EC5xx Mon Documentation

Version 1.07
© Würz-elektronik

Ingenieurbüro Edwin Würz
Im Burgfeld 4
D- 35781 Weilburg
Tel: ++49 6471 629 884; Fax: ++49 6471 629 885
info@wuerz-elektronik.com
http://www.wuerz-elektronik.com

Content

1Overview

The EC5xxMon is a small monitor which handles the basic operations for the MPC5xx processor and the peripherals of the EC5xx board. The monitor will first initialize the processor module then the memory interface and will support you some functions for testing and configuration.

1.1History

Vers.	Date	Change
1.00	2000-01-08	First final release
1.01	2000-21-09	History added
		Flash-size corrected
		Add flash-test
1.02	2002-02-01	Updated to version 0.45 of ec5xxmon
1.03	2002-05-09	Updated to version 0.50 of ec5xxmon
1.04	2002-08-01	Docum of application-interface
1.05	2002-10-15	Updated to version 0.59 of ec5xxmon
		Add ec5xxlightmon
1.06	2003-04-09	Updated to version 0.67 of ec5xxmon
1.07	2004-02-24	Add extensions for EC565 Module
		Updated to version 0.74 of ec5xxmon

2System Configuration

The EC5xxMon will initialize the MPC5xx-board, this will include the MPC5xx processor, the external devices and clock setting.

There are two versions of the EC5xx module (standard and light). The light version has no external flash and has a limitation to access up to three times 256 kBytes (with chip-select 0, 2,3).

2.1Clock configuration

The EC5xx board uses a 4 MHz crystal as clock source, the monitor will configure the internal system pll (spll) to multiply this frequency to the final frequency (MPC555: *10 = 40 MHz, MPC565: *14 = 56 MHz). This is the base of all other clocks in the system.

The timebase/decrementer (TB/DEC) use the systemclock (40/56 MHz) divided by 16, so we have a resolution of 400/286 ns (= 2.5/3.5 MHz). This configuration could be changed to fit for your application-software.

2.2Memory configuration

With the MPC5xx it is possible to connect peripherals in three different types. First the internal peripheral (which is mapped by an internal memory-map-register) supported by Motorola. Then there is a memory controller which supports up to four chip-select lines. These are used for external devices like the sram and the flash on the EC5xx board. Last it is possible to connect additional peripherals over the MPC5xx bus-interface (not used by the EC5xx / monitor, but you can connect peripherals over the system-bus of the EC5xx board).

2.2.1Internal memory configuration

There are two possible locations for the internal memory, this depends which type of monitor is used for the EC5xx-board. In the first configuration the monitor uses the external flash, then the internal memory is located at address 0x01c00000 and ends at 0x01ffffff. If the monitor is located in the internal flash, then start-location is 0x00000000 and the last cell is 0x003fffff. For the description of

the specific devices please consult the MPC5xx manual. The internal flash is enabled in both cases.

2.2.2Memory controller

The EC5xx board uses only chip-select 0 (external flash) and cs1 (external sram) of the MPC5xx memory controller. Chipselect 2 and 3 are available for additional applications.

2.2.2.1CS0 external flash

Title\Board	***EC555***	***EC555Light*)***	***EC565***
Wait-states / Timing	3	Not used	4
Burst	Not used	-	6-2-2-2
Accesstime	5 clocks = 125 ns	-	6 clocks = 107 ns
Burst access (4 beats)	20 clocks = 500 ns	-	12 clocks = 214 ns
Wait after read	1 clock	-	1 clock
Buswidth	32	-	32

*) external Flash not available on EC5xxlight (CS0 could be used by application).

2.2.2.2CS1 external sram

Title\Board	***EC555***	***EC555Light***	***EC565***
Wait-states / Timing	0	0	0
Burst	2-1-1-1	2-1-1-1	2-1-1-1
Accesstime	2 clocks = 50 ns	2 clocks = 50 ns	2 clocks = 18.9 ns
Burst access (4 beats)	5 clocks = 125 ns	5 clocks = 125 ns	5 clocks = 89,3 ns
Wait after read	0	0	0
Buswidth	32	32	32

2.2.3Address-location overview

2.2.3.1EC555light without external flash

	Size	***Mon in internal flash***	
Internal memory map	4 MB	0x00000000	0x003FFFFF
CS0: not - used	-		
CS1: external sram	1 MB	0x80000000	0x800FFFFF
CS2: not - used	-		
CS3: not - used			

2.2.3.2EC5xx with external flash

	Size	*Mon in internal flash*		*Mon in external flash*	
Internal memory map	4 MB	0x00000000	0x003FFFFF	0x01C00000	0x01FFFFFF
CS0: external flash	2 MB	0xC0000000	0xC01FFFFF	0x00000000	0x001FFFFF
	4 MB	0xC0000000	0xC03FFFFF	0x00000000	0x003FFFFF
	8 MB	0xC0000000	0xC07FFFFF	0x00000000	0x007FFFFF
CS1: external sram	1 MB	0x80000000	0x800FFFFF	0x80000000	0x800FFFFF
	2 MB	0x80000000	0x801FFFFF	0x80000000	0x801FFFFF
CS2: not - used					
CS3: not - used					

2.2.4System protection configuration

The MPC5xx has some protection-features implemented, like the watchdog and the bus monitor timeout counter. Since this setting could only be changed one time after a reset, the monitor will not access these devices.

The reset-configuration in this case is:

- Watchdog-timer is enabled and will generate a reset after 64k * 2048 clocks (~3.3 sec 40 MHz, ~2.4 sec 56 MHz).
- Bus-monitor-timer is disabled.

Since the bus-monitor-timer is disabled, you will not get a machine-check exception if you access a location which is not covered by the memory controller or inside the internal address-range. In this case the watchdog will generate a reset.

While the watchdog is enabled it is necessary to trigger the watchdog service-register, this is done by the EC5xxmon (in the decrementer-exception). So if you disable interrupts on the MPC5xx processor you should first disable the watchdog or trigger the service-logic (inside the 3.3/2.4 second period).

3Boot up sequence

The EC5xxMon will first initialize the MPC5xx processor (clock, interrupts, ...) and the external peripherals, like flash and sram. Each step will be printed out on SCI1 (on SCI2 there is only a message that it is the wrong line) which uses as a default 9600 baud. After the startup-message the monitor will wait for the selected baudrate (this is done by analyzing the start-bit of the serial communication), this is done by pressing [CR]. Baudrates from 300 up to 38400 bit/sec will work. If you use a different baudrate then 9600 you will not see the first 12 lines (only some trash).

The startup message looks like this:

```
01: Welcome to the EC5x5 board
02: ========================
03: EC5x5Mon Version 0.74g (use external ROM)
```

```
04: Compile-date: Feb 24 2004, time: 17:36:24
05: Sysclock: 56.00 MHz
06: Startup interrupt-controller -->OK
07: Flash ID = 0x2203 - AM29BL162C (Size = 0x00400000, Burst ON!)
08: Fast ext. Ram-Test (0x00200000) -->OK
09: init unused stack -->OK
10: Startup serial-driver -->OK
11: The actual time = 1.January 1970 - 01:01:28
12: Checking autoboot: external Flash, internal Flash  ==> Nothing!
13: Check baudrate (wait for CR-Key)
14: New configured baudrate = 9600
15: Now starting menu
16: (c) by Wuerz elektronik
17: e-mail: info@wuerz-elektronik.com
18: http://www.wuerz-elektronik.com
```

Line 1-4:	Startup-message (version, date of monitor)
Line 5:	used Systemclock of board
Line 6:	The interrupt-controller has to be configured very early because the ec5xxmon needs the decrementer exception for watchdog-trigger.
Line 7:	Check type of external flash
Line 8:	Next step is a quick external ram-test.
Line 9:	Stack-initialisation (only for stack-checking).
Line 10:	Startup interrupt driver serial device.
Line 11:	Prints the actual time (RTC has NO battery-backup).
Line 12:	Search internal and external flash for other applications (see chapter 4.3)
Line 13-14:	Check the actual baudrate (by pressing ENTER)
Line 16-:	EC5xx-message (if you have an autoboot-application, this will not be displayed)

After this message the monitor will display the ec5xx-menu. (see chapter 4.).

```
-- EC5xx Menu --    {help = ?}
========================================================================
```

4Menu-Operation

The EC5xxMon uses a easy to handle menu-system, to handle the monitor commands. The menu is able to handle new commands when the prompt character ':' is displayed.

4.1Handling and display of a menu

In the menu-system you have two types of entries, first a command and second a sub-menu. You can recognize a sub-menu with the text „--> next menu “ at the end of a line. To select a command or change to a sub-menu press only the key which is shown in the first column followed by a [CR]. It is possible to process multiple commands in one line (i.e. 'ABC [CR]' the commands ABC are executed), but its not possible to go into multiple sub-menus! It is not allowed to have spaces between the commands, because after the first space the rest of the line is used as parameters. You can start each command multiple times, if you add a decimal number before the command list. If you choose 0, the command-list is handled until you press the [ESC] key.

