LAN I2C Adapter VM

Interface Manual
LANIICVM.DLL
TCP/IP – I2C

(Rev 1.0)

COPTONIX

Falkentaler Steig 9
D – 13467 Berlin
Phone: +49 30 617 412 48
Fax: +49 30 617 412 47
www.coptonix.com

Dynamic Link Library DLL

1 General Functions

1.1 InitNetIIC

function initnetiic (pIRQCallBack, pSlvCallBack, pVMCallBack : Pointer) : DWORD Stdcall;
When starting the host application, this function must be called if callback functions are required. The arguments ***pIRQCallBack, pSlvCallBack and pVMCallback*** are the addresses of callback functions.

TIRQCallBack = Procedure; stdcall;
This function is called when an interrupt is detected and an IRQ IN Report (0x0920) is received.

TSlvCallBack = Procedure(const data : Pointer, wSize : Word); stdcall;
This function is called when the adapter is in SLAVE MODE and is receiving data from an I2C master and a SLAVE DATA IN Report (0x8230) is received. ***data*** is a pointer to the received data, and ***wSize*** is the number of bytes received.

TVMCallBack = Procedure(const data : Pointer, wSize : Word); stdcall;
This function is called when a VM DATA IN Report (0x5000) is received. ***data*** is a pointer to the data were sent by the virtual machine 8051, and ***wSize*** is the number of bytes were received.

If a callback function is not required, then just pass ***nil*** to the function.
e.g. iniusbiic(***nil***,***nil***,@vmcallback);

If the function fails the return value is non zero.

1.2 Connect

function connect (shost : PAnsiChar; wport : Word) : DWORD Stdcall;
This function is used to connect to the *LAN I2C Adapter VM* using the IP Address ***shost*** (e.g. “192.168.1.100”) and the port number ***wport*** (e.g. 5000). If the function fails the return value is non zero.

Default Network settings:

GAR (*Gateway IP Address Register*):	192.168.1.1
SUBR (*Subnet Mask Register*):	255.255.255.0
IP (*IP Address Register*):	192.168.1.100
SHAR (*Hardware Address register -MAC-*):	00-00-00-00-00-00
TCP/IP Port:	5000
UDP Port:	3000

1.3 Disconnect

function disconnect : DWORD Stdcall;
This function is used to disconnect the client from *LAN I2C Adapter VM*. If the function fails the return value is non zero.

1.4 Connected

function connected : DWORD Stdcall;
This function is used to check whether the connection is established. The function returns zero if connection is established.

1.5 SetMode

function setmode(var ucMode, ucSlvAddr : Byte; dwTimeOut : DWORD) : DWORD; stdcall;
This function selects one of following three operating modes: MASTER MODE, SLAVE MODE or PROG MODE.

ucMode = 0 -> will not effect any changes. The function returns current mode.
ucMode = 1 -> MASTER MODE
ucMode = 2 -> SLAVE MODE
ucMode = 3 -> PROG MODE

ucSlvAddr is the adapter's own slave address (only valid in SLAVE MODE). ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.6 GetNetConfig

function getnetconfig(var pNetConfig; wSize : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to read the current network settings from the *LAN I2C Adapter VM*. ***pNetConfig*** is a pointer to the buffer that receives the data read from the device. This buffer must remain valid for the duration of the read operation. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

```
TNetConfig = record
   GAR          : DWORD;              //e.g. 0xC0A80101 = 192.168.1.1
   SUBR         : DWORD;              //e.g. 0xFFFFFF00 = 255.255.255.0
   IP           : DWORD;              //e.g. 0xC0A80164 = 192.168.1.100
   TCPPORT      : WORD;               //e.g. 5000
   UDPPORT      : WORD;               //e.g. 3000
   VMTCPPORT : WORD;                  //e.g. 5001
   VMUDPPORT: WORD;                   //e.g. 3001
   SHAR         : array[0..5] of byte; //e.g. [0x00,0x00,0x00, 0x00,0x00,0x00]
   PHYCHECK   : Byte;                 //e.g. 12 corresponds to 60s (1 minute)
   KPALIVE      : Byte;               //e.g. 12 corresponds to 60s (1 minute)
   VMKPALIVE : Byte;                  //e.g. 12 corresponds to 60s (1 minute)
End;
PTNetConfig = ^TNetConfig;
```

