Application Note

Flash Loader for the eZ80 Evaluation Board

AN009402-1101

ZiLOG Worldwide Headquarters • 910 E. Hamilton Avenue • Campbell, CA 95008
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.zilog.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Table of Contents

List of Figures

List of Tables

Abstract

This Application Note offers the complete function codes for a Flash Loader application to run on the eZ80 Webserver Evaluation Board. The goal of this Application Note is to provide a tool that customers can use to program Flash memory with a Boot Loader and/or application software (firmware).

This Flash Loader implementation expects an Intel hexadecimal file (hereafter referred to as HEX) to be an upload file created with ZiLOG Developer Studio (ZDS). No other tools are necessary to convert the output file to another format. A user is able to access all function codes, and can therefore integrate a different method to upload a file to Flash memory.

This Flash Loader implementation only supports 8-MB Micron Technology Flash devices running in 8-bit mode. Both bottom-boot and top-boot Flash device types are supported. Other Micron Technology Flash memory sizes can be added easily by extending *FlashTable*, located in the *Flashld.c* file. Customers using the same Flash device in their own product can use this Flash Loader without any changes unless the startup features (or menu implementation) are not sufficient for the customer.

Other Flash devices are beyond the scope of this Application Note. However, customers can adjust the physical Flash routines located in *Flashsys.c* when a different Flash part is invoked.

General Overview

The Flash Loader application is designed to operate in ZiLOG's ZDS environment. It must be loaded via the ZiLOG Debug Interface (ZDI) into RAM on the eZ80 Webserver Evaluation Board. The Flash Loader can also be loaded into RAM as a resident Flash Loader, which is also called a Boot Loader. This Boot Loader must be physically located in the boot sector of the Flash device. The Boot Loader software starts directly out of Flash after the eZ80 Webserver Evaluation Board is powered on. The physical Flash routines for FLASH LOADER mode and BOOT LOADER mode differ from each other. When running out of Flash memory, physical Flash (Boot Loader) access is only possible when the corresponding Flash routines are copied into RAM and called from this location. The Flash Loader, however, is contained entirely in RAM, and therefore can load from ZDS via ZDI. To ensure correct compilation, the special compiler preprocessor definition *RESIDENT_FLASHLOADER* must be set within the project's compiler settings when the application is configured as a Boot Loader.

The user can access the functions of the Flash Loader via a standard terminal program (hereafter referred to in this document as HyperTerminal, for clarity). When the Flash Loader is first launched, a menu appears allowing the user to select from a choice of serial interface configurations (baud rate, flow control), memory block erase, Boot Loader, or another application program. Loading the Boot Loader program is necessary at initial launch because the Flash device does not yet contain data. The Boot Loader is write-protected and cannot be overwritten. Therefore, it only loads application code.

When compiling this project as a Boot Loader, a simple detection algorithm is implemented to allow the user to force the Boot Loader program to avoid loading the application code. This implementation is required because the application code does not usually support any function that jumps directly to a Boot Loader. To update firmware in Flash memory, press the SPACE bar on the keyboard of the terminal PC, or DTE, before turning on the power to the eZ80 Webserver Evaluation Board (or resetting the board) and wait until the Boot Loader startup menu appears. Essentially, when a SPACE character is detected immediately after power-on or reset, the Boot Loader program remains running. When no SPACE character is detected in the UART receive register, the Boot Loader program starts the application code by jumping directly to the address `0x20000h`, where the application code resides. The user can choose another address when necessary by changing the definition for *APPLICATION_START* in *Target.h*. The user's application code must then be linked from this new address.

Directory and File Structure

This Application Note offers a number of source and Help files. These files are located in the *eZ80FlashLoader* directory. See Figure 1 for this directory structure. The *eZ80FlashLoader* directory contains the *Doc* and *FlashLoader* subdirectories. The *Doc* subdirectory contains Help files and the *FlashLoader* subdirectory contains the source files, project files, and all files created by ZDS (*.hex*, *.ld*, *.sym*, *.map*, etc.). The *FlashLoader* subdirectory also contains a *BootLoad* subdirectory that includes project files and ZDS output files when the Flash Loader operates as a Boot Loader. Furthermore, an 'Include'subdirectoroy has been installed which containss all C header files of that project.

Figure 1. Directory Structure of the Flash Loader Application Note

The *Doc* subdirectory offers the Flash Loader Help files in Windows, WinWord, HTML, and RTF formats. These Help files contain information and definitions for the functions, type definitions, and enumerations mentioned in this Application Note. Figure 2 shows the contents of the *Doc* subdirectory.

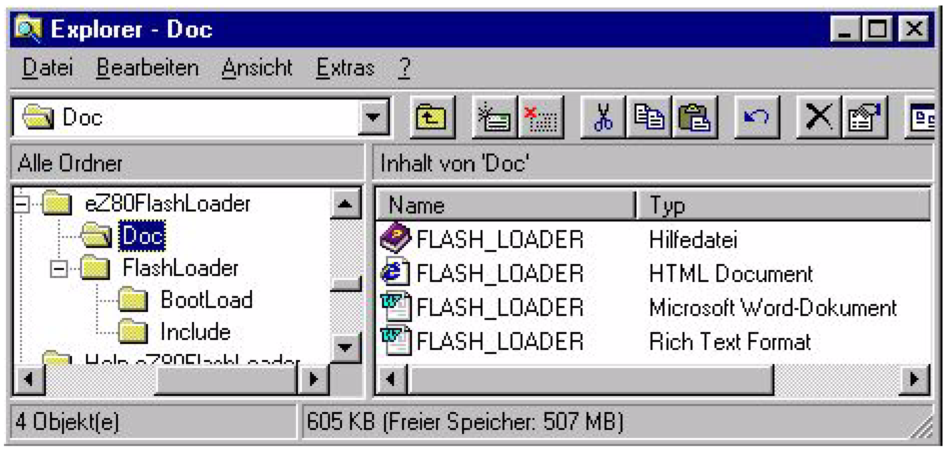

Figure 2. Contents of the *Doc* Subdirectory

The Flash Loader function codes are located in the *FlashLoader* subdirectory. Source fiiles are offered as Csource files (*.c), header files (*.h) and assembly files (*.s). All header files are located in the dedicated ‘Include’ subdirectory. The names of all files are shown in Fisures 3 and 4. To support both eZ80 evaluation boards (Crystal, Realtek) two separated prooject files can be found in the Flash-Loader subdirectory. The files FlashldCS.zws and Flashld.wsp are the corresponding ZDS project files to these source files, incorporating specific settings for the Flash Loader mode when using the Crystal evaluation board. The files FlashldRT.xws and FlashldRT.wsp are the corresponding ZDS project files, incorporating specific settings when using the Realtek evaluation board.

Note: *Target.h* contains target-specific settings that users can change according to the requirements of target hardware. For example, *MCLK* must be adjusted when an oscillator with a clock speed other than 40 MHz is used to receive the correct UART baud rate.

A description of the contents of these source files can be found in the Appendix 1—Function Code Reference section of this document, on page 39.

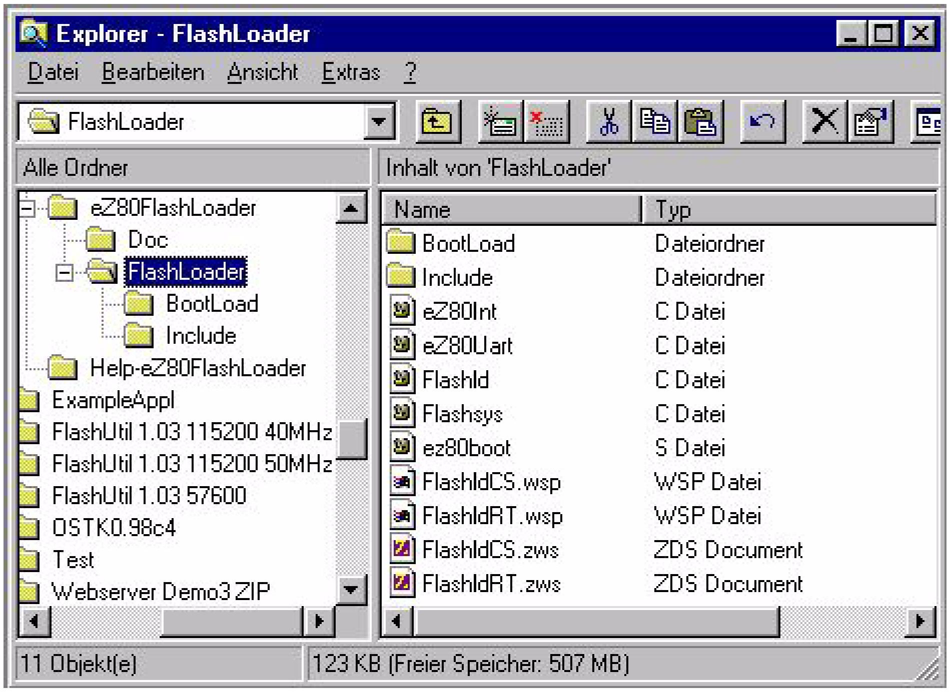

Figure 3. Contents of the *FlashLoader* Subdirectory

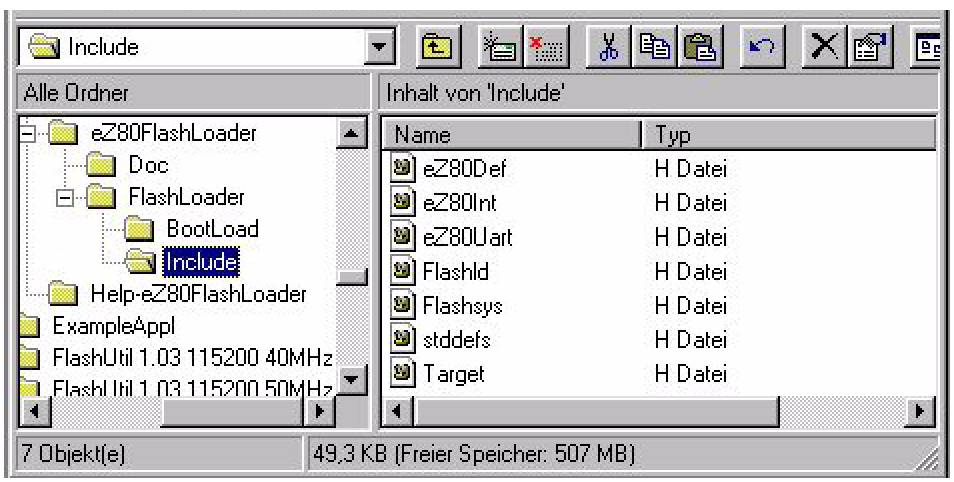

Figure 4. Contents of the Include Subdirectory

The *BootLoad* subdirectory contains ZDS project files that incorporate specific settings for the Boot Loader. While BootLoadCS.zws amd BootLoadCS.wsp contain specific settings for the Crystal evaluation board, the project files Boot-LoadRT.zws and BootLoadRT.wsp are didicated for the Realtek evaluation board.

All output files created by ZDS, as well as compiler and linker files, are located in this subdirectory. Figure 5 shows an example of the HEX files used to upload Boot Loader code into the boot sector of Flash memory. The fileBootLoadCS.hex must be used for the Crystal evaluation board and the BootLoadRT.hex is used for the Realtek evaluation board.

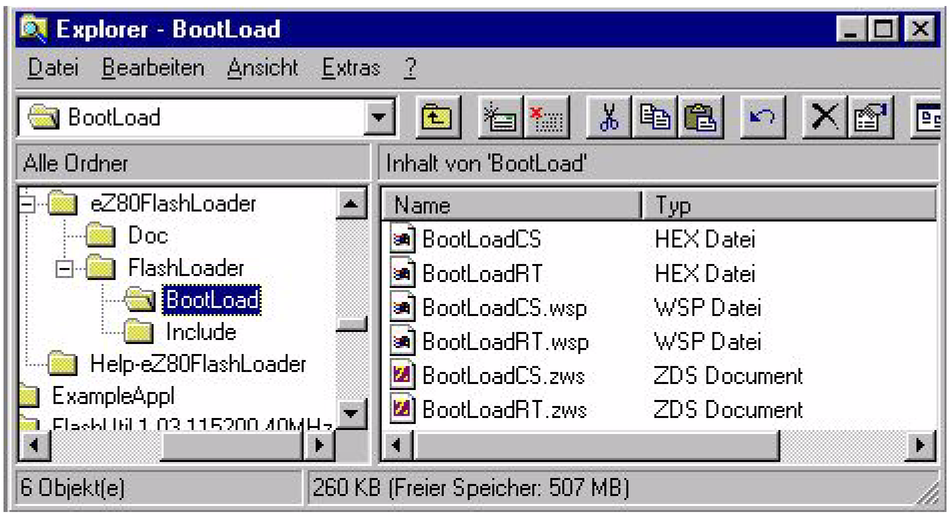

Figure 5. Contents of the BootLoad Subdirectory

Discussion

This section describes the basic operation of the Flash Loader. Detailed information about functions and data structures can be found in the Technical Support section on page 12 or can be requested directly from the Help files located in the *Doc* subdirectory. In the *Doc* subdirectory, all functions are explained with input parameters and return values and can be passed through by the user. These Help files are designed to be a complete reference for the Flash Loader.

Theory of Operation

The Flash Loader can be used for programming a resident Flash Loader (Boot Loader) or the user's application software into the Flash device in different ways.

As long as the user is developing application software within ZDS, the Flash Loader is unnecessary. Application code can be directly uploaded into eZ80 Webserver Evaluation Board's RAM via ZDI. The integrated debugger and the ZDI interface start the application code from an address defined in the **Initial Settings** menu of ZDS.

When the Flash device is empty of data, or a new Boot Loader should be programmed into Flash, the Flash Loader program must be used. The Flash Loader program is launched in a manner similar to any other application that operates within ZDS and ZDI. The Flash Loader is uploaded into RAM and started when the user clicks the RESET+GO button located on the ZDS toolbar. A new Boot Loader or application code can be loaded using ASCI (TEXT) FILE TRANSFER mode on the DTE. The DTE must be connected to the eZ80 UART0 (the default setting). The new software is sent as an Intel HEX file over this link. See the Flash Loader section on page 7 for details.

Using the eZ80 Webserver Evaluation Board as standalone version, the application code must reside in Flash memory. Usually a Boot Loader program is started at power-on. This loader must be located in the boot sector of the Flash device beginning with the address where the eZ80 jumps out of reset. For the eZ80 and the Flash device on the eZ80 Webserver Evaluation Board, the reset vector is `0x0h`. The Boot Loader is implemented to force the Boot Loader program to run its own code or to call the application code and process it. The application code, however, remains unaffected by this call.

The Boot Loader must be recognize the start address of the application code. The address is defined in the Boot Loader code using pertinent parameter blocks. The application code must then be linked from this address.

When this startup condition resides in the Boot Loader code, the user can download or update the application software with the same methodology as the Flash Loader. See the Boot Loader section on page 8 for details.

To download any file to Flash, the user only requires the Intel HEX file that ZDS outputs when the project is built. This resulting HEX file must be copied to the machine running HyperTerminal. To process frames, both the Flash and Boot Loaders run through the following sequence:

- Recognize the Intel HEX file
- Receive a frame
- Check the frame
- Process the frame according to its type
- Calculate the sector address to which data must be stored
- Convert the frame data into binary format
- Program the Flash device

This processing is performed on a frame-by-frame basis. While processing and storing the frame, the Flash Loader and Boot Loader software stop the DTE by issuing the XOFF character (software flow control) or CTS signal (hardware flow control). Only after a frame is successfully processed are the Flash and Boot Loaders ready for the next frame. The DTE transmission is enabled by sending the XON character or releasing the CTS signal.

If the user changes or adds some functions, the Help files offer navigation through all Flash Loader functions and types.

A complete list of all Flash Loader functions can be found in the Appendix 1—Function Code Reference section on page 39.

Flash Loader

The Flash Loader is designed to run from ZDS. The project file is loaded into ZDS and uploaded to the eZ80 RAM located on the eZ80 Webserver Evaluation Board. The program is started by clicking on the RESET/GO button on the tool bar of ZDS. The integrated debugger sets the program counter to the configured address and issues the RUN command.

