eZ80Acclaim! and eZ80AcclaimPlus!

eZ80 Boot Loader

Product User Guide

PUG003501-0217

Introduction

The eZ80 Boot Loader is a target-resident utility for the eZ80Acclaim! and eZ80Acclaim Plus! families of microcontrollers and microprocessors that allows an application programmed into (internal and/or external) Flash memory on the target system to be updated over a serial cable connected to an eZ80 UART. The eZ80 Boot Loader can be used to provide field-upgradable firmware to applications targeting the eZ80Acclaim! series of devices.

A terminal program is used on the PC to transmit the contents of an Intel Hex file generated by the Zilog Developer Studio (ZDS II) Integrated Development Environment (IDE) containing the program image to the eZ80 Boot Loader running on the target device. The Boot Loader converts the Intel Hex file to binary format and programs the application into Flash memory. After programming is complete, the target can be rebooted, causing the user-application to take control of the system. The Boot Loader can also be used to display information such as size and geometry about the external Flash device and to display, program, and erase the contents of Flash from an interactive console menu.

The PC used to download the Intel Hex file does not require the ZDS II IDE software to be installed. The only requirements are a copy of the Hex file containing the application program to be downloaded and a terminal program capable of transferring data over a serial cable, such as Tera Term.

Features

Key features of the eZ80 Boot Loader application include the following:

- Provides a simple environment for end-users, production testers, or SW developers to load an application into internal and/or external Flash
- Used to reprogram application firmware independently of ZDS

Requirements

This section lists the software and hardware requirements for the eZ80 Boot Loader.

Software Requirements

Initial deployment (including optional customization) of the eZ80 Boot Loader requires the following software:

- ZDS II IDE version 5.3.0 (or later)
- Zilog Flash Library (ZFL)
  - Included with the ZDS II – eZ80Acclaim! software version 5.3.0 (and later)

The PC used to interact with the eZ80 Boot Loader and transfer an Intel Hex file containing the user application requires the following software:

- Terminal program such as Tera Term or HyperTerminal. The version used must be capable of:
  - Data transfer at 57.6kbps with 8 data bits, no parity and 1 stop bit
  - Transferring an ASCII file (in Intel Hex format)

Hardware Requirements

Initial deployment (including optional customization) of the eZ80 Boot Loader requires the following hardware:

- One of the supported eZ80 Development Platforms with associated power supply
  - eZ80F910200ZCO, eZ80F910200KITG, eZ80F910300KITG, eZ80F910300ZCOG, eZ80F920200ZCOG, eZ80F930200ZCO
  - Z80L920210ZCO with eZ80L925048MODG or eZ80L925148MODG
  - Zilog USB Smart Cable (ZUSBSC00100ZACG)

The PC used to interact with the eZ80 Boot Loader and transfer an Intel Hex file containing the user application requires the following hardware:

- Serial cable (DB9 male to DB9 female)
  - Alternatively, a USB to serial adapter can be used, provided there is a DB9 male connector available to connect to the eZ80 development platform

    If the serial adapter does not include a male DB9 connector and only provides connections to TTL signals, connect the following signals between the USB to serial adapter and the eZ80 main board.

USB to Serial Adapter TTL Signal (Equivalent DB9 Pin Number	eZ80 Main Board Signal (J6 Header Strip Pin Number)
TxD	PD1_RXD0 (J6.34)
RxD	PD0_TxD (J6.36)
GND	GND (J6.55 through J6.60)

Settings

The terminal program used to interact with the eZ80 Boot Loader should be configured as follows:

- Baud rate of 57.6 kbps
- 8 data bits
- No parity
- 1 stop bit

By default, the eZ80 Boot Loader uses a baud rate of 57.6kbps for compatibility with sample programs that use the Zilog TCP/IP (ZTP) Software Suite which include a command shell running on UART 0 at 57.6kbps. To reduce the amount of time required to program large application Hex files, the eZ80 Boot Loader can be recompiled to use a higher baud rate. The eZ80 Boot Loader can also be modified to use UART1 instead of UART0 based on the target HW platform. For more information, refer to the Customizing the eZ80 Boot Loader section on page 25.

Jumper Settings

Consult the target eZ80 Development Platform User Manual to ensure that any jumpers used to enable write access to external Flash (and internal Flash for eZ80F91x150 modules) are configured such that write-access is enabled.

- There is a jumper on the eZ80F917150MODG module that is used to enable write access to the first 32KB of internal Flash in the eZ80F91 microcontroller (referred to as the eZ80F91 boot-block). This jumper must be configured to allow write access to the eZ80F91 boot-block for initial programming of the eZ80F91 Boot Loader.
- Other eZ80 Development kits typically include a Flash Write Enable (Flash WE) jumper that must be configured to allow write access to the boot block of external Flash devices for initial programming of the eZ80F91 Boot Loader and/or programming the application Hex file into external Flash.
- eZ80 Development kits also typically include a jumper that can be used to disable access to external Flash. If present, this jumper must be configured to enable external Flash in order for the eZ80 Boot Loader to be able to program the application into external Flash memory.

Memory Map

This section describes the memory map of the eZ80 Boot Loader and the user application applicable to the eZ80 and eZ80Acclaim! families of devices.

Memory Map for the eZ80F9x Series of Microcontrollers

Each of the microcontrollers in the eZ80F9x series of MCUs includes internal Flash memory. After Power-On Reset (POR), the system starts executing code from internal Flash at address 0x0. Because the eZ80F9x Boot Loader needs to take control of the system after POR, it must be programmed into the beginning of eZ80F9x internal Flash memory.

The default build of the eZ80F9x Boot Loader requires slightly less than 16KB of Flash memory. On the eZ80F92 and eZ80F93 microcontrollers, the smallest block of Flash that can be protected by setting bit 0 of the Flash Protection register is 16KB. Therefore, after POR, the eZ80 Boot Loader sets bit 0 in the Flash Protection register, effectively reserving the first 16KB of internal Flash memory and protecting itself from accidental erasure. However, bits 1 through 7 in the Flash Protection register are cleared, allowing the eZ80F9x Boot Loader to modify the non-reserved portion of internal Flash.

On the eZ80F91 microcontroller, the smallest block of Flash that can be protected by setting bit 0 in the Flash Protection register is 32KB. Therefore, when running in the eZ80F91 microcontroller, the eZ80F9x Boot Loader reserves the first 32KB of internal Flash memory. This block of eZ80F91 internal Flash is referred to as the boot-block and is further protected by the WP pin (active low) via a jumper on the eZ80F91 development kit/module.

Because the eZ80F9x Boot Loader reserves the first block of eZ80F9x internal Flash (protected by bit 0 of the Flash Protection register), the user-application must start at the address corresponding to the beginning of the 2nd block of internal Flash protected by bit 1 in the Flash Protection register as shown in Table 1. This is referred to as the APPLICATION_START_ADDR throughout this document.