1.7 GetMemProtect

function getmemprotect(var ucProtect : Byte; dwTimeOut : DWORD) : DWORD; stdcall;
The function is used to read the current settings of memory protection. ***ucProtect*** returns the current settings. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

ucProtect = 0 -> Write operations enabled
ucProtect = 1 -> Write Operations disabled

1.8 GetIRQ

function getirq(var ucIRQMode : Byte; dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to read the configuration of the interrupt input while operation in MASTER MODE. ***ucIRQMode*** returns the current configuration of the interrupt input. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

Possible values for ***ucIRQMode***:
0: Interrupt disabled.
1: Interrupt enabled. Interrupt is falling-edge sensitive.
2: Interrupt enabled. Interrupt is rising-edge sensitive.

1.9 VMStartRequest

function vmstartrequest(dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to send a request to start the 8051 virtual machine. If the bytecode is already loaded, the VM starts executing the bytecode and the VM status is set to VM_RUNNING. Note: The response to this command means the command has been submitted, but the VM may not have started yet. Read the current VM status using the function ***vmstate***. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.10 VMStopRequest

function vmstoprequest(dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to send a request to stop the 8051 virtual machine. The VM stops executing the bytecode and the VM status is set to VM_ABORTED. Note: The response to this command means the command has been submitted, but the VM may not have been stopped yet. Read the current VM status using the function ***vmstate***. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.11 VMGetState

function vmgetstate(var ucstate : Byte; dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to read the state of the 8051 virtual machine. ***ucstate*** returns the current status of the VM. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

Possible values for ***ucstate***:
0: VM_ABORTED.
1: VM_RUNNING
2: VM_ABORT_REQUEST
3: VM_START_REQUEST

1.12 VMIRQRequest

function vmirqrequest(const pvmdata; ucSize : Byte;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to send up to 255 bytes to the 8051 virtual machine. If the VM is already running and the interrupt is enabled, the VM jumps into the corresponding service routine. Call the function ***VMIRQResult*** to check if the VM received and processed the data. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.13 VMIRQResult

function vmirqresult(var ucIrqResult : byte;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to read the result of the last call of a ***VMIRQRequest***. ***ucIrqResult*** is equal to 1 if the data has been received by the VM and the data is available for further processing. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.14 VMReset

function vmreset(dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to delete the VM's code memory and to reset the VM. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.15 VMCodeLoaded

function vmcodeloaded(dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to check if a binary file has already been loade into the VM's code memory and verified. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.15 VMLoadCode_A

function vmloadcode_a(pImage : PAnsiChar; var dwImageChkSum : DWORD;
var wImageSize : WORD; pVMFlashCallBack : Pointer;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to write the bytecode to the VM's code memory. ***pImage*** is a pointer to the path of a binary file of type ***PAnsiChar***. ***dwImageChkSum*** returns the calculated checksum of the bytecode. ***wImageSize*** returns the size of the bytecode written the to VM. ***pVMFlashCallBack*** is a pointer to a callback function that returns the progress status. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

TVMFlashCallBack = Procedure(wImageSize, BytesWritten : Word;
dwErr : DWORD); stdcall;

1.16 VMLoadCode_W

function vmloadcode_w(pImage : PChar; var dwImageChkSum : DWORD;
var wImageSize : WORD; pVMFlashCallBack : Pointer;
dwTimeOut : DWORD) : DWORD; stdcall;

This function behaves like ***VMLoadCode_A***, except that this function accepts the argument ***pImage*** of type PChar (WideChar).

1.17 VMVerifyCode

function vmverifycode(dwImageChkSum : DWORD; wImageSize : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;

The next step after loading the bytecode into the VM's memory is to verify the bytecode. This is done by writing the checksum of the bytecode to the VM. The VM calculates the checksum and compares it with the received checksum. The checksum is a 32Bit 2's complement of the sum of the bytecode. The sum of the bytecode and the checksum should add up to 0x00000000.