Using the Flash Loader, the user can program another Boot Loader or update the application code. Usually, the Flash Loader is used to program Flash memory with a resident Flash Loader (Boot Loader). After uploading a Boot Loader, the user can update or reprogram Flash with new application code only by using the Boot Load capability. ZDS is not required to upload new application code.

The Flash Loader works with the following default configurations:

- UART0 (console port P3 on the eZ80 Webserver Evaluation Board) with 57.6KB
- XON/XOFF flow control

To use the Flash Loader, the PC running ZDS must be connected to ZPAK via a serial interface cable. ZPAK must be connected to the ZDI interface of the eZ80. This interface is also located on the eZ80 Webserver Evaluation Board. A second PC (or the same, when more than one serial interface port is available) must be connected to the console port P3 of the eZ80 Webserver Evaluation Board. A standard terminal program, such as HyperTerminal, is required and must be installed on this second PC. HyperTerminal should be configured with a baud rate setting of 57.6KB and the flow control set to XON/XOFF.

After the Flash Loader program is started from ZDS, and assuming all Flash Loader project settings are correct and HyperTerminal is launched, the user should be able to see the Flash Loader startup menu. See the Flash Loader Project Settings section on page 20 for details.

The **Flash Loader** menu lists functions that allow the user to erase only one block of Flash memory, or the entire contents of Flash memory. Additionally, a file download of either Boot Loader or application code can occur.

To download a new Boot Loader, it is not necessary to erase the contents of Flash memory. When selecting *Download Intel HEX Boot Loader File* from the **File** menu, the first four sectors (physical address `0x0h` to `0x1FFFFh`) are erased automatically. The Flash Loader then waits for the Intel HEX file. The user must initiate an ASCI file transfer of the appropriate file. When using a separate PC as a DTE, the user must ensure that the corresponding Intel HEX file is copied onto this PC.

A status indicator in the HyperTerminal window shows the progress of the ASCI file transfer. The user is notified when the Flash Loader completes a successful download. Should a failure occur while the download is in progress, the Flash Loader stops all activities and sends a corresponding error message to the user.

To program the Flash device with a Boot Loader, the WRITE PROTECT feature must be disabled. To disable WRITE PROTECT, open jumper 3 on the eZ80 Webserver Evaluation Board. If the user attempts to write to the boot sector while WRITE PROTECT is enabled, an error message is issued.

When selecting **Download Intel HEX Application File** from the **Flash Loader** menu, a new application code can be downloaded into Flash. All Flash memory sectors from address `0x20000h` (block numbers 4 to 10) are erased automatically before the user is requested to initiate an ASCI file transfer.

Note: The appropriate file must be located on the DTE.

Project settings and configuration parameters for the Flash Loader project can be found in the Technical Support and Test Procedure sections.

Boot Loader

The *BootLoad* subdirectory contains the project file *BootLoad.zws* to generate the resident version of the Flash Loader. Load this project in ZDS and click on the **BUILD** button. The corresponding HEX file can then be found in the *BootLoad* subdirectory. Use this file to download this HEX file to the Flash boot sector.

The major difference between the Flash Loader and Boot Loader projects is the additional preprocessor variable RESIDENT_FLASHLOADER and the linker settings set in ZDS' **Linker** menu. It is possible that the Flash Loader can run out of RAM space, and that the Boot Loader can run out of Flash memory space. Use of Flash commands (*Erase*, *Write*, *GetFlashID*) is limited and therefore cannot be processed while the eZ80 is simultaneously fetching program code from Flash. All physical Flash routines must first be copied into RAM before calling them. This transfer of routines is guaranteed when the preprocessor variable RESIDENT_FLASHLOADER is defined as a project compiler setting.

Additionally, the code and data segments must be linked for different physical addresses. Code segments must be linked to Flash (`0x0h` to `0xFFFFFh`), and data segments must be linked to RAM (`0x200000h` to `0x2FFFFFh`).

Project settings and configuration parameters for the Boot Loader project can be found in the Technical Support and Test Procedure sections.

When the Boot Loader is programmed into Flash, new application code can be written into Flash without ZDS, ZPAK, or ZDI. In this configuration, the eZ80 Webserver Evaluation Board operates as a standalone version.

The Boot Loader incorporates a simple startup algorithm to start the application code or to stay in the own code. When nothing happens after power-on or reset, the Boot Loader starts the application code by jumping directly to the defined application start address `0x20000h`. To update the application code, the Boot Loader waits for a SPACE character after power-on or reset. When a SPACE character is detected, the Boot Loader does not start the application code, but instead continues processing Boot Loader code. Thus, when new application code is written to Flash memory, the user must press the SPACE bar on the DTE, issue a hard reset or power-on, and wait until the Boot Loader detects the SPACE character and responds by displaying the Boot Loader startup menu. Assuming the same HyperTerminal configuration as the Flash Loader (57.6 baud, XON/XOFF), the user should see the menu in the HyperTerminal window.

Within the Boot Loader, only **Download Intel HEX Application File** can be selected from the Boot Loader startup menu. Downloading new Boot Loader code is not permitted here.

When the user must change the default startup algorithm (SPACE bar detection), a second method can be implemented. To force Boot Loader code after power-on or reset, Pin 0 of Port A must be held at a High level. Otherwise, application code starts when PA0 is Low. To enable this method, the preprocessor variable BOOTLOAD_PA0 must be defined under project's compiler setting.

Another Pin can be used by changing the PIN_DOWNLOAD definition in *Target.h* to any other Pin. Also, another Port can be used by changing the START_APPLICATION() definition in *Flashld.h* to any other port. For example, PD0 (port D Pin 0) is used instead of PA0, when changing START_APPLICATION() as follows:

```
#define START_APPLICATION() (PDR(PORTD) & PIN_DOWNLOAD)
```

Overall, managing the Boot Loader is similar to managing the Flash Loader.

User Application Code—Firmware

Application code is not the focus of this Application Note. The user can create code by creating a new project within ZDS. When automatic insertion of the standard boot file is enabled, ZDS inserts the *eZ80boot.s* file to the project file list. Then additional files can be added to form customer's application. There is no special startup sequence required. The only important thing is to link the application code to the appropriate address. This address must be equal to the defined application code start address. In the implementation described here, this address is `0x20000h`. When this address does not meet the user's configuration requirements, the link address can be changed.

Note: When changing the link address, the user must also adjust the Flash Loader or Boot Loader code by redefining APPLICATION_START in *Target.h* to the same address.

When the application code uses a different chip select configuration and/or memory settings, the user must ensure that all corresponding chip select control registers are set accordingly.

Results of Operation

This application code is verified for both FLASH LOADER and BOOT LOADER mode on both the Crystal and the Realtek Evaluation Board. Both C-Compiler versions (1.00 and 1.01) were tested to ensure proper operation. Additionally, software and hardware flow control could be erified and works correctly. When using hardware flow control on the Crystal evaluation board, the modem port P4 (UART1) must be used together with a NULL modem adapter between the DB9 connector and the serial cable because of missing handshake signals on the console port P3. Also, the UART1 must be selected when using modem port P4. The Realtek evaluation board contains full-featured ports (all modem handshaking signals available), and no changes are required when working wit the Realtek board.

Because there was no PC available for testing that could manage baud rates greater than 115KB, selectable baud rates could be verified only up to 115KB. To verify the Flash Loader application, the following test steps were performed:

1. Build the development environment (ZDS, eZ80 Webserver Evaluation Board, ZPAK, second PC).
2. Load the Flash Loader project file *Flashld.zws* or FlashldRT.zws caccording to the used evaluation board. Build and start the Flash Loader.
3. Test all menu items (change baud rate, change flow control, erase block, erase Flash).
4. Download Boot Loader code with software and hardware flow control enabled.
5. Download sample application code with software and hardware flow control enabled.
6. Start the application code from the **Flash Loader** and **Boot Loader** menus.

7. Disconnect ZPAK from the eZ80 Webserver Evaluation Board; reset board to operate as a standalone part.
8. Start the application code after reset or power-on.
9. Press the SPACE bar on the DTE and reset the eZ80 Webserver Evaluation Board. Press the SPACE bar until the Boot Loader is started.
10. Verify all menu items on the **Boot Loader** menu.
11. Download new application code with software and hardware flow control enabled.
12. Start the application code from the **Boot Loader** menu.

The Boot Loader code should be stored in the boot sector of the Flash device. When the Boot Loader is programmed into Flash, WRITE PROTECT should be enabled (Jumper 1 closed) to ensure that the Boot Loader is not overwritten. It is assumed that the size of the Boot Loader code is equal to or smaller than the size of the boot sector. The size of the boot sector of the Micron Technology Flash device is 16KB. However, the compiled version of this Boot Loader is greater than 40KB due to the current version of the linker (the linker includes all library functions independent of its being called from the Boot Loader code or not). The Boot Loader resides in Flash blocks 1, 2, 3, and 4.

Note: Flash block 1 is the boot block.

Because only 16KB of memory can be write-protected, all other parts of the Boot Loader can be overwritten. Because the Boot Loader lies within the first 64KB, it is not possible to use address `0x10000h` as an application start address. This address is located in the middle of the 4th block and must be erased completely before writing new code into the block. As a result of erasing, the part of the Boot Loader that resides in block 4 no longer exists, and operation is discontinued.

To avoid this problem, change the application start address to `0x20000h` to start the application code in block 5. The next release of the linker includes referenced library functions to allow the resulting Boot Loader code to fit into the 16-KB boot block.

Summary

This implementation of Flash and the capability to compile Flash software for different environments using preprocessor variables offers customers a flexible tool for addressing project-specific requirements. The user retains the ability to modify parts of this software. Also, the software architecture allows the user to modify the function codes to support other Flash memory devices when necessary. At this time, however, not all Flash memory types available in the market are supported.

For purposes of this Application Note, the supported Flash devices are limited to Micron Technology Flash types MT28F008XX and MT28F800XX.

Technical Support

Compiler Preprocessor Definitions

There are some preprocessor definitions necessary to compile the Flash Loader for different environments. Much is dependent upon the user's configuration and tools and the version of the compiler. The preprocessor variables listed in Table 1 are defined, and must be set, in ZDS by navigating to **Project Settings** → **Compiler** → **Preprocessor**.

Table 1. Preprocessor Definitions for the Flash Loader

Preprocessor Variable	Description
EZ80EB_REV_C	When working with the eZ80 Webserver Evaluation Board Revision C (Crystal board), this preprocessor variable must be defined because the data bus on the Flash and RAM is reversed.
CC_VER_1_0	Define this variable when working with C-Compiler version 1.00. When working with C-Compiler v1.01 or higher, this variable can be deleted from the Preprocessor list.
READ_BACK_VERIFY	Define this variable to verify the last written sector. In this case, the last sector is read from Flash memory and compared with the raw data still in memory. Delete this variable from the Preprocessor list if it is unnecessary.
RESIDENT_FLASHLOADER	Define this variable to compile the function codes as a Boot Loader. The result is a Boot Loader output file that can be used directly out of Flash. When operating in BOOT LOADER mode, all Flash functions must be copied into RAM before trying to erase a Flash block or write data.
DEFAULT_BAUD_115200	Define this variable to change the default baud rate to 115.2 kbps. If this variable is undefined, the default baud rate is 57.6 kbps. This preprocessor definition takes effect only when the RESIDENT_FLASHLOADER variable is also defined.

Table 1. Preprocessor Definitions for the Flash Loader (Continued)

Preprocessor Variable	Description
USE_UART_1	Define this variable to use UART1 of the eZ80190. This port is directed to the modoem port (P4) on the eZ80 Webserver evaluation boards (Crystal and Realtek). When using the Crystal evaluation board with enabled hardware flow control, only the modem port (P4) can be used. For that purpose, the UART1 must be selected and a NULL modem adapter must be inserted between the corresponding DIP9 connector and the serial cable to the DTE.
BOOTLOAD_PA0	Define this variable to use Port A Pin 0 as a signal to force the software to remain in BOOT LOADER mode. When this signal is active High at startup, the application software does not start, and access to the Boot Loader is granted to upload new application code. When this signal is active Low, the application code is started immediately after turning on power to the device or immediately after reset. If this variable remains undefined, the Flash Loader defaults to waiting for a SPACE character to arrive from the serial interface to force Boot Loader code.

Flash Loader Preprocessor Requirements

Use the following standard preprocessor settings to configure the Flash Loader.

- UART0 → Console Port P3 on the eZ80 Webserver Evaluation board
- 57.6 kbps default baud rate
- eZ80 Webserver Evaluation Board Revision C–Crystal board
- C-Compiler v1.01
- Verify written sectors

The Flash Loader project must be compiled with the following preprocessor definition by navigating to **Project Settings** → **Compiler** → **Preprocessor**:

```
EZ80EB_REV_C; READ_BACK_VERIFY
```

Additional preprocessor definitions can be inserted. For instance, `CC_VER_1_0` activates C-Compiler v1.00 or DEFAULT_BAUD_115200 when the default baud rate should be 115.2 kbps instead of 57.6 kbps.

Note: When using the Realtek board, which is part of the eZ80 Webserver Developer’s Kit, the user is instructed to delete the preprocessor variable

EZ80EB_REV_C in ZDS by navigating to **Project Settings** → **Compiler** → **Preprocessor**.

Boot Loader Preprocessor Requirements

Use the following standard preprocessor definitions to configure the Boot Loader.

- UART0 → Console Port P3 on the eZ80 Webserver Evaluation Board
- 57.6kbps default baud rate
- eZ80 Webserver Evaluation Board Revision C–Crystal board
- C-Compiler v1.01
- SPACE bar detection for Boot Loader activation
- Verify written sectors

The Boot Loader project must be compiled with the following preprocessor definitions by navigating to **Project Settings** → **Compiler** → **Preprocessor**:

```
EZ80EB_REV_C; READ_BACK_VERIFY; RESIDENT_FLASHLOADER
```

Additional preprocessor definitions can be inserted. For example, `CC_VER_1_0` activates C-Compiler v1.00 or DEFAULT_BAUD_115200 when the default baud rate should be 115.2kbps instead of 57.6kbps.

Note: When using the Realtek board, which is part of the eZ80 Websrver Developer's Kit, the user is instructed to delete the proprocessor variable EZ80EB_REV_C in ZDS by navigating to **Project Settings** → **Compiler** → **Preprocessor**.

Memory Usage

This section describes all address locations used in the Flash Loader and Boot Loader software. This information is required to modify or add functions.

Caution: Care must be taken when modifying addresses. Modification of any addresses within the project settings or the function codes can cause unpredictable errors.

The Flash Loader and the Boot Loader take advantage of the interrupt capability of the eZ80 on-chip UART. The Receive interrupt is enabled to store the serial data in a buffer. To setup the eZ80 interrupt table, internal RAM space is used to store all interrupt vectors and the corresponding preinterrupt handlers. The file *Target.h* defines the interrupt vector table address IVECTOR_TABLE as `0xFC00h`. Due to the 64-KB limitation of the interrupt vector table, the memory page must be set to 0 when enabling internal RAM. See RAMCTRL0 and RAMCTRL1 of **Symbol Definitions → Linker Settings**.

Because the interrupt vector table and preinterrupt routines allocate the RAM space to the address range `0xFC00h–0xFFFFh`, the short stack pointer SPS must be set to `0xFBFFh`. This location must be confirmed in the initial settings and in the corresponding symbol definition of the linker setting. These project settings are listed in the Test Procedure section beginning on page 17.

Both eZ80 Webserver evaluation boards, the Crystal and the Realtek board feature 1MB of Flash memory (CS0) and 1MB of RAM (CS1: 2x512KB and CS1/CS2 for Crystal board, CS1 only for the Realtek board), memory usage is defined as follows:

Flash Loader

- SPS = `0xFBFFh` (internal RAM Page 0)
- SPL = `0x2FFFFFh` (external RAM)
- IVECTOR_TABLE = `0xFC00h` (internal RAM Page 0)
- ROM = RAM = `0x200000h`–`0x2FFFFFh` (external RAM)

Boot Loader

- SPS = `0xFBFFh` (internal RAM Page 0)
- SPL = `0x2FFFFFh` (external RAM)
- IVECTOR_TABLE = `0xFC00h` (internal RAM Page 0)
- ROM = `0x000000h`—`0x0FFFFFh` (external Flash)
- RAM = `0x200000h`—`0x2FFFFFh` (external RAM)
- RAM Function Start Address = `0x280000h` (external RAM)

The RAM function start address defines the address location that the Flash routines must be copied to when BOOT LOADER mode is invoked. Flash routines are called by pointers.