Table 1. eZ80F9x Application Start Address

Device	Internal Flash Reserved for eZ80F9x Boot Loader (Hex)	Start of User Application in Internal Flash (Hex)
eZ80F91	0000-7FFF	8000
eZ80F92	0000-3FFF	4000
eZ80F93	0000-3FFF	4000

The eZ80 Boot Loader only requires approximately 20 KB of external RAM for proper operation. All Zilog development kits include at least 128KB of external RAM (typically on CS1 and/or CS2) and the default target configuration files (*.ZTGT) included with ZDS ensure that one of the eZ80 chip selects accesses external RAM between addresses 0xB80000 to 0xB9FFFF. Consequently, the default build configuration of all eZ80 Boot Loaders requires the target to have 128KB of RAM accessible between 0xB80000 and 0xB9FFFF.

Figure 1 shows the system memory map when the eZ80F9x Boot Loader is programmed into the first protect-block (Block 0) of an eZ80F9x microcontroller. Although the eZ80F9x Boot Loader is programmed into the eZ80F9x boot-block, the program runs from external RAM.

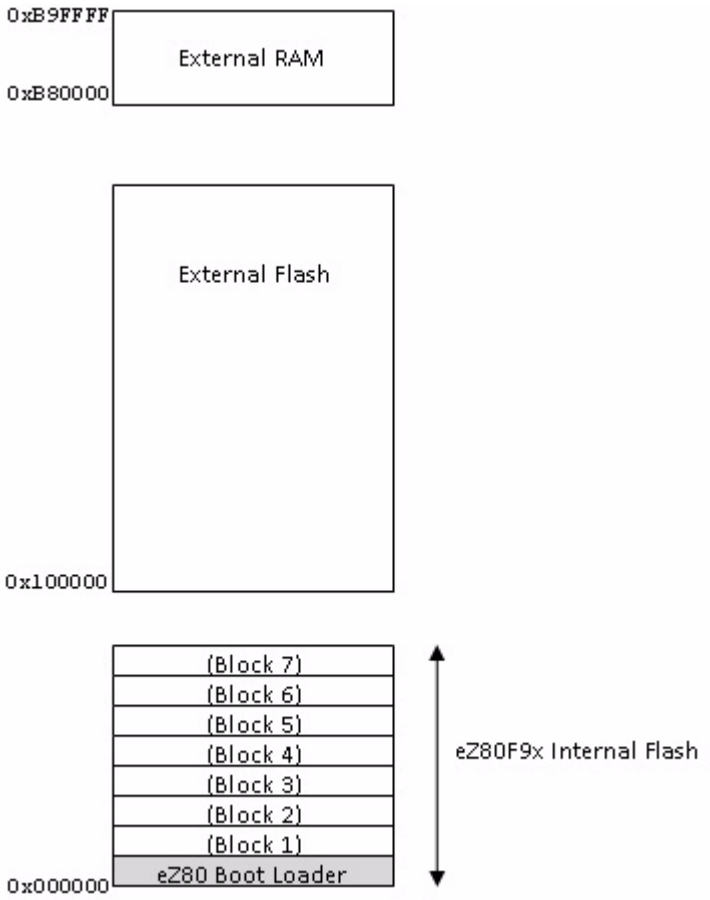

Figure 1. eZ80 Boot Loader Memory Map on the eZ80F9x Series

The user-application starts at the beginning of Block 1 in eZ80F9x internal Flash and extends upwards through internal Flash and into the external Flash address space as necessary. In Figure 1, the address of the last byte of external Flash is not shown because different target platforms have different amounts of external Flash. Zilog development kits typically have between 1MB and 8MB of external Flash.

For compatibility with the ZDS default target configuration files, the eZ80F9x Boot Loader locates external Flash at address `0x100000`. When used on Zilog development kits with 8MB external Flash and an LED matrix starting at `0x800000` (for example, eZ80F910300ZCOG), only the first 7MB of external Flash will be accessible using the eZ80Boot Loader.

Before activating the user application, the eZ80F9x Boot Loader sets all bits in the Flash Protection register to 1 to mimic the POR state of this register. While the application is running, if an NMI occurs, or the program executes an *RST n* instruction (with *n* equal to `8h`, `10h`, `18h`, `20h`, `28h`, `30h` or `38h`), the corresponding exception handler in the eZ80F9x Boot Loader code space executes. These exception handlers simply jump to the

corresponding handler at the beginning of the application code at APPLICATION_START_ADDR (as defined in Table 1) in external Flash. If an *RST 0* instruction executes, the boot loader is re-initialized, which will typically cause the application to get restarted.

Interrupt and Exception Handling on the eZ80F92 and eZ80F93

The I-register in the eZ80F92 and eZ80F93 microcontrollers is only 8 bits (versus 16 bits in the eZ80F91), requiring the eZ80 interrupt vector table to be located on a 256-byte boundary in the first 64KB of external Flash. Additionally, each vector is only 16-bit wide (versus 32-bit wide in the eZ80F91). Therefore, all application interrupt service routines must be routed through the eZ80F92/eZ80F93 Boot Loaders.

The interrupt handlers in the eZ80F92 and eZ80F93 Boot Loaders vector through the corresponding application interrupt jump table at the start of external RAM (0xB80000). For example, when the interrupt for vector 0x0A fires, the boot loader's interrupt handler will jump to address 0xB80000 + (0x0A * 2), corresponding to the ISR the application installs for vector 0x0A via the set_vector() runtime library routine. This requires the user application to be built with its RAM-based interrupt vector table located at 0xB80000, else the eZ80F92/eZ80F93 Boot Loader must be rebuilt with the IVJMPTBL located at the address of the application interrupt vector table.

NMI and reset handlers are handled the same as on the eZ80F91 MCU.

Memory Map for the eZ80L92 of Microprocessor

The eZ80L92 microprocessor does not contain any internal memory (FLASH or RAM). After POR, the system starts executing code from external Flash on CS0 at address 0x0. Because the eZ80L92 Boot Loader is required to take control of the system after POR, it must be programmed into the beginning of external Flash memory and CS0 must be mapped to start at address 0x0.

The default build of the eZ80L92 Boot Loader requires slightly less than 16KB of Flash memory. Depending on the geometry of the Flash connected to CS0, this may be smaller than, equal to, or larger than the first (boot-block) of external Flash. If the external Flash has a HW signal to protect its boot-block from being programmed or erased, this signal will need to be disabled when the eZ80L92 Boot Loader is loaded into external Flash.The eZ80L92 Boot Loader does not attempt to protect the boot-block in external Flash. Therefore, after the eZ80L92 Boot Loader has been programmed into external Flash, the user should ensure the signal that prevents modification of the boot-block in external Flash is once again enabled.

Depending on which eZ80L92 development kit is used, the actual size of external Flash will range between 1MB and 8MB. Although eZ80L92 development kits typically include 1MB of RAM, the eZ80L92 Boot Loader only accesses 128KB of RAM between 0xB80000 and 0xB9FFFF.

Figure 2 shows the memory map when the eZ80L92 Boot Loader is programmed into the first 16KB of external Flash. Because the eZ80L92 Boot Loader cannot simultaneously execute code from external Flash and program/erase external Flash, the Boot Loader must run from external RAM.