Here is a short list of possibilities to execute commands (not sub-menus):

Execute one command (A):	*A[CR]*	
Execute one command 3 times (A):	*3A[CR]*	
Execute one command endless (A):	*0A[CR]*	*-- press [ESC] to abort --*
Execute two commands 3 times (AC):	*3AC[CR]*	

Each command could have a parameter, this is signaled by two brackets after the command-name.

```
B) Addr Bus Test (addr,len)
```

In this case we have two parameters addr and len. If you enter no parameter, by only press B[CR] the command could select a default. In this special case the complete external ram will do an address-bus test, since a board could have different size of ram a constant default parameter is not so powerfull.
A function could also have a default parameter:

```
D) Applications (erom,irom,ram)                110
```

In this case '110' means that the function will use data '110' if you do not enter an own parameter. All text which follows the first space-character is interpreted as parameters. So you could call this command „Applications“ with a parameter of '100' by typing „A [SPACE] 100 [CR].
If you select multiple commands all commands will get the same parameter.
After each command you get a response (minimum OK or ERROR).

4.2Commands

Now lets look at all commands in the EC5xxMon:

4.2.1EC5xx Menu

This is the base menu of the EC5xxMon.

```
-- EC5xx Menu --    {help = ?}
=====================================================================
A) Tools                                        --> next menu
B) Configuration                                --> next menu
C) Test                                         --> next menu
D) Applications (erom,irom)                     11
.) Quit

-- EC5xx Menu --    {help = ?}
=====================================================================
:
```

Points A to C are all submenus which we will see later.
Attention, if you press '.' (== Quit) in this menu the EC5xxMon will do a reboot, because there is nothing else to do!

4.2.1.1Applications

This menu-entry will search in the rom-areas for a specific data-block, which indicates an additional application. If there is found minimum one application this could be started. Which memory regions should be processed can be defined by the parameter. There are three different locations, external-, internal- flash and external ram. The default is 11 this means the external and internal flash is checked (01 = only internal rom, 10 = only external rom).
For building own applications, please look at chapter 4.3..

4.2.2Tools List

In the tools-list menu you find commands to make some tests, downloads or read/write functions.

```
-- Tools List --    {help = ?}
=====================================================================
A) MPC5xx Tools                                --> next menu
B) RamAcc Tools                                --> next menu
C) RamLoad Tools                               --> next menu
D) Internal Flash Tools                        --> next menu
E) External Flash Tools                        --> next menu
F) Can Tools                                   --> next menu
G) Eth Tools                                   --> next menu
H) Serial EEPROM Tools                         --> next menu
I) PPC Tools                                   --> next menu
J) RUN Led                                     --> next menu
.) Quit
```

Here we find only sub-menus:

- MPC5xx Tools: get information of mpc5xx like chip-select configuration, ...
- RamAcc Tools: reading / writing memory-locations, ...
- RamLoad Tools: load or store data
- Internal Flash Tools: programming, erasing, ...
- External Flash Tools: programming, erasing, ...
- Can Tools: init, transmit, ...
- Eth Tools: init, send ping, receive ping, ...
- Serial EEPROM Tools: read, write, ...
- PPC Tools: disassemble, assemble, ...
- RUN Led: Software-programmable LED (only on V4 boards and later)

For a detailed description please refer to the corresponding chapters.

4.2.3MPC5xx Tools

In the MPC5xx-Tools menu we find some configuration and information functions which belong to the MPC5xx (main task is the processor, not the io-peripherals).

```
-- MPC5xx-Tool --    {help = ?}
=====================================================================
A) Display Memory
B) Show memory-usage
C) Switch Regions [0/1]                         1
D) Switch Burst-Buffer [0/1]                    1
E) Print Serialize
F) Switch Serialize [0/1]                        0
G) Display DEC / TB
H) Switch TB/DEC [0/1]                          1
I) Read Reset Status
J) Clear Reset Status
K) Print IRQ-Regs
L) Print IRQ-Config
M) Jump to Address
.) Quit
```

4.2.3.1Display Memory

Here you get some information about the memory-configuration of the MPC5xx, like locations and the access-options for the ram and rom devices.

4.2.3.2Show memory-usage

Prints information which address-space is used by the monitor (including used stack, data-section and used rom).

4.2.3.3Switch regions

The MPC5xx has no memory-management-unit (MMU), but some basic register-support. This could improve the performance of the PowerPC code. With this command you can enable and disable the MMU-functions of the MPC5xx:

```
C 1 [ENTER]                 enables the MMU-registers
C 0 [ENTER]                 disables the MMU-registers
```

It is very important to have a blank between the command 'C' and the parameter ('0' or '1'), after the parameter there is no need for a blank (it's only used for better readability).

4.2.3.4Switch Burst-Buffer

With this feature you can enable / disable the MPC5xx-Burst buffer. If you have an external device which supports bursts (like the SRAM of the EC5xx), you can improve performance of the MPC5xx. If no device is burstable, the system will still work fine but there is no performance increase in this case.

```
D 1 [ENTER]                 enables the Burst-Buffer
D 0 [ENTER]                 disables the Burst-Buffer
```

It is very important to have a blank between the command 'D' and the parameter ('0' or '1'), after the parameter there is no need for a blank (it's only used for better readability).

4.2.3.5Print Serialize

The MPC5xx has a special feature to switch the processor to serialized mode (this is much slower, because each instruction is handled separately not in a pipeline). This feature has some advantages for debugging with an emulator or for traces. Here you can show the actual configuration of the MPC5xx (after a reset the default is full serialized == slow, but the monitor will switch to non-serialized).

4.2.3.6Switch Serialize

Here you can switch the actual configuration of the MPC5xx for serialization (see chapter 4.2.3.5).

```
F 1 [ENTER]                 switch to full serialized (slow)
F 0 [ENTER]                 switch to not serialized (fast)
```

It is very important to have a blank between the command 'F' and the parameter ('0' or '1'), after the parameter there is no need for a blank (it's only used for better readability).

4.2.3.7Display decrementer (DEC) or timebase (TB)

Shows the actual count value of the timebase and the decrementer.

4.2.3.8Switch TB/DEC

The timebase and the decrementer can only be enabled / disabled both at a time! The monitor uses the DEC-exception to handle the watchdog!

```
H 1 [ENTER]                 enable DEC/TB
H 0 [ENTER]                 disable DEC/TB
```

It is very important to have a blank between the command 'H' and the parameter ('0' or '1'), after the parameter there is no need for a blank (it's only used for better readability).

*** ATTENTION *** when you disable the DEC the watchdog is not triggered any more. This means you will get a reset after some seconds! Only if you disable the watchdog the monitor will continue operation.

4.2.3.9Read Reset-Status

This command displays the cause of the last reset (or the last n resets!).

4.2.3.10Clear Reset-Status

This is very important, because on the next reset you can have two reset-cause bits (for example watchdog-reset and checkstop-reset (but which was it????). This should be done if you have checked

the last cause!

4.2.3.11Print IRQ-Regs

Here you can check the MPC5xx- interrupt-controller, which interrupts are enabled, and which interrupts are pending.

4.2.3.12Print IRQ-Config

Prints all possible IRQ´s of the MPC5xx and which IRQ's are enabled in the monitor.

4.2.3.13Jump to address

Jump to specific location (given as a parameter).

```
Example:       M 0x12345678 [ENTER]                jump to 0x12345678
```

4.2.4Ram-tools

With the ram-tools you have the possibility to examine any valid location in the system. You can also modify all memory-mapped devices of the MPC5xx (like ram or internal registers).

```
-- RamAcc-tools --    {help = ?}
=========================================================================
A) Fill Memory (start,cnt,data)               0 0 0
B) Copy Memory (dst,src,cnt)                  0 0 0
C) Find 1. Diff (s1,s2)                       0 1
D) Dump Memory (start,len)
E) Read  8-Bit (addr[,cnt])                   0
F) Read 16-Bit (addr[,cnt])                   0
G) Read 32-Bit (addr[,cnt])                   0
H) Write  8-bit (addr,data)                   0 0
I) Write 16-bit (addr,data)                   0 0
J) Write 32-bit (addr,data)                   0 0
K) WriteVerfy  8-bit (addr,data)              0 0
L) WriteVerfy 16-bit (addr,data)              0 0
M) WriteVerfy 32-bit (addr,data)              0 0
.) Quit
```

4.2.4.1Fill memory

The fill-memory can be used to program HEX values (only hex-data) to the memory. This function has three parameters:

start = start-location to program (hex/dec)

cnt = count of repetition of the data (hex/dec)

data = is a byte-array (ONLY IN HEX - no 0x allowed)

```
Example program Data: 0x12, 0x34, 0x88 to location 0x20000 3 times:
E 0x20000 3 12 34 88 [ENTER]

if you make a dump you to 0x2000 you will get this result:
0x00020000 - 12 34 88 12 34 88 12 34 88 xx xx xx xx xx xx xx ???????????????
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.4.2Copy memory

With this function you can copy any number bytes from source to destination.

```
Example copy 128k from 0 to 0x80000:
F 0x80000 0 0x20000 [ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for

a blank (it's only used for better readability).

4.2.4.3Find first differences

This function finds the first differences between source 1 and source 2. ATTENTION if the complete device is identical you will get an invalid access to an unassigned location --> RESET!

```
Example: find first difference between ram (0x80000000) and external flash (0x00000000)
G 0x80000000 0 [ENTER]
Example 2: find first not empty cell of the flash
G 0 1 [ENTER]                * the first cell should be 0xff!
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.4.4Dump memory

Dump memory makes a hex-dump to the terminal of any valid address-location of the EC5xx board. The default writes the address location 0 to 16 to the terminal. You can enter either decimal or hex address / count values.