Function Codes

The complete Flash Loader application is written in C language, except for the startup assembly file *eZ80boot.s*. This assembly file is automatically inserted by ZDS when creating a new eZ80 project.

The following files contain complete function codes and can be viewed or changed after extracting the application onto a local computer. The archive file *eZ80Flashloader.zip* contains the entire application, including all source, project, and Help files.

Following is a list of all source files and a short description of their contents.

C Files

- `flashld.c`—contains all functions related to the Flash Loader application
- `flashsys.c`—contains all Flash device-specific functions
- `ez80int.c`—contains all Interrupt-related functions used for the Flash Loader
- `ez80uart.c`—contains all UART-related functions of the eZ80190

Header Files

- `flashld.h`—the corresponding header file for *Flashld.c*
- `flashsys.h`—the corresponding header file for *Flashsys.c*
- `ez80int.h`—the corresponding header file for *ez80int.c*
- `ez80uart.h`—the corresponding header file for *ez80uart.c*
- `ez80def.h`—contains all eZ80 register and bit definitions used in the Flash Loader application
- `stddefs.h`—contains the global type definition for this application
- `target.h`—contains target-specific user-adjustable definitions

Assembly Files

`ez80boot.s`—startup boot file

Project Files

- `FlashldCs.zws`—XDS project file for the FLASH LOADER mode (Crystal)
- `FlashldCS.wsp`—ZDS Work Space file for the FlashldCS.zws project
- `FlashldRT.zws`—ZDS project file for the FLASH LOADER mode (Realtek)
- `FlshldRT.wsp`—ZDS Work Space file for the FlashldRT.zws project
- `BootldCS.zws`—ZDS project file for the BOOT LOADER mode (Crystal)
- `BootldCS.wsp`—ZDS Work Space file for the FlashldCS.zws project
- `BootldRT.zws`—ZDS project file for the FLASH LOADER mode (Realtek)
- `BootldRT.wsp`—ZDS Work Space file for the FlashldRT.zws project

When opening a project file, the project file list window within ZDS must display all source files correctly, reflecting full path names. If such is not the case, the project file list can be refreshed with the following instruction:

1. Delete all files from the project file list.
2. Add all four C files and the *ez80boot.s* file to the project file list by navigating to **Project** → **Add to Project** → **Files**.
3. Save the project.

Test Procedure

Equipment Used

- PC/Laptop running Windows 95/98/NT
- Second PC/Laptop with the HyperTerminal application installed. HyperTerminal's default settings should be:
  - 57.6kbps
  - XON/XOFF software flow control
  - ASCI (text) file transfer capability

Note: A second PC is required when the host PC running ZDS offers only one COM interface.

- eZ80 C-Compiler v1.00 or higher (the latest version should be used)
- ZDS v3.65 or higher
- eZ80 Webserver Evaluation Board Revision C or higher
- ZPAK
- Power Supply for the eZ80 Webserver Evaluation Board and for ZPAK
- Serial Interface cable to connect ZPAK with the PC/Laptop
- Serial Interface cable to connect a Terminal with the eZ80
- NULL modem adapter when using the modem port on the eZ80 Webserver Evaluation Board

Note: The NULL modem adapter is required only when the modem port (P4/UART1) is used to connect the eZ80 Webserver Evaluation Board with the DTE. The NULL modem adapter must swap DTR/DSR, RTS/CTS, and RX/TX.

- Flash Loader Function Codes (project files)
- Appropriate Boot Loader in Intel HEX format

- User's application code in Intel HEX format linked from address `0x20000h`
- ZiLOG Debug Interface (ZDI)

General Test Setup and Execution

Jumper Settings – Crystal Board

Ensure that the jumpers on the eZ80 Webserver Evaluation Board Revision C reflect the following settings:

J1 Open: Flash Write Protect Disabled

Open: Flash Write Protect Disabled

J2 Closed: use CS2 for RAM2

J3 1–2: use a 40-MHz oscillator

J7 1–2: 1-MB Flash size

J8 Closed: Flash from `0x0h`–`0xFFFFFh`

Jumper Settings – Realtek Board

Ensure that the jumpers on the eZ80 Webserver evaluation board with Realtek EMAC reflects the following settings:

J1 1-2: use a 40Mhz oscillator

J2 Open: Flash Write Protect Disabled

J3 Closed: Test Pin High (Test function disabled)

J5 2-3:EEPRM CS is EECS from Realtek

J7 1-2: Reset Select ($\overline{\text{PRESET}}$ signal)

J10 2-3: Ring signal Select (use R11_NB signal)

J11 Closed: IOCS16 enabled

Installation

The eZ80 C-Compiler (v1.00, v1.01 or greater) and ZDS (3.66 or greater) must be installed. The latest version of ZDS can be downloaded from ZiLOG.

After successful installation of the C-Compiler and ZDS, install the eZ80 Flash Loader application. After unzipping the *eZ80FlashLoader.zip* file, the same directory and file structure that is documented in the Directory and File Structure section should be available. See page 2.

Prepare a second PC (or the same when more than one serial interface is available) to act as a terminal device, or DTE. Open the HyperTerminal program and configure the serial interface for baud rate of 57.6KB, a polarity of 8N1, and for software flow control. Use a standard serial cable to connect the terminal device with the Console Port P3 of the eZ80 Webserver Evaluation Board. Connect ZPAK with ZDI using the small blue cable provided. Use another standard serial cable to connect ZPAK with ZDS (the PC with ZDS, the C-Compiler, and the eZ80 Flash Loader Application installed). Power-on both ZPAK and the eZ80 Web-server Evaluation Board.

Load the Flash Loader project into ZDS by double-clicking either the *FlashldCS.zws* or *FlsahldRT.zws* (according to the used evaluation board) file in Windows Explorer. Assuming that the ZDS installation is completed and correctly registered, ZDS can be launched. Because the Flash Loader is running on another machine, ZDS recognizes that this project has moved to another location. At this point, only the compiler *include* and *library* paths in the project settings must be modified to the local configuration of the eZ80 C-Compiler. When complete, the project should be compiled and linked without any errors. However, the user should verify all settings before compiling and linking. The Flash Loader Project Settings section on page 20 lists the default settings for the Flash Loader project.

After a successful build process, press the RESET+GO button to load the Flash Loader code into RAM and launch it. The startup menu is displayed within the terminal window. Try to erase some blocks, or Flash, to be sure that the software is working.

Proceed with the Boot Loader project (*BootLoad.zws* or BootloadRT.zws file according to the used evaluation board) in the same manner. Build the project and copy the *resulting BootLoadXX.*hex file to the terminal PC. See the Boot Loader Project Settings section on page 32 for all necessary settings.

The user can again start the Flash Loader and upload the Boot Loader code by initiating an ASCI file transfer of *BootLoadXX.hex*.

Note: The WRITE PROTECT jumper must be released.

Verify the Boot Loader in the following way:

1. Disconnect ZDS and ZDI from the eZ80 Webserver Evaluation Board.
2. Press the SPACE bar on the DTE while resetting the board.
3. When the Boot Loader code upload is successful, the Boot Loader startup menu is displayed on the DTE.

The user's application code can be programmed into Flash by selecting **Download Intel HEX Application File** from the **Boot Loader** menu. When completed, reset the board to start the application.

Flash Loader Project Settings

Access the Flash Loader Project Settings in ZDS by navigating to **Project** → **Target** → **Initial Settings**. In the **Initial Settings** dialog, the program start address, stack pointer, and chip select configurations can be defined. ZDS sets the values in the corresponding eZ80 control register via ZDI before program download and start.

Figures 6 through 9 show the necessary settings for FLASH LOADER mode. The code is compiled for a RAM environment accessible via Chip Select settings CS1 and CS2. Because the Crystal Board and the Realtek board use a different address decoding scheme, attention must be paid when setting CS1 and CS2. For the Crystal board, CS1 is connected to the first 512KB RAM and CS2 is connected to the second 512KB RAM.

For the Realtek board only CS1 is connected to the RAM. Therefore, the address range for CS1 on the Realtek board covers the entire range of 1MB (`0x200000h` to `0x2FFFFFh`).

On the Crystal board, CS1/CS2 covers only 512KB each which results in 1MB RAM (CS1:`0x200000h` to `0x27FFFFh`; CS2: `0x280000h` to `0X2FFFFh`). Except for internal RAM (`0xE000h`–`0xFFFFh`), no other memory locations are used in FLASH LOADER mode.

Note: Settings which are different for the Crystal and Realtek boards are illustrated separately in corresponding figures.

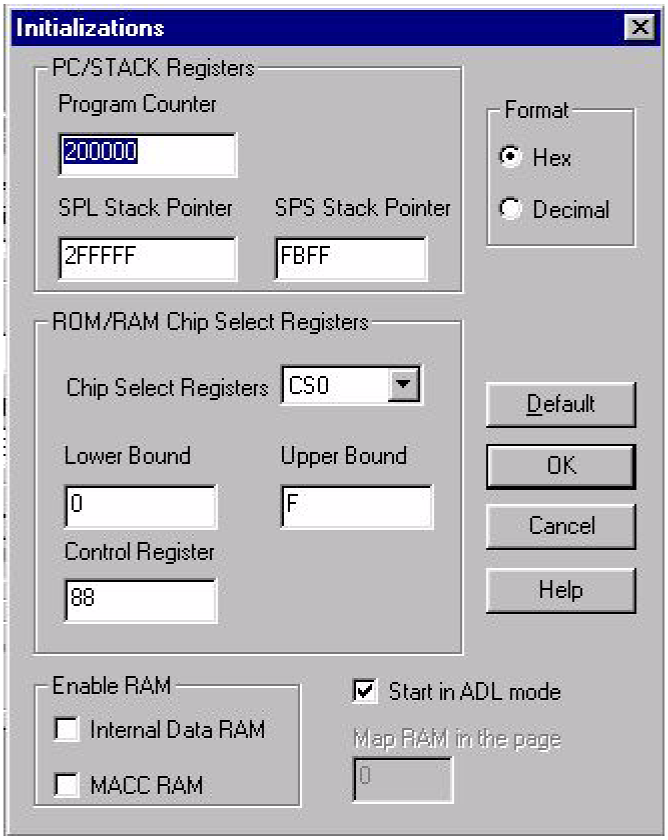

Figure 6. Initializations Screen—Initial Settings for CS0

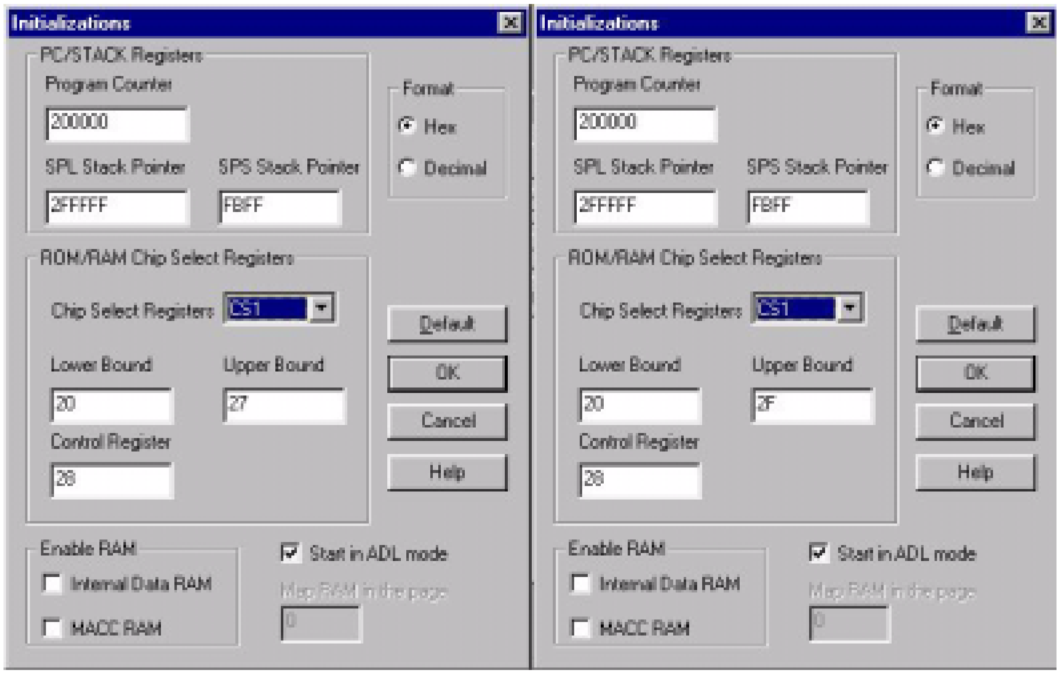

Figure 7. Project Target Menu—Initial Settings for CS1 (Left, Crystal Board; Right: Realtek Board)

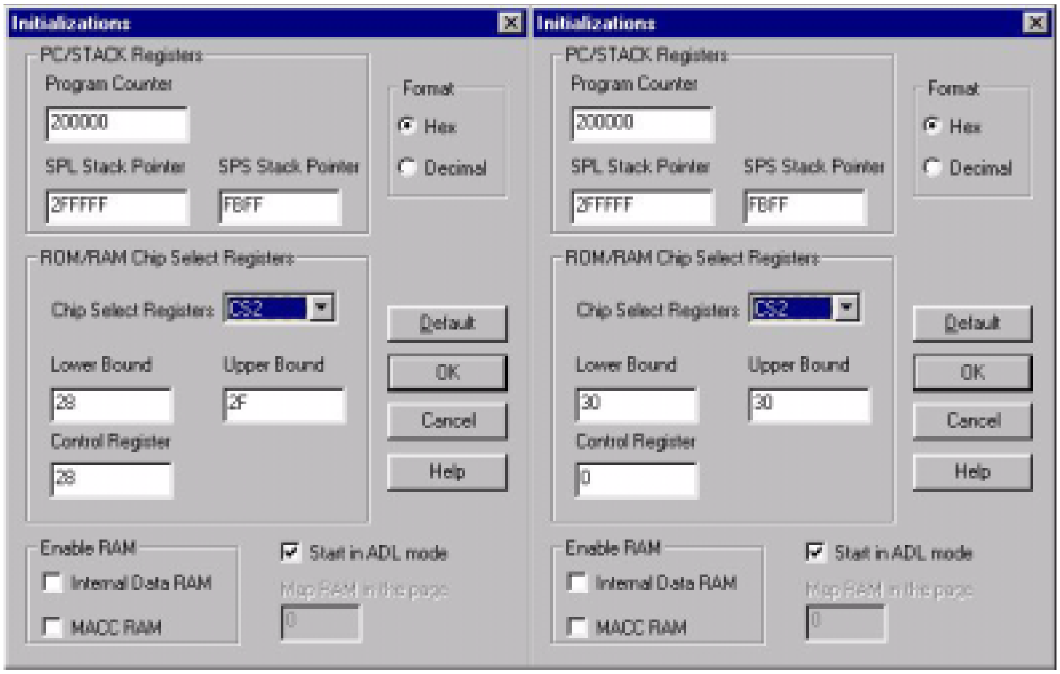

Figure 8. Project Target Menu—Initial Settings for both Realtek and Crystal Board

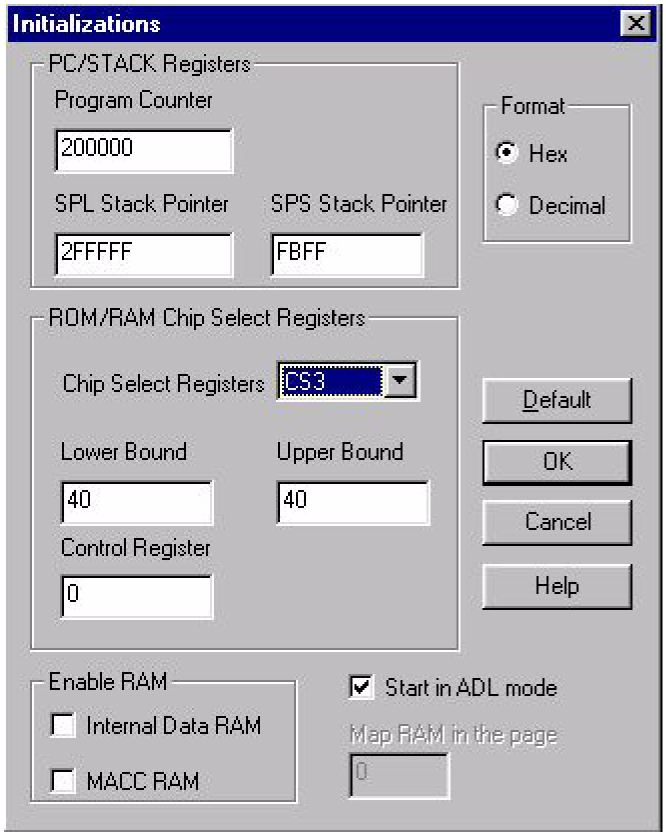

Figure 9. Project Target Menu—Initial Settings for CS3 for both Crystal and Realtek Boards

Figure 10 shows the **Compiler** settings of the **Settings Options** dialog box. With **Preprocessor** chosen as the category, all preprocessor variables and the C-Compiler include path must be set. When loading the Flash Loader project the first time, the include path must be modified to the local compiler path.