1.18 VMLoadVerifyCode_A

function vmloadverifycode_a(pImage : PAnsiChar; var dwImageChkSum : DWORD;
var wImageSize : WORD; pVMFlashCallBack : Pointer;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to write the bytecode to the VM's code memory and to verify the uploaded bytecode. ***pImage*** is a pointer to the path of a binary file of type ***PAnsiChar***. ***dwImageChkSum*** returns the calculated checksum of the bytecode. ***wImageSize*** returns the size of the bytecode written the to VM. ***pVMFlashCallBack*** is a pointer to a callback function that returns the progress status. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

TVMFlashCallBack = Procedure(wImageSize, BytesWritten : Word;
dwErr : DWORD); stdcall;

1.19 VMLoadVerifyCode_W

function vmloadverifycode_w(pImage : PChar; var dwImageChkSum : DWORD;
var wImageSize : WORD; pVMFlashCallBack : Pointer;
dwTimeOut : DWORD) : DWORD; stdcall;

This function behaves like ***VMLoadVerifyCode_A***, except that this function accepts the argument ***pImage*** of type PChar (WideChar).

1.20 VMLoadToEEP_A

function vmloadtoeep_a(pImage : PAnsiChar; wAddr : WORD;
ucPageSize : Byte; pVMFlashCallBack : Pointer;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to write the bytecode into the onboard memory (EEPROM). ***pImage*** is a pointer to the path of a binary file of type ***PAnsiChar***. ***wAddr*** is the starting address where the bytecode is to be written into the EEPROM. ***ucPageSize*** the page size of the EEPROM (64 bytes). ***pVMFlashCallBack*** is a pointer to a callback function that returns the progress status. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.21 VMLoadToEEP_W

function vmloadtoeep_w(pImage : PChar; wAddr : WORD;
ucPageSize : Byte; pVMFlashCallBack : Pointer;
dwTimeOut : DWORD) : DWORD; stdcall;

This function behaves like ***VMLoadToEEP_A***, except that this function accepts the argument ***pImage*** of type PChar (WideChar).

1.22 VMLoadFromEEP

function vmloadfromeep(wAddr : WORD; dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to load the bytecode from the EEPROM at the givin address (***wAddr***) to the VM's code memory. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.23 VMSetParam

function vmsetparam(ucLoad : Byte; wAddr : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to set the VM's onboot options. The bytecode can be automatically loaded from the EEPROM into the VM's code memory after the adapter powered on and booted. ***Addr*** is the address where the bytecode is located. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

Addr	2	0x0100..0x7FFF	The size of the EEPROM is 32K bytes. Memory address in the range 0x0100 to 0x7FFF.
Load Options	1	Bit[0]	0: do not load; 1 : load bytecode into VM after boot
		Bit[1]	0: do not start; 1 : start bytecode in VM after boot
		Bit[7:2]	Reserved. Write zeros

1.24 VMGetParam

function vmgetparam(var ucLoad : Byte; var wAddr : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to read the VM's onboot options.. ***Addr*** returns the address where the bytecode is located, ***ucLoad*** returns the onboot options. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.25 GetVersion

function getversion(var wVersion : Word; dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to read the version of the firmware. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.26 MemWR8Bit

function memwr8bit(wAddr : WORD; ucVal : Byte;
dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to write an 8 Bit value (***ucVal***) to the onboard memory at address ***wAddr***. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.27 MemWR16Bit

function memwr16bit(wAddr : WORD; wVal : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to write a 16 Bit value (***wVal***) to the onboard memory at address ***wAddr***. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.28 MemWR32Bit

function memwr32bit(wAddr : WORD; dwVal : DWORD;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to write a 32 Bit value (***dwVal***) to the onboard memory at address ***wAddr***. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.29 MemWRBlock

function memwrblock(wAddr : WORD; const pmemdata; wSize : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to write a block of data to the onboard memory. ***pmemdata*** is a pointer to the buffer containing the data to be written to memory. This buffer must remain valid for the duration of the write operation. ***wSize*** is the number of bytes to write to memory. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.30 MemRD8Bit

function memrd8bit(wAddr : WORD; var ucVal : Byte;
dwTimeOut : DWORD) : DWORD; stdcall;

