example	257
Ethernet PIC18FxxJ60 Library	258
Library Routines	259
Ethernet_Init	260
Ethernet_Enable	261
Ethernet_Disable	262
Ethernet_doPacket	263
Ethernet_putByte	264
Ethernet_putBytes	264
Ethernet_putConstBytes	265
Ethernet_putString	265
Ethernet_putConstString	266
Ethernet_getByte	266
Ethernet_getBytes	267
Ethernet_UserTCP	268
Ethernet_UserUDP	269
Ethernet_getIpAddress	270
Ethernet_getGwIpAddress	270
Ethernet_getDnsIpAddress	271
Ethernet_getIpMask	271
Ethernet_confNetwork	272
Ethernet_arpResolve	273
Ethernet_sendUDP	274
Ethernet_dnsResolve	275
Ethernet_initDHCP	276
Ethernet_doDHCPLeaseTime	277
Ethernet_renewDHCP	277
Library Example	278
Flash Memory Library	286
Library Routines	286
FLASH_Read	287
FLASH_Read_N_Bytes	287
FLASH_Write	288
FLASH_Erase	289
FLASH_Erase_Write	289

Library Example	290
Graphic Lcd Library	292
External dependencies of Graphic Lcd Library	292
Library Routines	293
Glcd_Init	294
Glcd_Set_Side	295
Glcd_Set_X	295
Glcd_Set_Page	296
Glcd_Read_Data	296
Glcd_Write_Data	297
Glcd_Fill	297
Glcd_Dot	298
Glcd_Line	298
Glcd_V_Line	299
Glcd_H_Line	299
Glcd_Rectangle	300
Glcd_Box	300
Glcd_Circle	301
Glcd_Set_Font	301
Glcd_Write_Char	302
Glcd_Write_Text	303
Glcd_Image	303
Library Example	304
HW Connection	306
I ² C Library	307
Library Routines	307
I2C1_Init	307
I2C1_Start	308
I2C1_Repeated_Start	308
I2C1_Is_Idle	308
I2C1_Rd	308
I2C1_Wr	309
I2C1_Stop	309
Library Example	309
HW Connection	310
Keypad Library	311

External dependencies of Keypad Library	311
Library Routines	311
Keypad_Init	311
Keypad_Key_Press	312
Keypad_Key_Click	312
Library Example	312
This is a simple example of using	312
the Keypad Librar	312
HW Connection	315
Lcd Library	316
External dependencies of Lcd Library	316
Library Routines	317
Lcd_Init	317
Lcd_Out	318
Lcd_Out_CP	318
Lcd_Chr	319
Lcd_Chr_CP	319
Lcd_Cmd	320
Available Lcd Commands	320
Library Example	321
HW connection	323
Lcd HW connecti	323
Manchester Code Library	324
External dependencies of Manchester Code Library	324
Library Routines	325
Man_Receive_Init	325
Man_Receive	326
Man_Send_Init	326
Man_Send	327
Man_Synchro	327
Man_Break	328
Library Example	329
Connection Example	332
Multi Media Card Library	333
Secure Digital Card	333
External dependencies of MMC Library	334

Library Routines	334
Mmc_Init	335
Mmc_Read_Sector	336
Mmc_Write_Sector	336
Mmc_Read_Cid	337
Mmc_Read_Csd	337
Mmc_Fat_Init	338
Mmc_Fat_QuickFormat	339
Mmc_Fat_Assign	340
Mmc_Fat_Reset	341
Mmc_Fat_Read	341
Mmc_Fat_Rewrite	342
Mmc_Fat_Append	342
Mmc_Fat_Delete	342
Mmc_Fat_Write	343
Mmc_Fat_Set_File_Date	344
Mmc_Fat_Get_File_Date	345
Mmc_Fat_Get_File_Size	345
Mmc_Fat_Get_Swap_File	346
Library Example	347
The following exa	347
HW Connection	350
OneWire Library	351
Library Routines	351
Ow_Reset	352
Ow_Read	352
Ow_Write	352
Library Example	352
This example reads the tem	352
erature using DS	352
HW Connection	355
Port Expander Library	356
External dependencies of Port Expander Library	356
Library Routines	356
Expander_Init	357
Expander_Read_Byte	358

Expander_Write_Byte	358
Expander_Read_PortA	359
Expander_Read_PortB	360
Expander_Read_PortAB	361
Expander_Write_PortA	362
Expander_Write_PortB	363
Expander_Write_PortAB	364
Expander_Set_DirectionPortA	365
Expander_Set_DirectionPortB	365
Expander_Set_DirectionPortAB	366
Expander_Set_PullUpsPortA	366
Expander_Set_PullUpsPortB	367
Expander_Set_PullUpsPortAB	367
Library Example	368
HW Connection	369
PS/2 Library	370
External dependencies of PS/2 Library	370
Library Routines	370
Ps2_Config	371
Ps2_Key_Read	372
Special Function Keys	373
Library Example	374
HW Connection	375
PWM Library	376
Library Routines	376
PWM1_Init	376
PWM1_Set_Duty	377
PWM1_Start	377
PWM1_Stop	377
Library Example	377
HW Connection	379
RS-485 Library	380
External dependencies of RS-485 Library	380
Library Routines	381
RS485Master_Init	381
RS485Master_Receive	382

RS485Master_Send	383
RS485Slave_Init	384
RS485slave_Receive	385
RS485Slave_Send	386
Library Example	386
HW Connection	390
Message format and CRC calculations	391
Software I ² C Library	392
External dependencies of Software I ² C Library	392
Library Routines	392
Soft_I2C_Init	393
Soft_I2C_Start	393
Soft_I2C_Read	394
Soft_I2C_Write	394
Soft_I2C_Stop	395
Soft_I2C_Break	395
Library Example	396
Software SPI Library	399
External dependencies of Software SPI Library	399
Library Routines	399
Soft_SPI_Init	400
Soft_SPI_Read	401
Soft_SPI_Write	401
Library Example	401
Software UART Library	404
Library Routines	404
Soft_UART_Init	405
Soft_UART_Read	406
Soft_UART_Write	407
Soft_UART_Break	408
Library Example	409
Sound Library	410
Library Routines	410
Sound_Init	410
Sound_Play	411
Library Example	411

HW Connection	414
SPI Library	415
Library Routines	415
SPI1_Init	415
SPI1_Init_Advanced	416
SPI1_Read	417
SPI1_Write	417
SPI_Set_Active	417
Library Example	418
HW Connection	419
SPI Ethernet Library	420
External dependencies of SPI Ethernet Library	421
Library Routines	422
PIC16 and PIC18:	422
PIC18 Only:	422
SPI_Ethernet_Init	423
SPI_Ethernet_Enable	425
SPI_Ethernet_Disable	426
SPI_Ethernet_doPacket	427
SPI_Ethernet_putByte	428
SPI_Ethernet_putBytes	428
SPI_Ethernet_putConstBytes	429
SPI_Ethernet_putString	429
SPI_Ethernet_putConstString	430
SPI_Ethernet_getByte	430
SPI_Ethernet_getBytes	431
SPI_Ethernet_UserTCP	432
SPI_Ethernet_UserUDP	433
SPI_Ethernet_getIpAddress	434
SPI_Ethernet_getGwIpAddress	434
SPI_Ethernet_getDnsIpAddress	435
SPI_Ethernet_getIpMask	435
SPI_Ethernet_confNetwork	436
SPI_Ethernet_arpResolve	437
SPI_Ethernet_sendUDP	438
SPI_Ethernet_dnsResolve	439

SPI_Ethernet_initDHCP	440
SPI_Ethernet_doDHCPLeaseTime	441
SPI_Ethernet_renewDHCP	441
Library Example	442
HW Connection	449
SPI Graphic Lcd Library	450
External dependencies of SPI Graphic Lcd Library	450
Library Routines	450
SPI_Glcd_Init	451
SPI_Glcd_Set_Side	452
SPI_Glcd_Set_Page	452
SPI_Glcd_Set_X	453
SPI_Glcd_Read_Data	453
SPI_Glcd_Write_Data	454
SPI_Glcd_Fill	454
SPI_Glcd_Dot	455
SPI_Glcd_Line	455
SPI_Glcd_V_Line	456
SPI_Glcd_H_Line	456
SPI_Glcd_Rectangle	457
SPI_Glcd_Box	458
SPI_Glcd_Circle	458
SPI_Glcd_Set_Font	459
SPI_Glcd_Write_Char	460
SPI_Glcd_Write_Text	461
SPI_Glcd_Image	462
Library Example	462
HW Connection	465
SPI Lcd Library	466
External dependencies of SPI Lcd Library	466
Library Routines	466
SPI_Lcd_Config	467
SPI_Lcd_Out	468
SPI_Lcd_Out_CP	468
SPI_Lcd_Chr	469
SPI_Lcd_Chr_CP	469

SPI_Lcd_Cmd	470
Available SPI Lcd Commands	470
Library Example	471
HW Connection	472
SPI Lcd8 (8-bit interface) Library	473
External dependencies of SPI Lcd Library	473
Library Routines	473
SPI_Lcd8_Config	474
SPI_Lcd8_Out	475
SPI_Lcd8_Out_CP	475
SPI_Lcd8_Chr	476
SPI_Lcd8_Chr_CP	476
SPI_Lcd8_Cmd	477
Available SPI Lcd8 Commands	477
Library Example	478
HW Connection	479
SPI T6963C Graphic Lcd Library	480
External dependencies of SPI T6963C Graphic Lcd Library	480
Library Routines	481
SPI_T6963C_Config	482
SPI_T6963C_WriteData	483
SPI_T6963C_WriteCommand	484
SPI_T6963C_SetPtr	484
SPI_T6963C_WaitReady	484
SPI_T6963C_Fill	485
SPI_T6963C_Dot	485
SPI_T6963C_Write_Char	486
SPI_T6963C_Write_Text	487
SPI_T6963C_Line	488
SPI_T6963C_Rectangle	488
SPI_T6963C_Box	489
SPI_T6963C_Circle	489
SPI_T6963C_Image	490
SPI_T6963C_Sprite	490
SPI_T6963C_Set_Cursor	491
SPI_T6963C_ClearBit	491

SPI_T6963C_SetBit	491
SPI_T6963C_NegBit	492
SPI_T6963C_DisplayGrPanel	492
SPI_T6963C_DisplayTxtPanel	493
SPI_T6963C_SetGrPanel	493
SPI_T6963C_SetTxtPanel	494
SPI_T6963C_PanelFill	494
SPI_T6963C_GrFill	494
SPI_T6963C_TxtFill	495
SPI_T6963C_Cursor_Height	495
SPI_T6963C_Graphics	495
SPI_T6963C_Text	496
SPI_T6963C_Cursor	496
SPI_T6963C_Cursor_Blink	496
Library Example	497
HW Connection	502
T6963C Graphic Lcd Library	503
External dependencies of T6963C Graphic Lcd Library	504
Library Routines	505
T6963C_Init	506
T6963C_WriteData	507
T6963C_WriteCommand	507
T6963C_SetPtr	508
T6963C_WaitReady	508
T6963C_Fill	508
T6963C_Dot	509
T6963C_Write_Char	509
T6963C_Write_Text	510
T6963C_Line	511
T6963C_Rectangle	511
T6963C_Box	512
T6963C_Circle	512
T6963C_Image	513
T6963C_Sprite	513
T6963C_Set_Cursor	514
T6963C_DisplayGrPanel	514

T6963C_DisplayTxtPanel	514
T6963C_SetGrPanel	515
T6963C_SetTxtPanel	515
T6963C_PanelFill	515
T6963C_GrFill	516
T6963C_TxtFill	516
T6963C_Cursor_Height	516
T6963C_Graphics	517
T6963C_Text	517
T6963C_Cursor	517
T6963C_Cursor_Blink	518
Library Example	518
HW Connection	523
UART Library	524
Library Routines	524
UART1_Init	525
UART1_Data_Ready	526
UART1_Tx_Idle	526
UART1_Read	527
UART1_Read_Text	527
UART1_Write	528
UART1_Write_Text	528
UART_Set_Active	529
Library Example	529
HW Connection	530
USB HID Library	531
Descriptor File	531
Library Routines	531
Hid_Enable	531
Hid_Read	532
Hid_Write	532
Hid_Disable	532
Library Example	532
HW Connection	534
Miscellaneous Libraries	535
Button Library	535

Button	535
Conversions Library	536
Library Routines	536
ByteToStr	536
ShortToStr	537
WordToStr	537
IntToStr	538
LongintToStr	538
LongWordToStr	539
FloatToStr	540
StrToInt	541
StrToWord	541
Dec2Bcd	541
Bcd2Dec16	542
Dec2Bcd16	542
Math Library	543
Library Functions	543
acos	543
asin	543
atan	544
atan2	544
ceil	544
cos	544
cosh	544
eval_poly	544
exp	544
fabs	545
floor	545
frexp	545
ldexp	545
log	545
log10	545
modf	545
pow	546
sin	546
sinh	546

sqrt	546
tan	546
tanh	546
String Library	547
Library Functions	547
memchr	547
memcmp	548
memcpy	548
memmove	548
memset	549
strcat	549
strchr	549
strcmp	549
strcpy	550
strcspn	550
strlen	550
strncat	550
strncmp	550
strncpy	551
strpbrk	551
strrchr	551
strspn	551
strstr	551
Time Library	552
Library Routines	552
Time_dateToEpoch	552
Time_epochToDate	553
Time_dateDiff	553
Library Example	554
TimeStruct type definition	554
Trigonometry Library	555
Library Routines	555
sinE3	555
cosE3	556

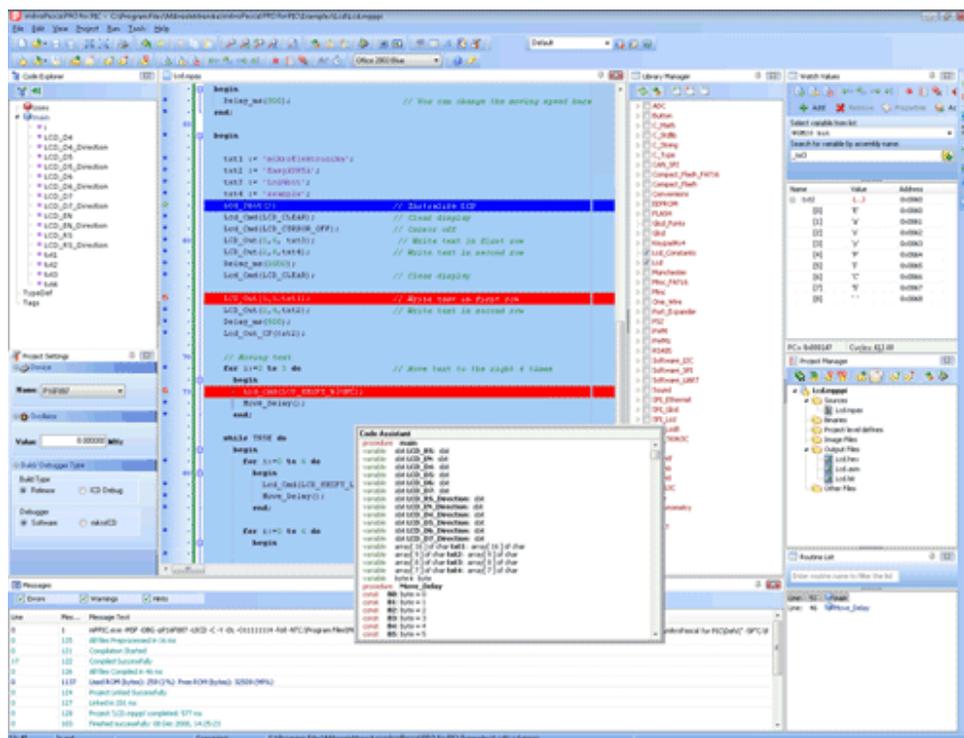
CHAPTER

1

Introduction to mikroPascal PRO for PIC

Help version: 2009/03/19

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



mikroPascal PRO for PIC IDE

Features

mikroPascal PRO for PIC allows you to quickly develop and deploy complex applications:

- Write your Pascal source code using the built-in Code Editor (Code and Parameter Assistants, Code Folding, Syntax Highlighting, Auto Correct, Code Templates, and more.)
- Use included mikroPascal PRO for PIC 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.
- Use the integrated mikroICD (In-Circuit Debugger) Real-Time debugging tool to monitor program execution on the hardware level.
- 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.
- mikroPascal PRO for PIC 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 PIC microcontrollers, read carefully the PIC Specifics chapter. It might give you some useful pointers on PIC constraints, code portability, and good programming practices.
- If you are experienced in Pascal programming, you will probably want to consult mikroPascal PRO for PIC Specifics first. For language issues, you can always refer to the comprehensive Language Reference. A complete list of included libraries is available at mikroPascal PRO for PIC Libraries.
- If you are not very experienced in Pascal programming, don't panic! mikroPascal PRO for PIC 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

Visegradska 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 mikroPascal PRO for PIC 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 mikroPascal PRO for PIC 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 mikroPascal PRO for PIC, then you should consider the possibility of purchasing the license key.

Who Gets the License Key

Buyers of the mikroPascal PRO for PIC are entitled to the license key. After you have completed the payment procedure, you have an option of registering your mikroPascal. 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 Jovanovic
ADDRESS	Enter your address
INVOICE	Enter invoice number if available in the form AAAAA/BB
2CO Number	Enter 2CheckOut Order Number if available (10 digits)
E-MAIL*	marko@mikroe.com
E-MAIL*	marko@mikroe.com
COMPANY	Enter company name
PRODUCT ID	515C-557269-6F6D72-6755
COMMENTS:	Enter comments on your order
DISTRIBUTOR*	<div style="display: flex; justify-content: space-between;"> mikroElektronika key@mikroe.com </div>

* Required fields

I have made the payment and I wish to request activation key for mikroPascal PRO for PIC

Name:
Marko Jovanovic

Address:

Invoice number:

2CheckOut order number:

Company:

E-Mail:
marko@mikroe.com

Product key:
515C-557269-6F6D72-6755

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 mikroPascal PRO for PIC 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.**

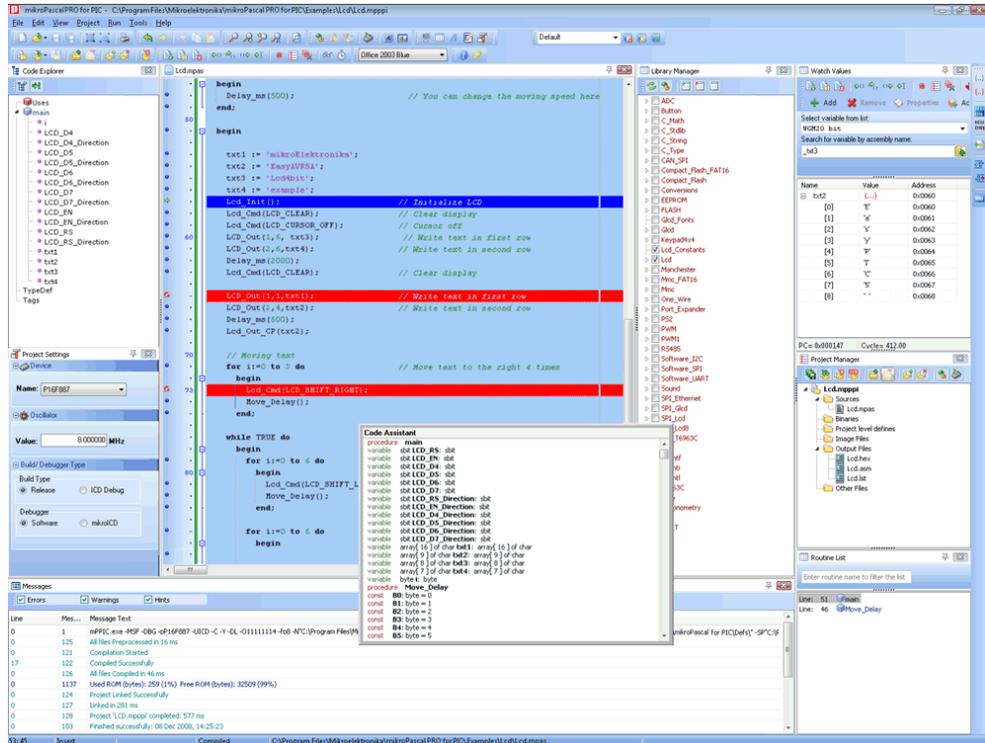
CHAPTER

2

mikroPASCAL PRO for PIC Environment

IDE OVERVIEW

The mikroPascal PRO for PIC 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 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 mikroPascal PRO for PIC 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

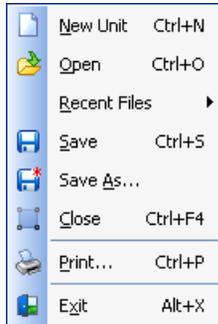
Tools

Help

Related topics: Keyboard shortcuts

FILE MENU OPTIONS

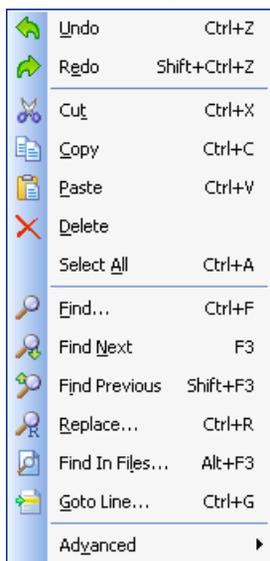
The File menu is the main entry point for manipulation with the source files.



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

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

EDIT MENU OPTIONS

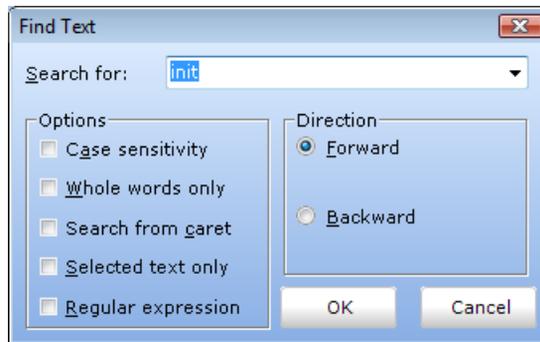


Edit	Description
 Undo Ctrl+Z	Undo last change.
 Redo Shift+Ctrl+Z	Redo last change.
 Cut Ctrl+X	Cut selected text to clipboard.
 Copy Ctrl+C	Copy selected text to clipboard.
 Paste Ctrl+V	Paste text from clipboard.
 Delete	Delete selected text.
Select All Ctrl+A	Select all text in active editor.
 Find... 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>a</u> Lowercase Ctrl+Alt+L	Changes selected text case to lowercase.
 <u>A</u> Uppercase Ctrl+Alt+U	Changes selected text case to uppercase.
 <u>T</u> itlecase Ctrl+Alt+T	Changes selected text case to titlecase.

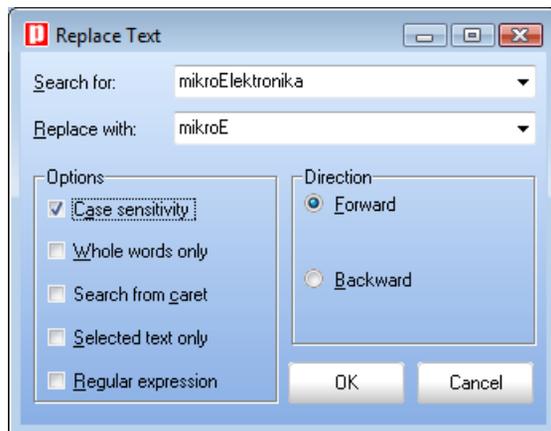
Find Text

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

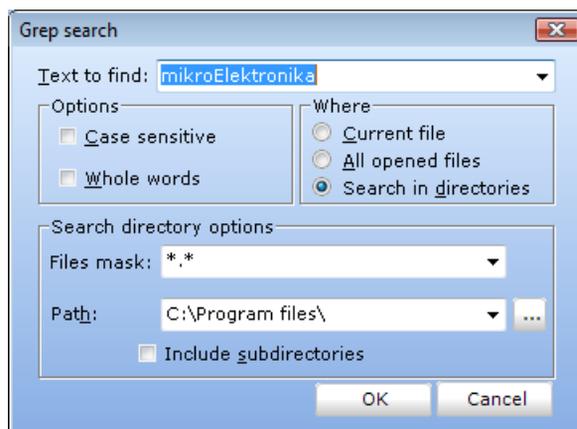
Dialog box for searching for a text string in file and replacing it with another text string.



Find In Files

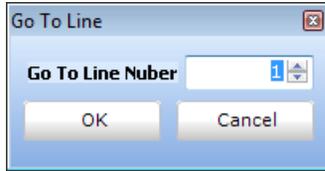
Dialog box for searching for a text string in current file, all opened files, or in files on a disk.

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.



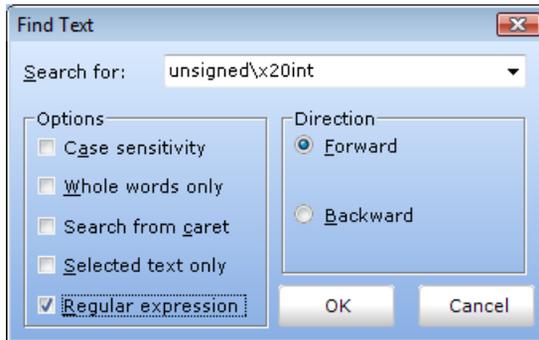
Go To Line

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



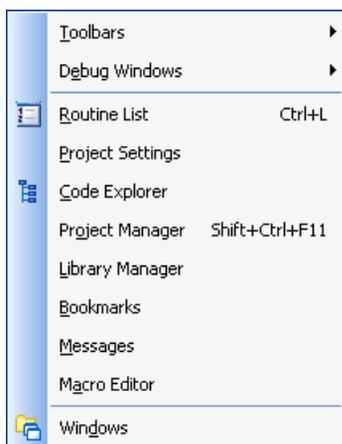
Regular expressions

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 Software Simulator/mikroICD (In-Circuit Debugger) 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
	New project.
	Open Project
	Save Project
	Close current project.
	Edit project settings.
	Add existing project to project group.
	Remove existing project from project group.
	Add File To Project
	Remove File From 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.
	Open listing file in editor.
	View statistics for current project.

Debugger



Debugger Toolbar comes with following options:

Icon	Description
	Start Software Simulator or mikroICD (In-Circuit Debugger).
	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



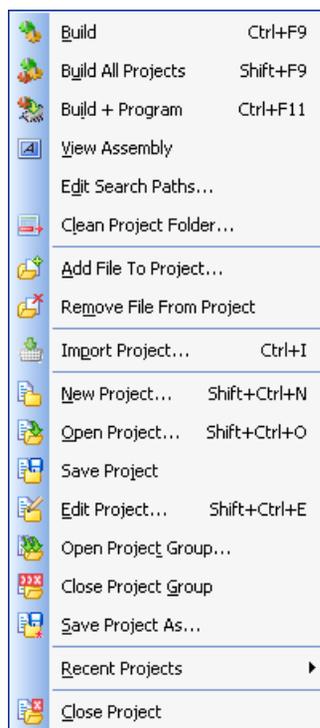
Tools Toolbar comes with following default options:

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

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
 <u>B</u> uild Ctrl+F9	Build active project.
 <u>B</u> uild All Projects Shift+F9	Build all projects.
 <u>B</u> uild + Program Ctrl+F11	Build and program active project.
 <u>V</u> iew Assembly	View Assembly.
<u>E</u> dit Search Paths...	Edit search paths.
 <u>C</u> lean Project Folder...	Clean Project Folder
 <u>A</u> dd File To Project...	Add file to project.
 <u>R</u> emove File From Project	Remove file from project.
 <u>I</u> mport Project... Ctrl+I	Import projects created in previous versions of mikroPascal.
 <u>N</u> ew Project...	Open New Project Wizard
 <u>O</u> pen Project... Shift+Ctrl+O	Open existing project.
 <u>S</u> ave Project	Save current project.
 <u>E</u> dit Project... Shift+Ctrl+E	Edit project settings
 <u>O</u> pen Project Group...	Open project group.
 <u>C</u> lose Project Group	Close project group.
 <u>S</u> ave Project As...	Save active project file with the different name.
<u>R</u> ecent Projects ►	Open recently used project.
 <u>C</u> lose Project	Close active project.

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

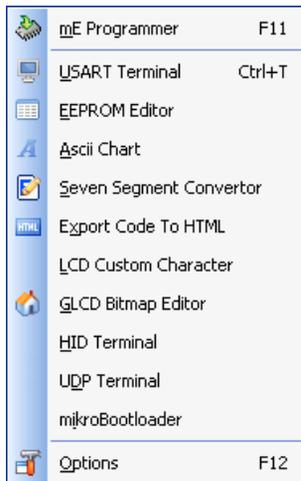
RUN MENU OPTIONS

	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 or mikroICD (In-Circuit Debugger).
 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 Pascal source and disassembly.

Related topics: Keyboard shortcuts, Debug Toolbar

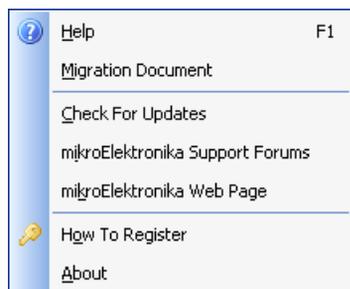
TOOLS MENU OPTIONS



Tools	Description
 m E Programmer F11	Run mikroElektronika Programmer
 U SART Terminal Ctrl+T	Run USART Terminal
 E EPR O M Editor	Run EEPROM Editor
 A scii Chart	Run ASCII Chart
 S even Segment Convertor	Run Seven Segment Convertor
 E xport Code To HTML	Generate HTML code suitable for publishing source code on the web.
L CD Custom Character	Run Lcd custom character
 G LCD Bitmap Editor	Run Glcd bitmap editor
H ID Terminal	Run HID Terminal
U DP Terminal	Run UDP communication terminal
m ikro B ootloader	Run mikroBootloader
 O ptions F12	Open Options window

Related topics: Keyboard shortcuts, Tools Toolbar

HELP MENU OPTIONS



File	Description
 <u>H</u> elp F1	Open Help File.
<u>M</u> igration Document	Open Code Migration Document.
<u>C</u> heck For Updates	Check if new compiler version is available.
<u>m</u> ikroElektronika Support Forums	Open mikroElektronika Support Forums in a default browser.
<u>m</u> ikroElektronika Web Page	Open mikroElektronika Web Page in a default browser.
 <u>H</u> ow To Register	Information on how to register
<u>A</u> bout	Open About window.

Related topics: Keyboard shortcuts

KEYBOARD SHORTCUTS

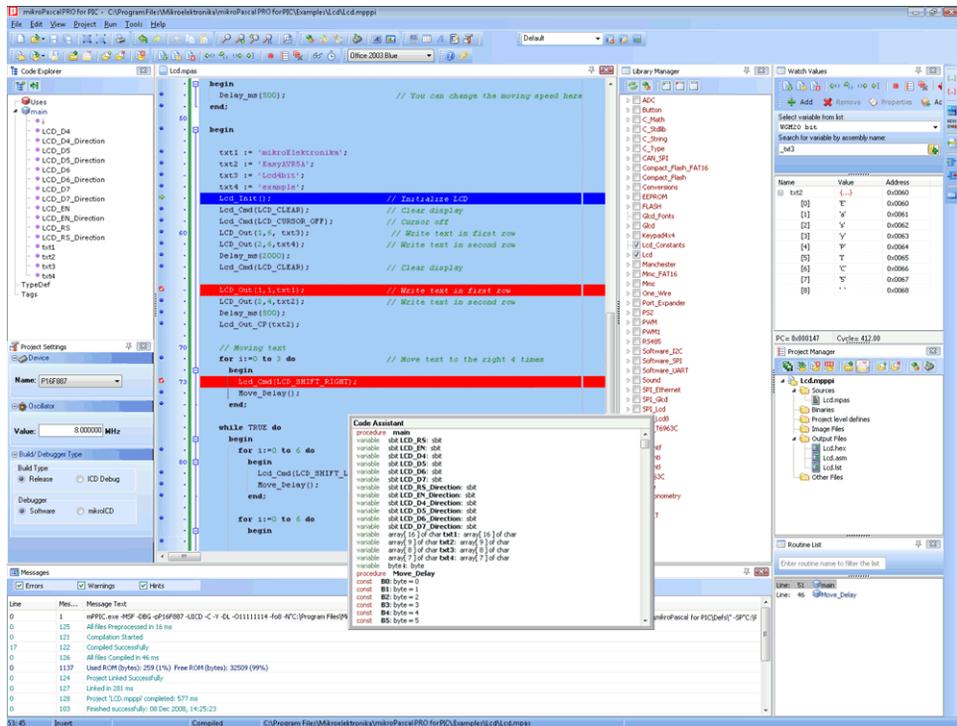
Below is a complete list of keyboard shortcuts available in mikroPascal PRO for PIC IDE.

IDE Shortcuts		Ctrl+S	Save Unit
F1	Help	Ctrl+Shift+S	Save All
Ctrl+N	New Unit	Ctrl+V	Paste
Ctrl+O	Open	Ctrl+X	Cut
Ctrl+Shift+O	Open Project	Ctrl+Y	Delete Entire Line
Ctrl+Shift+N	New Project	Ctrl+Z	Undo
Ctrl+K	Close Project	Ctrl+Shift+Z	Redo
Ctrl+F4	Close unit	Advanced Editor Shortcuts	
Ctrl+Shift+E	Edit Project	Ctrl+Space	Code Assistant
Ctrl+F9	Build	Ctrl+Shift+Space	Parameters Assistant
Shift+F9	Build All	Ctrl+D	Find Declaration
Ctrl+F11	Build And Program	Ctrl+E	Incremental Search
Shift+F4	View Breakpoints	Ctrl+L	Routine List
Ctrl+Shift+F5	Clear Breakpoints	Ctrl+G	Goto Line
F11	Start mE Programmer	Ctrl+J	Insert Code Template
Ctrl+Shift+F11	Project Manager	Ctrl+Shift+.	Comment Code
F12	Options	Ctrl+Shift+,	Uncomment Code
Alt + X	Close mikroPascal PRO for PIC	Ctrl+number	Goto Bookmark
Basic Editor Shortcuts		Ctrl+Shift+number	Set Bookmark
F3	Find, Find Next	Ctrl+Shift+I	Indent Selection
Shift+F3	Find Previous	Ctrl+Shift+U	Unindent Selection
Alt+F3	Grep Search, Find In Files	TAB	Indent Selection
Ctrl+A	Select All	Shift+TAB	Unindent selection
Ctrl+C	Copy	Alt+Select	Select Columns
Ctrl+F	Find	Ctrl+Alt+Select	Select Columns
Ctrl+R	Replace	Ctrl+Alt+L	Convert Selection to Lowercase
Ctrl+P	Print	Ctrl+Alt+U	Convert Selection to Uppercase

Ctrl+Alt+T	Convert to Titlecase
mikroICD Debugger and 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	Stop Debugger
Ctrl+F5	Add to Watch List
Ctrl+F8	Step Out
Alt+D	Dissassembly View
Shift+F5	Open Watch Window
Ctrl+Shift+A	Show Advanced Breakpoints

IDE OVERVIEW

The mikroPascal PRO for PIC 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 typoes and Code Templates (Auto Complete).
- The Code Explorer 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 mikroPascal PRO for PIC 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

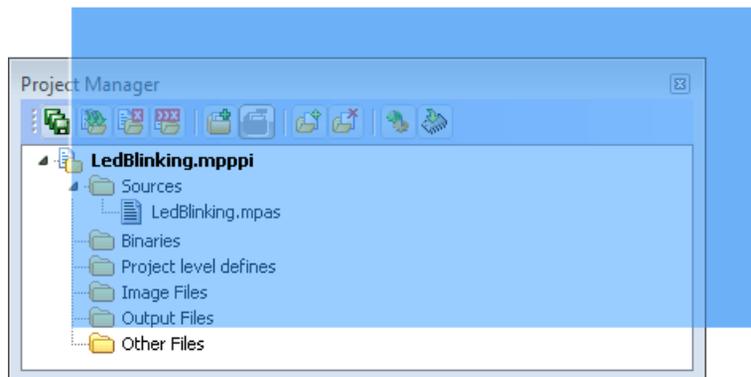
Docking Windows

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

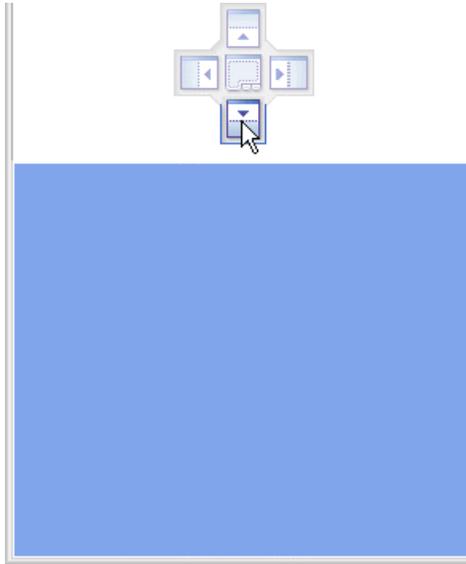
Step 1: Click the window you want to dock, to give it focus.



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 .

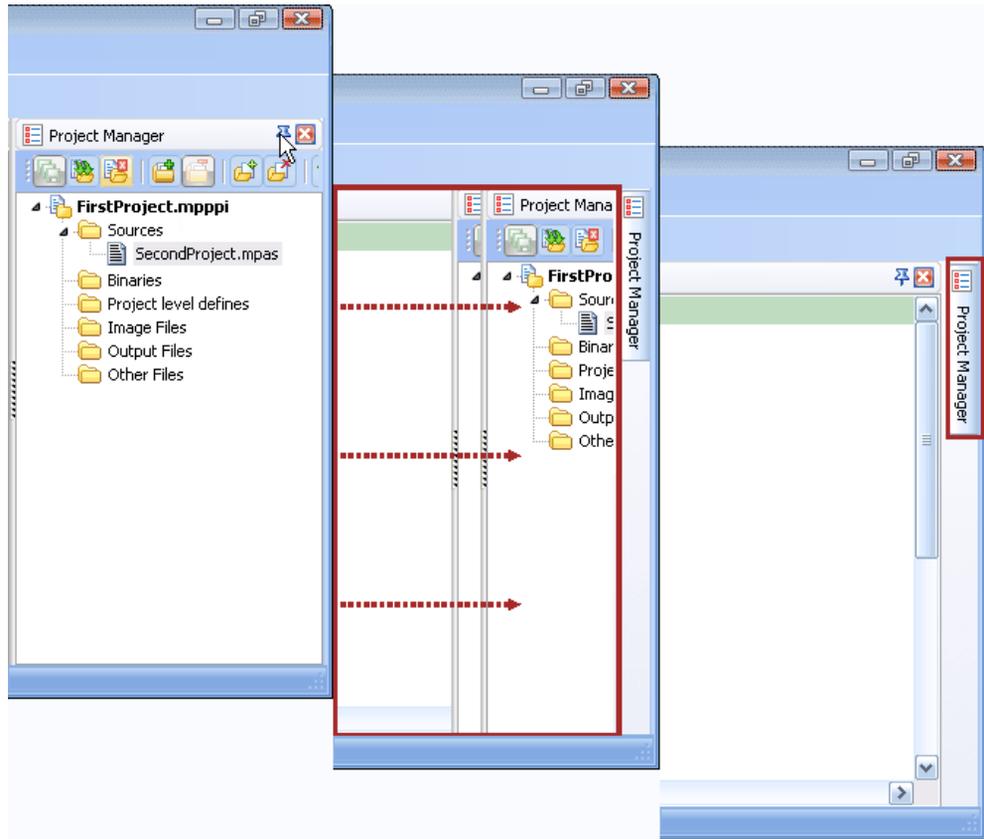
To remove the layout from the drop-down list, select the desired layout from the list and click the Delete Layout Icon .



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.

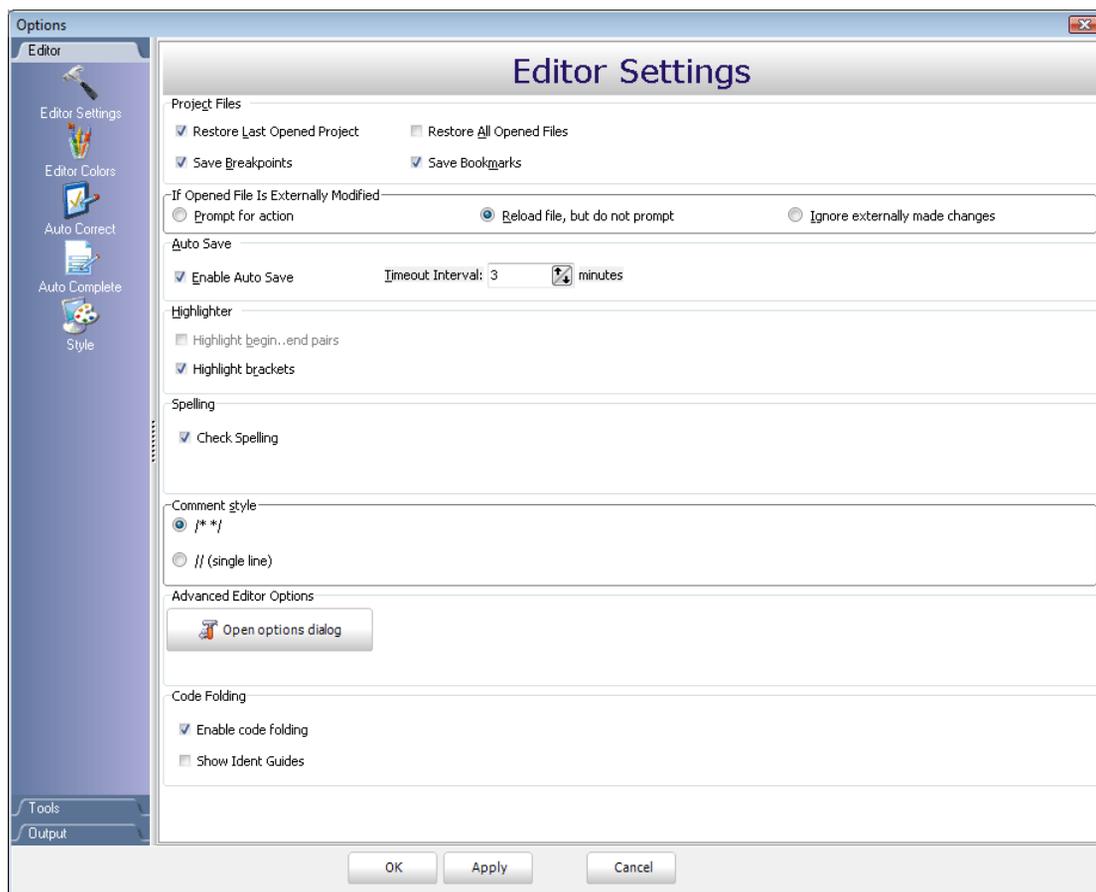
ADVANCED CODE EDITOR

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

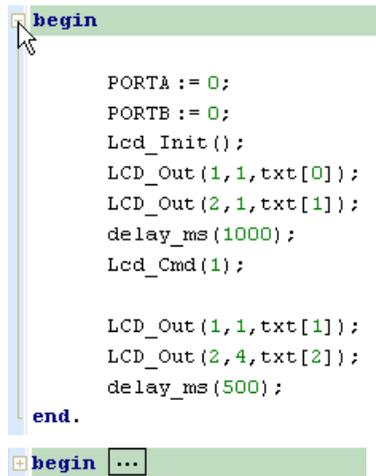
If you type the first few letters of a word and then press Ctrl+Space, all valid identifiers matching the letters you have typed will be prompted in a floating panel (see the image below). Now you can keep typing to narrow the choice, or you can select one from the list using the keyboard arrows and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



```

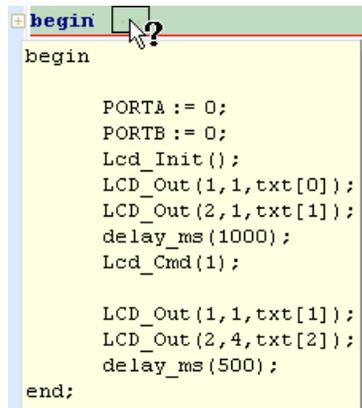
begin
    PORTA := 0;
    PORTB := 0;
    Lcd_Init ();
    LCD_Out (1, 1, txt [0] );
    LCD_Out (2, 1, txt [1] );
    delay_ms (1000);
    Lcd_Cmd (1);

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

```

 begin ...

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



```

begin
    PORTA := 0;
    PORTB := 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 mikroPascal PRO for PIC 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 they 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 short-cut 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.



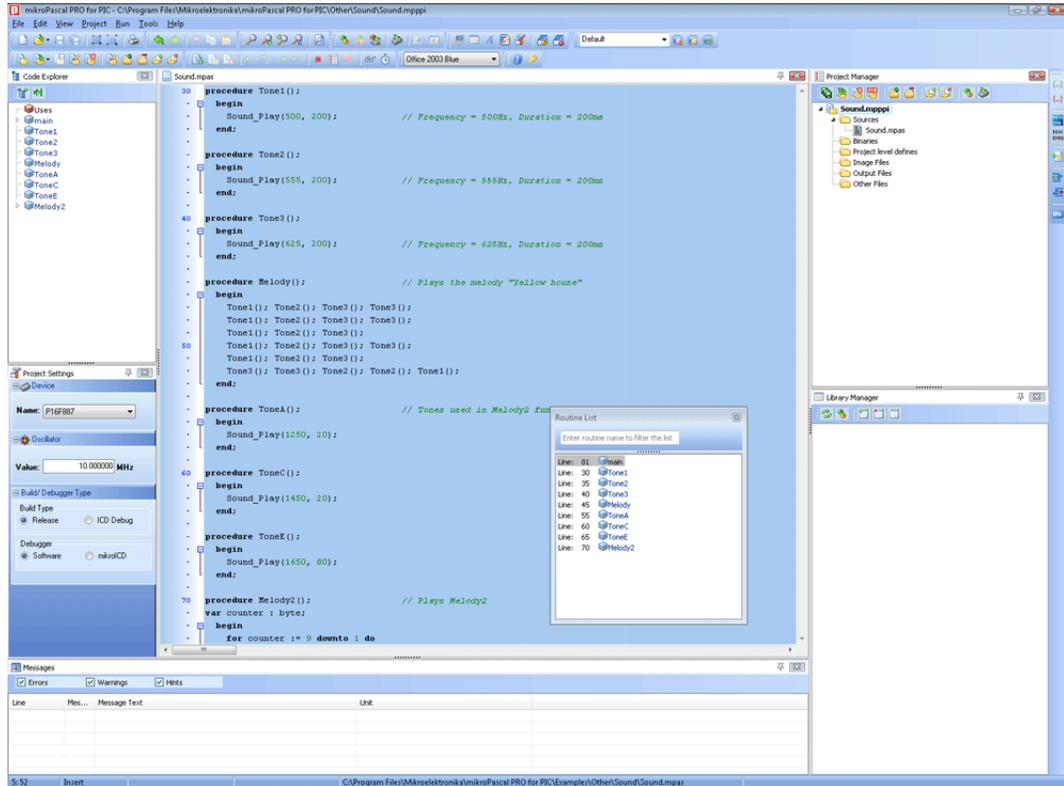
Following options are available in the Code Explorer:

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

ROUTINE LIST

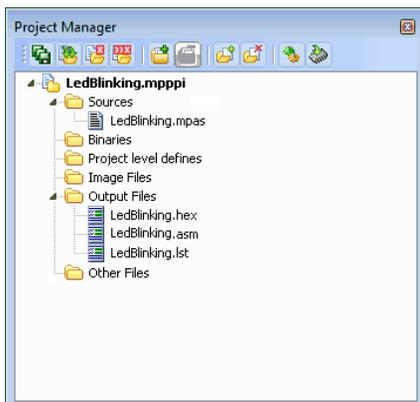
Routine list displays list of routines, and enables filtering routines by name. Routine list window can be accessed by pressing Ctrl+L.

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.

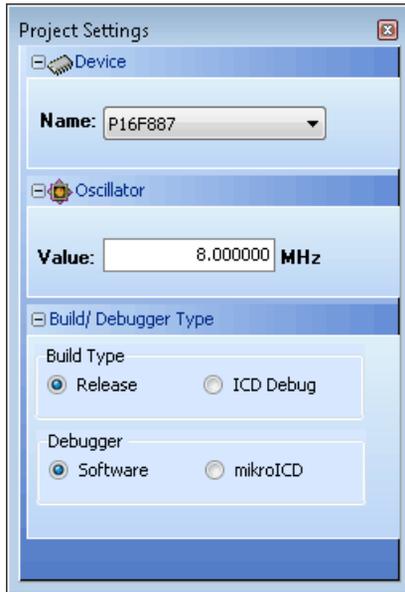
For details about adding and removing files from project see Add/Remove Files from Project.

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.
- Build/Debugger Type - choose debugger type.



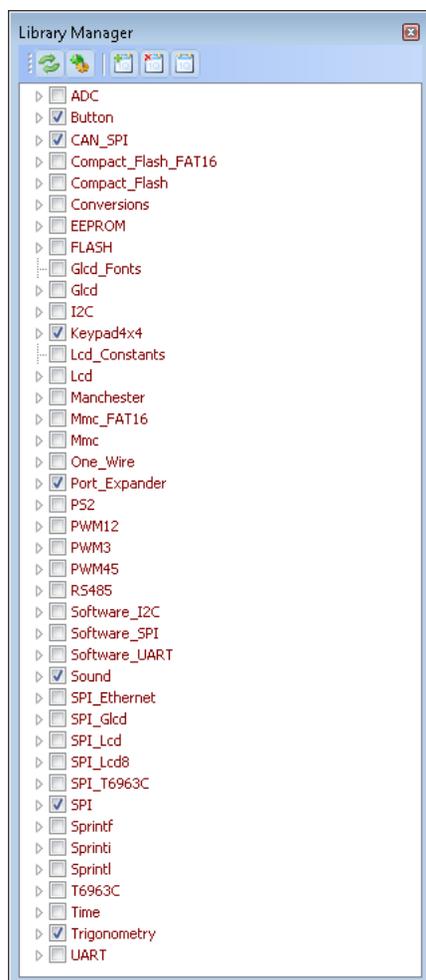
Related topics: Edit Project, Customizing Projects

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.



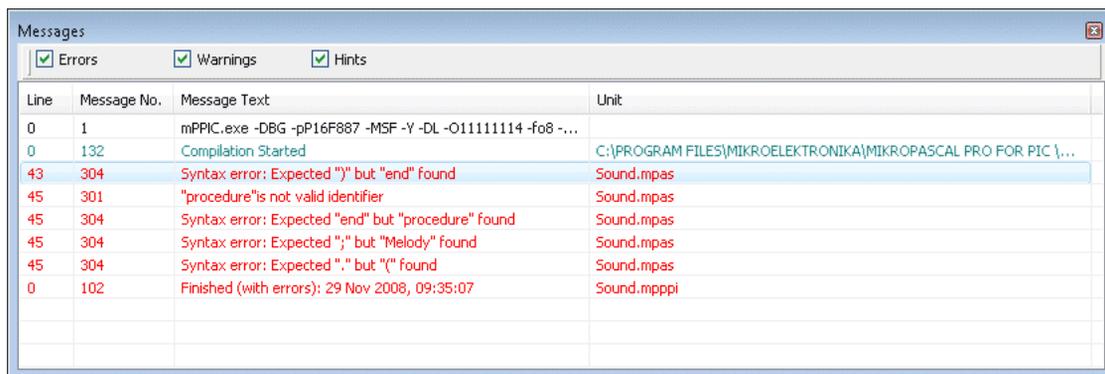
Icon	Description
	Refresh Library by scanning files in "Uses" folder. Useful when new libraries are added by copying files to "Uses" folder.
	Rebuild all available libraries. Useful when library sources are available and need refreshing.
	Include all available libraries in current project.
	No libraries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

Related topics: mikroPascal PRO for PIC 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.



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

Related topics: Error Messages

STATISTICS

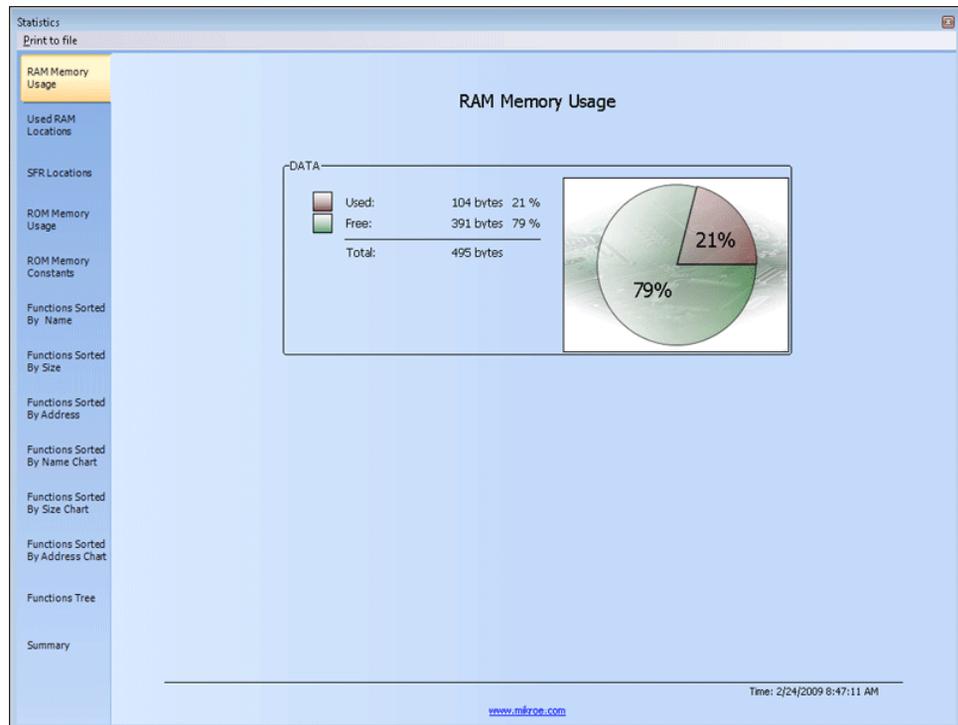
After successful compilation, you can review statistics of your code. Click the Statistics Icon .

Memory Usage Windows

Provides overview of RAM and ROM usage in the various forms.

RAM Memory Usage

Displays RAM memory usage in a pie-like form.



Used RAM Locations

Displays used RAM memory locations and their names.

Address	Name	Address	Name	Address	Name
0x0070	R0	0x0064	yy	0x00A4	fraction
0x0071	R1	0x0065	page	0x00A6	x_start
0x0072	R2	0x0063	activeFont	0x00A7	x_end
0x0073	R3	0x0065	aFontWidth	0x00A8	y_pos
0x0074	R4	0x0066	aFontHeight	0x00A9	color
0x0075	R5	0x0067	aFontOffs	0x00AA	loc
0x0076	R6	0x00A6	chr	0x00A6	y_start
0x0077	R7	0x00A7	x_pos	0x00A7	y_end
0x0078	R8	0x00A8	page_num	0x00A8	x_pos
0x0079	R9	0x00A9	color	0x00A9	color
0x007A	R10	0x00AA	ii	0x00AA	loc
0x007B	R11	0x00AB	rdata	0x0063	x_center
0x007C	R12	0x00AC	dama	0x0065	y_center
0x007D	R13	0x00AD	nema	0x0067	radius
0x007E	R14	0x00AE	pointer	0x0069	color
0x007F	R15	0x0063	text	0x006A	tswitch
0x0004	FSRPTR	0x0064	x_pos	0x006C	y
0x0028	?lstr1_Glcd	0x0065	page_num	0x006E	x
0x002F	?lstr2_Glcd	0x0066	color	0x00A0	d
0x0038	?lstr3_Glcd	0x0067	i	0x0063	image
0x004A	?lstr4_Glcd	0x00B0	x_pos	0x0065	col
0x0053	?lstr5_Glcd	0x00B1	y_pos	0x0066	pg
0x0061	ii	0x00B2	color	0x0067	clan_niza
0x0062	someText	0x00B3	bit_mask1	0x0063	x_upper_left
0x0020	_DoICPAddr	0x00B4	bit_mask0	0x0064	y_upper_left
0x0022	fontW	0x00B5	ddata	0x0065	x_bottom_right

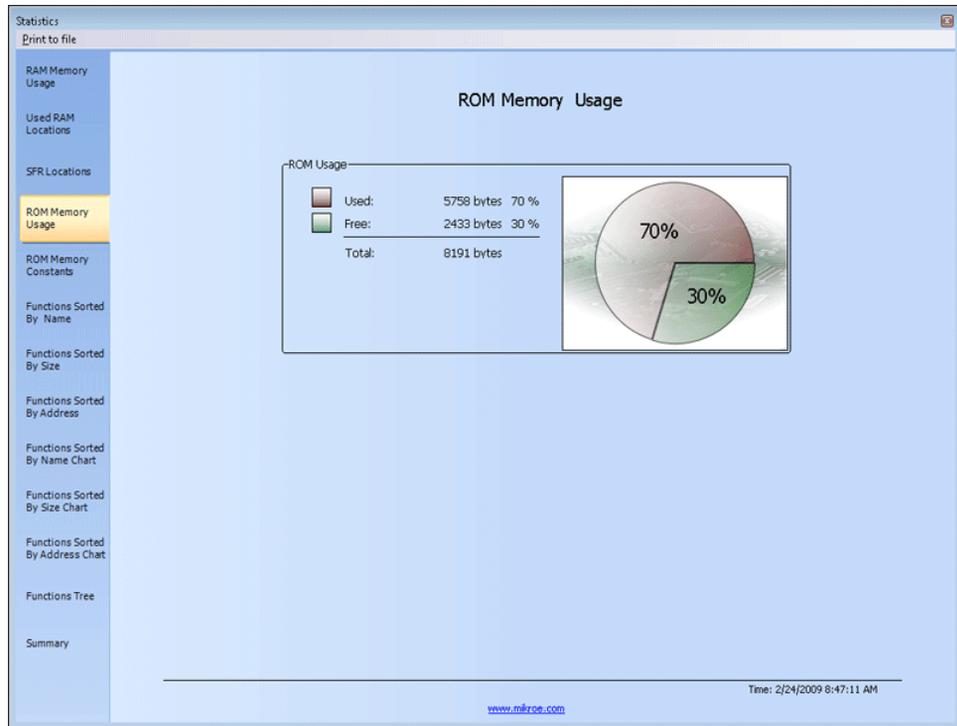
SFR Locations

Displays list of used SFR locations.

Address	Name	Address	Name	Address	Name
0x0000	INDF	0x0018	RX9D_bit	0x009C	PSSAC0_bit
0x0001	TMR0	0x0018	RCD8_bit	0x009C	PSSBD1_bit
0x0002	PCL	0x001D	CCP2X_bit	0x009C	PSSBD0_bit
0x0003	STATUS	0x001D	DC2B1_bit	0x009D	STRSYNC_bit
0x0004	FSR	0x001D	CCP2Y_bit	0x009D	STRD_bit
0x000A	PCLATH	0x001D	DC2B0_bit	0x009D	STRC_bit
0x000B	INTCON	0x001D	CCP2M3_bit	0x009D	STRB_bit
0x000C	PIR1	0x001D	CCP2M2_bit	0x009D	STRA_bit
0x000D	PIR2	0x001D	CCP2M1_bit	0x009F	ADFM_bit
0x000E	TMR1L	0x001D	CCP2M0_bit	0x009F	VCFG1_bit
0x000F	TMR1H	0x001F	ADCS1_bit	0x009F	VCFG0_bit
0x0010	T1CON	0x001F	ADCS0_bit	0x0105	WDTPS3_bit
0x0011	TMR2	0x001F	CHS3_bit	0x0105	WDTPS2_bit
0x0012	T2CON	0x001F	CHS2_bit	0x0105	WDTPS1_bit
0x0013	SSPBUF	0x001F	CHS1_bit	0x0105	WDTPS0_bit
0x0014	SSPCON	0x001F	CHS0_bit	0x0105	SWDTEN_bit
0x0015	CCPR1L	0x001F	GO_bit	0x0107	C1ON_bit
0x0016	CCPR1H	0x001F	NOT_DONE_bit	0x0107	C1OUT_bit
0x0017	CCP1CON	0x001F	GO_DONE_bit	0x0107	C1OE_bit
0x0018	RCSTA	0x001F	ADON_bit	0x0107	C1POL_bit
0x0019	TXREG	0x0081	NOT_RBPU_bit	0x0107	C1R_bit
0x001A	RCREG	0x0081	INTEDG_bit	0x0107	C1CH1_bit
0x001B	CCPR2L	0x0081	T0CS_bit	0x0107	C1CH0_bit
0x001C	CCPR2H	0x0081	T0SE_bit	0x0108	C2ON_bit
0x001D	CCP2CON	0x0081	PSA_bit	0x0108	C2OUT_bit
0x001E	ADRESH	0x0081	PS2_bit	0x0108	C2OE_bit

ROM Memory Usage

Displays ROM memory space usage in a pie-like form.



ROM Memory Constants

Displays ROM memory constants and their addresses.



The screenshot displays the mikroPASCAL PRO interface. On the left, a vertical menu lists various statistics and memory usage options. The 'ROM Memory Constants' option is highlighted in yellow. The main window area is titled 'ROM Memory Constants' and contains a table with two columns: 'Address' and 'Name'. The table lists several constants with their corresponding addresses and names. At the bottom right of the window, the time is displayed as 'Time: 2/24/2009 8:47:11 AM' and the website 'www.mikroe.com' is visible at the bottom center.

Address	Name
0x0780	?1CS?1str1_Glcd
0x0787	?1CS?1str2_Glcd
0x0790	?1CS?1str3_Glcd
0x07A2	?1CS?1str4_Glcd
0x07AB	?1CS?1str5_Glcd
0x0800	Character8x7
0x15EA	font5x7
0x1400	FontSystem5x7_v2
0x06BD	System3x5
0x1000	truck_bmp

Functions Sorted By Name

Sorts and displays functions by their addresses, symbolic names, and unique assembler names.

Address	Name	Unique Assembler Name
0x006F	___DoICP	___DoICP
0x01D6	___CC2DW	___CC2DW
0x0005	Delay_10us	__Delay_10us
0x0118	Delay_1us	__Delay_1us
0x0013	Delay_50us	__Delay_50us
0x0540	delay25	__delay25
0x0524	Glcd_Box	__Glcd_Box
0x0283	Glcd_Circle	__Glcd_Circle
0x00A1	Glcd_Dot	__Glcd_Dot
0x055D	Glcd_Fill	__Glcd_Fill
0x018B	Glcd_H_Line	__Glcd_H_Line
0x0228	Glcd_Image	__Glcd_Image
0x0481	Glcd_Init	__Glcd_Init
0x037F	Glcd_Line	__Glcd_Line
0x0034	Glcd_Read_Data	__Glcd_Read_Data
0x01E2	Glcd_Rectangle	__Glcd_Rectangle
0x054E	Glcd_Set_Font	__Glcd_Set_Font
0x0018	Glcd_Set_Page	__Glcd_Set_Page
0x0023	Glcd_Set_Side	__Glcd_Set_Side
0x004D	Glcd_Set_X	__Glcd_Set_X
0x00FD	Glcd_V_Line	__Glcd_V_Line
0x011B	Glcd_Write_Char	__Glcd_Write_Char
0x0064	Glcd_Write_Data	__Glcd_Write_Data
0x04E6	Glcd_Write_Text	__Glcd_Write_Text
0x0590	main	__main
0x0076	Mul_16x16_U	__Mul_16x16_U

Functions Sorted By Size

Sorts and displays functions by their size, in the ascending order.

Name	Unique Assembler Name	Size (bytes)
Delay_1us	._Delay_1us	3
Delay_50us	._Delay_50us	5
Delay_10us	._Delay_10us	6
___DoICP	___DoICP	7
Strobe	._Lib_Glcd_Strobe	8
Glcd_Write_Data	._Glcd_Write_Data	11
Glcd_Write_Data	._Glcd_Write_Data	11
___CC2DW	___CC2DW	12
delay2S	._delay2S	14
Glcd_Set_Font	._Glcd_Set_Font	15
Glcd_Set_Side	._Glcd_Set_Side	17
Glcd_Set_X	._Glcd_Set_X	23
Glcd_Read_Data	._Glcd_Read_Data	25
Glcd_Read_Data	._Glcd_Read_Data	25
Glcd_V_Line	._Glcd_V_Line	27
Glcd_V_Line	._Glcd_V_Line	27
Glcd_Box	._Glcd_Box	28
Glcd_Write_Text	._Glcd_Write_Text	37
Mul_16x16_U	._Mul_16x16_U	43
Glcd_Fill	._Glcd_Fill	51
Glcd_Init	._Glcd_Init	53
Glcd_Rectangle	._Glcd_Rectangle	70
Glcd_Image	._Glcd_Image	91
Glcd_Dot	._Glcd_Dot	92
Glcd_Write_Char	._Glcd_Write_Char	160
Glcd_Circle	._Glcd_Circle	252

Functions Sorted By Addresses

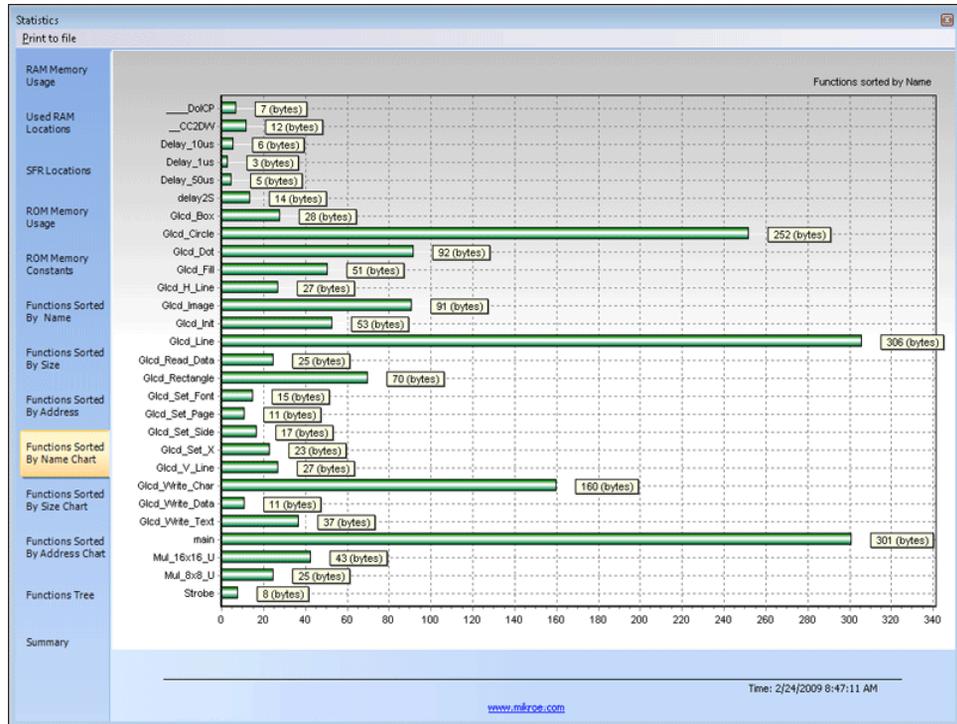
Sorts and displays functions by their addresses, in the ascending order.

The screenshot shows the 'Functions Sorted By Address' window in mikroPASCAL PRO. The window title is 'Functions Sorted By Address'. The left sidebar contains various navigation options, with 'Functions Sorted By Address' highlighted in yellow. The main area displays a table with the following data:

Address	Name	Unique Assembler Name
0x0005	Delay_10us	__Delay_10us
0x000B	Strobe	__Lib_Glcd_Strobe
0x0013	Delay_50us	__Delay_50us
0x0018	Glcd_Set_Page	__Glcd_Set_Page
0x0023	Glcd_Set_Side	__Glcd_Set_Side
0x0034	Glcd_Read_Data	__Glcd_Read_Data
0x004D	Glcd_Set_X	__Glcd_Set_X
0x0064	Glcd_Write_Data	__Glcd_Write_Data
0x006F	__DoICP	__DoICP
0x0076	Mul_16x16_U	__Mul_16x16_U
0x00A1	Glcd_Dot	__Glcd_Dot
0x00FD	Glcd_V_Line	__Glcd_V_Line
0x0118	Delay_1us	__Delay_1us
0x011B	Glcd_Write_Char	__Glcd_Write_Char
0x01BB	Glcd_H_Line	__Glcd_H_Line
0x01D6	__CC2DW	__CC2DW
0x01E2	Glcd_Rectangle	__Glcd_Rectangle
0x0228	Glcd_Image	__Glcd_Image
0x0283	Glcd_Circle	__Glcd_Circle
0x037F	Glcd_Line	__Glcd_Line
0x04B1	Glcd_Init	__Glcd_Init
0x04E6	Glcd_Write_Text	__Glcd_Write_Text
0x050B	Mul_8x8_U	__Mul_8x8_U
0x0524	Glcd_Box	__Glcd_Box
0x0540	delay2S	__delay2S
0x054E	Glcd_Set_Font	__Glcd_Set_Font

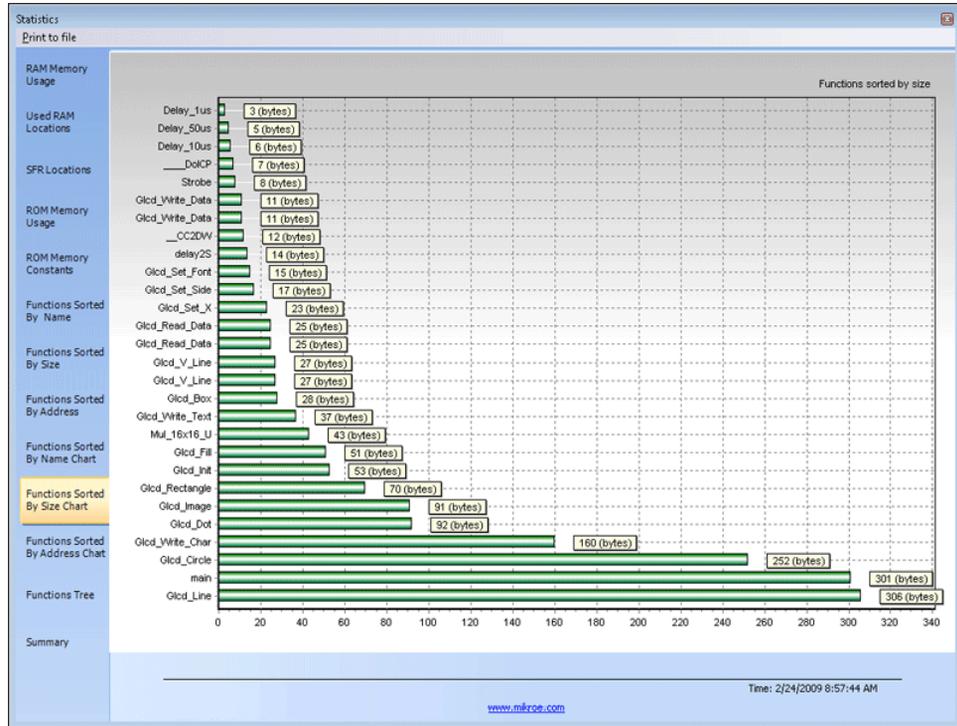
Functions Sorted By Name Chart

Sorts and displays functions by their names in a chart-like form.



Functions Sorted By Size Chart

Sorts and displays functions by their sizes in a chart-like form.



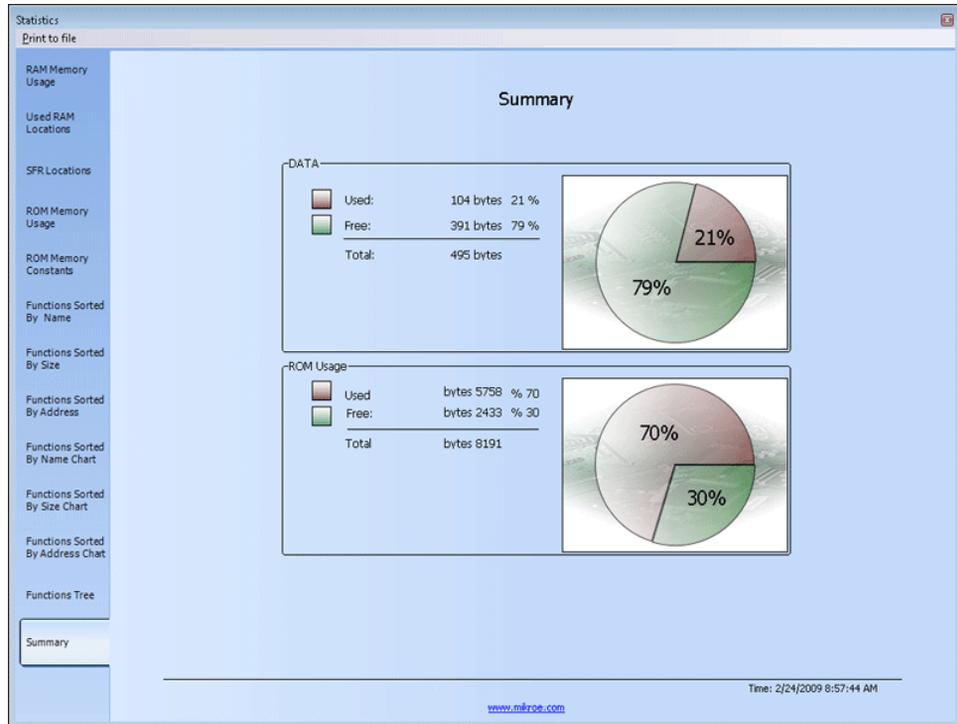
Functions Sorted By Addresses Chart

Sorts and displays functions by their addresses in a chart-like form.



Memory Summary

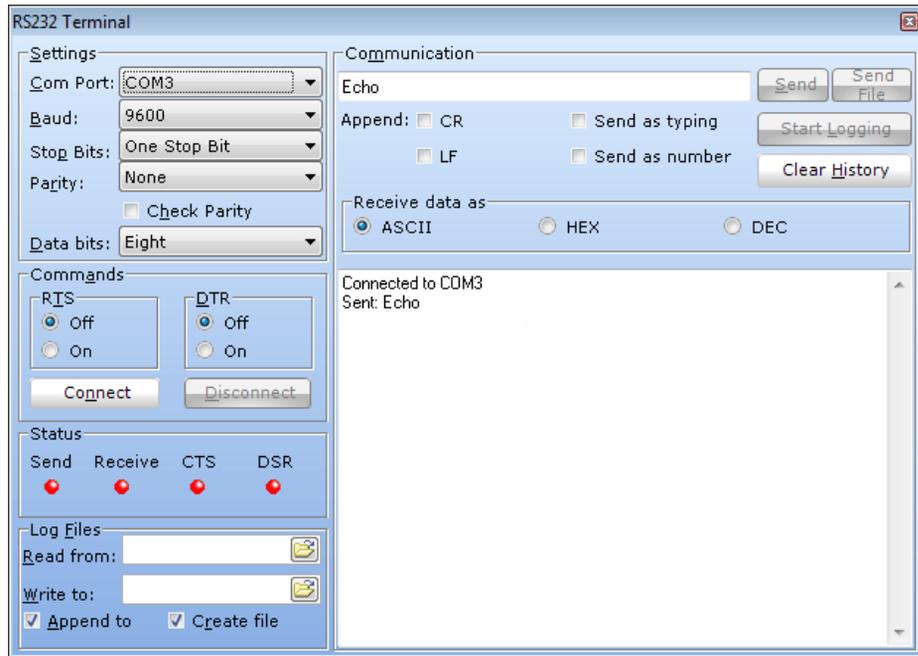
Displays summary of RAM and ROM memory in a pie-like form.



INTEGRATED TOOLS

USART Terminal

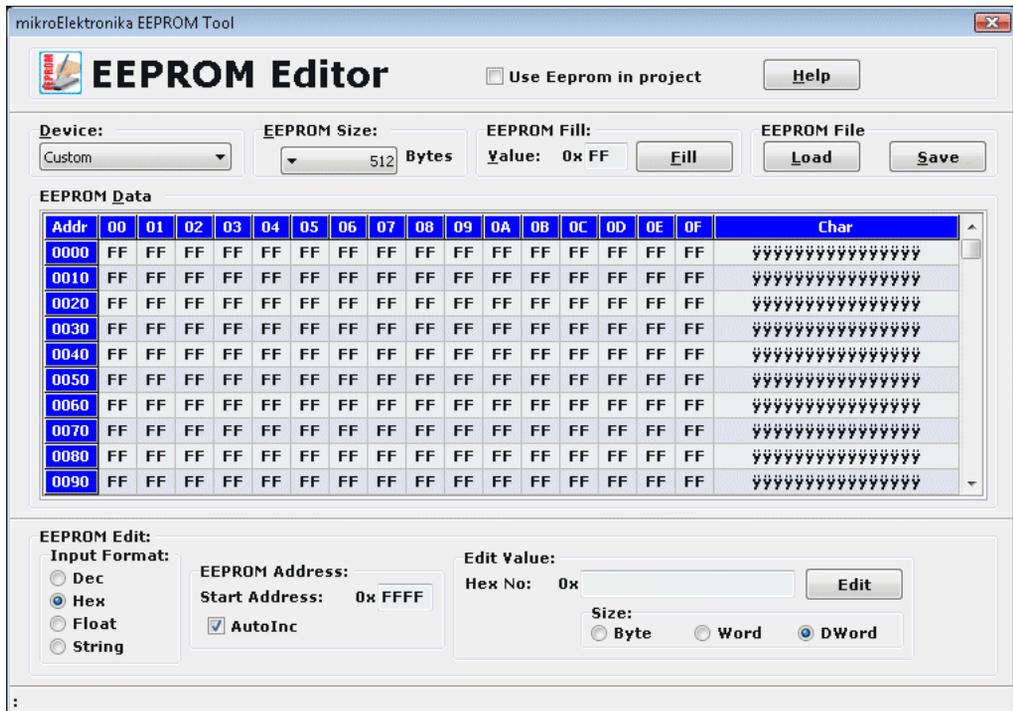
The mikroPascal PRO for PIC includes the USART communication terminal for RS232 communication. You can launch it from the drop-down menu **Tools > USART Terminal** or by clicking the USART Terminal Icon  from Tools toolbar.



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 mikroPascal PRO for PIC IDE - `project_name.hex` file will be loaded automatically while ihex file must be loaded manually.



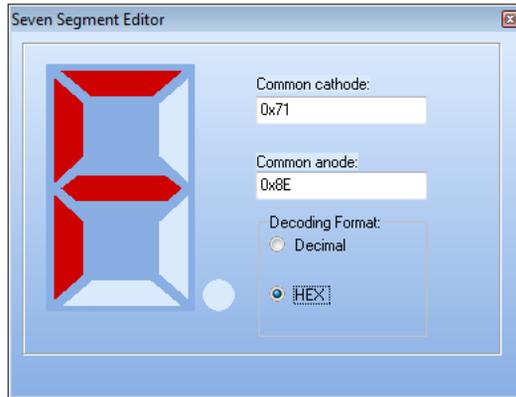
ASCII Chart

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.

	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
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SPC	!	"	#	\$	%	'	()	*	+	,	-	.	/	
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
8	€	,	f	„	…	†	‡	ˆ	˜	™	§	»	œ	ž	ÿ	
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	i	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯	
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	÷	ø	ù	ú	û	ü	ý	þ	ÿ	
	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

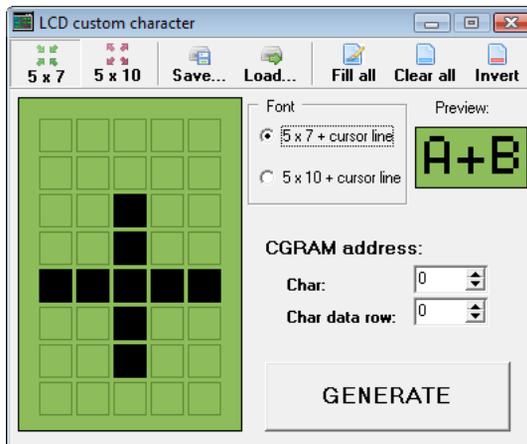
Seven Segment Converter

The Seven Segment Display Converter 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** › **Seven Segment Converter** or by clicking the Seven Segment Icon  from Tools toolbar.



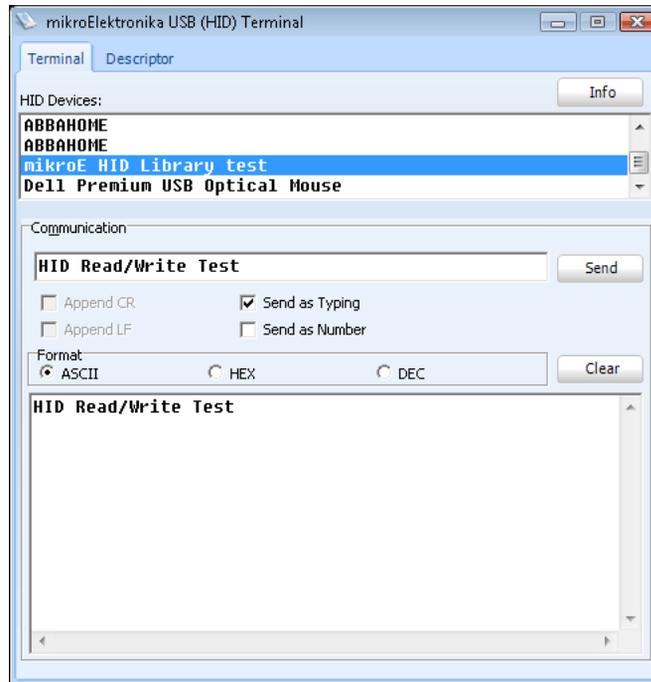
Lcd Custom Character

mikroPascal PRO for PIC includes the Lcd Custom Character. Output is mikroPascal PRO for PIC compatible code. You can launch it from the drop-down menu **Tools** › **Lcd Custom Character**.



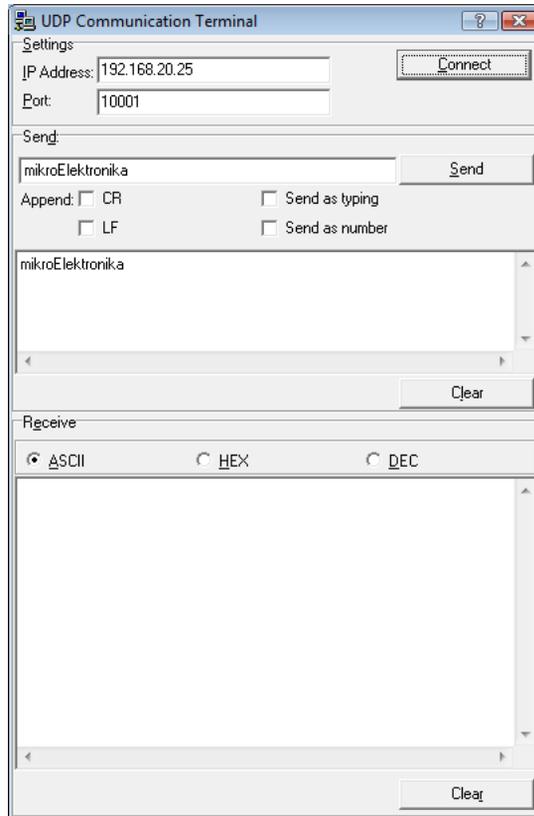
HID Terminal

The mikroPascal PRO for PIC includes the HID communication terminal for USB communication. You can launch it from the drop-down menu **Tools > HID Terminal**.



UDP Terminal

The mikroPascal PRO for PIC includes the UDP Terminal. You can launch it from the drop-down menu **Tools** › **UDP Terminal**.



mikroBootloader

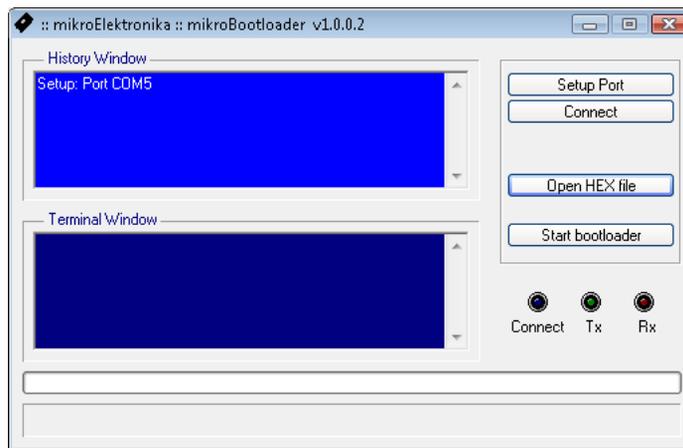
What is a Bootloader

(From Microchip's document AN732) The PIC16F87X family of microcontrollers has the ability to write to their own program memory. This feature allows a small bootloader program to receive and write new firmware into memory. In its most simple form, the bootloader starts the user code running, unless it finds that new firmware should be downloaded. If there is new firmware to be downloaded, it gets the data and writes it into program memory. There are many variations and additional features that can be added to improve reliability and simplify the use of the bootloader.

Note: mikroBootloader can be used only with PIC MCUs that support flash write.

How to use mikroBootloader

1. Load the PIC with the appropriate hex file using the conventional programming techniques (e.g. for PIC16F877A use [p16f877a.hex](#)).
2. Start mikroBootloader from the drop-down menu **Tools > Bootloader**.
3. Click on **Setup Port** and select the COM port that will be used. Make sure that BAUD is set to 9600 Kpbs.
4. Click on **Open File** and select the HEX file you would like to upload.
5. Since the bootcode in the PIC only gives the computer 4-5 sec to connect, you should reset the PIC and then click on the **Connect** button within 4-5 seconds.
6. The last line in then history window should now read "Connected".
7. To start the upload, just click on the **Start Bootloader** button.
8. Your program will written to the PIC flash. Bootloader will report an errors that may occur.
9. Reset your PIC and start to execute.



Features

The boot code gives the computer 5 seconds to get connected to it. If not, it starts running the existing user code. If there is a new user code to be downloaded, the boot code receives and writes the data into program memory.

The more common features a bootloader may have are listed below:

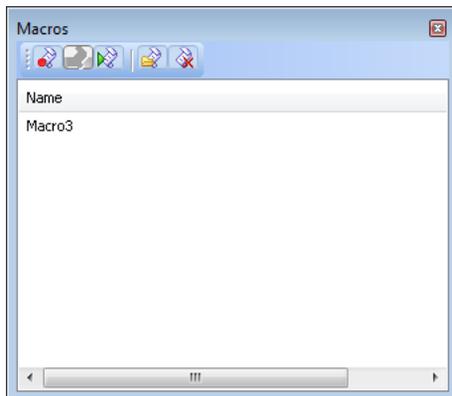
- Code at the Reset location.
- Code elsewhere in a small area of memory.
- Checks to see if the user wants new user code to be loaded.
- Starts execution of the user code if no new user code is to be loaded.
- Receives new user code via a communication channel if code is to be loaded.
- Programs the new user code into memory.

Integrating User Code and Boot Code

The boot code almost always uses the Reset location and some additional program memory. It is a simple piece of code that does not need to use interrupts; therefore, the user code can use the normal interrupt vector at 0x0004. The boot code must avoid using the interrupt vector, so it should have a program branch in the address range 0x0000 to 0x0003. The boot code must be programmed into memory using conventional programming techniques, and the configuration bits must be programmed at this time. The boot code is unable to access the configuration bits, since they are not mapped into the program memory space.

MACRO EDITOR

A macro is a series of keystrokes that have been 'recorded' in the order performed. A macro allows you to 'record' a series of keystrokes and then 'playback', or repeat, the recorded keystrokes.



The Macro offers the following commands:

Icon	Description
	Starts 'recording' keystrokes for later playback.
	Stops capturing keystrokes that was started when the Start Recording command was selected.
	Allows a macro that has been recorded to be replayed.
	New macro.
	Delete macro.

Related topics: Advanced Code Editor, Code Templates

OPTIONS

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

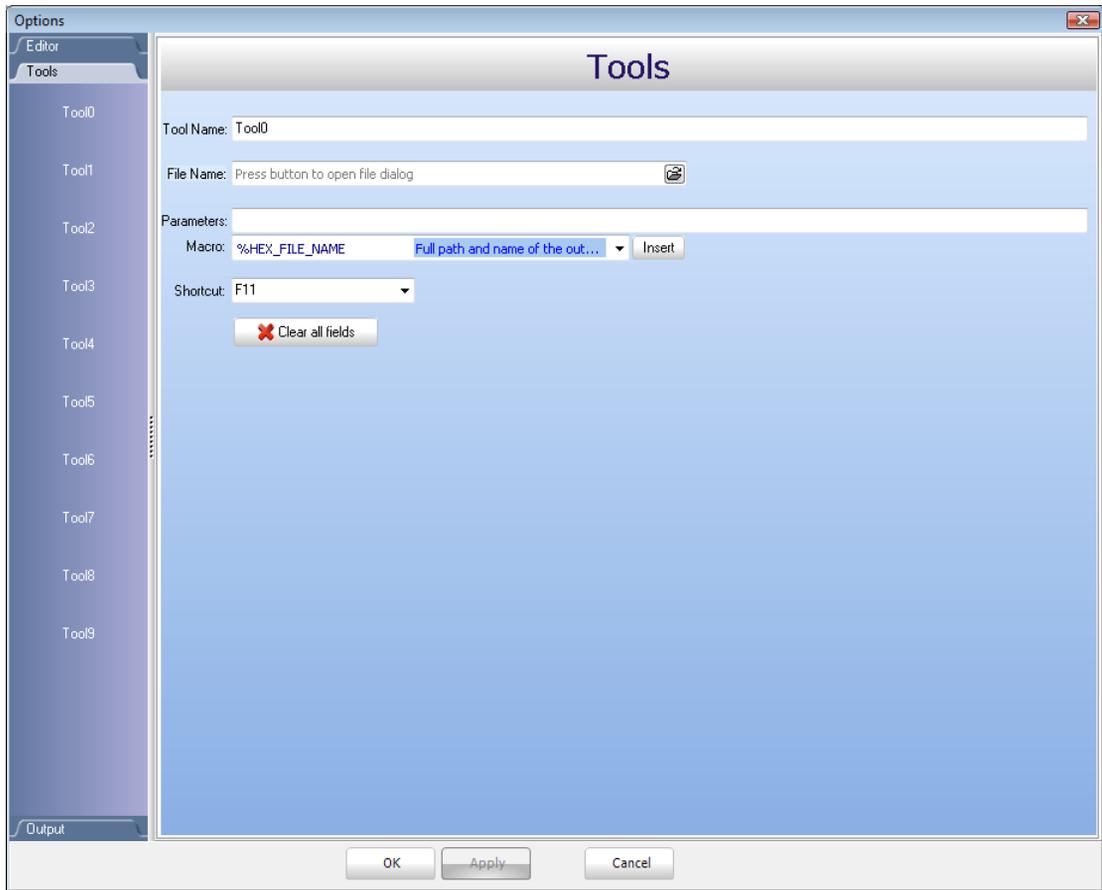
Code editor

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

Tools

The mikroPascal PRO for PIC 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.



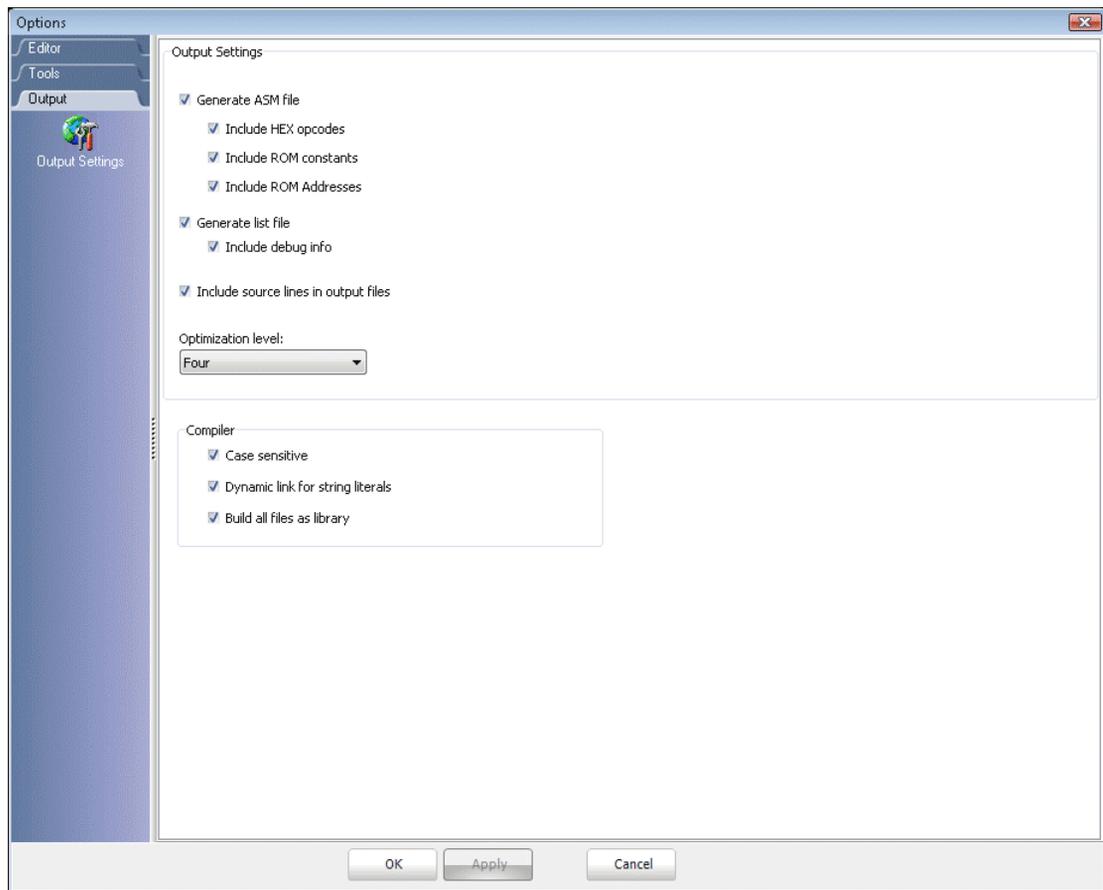
Output settings

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.

Also, user can choose optimization level, and compiler specific settings, which include case sensitivity, dynamic link for string literals setting (described in mikroPascal PRO for PIC specifics).

Build all files as library enables user to use compiled library (*.mcl) on any PIC MCU (when this box is checked), or for a selected PIC MCU (when this box is left unchecked).

For more information on creating new libraries, see Creating New Library.



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 :