```
Example: Dump address 0xc0000000 (256 Bytes)
D 0xc0000000 256 [ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.4.5Read functions

The read-functions are used to read a 8-bit value (h), 16-bit (i) and 32-bit (j). You get the result in hex and decimal. You have to provide an address (default would be zero).

4.2.4.6Write functions

The write-functions are used to write a 8-bit value (k), 16-bit (l) and 32-bit (m). You have to provide the address plus the data to write. There is no verification of the written data.

4.2.4.7Write and verify functions

The write-functions are used to write a 8-bit value (n), 16-bit (o) and 32-bit (p). You have to provide the address plus the data to write. The written data is verified.

4.2.5RamLoad-tools

With the ram-tools you have the possibility download data to ram, or to upload data from any valid location.

```
-- RamLoad-tools --    {help = ?}
========================================================================
A) Fill Memory (start,cnt,data)               0 0 0
B) Copy Memory (dst,src,cnt)                  0 0 0
C) Find 1. Diff (s1,s2)                       0 1
D) Dump Memory (start,len)
E) Download Srec (off)
F) Download XModem (off)
G) Upload XModem (addr,cnt)
H) Upload XModem1k (addr,cnt)
I) Verify Srec (off)
J) Verify Srec (abs)                          0
K) Verify XModem (off)
L) Verify XModem (abs)                        0
.) Quit
```

4.2.5.1Fill Memory,Copy Memory, Find 1.Diff, Dump Memory

Please look at the corresponding chapters 4.2.4.x.

4.2.5.2Download S-record to sram

With this option you can download Motorola s-records (all types S1,S2 and S3). The s-record file contains an address where to store the information. This address information is treated as an offset to the local sram. You can add an additional offset to the address location with the parameter.

```
Some examples:
A           [ENTER]                         : address = SRAM + SREC-Addr
A 0         [ENTER]                         : address = SRAM + SREC-Addr
A 0x1000    [ENTER]                         : address = SRAM + 0x1000 + SREC-Addr
To download in an address outside the SRAM: (sram = 0x80000000, location = 0x12345678)
A 0x92345678[ENTER]                         : address = SRAM + 0x92345678 + SREC-Addr
[you have to add (0 - SRAM-location) + offset]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.5.3Download XModem to sram

Download using the xmodem protocol. This will support standard packet size (128 Bytes of data) and also the 1k packet size. The download will use the normal checksum for error detection.

Since the xmodem-protocol does not have any size-information, this protocol will transmit up to 127 bytes (if your terminal use only 1k blocks up to 1023 bytes) of extra data, which will be written to ram.

4.2.5.4Upload Xmodem from any valid location

Send binary data to terminal using the Xmodem protocol, this will transmit 128 bytes payload per frame. It supports normal checksum and CRC16 to validate the frame. The type of checking depends on the receiver.

4.2.5.5Upload Xmodem1k from any valid location

Send binary data to terminal using the Xmodem1k protocol, this will transmit 1024 bytes payload per frame. It supports normal checksum and CRC16 to validate the frame. The type of checking depends on the receiver.

4.2.5.6Download XModem to sram

Download using the xmodem protocol. This will support standard packet size (128 Bytes of data) and also the 1k packet size. The download will use the normal checksum for error detection.

4.2.5.7Verify srecord

This will download a srecord-file and verify the data which should be there. There are two versions one with offset (to the SRAM start) and another with absolute system-addresses.

4.2.5.8Verify XModem

This will download a binary-file using the xmodem-protocol and verify the data which should be there. This will support the same options as xmodem-download. There are two versions one with offset (to the SRAM start) and another with absolute system-addresses.

4.2.5.9Download Binary

This options is obsolete (only available on older versions of the monitor), please use the xmodem-implementation instead.

Download binary is basically identical to the download s-record. The differences are:

- The transmitted bytes are binary not coded to ascii like s-records (so binary filesize is 1/3 of the srecord filesize)
- There is no address-offset like s-record (so we have only the starting-location)
- There are some additional bytes/words which are required for the receive process.

Fileformat:

4 Bytes filesize (big endian: most significant byte will be received first)
n Data-bytes (the data stream has an escape sequence)
4 Bytes CRC32 checksum (polonium is appended)

To support abort and some additional signaling the ESC-character (0x1b) is handled specially. If the EC5xxmon receives an ESC the monitor will check the next character, if this is a 0x00 it will append 0x1b to the data, if there is again a 0x1b the monitor will abort the receive process. A code piece for download is appended in the chapter 5.2..

4.2.6Internal Flash commands

With this library you can modify the internal mpc5xx flash, like erase, program, ...)

```
-- Internal Flash Commands --    {help = ?}
=====================================================================
A) Program Srec (off)
B) Program XModem (off)                        0x40000
C) Program Data (off,sa,cnt)
D) Erase all
E) Erase (start,cnt)                           0 0
F) Verify Srec (off)
G) Verify XModem (off)
H) Blank-Check (off,cnt)
I) Set RxBufsize (addr,size)
J) Set RxBufsize (addr,size)                   0x40000 0xc0000
K) Read Reset Config
L) Write Reset Config                          0
M) Erase Reset Config
N) Read 64 Bytes (off)                         0
O) Decensore Flash (A,B)
P) Censor Flash (A,B)
Q) Switch Mon to ram
R) Prog Ram-Mon to Flash
.) Quit
```

4.2.6.1Download S-record to internal flash

With this option you could download Motorola s-records (all types S1,S2 and S3). The s-record file contains an address where to store the information. This address information is treated as an offset to the internal flash. You can add an additional offset to the address location with the parameter. To program new data to the flash, this device must be erased! The received data is programmed immediately to the internal flash. The programming feature must be enabled with the corresponding jumper (please check the mpc5xx hardware manual).

```
Some examples:
A           [ENTER]                        : address = IFLASH + SREC-Addr
A 0         [ENTER]                        : address = IFLASH + SREC-Addr
A 0x1000    [ENTER]                        : address = IFLASH + 0x1000 + SREC-Addr
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

Dependent on the programming-speed and the download rate, there could be a loss of data on the download, since the monitor does not use any handshake. To minimize the data-loss problem, you can use the external ram as receive buffer, please check “set RxBufsize”.

The programming voltage must be enabled with the corresponding jumper (please check the MPC5xx hardware manual).

ATTENTION: On the MPC555 the programming time is highly dependent to the supply voltage, there could be download-errors on high-bitrates (to get better results please use minimum 5.0 V).

4.2.6.2Download xmodem to internal flash

Download using the xmodem protocol. This will support standard packetsize (128 Bytes of data) and also the 1k packet size. The download will use the normal checksum for error detection.

Since the xmodem-protocol does not have any size-information, this protocol will transmit up to 127 bytes (if your terminal use only 1k blocks up to 1023 bytes) of extra data, which will be written to ram.

The receive-buffer is large enough to hold a complete xmodem frame, so there is no need to expand the rx-buffer.

The programming voltage must be enabled with the corresponding jumper (please check the MPC5xx hardware manual).

ATTENTION: On the MPC555 the programming time is highly dependent on the supply voltage, there could be download-errors on high-bitrates (to get better results please use minimum 5.0 V).

4.2.6.3Program data to internal flash

With this feature you can program any data, which is available on the MPC5xx processor, to the internal flash, we have three parameters, the destination offset (starting from internal flash start), the source address (should be outside this flash-device) on any location plus a count-value where the amount of data is configured.

```
To program the 96k from location 0x8000000 to the internal flash (0x01c00000)
C 0x01c00000 0x80000000 0x18000 [ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

The programming voltage must be enabled with the corresponding jumper (please check the MPC5xx hardware manual).

ATTENTION: On the MPC555 the programming time is highly dependent to the supply voltage (to get better results please use minimum 5.0 V).

4.2.6.4Erase complete flash

Both flashes (MPC555: A = 256k + B = 192 k, MPC565: A = 512k + B = 512k) will be erased completely.
The programming voltage must be enabled with the corresponding jumper (please check the

MPC5xx hardware manual).

ATTENTION: When you erase the internal flash, the reset-configuration-word will be erased too, and have to be reprogrammed!

4.2.6.5Erase blocks

This function erases one or more blocks of the flash (one block has the size of 32 kBytes on MPC555 and 64 kBytes on MPC565). The starting location (offset of flash-start) plus the count of bytes which should be erased must be given as a parameter (the count is rounded UP to the next 32k/64k-boundary).
The programming voltage must be enabled with the corresponding jumper (please check the MPC5xx hardware manual).

```
Example to erase the first 128 k of internal flash
e 0 0x20000 [ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

ATTENTION: When you erase the first sector (0..0x8000 on MPC555, 0..0x10000 on MPC565) the reset-configuration-word will be erased too, and have to be reprogrammed!

4.2.6.6Verify srecord with internal flash

Is similar to download srecord, except this code will verify instead of write to the flash. For a detailed documentation pleas look to chapter 4.2.6.1.. This function is identical to the SRAM version.

4.2.6.7Verify xmodem with internal flash

It is similar to download xmodem, except this code will verify instead of write to the flash. For a detailed documentation please refer to chapter 4.2.6.2.. This function is identical to the SRAM version.