This command is used to read an 8 Bit value from memory at address ***wAddr***. ***ucVal*** returns the 8 Bit value read. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.31 MemRD16Bit

function memrd16bit(wAddr : WORD; var wVal : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;

This command is used to read a 16 Bit value from memory at address ***wAddr***. ***wVal*** returns the 16 Bit value read. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.32 MemRD32Bit

function memrd32bit(wAddr : WORD; var dwVal : DWORD;
dwTimeOut : DWORD) : DWORD; stdcall;

This command is used to read a 32 Bit value from memory at address ***wAddr***. ***dwVal*** returns the 32 Bit value read. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.33 MemRDBlock

function memrdblock(wAddr : WORD; var pmemdata; var wSize : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to read number of bytes (***wSize***) of data from memory starting at address ***wAddr***. ***pi2cdata*** is a pointer to a buffer that receives the data read from a memory. This buffer must remain valid for the duration of the read operation. ***wSize*** is the number of bytes to read from memory. ***wSize*** also returns the number of bytes read from memory. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.34 SaveSettings

function savesettings(dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to save the settings (operation mode, slave address and SCL frequency) to onboard memory. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

1.35 ResetSettings

function resetsettings(dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to restore factory (default) settings:
MASTER_MODE
SLAVE ADDRESS = 0x00
SCL FREQUENCY = 100 kHz
dwTimeOut is time out value in millisecond. If the function fails the return value is non zero.

1.36 GetErrorString

function geterrorstring(dwErr : DWORD) : PAnsiChar; stdcall;
This function is used to convert an error code to a readable string. For a full list of error codes provided by the DLL, see section *5 Error codes*. ***dwErr*** is the return value of all functions mentioned in this manual. The return value is the error string.

2 PROG MODE Functions

2.1 SetGAR

function setgar (ucVal1, ucVal2, ucVal3, ucVal4: Byte;
dwTimeOut : DWORD) : DWORD Stdcall;

This function is used to set new value for GAR (Gateway IP Address Register). The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command.
e.g. 192.168.1.1
ucVal1 = 192
ucVal2 = 168
ucVal3 = 1
ucVal4 = 1

dwTimeOut is time out value in millisecond. If the function fails the return value is non zero.

2.2 SetSUBR

function setsubr (ucVal1, ucVal2, ucVal3, ucVal4: Byte;
dwTimeOut : DWORD) : DWORD Stdcall;

This function is used to set new value for SUBR (Subnet Mask Register). The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command.
e.g. 255.255.255. 0
ucVal1 = 255
ucVal2 = 255
ucVal3 = 255
ucVal4 = 0

dwTimeOut is time out value in millisecond. If the function fails the return value is non zero.

2.3 SetSHAR

function setshar (ucVal1, ucVal2, ucVal3, ucVal4, ucVal5, ucVal6: Byte;
dwTimeOut : DWORD) : DWORD Stdcall;

This function is used to set new value for SHAR (Hardware Address Register -MAC-). The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command.
e.g. 00-50-C2-00-00-00
ucVal1 = 0x00
ucVal2 = 0x50
ucVal3 = 0xC2
ucVal4 = 0x00
ucVal5 = 0x00
ucVal6 = 0x00

dwTimeOut is time out value in millisecond. If the function fails the return value is non zero.

2.4 SetIP

function setip (ucVal1, ucVal2, ucVal3, ucVal4: Byte;
dwTimeOut : DWORD) : DWORD Stdcall;

This function is used to set new value for IP Address (IP Address Register). The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command.
e.g. 192.168.1.100
ucVal1 = 192
ucVal2 = 168
ucVal3 = 1
ucVal4 = 100

dwTimeOut is time out value in millisecond. If the function fails the return value is non zero.