```
unsigned matches string 'unsigned'  
^unsigned matches string '^unsigned'
```

Escape sequences

Characters may be specified using a escape sequences: `"\n"` matches a newline, `"\t"` a tab, etc. More generally, `\xnn`, 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:

```
unsigned\x20int matches 'unsigned int' (note space in the middle)  
\tunsigned matches 'unsigned' (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 precede it with a backslash.

If you want `']'`, you may place it at the start of list or precede 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 `']'` to `'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 <http://www.unicode.org/unicode/reports/tr18/>

Metacharacters - Predefined classes

`\w` - an alphanumeric character (including "`_`")
`\W` - a nonalphanumeric character
`\d` - a numeric character
`\D` - a non-numeric character
`\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 an alphanumeric character ("`\w`") on one side, and a nonalphanumeric character ("`\W`") on the other side (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 - matches strings like 'counter', 'countelkjdf1kj9r' and 'count'
- count.+r - matches strings like 'counter', 'countelkjdf1kj9r' but not 'count'
- 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

MIKROPASCAL PRO FOR PIC COMMAND LINE OPTIONS

Usage: `mPPIC.exe` [-<opts> [-<opts>]] [<infile> [-<opts>]] [-<opts>]]
 Infile can be of *.mpas and *.mcl type.

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

- P : MCU for which compilation will be done.
- FO : Set oscillator [in MHz].
- 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.
- L : Check and rebuild new libraries.
- DL : Build all files as libraries.
- Y : Dynamic link for string literals.
- UICD : ICD build type.

Example:

```
mPPIC.exe -MSF -DBG -p16F887 -O11111114 -fo8 -N"C:\Lcd\Lcd.mpppi" -SP"C:\Program Files\Mikroelektronika\mikroPascal PRO for PIC\Defs\" -SP"C:\Program Files\Mikroelektronika\mikroPascal PRO for PIC\Uses\P16\" -SP"C:\Lcd\" "Lcd.mpas" "__Lib_Math.mcl" "__Lib_MathDouble.mcl" "__Lib_System.mcl" "__Lib_Delays.mcl" "__Lib_LcdConsts.mcl" "__Lib_Lcd.mcl"
```

Parameters used in the example:

- MSF : Short Message Format; used for internal purposes by IDE.
- DBG : Generate debug info.
- p16F887 : MCU 16F887 selected.
- O11111114 : Miscellaneous output options.
- fo8 : Set oscillator frequency [in MHz].
- N"C:\Lcd\Lcd.mpppi" -SP"C:\Program Files\Mikroelektronika\mikroPascal PRO for PIC\defs\" : Output files generated to file path specified by filename.
- SP"C:\Program Files\Mikroelektronika\mikroPascal PRO for PIC\Defs\" : Add directory to the search path list.
- SP"C:\Program Files\Mikroelektronika\mikroPascal PRO for PIC\Uses\P16\" : Add directory to the search path list.
- SP"C:\Lcd\" : Add directory to the search path list.
- "Lcd.mpas" "__Lib_Math.mcl" "__Lib_MathDouble.mcl" "__Lib_System.mcl" "__Lib_Delays.mcl" "__Lib_LcdConsts.mcl" "__Lib_Lcd.mcl" : Specify input files.

PROJECTS

The mikroPascal PRO for PIC organizes applications into projects, consisting of a single project file (extension `.mppi`) and one or more source files (extension `mpas`). mikroPascal PRO for PIC 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,
- device flags (config word),
- device clock,
- list of the project source files with paths,
- image files,
- other files.

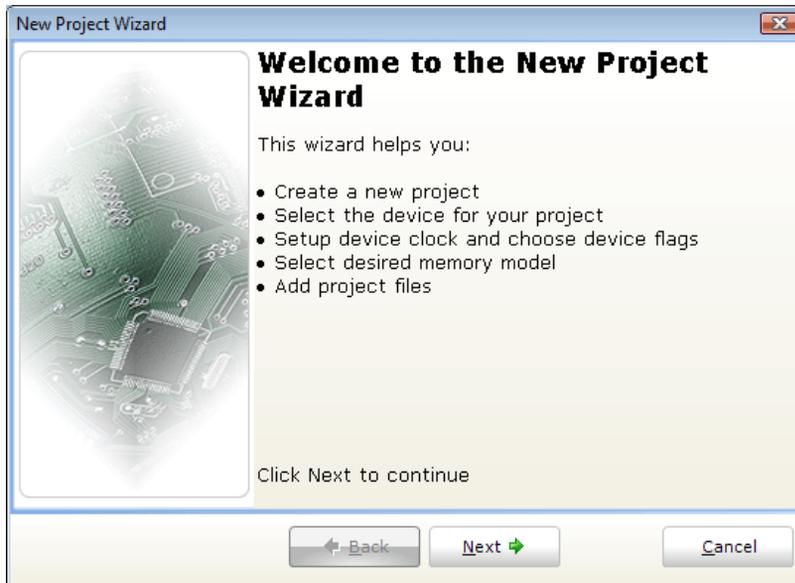
Note that the project does not include files in the same way as preprocessor does, see Add/Remove Files from Project.

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

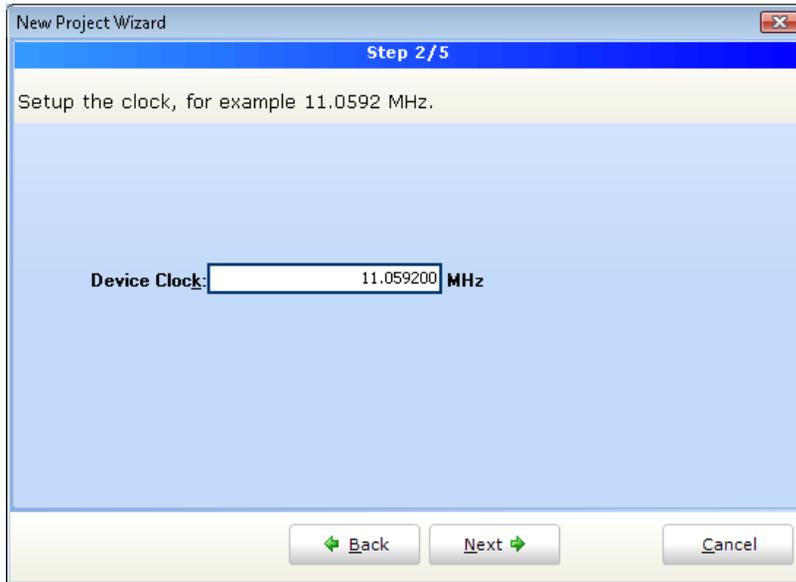
Start creating your New project, by clicking Next button:



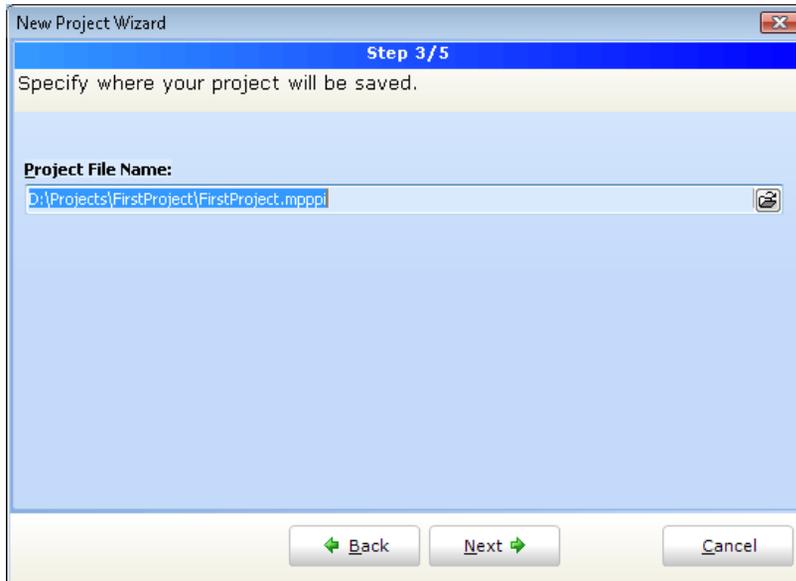
Step One - Select the device from the device drop-down list.



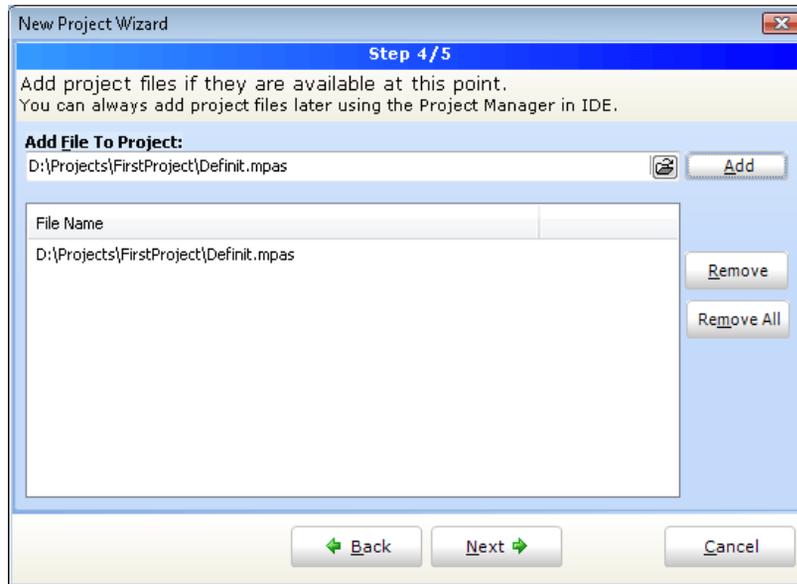
Step Two - Enter the oscillator frequency value.



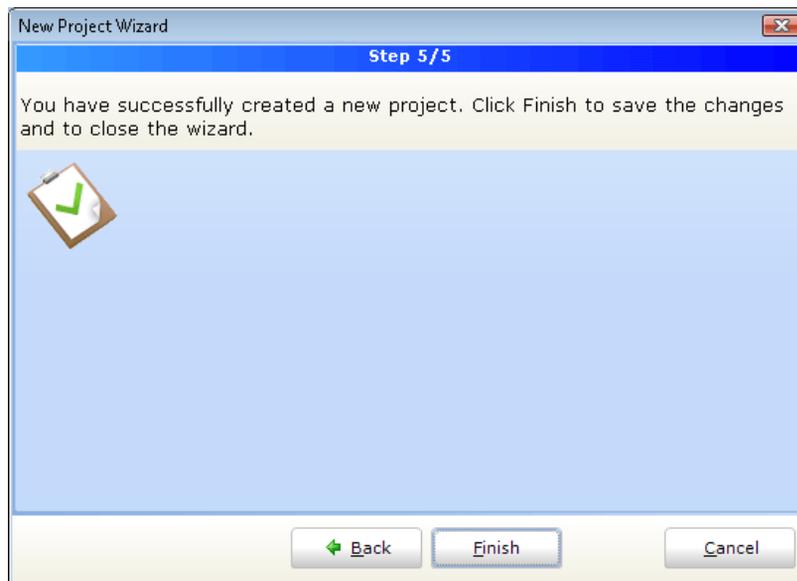
Step Three - Specify the location where your project will be saved.



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



Step Five - Click Finish button to create your New Project:



Related topics: Project Manager, Project Settings

CUSTOMIZING PROJECTS

You can change basic project settings in the Project Settings window, like chip and oscillator frequency. 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. Also, you can change configuration bits of the selected chip in the Edit Project window.

Managing Project Group

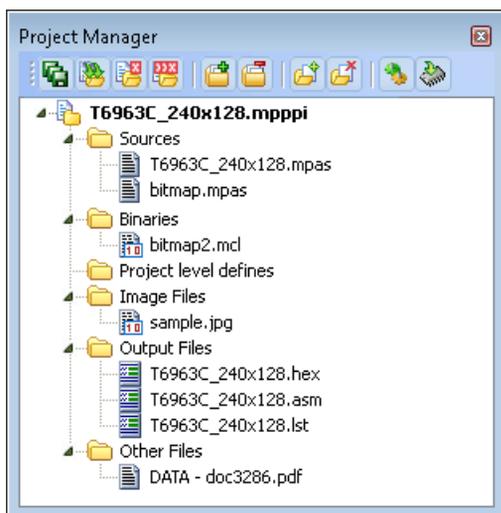
mikroPascal PRO for PIC IDE provides convenient option which enables several projects to be open simultaneously. If you have several projects being connected in some way, you can create a project group.

The project group may be saved by clicking the Save Project Group Icon  from the Project Manager window. The project group may be reopend 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:

- `.mpas` source files
- `.mcl` binary files
- `.pld` project level defines files
- 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 `.mpppi`).

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 .

See File Inclusion for more information.

Project Level Defines

Project Level Defines (`.pld`) files can also be added to project. Project level define files enable you to have defines that are visible in all source files in the project. One project may contain several pld files. A file must contain one definition per line, for example:

```
ANALOG  
DEBUG  
TEST
```

There are some predefined project level defines. See predefined project level defines

Related topics: Project Manager, Project Settings, Edit Project

SOURCE FILES

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

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 `.mpas`, will be created automatically. The mikroPascal PRO for PIC 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

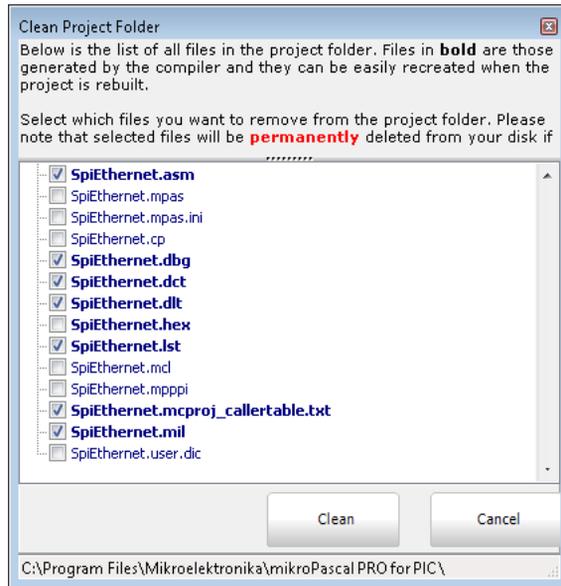
1. Make sure that the tab containing the file that you want to close is the active tab.
2. Select **File** › **Close** from the drop-down menu, or right click the tab of the file that you want to close and select **Close** option from the context menu.
3. If the file has been changed since it was last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

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

CLEAN PROJECT FOLDER

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.



Related topics: Customizing Projects

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 mikroPascal PRO for PIC will generate output files.

Output Files

Upon successful compilation, the mikroPascal PRO for PIC will generate output files in the project folder (folder which contains the project file `.mppi`). Output files are summarized in the table below:

Format	Description	File Type
Intel HEX	Intel style hex records. Use this file to program PIC 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 PIC 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 mikroPascal PRO for PIC, 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.
- Actual 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.
 - Complex type 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 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.
 - Destination size ("%s") does not match source size ("%s").
 - Routine prototype is different from previous declaration.
 - Division by zero.
 - Uart module cannot be initialized with selected baud rate/device clock.
 - "%s" cannot be of "%s" type.
 - Array of "%s" can not be declared.
 - Incomplete variable declaration: "%s".
 - Recursive build of units is not allowed ("%s").
 - Object must be smaller than 64kb in size: "%s".
 - Index out of bounds.
 - With statement cannot be used with this argument "%s".
 - Reset directive is available only on P18 family.

Warning Messages:

- Variable "%s" is not initialized.
- Return value of the function "%s" is not defined.
- Identifier "%s" overrides declaration in unit "%s".
- Generated baud rate is %s bps (error = %s percent).
- Result size may exceed destination array size.
- Infinite loop.
- Implicit typecast performed from "%s" to "%s".
- Implicit typecast of integral value to pointer.
- Library "%s" was not found in search path.
- Interrupt context saving has been turned off.
- Source size (%s) does not match destination size (%s).
- Aggregate padded with zeros (%s) in order to match declared size (%s).
- Suspicious pointer conversion.
- Source size may exceed destination size.

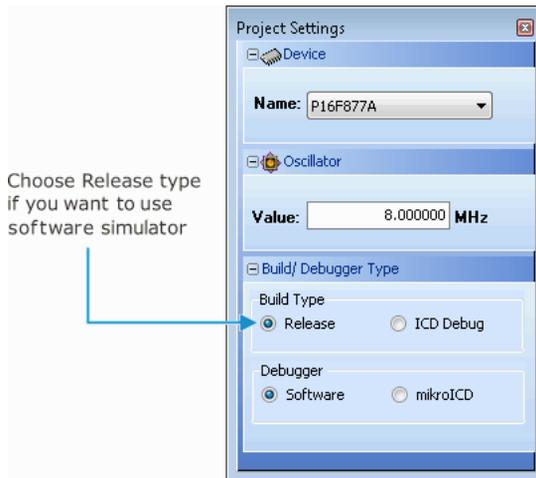
HINT MESSAGES:

- Constant "%s" has been declared, but not used.
- Variable "%s" has been declared, but not used.
- Unit "%s" has been recompiled.
- Variable "%s" has been eliminated by optimizer.
- Compiling unit "%s".

SOFTWARE SIMULATOR OVERVIEW

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

Upon completion of writing your program, choose **Release** build Type in the Project Settings window:

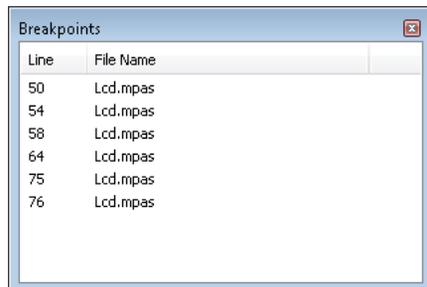


After you have successfully compiled your project, you can run the Software Simulator by selecting **Run > Start Debugger** from the drop-down menu, or by clicking the Start Debugger Icon  from the Debugger Toolbar. Starting the Software Simulator makes more options available: Step Into, Step Over, Step 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 PIC device behavior, i.e. it doesn't update timers, interrupt flags, etc.

Breakpoints Window

The Breakpoints window manages the list of currently set breakpoints in the project. Doubleclicking the desired breakpoint will cause cursor to navigate to the corresponding location in source code.

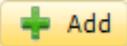
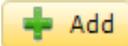


Watch Window

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 Pascal 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  .

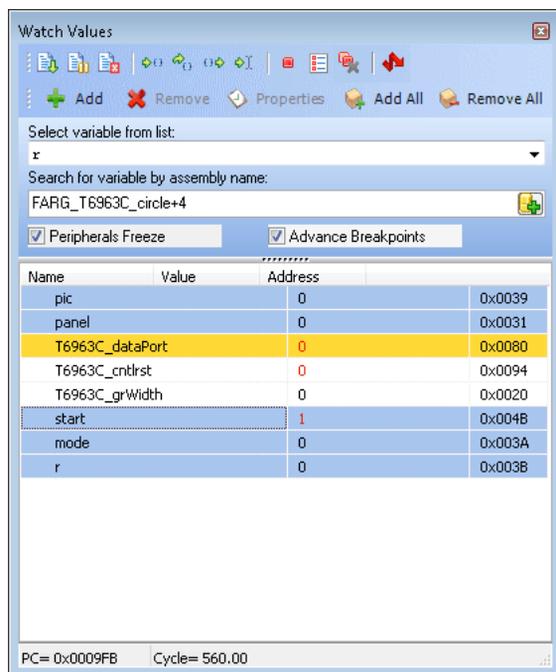
Variables can also be removed from the Watch window, just select the variable that you want to remove and then click the Remove Button  .

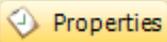
Add All Button  adds all variables.

Remove All Button  removes all variables.

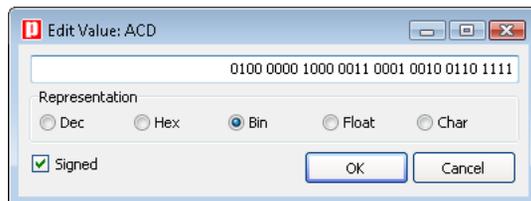
You can also expand/collapse complex variables, i.e. struct type variables, strings...

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.



View RAM Window

Debugger View RAM Window is available from the drop-down menu, **View > Debug Windows > View RAM**.

The View RAM Window displays the map of PIC's RAM, with recently changed items colored red.

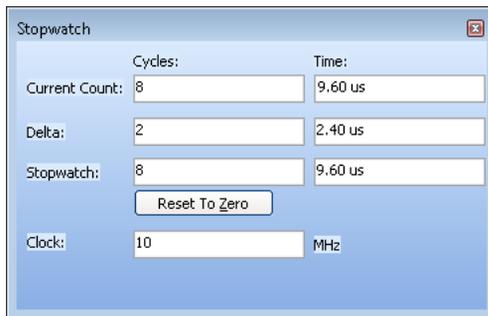
RAM	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	00	0C	96	0C	00	00	3F	00	00	00	00	00	00	00	D9	22	...
0010	00	00	00	DF	00	10	A0	00	00	00	00	C0	A0	00	00	0F	...
0020	00	00	04	00	00	E0	80	00	0C	84	81	00	01	08	20	00	...
0030	11	00	25	00	49	08	00	40	04	00	44	10	00	10	00	00	...
0040	00	84	20	00	12	80	00	22	00	10	00	01	80	04	00	48	...
0050	00	00	00	04	00	00	20	80	01	00	00	00	02	00	01	00	...
0060	4C	06	20	00	08	02	00	08	08	14	01	19	00	10	00	A0	...
0070	00	08	15	60	06	00	01	08	10	01	21	66	26	50	00	00	...
0080	00	FF	96	0C	00	3F	7F	FF	FF	07	00	00	00	00	00	00	...
0090	00	00	FF	00	00	00	00	00	02	00	00	00	07	00	00	00	...
00A0	80	01	84	00	40	20	03	00	00	04	00	18	00	41	20	00	...
00B0	08	00	01	10	02	00	00	00	00	00	04	40	02	28	24	82	...
00C0	00	22	01	11	06	70	11	58	84	00	02	10	00	80	28	80	...
00D0	00	00	01	00	10	00	10	10	41	04	00	00	00	01	40	84	...
00E0	00	00	04	20	50	20	00	90	00	40	40	84	21	14	80	28	...
00F0	00	08	15	60	06	00	01	08	10	01	21	66	26	50	00	00	...

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.



SOFTWARE SIMULATOR OPTIONS

Format	Description	File Type	Toolbar Icon
Start Debugger	Start Software Simulator.	[F9]	
Run/Pause Debugger	Run or pause Software Simulator.	[F6]	
Stop Debugger	Stop Software Simulator.	[Ctrl+F2]	
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.	[F5]	
Run to cursor	Execute all instructions between the current instruction and cursor position.	[F4]	
Step Into	Execute the current Pascal (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.	[F7]	
Step Over	Execute the current Pascal (single or multi-cycle) instruction, then halt.	[F8]	
Step Out	Execute all remaining instructions in the current routine, return and then halt.	[Ctrl+F8]	

Related topics: Run Menu, Debug Toolbar

CREATING NEW LIBRARY

mikroPascal PRO for PIC allows you to create your own libraries. In order to create a library in mikroPascal PRO for PIC follow the steps below:

1. Create a new Pascal source file, see Managing Source Files
2. Save the file in one of the subfolders of the compiler's Uses folder:
`DriveName:\Program Files\Mikroelektronika\mikroPascal PRO for PIC\Uses\P16\
DriveName:\Program Files\Mikroelektronika\mikroPascal PRO for PIC\Uses\P18\`

If you are creating library for PIC16 MCU family the file should be saved in P16 folder. If you are creating library for PIC18 MCU family the file should be saved in P18 folder. If you are creating library for PIC16 and PIC18 MCU families the file should be saved in both folders.

3. Write a code for your library and save it.
4. Add `__Lib_Example` file in some project, see Project Manager. Recompile the project.

If you wish to use this library for all MCUs, then you should go to **Tools > Options > Output settings**, and check **Build all files as library** box.

This will build libraries in a common form which will work with all MCUs. If this box is not checked, then library will be built for selected MCU.

Bear in mind that compiler will report an error if a library built for specific MCU is used for another one.

5. Compiled file `__Lib_Example.mcl` should appear in `...\mikroPascal PRO for PIC\Uses\P16\` 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\mikroPascal PRO for PIC\Defs\  
and it is named MCU_NAME.mlk, for example 16F887.mlk
```

7. Add the the following segment of code to `<LIBRARIES>` node of the definition file (definition file is in XML format):

```
<LIB>  
  <ALIAS>Example_Library</ALIAS>  
  <FILE>__Lib_Example</FILE>  
  <TYPE>REGULAR</TYPE>  
</LIB>
```

8. Add Library to mlk file for each MCU that you want to use with your library.
9. Click Refresh button in Library Manager
10. `Example_Library` should appear in the Library manager window.

Multiple Library Versions

Library Alias represents unique name that is linked to corresponding Library .mcl file. For example UART library for 16F887 is different from UART library for 18F4520 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

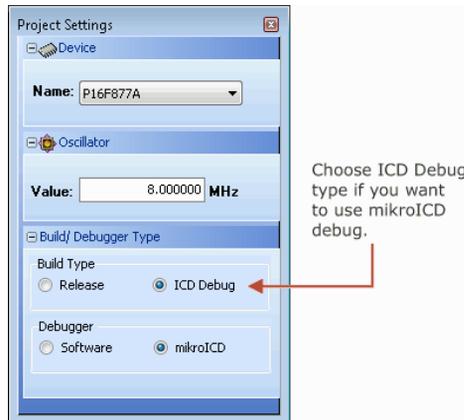
3

mikroICD (In-Circuit Debugger)

mikroICD is highly effective tool for **Real-Time debugging** on hardware level. ICD debugger enables you to execute a mikroPascal program on a host PIC microcontroller and view variable values, Special Function Registers (SFR), memory and EEPROM as the program is running.

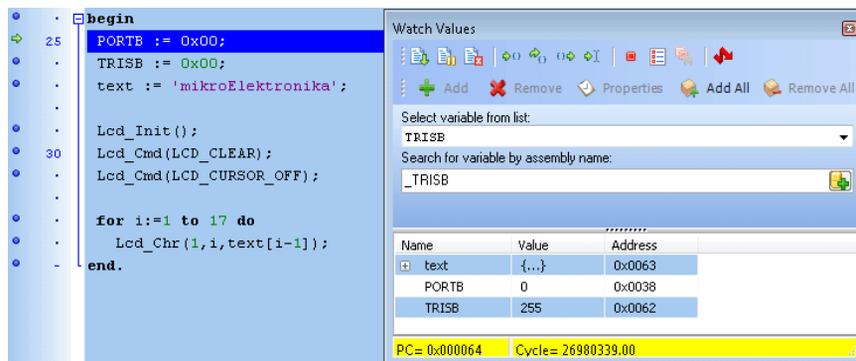
Step No. 1

If you have appropriate hardware and software for using mikroICD, then, upon completion of writing your program, you will have to choose **ICD Debug** build type.



Step No. 2

You can run the mikroICD by selecting **Run > Debug** from the drop-down menu, or by clicking Debug Icon . Starting the Debugger makes more options available: Step Into, Step Over, Run to Cursor, etc. Line that is to be executed is color highlighted (blue by default). There is also notification about program execution and it can be found on Watch Window (yellow status bar). Note that some functions take time to execute, so running of program is indicated on Watch Window.



MIKROICD DEBUGGER OPTIONS

Name	Description	Function Key
Debug	Starts Debugger.	[F9]
Run/ Pause Debugger	Run or pause Debugger.	[F6]
Toggle Breakpoints	Toggle breakpoint at the current cursor position. To view all the breakpoints, select Run > View Breakpoints from the drop-down menu. Double clicking an item in window list locates the breakpoint.	[F5]
Run to cursor	Execute all instructions between the current instruction and the cursor position.	[F4]
Step Into	Execute the current C (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.	[F7]
Step Over	Execute the current C (single- or multi-cycle) instruction, then halt. If the instruction is a routine call, skip it and halt at the first instruction following the call.	[F8]
Flush RAM	Flushes current PIC RAM. All variable values will be changed according to values from watch window.	N/A
Disassembly View	Toggle between disassembly and Pascal source view.	[Alt+D]

MIKROICD DEBUGGER EXAMPLE

Here is a step by step mikroLCD Debugger Example.

Step No. 1

First you have to write a program. We will show how mikroLCD works using this example:

```
program Lcd_Demo;

var LCD_RS : sbit at RB4_bit;
var LCD_EN : sbit at RB5_bit;
var LCD_D4 : sbit at RB0_bit;
var LCD_D5 : sbit at RB1_bit;
var LCD_D6 : sbit at RB2_bit;
var LCD_D7 : sbit at RB3_bit;

var LCD_RS_Direction : sbit at TRISB4_bit;
var LCD_EN_Direction : sbit at TRISB5_bit;
var LCD_D4_Direction : sbit at TRISB0_bit;
var LCD_D5_Direction : sbit at TRISB1_bit;
var LCD_D6_Direction : sbit at TRISB2_bit;
var LCD_D7_Direction : sbit at TRISB3_bit;

var text : array[16] of char;
    i : byte;

begin
    PORTB := 0;
    TRISB := 0;
    ANSEL := 0;
    ANSELH := 0;
    text := 'mikroElektronika';

    Lcd_Init();
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Cmd(_LCD_CURSOR_OFF);

    for i := 1 to 17 do
        Lcd_Ch(1, i, text[i-1]);
    end.
```

Step No. 2

After successful compilation and PIC programming press **F9** for starting mikroICD. After mikroICD initialization blue active line should appear.

The screenshot shows the mikroPASCAL PRO IDE interface. On the left, the code editor displays the following code:

```

20 begin
21 PORTB := 0;
   TRISB := 0;
   ANSEL := 0;
   ANSELH := 0;
   text := 'mikroElektronika';
   .
   .
   Lcd_Init();
   Lcd_Cmd(LCD_CLEAR);
   Lcd_Cmd(LCD_CURSOR_OFF);
30
   for i := 1 to 17 do
   .
   .
     Lcd_Chr(1, i, text[i-1]);
   .
   .
end.

```

On the right, the Watch Values window is open, showing a table of variables and their values:

Name	Value	Address
PORTB	0	0x0006
TRISB	127	0x0086
ANSEL	255	0x0188
ANSELH	63	0x0189
text	{...}	0x0022

At the bottom of the Watch Values window, it shows PC= 0x000005 and Cycle= 0.00.

Step No. 3

We will debug program line by line. Pressing **F8** we are executing code line by line. It is recommended that user does not use Step Into [F7] and Step Over [F8] over Delays routines and routines containing delays. Instead use Run to cursor [F4] and Breakpoints functions.

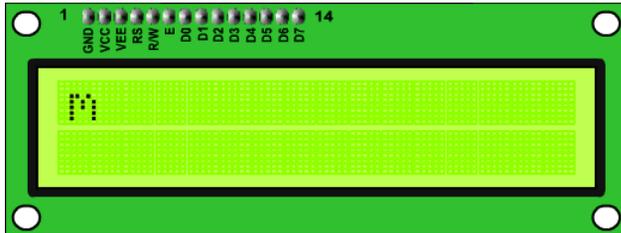
All changes are read from PIC and loaded into Watch Window. Note that **TRISB** changed its value from 255 to 0.

The screenshot shows the mikroPASCAL PRO IDE interface, identical to Step No. 2. The code editor displays the same code. The Watch Values window is open, showing the same table of variables and their values:

Name	Value	Address
PORTB	0	0x0006
TRISB	127	0x0086
ANSEL	255	0x0188
ANSELH	63	0x0189
text	{...}	0x0022

At the bottom of the Watch Values window, it shows PC= 0x000005 and Cycle= 0.00.

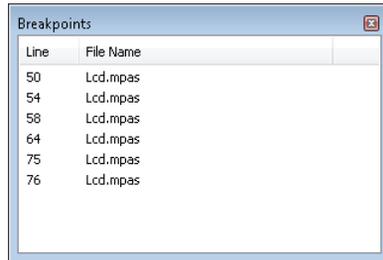
Breakpoints has been separated into two groups. There are hardware and software break points. Hardware breakpoints are placed in PIC and they provide fastest debug. Number of hardware breakpoints is limited (1 for P16 and 1 or 3 or 5 for P18). If all hardware breakpoints are used, next breakpoints that will be used are software breakpoint. Those breakpoints are placed inside mikro ICD, and they simulate hardware breakpoints. Software breakpoints are much slower than hardware breakpoints. This differences between hardware and software differences are not visible in mikroICD software but their different timings are quite notable, so it is important to know that there is two types of breakpoints.



MIKROICD (IN-CIRCUIT DEBUGGER) OVERVIEW

Breakpoints Window

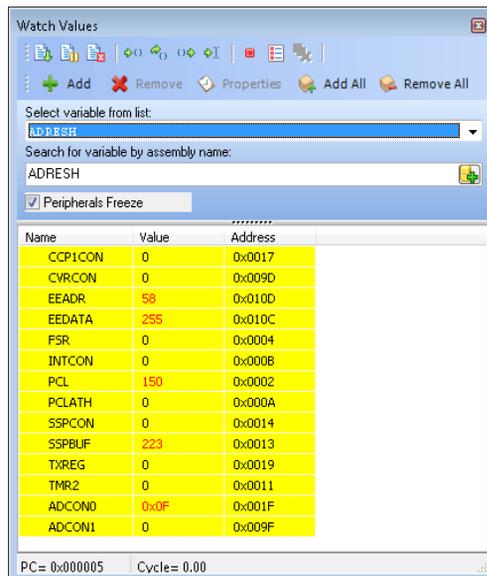
The Breakpoints window manages the list of currently set breakpoints in the project. Doubleclicking the desired breakpoint will cause cursor to navigate to the corresponding location in source code.



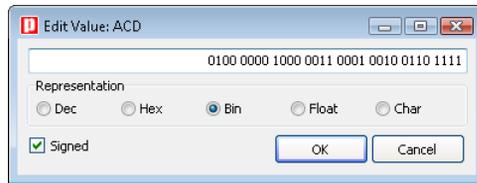
Watch Window

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

The Watch Window displays variables and registers of PIC, with their addresses and values. Values are updated as you go through the simulation. Use the drop-down menu to add and remove the items that you want to monitor. Recently changed items are colored red.



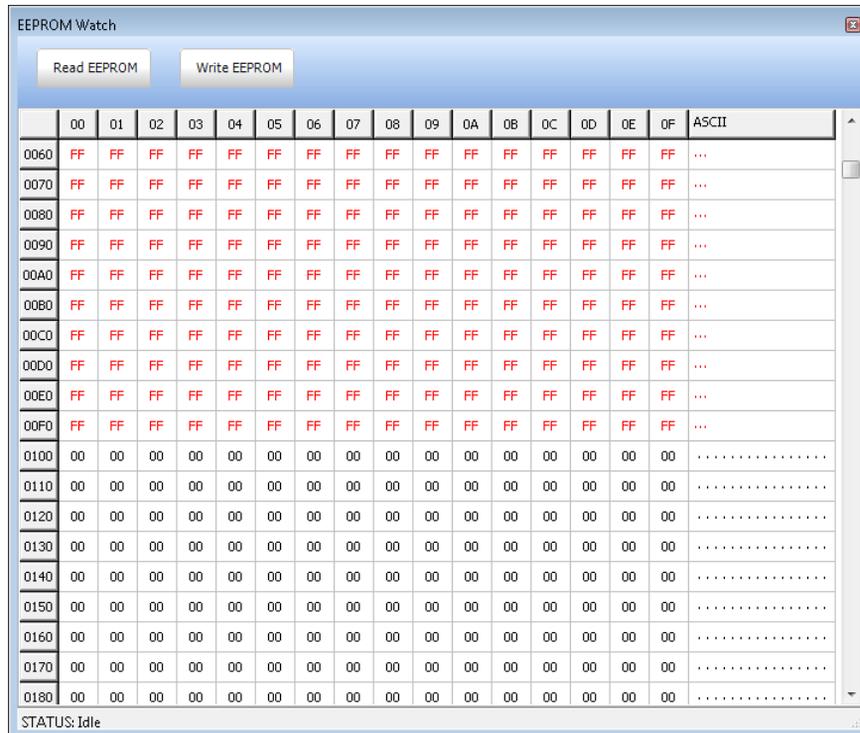
Double clicking an item opens the Edit Value window in which you can assign a new value to the selected variable/register. Also, you can change view to binary, hex, char, or decimal for the selected item.



EEPROM Watch Window

mikroICD EEPROM Watch Window is available from the drop-down menu, **View » Debug Windows » View EEPROM**.

The EEPROM Watch window shows current values written into PIC internal EEPROM memory. There are two action buttons concerning EEPROM Watch window - **Write EEPROM** and **Read EEPROM**. **Write EEPROM** writes data from EEPROM Watch window into PIC internal EEPROM memory. **Read EEPROM** reads data from PIC internal EEPROM memory and loads it up in EEPROM window.

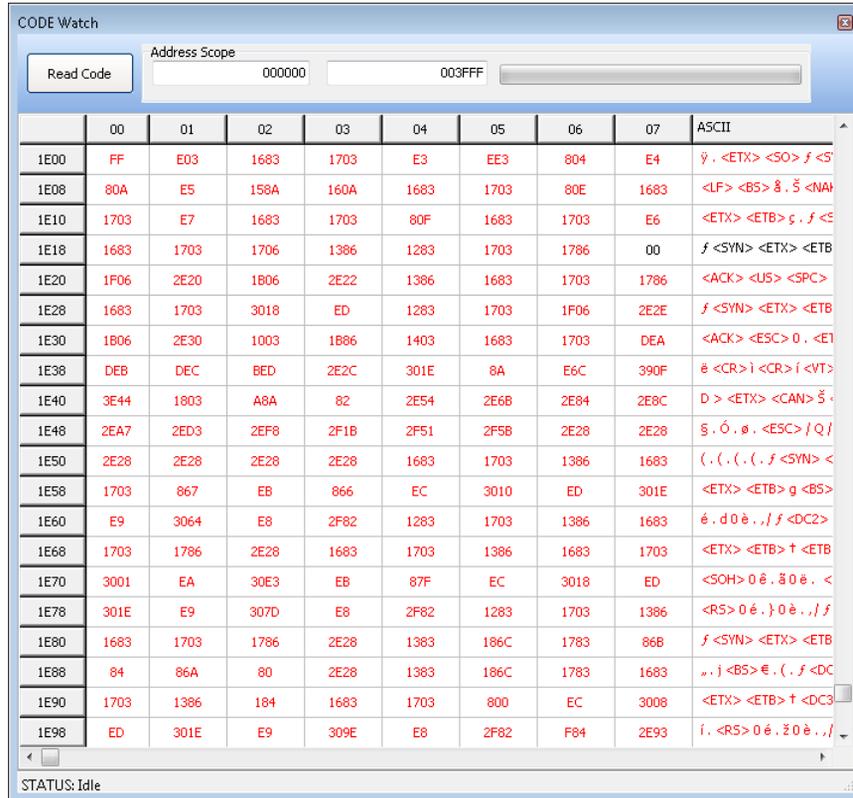


Code Watch Window

mikroICD Code Watch Window is available from the drop-down menu, **View » Debug Windows » View Code**.

The Code Watch window shows code (hex code) written into PIC. There is action button concerning Code Watch window - **Read Code**. **Read Code** reads code from PIC and loads it up in View Code Window.

Also, you can set an address scope in which hex code will be read.



View RAM Window

Debugger View RAM Window is available from the drop-down menu, **View > Debug Windows > View RAM**.

The View RAM Window displays the map of PIC's RAM, with recently changed items colored red.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	00	0C	96	0C	00	00	3F	00	00	00	00	00	00	00	D9	22	...
0010	00	00	00	DF	00	10	A0	00	00	00	00	C0	A0	00	00	0F	...
0020	00	00	04	00	00	E0	80	00	0C	84	81	00	01	08	20	00	...
0030	11	00	25	00	49	08	00	40	04	00	44	10	00	10	00	00	...
0040	00	84	20	00	12	80	00	22	00	10	00	01	80	04	00	48	...
0050	00	00	00	04	00	00	20	80	01	00	00	00	02	00	01	00	...
0060	4C	06	20	00	08	02	00	08	08	14	01	19	00	10	00	A0	...
0070	00	08	15	60	06	00	01	08	10	01	21	66	26	50	00	00	...
0080	00	FF	96	0C	00	3F	7F	FF	FF	07	00	00	00	00	00	00	...
0090	00	00	FF	00	00	00	00	00	02	00	00	00	07	00	00	00	...
00A0	80	01	84	00	40	20	03	00	00	04	00	18	00	41	20	00	...
00B0	08	00	01	10	02	00	00	00	00	00	04	40	02	28	24	82	...
00C0	00	22	01	11	06	70	11	58	84	00	02	10	00	80	28	80	...
00D0	00	00	01	00	10	00	10	10	41	04	00	00	00	01	40	84	...
00E0	00	00	04	20	50	20	00	90	00	40	40	84	21	14	80	28	...
00F0	00	08	15	60	06	00	01	08	10	01	21	66	26	50	00	00	...

Common Errors

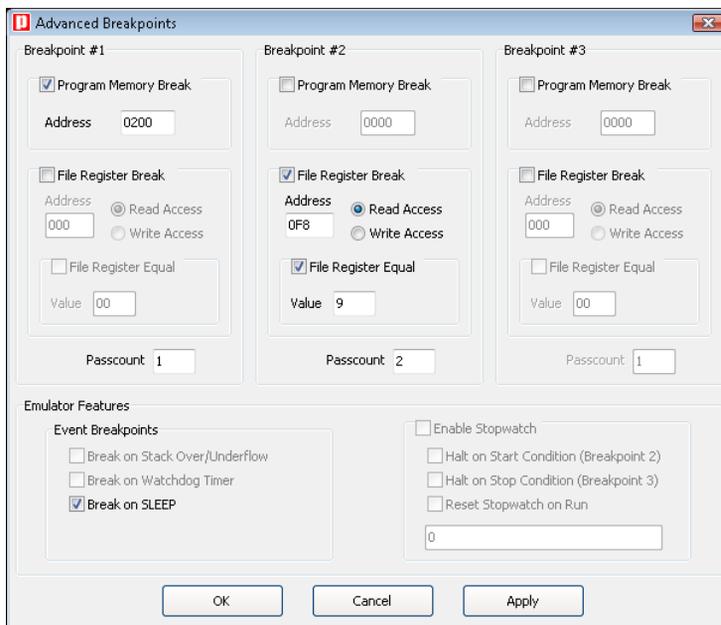
- Trying to program PIC while mikroICD is active.
- Trying to debug **Release** build Type version of program.
- Trying to debug changed program code which hasn't been compiled and programmed into PIC.
- Trying to select line that is empty for Run to cursor [**F4**] and Toggle Breakpoints [**F5**] functions.
- Trying to debug PIC with mikroICD while Watch Dog Timer is enabled.
- Trying to debug PIC with mikroICD while Power Up Timer is enabled.
- It is not possible to force Code Protect while trying to debug PIC with mikroICD.
- Trying to debug PIC with ICD with pull-up resistors set to ON on RB6 and RB7.
- For correct mikroICD debugging do not use pull-ups.

MIKROICD ADVANCED BREAKPOINTS

mikroICD provides the possibility to use the Advanced Breakpoints. Advanced Breakpoints can be used with PIC18 and PIC18FJ MCUs. To enable Advanced Breakpoints set the Advanced Breakpoints checkbox inside Watch window :

Advanced Breakpoints

To configure Advanced Breakpoints, start mikroICD [F9] and select **View > Debug Windows > Advanced Breakpoints** option from the drop-down menu or use [Ctrl+Shift+A] shortcut.



Note: When Advanced Breakpoints are enabled mikroICD operates in Real-Time mode, so it will support only the following set of commands: Start Debugger [F9], Run/Pause Debugger [F6] and Stop Debugger [Ctrl+F2]. Once the Advanced Breakpoint is reached, the Advanced Breakpoints feature can be disabled and mikroICD debugging can be continued with full set of commands. If needed, Advanced Breakpoints can be re-enabled without restarting mikroICD.

Note: Number of Advanced Breakpoints is equal to number of Hardware breakpoints and it depends on used MCU.

Program Memory Break

Program Memory Break is used to set the Advanced Breakpoint to the specific address in program memory. Because of PIC pipelining mechanism program execution may stop one or two instructions after the address entered in the [Address](#) field. Value entered in the Address field must be in hex format.

Note: Program Memory Break can use the Passcount option. The program execution will stop when the specified program address is reached for the N-th time, where N is the number entered in the Passcount field. When some Advanced Breakpoint stops the program execution, passcount counters for all Advanced Breakpoints will be cleared.

File Register Break

File Register Break can be used to stop the code execution when read/write access to the specific data memory location occurs. If Read Access is selected, the File Register Equal option can be used to set the matching value. The program execution will be stopped when the value read from the specified data memory location is equal to the number written in the Value field. Values entered in the [Address](#) and [Value](#) fields must be in hex format.

Note: File Register Break can also use the Passcount option in the same way as Program Memory Break.

Emulator Features

Event Breakpoints

- **Break on Stack Overflow/Underflow** : not implemented.
- **Break on Watchdog Timer** : not implemented.
- **Break on SLEEP** : break on SLEEP instruction. SLEEP instruction will not be executed. If you choose to continue the mikroICD debugging [**F6**] then the program execution will start from the first instruction following the SLEEP instruction.

Stopwatch

Stopwatch uses [Breakpoint#2](#) and [Breakpoint#3](#) as a Start and Stop conditions. To use the Stopwatch define these two Breakpoints and check the [Enable Stopwatch](#) checkbox.

Stopwatch options:

- **Halt on Start Condition (Breakpoint#2)** : when checked, the program execution will stop on `Breakpoint#2`. Otherwise, `Breakpoint#2` will be used only to start the Stopwatch.
- **Halt on Stop Condition (Breakpoint#3)** : when checked, the program execution will stop on `Breakpoint#3`. Otherwise, `Breakpoint#3` will be used only to stop the Stopwatch.
- **Reset Stopwatch on Run** : when checked, the Stopwatch will be cleared before continuing program execution and the next counting will start from zero. Otherwise, the next counting will start from the previous Stopwatch value.

CHAPTER

4

mikroPascal PRO for PIC Specifics

The following topics cover the specifics of mikroPascal PRO for PIC compiler:

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

PASCAL STANDARD ISSUES

Divergence from the Pascal Standard

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

Pascal Language Extensions

mikroPascal PRO for PIC has additional set of keywords that do not belong to the standard Pascal language keywords:

- `code`
- `data`
- `rx`
- `sfr`
- `at`
- `sbit`
- `bit`

Related topics: Keywords, PIC Specific

PREDEFINED GLOBALS AND CONSTANTS

SFRs and related constants

All PIC 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 mikroPascal PRO for PIC will include an appropriate (*.mpas) 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 mikroPascal PRO for PIC 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 mikroPascal PRO for PIC:

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

Predefined project level defines

These defines are based on a value that you have entered/edited in the current project, and it is equal to the name of selected device for the project.

If 16F887 is selected device, then 16F887 token will be defined as 1, so it can be used for conditional compilation:

```
{ $IFDEF P16F887}
...
{ $ENDIF}
```

Related topics: Project level defines

ACCESSING INDIVIDUAL BITS

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

Accessing Individual Bits Of Variables

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

```
// Clear Global Interrupt Bit (GIE)
GIE_bit := 0;
```

Also, you can simply use the direct member selector (.) with a variable, followed by a primary expression. Primary expression can be variable, constant, function call or an expression enclosed by parentheses. For individual bit access there are predefined global constants `B0`, `B1`, ... , `B7` with `B7` being the most significant bit:

```
// predefined globals as bit designators
// Clear bit 0 in INTCON register
INTCON.B0 := 0;

// literal constant as bit designator
// Set bit 5 in ADCON0 register
ADCON0.5 := 1;

// expression as bit designator
// Set bit 6 in STATUS register
i := 5;
STATUS.(i+1) := 1;
```

When using literal constants as bit designators instead of predefined ones, make sure not to exceed the appropriate type size.

There is no need of any special declarations. This kind of selective access is an intrinsic feature of mikroPascal PRO for PIC and can be used anywhere in the code. Identifiers `B0`–`B7` are not case sensitive and have a specific namespace. You may override them with your own members `B0`–`B7` within any given structure.

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

sbit type

The mikroPascal PRO for PIC compiler has sbit data type which provides access to bit-addressable SFRs. You can access them in several ways:

```
var LEDA : sbit at PORTA.B0;
var bit_name : sbit at sfr-name.B<bit-position>;

var LEDB : sbit at PORTB.0;
var bit_name : sbit at sfr-name.<bit-position>;

// If you are familiar with a particular MCU and its ports and direc-
// tion registers (TRIS), you can access bits by their names:
var LEDC : sbit at RC0_bit;
var bit_name : sbit at R<port-letter><bit-position>_bit;

var TRISCO : sbit at TRISCO_bit;
var bit_name : sbit at TRIS<port-letter><bit-position>_bit;
```

bit type

The mikroPascal PRO for PIC 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.