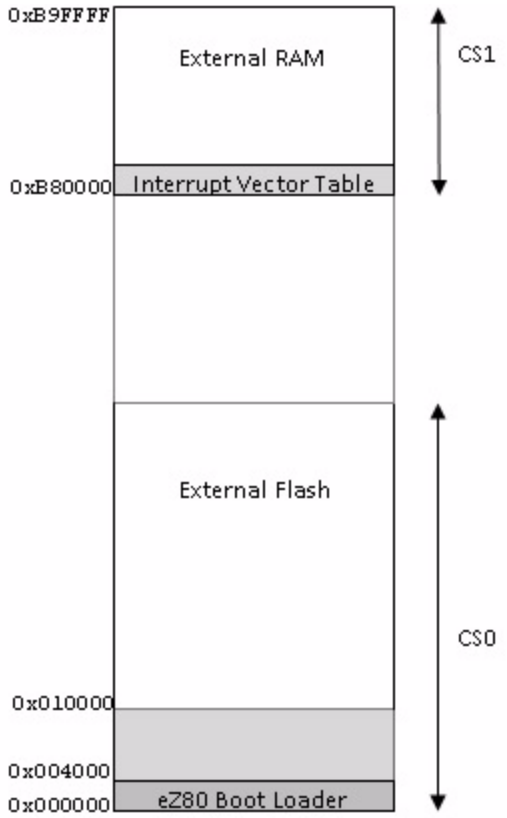

Figure 2. eZ80 Boot Loader Memory Map on the eZ80L92 Microprocessor

The user application that the eZ80L92 Boot Loader programs into external Flash must not reside in the same erase-block as the eZ80L92 Boot Loader program image. Otherwise, the eZ80L92 Boot Loader would erase itself from external Flash while erasing the application image. Depending on which external Flash device is used, the 16KB of Flash consumed by the eZ80L92 Boot Loader may be smaller than, equal to, or larger than the external Flash erase-block size. To ensure compatibility with a large number of external Flash devices, the eZ80L92 Boot Loader requires the application program to start at address `0x010000`[1] as most Flash devices use an erase block size of 64KB or less, especially with bottom-boot Flash devices[2].

Notes:

1. When used with the eZ80L920210ZCO with eZ80L925048MODG, the application start address should be 0x008000.
2. Zilog strongly recommends using only-bottom boot devices with the eZ80 Boot Loader because the Common Flash Interface (CFI) geometry table used by older AMD-compatible Flash devices is always presented in bottom-boot format even if Flash is a top-boot device, which can cause the eZ80 Boot Loader to misinterpret the geometry.

Interrupt and Exception handling on the eZ80L92

The I-register in the eZ80L92 microprocessor is only 8-bits wide, requiring the eZ80 interrupt vector table to be located on a 256-byte boundary in the first 64KB of external Flash. Additionally, each vector is only 16-bit wide, requiring all application interrupt service routines to be routed through the eZ80L92 Boot Loader.

The interrupt handlers in the eZ80L92 Boot Loader vector through the corresponding application interrupt jump table at the start of external RAM (`0xB80000`). For example, when the interrupt for vector `0x0A` fires, the eZ80's default interrupt handler will jump to address `0xB80000` + (`0x0A` * 2), corresponding to the ISR the application installs for vector `0x0A` via the `set_vector()` runtime library routine. This requires that the application is built with its RAM-based interrupt vector table located at `0xB80000`, or that the eZ80L92 Boot Loader is rebuilt with the IVJMPTBL located at the RAM address used by the application.

If an NMI occurs while the user application is running, or the program executes an *RST n* instruction (with *n* equal to `8h`, `10h`, `18h`, `20h`, `28h`, `30h` or `38h`), the corresponding exception handler in the eZ80L92 Boot Loader code space executes. These exception handlers simply jump to the corresponding handler at the beginning of the application code at `0x010000` (default value of `APPLICATION_START_ADDR` on the eZ80L92 microprocessor) in external Flash. If an *RST 0* instruction executes, the boot loader is re-initialized, which will typically cause the application to get restarted.

Installing the eZ80 Boot Loader into Flash Memory on the Target

This section describes how to initially build and program the eZ80 Boot Loader into Flash memory on a supported development kit.

1. Launch the ZDS II – eZ80Acclaim! 5.3.0 (or later version) IDE.
2. Navigate to the `<ZDS II Installation folder>\applications\Boot-Loader` folder and open the appropriate `eZ80xxx_BootLoader.zdsproj` project file, where `xxx` identifies the target MCU, such as F91 or L92.
3. From the **Build configuration** menu, select the target Zilog development kit based on the Zilog development kit number (silkscreen lettering visible on the development board). Figure 3 shows examples of build configuration options for select eZ80F91 development kits.

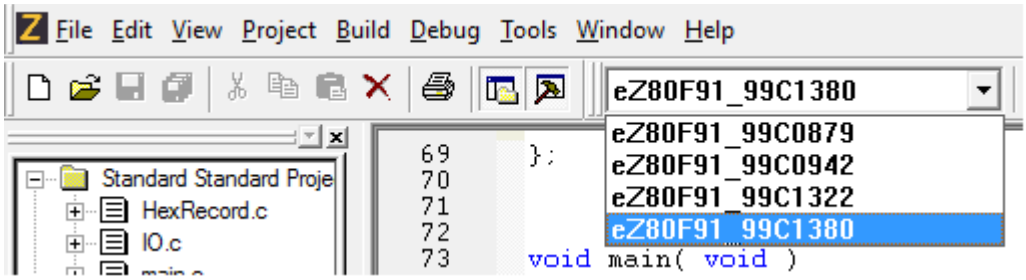

Figure 3. Example of Build Configuration Options for eZ80F91 Development Kits

4. From the **Build** menu, select **Rebuild All** or click the Rebuild All icon  to regenerate the Boot Loader application.
5. Refer to the appropriate eZ80 development kit user manual to ensure that any jumpers on the target development board that protect the boot-block and enable write access into Flash are configured such that the boot-block is not write-protected and write access to (external) Flash is enabled (only applicable to the eZ80L92 microprocessor).
6. Refer to the appropriate eZ80 development kit user manual to connect the appropriate Zilog debug device (such as the USB SmartCable) between the PC running ZDS and the target development kit.
7. To use the eZ80 Boot Loader interactive console, it is necessary to connect a serial cable (or a USB to serial dongle) between the target eZ80 development kit and a PC running a terminal program such as Tera Term or HyperTerminal. Complete instructions on this procedure can be found in the appropriate eZ80 development kit user manual.
8. Open the terminal program on the PC to which the serial cable is connected and ensure the terminal program is configured to operate at 57.6kbps with 8 data bits, no parity and 1 stop.

9. From the **Debug** menu, select **Download Code** or click the Download Code icon  to download the Boot Loader application into (internal or external) Flash memory on the target development kit.
10. After the Boot Loader has been programmed into Flash memory on the target, the Zilog debug cable can be disconnected from the target development kit and the development kit can be reset, via the reset button, if present, or by disconnecting and then reconnecting power to the development kit, to activate the eZ80 Boot Loader. Alternatively, from the **Debug** menu, select **Go** or click the Go icon  to activate the eZ80 Boot Loader.
11. Proceed to Step 3 in the Using the Boot Loader to Download a User Application section.