Note: ZDS version 3.65 and higher allows the user to set different values for each file in the project list. The project file must be selected to guarantee that these values are valid for each file.

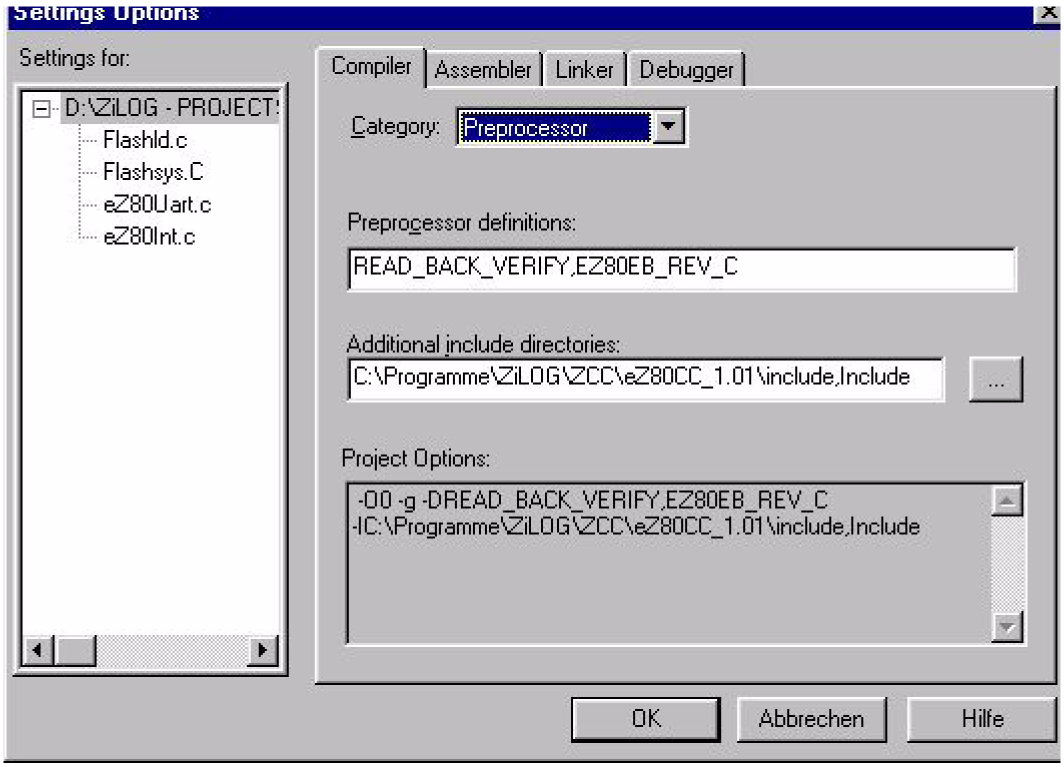

Figure 10. Project Settings Menu—Compiler Preprocessor

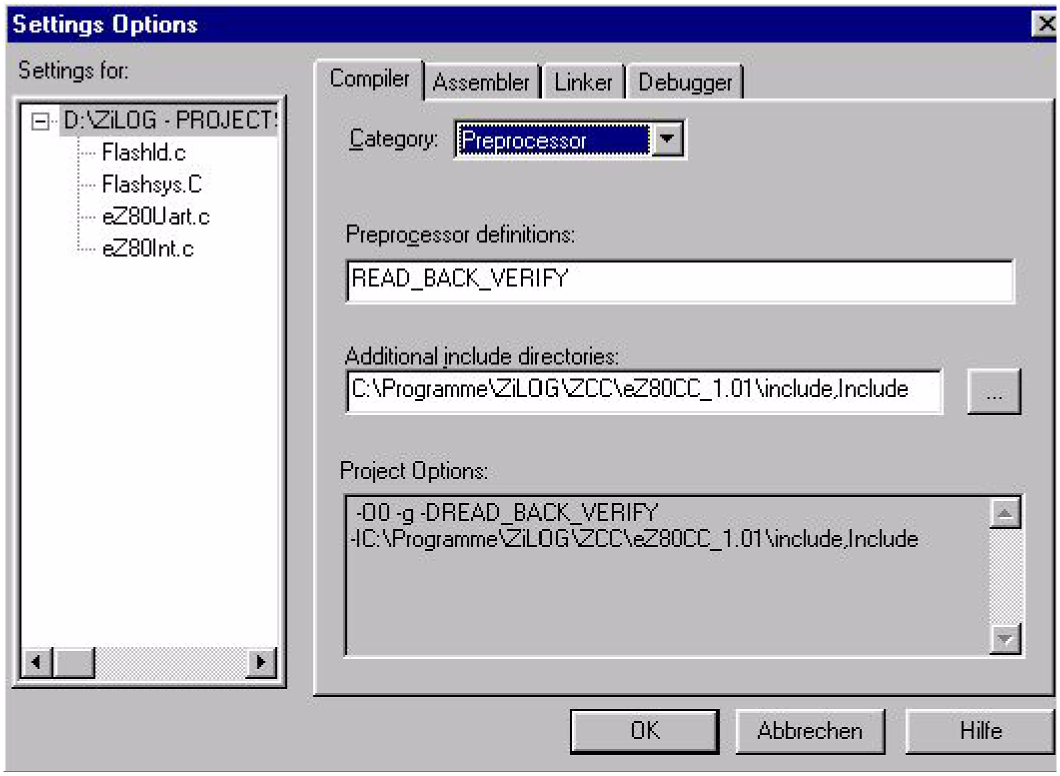

Figure 11. Project Settings Menu - Compiler Preprocessor for the Realtek Board

To change some loader configurations, the user can add further preprocessor definition in the editor window. For example, DEFAULT_BAUD_115200 sets the default baud rate to 115KB.

The memory map linker settings for the Flash Loader are shown in Figure 12. Most important are the ROM start and end addresses fields, as these addresses define the location of external RAM.

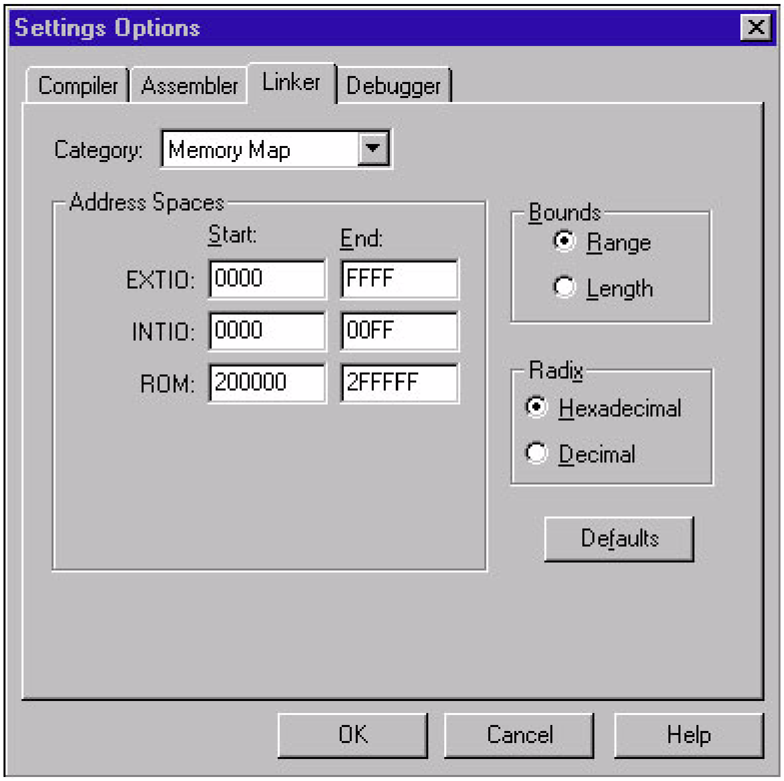

Figure 12. Project Settings Menu—Linker Memory Map

Figures 13 and 14 show the symbol definitions required to build the Flash Loader project. The .CSXLB, .CSXUB, .CSXCR, and .RAMCTLX symbols must be set according to the initial CS settings (see Figure 6 on page 21). These symbols are used within the *ez80boot.s* startup file to configure all chip select and RAM control registers first before any other instruction is processed. For better understanding, symbol names are chosen according to the eZ80 on-chip register names. For details, refer to the eZ80 CPU User's Manual.

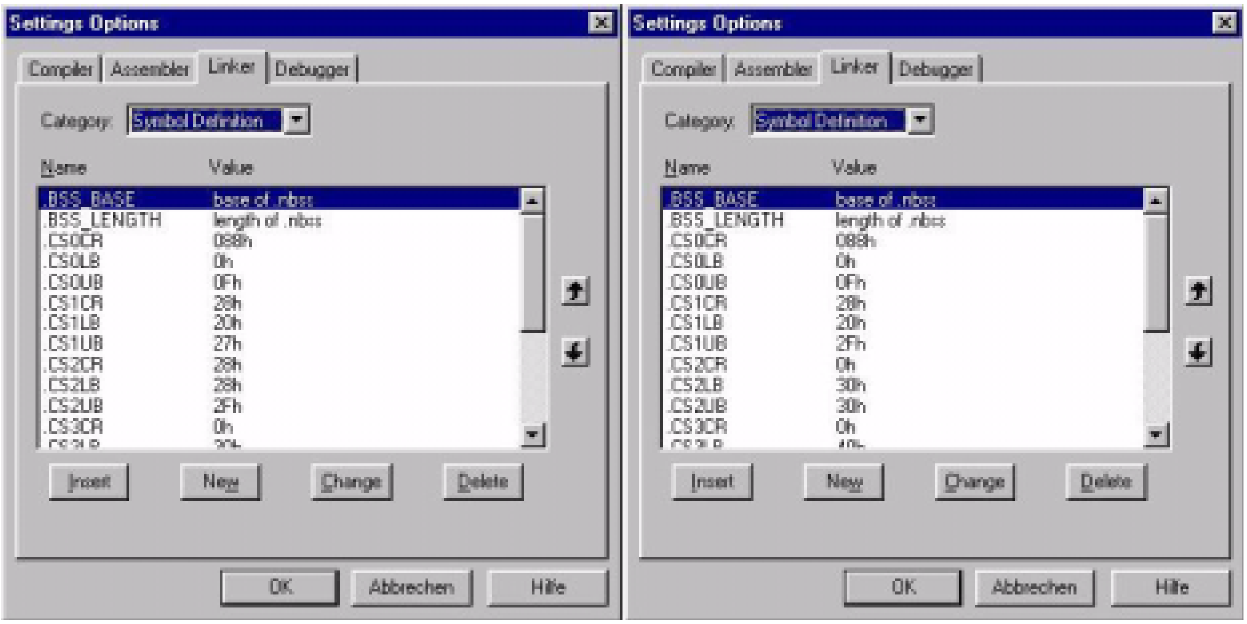

Figure 13. Project Settings Menu—Linker Symbol Definition, (Left: Crystal Board; Right: Realtek Board) 1 of 2

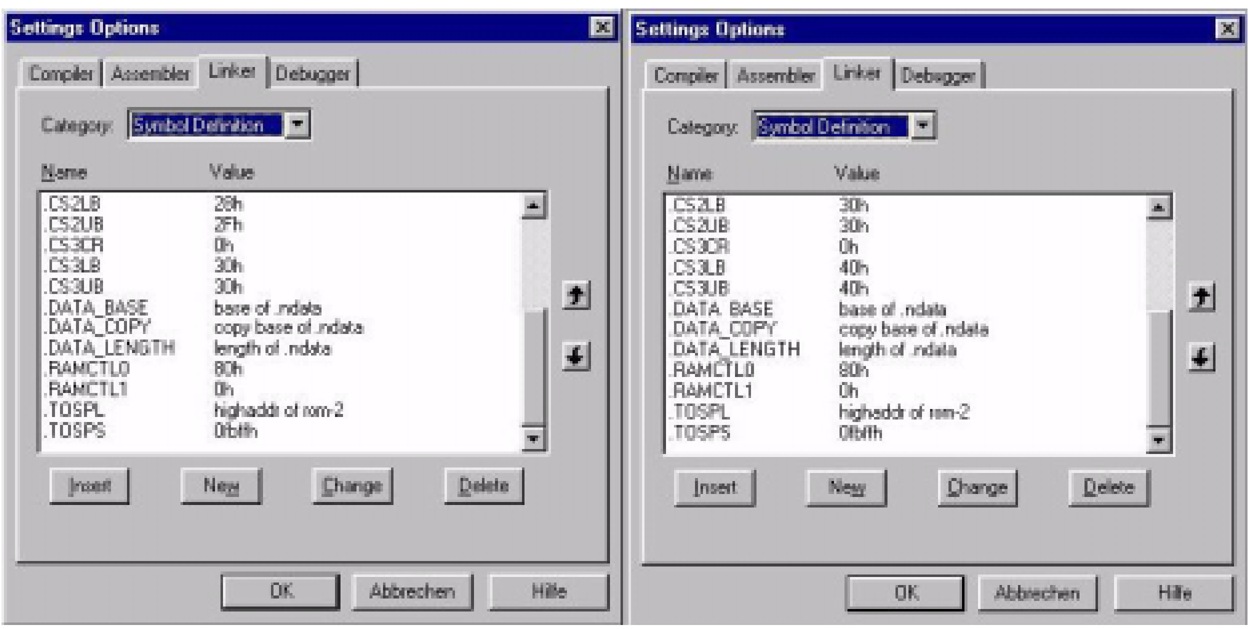

Figure 14. Project Settings Menu—Linker Symbol Definition, (Left: Crystal Board; Right: Realtek Board) 2 of 2

The ordering sequence must be set as indicated in Figure 15. Only the *.startup* segment is listed to ensure that the *ez80boot.s* file is linked first.