```
var bf : bit;    // bit variable
```

There are no pointers to bit variables:

```
var ptr : ^bit;    // invalid
```

An array of type bit is not valid:

```
var arr[5] : bit;    // invalid
```

Note :

- Bit variables can not be initialized.
- Bit variables can not be members of records.
- Bit variables do not have addresses, therefore unary operator `@` (address of) is not applicable to these variables.

Related topics: Predefined globals and constants

INTERRUPTS

Interrupts can be easily handled by means of reserved word `interrupt`. mikroPascal PRO for PIC implicitly declares procedure `interrupt` which cannot be redeclared.

Write your own procedure body to handle interrupts in your application. Note that you cannot call routines from within interrupt due to stack limitations.

mikroPascal PRO for PIC saves the following SFR on stack when entering interrupt and pops them back upon return:

- PIC12 family: `W`, `STATUS`, `FSR`, `PCLATH`
- PIC16 family: `W`, `STATUS`, `FSR`, `PCLATH`
- PIC18 family: `FSR` (fast context is used to save `WREG`, `STATUS`, `BSR`)

P18 priority interrupts

Note: For the P18 family both low and high interrupts are supported.

For P18 low priority interrupts reserved word is `interrupt_low`:

1. function with name `interrupt` will be linked as ISR (interrupt service routine) for high level interrupt
2. function with name `interrupt_low` will be linked as ISR for low level interrupt_low

If interrupt priority feature is to be used then the user should set the appropriate SFR bits to enable it. For more information refer to datasheet for specific device.

Routine Calls from Interrupt

Calling functions and procedures from within the interrupt routine is now possible. The compiler takes care about the registers being used, both in "interrupt" and in "begin" thread, and performs "smart" context-switching between the two, saving only the registers that have been used in both threads.

The functions and procedures that don't have their own frame (no arguments and local variables) can be called both from the interrupt and the "begin" thread.

Interrupt Examples

Here is a simple example of handling the interrupts from `TMRO` (if no other interrupts are allowed):

```
procedure interrupt;  
begin  
    counter := counter + 1;  
    TMRO := 96;  
    INTCON := $20;  
end;
```

In case of multiple interrupts enabled, you need to test which of the interrupts occurred and then proceed with the appropriate code (interrupt handling):

```
procedure interrupt;  
begin  
  
    if TestBit(INTCON, TMR0IF) = 1 then  
    begin  
        counter := counter + 1;  
        TMRO := 96;  
        ClearBit(INTCON, TMR0F);  
        // ClearBit is realised as an inline function,  
        // and may be called from within an interrupt  
    end  
  
    else  
  
        if TestBit(INTCON, RBIF) = 1 then  
        begin  
            counter := counter + 1;  
            TMRO := 96;  
            ClearBit(INTCON, RBIF);  
        end;  
    end;  
end;
```

LINKER DIRECTIVES

mikroPascal PRO for PIC uses internal algorithm to distribute objects within memory. If you need to have a variable or a 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`

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

Directive `absolute` is appended to the declaration of a variable:

```
var x : word; absolute $32;
// Variable x will occupy 1 word (16 bits) at address $32

    y : longint; absolute $34;
// Variable y will occupy 2 words at addresses $34 and $36
```

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

```
var i : word; absolute $42;
// Variable i will occupy 1 word at address $42;

    jj : longint; absolute $40;
// Variable will occupy 2 words at $40 and $42; thus,
// changing i changes jj at the same time and vice versa
```

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

Directive org

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

```
procedure proc(par : byte); org $200;
begin
// Procedure will start at address $200;
...
end;
```

org directive can be used with main routine too. For example:

```
program Led_Blinking;

procedure some_proc();
begin
...
end;

org 0x800;           // main procedure starts at 0x800
begin
    TRISB := 0;

    while TRUE do
        begin
            PORTB := 0x00;
            Delay_ms(500);
            PORTB := 0xFF;
            Delay_ms(500);
        end;
    end.
```

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

Directive orgall

Use the orgall directive to specify the address above which all routines, constants will be placed. Example:

```
begin
    orgall(0x200) // All the routines, constants in main program will
    be above the address 0x200

    ...

end.
```

BUILT-IN ROUTINES

The mikroPascal PRO for PIC 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 Pascal routines. Their sources can be found in `Delays.mpas` file located in the `uses` folder of the compiler.

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

- `Inc`
- `Dec`

- `SetBit`
- `ClearBit`
- `TestBit`

- `Delay_us`
- `Delay_ms`

- `Clock_KHz`
- `Clock_MHz`

- `Reset`
- `ClrWdt`

- `DisableContextSaving`

- `SetFuncCall`

- `GetDateTime`
- `GetVersion`

Lo

Prototype	<code>function Lo(number: longint): 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>function Hi(number: longint): 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>function Higher(number: longint): 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>function Highest(number: longint): 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	<code>d := 0x1AC30F4; tmp := Highest(d); // Equals 0x01</code>

Inc

Prototype	<code>procedure Inc(var par : longint);</code>
Returns	Nothing.
Description	Increases parameter par by 1.
Requires	Nothing.
Example	<code>p := 4; Inc(p); // p is now 5</code>

Dec

Prototype	<code>procedure Dec(var par : longint);</code>
Returns	Nothing.
Description	Decreases parameter par by 1.
Requires	Nothing.
Example	<code>p := 4; Dec(p); // p is now 3</code>

SetBit

Prototype	<code>procedure SetBit(var register : byte; rbit : byte);</code>
Returns	Nothing.
Description	<p>Function sets the bit <code>rbit</code> of <code>register</code>. Parameter <code>rbit</code> needs to be a variable or literal with value 0..7. See Predefined globals and constants for more information on register identifiers.</p> <p>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.</p>
Requires	Nothing.
Example	<code>SetBit(PORTB, 2); // Set RB2</code>

ClearBit

Prototype	<code>procedure ClearBit(var register : byte; rbit : byte);</code>
Returns	Nothing.
Description	<p>Function clears the bit <code>rbit</code> of <code>register</code>. Parameter <code>rbit</code> needs to be a variable or literal with value 0..7. See Predefined globals and constants for more information on register identifiers.</p> <p>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.</p>
Requires	Nothing.
Example	<code>ClearBit(PORTC, 7); // Clear RC7</code>

TestBit

Prototype	<code>function TestBit(register, rbit : byte) : byte;</code>
Returns	If the bit is set, returns 1, otherwise returns 0.
Description	<p>Function tests if the bit <code>rbit</code> of <code>register</code> is set. If set, function returns 1, otherwise returns 0. Parameter <code>rbit</code> needs to be a variable or literal with value 0..7. See Predefined globals and constants for more information on register identifiers.</p> <p>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.</p>
Requires	Nothing.
Example	<code>flag := TestBit(PORTE, 2); // 1 if RE2 is set, otherwise 0</code>

Delay_us

Prototype	<code>procedure Delay_us(time_in_us: const 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>procedure Delay_ms(time_in_ms: const 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>

Clock_KHz

Prototype	<code>function Clock_KHz(): 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>function Clock_MHz(): 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>

Reset

Prototype	<code>procedure Reset;</code>
Returns	Nothing.
Description	This procedure is equal to assembler instruction reset . This procedure works only for P18.
Requires	Nothing.
Example	<code>Reset; //Resets the PIC MCU</code>

ClrWdt

Prototype	<code>procedure ClrWdt();</code>
Returns	Nothing.
Description	This procedure is equal to assembler instruction clrwtd .
Requires	Nothing.
Example	<code>ClrWdt; //Clears PIC's WDT</code>

DisableContextSaving

Prototype	<code>procedure DisableContextSaving();</code>
Returns	Nothing.
Description	Use the DisableContextSaving() to instruct the compiler not to automatically perform context-switching. This means that no register will be saved/restored by the compiler on entrance/exit from interrupt service routine. This enables the user to manually write code for saving registers upon entrance and to restore them before exit from interrupt.
Requires	This routine must be called from main.
Example	<code>DisableContextSaving(); //instruct the compiler not to automatically perform context-switching</code>

SetFuncCall

Prototype	<code>procedure SetFuncCall(FuncName: string);</code>
Returns	Nothing.
Description	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. Function prepares the caller tree, and informs linker about the procedure usage, making it possible to link the called routine.
Requires	Nothing.
Example	<code>procedure first(p, q: byte); begin ... SetFuncCall(second); // let linker know that we will call the routine 'second' ... end</code>

GetDateTime

Prototype	<code>function GetDateTime() : string;</code>
Returns	String with date and time when this routine is compiled.
Description	Use the GetDateTime() to get date and time of compilation as string in your code.
Requires	Nothing.
Example	<code>str := GetDateTime();</code>

GetVersion

Prototype	<code>function GetVersion() : string;</code>
Returns	String with current compiler version.
Description	Use the GetVersion() to get the current version of compiler.
Requires	Nothing.
Example	<code>str := GetVersion(); // for example, str will take the value of '8.2.1.6'</code>

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. $(3 + 5 \rightarrow 8)$;

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 payed to implement specific solutions for the code "building bricks" that further reduce output code size.

CHAPTER

5

PIC Specifics

In order to get the most from your mikroPascal PRO for PIC compiler, you should be familiar with certain aspects of PIC MCU. This knowledge is not essential, but it can provide you a better understanding of PICs' capabilities and limitations, and their impact on the code writing.

Types Efficiency

First of all, you should know that PIC's ALU, which performs arithmetic operations, is optimized for working with bytes. Although mikroPascal PRO for PIC is capable of handling very complex data types, PIC 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.

Get to know your tool. When it comes down to calculus, not all PIC MCUs are of equal performance. For example, PIC16 family lacks hardware resources to multiply two bytes, so it is compensated by a software algorithm. On the other hand, PIC18 family has HW multiplier, and as a result, multiplication works considerably faster.

Nested Calls Limitations

Nested call represents a function call within function body, either to itself (recursive calls) or to another function. Recursive function calls are supported by mikroPascal PRO for PIC but with limitations. Recursive function calls can't contain any function parameters and local variables due to the PIC's stack and memory limitations.

mikroPascal PRO for PIC limits the number of non-recursive nested calls to:

- 8 calls for PIC12 family,
- 8 calls for PIC16 family,
- 31 calls for PIC18 family.

Note that some of the built-in routines do not count against this limit, due to their "inline" implementation.

Number of the allowed nested calls decreases by one if you use any of the following operators in the code: * / %. It further decreases if you use interrupts in the program. Number of decreases is specified by number of functions called from interrupt. Check functions reentrancy.

If the allowed number of nested calls is exceeded, the compiler will report a stack overflow error.

PIC18FxxJxx Specifics

Shared Address SFRs

mikroPascal PRO for PIC does not provide auto setting of bit for accessing alternate register. This is new feature added to pic18fxxjxx family and will be supported in future. In several locations in the SFR bank, a single address is used to access two different hardware registers. In these cases, a “legacy” register of the standard PIC18 SFR set (such as OSCCON, T1CON, etc.) shares its address with an alternate register. These alternate registers are associated with enhanced configuration options for peripherals, or with new device features not included in the standard PIC18 SFR map. A complete list of shared register addresses and the registers associated with them is provided in datasheet.

PIC16 Specifics

Breaking Through Pages

In applications targeted at PIC16, no single routine should exceed one page (2,000 instructions). If routine does not fit within one page, linker will report an error. When confront with this problem, maybe you should rethink the design of your application – try breaking the particular routine into several chunks, etc.

Limits of Indirect Approach Through FSR

Pointers with PIC16 are “near”: they carry only the lower 8 bits of the address. Compiler will automatically clear the 9th bit upon startup, so that pointers will refer to banks 0 and 1. To access the objects in banks 2 or 3 via pointer, user should manually set the IRP, and restore it to zero after the operation. The stated rules apply to any indirect approach: arrays, structures and unions assignments, etc.

Note: It is very important to take care of the IRP properly, if you plan to follow this approach. If you find this method to be inappropriate with too many variables, you might consider upgrading to PIC18.

Note: If you have many variables in the code, try rearranging them with the linker directive absolute. Variables that are approached only directly should be moved to banks 3 and 4 for increased efficiency.

Related topics: mikroPascal PRO for PIC specifics

MEMORY TYPE SPECIFIERS

The mikroPascal PRO for PIC 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.

The following memory type specifiers can be used:

- code
- data
- rx
- sfr

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

```
var data_buffer : char; data;           // puts data_buffer in data ram
const txt = 'Enter parameter'; code; // puts text in program memory
```

code

Description	The <code>code</code> memory type may be used for allocating constants in program memory.
Example	<pre>// puts txt in program memory const txt = 'Enter parameter'; code;</pre>

data

Description	This memory specifier is used when storing variable to the internal data SRAM.
Example	<pre>// puts data_buffer in data ram var data_buffer : char; data;</pre>

rx

Description	This memory specifier allows variable to be stored in the Rx space (Register file). Note: In most of the cases, there will be enough space left for the user variables in the Rx space. However, since compiler uses Rx space for storing temporary variables, it might happen that user variables will be stored in the internal data SRAM, when writing complex programs.
Example	<pre>// puts y in Rx space var y : char; rx;</pre>

sfr

Description	This memory specifier in combination with (<code>rx</code> , <code>io</code> , <code>data</code>) allows user to access special function registers. It also instructs compiler to maintain same identifier in Pascal and assembly.
Example	<code>var y : char; rx; sfr; // puts y in Rx space</code>

Note: If none of the memory specifiers are used when declaring a variable, `data` specifier will be set as default by the compiler.

Related topics: Accessing individual bits, SFRs, Constants, Functions

CHAPTER



mikroPascal PRO for PIC Language Reference

The mikroPascal PRO for AVR Language Reference describes the syntax, semantics and implementation of the mikroPascal PRO for AVR language.

The aim of this reference guide is to provide a more understandable description of the mikroPascal PRO for AVR language to the user.

Lexical Elements

- Whitespace
- Comments
- Tokens
 - Literals
 - Keywords
 - Identifiers
 - Punctuators

Program Organization

- Program Organization
- Scope and Visibility
- Units

Variables

Constants

Labels

Functions and Procedures

- Functions
- Procedures

Types

- Simple Types
- Arrays
- Strings
- Pointers
- Records
- 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
- Compound Statements (Blocks)
- Conditional Statements

- If Statement
- Case Statement
- Iteration Statements (Loops)
 - For Statement
 - While Statement
 - Repeat Statement
- Jump Statements
 - Break and Continue Statements
 - Exit Statement
 - Goto Statement
- asm Statement
- Directives**
 - Compiler Directives
 - Linker Directives

LEXICAL ELEMENTS OVERVIEW

The following topics provide a formal definition of the mikroPascal PRO for PIC lexical elements. They describe different categories of word-like units (tokens) recognized by mikroPascal PRO for PIC.

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

WHITESPACE

Whitespace is a collective name given to spaces (blanks), horizontal and vertical tabs, newline characters and comments. Whitespace can serve to indicate where tokens start and end, but beyond this function, any surplus whitespace is discarded. For example, two sequences

```
var i : char;  
    j : word;and
```

```
var  
i : char;  
  
    j : word;
```

are lexically equivalent and parse identically to give nine tokens:

```
var  
i  
:  
char  
;  
j  
:  
word  
;
```

Whitespace in Strings

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

```
some_string := 'mikro foo';
```

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

```
some_string  
:=  
'mikro foo'  
;
```

COMMENTS

Comments are pieces of a 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.

There are two ways to create comments in mikroPascal. You can use multi-line comments which are enclosed with braces or (* and *):

```
{ All text between left and right brace  
  constitutes a comment. May span multiple lines. }
```

```
(* Comment can be  
   written in this way too. *)
```

or single-line comments:

```
// Any text between a double-slash and the end of the  
// line constitutes a comment spanning one line only.
```

Nested comments

mikroPascal PRO for PIC doesn't allow nested comments. The attempt to nest a comment like this

```
{ i { identifier } : word; }
```

fails, because the scope of the first open brace "{" ends at the first closed brace "}". This gives us

```
: word; }
```

which would generate a syntax error.

TOKENS

Token is the smallest element of the Pascal program that compiler can recognize. The parser separates tokens from the input stream by creating the longest token possible using the input characters in a left-to-right scan.

mikroPascal PRO for PIC 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. Take a look at the following example code sequence:

```
end_flag := 0;
```

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

The compiler would parse it as the following four tokens:

```
end_flag // variable identifier  
:=      // assignment operator  
0       // literal  
;       // statement terminator
```

Note that `:=` parses as one token (the longest token possible), not as token `:` followed by token `=`.

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, `%01010000`).

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 mikroPascal PRO for PIC – `longint`. Compiler will report an error if the literal exceeds `2147483647` (`$7FFFFFFF`).

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 the decimal integer or decimal fraction (but not both).

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

mikroPascal PRO for PIC limits floating-point constants to range $\pm 1.17549435082 * 10^{-38} .. \pm 6.80564774407 * 10^{38}$.

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 apostrophes.

Character literal can be assigned to variables of the 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.

Note: Quotes ("") have no special meaning in mikroPascal PRO for PIC.

String Literals

String literal is a sequence of characters from the extended ASCII character set, written in one line and enclosed with apostrophes. 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 apostrophes (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 apostrophe itself cannot be a part of the string literal, i.e. there is no escape sequence. You can use the built-in function `Chr` to print an apostrophe: `Chr(39)`. Also, see String Splicing.

KEYWORDS

Keywords are the words reserved for special purposes and must not be used as normal identifier names.

Beside standard Pascal keywords, all relevant SFRs are defined as global variables and represent reserved words that cannot be redefined (for example: W0, TMR1, T1CON, etc). Probe the Code Assistant for specific letters (Ctrl+Space in Editor) or refer to Predefined Globals and Constants.

Here is the alphabetical listing of keywords in Pascal:

- absolute	- for	- procedure
- abstract	- forward	- program
- and	- goto	- property
- array	- helper	- protected
- as	- idata	- public
- asm	- if	- published
- assembler	- ilevel	- raise
- at	- implementation	- read
- automated	- implements	- readonly
- bdata	- in	- record
- begin	- index	- register
- bit	- inherited	- reintroduce
- case	- initialization	- repeat
- cdecl	- inline	- requires
- class	- interface	- safecall
- code	- is	- sbit
- compact	- label	- sealed
- const	- library	- set
- constructor	- message	- shl
- contains	- mod	- shr
- data	- name	- small
- default	- near	- stdcall
- deprecated	- nil	- stored
- destructor	- nodefault	- string
- dispid	- not	- threadvar
- dispinterface	- object	- to
- div	- of	- try
- do	- on	- type
- downto	- operator	- unit
- dynamic	- or	- until
- end	- org	- uses
- except	- out	- var
- export	- overload	- virtual
- exports	- override	- volatile
- external	- package	- while
- far	- packed	- with
- file	- pascal	- write
- final	- pdata	- writeonly
- finalization	- platform	- xdata
- finally	- private	- xor

Also, mikroPascal PRO for PIC includes a number of predefined identifiers used in libraries. You can replace them by your own definitions, if you plan to develop your own libraries. For more information, see mikroPascal for PIC 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 get confused about the meaning of object in object-oriented programming).

Identifiers can contain the letters a to z and A to Z, underscore character “_”, and digits from 0 to 9. The only restriction is that the first character must be a letter or an underscore.

Case Sensitivity

Pascal is not case sensitive, so `Sum`, `sum`, and `suM` are an equivalent identifier.

Uniqueness and Scope

Although identifier names are arbitrary (according to the stated rules), if the same name is used for more than one identifier within the same scope then error arises. Duplicated names are illegal within 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 mikroPascal punctuators (also known as separators) are:

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

Brackets

Brackets [] indicate single and multidimensional array subscripts:

```
var alphabet : array[1..30] of byte;
// ...
alphabet[3] := '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
function func2(n : word); // Function declaration with parameters
```

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

Comma

Comma (,) separates the arguments in function calls:

```
LCD_Out(1, 1, txt);
```

Further, the comma separates identifiers in declarations:

```
var i, j, k : byte;
```

The comma also separates elements of array in initialization lists:

```
const MONTHS : array[1..12] of byte =
(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
```

Semicolon

Semicolon (;) is a statement terminator. Every statement in Pascal must be terminated with a semicolon. The exceptions are: the last (outer most) `end` statement in the program which is terminated with a dot and the last statement before `end` which doesn't need to be terminated with a semicolon.

For more information, see Statements.

Colon

Colon (:) is used in declarations to separate identifier list from type identifier. For example:

```
var
  i, j : byte;
  k    : word;
```

In the program, use the colon to indicate a labeled statement:

```
start:  nop;
  ...
goto start;
```

For more information, refer to Labels.

Dot

Dot (.) indicates an access to a field of a record. For example:

```
person.surname := 'Smith';
```

For more information, refer to Records.

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

PROGRAM ORGANIZATION

Pascal imposes quite 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 Units and Scope and Visibility.

Organization of Main Unit

Basically, the 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 unit should look like this:

```

program { program name }
uses { include other units }

//*****
/* Declarations (globals):
//*****

{ constants declarations }
const ...

{ types declarations }
type ...

{ variables declarations }
var Name[, Name2...] : [ ^ ]type; [ absolute 0x123;] [ external;]
[ volatile;] [ register;] [ sfr;]

{ labels declarations }
label ...

{ procedures declarations }
procedure procedure_name(parameter_list);
  { local declarations }
  begin
    ...
  end;

{ functions declarations }
function function_name(parameter_list) : return_type;
  { local declarations }
  begin
    ...
  end;

```

```

//*****
/* Program body:
//*****

begin
  { write your code here }
end.

```

Organization of Other Units

Units other than main start with the keyword `unit`. Implementation section starts with the keyword `implementation`. Follow the model presented below:

```

unit { unit name }
uses { include other units }

//*****
/* Interface (globals):
//*****

{ constants declarations }
const ...

{ types declarations }
type ...

{ variables declarations }
var Name[, Name2...] : [ ^ ]type; [ absolute 0x123;] [ external;]
[ volatile;] [ register;] [ sfr;]

{ procedures prototypes }
procedure procedure_name([ var] [ const] ParamName : [ ^ ]type; [ var]
[ const] ParamName2, ParamName3 : [ ^ ]type);

{ functions prototypes }
function function_name([ var] [ const] ParamName : [ ^ ]type; [ var]
[ const] ParamName2, ParamName3 : [ ^ ]type) : [ ^ ]type;

//*****
/* Implementation:
//*****

implementation

{ constants declarations }
const ...

{ types declarations }
type ...

```

```

{ variables declarations }
var Name[, Name2...] : [ ^ ] type; [ absolute 0x123;] [ external;]
[ volatile;] [ register;] [ sfr;]

{ labels declarations }
label ...

{ procedures declarations }
procedure procedure_name([ var] [ const] ParamName : [ ^ ] type; [ var]
[ const] ParamName2, ParamName3 : [ ^ ] type); [ ilevel 0x123;] [ over-
load;] [ forward;]
  { local declarations }
  begin
  ...
  end;

{ functions declarations }
function function_name([ var] [ const] ParamName : [ ^ ] type; [ var]
[ const] ParamName2, ParamName3 : [ ^ ] type) : [ ^ ] type; [ ilevel 0x123;]
[ overload;] [ forward;]
  { local declarations }
  begin
  ...
  end;

end.

```

Note: constants, types and variables used in the `implementation` section are inaccessible to other units. This feature is not applied to the procedures and functions in the current version, but it will be added to the future ones.

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

Place of declaration	Scope
Identifier is declared in the declaration of a program, function, or procedure	Scope extends from the point where it is declared to the end of the current block, including all blocks enclosed within that scope. Identifiers in the outermost scope (file scope) of the main unit are referred to as globals, while other identifiers are locals.
Identifier is declared in the interface section of a unit	Scope extends the interface section of a unit from the point where it is declared to the end of the unit, and to any other unit or program that uses that unit.
Identifier is declared in the implementation section of a unit, but not within the block of any function or procedure	Scope extends from the point where it is declared to the end of the unit. The identifier is available to any function or procedure in the unit.

Visibility

The visibility of an identifier is that region of the program source code from which 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, i.e. 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.

UNITS

In mikroPascal PRO for PIC, each project consists of a single project file and one or more unit files. Project file, with extension `.mpppi` contains information about the project, while unit files, with extension `.mpas`, contain the actual source code.

Units allow you to:

- break large programs into encapsulated parts 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 unit is stored in its own file and compiled separately. Compiled units are linked to create an application. In order to build a project, the compiler needs either a source file or a compiled unit file (`.mcl` file) for each unit.

Uses Clause

mikroPascal PRO for PIC includes units by means of the uses clause. It consists of the reserved word `uses`, followed by one or more comma-delimited unit names, followed by a semicolon. Extension of the file should not be included. There can be at most one uses clause in each source file, and it must appear immediately after the program (or unit) name.

Here's an example:

```
uses utils, strings, Unit2, MyUnit;
```

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

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

Main Unit

Every project in mikroPascal PRO for PIC requires a single main unit file. The main unit file 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 the Project Wizard, the Code Editor will display a new main unit. It contains the bare-bones of the Pascal program:

```
program MyProject;  
  
{ main procedure }  
begin  
  { Place program code here }  
end.
```

Nothing should precede the keyword `program` except comments. After the program name, you can optionally place the `uses` clause.

Place all global declarations (constants, variables, types, labels, routines) before the keyword `begin`.

Other Units

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

```
unit MyUnit;  
  
implementation  
  
end.
```

Other than comments, nothing should precede the keyword `unit`. After the unit name, you can optionally place the `uses` clause.

Interface Section

Part of the unit above the keyword `implementation` is referred to as interface section. Here, you can place global declarations (constants, variables, labels and types) for the project.

You 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 unit. Prototypes must match the declarations exactly.

Implementation Section

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

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

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

VARIABLES

Variable is 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 a variable.

Variables are declared in the declaration part of the file or routine — each variable needs to be declared before being used. Global variables (those that do not belong to any enclosing block) are declared below the uses statement, above the keyword `begin`.

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

```
var identifier_list : type;
```

`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.

Pascal allows shortened syntax with only one keyword `var` followed by multiple variable declarations. For example:

```
var i, j, k : byte;  
    counter, temp : word;  
    samples : array[100] of word;
```

External Modifier

Use the external modifier to indicate that the actual place and initial value of the variable, or body of the function, is defined in a separate source code unit.

Variables and PIC

Every declared variable consumes part of RAM. Data type of variable determines not only allowed range of values, but also the space variable occupies in RAM. Bear in mind that operations using different types of variables take different time to be completed. mikroPascal PRO for PIC 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 SFRs explicitly, as mikroPascal PRO for PIC automatically declares relevant registers as global variables of `volatile word` see SFR for details.

CONSTANTS

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

Constants are declared in the declaration part of a program or routine. You can declare any number of constants after the keyword `const`:

```
const constant_name [: 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 `type`, the compiler assumes the “smallest” of all types that can accommodate `value`.

Note: You cannot omit `type` when declaring a constant array.

Pascal allows shorthand syntax with only one keyword `const` followed by multiple constant declarations. Here’s an example:

```
const
  MAX : longint = 10000;
  MIN = 1000;      // compiler will assume word type
  SWITCH = 'n';   // compiler will assume char type
  MSG = 'Hello';  // compiler will assume string type
  MONTHS : array[ 1..12] of byte =
(31,28,31,30,31,30,31,31,30,31,30,31);
```

LABELS

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

```
label_identifier : statement
```

Before marking a statement, you must declare a label. Labels are declared in declaration part of unit or routine, similar to variables and constants. Declare labels using the keyword `label`:

```
label label1, ..., labeln;
```

Name of the label needs to be a valid identifier. The label declaration, marked statement, and goto statement must belong to the same block. Hence it is not possible to jump into or out of a procedure or function. Do not mark more than one statement in a block with the same label.

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

```
label loop;  
...  
loop:  
    Beep;  
    goto loop;
```

Note: label should be followed by end of line (CR) otherwise compiler will report an error:

```
label loop;  
...  
loop: Beep; // compiler will report an error  
loop: // compiler will report an error
```

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 a value while procedure does not.

mikroPascal PRO for PIC does not support inline routines.

Functions

A function is declared like this:

```
function function_name(parameter_list) : return_type;
    { local declarations }
begin
    { function body }
end;
```

`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 very similar to variable declaration. In Pascal, parameters are always passed to a function by the value — to pass an argument by address, add the keyword `var` 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 Pascal, a `function_name` is automatically created local variable that can be used for returning a value of a function. mikroPascal PRO for PIC 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 ($n > 0$):

```
function power(x, n : byte) : longint;  
var i : byte;  
begin  
    i := 0; result := 1;  
    if n > 0 then  
        for i := 1 to n do result := result*x;  
    end;
```

Now we could call it to calculate 3^{12} for example:

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

PROCEDURES

Procedure is declared like this:

```
procedure procedure_name(parameter_list);  
    { local declarations }  
begin  
    { procedure body }  
end;
```

`procedure_name` represents a procedure's name and can be any valid identifier. Within parentheses, `parameter_list` is a formal parameter list very similar to variable declaration. In Pascal, parameters are always passed to a procedure by the value — to pass an argument by address, add the keyword `var` 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 the 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:

```
procedure time_prep(var sec, min, hr : byte);
begin
  sec := ((sec and $F0) shr 4)*10 + (sec and $0F);
  min := ((min and $F0) shr 4)*10 + (min and $0F);
  hr := ((hr and $F0) shr 4)*10 + (hr and $0F);
end;
```

A function can return a complex type. Follow the example bellow to learn how to declare and use a function which returns a complex type.

Example:

This example shows how to declare a function which returns a complex type.

```
program Example;

type TCircle = record // Record
  CenterX, CenterY: word;
  Radius: byte;
end;

var MyCircle: TCircle; // Global variable

function DefineCircle(x, y: word; r: byte): TCircle; // DefineCircle
function returns a Record

begin
  result.CenterX := x;
  result.CenterY := y;
  result.Radius := r;
end;
```


TYPES

Pascal 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.

mikroPascal PRO for PIC supports many standard (predefined) and user-defined data types, including signed and unsigned integers of various sizes, arrays, strings, pointers and records.

Type Categories

Types can be divided into:

- simple types
- arrays
- strings
- pointers
- records

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 mikroPascal PRO for PIC has 16 bits.

Here is an overview of simple types in mikroPascal PRO for PIC:

Type	Size	Range
<code>byte</code> , <code>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>dword</code>	32-bit	0 .. 4294967295
<code>longint</code>	32-bit	-2147483648 .. 2147483647
<code>real</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). Because 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:

```
array[ index_start .. index_end] of type
```

Each of the elements of an array is numbered from `index_start` through `index_end`. The specifier `index_start` can be omitted along with dots, in which case it defaults to zero.

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:

```
var
  weekdays : array[ 1..7] of byte;
  samples  : array[ 50] of word;

begin
  // 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      : array[ 1..12]      of    byte    =
(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
```

The number of assigned values must not exceed the specified length. The opposite is possible, when the trailing “excess” elements are assigned zeroes.

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

Multi-dimensional Arrays

Multidimensional arrays are constructed by declaring arrays of array type. These arrays are stored in memory in such way that the right most subscript changes fastest, i.e. arrays are stored “in rows”. Here is a sample 2-dimensional array:

```
m : array[ 5] of array[ 10] of byte;    // 2-dimensional array of size
5x10
```

A variable `m` is an array of 5 elements, which in turn are arrays of 10 byte each. Thus, we have a matrix of 5x10 elements where the first element is `m[0][0]` and last one is `m[4][9]`. The first element of the 4th row would be `m[3][0]`.

STRINGS

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

```
string_name : string[ length]
```

The specifier `length` is a number of characters the string consists of. String is stored internally as the given sequence of characters plus a final null character which is introduced to terminate the string. It does not count against the 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. String on the right side of an assignment operator has to be shorter or of equal length than the one on the right side. For example:

```
var
  msg1 : string[ 20];
  msg2 : string[ 19];

begin
  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:

```

var s : 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.

String Concatenating

mikroPascal PRO for PIC allows you to concatenate strings by means of plus operator. This kind of concatenation is applicable to string variables/literals, character variables/literals. For control characters, use the non-quoted hash sign and a numeral (e.g. #13 for CR).

Here is an example:

```

var msg      : string[ 20];
    res_txt  : string[ 5];
    res, channel : word;

begin
    //...

    // Get result of ADC
    res := Adc_Read(channel);

    // Create string out of numeric result
    WordToStr(res, res_txt);

    // Prepare message for output
    msg := 'Result is ' +      // Text "Result is"
           res_txt           ; // Result of ADC

    //...

```

Note: In current version plus operator for concatenating strings will accept at most two operands.

Note

mikroPascal PRO for PIC includes a 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 carat prefix (^) before type. For example, in order to create a pointer to an integer, write:

`^integer;` In order to access data at the pointer's memory location, add a carat after the variable name. For example, let's declare variable `p` which points to a word, and then assign value 5 to the pointed memory location:

```
var p : ^word;  
...  
p^ := 5;
```

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

Pointers to program memory space are declared using the keyword `const`:

```
program const_ptr;  
  
// constant array will be stored in program memory  
const b_array: array[5] of byte = (1,2,3,4,5);  
  
const ptr: ^byte;      // ptr is pointer to program memory space  
  
begin  
    ptr := @b_array;  // ptr now points to b_array[0]  
    P0 := ptr^;  
    ptr := ptr + 3;   // ptr now points to b_array[3]  
    P0 := ptr^;  
end.
```

Function Pointers

Function pointers are allowed in mikroPascal PRO for PIC. 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;

    type TMyFunctionType = function (param1, param2: byte; param3: word)
    : word; // First, define the procedural type
        var MyPtr: ^TMyFunctionType;
// This is a pointer to previously defined type
    Sample: word;

    function Func1(p1, p2: byte; p3: word): word; // Now,
define few functions which will be pointed to. Make sure that param-
eters match the type definition
    begin
        result := p1 and p2 or p3; // return something
    end;

    function Func2(abc: byte; def: byte; ghi: word): word; // Another
function of the same kind. Make sure that parameters match the type
definition
    begin
        result := abc * def + ghi; // return something
    end;

    function Func3(first, yellow: byte; monday: word): word; // Yet
another function. Make sure that parameters match the type defini-
tion
    begin
        result := monday - yellow - first; // return something
    end;

// main program:
begin
    MyPtr := @Func1; // MyPtr now points to Func1
    Sample := MyPtr^(1, 2, 3); // Perform function call via
pointer, call Func1, the return value is 3
    MyPtr := @Func2; // MyPtr now points to Func2
    Sample := MyPtr^(1, 2, 3); // Perform function call via
pointer, call Func2, the return value is 5
    MyPtr := @Func3; // MyPtr now points to Func3
    Sample := MyPtr^(1, 2, 3); // Perform function call via
pointer, call Func3, the return value is 0
end.

```

@ Operator

The @ operator constructs a pointer to its operand. The following rules are applied to @:

- If *x* is a variable, @*x* returns a pointer to *x*.

Note: If variable *x* is of array type, the @ operator will return pointer to it's first basic element, except when the left side of the statement in which *x* is used is an array pointer.

In this case, the @ operator will return pointer to array, not to it's first basic element.

program example;

```
var w      : word;
    ptr_b   : ^byte;
    ptr_arr : ^array[10] of byte;
    arr     : array[10] of byte;
```

begin

```
    ptr_b := @arr; // @ operator will return ^byte
    w     := @arr; // @ operator will return ^byte
    ptr_arr := @arr; // @ operator will return ^array[10] of byte
```

end.

- If *F* is a routine (a function or procedure), @*F* returns a pointer to *F*.

Related topics: Pointer Arithmetic

RECORDS

A record (analogous to a structure in some languages) represents a heterogeneous set of elements. Each element is called a field. The declaration of the record type specifies a name and type for each field. The syntax of a record type declaration is

```
type recordTypeName = record
    fieldList1 : type1;
    ...
    fieldListn : typen;
end;
```

where `recordTypeName` is a valid identifier, each type denotes a type, and each `fieldList` is a valid identifier or a comma-delimited list of identifiers. The scope of a field identifier is limited to the record in which it occurs, so you don't have to worry about naming conflicts between field identifiers and other variables.

Note: In mikroPascal PRO for PIC, you cannot use the `record` construction directly in variable declarations, i.e. without `type`.

For example, the following declaration creates a record type called `TDot`:

```
type
    TDot = record
        x, y : real;
    end;
```

Each `TDot` contains two fields: `x` and `y` coordinates. Memory is allocated when you declare the record, like this:

```
var m, n: TDot;
```

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

A field can be of previously defined record type. For example:

```
// Structure defining a circle:
type
    TCircle = record
        radius : real;
        center : TDot;
    end;
```

Accessing Fields

You can access the fields of a record by means of dot (.) as a direct field selector. If we have declared variables `circle1` and `circle2` of previously defined type `TCircle`:

```
var circle1, circle2 : TCircle;
```

we could access their individual fields like this:

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

Accessing the fields is possible via the with statement as well.

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

```
circle2 := circle1; // This will copy values of all fields
```

TYPES CONVERSIONS

Conversion of variable of one type to a variable of another type is typecasting. mikroPascal PRO for PIC 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 type to more complex type taking the following steps:

```
byte/char    → word
short        → integer
short        → longint
integer      → longint
integer      → real
```

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:

```
var a : byte; b : 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 a more complex type than expected, excess of data will be simply clipped (higher bytes are lost).

```
var i : byte; j : 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 `real`) ahead of an expression to be converted. The expression must be enclosed in parentheses. Explicit conversion can be performed only on the operand right of the assignment operator.

Special case is 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:

```
var a : byte; b : 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 can't execute explicit conversion on the operand left of the assignment operator:

```
word(b) := a;    // Compiler will report an error
```

Conversions Examples

Here is an example of conversion:

```
var a, b, c : byte;
    d : word;
...
a := 241;
b := 128;

c := a + b;           // equals 113
c := word(a + b);    // equals 113
d := a + b;           // equals 369
```

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 mikroPascal PRO for PIC:

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

OPERATORS PRECEDENCE AND ASSOCIATIVITY

There are 4 precedence categories in mikroPascal PRO for PIC. 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 shl shr	\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
<code>+</code>	addition	byte, short, word, integer, longint, dword, real	byte, short, word, integer, longint, dword, real
<code>-</code>	subtraction	byte, short, word, integer, longint, dword, real	byte, short, word, integer, longint, dword, real
<code>*</code>	multiplication	byte, short, word, integer, longint, dword, real	word, integer, longint, dword, real
<code>/</code>	division, floating-point	byte, short, word, integer, longint, dword, real	real
<code>div</code>	division, rounds down to nearest integer	byte, short, word, integer, longint, dword	byte, short, word, integer, longint, dword
<code>mod</code>	modulus, returns the remainder of integer division (cannot be used with floating points)	byte, short, word, integer, longint, dword	byte, short, word, integer, longint, dword

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
<code>=</code>	equal
<code><></code>	not equal
<code>></code>	greater than
<code><</code>	less than
<code>>=</code>	greater than or equal
<code><=</code>	less than or equal

All relational operators associate from left to right.

Relational Operators in Expressions

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

```
a + 5 >= c - 1.0 / e // ? (a + 5) >= (c - (1.0 / e))
```

BITWISE OPERATORS

Use bitwise operators to modify individual bits of numerical operands. Operands need to be either both signed or both unsigned.

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>shl</code>	bitwise shift left; moves the bits to the left, discards the far left bit and assigns 0 to the right most bit.
<code>shr</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
0	1	0

Bitwise operators `and`, `or`, and `xor` perform logical operations on the appropriate pairs of bits of their operands. `not` operator 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 a number is converted from less complex to more complex data type, the upper bytes are filled with zeroes. If a number is converted from more complex to less complex data type, the data is simply truncated (the upper bytes are lost).

For example:

```

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

For example:

```

var a : byte; b : word;
...
a := -12;
b := $70FF;
b := b and a;

{ a is sign extended, with the upper byte equal to $FF;
  b becomes $70F4 }

```

Bitwise Shift Operators

Binary operators `shl` and `shr` 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 (`shl`), 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 (`shr`), 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 mikroPascal PRO for PIC 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. More complex expressions can be created from primary expressions by using operators. 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 which depend on the operators in use, 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 mikroPascal PRO for PIC.

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
- Compound Statements (Blocks)
- Conditional Statements
- Iteration Statements (Loops)
- Jump Statements

- asm Statement

ASSIGNMENT STATEMENTS

Assignment statements have the form:

```
variable := expression;
```

The statement evaluates expression and assigns its value to `variable`. All the 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. Also note that, although similar, the construction is not related to the declaration of constants.

COMPOUND STATEMENTS (BLOCKS)

Compound statement, or block, is a list of statements enclosed by keywords begin and end:

```
begin
    statements
end;
```

Syntactically, a block is considered to be a single statement which is allowed to be used when Pascal syntax requires a single statement. Blocks can be nested up to the limits of memory.

For example, the while loop expects one statement in its body, so we can pass it a compound statement:

```
while i < n do
    begin
        temp := a[ i ];
        a[ i ] := b[ i ];
        b[ i ] := temp;
        i := i + 1;
    end;
```

Conditional Statements

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

- if
- case

IF STATEMENT

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

```
if expression then statement1 [ else statement2]
```

If `expression` evaluates to true then `statement1` executes. If `expression` is false then `statement2` executes. The `expression` must convert to a boolean type; otherwise, the condition is ill-formed. The `else` keyword with an alternate statement (`statement2`) is optional.

There should never be a semicolon before the keyword `else`.

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
```

The compiler treats the construction in this way:

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

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

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

CASE STATEMENT

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

```
case selector of
  value_1 : statement_1
  ...
  value_n : statement_n
  [else default_statement]
end;
```

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

The `else` clause is optional. If using the `else` branch, note that there should never be a semicolon before the keyword `else`.

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

Here's a simple example of the case statement:

```
case operator of
  '*' : result := n1 * n2;
  '/' : result := n1 / n2;
  '+' : result := n1 + n2;
  '-' : result := n1 - n2
else result := 0;
end;
```

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

```
case reg of
  0:      opmode := 0;
  1,2,3,4: opmode := 1;
  5,6,7:  opmode := 2;
end;
```

In mikroPascal PRO for PIC, `values` in the `case` statement can be variables too:

```
case byte_variable of

    byte_var1: opmode := 0; // this will be compiled correctly

    byte_var2:
        opmode := 1; // avoid this case, compiler will parse
                    // a variable followed by colon sign as
label

    byte_var3: //          adding a comment solves the parsing
problem        opmode := 2;

end;
```

Nested Case statement

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

ITERATION STATEMENTS

Iteration statements let you loop a set of statements. There are three forms of iteration statements in mikroPascal PRO for PIC:

- 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 do statement
// or
for counter := initial_value downto final_value do statement
```

`counter` is a variable which increments (or decrements if you use `downto`) with each iteration of the loop. Before the first iteration, `counter` is set to `initial_value` and will increment (or decrement) until it reaches `final_value`. With each iteration, `statement` will be executed.

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

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 do
  s := s + a[i] * b[i];
```

Endless Loop

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

More legible way to create an endless loop in Pascal is to use the statement `while TRUE do`.

WHILE STATEMENT

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

```
while expression do statement
```

`statement` is executed repeatedly as long as `expression` evaluates true. The test takes place before the `statement` is 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 do
begin
  s := s + a[ i ] * b[ i ];
  i := i + 1;
end;
```

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

```
while TRUE do ...;
```

REPEAT STATEMENT

The `repeat` statement executes until the condition becomes false. The syntax of the `repeat` statement is:

```
repeat statement until expression
```

`statement` is executed repeatedly as long as `expression` evaluates true. The `expression` is evaluated after each iteration, so the loop will execute `statement` at least once.

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

```
s := 0; i := 0;
...
repeat
begin
  s := s + a[ i ] * b[ i ];
  i := i + 1;
end;
until i = n;
```

JUMP STATEMENTS

A jump statement, when executed, transfers control unconditionally. There are four such statements in mikroPascal PRO for PIC:

- break
- continue
- exit
- goto

BREAK AND CONTINUE STATEMENTS

Break Statement

Sometimes, you might need to stop the loop from within its body. Use the break statement within loops to pass control to the first statement following the innermost loop (for, while, or repeat block).

For example:

```
Lcd_Out(1,1,'Insert CF card');  
  
// Wait for CF card to be plugged; refresh every second  
while TRUE do  
begin  
    if Cf_Detect() = 1 then break;  
    Delay_ms(1000);  
end;  
  
// Now we can work with CF card ...  
Lcd_Out(1,1,'Card detected  ');
```

Continue Statement

You can use the continue statement within loops to “skip the cycle”:

- `continue` statement in `for` loop moves program counter to the line with keyword `for`
- `continue` statement in `while` loop moves program counter to the line with loop condition (top of the loop),
- `continue` statement in `repeat` loop moves program counter to the line with loop condition (bottom of the loop).

```
// continue jumps here
for i := ... do
  begin
    ...
    continue;
    ...
  end;

// continue jumps here
while condition do
  begin
    ...
    continue;
    ...
  end;

repeat
  begin
    ...
    continue;
    ...
  // continue jumps here
until condition;
```

EXIT STATEMENT

The `exit` statement allows you to break out of a routine (function or procedure). It passes the control to the first statement following the routine call.

Here is a simple example:

```
procedure Procl();
var error: byte;
begin
  ... // we're doing something here
  if error = TRUE then exit;
  ... // some code, which won't be executed if error is true
end;
```

Note: If breaking out of a function, return value will be the value of the local variable `result` at the moment of exit.

GOTO STATEMENT

Use the `goto` statement to unconditionally jump to a local label — for more information, refer to Labels. Syntax of `goto` statement is:

```
goto label_name;
```

This will transfer control to the location of a local label specified by `label_name`. The `goto` line can come before or after the label.

The label declaration, marked statement and `goto` statement must belong to the same block. Hence it is not possible to jump into or out of a procedure or function.

You can use `goto` to break out from any level of nested control structures. Never jump into a loop or other structured statement, since this can have unpredictable effects.

Use of `goto` statement is generally discouraged as practically every algorithm can be realized without it, resulting in legible structured programs. One possible application of `goto` statement is breaking out from deeply nested control structures:

```
for (...) do
  begin
    for (...) do
      begin
        ...
        if (disaster) then goto Error;
        ...
      end;
    end;
  .
  .
  .
Error: // error handling code
```

asm Statement

mikroPascal PRO for PIC 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;
```

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

Note: Compiler doesn't expect memory banks to be changed inside the assembly code. If the user wants to do this, then he must restore the previous bank selection.

DIRECTIVES

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

The following directives are available for use:

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

COMPILER DIRECTIVES

mikroPascal PRO for PIC treats comments beginning with a “\$” immediately following an opening brace as a compiler directive; for example, { \$ELSE} . The compiler directives are not case sensitive.

You can use a 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 have 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.

Note: Pascal does not support macros; directives \$DEFINE and \$UNDEFINE do not create/destroy macros. They only provide flags for directive \$IFDEF to check against.

Directives \$IFDEF, \$IFNDEF, \$ELSE

Conditional compilation is carried out by the \$IFDEF and \$IFNDEF directives. \$IFDEF tests whether a flag is currently defined, and \$IFNDEF if the flag is not defined, i.e. whether a previous \$DEFINE directive has been processed for that flag and is still in force.

Directives \$IFDEF and \$IFNDEF are terminated with the \$ENDIF directive, and can have an optional \$ELSE clause:

```
{ $IFDEF flag}
    <block of code>
{ $ELSE}
    <alternate block of code>
{ $ENDIF}
```

First, `$IFDEF` checks if flag is defined by means of `$DEFINE`. If so, only `<block of code>` will be compiled. Otherwise, `<alternate block of code>` 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`.

Unlike `$IFDEF`, `$IFNDEF` checks if flag is not defined by means of `$DEFINE`, thus producing the opposite results.

```
// Uncomment the appropriate flag for your application:
//{{ $DEFINE resolution10}
//{{ $DEFINE resolution12}

{$IFDEF resolution10}
    // <code specific to 10-bit resolution>
{$ELSE}
    {$IFDEF resolution12}
        // <code specific to 12-bit resolution>
    {$ELSE}
        // <default code>
    {$ENDIF}
{$ENDIF}
```

Include Directive `$I`

The `$I` parameter directive instructs mikroPascal PRO for PIC to include the named text file in the compilation. In effect, the file is inserted in the compiled text right after the `{ $I filename }` directive. If filename does not specify a directory path, then, in addition to searching for the file in the same directory as the current unit, mikroPascal PRO for PIC will search for file in order specified by the search paths.

To specify a filename that includes a space, surround the file name with quotation marks: `{ $I "My file" }`.

There is one restriction to the use of include files: An include file can't be specified in the middle of a statement part. In fact, all statements between the begin and end of a statement part must exist in the same source file.

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 16F887}    // If 16F887 MCU is selected
{ $IFNDEF 18F4550} // IF 18F4550 MCU is selected
```

See also predefined project level defines.

LINKER DIRECTIVES

mikroPascal PRO for PIC uses internal algorithm to distribute objects within memory. If you need to have a variable or a 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`

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

Directive `absolute` is appended to the declaration of a variable:

```
var x : word; absolute $32;
// Variable x will occupy 1 word (16 bits) at address $32

    y : longint; absolute $34;
// Variable y will occupy 2 words at addresses $34 and $36
```

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

```
var i : word; absolute $42;
// Variable i will occupy 1 word at address $42;

    jj : longint; absolute $40;
// Variable will occupy 2 words at $40 and $42; thus,
// changing i changes jj at the same time and vice versa
```

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

Directive `org`

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

```
procedure proc(par : byte); org $200;
begin
// Procedure will start at address $200;
...
end;
```

org directive can be used with `main` routine too. For example:

```
program Led_Blinking;

procedure some_proc();
begin
...
end;

org 0x800; // main procedure starts at 0x800
begin
  TRISB := 0;

  while TRUE do
  begin
    PORTB := 0x00;
    Delay_ms(500);
    PORTB := 0xFF;
    Delay_ms(500);
  end;
end.
```

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

Directive `orgall`

Use the `orgall` directive to specify the address above which all routines, constants will be placed. Example:

```
begin
  orgall(0x200) // All the routines, constants in main program will
  be above the address 0x200

  ...

end.
```

CHAPTER

7

mikroPascal PRO for PIC Libraries

mikroPascal PRO for PIC provides a set of libraries which simplify the initialization and use of PIC compliant MCUs and their modules:

Use Library manager to include mikroPascal PRO for PIC Libraries in you project.

Hardware PIC-specific Libraries

- ADC Library
- CAN Library
- CANSPI Library
- Compact Flash Library
- EEPROM Library
- Ethernet PIC18FxxJ60 Library
- Flash Memory Library
- Graphic Lcd Library
- I₂C Library
- Keypad Library
- Lcd Library
- Manchester Code Library
- Multi Media Card library
- OneWire Library
- Port Expander Library
- PS/2 Library
- PWM 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
- USB HID Library

Miscellaneous Libraries

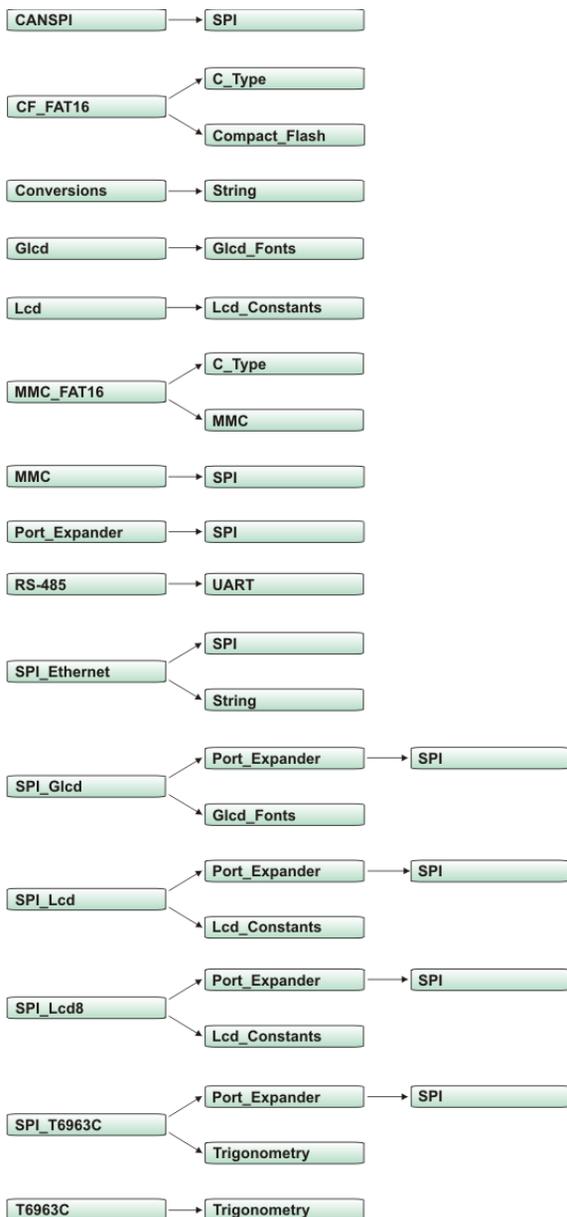
- Button Library
- Conversions Library
- Math Library
- String Library
- Time Library
- Trigonometry Library

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,
PIC Libraries

HARDWARE LIBRARIES

- ADC Library
- CAN Library
- CANSPI Library
- Compact Flash Library
- EEPROM Library
- Ethernet PIC18FxxJ60 Library
- Flash Memory Library
- Graphic Lcd Library
- I₂C Library
- Keypad Library
- Lcd Library
- Manchester Code Library
- Multi Media Card library
- OneWire Library
- Port Expander Library
- PS/2 Library
- PWM 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
- USB HID Library

ADC LIBRARY

ADC (Analog to Digital Converter) module is available with a number of PIC MCUs. Library function `ADC_Read` is included to provide you comfortable work with the module.

ADC_Read

Prototype	<code>function ADC_Read(channel : byte) : word;</code>
Returns	Initializes PIC's internal ADC module to work with RC clock. Clock determines the time period necessary for performing AD conversion (min 12TAD). Parameter channel represents the channel from which the analog value is to be acquired. Refer to the appropriate datasheet for channel-to-pin mapping.
Description	Initializes PIC 's internal ADC module to work with XTAL frequency prescaled by 128. Clock determines the time period necessary for performing A/D conversion. Parameter channel represents the channel from which the analog value is to be acquired. Refer to the appropriate datasheet for channel-to-pin mapping.
Requires	Nothing.
Example	<pre>var tmp : word; ... tmp := ADC_Read(2); // Read analog value from channel 2</pre>

Library Example

This example code reads analog value from channel 2 and displays it on PORTB and PORTC.

```

program ADC_on_LEDs;

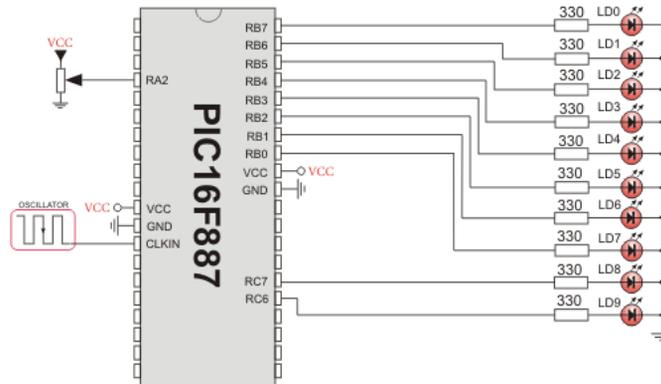
var temp_res : word;

begin
    ANSEL := 0x04;           // Configure AN2 pin as analog
    TRISA := 0xFF;          // PORTA is input
    ANSELH := 0;           // Configure other AN pins as digital I/O
    TRISC := 0x3F;         // Pins RC7, RC6 are outputs
    TRISB := 0;           // PORTB is output

    while (TRUE) do
        begin
            temp_res := Adc_Read(2); // Get 10-bit results of AD conversion
            PORTB := temp_res;        // Send lower 8 bits to PORTB
            PORTC := word(temp_res shr 2); // Send 2 most significant bits to RC7, RC6
        end;
    end;

```

HW Connection



ADC HW connection

CAN LIBRARY

mikroPascal PRO for PIC provides a library (driver) for working with the CAN module.

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

Data transfer rates vary from up to 1 Mbit/s at network lengths below 40m to 250 Kbit/s at 250m cables, and can go even lower at greater network distances, down to 200Kbit/s, which is the minimum bitrate defined by the standard. Cables used are shielded twisted pairs, and maximum cable length is 1000m.

CAN supports two message formats:

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

Note: Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.

Library Routines

- CANSetOperationMode
- CANGetOperationMode
- CANInitialize
- CANSetBaudRate
- CANSetMask
- CANSetFilter
- CANRead
- CANWrite

Following routines are for the internal use by compiler only:

- RegsToCANID
- CANIDToRegs

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

CANSetOperationMode

Prototype	<code>procedure CANSetOperationMode(mode : byte, wait_flag : byte);</code>
Returns	Nothing.
Description	<p>Sets CAN to requested mode, i.e. copies mode to CANSTAT. Parameter mode needs to be one of <code>CAN_OP_MODE</code> constants (see CAN constants).</p> <p>Parameter <code>wait_flag</code> needs to be either 0 or \$FF:</p> <ul style="list-style-type: none"> - If set to \$FF, this is a blocking call – the function won't "return" until the requested mode is set. - If 0, this is a non-blocking call. It does not verify if CAN module is switched to requested mode or not. Caller must use <code>CANGetOperationMode</code> to verify correct operation mode before performing mode specific operation.
Requires	Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.
Example	<code>CANSetOperationMode(_CAN_MODE_CONFIG, \$FF);</code>

CANGetOperationMode

Prototype	<code>function CANGetOperationMode : byte;</code>
Returns	Current opmode.
Description	Function returns current operational mode of CAN module.
Requires	Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.
Example	<code>if (CANGetOperationMode = _CAN_MODE_NORMAL) then ...</code>

CANInitialize

Prototype	<code>procedure CANInitialize(SJW, BRP, PHSEG1, PHSEG2, PROPSEG, CAN_CONFIG_FLAGS : byte);</code>
Returns	Nothing.
Description	<p>Initializes CAN. All pending transmissions are aborted. Sets all mask registers to 0 to allow all messages.</p> <p>Filter registers are set according to flag value:</p> <pre> if (_CAN_CONFIG_FLAGS and _CAN_CONFIG_VALID_XTD_MSG) <> 0 // Set all filters to XTD_MSG else if (config and CONFIG_VALID_STD_MSG) <> 0 // Set all filters to STD_MSG else // Set half of the filters to STD, and the rest to XTD_MSG. </pre> <p>Parameters:</p> <ul style="list-style-type: none"> - SJW as defined in 18XXX8 datasheet (1–4) - BRP as defined in 18XXX8 datasheet (1–64) - PHSEG1 as defined in 18XXX8 datasheet (1–8) - PHSEG2 as defined in 18XXX8 datasheet (1–8) - PROPSEG as defined in 18XXX8 datasheet (1–8) - CAN_CONFIG_FLAGS is formed from predefined constants (see CAN constants)
Requires	<p>CAN must be in Config mode; otherwise the function will be ignored.</p> <p>Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.</p>
Example	<pre> init := _CAN_CONFIG_SAMPLE_THRICE and _CAN_CONFIG_PHSEG2_PRG_ON and _CAN_CONFIG_STD_MSG and _CAN_CONFIG_DBL_BUFFER_ON and _CAN_CONFIG_VALID_XTD_MSG and _CAN_CONFIG_LINE_FILTER_OFF; ... CANInitialize(1, 1, 3, 3, 1, init); // initialize CAN </pre>

CANSetBaudRate

Prototype	<code>procedure CANSetBaudRate(SJW, BRP, PHSEG1, PHSEG2, PROPSEG, CAN_CONFIG_FLAGS : byte);</code>
Returns	Nothing.
Description	<p>Sets CAN baud rate. Due to complexity of CAN protocol, you cannot simply force a bps value. Instead, use this function when CAN is in Config mode. Refer to datasheet for details.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>SJW</code> as defined in 18XXX8 datasheet (1–4) - <code>BRP</code> as defined in 18XXX8 datasheet (1–64) - <code>PHSEG1</code> as defined in 18XXX8 datasheet (1–8) - <code>PHSEG2</code> as defined in 18XXX8 datasheet (1–8) - <code>PROPSEG</code> as defined in 18XXX8 datasheet (1–8) - <code>CAN_CONFIG_FLAGS</code> is formed from predefined constants (see CAN constants)
Requires	<p>CAN must be in Config mode; otherwise the function will be ignored.</p> <p>Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.</p>
Example	<pre>init := _CAN_CONFIG_SAMPLE_THRICE and _CAN_CONFIG_PHSEG2_PRG_ON and _CAN_CONFIG_STD_MSG and _CAN_CONFIG_DBL_BUFFER_ON and _CAN_CONFIG_VALID_XTD_MSG and _CAN_CONFIG_LINE_FILTER_OFF; ... CANSetBaudRate(1, 1, 3, 3, 1, init);</pre>

CANSetMask

Prototype	<code>procedure CANSetMask(CAN_MASK : byte; value : longint; CAN_CONFIG_FLAGS : byte);</code>
Returns	Nothing.
Description	<p>Function sets mask for advanced filtering of messages. Given value is bit adjusted to appropriate buffer mask registers.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>CAN_MASK</code> is one of predefined constant values (see CAN constants) - <code>value</code> is the mask register value - <code>CAN_CONFIG_FLAGS</code> selects type of message to filter, either <code>_CAN_CONFIG_XTD_MSG</code> or <code>_CAN_CONFIG_STD_MSG</code>
Requires	<p>CAN must be in Config mode; otherwise the function will be ignored.</p> <p>Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.</p>
Example	<pre>// Set all mask bits to 1, i.e. all filtered bits are relevant: CANSetMask(_CAN_MASK_B1, -1, _CAN_CONFIG_XTD_MSG); // Note that -1 is just a cheaper way to write \$FFFFFFFF. // Complement will do the trick and fill it up with ones.</pre>

CANSetFilter

Prototype	<code>procedure CANSetFilter(CAN_FILTER : byte; value : longint; CAN_CONFIG_FLAGS : byte);</code>
Returns	Nothing.
Description	<p>Function sets message filter. Given value is bit adjusted to appropriate buffer mask registers.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>CAN_FILTER</code> is one of predefined constant values (see CAN constants) - <code>value</code> is the filter register value - <code>CAN_CONFIG_FLAGS</code> selects type of message to filter, either <code>_CAN_CONFIG_XTD_MSG</code> or <code>_CAN_CONFIG_STD_MSG</code>
Requires	<p>CAN must be in Config mode; otherwise the function will be ignored.</p> <p>Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.</p>
Example	<pre>// Set id of filter B1_F1 to 3: CANSetFilter(_CAN_FILTER_B1_F1, 3, _CAN_CONFIG_XTD_MSG);</pre>

CANRead

Prototype	<code>function CANRead(var id : longint; var data : array[8] of byte; var datalen: byte; var CAN_RX_MSG_FLAGS : byte) : byte;</code>
Returns	Message from receive buffer or zero if no message found.
Description	<p>Function reads message from receive buffer. If at least one full receive buffer is found, it is extracted and returned. If none found, function returns zero.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>id</code> is message identifier - <code>data</code> is an array of bytes up to 8 bytes in length - <code>datalen</code> is data length, from 1–8. - <code>CAN_RX_MSG_FLAGS</code> is value formed from constants (see CAN constants)
Requires	<p>CAN must be in mode in which receiving is possible.</p> <p>Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.</p>
Example	<pre>var rcv, rx, len : byte; data : array[8] of byte; id : longint; // ... rx := 0; // ... rcv := CANRead(id, data, len, rx);</pre>

CANWrite

Prototype	<code>function CANWrite(id : longint; var data : array[8] of byte; datalen, CAN_TX_MSG_FLAGS : byte) : byte;</code>
Returns	Returns zero if message cannot be queued (buffer full).
Description	<p>If at least one empty transmit buffer is found, function sends message on queue for transmission. If buffer is full, function returns 0.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>id</code> is CAN message identifier. Only 11 or 29 bits may be used depending on message type (standard or extended) - <code>data</code> is array of bytes up to 8 bytes in length - <code>datalen</code> is data length from 1–8 - <code>CAN_TX_MSG_FLAGS</code> is value formed from constants (see CAN constants)
Requires	<p>CAN must be in Normal mode.</p> <p>Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.</p>
Example	<pre>var tx, data : byte; id : longint; // ... tx := _CAN_TX_PRIORITY_0 and _CAN_TX_XTD_FRAME; // ... CANWrite(id, data, 2, tx);</pre>

CAN Constants

There is a number of constants predefined in CAN library. To be able to use the library effectively, you need to be familiar with these. You might want to check the example at the end of the chapter.

CAN_OP_MODE

`CAN_OP_MODE` constants define CAN operation mode. Function `CANSetOperationMode` expects one of these as its argument:

```
const _CAN_MODE_BITS      = $E0; // Use it to access mode bits
const _CAN_MODE_NORMAL   = 0;
const _CAN_MODE_SLEEP    = $20;
const _CAN_MODE_LOOP     = $40;
const _CAN_MODE_LISTEN   = $60;
const _CAN_MODE_CONFIG   = $80;
```

CAN_CONFIG_FLAGS

`CAN_CONFIG_FLAGS` constants define flags related to CAN module configuration. Functions `CANInitialize` and `CANSetBaudRate` expect one of these (or a bitwise combination) as their argument:

```

const _CAN_CONFIG_DEFAULT          = $FF; // 11111111

const _CAN_CONFIG_PHSEG2_PRG_BIT  = $01;
const _CAN_CONFIG_PHSEG2_PRG_ON   = $FF; // XXXXXXX1
const _CAN_CONFIG_PHSEG2_PRG_OFF  = $FE; // XXXXXXX0

const _CAN_CONFIG_LINE_FILTER_BIT = $02;
const _CAN_CONFIG_LINE_FILTER_ON  = $FF; // XXXXXX1X
const _CAN_CONFIG_LINE_FILTER_OFF = $FD; // XXXXXX0X

const _CAN_CONFIG_SAMPLE_BIT      = $04;
const _CAN_CONFIG_SAMPLE_ONCE     = $FF; // XXXXX1XX
const _CAN_CONFIG_SAMPLE_THRICE   = $FB; // XXXXX0XX

const _CAN_CONFIG_MSG_TYPE_BIT    = $08;
const _CAN_CONFIG_STD_MSG         = $FF; // XXXX1XXX
const _CAN_CONFIG_XTD_MSG         = $F7; // XXXX0XXX

const _CAN_CONFIG_DBL_BUFFER_BIT  = $10;
const _CAN_CONFIG_DBL_BUFFER_ON   = $FF; // XXX1XXXX
const _CAN_CONFIG_DBL_BUFFER_OFF  = $EF; // XXX0XXXX

const _CAN_CONFIG_MSG_BITS        = $60;
const _CAN_CONFIG_ALL_MSG         = $FF; // X11XXXXX
const _CAN_CONFIG_VALID_XTD_MSG   = $DF; // X10XXXXX
const _CAN_CONFIG_VALID_STD_MSG   = $BF; // X01XXXXX
const _CAN_CONFIG_ALL_VALID_MSG   = $9F; // X00XXXXX

```

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

```

init := _CAN_CONFIG_SAMPLE_THRICE and
        _CAN_CONFIG_PHSEG2_PRG_ON and
        _CAN_CONFIG_STD_MSG and
        _CAN_CONFIG_DBL_BUFFER_ON and
        _CAN_CONFIG_VALID_XTD_MSG and
        _CAN_CONFIG_LINE_FILTER_OFF;
...
CANInitialize(1, 1, 3, 3, 1, init); // Initialize CAN

```

CAN_TX_MSG_FLAGS

`CAN_TX_MSG_FLAGS` are flags related to transmission of a CAN message:

```
const _CAN_TX_PRIORITY_BITS = $03;
const _CAN_TX_PRIORITY_0   = $FC; // XXXXXX00
const _CAN_TX_PRIORITY_1   = $FD; // XXXXXX01
const _CAN_TX_PRIORITY_2   = $FE; // XXXXXX10
const _CAN_TX_PRIORITY_3   = $FF; // XXXXXX11

const _CAN_TX_FRAME_BIT    = $08;
const _CAN_TX_STD_FRAME    = $FF; // XXXXX1XX
const _CAN_TX_XTD_FRAME    = $F7; // XXXXX0XX

const _CAN_TX_RTR_BIT      = $40;
const _CAN_TX_NO_RTR_FRAME = $FF; // X1XXXXXX
const _CAN_TX_RTR_FRAME   = $BF; // X0XXXXXX
```

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

```
// form value to be used with CANSendMessage:
send_config := _CAN_TX_PRIORITY_0 and
               _CAN_TX_XTD_FRAME and
               _CAN_TX_NO_RTR_FRAME;
...
CANSendMessage(id, data, 1, send_config);
```

CAN_RX_MSG_FLAGS

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

```
const _CAN_RX_FILTER_BITS = $07; // Use it to access filter bits
const _CAN_RX_FILTER_1   = $00;
const _CAN_RX_FILTER_2   = $01;
const _CAN_RX_FILTER_3   = $02;
const _CAN_RX_FILTER_4   = $03;
const _CAN_RX_FILTER_5   = $04;
const _CAN_RX_FILTER_6   = $05;
const _CAN_RX_OVERFLOW   = $08; // Set if Overflowed; else clear
const _CAN_RX_INVALID_MSG = $10; // Set if invalid; else clear
const _CAN_RX_XTD_FRAME   = $20; // Set if XTD message; else clear
const _CAN_RX_RTR_FRAME   = $40; // Set if RTR message; else clear
const _CAN_RX_DBL_BUFFERED = $80; // Set if message was
                                   // hardware double-buffered
```

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

```
if (MsgFlag and _CAN_RX_OVERFLOW) = 0 then
...
// Receiver overflow has occurred.
// We have lost our previous message.
```

CAN_MASK

CAN_MASK constants define mask codes. Function CANSetMask expects one of these as its argument:

```
const _CAN_MASK_B1 = 0;
const _CAN_MASK_B2 = 1;
```

CAN_FILTER

CAN_FILTER constants define filter codes. Function CANSetFilter expects one of these as its argument:

```
const _CAN_FILTER_B1_F1 = 0;
const _CAN_FILTER_B1_F2 = 1;
const _CAN_FILTER_B2_F1 = 2;
const _CAN_FILTER_B2_F2 = 3;
const _CAN_FILTER_B2_F3 = 4;
const _CAN_FILTER_B2_F4 = 5;
```

Library Example

This is a simple demonstration of CAN 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 CAN node:

```
program Can_1st;

var Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags, Rx_Data_Len :
byte; // can flags
    RxTx_Data : array[8] of byte; //
CAN rx/tx data buffer
    Rx_ID : longint;
    Msg_Rcvd : byte;

const ID_1st : longint = 12111;
const ID_2nd : longint = 3; // node IDs
```

```
begin

    PORTC := 0;           // clear PORTC
    TRISC := 0;          // set PORTC as output

    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 CANWrite
                     _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 CANInit
                     _CAN_CONFIG_XTD_MSG and
                     _CAN_CONFIG_DBL_BUFFER_ON and
                     _CAN_CONFIG_VALID_XTD_MSG;

    CANInitialize(1,3,3,3,1,Can_Init_Flags); // Initialize CAN module
    CANSetOperationMode(_CAN_MODE_CONFIG,0xFF); // set
CONFIGURATION mode
    CANSetMask(_CAN_MASK_B1,-1,_CAN_CONFIG_XTD_MSG); // set
all mask1 bits to ones
    CANSetMask(_CAN_MASK_B2,-1,_CAN_CONFIG_XTD_MSG); // set
all mask2 bits to ones
    CANSetFilter(_CAN_FILTER_B2_F4,ID_2nd,_CAN_CONFIG_XTD_MSG); // set
id of filter B2_F4 to 2nd node ID

    CANSetOperationMode(_CAN_MODE_NORMAL,0xFF); // set NORMAL mode

    RxTx_Data[0] := 9; // set initial data to be sent

    CANWrite(ID_1st, RxTx_Data, 1, Can_Send_Flags); // send initial message

    while ( TRUE ) do
// endless loop
    begin
        Msg_Rcvd := CANRead(Rx_ID , RxTx_Data , Rx_Data_Len,
Can_Rcv_Flags); // receive message
        if ((Rx_ID = ID_2nd) and Msg_Rcvd) <> 0 ) then
// if message received check id
        begin
            PORTC := RxTx_Data[0];
// id correct, output data at PORTC
            Inc(RxTx_Data[0]);
// increment received data
            Delay_ms(10);
            CANWrite(ID_1st, RxTx_Data, 1, Can_Send_Flags);
// send incremented data back
        end;
    end;
end.
```

Code for the second CAN node:

```

program Can_2nd;

var Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags, Rx_Data_Len   :
byte; // can flags
    RxTx_Data   : array[ 8] of byte; // can rx/tx data buffer
    Rx_ID       : longint;
    Msg_Rcvd    : byte; // reception flag

const ID_1st : longint = 12111;
const ID_2nd : longint = 3; // node IDs

begin
    PORTC := 0; // clear PORTC
    TRISC := 0; // set PORTC as output

    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 CANWrite
                     _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 CANInit
                     _CAN_CONFIG_XTD_MSG and
                     _CAN_CONFIG_DBL_BUFFER_ON and
                     _CAN_CONFIG_VALID_XTD_MSG and
                     _CAN_CONFIG_LINE_FILTER_OFF;

    CANInitialize(1,3,3,3,1,Can_Init_Flags); // initialize exter-
    nal CAN module
    CANSetOperationMode(_CAN_MODE_CONFIG,0xFF); // set
    CONFIGURATION mode
    CANSetMask(_CAN_MASK_B1,-1,_CAN_CONFIG_XTD_MSG); // set
    all mask1 bits to ones
    CANSetMask(_CAN_MASK_B2,-1,_CAN_CONFIG_XTD_MSG); // set
    all mask2 bits to ones
    CANSetFilter(_CAN_FILTER_B2_F3,ID_1st,_CAN_CONFIG_XTD_MSG); // set
    id of filter B2_F3 to 1st node ID