4.2.6.8Blank Check (off, cnt)

With this feature you can check if the Flash is empty at the specified locations (start-address bytecount). It will run too, if you work currently with dual-mapping (internal monitor works from the external ram).

4.2.6.9Set RxBufsize (addr, size)

Here you can use the external ram as the io-buffer for serial data. Since the internal flash-programming takes much time, there could be a buffer-overrun. With this feature you can assign the complete external ram as io-buffer. The second option (with default parameters is fine for a 1MB – ec5xx-board running the monitor from external ram.

4.2.6.10Read reset-configuration value of shadow-flash

The MPC5xx has a special feature which supports a programmable reset-configuration. This could be read with the command (check MPC5xx Reference manual for detail description).

4.2.6.11Write reset-configuration

To enable the boot-process from the internal flash you have to write into the reset-configuration-shadow-flash. To program new data to the reset-configuration, this area must be erased first! The received data is programmed immediately to the internal flash.

The programming voltage must be enabled with the corresponding jumper (please check the MPC5xx hardware manual).

4.2.6.12Erase reset configuration

To change cleared bits in the reset-configuration-word you have to erase this value first. Then you can program a new value.

The programming voltage must be enabled with the corresponding jumper (please check the MPC5xx hardware manual).

ATTENTION: When you erase the reset-configuration-value the first 32/64 k (MPC555/565) of internal flash are also erased, please backup the data and reprogram it later!!!!!

4.2.6.13Read 64 Bytes (off)

With this entry you are able to read from the internal flash, even if you have copied the internal monitor to ram (dual-mapping). If you dump the content of the internal ram (while in dual-mapping), you will get the content of the external ram!

4.2.6.14Decensore Flash

Here you can decensore the internal flash (make it readable again), but this operation **will erase the complete flash-module**!

4.2.6.15Censore Flash

With this option you are able to censor the internal flash (protect from reading, if an application outside the internal flash is running). **To disable censore, this internal flash-module has to be erased completely!**

4.2.6.16Switch monitor to ram

This feature is only available when using the internal monitor (monitor used to work in internal flash).

If you have to erase/program the internal flash you need to switch the monitor to ram first. The basic operation is that the monitor is copied to ram and then the dual-mapping is activated for the monitor-range. So the monitor runs now from external ram. Now you are able to erase/program the internal flash. If you like to read content of the flash, you have to use “Read 64 Bytes <off>” (see chapter 4.2.6.13.).

ATTENTION: If your monitor runs in internal flash and you modify this, the monitor could be lost! So you may need an ICE to reprogram the monitor.

4.2.6.17Prog Ram-Mon to Flash

This feature is only available when using the internal monitor (monitor used to work in internal flash).

When loading the monitor with a debugger, you are able to program it directly to the internal flash. The Reset-Configuration-Word will be programmed also on the MPC555 to be able to start this monitor from internal flash.

4.2.6.18Download binary to internal flash

This option is obsolete (only available on older versions of the monitor), please use the xmodem-implementation instead.

Download binary is basically identical to the download s-record to internal flash. The differences are:

- The data is binary not coded to ASCII like s-record (so filesize is 1/3 of the srecord filesize)
- There is no address-offset like s-record (so we have only the starting-location)
- There is some additional data which are required for the receive process.
- The data must be in one continuous range (a s-record could contain multiple blocks)

To program new data to the flash, this device must be erased!

The programming voltage must be enabled with the corresponding jumper (please check the MPC5xx hardware manual).

Fileformat:

4 Bytes filesize (big endian: most significant byte will be received first)
n Data-bytes (some additional encoding)
4 Bytes CRC32 checksum (polonium is appended)

To support abort and some additional signaling the ESC-character (0x1b) is handled specially. If the EC5xxmon receives an ESC the monitor will check the next character, if this is a 0x00 it will append 0x1b to the data, if there is again a 0x1b the monitor will abort receive process. A code piece for download is appended in the appendix A.

Dependent on the programming-speed and the download rate, there could be a loss of data on the download, since the monitor does not use any handshake. To minimize the data-loss problem, you can use the external ram as receive buffer, please check “set RxBufsize”.

ATTENTION: since the programming time is highly dependent to the supply voltage, there could be download-errors on high-bitrates (to get better results please use minimum 5.0 V).

4.2.7External Flash commands

With this library you can modify an external flash (which has AMD-like programming algorithm). This commands are **not** available on the light version of EC5xx!

```
A) Program Srec (off)
B) Program XModem (off)                     0x40000
C) Program data (off,sa,cnt)                0 0 0
D) Chip Erase
E) Sector Erase (off,cnt)                   0 0
F) Verify Srec (off)
G) Verify XModem (off)
H) Blank-Check (off,cnt)
I) Set RxBufsize (addr,size)
J) Set RxBufsize (addr,size)                0x40000 0xc0000
K) On/Off SyncMode                          0
L) On/Off Burst (MemCtrl)                   0
M) Switch Mon to ram
N) Prog Ram-Mon to Flash
.) Quit
```

4.2.7.1Download S-record to external flash

With this option you can download Motorola s-records (all types S1,S2 and S3). The s-record file contains an address where to store the information. This address information is treated as an offset to the start of the external flash. You can add an additional offset to the address location with the parameter. To program new data to the flash, this device must be erased! The received data is programmed immediately to the external flash.

```
Some examples:
A           [ENTER]                         : address = EFLASH + SREC-Addr
A 0         [ENTER]                         : address = EFLASH + SREC-Addr
A 0x1000    [ENTER]                         : address = EFLASH + 0x1000 + SREC-Addr
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

Dependent on the programming-speed and the download rate, there could be a loss of data on the download, since the monitor does not use any handshake. To minimize the data-loss problem, you can use the external ram as receive buffer, please check “set RxBufsize”.

4.2.7.2Download xmodem to external flash

Download using the xmodem protocol. This will support standard packet size (128 Bytes of data) and also the 1k packet size. The download will use the normal checksum for error detection.

Since the xmodem-protocol does not have any size-information, this protocol will transmit up to 127 bytes (if your terminal use only 1k blocks up to 1023 bytes) of extra data, which will be written to ram.

The receive-buffer is large enough to hold a complete xmodem frame, so theree is no need to expand the rx-buffer.

4.2.7.3Program data to external flash

With this feature you can program any data which is available to the MPC5xx processor to the external flash, we have three parameters, the destination address (should be in the external flash), the source address (outside this flash-device) on any location plus a count-value where the amount of data is configured.

```
To program the 96k from location 0x8000000 to the external flash (0x01c00000)
C 0x01c00000 0x80000000 0x18000 [ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.7.4Chip erase

The complete flash will be erased.

4.2.7.5Sector erase

This function erases one or more sectors of the external flash, the mapping depends on the version of flash (top/bottom type or 1 / 4 MB). The starting location (offset of flash-start) plus the count of bytes which should be erased must be given as a parameter (the count is rounded UP to the next sector).

```
Example to erase the first 128 k of external flash
e 0 0x20000 [ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.7.6Verify srecord with external flash

Is similar to download srecord, except this code will verify instead of write to the ram. For a detailed documentation please refer to chapter 4.2.7.1.. This function is identical to the SRAM version.

4.2.7.7Verify xmodem with external flash

Is similar to download srecord, except this code will verify instead of write to the ram. For a detailed documentation please look to chapter 4.2.7.2.. This function is identical to the SRAM version.

4.2.7.8Blank check

Check the external-flash area if it is empty.

4.2.7.9Set RxBufsize

Please see chapter 4.2.5.13

4.2.7.10Switch SyncMode

You can enable/disable the sync-mode of the external flash. If the flash does not support sync-mode the function will return an error. The Sync-Mode is required to activate burst. If Burst are enabled the Sync-Mode could not be disabled.

4.2.7.11Siwtch Burst on/off

You can enable/disable burst for the external flash, if the device supports this. The flash-device will also be switched to Sync-Mode to support bursts.

4.2.7.12Switch monitor to ram

This feature is only available when using the external monitor (monitor used to work in external flash).

This feature is only required when the monitor runs in the external flash, and the external flash should be modified! Basically the monitor is copied to the SRAM and then the address-locations of the SRAM and the FLASH is moved:

	New Start	***New End***	***Old Start***	***Old End***
SRAM (1 MB)	0x00000000	0x000FFFFF	0x80000000	0x800FFFFF
SRAM (2 MB)	0x00000000	0x000FFFFF	0x80000000	0x801FFFFF
Flash (1 MB)	0xC0000000	0xC00FFFFF	0x00000000	0x000FFFFF
Flash (4 MB)	0xC0000000	0xC03FFFFF	0x00000000	0x003FFFFF
Flash (8 MB)	0xC0000000	0xC07FFFFF	0x00000000	0x007FFFFF

The SRAM is available on both locations (old and new) - The flash only at the new.

After this step you can program / erase the external flash, even if the monitor has started from this device. This configuration will stay until you press RESET! If you get an error-response when you program the flash, it is possible to continue working with the monitor in ram until you overwrite the

ram or you press RESET.

ATTENTION: If your monitor runs in external flash and you modify this, the monitor could be lost! So you may need an ICE to reprogram the monitor.

4.2.7.13Prog Ram-Mon to flash

This feature is only available when using the external monitor (monitor used to work in external flash).

If you have downloaded a monitor-version to ram (with an ICE) then you can program this version directly to the external flash!

4.2.7.14Download binary to external flash

This options is obsolete (only available on older versions of the monitor), please use the xmodem-implementation instead.

Download binary is basically identical to the download s-record to external flash. The differences are:

- The data is binary not coded to ascii like s-record (so filesize is 1/3 of the srecord filesize)
- There is no address-offset like s-record (so we have only the starting-location)
- There is some additional data which are required for the receive process.
- The data must be in one continuous range (a s-record could contain multiple blocks)

To program new data to the flash, this device must be erased!

Fileformat:

4 Bytes filesize (big endian: most significant byte will be received first)
n Data-bytes (some additional encoding)
4 Bytes CRC32 checksum (polonium is appended)

To support abort and some additional signaling the ESC-character (0x1b) is handled specially. If the EC5xxmon receives an ESC the monitor will check the next character, if this is a 0x00 it will append 0x1b to the data, if there is again a 0x1b the monitor will abort receive process. A code piece for download is appended in the appendix A.

Dependent on the programming-speed and the download rate, there could be a loss of data on the download, since the monitor does not use any handshake. To minimize the data-loss problem, you can use the external ram as receive buffer, please check “set RxBufsize”.

4.2.8Can Tools

A small library to send/receive can-messages.