Using the Boot Loader to Download a User Application

This section describes how to download a user application into Flash memory on the target eZ80 development kit using the eZ80 Boot Loader. Refer to the Creating an Application Compatible with the eZ80 Boot Loader section for a description of the project setting changes required to make an existing application bootable using the eZ80 Boot Loader. For a description of all the eZ80 Boot Loader menu options, refer to the eZ80 Boot Loader Menu Options section.

After the Boot Loader has been programmed into Flash on the target development platform, use the following procedure to program a user application into Flash using a ZDS-generated Intel Hex file.

1. To use the eZ80 Boot Loader interactive console, connect a serial cable (or a USB to serial dongle) between the target eZ80 development kit and a PC running a terminal program such as Tera Term or HyperTerminal. Complete instructions on this procedure can be found in the appropriate eZ80 development kit user manual.
2. Open the terminal program on the PC to which the serial cable is connected and ensure the terminal program is configured to operate at 57.6kbps with 8 data bits, no parity and 1 stop.
3. In the terminal program window press the *z* key and then power on/reset the target eZ80 development kit. After *z* characters are displayed in the console window, release the *z* key. This action will cause the eZ80 Boot Loader main menu to appear. If the *z* key is not pressed within approximately 250 milliseconds of when the target CPU comes out of reset, the eZ80 Boot Loader checks to see if the application reset handler is valid (first 3 bytes of the application reset handler at `APPLICATION_START_ADDR` are not `0xFFFFFF`). If so, the application reset handler is executed, otherwise the Boot Loader main menu is displayed.
4. Before attempting to program the user application into Flash memory, if the target eZ80 development kit includes a jumper used to enable write access to external Flash, ensure that this jumper is configured to allow write access to external Flash. Perform this step only if the user application requires the use of external Flash.

5. After the Boot Loader menu is displayed, press the *e* key to erase all internal and external Flash memory except for the portion of Flash that contains the eZ80 Boot Loader program image. When prompted, press *y* to confirm that all Flash memory should be erased. Alternatively, if the user knows which blocks of Flash needs to be programmed, then the block erase (b) menu command can be used to erase only those blocks of Flash memory.
6. Press the *p* key to program the user application into Flash from the contents of an Intel Hex file generated using ZDS. The eZ80 Boot Loader then displays a message indicating that it is waiting for the Hex file to be sent.
7. In the terminal program, select the menu option to transfer an ASCII text file without the use of any modem protocol such as XMODEM or Kermit. For example, using Tera Term, the Hex file can be sent by selecting **Send File…** from the **File** menu. This causes the Tera Term **Send File** dialog box to open. Navigate to the location of the application Hex file and double click the file to start the transfer.
8. As each hex record is processed, a period (.) is displayed on the console for visual feedback. If an error occurs during the download operation, or during Flash programming, an error code is displayed on the console. The meaning of the error code can be determined by cross-referencing the list of error codes in the `FlashLib.h` header file located in the `<ZDS install>\applications\Inc` folder.
9. If no errors occur, the boot loader prompt is displayed after programming is complete, as shown in Figure 4.

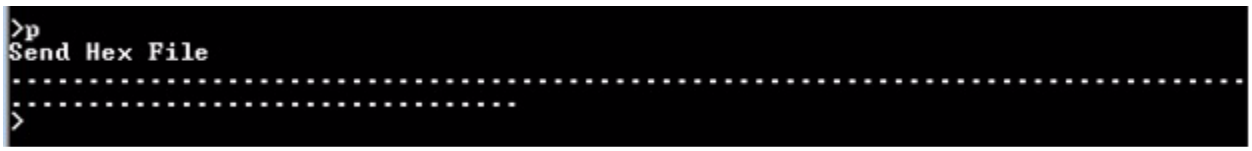

Figure 4. Boot Loader Prompt After Programming is Complete

10. In this instance, the application can be activated by pressing the *s* key in the terminal window, or by pressing the reset button on the target development kit, if available, or by disconnecting and then reconnecting power from the development kit.

Creating an Application Compatible with the eZ80 Boot Loader

Almost any user application with a *Flash* or *Copy to RAM* build configuration can be made compatible with the eZ80 Boot Loader. Typically, the only requirement is to change the application start address to `APPLICATION_START_ADDRESS` and add a linker directive to specify the location of the first-level interrupt vector table.

Projects targeting the eZ80F91 device are not required to add a linker directive specifying the location of the interrupt vector table since it can be located on any 256-byte boundary in the 16MB eZ80 address space. Projects that do not target the eZ80F91 must be built

such that the eZ80 Boot Loader and user application both locate the RAM-based interrupt vector table at the same physical address.

The default configuration of most Zilog sample projects includes at least 128KB of external RAM in the range of `0xB80000` to `0xB9FFFF` (typically there is at least 512KB of RAM in the range of `0xB80000` to `0xBFFFFF`). Consequently, the eZ80 Boot Loader uses the address `0xB80000` as the default location of the RAM-based interrupt vector table when running on all eZ80 devices except the eZ80F91.

For most applications, the base address of external RAM is arbitrary and the chip select setting of external RAM can be relocated to the start of any 64KB region in the 16MB eZ80 address space by modifying the value of the appropriate chip select's lower and upper bound values in the **Configure Target** dialog box, which can be accessed from **Project → Settings** by clicking the **Debugger** heading on the left and then clicking the **Setup** button in the Target section on the right. If the chip select settings used by the application are such that it is impossible to locate external RAM at `0xB80000`, it will be necessary to rebuild the eZ80 Boot Loader to use a compatible ZTGT file and interrupt vector table location as the application. In this instance, see the Modifying the eZ80 Boot Loader to be Compatible with the User Application section for more information.

Use the following procedure to modify the project settings of an existing eZ80 application to make it compatible with the eZ80 Boot Loader.

1. Launch the ZDS II – eZ80Acclaim! 5.3.0 (or later version) IDE.
2. Navigate to the folder containing the user application to be loaded by the eZ80 Boot Loader.
3. In the build configuration pull-down, select an option that is not based on the *All RAM* link configuration. When a project is created, ZDS allows the user to select between several pre-defined build configurations: *Standard*, *Copy to RAM*, and *All RAM*. The Standard and Copy to RAM build configurations assume the project starts running from Flash after POR. The All RAM build configuration is used to download the entire application directly into RAM via ZDS. Because projects that are based on the All Ram build configuration do not target Flash memory, they are not compatible with the eZ80 Boot Loader.
4. From the **Project** menu, select **Settings** and click **Address Spaces** under the Linker heading in the left side of the project settings dialog box.
5. Click in the ROM address space text box displayed on the right side of the project settings dialog box. Change the value of the lower bound of the ROM address space from 0 to the value displayed in Table 1 for the eZ80F9x device. For projects targeting the eZ80L92 microprocessor, use a value of `0x010000` (or `0x008000` for the eZ80L920210 with eZ80L925048MODG) to match the default build of the eZ80 Boot Loader. If the particular external Flash device used with the eZ80L92 device does not have an erase block starting at an offset of 64KB from the beginning of Flash, use the starting offset of the first erase block located at least 16KB from the beginning of external Flash. In this instance, it will be necessary to modify the value of the `APPLICATION_START_ADDR` macro in `eZ80BootLoader.h` and rebuild the eZ80 Boot Loader.