    CANSetOperationMode(_CAN_MODE_NORMAL,0xFF); // set NORMAL mode

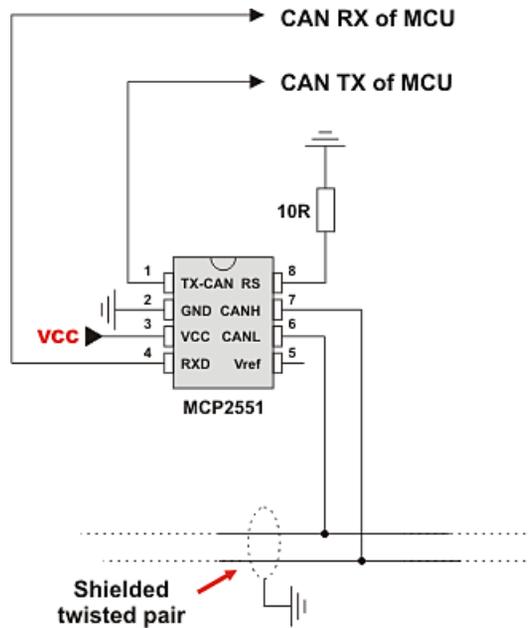
```

```

while (TRUE) do
// endless loop
  begin
    Msg_Rcvd := CANRead(Rx_ID , RxTx_Data , Rx_Data_Len,
Can_Rcv_Flags); // receive message
    if ((Rx_ID = ID_1st) and Msg_Rcvd) <> 0 ) then
// if message received check id
      begin
        PORTC := RxTx_Data[ 0 ] ;
// id correct, output data at PORTC
        inc (RxTx_Data[ 0 ] ) ;
// increment received data
        CANWrite(ID_2nd, RxTx_Data, 1, Can_Send_Flags);
// send incremented data back
      end;
    end;
  end.

```

HW Connection



Example of interfacing CAN transceiver with MCU and bus

CANSPI LIBRARY

The SPI module is available with a number of the PIC compliant MCUs. The mikroPascal PRO for PIC 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.
- The library uses the SPI module for communication. User must initialize SPI module before using the SPI Graphic Lcd Library.
For MCUs with two SPI modules it is possible to initialize both of them and then switch by using the SPI_Set_Active() routine.
- 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>var CanSpi_CS : sbit; sfr; external;</code>	Chip Select line.	<code>var CanSpi_CS : sbit at RC0_bit;</code>
<code>var CanSpi_Rst : sbit; sfr; external;</code>	Reset line.	<code>var CanSpi_Rst : sbit at RC2_bit;</code>
<code>var CanSpi_CS_Direction : sbit; sfr; external;</code>	Direction of the Chip Select pin.	<code>var CanSpi_CS_Direction : sbit at TRISC0_bit;</code>
<code>var CanSpi_Rst_Direction : sbit; sfr; external;</code>	Direction of the Reset pin.	<code>var CanSpi_Rst_Direction : sbit at TRISC2_bit;</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 functions.

CANSPISetOperationMode

Prototype	<code>procedure CANSPISetOperationMode(mode: byte; WAIT: 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 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 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>function CANSPIGetOperationMode(): byte;</code>
Returns	Current operation mode.
Description	The 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	The CANSPI routines are supported only by MCUs with the SPI module. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.
Example	<pre>// check whether the CANSPI module is in Normal mode and if it is do something. if (CANSPIGetOperationMode() = _CANSPI_MODE_NORMAL) then begin ... end;</pre>

CANSPIInitialize

Prototype	<code>procedure CANSPIInitialize(SJW: byte; BRP: byte; PHSEG1: byte; PHSEG2: byte; PROPSEG: byte; CANSPI_CONFIG_FLAGS: 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>CANSPI_CONFIG_FLAGS</code> value <p><code>SAM</code>, <code>SEG2PHTS</code>, <code>WAKFIL</code> and <code>DBEN</code> bits are set according to <code>CANSPI_CONFIG_FLAGS</code> value.</p>

Description	<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>CANSPI_CONFIG_FLAGS</code> is formed from predefined constants (see CANSPI constants)
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>CanSpi_CS</code>: Chip Select line - <code>CanSpi_Rst</code>: Reset line - <code>CanSpi_CS_Bit_Direction</code>: Direction of the Chip Select pin - <code>CanSpi_Rst_Bit_Direction</code>: Direction of the Reset pin <p>must be defined before using this 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>SPI1_Init</code> and <code>SPI1_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	<pre>// CANSPI module connections var CanSpi_CS : sbit at RC0_bit; CanSpi_CS_Direction : sbit at TRISC0_bit; CanSpi_Rst : sbit at RC2_bit; CanSpi_Rst_Direction : sbit at TRISC2_bit; // End CANSPI module connections var Can_Init_Flags: byte; ... Can_Init_Flags := _CANCONFIG_SAMPLE_THRICE and // form value to be used _CANCONFIG_PHSEG2_PRG_ON and // with CANSPIInitialize _CANCONFIG_XTD_MSG and _CANCONFIG_DBL_BUFFER_ON and _CANCONFIG_VALID_XTD_MSG; ... SPI1_Init(); // initialize SPI module CANSPIInitialize(1,3,3,3,1,Can_Init_Flags); // initialize external CANSPI module</pre>

CANSPISetBaudRate

Prototype	<code>procedure CANSPISetBaudRate(SJW: byte; BRP: byte; PHSEG1: byte; PHSEG2: byte; PROPSEG: byte; CANSPI_CONFIG_FLAGS: 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 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>CANSPI_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>CANSPI_CONFIG_FLAGS</code> is formed from predefined constants (see CANSPI constants)
Requires	<p>The CANSPI module must be in Config mode, otherwise the 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 var can_config_flags: 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>procedure CANSPISetMask(CANSPI_MASK: byte; val: longint; CANSPI_CONFIG_FLAGS: 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>CANSPI_MASK</code>: CANSPI module mask number. Valid values: <code>CANSPI_MASK</code> constants (see CANSPI constants) - <code>val</code>: mask register value - <code>CANSPI_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 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>procedure CANSPISetFilter(CANSPI_FILTER: byte; val: longint; CANSPI_CONFIG_FLAGS: byte);</code>
Returns	Nothing.
Description	<p>Configures message filter. The parameter value is bit-adjusted to the appropriate filter registers.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>CANSPI_FILTER</code>: CANSPI module filter number. Valid values: <code>CANSPI_FILTER</code> constants (see CANSPI constants) - <code>val</code>: filter register value - <code>CANSPI_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 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	<code>function CANSPiRead(var id: longint; var rd_data: array[8] of byte; data_len: byte; var CANSPi_RX_MSG_FLAGS: byte): byte;</code>
Returns	- 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>CANSPi_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>CANSPi_RX_MSG_FLAGS</code>: message flags storage address
Requires	<p>The CANSPi module must be in a mode in which receiving is possible. 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>// check the CANSPi module for received messages. If any was // received do something. var msg_rcvd, rx_flags, data_len: byte; rd_data: array[8] of byte; msg_id: 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)) begin ... end;</pre>

CANSPIWrite

Prototype	<code>function CANSPIWrite(id: longint; var wr_data: array[8] of byte; data_len: byte; CANSPI_TX_MSG_FLAGS: byte): byte;</code>
Returns	- 0 if all Transmit Buffers are busy - 0xFF if at least one Transmit Buffer is available
Description	If at least one empty Transmit Buffer is found, the function sends message in the queue for transmission. Parameters: - <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>CANSPI_RX_MSG_FLAGS</code> : message flags
Requires	The CANSPI module must be in mode in which transmission is possible. See <code>CANSPISetOperationMode</code> . The CANSPI routines are supported only by MCUs with the SPI module. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.
Example	<pre>// send message extended CAN message with the appropriate ID and data var tx_flags: byte; rd_data: array[8] of byte; msg_id: longint; ... CANSPISetOperationMode(_CANSPI_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. Function `CANSPISetOperationMode` expects one of these as its 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 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; // XXXXXXX1
    _CANSPI_CONFIG_PHSEG2_PRG_OFF = 0xFE; // XXXXXXX0

    _CANSPI_CONFIG_LINE_FILTER_BIT = 0x02;
    _CANSPI_CONFIG_LINE_FILTER_ON  = 0xFF; // XXXXXXX1X
    _CANSPI_CONFIG_LINE_FILTER_OFF = 0xFD; // XXXXXXX0X

    _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_DBL_BUFFER_BIT  = 0x10;
    _CANSPI_CONFIG_DBL_BUFFER_ON   = 0xFF; // XXX1XXXX
    _CANSPI_CONFIG_DBL_BUFFER_OFF  = 0xEF; // XXX0XXXX

    _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 hardware double-buffered
```

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

```
if (MsgFlag and _CANSPI_RX_OVERFLOW <> 0) then
begin
  ...
  // Receiver overflow has occurred.
  // We have lost our previous message.
end;
```

CANSPI_MASK

The `CANSPI_MASK` constants define mask codes. 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. 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 Can_Spi_1st;

var Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags : byte;    // can
    flags
    Rx_Data_Len : byte;          // received data length in bytes
    RxTx_Data   : array[8] of byte; // can rx/tx data buffer
    Msg_Rcvd   : byte;          // reception flag
    Tx_ID, Rx_ID : longint;     // can rx and tx ID

// CANSPI module connections
var CanSpi_CS : sbit at RC0_bit;
    CanSpi_CS_Direction : sbit at TRISC0_bit;
    CanSpi_Rst : sbit at RC2_bit;
    CanSpi_Rst_Direction : sbit at TRISC2_bit;
// End CANSPI module connections

begin

    ANSEL := 0;          // Configure AN pins as digital I/O
    ANSELH := 0;
    PORTB := 0;
    TRISB := 0;

    Can_Init_Flags := 0;          //
    Can_Send_Flags := 0;         // clear flags
    Can_Rcv_Flags := 0;          //

    Can_Send_Flags := _CANSPI_TX_PRIORITY_0 and //form value to be used
                      _CANSPI_TX_XTD_FRAME and //      with CANSPIWrite
                      _CANSPI_TX_NO_RTR_FRAME;

    Can_Init_Flags := _CANSPI_CONFIG_SAMPLE_THRICE and // form
value to be used
                      _CANSPI_CONFIG_PHSEG2_PRG_ON and // with
CANSPIInit
                      _CANSPI_CONFIG_XTD_MSG and
                      _CANSPI_CONFIG_DBL_BUFFER_ON and
                      _CANSPI_CONFIG_VALID_XTD_MSG;
```

```
SPI1_Init(); //
initialize SPI module
  CANSPIInitialize(1,3,3,3,1,Can_Init_Flags); //
Initialize external CANSPI module
  CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF); //
set CONFIGURATION mode
  CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG);
// set all mask1 bits to ones
  CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG);
// set all mask2 bits to ones
  CANSPISetFilter(_CANSPI_FILTER_B2_F4,3,_CANSPI_CONFIG_XTD_MSG);
// set id of filter B1_F1 to 3

  CANSPISetOperationMode(_CANSPI_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) do
    begin // endless loop
      Msg_Rcvd := CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len,
Can_Rcv_Flags); // receive message
      if ((Rx_ID = 3) and Msg_Rcvd) then
// if message received check id
        begin
          PORTB := RxTx_Data[0];
// id correct, output data at PORTB
          Inc(RxTx_Data[0]) ;
// increment received data
          Delay_ms(10);
          CANSPIWrite(Tx_ID, RxTx_Data, 1, Can_Send_Flags);
// send incremented data back
        end;
    end;
end.
```

Code for the second CANSPI node:

```
program Can_Spi_2nd;

var Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags : byte;    // can
    flags
    Rx_Data_Len : byte;          // received data length in bytes
    RxTx_Data   : array[8] of byte; // CAN rx/tx data buffer
    Msg_Rcvd   : byte;          // reception flag
    Tx_ID, Rx_ID : longint;     // can rx and tx ID

// CANSPI module connections
var CanSpi_CS : sbit at RC0_bit;
    CanSpi_CS_Direction : sbit at TRISC0_bit;
    CanSpi_Rst : sbit at RC2_bit;
    CanSpi_Rst_Direction : sbit at TRISC2_bit;
// End CANSPI module connections

begin

    ANSEL := 0; //
    Configure AN pins as digital I/O
    ANSELH := 0;
    PORTB := 0; //
    clear PORTC
    TRISB := 0; //
    set PORTC as output

    Can_Init_Flags := 0; //
    Can_Send_Flags := 0; //
    clear flags
    Can_Rcv_Flags := 0; //

    Can_Send_Flags := _CANSPI_TX_PRIORITY_0 and //
    form value to be used //
    _CANSPI_TX_XTD_FRAME and //
with CANSPIWrite //
    _CANSPI_TX_NO_RTR_FRAME;

    Can_Init_Flags := _CANSPI_CONFIG_SAMPLE_THRICE and //
    Form value to be used //
    _CANSPI_CONFIG_PHSEG2_PRG_ON and //
with CANSPIInit //
    _CANSPI_CONFIG_XTD_MSG and
    _CANSPI_CONFIG_DBL_BUFFER_ON and
    _CANSPI_CONFIG_VALID_XTD_MSG and
    _CANSPI_CONFIG_LINE_FILTER_OFF;
```

```

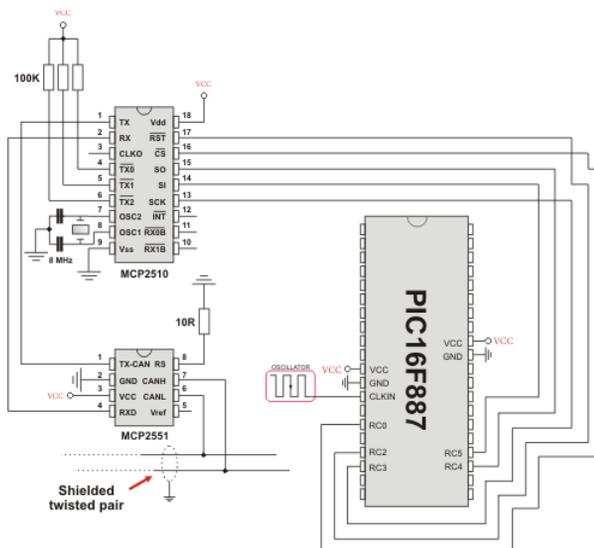
SPI1_Init(); // initialize SPI1 module
CANSPIInitialize(1,3,3,3,1,Can_Init_Flags); // initialize
external CANSPI module
CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF); // set
CONFIGURATION mode
CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG); //
set all mask1 bits to ones
CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG); //
set all mask2 bits to ones

CANSPISetFilter(_CANSPI_FILTER_B2_F3,12111,_CANSPI_CONFIG_XTD_MSG);
// set id of filter B1_F1 to 3
CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF); //
set NORMAL mode
Tx_ID := 3; // set tx ID

while (TRUE) do // endless loop
begin
Msg_Rcvd := CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len,
Can_Rcv_Flags); // receive message
if ((Rx_ID = 12111) and Msg_Rcvd) then
// if message received check id
begin
PORTB := RxTx_Data[0]; // id correct, output data at PORTB
Inc(RxTx_Data[0]); // increment received data
CANSPIWrite(Tx_ID, RxTx_Data, 1, Can_Send_Flags);
// send incremented data back
end;
end;
end.

```

HW Connection



Example of interfacing CAN transceiver MCP2510 with MCU via SPI interface

COMPACT FLASH LIBRARY

The Compact Flash Library provides routines for accessing data on Compact Flash card (abbr. CF further in text). CF cards are widely used memory elements, commonly used with digital cameras. Great capacity and excellent access time of only a few microseconds make them very attractive for the microcontroller applications.

In CF card, data is divided into sectors. One sector usually comprises 512 bytes. Routines for file handling, the Cf_Fat routines, are not performed directly but successively through 512B buffer.

Note: Routines for file handling can be used only with FAT16 file system.

Note: Library functions create and read files from the root directory only.

Note: Library functions populate both FAT1 and FAT2 tables when writing to files, but the file data is being read from the FAT1 table only; i.e. there is no recovery if the FAT1 table gets corrupted.

Note: If MMC/SD card has Master Boot Record (MBR), the library will work with the first available primary (logical) partition that has non-zero size. If MMC/SD card has Volume Boot Record (i.e. there is only one logical partition and no MBRs), the library works with entire card as a single partition. For more information on MBR, physical and logical drives, primary/secondary partitions and partition tables, please consult other resources, e.g. Wikipedia and similar.

Note: Before writing operation, make sure not to overwrite boot or FAT sector as it could make your card on PC or digital camera unreadable. Drive mapping tools, such as Winhex, can be of great assistance.

External dependencies of Compact Flash Library

The following variables must be defined in all projects using Compact Flash Library:	Description:	Example :
<code>var CF_Data_Port : byte; sfr; external;</code>	Compact Flash Data Port.	<code>var CF_Data_Port : byte at PORTD;</code>
<code>var CF_RDY : sbit; sfr; external;</code>	Ready signal line.	<code>var CF_RDY : sbit at RB7_bit;</code>
<code>var CF_WE : sbit; sfr; external;</code>	Write Enable signal line.	<code>var CF_WE : sbit at RB6_bit;</code>
<code>var CF_OE : sbit; sfr; external;</code>	Output Enable signal line.	<code>var CF_OE : sbit at RB5_bit;</code>
<code>var CF_CD1 : sbit; sfr; external;</code>	Chip Detect signal line.	<code>var CF_CD1 : sbit at RB4_bit;</code>
<code>var CF_CE1 : sbit; sfr; external;</code>	Chip Enable signal line.	<code>var CF_CE1 : sbit at RB3_bit;</code>
<code>var CF_A2 : sbit; sfr; external;</code>	Address pin 2.	<code>var CF_A2 : sbit at RB2_bit;</code>
<code>var CF_A1 : sbit; sfr; external;</code>	Address pin 1.	<code>var CF_A1 : sbit at RB1_bit;</code>
<code>var CF_A0 : sbit; sfr; external;</code>	Address pin 0.	<code>var CF_A0 : sbit at RB0_bit;</code>
<code>var CF_RDY_direction : sbit; sfr; external;</code>	Direction of the Ready pin.	<code>var CF_RDY_direction : sbit at TRISB7_bit;</code>
<code>var CF_WE_direction : sbit; sfr; external;</code>	Direction of the Write Enable pin.	<code>var CF_WE_direction : sbit at TRISB6_bit;</code>
<code>var CF_OE_direction : sbit; sfr; external;</code>	Direction of the Output Enable pin.	<code>var CF_OE_direction : sbit at TRISB5_bit;</code>
<code>var CF_CD1_direction : sbit; sfr; external;</code>	Direction of the Chip Detect pin.	<code>var CF_CD1_direction : sbit at TRISB4_bit;</code>
<code>var CF_CE1_direction : sbit; sfr; external;</code>	Direction of the Chip Enable pin.	<code>var CF_CE1_direction : sbit at TRISB3_bit;</code>
<code>var CF_A2_direction : sbit; sfr; external;</code>	Direction of the Address 2 pin.	<code>var CF_A2_direction : sbit at TRISB2_bit;</code>
<code>var CF_A1_direction : sbit; sfr; external;</code>	Direction of the Address 1 pin.	<code>var CF_A1_direction : sbit at TRISB1_bit;</code>
<code>var CF_A0_direction : sbit; sfr; external;</code>	Direction of the Address 0 pin.	<code>var CF_A0_direction : sbit at TRISB0_bit;</code>

Library Routines

- Cf_Init
- Cf_Detect
- Cf_Enable
- Cf_Disable
- Cf_Read_Init
- Cf_Read_Byte
- Cf_Write_Init
- Cf_Write_Byte
- Cf_Read_Sector
- Cf_Write_Sector

Routines for file handling:

- Cf_Fat_Init
- Cf_Fat_QuickFormat
- Cf_Fat_Assign
- Cf_Fat_Reset
- Cf_Fat_Read
- Cf_Fat_Rewrite
- Cf_Fat_Append
- Cf_Fat_Delete
- Cf_Fat_Write
- Cf_Fat_Set_File_Date
- Cf_Fat_Get_File_Date
- Cf_Fat_Get_File_Size
- Cf_Fat_Get_Swap_File

Cf_Init

Prototype	<code>procedure Cf_Init();</code>
Returns	Nothing.
Description	Initializes ports appropriately for communication with CF card.
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>CF_Data_Port</code> : Compact Flash data port - <code>CF_RDY</code> : Ready signal line - <code>CF_WE</code> : Write enable signal line - <code>CF_OE</code> : Output enable signal line - <code>CF_CD1</code> : Chip detect signal line - <code>CF_CE1</code> : Enable signal line - <code>CF_A2</code> : Address pin 2 - <code>CF_A1</code> : Address pin 1 - <code>CF_A0</code> : Address pin 0 - <code>CF_RDY_direction</code> : Direction of the Ready pin - <code>CF_WE_direction</code> : Direction of the Write enable pin - <code>CF_OE_direction</code> : Direction of the Output enable pin - <code>CF_CD1_direction</code> : Direction of the Chip detect pin - <code>CF_CE1_direction</code> : Direction of the Chip enable pin - <code>CF_A2_direction</code> : Direction of the Address 2 pin - <code>CF_A1_direction</code> : Direction of the Address 1 pin - <code>CF_A0_direction</code> : Direction of the Address 0 pin <p>must be defined before using this function.</p>
Example	<pre>// set compact flash pinout var CF_Data_Port : byte at PORTD; var CF_RDY : sbit at RB7_bit; var CF_WE : sbit at RB6_bit; var CF_OE : sbit at RB5_bit; var CF_CD1 : sbit at RB4_bit; var CF_CE1 : sbit at RB3_bit; var CF_A2 : sbit at RB2_bit; var CF_A1 : sbit at RB1_bit; var CF_A0 : sbit at RB0_bit; var CF_RDY_direction : sbit at TRISB7_bit; var CF_WE_direction : sbit at TRISB6_bit; var CF_OE_direction : sbit at TRISB5_bit; var CF_CD1_direction : sbit at TRISB4_bit; var CF_CE1_direction : sbit at TRISB3_bit; var CF_A2_direction : sbit at TRISB2_bit; var CF_A1_direction : sbit at TRISB1_bit; var CF_A0_direction : sbit at TRISB0_bit; // end_of_cf pinout // Init CF begin Cf_Init(); end;</pre>

Cf_Detect

Prototype	<code>function Cf_Detect() : byte;</code>
Returns	- 1 - if CF card was detected - 0 - otherwise
Description	Checks for presence of CF card by reading the chip <code>detect</code> pin.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See <code>Cf_Init</code> .
Example	<pre>// Wait until CF card is inserted: while (Cf_Detect() = 0) do nop;</pre>

Cf_Enable

Prototype	<code>procedure Cf_Enable();</code>
Returns	Nothing.
Description	Enables the device. Routine needs to be called only if you have disabled the device by means of the <code>Cf_Disable</code> routine. These two routines in conjunction allow you to free/occupy data line when working with multiple devices.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See <code>Cf_Init</code> .
Example	<pre>// enable compact flash Cf_Enable();</pre>

Cf_Disable

Prototype	<code>procedure Cf_Disable();</code>
Returns	Nothing.
Description	Routine disables the device and frees the data lines for other devices. To enable the device again, call <code>Cf_Enable</code> . These two routines in conjunction allow you to free/occupy data line when working with multiple devices.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See <code>Cf_Init</code> .
Example	<pre>// disable compact flash Cf_Disable();</pre>

Cf_Read_Init

Prototype	<code>procedure Cf_Read_Init(address : dword; sector_count : byte);</code>
Returns	Nothing.
Description	<p>Initializes CF card for reading.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>address</code>: the first sector to be prepared for reading operation. - <code>sector_count</code>: number of sectors to be prepared for reading operation.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.
Example	<pre>// initialize compact flash for reading from sector 590 Cf_Read_Init(590, 1);</pre>

Cf_Read_Byte

Prototype	<code>function CF_Read_Byte() : byte;</code>
Returns	<p>Returns a byte read from Compact Flash sector buffer.</p> <p>Note: Higher byte of the <code>unsigned</code> return value is cleared.</p>
Description	Reads one byte from Compact Flash sector buffer location currently pointed to by internal read pointers. These pointers will be autoincremented upon reading.
Requires	<p>The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.</p> <p>CF card must be initialized for reading operation. See Cf_Read_Init.</p>
Example	<pre>// Read a byte from compact flash: var data : byte; ... data := Cf_Read_Byte();</pre>

Cf_Write_Init

Prototype	<code>procedure Cf_Write_Init(address : dword; sectcnt : byte);</code>
Returns	Nothing.
Description	<p>Initializes CF card for writing.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>address</code>: the first sector to be prepared for writing operation. - <code>sectcnt</code>: number of sectors to be prepared for writing operation.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.
Example	<pre>// initialize compact flash for writing to sector 590 Cf_Write_Init(590, 1);</pre>

Cf_Write_Byte

Prototype	<code>procedure Cf_Write_Byte(data_ : byte);</code>
Returns	Nothing.
Description	<p>Writes a byte to Compact Flash sector buffer location currently pointed to by writing pointers. These pointers will be autoincremented upon reading. When sector buffer is full, its content will be transferred to appropriate flash memory sector.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>data_</code>: byte to be written.
Requires	<p>The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.</p> <p>CF card must be initialized for writing operation. See Cf_Write_Init.</p>
Example	<pre>var data_ : byte; ... data := 0xAA; Cf_Write_Byte(data);</pre>

Cf_Read_Sector

Prototype	<code>procedure Cf_Read_Sector(sector_number : dword; var buffer : array[512] of byte);</code>
Returns	Nothing.
Description	<p>Reads one sector (512 bytes). Read data is stored into buffer provided by the <code>buffer</code> parameter.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>sector_number</code>: sector to be read. - <code>buffer</code>: data buffer of at least 512 bytes in length.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See <code>Cf_Init</code> .
Example	<pre>// read sector 22 var data : array[512] of byte; ... Cf_Read_Sector(22, data);</pre>

Cf_Write_Sector

Prototype	<code>procedure Cf_Write_Sector(sector_number : dword; var buffer : array[512] of byte);</code>
Returns	Nothing.
Description	<p>Writes 512 bytes of data provided by the buffer parameter to one CF sector.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>sector_number</code>: sector to be written to. - <code>buffer</code>: data buffer of 512 bytes in length.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See <code>Cf_Init</code> .
Example	<pre>// write to sector 22 var data : array[512] of byte; ... Cf_Write_Sector(22, data);</pre>

Cf_Fat_Init

Prototype	<code>function Cf_Fat_Init(): byte;</code>
Returns	- 0 - if CF card was detected and successfully initialized - 1 - if FAT16 boot sector was not found - 255 - if card was not detected
Description	Initializes CF card, reads CF FAT16 boot sector and extracts data needed by the library.
Requires	Nothing.
Example	<pre>//init the FAT library if (Cf_Fat_Init() = 0) then begin ... end</pre>

Cf_Fat_QuickFormat

Prototype	<code>function Cf_Fat_QuickFormat(var cf_fat_label : string[11]) : byte;</code>
Returns	- 0 - if CF card was detected, successfully formatted and initialized - 1 - if FAT16 format was unseccessful - 255 - if card was not detected
Description	<p>Formats to FAT16 and initializes CF card.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>cf_fat_label</code>: volume label (11 characters in length). If less than 11 characters are provided, the label will be padded with spaces. If an empty string is passed, the volume will not be labeled. <p>Note: This routine can be used instead or in conjunction with the Cf_Fat_Init routine.</p> <p>Note: If CF card already contains a valid boot sector, it will remain unchanged (except volume label field) and only FAT and ROOT tables will be erased. Also, the new volume label will be set.</p>
Requires	Nothing.
Example	<pre>// format and initialize the FAT library if (Cf_Fat_QuickFormat('mikroE') = 0) then begin ... end;</pre>

Cf_Fat_Assign

Prototype	<code>function Cf_Fat_Assign(var filename: array[12] of char; file_cre_attr: byte): byte;</code>																												
Returns	<ul style="list-style-type: none"> - 0 if file does not exist and no new file is created. - 1 if file already exists or file does not exist but a new file is created. 																												
Description	<p>Assigns file for file operations (read, write, delete...). All subsequent file operations will be applied to the assigned file.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>filename</code>: name of the file that should be assigned for file operations. The file name should be in DOS 8.3 (file_name.extension) format. The file name and extension will be automatically padded with spaces by the library if they have less than length required (i.e. "mikro.tx" -> "mikro .tx "), so the user does not have to take care of that. The file name and extension are case insensitive. The library will convert them to the proper case automatically, so the user does not have to take care of that. Also, in order to keep backward compatibility with the first version of this library, file names can be entered as UPPERCASE string of 11 bytes in length with no dot character between the file name and extension (i.e. "MIKROELETXT" -> MIKROELE.TXT). In this case the last 3 characters of the string are considered to be file extension. - <code>file_cre_attr</code>: file creation and attributs flags. Each bit corresponds to the appropriate file attribut: <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">Bit</th> <th style="width: 15%;">Mask</th> <th style="width: 75%;">Description</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0x01</td> <td>Read Only</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0x02</td> <td>Hidden</td> </tr> <tr> <td style="text-align: center;">2</td> <td style="text-align: center;">0x04</td> <td>System</td> </tr> <tr> <td style="text-align: center;">3</td> <td style="text-align: center;">0x08</td> <td>Volume Label</td> </tr> <tr> <td style="text-align: center;">4</td> <td style="text-align: center;">0x10</td> <td>Subdirectory</td> </tr> <tr> <td style="text-align: center;">5</td> <td style="text-align: center;">0x20</td> <td>Archive</td> </tr> <tr> <td style="text-align: center;">6</td> <td style="text-align: center;">0x40</td> <td>Device (internal use only, never found on disk)</td> </tr> <tr> <td style="text-align: center;">7</td> <td style="text-align: center;">0x80</td> <td>File creation flag. If the file does not exist and this flag is set, a new file with specified name will be created.</td> </tr> </tbody> </table> <p>Note: Long File Names (LFN) are not supported.</p>		Bit	Mask	Description	0	0x01	Read Only	1	0x02	Hidden	2	0x04	System	3	0x08	Volume Label	4	0x10	Subdirectory	5	0x20	Archive	6	0x40	Device (internal use only, never found on disk)	7	0x80	File creation flag. If the file does not exist and this flag is set, a new file with specified name will be created.
Bit	Mask	Description																											
0	0x01	Read Only																											
1	0x02	Hidden																											
2	0x04	System																											
3	0x08	Volume Label																											
4	0x10	Subdirectory																											
5	0x20	Archive																											
6	0x40	Device (internal use only, never found on disk)																											
7	0x80	File creation flag. If the file does not exist and this flag is set, a new file with specified name will be created.																											
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init.																												
Example	<code>// create file with archive attribut if it does not already exist Cf_Fat_Assign('MIKRO007.TXT', 0xA0);</code>																												

Cf_Fat_Reset

Prototype	<code>procedure Cf_Fat_Reset(var size: dword);</code>
Returns	Nothing.
Description	<p>Opens currently assigned file for reading.</p> <p>Parameters :</p> <p>- <code>size</code>: buffer to store file size to. After file has been open for reading its size is returned through this parameter.</p>
Requires	<p>CF card and CF library must be initialized for file operations. See Cf_Fat_Init.</p> <p>File must be previously assigned. See Cf_Fat_Assign.</p>
Example	<pre>var size : dword; ... Cf_Fat_Reset(size);</pre>

Cf_Fat_Read

Prototype	<code>procedure Cf_Fat_Read(var bdata: byte);</code>
Returns	Nothing.
Description	<p>Reads a byte from currently assigned file opened for reading. Upon function execution file pointers will be set to the next character in the file.</p> <p>Parameters :</p> <p>- <code>bdata</code>: buffer to store read byte to. Upon this function execution read byte is returned through this parameter.</p>
Requires	<p>CF card and CF library must be initialized for file operations. See Cf_Fat_Init.</p> <p>File must be previously assigned. See Cf_Fat_Assign.</p> <p>File must be open for reading. See Cf_Fat_Reset.</p>
Example	<pre>var character : byte; ... Cf_Fat_Read(character);</pre>

Cf_Fat_Rewrite

Prototype	<code>procedure Cf_Fat_Rewrite();</code>
Returns	Nothing.
Description	Opens currently assigned file for writing. If the file is not empty its content will be erased.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. The file must be previously assigned. See Cf_Fat_Assign.
Example	<pre>// open file for writing Cf_Fat_Rewrite();</pre>

Cf_Fat_Append

Prototype	<code>procedure Cf_Fat_Append();</code>
Returns	Nothing.
Description	Opens currently assigned file for appending. Upon this function execution file pointers will be positioned after the last byte in the file, so any subsequent file writing operation will start from there.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign.
Example	<pre>// open file for appending Cf_Fat_Append();</pre>

Cf_Fat_Delete

Prototype	<code>procedure Cf_Fat_Delete();</code>
Returns	Nothing.
Description	Deletes currently assigned file from CF card.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign.
Example	<pre>// delete current file Cf_Fat_Delete();</pre>

Cf_Fat_Write

Prototype	<code>procedure Cf_Fat_Write(var fdata: array[512] of byte; data_len: word);</code>
Returns	Nothing.
Description	Writes requested number of bytes to currently assigned file opened for writing. Parameters : - <code>fdata</code> : data to be written. - <code>data_len</code> : number of bytes to be written.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign. File must be open for writing. See Cf_Fat_Rewrite or Cf_Fat_Append.
Example	<code>var file_contents : array[42] of byte; ... Cf_Fat_Write(file_contents, 42); // write data to the assigned file</code>

Cf_Fat_Set_File_Date

Prototype	<code>procedure Cf_Fat_Set_File_Date(year: word; month: byte; day: byte; hours: byte; mins: byte; seconds: byte);</code>
Returns	Nothing.
Description	Sets the date/time stamp. Any subsequent file writing operation will write this stamp to currently assigned file's time/date attributes. Parameters : - <code>year</code> : year attribute. Valid values: 1980-2107 - <code>month</code> : month attribute. Valid values: 1-12 - <code>day</code> : day attribute. Valid values: 1-31 - <code>hours</code> : hours attribute. Valid values: 0-23 - <code>mins</code> : minutes attribute. Valid values: 0-59 - <code>seconds</code> : seconds attribute. Valid values: 0-59
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign. File must be open for writing. See Cf_Fat_Rewrite or Cf_Fat_Append.
Example	<code>Cf_Fat_Set_File_Date(2005, 9, 30, 17, 41, 0);</code>

Cf_Fat_Get_File_Date

Prototype	<code>procedure Cf_Fat_Get_File_Date(var year: word; var month: byte; var day: byte; var hours: byte; var mins: byte);</code>
Returns	Nothing.
Description	<p>Reads time/date attributes of currently assigned file.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>year</code>: buffer to store year attribute to. Upon function execution year attribute is returned through this parameter. - <code>month</code>: buffer to store month attribute to. Upon function execution month attribute is returned through this parameter. - <code>day</code>: buffer to store day attribute to. Upon function execution day attribute is returned through this parameter. - <code>hours</code>: buffer to store hours attribute to. Upon function execution hours attribute is returned through this parameter. - <code>mins</code>: buffer to store minutes attribute to. Upon function execution minutes attribute is returned through this parameter.
Requires	<p>CF card and CF library must be initialized for file operations. See <code>Cf_Fat_Init</code>.</p> <p>File must be previously assigned. See <code>Cf_Fat_Assign</code>.</p>
Example	<pre>var year : word; month, day, hours, mins : byte; ... Cf_Fat_Get_File_Date(year, month, day, hours, mins);</pre>

Cf_Fat_Get_File_Size

Prototype	<code>function Cf_Fat_Get_File_Size(): dword;</code>
Returns	Size of the currently assigned file in bytes.
Description	This function reads size of currently assigned file in bytes.
Requires	<p>CF card and CF library must be initialized for file operations. See <code>Cf_Fat_Init</code>.</p> <p>File must be previously assigned. See <code>Cf_Fat_Assign</code>.</p>
Example	<pre>var my_file_size : dword; ... my_file_size := Cf_Fat_Get_File_Size();</pre>

Cf_Fat_Get_Swap_File

Prototype	<code>function Cf_Fat_Get_Swap_File(sectors_cnt: longint; var filename : string[11]; file_attr : byte): dword;</code>
Returns	<ul style="list-style-type: none"> - Number of the start sector for the newly created swap file, if there was enough free space on CF card to create file of required size. - 0 - otherwise.
Description	<p>This function is used to create a swap file of predefined name and size on the CF media. If a file with specified name already exists on the media, search for consecutive sectors will ignore sectors occupied by this file. Therefore, it is recommended to erase such file if it exists before calling this function. If it is not erased and there is still enough space for a new swap file, this function will delete it after allocating new memory space for a new swap file.</p> <p>The purpose of the swap file is to make reading and writing to CF media as fast as possible, by using the Cf_Read_Sector() and Cf_Write_Sector() functions directly, without potentially damaging the FAT system. The swap file can be considered as a "window" on the media where the user can freely write/read data. Its main purpose in the mikroPascal's library is to be used for fast data acquisition; when the time-critical acquisition has finished, the data can be re-written into a "normal" file, and formatted in the most suitable way.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>sectors_cnt</code>: number of consecutive sectors that user wants the swap file to have. - <code>filename</code>: name of the file that should be assigned for file operations. The file name should be in DOS 8.3 (file_name.extension) format. The file name and extension will be automatically padded with spaces by the library if they have less than length required (i.e. "mikro.tx" -> "mikro .tx "), so the user does not have to take care of that. The file name and extension are case insensitive. The library will convert them to the proper case automatically, so the user does not have to take care of that. <p>Also, in order to keep backward compatibility with the first version of this library, file names can be entered as UPPERCASE string of 11 bytes in length with no dot character between the file name and extension (i.e. "MIKROELETXT" -> MIKROELE.TXT). In this case the last 3 characters of the string are considered to be file extension.</p> <ul style="list-style-type: none"> - <code>file_attr</code>: file creation and attributes flags. Each bit corresponds to the appropriate file attribute:

Description	Bit	Mask	Description
	0	0x01	Read Only
	1	0x02	Hidden
	2	0x04	System
	3	0x08	Volume Label
	4	0x10	Subdirectory
	5	0x20	Archive
	6	0x40	Device (internal use only, never found on disk)
	7	0x80	Not used
Note: Long File Names (LFN) are not supported.			
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init.		
Example	<pre> // Try to create a swap file with archive attribute, whose size // will be at least 1000 sectors. // If it succeeds, it sends the No. of start sec- // tor over UART var size : dword; ... size := Cf_Fat_Get_Swap_File(1000, "mikroE.txt", 0x20); if (size <> 0) then begin UART1_Write(0xAA); UART1_Write(Lo(size)); UART1_Write(Hi(size)); UART1_Write(Higher(size)); UART1_Write(Highest(size)); UART1_Write(0xAA); end ... </pre>		

Library Example

The following example demonstrates various aspects of the Cf_Fat16 library : Creation of new file and writing down to it; Opening existing file and re-writing it (writing from start-of-file); Opening existing file and appending data to it (writing from end-of-file); Opening a file and reading data from it (sending it to USART terminal); Creating and modifying several files at once;

```
program Cf_Fat16_Test;

var
  // set compact flash pinout
  Cf_Data_Port : byte at PORTD;

  CF_RDY : sbit at RB7_bit;
  CF_WE  : sbit at RB6_bit;
  CF_OE  : sbit at RB5_bit;
  CF_CD1 : sbit at RB4_bit;
  CF_CE1 : sbit at RB3_bit;
  CF_A2  : sbit at RB2_bit;
  CF_A1  : sbit at RB1_bit;
  CF_A0  : sbit at RB0_bit;

  CF_RDY_direction : sbit at TRISB7_bit;
  CF_WE_direction  : sbit at TRISB6_bit;
  CF_OE_direction  : sbit at TRISB5_bit;
  CF_CD1_direction : sbit at TRISB4_bit;
  CF_CE1_direction : sbit at TRISB3_bit;
  CF_A2_direction  : sbit at TRISB2_bit;
  CF_A1_direction  : sbit at TRISB1_bit;
  CF_A0_direction  : sbit at TRISB0_bit;
  // end of cf pinout

const LINE_LEN = 39;
var
  err_txt : string[20];
  file_contents : string[LINE_LEN];

  filename : string[14]; // File names

  character : byte;
  loop, loop2 : byte;
  i, size : longint;

  Buffer : array[512] of byte;

  // UART write text and new line (carriage return + line feed)
procedure UART1_Write_Line( var uart_text : byte );
```

```
begin
    UART1_Write_Text(uart_text);
    UART1_Write(13);
    UART1_Write(10);
end;

//----- Creates new file and writes some data to it
procedure M_Create_New_File();
begin
    filename[ 7] := 'A';
    Cf_Fat_Assign(filename, 0xA0);           // Will not find file and then
create file
    Cf_Fat_Rewrite();           // To clear file and start with new data
    for loop:=1 to 90 do      // We want 5 files on the MMC card
        begin
            UART1_Write('.');
            file_contents[ 0] := loop div 10 + 48;
            file_contents[ 1] := loop mod 10 + 48;
            Cf_Fat_Write(file_contents, LINE_LEN-1); // write data to
the assigned file
        end;
    end;
end;

//----- Creates many new files and writes data to them
procedure M_Create_Multiple_Files();
begin
    for loop2 := 'B' to 'Z' do
        begin
            UART1_Write(loop2); // this line can slow down the performance
            filename[ 7] := loop2;           // set filename
            Cf_Fat_Assign(filename, 0xA0);   // find existing file
or create a new one
            Cf_Fat_Rewrite(); // To clear file and start with new data
            for loop := 1 to 44 do
                begin
                    file_contents[ 0] := loop div 10 + 48;
                    file_contents[ 1] := loop mod 10 + 48;
                    Cf_Fat_Write(file_contents, LINE_LEN-1); // write data to
the assigned file
                end;
            end;
        end;
end;

//----- Opens an existing file and rewrites it
procedure M_Open_File_Rewrite();
begin
    filename[ 7] := 'C';           // Set filename for single-file tests
    Cf_Fat_Assign(filename, 0);
    Cf_Fat_Rewrite();
    for loop := 1 to 55 do
```

```
begin
    file_contents[ 0] := byte(loop div 10 + 48);
    file_contents[ 1] := byte(loop mod 10 + 48);
    Cf_Fat_Write(file_contents, LINE_LEN-1); // write data to the
assigned file
end;
end;

//----- Opens an existing file and appends data to it
//          (and alters the date/time stamp)
procedure M_Open_File_Append();
begin
    filename[ 7] := 'B';
    Cf_Fat_Assign(filename, 0);
    Cf_Fat_Set_File_Date(2005,6,21,10,35,0);
    Cf_Fat_Append();
    file_contents := ' for mikroElektronika 2005'; // Prepare file
for append
end;

//----- Opens an existing file, reads data from it and puts
it to USART
procedure M_Open_File_Read();
begin
    filename[ 7] := 'B';
    Cf_Fat_Assign(filename, 0);
    Cf_Fat_Reset(size); // To read file, procedure returns
size of file
    while size > 0 do
        begin
            Cf_Fat_Read(character);
            UART1_Write(character); // Write data to USART
            Dec(size);
        end;
    end;

//----- Deletes a file. If file doesn't exist, it will first
be created
//          and then deleted.
procedure M_Delete_File();
begin
    filename[ 7] := 'F';
    Cf_Fat_Assign(filename, 0);
    Cf_Fat_Delete();
end;

//----- Tests whether file exists, and if so sends its cre-
ation date
//          and file size via USART
```

```
procedure M_Test_File_Exist();
var
  fsize : longint;
  year : word;
  month, day, hour, minute : byte;
  outstr : array[12] of byte;
begin
  filename[7] := 'B';          //uncomment this line to search for file
  that DOES exists
  // filename[7] := 'F';      //uncomment this line to search for
  file that DOES NOT exist
  if Cf_Fat_Assign(filename, 0) <> 0 then
    begin
      //--- file has been found - get its date
      Cf_Fat_Get_File_Date(year,month,day,hour,minute);
      WordToStr(year, outstr);
      UART1_Write_Text(outstr);
      ByteToStr(month, outstr);
      UART1_Write_Text(outstr);
      WordToStr(day, outstr);
      UART1_Write_Text(outstr);
      WordToStr(hour, outstr);
      UART1_Write_Text(outstr);
      WordToStr(minute, outstr);
      UART1_Write_Text(outstr);
      //--- get file size
      fsize := Cf_Fat_Get_File_Size;
      LongIntToStr(fsize, outstr);
      UART1_Write_Text(outstr);
    end
  else begin
    //--- file was not found - signal it
    UART1_Write(0x55);
    Delay_ms(1000);
    UART1_Write(0x55);
  end;
end;

//----- Tries to create a swap file, whose size will be at
// least 100
//          sectors (see Help for details)
procedure M_Create_Swap_File();
var i : word;
begin
  for i:=0 to 511 do
    Buffer[i] := i;

    size := Cf_Fat_Get_Swap_File(5000, 'mikroE.txt', 0x20); // see
    help on this function for details
```

```
if (size <> 0) then
  begin
    LongIntToStr(size, err_txt);
    UART1_Write_Line(err_txt);

    for i:=0 to 4999 do
      begin
        Cf_Write_Sector(size, Buffer);
        Inc(size);
        UART1_Write('.');
      end;
    end;
  end;

//----- Main. Uncomment the function(s) to test the desired
operation(s)
begin
  err_txt := 'FAT16 not found';
  file_contents := 'XX CF FAT16 library by Anton Rieckert';
  filename := 'MIKRO00xTXT';

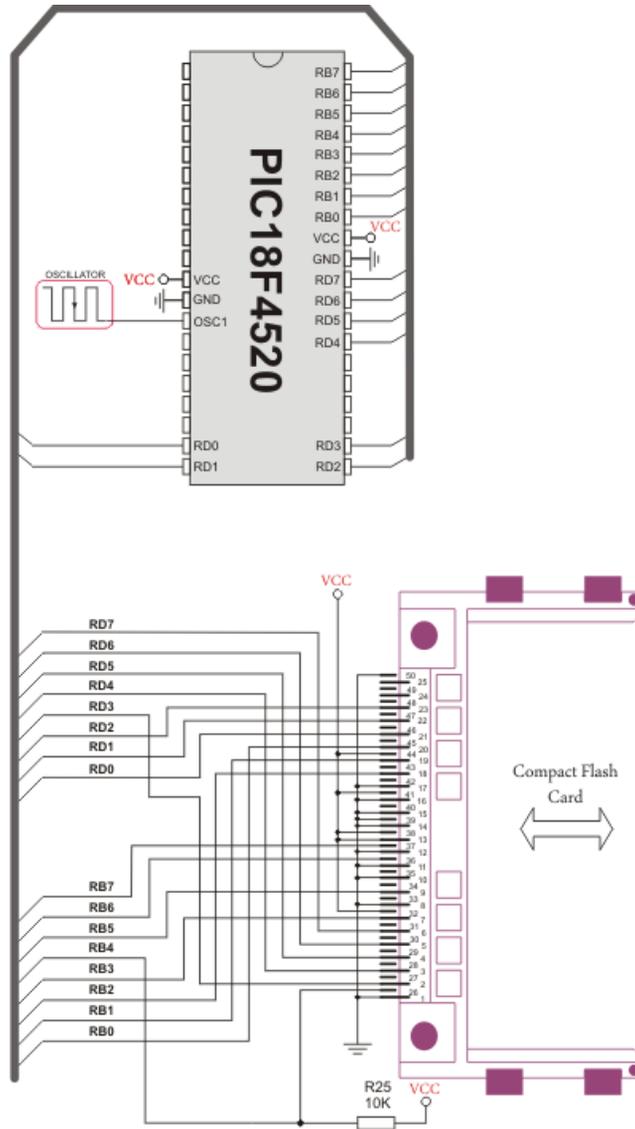
  // we will use PORTC to signal test end
  ADCON1 := ADCON1 or 0x0F; // Configure AN pins as digital
  CMCON := CMCON or 7; // Turn off comparators

  UART1_Init(19200); // Set up USART for file reading
  Delay_ms(10);
  UART1_Write_Line('PIC-Started'); // PIC present report

  // --- Init the FAT library
  // --- use Cf_Fat_QuickFormat instead of init routine if a for-
mat is needed
  if Cf_Fat_Init() = 0 then
    begin
      Delay_ms(2000); // wait for a while until the card is
stabilized // period depends on used CF card

      //--- Test start
      UART1_Write_Line('Test Start. ');
      M_Create_New_File();
      M_Create_Multiple_Files();
      M_Open_File_Rewrite();
      M_Open_File_Append();
      M_Open_File_Read();
      M_Delete_File();
      M_Test_File_Exist();
      M_Create_Swap_File();
      UART1_Write_Line('Test End. ');
    end
  else
    begin
      UART1_Write_Line(err_txt); // Note: Cf_Fat_Init tries to
initialize a card more than once. // If card is not
present, initialization may last longer (depending on clock speed)
    end;
  end;
end.
```

HW Connection



Pin diagram of CF memory card

EEPROM LIBRARY

EEPROM data memory is available with a number of PIC MCUs. mikroPascal PRO for PIC includes library for comfortable work with EEPROM.

Library Routines

- EEPROM_Read
- EEPROM_Write

EEPROM_Read

Prototype	<code>function EEPROM_Read(Address: word): byte;</code>
Returns	Returns byte from specified address.
Description	Reads data from specified <code>address</code> . Parameter <code>address</code> is of byte type, which means it can address only 256 locations. For PIC18 MCUs with more EEPROM data locations, it is programmer's responsibility to set SFR EEADRH register appropriately.
Requires	Requires EEPROM module. Ensure minimum 20ms delay between successive use of routines <code>EEPROM_Write</code> and <code>EEPROM_Read</code> . Although PIC will write the correct value, <code>EEPROM_Read</code> might return an undefined result.
Example	<code>tmp := EEPROM_Read(\$3F);</code>

EEPROM_Write

Prototype	<code>procedure EEPROM_Write(Address : word; Data_: byte);</code>
Returns	Nothing.
Description	Writes data to specified <code>address</code> . Parameter <code>address</code> is of byte type, which means it can address only 256 locations. For PIC18 MCUs with more EEPROM data locations, it is programmer's responsibility to set SFR EEADRH register appropriately. Be aware that all interrupts will be disabled during execution of <code>EEPROM_Write</code> routine (GIE bit of INTCON register will be cleared). Routine will set this bit on exit.
Requires	Requires EEPROM module. Ensure minimum 20ms delay between successive use of routines <code>EEPROM_Write</code> and <code>EEPROM_Read</code> . Although PIC will write the correct value, <code>EEPROM_Read</code> might return an undefined result.
Example	<code>EEPROM_Write(\$32);</code>

Library Example

The example demonstrates using EEPROM Library.

```

program Eeprom;

var counter : byte;           // loop variable
begin
    ANSEL := 0;                // Configure AN pins as digital I/O
    ANSELH := 0;

    PORTB := 0;
    PORTC := 0;
    PORTD := 0;

    TRISB := 0;
    TRISC := 0;
    TRISD := 0;

    for counter := 0 to 31 do           // Fill data buffer
        EEPROM_Write(0x80+counter, counter); // Write data to address
0x80+ii

    EEPROM_Write(0x02,0xAA); // Write some data at address 2
    EEPROM_Write(0x50,0x55); // Write some data at address 0150

    Delay_ms(1000);           // Blink PORTB and PORTC diodes
    PORTB := 0xFF;           // to indicate reading start
    PORTC := 0xFF;
    Delay_ms(1000);
    PORTB := 0x00;
    PORTC := 0x00;
    Delay_ms(1000);

    PORTB := EEPROM_Read(0x02);           // Read data from address
2 and display it on PORTB
    PORTC := EEPROM_Read(0x50);           // Read data from address
0x50 and display it on PORTC

    Delay_ms(1000);

    for counter := 0 to 31 do           // Read 32 bytes block
from address 0x100
        begin
            PORTD := EEPROM_Read(0x80+counter); // and display data on
PORTC
            Delay_ms(300);
        end;
end.

```

ETHERNET PIC18FXXJ60 LIBRARY

`PIC18FxxJ60` family of microcontrollers feature an embedded Ethernet controller module. This is a complete connectivity solution, including full implementations of both Media Access Control (MAC) and Physical Layer transceiver (PHY) modules. Two pulse transformers and a few passive components are all that are required to connect the microcontroller directly to an Ethernet network.

The Ethernet module meets all of the IEEE 802.3 specifications for 10-BaseT connectivity to a twisted-pair network. 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. Provisions are also made for two LED outputs to indicate link and network activity

This library provides the possibility to easily utilize ethernet feature of the above mentioned MCUs.

Ethernet PIC18FxxJ60 library supports:

- IPv4 protocol.
- ARP requests.
- ICMP echo requests.
- UDP requests.
- TCP requests (no stack, no packet reconstruction).
- ARP client with cache.
- DNS client.
- UDP client.
- DHCP client.
- packet fragmentation is **NOT** supported.

Note: Global library variable `Ethernet_userTimerSec` is used to keep track of time for all client implementations (ARP, DNS, UDP and DHCP). It is user responsibility to increment this variable each second in it's code if any of the clients is used.

Note: For advanced users there are header files ("`eth_j60LibDef.h`" and "`eth_j60LibPrivate.h`") in `Uses\P18` folder of the compiler with description of all routines and global variables, relevant to the user, implemented in the Ethernet PIC18FxxJ60 Library.

Library Routines

- Ethernet_Init
- Ethernet_Enable
- Ethernet_Disable
- Ethernet_doPacket
- Ethernet_putByte
- Ethernet_putBytes
- Ethernet_putString
- Ethernet_putConstString
- Ethernet_putConstBytes
- Ethernet_getByte
- Ethernet_getBytes
- Ethernet_UserTCP
- Ethernet_UserUDP
- Ethernet_getIpAddress
- Ethernet_getGwIpAddress
- Ethernet_getDnsIpAddress
- Ethernet_getIpMask
- Ethernet_confNetwork
- Ethernet_arpResolve
- Ethernet_sendUDP
- Ethernet_dnsResolve
- Ethernet_initDHCP
- Ethernet_doDHCPLeaseTime
- Ethernet_renewDHCP

Ethernet_Init

Prototype	<code>procedure Ethernet_Init(var mac : array[6] of byte; var ip : array[4] of byte; fullDuplex : byte);</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It initializes Ethernet controller. This function is internally splitted into 2 parts to help linker when coming short of memory.</p> <p>Ethernet 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. - half duplex loopback disabled. - LED configuration: default (LEDA-link status, LEDB-link activity). <p>Parameters:</p> <ul style="list-style-type: none"> - mac: RAM buffer containing valid MAC address. - ip: RAM buffer containing valid IP address. - fullDuplex: ethernet duplex mode switch. Valid values: 0 (half duplex mode, predefined library const <code>Ethernet_HALFDUPLEX</code>) and 1 (full duplex mode, predefined library const <code>Ethernet_FULLLDUPLEX</code>). <p>Note: If a DHCP server is to be used, IP address should be set to 0.0.0.0.</p>
Requires	Nothing.
Example	<pre>var myMacAddr : array[6] of byte; // my MAC address myIpAddr : array[4] of byte; // my IP addr ... myMacAddr[0] := 0x00; myMacAddr[1] := 0x14; myMacAddr[2] := 0xA5; myMacAddr[3] := 0x76; myMacAddr[4] := 0x19; myMacAddr[5] := 0x3F; myIpAddr[0] := 192; myIpAddr[1] := 168; myIpAddr[2] := 20; myIpAddr[3] := 60; Ethernet_Init(myMacAddr, myIpAddr, Ethernet_FULLLDUPLEX);</pre>

Description	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 MCU's internal Ethernet module. The MCU's internal Ethernet module should be properly configured by the means of Ethernet_Init routine.
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre>Ethernet_Enable(_Ethernet_CRC or _Ethernet_UNICAST); // enable CRC checking and Unicast traffic</pre>

Ethernet_Disable

Prototype	<code>procedure Ethernet_Disable(disFlt : byte);</code>		
Returns	Nothing.		
Description	<p>This is MAC module routine. This routine disables appropriate network traffic on the MCU's internal Ethernet 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>_Ethernet_BROADCAST</code>	
		<code>_Ethernet_MULTICAST</code>	
		none	
		none	
		none	
		<code>_Ethernet_CRC</code>	
		none	
		<code>_Ethernet_UNICAST</code>	

Description	<p>Note: Advance filtering available in the MCU's internal Ethernet module such as Pattern Match, Magic Packet and Hash Table 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 MCU's internal Ethernet module. The MCU's internal Ethernet module should be properly configured by the means of <code>Ethernet_Init</code> routine.</p>
Requires	Ethernet module has to be initialized. See <code>Ethernet_Init</code> .
Example	<code>Ethernet_Disable(_Ethernet_CRC or _Ethernet_UNICAST); // disable CRC checking and Unicast traffic</code>

Ethernet_doPacket

Prototype	<code>function Ethernet_doPacket() : 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. Ethernet 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>Ethernet_UserTCP</code> function is called for further processing. - upon UDP request the <code>Ethernet_UserUDP</code> function is called for further processing. <p>Note: <code>Ethernet_doPacket</code> must be called as often as possible in user's code.</p>
Requires	Ethernet module has to be initialized. See <code>Ethernet_Init</code> .
Example	<pre>while true do begin ... Ethernet_doPacket(); // process received packets ... end;</pre>

Ethernet_putByte

Prototype	<code>procedure Ethernet_putByte(v : byte);</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It stores one byte to address pointed by the current Ethernet controller's write pointer (<code>EWRPT</code>).</p> <p>Parameters:</p> <ul style="list-style-type: none">- <code>v</code>: value to store
Requires	Ethernet module has to be initialized. See <code>Ethernet_Init</code> .
Example	<pre>var data : byte; ... Ethernet_putByte(data); // put an byte into ethernet buffer</pre>

Ethernet_putBytes

Prototype	<code>procedure Ethernet_putBytes(ptr : ^byte; n : byte);</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It stores requested number of bytes into Ethernet controller's RAM starting from current Ethernet controller's write pointer (<code>EWRPT</code>) location.</p> <p>Parameters:</p> <ul style="list-style-type: none">- <code>ptr</code>: RAM buffer containing bytes to be written into Ethernet controller's RAM.- <code>n</code>: number of bytes to be written.
Requires	Ethernet module has to be initialized. See <code>Ethernet_Init</code> .
Example	<pre>var buffer : array[17] of byte; ... buffer := 'mikroElektronika'; ... Ethernet_putBytes(buffer, 16); // put an RAM array into ethernet buffer</pre>

Ethernet_putConstBytes

Prototype	<code>procedure Ethernet_putConstBytes(const ptr : ^byte; n : byte);</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It stores requested number of const bytes into Ethernet controller's RAM starting from current Ethernet controller's write pointer (EWRPT) location.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>ptr</code>: const buffer containing bytes to be written into Ethernet controller's RAM. - <code>n</code>: number of bytes to be written.
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre>const buffer : array[17] of byte; ... buffer := 'mikroElektronika'; ... Ethernet_putConstBytes(buffer, 16); // put a const array into ethernet buffer</pre>

Ethernet_putString

Prototype	<code>function Ethernet_putString(ptr : ^byte) : word;</code>
Returns	Number of bytes written into Ethernet controller's RAM.
Description	<p>This is MAC module routine. It stores whole string (excluding null termination) into Ethernet controller's RAM starting from current Ethernet controller's write pointer (EWRPT) location.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>ptr</code>: string to be written into Ethernet controller's RAM.
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre>var buffer : string[16]; ... buffer := 'mikroElektronika'; ... Ethernet_putString(buffer); // put a RAM string into ethernet buffer</pre>

Ethernet_putConstString

Prototype	<code>function Ethernet_putConstString(const ptr : ^byte) : word;</code>
Returns	Number of bytes written into Ethernet controller's RAM.
Description	<p>This is MAC module routine. It stores whole const string (excluding null termination) into Ethernet controller's RAM starting from current Ethernet controller's write pointer (<code>EWRTPT</code>) location.</p> <p>Parameters:</p> <p>- <code>ptr</code>: const string to be written into Ethernet controller's RAM.</p>
Requires	Ethernet module has to be initialized. See <code>Ethernet_Init</code> .
Example	<pre>const buffer : string[16]; ... buffer := 'mikroElektronika'; ... Ethernet_putConstString(buffer); // put a const string into ethernet buffer</pre>

Ethernet_getByte

Prototype	<code>function Ethernet_getByte() : byte;</code>
Returns	Byte read from Ethernet controller's RAM.
Description	This is MAC module routine. It fetches a byte from address pointed to by current Ethernet controller's read pointer (<code>ERDPT</code>).
Requires	Ethernet module has to be initialized. See <code>Ethernet_Init</code> .
Example	<pre>var buffer : byte; ... buffer := Ethernet_getByte(); // read a byte from ethernet buffer</pre>

Ethernet_getBytes

Prototype	<code>procedure Ethernet_getBytes(ptr : ^byte; addr : word; n : byte);</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It fetches requested number of bytes from Ethernet controller's RAM starting from given address. If value of <code>0xFFFF</code> is passed as the address parameter, the reading will start from current Ethernet controller's read pointer (<code>ERDPT</code>) location.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>ptr</code>: buffer for storing bytes read from Ethernet controller's RAM. - <code>addr</code>: Ethernet controller's RAM start address. Valid values: <code>0..8192</code>. - <code>n</code>: number of bytes to be read.
Requires	Ethernet module has to be initialized. See <code>Ethernet_Init</code> .
Example	<pre>var buffer: array[16] of byte; ... Ethernet_getBytes(buffer, 0x100, 16); // read 16 bytes, starting from address 0x100</pre>

Ethernet_UserTCP

Prototype	<code>function Ethernet_UserTCP(var remoteHost : array[4] of byte; remotePort, localPort, reqLength : word) : word;</code>
Returns	- 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 Ethernet_get routines. The user puts data in the transmit buffer by using some of the 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 <code>return(0)</code> 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 Ethernet_Init.
Example	This function is internally called by the library and should not be called by the user's code.

Ethernet_UserUDP

Prototype	<code>function Ethernet_UserUDP(var remoteHost : array[4] of byte; remotePort, destPort, reqLength : word) : word;</code>
Returns	- 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 Ethernet_get routines. The user puts data in the transmit buffer by using some of the 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 Ethernet_Init.
Example	This function is internally called by the library and should not be called by the user's code.

Ethernet_getIpAddress

Prototype	<code>function Ethernet_getIpAddress() : word;</code>
Returns	Ponter to the global variable holding IP address.
Description	This routine should be used when DHCP server is present on the network to fetch assigned IP address. Note: User should always copy the IP address from the RAM location returned by this routine into it's own IP address buffer. These locations should not be altered by the user in any case!
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre>var ipAddr : array[4] of byte; // user IP address buffer ... memcpy(ipAddr, Ethernet_getIpAddress(), 4); // fetch IP address</pre>

Ethernet_getGwIpAddress

Prototype	<code>function Ethernet_getGwIpAddress() : word;</code>
Returns	Ponter to the global variable holding gateway IP address.
Description	This routine should be used when DHCP server is present on the network to fetch assigned gateway IP address. Note: User should always copy the IP address from the RAM location returned by this routine into it's own gateway IP address buffer. These locations should not be altered by the user in any case!
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre>var gwIpAddr : array[4] of byte; // user gateway IP address buffer ... memcpy(gwIpAddr, Ethernet_getGwIpAddress(), 4); // fetch gate- way IP address</pre>

Ethernet_getDnsIpAddress

Prototype	<code>function Ethernet_getDnsIpAddress() : word;</code>
Returns	Ponter to the global variable holding DNS IP address.
Description	This routine should be used when DHCP server is present on the network to fetch assigned DNS IP address. Note: User should always copy the IP address from the RAM location returned by this routine into it's own DNS IP address buffer. These locations should not be altered by the user in any case!
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre>var dnsIpAddr : array[4] of byte; // user DNS IP address buffer ... memcpy(dnsIpAddr, Ethernet_getDnsIpAddress(), 4); // fetch DNS server address</pre>

Ethernet_getIpMask

Prototype	<code>function Ethernet_getIpMask() : word;</code>
Returns	Ponter to the global variable holding IP subnet mask.
Description	This routine should be used when DHCP server is present on the network to fetch assigned IP subnet mask. Note: User should always copy the IP address from the RAM location returned by this routine into it's own IP subnet mask buffer. These locations should not be altered by the user in any case!
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre>var IpMask : array[4] of byte; // user IP subnet mask buffer ... memcpy(IpMask, Ethernet_getIpMask(), 4); // fetch IP subnet mask</pre>

Ethernet_confNetwork

Prototype	<code>procedure Ethernet_confNetwork(var ipMask, gwIpAddr, dnsIpAddr : array[4] of byte);</code>
Returns	Nothing.
Description	<p>Configures network parameters (IP subnet mask, gateway IP address, DNS IP address) when DHCP is not used.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>ipMask</code>: IP subnet mask. - <code>gwIpAddr</code> gateway IP address. - <code>dnsIpAddr</code>: DNS IP address. <p>Note: The above mentioned network parameters should be set by this routine only if DHCP module is not used. Otherwise DHCP will override these settings.</p>
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre> var ipMask : array[4] of byte; // network mask (for example : 255.255.255.0) gwIpAddr : array[4] of byte; // gateway (router) IP address dnsIpAddr : array[4] of byte; // DNS server IP address ... gwIpAddr[0] := 192; gwIpAddr[1] := 168; gwIpAddr[2] := 20; gwIpAddr[3] := 6; dnsIpAddr[0] := 192; dnsIpAddr[1] := 168; dnsIpAddr[2] := 20; dnsIpAddr[3] := 100; ipMask[0] := 255; ipMask[1] := 255; ipMask[2] := 255; ipMask[3] := 0; ... Ethernet_confNetwork(ipMask, gwIpAddr, dnsIpAddr); // set net- work configuration parameters </pre>

Ethernet_arpResolve

Prototype	<code>function Ethernet_arpResolve(var ip : array[4] of byte; tmax : byte) : word;</code>
Returns	- MAC address behind the IP address - the requested IP address was resolved. - 0 - otherwise.
Description	<p>This is ARP module routine. It sends an ARP request for given IP address and waits for ARP reply. If the requested IP address was resolved, an ARP cash entry is used for storing the configuration. ARP cash can store up to 3 entries. For ARP cash structure refer to "eth_j60LibDef.h" header file in the compiler's Uses/P18 folder.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>ip</code>: IP address to be resolved. - <code>tmax</code>: time in seconds to wait for an reply. <p>Note: The Ethernet services are not stopped while this routine waits for ARP reply. The incoming packets will be processed normally during this time.</p>
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre> var IpAddr : array[4] of byte; // IP address ... IpAddr[0] := 192; IpAddr[0] := 168; IpAddr[0] := 1; IpAddr[0] := 1; ... Ethernet_arpResolve(IpAddr, 5); // get MAC address behind the above IP address, wait 5 secs for the response </pre>

Ethernet_sendUDP

Prototype	<code>function Ethernet_sendUDP(var destIP : array[4] of byte; sourcePort, destPort : word; pkt : ^byte; pktLen : word) : byte;</code>
Returns	1 - UDP packet was sent successfully. 0 - otherwise.
Description	This is UDP module routine. It sends an UDP packet on the network. Parameters: - <code>destIP</code> : remote host IP address. - <code>sourcePort</code> : local UDP source port number. - <code>destPort</code> : destination UDP port number. - <code>pkt</code> : packet to transmit. - <code>pktLen</code> : length in bytes of packet to transmit.
Requires	Ethernet module has to be initialized. See <code>Ethernet_Init</code> .
Example	<pre>var IpAddr : array[4] of byte; // remote IP address ... IpAddr[0] := 192; IpAddr[1] := 168; IpAddr[2] := 1; IpAddr[3] := 1; ... Ethernet_sendUDP(IpAddr, 10001, 10001, 'Hello', 5); // send Hello message to the above IP address, from UDP port 10001 to UDP port 10001</pre>

Ethernet_dnsResolve

Prototype	<code>function Ethernet_dnsResolve(var host : array[4] of byte; tmax : byte) : word;</code>
Returns	- pointer to the location holding the IP address - the requested host name was resolved. - 0 - otherwise.
Description	<p>This is DNS module routine. It sends an DNS request for given host name and waits for DNS reply. If the requested host name was resolved, it's IP address is stored in library global variable and a pointer containing this address is returned by the routine. UDP port 53 is used as DNS port.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>host</code>: host name to be resolved. - <code>tmax</code>: time in seconds to wait for an reply. <p>Note: The Ethernet services are not stopped while this routine waits for DNS reply. The incoming packets will be processed normally during this time.</p> <p>Note: User should always copy the IP address from the RAM location returned by this routine into it's own resolved host IP address buffer. These locations should not be altered by the user in any case!</p>
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre>var remoteHostIpAddr : array[4] of byte; // user host IP address buffer ... // SMTP server: // Zurich, Switzerland: Integrated Systems Lab, Swiss Fed. Inst. of Technology // 129.132.2.21: swisstime.ethz.ch // Service Area: Switzerland and Europe memcpy(remoteHostIpAddr, Ethernet_dnsResolve('swisstime.ethz.ch', 5), 4);</pre>

Ethernet_initDHCP

Prototype	<code>function Ethernet_initDHCP(tmax : byte) : byte;</code>
Returns	<ul style="list-style-type: none"> - 1 - network parameters were obtained successfully. - 0 - otherwise.
Description	<p>This is DHCP module routine. It sends an DHCP request for network parameters (IP, gateway, DNS addresses and IP subnet mask) and waits for DHCP reply. If the requested parameters were obtained successfully, their values are stored into the library global variables.</p> <p>These parameters can be fetched by using appropriate library IP get routines:</p> <ul style="list-style-type: none"> - Ethernet_getIpAddress - fetch IP address. - Ethernet_getGwIpAddress - fetch gateway IP address. - Ethernet_getDnsIpAddress - fetch DNS IP address. - Ethernet_getIpMask - fetch IP subnet mask. <p>UDP port 68 is used as DHCP client port and UDP port 67 is used as DHCP server port.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>tmax</code>: time in seconds to wait for an reply. <p>Note: The Ethernet services are not stopped while this routine waits for DNS reply. The incoming packets will be processed normally during this time.</p> <p>Note: When DHCP module is used, global library variable <code>Ethernet_userTimerSec</code> is used to keep track of time. It is user responsibility to increment this variable each second in it's code.</p>
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre> ... Ethernet_initDHCP(5); // get network configuration from DHCP server, wait 5 sec for the response ... </pre>

Ethernet_doDHCPLeaseTime

Prototype	<code>function Ethernet_doDHCPLeaseTime() : byte;</code>
Returns	- 0 - lease time has not expired yet. - 1 - lease time has expired, it's time to renew it.
Description	This is DHCP module routine. It takes care of IP address lease time by decrementing the global lease time library counter. When this time expires, it's time to contact DHCP server and renew the lease.
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre> while true do begin ... if(Ethernet_doDHCPLeaseTime() <> 0) then begin ... // it's time to renew the IP address lease end; end; end; end; </pre>

Ethernet_renewDHCP

Prototype	<code>function Ethernet_renewDHCP(tmax : byte) : byte;</code>
Returns	- 1 - upon success (lease time was renewed). - 0 - otherwise (renewal request timed out).
Description	This is DHCP module routine. It sends IP address lease time renewal request to DHCP server. Parameters: - <code>tmax</code> : time in seconds to wait for an reply.
Requires	Ethernet module has to be initialized. See Ethernet_Init.
Example	<pre> while true do begin ... if(Ethernet_doDHCPLeaseTime() <> 0) then begin Ethernet_renewDHCP(5); // it's time to renew the IP address lease, with 5 secs for a reply end; ... end; end; </pre>

Library Example

This code shows how to use the PIC18FxxJ60 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 RD0 to RD7 bit and return HTML main page
- all other requests return also HTML main page.