```
A) Init (dev) [1M,500k,250k,125k,10k]          A 1M
B) Disable (dev)                               A
C) Print state (dev)                           A
D) Prep receive (dev,buf-idx,id)               A 0 111
E) Transmit (dev,buf-idx,id, data)             A 0 111 GUENTER!
F) Print Buffer (dev,buf-idx)                  A 0
G) Print all Buf
.) Quit
```

4.2.8.1Init

Sets the can-module-registers to operation state (including the configured baudrate). You have to enter first the device (MPC555: a-b, MPC565: a-c) and then one of the shown baudrates, the default is 1 Megabit/sec.

4.2.8.2Disable (dev)

Switch device off.

4.2.8.3Print state (dev)

Shows if the can bus is active / or if it is in error-state.

4.2.8.4Prep receive (dev,buf-idx,id)

Configure one of the 16-can-buffer (buf-idx 0..15) for receive-operation. This buffer will wait for a specific id.

```
A [ENTER]                          ; configures the CAN-A to 1MBit/s
E [ENTER]                          ; configures buffer0 to wait for a message with id 111
0G [ENTER]                         ; here the Can-Buffer A is polled (continually)
```

4.2.8.5Transmit (dev,buf-idx,id,data)

Sends one Package to the can-bus, with your data:

```
A [ENTER]                          ; configures the CAN-A to 1MBit/s
F A 3 MyMessag[ENTER]              ; Sends the message „MyMessag“ over buffer 3 with can-a.
```

4.2.8.6Print Buffer (dev,buf-id)

Shows the state/content of the specified buffer.

4.2.8.7Print all Buf

Shows all 16 message-buffer of both can-modules.

4.2.9Eth Tools

The Eth-Tools are only usable if you have an additional eth-module of Würz-Elektronik. When you have mounted this device you can make some „basic“ operations, there is no higher level support (like tcp/ip). There is only some register-access and a “simple” ping function.

```
A) Activate ChipSelect of The (1=on/0=off,CS)   1 2
B) Reset to ETH
C) Read RegisterIo                              0x20
D) Write RegisterIo
E) Read RegisterMem                             0x20
F) Write Register
G) Init Rx/Tx (mac)                             000017091964
H) Set destination (mac)                        fffffffffffff
I) Read dest-mac from frame (timeout/ms)        10000
J) Print statistics
K) Read Rx packets (print,timeout/ms)           0 10000
L) Transmit packet (addr,size)                  0 48
M) Toggle leds (cnt) times                      10
N) Set my IP-address                            192.168.222.8
O) print my IP-address
P) Send ping to (print,ip-addr,timeout)         0 192.168.222.222 4000
Q) Wait for ping (print,timeout)                0 10000
.) Quit
```

4.2.9.1Activate Chipselect of Eth (1=on/0=0ff,CS)

Here you can activate/deactivate the Eth-Module. With the first parameter you can activate (1) or deactivate (0) the CS. The parameters of the CS you can see in the MPC5xx-CS description.

4.2.9.2Reset to Eth

Makes a hardware reset to the eth-module. After the reset the module is checked if it is alive. If there is no eth-module found, you get an error-message.

4.2.9.3Read RegisterIo (roff)

You can read any onchip register of the CS8900-device (in this case over the io-interface).

4.2.9.4Write RegisterIo (roff,val)

You can write any on chip register of the CS8900-device (in this case over the io-interface). The first parameter is the register-number and the second the 16-bit-value.

4.2.9.5Read RegisterMem (roff)

Same as 4.2.8.3, except it uses the memory-interface.

4.2.9.6Write RegisterMem(roff,val)

Same as 4.2.8.4, except it uses the memory-interface.

4.2.9.7Init Rx/Tx (mac)

A mac-address is programmed to the ethernet-module, ATTENTION this address is not a valid Ethernet-Address (not unique). For this module there is a separate ID for Ethernet-Operations. With this function you can program any address to the device.

4.2.9.8Set destination (mac)

If you try to send a packet to some other devices. You can configure the destination-address here.

4.2.9.9Read dest-mac from frame (timeout/ms)

It is also possible to take the mac-address from the first receive packet (i.e. A Broadcast).

4.2.9.10Print Statistics

Shows some counters/statistics of the cs8900 chip (i.e. Collision-count, ...)

4.2.9.11Read Rx packets (print,timeout/ms)

Show all received packets, which are receive until the time-out is reached. If you disable printing you get only a message that a packet from a different device is receive, but not the content of the packet.

4.2.9.12Transmit packet (addr,size)

Transmit an eth-packet to the destination address, the sent data is taken from address ADDR and SIZE bytes are sent!

4.2.9.13Toggle leds (cnt) times

Show only the LEDs flushing (of the ethernet-module).

4.2.9.14Set my IP-address

Configures the used IP-address for ping-operations. You have to input 4 bytes in the notation b0.b1.b2.b3: “192.168.222.9”.

4.2.9.15Print my IP-address

Shows the acutal configured IP-address.

4.2.9.16Send ping to ...

Send a pint to the entered IP-address. You have to enter 3 parameters:

- print: 0 = few text-output, 1 = more text-output, 2 = dump all
- ip-addr: ip-address like “Set my IP-address”
- timeout: enter timeout in msecs.

To get the ping working you have to first initialize the ethernet-module. Please run the following steps:

- A) Configure CS
- B) Reset to Eth
- G) Init Rx/Tx (mac)
- N) Configure the IP-address

The ping-target have to be in the same subnet (no gateway or something like that should between the mpc5xx and the other device).

4.2.9.17Wait for ping

Here you can wait for a ping-request, and the mpc5xx will reply.

Send a pint to the entered IP-address. You have to enter 3 parameters:

- print: 0 = few text-output, 1 = more text-output, 2 = dump all
- timeout: enter timeout in msecs.

To get the ping working you have to first initialize the ethernet-module. Please run the following steps:

- A) Configure CS
- B) Reset to Eth
- G) Init Rx/Tx (mac)
- N) Configure the IP-address

The ping-requester have to be in the same subnet (no gateway or something like that should between the mpc5xx and the other device).

4.2.10Serial EEPROM Tools

The serial EEPROM is not available on all boards. If it is not inserted, you get a busy-error message, when reading/writing from/to the eeprom.

```
A) Open Device/SPI
B) Close Device/SPI
C) Status Register Read
D) Set Protection
E) Enable Writing
F) Disable Writing
G) Dump Rom (off,cnt)                                    0 16
H) Write Rom (off,cnt,data...)
I) Erase Rom
.) Quit
```

4.2.10.1Open Device/SPI

Before any operation to the EEPROM you have to open the device (which does the basic configuration for the SPI of the MPC5xx.

4.2.10.2Close Device/SPI

After operation you should close the serial eeprom, to make the resource available for other operations.

4.2.10.3Status Register Read

To get the actual state of the serial eeprom. You get information like busy-state, writing enabled, ... If the serial eeprom is not present, you read always 0xff (which means that the device is busy). Beside the hex-value, you get also a short text, which bits are set.

4.2.10.4Set Protection

You can writeprotect the serial device (parts or complete). You have to enter an integer value to select one of four different protection levels:

Value	*Configuration*
0	No protection (all areas are programmable) – default
1	Last 1/4 is writeprotected (from 0x180..0x1ff)
2	Last 1/2 is writeprotected (from 0x100..0x1ff)
3	Complete device is writeprotected

4.2.10.5Enable Writing

Make EEPROM writeable (could be proved by the Status Register Read – Command).

4.2.10.6Disable Writing

Make EEPROM write-protected (could be proved by the Status Register Read – Command).

4.2.10.7Dump Rom

With this function you can read the actual content of the serial device. You need two parameters, the first is the offset (where to read the eprom), and the second is the count of bytes to read.

4.2.10.8Write Rom

You can write up to 16 bytes to the serial eprom with this command. As an example:

To program 0x11, 0x22, 0x33, 0x44, 0x55 to the serial eprom at location 0x120, you have to enter this command:

```
F 0x120 0x11 0x22 0x33 0x44 0x55 [ENTER]
```

4.2.10.9Erase Rom

This command will write 0 to all locations.

4.2.11PowerPC Tool library

This is only a small function-library to make some basic assembly/disassembly of PPC-code.