6. If the application is targeting eZ80F91, proceed to the next step. For all other targets, while still in the Project Settings dialog, click **Commands** under the Linker heading. Ensure the **Always Generate From Settings** radio button is selected, and that the **Additional Directives** checkbox is selected, and then click the **Edit...** button. In the Additional Directives dialog box, add the following text after any existing directives: LOCATE IVJMPTBL AT $B80000.
7. Click **OK** to exit the Project Settings dialog. If prompted to rebuild the application, click YES. Select **Rebuild All** from the **Build** menu or click the Rebuild All icon to regenerate the application Hex file.

Note: When rebuilding a project targeting the eZ80F92, eZ80F93, or eZ80L92 devices with the value of APPLICATION_START_ADDR>0xFFFF, the linker will generate a warning message "WARNING(915) ...\vectors16.obj --> Expression range error" because the address of the `__vector_table` used in the `_init_default_vectors` routine is above `0xFFFF`. This warning message can safely be ignored and the project will execute as expected. Alternatively, this warning message can be eliminated by copying the Zilog standard startup files (`vectors16.asm`, `cstartup.asm` and `init_params_xxx.asm`; where `xxx` is either F92, F93 or L92) into the project folder and modifying the instruction used to initialize the 8-bit I register as shown below:

```
;***WARNING(915)***ld a, __vector_table >> 8
ld a, low(__vector_table >> 8)    ;eliminates WARNING(915)
```

If the standard startup files are moved into the application folder, it will also be necessary to click the **Included In Project** radio button in the C Startup Module section of the Objects and Libraries dialog (accessed from the **Project →Settings** menu option by clicking the Object and Libraries option under the Linker category).

8. Note the location of the ZDS-generated Hex file and use the procedure described in the Using the Boot Loader to Download a User Application section to program the application image into Flash on the target.

Modifying the eZ80 Boot Loader to be Compatible with the User Application

If the target HW platform is not an eZ80 development kit and has a unique memory map that will not allow the chip select connected to external RAM to be located at address `0xB80000`, use the following procedure to modify the eZ80 Boot Loader to make it compatible with the user application.

1. Follow steps 1 through 3 in the Installing the eZ80 Boot Loader into Flash Memory on the Target section to open the appropriate eZ80 Boot Loader project in the ZDS IDE.
2. From the **Project** menu, select **Settings** and then click **Debugger** on the left side of the dialog.

3. From the **Target** section on the right side of the dialog, click **Copy**.
4. From the **Source** section of the **Target Copy or Move** dialog, select the **Target File** radio button. To navigate to the user application folder containing the custom ZTGT file that must be used by the eZ80 Boot Loader, click the ellipses button.

5. After navigating to the folder containing the custom ZTGT file, double-click the appropriate target file to select it as the source of the target-copy operation.
6. In the **Destination** section of the **Target Copy or Move** dialog, ensure the **Project Directory** radio button is checked in the **Place Target File In** section, and then click **OK**. This will place a copy of the application's ZTGT file in the Boot Loader directory. The copied target file will appear in the list under the Target Name column.
7. Click the target name copied in the previous step to make it the active target.
8. While still in the Project Settings dialog, click **Address Space** under the Linker heading and modify the location of the RAM address space as appropriate. The Boot Loader project should be configured to use the same RAM address space settings as the application.
9. While still in the Project Settings dialog, click **Commands** under the Linker heading. Ensure the **Always Generate From Settings** radio button is selected and **Additional Directives** is checked and then click the **Edit…** button
10. Modify the address in the LOCATE linker directive to locate the IVJMPTBL at the same address as used by the application, then click **OK** and **OK**.
11. If prompted to rebuild the application, click **YES**; otherwise, select **Rebuild All** from the **Build** menu or click the Rebuild All icon to regenerate the application Hex file.
12. Follow steps 5 through 11 in the Installing the eZ80 Boot Loader into Flash Memory on the Target section to program the modified Boot Loader image into the Flash boot-block of the target device.

eZ80 Boot Loader Menu Options

This section describes the function of each option in the Boot Loader menu. A sample menu is displayed below. The View Info Page (1), Program Info Page (2), and Erase Info Page (3) menu options are only displayed when the eZ80 Boot Loader is running on an eZ80F9x device.

```
Press (key) to activate the corresponding menu option
 (b) Block erase
 (c) Display CFI data
 (d) Display memory
 (e) Erase application Flash memory
 (g) Display Flash Geometry
 (i) Display Flash Information
 (p) Program Hex file
 (r) Reboot
 (s) Start application
```

```
 (w) Write memory
 (1) View Info Page
 (2) Program Info Page
 (3) Erase Info Page
 (?) Display this help menu

>
```

The eZ80 Boot Loader displays the prompt character > whenever it is waiting for the operator to select one of the menu options. To activate a menu option, press the key corresponding to the letter in parenthesis that prefixes each menu option. For example, to display the contents of memory, press the *d* key in the terminal program.

After the target comes out of reset, the eZ80 Boot Loader takes control of the system and waits for the user to press the *z* key in the terminal program for approximately 250 milliseconds. If the user presses the *z* key before the timeout, then the Boot Loader menu is displayed; otherwise, the eZ80 Boot Loader checks to see if a user application has been programmed into Flash. See the Start Application (s) section for more information. If the user application has not been programmed into Flash, the Boot Loader menu is displayed.

Block Erase (b)

The Block Erase menu command can be used to erase a block (also called a page, sector, or erase-block) of external or internal Flash memory as determined by the target address range. This command cannot be used to erase internal or external RAM or the Flash Information Page available on eZ80F9x series devices.

After selecting the erase block menu option, the user is prompted to enter an address corresponding to the block(s) to be erased and the number of consecutive blocks to erase starting with the block containing the target address. Values entered must be in hexadecimal format, as shown below.

```
>b
Enter an address within any Flash block to erase the entire block
1b000f
Enter the number of consecutive blocks to erase (in hexadecimal)
(press 0 or <Enter> to abort)
4
....
>
```

Typically, it is not necessary to enter an address that corresponds to the beginning of an erase block in external Flash. Most external Flash devices ignore the least significant (offset) bits in the command sequence used to erase the block. Consequently, any address within a block can be specified to erase the entire block. Similarly, when erasing internal Flash, any address within a page can be specified to erase the entire page of internal Flash memory. If the specified address does not reside within internal or external Flash, the eZ80 Boot Loader displays an error code (see `FlashLib.h`) and aborts the erase operation.

Entering a value of 0 when prompted for the number of blocks to erase causes the eZ80 Boot Loader to abort the operation without erasing any Flash memory. If the number of blocks to erase exceeds the number of erase blocks from the start of the block containing the target address to the end of internal or external Flash, the eZ80 Boot Loader will erase all blocks from the start of the block containing the specified address until the end of Flash and then return an error (see `FlashLib.h`) indicating that the remaining blocks could not be erased. Although this command can be used to erase either blocks of internal or external Flash, it cannot be used to erase contiguous blocks that span between internal and external Flash.

Display CFI Data (c)

The Display CFI Data menu command is used to display data from the Common Flash Interface (CFI) query table that resides within a special region of CFI-compliant Flash devices. The structure of the query table data is described in the Common Flash Interface Specification.