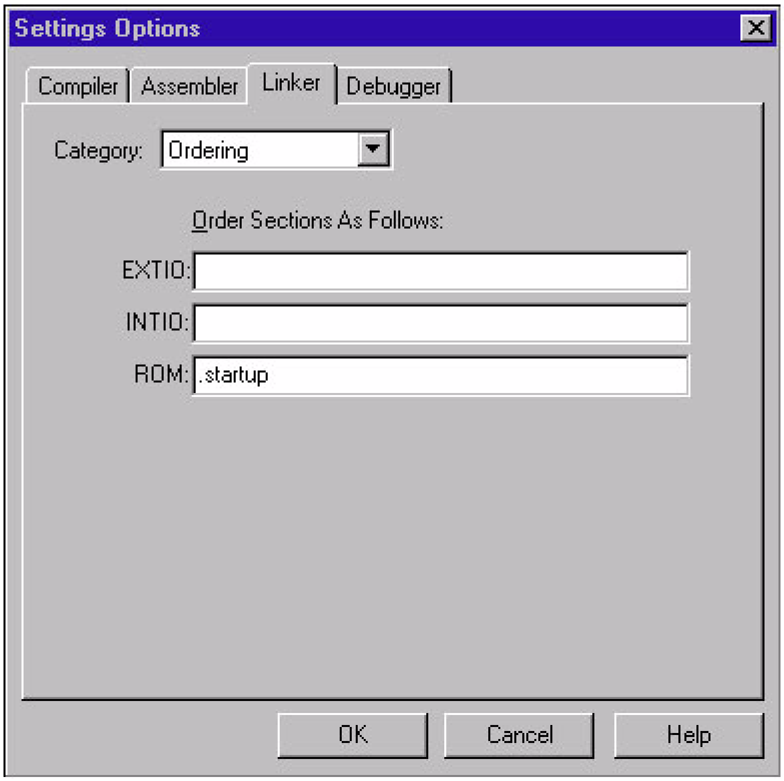

Figure 15. Project Settings Menu—Linker Ordering

Be sure that both the *.const* and *.startup* segments are assigned to ROM address space as indicated in Figure 16.

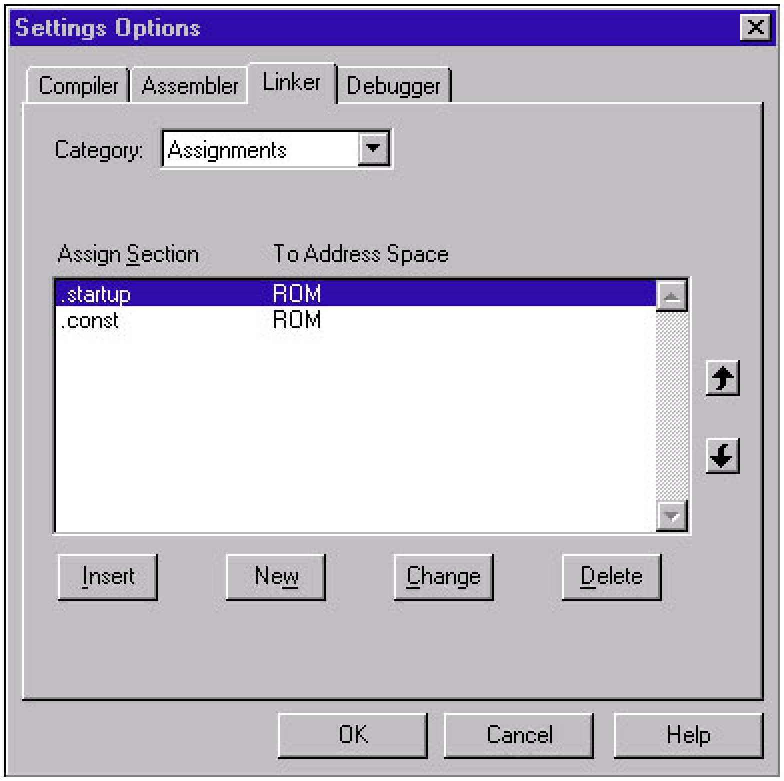

Figure 16. Project Settings Menu—Linker Assignments

The compiler must create the initialized data section *.ndata* in ROM. In the *ez80boot.s* startup file, this section is copied into the assigned RAM location. For this purpose, *.ndata* must be defined as copy sections located in ROM. Figure 17 shows this setting.

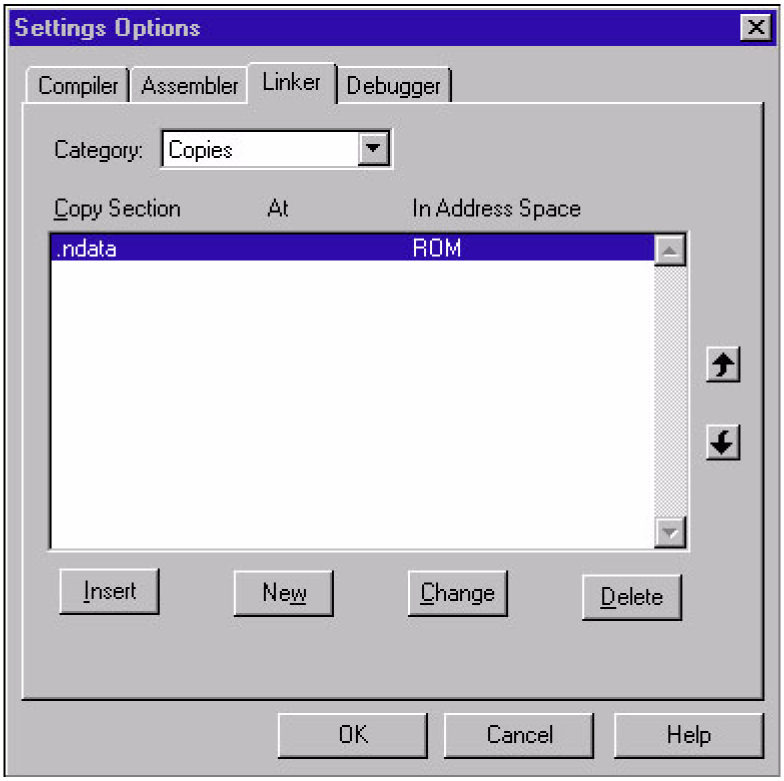

Figure 17. Project Settings Menu—Linker Copies

Figure 18 shows the general linker settings. The library path must be modified according to the local compiler path. The project cannot be built when the path is not correct.

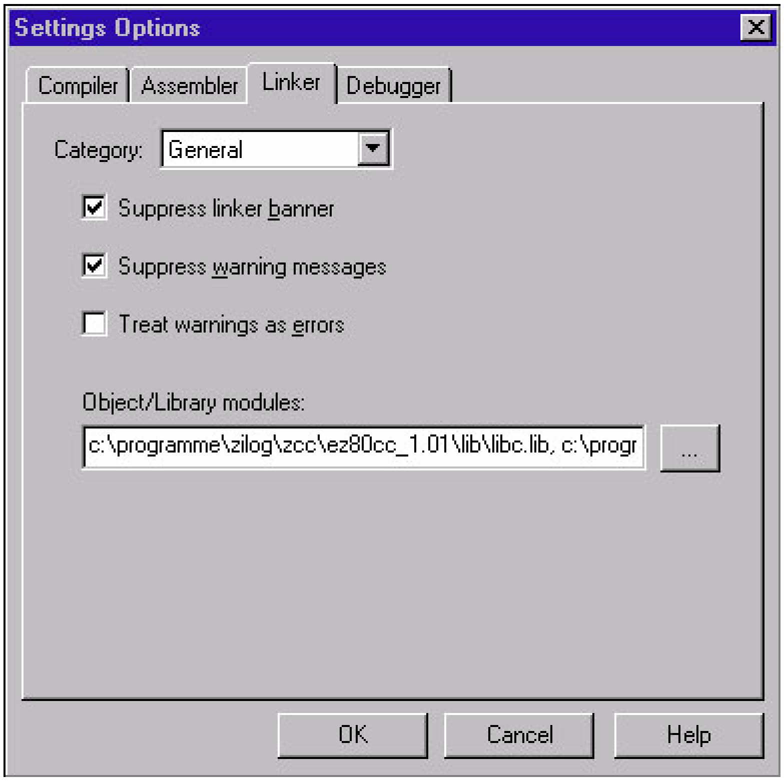

Figure 18. Project Settings Menu—Linker General

Boot Loader Project Settings

When the Flash Loader function codes are compiled with the preprocessor value RESIDENT_FLASHLOADER set, the result is Boot Loader code that can be programmed into the Flash boot sector. The resulting project file *BootLoadCS.zws* and *BootLoadRT.zws* in the *BootLoad* subdirectory includes a number of address settings. Flash memory can be accessed via CS0 in the range `0x000000h`–`0xFFFFFh`. The external RAM configuration is similar to FLASH LOADER mode (`0x200000h`–`0x2FFFFFh`) and is accessible via CS1 and CS2 in *BootLoad.zws* (Crystal board) or CS1 only in *BootLoadRT.zws* (Realtek board).

The initial settings do not affect the Boot Loader project because it is designed to be compiled and linked for the target hardware. Therefore, these settings are not listed here. Some settings are the same as those in the Flash Loader project. The general linker setting, the assignments, the copy sections, the ordering sequence, and all symbol definitions are the same. These default settings are listed in the Flash Loader Project Settings section on page 20. Only the settings that differ from the default settings are described here.

Note: The general linker setting requires a modification of the library pathname.

For BOOT LOADER mode, the additional preprocessor definition RESIDENT_FLASHLOADER must be defined as indicated in Figure 19.and Figure 21. Additionally, the compiler include path must also be change according to the local compiler installation path. A second include path '..\Include' must defined to find the header files of the Flash Loader project when starting the build process. This action is necessary because the Boot Loader project file is located in the *BootLoad* subdirectory, but all source files are still in the *Flash Loader/ Include* directory. If these settings are valid for all files, the project file can be selected.

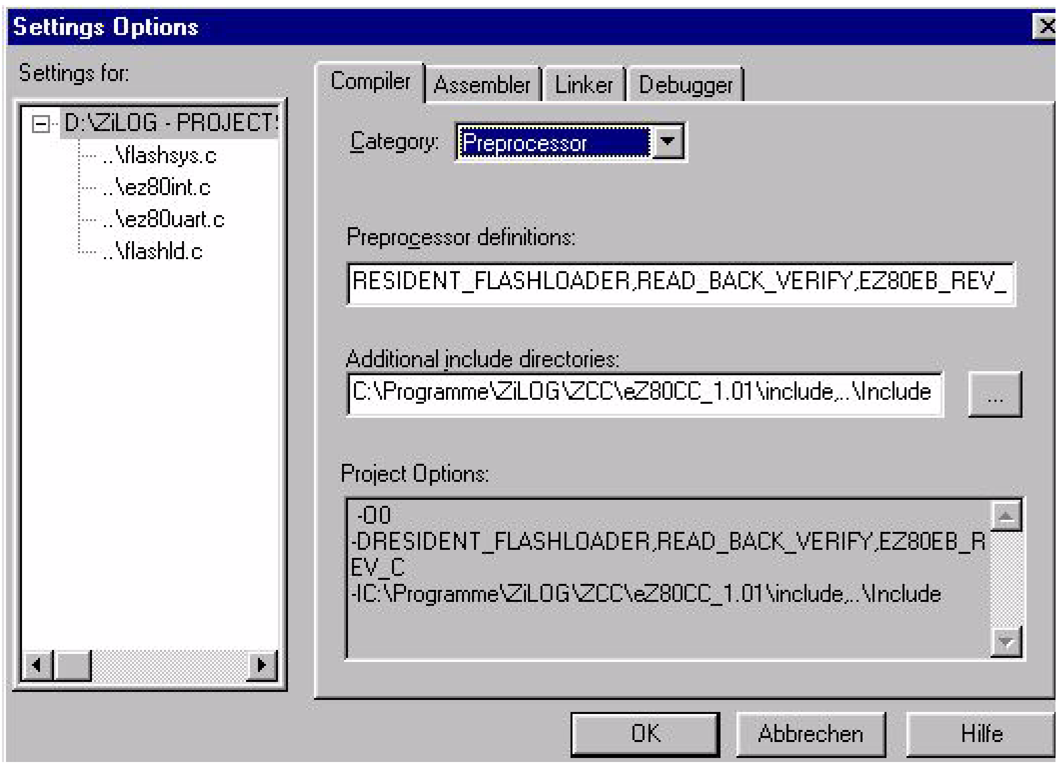

Figure 19. Project Settings Menu—Compiler Preprocessor (Crystal Board)

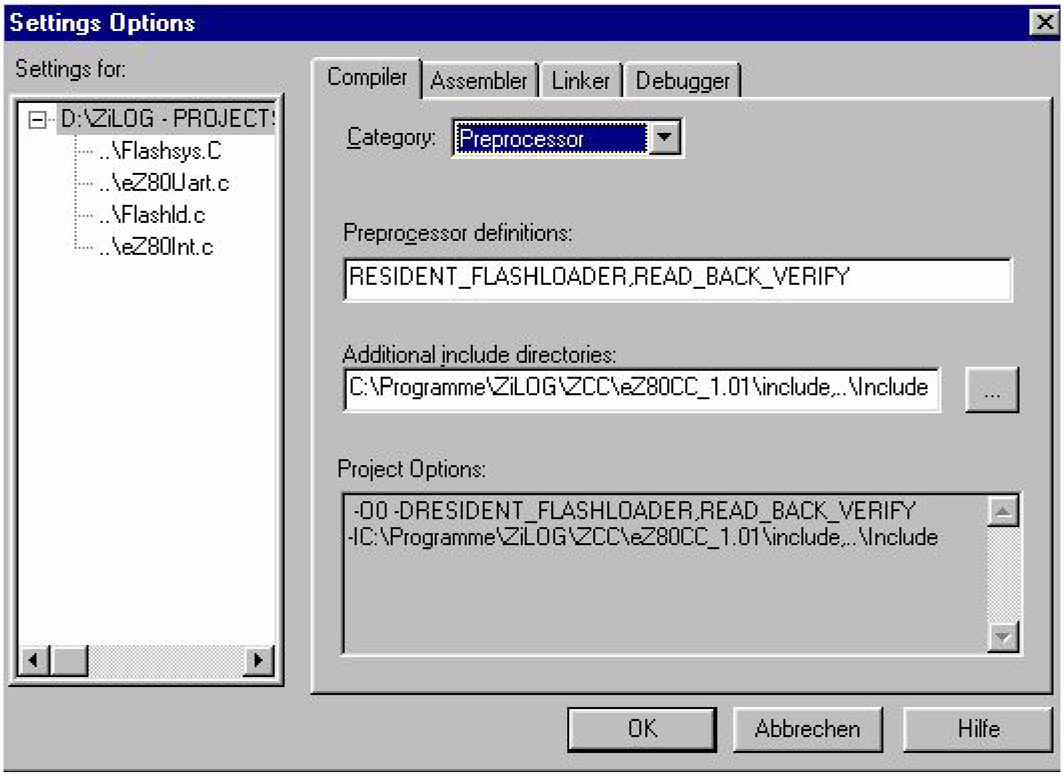

Figure 20. Project Settings Menu - Compiler Preprocessor (Realtek Board)

Figure 21 indicates the memory map setting for BOOT LOADER mode. The memory space must start at address `0x0h` and end at address `0x2FFFFFh`. This memory space includes ROM and RAM, because ZDS does not differentiate between RAM and ROM. To distinguish between RAM and ROM, all data sections must be reassigned to RAM using the Locate menu as shown in Figure 22. This setting forces the linker to reassign both the *.ndata* and the *.nbss* segments to their appropriate addresses. Only one segment makes the first RAM address available.