```
program enc_ethernet;

{ *****
  * RAM variables
  *}

var myMacAddr      : array[ 6] of byte  ; // my MAC address
    myIpAddr       : array[ 4] of byte  ; // my IP address
    gwIpAddr       : array[ 4] of byte  ; // gateway (router) IP address
    ipMask          : array[ 4] of byte  ; // network mask (for example
: 255.255.255.0)
    dnsIpAddr      : array[ 4] of byte  ; // DNS server IP address

{ *****
  * ROM constant strings
  *}
const httpHeader : string[ 30] = 'HTTP/1.1 200 OK'+#10+'Content-type:
' ; // HTTP header
const httpMimeTypeHTML : string[ 11]      = 'text/html'+#10+#10 ;
// HTML MIME type
const httpMimeTypeScript : string[ 12]    = 'text/plain'+#10+#10 ;
// TEXT MIME type
const httpMethod : string[ 5] = 'GET /';
{ *
  * web page, splitted 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 indexPage : string[ 759] =
    '<meta http-equiv="refresh"
content="3;url=http://192.168.20.60">' +
    '<HTML><HEAD></HEAD><BODY>'+
    '<h1>PIC18FxxJ60 Mini Web Server</h1>'+
    '<a href=/>Reload</a>'+
    '<script src=/s></script>'+
    '<table><tr><td valign=top><table border=1
style="font-size:20px ;font-family: terminal ;">'+
    '<tr><th colspan=2>ADC</th></tr>'+

'<tr><td>AN2</td><td><script>document.write(AN2)</script></td></tr>'
+

'<tr><td>AN3</td><td><script>document.write(AN3)</script></td></tr>'
+

    '</table></td><td><table border=1 style="font-
size:20px ;font-family: terminal ;">'+
    '<tr><th colspan=2>PORTB</th></tr>'+
    '<script>'+
    'var str,i;'+
    'str="";'+
    'for(i=0;i<8;i++)'+
    '{ str+="<tr><td bgcolor=pink>BUTTON #" +i+"</td>";'+
    'if(PORTB&(1<<i)){ str+="<td bgcolor=red>ON";}' +
    'else { str+="<td bgcolor=#cccccc>OFF";}' +
    'str+="</td></tr>";}' +
    'document.write(str) ;'+
    '</script>';

const indexPage2 : string[ 466] =
    '</table></td><td>'+
    '<table border=1 style="font-size:20px ;font-
family: terminal ;">'+
    '<tr><th colspan=3>PORTD</th></tr>'+
    '<script>'+
    'var str,i;'+
    'str="";'+
    'for(i=0;i<3;i++)'+
    '{ str+="<tr><td bgcolor=yellow>LED #" +i+"</td>";'+
    'if(PORTD&(1<<i)){ str+="<td bgcolor=red>ON";}' +
    'else { str+="<td bgcolor=#cccccc>OFF";}' +
    'str+="</td><td><a
href=/t"+i+">Toggle</a></td></tr>";}' +
    'document.write(str) ;'+
    '</script>'+
    '</table></td></tr></table>'+
    'This is HTTP request
#<script>document.write(REQ)</script></BODY></HTML>';

```

```
var    getRequest  : array[ 15] of byte; // HTTP request buffer
      dyna         : array[ 30] of byte; // buffer for dynamic response
      httpCounter  : word ;           // counter of HTTP requests

{*****
 * user defined functions
 *}

{ *
 * this function is called by the library
 * the user accesses to the HTTP request by successive calls to
Ethernet_getByte()
 * the user puts data in the transmit buffer by successive calls to
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
 *
 *}

function Ethernet_UserTCP(var remoteHost : array[ 4] of byte;
                          remotePort, localPort, reqLength
: word) : word;
var    i : word ;           // my reply length
      bitMask : byte ; // for bit mask
      tmp: string[ 11]; // to copy const array to ram for memcmp
begin
  result := 0;

  if(localPort <> 80) then // I listen only to web request on port 80
  begin
    result := 0;
    exit;
  end;

  // get 10 first bytes only of the request, the rest does not mat-
  ter here
  for i := 0 to 9 do
    getRequest[ i ] := Ethernet_getByte() ;
    getRequest[ i ] := 0 ;

  // copy httpMethod to ram for use in memcmp routine
  for i := 0 to 4 do
    tmp[ i ] := httpMethod[ i ] ;

  if(memcmp(@getRequest, @tmp, 5) <> 0) then // only GET method
  is supported here
  begin
    result := 0 ;
    exit;
  end;
```

```
Inc(httpCounter) ; // one more request done

    if(getRequest[ 5] = 's') then // if request path
name starts with s, store dynamic data in transmit buffer
    begin
        // the text string replied by this request can be interpret-
ed as javascript statements
        // by browsers
        result := Ethernet_putConstString(@httpHeader) ;// HTTP header
        result := result +
Ethernet_putConstString(@httpMimeTypeScript) ; // with text MIME type

        // add AN2 value to reply
        WordToStr(ADC_Read(2), dyna) ;
        tmp := 'var AN2=';
        result := result + Ethernet_putString(@tmp) ;
        result := result + Ethernet_putString(@dyna) ;
        tmp := ';;';
        result := result + Ethernet_putString(@tmp) ;

        // add AN3 value to reply
        WordToStr(ADC_Read(3), dyna) ;
        tmp := 'var AN3=';
        result := result + Ethernet_putString(@tmp) ;
        result := result + Ethernet_putString(@dyna) ;
        tmp := ';;';
        result := result + Ethernet_putString(@tmp) ;

        // add PORTB value (buttons) to reply
        tmp := 'var PORTB=' ;
        result := result + Ethernet_putString(@tmp) ;
        WordToStr(PORTB, dyna) ;
        result := result + Ethernet_putString(@dyna) ;
        tmp := ';;';
        result := result + Ethernet_putString(@tmp) ;

        // add PORTD value (LEDs) to reply
        tmp := 'var PORTD=' ;
        result := result + Ethernet_putString(@tmp) ;
        WordToStr(PORTD, dyna) ;
        result := result + Ethernet_putString(@dyna) ;
        tmp := ';;';
        result := result + Ethernet_putString(@tmp) ;

        // add HTTP requests counter to reply
        WordToStr(httpCounter, dyna) ;
        tmp := 'var REQ=' ;
        result := result + Ethernet_putString(@tmp) ;
        result := result + Ethernet_putString(@dyna) ;
        tmp := ';;';
        result := result + Ethernet_putString(@tmp) ;
    end
```

```
else
    if(getRequest[ 5] = 't') then // if request path
name starts with t, toggle PORTD (LED) bit number that comes after
    begin
        bitMask := 0;
        if(isdigit(getRequest[ 6]) <> 0) then // if 0
<= bit number <= 9, bits 8 & 9 does not exist but does not matter
            begin
                bitMask := getRequest[ 6] - '0' ; // con-
vert ASCII to integer
                bitMask := 1 shl bitMask ; // cre-
ate bit mask
                PORTD := PORTD xor bitMask ; // tog-
gle PORTD with xor operator
            end;
        end;

        if(result = 0) then // what do to by default
            begin
                result := Ethernet_putConstString(@httpHeader) ;
// HTTP header
                result := result + Ethernet_putConstString(@httpMimeTypeHTML)
; // with HTML MIME type
                result := result + Ethernet_putConstString(@indexPath) ;
// HTML page first part
                result := result + Ethernet_putConstString(@indexPath2) ;
// HTML page second part
            end;
// return to the library with the number of bytes to transmit
end;

{ *
* this function is called by the library
* the user accesses to the UDP request by successive calls to
Ethernet_getByte()
* the user puts data in the transmit buffer by successive calls to
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
*
*}
function Ethernet_UserUDP(var remoteHost : array[ 4] of byte;
                           remotePort, destPort, reqLength
: word) : word;
    var tmp : string[ 5];
```

```
begin
    result := 0;
    // reply is made of the remote host IP address in human readable
format
    byteToStr(remoteHost[ 0], dyna) ;           // first IP address
byte
    dyna[ 3] := '.' ;
    byteToStr(remoteHost[ 1], tmp) ;           // second
    dyna[ 4] := tmp[ 0] ;
    dyna[ 5] := tmp[ 1] ;
    dyna[ 6] := tmp[ 2] ;
    dyna[ 7] := '.' ;
    byteToStr(remoteHost[ 2], tmp) ;           // second
    dyna[ 8] := tmp[ 0] ;
    dyna[ 9] := tmp[ 1] ;
    dyna[ 10] := tmp[ 2] ;
    dyna[ 11] := '.' ;
    byteToStr(remoteHost[ 3], tmp) ;           // second
    dyna[ 12] := tmp[ 0] ;
    dyna[ 13] := tmp[ 1] ;
    dyna[ 14] := tmp[ 2] ;

    dyna[ 15] := ':' ;                          // add separator

    // then remote host port number
    WordToStr(remotePort, tmp) ;
    dyna[ 16] := tmp[ 0] ;
    dyna[ 17] := tmp[ 1] ;
    dyna[ 18] := tmp[ 2] ;
    dyna[ 19] := tmp[ 3] ;
    dyna[ 20] := tmp[ 4] ;
    dyna[ 21] := '[' ;
    WordToStr(destPort, tmp) ;
    dyna[ 22] := tmp[ 0] ;
    dyna[ 23] := tmp[ 1] ;
    dyna[ 24] := tmp[ 2] ;
    dyna[ 25] := tmp[ 3] ;
    dyna[ 26] := tmp[ 4] ;
    dyna[ 27] := ']' ;
    dyna[ 28] := 0 ;

    // the total length of the request is the length of the dynamic
string plus the text of the request
    result := 28 + reqLength ;

    // puts the dynamic string into the transmit buffer
    Ethernet_putBytes(@dyna, 28) ;
```

```
// then puts the request string converted into upper char into the
transmit buffer
    while(reqLength <> 0) do
        begin
            Ethernet_putByte(Ethernet_getByte() ) ;
            reqLength := reqLength - 1;
        end;
    // back to the library with the length of the UDP reply
end;

begin
    ADCON1 := 0x0B ;           // ADC convertors will be used with AN2
and AN3
    CMCON  := 0x07 ;           // turn off comparators

    PORTA := 0 ;
    TRISA := 0x0C ;           // RA2:RA3 - analog inputs
                                // RA1:RA0 - ethernet LEDA:LEDB

    PORTB := 0 ;
    TRISB := 0xff ;           // set PORTB as input for buttons

    PORTD := 0 ;
    TRISD := 0 ;             // set PORTD as output

    httpCounter := 0;

    // set mac address
    myMacAddr[ 0] := 0x00;
    myMacAddr[ 1] := 0x14;
    myMacAddr[ 2] := 0xA5;
    myMacAddr[ 3] := 0x76;
    myMacAddr[ 4] := 0x19;
    myMacAddr[ 5] := 0x3F;

    // set IP address
    myIpAddr[ 0] := 192;
    myIpAddr[ 1] := 168;
    myIpAddr[ 2] := 20;
    myIpAddr[ 3] := 60;

    // set gateway address
    gwIpAddr[ 0] := 192;
    gwIpAddr[ 1] := 168;
    gwIpAddr[ 2] := 20;
    gwIpAddr[ 3] := 6;
```

```
// set dns address
dnsIpAddr[ 0] := 192;
dnsIpAddr[ 1] := 168;
dnsIpAddr[ 2] := 20;
dnsIpAddr[ 3] := 1;

// set subnet mask
ipMask[ 0] := 255;
ipMask[ 1] := 255;
ipMask[ 2] := 255;
ipMask[ 3] := 0;

{ *
 * starts ENC28J60 with :
 * reset bit on PORTC.B0
 * CS bit on PORTC.B1
 * my MAC & IP address
 * full duplex
 *}

Ethernet_Init(myMacAddr, myIpAddr, _Ethernet_FULLDUPLEX) ; // init
ethernet module
Ethernet_setUserHandlers (@Ethernet_UserTCP, @Ethernet_UserUDP); //
set user handlers

// dhcp will not be used here, so use preconfigured addresses
Ethernet_confNetwork(ipMask, gwIpAddr, dnsIpAddr) ;

while true do // do forever
begin
Ethernet_doPacket() ; // process incoming Ethernet packets

{ *
 * add your stuff here if needed
 * Ethernet_doPacket() must be called as often as possible
 * otherwise packets could be lost
 *}
end;
end.
```

FLASH MEMORY LIBRARY

This library provides routines for accessing microcontroller Flash memory. Note that prototypes differ for PIC16 and PIC18 families.

Note: Due to the P16/P18 family flash specifics, flash library is MCU dependent. Since the P18 family differ significantly in number of bytes that can be erased and/or written to specific MCUs, the appropriate suffix is added to the names of functions in order to make it easier to use them. Flash memory operations are MCU dependent :

1. **Read** operation supported. For this group of MCU's only read function is implemented.
2. **Read** and **Write** operations supported (write is executed as erase-and-write). For this group of MCU's read and write functions are implemented. Note that write operation which is executed as erase-and-write, may write less bytes than it erases.
3. **Read**, **Write** and **Erase** operations supported. For this group of MCU's read, write and erase functions are implemented. Further more, flash memory block has to be erased prior to writing (write operation is not executed as erase-and-write).

Please refer to MCU datasheet before using flash library.

Library Routines

- FLASH_Read
- FLASH_Read_N_Bytes
- FLASH_Write
- FLASH_Write_8
- FLASH_Write_16
- FLASH_Write_32
- FLASH_Write_64
- FLASH_Erase
- FLASH_Erase_64
- FLASH_Erase_1024
- FLASH_Erase_Write
- FLASH_Erase_Write_64
- FLASH_Erase_Write_1024

FLASH_Read

Prototype	<pre>// for PIC16 function FLASH_Read(Address: word): word; // for PIC18 function FLASH_Read(address : dword) : byte;</pre>
Returns	Returns data byte from Flash memory.
Description	Reads data from the specified address in Flash memory.
Requires	Nothing.
Example	<pre>// for PIC18 var tmp : byte; ... begin ... tmp := FLASH_Read(0x0D00); ... end.</pre>

FLASH_Read_N_Bytes

Prototype	<pre>// for PIC18 procedure FLASH_Read_N_Bytes(address : longint; var data : byte; N : word);</pre>
Returns	Nothing.
Description	Reads N data from the specified <code>address</code> in Flash memory to variable pointed by <code>data</code>
Requires	Nothing.
Example	<code>FLASH_Read_N(0x0D00, data_buffer, sizeof(data_buffer));</code>

FLASH_Write

Prototype	<pre>// for PIC16 procedure FLASH_Write(Address : word; var Data : array[4] of word); // for PIC18 procedure FLASH_Write_8(address : dword; var data: array[8] of byte); procedure FLASH_Write_16(address : dword; var data: array[16] of byte); procedure FLASH_Write_32(address : dword; var data: array[32] of byte); procedure FLASH_Write_64(address : dword; var data: array[64] of byte);</pre>
Returns	Nothing.
Description	<p>Writes block of data to Flash memory. Block size is MCU dependent.</p> <p>P16: This function may erase memory segment before writing block of data to it (MCU dependent). Furthermore, memory segment which will be erased may be greater than the size of the data block that will be written (MCU dependent). Therefore it is recommended to write as many bytes as you erase. FLASH_Write writes 4 flash memory locations in a row, so it needs to be called as many times as it is necessary to meet the size of the data block that will be written.</p> <p>P18: This function does not perform erase prior to write.</p>
Requires	Flash memory that will be written may have to be erased before this function is called (MCU dependent). Refer to MCU datasheet for details.
Example	<p>Write consecutive values in 64 consecutive locations, starting from 0x0D00:</p> <pre>var toWrite : array[64] of byte; ... begin ... // initialize array: for i := 0 to 63 do toWrite[i] := i; ... // write contents of the array to the address 0x0D00: FLASH_Write_64(0x0D00, toWrite); ... end.</pre>

FLASH_Erase

Prototype	<pre>// for PIC16 procedure FLASH_Erase(address : word); // for PIC18 procedure FLASH_Erase_64(address : dword); procedure FLASH_Erase_1024(address : dword);</pre>
Returns	Nothing.
Description	Erases memory block starting from a given address. For P16 family is implemented only for those MCU's whose flash memory does not support erase-and-write operations (refer to datasheet for details).
Requires	Nothing.
Example	<pre>Erase 64 byte memory memory block, starting from address \$0D00: FLASH_Erase_64(\$0D00);</pre>

FLASH_Erase_Write

Prototype	<pre>// for PIC18 procedure FLASH_Erase_Write_64(address : dword; var data : array[64] of byte); procedure FLASH_Erase_Write_1024(address : dword; var data : array[1024] of byte);</pre>
Returns	None.
Description	Erase then write memory block starting from a given address.
Requires	None.
Example	<pre>var toWrite : array[64] of byte; ... begin ... // initialize array: for i := 0 to 63 do toWrite[i] := i; ... // erase block of memory at address 0x0D00 then write contents of the array to the address 0x0D00: FLASH_Erase_Write_64(0x0D00, toWrite); ... end.</pre>

Library Example

This is a simple demonstration how to use to PIC16 internal flash memory to store data. The data is being written starting from the given location; then, the same locations are read and the data is displayed on PORTB and PORTC.

```
var counter : byte;
    addr, data_ : word;
    dataAR: array[ 5] of array[ 4] of word;

begin
    ANSEL := 0; // configure AN pins as digital I/O
    ANSELH := 0;
    PORTB := 0; // initial PORTB value
    TRISB := 0; // set PORTB as output
    PORTC := 0; // initial PORTC value
    TRISC := 0; // set PORTC as output
    Delay_ms(500);

    dataAR[ 0][ 0] := $FFAA+0;
    dataAR[ 0][ 1] := $FFAA+1;
    dataAR[ 0][ 2] := $FFAA+2;
    dataAR[ 0][ 3] := $FFAA+3;
    dataAR[ 1][ 0] := $FFAA+4;
    dataAR[ 1][ 1] := $FFAA+5;
    dataAR[ 1][ 2] := $FFAA+6;
    dataAR[ 1][ 3] := $FFAA+7;
    dataAR[ 2][ 0] := $FFAA+8;
    dataAR[ 2][ 1] := $FFAA+9;
    dataAR[ 2][ 2] := $FFAA+10;
    dataAR[ 2][ 3] := $FFAA+11;
    dataAR[ 3][ 0] := $FFAA+12;
    dataAR[ 3][ 1] := $FFAA+13;
    dataAR[ 3][ 2] := $FFAA+14;
    dataAR[ 3][ 3] := $FFAA+15;
    dataAR[ 4][ 0] := $FFAA+16;
    dataAR[ 4][ 1] := $FFAA+17;
    dataAR[ 4][ 2] := $FFAA+18;
    dataAR[ 4][ 3] := $FFAA+19;

    addr := 0x0430; // starting Flash address,
    valid for P16F887
    for counter := 0 to 4 do
        begin // write some data to Flash
            Delay_ms(100);
            Flash_Write(addr+counter*4, dataAR[ counter]);
        end;
```

```
    Delay_ms(500);

    addr := 0x0430;
    for counter := 0 to 19 do
        begin
            data_ := Flash_Read(addr);           // P16's FLASH is 14-bit
wide, so
            Inc(addr);
            Delay_us(10);                       // two MSB's will always
be '00'
            PORTB := data_;                     // display data on PORTB
LS Byte
            PORTC := word(data_ shr 8);        // and PORTC MS Byte
            Delay_ms(500);
        end;
    end.
```

GRAPHIC LCD LIBRARY

The mikroPascal PRO for PIC 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>var GLCD_DataPort : byte; sfr; external;</code>	Glcd Data Port.	<code>var GLCD_DataPort : byte at PORTD;</code>
<code>var GLCD_CS1 : sbit; sfr; external;</code>	Chip Select 1 line.	<code>var GLCD_CS1 : sbit at RB0_bit;</code>
<code>var GLCD_CS2 : sbit; sfr; external;</code>	Chip Select 2 line.	<code>var GLCD_CS2 : sbit at RB1_bit;</code>
<code>var GLCD_RS : sbit; sfr; external;</code>	Register select line.	<code>var GLCD_RS : sbit at RB2_bit;</code>
<code>var GLCD_RW : sbit; sfr; external;</code>	Read/Write line.	<code>var GLCD_RW : sbit at RB3_bit;</code>
<code>var GLCD_EN : sbit; sfr; external;</code>	Enable line.	<code>var GLCD_EN : sbit at RB4_bit;</code>
<code>var GLCD_RST : sbit; sfr; external;</code>	Reset line.	<code>var GLCD_RST : sbit at RB5_bit;</code>
<code>var GLCD_CS1_Direction : sbit; sfr; external;</code>	Direction of the Chip Select 1 pin.	<code>var GLCD_CS1_Direction : sbit at TRISB0_bit;</code>
<code>var GLCD_CS2_Direction : sbit; sfr; external;</code>	Direction of the Chip Select 2 pin.	<code>var GLCD_CS2_Direction : sbit at TRISB1_bit;</code>
<code>var GLCD_RS_Direction : sbit; sfr; external;</code>	Direction of the Register select pin.	<code>var GLCD_RS_Direction : sbit at TRISB2_bit;</code>
<code>var GLCD_RW_Direction : sbit; sfr; external;</code>	Direction of the Read/Write pin.	<code>var GLCD_RW_Direction : sbit at TRISB3_bit;</code>
<code>var GLCD_EN_Direction : sbit; sfr; external;</code>	Direction of the Enable pin.	<code>var GLCD_EN_Direction : sbit at TRISB4_bit;</code>
<code>var GLCD_RST_Direction : sbit; sfr; external;</code>	Direction of the Reset pin.	<code>var GLCD_RST_Direction : sbit at TRISB5_bit;</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>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"> - <code>GLCD_CS1</code> : Chip select 1 signal pin - <code>GLCD_CS2</code> : Chip select 2 signal pin - <code>GLCD_RS</code> : Register select signal pin - <code>GLCD_RW</code> : Read/Write Signal pin - <code>GLCD_EN</code> : Enable signal pin - <code>GLCD_RST</code> : Reset signal pin - <code>GLCD_DataPort</code> : Data port - <code>GLCD_CS1_Direction</code> : Direction of the Chip select 1 pin - <code>GLCD_CS2_Direction</code> : Direction of the Chip select 2 pin - <code>GLCD_RS_Direction</code> : Direction of the Register select signal pin - <code>GLCD_RW_Direction</code> : Direction of the Read/Write signal pin - <code>GLCD_EN_Direction</code> : Direction of the Enable signal pin - <code>GLCD_RST_Direction</code> : Direction of the Reset signal pin <p>must be defined before using this function.</p>
Example	<pre>// Glcd module connections var GLCD_DataPort : byte at PORTD; var GLCD_CS1 : sbit at RB0_bit; GLCD_CS2 : sbit at RB1_bit; GLCD_RS : sbit at RB2_bit; GLCD_RW : sbit at RB3_bit; GLCD_EN : sbit at RB4_bit; GLCD_RST : sbit at RB5_bit; var GLCD_CS1_Direction : sbit at TRISB0_bit; GLCD_CS2_Direction : sbit at TRISB1_bit; GLCD_RS_Direction : sbit at TRISB2_bit; GLCD_RW_Direction : sbit at TRISB3_bit; GLCD_EN_Direction : sbit at TRISB4_bit; GLCD_RST_Direction : sbit at TRISB5_bit; // End Glcd module connections ... ANSEL = 0; ANSELH = 0; Glcd_Init();</pre>

Glcd_Set_Side

Prototype	<code>procedure Glcd_Set_Side(x_pos: 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 Glcd_Init 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>procedure Glcd_Set_X(x_pos: 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 Glcd_Init routine.
Example	<pre>Glcd_Set_X(25);</pre>

Glcd_Set_Page

Prototype	<code>procedure Glcd_Set_Page(page: 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 Glcd_Init routine.
Example	<code>Glcd_Set_Page(5);</code>

Glcd_Read_Data

Prototype	<code>function Glcd_Read_Data(): 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 Glcd_Init routine.</p> <p>Glcd side, x-axis position and page should be set first. See functions Glcd_Set_Side, Glcd_Set_X, and Glcd_Set_Page.</p>
Example	<pre>var data: byte; ... data := Glcd_Read_Data();</pre>

Glcd_Write_Data

Prototype	<code>procedure Glcd_Write_Data(ddata: 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>var data: byte; ... Glcd_Write_Data(data);</pre>

Glcd_Fill

Prototype	<code>procedure Glcd_Fill(pattern: byte);</code>
Returns	Nothing.
Description	Fills Glcd memory with the byte pattern. 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>procedure Glcd_Dot(x_pos: byte; y_pos: byte; color: 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 Glcd_Init routine.
Example	<pre>' Invert the dot in the upper left corner Glcd_Dot(0, 0, 2);</pre>

Glcd_Line

Prototype	<code>procedure Glcd_Line(x_start: integer; y_start: integer; x_end: integer; y_end: integer; color: 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 Glcd_Init 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>procedure Glcd_V_Line(y_start: byte; y_end: byte; x_pos: byte; color: 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 color determines the line color: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	<code>' Draw a vertical line between dots (10,5) and (10,25) Glcd_V_Line(5, 25, 10, 1);</code>

Glcd_H_Line

Prototype	<code>procedure Glcd_V_Line(x_start: byte; x_end: byte; y_pos: byte; color: 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 color determines the line color: 0 white, 1 black, and 2 inverts each dot.</p>
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	<code>' Draw a horizontal line between dots (10,20) and (50,20) Glcd_H_Line(10, 50, 20, 1);</code>

Glcd_Rectangle

Prototype	<code>procedure Glcd_Rectangle(x_upper_left: byte; y_upper_left: byte; x_bottom_right: byte; y_bottom_right: byte; color: 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 Glcd_Init 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>procedure Glcd_Box(x_upper_left: byte; y_upper_left: byte; x_bottom_right: byte; y_bottom_right: byte; color: 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 Glcd_Init 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>procedure Glcd_Circle(x_center: integer; y_center: integer; radius: integer; color: 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>procedure Glcd_Set_Font(const ActiveFont: ^byte; FontWidth: byte; FontHeight: byte; FontOffs: word);</code>
Returns	Nothing.
Description	<p>Sets font that will be used with <code>Glcd_Write_Char</code> and <code>Glcd_Write_Text</code> 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 byte - <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 mikroPascal PRO for PIC character set and regular ASCII set (eg. if 'A' is 65 in ASCII character, and 'A' is 45 in the mikroPascal PRO for PIC character set, <code>aFontOffs</code> 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 “<code>__Lib_GLCDFonts.mpas</code>” file located in the Uses folder or create his own fonts.</p>
Requires	Glcd needs to be initialized, see <code>Glcd_Init</code> 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>procedure Glcd_Write_Char(chr: byte; x_pos: byte; page_num: byte; color: 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 color 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 Glcd_Init routine. Use Glcd_Set_Font 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 character 'C' on the position 10 inside the page 2: Glcd_Write_Char('C', 10, 2, 1);</code>

Glcd_Write_Text

Prototype	<pre>procedure Glcd_Write_Text(var text: array[20] of char; x_pos: byte; page_num: byte; color: byte);</pre>
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 color 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 Glcd_Init routine. Use Glcd_Set_Font 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 text "Hello world!" on the position 10 inside the page 2: Glcd_Write_Text("Hello world!", 10, 2, 1);</pre>

Glcd_Image

Prototype	<pre>procedure Glcd_Image(const image: ^byte);</pre>
Returns	Nothing.
Description	<p>Displays bitmap on Glcd.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>image</code>: image to be displayed. Bitmap array must be located in code memory. <p>Use the mikroPascal PRO for PIC integrated Glcd Bitmap Editor to convert image to a constant array suitable for displaying on Glcd.</p>
Requires	Glcd needs to be initialized, see Glcd_Init 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_Test;

uses bitmap;

// Glcd module connections
var GLCD_DataPort : byte at PORTD;

var GLCD_CS1 : sbit at RB0_bit;
    GLCD_CS2 : sbit at RB1_bit;
    GLCD_RS  : sbit at RB2_bit;
    GLCD_RW  : sbit at RB3_bit;
    GLCD_EN  : sbit at RB4_bit;
    GLCD_RST : sbit at RB5_bit;

var GLCD_CS1_Direction : sbit at TRISB0_bit;
    GLCD_CS2_Direction : sbit at TRISB1_bit;
    GLCD_RS_Direction  : sbit at TRISB2_bit;
    GLCD_RW_Direction  : sbit at TRISB3_bit;
    GLCD_EN_Direction  : sbit at TRISB4_bit;
    GLCD_RST_Direction : sbit at TRISB5_bit;
// End Glcd module connections

var counter : byte;
    someText : array[18] of char;

procedure Delay2S();           // 2 seconds delay function
begin
    Delay_ms(2000);
end;

begin

    ANSEL  := 0;                // Configure AN pins as digital I/O
    ANSELH := 0;

    Glcd_Init();                // Initialize Glcd
    Glcd_Fill(0x00);            // Clear Glcd

    while TRUE do
        begin
            Glcd_Image(@truck_bmp);           // Draw image
            Delay2S(); delay2S();

            Glcd_Fill(0x00);                // Clear Glcd
```

```
    Glcd_Box(62,40,124,63,1);           // Draw box
    Glcd_Rectangle(5,5,84,35,1);       // Draw rectangle
    Glcd_Line(0, 0, 127, 63, 1);       // Draw line
    Delay2S();
    counter := 5;

    while (counter <= 59) do // Draw horizontal and vertical lines
    begin
        Delay_ms(250);
        Glcd_V_Line(2, 54, counter, 1);
        Glcd_H_Line(2, 120, counter, 1);
        Counter := counter + 5;
    end;

    Delay2S();

    Glcd_Fill(0x00);                     // Clear Glcd

        Glcd_Set_Font(@Character8x7, 8, 7, 32); // Choose font
"Character8x7"
    Glcd_Write_Text('mikroE', 1, 7, 2); // Write string

    for counter := 1 to 10 do           // Draw circles
        Glcd_Circle(63,32, 3*counter, 1);
    Delay2S();

    Glcd_Box(10,20, 70,63, 2);         // Draw box}
    Delay2S();

    Glcd_Fill(0xFF);                     // Fill Glcd
    Glcd_Set_Font(@Character8x7, 8, 7, 32); // Change font
    someText := '8x7 Font';
    Glcd_Write_Text(someText, 5, 0, 2); // Write string
    delay2S();

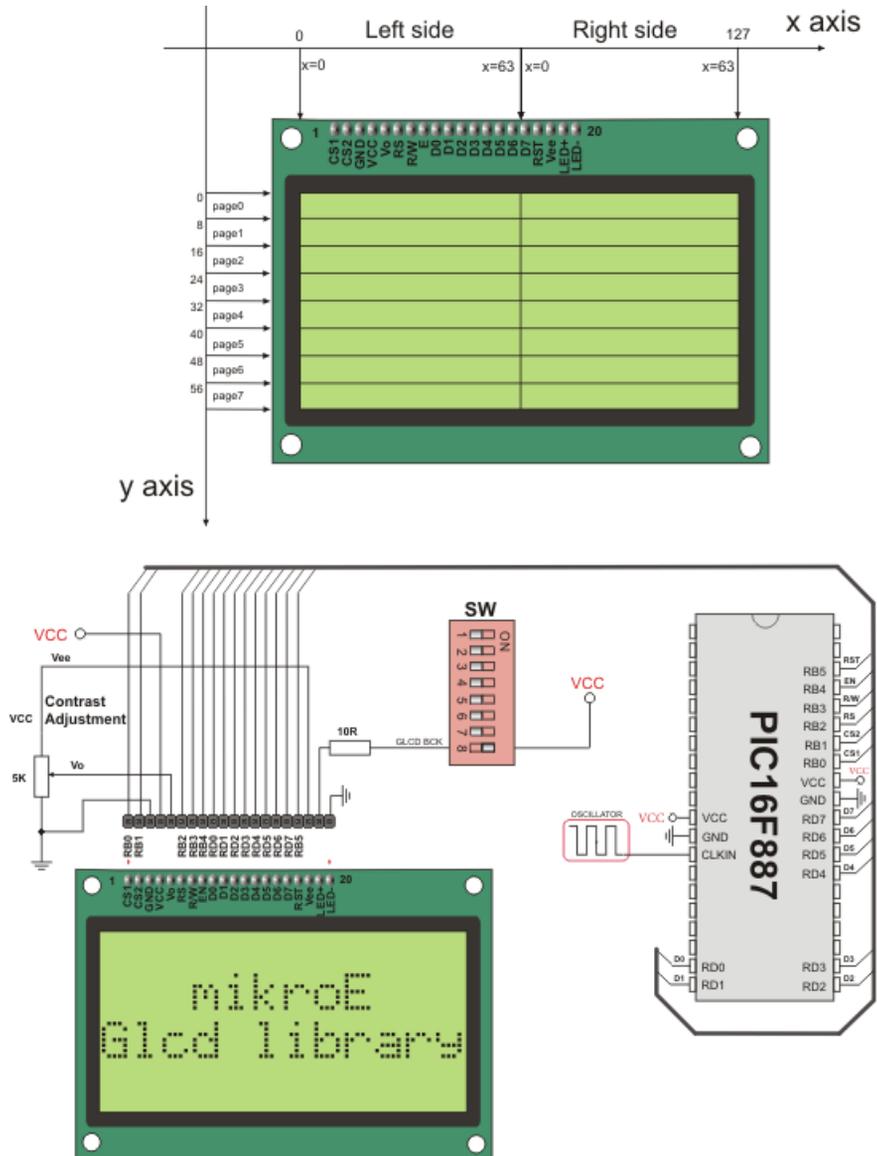
    Glcd_Set_Font(@System3x6, 3, 5, 32); // Change font
    someText := '3X5 CAPITALS ONLY';
    Glcd_Write_Text(someText, 60, 2, 2); // Write string
    delay2S();

    Glcd_Set_Font(@font5x7, 5, 7, 32); // Change font
    someText := '5x7 Font';
    Glcd_Write_Text(someText, 5, 4, 2); // Write string
    delay2S();

    Glcd_Set_Font(@FontSystem5x7_v2, 5, 7, 32); // Change font
    someText := '5x7 Font (v2)';
    Glcd_Write_Text(someText, 50, 6, 2); // Write string
    delay2S();

end;
end.
```

HW Connection



Glcd HW connection

I²C Library

I²C full master MSSP module is available with a number of PIC MCU models. mikroPascal PRO for PIC provides library which supports the master I²C mode.

Note: Some MCUs have multiple I²C modules. In order to use the desired I²C library routine, simply change the number 1 in the prototype with the appropriate module number, i.e. `I2C1_Init(100000);`

Library Routines

- I2C1_Init
- I2C1_Start
- I2C1_Repeated_Start
- I2C1_Is_Idle
- I2C1_Rd
- I2C1_Wr
- I2C1_Stop

I2C1_Init

Prototype	<code>procedure I2C1_Init(clock : dword);</code>
Returns	Nothing.
Description	Initializes I ² C with desired clock (refer to device data sheet for correct values in respect with Fosc). Needs to be called before using other functions of I ² C Library. You don't need to configure ports manually for using the module; library will take care of the initialization.
Requires	Library requires MSSP module on PORTB or PORTC. Note: Calculation of the I ² C clock value is carried out by the compiler, as it would produce a relatively large code if performed on the library level. Therefore, compiler needs to know the value of the parameter in the compile time. That is why this parameter needs to be a constant, and not a variable.
Example	<code>I2C1_Init(100000);</code>

I2C1_Start

Prototype	<code>function I2C1_Start() : byte;</code>
Returns	If there is no error, function returns 0.
Description	Determines if I ² C bus is free and issues START signal.
Requires	I ₂ C must be configured before using this function. See I2C1_Init.
Example	<pre>if (I2C1_Start() = 0) then ... </pre>

I2C1_Repeated_Start

Prototype	<code>procedure I2C1_Repeated_Start();</code>
Returns	Nothing.
Description	Issues repeated START signal.
Requires	I ² C must be configured before using this function. See I2C1_Init.
Example	<pre>I2C1_Repeated_Start();</pre>

I2C1_Is_Idle

Prototype	<code>function I2C1_Is_Idle() : byte;</code>
Returns	Returns <code>TRUE</code> if I ² C bus is free, otherwise returns <code>FALSE</code> .
Description	Tests if I ² C bus is free.
Requires	I ² C must be configured before using this function. See I2C1_Init.
Example	<pre>if (I2C1_Is_Idle) then ... </pre>

I2C1_Rd

Prototype	<code>function I2C1_Rd(ack : byte) : byte;</code>
Returns	Returns one byte from the slave.
Description	Reads one byte from the slave, and sends not acknowledge signal if parameter <code>ack</code> is 0, otherwise it sends acknowledge.
Requires	I ² C must be configured before using this function. See I2C1_Init. Also, START signal needs to be issued in order to use this function. See I2C1_Start.
Example	Read data and send not acknowledge signal: <pre>tmp := I2C1_Rd(0);</pre>

I2C1_Wr

Prototype	<code>function I2C1_Wr(data : byte) : byte;</code>
Returns	Returns 0 if there were no errors.
Description	Sends data byte (parameter <code>data</code>) via I ² C bus.
Requires	I ² C must be configured before using this function. See I2C1_Init. Also, START signal needs to be issued in order to use this function. See I2C1_Start.
Example	<code>I2C1_Write(\$A3);</code>

I2C1_Stop

Prototype	<code>procedure I2C1_Stop();</code>
Returns	Nothing.
Description	Issues STOP signal.
Requires	I ² C must be configured before using this function. See I2C1_Init.
Example	<code>I2C1_Stop();</code>

Library Example

This code demonstrates use of I²C Library procedures and functions. PIC MCU is connected (SCL, SDA pins) to 24c02 EEPROM. Program sends data to EEPROM (data is written at address 2). Then, we read data via I²C from EEPROM and send its value to PORTD, to check if the cycle was successful. Check the figure below.

```

program I2C_Simple;

begin
  ANSEL  := 0;           // Configure AN pins as digital I/O
  ANSELH := 0;
  PORTB := 0;
  TRISB := 0;           // Configure PORTB as output

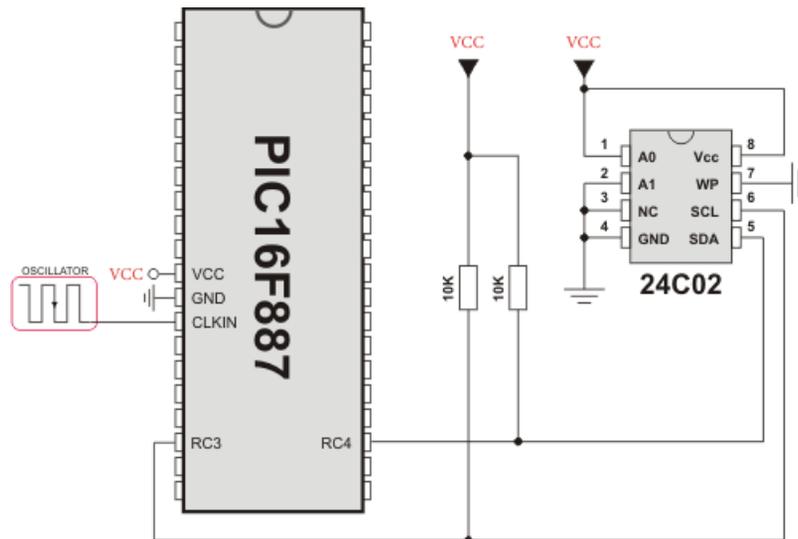
  I2C1_Init(100000);    // initialize I2C communication
  I2C1_Start();         // issue I2C start signal
  I2C1_Wr(0xA2);        // send byte via I2C (device address + W)
  I2C1_Wr(2);           // send byte (address of EEPROM location)
  I2C1_Wr(0xF0);        // send data (data to be written)
  I2C1_Stop();          // issue I2C stop signal

  Delay_100ms();

  I2C1_Start();         // issue I2C start signal
  I2C1_Wr(0xA2);        // send byte via I2C (device address + W)
  I2C1_Wr(2);           // send byte (data address)
  I2C1_Repeated_Start(); // issue I2C signal repeated start
  I2C1_Wr(0xA3);        // send byte (device address + R)
  PORTB := I2C1_Rd(0);  // Read the data (NO acknowledge)
  I2C1_Stop();          // issue I2C stop signal
end.

```

HW Connection



Interfacing 24c02 to PIC via I²C

Keypad Library

The mikroPascal PRO for PIC 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.

External dependencies of Keypad Library

The following variable must be defined in all projects using Keypad Library:	Description:	Example :
<code>var keypadPort : byte; sfr; external;</code>	Keypad Port.	<code>var keypadPort : byte at PORTD;</code>

Library Routines

- Keypad_Init
- Keypad_Key_Press
- Keypad_Key_Click

Keypad_Init

Prototype	<code>procedure Keypad_Init();</code>
Returns	Nothing.
Description	Initializes port for working with keypad.
Requires	Global variables : - <code>keypadPort_Reg</code> - Keypad port must be defined before using this function.
Example	<pre>// Initialize PORTB for communication with keypad var keypadPort : byte at PORTD; ... Keypad_Init();</pre>

Keypad_Key_Press

Prototype	<code>function Keypad_Key_Press(): 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>var kp : byte; ... kp := Keypad_Key_Press();</pre>

Keypad_Key_Click

Prototype	<code>function Keypad_Key_Click(): 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>var kp : 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_Test;

var kp, cnt, oldstate : byte;
    txt : array[ 6] of byte;

// Keypad module connections
var keypadPort : byte at PORTD;
// End Keypad module connections

// Lcd module connections
var LCD_RS : sbit at RB4_bit;
    LCD_EN : sbit at RB5_bit;
    LCD_D4 : sbit at RB0_bit;
    LCD_D5 : sbit at RB1_bit;
    LCD_D6 : sbit at RB2_bit;
    LCD_D7 : sbit at RB3_bit;

var LCD_RS_Direction : sbit at TRISB4_bit;
    LCD_EN_Direction : sbit at TRISB5_bit;
    LCD_D4_Direction : sbit at TRISB0_bit;
    LCD_D5_Direction : sbit at TRISB1_bit;
    LCD_D6_Direction : sbit at TRISB2_bit;
    LCD_D7_Direction : sbit at TRISB3_bit;
// End Lcd module connections

begin
    oldstate := 0;
    cnt := 0; // Reset counter
    Keypad_Init(); // Initialize Keypad
    ANSEL := 0; // Configure AN pins as digital I/O
    ANSELH := 0;
    Lcd_Init(); // Initialize Lcd
    Lcd_Cmd(_LCD_CLEAR); // Clear display
    Lcd_Cmd(_LCD_CURSOR_OFF); // Cursor off
    Lcd_Out(1, 1, 'Key :'); // Write message text on Lcd
    Lcd_Out(2, 1, 'Times:');

while TRUE do
    begin
        kp := 0; // Reset key code variable

// Wait for key to be pressed and released
        while ( kp = 0 ) do
            kp := Keypad_Key_Click(); // Store key code in
// Prepare value for output, transform key to it's ASCII
// value

```

```
case kp of
    //case 10: kp = 42;    // '*'        // Uncomment this block
for keypad4x3
    //case 11: kp = 48;    // '0'
    //case 12: kp = 35;    // '#'
    //default: kp += 48;

        1: kp := 49;    // 1                // Uncomment this
block for keypad4x4
    2: kp := 50;    // 2
    3: kp := 51;    // 3
    4: kp := 65;    // A
    5: kp := 52;    // 4
    6: kp := 53;    // 5
    7: kp := 54;    // 6
    8: kp := 66;    // B
    9: kp := 55;    // 7
    10: kp := 56;    // 8
    11: kp := 57;    // 9
    12: kp := 67;    // C
    13: kp := 42;    // *
    14: kp := 48;    // 0
    15: kp := 35;    // #
    16: kp := 68;    // D

end;

if (kp <> oldstate) then // Pressed key differs from previous
begin
    cnt := 1;
    oldstate := kp;
end
else // Pressed key is same as previous
    Inc(cnt);

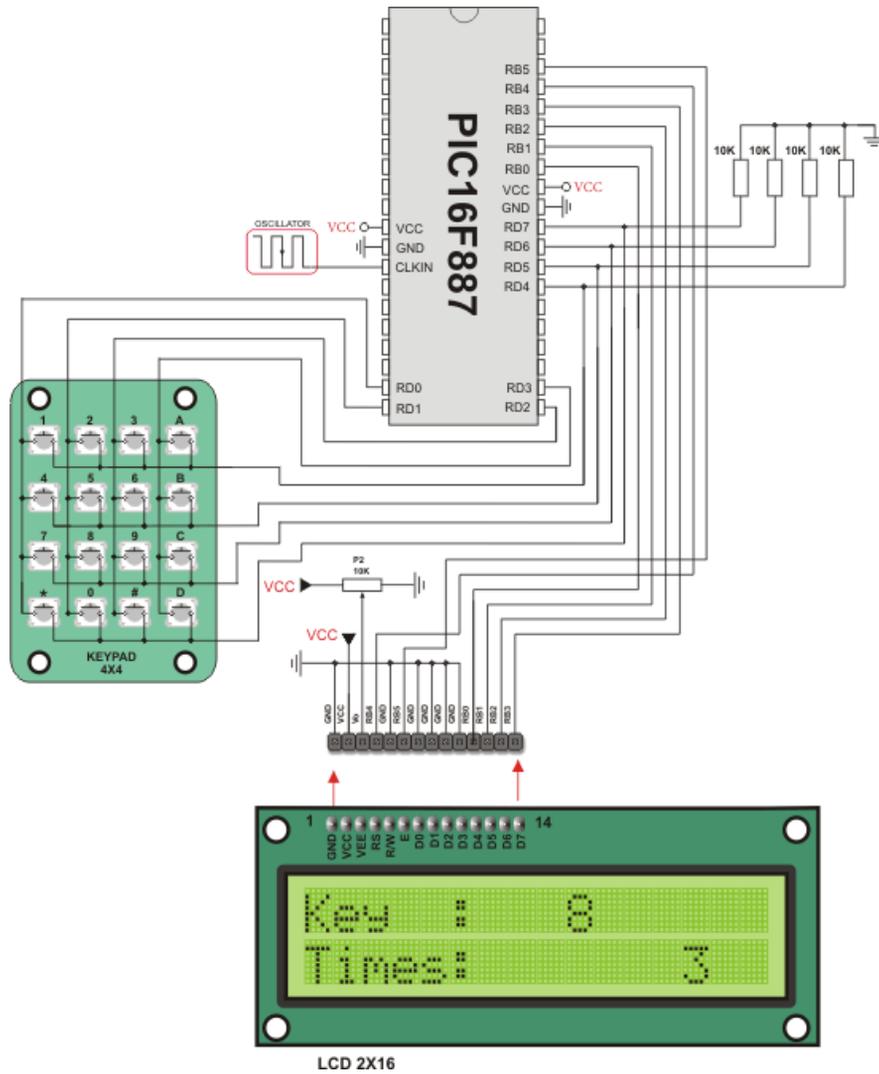
Lcd_Ch(1, 10, kp); // Print key ASCII value on Lcd

if (cnt = 255) then // If counter variable overflow
begin
    cnt := 0;
    Lcd_Out(2, 10, ' ');
end;

WordToStr(cnt, txt); // Transform counter value to string

Lcd_Out(2, 10, txt); // Display counter value on Lcd
end;
end.
```

HW Connection



4x4 Keypad connection scheme

LCD LIBRARY

The mikroPascal PRO for PIC 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>var LCD_RS : sbit; sfr; external;</code>	Register Select line.	<code>var LCD_RS : sbit at RB4_bit;</code>
<code>var LCD_EN : sbit; sfr; external;</code>	Enable line.	<code>var LCD_EN : sbit at RB5_bit;</code>
<code>var LCD_D7 : sbit; sfr; external;</code>	Data 7 line.	<code>var LCD_D7 : sbit at RB3_bit;</code>
<code>var LCD_D6 : sbit; sfr; external;</code>	Data 6 line.	<code>var LCD_D6 : sbit at RB2_bit;</code>
<code>var LCD_D5 : sbit; sfr; external;</code>	Data 5 line.	<code>var LCD_D5 : sbit at RB1_bit;</code>
<code>var LCD_D4 : sbit; sfr; external;</code>	Data 4 line.	<code>var LCD_D4 : sbit at RB0_bit;</code>
<code>var LCD_RS_Direction : sbit; sfr; external;</code>	Register Select direction pin.	<code>var LCD_RS_Direction : sbit at TRISB4_bit;</code>
<code>var LCD_EN_Direction : sbit; sfr; external;</code>	Enable direction pin.	<code>var LCD_EN_Direction : sbit at TRISB5_bit;</code>
<code>var LCD_D7_Direction : sbit; sfr; external;</code>	Data 7 direction pin.	<code>var LCD_D7_Direction : sbit at TRISB3_bit;</code>
<code>var LCD_D6_Direction : sbit; sfr; external;</code>	Data 6 direction pin.	<code>var LCD_D6_Direction : sbit at TRISB2_bit;</code>
<code>var LCD_D5_Direction : sbit; sfr; external;</code>	Data 5 direction pin.	<code>var LCD_D5_Direction : sbit at TRISB1_bit;</code>
<code>var LCD_D4_Direction : sbit; sfr; external;</code>	Data 4 direction pin.	<code>var LCD_D4_Direction : sbit at TRISB0_bit;</code>

Library Routines

Lcd_Init
Lcd_Out
Lcd_Out_CP
Lcd_Chr
Lcd_Chr_Cp
Lcd_Cmd

Lcd_Init

Prototype	<code>procedure Lcd_Init();</code>
Returns	Nothing.
Description	Initializes Lcd module.
Requires	<p>Global variables:</p> <ul style="list-style-type: none"> - LCD_D7: Data bit 7 - LCD_D6: Data bit 6 - LCD_D5: Data bit 5 - LCD_D4: Data bit 4 - LCD_RS: Register Select (data/instruction) signal pin - LCD_EN: Enable signal pin - LCD_D7_Direction: Direction of the Data 7 pin - LCD_D6_Direction: Direction of the Data 6 pin - LCD_D5_Direction: Direction of the Data 5 pin - LCD_D4_Direction: Direction of the Data 4 pin - LCD_RS_Direction: Direction of the Register Select pin - LCD_EN_Direction: Direction of the Enable signal pin <p>must be defined before using this function.</p>
Example	<pre>// Lcd module connections var LCD_RS : sbit at RB4_bit; var LCD_EN : sbit at RB5_bit; var LCD_D4 : sbit at RB3_bit; var LCD_D5 : sbit at RB2_bit; var LCD_D6 : sbit at RB1_bit; var LCD_D7 : sbit at RB0_bit; var LCD_RS_Direction : sbit at TRISB4_bit; var LCD_EN_Direction : sbit at TRISB5_bit; var LCD_D4_Direction : sbit at TRISB3_bit; var LCD_D5_Direction : sbit at TRISB2_bit; var LCD_D6_Direction : sbit at TRISB1_bit; var LCD_D7_Direction : sbit at TRISB0_bit; // End Lcd module connections ... Lcd_Init();</pre>

Lcd_Out

Prototype	<code>procedure Lcd_Out(row: byte; column: byte; var text: array [20] of char);</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	The Lcd module needs to be initialized. See Lcd_Init routine.
Example	<pre>// Write text "Hello!" on Lcd starting from row 1, column 3: Lcd_Out(1, 3, "Hello!");</pre>

Lcd_Out_CP

Prototype	<code>procedure Lcd_Out_CP(var text: array [20] of char);</code>
Returns	Nothing.
Description	<p>Prints text on Lcd at current cursor position. Both string variables and literals can be passed as a text.</p> <p>Parameters :</p> <ul style="list-style-type: none">- <code>text</code>: text to be written
Requires	The Lcd module needs to be initialized. See Lcd_Init routine.
Example	<pre>// Write text "Here!" at current cursor position: Lcd_Out_CP("Here!");</pre>

Lcd_Chr

Prototype	<code>procedure Lcd_Chr(row: byte; column: byte; out_char: byte);</code>
Returns	Nothing.
Description	<p>Prints character on Lcd at specified position. Both variables and literals can be passed as a character.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <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	<pre>// Write character "i" at row 2, column 3: Lcd_Chr(2, 3, 'i');</pre>

Lcd_Chr_CP

Prototype	<code>procedure Lcd_Chr_CP(out_char: byte);</code>
Returns	Nothing.
Description	<p>Prints character on Lcd at current cursor position. Both variables and literals can be passed as a character.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>out_char</code>: character to be written
Requires	The Lcd module needs to be initialized. See Lcd_Init routine.
Example	<pre>// Write character "e" at current cursor position: Lcd_Chr_CP('e');</pre>

Lcd_Cmd

Prototype	<code>procedure Lcd_Cmd(out_char: byte);</code>
Returns	Nothing.
Description	<p>Sends command to Lcd.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>out_char</code>: command to be sent <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
var LCD_RS : sbit at RB4_bit;
var LCD_EN : sbit at RB5_bit;
var LCD_D4 : sbit at RB0_bit;
var LCD_D5 : sbit at RB1_bit;
var LCD_D6 : sbit at RB2_bit;
var LCD_D7 : sbit at RB3_bit;

var LCD_RS_Direction : sbit at TRISB4_bit;
var LCD_EN_Direction : sbit at TRISB5_bit;
var LCD_D4_Direction : sbit at TRISB0_bit;
var LCD_D5_Direction : sbit at TRISB1_bit;
var LCD_D6_Direction : sbit at TRISB2_bit;
var LCD_D7_Direction : sbit at TRISB3_bit;
// End LCD module connections

var txt1 : array[ 16] of char;
    txt2 : array[ 9]  of char;
    txt3 : array[ 8]  of char;
    txt4 : array[ 7]  of char;
    i    : byte;                                // Loop variable

procedure Move_Delay(); // Function used for text moving
begin
    Delay_ms(500);      // You can change the moving speed here
end;

begin

    TRISB := 0;
    PORTB := 0xFF;
    TRISB := 0xFF;
    ANSEL := 0;          // Configure AN pins as digital I/O
    ANSELH := 0;

    txt1 := 'mikroElektronika';
    txt2 := 'EasyPIC5';
    txt3 := 'Lcd4bit';
    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(2,5,txt2);           // Write text in second row
Delay_ms(500);

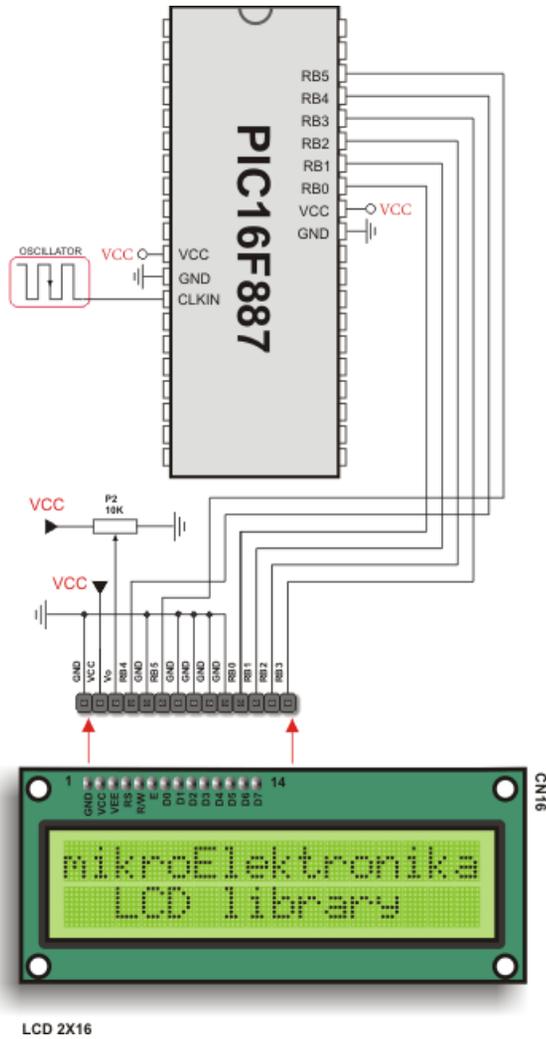
// Moving text
for i:=0 to 3 do             // Move text to the right 4 times
begin
    Lcd_Cmd(_LCD_SHIFT_RIGHT);
    Move_Delay();
end;

while TRUE do               // Endless loop
begin
    for i:=0 to 6 do         // Move text to the left 7 times
begin
    Lcd_Cmd(_LCD_SHIFT_LEFT);
    Move_Delay();
end;

    for i:=0 to 6 do         // Move text to the right 7 times
begin
    Lcd_Cmd(_LCD_SHIFT_RIGHT);
    Move_Delay();
end;

end;
end.
```

HW connection

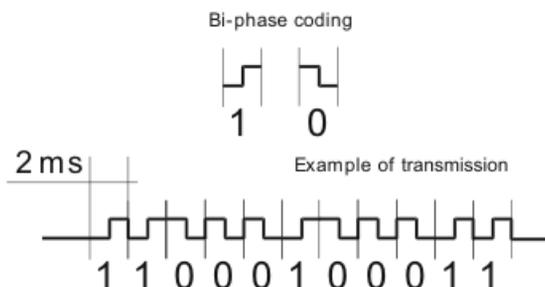
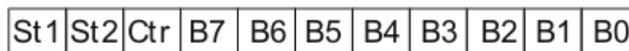


Lcd HW connection

MANCHESTER CODE LIBRARY

The mikroPascal PRO for PIC 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



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).

Note: Manchester code library implements time-based activities, so interrupts need to be disabled when using it.

External dependencies of Manchester Code Library

The following variables must be defined in all projects using Manchester Code Library:	Description:	Example :
<code>var MANRXPIN : sbit; sfr; external;</code>	Receive line.	<code>var MANRXPIN : sbit at RB0_bit;</code>
<code>var MANTXPIN : sbit; sfr; external;</code>	Transmit line.	<code>var MANTXPIN : sbit at RB1_bit;</code>
<code>var MANRXPIN_Direction : sbit; sfr; external;</code>	Direction of the Receive pin.	<code>var MANRXPIN_Direction : sbit at TRISB0_bit;</code>
<code>var MANTXPIN_Direction : sbit; sfr; external;</code>	Direction of the Transmit pin.	<code>var MANTXPIN_Direction : sbit at TRISB1_bit;</code>

Library Routines

- Man_Receive_Init
- Man_Receive
- Man_Send_Init
- Man_Send
- Man_Synchro
- Man_Break

The following routines are for the internal use by compiler only:

- Manchester_0
- Manchester_1
- Manchester_Out

Man_Receive_Init

Prototype	<code>function Man_Receive_Init(): word;</code>
Returns	<ul style="list-style-type: none"> - 0 - if initialization and synchronization were successful. - 1 - upon unsuccessful synchronization. - 255 - upon user abort.
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	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>MANRXPIN</code> : Receive line - <code>MANRXPIN_Direction</code> : Direction of the receive pin <p>must be defined before using this function.</p>
Example	<pre>// Initialize Receiver var MANRXPIN : sbit at RB0_bit; var MANRXPIN_Direction : sbit at TRISB0_bit; ... Man_Receive_Init();</pre>

Man_Receive

Prototype	<code>function Man_Receive(var error: byte): byte;</code>
Returns	A byte read from the incoming signal.
Description	The function extracts one byte from incoming signal. Parameters : - <code>error</code> : error flag. If signal format does not match the expected, the error flag will be set to non-zero.
Requires	To use this function, the user must prepare the MCU for receiving. See <code>Man_Receive_Init</code> .
Example	<pre> var data, error : byte ... data := 0 error := 0 data := Man_Receive(&error); if (error <> 0) then begin // error handling end; </pre>

Man_Send_Init

Prototype	<code>procedure Man_Send_Init();</code>
Returns	Nothing.
Description	The function configures Transmitter pin.
Requires	Global variables : - <code>MANRXPIN</code> : Receive line - <code>MANRXPIN_Direction</code> : Direction of the receive pin must be defined before using this function.
Example	<pre> // Initialize Transmitter: var MANTXPIN : sbit at RB1_bit; var MANTXPIN_Direction : sbit at TRISB1_bit; ... Man_Send_Init(); </pre>

Man_Send

Prototype	<code>procedure Man_Send(tr_data: byte);</code>
Returns	Nothing.
Description	<p>Sends one byte.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>tr_data</code>: data to be sent <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>var msg : byte; ... Man_Send(msg);</pre>

Man_Synchro

Prototype	<code>function Man_Synchro(): byte;</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 <code>Man_Receive_Init</code> .
Example	<pre>var man_half_bit_len : word; ... man_half_bit_len := Man_Synchro();</pre>

Man_Break

Prototype	<code>procedure Man_Break();</code>
Returns	Nothing.
Description	<p>Man_Receive is blocking routine and it can block the program flow. Call this routine from interrupt to unblock the program execution. This mechanism is similar to WDT.</p> <p>Note: Interrupts should be disabled before using Manchester routines again (see note at the top of this page).</p>
Requires	Nothing.
Example	<pre> var datal, error, counter : byte; procedure interrupt(); begin if (INTCON.TOIF <> 0) then if (counter >= 20) then begin Man_Break(); counter := 0; // reset counter end else Inc(counter); // increment counter INTCON.TOIF := 0; // Clear Timer0 overflow inter- rupt flag end; begin counter := 0; OPTION_REG := 0x04; // TMR0 prescaler set to 1:32 ... Man_Receive_Init(); ... // try Man_Receive with blocking prevention mechanism INTCON.GIE := 1; // Global interrupt enable INTCON.TOIE := 1; // Enable Timer0 overflow inter- rupt datal := Man_Receive(error); INTCON.GIE := 0; // Global interrupt disable end. </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
var LCD_RS : sbit at RB4_bit;
    LCD_EN : sbit at RB5_bit;
    LCD_D4 : sbit at RB0_bit;
    LCD_D5 : sbit at RB1_bit;
    LCD_D6 : sbit at RB2_bit;
    LCD_D7 : sbit at RB3_bit;

var LCD_RS_Direction : sbit at TRISB4_bit;
    LCD_EN_Direction : sbit at TRISB5_bit;
    LCD_D4_Direction : sbit at TRISB0_bit;
    LCD_D5_Direction : sbit at TRISB1_bit;
    LCD_D6_Direction : sbit at TRISB2_bit;
    LCD_D7_Direction : sbit at TRISB3_bit;
// End LCD module connections

// Manchester module connections
var MANRXPIN : sbit at RB0_bit;
    MANRXPIN_Direction : sbit at TRISB0_bit;
    MANTXPIN : sbit at RB1_bit;
    MANTXPIN_Direction : sbit at TRISB1_bit;
// End Manchester module connections

var error, ErrorCount, temp : byte;

begin
    ErrorCount := 0;
    ANSEL := 0;           // Configure AN pins as digital I/O
    ANSELH := 0;
    C1ON_bit := 0;       // Disable comparators
    C2ON_bit := 0;
    Lcd_Init();          // Initialize LCD
    Lcd_Cmd(_LCD_CLEAR); // Clear LCD display

    Man_Receive_Init(); // Initialize Receiver

    while TRUE do       // Endless loop
        begin
            Lcd_Cmd(_LCD_FIRST_ROW); // Move cursor to the 1st row

            while TRUE do // Wait for the "start" byte
                begin
```

```

temp := Man_Receive(error);    // Attempt byte receive
if (temp = 0x0B) then         // "Start" byte, see
Transmitter example
    break;                    // We got the starting sequence
if (error <> 0) then          // Exit so we do not loop forever
    break;
end;

repeat
begin
temp := Man_Receive(error);    // Attempt byte receive
if (error <> 0) then           // If error occurred
begin
    Lcd_Chr_CP('?');          // Write question mark on LCD
    Inc(ErrorCount);         // Update error counter
    if (ErrorCount > 20) then // In case of multi-
ple errors
begin
again
temp := Man_Synchro();        // Try to synchronize
//Man_Receive_Init();       // Alternative, try to
Initialize Receiver again
ErrorCount := 0;             // Reset error counter
end;
end
else                           // No error occurred
begin
if (temp <> 0x0E) then         // If "End" byte was
received(see Transmitter example)
    Lcd_Chr_CP(temp);        // do not write received
byte on LCD
end;
Delay_ms(25);
end;
until ( temp = 0x0E );
end;                            // If "End" byte was received exit do loop
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
var MANRXPIN : sbit at RBO_bit;
    MANRXPIN_Direction : sbit at TRISB0_bit;
    MANTXPIN : sbit at RB1_bit;
    MANTXPIN_Direction : sbit at TRISB1_bit;
// End Manchester module connections

var index, character : byte;
    s1 : array[17] of char;

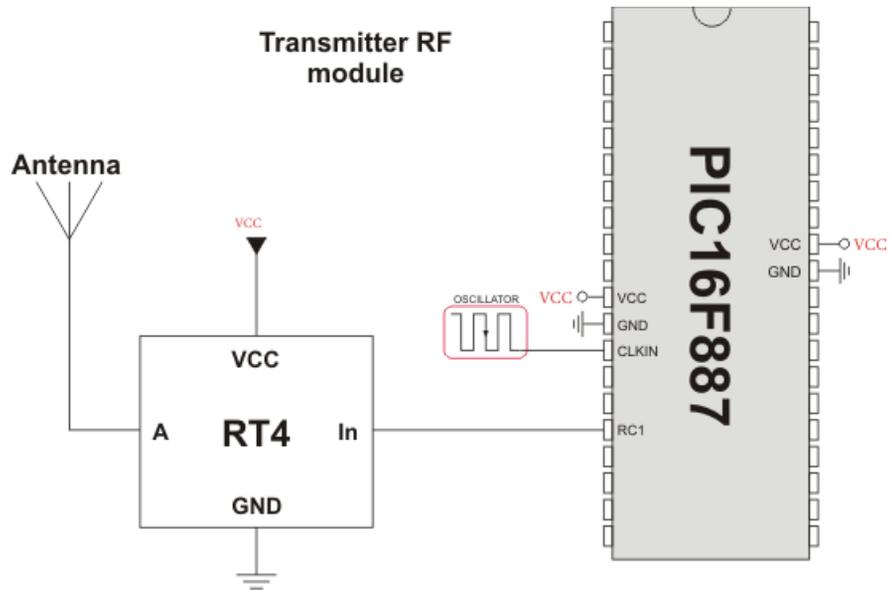
begin
    s1 := 'mikroElektronika';
    ANSEL := 0; // Configure AN pins as digital I/O
    ANSELH := 0;
    C1ON_bit := 0; // Disable comparators
    C2ON_bit := 0;