After selecting the CFI query menu option, the user is prompted to enter the starting hexadecimal offset within the CFI query table and number of bytes of data to be displayed (in hexadecimal). Up to `0x100` (256 decimal) bytes of information from the CFI query table can be displayed. If the user enters a value of 0 when prompted for the number of bytes to display, nothing is displayed. If the user enters a number larger than `0x100` when prompted for the number of bytes to display, the output is truncated to 256 bytes.

The example below shows the query table information obtained from a Spansion S29GL064N Flash that includes the CFI identification string, the system interface information, and the device geometry information. For an explanation of the values returned, refer to the CFI specification:

```
>c
Enter staring CFI offset (in hexadecimal): 10
Enter number of bytes to display (in hexadecimal, max 100): 25

000010: 51 52 59 02 00 40 00 00 - 00 00 00 27 36 00 00 07
QRY..@.....'6...
000020: 07 0a 00 03 05 04 00 17 - 02 00 05 00 02 07 00
20..............
000030: 00 7e 00 00 01

>
```

If this command is issued on a non-CFI compliant Flash, the eZ80 Boot Loader displays an error code (see `FlashLib.h`).

```
>c
Enter starting CFI offset (in hexadecimal): 10
Enter number of bytes to display (in hexadecimal, max 100): 25

Status -9

>
```

Display Memory (d)

The display memory command is used to display the contents of a range of internal or external RAM or Flash memory. After selecting the display memory command the user is prompted to enter the starting hexadecimal address of the memory to be displayed and the number of bytes to be displayed (in hexadecimal), as shown in the following example.

```
>d
Enter starting address (in hexadecimal): 800
Enter number of bytes to display (in hexadecimal): 40

000800: 0f fc dd 36 ff 00 cd cf - 19 b8 dd 77 fb fe 0d
20...6.......w...
000810: 1e 01 62 1d b8 ed 43 58 - 40 b8 01 6b 39 b8 c5 cd
..b...CX@..k9...
000820: 9e 1b b8 c1 01 36 1e b8 - ed 43 58 40 b8 18 7c dd
.....6...CX@..|.
000830: 7e ff dd be 09 20 0c 01 - 07 00 00 c5 cd 22 1a b8
~..........."..

>
```

This command can be used to display the contents of contiguous memory blocks of different types of memory. For example, if the last byte of internal Flash resides at `0x3FFF` and the first byte of external RAM is located at `0x40000`, reading `0x20` bytes of data from `0x3FFF0` will display 16 bytes of internal Flash and 16 bytes of external RAM.

Attempting to display the contents of memory ranges that are not mapped to physical RAM or Flash will return meaningless data. Typically, the same data value will be displayed for each memory location in the (invalid) range as shown below.

```
>d
Enter starting address (in hexadecimal): f00000
Enter number of bytes to display (in hexadecimal): 40

F00000: b7 b7 b7 b7 b7 b7 b7 b7 - b7 b7 b7 b7 b7 b7 b7 b7
................
F00010: b7 b7 b7 b7 b7 b7 b7 b7 - b7 b7 b7 b7 b7 b7 b7 b7
................
F00020: b7 b7 b7 b7 b7 b7 b7 b7 - b7 b7 b7 b7 b7 b7 b7 b7
................
F00030: b7 b7 b7 b7 b7 b7 b7 b7 - b7 b7 b7 b7 b7 b7 b7 b7
................

>
```

Erase Application Flash Memory (e)

As described in the Memory Map section of this document, the application can be loaded into external and/or internal Flash depending on the value of the `APPLICATION_START_ADDR` macro and the size of the user application. This menu option is used to erase internal and/or external Flash from `APPLICATION_START_ADDR`

to the end of external Flash. The only region of Flash not erased by this command is the region reserved for use by the eZ80 Boot Loader (between 0x00 and APPLICATION_START_ADDR – 1). On the eZ80F9x series of devices, this menu option does not erase the contents of the Flash Info Page.

The Erase Application Flash Memory command is typically issued prior to programming Flash with a new user application. The eZ80 Boot Loader will not be able to program a new application image into Flash unless the Flash memory occupied by the new application image is completely erased prior to programming.

After selecting the Erase Application Flash Memory option, the user is prompted to confirm that the application in Flash memory should be erased. If the user presses the *y* key in response, the eZ80 Boot Loader erases the application area of internal and/or external Flash memory. If the user presses any other key, nothing is erased. As each block of internal and/or external Flash memory is erased, the eZ80 Boot Loader displays a period to provide visual feedback to the user during the erase operation, as shown in the following example:

```
>e
Do you want to completely erase the application Flash?
   Press y to confirm, any other key aborts
Erasing application Flash.............................

>
```

Display Flash Geometry (g)

The Display Flash Geometry menu option can be used to display the memory location of each erase block in internal Flash memory (only applicable to the eZ80F9x series) and/or external Flash memory. This information can be used with the Erase Flash Block (b) menu command to selectively erase contiguous blocks of Flash. An example of the geometry information displayed by this command on an eZ80F92 device with a 1MB external Flash is shown below (internal Flash pages 11 through 119 deleted for brevity).

```
>g

            Internal Flash Geometry
            -----------------------
 Page 0   Address Range: 000000 - 0003FF
 Page 1   Address Range: 000400 - 0007FF
 Page 2   Address Range: 000800 - 000BFF
 Page 3   Address Range: 000C00 - 000FFF
 Page 4   Address Range: 001000 - 0013FF
 Page 5   Address Range: 001400 - 0017FF
 Page 6   Address Range: 001800 - 001BFF
 Page 7   Address Range: 001C00 - 001FFF
 Page 8   Address Range: 002000 - 0023FF
 Page 9   Address Range: 002400 - 0027FF
 Page 10 Address Range: 002800 - 002BFF

 ...
```

```
 Page 120 Address Range: 01E000 - 01E3FF
 Page 121 Address Range: 01E400 - 01E7FF
 Page 122 Address Range: 01E800 - 01EBFF
 Page 123 Address Range: 01EC00 - 01EFFF
 Page 124 Address Range: 01F000 - 01F3FF
 Page 125 Address Range: 01F400 - 01F7FF
 Page 126 Address Range: 01F800 - 01FBFF
 Page 127 Address Range: 01FC00 - 01FFFF


            External Flash Geometry
            -----------------------
Region 0
  Block 0   Address Range: 100000 - 103FFF
Region 1
  Block 0   Address Range: 104000 - 105FFF
  Block 1   Address Range: 106000 - 107FFF
Region 2
  Block 0   Address Range: 108000 - 11FFFF
Region 3
  Block 0   Address Range: 120000 - 13FFFF
  Block 1   Address Range: 140000 - 15FFFF
  Block 2   Address Range: 160000 - 17FFFF
  Block 3   Address Range: 180000 - 19FFFF
  Block 4   Address Range: 1A0000 - 1BFFFF
  Block 5   Address Range: 1C0000 - 1DFFFF
  Block 6   Address Range: 1E0000 - 1FFFFF
>
```

Display Flash Information (i)

The Display Flash Information menu option is used to display information about internal Flash memory (only applicable to the eZ80F9x series) and/or external Flash memory and the expected starting address of the application (`APPLICATION_START_ADDR`), as shown in the following example.

```
>i

 Internal Flash Information
 ==========================
 Row size:     256
 Page size:    2048
 Total pages:  128
 Blocks:       8
 Flash Size:   262144 (256 KB)

External Flash Information
 ==========================
 Chip Select:     CS0
 CS Base Address: 100000
```

```
 CS End Address: 7FFFFF
 CFI-compliant:   YES
 Manufacturer ID: 01
 Device ID:       7E
 Flash Size:      800000 (8 MB)

Application start address: 008000

>
```

The internal Flash information is only displayed for eZ80F9x series devices. The eZ80 Boot Loader can be recompiled with the definition of the SHOW_IFL_INFO macro commented out. This will prevent internal Flash information from being displayed on all eZ80F9x devices.