2.5 SetTCPPort

function settcpport (wPort: Word; dwTimeOut : DWORD) : DWORD Stdcall;

This function is used to set new value for TCP/IP listening port. The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command. ***wPort*** is the port number (e.g. 5000). ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

2.6 SetUDPPort

function setudpport (wPort: Word; dwTimeOut : DWORD) : DWORD Stdcall;

This function is used to set new value for UDP port.T he new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command. ***wPort*** is the port number (e.g. 3000). ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

2.7 SetVMTCPPort

function settcpport (wPort: Word; dwTimeOut : DWORD) : DWORD Stdcall;

This function is used to set new value for TCP/IP listening port for direct communication with the virtual machine 8051. The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command. ***wPort*** is the port number (e.g. 5001). ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

2.8 SetVMUDPPort

function setudpport (wPort: Word; dwTimeOut : DWORD) : DWORD Stdcall;

This function is used to set new value for UDP port for direct communication with the virtual machine 8051. The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command. ***wPort*** is the port number (e.g. 3001). ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

2.9 SetKeepAlive

function setkeepalive (ucKPAlive: Byte; dwTimeOut : DWORD) : DWORD Stdcall;
This function is used to set new value for the time interval for the KEEP ALIVE packets of socket 1 (TCP/IP). The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command. ***ucKPALIVE*** is the time interval. The time unit is 5 seconds. A value of e.g. 12 corresponds to 60 seconds, in this case KEEP ALIVE packets are sent every 60 seconds (1 minute). KEEP ALIVE packets are only transmitted after data has been received from and sent to a client. If a client fails to answer a KEEP ALIVE packet, the socket is disconnected. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

2.10 SetVMKeepAlive

function setvmkeepalive (ucKPAlive: Byte; dwTimeOut : DWORD) : DWORD Stdcall;
This function is used to set new value for the time interval for the KEEP ALIVE packets of socket 3 (TCP/IP for direct communication between a client and the VM 8051). The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command. ***ucKPALIVE*** is the time interval. The time unit is 5 seconds. A value of e.g. 12 corresponds to 60 seconds, in this case KEEP ALIVE packets are sent every 60 seconds (1 minute). KEEP ALIVE packets are only transmitted after data has been received from and sent to a client. If a client fails to answer a KEEP ALIVE packet, the socket is disconnected. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

2.11 SetPHYCheck

function setphycheck (ucPhyCheck: Byte; dwTimeOut : DWORD) : DWORD Stdcall;
This function is used to set new value for the time interval for checking network cable connection. The new value is written directly into the memory (EEPROM). The new value is valid after power-up or after a REINIT command. ***ucPhyCheck*** is the time interval. The time unit is 5 seconds. A value of e.g. 12 corresponds to 60 seconds, in this case the PHYLINK and the cable connection are checked every 60 seconds (1 minute). ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

2.12 LoadFactorySettings

function loadfactorysettings (dwTimeOut : DWORD) : DWORD Stdcall;
loads factory (default) network settings. On next power-on **or** after calling the function *ReInit* the converter uses new settings. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

Default Network settings:

GAR (*Gateway IP Address Register*):	192.168.1.1
SUBR (*Subnet Mask Register*):	255.255.255.0
IP (*IP Address Register*):	192.168.1.100
SHAR (*Hardware Address register -MAC-*):	00-00-00-00-00-00
TCP/IP Port:	5000
UDP Port:	3000
TCP/IP Port (VM):	5001
UDP Port (VM):	3001
KEEP ALIVE Time Interval:	0 (disabled)
KEEP ALIVE Time Interval (VM):	0 (disabled)
PHY LINK Check Time Interval:	12 (60 seconds)

2.13 ReInit

function reinit (dwTimeOut : DWORD) : DWORD Stdcall;
New Network settings becomes valid after power-on or after calling the function "*ReInit*". The Function "*ReInit*" reinitialize the adapter. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

2.14 SetMemProtect

function setmemprotect(ucProtect : Byte; dwTimeOut : DWORD) : DWORD; stdcall;
The function is used to enable/disable write operations to the onboard memory (only user memory > 0xFF) from accidental write operations. Write operations are disabled to the memory when *ucProtect* is set to 0x01. If *ucProtect* is set to 0x00, protection is disabled and writes are allowed. The new value is written directly into the memory (EEPROM). ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

ucProtect = 0 -> Write operations enabled
ucProtect = 1 -> Write Operations disabled