    Man_Send_Init(); // Initialize transmitter

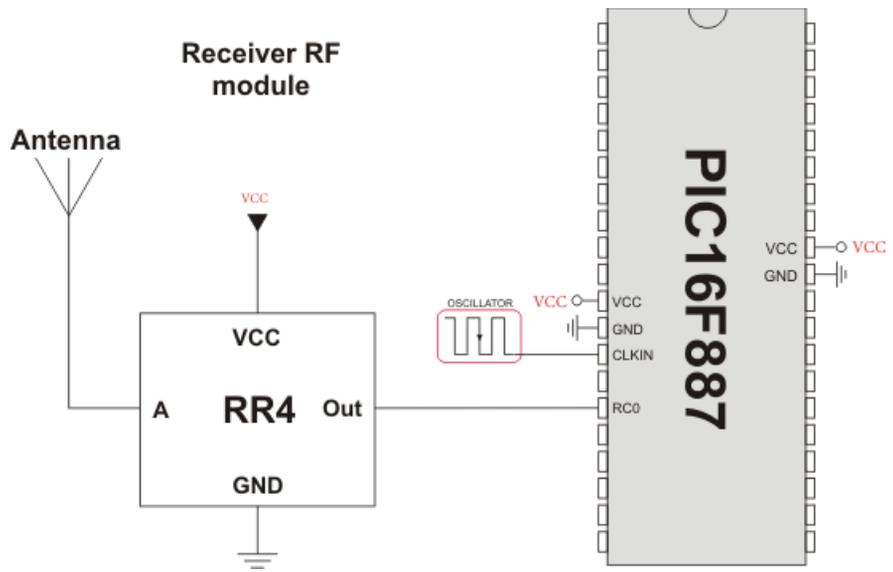
    while TRUE do // Endless loop
        begin
            Man_Send(0x0B); // Send "start" byte
            Delay_ms(100); // Wait for a while

            character := s1[0]; // Take first char from string
            index := 0; // Initialize index variable
            while (character <> 0) do // String ends with zero
                begin
                    Man_Send(character); // Send character
                    Delay_ms(90); // Wait for a while
                    Inc(index); // Increment index variable
                    character := s1[index]; // Take next char from string
                end;
            Man_Send(0x0E); // Send "end" byte
            Delay_ms(1000);
        end;
    end.
```

Connection Example



Simple Transmitter connection



Simple Receiver connection

MULTI MEDIA CARD LIBRARY

The Multi Media Card (MMC) is a flash memory card standard. MMC cards are currently available in sizes up to and including 1 GB, and are used in cell phones, mp3 players, digital cameras, and PDA's.

mikroPascal PRO for PIC provides a library for accessing data on Multi Media Card via SPI communication. This library also supports SD(Secure Digital) memory cards.

Secure Digital Card

Secure Digital (SD) is a flash memory card standard, based on the older Multi Media Card (MMC) format.

SD cards are currently available in sizes of up to and including 2 GB, and are used in cell phones, mp3 players, digital cameras, and PDAs.

Notes:

- Library works with PIC18 family only;
- The library uses the SPI module for communication. User must initialize SPI module before using the SPI Graphic Lcd Library.
For MCUs with two SPI modules it is possible to initialize both of them and then switch by using the `SPI_Set_Active()` routine.
- Routines for file handling can be used only with FAT16 file system.
- Library functions create and read files from the root directory only;
- Library functions populate both FAT1 and FAT2 tables when writing to files, but the file data is being read from the FAT1 table only; i.e. there is no recovery if FAT1 table is corrupted.

Note: The SPI module has to be initialized through `SPI1_Init_Advanced` routine with the following parameters:

- SPI Master
- 8bit mode
- primary prescaler 16
- Slave Select disabled
- data sampled in the middle of data output time
- clock idle low
- Serial output data changes on transition from idle clock state to active clock state

`SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV16, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);` must be called before initializing `Mmc_Init`.

Note: Once the MMC/SD card is initialized, the user can reinitialize SPI at higher speed. See the `Mmc_Init` and `Mmc_Fat_Init` routines.

External dependencies of MMC Library

The following variable must be defined in all projects using MMC library:	Description:	Example :
<code>var Mmc_Chip_Select : sbit; sfr; external;</code>	Chip select pin.	<code>var Mmc_Chip_Select : sbit at RC2_bit;</code>
<code>var Mmc_Chip_Select_Direction : sbit; sfr; external;</code>	Direction of the chip select pin.	<code>var Mmc_Chip_Select_Direction : sbit at TRISCO_bit;</code>

Library Routines

- Mmc_Init
- Mmc_Read_Sector
- Mmc_Write_Sector
- Mmc_Read_Cid
- Mmc_Read_Csd

Routines for file handling:

- Mmc_Fat_Init
- Mmc_Fat_QuickFormat
- Mmc_Fat_Assign
- Mmc_Fat_Reset
- Mmc_Fat_Read
- Mmc_Fat_Rewrite
- Mmc_Fat_Append
- Mmc_Fat_Delete
- Mmc_Fat_Write
- Mmc_Fat_Set_File_Date
- Mmc_Fat_Get_File_Date
- Mmc_Fat_Get_File_Size
- Mmc_Fat_Get_Swap_File

Mmc_Init

Prototype	<code>function Mmc_Init(): byte;</code>
Returns	- 0 - if MMC/SD card was detected and successfully initialized - 1 - otherwise
Description	Initializes MMC through hardware SPI interface. Parameters: - <code>port</code> : chip select signal port address. - <code>cspin</code> : chip select pin.
Requires	Global variables : - <code>Mmc_Chip_Select</code> : Chip Select line - <code>Mmc_Chip_Select_Direction</code> : Direction of the Chip Select pin must be defined before using this function. The appropriate hardware SPI module must be previously initialized. See the <code>SPI1_Init</code> , <code>SPI1_Init_Advanced</code> routines.
Example	<pre>// MMC module connections var Mmc_Chip_Select : sbit at RC2_bit; var Mmc_Chip_Select_Direction : sbit at TRISC2_bit; // MMC module connections error = Mmc_Init(); // Init with CS line at RC.2 var i : byte; ... SPI1_Init(); i = Mmc_Init();</pre>

Mmc_Read_Sector

Prototype	<code>function Mmc_Read_Sector(sector: longint; var dbuff: array[512] of byte): byte;</code>
Returns	- 0 - if reading was successful - 1 - if an error occurred
Description	The function reads one sector (512 bytes) from MMC card. Parameters: - <code>sector</code> : MMC/SD card sector to be read. - <code>data</code> : buffer of minimum 512 bytes in length for data storage.
Requires	MMC/SD card must be initialized. See <code>Mmc_Init</code> .
Example	<pre>// read sector 510 of the MMC/SD card var error : byte; sectorNo : longint; dataBuffer : array[512] of byte; ... sectorNo := 510; error := Mmc_Read_Sector(sectorNo, dataBuffer);</pre>

Mmc_Write_Sector

Prototype	<code>function Mmc_Write_Sector(sector: longint; var data_: array[512] of byte): byte;</code>
Returns	- 0 - if writing was successful - 1 - if there was an error in sending write command - 2 - if there was an error in writing (data rejected)
Description	The function writes 512 bytes of data to one MMC card sector. Parameters: - <code>sector</code> : MMC/SD card sector to be written to. - <code>data</code> : data to be written (buffer of minimum 512 bytes in length).
Requires	MMC/SD card must be initialized. See <code>Mmc_Init</code> .
Example	<pre>// write to sector 510 of the MMC/SD card var error : byte; sectorNo : longint; dataBuffer : array[512] of byte; ... sectorNo := 510; error := Mmc_Write_Sector(sectorNo, dataBuffer);</pre>

Mmc_Read_Cid

Prototype	<code>function Mmc_Read_Cid(var data_cid: array[16] of byte): byte;</code>
Returns	- 0 - if CID register was read successfully - 1 - if there was an error while reading
Description	The function reads 16-byte CID register. Parameters: - <code>data_cid</code> : buffer of minimum 16 bytes in length for storing CID register content.
Requires	MMC/SD card must be initialized. See <code>Mmc_Init</code> .
Example	<pre>var error : byte; dataBuffer : array[16] of byte; ... error := Mmc_Read_Cid(dataBuffer);</pre>

Mmc_Read_Csd

Prototype	<code>function Mmc_Read_Csd(var data_for_registers: array[16] of byte): byte;</code>
Returns	- 0 - if CSD register was read successfully - 1 - if there was an error while reading
Description	The function reads 16-byte CSD register. Parameters: - <code>data_for_registers</code> : buffer of minimum 16 bytes in length for storing CSD register content.
Requires	MMC/SD card must be initialized. See <code>Mmc_Init</code> .
Example	<pre>var error : word; data_for_registers : array[16] of byte; ... error := Mmc_Read_Csd(data_for_registers);</pre>

Mmc_Fat_Init

Prototype	<code>function Mmc_Fat_Init(): byte;</code>
Returns	<ul style="list-style-type: none"> - 0 - if MMC/SD card was detected and successfully initialized - 1 - if FAT16 boot sector was not found - 255 - if MMC/SD card was not detected
Description	<p>Initializes MMC/SD card, reads MMC/SD FAT16 boot sector and extracts necessary data needed by the library.</p> <p>Note: MMC/SD card has to be formatted to FAT16 file system.</p>
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>Mmc_Chip_Select</code>: Chip Select line - <code>Mmc_Chip_Select_Direction</code>: Direction of the Chip Select pin <p>must be defined before using this function. The appropriate hardware SPI module must be previously initialized. See the <code>SPI1_Init</code>, <code>SPI1_Init_Advanced</code> routines.</p>
Example	<pre>// MMC module connections var Mmc_Chip_Select : sbit at RC2_bit; var Mmc_Chip_Select_Direction : sbit at TRISC2_bit; // MMC module connections SPI1_Init(); // init the FAT library if (Mmc_Fat_Init() = 0) then begin SPI1_Init(); ... end</pre>

Mmc_Fat_QuickFormat

Prototype	<code>function Mmc_Fat_QuickFormat(var mmc_fat_label : string[11]) : byte;</code>
Returns	<ul style="list-style-type: none"> - 0 - if MMC/SD card was detected, successfully formatted and initialized - 1 - if FAT16 format was unseccessful - 255 - if MMC/SD card was not detected
Description	<p>Formats to FAT16 and initializes MMC/SD card.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>port</code>: chip select signal port address. - <code>pin</code>: chip select pin. - <code>mmc_fat_label</code>: volume label (11 characters in length). If less than 11 characters are provided, the label will be padded with spaces. If an empty string is passed, the volume will not be labeled. <p>Note: This routine can be used instead or in conjunction with the <code>Mmc_Fat_Init</code> routine.</p> <p>Note: If MMC/SD card already contains a valid boot sector, it will remain unchanged (except volume label field) and only FAT and ROOT tables will be erased. Also, the new volume label will be set.</p>
Requires	The appropriate hardware SPI module must be previously initialized.
Example	<pre>// format and initialize the FAT library if (Mmc_Fat_QuickFormat('mikroE') = 0) then begin ... end;</pre>

Mmc_Fat_Assign

Prototype	<code>function Mmc_Fat_Assign(var filename: array[12] of char; file_cre_attr: byte): byte;</code>																												
Returns	<ul style="list-style-type: none"> - 1 - if file already exists or file does not exist but a new file is created. - 0 - if file does not exist and no new file is created. 																												
Description	<p>Assigns file for file operations (read, write, delete...). All subsequent file operations will be applied on an assigned file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>filename</code>: name of the file that should be assigned for file operations. File name should be in DOS 8.3 (<code>file_name.extension</code>) format. The file name and extension will be automatically padded with spaces by the library if they have less than length required (i.e. "mikro.tx" -> "mikro .tx "), so the user does not have to take care of that. The file name and extension are case insensitive. The library will convert them to proper case automatically, so the user does not have to take care of that. Also, in order to keep backward compatibility with the first version of this library, file names can be entered as UPPERCASE string of 11 bytes in length with no dot character between file name and extension (i.e. "MIKROELETXT" -> MIKROELE.TXT). In this case last 3 characters of the string are considered to be file extension. - <code>file_cre_attr</code>: file creation and attributs flags. Each bit corresponds to the appropriate file attribut: <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">Bit</th> <th style="width: 15%;">Mask</th> <th style="width: 75%;">Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0x01</td> <td>Read Only</td> </tr> <tr> <td>1</td> <td>0x02</td> <td>Hidden</td> </tr> <tr> <td>2</td> <td>0x04</td> <td>System</td> </tr> <tr> <td>3</td> <td>0x08</td> <td>Volume Label</td> </tr> <tr> <td>4</td> <td>0x10</td> <td>Subdirectory</td> </tr> <tr> <td>5</td> <td>0x20</td> <td>Archive</td> </tr> <tr> <td>6</td> <td>0x40</td> <td>Device (internal use only, never found on disk)</td> </tr> <tr> <td>7</td> <td>0x80</td> <td>File creation flag. If the file does not exist and this flag is set, a new file with specified name will be created.</td> </tr> </tbody> </table> <p>Note: Long File Names (LFN) are not supported.</p>		Bit	Mask	Description	0	0x01	Read Only	1	0x02	Hidden	2	0x04	System	3	0x08	Volume Label	4	0x10	Subdirectory	5	0x20	Archive	6	0x40	Device (internal use only, never found on disk)	7	0x80	File creation flag. If the file does not exist and this flag is set, a new file with specified name will be created.
Bit	Mask	Description																											
0	0x01	Read Only																											
1	0x02	Hidden																											
2	0x04	System																											
3	0x08	Volume Label																											
4	0x10	Subdirectory																											
5	0x20	Archive																											
6	0x40	Device (internal use only, never found on disk)																											
7	0x80	File creation flag. If the file does not exist and this flag is set, a new file with specified name will be created.																											
Requires	MMC/SD card and MMC library must be initialized for file operations. See <code>Mmc_Fat_Init</code> .																												
Example	<pre>// create file with archive attribut if it does not already exist Mmc_Fat_Assign('MIKRO007.TXT',0xA0);</pre>																												

Mmc_Fat_Reset

Prototype	<code>procedure Mmc_Fat_Reset(var size: dword);</code>
Returns	Nothing.
Description	Opens currently assigned file for reading. Parameters: - <code>size</code> : buffer to store file size to. After file has been open for reading, its size is returned through this parameter.
Requires	MMC/SD card and MMC library must be initialized for file operations. See <code>Mmc_Fat_Init</code> . The file must be previously assigned. See <code>Mmc_Fat_Assign</code> .
Example	<pre>var size : dword; ... Mmc_Fat_Reset(size);</pre>

Mmc_Fat_Read

Prototype	<code>procedure Mmc_Fat_Read(var bdata: byte);</code>
Returns	Nothing.
Description	Reads a byte from the currently assigned file opened for reading. Upon function execution, file pointers will be set to the next character in the file. Parameters: - <code>bdata</code> : buffer to store read byte to. Upon this function execution read byte is returned through this parameter.
Requires	MMC/SD card and MMC library must be initialized for file operations. See <code>Mmc_Fat_Init</code> . The file must be previously assigned. See <code>Mmc_Fat_Assign</code> . The file must be opened for reading. See <code>Mmc_Fat_Reset</code> .
Example	<pre>var character : byte; ... Mmc_Fat_Read(character);</pre>

Mmc_Fat_Rewrite

Prototype	<code>procedure Mmc_Fat_Rewrite();</code>
Returns	Nothing.
Description	Opens the currently assigned file for writing. If the file is not empty its content will be erased.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign.
Example	<pre>// open file for writing Mmc_Fat_Rewrite();</pre>

Mmc_Fat_Append

Prototype	<code>procedure Mmc_Fat_Append();</code>
Returns	Nothing.
Description	Opens the currently assigned file for appending. Upon this function execution file pointers will be positioned after the last byte in the file, so any subsequent file writing operation will start from there.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign.
Example	<pre>// open file for appending Mmc_Fat_Append();</pre>

Mmc_Fat_Delete

Prototype	<code>procedure Mmc_Fat_Delete();</code>
Returns	Nothing.
Description	Deletes currently assigned file from MMC/SD card.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign.
Example	<pre>// delete current file Mmc_Fat_Delete();</pre>

Mmc_Fat_Write

Prototype	<code>procedure Mmc_Fat_Write(var fdata: array[512] of byte; data_len: word);</code>
Returns	Nothing.
Description	Writes requested number of bytes to the currently assigned file opened for writing. Parameters: - <code>fdata</code> : data to be written. - <code>data_len</code> : number of bytes to be written.
Requires	MMC/SD card and MMC library must be initialized for file operations. See <code>Mmc_Fat_Init</code> . The file must be previously assigned. See <code>Mmc_Fat_Assign</code> . The file must be opened for writing. See <code>Mmc_Fat_Rewrite</code> or <code>Mmc_Fat_Append</code> .
Example	<code>var file_contents : array[42] of byte; ... Mmc_Fat_Write(file_contents, 42); // write data to the assigned file</code>

Mmc_Fat_Set_File_Date

Prototype	<code>procedure Mmc_Fat_Set_File_Date(year: word; month: byte; day: byte; hours: byte; mins: byte; seconds: byte);</code>
Returns	Nothing.
Description	<p>Sets the date/time stamp. Any subsequent file writing operation will write this stamp to the currently assigned file's time/date attributs.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>year</code>: year attribute. Valid values: 1980-2107 - <code>month</code>: month attribute. Valid values: 1-12 - <code>day</code>: day attribute. Valid values: 1-31 - <code>hours</code>: hours attribute. Valid values: 0-23 - <code>mins</code>: minutes attribute. Valid values: 0-59 - <code>seconds</code>: seconds attribute. Valid values: 0-59
Requires	<p>MMC/SD card and MMC library must be initialized for file operations. See <code>Mmc_Fat_Init</code>.</p> <p>The file must be previously assigned. See <code>Mmc_Fat_Assign</code>.</p> <p>The file must be opened for writing. See <code>Mmc_Fat_Rewrite</code> or <code>Mmc_Fat_Append</code>.</p>
Example	<code>Mmc_Fat_Set_File_Date(2005,9,30,17,41,0);</code>

Mmc_Fat_Get_File_Date

Prototype	<code>procedure Mmc_Fat_Get_File_Date(var year: word; var month: byte; var day: byte; var hours: byte; var mins: byte);</code>
Returns	Nothing.
Description	<p>Reads time/date attributes of the currently assigned file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>year</code>: buffer to store year attribute to. Upon function execution year attribute is returned through this parameter. - <code>month</code>: buffer to store month attribute to. Upon function execution month attribute is returned through this parameter. - <code>day</code>: buffer to store day attribute to. Upon function execution day attribute is returned through this parameter. - <code>hours</code>: buffer to store hours attribute to. Upon function execution hours attribute is returned through this parameter. - <code>mins</code>: buffer to store minutes attribute to. Upon function execution minutes attribute is returned through this parameter.
Requires	<p>MMC/SD card and MMC library must be initialized for file operations. See <code>Mmc_Fat_Init</code>.</p> <p>The file must be previously assigned. See <code>Mmc_Fat_Assign</code>.</p>
Example	<pre>var year : word; month, day, hours, mins : byte; ... Mmc_Fat_Get_File_Date(year, month, day, hours, mins);</pre>

Mmc_Fat_Get_File_Size

Prototype	<code>function Mmc_Fat_Get_File_Size(): dword;</code>
Returns	Size of the currently assigned file in bytes.
Description	This function reads size of the currently assigned file in bytes.
Requires	<p>MMC/SD card and MMC library must be initialized for file operations. See <code>Mmc_Fat_Init</code>.</p> <p>The file must be previously assigned. See <code>Mmc_Fat_Assign</code>.</p>
Example	<pre>var my_file_size : dword; ... my_file_size := Mmc_Fat_Get_File_Size();</pre>

Mmc_Fat_Get_Swap_File

Prototype	<pre>function Mmc_Fat_Get_Swap_File(sectors_cnt: dword; var filename : string[11]; file_attr : byte) : dword;</pre>
Returns	<ul style="list-style-type: none"> - Number of the start sector for the newly created swap file, if there was enough free space on the MMC/SD card to create file of required size. - 0 - otherwise.
Description	<p>This function is used to create a swap file of predefined name and size on the MMC/SD media. If a file with specified name already exists on the media, search for consecutive sectors will ignore sectors occupied by this file. Therefore, it is recommended to erase such file if it already exists before calling this function. If it is not erased and there is still enough space for a new swap file, this function will delete it after allocating new memory space for a new swap file.</p> <p>The purpose of the swap file is to make reading and writing to MMC/SD media as fast as possible, by using the Mmc_Read_Sector() and Mmc_Write_Sector() functions directly, without potentially damaging the FAT system. The swap file can be considered as a "window" on the media where the user can freely write/read data. Its main purpose in the mikroPascal's library is to be used for fast data acquisition; when the time-critical acquisition has finished, the data can be re-written into a "normal" file, and formatted in the most suitable way.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>sectors_cnt</code>: number of consecutive sectors that user wants the swap file to have. - <code>filename</code>: name of the file that should be assigned for file operations. File name should be in DOS 8.3 (file_name.extension) format. The file name and extension will be automatically padded with spaces by the library if they have less than length required (i.e. "mikro.tx" -> "mikro .tx "), so the user does not have to take care of that. The file name and extension are case insensitive. The library will convert them to proper case automatically, so the user does not have to take care of that. <p>Also, in order to keep backward compatibility with the first version of this library, file names can be entered as UPPERCASE string of 11 bytes in length with no dot character between file name and extension (i.e. "MIKROELETXT" -> MIKROELE.TXT). In this case the last 3 characters of the string are considered to be file extension.</p> <ul style="list-style-type: none"> - <code>file_attr</code>: file creation and attributes flags. Each bit corresponds to the appropriate file attribute:

Description	Bit	Mask	Description
	0	0x01	Read Only
	1	0x02	Hidden
	2	0x04	System
	3	0x08	Volume Label
	4	0x10	Subdirectory
	5	0x20	Archive
	6	0x40	Device (internal use only, never found on disk)
	7	0x80	Not used

Note: Long File Names (LFN) are not supported.

Requires	
	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init.

Example	
	<pre>//----- Try to create a swap file with archive attribute, // whose size will be at least 1000 sectors. // If it succeeds, it sends No. of start sector // over USART var size : dword; ... size := Mmc_Fat_Get_Swap_File(1000, 'mikroE.txt', 0x20); if (size <> 0) then begin UART1_Write(0xAA); UART1_Write(Lo(size)); UART1_Write(Hi(size)); UART1_Write(Higher(size)); UART1_Write(Highest(size)); UART1_Write(0xAA); end;</pre>

Library Example

The following example demonstrates MMC library test. Upon flashing, insert a MMC/SD card into the module, when you should receive the "Init-OK" message. Then, you can experiment with MMC read and write functions, and observe the results through the Usart Terminal.

```
program MMC_Test;

{ $DEFINE RS232_debug}

var MMC_chip_select : sbit at RC2_bit;
var MMC_chip_select_direction : sbit at TRISC2_bit;

const FILL_CHAR = 'm';
var i, SectorNo : word;
var mmc_error : byte;
var data_ok : bit;

// Variables for MMC routines
SectorData : array[512] of byte; // Buffer for MMC sector
reading/writing
data_for_registers : array[16] of byte; // buffer for CID and CSD
registers

// UART write text and new line (carriage return + line feed)
procedure UART_Write_Line( var uart_text : byte );
begin
    UART1_Write_Text(uart_text);
    UART1_Write(13);
    UART1_Write(10);
end;

// Display byte in hex
// Display byte in hex
procedure printhex( i : byte );
var high, low : byte;
begin
    high := i and 0xF0;           // High nibble
    high := high shr 4;
    high := high + '0';
    if ( high > '9' ) then
        high := high + 7;
    low := (i and 0x0F) + '0';    // Low nibble
    if ( low > '9' ) then
        low := low + 7 ;

    UART1_Write(high);
    UART1_Write(low);
end;

begin

ADCON1 := ADCON1 or 0x0F;       // Configure AN pins as digital
CMCON := CMCON or 7;           // Turn off comparators
```

```
// Initialize UART1 module
UART1_Init(19200);
Delay_ms(10);

UART_Write_Line('PIC-Started'); // PIC present report

// Initialize SPI1 module
SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV64, _SPI_DATA_SAMPLE_MIDDLE,
_SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);

// initialise a MMC card
mmc_error := Mmc_Init();
if ( mmc_error = 0 ) then
    UART_Write_Line('MMC Init-OK') // If MMC present report
else
    UART_Write_Line('MMC Init-error'); // If error report

// Fill MMC buffer with same characters
for i := 0 to 511 do
    SectorData[i] := FILL_CHAR;

// Write sector
mmc_error := Mmc_Write_Sector(SectorNo, SectorData);
if ( mmc_error = 0 ) then
    UART_Write_Line('Write-OK')
else // if there are errors.....
    UART_Write_Line('Write-Error');

// Reading of CID register
mmc_error := Mmc_Read_Cid(data_for_registers);
if ( mmc_error = 0 ) then
    begin
        UART1_Write_Text('CID : ');
        for i := 0 to 15 do
            printhex(data_for_registers[i]);
        UART_Write_Line(' ');
    end
else
    UART_Write_Line('CID-error');

// Reading of CSD register
mmc_error := Mmc_Read_Csd(data_for_registers);
if ( mmc_error = 0 ) then
    begin
        UART1_Write_Text('CSD : ');
        for i := 0 to 15 do
            printhex(data_for_registers[i]);
        UART_Write_Line(' ');
    end
else
```

```

UART_Write_Line('CSD-error');

// Read sector
mmc_error := Mmc_Read_Sector(SectorNo, SectorData);
if ( mmc_error = 0 ) then
    UART_Write_Line('Read-OK')
else // if there are errors.....
    UART_Write_Line('Read-Error');

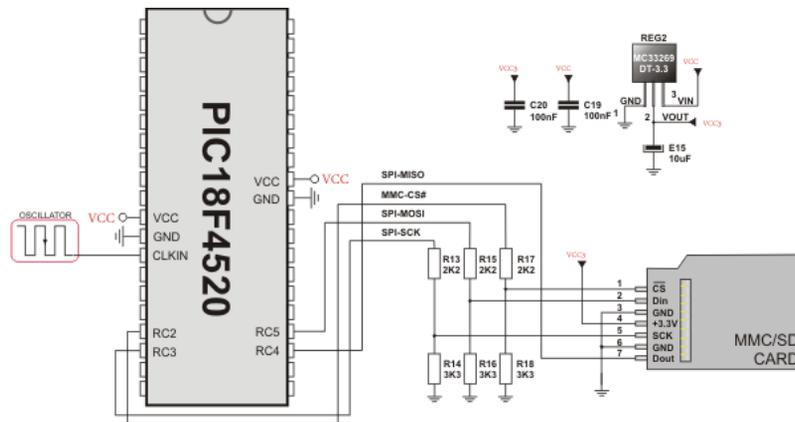
// Chech data match
data_ok := 1;
for i := 0 to 511 do
    begin
        UART1_Write(SectorData[ i ]);
        if (SectorData[ i] <> FILL_CHAR) then
            begin
                data_ok := 0;
                break;
            end;
        end;
    end;

if ( data_ok <> 0 ) then
    UART_Write_Line('Content-OK')
else
    UART_Write_Line('Content-Error');

// Signal test end
UART_Write_Line('Test End. ');
end.

```

HW Connection



Pin diagram of MMC memory card

ONEWIRE LIBRARY

The OneWire library provides routines for communication via the Dallas OneWire protocol, for example 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.

Note: This library implements time-based activities, so interrupts need to be disabled when using OneWire library.

Library Routines

- Ow_Reset
- Ow_Read
- Ow_Write

Ow_Reset

Prototype	<code>function Ow_Reset(var port : byte; pin : byte) : byte;</code>
Returns	0 if DS1820 is present, and 1 if not present.
Description	Issues OneWire reset signal for DS1820. Parameters port and pin specify the location of DS1820.
Requires	Works with Dallas DS1820 temperature sensor only.
Example	To reset the DS1820 that is connected to the RA5 pin: <code>Ow_Reset(PORTA, 5);</code>

Ow_Read

Prototype	<code>function Ow_Read(var port : byte; pin : byte) : byte;</code>
Returns	Data read from an external device over the OneWire bus.
Description	Reads one byte of data via the OneWire bus.
Requires	Works with Dallas DS1820 temperature sensor only.
Example	<code>tmp := Ow_Read(PORTA, 5);</code>

Ow_Write

Prototype	<code>procedure Ow_Write(var port : byte; pin, par : byte);</code>
Returns	Nothing.
Description	Writes one byte of data (argument par) via OneWire bus.
Requires	Works with Dallas DS1820 temperature sensor only.
Example	<code>Ow_Write(PORTA, 5, \$CC);</code>

Library Example

This example reads the temperature using DS18x20 connected to pin PORTA.5. After reset, MCU obtains temperature from the sensor and prints it on the Lcd. Make sure to pull-up PORTA.5 line and to turn off the PORTA LEDs.

```
program OneWire;

// LCD module connections
var LCD_RS : sbit at RB4_bit;
    LCD_EN : sbit at RB5_bit;
    LCD_D4 : sbit at RB0_bit;
    LCD_D5 : sbit at RB1_bit;
    LCD_D6 : sbit at RB2_bit;
    LCD_D7 : sbit at RB3_bit;
    LCD_RS_Direction : sbit at TRISB4_bit;
    LCD_EN_Direction : sbit at TRISB5_bit;
    LCD_D4_Direction : sbit at TRISB0_bit;
    LCD_D5_Direction : sbit at TRISB1_bit;
    LCD_D6_Direction : sbit at TRISB2_bit;
    LCD_D7_Direction : sbit at TRISB3_bit;
// End LCD module connections

// 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 : byte = 9;

var text : array[9] of byte;
    temp : word;

procedure Display_Temperature( temp2write : word );
const RES_SHIFT = TEMP_RESOLUTION - 8;

var temp_whole : byte;
    temp_fraction : word;

begin
    text := '000.0000';
    // check if temperature is negative
    if (temp2write and 0x8000) then
        begin
            text[0] := '-';
            temp2write := not temp2write + 1;
        end;

    // extract temp_whole
    temp_whole := word(temp2write shr RES_SHIFT);

    // convert temp_whole to characters
    if ( temp_whole div 100 ) then
        text[0] := temp_whole div 100 + 48
    else
        text[0] := '0';
```

```
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 := word(temp2write shl (4-RES_SHIFT));
temp_fraction := temp_fraction and 0x000F;
temp_fraction := temp_fraction * 625;

// convert temp_fraction to characters
text[4] := word(temp_fraction div 1000) + 48; //
Extract thousands digit
text[5] := word((temp_fraction div 100)mod 10 + 48); //
Extract hundreds digit
text[6] := word((temp_fraction div 10)mod 10 + 48); //
Extract tens digit
text[7] := word(temp_fraction mod 10) + 48; //
Extract ones digit

// print temperature on LCD
Lcd_Out(2, 5, text);
end;

begin

ANSEL := 0; // Configure AN pins as digital I/O
ANSELH := 0;

text := '000.0000';
Lcd_Init(); // Initialize Lcd
Lcd_Cmd(_LCD_CLEAR); // Clear Lcd
Lcd_Cmd(_LCD_CURSOR_OFF); // Turn cursor off
Lcd_Out(1, 1, ' Temperature: ');

Lcd_Chr(2,13,223); // Print degree character, 'C' for Centigrades
// different Lcd displays have different char code for degree
Lcd_Chr(2,14,'C'); // if you see greek alpha letter try typing
178 instead of 223

//--- main loop
while (TRUE) do
  begin
    //--- perform temperature reading
    Ow_Reset(PORTA, 5); // Onewire reset signal
    Ow_Write(PORTA, 5, 0xCC); // Issue command SKIP_ROM
    Ow_Write(PORTA, 5, 0x44); // Issue command CONVERT_T
    Delay_us(120);
```

```
Ow_Reset(PORTA, 5);
Ow_Write(PORTA, 5, 0xCC); // Issue command SKIP_ROM
Ow_Write(PORTA, 5, 0xBE); // Issue command READ_SCRATCHPAD

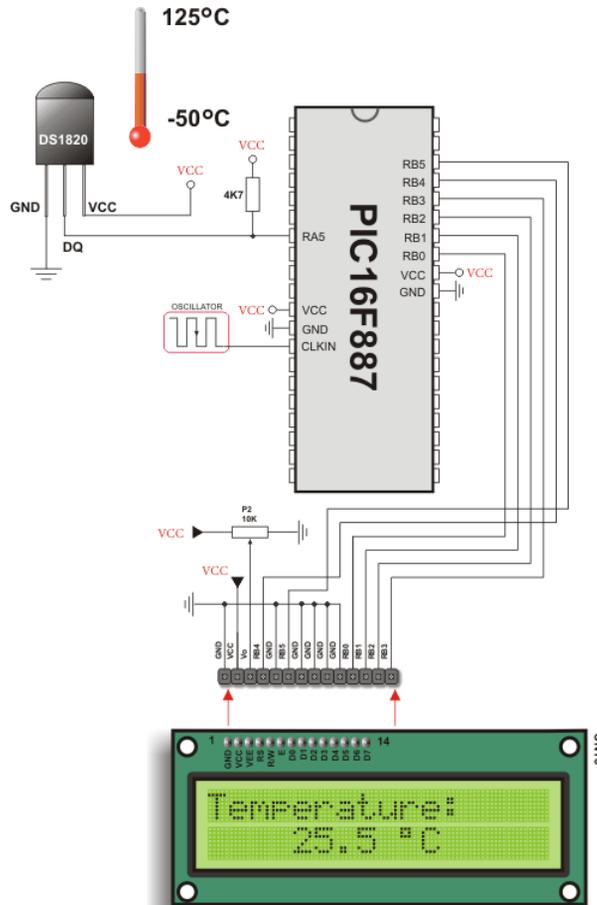
temp := Ow_Read(PORTA, 5);
temp := (Ow_Read(PORTA, 5) shl 8) + temp;

//--- Format and display result on Lcd

Display_Temperature(temp);

Delay_ms(520);
end;
end.
```

HW Connection



LCD 2X16

Example of DS1820 connection

PORT EXPANDER LIBRARY

The mikroPascal PRO for PIC provides a library for communication with the Microchip's Port Expander MCP23S17 via SPI interface. Connections of the PIC 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>var SPExpanderRST : sbit; sfr; external;</code>	Reset line.	<code>var SPExpanderRST : sbit at RC0_bit;</code>
<code>var SPExpanderCS : sbit; sfr; external;</code>	Chip Select line.	<code>var SPExpanderCS : sbit at RC1_bit;</code>
<code>var SPExpanderRST_Direction : sbit; sfr; external;</code>	Direction of the Reset pin.	<code>var SPExpanderRST_Direction : sbit at TRISC0_bit;</code>
<code>var SPExpanderCS_Direction : sbit; sfr; external;</code>	Direction of the Chip Select pin.	<code>var SPExpanderCS_Direction : sbit at TRISC1_bit;</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>procedure Expander_Init(ModuleAddress : 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>Global variables :</p> <ul style="list-style-type: none"> - <code>SPExpanderCS</code>: Chip Select line - <code>SPExpanderRST</code>: Reset line - <code>SPExpanderCS_Direction</code>: Direction of the Chip Select pin - <code>SPExpanderRST_Direction</code>: Direction of the Reset pin <p>must be defined before using this function.</p> <p>SPI module needs to be initialized. See <code>SPI1_Init</code> and <code>SPI1_Init_Advanced</code> routines.</p>
Example	<pre>// Port Expander module connections var SPExpanderRST : sbit at RC0_bit; SPExpanderCS : sbit at RC1_bit; SPExpanderRST_Direction : sbit at TRISC0_bit; SPExpanderCS_Direction : sbit at TRISC1_bit; // End of Port Expander module connections ... SPI_Init(); // initialize SPI module Expander_Init(0); // initialize port expander</pre>

Expander_Read_Byte

Prototype	<code>function Expander_Read_Byte(ModuleAddress : byte; RegAddress : byte) : 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 var read_data : byte; ... read_data := Expander_Read_Byte(0,1);</pre>

Expander_Write_Byte

Prototype	<code>procedure Expander_Write_Byte(ModuleAddress: byte; RegAddress: byte; Data_: 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>function Expander_Read_PortA(ModuleAddress: byte): byte;</code>
Returns	Byte read.
Description	<p>The function reads byte from Port Expander's PortA.</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 should be configured as input. See <code>Expander_Set_DirectionPortA</code> and <code>Expander_Set_DirectionPortAB</code> routines.</p>
Example	<pre>// Read a byte from Port Expander's PORTA var read_data : 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>function Expander_Read_PortB(ModuleAddress: byte): byte;</code>
Returns	Byte read.
Description	<p>The function reads byte from Port Expander's PortB.</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 PortB should be configured as input. See <code>Expander_Set_DirectionPortB</code> and <code>Expander_Set_DirectionPortAB</code> routines.</p>
Example	<pre>// Read a byte from Port Expander's PORTB var read_data : 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>function Expander_Read_PortAB(ModuleAddress: byte): 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 var read_data : 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>procedure Expander_Write_PortA(ModuleAddress: byte; Data_: 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>procedure Expander_Write_PortB(ModuleAddress: byte; Data_: 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>procedure Expander_Write_PortAB(ModuleAddress: byte; Data_: 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>procedure Expander_Set_DirectionPortA(ModuleAddress: byte; Data_: 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>procedure Expander_Set_DirectionPortB(ModuleAddress: byte; Data_: 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>procedure Expander_Set_DirectionPortAB(ModuleAddress: byte; Direction: 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>procedure Expander_Set_PullUpsPortA(ModuleAddress: byte; Data_: 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>procedure Expander_Set_PullUpsPortB(ModuleAddress: byte; Data_: 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>procedure Expander_Set_PullUpsPortAB(ModuleAddress: byte; PullUps: 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;

// Port Expander module connections
var SPExpanderRST : sbit at RC0_bit;
    SPExpanderCS   : sbit at RC1_bit;
    SPExpanderRST_Direction : sbit at TRISCO_bit;
    SPExpanderCS_Direction  : sbit at TRISC1_bit;
// End Port Expander module connections

var counter : byte;// = 0;

begin
    counter := 0;
    ANSEL   := 0;           // Configure AN pins as digital I/O
    ANSELH  := 0;
    TRISB   := 0;           // Set PORTB as output
    PORTB   := 0xFF;

    // If Port Expander Library uses SPI1 module
    SPI1_Init();           // Initialize SPI module used with PortExpander

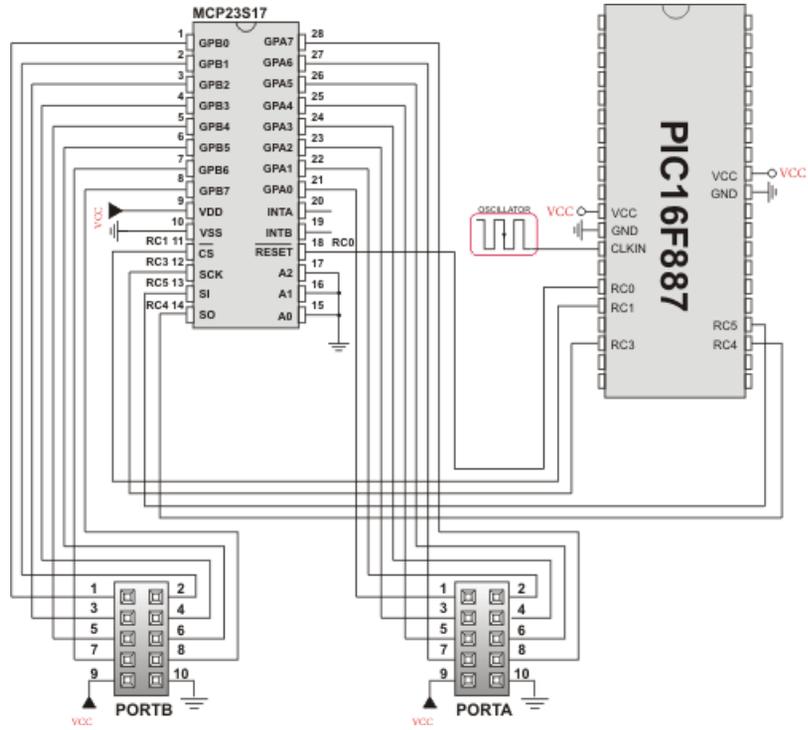
    Expander_Init(0);      // Initialize Port Expander

    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 ) do      // Endless loop
        begin
            Expander_Write_PortA(0, counter); // Write i to expander's
PORTA
            Inc(counter);
            PORTB := Expander_Read_PortB(0); // Read expander's PORTB
and write it to LEDs
            Delay_ms(100);
        end;
    end.
end.
```

HW Connection



Port Expander HW connection

PS/2 LIBRARY

The mikroPascal PRO for PIC 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>var PS2_Data : sbit; sfr; external;</code>	PS/2 Data line.	<code>var PS2_Data : sbit at RC0_bit;</code>
<code>var PS2_Clock : sbit; sfr; external;</code>	PS/2 Clock line.	<code>var PS2_Clock : sbit at RC1_bit;</code>
<code>var PS2_Data_Direction : sbit; sfr; external;</code>	Direction of the PS/2 Data pin.	<code>var PS2_Data_Direction : sbit at TRISC0_bit;</code>
<code>var PS2_Clock_Direction : sbit; sfr; external;</code>	Direction of the PS/2 Clock pin.	<code>var PS2_Clock_Direction : sbit at TRISC1_bit;</code>

Library Routines

- Ps2_Config
- Ps2_Key_Read

Ps2_Config

Prototype	<code>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"> - PS2_Data: Data signal line - PS2_Clock: Clock signal line - PS2_Data_Direction: Direction of the Data pin - PS2_Clock_Direction: Direction of the Clock pin <p>must be defined before using this function.</p>
Example	<pre>// PS2 pinout definition var PS2_Data : sbit at RC0_bit; var PS2_Clock : sbit at RC1_bit; var PS2_Data_Direction : sbit at TRISC0_bit; var PS2_Clock_Direction : sbit at TRISC1_bit; // End of PS2 pinout definition ... Ps2_Config(); // Init PS/2 Keyboard</pre>

Ps2_Key_Read

Prototype	<code>function Ps2_Key_Read(var value: byte; var special: byte; var pressed: byte): byte;</code>
Returns	- 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>var value, special, pressed: byte; ... // Press Enter to continue: repeat if (Ps2_Key_Read(value, special, pressed)) then if ((value = 13) and (special = 1)) then break; 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_Example;

var keydata, special, down : byte;

var PS2_Data      : sbit at RC0_bit;
    PS2_Clock     : sbit at RC1_bit;

    PS2_Data_Direction : sbit at TRISC0_bit;
    PS2_Clock_Direction : sbit at TRISC1_bit;

begin

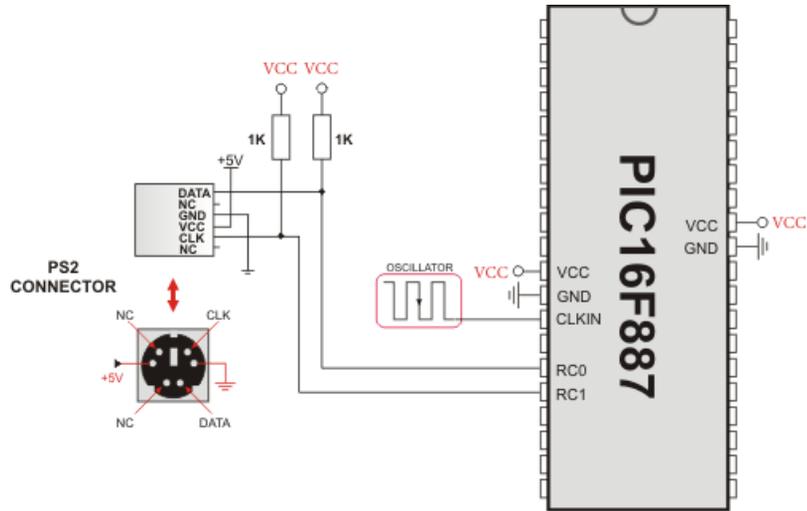
    ANSEL := 0;           // Configure AN pins as digital I/O
    ANSELH := 0;

    keydata := 0;
    special := 0;
    down := 0;

    UART1_Init(19200);   // Initialize UART module at 19200 bps
    Ps2_Config();       // Init PS/2 Keyboard
    Delay_ms(100);      // Wait for keyboard to finish
    UART1_Write_Text('Ready');

    while TRUE do      // Endless loop
        begin
            if Ps2_Key_Read(keydata, special, down) then // If data
was read from PS/2
                begin
                    if (down <> 0) and (keydata = 16) then // Backspace read
                        begin
                            UART1_Write(0x08); //Send Backspace to usart terminal
                        end
                    else if (down <> 0) and (keydata = 13) then // Enter read
                        begin
                            UART1_Write(10);           // Send
carriage return to usart terminal
                            UART1_Write(13);           // Uncomment this line
if usart terminal also expects line feed
                                                    // for new line transition
                        end
                    else if (down <> 0) and (special = 0) and (keydata <>
0) then // Common key read
                        begin
                            UART1_Write(keydata); // Send key to usart terminal
                        end;
                    end;
                    Delay_ms(1);           // Debounce period
                end;
        end;
end.
```

HW Connection



Example of PS2 keyboard connection

PWM LIBRARY

CCP module is available with a number of PIC MCUs. provides library which simplifies using PWM HW Module.

Note: Some MCUs have multiple CCP modules. In order to use the desired CCP library routine, simply change the number 1 in the prototype with the appropriate module number, i.e.

```
PWM2_Start();
```

Library Routines

- PWM1_Init
- PWM1_Set_Duty
- PWM1_Start
- PWM1_Stop

PWM1_Init

Prototype	<code>procedure PWM1_Init(freq : longint);</code>
Returns	Nothing.
Description	<p>Initializes the PWM module with duty ratio 0. Parameter freq is a desired PWM frequency in Hz (refer to device data sheet for correct values in respect with Fosc).</p> <p>This routine needs to be called before using other functions from PWM Library.</p>
Requires	<p>MCU must have CCP module.</p> <p>Note: Calculation of the PWM frequency value is carried out by the compiler, as it would produce a relatively large code if performed on the library level. Therefore, compiler needs to know the value of the parameter in the compile time. That is why this parameter needs to be a constant, and not a variable.</p>
Example	<p>Initialize PWM module at 5KHz:</p> <pre>PWM1_Init(5000);</pre>

PWM1_Set_Duty

Prototype	<code>procedure PWM1_Set_Duty(duty_ratio : byte);</code>
Returns	Nothing.
Description	sets PWM duty ratio. Parameter duty takes values from 0 to 255, where 0 is 0%, 127 is 50%, and 255 is 100% duty ratio. Other specific values for duty ratio can be calculated as $(\text{Percent} * 255) / 100$.
Requires	MCU must have CCP module. PWM1_Init must be called before using this routine.
Example	Set duty ratio to 75%: <code>PWM1_Set_Duty(192);</code>

PWM1_Start

Prototype	<code>procedure PWM1_Start();</code>
Returns	Nothing.
Description	Starts PWM.
Requires	MCU must have CCP module. PWM1_Init must be called before using this routine.
Example	<code>PWM1_Start();</code>

PWM1_Stop

Prototype	<code>procedure PWM1_Stop();</code>
Returns	Nothing.
Description	Stops PWM.
Requires	MCU must have CCP module. PWM1_Init must be called before using this routine. PWM1_Start should be called before using this routine, otherwise it will have no effect as the PWM module is not running.
Example	<code>PWM1_Stop();</code>

Library Example

The example changes PWM duty ratio on RC1 and RC2 pins continually. If LED is connected to these pins, you can observe the gradual change of emitted light.

```
program PWM_Test;

var current_duty, current_duty1, old_duty, old_duty1 : byte;

procedure InitMain();
begin
    ANSEL := 0;           // Configure AN pins as digital I/O
    ANSELH := 0;

    PORTA := 255;
    TRISA := 255;        // configure PORTA pins as input
    PORTB := 0;         // set PORTB to 0
    TRISB := 0;         // designate PORTB pins as output
    PORTC := 0;         // set PORTC to 0
    TRISC := 0;         // designate PORTC pins as output
    PWM1_Init(5000);    // Initialize PWM1 module at 5KHz
    PWM2_Init(5000);    // Initialize PWM2 module at 5KHz
end;

begin
    InitMain();
    current_duty := 16; // initial value for current_duty
    current_duty1 := 16; // initial value for current_duty1

    PWM1_Start();      // start PWM1
    PWM2_Start();      // start PWM2
    PWM1_Set_Duty(current_duty); // Set current duty for PWM1
    PWM2_Set_Duty(current_duty1); // Set current duty for PWM2

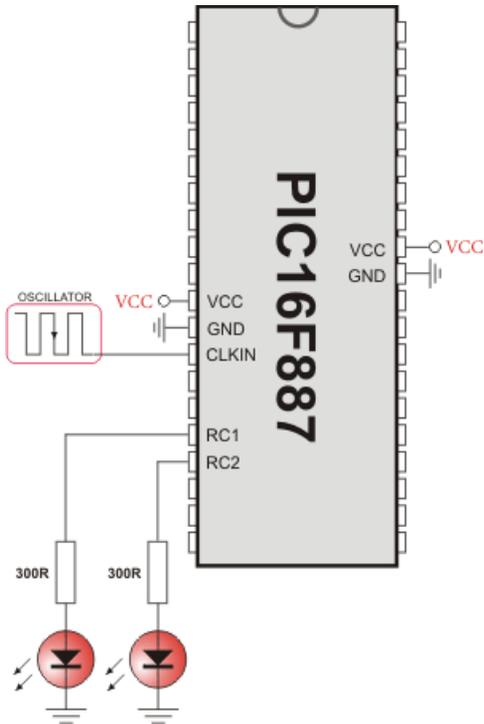
    while (TRUE) do // endless loop
    begin
        if (RA0_bit <> 0) then // button on RA0 pressed
        begin
            Delay_ms(40);
            Inc(current_duty); // increment current_duty
            PWM1_Set_Duty(current_duty);
        end;

        if (RA1_bit <> 0) then // button on RA1 pressed
        begin
            Delay_ms(40);
            Dec(current_duty); // decrement current_duty
            PWM1_Set_Duty(current_duty);
        end;

        if (RA2_bit <> 0) then // button on RA2 pressed
        begin
            Delay_ms(40);
            Inc(current_duty1); // increment current_duty1
        end;
    end;
end;
```

```
PWM2_Set_Duty(current_duty1);  
    end;  
  
    if (RA3_bit <> 0) then           // button on RA3 pressed  
        begin  
            Delay_ms(40);  
            Dec(current_duty1);       // decrement current_duty1  
            PWM2_Set_Duty(current_duty1);  
        end;  
  
        Delay_ms(5);                 // slow down change pace a little  
    end;  
end.
```

HW Connection



PWM demonstration

RS-485 LIBRARY

RS-485 is a multipoint communication which allows multiple devices to be connected to a single bus. The mikroPascal PRO for PIC 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).

Note: The library uses the UART module for communication. The user must initialize the appropriate UART module before using the RS-485 Library. For MCUs with two UART modules it is possible to initialize both of them and then switch by using the `UART_Set_Active` function. See the UART Library functions.

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 :
<code>var RS485_rxtx_pin : sbit; sfr; external;</code>	Control RS-485 Transmit/Receive operation mode	<code>var RS485_rxtx_pin : sbit at RC2_bit;</code>
<code>var RS485_rxtx_pin_direction : sbit; sfr; external;</code>	Direction of the RS-485 Transmit/Receive pin	<code>var RS485_rxtx_pin_direction : sbit at TRISC2_bit;</code>

Library Routines

- RS485Master_Init
- RS485Master_Receive
- RS485Master_Send
- RS485Slave_Init
- RS485Slave_Receive
- RS485Slave_Send

RS485Master_Init

Prototype	<code>procedure RS485Master_Init();</code>
Returns	Nothing.
Description	Initializes MCU as a Master for RS-485 communication.
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>RS485_rxtx_pin</code> - 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. - <code>RS485_rxtx_pin_direction</code> - direction of the RS-485 Transmit/Receive pin <p>must be defined before using this function.</p> <p>UART HW module needs to be initialized. See <code>UARTx_Init</code>.</p>
Example	<pre>// RS485 module pinout var RS485_rxtx_pin : sbit at RC2_bit; var RS485_rxtx_pin_direction : sbit at TRISC2_bit; // End of RS485 module pinout ... UART1_Init(9600); // initialize UART module RS485Master_Init(); // intialize MCU as a Master for RS-485 communication</pre>

RS485Master_Receive

Prototype	<code>procedure RS485Master_Receive(var data_buffer: array[5] 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>var msg : array[20] of byte; ... RS485Master_Receive(msg);</pre>

RS485Master_Send

Prototype	<code>procedure RS485Master_Send(var data_buffer: array[20] of byte; datalen: byte; slave_address: 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>var msg : array[20] of byte; ... // send 3 bytes of data to Slave with address 0x12 RS485Master_Send(msg, 3, 0x12);</pre>

RS485Slave_Init

Prototype	<code>procedure RS485Slave_Init(slave_address: byte);</code>
Returns	Nothing.
Description	<p>Initializes MCU as a Slave for RS-485 communication.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>slave_address</code>: Slave address
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>RS485_rxtx_pin</code> - 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: <code>1</code> (for transmitting) and <code>0</code> (for receiving) - <code>RS485_rxtx_pin_direction</code> - direction of the RS-485 Transmit/Receive pin <p>must be defined before using this function.</p> <p>UART HW module needs to be initialized. See <code>UARTx_Init</code>.</p>
Example	<pre>// RS485 module pinout var RS485_rxtx_pin : sbit at RC2_bit; var RS485_rxtx_pin_direction : sbit at TRISC2_bit; // End of RS485 module pinout ... UART1_Init(9600); // initialize UART module RS485Slave_Init(160); // intialize MCU as a Slave for RS-485 communication with address 160</pre>

RS485slave_Receive

Prototype	<code>procedure RS485Slave_Receive(var data_buffer: 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> <p>data_buffer: 6 byte buffer for storing received data, in the following manner:</p> <ul style="list-style-type: none"> - data[0..2] : message content - data[3] : number of message bytes received, 1–3 - data[4] : is set to 255 when message is received - data[5] : is set to 255 if error has occurred <p>The function automatically adjusts data[4] and data[5] 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 RS485Slave_Init.
Example	<pre>var msg : array[20] of byte; ... RS485Slave_Read(msg);</pre>

RS485Slave_Send

Prototype	<code>procedure RS485Slave_Send(var data_buffer: array[20] of byte; datalen : 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 RS485Slave_Init. It is the user's responsibility to ensure (by protocol) that only one device sends data via 485 bus at a time.
Example	<code>var msg : array[8] of byte; ... // send 2 bytes of data to the Master RS485Slave_Send(msg, 2);</code>

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 PORTB, while error on receive (0xAA) and number of consecutive unsuccessful retries are displayed on PORTD. Slave displays received data on PORTB, while error on receive (0xAA) is displayed on PORTD. Hardware configurations in this example are made for the EasyPIC5 board and 16F887.

RS485 Master code:

```
program RS485_Master_Example;

var dat : array[ 10] of byte ;           // buffer for receiving/sending messages
    i, j : byte;
    cnt : longint;
```

```
var rs485_rxtx_pin   : sbit at RC2_bit;           // set transceiver pin
    rs485_rxtx_pin_direction : sbit at TRISC2_bit; // set transceiver pin direction

// Interrupt routine
procedure interrupt(); org 0x16;
begin
    RS485Master_Receive(dat);
end;

begin
    cnt := 0;
    ANSEL := 0;           // Configure AN pins as digital I/O
    ANSELH := 0;

    PORTB := 0;
    PORTD := 0;
    TRISB := 0;
    TRISD := 0;

    UART1_Init(9600);           // initialize UART1 module
    Delay_ms(100);

    RS485Master_Init();        // initialize MCU as Master
    dat[ 0] := 0xAA;
    dat[ 1] := 0xF0;
    dat[ 2] := 0x0F;
    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);

    PIE1.RCIE := 1;         // enable interrupt on UART1 receive
    PIE2.TXIE := 0;         // disable interrupt on UART1 transmit
    INTCON.PEIE := 1;       // enable peripheral interrupts
    INTCON.GIE := 1;        // enable all interrupts

    while (TRUE) do
        begin           // upon completed valid message receiving
                        //   data[ 4] is set to 255

Inc(cnt);
        if (dat[ 5] <> 0) then // if an error detected, signal it
            PORTD := 0xAA;       //   by setting portd to 0xAA
        if (dat[ 4] <> 0) then // if message received successfully
            begin
```

```
        cnt := 0;
        dat[ 4] := 0;           // clear message received flag
        j := dat[ 3];
        for i := 1 to dat[ 3] do           // show data on PORTB
            PORTB := dat[ i-1];
        dat[ 0] := dat[ 0] +1;           // send back to master
        Delay_ms(1);
        RS485Master_Send(dat,1,160);
    end;

    if (cnt > 100000) then // if in 100000 poll-cycles the answer
    begin
        Inc(PORTD);           // was not detected, signal
        cnt := 0;           // failure of send-message
        RS485Master_Send(dat,1,160);
        if (PORTD > 10) then // if sending failed 10 times
            begin
                RS485Master_Send(dat,1,50); // send message on
broadcast address
            end;
        end;
    end;
end; // function to be properly linked.
end.
```

RS485 Slave code:

```
program RS485_Slave_Example;

var dat : array[ 20] of byte;           // buffer for receving/send-
ing messages
    i, j : byte;

var rs485_rxtx_pin : sbit at RC2_bit; // set transcieve pin
    rs485_rxtx_pin_direction : sbit at TRISC2_bit; // set tran-
scieve pin direction

// Interrupt routine
procedure interrupt();
begin
    RS485Slave_Receive(dat);
end;

begin

    ANSEL := 0; // Configure AN pins as digital I/O
    ANSELH := 0;
```

```
PORTB := 0;
PORTD := 0;
TRISB := 0;
TRISD := 0;

UART1_Init(9600);           // initialize UART1 module
Delay_ms(100);
RS485Slave_Init(160);      // Initialize MCU as slave, address 160

dat[ 4] := 0;              // ensure that message received flag is 0
dat[ 5] := 0;              // ensure that message received flag is 0
dat[ 6] := 0;              // ensure that error flag is 0

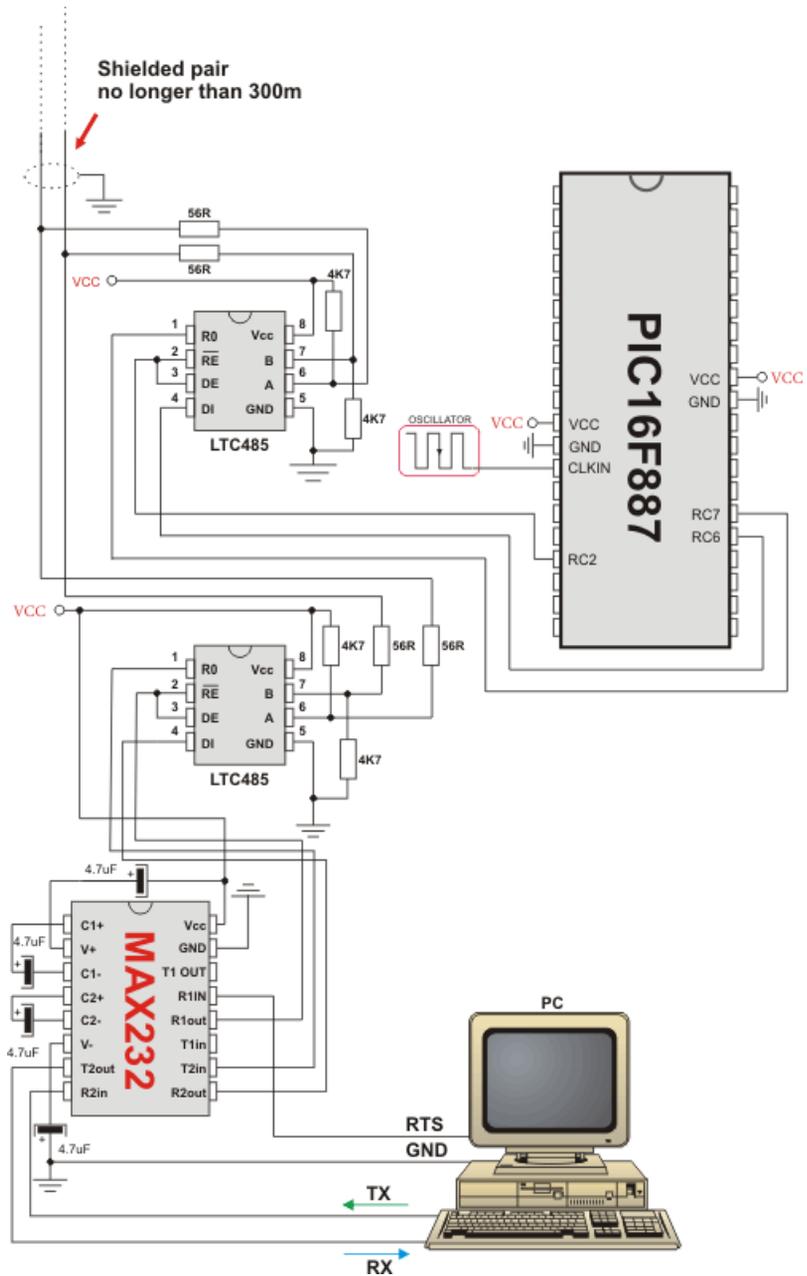
PIE1.RCIE := 1;           // enable interrupt on UART1 receive
PIE2.TXIE := 0;           // disable interrupt on UART1 transmit
INTCON.PEIE := 1;         // enable peripheral interrupts
INTCON.GIE := 1;         // enable all interrupts

while (TRUE) do
begin
  if (dat[ 5] <> 0) then // if an error detected, signal it by
  begin
    PORTD := 0xAA;        // setting portd to 0xAA
    dat[ 5] := 0;
  end;
  if (dat[ 4] <> 0) then // upon completed valid message receive
  begin
    dat[ 4] := 0;        // data[ 4] is set to 0xFF
    j := dat[ 3];

    for i := 1 to dat[ 3] do // show data on PORTB
      PORTB := dat[ i-1];

    dat[ 0] := dat[ 0] +1; // increment received dat[ 0]
    Delay_ms(1);
    RS485Slave_Send(dat,1); // and send it back to master
  end;
end;
end.
```

HW Connection



Example of interfacing PC to PIC 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 xor address;
crc_send := crc_send xor data[ 0]; // if exists
crc_send := crc_send xor data[ 1]; // if exists
crc_send := crc_send xor data[ 2]; // if exists
crc_send := not crc_send;
if ((crc_send = START_BYTE) or (crc_send = STOP_BYTE)) then
  Inc(crc_send);
```

```
NOTE: DATALEN<4..0> can not take the START_BYTE<4..0> or
STOP_BYTE<4..0> values.
```

SOFTWARE I²C LIBRARY

The mikroPascal PRO for PIC 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 Software I²C Library functions are blocking-call functions (they are waiting for I²C clock line to become logical one).

Note: The pins used for the Software 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 Software I²C Library

The following variables must be defined in all projects using Software I ² C Library:	Description:	Example :
<code>var Soft_I2C_Scl : sbit; sfr; external;</code>	Soft I ² C Clock line.	<code>var Soft_I2C_Scl : sbit at RC3_bit;</code>
<code>var Soft_I2C_Sda : sbit; sfr; external;</code>	Soft I ² C Data line.	<code>var Soft_I2C_Sda : sbit at RC4_bit;</code>
<code>var Soft_I2C_Scl_Direction : sbit; sfr; external;</code>	Direction of the Soft I ² C Clock pin.	<code>var Soft_I2C_Scl_Direction : sbit at TRISC3_bit;</code>
<code>var Soft_I2C_Sda_Direction : sbit; sfr; external;</code>	Direction of the Soft I ² C Data pin.	<code>var Soft_I2C_Sda_Direction : sbit at TRISC4_bit;</code>

Library Routines

- Soft_I2C_Init
- Soft_I2C_Start
- Soft_I2C_Read
- Soft_I2C_Write
- Soft_I2C_Stop
- Soft_I2C_Break

Soft_I2C_Init

Prototype	<code>procedure Soft_I2C_Init();</code>
Returns	Nothing.
Description	Configures the software I ² C module.
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>Soft_I2C_Scl</code>: Soft I²C clock line - <code>Soft_I2C_Sda</code>: Soft I²C data line - <code>Soft_I2C_Scl_Direction</code>: Direction of the Soft I²C clock pin - <code>Soft_I2C_Sda_Direction</code>: Direction of the Soft I²C data pin <p>must be defined before using this function.</p>
Example	<pre>// Soft_I2C pinout definition var Soft_I2C_Scl : sbit at RC3_bit; var Soft_I2C_Sda : sbit at RC4_bit; var Soft_I2C_Scl_Direction : sbit at TRISC3_bit; var Soft_I2C_Sda_Direction : sbit at TRISC4_bit; // End of Soft_I2C pinout definition ... Soft_I2C_Init();</pre>

Soft_I2C_Start

Prototype	<code>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>function Soft_I2C_Read(ack: word): 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>var take : word; ... // Read data and send the not_acknowledge signal take := Soft_I2C_Read(0);</pre>

Soft_I2C_Write

Prototype	<code>function Soft_I2C_Write(_data: byte): 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>var _data, error : byte; ... error := Soft_I2C_Write(data); error := Soft_I2C_Write(0xA3);</pre>

Soft_I2C_Stop

Prototype	<code>procedure Soft_I2C_Stop();</code>
Returns	Nothing.
Description	Issues STOP signal.
Requires	Soft I ² C must be configured before using this function. See Soft_I2C_Init routine.
Example	<pre>// Issue STOP signal Soft_I2C_Stop();</pre>

Soft_I2C_Break

Prototype	<code>procedure Soft_I2C_Break();</code>
Returns	Nothing.
Description	<p>All Software I²C Library functions can block the program flow (see note at the top of this page). Calling this routine from interrupt will unblock the program execution. This mechanism is similar to WDT.</p> <p>Note: Interrupts should be disabled before using Software I²C routines again (see note at the top of this page).</p>
Requires	Nothing.
Example	<pre>var data1, error, counter : byte; procedure interrupt(); begin if (INTCON.TOIF <> 0) then if (counter >= 20) then begin Soft_I2C_Break(); counter := 0; // reset counter end else Inc(counter); // increment counter INTCON.TOIF := 0; // Clear Timer0 overflow inter- end; rupt flag begin counter := 0; OPTION_REG := 0x04; // TMR0 prescaler set to 1:32 ... // try Soft_I2C_Init with blocking prevention mechanism INTCON.GIE := 1; // Global interrupt enable INTCON.TOIE := 1; // Enable Timer0 overflow inter- end; rupt Soft_I2C_Init(); INTCON.GIE := 0; // Global interrupt disable end.</pre>

Library Example

The example demonstrates Software I₂C Library routines usage. The PIC 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 RTC_Read;

var seconds, minutes, hours, day, month, year : byte;      // Global
date/time variables

// Software I2C connections
var Soft_I2C_Scl      : sbit at RC3_bit;
    Soft_I2C_Sda      : sbit at RC4_bit;
    Soft_I2C_Scl_Direction : sbit at TRISC3_bit;
    Soft_I2C_Sda_Direction : sbit at TRISC4_bit;
// End Software I2C connections

// LCD module connections
var LCD_RS : sbit at RB4_bit;
    LCD_EN : sbit at RB5_bit;
    LCD_D4 : sbit at RB0_bit;
    LCD_D5 : sbit at RB1_bit;
    LCD_D6 : sbit at RB2_bit;
    LCD_D7 : sbit at RB3_bit;
    LCD_RS_Direction : sbit at TRISB4_bit;
    LCD_EN_Direction : sbit at TRISB5_bit;
    LCD_D4_Direction : sbit at TRISB0_bit;
    LCD_D5_Direction : sbit at TRISB1_bit;
    LCD_D6_Direction : sbit at TRISB2_bit;
    LCD_D7_Direction : sbit at TRISB3_bit;
// End LCD module connections

//----- Reads time and date information from RTC
(PCF8583)
procedure Read_Time();
begin
    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;
```

```

//----- Formats date and time
procedure Transform_Time() ;
  begin
    seconds := ((seconds and 0xF0) shr 4)*10 + (seconds and 0x0F);
  // Transform seconds
    minutes := ((minutes and 0xF0) shr 4)*10 + (minutes and 0x0F);
  // Transform months
    hours := ((hours and 0xF0) shr 4)*10 + (hours and 0x0F);
  // Transform hours
    year := (day and 0xC0) shr 6; // Transform year
    day := ((day and 0x30) shr 4)*10 + (day and 0x0F);
  // Transform day
    month := ((month and 0x10) shr 4)*10 + (month and 0x0F);
  // Transform month
  end;

//----- Output values to LCD
procedure Display_Time();
  begin
    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;

//----- Performs project-wide init
procedure Init_Main();
  begin

    TRISB := 0;
    PORTB := 0xFF;
    TRISB := 0xff;
    ANSEL := 0; // Configure AN pins as digital I/O
    ANSELH := 0;

    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;

//----- Main procedure
begin

    Delay_ms(1000);

    Init_Main(); // Perform initialization

    while TRUE do // Endless loop
        begin
            Read_Time(); // Read time from RTC(PCF8583)
            Transform_Time(); // Format date and time
            Display_Time(); // Prepare and display on LCD
        end;
    end.
```

SOFTWARE SPI LIBRARY

The mikroPascal PRO for PIC 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.

External dependencies of Software SPI Library

The following variables must be defined in all projects using Software SPI Library:	Description:	Example :
<code>var SoftSpi_SDI : sbit; sfr; external;</code>	Data In line.	<code>var SoftSpi_SDI : sbit at RC4_bit;</code>
<code>var SoftSpi_SDO : sbit; sfr; external;</code>	Data Out line.	<code>var SoftSpi_SDO : sbit at RC5_bit;</code>
<code>var SoftSpi_CLK : sbit; sfr; external;</code>	Clock line.	<code>var SoftSpi_CLK : sbit at RC3_bit;</code>
<code>var SoftSpi_SDI_Direction : sbit; sfr; external;</code>	Direction of the Data In pin.	<code>var SoftSpi_SDI_Direction : sbit at TRISC4_bit;</code>
<code>var SoftSpi_SDO_Direction : sbit; sfr; external;</code>	Direction of the Data Out pin	<code>var SoftSpi_SDO_Direction : sbit at TRISC5_bit;</code>
<code>var SoftSpi_CLK_Direction : sbit; sfr; external;</code>	Direction of the Clock pin.	<code>var SoftSpi_CLK_Direction : sbit at TRISC3_bit;</code>

Library Routines

- Soft_SPI_Init
- Soft_SPI_Read
- Soft_SPI_Write

Soft_SPI_Init

Prototype	<code>procedure Soft_SPI_Init();</code>
Returns	Nothing.
Description	Configures and initializes the software SPI module.
Requires	<p>Global variables:</p> <ul style="list-style-type: none"> - <code>Chip_Select</code>: Chip select line - <code>SoftSpi_SDI</code>: Data in line - <code>SoftSpi_SDO</code>: Data out line - <code>SoftSpi_CLK</code>: Data clock line - <code>Chip_Select_Direction</code>: Direction of the Chip select pin - <code>SoftSpi_SDI_Direction</code>: Direction of the Data in pin - <code>SoftSpi_SDO_Direction</code>: Direction of the Data out pin - <code>SoftSpi_CLK_Direction</code>: Direction of the Data clock pin <p>must be defined before using this function.</p>
Example	<pre>// Software SPI module connections var Chip_Select : sbit at RC1; var SoftSpi_SDI : sbit at RC4_bit; var SoftSpi_SDO : sbit at RC5_bit; var SoftSpi_CLK : sbit at RC3_bit; var Chip_Select_Direction : sbit at TRISC1_bit; var SoftSpi_SDI_Direction : sbit at TRISC4_bit; var SoftSpi_SDO_Direction : sbit at TRISC5_bit; var SoftSpi_CLK_Direction : sbit at TRISC3_bit; // End Software SPI module connections ... Soft_SPI_Init(); // Init Soft_SPI</pre>

Soft_SPI_Read

Prototype	<code>function Soft_SPI_Read(sdata: byte): word;</code>
Returns	Byte received via the SPI bus.
Description	This routine performs 3 operations simultaneously. It provides clock for the Software SPI bus, reads a byte and sends a byte. 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>var data_read : word; data_send : 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>procedure Soft_SPI_Write(sdata: 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
var Chip_Select : sbit at RC1_bit;
    SoftSpi_CLK : sbit at RC3_bit;
    SoftSpi_SDI : sbit at RC4_bit;    // Note: Input signal
    SoftSpi_SDO : sbit at RC5_bit;

var Chip_Select_Direction : sbit at TRISC1_bit;
    SoftSpi_CLK_Direction : sbit at TRISC3_bit;
    SoftSpi_SDI_Direction : sbit at TRISC4_bit;
    SoftSpi_SDO_Direction : sbit at TRISC5_bit;
// End DAC module connections

var value : word;

procedure InitMain();
begin
    TRISA0_bit := 1;           // Set RA0 pin as input
    TRISA1_bit := 1;           // Set RA1 pin as input
    Chip_Select := 1;          // Deselect DAC
    Chip_Select_Direction := 0; // Set CS# pin as Output
    SoftSpi_Init();           // Initialize Soft_SPI
end;

// DAC increments (0..4095) --> output voltage (0..Vref)
procedure DAC_Output( valueDAC : word);
var temp : byte; volatile;
begin
    Chip_Select := 0;          // Select DAC chip

    // Send High Byte
    temp := word(valueDAC shr 8) and 0x0F; // Store valueDAC[11..8]
to temp[3..0]
    temp := temp 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;

begin

    ANSEL := 0;
    ANSELH := 0;
    C1ON_bit := 0;           // Disable comparators
    C2ON_bit := 0;
```

```
InitMain(); // Perform main initialization

value := 2048; // When program starts, DAC gives
              // the output in the mid-range

while (TRUE) do // Endless loop
begin

    if ((RA0_bit) and (value < 4095)) then // If PA0 button
is pressed
        Inc(value) // increment value
    else
    begin
        if ((RA1_bit) and (value > 0)) then // If PA1 button
is pressed
            Dec(value); // decrement value
        end;

        DAC_Output(value); // Send value to DAC chip
        Delay_ms(1); // Slow down key repeat pace
    end;
end.
```

SOFTWARE UART LIBRARY

The mikroPascal PRO for PIC 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.

Library Routines

- Soft_UART_Init
- Soft_UART_Read
- Soft_UART_Write
- Soft_UART_Break

Soft_UART_Init

Prototype	<code>function Soft_Uart_Init(var port : byte; rx_pin, tx_pin, baud_rate : dword, inverted : byte): byte;</code>
Returns	<ul style="list-style-type: none"> - 2 - error, requested baud rate is too low - 1 - error, requested baud rate is too high - 0 - successfull initialization
Description	<p>Configures and initializes the software UART module.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>port</code>: port to be used. - <code>rx_pin</code>: sets rx_pin to be used. - <code>tx_pin</code>: sets tx_pin to be used. - <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. <p>Software UART routines use Delay_Cyc routine. If requested baud rate is too low then calculated parameter for calling Delay_Cyc exceeds Delay_Cyc argument range.</p> <p>If requested baud rate is too high then rounding error of Delay_Cyc argument corrupts Software UART timings.</p>
Requires	Nothing.
Example	<pre>var error : byte; ... // Initialize Software UART communication on pins Rx, Tx, at 9600 bps error := Soft_UART_Init(PORTC, 7, 6, 14400, 0);</pre>

Soft_UART_Read

Prototype	<code>function Soft_UART_Read(var error: byte): byte;</code>
Returns	Byte received via UART.
Description	<p>The function receives a byte via software UART.</p> <p>This is a blocking function call (waits for start bit). Programmer can unblock it by calling Soft_UART_Break routine.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>error</code>: Error flag. Error code is returned through this variable. <ul style="list-style-type: none"> 0 - no error 1 - stop bit error 255 - user abort, Soft_UART_Break called
Requires	Software UART must be initialized before using this function. See the Soft_UART_Init routine.
Example	<pre> var data : byte; error : byte; ... // wait until data is received repeat data := Soft_UART_Read(error); until (error=0); // Now we can work with data: if (data) then begin ... end </pre>

Soft_UART_Write

Prototype	<code>procedure Soft_UART_Write(udata: byte);</code>
Returns	Nothing.
Description	<p>This routine sends one byte via the Software UART bus.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>udata</code>: data to be sent.
Requires	<p>Software UART must be initialized before using this function. See the <code>Soft_UART_Init</code> routine.</p> <p>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.</p>
Example	<pre>var some_byte : byte; ... // Write a byte via Soft UART some_byte := 0x0A; Soft_UART_Write(some_byte);</pre>

Soft_UART_Break

Prototype	<code>procedure Soft_UART_Break();</code>
Returns	Nothing.
Description	<p>Soft_UART_Read is blocking routine and it can block the program flow. Call this routine from interrupt to unblock the program execution. This mechanism is similar to WDT.</p> <p>Note: Interrupts should be disabled before using Software UART routines again (see note at the top of this page).</p>
Requires	Nothing.
Example	<pre>var datal, error, counter : byte; procedure interrupt(); begin if (INTCON.TOIF <> 0) then if (counter >= 20) then begin Soft_UART_Break(); counter := 0; // reset counter end else Inc(counter); // increment counter INTCON.TOIF := 0; // Clear Timer0 overflow inter- rupt flag end; begin counter := 0; OPTION_REG := 0x04; // TMR0 prescaler set to 1:32 ... if (Soft_UART_Init(PORTC, 7, 6, 9600, 0) = 0) then Soft_UART_Write(0x55); ... // try Soft_UART_Read with blocking prevention mechanism INTCON.GIE := 1; // Global interrupt enable INTCON.TOIE := 1; // Enable Timer0 overflow inter- rupt datal := Soft_UART_Read(error); INTCON.GIE := 0; // Global interrupt disable end.</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 mikroPascal PRO for PIC USART Terminal Tool.

```

program Soft_UART;

var error : byte;
    counter, byte_read : byte;           // Auxiliary variables

begin

    ANSEL  := 0;           // Configure AN pins as digital I/O
    ANSELH := 0;

    TRISB := 0x00;       // Set PORTB as output (error signalization)
    PORTB := 0;          // No error

    error := Soft_UART_Init(PORTC, 7, 6, 14400, 0); // Initialize Soft
UART at 14400 bps
    if (error > 0) then
        begin
            PORTB := error;           // Signalize Init error
            while (TRUE) do nop;     // Stop program
        end;
        Delay_ms(100);

    for counter := 'z' downto 'A' do           // Send bytes from 'z'
downto 'A'
        begin
            Soft_UART_Write(counter);
            Delay_ms(100);
        end;

    while TRUE do                               // Endless loop
        begin
            byte_read := Soft_UART_Read(error); // Read byte, then test
error flag
            if (error <> 0) then             // If error was detected
                PORTB := error               // signal it on PORTB
            else
                Soft_UART_Write(byte_read); // If error was not detect-
ed, return byte read
            end;
    end.

```

SOUND LIBRARY

The mikroPascal PRO for PIC 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).