Applications loaded into Flash memory using the program Hex file menu option (p) must be compiled such that the application's reset vector is located at the Application start address (APPLICATION_START_ADDR) displayed by the Display Flash Information (i) command. On eZ80F9x series devices, the application start address will be within internal Flash at the start of the first page of Flash not reserved for use by the eZ80 Boot Loader. However, the eZ80 Boot Loader can be recompiled with the value assigned to the APPLICATION_START_ADDR macro modified such that the application will reside entirely within external Flash memory, allowing the application to use the unreserved region of internal Flash for some other purpose.

The eZ80 Boot Loader always configures CS0 such that external Flash is located at a base address of 0xx0 (eZ80L92) or 0x100000 (eZ80F9x series) for compatibility with the default ZDS target file settings. The upper bound of CS0 is set 7MB above the CS0 base address. However, the actual address range of external Flash will depend on the particular external Flash device used. The external Flash device in the previous example was 8MB in size; therefore, the default CS0 settings will only allow the first 7MB of external Flash to be accessed. If the external Flash was 1MB in size, the address between 0x100000 and 0x1FFFFF corresponds to memory locations within external Flash, and the remainder of the CS0 address space between 0x200000 and 0x7FFFFF is not usable.

Program Hex File (p)

The Program Hex File menu option is used to program an application image into internal and/or external Flash memory from a file in Intel hex format. The Hex file is generated by ZDS when the Build or Rebuild All menu options or icons are clicked from within the IDE.

Prior to using the Program Hex File menu option, Flash memory targeted by the Hex file must be erased (all bytes in the application memory range must be 0xFF). Typically, this is accomplished by using the Erase Application Flash Memory (e) or the Block Erase (b) menu options.

After selecting the Program Hex File menu option, a message instructing the user to send the hex file using the terminal program is displayed. Refer to the terminal program's documentation for instructions on how to transfer ASCII text files. When sending the hex file,

do not use any transfer protocols such as XMODEM or Kermit. It is not necessary to enable flow control while transferring the Hex file.

If an error occurs, the program operation is aborted and the eZ80 Boot Loader will display an error code in the terminal window. Otherwise, after the transfer completes, the eZ80 Boot Loader prompt appears, indicating that the transfer completed. As each data record in the hex file is processed, a period (.) character is displayed in the terminal window to provide visual feedback to the operator. A sample of the output generated by the Program Hex File menu option is displayed below.

```
>p
Send Hex File
.....................................................................
.........................................
>
```

Reboot (r)

The Reboot menu option causes the eZ80 CPU to jump to the reset vector at Flash address 0x0. Although this does not reinitialize eZ80 peripherals, it effectively causes a SW reset of the system, forcing the boot loader to go through its POR reset sequence.

When the eZ80 Boot Loader startup code begins executing, it waits for the user to press the *z* key in the terminal window for approximately 250ms. If the user presses the *z* key before the timeout, the eZ80 Boot Loader main menu is displayed; otherwise, the eZ80 Boot Loader reads the first three bytes of the application program located at APPLICATION_START_ADDR. If these bytes are 0xFFFFFF, the eZ80 Boot Loader main menu is displayed; otherwise, the eZ80 Boot Loader transfers control of the CPU to the user application, as shown in the following example.

```
>r
Rebooting...Activating user application...ø

 Zilog Development Board Demonstration Program (1.0)
        Featuring the eZ80F91

 Type a character and see it appear on the LED Matrix:
```

Start Application (s)

This option is used to begin execution of the user application stored in Flash. Before activating the user application, the Boot Loader determines whether the application has been programmed into Flash Memory at the application start address (APPLICATION_START_ADDR) shown in Table 1. If the first 3 bytes of Flash located at APPLICATION_START_ADDR are all 0xFF, the application has not been programmed into Flash memory. In this instance, the eZ80 Boot Loader will display an error message and the prompt character to indicate it is waiting for the operator to enter another command, as shown below.

```
>s
Status -1
```

```
>
```

If the user application has been programmed into Flash, the first 3 bytes of the application reset handler will not contain the value 0xFFFFFF. In this instance, the eZ80 Boot Loader displays a message indicating that the user application is being activated and control is passed to the application reset handler located at APPLICATION_START_ADDR.

```
>s
Activating user application...
```

Prior to activating the user application on the eZ80F9x series of devices, the eZ80 Boot Loader sets the Flash Protection register to 0xFF (write-protecting all blocks of internal Flash) and sets the FLASH_PAGE register to 0.

Write Memory (w)

The Write Memory menu option is used to modify the contents of internal and/or external Flash, or RAM, depending on the location of the target memory range. After selecting this menu option, the user is prompted to enter the hexadecimal address of the first byte to be modified and up to 16 bytes (32 hexadecimal characters) of data to write into consecutive memory locations starting with the specified address. An example of the Write Memory command that modifies the contents of RAM address between 0xB90010 and 0xB90013 is shown below.

```
>d
Enter starting address (in hexadecimal): b90000
Enter number of bytes to display (in hexadecimal): 40

B90000: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
................
B90010: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
................
B90020: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
................
B90030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
................

>w
Enter starting address (in hexadecimal): b90010
Enter up to 16 bytes of Hexadecimal data
11223344

>d
Enter starting address (in hexadecimal): b90000
Enter number of bytes to display (in hexadecimal): 40

B90000: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
................
B90010: 11 22 33 44 00 00 00 00 - 00 00 00 00 00 00 00 00
."3D............
```

```
B90020: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
................
B90030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
................

>
```

If this command is used to program the contents of Flash memory, the range of memory locations targeted by this command must have previously been erased (all bytes `0xFF`). Otherwise, the write to memory operation will typically fail and an error code is displayed on the console. Refer to the `FlashLib.h` header file for a description of the error code.

Although this command can be used to modify the contents of RAM or Flash, it should not be used to modify different types of contiguous memory. For example, if internal Flash memory is located between `0x0` and `0x03FFFF` and external Flash memory begins at `0x040000`, attempting to use the write memory command to modify 8 consecutive bytes of memory starting at `0x03FFFC` will cause the last 4 bytes of internal Flash to be programmed (if initially erased) and may cause unpredictable results when attempting to write to the first 4 bytes of external Flash.

View Info Page (1)

The view information page menu option is only available when the eZ80 Boot Loader is running on a eZ80F9x series microcontroller. This menu command is used to display a subset of the data stored in the eZ80F9x Flash Information Page.