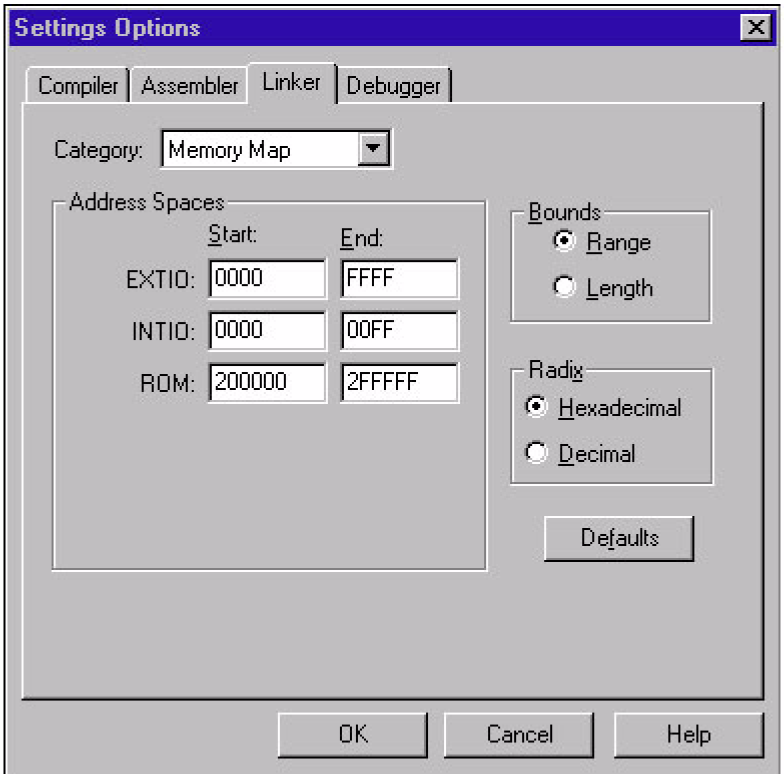

Figure 21. Project Settings Menu—Linker Memory Map

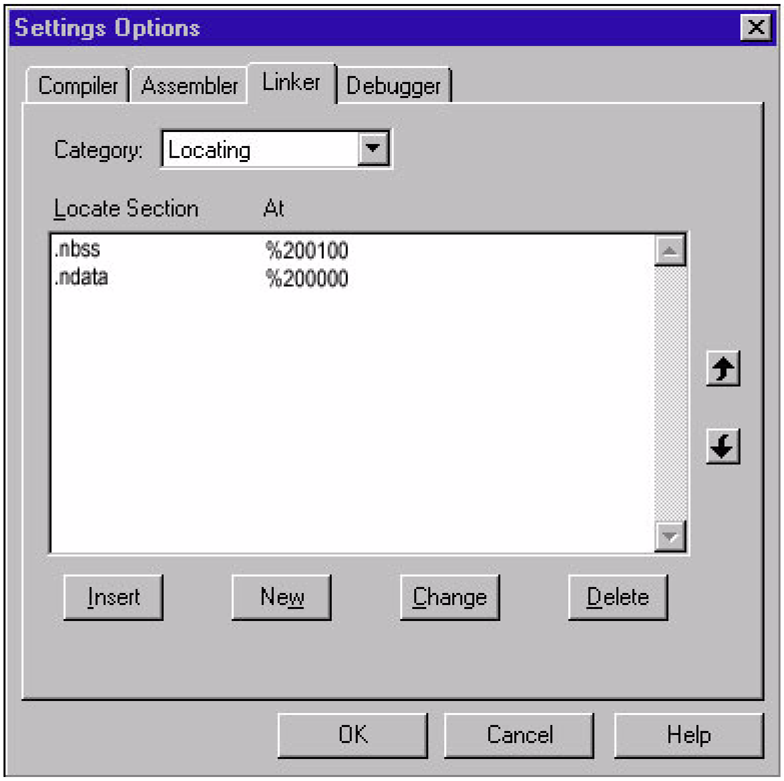

Figure 22. Project Settings Menu—Linker Locating

Test Results

All functions are verified and operate as expected. Though the Flash Loader and Boot Loader applications work correctly when using the C-Compiler version 1.00, version 1.01 is the preferred version due to improved interrupt support.

Additionally, the program is verifiable with both the first eZ80190 silicon and the new version of the silicon. Each test was performed on both the Crystal (Revision C) and the Realtek eZ80 Webserver Evaluation Board.

References

The first four listings below can be found on the ZiLOG web site. The others can be found on their respective company sites.

- eZ80190 Product Specification
- eZ80 CPU User's Manual

- eZ80 Webserver Evaluation Kit User's Manual
- eZ80 C-Compiler User's Manual
- MT28F008B3/MT28F800B3 Micron Technology Flash Memory Product Specification
- Intel HEX File Format Specification

Glossary

Boot Loader	Offers nearly the same functionality as the Flash Loader, but compiled for the end product. The program code must be linked from `0x0h`. All RAM variables must be linked to `0x0h` so that the RAM device can be accessed.
CS	Chip Select.
CTS	Clear To Send—a serial interface control signal for hardware flow control.
DCE	Data Communication Equipment, such as a modem or embedded controller.
DTE	Data Terminal Equipment, such as a PC.
DSR	Data Set Ready—a control signal on the serial interface for the DCE.
DTR	Data Terminal Ready—a control signal on the serial interface for the DTE.
Flash Loader	A program that is able to receive a file, convert it to binary form, and write data to the appropriate address in Flash memory. The Flash Loader requires physical access to the Flash device. Usually the Flash Loader loads into RAM from ZDS via ZDI.
MSB	Most Significant Byte.
PC	Personal Computer.
RAM	Random Access Memory.
ROM	Read Only Memory.
RTS	Ready To Send—a serial interface control signal for hardware flow control.
RX/TX	Receive Signal.
SPL	Stack Pointer Long; a 24-bit stack pointer.

SPS	Stack Pointer Short; a 16-bit stack pointer.
TX	Transmit Signal.
UART	Universal Asynchronous Receiver/Transmitter.
XON/XOFF	Control characters for software flow control (`0x11h`/`0x13h`).
ZDS	ZiLOG Developer Studio.

Appendix 1—Function Code Reference

This section describes the characteristics of each function. The following pages lists each function by its name, followed by its usage in standard syntax, a short description of the function, and a table of parameters.

The standard syntax for C structures is:

TypeDefinition FunctionCode(parameter 1, parameter 2, … parameter n).

ChangeBaudrate

void ChangeBaudrate(void)

This function changes the baud rate.

Defined in:	FLASHLD.C	
Return Value:	None	
Parameters:	void	Void parameter

ChangeFlowControl

void ChangeFlowControl(void)

This function changes the software flow control (XON/XOFF) or hardware flow control (RTS/CTS). This selection is stored in the variable *AppFlowControl*.

Defined in:	FLASHLD.C	
Return Value:	None	
Parameters	void	Void parameter

CheckBlank

BOOL CheckBlank(U8 FlashIndex, U8 BlkNum)

This function checks if the specified Flash block is empty. When empty, the read data must be `0xFFh`.

Defined in:	FLASHLD.C	
Return Value:	TRUE	Flash block is empty
	FALSE	Flash block is not empty and must be erased before writing
Parameters:	FlashIndex	The Flash index is used to access the correct data from the Flash table
	BlkNum	Block number

CheckForSpaceCharacter

S8 CheckForSpaceCharacter(U8 ChannelBase)

This function checks if a SPACE keyboard character is received immediately after power-on. The waiting time for the SPACE character is only a few milliseconds.

Defined in:	FLASHLD.C	
Return Value:	0xFFh	Returns 0xFFh when there is no character in the UART register
	0–256	Returns the last character received by the UART
Parameters:	ChannelBase	Base address of the UART

ChkRxRdy

U8 ChkRxRdy(U8 ChannelBase)

This function checks if a character is available. All bits except RxDataRdy are cleared when returning the value.

Defined in:	EZ80UART.C	
Return Value:	0	No character is available
	RxDataRdy	Rx data is available and can be read
Parameters:	ChannelBase	Base address of the UART

ChkTxRdy

U8 ChkTxRdy(U8 ChannelBase)

This function checks the transmitter state if a new character can be sent. When returning the value, all bits other than TxEmpty and TxRdy are 0.

Defined in:	EZ80UART.C	
Return Value:	0	Transmitter is not ready
	TxEmpty	Transmit register is empty and can be written
	TxRdy	Transmission of the last byte is completed
	TxEmpty+TxRdy	Transmission completed and THR register empty
Parameters:	ChannelBase	Base address of the UART

ClearInterruptVector

BOOL ClearInterruptVector(EZ80_INT_VECTOR Vector)

This function replaces the current interrupt vector with the default interrupt vector for the specified vector number, EZ80_INT_VECTOR.

Note: The DI command must be invoked before calling this function.

Defined in:	EZ80INT.C	
Return Value:	TRUE	Function success
	FALSE	Function failure
Parameters:	Vector	Interrupt vector number according to EZ80_INT_VECTOR

ConfigureFlashRAMFunction

void ConfigureFlashRAMFunction(U32 RamAdr)

This function configures all RAM routines necessary to access Flash for erasing and writing when the Flash Loader is called from external Flash memory. All functions are copied into RAM and the function pointer structure is initialized.

Defined in:	FLASHSYS.C	
Return Value:	None	
Parameters:	RamAdr	RAM address to where all functions should be copied

DisableInterrupt

void DisableInterrupt(void)

This function disables the global eZ80 interrupt.

Defined in:	EZ80INT.C	
Return Value:	None	
Parameters:	void	Void parameter

DisableUARTInterrupt

BOOL DisableUARTInterrupt(U8 ChannelBase, U8 IntMask)

This function disables the UART interrupt according to the given interrupt mask. The interrupt mask must be a valid value for the UART_IER register.

Defined in:	EZ80UART.C	
Return Value:	TRUE	Interrupt Disable success
	FALSE	Interrupt Disable failure
Parameters:	ChannelBase	UART Channel base address
	IntMask	Interrupt mask with set bits for disabling the corresponding interrupt

DoFlowControl

BOOL DoFlowControl(U8 ChannelBase, FLOW_CONTROL FlowControl, FLOW_ACTION Action)

This function controls the flow between the eZ80 and the connected DTE. Software or hardware flow control are invoked by the user.

Defined in:	EZ80UART.C	
Return Value:	TRUE	Success
	FALSE	Failure
Parameters:	ChannelBase	UART Channel base address
	FlowControl	Current flow control method (see FLOW-CONTROL)
	Action	Action to be performed for flow control (see FLOW_ACTION)

DownloadFile

void DownloadFile(FILE_TYPE File)

This function downloads a file into Flash memory. The file is sent as an Intel HEX file from HyperTerminal using the ASCI file transfer mode. It is necessary to invoke flow control.

Defined in:	FLASHLD.C	
Return Value:	None	
Parameters:	File	File type identifier according to FILE_TYPE

DownloadFileToFlash

BOOL DownloadFileToFlash(void)

This function receives the Intel HEX file from the DTE and writes the binary data extracted from this file into Flash memory. The write sequence is performed sector-by-sector to be compatible to other Flash memories. For the Micron Technology devices, the sector size can be set to any value and is defined in the Flash table.

Defined in:	FLASHLD.C	
Return Value:	TRUE	Download and program file successfully
	FALSE	Download or program failure
Parameters:	void	Void parameter

EnableInterrupt

void EnableInterrupt(void)

This function enables the global eZ80 interrupt.

Defined in:	EZ80INT.C	
Return Value:	None	
Parameters:	void	Void parameter

EnableUART

BOOL EnableUART(U8 ChannelBase, U8 PinMask)

This function enables the UART by setting the UZI control register to UART_MODE and setting the corresponding GPIO port to Interrupt Mode 7. This mode sets the GPIO pins to an alternative function controllable by the UART via a pin mask parameter. The user can decide if all port pins should be used.

Defined in:	EZ80UART.C	
Return Value:	TRUE	UART successfully enabled
	FALSE	UART not enabled due to wrong channel base address
Parameters:	ChannelBase	Base address of the UART
	PinMask	Pin mask for alternative function

EnableUARTInterrupt

BOOL EnableUARTInterrupt(U8 ChannelBase, U8 IntMask, U8 pHandler)

This function enables the UART interrupt according to a given interrupt mask. The interrupt mask must be a valid value for the UART_IER register. The pointer to the application handler table contains the application handler address for the corresponding UART interrupt (Modem Int, Line Status Int, RX Int, Tx Int). The handler table *TypeUartHandlerTable* must contain valid addresses or NULL when not used.

Defined in:	EZ80UART.C	
Return Value:	TRUE	Interrupt Enable success
	FALSE	Interrupt Enable failure
Parameters:	ChannelBase	Channel base address of the UART channel
	IntMask	Interrupt mask with bits set for enabling the corresponding interrupt
	pHandler	Pointer to the UART handler table TypeUart HandlerTable

EraseBlock

MT_ERROR_CODE EraseBlock(BOOL UserInput, U8 BlockNumber)

This function erases one block of the Flash memory. The user is requested to input the block number. This number is checked according to the Flash recognized. The start address of the selected block is read from the Flash table.

Defined in:	FLASHLD.C	
Return Value:	MT_ERROR_CODE	Error code according to MT_ERROR_CODE
Parameters:	UserInput	Indicates if user input is required
	BlockNumber	Block number to erase when UserInput = FALSE

EraseFlash

MT_ERROR_CODE EraseFlash(U8 FirstBlock, U8 LastBlock)

This function erases the Flash memory without the boot block and both parameter blocks. When erasing is completed, the data address in all locations must be `0xFFh`.

Defined in:	FLASHLD.C	
Return Value:	MT_ERROR_CODE	Error code according to MT_ERROR_CODE
Parameters:	FirstBlock	First Block to erase
	LastBlock	Last Block to erase

ErrorHandler

BOOL ErrorHandler(MT_ERROR_CODE ErrorCode)

This function outputs a string according to a given error code.

Defined in:	FLASHLD.C	
Return Value:	TRUE	Success
	FALSE	Failure
Parameters:	ErrorCode	Error code according to MT_ERROR_CODE

GetFlashType

U8 GetFlashType(void)

This function requests the Device ID and the Manufacturer ID from the Flash memory. Both values are merged into one 16-bit FlashID whereby the LSB contains the Device ID, the MSB the Manufacturer ID. This FlashID is searched in the Flash table. When found, the variable *ThisFlash* is set to the index of the table entry. The value *ThisFlash* is used by all Flash to work with the correct data. Additionally, when more Flash memory types are supported, *ThisFlash* can be used to distinguish between different Flash algorithms.

Defined in:	FLASHLD.C	
Return Value:	NO_FLASH	FlashID not found in Flash table—failure
	0 to NB_FLASH_ SUPPORTED	Flash table Index
Parameters:	void	Void parameter

GetHexNumber

BOOL GetHexNumber(U8 * pSourceAdr, U8 Count, U32 * pDestAdr)

This function determines the HEX value of an ASCI-coded HEX number located at a specified address. The *Count* in the parameter field indicates the number of ASCI-coded bytes that must be passed. The result is written to an address indicated by the pointer *pDestAdr*.

Defined in:	FLASHLD.C	
Return Value:	TRUE	Converting HEX value successful
	FALSE	Converting HEX value not successful
Parameters:	pSourceAdr	String source address
	Count	Number of digits to convert
	pDestAdr	Destination address where the result must be stored

GetIHexFrame

BOOL GetIHexFrame(U8 * pFrameBuffer, U16 Size)

This function reads one Intel HEX frame from the RxBuffer. The frame data is stored in the location pointed to by *pFrameBuffer*.

Defined in:	FLASHLD.C	
Return Value:	TRUE	Success
	FALSE	Failure
Parameters:	pFrameBuffer	Pointer to the frame buffer
	Size	Maximal number of bytes to receive

GetLineStatus

U8 GetLineStatus(U8 ChannelBase)

This function returns the UART line status.

Defined in:	EZ80UART.C	
Return value:	U8	Contents of the line status register of the selected UART
Parameters:	ChannelBase	Base address of the UART; returns the UART Line Status Register contents

GetModemStatus

U8 GetModemStatus(U8 ChannelBase)

This function returns the status of the UART modem.

Defined in:	EZ80UART.C	
Return value:	U8	Contents of the modem status register of the selected UART
Parameters:	ChannelBase	Base address of the UART; returns the UART Line Status Register contents

GetNextChar

BOOL GetNextChar(U8 * pDest)

This function reads the next character from the receive buffer. When the end of buffer is reached, the read pointer is reset to the start of the receive buffer.

Defined in:	FLASHLD.C	
Return value:	TRUE	Success
	FALSE	No data in buffer
Parameters:	pDest	Pointer to storage location of next character

GetRTS

U8 GetRTS(U8 ChannelBase)

This function returns the current status of the RTS signal. When the RTS signal status is 0, the host PC requests a stop in transmission.

Defined in:	EZ80UART.C	
Return value:	0	Host activated the RTS signal to stop transfer (FlowControl = ON)
	> 0	Host deactivates the RTS signal (FlowControl = OFF)
Parameters:	ChannelBase	Channel Base Address of the UART

InitEZ80Hardware

BOOL InitEZ80Hardware(void)

This function initializes the eZ80 hardware. Further parameters can be included by the user when required.

Note: The DI command must be invoked before calling this function.