```
A) Disassemble (start,cnt)
B) Assemble
C) call (addr)
D) Display Regs
E) Set Reg (reg, val)
F) Dump memory (start,len)
G) Fill Memory (start,cnt,data)
.) Quit
```

4.2.11.1Disassemble

Here you can disassemble code, but it will show you only the „normal“ opcodes, not the „simplified“! You have to give an address and a cnt (count of opcodes). Default would be one opcode at locaion 0.

```
Example: Disasseble Reset-Handler
A 0x100 0x40 [ENTER]
```

4.2.11.2Assemble

Here you can try your powerpc knowledge. Attention this line-assembler does know only the primary-opcodes (not the simplified). Also register-Names are not known (lr, ...). You can quit the line-assembler by typing a dot '.'. If you enter an empty line, the lineassembler use the previous opcode, and advance to the next location.

```
Example: Assemble at location 0x80000
B 0x80000 [ENTER]
Enter empty line to exit!

0x00080000->0x7cf202a6 = mfspr   r7,0x12    # 18 = dsisr
```

4.2.11.3Call (addr)

Here you can try your code, before the code is started, the monitor will switch to the register-configuration, which could be shown with point D. One exception is, that the r1 could not be changed, since this is the system-stack-pointer, and that is required for the Interrupt handling. The called function could return to the monitor by a normal return instruction (bclr 20,0).

ATTENTION: if you overwrite the stack pointer (r1) then an interrupt-handler will write to this new location. If there is an invalid pointer in r1 the system will crash (reboot)!

4.2.11.4Display Regs

Here the register-content which is used for the call, is displayed (only r0..r31, cr, xer, ctr).

4.2.11.5Set Regs

The registers which are used in the call-operation could be configured here. You have to provide the register-name (r00) and the new content.

```
E r03 0x12345678                                ; new value of r03 = 0x12345678
```

4.2.11.6Dump Memory (start, len)

Please see chapter 4.2.4.4.

4.2.11.7Fill Memory (start,cnt,data)

Please see chapter 4.2.4.5.

4.2.12RUN Led - Tools

Functions to switch the RUN-LED on/off and let it flash! The LED is connected to the MIOS-Port on pin 15.

The RUN-LED is only available on EC5xx version 4 and later, and on the EC5xxlight -board!

```
A) Activate LED
B) Deactivate LED
C) Toggle LED (ticks/sec)
.) Quit
```

4.2.12.1Activate LED

Switch the RUN-LED on. The IRQ for LED-flashing is disabled.

4.2.12.2Deactivate LED

Switch the RUN-LED off. The IRQ for LED-flashing is also disabled.

4.2.12.3Toggle LED

The PIT-Timer is used to generate interrupts at a rate to support the flash-rate.

4.2.13Configuration commands

With the configuration commands you can modify the actual settings of the MPC5xx until you reboot.

```
-- Configuration --    {help = ?}
=========================================================================
A) Baudrate                                    9600
B) Watch-Dog Timeout (wd,bm,freeze,NMI)
C) Print Clock
D) Setup Clock
E) Print Alarm
F) Setup Alarm
G) Disable Alarm
H) show UIMB/SGPIO pullup/down state
I) UIMB/SGPIO pullup/down (0=dis, 1=en)        1
.) Quit
```

4.2.13.1Baudrate

Here you can change the baudrate of the ec5xx board. You can request any baudrate even not common values like 8000 baud. The only limitation is that on high-bitrates the accuracy could be not enough. The input clock to the divider is 1.25 MHz. The monitor prints the new selected baudrate and you can check if it's fine enough.

4.2.13.2Watch-Dog Timeout

The mpc5xx has an internal watchdog-timer which could be configured only one time after reset. To enable this feature to the application the EC5xxMon does not access the watchdog/busmonitor feature, but the user could setup/disable this feature with the monitor. As long as the watchdog timer is active it is not allowed to disable the interrupts of the PowerPC, because the decrementer exception is used to trigger the watchdog. Since watchdog and bus-monitor timeout counter are configured with the same register it is only possible to configure both or none.

The command has four parameters:

wd	*:*	*watchdog-timeout in clocks (0..0xFFFF in 1-clock steps, 0xFFFF..0x7FFF800 in 2048 clock-steps)*	
bm	*:*	*bus-monitor timeout (0x0 .. 0x7F8 in 8-clock steps)*	
freeze	*:*	*0*	*= CPU enters debug-mode the watchdog continues counting --> Reset after ~ 3 sec*
		1	*= CPU enters debug-mode the watchdog is stopped.*
NMI	*:*	*0*	*= CPU gets an reset after watchdog timer runs over*
		1	*= CPU gets an NMI after watchdog timer runs over*

Please be aware that the NMI (none maskable interrupt) is critical on the MPC5xx, because this could generate a system hang if you leave the exception-handler (This is because the PowerPC architecture does not support NMI)

Since the decrementer is only started every 10 ms you should not program a watchdog-timeout which is smaller then 2 000 000 clocks.

For the bus-monitor timeout-counter you should not program less then 150 ns, since your external devices require 100 ns access-time.

Configuration example: Watchdog disabled, busmontior-timeout = 400 ns (16 clocks)

```
b 0 16 0 0 [ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.13.3Print Clock

The MPC5xx has an internal counter which could be used as a realtime clock. On the EC5xx board the clock is NOT battery-backuped, so the time is lost after power-down. The clock use the ANSI-time of the „c“ libraries.

4.2.13.4Setup Clock

With this option you can setup time and date of the internal clock, the monitor will ask you for the needed values.

4.2.13.5Print Alarm

In the MPC5xx there is also an alarm support, which could be displayed by the monitor.

4.2.13.6Setup Alarm

Here you can enter an alarm time to the MPC5xx, (this does only print a message to the terminal. This is only to test the alarm-function and is not thought for normal use.

4.2.13.7Disable Alarm

If you have enabled the alarm-timer you can disable it with this command.

4.2.13.8Show UIMB/SGPIO pullup/down state

Here you can check if the internal resistors for pullup/pulldown the UIMB/SGPIO pins in the MPC5xx. This is important if you connect analog signals to the QADC.

4.2.13.9UIMB/SGPIO pullup/down

You can switch on/off the internal resistors of the UIMB-/SGPIO pins. To get accurate values of the QADC you should disable internal pullup/pulldown resistors (0). Where this is applicable you should configure unused pins to output, or put external resistors to unused input pins.

4.2.14Test commands

The MPC5xx has also some support for tests in this version it is only some ram-tests:

```
-- Test Commands --    {help = ?}
=====================================================================
A) Data Bus Test (addr)
B) Addr Bus Test (addr,len)
C) Pattern Test (addr,len)
D) Shift Test (addr,len)
E) External Flash test
F) CAN test
G) Eth test (cs,mac,myip,otherip)        2 000017091964 192.168.222.8 222
H) Serial EE-Prom test (off)             0
I) Test QSC2 – Rx+Tx bridge
J) IO-Test           -testboard
K) TPU-CLK test      -testboard
L) QADC-Test         -testboard
M) QADCA-Show (secs)
N) QADCB-Show (secs)
.) Quit
```

4.2.14.1Data Bus Test

The data-bus test is used to test the data lines to a ram. The test writes different 32-bit patterns to this location (on a device which is smaller then 32-bit, with the data-bus-test also the lowest address lines will be tested: 16-bit --> addressline A30[PPC], 8-bit A30/31[PPC]).

This command requires one address-parameter (which location should be tested), if you do not support a parameter the monitor uses as a default the first SRAM location.

```
A 0x80000000 [ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.14.2Address Bus Test

The address-bus test writes to each location of the address-range the address-information (32-bit), after all locations are written, the MPC5xx will verify all words. The command expects two parameters: starting-address and count of bytes (should be 32-bit aligned). If you do not support a parameter the monitor uses as a default the complete SRAM for testing). Pay attention if you request this test, and the monitor is in SRAM your program will crash!

```
B 0x80000000 0x00100000[ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.14.3Pattern Test

The pattern-test will write to each location in the given address-range the following patterns:

0xFFFFFFFF, 0x00000000, 0x5xx5xx55, 0xAAAAAAAA. Then the monitor will verify if all data is written correctly.

If you do not support an address-range (start and count parameter) the monitor will use the complete SRAM for testing.

```
A 0x80000000 0x00100000[ENTER]
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.14.4Shift Test

This is the slowest test. The monitor will write to each 32-bit location a shifting '1' and a shifting '0' (this means each location is written and verified (2 * 32) times!).

If you do not support an address-range (start and count parameter) the monitor will use the complete SRAM for testing.