3 MASTER MODE Functions

3.1 WriteI2C

function writei2c(ucSlvAddr : Byte; const pi2cdata; wSize : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to write data to an I2C device. ***ucSlvAddr*** is the slave address of an I2C device. ***pi2cdata*** is a pointer to the buffer containing the data to be written to a device. This buffer must remain valid for the duration of the write operation. ***wSize*** is the number of bytes to write to the device. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

3.2 ReadI2C

function readi2c(ucSlvAddr : Byte; var pi2cdata; var wSize : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;

This funtion is used to read data from an I2C device. ***ucSlvAddr*** is the slave address of an I2C device. ***pi2cdata*** is a pointer to a buffer that will receive data read from a device. This buffer must remain valid for the duration of the read operation. ***wSize*** is the number of bytes to be read from the device. ***wSize*** also returns the number of bytes read from the device. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

3.3 WRDI2C

function wrdi2c(ucSlvAddr : Byte; const pwrdata; wwrSize : WORD;
var prddata; var wrdSize, : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to write and read data from an I2C device separated by a RepeatedStart. ***ucSlvAddr*** is the slave address of an I2C device. ***pwrdata*** is a pointer to the buffer containing the data to be written to a device. This buffer must remain valid for the duration of the write operation. ***wwrSize*** is the number of bytes to be write to the device. ***prddata*** is a pointer to a buffer that will receive the data read from a device. This buffer must remain valid for the duration of the read operation. ***wrdSize*** is the number of bytes to be read from the device. ***wrdSize*** also returns the number bytes read from the device. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

3.4 ChkSlvAddr

function chkslvaddr(ucslvaddr : Byte; var ucConnected : byte;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to check if the slave address ***ucSlvAddr*** is connected to the I2C bus. If ***ucConnected*** returns 0, the device is not connected. If ***ucConnected*** returns 1, the device is connected to the I2C bus. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

3.5 ScanI2C

function scani2c(var pscandata; var ucSize : byte;
dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to scan the I2C bus for devices currently connected to the bus. ***pscandata*** is a pointer to a buffer that receives the list of detected slave addresses (devices). This buffer must remain valid for the duration of the scan operation. ***ucSize*** returns the number of discovered devices. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

3.6 SetSCL

function setscl(dwfreq : DWORD; dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to set a new value for I2C clock frequency. ***dwfreq*** is the new frequency (in [Hz]) in the range of 500Hz through 400kHz. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

3.7 GetSCL

function getscl(var dwfreq : DWORD; dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to read the current value of the I2C clock frequency. ***dwfreq*** is the frequency (in [Hz]) in the range of 500Hz through 400kHz. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

3.8 SetIRQ

function setirq(ucIRQMode : Byte; dwTimeOut : DWORD) : DWORD; stdcall;

This function is used to set the IRQmode while operating in MASTER MODE. The adapter has an interrupt input. e.g. if an IO-Expander is used and its interrupt output is connected to the interrupt input of the adapter, then the software recognizes all events of the IO-Expander when the state of the IOs changes. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

Possible values for ***ucIRQMode***:
0: Disable interrupt.
1: Enable interrupt. Interrupt is falling-edge sensitive.
2: Enable interrupt. Interrupt is rising-edge sensitive.

4 SLAVE MODE Functions

4.1 SetI2CSlvAddr

function seti2cslvaddr(var ucslvaddr : Byte; dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to change the slave address while operating in SLAVE MODE. ***ucslvaddr*** is the adapter's new slave address. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

4.2 WriteI2CSlaveBuf

function writei2cslvbuf(const pi2cdata; wSize : WORD; ucEvent: Byte;
dwTimeOut : DWORD) : DWORD; stdcall;
This function is used to write data to the I2C slave output buffer. ***pi2cdata*** is a pointer to the buffer containing the data to be written to the buffer. This buffer must remain valid for the duration of the write operation. ***wSize*** (max. 1024 bytes) is the number of bytes to write to the buffer. If ***ucEvent*** is set to 1, the adapter sends an interrupt signal to the master. In this way, the adapter notifies the master that data is ready to read. ***dwTimeOut*** is time out value in millisecond. If the function fails the return value is non zero.