Defined in:	FLASHLD.C	
Return value:	TRUE	Hardware initialization success
	FALSE	Hardware initialization failure
Parameters:	void	Void parameter

InitEZ80Interrupt

BOOL InitEZ80Interrupt(void)

This function initializes the eZ80 interrupt. This implementation only requires the UART RX interrupt.

The interrupt vector table and this function reside at address `0xFC00h` of internal RAM.

Note: The DI command must be invoked before calling this function.

Defined in:	FLASHLD.C	
Return value:	TRUE	Interrupt initialization success
	FALSE	Interrupt initialization failure
Parameters:	void	Void parameter

InitIVectorTable

void InitIVectorTable(U16 ivt, U16 ift)

This function initializes the interrupt system of the eZ80. The two parameters *ivt* and *ift* contain the starting address of the interrupt vector table and the interrupt function table. This address must be a 16-bit address.

A default interrupt handler is set, and all interrupt vectors are set to point to this default handler. The vector address must then be set in the eZ80 *I* register.

Note: The DI command must be invoked before calling this function.

Defined in:	EZ80INT.C	
Return value:	TRUE	Function success
	FALSE	Function failure
Parameters:	ivt	16-bit interrupt vector table address
	ift	16-bit interrupt function table address

InitUART

BOOL InitUART(U8 ChannelBase, BAUDRATE BaudRate, U8 BitsPerChar, U8 StopBits, U8 Parity)

This function initializes the UART for proper services. The BRG divisor value is set according to the appropriate baud rate. Additionally, the character framing is configured.

Note: The DI command must be invoked before calling this function.

Defined in:	EZ80UART.C	
Return value:	TRUE	Successful initialization
Parameters:	U8 ChannelBase	Base address of the UART
	BAUDRATE BaudRate	Appropriate baud rate
	BitsPerChar	Number of bits per character
	StopBits	Number of stop bits
	Parity	Parity selection

IrqHandlerDefault

#pragma interrupt void IrqHandlerDefault(void)

This default C-interrupt handler is used when no other interrupts are enabled.

Defined in:	EZ80INT.C	
Return value:	TRUE	Function success
	FALSE	Function failure
Parameters:	void	Void parameter

IrqHandlerUART

void IrqHandlerUART(U8 ChannelBase)

This UART interrupt handler calls the corresponding application interrupt handler. The Line Status and the Modem Status are passed to the application handler. This handler is valid for both UART channels.

Defined in:	EZ80UART.C	
Return value:	None	
Parameters:	ChannelBase	UART channel base address

IrqHandlerUART0

#pragma interrupt void IrqHandlerUART0(void)

This function serves as a UART0 interrupt handler entry.

Defined in:	EZ80UART.C	
Return value:	None	
Parameters:	Void	Void parameter

IrqHandlerUART1

#pragma interrupt void IrqHandlerUART1(void)

This function serves as a UART1 interrupt handler entry.

Defined in:	EZ80UART.C	
Return value:	None	
Parameters:	Void	Void parameter

JumpToApplicationCode

void JumpToApplicationCode(U32 Adr)

This function starts the application code linked at address `0x20000h`. For a Boot Loader, the application code should be verified by using a standard 16-bit CRC checksum that can be located in the parameter sector of Flash memory. In this example, a long jump is executed to the start address of the application code.

Defined in:	FLASHLD.C	
Return value:	None	
Parameters:	Adr	Address of the application code

main

void main(void)

This function is the main entry of the Flash Loader.

Defined in:	FLASHLD.C	
Return value:	None	
Parameters:	Void	Void parameter

mtBlockEraseSeq

MT_ERROR_CODE mtBlockEraseSeq(U32 BA)

This function manages the complete block erase sequence according to the Micron Technology Flash specification. The block erase sequence is a 2-step algorithm and must contain the Erase Setup and the Erase confirmation sequence.

This function does not support SUSPEND ERASE mode.

Defined in:	FLASHSYS.C	
Return value:	MT_ERROR_CODE	Returns one of the error codes defined in MT_ERROR_CODE
Parameters:	BA	Block start address

mtDeviceID

U8 mtDeviceID(U32 FlashBase)

This function performs the sequence to request the device ID of the Micron Technology Flash memory. Together with the Manufacture ID this value builds the Flash ID used as an index in the global Flash table.

Defined in:	FLASHSYS.C	
Return value:	IDDev	Device ID of the current Flash memory
Parameters:	FlashBase	Flash base address

mtManufactureComp

U8 mtManufactureComp(U32 FlashBase)

This function requests the Manufacture ID. The Manufacture ID of the Micron Technology Flash device is `0x89h`. The Manufacture ID and the Device ID build the Flash ID, which is used as an index in the global Flash table.

Defined in:	FLASHSYS.C	
Return value:	ManID	Manufacture ID of the current Flash memory
Parameters:	FlashBase	Flash base address

mtReadBackVerify

BOOL mtReadBackVerify(U32 FlashAddr, U32 NoBytes, U8 * pRamAddr)

This function compares the data from the specified RAM address and the Flash address. Usually this function is called afterwards a WRITE sequence to verify the correctness of written data.

Defined in:	FLASHSYS.C	
Return value:	TRUE	Read back verify success
	FALSE	Read back verify failure
Parameters:	FlashAddr	Flash start address
	NoBytes	Number of bytes to compare
	pRamAddr	Pointer to the RAM start

mtWriteSequence

MT_ERROR_CODE mtWriteSequence(U32 FlashAddr, U32 NoBytes, U8 * pRamAddr)

This function programs Flash memory physically. According to the Flash specification, programming the entire block is unnecessary. The number of bytes to be written range from 1 to $(2^{32})–1$. This function is optimized to call it completely from RAM.

Defined in:	FLASHSYS.C	
Return value:	MT_ERROR_CODE	Returns one of the error code defined in MT_ERROR_CODE
Parameters:	FlashAddr	Flash start address
	NoBytes	Number of bytes to write
	pRamAddr	RAM start address

OutString

void OutString(void)

Writes a string to the UART port.

Defined in:	FLASHLD.C	
Return value:	None	
Parameters:	Void	Void parameter

RAMFunctionEnd

void RAMFunctionEnd(void)

This Help function defines the end address of functions that must be copied from Flash memory into RAM when using the Flash Loader as a Boot Loader.

Defined in:	FLASHSYS.C	
Return value:	None	
Parameters:	Void	Void parameter

RAMFunctionStart

void RAMFunctionStart(void)

This Help function defines the start address of functions that must be copied from Flash memory into RAM when using the Flash Loader as a Boot Loader.

Defined in:	FLASHSYS.C	
Return value:	None	
Parameters:	Void	Void parameter

ReadFlashID

BOOL ReadFlashID

This application function reads the Flash ID, which contains the Device ID and the Manufacturer ID.

Defined in:	FLASHLD.C	
Return value:	TRUE	Read Flash ID success
	FALSE	Read Flash ID failure
Parameters:	Void	Void parameter

ReadSector

IHEX_FRAME_TYP ReadSector(U8 * pSector, U32 * pSectorSize, U32 * pSectorAddr)

This function reads one sector. This implementation reads only one Intel HEX frame. When a complete frame is correctly received, the binary data are stored in the sector buffer. The return value informs the calling function about the type of the frame. When an address frame is received (Type 2 or 4), the address is calculated and stored. Additionally, the number of bytes in this frame is stored.

Defined in:	FLASHLD.C	
Return value:	IHEX_FRAME_TYP	Returns the corresponding frame type received; see IHEX_FRAME_TYP
Parameters:	pSector	Pointer to the sector buffer
	pSectorSize	Pointer to store the sector size
	pSectorAddr	Pointer to store the current sector address

RecvChar

S8 RecvChar(U8 ChannelBase)

This function waits for a character received by the UART channel selected. The function polls the receive status register until a character is received.

Channel Base is one of the 2 UART channel base addresses defined in EZ80REG.H file.

Defined in:	EZ80UART.C	
Return value:	S8	Returns the received character of the selected UART
Parameters:	ChannelBase	Base address of the UART

ResetCTS

void ResetCTS(U8 ChannelBase)

This function resets the CTS signal. The DTE continues transmission when detecting an active Low CTS signal.

Defined in:	EZ80UART.C	
Return value:	None	
Parameters:	ChannelBase	Channel Base Address of the UART

ResetDSR

void ResetDSR(U8 ChannelBase)

This function resets the DSR signal. When using the UART on the DTE, the DSR signal indicates a readiness state to the DTE. Due to this inverse operation, the DTR bit must be reset in the UART modem control register.

Defined in:	EZ80UART.C	
Return value:	None	
Parameters:	ChannelBase	Channel Base address of the UART

SetCTS

void SetCTS(U8 ChannelBase)

Sets CTS signal for hardware flow control. The DTE stops transmission when detecting an active Low CTS signal.

Defined in:	EZ80UART.C	
Return value:	None	
Parameters:	ChannelBase	Channel Base address of the UART

SetDSR

void SetDSR(U8 ChannelBase)

Sets DSR signal. When using the UART as DTE UART, the DSR signal indicates the readiness to the DTE. Due to inverse operation, we must set the DTR bit in the UART modem control register.

Defined in:	EZ80UART.C	
Return value:	None	
Parameters:	ChannelBase	Channel Base address of the UART

SetInterruptMode2

void SetInterruptMode2(U16 ivt)

This function sets the *I* register of the eZ80. The *I* register contains the MSB of the interrupt vector table address. For this implementation, the parameter *ivt* is passed in the *DE* register. Thus, access is granted to the interrupt vector table address by reading the *D* register and storing this value in the *I* register.

Additionally, Interrupt Mode 2 is set for proper operation.

Defined in:	EZ80INT.C	
Return value:	None	
Parameters:	ivt	Interrupt Vector Table address

SetInterruptVector

BOOL SetInterruptVector(EZ80_INT_VECTOR Vector, TypeZ80IntHandler Handler)

This function sets the interrupt vector to the corresponding 5-byte interrupt function and replaces the 24-bit handler address with the handler address given as parameter. The vector location is specified by the parameter vector of type EZ80_INT_VECTOR. With this function, a C user application can setup a certain interrupt vector in real time, whereby the C-interrupt handler can be located anywhere in the 16-MB address space of the eZ80.

Note: The DI command must be invoked before calling this function.

Defined in:	EZ80INT.C	
Return value:	TRUE	Function success
	FALSE	Function failure
Parameters:	Vector	Interrupt vector number according to EZ80_INT_VECTOR
	Handler	Interrupt handler address of type TypeZ80IntHandler

SetULCR

void SetULCR(U8 ChannelBase, U8 BitsPerChar, U8 StopBits, U8 Parity)

Sets the ULCR register of the selected UART. The bit position of this register is set according to the given parameters.

Defined in:	EZ80UART.C	
Return value:	None	
Parameters:	ChannelBase	Base address of the UART
	BitsPerChar	Number of bits per character
	StopBits	Number of stop bits
	Parity	Parity selection

UartRxHandler

void UartRxHandler(U8 ChannelBase, U8 LineStatus, U8 ModemStatus)

ISR for the UART channel. The received data is put in the receive buffer. In the case of the end of the receive buffer, the buffer pointer is set to the start address.

When software flow control is invoked, this handler sets the variable *DTEFLow-Control* every time a XON/XOFF control character is received.

Defined in:	FLASHLD.C	
Return value:	None	
Parameters:	ChannelBase	UART channel base address
	LineStatus	Line status of this channel
	ModemStatus	Modem status of this channel

UartRxToHandler

void UartRxToHandler(U8 ChannelBase, U8 LineStatus, U8 ModemStatus)

Receive Character Time Out interrupt handler. This event only occurs when FIFO is enabled. Because the RX interrupt flag is set only when the number of bytes in the FIFO reaches the programmed threshold, the character time out interrupt is necessary to get access to the data independent of the number of bytes in the FIFO. For detailed information see eZ80 user manual.

Defined in:	FLASHLD.C	
Return value:	None	
Parameters:	ChannelBase	UART channel base address
	LineStatus	Line status of this channel
	ModemStatus	Modem status of this channel

XmitChar

void XmitChar(U8 ChannelBase, S8 c)

This function sends one character to the selected UART port. The character is sent when the transmitter is ready for the next byte.

Defined in:	EZ80UART.C	
Return value:	None	
Parameters:	ChannelBase	Base address of the UART
	c	Character to send

Appendix 2—Type Definition Reference

BAUDRATE

```
enum BAUDRATE {
  BAUD_110,
  BAUD_300,
  BAUD_600,
  BAUD_1200,
  BAUD_2400,
  BAUD_4800,
  BAUD_9600,
  BAUD_19200,
  BAUD_38400,
  BAUD_57600,
  BAUD_115200,
  BAUD_230400,
  BAUD_460800,
  MAX_BAUDRATE,
};
```

Definition of baud rate values to adjust the transfer speed.

Defined in:	EZ80UART.H	
Members	BAUD_110	110 baud
	BAUD_300	300 baud
	BAUD_600	600 baud
	BAUD_1200	1200 baud
	BAUD_2400	2400 baud
	BAUD_4800	4600 baud
	BAUD_9600	9600 baud
	BAUD_19200	19200 baud
	BAUD_38400	38400 baud
	BAUD_57600	57600 baud
	BAUD_115200	115200 baud
	BAUD_230400	230400 baud

Members, cont'd	BAUD_460800	460800 baud
	MAX_BAUDRATE	Maximum number of baud rates supported

BOOL

Type definition of BOOL. Variables from this type contain 8 boolean bits (TRUE or FALSE).

Defined in:	STDDEFS.H

BYTE

Type definition of byte. Variables of this type contain 8 bits.

Defined in:	STDDEFS.H

CHAR

Type definition of signed character. Variables of this type contain 8 bits.

Definition:	STDDEFS.H

DOUBLE

Type definition double. Variables of this type contain 64 bits.

Definition:	STDDEFS.H

DWORD

Type definition of double word. Variables of this type contain 32 bits.

Definition:	STDDEFS.H

EZ80_INT_VECTOR

enum EZ80_INT_VECTOR {

```
IVEC_MACC,
IVEC_DMA0,
```

```
  IVEC_DMA1,
  IVEC_TIMER0,
  IVEC_TIMER1,
  IVEC_TIMER2,
  IVEC_TIMER3,
  IVEC_TIMER4,
  IVEC_TIMER5,
  IVEC_UZI0,
  IVEC_UZI1,
  IVEC_PA0,
  IVEC_PA1,
  IVEC_PA2,
  IVEC_PA3,
  IVEC_PA4,
  IVEC_PA5,
  IVEC_PA6,
  IVEC_PA7,
  IVEC_PB0,
  IVEC_PB1,
  IVEC_PB2,
  IVEC_PB3,
  IVEC_PB4,
  IVEC_PB5,
  IVEC_PB6,
  IVEC_PB7,
  IVEC_PC0,
  IVEC_PC1,
  IVEC_PC2,
  IVEC_PC3,
  IVEC_PC4,
  IVEC_PC5,
  IVEC_PC6,
  IVEC_PC7,
  IVEC_PD0,
  IVEC_PD1,
  IVEC_PD2,
  IVEC_PD3,
  IVEC_PD4,
  IVEC_PD5,
  IVEC_PD6,
  IVEC_PD7,
  IVEC_DEFAULT,
};
```

eZ80 interrupt vector number definition.