Library Routines

- Sound_Init
- Sound_Play

Sound_Init

Prototype	<code>procedure Sound_Init(var snd_port : byte; snd_pin : byte);</code>
Returns	Nothing.
Description	Configures the appropriate MCU pin for sound generation. Parameters : - <code>snd_port</code> : sound output port address - <code>snd_pin</code> : sound output pin
Requires	Nothing.
Example	<code>Sound_Init(PORTD, 3); // Initialize sound at RD3</code>

Sound_Play

Prototype	<code>procedure Sound_Play(freq_in_hz, duration_ms : word);</code>
Returns	Nothing.
Description	<p>Generates the square wave signal on the appropriate pin.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>freq_in_hz</code>: signal frequency in Hertz (Hz) - <code>duration_ms</code>: signal duration in milliseconds (ms) <p>Note: frequency range is limited by Delay_Cyc parameter. Maximum frequency that can be produced by this function is $\text{Freq_max} = \text{Fosc} / (80 * 3)$. Minimum frequency is $\text{Freq_min} = \text{Fosc} / (80 * 255)$. Generated frequency may differ from the <code>freq_in_hz</code> parameter due to integer arithmetics.</p>
Requires	In order to hear the sound, you need a piezo speaker (or other hardware) on designated port. Also, you must call Sound_Init to prepare hardware for output before using this function.
Example	<pre>// Play sound of 1KHz in duration of 100ms Sound_Play(1000, 100);</pre>

Library Example

The example is a simple demonstration of how to use the Sound Library for playing tones on a piezo speaker.

```
program Sound;

procedure Tone1();
begin
    Sound_Play(659, 250);           // Frequency = 659Hz, duration = 250ms
end;

procedure Tone2();
begin
    Sound_Play(698, 250);           // Frequency = 698Hz, duration = 250ms
end;

procedure Tone3();
begin
    Sound_Play(784, 250);           // Frequency = 784Hz, duration = 250ms
end;
```

```

procedure Melody();           // Plays the melody "Yellow house"
begin
    Tone1(); Tone2(); Tone3(); Tone3();
    Tone1(); Tone2(); Tone3(); Tone3();
    Tone1(); Tone2(); Tone3();
    Tone1(); Tone2(); Tone3(); Tone3();
    Tone1(); Tone2(); Tone3();
    Tone3(); Tone3(); Tone2(); Tone2(); Tone1();
end;

procedure ToneA();           // Tones used in Melody2 function
begin
    Sound_Play( 880, 50);
end;

procedure ToneC();
begin
    Sound_Play(1046, 50);
end;

procedure ToneE();
begin
    Sound_Play(1318, 50);
end;

procedure Melody2();         // Plays Melody2
var counter : byte;
begin
    for counter := 9 downto 1 do
        begin
            ToneA();
            ToneC();
            ToneE();
        end;
    end;

begin

    ANSEL := 0;                // Configure AN pins as digital I/O
    ANSELH := 0;

    C1ON_bit := 0;            // Disable comparators
    C2ON_bit := 0;

    TRISB := 0xF0;           // Configure RB7..RB4 as input
    TRISD := 0xF7           // Configure RD3 as output

    Sound_Init(PORTD, 3);
    Sound_Play(880, 5000);

```

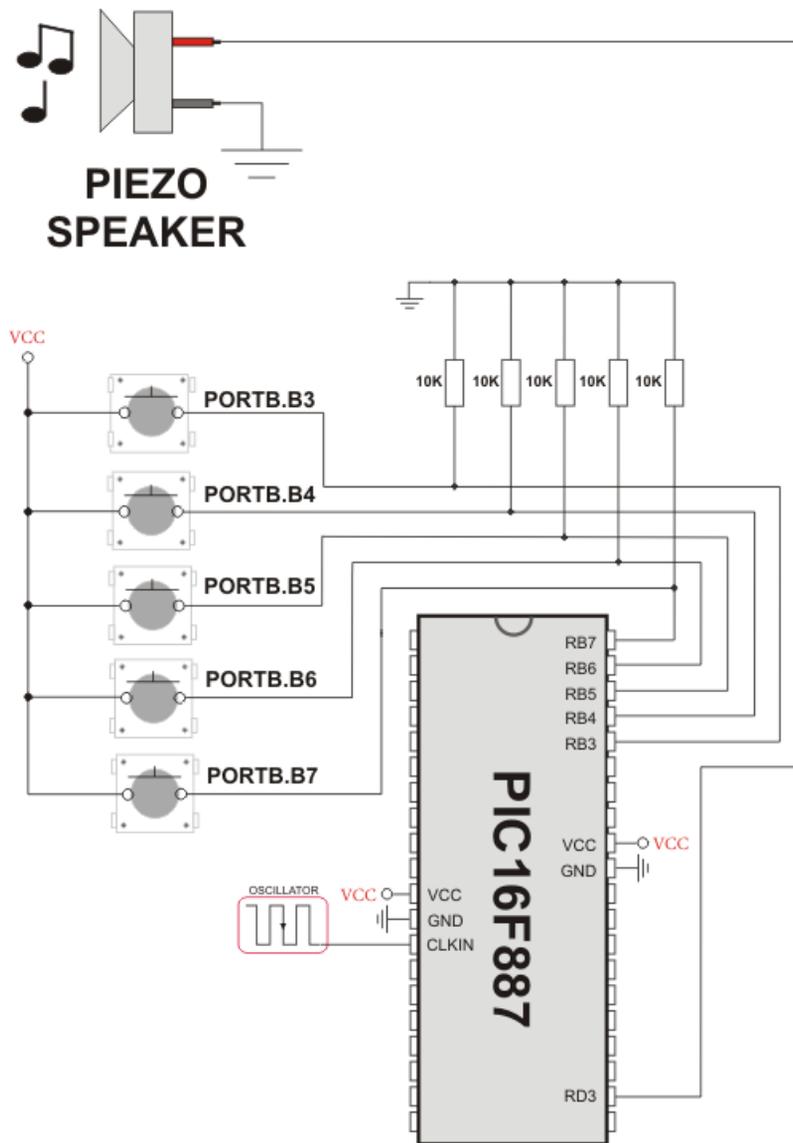
```
while TRUE do // endless loop
begin
  if (Button(PORTB,7,1,1)) then // If PORTB.7 is pressed play Tone1
  begin
    Tone1();
    while (RB7_bit <> 0) do nop; // Wait for button to be released
  end;

  if (Button(PORTB,6,1,1)) then // If PORTB.6 is pressed play Tone1
  begin
    Tone2();
    while (RB6_bit <> 0) do nop; // Wait for button to be released
  end;

  if (Button(PORTB,5,1,1)) then // If PORTB.5 is pressed play Tone1
  begin
    Melody2();
    while (RB5_bit <> 0) do nop; // Wait for button to be released
  end;

  if (Button(PORTB,4,1,1)) then // If PORTB.4 is pressed play Tone1
  begin
    Melody();
    while (RB4_bit <> 0) do nop; // Wait for button to be released
  end;
end;
end.
```

HW Connection



Example of Sound Library sonnection

SPI LIBRARY

SPI module is available with a number of PIC MCU models. mikroPascal PRO for PIC provides a library for initializing Slave mode and comfortable work with Master mode. PIC can easily communicate with other devices via SPI: A/D converters, D/A converters, MAX7219, LTC1290, etc. You need PIC MCU with hardware integrated SPI (for example, PIC16F877).

Note: Some PIC18 MCUs have multiple SPI modules. Switching between the SPI modules in the SPI library is done by the SPI_Set_Active function (SPI module has to be previously initialized).

Note: In order to use the desired SPI library routine, simply change the number **1** in the prototype with the appropriate module number, i.e. SPI2_Init();

Library Routines

- SPI1_Init
- SPI1_Init_Advanced
- SPI1_Read
- SPI1_Write
- SPI_Set_Active

SPI1_Init

Prototype	<code>procedure SPI1_Init();</code>
Returns	Nothing.
Description	<p>Configures and initializes SPI with default settings. SPI1_Init_Advanced or SPI1_Init needs to be called before using other functions from SPI Library.</p> <p>Default settings are:</p> <ul style="list-style-type: none"> - master mode - clock Fosc/4 - clock idle state low - data transmitted on low to high edge - input data sampled at the middle of interval
Requires	You need PIC MCU with hardware integrated SPI.
Example	<code>SPI1_Init(); // Initialize the SPI module with default settings</code>

SPI1_Init_Advanced

Prototype	<code>procedure SPI1_Init_Advanced(master, data_sample, clock_idle, transmit_edge : byte);</code>																																		
Returns	Nothing.																																		
Description	Configures and initializes SPI. SPI1_Init_Advanced or SPI1_Init needs to be called before using other functions of SPI Library.																																		
	Parameters <code>mode</code> , <code>data_sample</code> and <code>clock_idle</code> configure the SPI module, and can have the following values:																																		
	<table border="1"> <thead> <tr> <th>Description</th> <th>Predefined library const</th> </tr> </thead> <tbody> <tr> <td colspan="2" style="text-align: center;">SPI work mode:</td> </tr> <tr> <td>Master clock = $F_{osc}/4$</td> <td><code>_SPI_MASTER_OSC_DIV4</code></td> </tr> <tr> <td>Master clock = $F_{osc}/16$</td> <td><code>_SPI_MASTER_OSC_DIV16</code></td> </tr> <tr> <td>Master clock = $F_{osc}/64$</td> <td><code>_SPI_MASTER_OSC_DIV64</code></td> </tr> <tr> <td>Master clock source TMR2</td> <td><code>_SPI_MASTER_TMR2</code></td> </tr> <tr> <td>Slave select enabled</td> <td><code>_SPI_SLAVE_SS_ENABLE</code></td> </tr> <tr> <td>Slave select disabled</td> <td><code>_SPI_SLAVE_SS_DIS</code></td> </tr> <tr> <td colspan="2" style="text-align: center;">Data sampling interval:</td> </tr> <tr> <td>Input data sampled in middle of interval</td> <td><code>_SPI_DATA_SAMPLE_MIDDLE</code></td> </tr> <tr> <td>Input data sampled at the end of interval</td> <td><code>_SPI_DATA_SAMPLE_END</code></td> </tr> <tr> <td colspan="2" style="text-align: center;">SPI clock idle state:</td> </tr> <tr> <td>Clock idle HIGH</td> <td><code>_SPI_CLK_IDLE_HIGH</code></td> </tr> <tr> <td>Clock idle LOW</td> <td><code>_SPI_CLK_IDLE_LOW</code></td> </tr> <tr> <td colspan="2" style="text-align: center;">Transmit edge:</td> </tr> <tr> <td>Data transmit on low to high edge</td> <td><code>_SPI_LOW_2_HIGH</code></td> </tr> <tr> <td>Data transmit on high to low edge</td> <td><code>_SPI_HIGH_2_LOW</code></td> </tr> </tbody> </table>	Description	Predefined library const	SPI work mode:		Master clock = $F_{osc}/4$	<code>_SPI_MASTER_OSC_DIV4</code>	Master clock = $F_{osc}/16$	<code>_SPI_MASTER_OSC_DIV16</code>	Master clock = $F_{osc}/64$	<code>_SPI_MASTER_OSC_DIV64</code>	Master clock source TMR2	<code>_SPI_MASTER_TMR2</code>	Slave select enabled	<code>_SPI_SLAVE_SS_ENABLE</code>	Slave select disabled	<code>_SPI_SLAVE_SS_DIS</code>	Data sampling interval:		Input data sampled in middle of interval	<code>_SPI_DATA_SAMPLE_MIDDLE</code>	Input data sampled at the end of interval	<code>_SPI_DATA_SAMPLE_END</code>	SPI clock idle state:		Clock idle HIGH	<code>_SPI_CLK_IDLE_HIGH</code>	Clock idle LOW	<code>_SPI_CLK_IDLE_LOW</code>	Transmit edge:		Data transmit on low to high edge	<code>_SPI_LOW_2_HIGH</code>	Data transmit on high to low edge	<code>_SPI_HIGH_2_LOW</code>
	Description	Predefined library const																																	
	SPI work mode:																																		
	Master clock = $F_{osc}/4$	<code>_SPI_MASTER_OSC_DIV4</code>																																	
	Master clock = $F_{osc}/16$	<code>_SPI_MASTER_OSC_DIV16</code>																																	
	Master clock = $F_{osc}/64$	<code>_SPI_MASTER_OSC_DIV64</code>																																	
	Master clock source TMR2	<code>_SPI_MASTER_TMR2</code>																																	
	Slave select enabled	<code>_SPI_SLAVE_SS_ENABLE</code>																																	
	Slave select disabled	<code>_SPI_SLAVE_SS_DIS</code>																																	
	Data sampling interval:																																		
	Input data sampled in middle of interval	<code>_SPI_DATA_SAMPLE_MIDDLE</code>																																	
	Input data sampled at the end of interval	<code>_SPI_DATA_SAMPLE_END</code>																																	
	SPI clock idle state:																																		
Clock idle HIGH	<code>_SPI_CLK_IDLE_HIGH</code>																																		
Clock idle LOW	<code>_SPI_CLK_IDLE_LOW</code>																																		
Transmit edge:																																			
Data transmit on low to high edge	<code>_SPI_LOW_2_HIGH</code>																																		
Data transmit on high to low edge	<code>_SPI_HIGH_2_LOW</code>																																		
Requires	You need PIC MCU with hardware integrated SPI.																																		
Example	<pre>// Set SPI to master mode, clock = Fosc/4, data sampled at the // middle of interval, clock idle state low and data transmitted at // low to high edge: SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV4, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);</pre>																																		

SPI1_Read

Prototype	<code>function SPI1_Read(buffer : byte) : byte;</code>
Returns	Returns the received data.
Description	Provides clock by sending buffer and receives data at the end of period.
Requires	You need PIC MCU with hardware integrated SPI. SPI must be initialized and communication established before using this function. See SPI1_Init_Advanced or SPI1_Init.
Example	<pre>var buffer, take : byte; begin take := SPI1_Read(buffer); end.</pre>

SPI1_Write

Prototype	<code>procedure SPI1_Write(data_ : byte);</code>
Returns	Nothing.
Description	Writes byte data to SSPBUF, and immediately starts the transmission.
Requires	You need PIC MCU with hardware integrated SPI. SPI must be initialized and communication established before using this function. See SPI1_Init_Advanced or SPI1_Init.
Example	<code>SPI1_Write(1);</code>

SPI_Set_Active

Prototype	<code>procedure SPI_Set_Active (read_ptr : ^TSpi_Rd_Ptr);</code>
Returns	Nothing.
Description	Sets the active SPI module which will be used by the SPI routines. Parameters : - <code>read_ptr</code> : SPI1_Read handler
Requires	Routine is available only for MCUs with two SPI modules. Used SPI module must be initialized before using this function. See the SPI1_Init, SPI1_Init_Advanced
Example	<code>SPI_Set_Active(@SPI2_Read); // Sets the SPI2 module active</code>

Library Example

The code demonstrates how to use SPI library functions for communication between SPI module of the MCU and Microchip's MCP4921 12-bit D/A converter

```
program SPI;

// DAC module connections
var Chip_Select : sbit at RCl_bit;
    Chip_Select_Direction : sbit at TRISCl_bit;
// End DAC module connections

var value : word;

procedure InitMain();
begin
    TRISB0_bit := 1;           // Set RA0 pin as input
    TRISB1_bit := 1;           // Set RA1 pin as input
    Chip_Select := 1;          // Deselect DAC
    Chip_Select_Direction := 0; // Set CS# pin as Output
    SPI1_Init();               // Initialize SPI module
end;

// DAC increments (0..4095) --> output voltage (0..Vref)
procedure DAC_Output( valueDAC : word);
var temp : byte;
begin
    Chip_Select := 0;           // Select DAC chip

    // Send High Byte
    temp := word(valueDAC shr 8) and 0x0F; // Store
valueDAC[11..8] to temp[3..0]
    temp := temp or 0x30; // Define DAC setting, see MCP4921 datasheet
    SPI1_Write(temp);        // Send high byte via SPI

    // Send Low Byte
    temp := valueDAC; // Store valueDAC[7..0] to temp[7..0]
    SPI1_Write(temp); // Send low byte via SPI

    Chip_Select := 1; // Deselect DAC chip
end;

begin

    ANSEL := 0;
    ANSELH := 0;

    InitMain(); // Perform main initialization
```


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 PIC with integrated SPI and more than 4 Kb ROM memory. 38 to 40 MHz clock is recommended to get from 8 to 10 Mhz SPI clock, otherwise PIC should be clocked by ENC28J60 clock output due to its silicon bug in SPI hardware. If you try lower PIC clock speed, there might be board hang or miss some requests.

SPI Ethernet library supports:

- IPv4 protocol.
- ARP requests.
- ICMP echo requests.
- UDP requests.
- TCP requests (no stack, no packet reconstruction).
- ARP client with cache.
- DNS client.
- UDP client.
- DHCP client.
- packet fragmentation is **NOT** supported.

Note: Due to PIC16 RAM/Flash limitations PIC16 library does **NOT** have ARP, DNS, UDP and DHCP client support implemented.

Note: Global library variable `SPI_Ethernet_userTimerSec` is used to keep track of time for all client implementations (ARP, DNS, UDP and DHCP). It is user responsibility to increment this variable each second in it's code if any of the clients is used.

Note: For advanced users there are header files ("`eth_enc28j60LibDef.h`" and "`eth_enc28j60LibPrivate.h`") in Uses\P16 and Uses\P18 folders of the compiler with description of all routines and global variables, relevant to the user, implemented in the SPI Ethernet Library.

Note: The appropriate hardware SPI module must be initialized before using any of the SPI Ethernet library routines. Refer to SPI Library. For MCUs with two SPI modules it is possible to initialize both of them and then switch by using the `SPI_Set_Active()` routine.

External dependencies of SPI Ethernet Library

The following variables must be defined in all projects using SPI Ethernet Library:	Description:	Example :
<code>var SPI_Ethernet_CS : sbit; sfr; external;</code>	ENC28J60 chip select pin.	<code>var SPI_Ethernet_CS : sbit at RC1_bit;</code>
<code>var SPI_Ethernet_RST : sbit; sfr; external;</code>	ENC28J60 reset pin.	<code>var SPI_Ethernet_RST : sbit at RC0_bit;</code>
<code>var SPI_Ethernet_CS_Direction : sbit; sfr; external;</code>	Direction of the ENC28J60 chip select pin.	<code>var SPI_Ethernet_CS_Direction : sbit at TRISC1_bit;</code>
<code>var SPI_Ethernet_RST_Direction : sbit; sfr; external;</code>	Direction of the ENC28J60 reset pin.	<code>var SPI_Ethernet_RST_Direction : sbit at TRISCO_bit;</code>

The following routines must be defined in all project using SPI Ethernet Library:	Description:	Example :
<code>function SPI_Ethernet_UserTCP(remoteHost : ^byte, remotePort : word, localPort : word, reqLength : word): word;</code>	TCP request handler.	Refer to the library example at the bottom of this page for code implementation.
<code>function SPI_Ethernet_UserUDP(remoteHost : ^byte, remotePort : word, destPort : word, reqLength : word): word;</code>	UDP request handler.	Refer to the library example at the bottom of this page for code implementation.

Library Routines

PIC16 and PIC18:

- 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

PIC18 Only:

- SPI_Ethernet_getIpAddress
- SPI_Ethernet_getGwIpAddress
- SPI_Ethernet_getDnsIpAddress
- SPI_Ethernet_getIpMask
- SPI_Ethernet_confNetwork
- SPI_Ethernet_arpResolve
- SPI_Ethernet_sendUDP
- SPI_Ethernet_dnsResolve
- SPI_Ethernet_initDHCP
- SPI_Ethernet_doDHCPLeaseTime
- SPI_Ethernet_renewDHCP

SPI_Ethernet_Init

Prototype	<code>procedure SPI_Ethernet_Init(mac: ^byte; ip: ^byte; fullDuplex: byte);</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It initializes <code>ENC28J60</code> controller. This function is internally splitted into 2 parts to help linker when coming short of memory.</p> <p><code>ENC28J60</code> controller settings (parameters not mentioned here are set to default):</p> <ul style="list-style-type: none"> - receive buffer start address : <code>0x0000</code>. - receive buffer end address : <code>0x19AD</code>. - transmit buffer start address: <code>0x19AE</code>. - transmit buffer end address : <code>0x1FFF</code>. - 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 <code>1518</code>. - Back-to-Back Inter-Packet Gap: <code>0x15</code> in full duplex mode; <code>0x12</code> in half duplex mode. - Non-Back-to-Back Inter-Packet Gap: <code>0x0012</code> in full duplex mode; <code>0x0C12</code> in half duplex mode. - Collision window is set to 63 in half duplex mode to accomodate some <code>ENC28J60</code> 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	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>SPI_Ethernet_CS</code>: Chip Select line - <code>SPI_Ethernet_CS_Direction</code>: Direction of the Chip Select pin - <code>SPI_Ethernet_RST</code>: Reset line - <code>SPI_Ethernet_RST_Direction</code>: Direction of the Reset pin <p>must be defined before using this function.</p> <p>The SPI module needs to be initialized. See the <code>SPI1_Init</code> and <code>SPI1_Init_Advanced</code> routines.</p>

Example

```
// SPI Ethernet module connections
var SPI_Ethernet_RST : sbit at RC0_bit;
var SPI_Ethernet_CS : sbit at RC1_bit;
var SPI_Ethernet_RST_Direction : sbit at TRISC0_bit;
var SPI_Ethernet_CS_Direction : sbit at TRISC1_bit;

const SPI_Ethernet_HALFDUPLEX = 0;
const SPI_Ethernet_FULLDUPLEX = 1;

var
  myMacAddr : array[ 6] of byte; // my MAC address
  myIpAddr  : array[ 4] of byte; // my IP addr
  ...
  myMacAddr[ 0] := 0x00;
  myMacAddr[ 1] := 0x14;
  myMacAddr[ 2] := 0xA5;
  myMacAddr[ 3] := 0x76;
  myMacAddr[ 4] := 0x19;
  myMacAddr[ 5] := 0x3F;

  myIpAddr[ 0] := 192;
  myIpAddr[ 1] := 168;
  myIpAddr[ 2] := 1;
  myIpAddr[ 3] := 60;

  SPI_Init();
  SPI_Ethernet_Init(myMacAddr, myIpAddr, SPI_Ethernet_FULLDUPLEX);
```

SPI_Ethernet_Enable

Prototype	<code>procedure SPI_Ethernet_Enable(enFlt : 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"> - enFlt: 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	
		<code>_SPI_Ethernet_BROADCAST</code>	
		<code>_SPI_Ethernet_MULTICAST</code>	
		none	
		none	
		none	
		<code>_SPI_Ethernet_CRC</code>	
		none	
		<code>_SPI_Ethernet_UNICAST</code>	
	<p>Note: Advance filtering available in the ENC28J60 module such as <code>Pattern Match</code>, <code>Magic Packet</code> and <code>Hash Table</code> 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>		

Description	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 configured by the means of SPI_Ethernet_Init routine.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	<pre>SPI_Ethernet_Enable(_SPI_Ethernet_CRC or _SPI_Ethernet_UNICAST); // enable CRC checking and Unicast traffic</pre>

SPI_Ethernet_Disable

Prototype	<code>procedure SPI_Ethernet_Disable(disFlt : 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: <table border="1"> <thead> <tr> <th>Bit</th> <th>Mask</th> <th>Description</th> <th>Predefined library const</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0x01</td> <td>MAC Broadcast traffic/receive filter flag. When set, MAC broadcast traffic will be disabled.</td> <td><code>_SPI_Ethernet_BROADCAST</code></td> </tr> <tr> <td>1</td> <td>0x02</td> <td>MAC Multicast traffic/receive filter flag. When set, MAC multicast traffic will be disabled.</td> <td><code>_SPI_Ethernet_MULTICAST</code></td> </tr> <tr> <td>2</td> <td>0x04</td> <td>not used</td> <td>none</td> </tr> <tr> <td>3</td> <td>0x08</td> <td>not used</td> <td>none</td> </tr> <tr> <td>4</td> <td>0x10</td> <td>not used</td> <td>none</td> </tr> <tr> <td>5</td> <td>0x20</td> <td>CRC check flag. When set, CRC check will be disabled and packets with invalid CRC field will be accepted.</td> <td><code>_SPI_Ethernet_CRC</code></td> </tr> <tr> <td>6</td> <td>0x40</td> <td>not used</td> <td>none</td> </tr> <tr> <td>7</td> <td>0x80</td> <td>MAC Unicast traffic/receive filter flag. When set, MAC unicast traffic will be disabled.</td> <td><code>_SPI_Ethernet_UNICAST</code></td> </tr> </tbody> </table>	Bit	Mask	Description	Predefined library const	0	0x01	MAC Broadcast traffic/receive filter flag. When set, MAC broadcast traffic will be disabled.	<code>_SPI_Ethernet_BROADCAST</code>	1	0x02	MAC Multicast traffic/receive filter flag. When set, MAC multicast traffic will be disabled.	<code>_SPI_Ethernet_MULTICAST</code>	2	0x04	not used	none	3	0x08	not used	none	4	0x10	not used	none	5	0x20	CRC check flag. When set, CRC check will be disabled and packets with invalid CRC field will be accepted.	<code>_SPI_Ethernet_CRC</code>	6	0x40	not used	none	7	0x80	MAC Unicast traffic/receive filter flag. When set, MAC unicast traffic will be disabled.	<code>_SPI_Ethernet_UNICAST</code>
Bit	Mask	Description	Predefined library const																																		
0	0x01	MAC Broadcast traffic/receive filter flag. When set, MAC broadcast traffic will be disabled.	<code>_SPI_Ethernet_BROADCAST</code>																																		
1	0x02	MAC Multicast traffic/receive filter flag. When set, MAC multicast traffic will be disabled.	<code>_SPI_Ethernet_MULTICAST</code>																																		
2	0x04	not used	none																																		
3	0x08	not used	none																																		
4	0x10	not used	none																																		
5	0x20	CRC check flag. When set, CRC check will be disabled and packets with invalid CRC field will be accepted.	<code>_SPI_Ethernet_CRC</code>																																		
6	0x40	not used	none																																		
7	0x80	MAC Unicast traffic/receive filter flag. When set, MAC unicast traffic will be disabled.	<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	<pre>SPI_Ethernet_Disable(_SPI_Ethernet_CRC or _SPI_Ethernet_UNICAST); // disable CRC checking and Unicast traffic</pre>

SPI_Ethernet_doPacket

Prototype	<code>function SPI_Ethernet_doPacket() : 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 do begin ... SPI_Ethernet_doPacket(); // process received packets ... end;</pre>

SPI_Ethernet_putByte

Prototype	<code>procedure SPI_Ethernet_putByte(v : byte);</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It stores one byte to address pointed by the current ENC28J60 write pointer (<code>EWRPT</code>).</p> <p>Parameters:</p> <ul style="list-style-type: none">- <code>v</code>: value to store
Requires	Ethernet module has to be initialized. See <code>SPI_Ethernet_Init</code> .
Example	<pre>var data : byte; ... SPI_Ethernet_putByte(data); // put an byte into ENC28J60 buffer</pre>

SPI_Ethernet_putBytes

Prototype	<code>procedure SPI_Ethernet_putBytes(ptr : ^byte; n : byte);</code>
Returns	Nothing.
Description	<p>This is MAC module routine. It stores requested number of bytes into ENC28J60 RAM starting from current ENC28J60 write pointer (<code>EWRPT</code>) location.</p> <p>Parameters:</p> <ul style="list-style-type: none">- <code>ptr</code>: RAM buffer containing bytes to be written into ENC28J60 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>var buffer : array[17] of byte; ... buffer := 'mikroElektronika'; ... SPI_Ethernet_putBytes(buffer, 16); // put an RAM array into ENC28J60 buffer</pre>

SPI_Ethernet_putConstBytes

Prototype	<code>procedure SPI_Ethernet_putConstBytes(const ptr : ^byte; n : 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"> - <code>ptr</code>: const buffer containing bytes to be written into ENC28J60 RAM. - <code>n</code>: number of bytes to be written.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	<pre>const buffer : array[17] of byte; ... buffer := 'mikroElektronika'; ... SPI_Ethernet_putConstBytes(buffer, 16); // put a const array into ENC28J60 buffer</pre>

SPI_Ethernet_putString

Prototype	<code>function SPI_Ethernet_putString(ptr : ^byte) : 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"> - <code>ptr</code>: string to be written into ENC28J60 RAM.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	<pre>var buffer : string[16]; ... buffer := 'mikroElektronika'; ... SPI_Ethernet_putString(buffer); // put a RAM string into ENC28J60 buffer</pre>

SPI_Ethernet_putConstString

Prototype	<code>function SPI_Ethernet_putConstString(const ptr : ^byte) : word;</code>
Returns	Number of bytes written into ENC28J60 RAM.
Description	<p>This is MAC module routine. It stores whole const string (excluding null termination) into ENC28J60 RAM starting from current ENC28J60 write pointer (EWRPT) location.</p> <p>Parameters:</p> <p>- <code>ptr</code>: const string to be written into ENC28J60 RAM.</p>
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	<pre>const buffer : string[16]; ... buffer := 'mikroElektronika'; ... SPI_Ethernet_putConstString(buffer); // put a const string into ENC28J60 buffer</pre>

SPI_Ethernet_getByte

Prototype	<code>function SPI_Ethernet_getByte() : byte;</code>
Returns	Byte read from ENC28J60 RAM.
Description	<p>This is MAC module routine. It fetches a byte from address pointed to by current ENC28J60 read pointer (ERDPT).</p>
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	<pre>var buffer : byte; ... buffer := SPI_Ethernet_getByte(); // read a byte from ENC28J60 buffer</pre>

SPI_Ethernet_getBytes

Prototype	<code>procedure SPI_Ethernet_getBytes(ptr : ^byte; addr : word; n : 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>var buffer: array[16] of byte; ... SPI_Ethernet_getBytes(buffer, 0x100, 16); // read 16 bytes, starting from address 0x100</pre>

SPI_Ethernet_UserTCP

Prototype	<code>function SPI_Ethernet_UserTCP(var remoteHost : array[4] of byte; remotePort, localPort, reqLength : word) : word;</code>
Returns	- 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>function SPI_Ethernet_UserUDP(var remoteHost : array[4] of byte; remotePort, destPort, reqLength : word) : word;</code>
Returns	- 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.

SPI_Ethernet_getIpAddress

Prototype	<code>function SPI_Ethernet_getIpAddress() : word;</code>
Returns	Ponter to the global variable holding IP address.
Description	<p>This routine should be used when DHCP server is present on the network to fetch assigned IP address.</p> <p>Note: User should always copy the IP address from the RAM location returned by this routine into it's own IP address buffer. These locations should not be altered by the user in any case!</p>
Requires	<p>Ethernet module has to be initialized. See SPI_Ethernet_Init.</p> <p>Available for PIC18 family MCUs only.</p>
Example	<pre>var ipAddr : array[4] of byte; // user IP address buffer ... memcpy(ipAddr, SPI_Ethernet_getIpAddress(), 4); // fetch IP address</pre>

SPI_Ethernet_getGwIpAddress

Prototype	<code>function SPI_Ethernet_getGwIpAddress() : word;</code>
Returns	Ponter to the global variable holding gateway IP address.
Description	<p>This routine should be used when DHCP server is present on the network to fetch assigned gateway IP address.</p> <p>Note: User should always copy the IP address from the RAM location returned by this routine into it's own gateway IP address buffer. These locations should not be altered by the user in any case!</p>
Requires	<p>Ethernet module has to be initialized. See SPI_Ethernet_Init.</p> <p>Available for PIC18 family MCUs only.</p>
Example	<pre>var gwIpAddr : array[4] of byte; // user gateway IP address buffer ... memcpy(gwIpAddr, SPI_Ethernet_getGwIpAddress(), 4); // fetch gateway IP address</pre>

SPI_Ethernet_getDnsIpAddress

Prototype	<code>function SPI_Ethernet_getDnsIpAddress() : word;</code>
Returns	Ponter to the global variable holding DNS IP address.
Description	<p>This routine should be used when DHCP server is present on the network to fetch assigned DNS IP address.</p> <p>Note: User should always copy the IP address from the RAM location returned by this routine into it's own DNS IP address buffer. These locations should not be altered by the user in any case!</p>
Requires	<p>Ethernet module has to be initialized. See SPI_Ethernet_Init.</p> <p>Available for PIC18 family MCUs only.</p>
Example	<pre>var dnsIpAddr : array[4] of byte; // user DNS IP address buffer ... memcpy(dnsIpAddr, SPI_Ethernet_getDnsIpAddress(), 4); // fetch DNS server address</pre>

SPI_Ethernet_getIpMask

Prototype	<code>function SPI_Ethernet_getIpMask() : word;</code>
Returns	Ponter to the global variable holding IP subnet mask.
Description	<p>This routine should be used when DHCP server is present on the network to fetch assigned IP subnet mask.</p> <p>Note: User should always copy the IP address from the RAM location returned by this routine into it's own IP subnet mask buffer. These locations should not be altered by the user in any case!</p>
Requires	<p>Ethernet module has to be initialized. See SPI_Ethernet_Init.</p> <p>Available for PIC18 family MCUs only.</p>
Example	<pre>var IpMask : array[4] of byte; // user IP subnet mask buffer ... memcpy(IpMask, SPI_Ethernet_getIpMask(), 4); // fetch IP subnet mask</pre>

SPI_Ethernet_confNetwork

Prototype	<code>procedure SPI_Ethernet_confNetwork(var ipMask, gwIpAddr, dnsIpAddr : array[4] of byte);</code>
Returns	Nothing.
Description	<p>Configures network parameters (IP subnet mask, gateway IP address, DNS IP address) when DHCP is not used.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>ipMask</code>: IP subnet mask. - <code>gwIpAddr</code> gateway IP address. - <code>dnsIpAddr</code>: DNS IP address. <p>Note: The above mentioned network parameters should be set by this routine only if DHCP module is not used. Otherwise DHCP will override these settings.</p>
Requires	<p>Ethernet module has to be initialized. See <code>SPI_Ethernet_Init</code>.</p> <p>Available for PIC18 family MCUs only.</p>
Example	<pre> var ipMask : array[4] of byte; // network mask (for example : 255.255.255.0) gwIpAddr : array[4] of byte; // gateway (router) IP address dnsIpAddr : array[4] of byte; // DNS server IP address ... gwIpAddr[0] := 192; gwIpAddr[1] := 168; gwIpAddr[2] := 20; gwIpAddr[3] := 6; dnsIpAddr[0] := 192; dnsIpAddr[1] := 168; dnsIpAddr[2] := 20; dnsIpAddr[3] := 100; ipMask[0] := 255; ipMask[1] := 255; ipMask[2] := 255; ipMask[3] := 0; ... SPI_Ethernet_confNetwork(ipMask, gwIpAddr, dnsIpAddr); // set network configuration parameters </pre>

SPI_Ethernet_arpResolve

Prototype	<code>function SPI_Ethernet_arpResolve(var ip : array[4] of byte; tmax : byte) : word;</code>
Returns	- MAC address behind the IP address - the requested IP address was resolved. - 0 - otherwise.
Description	<p>This is ARP module routine. It sends an ARP request for given IP address and waits for ARP reply. If the requested IP address was resolved, an ARP cash entry is used for storing the configuration. ARP cash can store up to 3 entries. For ARP cash structure refer to "eth_enc28j60LibDef.h" header file in the compiler's Uses/P18 folder.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - ip: IP address to be resolved. - tmax: time in seconds to wait for an reply. <p>Note: The Ethernet services are not stopped while this routine waits for ARP reply. The incoming packets will be processed normaly during this time.</p>
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init. Available for PIC18 family MCUs only.
Example	<pre> var IpAddr : array[4] of byte; // IP address ... IpAddr[0] := 192; IpAddr[1] := 168; IpAddr[2] := 1; IpAddr[3] := 1; ... SPI_Ethernet_arpResolve(IpAddr, 5); // get MAC address behind the above IP address, wait 5 secs for the response </pre>

SPI_Ethernet_sendUDP

Prototype	<code>function SPI_Ethernet_sendUDP(var destIP : array[4] of byte; sourcePort, destPort : word; pkt : ^byte; pktLen : word) : byte;</code>
Returns	- 1 - UDP packet was sent successfully. - 0 - otherwise.
Description	This is UDP module routine. It sends an UDP packet on the network. Parameters: - <code>destIP</code> : remote host IP address. - <code>sourcePort</code> : local UDP source port number. - <code>destPort</code> : destination UDP port number. - <code>pkt</code> : packet to transmit. - <code>pktLen</code> : length in bytes of packet to transmit.
Requires	Ethernet module has to be initialized. See <code>SPI_Ethernet_Init</code> . Available for PIC18 family MCUs only.
Example	<pre>var IpAddr : array[4] of byte; // remote IP address ... IpAddr[0] := 192; IpAddr[1] := 168; IpAddr[2] := 1; IpAddr[3] := 1; ... SPI_Ethernet_sendUDP(IpAddr, 10001, 10001, 'Hello', 5); // send Hello message to the above IP address, from UDP port 10001 to UDP port 10001</pre>

SPI_Ethernet_dnsResolve

Prototype	<code>function SPI_Ethernet_dnsResolve(var host : array[4] of byte; tmax : byte) : word;</code>
Returns	- pointer to the location holding the IP address - the requested host name was resolved. - 0 - otherwise.
Description	<p>This is DNS module routine. It sends an DNS request for given host name and waits for DNS reply. If the requested host name was resolved, it's IP address is stored in library global variable and a pointer containing this address is returned by the routine. UDP port 53 is used as DNS port.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>host</code>: host name to be resolved. - <code>tmax</code>: time in seconds to wait for an reply. <p>Note: The Ethernet services are not stopped while this routine waits for DNS reply. The incoming packets will be processed normaly during this time.</p> <p>Note: User should always copy the IP address from the RAM location returned by this routine into it's own resolved host IP address buffer. These locations should not be altered by the user in any case!</p>
Requires	Ethernet module has to be initialized. See <code>SPI_Ethernet_Init</code> . Available for PIC18 family MCUs only.
Example	<pre>var remoteHostIpAddr : array[4] of byte; // user host IP address buffer ... // SNMP server: // Zurich, Switzerland: Integrated Systems Lab, Swiss Fed. Inst. of Technology // 129.132.2.21: swisstime.ethz.ch // Service Area: Switzerland and Europe memcpy(remoteHostIpAddr, SPI_Ethernet_dnsResolve('swisstime.ethz.ch', 5), 4);</pre>

SPI_Ethernet_initDHCP

Prototype	<code>function SPI_Ethernet_initDHCP(tmax : byte) : byte;</code>
Returns	- 1 - network parameters were obtained successfully. - 0 - otherwise.
Description	<p>This is DHCP module routine. It sends an DHCP request for network parameters (IP, gateway, DNS addresses and IP subnet mask) and waits for DHCP reply. If the requested parameters were obtained successfully, their values are stored into the library global variables.</p> <p>These parameters can be fetched by using appropriate library IP get routines:</p> <ul style="list-style-type: none"> - SPI_Ethernet_getIpAddress - fetch IP address. - SPI_Ethernet_getGwIpAddress - fetch gateway IP address. - SPI_Ethernet_getDnsIpAddress - fetch DNS IP address. - SPI_Ethernet_getIpMask - fetch IP subnet mask. <p>UDP port 68 is used as DHCP client port and UDP port 67 is used as DHCP server port.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - <code>tmax</code>: time in seconds to wait for an reply. <p>Note: The Ethernet services are not stopped while this routine waits for DNS reply. The incoming packets will be processed normally during this time.</p> <p>Note: When DHCP module is used, global library variable <code>SPI_Ethernet_userTimerSec</code> is used to keep track of time. It is user responsibility to increment this variable each second in it's code.</p>
Requires	<p>Ethernet module has to be initialized. See <code>SPI_Ethernet_Init</code>.</p> <p>Available for PIC18 family MCUs only.</p>
Example	<pre> ... SPI_Ethernet_initDHCP(5); // get network configuration from DHCP server, wait 5 sec for the response ... </pre>

SPI_Ethernet_doDHCPLeaseTime

Prototype	<code>function SPI_Ethernet_doDHCPLeaseTime() : byte;</code>
Returns	- 0 - lease time has not expired yet. - 1 - lease time has expired, it's time to renew it.
Description	This is DHCP module routine. It takes care of IP address lease time by decrementing the global lease time library counter. When this time expires, it's time to contact DHCP server and renew the lease.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init. Available for PIC18 family MCUs only.
Example	<pre> while true do begin ... if (SPI_Ethernet_doDHCPLeaseTime() <> 0) then begin ... // it's time to renew the IP address lease end; end; end; </pre>

SPI_Ethernet_renewDHCP

Prototype	<code>function SPI_Ethernet_renewDHCP(tmax : byte) : byte;</code>
Returns	- 1 - upon success (lease time was renewed). - 0 - otherwise (renewal request timed out).
Description	This is DHCP module routine. It sends IP address lease time renewal request to DHCP server. Parameters: - <code>tmax</code> : time in seconds to wait for an reply.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init. Available for PIC18 family MCUs only.
Example	<pre> while true do begin ... if (SPI_Ethernet_doDHCPLeaseTime() <> 0) then begin SPI_Ethernet_renewDHCP(5); // it's time to renew the IP address lease, with 5 secs for a reply end; ... end; </pre>

Library Example

This code shows how to use the PIC18 Ethernet mini 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 RD0 to RD7 bit and return HTML main page
 - all other requests return also HTML main page.

```
program enc_ethernet;

{ *****
  * RAM variables
  *}
var myMacAddr   : array[ 6] of byte ; // my MAC address
    myIpAddr    : array[ 4] of byte ; // my IP address

// mE ethernet NIC pinout
  SPI_Ethernet_Rst : sbit at PORTC.B0;
  SPI_Ethernet_CS  : sbit at PORTC.B1;
  SPI_Ethernet_Rst_Direction : sbit at TRISC.B0;
  SPI_Ethernet_CS_Direction  : sbit at TRISC.B1;
// end ethernet NIC definitions

{ *****
  * ROM constant strings
  *}
const httpHeader : string[ 30] = 'HTTP/1.1 200 OK'+#10+'Content-type:
' ; // HTTP header
const httpMimeTypeHTML : string[ 11] = 'text/html'+#10+#10 ;
// HTML MIME type
const httpMimeTypeScript : string[ 12] = 'text/plain'+#10+#10 ;
// TEXT MIME type
const httpMethod : string[ 5] = 'GET /';
{ *
  * web page, splitted 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 indexPage : string[ 759] =
    '<meta http-equiv="refresh"
content="3;url=http://192.168.20.60">' +
    '<HTML><HEAD></HEAD><BODY>'+
    '<h1>PIC + ENC28J60 Mini Web Server</h1>'+
    '<a href=/>Reload</a>'+
    '<script src=/s></script>'+
    '<table><tr><td valign=top><table border=1
style="font-size:20px ;font-family: terminal ;">'+
    '<tr><th colspan=2>ADC</th></tr>'+

'<tr><td>AN2</td><td><script>document.write(AN2)</script></td></tr>'
+
'<tr><td>AN3</td><td><script>document.write(AN3)</script></td></tr>'
+
    '</table></td><td><table border=1 style="font-
size:20px ;font-family: terminal ;">'+
    '<tr><th colspan=2>PORTB</th></tr>'+
    '<script>'+
    'var str,i;'+
    'str="";'+
    'for(i=0;i<8;i++)'+
    '{ str+="<tr><td bgcolor=pink>BUTTON #" +i+"</td>";'+
    'if(PORTB&(1<<i)){ str+="<td bgcolor=red>ON";}' +
    'else { str+="<td bgcolor=#cccccc>OFF";}' +
    'str+="</td></tr>";}' +
    'document.write(str) ;'+
    '</script>';

const indexPage2 : string[ 466] =
    '</table></td><td>'+
    '<table border=1 style="font-size:20px ;font-
family: terminal ;">'+
    '<tr><th colspan=3>PORTD</th></tr>'+
    '<script>'+
    'var str,i;'+
    'str="";'+
    'for(i=0;i<8;i++)'+
    '{ str+="<tr><td bgcolor=yellow>LED #" +i+"</td>";'+
    'if(PORTD&(1<<i)){ str+="<td bgcolor=red>ON";}' +
    'else { str+="<td bgcolor=#cccccc>OFF";}' +
    'str+="</td><td><a
href=/t"+i+">Toggle</a></td></tr>";}' +
    'document.write(str) ;'+
    '</script>'+
    '</table></td></tr></table>'+
    'This is HTTP request
#<script>document.write(REQ)</script></BODY></HTML>';

```

```
var    getRequest  : array[ 15] of byte; // HTTP request buffer
      dyna         : array[ 30] of byte; // buffer for dynamic
response
      tmp: string[ 11];                // auxiliary ram string
      httpCounter : word ;             // counter of HTTP requests

{*****}
* user defined functions
*}

{ *
* 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
*
*}

function SPI_Ethernet_UserTCP(var remoteHost : array[ 4] of byte;
                             remotePort, localPort, reqLength
: word) : word;
var  i : word ;           // my reply length
     bitMask : byte ; // for bit mask
begin
    result := 0;

    if(localPort <> 80) then // I listen only to web request on port
80
        begin
            result := 0;
            exit;
        end;

    // get 10 first bytes only of the request, the rest does not mat-
ter here
    for i := 0 to 9 do
        getRequest[ i ] := SPI_Ethernet_getByte() ;
        getRequest[ i ] := 0 ;

    // copy httpMethod to ram for use in memcmp routine
    for i := 0 to 4 do
        tmp[ i ] := httpMethod[ i ];
```

```
    if(memcmp(@getRequest, @tmp, 5) <> 0) then // only GET method is
supported here
    begin
        result := 0 ;
        exit;
    end;

    Inc(httpCounter) ; // one more request done

    if(getRequest[5] = 's') then // if request path name starts with
s, store dynamic data in transmit buffer
    begin
        // the text string replied by this request can be interpret-
ed as javascript statements
        // by browsers
        result := SPI_Ethernet_putConstString(@httpHeader) ;
// HTTP header
        result := result +
SPI_Ethernet_putConstString(@httpMimeTypeScript) ; // with text MIME
type

        // add AN2 value to reply
        WordToStr(ADC_Read(2), dyna) ;
        tmp := 'var AN2=';
        result := result + SPI_Ethernet_putString(@tmp) ;
        result := result + SPI_Ethernet_putString(@dyna) ;
        tmp := ';' ;
        result := result + SPI_Ethernet_putString(@tmp) ;

        // add AN3 value to reply
        WordToStr(ADC_Read(3), dyna) ;
        tmp := 'var AN3=';
        result := result + SPI_Ethernet_putString(@tmp) ;
        result := result + SPI_Ethernet_putString(@dyna) ;
        tmp := ';' ;
        result := result + SPI_Ethernet_putString(@tmp) ;

        // add PORTB value (buttons) to reply
        tmp := 'var PORTB=' ;
        result := result + SPI_Ethernet_putString(@tmp) ;
        WordToStr(PORTB, dyna) ;
        result := result + SPI_Ethernet_putString(@dyna) ;
        tmp := ';' ;
        result := result + SPI_Ethernet_putString(@tmp) ;

        // add PORTD value (LEDs) to reply
        tmp := 'var PORTD=' ;
        result := result + SPI_Ethernet_putString(@tmp) ;
        WordToStr(PORTD, dyna) ;
        result := result + SPI_Ethernet_putString(@dyna) ;
        tmp := ';' ;
        result := result + SPI_Ethernet_putString(@tmp) ;
```

```
        // add HTTP requests counter to reply
        WordToStr(httpCounter, dyna) ;
        tmp := 'var REQ=  ';
        result := result + SPI_Ethernet_putString(@tmp) ;
        result := result + SPI_Ethernet_putString(@dyna) ;
        tmp := ';;';
        result := result + SPI_Ethernet_putString(@tmp) ;
    end
else
    if(getRequest[ 5] = 't') then // if request path
name starts with t, toggle PORTD (LED) bit number that comes after
        begin
            bitMask := 0;
            if(isdigit(getRequest[ 6] ) <> 0) then // if 0 <= bit num-
ber <= 9, bits 8 & 9 does not exist but does not matter
                begin
                    bitMask := getRequest[ 6] - '0' ; // convert ASCII to
integer
                    bitMask := 1 shl bitMask ; // create bit mask
                    PORTD := PORTD xor bitMask ; // toggle PORTD with
xor operator
                end;
            end;

            if(result = 0) then // what do to by default
                begin
                    result := SPI_Ethernet_putConstString(@httpHeader) ; // HTTP
header
                    result := result +
SPI_Ethernet_putConstString(@httpMimeTypeHTML) ; // with HTML MIME
type
                    result := result + SPI_Ethernet_putConstString(@indexPath) ;
// HTML page first part
                    result := result + SPI_Ethernet_putConstString(@indexPath2)
; // HTML page second part
                end;
            // return to the library with the number of bytes to transmit
end;

{ *
* 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
*
*}
function SPI_Ethernet_UserUDP(var remoteHost : array[ 4] of byte;
remotePort, destPort, reqLength
: word) : word;
```

```
begin
    result := 0;
    // reply is made of the remote host IP address in human readable
format
    byteToStr(remoteHost[ 0], dyna) ; // first IP address byte
    dyna[ 3] := '.' ;
    byteToStr(remoteHost[ 1], tmp) ; // second
    dyna[ 4] := tmp[ 0];
    dyna[ 5] := tmp[ 1];
    dyna[ 6] := tmp[ 2];
    dyna[ 7] := '.' ;
    byteToStr(remoteHost[ 2], tmp) ; // second
    dyna[ 8] := tmp[ 0];
    dyna[ 9] := tmp[ 1];
    dyna[ 10] := tmp[ 2];
    dyna[ 11] := '.' ;
    byteToStr(remoteHost[ 3], tmp) ; // second
    dyna[ 12] := tmp[ 0];
    dyna[ 13] := tmp[ 1];
    dyna[ 14] := tmp[ 2];

    dyna[ 15] := ':' ; // add separator

    // then remote host port number
    WordToStr(remotePort, tmp) ;
    dyna[ 16] := tmp[ 0];
    dyna[ 17] := tmp[ 1];
    dyna[ 18] := tmp[ 2];
    dyna[ 19] := tmp[ 3];
    dyna[ 20] := tmp[ 4];
    dyna[ 21] := '[' ;
    WordToStr(destPort, tmp) ;
    dyna[ 22] := tmp[ 0];
    dyna[ 23] := tmp[ 1];
    dyna[ 24] := tmp[ 2];
    dyna[ 25] := tmp[ 3];
    dyna[ 26] := tmp[ 4];
    dyna[ 27] := ']' ;
    dyna[ 28] := 0 ;

    // the total length of the request is the length of the dynamic
string plus the text of the request
    result := 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) do
```

```
begin
    SPI_Ethernet_putByte(SPI_Ethernet_getByte()) ;
    reqLength := reqLength - 1;
end;
// back to the library with the length of the UDP reply
end;

begin
    ANSEL := 0x0C ;           // AN2 and AN3 convertors will be used
    PORTA := 0 ;
    TRISA := 0xff ;          // set PORTA as input for ADC

    ANSELH := 0 ;           // Configure other AN pins as digital I/O
    PORTB := 0 ;
    TRISB := 0xff ;         // set PORTB as input for buttons

    PORTD := 0 ;
    TRISD := 0 ;           // set PORTD as output

    httpCounter := 0;

    // set mac address
    myMacAddr[ 0] := 0x00;
    myMacAddr[ 1] := 0x14;
    myMacAddr[ 2] := 0xA5;
    myMacAddr[ 3] := 0x76;
    myMacAddr[ 4] := 0x19;
    myMacAddr[ 5] := 0x3F;

    // set IP address
    myIpAddr[ 0] := 192;
    myIpAddr[ 1] := 168;
    myIpAddr[ 2] := 20;
    myIpAddr[ 3] := 60;

    { *
      * starts ENC28J60 with :
      * reset bit on PORTC.B0
      * CS bit on PORTC.B1
      * my MAC & IP address
      * full duplex
      * }

    SPI1_Init(); // init spi module
    SPI_Ethernet_Init(myMacAddr, myIpAddr, _SPI_Ethernet_FULLLDUPLEX) ;
// init ethernet module
    SPI_Ethernet_setUserHandlers(@SPI_Ethernet_UserTCP,
    @SPI_Ethernet_UserUDP); // set user handlers
```

```

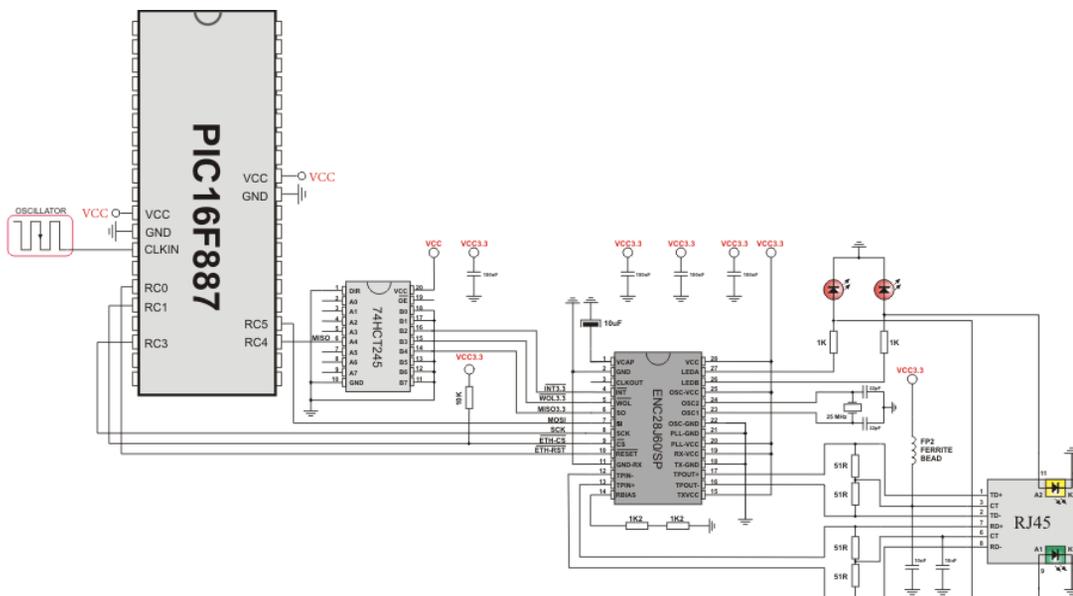
while true do // do forever
begin
    SPI_Ethernet_doPacket() ; // process incoming Ethernet packets

    { *
      * add your stuff here if needed
      * SPI_Ethernet_doPacket() must be called as often as possible
      * otherwise packets could be lost
      * }

end;
end.

```

HW Connection



SPI GRAPHIC LCD LIBRARY

The mikroPascal PRO for PIC 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.

For MCUs with two SPI modules it is possible to initialize both of them and then switch by using the SPI_Set_Active() routine.

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>procedure SPI_Glcd_Init(DeviceAddress : 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>Global variables :</p> <ul style="list-style-type: none"> - <code>SPExpanderCS</code>: Chip Select line - <code>SPExpanderRST</code>: Reset line - <code>SPExpanderCS_Direction</code>: Direction of the Chip Select pin - <code>SPExpanderRST_Direction</code>: Direction of the Reset pin <p>must be defined before using this function.</p> <p>SPI module needs to be initialized. See <code>SPI1_Init</code> and <code>SPI1_Init_Advanced</code> routines.</p>
Example	<pre>// port expander pinout definition var SPExpanderCS : sbit at RC1_bit; SPExpanderRST : sbit at RC0_bit; SPExpanderCS_Direction : sbit at TRISC1_bit; SPExpanderRST_Direction : sbit at TRISC_bit; ... // If Port Expander Library uses SPI1 module : SPI1_Init(); // Initialize SPI module used with PortExpander SPI_Glcd_Init(0);</pre>

SPI_Glcd_Set_Side

Prototype	<code>procedure SPI_Glcd_Set_Side(x_pos : 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 SPI_Glcd_Init 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>procedure SPI_Glcd_Set_Page(page : 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 SPI_Glcd_Init routines.
Example	<code>SPI_Glcd_Set_Page(5);</code>

SPI_Glcd_Set_X

Prototype	<code>procedure SPI_Glcd_Set_X(x_pos : 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 SPI_Glcd_Init routines.
Example	<code>SPI_Glcd_Set_X(25);</code>

SPI_Glcd_Read_Data

Prototype	<code>function SPI_Glcd_Read_Data() : 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	<p>Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routines.</p> <p>Glcd side, x-axis position and page should be set first. See the functions SPI_Glcd_Set_Side, SPI_Glcd_Set_X, and SPI_Glcd_Set_Page.</p>
Example	<pre>var data : byte; ... data := SPI_Glcd_Read_Data();</pre>

SPI_Glcd_Write_Data

Prototype	<code>procedure SPI_Glcd_Write_Data(ddata : 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 SPI_Glcd_Init routines. Glcd side, x-axis position and page should be set first. See the functions SPI_Glcd_Set_Side, SPI_Glcd_Set_X, and SPI_Glcd_Set_Page.
Example	<pre>var ddata : byte; ... SPI_Glcd_Write_Data(ddata);</pre>

SPI_Glcd_Fill

Prototype	<code>procedure SPI_Glcd_Fill(pattern: 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 SPI_Glcd_Fill(0). To fill the screen completely, use SPI_Glcd_Fill(0xFF).
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routines.
Example	<pre>// Clear screen SPI_Glcd_Fill(0);</pre>

SPI_Glcd_Dot

Prototype	<code>procedure SPI_Glcd_Dot(x_pos : byte; y_pos : byte; color : 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 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	<pre>// Invert the dot in the upper left corner SPI_Glcd_Dot(0, 0, 2);</pre>

SPI_Glcd_Line

Prototype	<code>procedure SPI_Glcd_Line(x_start : integer; y_start : integer; x_end : integer; y_end : integer; color : 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 SPI_Glcd_Init routines.
Example	<pre>// Draw a line between dots (0,0) and (20,30) SPI_Glcd_Line(0, 0, 20, 30, 1);</pre>

SPI_Glcd_V_Line

Prototype	<code>procedure SPI_Glcd_V_Line(y_start: byte; y_end: byte; x_pos: byte; color: 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 SPI_Glcd_Init routines.
Example	<pre>// Draw a vertical line between dots (10,5) and (10,25) SPI_Glcd_V_Line(5, 25, 10, 1);</pre>

SPI_Glcd_H_Line

Prototype	<code>procedure SPI_Glcd_V_Line(x_start : byte; x_end : byte; y_pos : byte; color : 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 SPI_Glcd_Init routines.
Example	<pre>// Draw a horizontal line between dots (10,20) and (50,20) SPI_Glcd_H_Line(10, 50, 20, 1);</pre>

SPI_Glcd_Rectangle

Prototype	<code>procedure SPI_Glcd_Rectangle(x_upper_left : byte; y_upper_left : byte; x_bottom_right : byte; y_bottom_right : byte; color : 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 SPI_Glcd_Init routines.
Example	<pre>// Draw a rectangle between dots (5,5) and (40,40) SPI_Glcd_Rectangle(5, 5, 40, 40, 1);</pre>

SPI_Glcd_Box

Prototype	<code>procedure SPI_Glcd_Box(x_upper_left : byte; y_upper_left : byte; x_bottom_right : byte; y_bottom_right : byte; color : 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 color 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 SPI_Glcd_Init routines.
Example	<pre>// Draw a box between dots (5,15) and (20,40) SPI_Glcd_Box(5, 15, 20, 40, 1);</pre>

SPI_Glcd_Circle

Prototype	<code>procedure SPI_Glcd_Circle(x_center : integer; y_center : integer; radius : integer; color : 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 color 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 SPI_Glcd_Init routine.
Example	<pre>// Draw a circle with center in (50,50) and radius=10 SPI_Glcd_Circle(50, 50, 10, 1);</pre>

SPI_Glcd_Set_Font

Prototype	<code>procedure SPI_Glcd_Set_Font(activeFont : longint; aFontWidth : byte; aFontHeight : byte; aFontOffs : 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 mikroPascal character set and regular ASCII set (eg. if 'A' is 65 in ASCII character, and 'A' is 45 in the mikroPascal 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 "<code>__Lib_GLCD_fonts.mpas</code>" 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	<pre>// Use the custom 5x7 font "myfont" which starts with space (32): SPI_Glcd_Set_Font(@myfont, 5, 7, 32);</pre>

SPI_Glcd_Write_Char

Prototype	<code>procedure SPI_Glcd_Write_Char(chr1 : byte; x_pos : byte; page_num : byte; color : 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 SPI_Glcd_Init routines.</p> <p>Use the SPI_Glcd_Set_Font 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 character 'C' on the position 10 inside the page 2: SPI_Glcd_Write_Char("C", 10, 2, 1);</code>

SPI_Glcd_Write_Text

Prototype	<pre>procedure SPI_Glcd_Write_Text(var text : array[40] of byte; x_pos : byte; page_num : byte; color : byte);</pre>
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 SPI_Glcd_Init routines.</p> <p>Use the SPI_Glcd_Set_Font 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 text "Hello world!" on the position 10 inside the page 2: SPI_Glcd_Write_Text("Hello world!", 10, 2, 1);</pre>

SPI_Glcd_Image

Prototype	<code>procedure SPI_Glcd_Image(const image : ^byte);</code>
Returns	Nothing.
Description	<p>Displays bitmap on Glcd.</p> <p>Parameters :</p> <ul style="list-style-type: none">- <code>image</code>: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroPascal PRO for PIC pointer to const and pointer to RAM equivalency). <p>Use the mikroPascal'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;

uses bitmap;

// Port Expander module connections
var SPExpanderRST : sbit at RC0_bit;
    SPExpanderCS   : sbit at RC1_bit;
    SPExpanderRST_Direction : sbit at TRISC0_bit;
    SPExpanderCS_Direction  : sbit at TRISC1_bit;
// End Port Expander module connections

var someText : array[20] of char;
    counter : byte;

procedure Delay2S;
begin
    Delay_ms(2000);
end;

begin
```

```
ANSEL := 0; // Configure AN pins as
digital
ANSELH := 0;
C1ON_bit := 0; // Disable comparators
C2ON_bit := 0;

// If Port Expander Library uses SPI1 module
SPI1_Init(); // Initialize SPI module used with PortExpander

// // If Port Expander Library uses SPI2 module
// SPI2_Init(); // Initialize SPI module used with PortExpander

SPI_Glcd_Init(0); // Initialize Glcd via SPI
SPI_Glcd_Fill(0x00); // Clear Glcd

while (TRUE) do
  begin

    SPI_Glcd_Image(@truck_bmp); // Draw image
    Delay2s(); Delay2s();

    SPI_Glcd_Fill(0x00); // Clear Glcd
    Delay2s;

    SPI_Glcd_Box(62,40,124,63,1); // Draw box
    SPI_Glcd_Rectangle(5,5,84,35,1); // Draw rectangle
    SPI_Glcd_Line(0, 0, 127, 63, 1); // Draw line
    Delay2s();
    counter := 5;
    while (counter < 60) do // Draw horizontal and vertical line
      begin
        Delay_ms(250);
        SPI_Glcd_V_Line(2, 54, counter, 1);
        SPI_Glcd_H_Line(2, 120, counter, 1);
        counter := counter + 5;
      end;
    Delay2s();

    SPI_Glcd_Fill(0x00); // Clear Glcd
    SPI_Glcd_Set_Font(@Character8x7, 8, 7, 32); // Choose font,
see __Lib_GLCDFonts.c in Uses folder
    SPI_Glcd_Write_Text('mikroE', 1, 7, 2); // Write string

    for counter := 1 to 10 do// Draw circles
      SPI_Glcd_Circle(63,32, 3*counter, 1);
      Delay2s();

    SPI_Glcd_Box(10,20, 70,63, 2); // Draw box
    Delay2s();
```

```
SPI_Glcd_Fill(0xFF); // Fill Glcd

SPI_Glcd_Set_Font(@Character8x7, 8, 7, 32); // Change font
someText := '8x7 Font';
SPI_Glcd_Write_Text(someText, 5, 0, 2); // Write string
Delay2s();

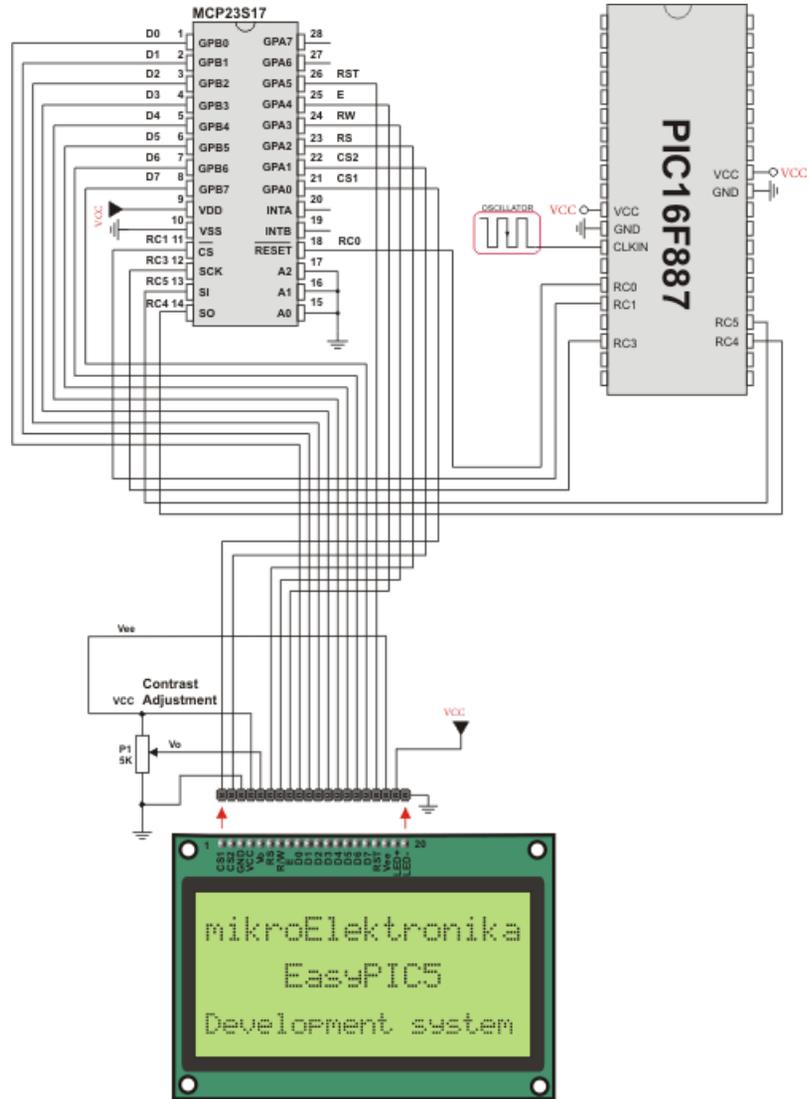
SPI_Glcd_Set_Font(@System3x6, 3, 5, 32); // Change font
someText := '3X5 CAPITALS ONLY';
SPI_Glcd_Write_Text(someText, 60, 2, 2); // Write string
Delay2s();

SPI_Glcd_Set_Font(@font5x7, 5, 7, 32); // Change font
someText := '5x7 Font';
SPI_Glcd_Write_Text(someText, 5, 4, 2); // Write string
Delay2s();

SPI_Glcd_Set_Font(@FontSystem5x7_v2, 5, 7, 32); // Change font
someText := '5x7 Font (v2)';
SPI_Glcd_Write_Text(someText, 50, 6, 2); // Write string
Delay2s();

end;
end.
```

HW Connection



SPI Glcd HW connection

SPI LCD LIBRARY

The mikroPascal PRO for PIC 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.

For MCUs with two SPI modules it is possible to initialize both of them and then switch by using the SPI_Set_Active() routine.

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>procedure SPI_Lcd_Config(DeviceAddress: 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>Global variables :</p> <ul style="list-style-type: none"> - <code>SPExpanderCS</code>: Chip Select line - <code>SPExpanderRST</code>: Reset line - <code>SPExpanderCS_Direction</code>: Direction of the Chip Select pin - <code>SPExpanderRST_Direction</code>: Direction of the Reset pin <p>must be defined before using this function.</p> <p>SPI module needs to be initialized. See <code>SPI1_Init</code> and <code>SPI1_Init_Advanced</code> routines.</p>
Example	<pre>// port expander pinout definition var SPExpanderCS : sbit at RC1_bit; SPExpanderRST : sbit at RC0_bit; SPExpanderCS_Direction : sbit at TRISC1_bit; SPExpanderRST_Direction : sbit at TRISC_bit; // If Port Expander Library uses SPI1 module SPI1_Init(); // Initialize SPI module used with PortExpander SPI_Lcd_Config(0); // initialize lcd over spi interface</pre>

SPI_Lcd_Out

Prototype	<code>procedure SPI_Lcd_Out(row: byte; column: byte; var text: array[20] of byte);</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 SPI_Lcd_Config routines.
Example	<code>// Write text "Hello!" on Lcd starting from row 1, column 3: SPI_Lcd_Out(1, 3, "Hello!");</code>

SPI_Lcd_Out_CP

Prototype	<code>procedure SPI_Lcd_Out_CP(var text : array[20] of byte);</code>
Returns	Nothing.
Description	<p>Prints text on the Lcd at current cursor position. Both string variables and literals can be passed as a text.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>text</code>: text to be written
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd_Config routines.
Example	<code>// Write text "Here!" at current cursor position: SPI_Lcd_Out_CP("Here!");</code>

SPI_Lcd_Chr

Prototype	<code>procedure SPI_Lcd_Chr(Row : byte; Column : byte; Out_Char : byte);</code>
Returns	Nothing.
Description	<p>Prints character on Lcd at specified position. Both variables and literals can be passed as character.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <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	<pre>// Write character "i" at row 2, column 3: SPI_Lcd_Chr(2, 3, 'i');</pre>

SPI_Lcd_Chr_CP

Prototype	<code>procedure SPI_Lcd_Chr_CP(Out_Char : byte);</code>
Returns	Nothing.
Description	<p>Prints character on Lcd at current cursor position. Both variables and literals can be passed as character.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>Out_Char</code>: character to be written
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd_Config routines.
Example	<pre>// Write character "e" at current cursor position: SPI_Lcd_Chr_CP('e');</pre>

SPI_Lcd_Cmd

Prototype	<code>procedure SPI_Lcd_Cmd(out_char : 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	Lcd needs to be initialized for SPI communication, see SPI_Lcd_Config routines.
Example	<pre>// Clear Lcd display: SPI_Lcd_Cmd(_LCD_CLEAR);</pre>

Available SPI 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

This example demonstrates how to communicate Lcd via the SPI module, using serial to parallel convertor MCP23S17.

```
program SPI_Lcd;

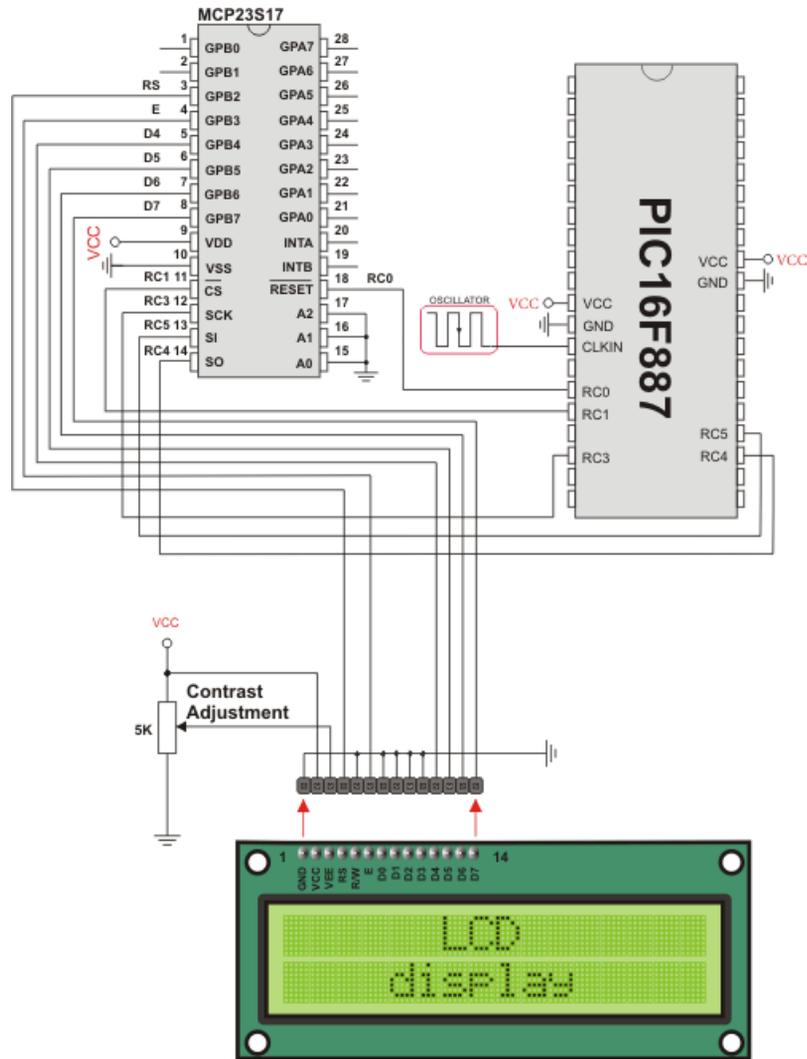
var text : array[16] of char;

// Port Expander module connections
var SPExpanderRST : sbit at RC0_bit;
    SPExpanderCS   : sbit at RC1_bit;
    SPExpanderRST_Direction : sbit at TRISC0_bit;
    SPExpanderCS_Direction  : sbit at TRISC1_bit;
// End Port Expander module connections

begin
    text := 'mikroElektronika';
    // If Port Expander Library uses SPI1 module
    SPI1_Init(); // Initialize SPI module used with PortExpander

    // If Port Expander Library uses SPI2 module
    // SPI2_Init(); // Initialize SPI module used with PortExpander
    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
end.
```

HW Connection



SPI Lcd HW connection

SPI LCD8 (8-BIT INTERFACE) LIBRARY

The mikroPascal PRO for PIC 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.

For MCUs with two SPI modules it is possible to initialize both of them and then switch by using the SPI_Set_Active() routine.

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>procedure SPI_Lcd8_Config(DeviceAddress : 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>Global variables :</p> <ul style="list-style-type: none"> - <code>SPExpanderCS</code>: Chip Select line - <code>SPExpanderRST</code>: Reset line - <code>SPExpanderCS_Direction</code>: Direction of the Chip Select pin - <code>SPExpanderRST_Direction</code>: Direction of the Reset pin <p>must be defined before using this function.</p> <p>SPI module needs to be initialized. See <code>SPI1_Init</code> and <code>SPI1_Init_Advanced</code> routines.</p>
Example	<pre>// port expander pinout definition var SPExpanderCS : sbit at RC1_bit; SPExpanderRST : sbit at RC0_bit; SPExpanderCS_Direction : sbit at TRISC1_bit; SPExpanderRST_Direction : sbit at TRISC_bit; SPI1_Init(); // Initialize SPI module used with PortExpander ... SPI_Lcd8_Config(0); // intialize lcd in 8bit mode via spi</pre>

SPI_Lcd8_Out

Prototype	<code>procedure SPI_Lcd8_Out(row: byte; column: byte; var text: array[20] of byte);</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	<code>// Write text "Hello!" on Lcd starting from row 1, column 3: SPI_Lcd8_Out(1, 3, "Hello!");</code>

SPI_Lcd8_Out_CP

Prototype	<code>procedure SPI_Lcd8_Out_CP(var text : array[20] of byte);</code>
Returns	Nothing.
Description	<p>Prints text on Lcd at current cursor position. Both string variables and literals can be passed as a text.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>text</code>: text to be written
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd8_Config routines.
Example	<code>// Write text "Here!" at current cursor position: SPI_Lcd8_Out_CP("Here!");</code>

SPI_Lcd8_Chr

Prototype	<code>procedure SPI_Lcd8_Chr(Row : byte; Column : byte; Out_Char : byte);</code>
Returns	Nothing.
Description	<p>Prints character on Lcd at specified position. Both variables and literals can be passed as character.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <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	<pre>// Write character "i" at row 2, column 3: SPI_Lcd8_Chr(2, 3, 'i');</pre>

SPI_Lcd8_Chr_CP

Prototype	<code>procedure SPI_Lcd8_Chr_CP(Out_Char : byte);</code>
Returns	Nothing.
Description	<p>Prints character on Lcd at current cursor position. Both variables and literals can be passed as character.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>out_char</code> : character to be written
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd8_Config routines.
Example	<p>Print "e" at current cursor position:</p> <pre>// Write character "e" at current cursor position: SPI_Lcd8_Chr_CP('e');</pre>

SPI_Lcd8_Cmd

Prototype	<code>procedure SPI_Lcd8_Cmd(out_char : 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	Lcd needs to be initialized for SPI communication, see SPI_Lcd8_Config routines.
Example	<pre>// Clear Lcd display: SPI_Lcd8_Cmd(_LCD_CLEAR);</pre>

Available SPI Lcd8 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

This example demonstrates how to communicate Lcd in 8-bit mode via the SPI module, using serial to parallel convertor MCP23S17.