5. Error codes

```
RET_OK                              = $00000000;

ERR_UNABLE_TO_CONNECT               = $00010001;
ERR_UNABLE_TO_DISCONNECT            = $00010002;
ERR_DISCONNECTED                    = $00010003;
ERR_COMM_UNKNOWN                    = $00020000;
ERR_NET_WRITE                       = $00020001;
ERR_NET_READ_TIMEOUT                = $00020002;
ERR_NET_READ_WAIT_FAILED            = $00020003;
ERR_NET_READ_WAIT_ABANDONED         = $00020004;
ERR_NET_IO_CALLBACK                 = $00020005;
ERR_RD_TH_TIMEOUT                   = $00030001;
ERR_RD_TH_WAIT_FAILED               = $00030002;
ERR_RD_TH_WAIT_ABANDONED            = $00030003;
ERR_CREATE_SYNC_EVENT               = $00040001;
ERR_IO_TO_TH_PULSEEVENT             = $00040002;
ERR_TH_TO_IO_SETEVENT               = $00040003;
ERR_TH_TO_IO_RESETEVENT             = $00040004;
ERR_IO_TO_TH_SETEVENT               = $00040005;
ERR_CREATE_MMF_HANDLE               = $00050001;
ERR_CREATE_MMF                      = $00050002;
ERR_CREATE_MMF_LOCK_HANDLE          = $00050003;
ERR_LOCK_MMF                        = $00050004;
ERR_MMF_MAX_APP_NUM_REACHED         = $00050005;
ERR_WRI2C_FAILED                    = $00070001;
ERR_RDI2C_FAILED                    = $00070002;
ERR_WRI2C_INVALID_SIZE              = $00070003;
ERR_RDI2C_INVALID_SIZE              = $00070004;
ERR_INVALID_SCL_FREQUENCY           = $00070005;
ERR_I2C_ERROR                       = $00070020;
ERR_I2C_ARBLOST                     = $00070021;
ERR_I2C_BUS_ERROR                   = $00070022;
ERR_I2C_INVALID_WLEN                = $00070023;
ERR_I2C_INVALID_RLEN                = $00070024;
ERR_I2C_NAK_WADDR                   = $00070025;
ERR_I2C_NAK_RADDR                   = $00070026;
ERR_I2C_NAK_WDAT                    = $00070027;
ERR_I2C_TIMEOUT                     = $00070028;
ERR_I2C_BUSY                        = $00070029;
ERR_I2C_DEVICE_NOT_CONNECTED        = $0007002A;
ERR_HW_UNKNOWN                      = $00080000;
ERR_MEMORY_WRITE_FAILED             = $00080001;
ERR_MEMORY_READ_FAILED              = $00080002;
ERR_MEMORY_RESTRICTED               = $00080003;
ERR_MEMORY_PROTECTED                = $00080004;
ERR_MEMORY_INVALID_ADDR             = $00080005;
ERR_MEMORY_INVALID_SIZE             = $00080006;
ERR_SAVE_SETTINGS_FAILED            = $00080007;
ERR_RESET_SETTINGS_FAILED           = $00080008;
ERR_VM_COMM                         = $00090001;
ERR_VM_RUNNING                      = $00090002;
ERR_VM_ABORTED                      = $00090003;
ERR_VM_CODE_ADDR                    = $00090004;
ERR_VM_CODE_SIZE                    = $00090005;
```

```
ERR_VM_CODE_CHECKSUM          = $00090006;
ERR_VM_CODE_NOT_LOADED        = $00090007;
ERR_VM_IRQ_DATA_SIZE          = $00090008;
ERR_VM_IRQ_REQUEST            = $00090009;
ERR_UNKNOWN_CMDID             = $000F0001;
ERR_UNKNOWN_CMD               = $000F0002;
ERR_MODE_CMD                  = $000F0003;
```

Revision history

Rev.	Date	Description
1.0	05 JULY 2022	Initial release of the Dynamic Link Library TCP/IP - I2C