Definition:	EZ80INT.H	
Members	IVEC_MACC	MACC interrupt vector
	IVEC_DMA0	DMA0 interrupt vector
	IVEC_DMA1	DMA1 interrupt vector
	IVEC_TIMER0	Timer 0 interrupt vector
	IVEC_TIMER1	Timer 1 interrupt vector
	IVEC_TIMER2	Timer 2 interrupt vector
	IVEC_TIMER3	Timer 3 interrupt vector
	IVEC_TIMER4	Timer 4 interrupt vector
	IVEC_TIMER5	Timer 5 interrupt vector
	IVEC_UZI0	UART0, SPI0 or IIC0 interrupt vector
	IVEC_UZI1	UART0, SPI0 or IIC0 interrupt vector
	IVEC_PA0	Port A Bit 0 interrupt vector
	IVEC_PA1	Port A Bit 1 interrupt vector
	IVEC_PA2	Port A Bit 2 interrupt vector
	IVEC_PA3	Port A Bit 3 interrupt vector
	IVEC_PA4	Port A Bit 4 interrupt vector
	IVEC_PA5	Port A Bit 5 interrupt vector
	IVEC_PA6	Port A Bit 6 interrupt vector
	IVEC_PA7	Port A Bit 7 interrupt vector
	IVEC_PB0	Port B Bit 0 interrupt vector
	IVEC_PB1	Port B Bit 1 interrupt vector
	IVEC_PB2	Port B Bit 2 interrupt vector
	IVEC_PB3	Port B Bit 3 interrupt vector
	IVEC_PB4 \	Port B Bit 4 interrupt vector
	IVEC_PB5	Port B Bit 5 interrupt vector
	IVEC_PB6	Port B Bit 6 interrupt vector
	IVEC_PB7	Port B Bit 7 interrupt vector
	IVEC_PC0	Port C Bit 0 interrupt vector

Members, cont'd	IVEC_PC1	Port C Bit 1 interrupt vector
	IVEC_PC2	Port C Bit 2 interrupt vector
	IVEC_PC3	Port C Bit 3 interrupt vector
	IVEC_PC4	Port C Bit 4 interrupt vector
	IVEC_PC5	Port C Bit 5 interrupt vector
	IVEC_PC6	Port C Bit 6 interrupt vector
	IVEC_PC7	Port C Bit 7 interrupt vector
	IVEC_PD0	Port D Bit 0 interrupt vector
	IVEC_PD1	Port D Bit 1 interrupt vector
	IVEC_PD2	Port D Bit 2 interrupt vector
	IVEC_PD3	Port D Bit 3 interrupt vector
	IVEC_PD4	Port D Bit 4 interrupt vector
	IVEC_PD5	Port D Bit 5 interrupt vector
	IVEC_PD6	Port D Bit 6 interrupt vector
	IVEC_PD7	Port D Bit 7 interrupt vector
	IVEC_DEFAULT	Vector for the default interrupt handler

FILE_TYPE

enum FILE_TYPE {

```
  FILE_APPLICATION,
  FILE_BOOTLOADER,
};
```

Defines file types that can be downloaded into Flash memory.

Defined in:	FLASHLD.H	
Members	FILE_APPLICATION	This application file is linked at `0x20000h`
	FILE_BOOTLOADER	This Boot Loader file is for the Flash boot sector

FLASH_SIZE

enum FLASH_SIZE {

```
  SIZE_1MBYTE,
  SIZE_2MBYTE,
  SIZE_4MBYTE,
  SIZE_8MBYTE,
};
```

Definition of Flash memory size. This mask is used when programming the device. When the original base address within the Intel HEX file (Type 4) indicates a address other than the base address used in the current Flash Loader's CS settings, this address must be adjusted to the current CS settings to be able to access Flash memory physically.

Defined in:	FLASHLD.H	
Members:	SIZE_1MBYTE	1MB Flash Memory
	SIZE_2MBYTE	2MB Flash memory
	SIZE_4MBYTE	4MB Flash Memory
	SIZE_8MBYTE	8MB FLash Memory

FLASH_TYPE

enum FLASH_TYPE {

```
  MT28F800B3_B,
  MT28F800B3_T,
  MT28F008B3_B,
  MT28F008B3_T,
  NB_FLASH_SUPPORTED,
  NO_FLASH,
};
```

This function defines Flash memory types. When necessary, more types can be defined.

Note: The value *NB_FLASH_SUPPORTED* must be the last entry to ensure proper operation.

Defined in:	FLASHLD.H	
Members	MT28F800B3_B	Micron Technology MT28F800B3 Bottom Boot 8-MB Flash
	MT28F800B3_T	Micron Technology MT28F800B3 Top Boot 8-MB Flash
	MT28F008B3_B	Micron Technology MT28F008B3 Bottom Boot 8-MB Flash
	MT28F008B3_T	Micron Technology MT28F008B3 Top Boot 8-MB Flash

NB_FLASH_SUPPORTED

Number of supported Flash memory types.

Defined in:	NO_FLASH	Value for NO FLASH as return value

FLOAT

This function defines the Float type. Variables of this type contain 32 bits.

Defined in:	STDDEFS.H

FLOW_ACTION

enum FLOW_ACTION {

```
  FLOW_ON,
  FLOW_OFF,
};
```

This function defines the action of the flow control mechanism.

Defined in:	EZ80UART.H	
Members	FLOW_ON	Flow control ON—stop DTE
	FLOW_OFF	Flow control OFF—allow data transfer

FLOW_CONTROL

enum FLOW_CONTROL {

```
  SW_FC,
  HW_FC,
};
```

This function defines types that control data flow.

Defined in:	EZ80UART.H	
Members	SW_FC	Software flow control using XON/XOFF
	HW_FC	Hardware flow control using RTS/CTS

IHEX_FRAME_TYP

enum IHEX_FRAME_TYP {

```
  IHEX_TYP0,
  IHEX_TYP1,
  IHEX_TYP2,
  IHEX_TYP3,
  IHEX_TYP4,
  IHEX_NO_TYP,
};
```

Intel HEX frame definition to distinguish between the different frames.

Defined in:	FLASHLD.H	
Members	IHEX_TYP0	Standard data frame type.
	IHEX_TYP1	End of File frame
	IHEX_TYP2	Enhanced 64-KB address control frame
	IHEX_TYP3	Not used
	IHEX_TYP4	Enhanced linear address control frame
	IHEX_NO_TYP	This indicates a wrong frame type.

INT16

Type definition of *signed short int*. Variables of this type contain 16 bits.

Defined in:	STDDEFS.H

INT32

Type definition of signed int. Variables of this type contain 32 bits.

Defined in:	STDDEFS.H

LONG

Type definition of signed long. Variables of this type contain 32 bits.

Defined in:	STDDEFS.H

MT_ERROR_CODE

enum MT_ERROR_CODE {

```
  MT_NO_ERROR,
  MT_Vpp_ERROR,
  MT_WRITE_ERROR,
  MT_WRITE_Vpp_ERROR,
  MT_ERASE_ERROR,
  MT_ERASE_Vpp_ERROR,
  MT_CMD_SEQ_ERROR,
  MT_CMD_SEQ_Vpp_ERROR,
  MT_WRONG_BLOCK_NUM,
  MT_FLASH_ID_ERROR,
  MT_NUM_ERROR,
};
```

This function defines an error code for Micron Technology Flash memory. This error code is derived from the status register.

Defined in:	FLASHLD.H	
Members	MT_NO_ERROR	No error occurred; command successful
	MT_Vpp_ERROR	V_{PP} voltage error
	MT_WRITE_ERROR	WRITE error
	MT_WRITE_Vpp_ERROR	WRITE error; V_{PP} not valid at time of write
	MT_ERASE_ERROR	ERASE error

Members, cont'd	MT_ERASE_Vpp_ERROR	ERASE error; V_{PP} not valid at time of erase
	MT_CMD_SEQ_ERROR	Command sequencing error or WRITE/ERASE error
	MT_CMD_SEQ_Vpp_ERROR	Command sequencing error or WRITE/ERASE error and V_{PP} error
	MT_WRONG_BLOCK_NUM	Wrong block number
	MT_FLASH_ID_ERROR	Flash ID error
	MT_NUM_ERROR	Maximal number of error codes

pEIO

Defines a pointer to a external 8-bit I/O register.

Defined in:	EZ80DEF.H

pIIO

Defines a pointer to an internal 8-bit I/O register.

Defined in:	EZ80DEF.H

pTypeFlashType

Defines a pointer to the structure TypeFlashType.

Defined in:	FLASHLD.H

pTypeRAMFunction

Defines a pointer to the structure TypeRAMFunction.

Defined in:	FLASHSYS.H

pTypeUARTHandlerTable

Defines a pointer to the structure TypeUartHandlerTable.

Defined in:	EZ80UART.H

pTypeUARTHandlerTable

Pointer to the structure TypeUartIrqTable.

Defined in:	EZ80UART.H

pTypeZ80IFunction

Defines a pointer to the function code TypeZ80IFunction.

Defined in:	EZ80INT.H

pTypeZ80IVector

Defines a pointer to the interrupt vector TypeZ80IVector.

Defined in:	EZ80INT.H

S16

Type definition for a signed 16-bit variable.

Defined in:	STDDEFS.H

S32

Type definition for a signed 32-bit variable.

Defined in:	STDDEFS.H

S8

Type definition for a signed 8-bit variable.

Defined in:	STDDEFS.H

SHORT

Type definition of signed short. Variables of this type contain 16 bits.

Defined in:	STDDEFS.H

TypeBlockEraseSeqFunction

BlockEraseSeq function used for erasing Flash memory block.

Defined in:	FLASHSYS.H

TypeDeviceIDFunction

DeviceID function used to get the device ID.

Defined in:	FLASHSYS.H

TypeFlashType Structure

```
struct {
  FLASH_TYPE FlashType;
  U16 FlashID;
  U8 NoBlocks;
  FLASH_SIZE SizeMask;
  U32 SectorSize;
  S8 FlashName[17];
  U32 FlashBlock[MAX_NB_BLOCK+1];
} TypeFlashType;
```

Structure with Flash memory specific data.

Defined in:	FLASHLD.H	
Members:	FlashType	Flash memory of type FLASH_TYPE
	FlashID	Flash ID containing Device ID (LSB) and Manufacture ID (MSB)
	NoBlocks	Number of blocks
	SizeMask	Flash size to adjust the address when writing data, see FLASH_SIZE
	SectorSize	Maximal Sector size for programming the Flash (not really necessary for Micron Technology)
	FlashName[17]	Flash memory name—maximum 16 characters
	FlashBlock[MAX_NB_BLOCK+1]	Flash block address table for this Flash memory

TypeManufactureCompFunction

ManufactureComp function used to get the manufacture ID.

Defined in:	FLASHSYS.H

TypeRAMFunction Structure

```
struct {
  TypeBlockEraseSeqFunction BlockEraseSeq;
  TypeWriteSequenceFunction WriteSequence;
  TypeDeviceIDFunction DeviceID;
  TypeManufactureCompFunction ManufactureComp;
  TypeReadBackVerifyFunction ReadBackVerify;
} TypeRAMFunction;
```

This structure contains the function pointer to Flash, located in RAM. When using the Flash loader as a Boot Loader, all Flash routines must be copied into RAM and called from this location.

Defined in:	FLASHSYS.H	
Members:	BlockEraseSeq	Block Erase sequence
	WriteSequence	Write Sequence function
	DeviceID	Get Device ID
	ManufactureComp	Get Manufacture ID
	ReadBackVerify	Read and verify function

TypeReadBackVerifyFunction

ReadBackVerify function used for erasing Flash memory block.

Defined in:	FLASHSYS.H

TypeUARTHandler

Global UART interrupt handler function. The application must create this type of handler and must enable a UART interrupt with this handler as parameter.

Parameter:	ChannelBase:	Channel base address of the UART requested this interrupt
	LineStatus:	Line status register value of this channel
	ModemStatus:	Modem status register value of this channel

When an application creates a UART channel interrupt handler, it must incorporate the following syntax:

```
void ApplicationIrqHandler(U8 ChannelBase, U8 LineStatus, U8
ModemStatus){...};
```

Defined in:	EZ80UART.H

TypeUartHandlerTable Structure

```
struct {
  TypeUARTHandler UartTxHandler;
  TypeUARTHandler UartRxHandler;
  TypeUARTHandler UartRxToHandler;
  TypeUARTHandler UartLsHandler;
  TypeUARTHandler UartMsHandler;
} TypeUartHandlerTable;
```

Structure for the interrupt handler table used to enable UART interrupts. See EnableUARTInterrupt.

Defined in:	EZ80UART.H	
Members:	UartTxHandler	Application Transmit interrupt handler
	UartRxHandler	Application Receive interrupt handler
	UartRxToHandler	Application Receive character time out interrupt handler
	UartLsHandler	Application Line status interrupt handler
	UartMsHandler	Application Modem status interrupt handler

TypeUartIrqTable Structure

```
struct {
  U8 IntMask;
  TypeUartHandlerTable HandlerTable;
} TypeUartIrqTable;
```

Structure for the interrupt table used to enable UART interrupts. See EnableUARTInterrupt

Defined in:	EZ80UART.H	
Members	IntMask	Interrupt mask for the UART channel
	HandlerTable	Application interrupt handler table

TypeWriteSequenceFunction

WriteSequence function used for programming Flash memory blocks.

Defined in:	FLASHSYS.H

TypeZ80IFunction Structure

```
struct {
  U16 OpCode;
  U8 AdrLo;
  U8 AdrMi;
  U8 AdrHi;
} TypeZ80IFunction;
```

Defines a minimal interrupt function to call the C-Interrupt handler defined with *#pragma interrupt*. This function contains the opcode `0xC3h`, `0x5Bh`, and a 24-bit address. The address of this function is placed in the interrupt vector table as a vector.

Defined in:	EZ80INT.H	
Members:	OpCode	Opcode *JP.LiL*.
	AdrLo	Low byte of the 24-bit C-Handler address
	AdrMi	Middle byte of the 24-bit C-Handler address
	AdrHi	High byte of the 24-bit C-Handler address

TypeZ80IntHandler

Defines the eZ80 interrupt entry function.

Defined in:	EZ80INT.H

TypeZ80IVector Structure

```
struct {
  U16 Vector;
} TypeZ80IVector;
```

Defines the interrupt vector of the eZ80.

Defined in:	EZ80INT.H	
Members:	Vector	16-bit interrupt vector

U16

Type definition for a unsigned 16 bit variable.

Defined in:	STDDEFS.H

U24

Type definition for a unsigned 16 bit variable.

Defined in:	STDDEFS.H

U32

Type definition for a unsigned 32 bit variable.

Defined in:	STDDEFS.H

U8

Type definition for a unsigned 8 bit variable.

Defined in:	STDDEFS.H

UCHAR

Type definition of unsigned character. Variables of this type contain 8 bits.

Defined in:	STDDEFS.H

UINT16

Type definition of unsigned short int. Variables of this type contain 16 bits.

Defined in:	STDDEFS.H

UINT24

Type definition of unsigned short int. Variables of this type contain 24 bits.

Defined in:	STDDEFS.H

UINT32

Type definition of unsigned int. Variables of this type contain 32 bits.

Defined in:	STDDEFS.H

ULONG

Type definition of unsigned long. Variables of this type contain 32 bits.

Defined in:	STDDEFS.H

USHORT

Type definition of Unsigned Short variable. Variables of this type contain 16 bits.

Defined in:	STDDEFS.H

WORD

Type definition of Word variable. Variables of this type contain 16 bits.

Defined in:	STDDEFS.H

Notes

This Flash Loader application is written to support customers using the eZ80 and its associated tools. This application offers users the possibility of programming a Flash device onto the eZ80 Webserver Evaluation Board (or user target board) to create a standalone version or to test the entire application in real time without a debug environment. With the Flash Loader application, the software designer does not require detail to use the Flash algorithm. Also, the Boot Load feature is offered as a compiler switch for end applications. Because it is not feasible to address all application requirements, the possibility exists that some issues within the Flash Loader implementation may require modification. So that ZiLOG may respond to these issues, the reader of this Application Note is invited to complete and return the Customer Feedback Form found on the last page of this document.

Customer Feedback Form

Flash Loader for the eZ80 Webserver Evaluation Board Application Note

If you experience any problems while operating this product, or if you note any inaccuracies while reading this Application Note, please copy and complete this form, then mail or fax it to ZiLOG (see *Return Information*, below). We also welcome your suggestions!

Product Information

eZ80190 Webserver
Serial # or Board Fab #/Rev. #
Software Version
Document Number
Host Computer Description/Type

Customer Information

Name	Country
Company	Phone
Address	Fax
City/State/Zip	E-Mail

Return Information

ZiLOG
System Test/Customer Support
910 E. Hamilton Avenue, Suite 110, MS 4–3
Campbell, CA 95008
Fax: (408) 558-8536
ZILOG World Wide Customer Support Center

Problem Description or Suggestion

Provide a complete description of the problem or your suggestion. If you are reporting a specific problem, include all steps leading up to the occurrence of the problem. Attach additional pages as necessary.