After selecting this menu command the user is prompted to enter a starting hexadecimal address within the Information Page and the number of bytes to be displayed (in hexadecimal), as shown in the following example:

```
>1
Enter starting address (in hexadecimal): 0
Enter number of bytes to display (in hexadecimal): 20

000000: 00 90 23 00 01 01 01 ac - 10 06 ba ff ff ff 00 ac
..#.............
000010: 10 06 01 00 ff ff ff ff - ff ff ff ff ff ff ff ff
................
```

Some application programs (including the Zilog TCP/IP Software Suite, ZTP) use the eZ80F9x Flash Information Page to store persistent parameter data, independently of the application. This allows the application's operating characteristics to be modified without having to reprogram the entire user application into Flash.

Attempting to read Information Page data from an address not within the Information Page will return meaningless values.

Program Info Page (2)

The program information page menu option is only available when the eZ80 Boot Loader is running on an eZ80F9x series microcontroller. This menu command is used to modify a subset of the data stored in the eZ80F9x Flash Information Page.

This menu option cannot be used to change bits in the programmed state (binary value of 0) to the erased state (binary value 1). Consequently, prior to programming new values into the Flash Information Page, it is recommended to use the Erase Info Page (3) menu option to erase data already programmed into the Information page.

After selecting this menu option, the user is prompted to enter the hexadecimal address of the first byte to be modified and up to 16 bytes (32 hexadecimal characters) of data to write into consecutive Information Page memory locations starting with the specified address. An example of the program information page command is shown below.

```
>1
Enter starting address (in hexadecimal): 0
Enter number of bytes to display (in hexadecimal): 20

000000: ff ff ff ff ff ff ff ff - ff ff ff ff ff ff ff ff
................
000010: ff ff ff ff ff ff ff ff - ff ff ff ff ff ff ff ff
................
>2
Enter starting address (in hexadecimal): 0
Enter up to 16 bytes of Hexadecimal data
11223344

>1
Enter starting address (in hexadecimal): 0
Enter number of bytes to display (in hexadecimal): 20

000000: 11 22 33 44 ff ff ff ff - ff ff ff ff ff ff ff ff
................
000010: ff ff ff ff ff ff ff ff - ff ff ff ff ff ff ff ff
................
>
```

Erase Info Page (3)

The Erase Information Page menu option is only available when the eZ80 Boot Loader is running on an eZ80F9x series microcontroller. This menu command is used to erase all data stored in the eZ80F9x Flash Information Page. Consequently, when this command finishes, all memory locations in the Information Page contain the value `0xFF`.

After selecting this menu option, the user is prompted to confirm that the contents of the Information Page should be deleted. If the *y* key is pressed in response, the eZ80 Boot Loader will proceed to erase the eZ80F9x Flash Information Page; otherwise, nothing is erased, as shown below.

```
>3
Do you want to completely erase the Information Page?
   Press y to confirm, any other key aborts
Nothing erased

>
```

Customizing the eZ80 Boot Loader

The eZ80 Boot Loader can be customized to modify its default operating behavior. Typically, this involves modifying a macro definition and then rebuilding the eZ80Boot Loader as described in the Installing the eZ80 Boot Loader into Flash Memory on the Target section on page 9. The remainder of this section describes each of the definitions that the user may need to modify.

Target Chip Select

The default build of the eZ80 Boot Loader assumes that CS0 is connected to the external Flash device of interest. If external Flash is connected to a different chip select, the value of the `EXT_FLASH_CHIP_SELECT` macro definition (defined in `eZ80BootLoader.h`) needs to be modified. For example, if the external Flash device is connected to CS2, the `EXT_FLASH_CHIP_SELECT` macro should be modified as shown below.

```
#define EXT_FLASH_CHIP_SELECT          2
```

Application Start Address

The application the eZ80 Boot Loader programs into Flash memory must be compiled with a starting address that matches the value of the APPLICATION_START_ADDR macro defined in eZ80BootLoader.h. The default value assigned to the APPLICATION_START_ADDR macro is determined based on which member of the eZ80 family of microcontrollers and microprocessors is targeted by the project settings, as shown below.

```
#ifdef _EZ80F91
   #define APPLICATION_START_ADDR      0x008000
#endif
#if( defined(_EZ80F92) || defined(_EZ80F93) )
   #define APPLICATION_START_ADDR      0x004000
#endif
#ifdef EZ80L925048MOD
   #define APPLICATION_START_ADDR      0x008000
#endif
#ifdef EZ80L925148MODG
   #define APPLICATION_START_ADDR      0x010000
#endif
```

If necessary, the application start address used by the boot loader can be modified to match application requirements. For proper operation of the eZ80 Boot Loader, the value assigned to the APPLICATION_START_ADDR macro must correspond to the start of an internal or external block of Flash.

Erasing the Boot Loader

By default, the Boot Loader will not allow the user to erase Flash memory locations where the Boot Loader image is stored. To allow the user to erase the area of Flash used to store the Boot Loader application, uncomment the definition of the ALLOW_BOOT_LOADER_ERASE macro definition in the eZ80BootLoader.h header file, as shown below.

```
#define ALLOW_BOOT_LOADER_ERASE
```

After the macro definition is uncommented, the block erase menu option (b) can be used to erase the portion of Flash memory containing the eZ80 Boot Loader. When prompted, enter a starting address of 0. Since the size of the boot loader is just under 16KB, the number of blocks of Flash to erase depends on the size of the erase block starting at address 0. On the eZ80F91, internal pages of Flash memory are all 2048 bytes long, requiring 8 pages of eZ80F91 internal Flash to be erased. For the eZ80F92 and eZ80F93 devices, each page of internal Flash memory is 1024 bytes long, requiring 16 pages of Flash to be erased. On the eZ80L92, it is necessary to refer to the data sheet of the Flash device on CS0 to determine the number of blocks of external Flash that must be erased.

UART0 or UART1 Selection

By default, the eZ80 Boot Loader uses UART0 for communications with the PC running the terminal program. If necessary, the eZ80 Boot Loader can be rebuilt to use UART1 by commenting out the USE_UART0 macro definition in `UART.h` and un-commenting the `USE_UART1` macro, as shown below.

```
//#define USE_UART0
#define USE_UART1
```

UART Baud Rate

By default, the eZ80 Boot Loader uses a UART baud rate of 57.6kbps. To reduce the time required to program large application images using the Program Hex File menu option, the value of the `UART_BAUD` macro can be modified (defined in `UART.h`). However, note that the maximum baud rate that can be supported by the target device is dependent on the operating frequency of the device.

```
#define UART_BAUD                       57600
```

Enabling Access to the eZ80F9x Information Page

By default, when the eZ80 Boot Loader is running on an eZ80F9x device, the display information ('i') and display geometry ('g') menu options will show information pertaining to eZ80F9x internal Flash as well as external Flash on CS0. This increases the size of the eZ80F9x boot Loader. To reduce the size of the eZ80F9x Boot Loader, the definition of the SHOW_IFL_INFO macro can be commented out in `eZ80BootLoader.h`, as shown below.

```
//#define SHOW_IFL_INFO
```

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

 Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

eZ80Acclaim! and eZ80Acclaim Plus! are trademarks or registered trademarks of Zilog, Inc. All other product or service names are the property of their respective owners.

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base or consider participating in the Zilog Forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.