```
C 0x80040000 0x00040000[ENTER]     test 256 kbytes
```

It is very important to have a blank between the command and the parameter, after the parameter there is no need for a blank (it's only used for better readability).

4.2.14.5External Flash Test

This test will test the complete external Flash. If the monitor is in the external flash, the code and data will be copied to the ram and the Flash could be tested.

******* ATTENTION *******

If your monitor is working in the external flash, the test should only be done if you have the possibilities to reprogram the monitor with an external tool (debugger, programmer, ...) or you don't need the monitor anymore. If you get an ERROR in this test it could be possible that the monitor is not restored at the test-end – the EC5xx-board will not reboot after a reset.

The EC5xxMon will do three steps for flash-testing:

1 Test chip-erase

2 Test data/addresslines (the monitor write a pattern to complete flash to test the address and data lines).

3 Test sector-erase

After the test the monitor will be copied to the external flash, so the flash is NOT empty after this test.

If the test is done from the INTERNAL monitor, the programmed version will NOT run in external rom, since there are some address-differences.

4.2.14.6CAN-Test

The can-modules are configured, and each can-controller sends some (8) packets to the other. This is done with different baud-rates (10kBit/s,...1MBit/s).

Here is a list of test which are done:

MPC555: TouCanA <-> TouCanB

MPC565: TouCanA <-> TouCanB, TouCanA <-> TouCanC, TouCanB <-> TouCanC

4.2.14.7Eth-Test

The Eth-Test has four major phases. 1. The device is initialized (activate chip-select, reset cs9800a). 2. Verify that there is an CS9800a device available. 3. Register-Test (write/read some registers). 4. Make a ping to another net-device. This function requires 4 parameters:

1 CS Used chips-select (default = 2) (there is no error if CS was already active)

2 MAC Device MAC address (default is only a test-MAC do NOT used for normal operations)

3 MYIP Used IP-address for the ping-request. (format vvv.www.xxx.yyy)

4 Otherip only the last byte of the target ip have to be entered.

Example: CS = 2, MAC = 0123456789ab, myip = 192.168.99.8, otherip = 192.168.99.218:

```
Enter: F 2 0123456789ab 192.168.99.8 218 [CR]
```

4.2.14.8Serial EE-Prom test

The serial eeprom test will try to write 16 bytes to the eeprom, and try to verify the written data. You can select which 16-bytes are tested by enter any start-location up to 0x1f0 (since the serial eeprom has only 0x200 (512) bytes).

The test will program some different patterns to check each bit in the eeprom.

4.2.14.9Test QSC2

This test will send some characters (~10) to the tx-side an expect exactly this data on the rx-side (done in polling mode). You have to connect RX-TX pins directly together for this test.

4.2.14.10IO-Test

For this test there have to be some connections between ports

Attention this test needs to disable some debug signals to succeed. If you use an emulator it could be disconnected after this test!

If the test starts and the debug-pins are enabled (default), you will be asked to disable the ICE-pins. If you answer with Y)es an emulator will get into trouble, but the test should complete without an error. If you answer with N)o the emulator will continue to work, but some pins will fail. If there is no debugger used there is no reason to disable these pins.

4.2.14.10.1IO-Tests on MPC555

List of pin-test which fails: **PIO000, PIO01, PIO02**, these pins are connected to **DASM11, DASM12, DASM13**

Here is a list of connections needed for this test:

```
TPU-A Pin 0    -      TPU-B Pin 0                DASM-28 -      PIO 8
TPU-A Pin 1    -      TPU-B Pin 1                DASM-29 -      PIO 9
TPU-A Pin 2    -      TPU-B Pin 2                DASM-30 -      PIO 10
...            -      ...                        DASM-31 -      PIO 11
TPU-A Pin 15   -      TPU-B Pin 15               PWM-1          -      PIO 13
```

```
DASM-11        -      PIO 0                    PWM-2          -      PIO 14
DASM-12        -      PIO 1                    PWM-3          -      PIO 15
DASM-13        -      PIO 2                    PWM-17         -      SIU-GPIO 06
DASM-14        -      PIO 5                    PWM-18         -      SIU-GPIO 07
DASM-15        -      PIO 6                    SPI-MOSI       -      SPI-MISO
DASM-27        -      PIO 7                    SPI-SCK -      SPI-PCS1
```

Each pin is used for input and output.

```
SPI-PCS0  XOR  SPI-PSC3   =>  SPI-PCS2
```

4.2.14.10.2IO-Tests on MPC565

<TODO>

4.2.14.11TPU-CLK Test

For the TPU-CLK test which uses an external clock-signal as a timebase checks the input-pins. To get a pass, you have to provide the ENCLK signal to this two input-pins. Additional the TPU-A0 and TPU-B0 pin have to be connected together.

For this test the TPU will do a PWM-Signal on one port, and make a puls-width measurement on the other port. If the detected value fits to the configured ENCLK the test pass.

4.2.14.12QADC-Test

This test will temporary deactivate the internal resistors of the MPC5xx. This is needed to get an accurate value from each port.

For the QADC-Test you need two variable resistors connected to the QADC-ports:

Nr.	Poti 1:	Poti 2:
01.	QADCA-AN0	QADCB-AN0
02.	QADCA-AN1	QADCB-AN1
03.	QADCA-AN2	QADCB-AN3
04.	QADCA-AN3	QADCB-AN48
05.	QADCA-AN48	QADCB-AN49
06.	QADCA-AN49	QADCB-AN50
07.	QADCA-AN50	QADCB-AN51
08.	QADCA-AN51	QADCB-AN52
09.	QADCA-AN52	QADCB-AN53
10.	QADCA-AN53	QADCB-AN54
11.	QADCA-AN54	QADCB-AN55
12.	QADCA-AN55	QADCB-AN56
13.	QADCA-AN56	QADCB-AN57
14.	QADCA-AN57	QADCB-AN58
15.	QADCA-AN58	QADCB-AN59
16.	QADCA-AN59	
17.	QADCB-AN2	

This test will request to move the resistors to four different positions (0%, 33%, 66% and 100%), if more then half of the ports have reached that value (with an offset of 5 counts) all ports are checked. All ports which are in range of 10 counts around the expected value, will be signaled OK.

This test will show for each position the following line of text:

Turn Poti (341)-> 0044 -> 15

- The field **(341)** shows the actual value which is requested (decimal, range 0 -> 1023).
- The field **-> 0044 ->** signals that you have to turn the poti to the right (clock-wise) and the best port has an offset of 44 counts.
- The last field **15** tells you that 15 ports are not in range of + - 10.

There could be a problem, when one port is open (not connected to the resistor). Because there could be a cross-talk to the open line. In that case this open port could have a better value than the rest of the good ports. In that case you could have an offset of 0, but only this (bad) port is signaled as OK:

```
Turn Poti ( 341)-> 0000 ->  14
```

If only one port gets OK, please try to move the resistor in both directions, to find another position where you get better results.

At the end of the test you get a summary which ports fails this test, and how many, or you get an OK.

4.2.14.13QADCA-Show

This function will show all 16-portvalues in hex-values (0..0x3ff). If you press only “M [ENTER]” you get the actual values read from each port. You can also use an optional parameter to show the results for some time (in seconds).

Example:

```
M [ENTER]                Show values one time
M 120 [ENTER]            Show values for two minutes
```

4.2.14.14QADCB-Show

Same as QADCA only for port QADCB (Please look at 4.2.13.13).

4.3Special features of the EC5xx

4.3.1MonApplication loader

With the ec5xxmon you have the possibility to start an application with the monitor. You can decide if the application is started automatically or manually. If you configure for an automatic boot, the program will be started after a programmable delay (min 2 secs). While this delay is running you can interrupt the autoboot by sending any character to the board. If you have aborted the autoboot the monitor is called.

Since the autoboot-option will start before the baudrate detection, you will not see any output if you have not configured the terminal to 9600 baud.

4.3.1.1Application configuration

To use an additional application, you need to have a header before the application image. This **must** be aligned to a sector border of the flash, in other cases the monitor will not find the application. Pay attention, since the sector startlocations may vary on different flash-devices!

Application Header content:

```
struct LoadHeader{
       uint32 magich;        /* 0x2a426165 */
       uint32 magicl;        /* 0x726c792a */
       uint32 head_size;     /* 0x30 at the moment */
       uint32 start_delay;   /* any time in 1/10 secs */
       uint32 load_size;     /* bytes */
```