```
program Spi_Lcd8_Test;

var text : array[16] of char;

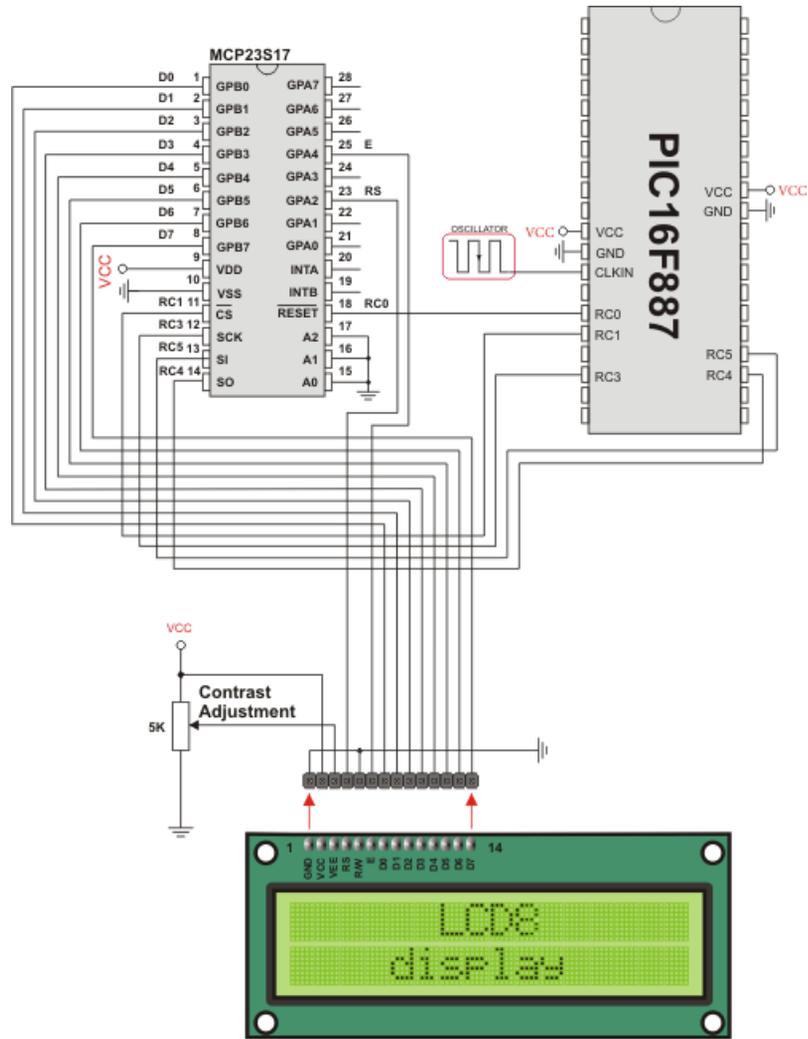
// Port Expander module connections
var SPExpanderRST : sbit at RC0_bit;
    SPExpanderCS   : sbit at RC1_bit;
    SPExpanderRST_Direction : sbit at TRISC0_bit;
    SPExpanderCS_Direction  : sbit at TRISC1_bit;
// End Port Expander module connections

begin
    text := 'mikroE';
    // If Port Expander Library uses SPI1 module
    SPI1_Init(); // Initialize SPI
module used with PortExpander

    // // If Port Expander Library uses SPI2 module
    // SPI2_Init(); // Initialize SPI
module used with PortExpander

    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 mikroPascal PRO for PIC 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.

For MCUs with two SPI modules it is possible to initialize both of them and then switch by using the `SPI_Set_Active()` routine.

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
- 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	<pre>procedure SPI_T6963C_Config(width : word; height : byte; fntW : byte; DeviceAddress : byte; wr : byte; rd : byte; cd : byte; rst : byte);</pre>
Returns	Nothing.
Description	<p>Initializes T6963C 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 #1 + + + +-----+ \</pre>

Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>SPExpanderCS</code>: Chip Select line - <code>SPExpanderRST</code>: Reset line - <code>SPExpanderCS_Direction</code>: Direction of the Chip Select pin - <code>SPExpanderRST_Direction</code>: Direction of the Reset pin <p>must be defined before using this function.</p> <p>SPI module needs to be initialized. See <code>SPI1_Init</code> and <code>SPI1_Init_Advanced</code> routines.</p>
Example	<pre>// port expander pinout definition var SPExpanderCS : sbit at RC1_bit; SPExpanderRST : sbit at RC0_bit; SPExpanderCS_Direction : sbit at TRISC1_bit; SPExpanderRST_Direction : sbit at TRISC_bit; ... // Initialize SPI module SPI1_Init(); SPI_T6963C_Config(240, 64, 8, 0, 0, 1, 3, 4);</pre>

SPI_T6963C_WriteData

Prototype	<code>procedure SPI_T6963C_WriteData(Ddata : byte);</code>
Returns	Nothing.
Description	<p>Writes data to T6963C controller via SPI interface.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>Ddata</code>: data to be written
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>procedure SPI_T6963C_WriteCommand(Ddata : byte);</code>
Returns	Nothing.
Description	Writes command to T6963C controller via SPI interface. Parameters : - <i>Ddata</i> : command to be written
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	<code>SPI_T6963C_WriteCommand(SPI_T6963C_CURSOR_POINTER_SET);</code>

SPI_T6963C_SetPtr

Prototype	<code>procedure SPI_T6963C_SetPtr(p : word; c : byte);</code>
Returns	Nothing.
Description	Sets the memory pointer p for command c. Parameters : - <i>p</i> : address where command should be written - <i>c</i> : 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>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>procedure SPI_T6963C_Fill(v : byte; start : word; len : word);</code>
Returns	Nothing.
Description	<p>Fills controller memory block with given byte.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>v</code>: byte to be written - <code>start</code>: starting address of the memory block - <code>len</code>: 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>procedure SPI_T6963C_Dot(x : integer; y : integer; color : 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>procedure SPI_T6963C_Write_Char(c : byte; x : byte; y : byte; mode : 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>procedure SPI_T6963C_write_text(var str : array[10] of byte; x, y, mode : 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>procedure SPI_T6963C_Line(x0 : integer; y0 : integer; x1 : integer; y1 : integer; pcolor : 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"> - 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: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	<code>SPI_T6963C_Line(0, 0, 239, 127, T6963C_WHITE);</code>

SPI_T6963C_Rectangle

Prototype	<code>procedure SPI_T6963C_Rectangle(x0 : integer; y0 : integer; x1 : integer; y1 : integer; pcolor : byte);</code>
Returns	Nothing.
Description	<p>Draws a rectangle on Glcd.</p> <p>Parameters :</p> <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: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	<code>SPI_T6963C_Rectangle(20, 20, 219, 107, T6963C_WHITE);</code>

SPI_T6963C_Box

Prototype	<code>procedure SPI_T6963C_Box(x0 : integer; y0 : integer; x1 : integer; y1 : integer; pcolor : 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: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	<code>SPI_T6963C_Box(0, 119, 239, 127, T6963C_WHITE);</code>

SPI_T6963C_Circle

Prototype	<code>procedure SPI_T6963C_Circle(x : integer; y : integer; r : longint; pcolor : word);</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: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	<code>SPI_T6963C_Circle(120, 64, 110, T6963C_WHITE);</code>

SPI_T6963C_Image

Prototype	<code>procedure SPI_T6963C_image(const pic : ^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 mikroPascal PRO for PIC pointer to const and pointer to RAM equivalency). <p>Use the mikroPascal's integrated Glcd Bitmap Editor (menu option Tools > Glcd Bitmap Editor) to convert image to a constant array suitable for displaying on Glcd.</p> <p>Note: Image dimension must match the display dimension.</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>procedure SPI_T6963C_sprite(px, py : byte; const pic : ^byte; sx, sy : 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>procedure SPI_T6963C_set_cursor(x, y : byte);</code>
Returns	Nothing.
Description	Sets cursor to row x and column y. Parameters : - 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>procedure SPI_T6963C_clearBit(b : byte);</code>
Returns	Nothing.
Description	Clears control port bit(s). Parameters : - 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>procedure SPI_T6963C_setBit(b : byte);</code>
Returns	Nothing.
Description	Sets control port bit(s). Parameters : - 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>procedure SPI_T6963C_negBit(b : byte);</code>
Returns	Nothing.
Description	<p>Negates control port bit(s).</p> <p>Parameters :</p> <p>- b: bit mask. The function will negate bit x on control port if bit x in bit mask is set to 1.</p>
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>procedure SPI_T6963C_DisplayGrPanel(n : byte);</code>
Returns	Nothing.
Description	<p>Display selected graphic panel.</p> <p>Parameters :</p> <p>- n: graphic panel number. Valid values: 0 and 1.</p>
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>procedure SPI_T6963C_DisplayGrPanel(n : 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_SetGrPanel

Prototype	<code>procedure SPI_T6963C_SetGrPanel(n : 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 : - n: 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>procedure SPI_T6963C_SetTxtPanel(n : 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 : - n : 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>procedure SPI_T6963C_PanelFill(v : byte);</code>
Returns	Nothing.
Description	Fill current panel in full (graphic+text) with appropriate value (0 to clear). Parameters : - v : 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>procedure SPI_T6963C_GrFill(v : byte);</code>
Returns	Nothing.
Description	Fill current graphic panel with appropriate value (0 to clear). Parameters : - v : 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>procedure SPI_T6963C_TxtFill(v : byte);</code>
Returns	Nothing.
Description	Fill current text panel with appropriate value (0 to clear). Parameters : - v : 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>procedure SPI_T6963C_Cursor_Height(n : byte);</code>
Returns	Nothing.
Description	Set cursor size. Parameters : - n : cursor height. Valid values: 0..7.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	<pre>SPI_T6963C_Cursor_Height(7);</pre>

SPI_T6963C_Graphics

Prototype	<code>procedure SPI_T6963C_Graphics(n : byte);</code>
Returns	Nothing.
Description	Enable/disable graphic displaying. Parameters : - n : 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	<pre>// enable graphic displaying SPI_T6963C_Graphics(1);</pre>

SPI_T6963C_Text

Prototype	<code>procedure SPI_T6963C_Text(n : byte);</code>
Returns	Nothing.
Description	<p>Enable/disable text displaying.</p> <p>Parameters :</p> <p>- n: text enable/disable parameter. Valid values: 0 (disable text displaying) and 1 (enable text displaying).</p>
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	<code>// enable text displaying SPI_T6963C_Text(1);</code>

SPI_T6963C_Cursor

Prototype	<code>procedure SPI_T6963C_Cursor(n : byte);</code>
Returns	Nothing.
Description	<p>Set cursor on/off.</p> <p>Parameters :</p> <p>- n: on/off parameter. Valid values: 0 (set cursor off) and 1 (set cursor on).</p>
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>procedure SPI_T6963C_Cursor_Blink(n : byte);</code>
Returns	Nothing.
Description	<p>Enable/disable cursor blinking.</p> <p>Parameters :</p> <p>- n: cursor blinking enable/disable parameter. Valid values: 0 (disable cursor blinking) and 1 (enable cursor blinking).</p>
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, EasyPIC5 board and 16F887.

```

program SPI_T6963C_240x128;
uses   __Lib_SPIT6963C_Const, bitmap, bitmap2;

var
// Port Expander module connections
    SPExpanderRST : sbit at RCO_bit;
    SPExpanderCS  : sbit at RC1_bit;
    SPExpanderRST_Direction : sbit at TRISCO_bit;
    SPExpanderCS_Direction  : sbit at TRISC1_bit;
// End Port Expander module connections

var   panel : byte;           // current panel
        i : word;             // general purpose register
        curs : byte;         // cursor visibility
        cposx,
        cposy : word;         // cursor x-y position
        txt, txt1 : string[ 29];

begin

    txt1 := ' EINSTEIN WOULD HAVE LIKED ME!';
    txt  := ' GLCD LIBRARY DEMO, WELCOME !!';

    ANSEL  := 0;                // Configure AN pins as digital I/O
    ANSELH := 0;
    C1ON_bit := 0;              // Disable comparators
    C2ON_bit := 0;

    TRISB0_bit := 1;           // Set RB0 as input
    TRISB1_bit := 1;           // Set RB1 as input
    TRISB2_bit := 1;           // Set RB2 as input
    TRISB3_bit := 1;           // Set RB3 as input
    TRISB4_bit := 1;           // Set RB4 as input

    // Initialize SPI module
    SPI1_Init();

    // // If Port Expander Library uses SPI2 module
    // Pass pointer to SPI Read function of used SPI module

    // Initialize SPI module used with PortExpander
    //      SPI2_Init_Advanced(_SPI_MASTER,      _SPI_FCY_DIV32,
    _SPI_CLK_HI_TRAILING);

```

```

{ *
  * 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 internaly set in library
  *}

// 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 ;
cposx := 0;
cposy := 0 ;

{ *
  * Text messages
  *}
SPI_T6963C_write_text(txt, 0, 0, SPI_T6963C_ROM_MODE_XOR) ;
SPI_T6963C_write_text(txt1, 0, 15, SPI_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, SPI_T6963C_WHITE) ;
SPI_T6963C_rectangle(20, 20, 219, 107, SPI_T6963C_WHITE) ;
SPI_T6963C_rectangle(40, 40, 199, 87, SPI_T6963C_WHITE) ;
SPI_T6963C_rectangle(60, 60, 179, 67, SPI_T6963C_WHITE) ;

```

```

{ *
  * Draw a cross
  *}
SPI_T6963C_line(0, 0, 239, 127, SPI_T6963C_WHITE) ;
SPI_T6963C_line(0, 127, 239, 0, SPI_T6963C_WHITE) ;

{ *
  * Draw solid boxes
  *}
SPI_T6963C_box(0, 0, 239, 8, SPI_T6963C_WHITE) ;
SPI_T6963C_box(0, 119, 239, 127, SPI_T6963C_WHITE) ;

{ *
  * Draw circles
  *}
SPI_T6963C_circle(120, 64, 10, SPI_T6963C_WHITE) ;
SPI_T6963C_circle(120, 64, 30, SPI_T6963C_WHITE) ;
SPI_T6963C_circle(120, 64, 50, SPI_T6963C_WHITE) ;
SPI_T6963C_circle(120, 64, 70, SPI_T6963C_WHITE) ;
SPI_T6963C_circle(120, 64, 90, SPI_T6963C_WHITE) ;
SPI_T6963C_circle(120, 64, 110, SPI_T6963C_WHITE) ;
SPI_T6963C_circle(120, 64, 130, SPI_T6963C_WHITE) ;

SPI_T6963C_sprite(76, 4, @einstein, 88, 119) ; // Draw a sprite

SPI_T6963C_setGrPanel(1) ; // Select other graphic panel

SPI_T6963C_sprite(0, 0, @me, 240, 64); // 240x128 can't
be stored in most of PIC16 MCUs
SPI_T6963C_sprite(0, 64, @me, 240, 64); // it is replaced
with smaller picture 240x64
// Smaller picture is drawn two times

while (TRUE) do // Endless loop
  begin

    { *
      * If PORTA_0 is pressed, toggle the display between graphic
      panel 0 and graphic 1
      *}
    if( RBO_bit <> 0) then
      begin
        Inc(panel) ;
        panel := panel and 1;
        SPI_T6963C_setPtr((SPI_T6963C_grMemSize +
SPI_T6963C_txtMemSize) * panel, SPI_T6963C_GRAPHIC_HOME_ADDRESS_SET)
        ;
        Delay_ms(300) ;
      end

```

```
{ *
 * If PORTA_1 is pressed, display only graphic panel
 *}
else
  if ( RB1_bit <> 0) then
    begin
      SPI_T6963C_graphics(1) ;
      SPI_T6963C_text(0) ;
      Delay_ms(300) ;
    end

{ *
 * If PORTA_2 is pressed, display only text panel
 *}
else
  if ( RB2_bit <> 0) then
    begin
      SPI_T6963C_graphics(0) ;
      SPI_T6963C_text(1) ;
      Delay_ms(300) ;
    end

{ *
 * If PORTA_3 is pressed, display text and graphic panels
 *}
else
  if ( RB3_bit <> 0) then
    begin
      SPI_T6963C_graphics(1) ;
      SPI_T6963C_text(1) ;
      Delay_ms(300) ;
    end

{ *
 * If PORTA_4 is pressed, change cursor
 *}
else
  if( RB4_bit <> 0) then
    begin
      Inc(curs);
      if (curs = 3) then
        curs := 0;
      case curs of
        0:
          // no cursor
          SPI_T6963C_cursor(0) ;
```

```

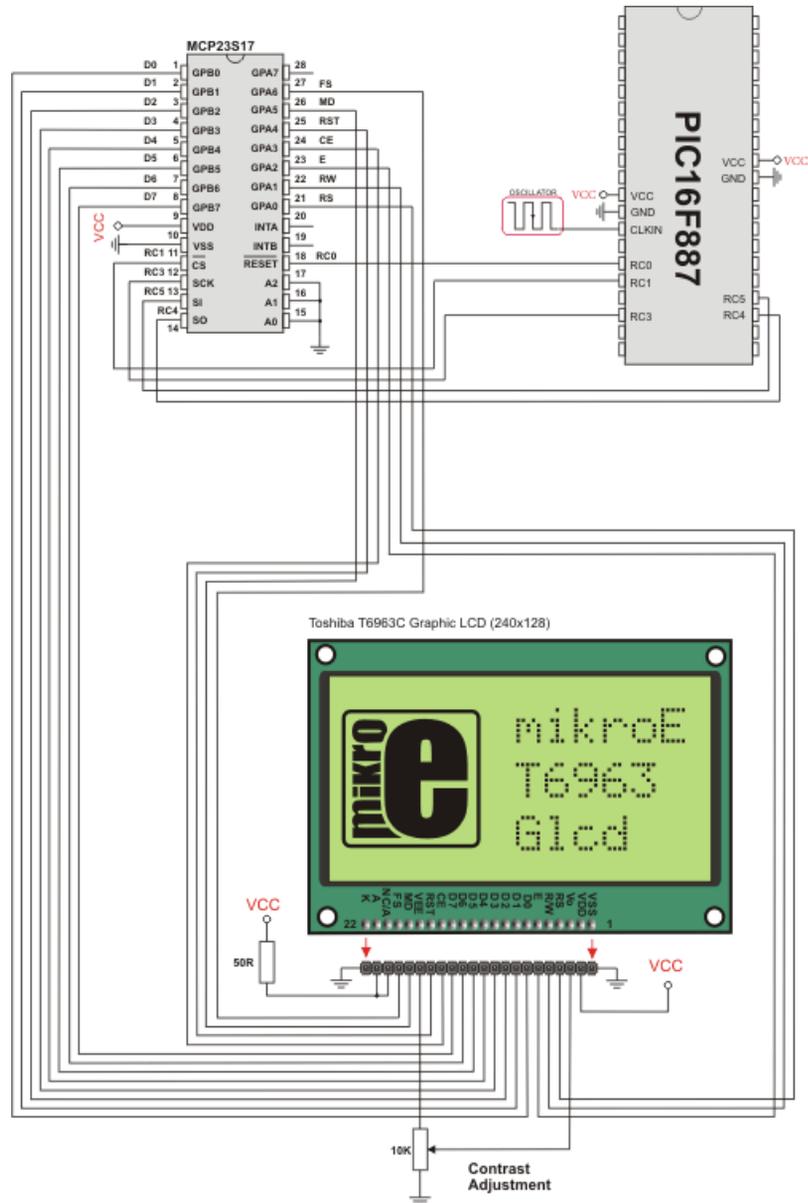
        1: begin
            // blinking cursor
            SPI_T6963C_cursor(1) ;
            SPI_T6963C_cursor_blink(1) ;
        end;
        2: begin
            // non blinking cursor
            SPI_T6963C_cursor(1) ;
            SPI_T6963C_cursor_blink(0) ;
        end;
    end;
    Delay_ms(300) ;
end;

{ *
 * Move cursor, even if not visible
 *}
Inc(cposx);
if (cpox = SPI_T6963C_txtCols) then
    begin
        cposx := 0 ;
        Inc(cposy);
        if (cposy = SPI_T6963C_grHeight /
SPI_T6963C_CHARACTER_HEIGHT) then
            cposy := 0 ;
        end;
        SPI_T6963C_set_cursor(cposx, cposy) ;

        Delay_ms(100) ;
    end;
end.

```

HW Connection



SPI T6963C Glcd HW connection

T6963C GRAPHIC LCD LIBRARY

The mikroPascal PRO for PIC 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>var T6963C_dataPort : byte; sfr; external;</code>	T6963C Control Port.	<code>var T6963C_dataPort : byte at PORTD;</code>
<code>var T6963C_ctrlwr : sbit; sfr; external;</code>	Write signal.	<code>var T6963C_ctrlwr : sbit at RC2_bit;</code>
<code>var T6963C_ctrlrd : sbit; sfr; external;</code>	Read signal.	<code>var T6963C_ctrlrd : sbit at RC1_bit;</code>
<code>var T6963C_ctrlcd : sbit; sfr; external;</code>	Command/Data signal.	<code>var T6963C_ctrlcd : sbit at RC0_bit;</code>
<code>var T6963C_ctrlrst : sbit; sfr; external;</code>	Reset signal.	<code>var T6963C_ctrlrst : sbit at RC4_bit;</code>
<code>var T6963C_ctrlwr_Direction : sbit; sfr; external;</code>	Direction of the Write pin.	<code>var T6963C_ctrlwr_Direction : sbit at TRISC2_bit;</code>
<code>var T6963C_ctrlrd_Direction : sbit; sfr; external;</code>	Direction of the Read pin.	<code>var T6963C_ctrlrd_Direction : sbit at TRISC0_bit;</code>
<code>var T6963C_ctrlcd_Direction : sbit; sfr; external;</code>	Direction of the Command/Data pin.	<code>var T6963C_ctrlcd_Direction : sbit at TRISC0_bit;</code>
<code>var T6963C_ctrlrst_Direction : sbit; sfr; external;</code>	Direction of the Reset pin.	<code>var T6963C_ctrlrst_Direction : sbit at TRISC4_bit;</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_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>procedure T6963C_init(width, height, fntW : 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 #1 + + + +-----+ \ </pre>
Requires	<p>Global variables :</p> <ul style="list-style-type: none"> - <code>T6963C_dataPort</code>: Data 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 - <code>T6963C_ctrlwr_Direction</code>: Direction of Write signal pin - <code>T6963C_ctrlrd_Direction</code>: Direction of Read signal pin - <code>T6963C_ctrlcd_Direction</code>: Direction of Command/Data signal pin - <code>T6963C_ctrlrst_Direction</code>: Direction of Reset signal pin <p>must be defined before using this function.</p>

Example	<pre>// T6963C module connections var T6963C_dataPort : byte at PORTD; var T6963C_ctrlwr : sbit at RC2_bit; var T6963C_ctrlrd : sbit at RC1_bit; var T6963C_ctrlcd : sbit at RC0_bit; var T6963C_ctrlrst : sbit at RC4_bit; var T6963C_ctrlwr_Direction : sbit at TRISC2_bit; var T6963C_ctrlrd_Direction : sbit at TRISC1_bit; var T6963C_ctrlcd_Direction : sbit at TRISC0_bit; var T6963C_ctrlrst_Direction : sbit at TRISC4_bit; // End of T6963C module connections ... // init display for 240 pixel width, 128 pixel height and 8 bits character width T6963C_init(240, 128, 8);</pre>
----------------	---

T6963C_WriteData

Prototype	<code>procedure T6963C_WriteData(mydata : 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>procedure T6963C_WriteCommand(mydata : 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>procedure T6963C_SetPtr(p : word; c : 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>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>procedure T6963C_Fill(v : byte; start, len : 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>procedure T6963C_Dot(x, y : integer; color : 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: 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>procedure T6963C_Write_Char(c, x, y, mode : 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>procedure T6963C_Write_Text(var str : array[10] of byte; x, y, mode : 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>procedure T6963C_Line(x0, y0, x1, y1 : integer; pcolor : 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>procedure T6963C_Rectangle(x0, y0, x1, y1 : integer; pcolor : 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>procedure T6963C_Box(x0, y0, x1, y1 : integer; pcolor : 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>procedure T6963C_Circle(x, y : integer; r : longint; pcolor : word);</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>procedure T6963C_Image(const code pic : ^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 mikroPascal PRO for PIC pointer to const and pointer to RAM equivalency). <p>Use the mikroPascal's integrated Glcd Bitmap Editor (menu option Tools > Glcd Bitmap Editor) to convert image to a constant array suitable for displaying on Glcd.</p> <p>Note: Image dimension must match the display dimension.</p>
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_Init routine.
Example	<code>T6963C_Image(mc);</code>

T6963C_Sprite

Prototype	<code>procedure T6963C_Sprite(px, py : byte; const pic : ^byte; sx, sy : 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 the T6963C_Init routine.
Example	<code>T6963C_Sprite(76, 4, einstein, 88, 119); // draw a sprite</code>

T6963C_Set_Cursor

Prototype	<code>procedure T6963C_Set_Cursor(x, y : 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 the T6963C_Init routine.
Example	<code>T6963C_Set_Cursor(cposx, cposy);</code>

T6963C_DisplayGrPanel

Prototype	<code>procedure T6963C_DisplayGrPanel(n : byte);</code>
Returns	Nothing.
Description	<p>Display selected graphic panel.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - n: graphic panel number. Valid values: 0 and 1.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_Init routine.
Example	<code>// display graphic panel 1 T6963C_DisplayGrPanel(1);</code>

T6963C_DisplayTxtPanel

Prototype	<code>procedure T6963C_DisplayTxtPanel(n : byte);</code>
Returns	Nothing.
Description	<p>Display selected text panel.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - n: text panel number. Valid values: 0 and 1.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_Init routine.
Example	<code>// display text panel 1 T6963C_DisplayTxtPanel(1);</code>

T6963C_SetGrPanel

Prototype	<code>procedure T6963C_SetGrPanel(n : 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	<pre>// set graphic panel 1 as current graphic panel. T6963C_SetGrPanel(1);</pre>

T6963C_SetTxtPanel

Prototype	<code>procedure T6963C_SetTxtPanel(n : 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	<pre>// set text panel 1 as current text panel. T6963C_SetTxtPanel(1);</pre>

T6963C_PanelFill

Prototype	<code>procedure T6963C_PanelFill(v : byte);</code>
Returns	Nothing.
Description	<p>Fill current panel in full (graphic+text) with appropriate value (0 to clear).</p> <p>Parameters :</p> <p>- <code>v</code>: value to fill panel with.</p>
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>procedure T6963C_GrFill(v : byte);</code>
Returns	Nothing.
Description	Fill current graphic panel with appropriate value (0 to clear). Parameters : - v : 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>procedure T6963C_TxtFill(v : byte);</code>
Returns	Nothing.
Description	Fill current text panel with appropriate value (0 to clear). Parameters : - v : 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>procedure T6963C_Cursor_Height(n : byte);</code>
Returns	Nothing.
Description	Set cursor size. Parameters : - n : 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>procedure T6963C_Graphics(n : byte);</code>
Returns	Nothing.
Description	<p>Enable/disable graphic displaying.</p> <p>Parameters :</p> <p>- n: on/off parameter. Valid values: 0 (disable graphic displaying) and 1 (enable graphic displaying).</p>
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>procedure T6963C_Text(n : byte);</code>
Returns	Nothing.
Description	<p>Enable/disable text displaying.</p> <p>Parameters :</p> <p>- n: on/off parameter. Valid values: 0 (disable text displaying) and 1 (enable text displaying).</p>
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>procedure T6963C_Cursor(n : byte);</code>
Returns	Nothing.
Description	<p>Set cursor on/off.</p> <p>Parameters :</p> <p>- n: on/off parameter. Valid values: 0 (set cursor off) and 1 (set cursor on).</p>
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_Init routine.
Example	<code>// set cursor on T6963C_Cursor(1);</code>

T6963C_Cursor_Blink

Prototype	<code>procedure T6963C_Cursor_Blink(n : byte);</code>
Returns	Nothing.
Description	Enable/disable cursor blinking. Parameters : - n : 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	<code>// enable cursor blinking T6963C_Cursor_Blink(1);</code>

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, EasyPIC5 board and 16F887.

```

program T6963C_240x128;

uses __Lib_T6963C_Consts, einstein_bmp, mikroe_bmp;

// T6963C module connections
var T6963C_dataPort : byte at PORTD; // DATA port

var T6963C_ctrlwr : sbit at RC2_bit; // WR write signal
var T6963C_ctrlrd : sbit at RC1_bit; // RD read signal
var T6963C_ctrlcd : sbit at RC0_bit; // CD command/data signal
var T6963C_ctrlrst : sbit at RC4_bit; // RST reset signal
var T6963C_ctrlwr_Direction : sbit at TRISC2_bit; // WR write signal direction
var T6963C_ctrlrd_Direction : sbit at TRISC1_bit; // RD read signal direction
var T6963C_ctrlcd_Direction : sbit at TRISC0_bit; // CD command/data signal direction
var T6963C_ctrlrst_Direction : sbit at TRISC4_bit; // RST reset signal direction

// Signals not used by library, they are set in main function
var T6963C_ctrlce : sbit at RC3_bit; // CE signal
var T6963C_ctrlfs : sbit at RC6_bit; // FS signal
var T6963C_ctrlmd : sbit at RC5_bit; // MD signal
var T6963C_ctrlce_Direction : sbit at TRISC3_bit; // CE signal direction
var T6963C_ctrlfs_Direction : sbit at TRISC6_bit; // FS signal direction
var T6963C_ctrlmd_Direction : sbit at TRISC5_bit; // MD signal direction
// End T6963C module connections

```

```
var   panel : byte;           // current panel
      i : word;              // general purpose register
      curs : byte;          // cursor visibility
      cposx,
      cposy : word;         // cursor x-y position
      txtcols : byte;      // number of text coloms
      txt, txt1 : string[ 29];

begin

  txt1 := ' EINSTEIN WOULD HAVE LIKED mE';
  txt  := ' GLCD LIBRARY DEMO, WELCOME !';

  ANSEL  := 0;                // Configure AN pins as digital I/O
  ANSELH := 0;
  C1ON_bit := 0;             // Disable comparators
  C2ON_bit := 0;

  TRISB0_bit := 1;          // Set RB0 as input
  TRISB1_bit := 1;          // Set RB1 as input
  TRISB2_bit := 1;          // Set RB2 as input
  TRISB3_bit := 1;          // Set RB3 as input
  TRISB4_bit := 1;          // Set RB4 as input

  T6963C_ctrlce_Direction := 0;
  T6963C_ctrlce := 0;       // Enable T6963C
  T6963C_ctrlfs_Direction := 0;
  T6963C_ctrlfs := 0;      // Font Select 8x8
  T6963C_ctrlmd_Direction := 0;
  T6963C_ctrlmd := 0;      // Column number select

  panel := 0;
  i := 0;
  curs := 0;
  cposx := 0;
  cposy := 0;

  // Initialize T6369C
  T6963C_init(240, 128, 8);

  { *
    * Enable both graphics and text display at the same time
    * }
  T6963C_graphics(1);
  T6963C_text(1);

  { *
    * 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_WHITE);
T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE);
T6963C_rectangle(40, 40, 199, 87, T6963C_WHITE);
T6963C_rectangle(60, 60, 179, 67, T6963C_WHITE);

{ *
  * Draw a cross
  *}
T6963C_line(0, 0, 239, 127, T6963C_WHITE);
T6963C_line(0, 127, 239, 0, T6963C_WHITE);

{ *
  * Draw solid boxes
  *}
T6963C_box(0, 0, 239, 8, T6963C_WHITE);
T6963C_box(0, 119, 239, 127, T6963C_WHITE);
//while true do nop;
{ *
  * Draw circles
  *}
T6963C_circle(120, 64, 10, T6963C_WHITE);
T6963C_circle(120, 64, 30, T6963C_WHITE);
T6963C_circle(120, 64, 50, T6963C_WHITE);
T6963C_circle(120, 64, 70, T6963C_WHITE);
T6963C_circle(120, 64, 90, T6963C_WHITE);
T6963C_circle(120, 64, 110, T6963C_WHITE);
T6963C_circle(120, 64, 130, T6963C_WHITE);

T6963C_sprite(76, 4, @einstein, 88, 119);           // Draw a sprite

T6963C_setGrPanel(1);                               // Select other
graphic panel

T6963C_sprite(0, 0, @mikroe_bmp, 240, 64); // 240x128 can't be
stored in most of PIC16 MCUs
T6963C_sprite(0, 64, @mikroe_bmp, 240, 64); // it is replaced with
smaller picture 240x64

// Smaller
picture is drawn two times

```

```
while (TRUE) do // Endless loop
begin

    { *
      * If PORTA_0 is pressed, toggle the display between graphic
panel 0 and graphic 1
      * }
    if( RB0_bit <> 0) then
        begin
            T6963C_graphics(1);
            T6963C_text(0);
            Delay_ms(300);
        end

    { *
      * If PORTA_1 is pressed, display only graphic panel
      * }
    else
        if ( RB1_bit <> 0) then
            begin
                Inc(panel) ;
                panel := panel and 1;
                T6963C_setPtr((T6963C_grMemSize + T6963C_txtMemSize) *
panel, T6963C_GRAPHIC_HOME_ADDRESS_SET) ;
                Delay_ms(300) ;
            end

    { *
      * If PORTA_2 is pressed, display only text panel
      * }
    else
        if ( RB2_bit <> 0) then
            begin
                T6963C_graphics(0) ;
                T6963C_text(1) ;
                Delay_ms(300) ;
            end

    { *
      * If PORTA_3 is pressed, display text and graphic panels
      * }
    else
        if ( RB3_bit <> 0) then
            begin
                T6963C_graphics(1) ;
                T6963C_text(1) ;
                Delay_ms(300) ;
            end
        end
    end
```

```

{ *
 * If PORTA_4 is pressed, change cursor
 *}
    else
        if( RB4_bit <> 0) then
            begin
                Inc(curs);
                if (curs = 3) then
                    curs := 0;
                case curs of
                    0:
                        // no cursor
                        T6963C_cursor(0) ;

                    1: begin
                        // blinking cursor
                        T6963C_cursor(1) ;
                        T6963C_cursor_blink(1) ;
                    end;
                    2: begin
                        // non blinking cursor
                        T6963C_cursor(1) ;
                        T6963C_cursor_blink(0) ;
                    end;
                end;
                Delay_ms(300) ;
            end;

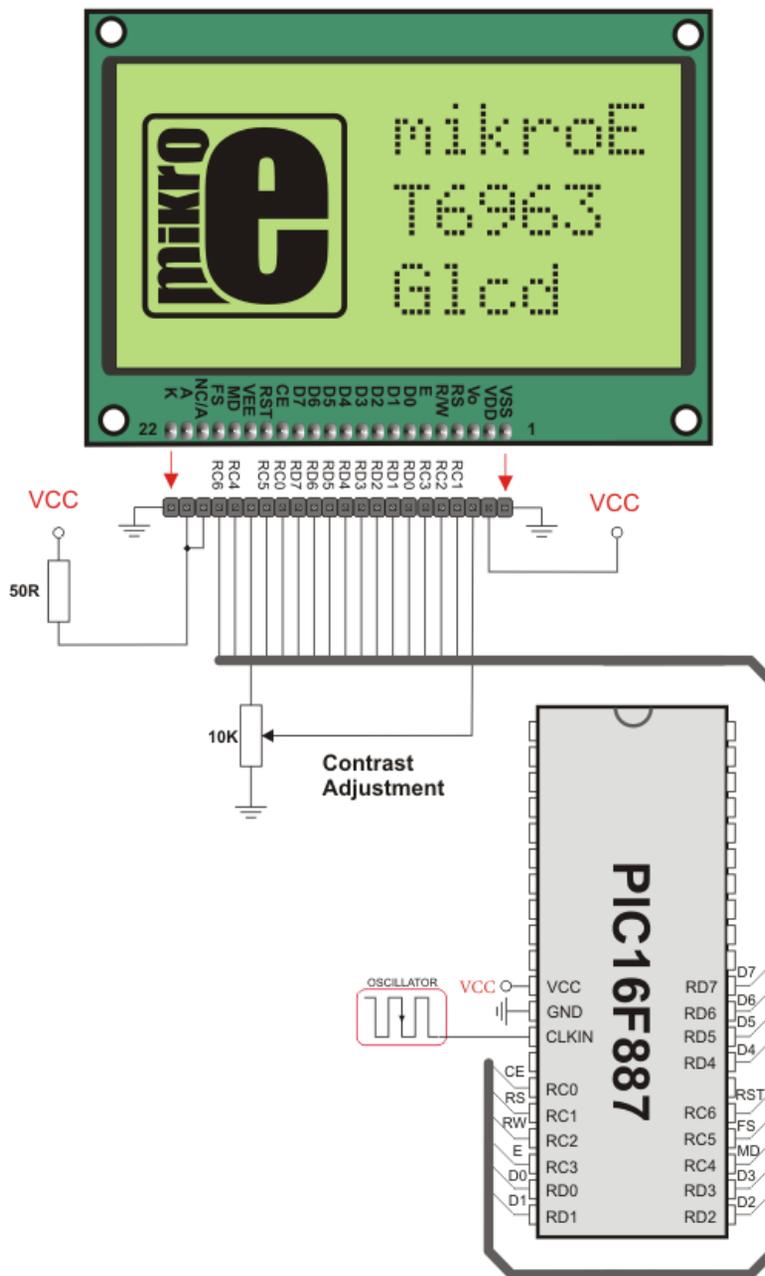
{ *
 * Move cursor, even if not visible
 *}
Inc(cposx);
if (cpox = T6963C_txtCols) then
    begin
        cposx := 0 ;
        Inc(cposy);
        if (cposy = T6963C_grHeight / T6963C_CHARACTER_HEIGHT) then
            cposy := 0 ;
        end;
        T6963C_set_cursor(cposx, cposy) ;

        Delay_ms(100) ;
    end;
end.

```

HW Connection

Toshiba T6963C Graphic LCD (240x128)



T6963C Glcd HW connection

UART LIBRARY

UART hardware module is available with a number of PIC MCUs. mikroPascal PRO for PIC UART Library provides comfortable work with the Asynchronous (full duplex) mode.

You can easily communicate with other devices via RS-232 protocol (for example with PC, see the figure at the end of the topic – RS-232 HW connection). You need a PIC MCU with hardware integrated UART, for example 16F887. Then, simply use the functions listed below.

Note: Some PIC18 MCUs have multiple UART modules. Switching between the UART modules in the UART library is done by the `UART_Set_Active` function (UART module has to be previously initialized).

Note: In order to use the desired UART library routine, simply change the number **1** in the prototype with the appropriate module number, i.e. `UART2_Init(2400);`

Library Routines

- `UART1_Init`
- `UART1_Data_Ready`
- `UART1_Tx_Idle`
- `UART1_Read`
- `UART1_Read_Text`
- `UART1_Write`
- `UART1_Write_Text`
- `UART_Set_Active`

UART1_Init

Prototype	<code>procedure UART1_Init(baud_rate: longint);</code>
Returns	Nothing.
Description	<p>Initializes hardware UART module with the desired baud rate. Refer to the device data sheet for baud rates allowed for specific <code>Fosc</code>. If you specify the unsupported baud rate, compiler will report an error.</p> <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 <code>Fosc</code>.</p>
Requires	<p>You'll need PIC MCU with hardware UART.</p> <p><code>UART1_Init</code> needs to be called before using other functions from UART Library.</p> <p>Note: Calculation of the UART baud rate value is carried out by the compiler, as it would produce a relatively large code if performed on the library level. Therefore, compiler needs to know the value of the parameter in the compile time. That is why this parameter needs to be a constant, and not a variable.</p>
Example	<p>This will initialize hardware UART1 module and establish the communication at 2400 bps:</p> <pre>UART1_Init(2400);</pre>

UART1_Data_Ready

Prototype	<code>function UART1_Data_Ready(): byte;</code>
Returns	Function returns 1 if data is ready or 0 if there is no data.
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 UART1_Init routine.
Example	<pre>var receive: byte; ... // read data if ready if (UART1_Data_Ready() = 1) then receive := UART1_Read();</pre>

UART1_Tx_Idle

Prototype	<code>function UART1_Tx_Idle(): byte</code>
Returns	- 1 if the data has been transmitted - 0 otherwise
Description	Use the function to test if the transmit shift register is empty or not.
Requires	UART HW module must be initialized and communication established before using this function. See UART1_Init.
Example	<pre>// If the previous data has been shifted out, send next data: if (UART1_Tx_Idle() <> 0) begin ... end;</pre>

UART1_Read

Prototype	<code>function UART1_Read(): byte;</code>
Returns	Received byte.
Description	The function receives a byte via UART. Use the <code>UART1_Data_Ready</code> 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 <code>UART1_Init</code> routine.
Example	<pre> var receive: byte; ... // read data if ready if (UART1_Data_Ready() = 1) then receive := UART1_Read(); </pre>

UART1_Read_Text

Prototype	<code>procedure UART1_Read_Text(var Output, Delimiter : array[255] of byte; Attempts : byte);</code>
Returns	Nothing.
Description	<p>Reads characters received via UART until the delimiter sequence is detected. The read sequence is stored in the parameter <code>output</code>; delimiter sequence is stored in the parameter <code>delimiter</code>.</p> <p>This is a blocking call: the delimiter sequence is expected, otherwise the procedure exits(if the delimiter is not found). Parameter <code>Attempts</code> defines number of received characters in which <code>Delimiter</code> sequence is expected. If <code>Attempts</code> is set to 255, this routine will continuously try to detect the <code>Delimiter</code> sequence.</p>
Requires	UART HW module must be initialized and communication established before using this function. See <code>UART1_Init</code> .
Example	<p>Read text until the sequence "OK" is received, and send back what's been received:</p> <pre> UART1_Init(4800); // initialize UART module Delay_ms(100); while TRUE do begin if (UART1_Data_Ready() = 1) then // if data is received begin UART1_Read_Text(output, 'delim', 10); // reads text until 'delim' is found UART1_Write_Text(output); // sends back text end; end; </pre>

UART1_Write

Prototype	<code>procedure UART1_Write(TxData: byte);</code>
Returns	Nothing.
Description	The function transmits a byte via the UART module. Parameters : - TxData: data to be sent
Requires	MCU with the UART module. The UART module must be initialized before using this routine. See UART1_Init routine.
Example	<pre>var data_: byte; ... data := 0x1E UART1_Write(data_);</pre>

UART1_Write_Text

Prototype	<code>procedure UART1_Write_Text(var uart_text : array[255] of byte);</code>
Returns	Nothing.
Description	Sends text (parameter <code>uart_text</code>) via UART. Text should be zero terminated.
Requires	UART HW module must be initialized and communication established before using this function. See UART1_Init.
Example	<p>Read text until the sequence “OK” is received, and send back what’s been received:</p> <pre>UART1_Init(4800); // initialize UART module Delay_ms(100); while TRUE do begin if (UART1_Data_Ready() = 1) then // if data is received begin UART1_Read_Text(output, 'delim', 10); // reads text until 'delim' is found UART1_Write_Text(output); // sends back text end; end;</pre>

UART_Set_Active

Prototype	<code>procedure UART_Set_Active (read_ptr : ^Tread_ptr; write_ptr : ^Twrite_ptr; ready_ptr : ^Tready_ptr; tx_idle_ptr : ^Ttx_idle_ptr);</code>
Returns	Nothing.
Description	<p>Sets active UART module which will be used by the UART library routines.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>read_ptr</code>: UART1_Read handler - <code>write_ptr</code>: UART1_Write handler - <code>ready_ptr</code>: UART1_Data_Ready handler - <code>tx_idle_ptr</code>: UART1_Tx_Idle handler
Requires	<p>Routine is available only for MCUs with two UART modules.</p> <p>Used UART module must be initialized before using this routine. See UART1_Init routine</p>
Example	<pre>// Activate UART2 module UART_Set_Active(@UART1_Read, @UART1_Write, @UART1_Data_Ready, @UART1_Tx_Idle);</pre>

Library Example

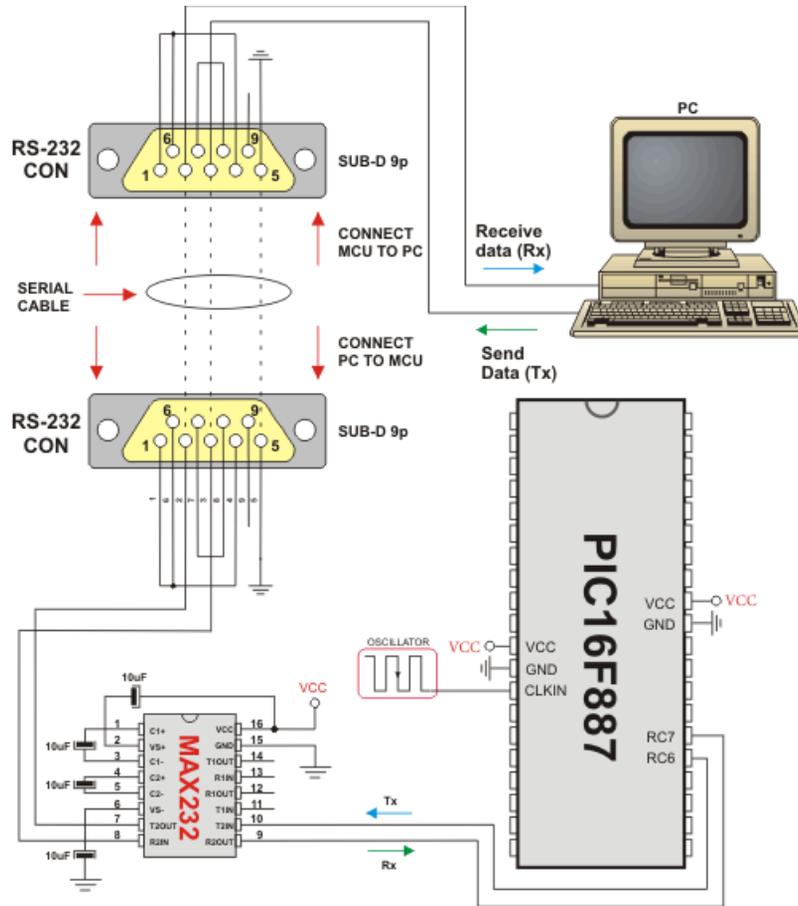
This example demonstrates simple data exchange via UART. If MCU is connected to the PC, you can test the example from the mikroPascal PRO for PIC USART Terminal.

```
program UART;
var uart_rd : byte;

begin
  UART1_Init(9600);           // Initialize UART module at 9600 bps
  Delay_ms(100);            // Wait for UART module to stabilize

  while (TRUE) do           // Endless loop
  begin
    if (UART1_Data_Ready() <> 0) then // If data is received,
    begin
      uart_rd := UART1_Read();        // read the received data,
      UART1_Write(uart_rd);          // and send data via UART
    end;
  end;
end.
```

HW Connection



UART HW connection

USB HID LIBRARY

Universal Serial Bus (USB) provides a serial bus standard for connecting a wide variety of devices, including computers, cell phones, game consoles, PDA's, etc.

mikroPascal PRO for PIC includes a library for working with human interface devices via Universal Serial Bus. A human interface device or HID is a type of computer device that interacts directly with and takes input from humans, such as the keyboard, mouse, graphics tablet, and the like.

Descriptor File

Each project based on the USB HID library should include a descriptor source file which contains vendor id and name, product id and name, report length, and other relevant information. To create a descriptor file, use the integrated USB HID terminal of mikroPascal PRO for PIC (**Tools** > **USB HID Terminal**). The default name for descriptor file is USBdsc.ppas, but you may rename it.

The provided code in the “Examples” folder works at 48MHz, and the flags should not be modified without consulting the appropriate datasheet first.

Library Routines

- Hid_Enable
- Hid_Read
- Hid_Write
- Hid_Disable

Hid_Enable

Prototype	<code>procedure Hid_Enable(readbuff, writebuff : word);</code>
Returns	Nothing.
Description	Enables USB HID communication. Parameters <code>readbuff</code> and <code>writebuff</code> are the addresses of Read Buffer and the Write Buffer, respectively, which are used for HID communication. You can pass buffer names with the <code>@</code> operator. This function needs to be called before using other routines of USB HID Library.
Requires	Nothing.
Example	<code>Hid_Enable(@rd, @wr);</code>

Hid_Read

Prototype	<code>function Hid_Read() : byte;</code>
Returns	Number of characters in the Read Buffer received from the host.
Description	Receives message from host and stores it in the Read Buffer. Function returns the number of characters received in the Read Buffer.
Requires	USB HID needs to be enabled before using this function. See Hid_Enable.
Example	<code>length := Hid_Read();</code>

Hid_Write

Prototype	<code>procedure Hid_Write(writebuff : word, len : byte);</code>
Returns	Nothing.
Description	Function sends data from Write Buffer writebuff to host. Write Buffer is the address of the parameter used in initialization; see Hid_Enable. You can pass a buffer name with the @ operator. Parameter len should specify a length of the data to be transmitted.
Requires	USB HID needs to be enabled before using this function. See Hid_Enable.
Example	<code>Hid_Write(@wr, len);</code>

Hid_Disable

Prototype	<code>procedure Hid_Disable();</code>
Returns	Nothing.
Description	Disables USB HID communication.
Requires	USB HID needs to be enabled before using this function. See Hid_Enable.
Example	<code>Hid_Disable();</code>

Library Example

This example establishes connection with the HID terminal that is active on the PC. Upon connection establishment, the HID Device Name will appear in the respective window. After that software will wait for data and it will return received data back. Examples uses USBdsc descriptor file, which is in the same folder, and can be created by the HID Terminal.

```
program USB_HID_test;

uses USBdsc;

var
  k, i, ch      : byte;
  userWR_buffer : array[ 64] of byte;
  userRD_buffer : array[ 64] of byte;

// Main Interrupt Routine
procedure interrupt;
begin
  HID_InterruptProc;
end;

// Initialization Routine
procedure Init_Main;
begin
  // Disable interrupts
  INTCON := 0;                                // Disable GIE, PEIE,
  TMR0IE,INT0IE,RBIE
  INTCON2 := 0xF5;
  INTCON3 := 0xC0;
  RCON.IPEN := 0;                            // Disable Priority Levels on interrupts
  PIE1 := 0;
  PIE2 := 0;
  PIR1 := 0;
  PIR2 := 0;

  ADCON1 := ADCON1 or 0x0F;                  // Configure all ports with ana-
log function as digital
  // Ports Configuration
  TRISA := 0xFF;
  TRISB := 0xFF;
  TRISC := 0xFF;
  TRISD := 0;
  TRISE := 0x07;

  LATA := 0;
  LATB := 0;
  LATC := 0;
  LATD := 0;
  LATE := 0;
end;

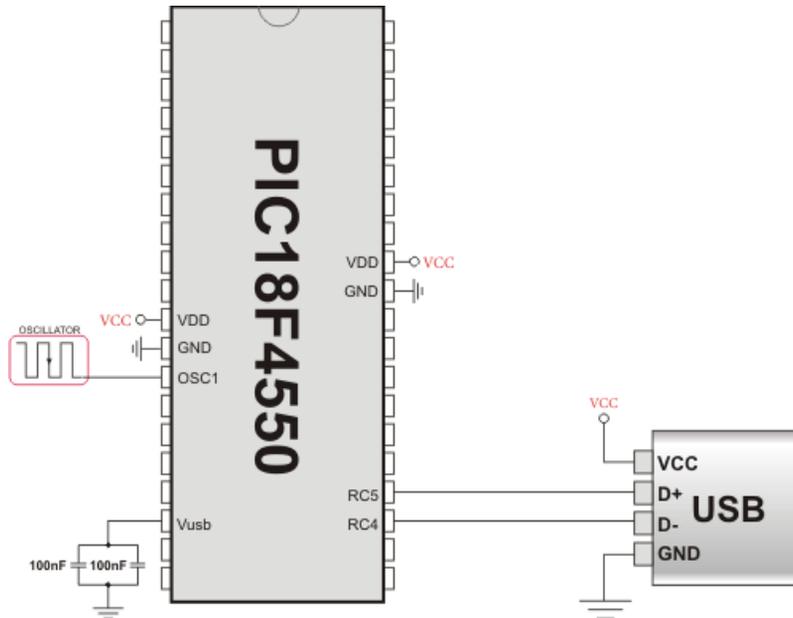
// Main Program Routine
begin
  Init_Main();
```

```

HID_Enable(@userRD_buffer, @userWR_buffer);
Delay_mS(1000);
Delay_mS(1000);
while true do
begin
k := HID_Read();
i := 0;
while i < k do
begin
ch := userRD_buffer[ 0 ];
userWR_buffer[ 0 ] := ch;
HID_Write(@userWR_buffer, 1);
inc(i);
end;
end;
HID_Disable();
end.

```

HW Connection



USB connection scheme

MISCELLANEOUS LIBRARIES

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

BUTTON LIBRARY

Button library contains miscellaneous routines useful for project development.

- Button

Button

Prototype	<code>function Button(var port : byte; pin, time, active_state : byte) : byte;</code>
Returns	Returns 0 or 255.
Description	Function eliminates the influence of contact flickering upon pressing a button (debouncing). Parameter <code>port</code> specifies the location of the button; parameter <code>pin</code> is the pin number on designated <code>port</code> and goes from 0..7; parameter <code>time</code> is a debounce period in milliseconds; parameter <code>active_state</code> can be either 0 or 1, and it determines if the button is active upon logical zero or logical one.
Requires	Button pin must be configured as input.
Example	Example reads RB0, to which the button is connected; on transition from 1 to 0 (release of button), PORTD is inverted: <pre>while true do begin if Button(PORTB, 0, 1, 1) then oldstate := 255; if oldstate and Button(PORTB, 0, 1, 0) then begin PORTD := not(PORTD); oldstate := 0; end; end;</pre>

CONVERSIONS LIBRARY

mikroPascal PRO for PIC 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
- StrToInt
- StrToWord

The following functions convert decimal values to BCD and vice versa:

- Dec2Bcd
- Bcd2Dec16
- Dec2Bcd16

ByteToStr

Prototype	<code>procedure ByteToStr(input : word; var output : array[3] of char);</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>var t : word; txt : array[3] of char; ... t := 24; ByteToStr(t, txt); // txt is " 24" (one blank here)</pre>

ShortToStr

Prototype	<code>procedure ShortToStr(input : short; var output : array[4] of char);</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>var t : short; txt : array[4] of char; ... t := -24; ByteToStr(t, txt); // txt is " -24" (one blank here)</pre>

WordToStr

Prototype	<code>procedure WordToStr(input : word; var output : array[5] of char);</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>var t : word; txt : array[5] of char; ... t := 437; WordToStr(t, txt); // txt is " 437" (two blanks here)</pre>

IntToStr

Prototype	<code>procedure IntToStr(input : integer; var output : array[6] of char);</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>var input : integer; txt : string[5]; //... begin input := -4220; IntToStr(input, txt); // txt is ' -4220'</pre>

LongintToStr

Prototype	<code>procedure LongintToStr(input : longint; var output : array[11] of char);</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>var input : longint; txt : array[11] of char; //... begin input := -12345678; IntToStr(input, txt); // txt is ' -12345678'</pre>

LongWordToStr

Prototype	<code>procedure LongWordToStr(input : dword; var output : array[10] of char);</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>var input : longint; txt : array[10] of char; //... begin input := 12345678; IntToStr(input, txt); // txt is ' 12345678'</pre>

FloatToStr

Prototype	<code>procedure FloatToStr(input : real; var output : array[23] of char);</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>var ff1, ff2, ff3 : real; txt : array[23] of char; ... 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>

StrToInt

Prototype	<code>function StrToInt(var input:string[6]):integer;</code>
Returns	Integer variable.
Description	Converts a string to integer.
Requires	The string is assumed to be a correct representation of a number.
Example	<pre>var ii:integer begin ii := StrToInt('-1234');</pre>

StrToWord

Prototype	<code>function StrToWord(var input:string[5]):word;</code>
Returns	Word variable.
Description	Converts a string to word.
Requires	<code>input</code> string with length of max 5 chars. The string is assumed to be a correct representation of a number.
Example	<pre>var ww:word; begin ww := StrToword('65432');</pre>

Dec2Bcd

Prototype	<code>function Dec2Bcd(decnum : byte) : 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	<pre>var a, b : byte; ... a := 22; b := Dec2Bcd(a); // b equals 34</pre>

Bcd2Dec16

Prototype	<code>function Bcd2Dec16(bcdnum : word) : word;</code>
Returns	Converted decimal value.
Description	<p>Converts 16-bit BCD numeral to its decimal equivalent.</p> <p>Parameters :</p> <p>- <code>bcdnum</code>: 16-bit BCD numeral to be converted</p>
Requires	Nothing.
Example	<pre>var a, b : word; ... a := 0x1234; // a equals 4660 b := Bcd2Dec16(a); // b equals 1234</pre>

Dec2Bcd16

Prototype	<code>function Dec2Bcd16(decnum : word) : 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>var a, b : word; ... a := 2345; b := Dec2Bcd16(a); // b equals 9029</pre>

MATH LIBRARY

The mikroPascal PRO for PIC 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>function acos(x : real) : 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>function asin(x : real) : 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>function atan(arg : real) : 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>function atan2(y : real; x : real) : 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>function ceil(x : real) : real;</code>
Description	The function returns value of parameter <code>x</code> rounded up to the next whole number.

cos

Prototype	<code>function cos(arg : real) : 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>function cosh(x : real) : 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>function eval_poly(x : real; var d : array[10] of real; n : integer) : real;</code>
Description	Function Calculates polynomial for number <code>x</code> , with coefficients stored in <code>d[]</code> , for degree <code>n</code> .

exp

Prototype	<code>function exp(x : real) : 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>function fabs(d : real) : real;</code>
Description	The function returns the absolute (i.e. positive) value of <code>d</code> .

floor

Prototype	<code>function floor(x : real) : real;</code>
Description	The function returns the value of parameter <code>x</code> rounded down to the nearest integer.

frexp

Prototype	<code>function frexp(value : real; var eptr : integer) : 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>function ldexp(value : real; newexp : integer) : 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>function log(x : real) : real;</code>
Description	The function returns the natural logarithm of <code>x</code> (i.e. $\log_e(x)$).

log10

Prototype	<code>function log10(x : real) : real;</code>
Description	The function returns the base-10 logarithm of <code>x</code> (i.e. $\log_{10}(x)$).

modf

Prototype	<code>function modf(val : real; var iptr : real) : 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>function pow(x : real; y : real) : real;</code>
Description	The function returns the value of x raised to the power y (i.e. x^y). If x is negative, the function will automatically cast y into <code>longint</code> .

sin

Prototype	<code>function sin(arg : real) : 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>function sinh(x : real) : real;</code>
Description	The function returns the hyperbolic sine of x , defined mathematically as $(e^x - e^{-x})/2$. If the value of x is too large (if overflow occurs), the function fails.

sqrt

Prototype	<code>function sqrt(x : real) : real;</code>
Description	The function returns the non negative square root of x .

tan

Prototype	<code>function tan(x : real) : real;</code>
Description	The function returns the tangent of x in radians. The return value spans the allowed range of floating point in mikroPascal PRO for PIC.

tanh

Prototype	<code>function tanh(x : real) : real;</code>
Description	The function returns the hyperbolic tangent of x , defined mathematically as $\sinh(x)/\cosh(x)$.

STRING LIBRARY

The mikroPascal PRO for PIC includes a library which automatizes string related tasks.

Library Functions

- memchr
- memcmp
- memcpy
- memmove
- memset
- strcat
- strchr
- strcmp
- strcpy
- strlen
- strncat
- strncmp
- strspn
- strcspn
- strncmp
- strpbrk
- strrchr
- strstr

memchr

Prototype	<code>function memchr(p : ^byte; ch : byte; n : word) : word;</code>
Description	<p>The function locates the first occurrence of the word ch in the initial n words of memory area starting at the address p. The function returns the offset of this occurrence from the memory address p or <code>0xFFFF</code> if ch was not found.</p> <p>For the parameter p 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>function memcmp(p1, p2 : ^byte; n : word) : short;</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 border="0"> <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>procedure memcpy(p1, p2 : ^byte; nn : 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>procedure memmove(p1, p2 : ^byte; nn : 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>procedure memset(p : ^byte; character : byte; n : word);</code>
Description	<p>The function fills the first <code>n</code> words in the memory area starting at the address <code>p</code> with the value of word <code>character</code>.</p> <p>For 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>

strcat

Prototype	<code>procedure strcat(var s1, s2 : string[100]);</code>
Description	The function appends the value of string <code>s2</code> to string <code>s1</code> and terminates <code>s1</code> with a null character.

strchr

Prototype	<code>function strchr(var s : string[100]; ch : byte) : word;</code>
Description	<p>The function searches the string <code>s</code> for the first occurrence of the character <code>ch</code>. The null character terminating <code>s</code> is not included in the search.</p> <p>The function returns the position (index) of the first character <code>ch</code> found in <code>s</code>; if no matching character was found, the function returns <code>0xFFFF</code>.</p>

strcmp

Prototype	<code>function strcmp(var s1, s2 : string[100]) : short;</code>								
Description	<p>The function lexicographically compares the contents of the strings <code>s1</code> and <code>s2</code> and returns a value indicating their relationship:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>< 0</code></td> <td><code>s1 "less than" s2</code></td> </tr> <tr> <td><code>= 0</code></td> <td><code>s1 "equal to" s2</code></td> </tr> <tr> <td><code>> 0</code></td> <td><code>s1 "greater than" s2</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>	Value	Meaning	<code>< 0</code>	<code>s1 "less than" s2</code>	<code>= 0</code>	<code>s1 "equal to" s2</code>	<code>> 0</code>	<code>s1 "greater than" s2</code>
Value	Meaning								
<code>< 0</code>	<code>s1 "less than" s2</code>								
<code>= 0</code>	<code>s1 "equal to" s2</code>								
<code>> 0</code>	<code>s1 "greater than" s2</code>								

strcpy

Prototype	<code>procedure strcpy(var s1, s2 : 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>function strcspn(var s1, s2 : string[100]) : 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>function strlen(var s : string[100]) : 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>procedure strncat(var s1, s2 : string[100]; size : 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>function strncmp(var s1, s2 : string[100]; len : byte) : short;</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"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>< 0</td> <td><code>s1</code> "less than" <code>s2</code></td> </tr> <tr> <td>= 0</td> <td><code>s1</code> "equal to" <code>s2</code></td> </tr> <tr> <td>> 0</td> <td><code>s1</code> "greater than" <code>s2</code></td> </tr> </tbody> </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	<code>s1</code> "less than" <code>s2</code>	= 0	<code>s1</code> "equal to" <code>s2</code>	> 0	<code>s1</code> "greater than" <code>s2</code>
Value	Meaning								
< 0	<code>s1</code> "less than" <code>s2</code>								
= 0	<code>s1</code> "equal to" <code>s2</code>								
> 0	<code>s1</code> "greater than" <code>s2</code>								

strncpy

Prototype	<code>procedure strncpy(var s1, s2 : string[100]; size : word);</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>function strpbrk(var s1, s2 : string[100]) : 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>function strrchr(var s : string[100]; ch : byte) : 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>function strspn(var s1, s2 : string[100]) : byte;</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>function strstr(var s1, s2 : string[100]) : 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>function Time_dateToEpoch(var ts : TimeStruct) : 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>var ts1 : TimeStruct; Epoch : longint; ... // what is the epoch of the date in ts ? epoch := Time_dateToEpoch(ts1);</pre>

Time_epochToDate

Prototype	<code>procedure Time_epochToDate(e: longint; var ts : TimeStruct);</code>
Returns	Nothing.
Description	<p>Converts the UNIX time to time and date.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>e</code>: UNIX time (seconds since UNIX epoch) - <code>ts</code>: time and date structure for storing conversion output
Requires	Nothing.
Example	<pre>var ts2 : TimeStruct; epoch : longint; ... //what date is epoch 1234567890 ? epoch := 1234567890; Time_epochToDate(epoch,ts2);</pre>

Time_dateDiff

Prototype	<code>function Time_dateDiff(t1 : ^TimeStruct; t2 : ^TimeStruct) : 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 <code>t1</code> is before <code>t2</code>, null if <code>t1</code> is the same as <code>t2</code> and negative if <code>t1</code> is after <code>t2</code>.</p> <p>Parameters :</p> <ul style="list-style-type: none"> - <code>t1</code>: time and date structure (the first comparison parameter) - <code>t2</code>: time and date structure (the second comparison parameter)
Requires	Nothing.
Example	<pre>var ts1, ts2 : TimeStruct; diff : 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;
program Time_Demo;
var epoch, diff : longint;
    ts1, ts2 : TimeStruct;
begin
    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);

    { *
      * What date is epoch 1234567890 ?
      * }
    epoch := 1234567890;
    Time_epochToDate(epoch, ts2);

    { *
      * How much seconds between this two dates ?
      * }
    diff := Time_dateDiff(ts1, ts2);
end.
```

TimeStruct type definition

```
type TimeStruct = record

    ss : byte;    // seconds
    mn : byte;    // minutes
    hh : byte;    // hours
    md : byte;    // day in month, from 1 to 31
    wd : byte;    // day in week, monday=0, tuesday=1, .... sunday=6
    mo : byte;    // month number, from 1 to 12 (and not from
0 to 11 as with unix C time !)
    yy : word;    // year Y2K compliant, from 1892 to 2038
end;
```

TRIGONOMETRY LIBRARY

The mikroPascal PRO for PIC 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>function sinE3(angle_deg : word): 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> <ul style="list-style-type: none"> - <code>angle_deg</code>: input angle in degrees <p>Note: Return value range: -1000..1000.</p>
Requires	Nothing.
Example	<pre>var res : integer; ... res := sinE3(45); // result is 707</pre>

cosE3

Prototype	<code>function cosE3(angle_deg : word): 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> <ul style="list-style-type: none">- <code>angle_deg</code>: input angle in degrees <p>Note: Return value range: -1000..1000.</p>
Requires	Nothing.
Example	<pre>var res: 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:

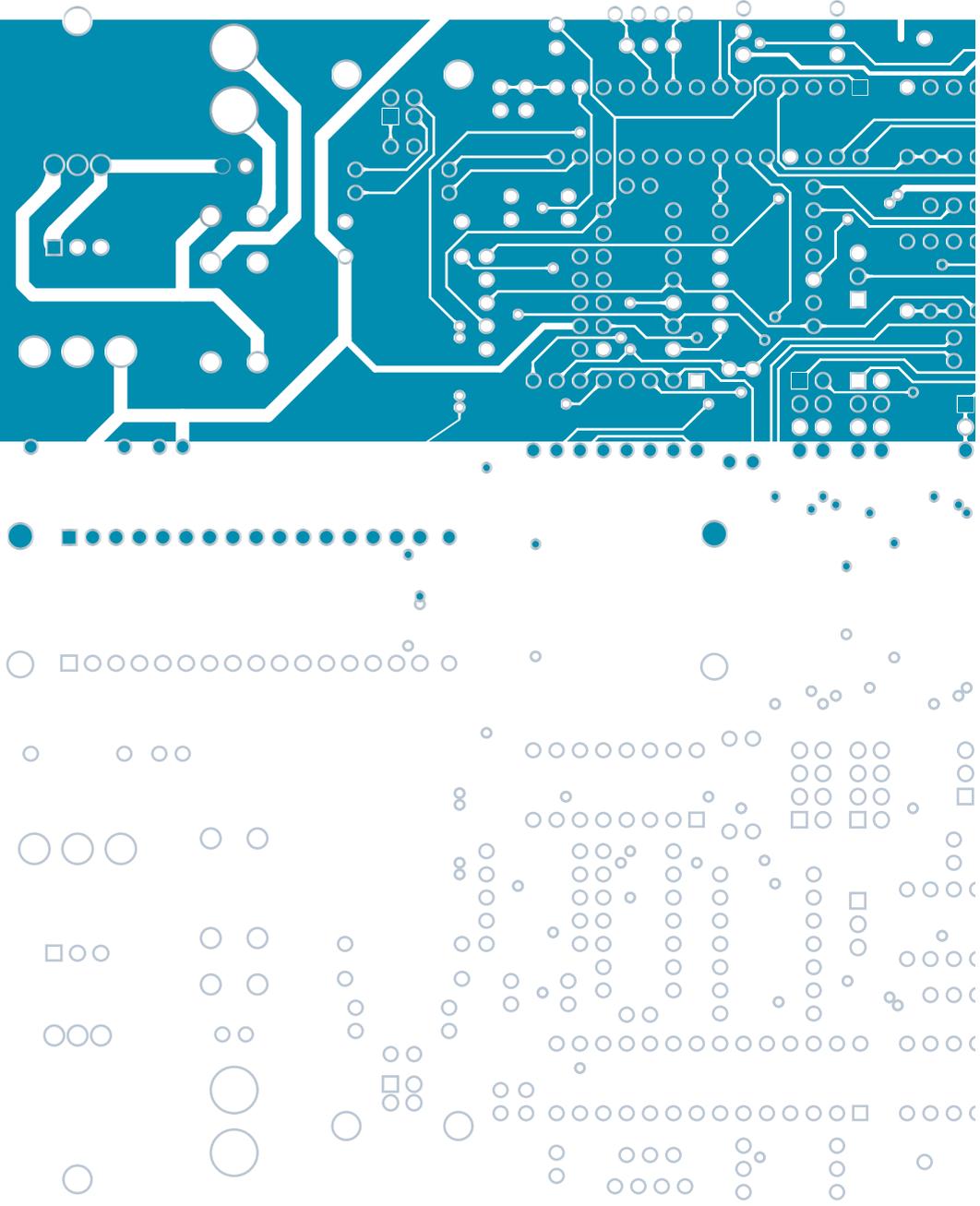
web: www.mikroe.com

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. TECHNICAL SUPPORT:

www.mikroe.com/en/support



Компания «Life Electronics» занимается поставками электронных компонентов импортного и отечественного производства от производителей и со складов крупных дистрибьюторов Европы, Америки и Азии.

С конца 2013 года компания активно расширяет линейку поставок компонентов по направлению коаксиальный кабель, кварцевые генераторы и конденсаторы (керамические, пленочные, электролитические), за счёт заключения дистрибьюторских договоров

Мы предлагаем:

- Конкурентоспособные цены и скидки постоянным клиентам.
- Специальные условия для постоянных клиентов.
- Подбор аналогов.
- Поставку компонентов в любых объемах, удовлетворяющих вашим потребностям.
- Приемлемые сроки поставки, возможна ускоренная поставка.
- Доставку товара в любую точку России и стран СНГ.
- Комплексную поставку.
- Работу по проектам и поставку образцов.
- Формирование склада под заказчика.
- Сертификаты соответствия на поставляемую продукцию (по желанию клиента).
- Тестирование поставляемой продукции.
- Поставку компонентов, требующих военную и космическую приемку.
- Входной контроль качества.
- Наличие сертификата ISO.

В составе нашей компании организован Конструкторский отдел, призванный помогать разработчикам, и инженерам.

Конструкторский отдел помогает осуществить:

- Регистрацию проекта у производителя компонентов.
- Техническую поддержку проекта.
- Защиту от снятия компонента с производства.
- Оценку стоимости проекта по компонентам.
- Изготовление тестовой платы монтаж и пусконаладочные работы.



Тел: +7 (812) 336 43 04 (многоканальный)

Email: org@lifeelectronics.ru