```
        uint32 start_offset;  /* from load-start i.e 0x100 for reset-handler */
        char appl_name[24];   /* MyApp-name */
};
```

Content	*Offset*	*Ends*	*Width*	*example*
MagicH value to recognize header	0x00000000	0x00000003	4 bytes	0x2A426165 * must have this value
MagicL value to recognize header	0x00000004	0x00000007	4 bytes	0x726c792a * must have this value
Header size (in bytes - offset to image)	0x00000008	0x0000000b	4 bytes	0x30 * acutal haeder-size
Startup delay (in 1/10 seconds)	0x0000000c	0x0000000f	4 bytes	40 -> 4 seconds
Load size (in bytes)	0x00000010	0x00000013	4 bytes	0x12345
Start-location (offset to image-start)	0x00000014	0x00000017	4 bytes	0x100
Application-Name	0x00000018	0x0000002f	24 bytes	MyTestApp

Please note that all values have to be written in BIG ENDIAN!

The magic values (at offset 0 + 4) are used to find an application, this must be programmed to the above data (in the example).

The header size is 0x30 (48) bytes.

With the startup-delay you can configure the time in 1/10 seconds to autostart from the monitor.

The loadsize is the image-size from your application (binary-file which you would program to the flash).

Start-Location is the entry-point of your application (offset from image start)

With application name you can give this application a name, if you have multiple applications in the flash you can select one from the monitor (but the autostart will always use the first application!

4.3.1.2Application startup

The monitor will do the following steps when starting an application:

1 unlock all key´d registers

2 copy the image to the ram

3 copy small trailer (at the moment 6 opcodes = 24 bytes) at the end of the image

4 disalbes interrupts (ppc + interrupt controller)

5 trigger watchdog

6 call trailer

7 trailer moves memory-regions (to have application at location 0)

8 trailer calls application

To 7: The trailer will move the memory-regions to new locations:

Internal-memory: 0x01c00000

External-flash: 0xc0000000

External-ram: 0x00000000

(Additional the external ram still is available at location 0x80000000)

The monitor has done the basic configuration of the ec5xx-board (including clock and memory-controller), so it is not necessary (but possible) to reconfigure these values (but if you modify the clock-multiplier you will get a reset!).

4.3.1.3Autostart-mode

The ec5xxmon could start one application automatically (the first one). In this application the delay-value is checked, and if this is different to 0 and 0xffffffff, the application will start without user intervention.

To be able to start the monitor (even an autostart-application is available) the monitor will wait delay-ticks (1/10 secs), until application-start. A minimum of 2 seconds is programmed, to be able to interrupt autostart, but you can program any longer time (until some years).

If you send any key while the autostarter is active the normal monitor is started.

5Appendix

5.1 Binary download - demo code

Binary download is obsolete (only available on older versions of the monitor), please use the xmodem-implementation instead.

Serial upload source:

```
- serial download ----------------------------------------------------------------------------
#include "crc32.h"

typedef struct
{
      uint32 BlockSize;
      uint8  Buffer;

} info;


static DWORD MakeBigEndian( DWORD val)
{
#if BIGENDIAN
       return val;
#else
       uint32 res;

       res  = (val << 24) & 0xFF000000;
       res |= (val <<  8) & 0x00FF0000;
       res |= (val >>  8) & 0x0000FF00;
       res |= (val >> 24) & 0x000000FF;

       return res;
#endif
}

uint32 WorkerFkt( info *Data )
{
       // first transmit size
       tmp = Data->BlockSize;
       tmp = MakeBigEndian( tmp);
       SerialWrite( (char *) &tmp, sizeof( tmp));

       // now transmit data
       for (i=0; i < Data->BlockSize; i ++)
       {
              SerialWrite( Data->Buffer[i]);
              if (Data->Buffer[i] == 0x1b)
                     SerialWrite( 0); // escape -- normal
       }
       if (i != Data->BlockSize)
       {
              // Abort
              tmp = 0x1b1b1b1b; // sending 4 ESC-chars
              SerialWrite( (char *) &tmp, sizeof( tmp));
       }
       else
       {
              // last transmit crc32
```

```
                tmp = CRC32_Calculate( Data->Buffer, Data->BlockSize);
                tmp = CRC32_Result( tmp);
                tmp = MakeBigEndian( tmp);
                cp->BlockWrite( (char *) &tmp, sizeof( tmp));
        }
        return 0;
}
- end of serial download ------------------------------------------------------------------------

- crc.h -----------------------------------------------------------------------------------------
#ifndef INCLUDE_CRC32
#define INCLUDE_CRC32

#include "mytypes.h"

#define CRC_INIT 0xffffffff

#ifdef __cplusplus
extern "C"
{
#endif

uint32 CRC32_Calculate( uint8 *data, uint32 len);
uint32 CRC32_Update( uint32 crc, uint8 *data, uint32 len);
uint32 CRC32_Result( uint32 crc);

#ifdef __cplusplus
}
#endif

#endif

- end of crc.h ----------------------------------------------------------------------------------

- crc.c -----------------------------------------------------------------------------------------
#include "mytypes.h"
#include "crc32.h"

/***************************************************************************
*                                                                          *
* cCrc32Tab                                                                *
*                                                                          *
* the CRC32 table (polynomial 0xedb88320)                                  *
*                                                                          *
***************************************************************************/

static const uint32 CRC32_Table[256] =
{ /* CRC polynomial 0xedb88320 */
        0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL,
        0x076dc419L, 0x706af48fL, 0xe963a535L, 0x9e6495a3L,
        0x0edb8832L, 0x79dcb8a4L, 0xe0d5e91eL, 0x97d2d988L,
        0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L, 0x90bf1d91L,
        0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
        0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L,
        0x136c9856L, 0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL,
        0x14015c4fL, 0x63066cd9L, 0xfa0f3d63L, 0x8d080df5L,
        0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L, 0xa2677172L,
        0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
        0x35b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L,
        0x32d86ce3L, 0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L,
        0x26d930acL, 0x51de003aL, 0xc8d75180L, 0xbfd06116L,
        0x21b4f4b5L, 0x56b3c423L, 0xcfba9599L, 0xb8bda50fL,
        0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
        0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL,
        0x76dc4190L, 0x01db7106L, 0x98d220bcL, 0xefd5102aL,
        0x71b18589L, 0x06b6b51fL, 0x9fbfe4a5L, 0xe8b8d433L,
        0x7807c9a2L, 0x0f00f934L, 0x9609a88eL, 0xe10e9818L,
        0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
        0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL,
        0x6c0695edL, 0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L,
        0x65b0d9c6L, 0x12b7e950L, 0x8bbeb8eaL, 0xfcb9887cL,
        0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L, 0xfbd44c65L,
        0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
        0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL,
        0x4369e96aL, 0x346ed9fcL, 0xad678846L, 0xda60b8d0L,
        0x44042d73L, 0x33031de5L, 0xaa0a4c5fL, 0xdd0d7cc9L,
        0x5005713cL, 0x270241aaL, 0xbe0b1010L, 0xc90c2086L,
        0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
        0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L,
        0x59b33d17L, 0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL,
        0xedb88320L, 0x9abfb3b6L, 0x03b6e20cL, 0x74b1d29aL,
```

```
        0xead54739L, 0x9dd277afL, 0x04db2615L, 0x73dc1683L,
        0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
        0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L,
        0xf00f9344L, 0x8708a3d2L, 0x1e01f268L, 0x6906c2feL,
        0xf762575dL, 0x806567cbL, 0x196c3671L, 0x6e6b06e7L,
        0xfed41b76L, 0x89d32be0L, 0x10da7a5aL, 0x67dd4accL,
        0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
        0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L,
        0xd1bb67f1L, 0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL,
        0xd80d2bdaL, 0xaf0a1b4cL, 0x36034af6L, 0x41047a60L,
        0xdf60efc3L, 0xa867df55L, 0x316e8eefL, 0x4669be79L,
        0xcb61b38cL, 0xbc66831aL, 0x256fd2a0L, 0x5268e236L,
        0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL,
        0xc5ba3bbeL, 0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L,
        0xc2d7ffa7L, 0xb5d0cf31L, 0x2cd99e8bL, 0x5bdeae1dL,
        0x9b64c2b0L, 0xec63f226L, 0x756aa39cL, 0x026d930aL,
        0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
        0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L,
        0x92d28e9bL, 0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L,
        0x86d3d2d4L, 0xf1d4e242L, 0x68ddb3f8L, 0x1fda836eL,
        0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L, 0x18b74777L,
        0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
        0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L,
        0xa00ae278L, 0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L,
        0xa7672661L, 0xd06016f7L, 0x4969474dL, 0x3e6e77dbL,
        0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L, 0x37d83bf0L,
        0xa9bcae53L, 0xdebb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
        0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L,
        0xbad03605L, 0xcdd70693L, 0x54de5729L, 0x23d967bfL,
        0xb3667a2eL, 0xc4614ab8L, 0x5d681b02L, 0x2a6f2b94L,
        0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL, 0x2d02ef8dL
};

/************************************************************************
*                                                                       *
* CalculateCRC32()                                                      *
*                                                                       *
* update the CRC of a memory block                                      *
*                                                                       *
* para:    crc - the previous crc (init to CRC_INIT (0xffffffff))       *
*          buf - pointer to memory block                                *
*          cnt - number of byte in the block                            *
* returns: the updated crc                                              *
*                                                                       *
************************************************************************/
uint32 CRC32_Update( uint32 crc, uint8 *buf, uint32 cnt)
{
        while(cnt--)
        {
                crc = CRC32_Table[*buf++ ^ (uint8)crc] ^ (crc >> 8);
        }
        return crc;
}

/************************************************************************
*                                                                       *
* CalculateCRC32()                                                      *
*                                                                       *
* calculate the CRC of a memory block                                   *
*                                                                       *
* para:    buf - pointer to memory block                                *
*          cnt - number of byte in the block                            *
* returns: the calculated crc                                           *
*                                                                       *
************************************************************************/
uint32 CRC32_Calculate( uint8 *buf, uint32 cnt)
{
        uint32 crc = CRC_INIT;

        return CRC32_Update( crc, buf, cnt);
}

/************************************************************************
*                                                                       *
* result of CRC32()                                                     *
*                                                                       *
* calculate the CRC of a memory block                                   *
*                                                                       *
* para:    buf - pointer to memory block                                *
*          cnt - number of byte in the block                            *
* returns: the calculated crc                                           *
```

```
*                                                                            *
*****************************************************************************/
uint32 CRC32_Result( uint32 crc)
{
        return ~crc;
}

/* end of crc calculation */
- end of crc.c ------------------------------------------------------------------------------------
```