Application Note

Flash Loader Utility for the Z8 Encore! XP® MCU

AN011806-0408

Abstract

This application note describes Flash Loader utility for the Zilog's Z8 Encore! XP® MCU that can be operated through the UART0 console. The goal of this document is to provide a tool for program On-Chip Flash memory on the Z8 Encore! XP MCU using the Flash Loader and/or application software (firmware).

This Flash Loader implementation expects an Intel hexadecimal file (IHX file) to be created and uploaded by Zilog Developer Studio II (ZDS II). No other tools are required to convert output files to another format. Access all function modules and integrate a different method to upload IHX files or read/write bytes to Flash memory.

Note: *The source code file associated with this application note, AN0118-SC01 is available for download on* www.zilog.com.

Z8 Encore! XP® Flash Microcontrollers

Zilog's Z8 Encore! XP products are based on the new eZ8™ CPU and introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows for faster development time and program changes in the field. The high-performance register-to-register based architecture of the eZ8 core maintains backward compatibility with Zilog's popular Z8 MCU.

Z8 Encore! XP MCUs combine a 20 MHz core with Flash memory, linear-register SRAM, and an extensive array of On-Chip peripherals. These peripherals make the Z8 Encore! XP MCU suitable for a variety of applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

General Overview

For all code development and debugging, the Z8F Debug Port (OCD) is used in conjunction with ZDS II. However, there might be a situation when some bytes contained in Flash memory must be read from or written to user application code when the core is running. Flash Loader comes to the rescue in such a situation.

The Flash Loader utility runs in On-Chip Flash to self-program Flash memory. Flash Loader is designed to operate (compile, link, and load) in the ZDS II environment. The Flash Loader application is loaded via On-Chip Debug port (Target Interface Module) into Flash memory on the Z8 Encore! XP MCU Development Board.

Flash Loader physically resides in Flash memory. Access the functions of Flash Loader using a standard dumb terminal program (HyperTerminal). When Flash Loader is first launched, a menu appears allowing to select from a set of choices. See Figure 1 on page 5 and Figure 2 on page 5, respectively.

Apart from downloading IHX files into the user application space within Flash memory, Flash Loader allows user Reads and Writes to the RAM and Flash memory locations using the HyperTerminal program. Loading code via Flash Loader through the ZDS II/ Target Interface Module is necessary only once. Flash Loader is not ensured to be write-protected; however, while downloading user application code, it does ensure that the application code is not overwrite the memory space occupied by the Flash Loader.

Discussion

This section describes basic operation of the Flash Loader, including its multiple software layers. For a complete listing of all functions and source code associated with the Flash Loader application, refer to the source code file *AN0118-SC01 available for download at* www.zilog.com.

Theory of Operation

The Flash Loader can be used to program a user's application software into a Flash device via HyperTerminal interface, eliminating the requirement for ZDS II to be installed on the host machine. Though this scenario does not apply when developing an application, it is convenient when the user code is ready and must be uploaded to Flash through a peripheral device such as a modem, an IrDA device, or HyperTerminal.

If developing application software within ZDS II, use of the Flash Loader program is not necessary. Application code can be directly uploaded into the Z8 Encore! XP MCU's Flash memory space using ZDS II and the Target Interface Module using the OCD port.

When the Flash device contains no data, the Flash Loader utility should be used to program Flash through ZDS II and the OCD port. After the Flash Loader program is executed, click the **Reset+Go** button on the ZDS II toolbar or click the **Reset** button on the Evaluation board. New application code can be loaded on top of the Boot Loader using the ASCII (text) file transfer mode of the DTE. The DTE must be connected to the Z8 Encore! XP UART0. The new application code is sent as an Intel IHX file over this link.

Developing the Flash Loader Application for the Z8 Encore! XP® MCU

The Boot Loader starts at power-up and is forced 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 defines the start of the application code. When executing a Build, the application code must account for the start address that the Boot Loader defines for the application code.

The Flash Loader utility features two basic modes of operation: FLASH LOADER mode and USER APPLICATION mode. Both are discussed below.

Flash Loader Mode

FLASH LOADER mode allows to examine Flash and RAM locations, read and write into Flash and RAM, and upload user application code into the area of memory space occupied by the user application. See Appendix A—Project Settings on page 10 for Flash Loader project settings and Appendix B—User Application Code Generation on page 13 for user code relocation.

Upon power-up or reset, the Flash Loader utility waits for the special sequence key (in this application, the *space* character on the keyboard) via console port (UART0). If the user inputs the special sequence key from the keyboard at the time of Hardware Reset, the Flash Boot Loader operate in FLASH LOADER mode; otherwise the Flash Loader operates by default in USER APPLICATION mode.

User Application Mode

This mode is the default mode on power-up, wherein the Flash Loader transfers program control to the user application. In this mode, the user application starts running the application code residing in the application space (in this case, `4000h` and above), unless the special sequence key is pressed. See Appendix A—Project Settings on page 10 for Flash

Loader project settings and Appendix B—User Application Code Generation on page 13 for user code relocation.

The user can move from USER APPLICATION mode to FLASH LOADER mode by pressing the special sequence key and pressing the **Reset** button on the evaluation board.

Flash Loader

The Flash Loader features the following layers for implementing code:

Startup Code

This piece of code is the default code used to bring the device to a known state, then relinquish control over to the start of the main program.

Serial Port Initialization

This layer initializes the serial port (the console port) for the Flash Loader.

Flash Read/Write Utilities

This layer provides basic Flash and RAM Read/Write capabilities and constitutes the core layer of the Flash Loader utility.

File Parser, Error Checker, and Flash Loader

This layer receives the Intel Hex file frame by frame, checks each frame, and processes each frame according to its type. The frame start address is derived and the frame data is directly programmed into Flash.

User Interface Layer

This layer provides for user interaction with the Flash Loader via HyperTerminal. Because this layer uses standard libraries such as `printf` and `scanf`, it constitutes the bulk of the Flash Loader code.

The Flash Loader uses UART0 to communicate via HyperTerminal console and works with the following default configurations:

- UART0 (console port on the Z8 Encore! XP Evaluation Board)
- 9600 bps
- 8-N-1
- Flow Control = none

The console port can be extended to a peripheral such as UART1, I^2C, GPIO, Modem, or IrDA. In other words, Z8 Encore! XP MCU Flash memory can be programmed remotely using a modem or by an IrDA device (for example, a Palm Pilot) that beams its signal.

Table 1 lists the memory locations in which user application code and the Flash Loader reside on the Z8 Encore! XP F64XX Series MCU.

Table 1. Flash Loader/User Application Memory Map

Memory Location	Description
0000h-0001h	Flash Option Bits
0002h-0007h	Reset/WDT/Trap Vectors
0008h-0037h	Interrupt Vectors
0038h-01FFh	Reserved
0200h-3FFFh	Flash Loader
4000h-FFFFh	User application code

Flash Loader starts on a fresh 512 byte boundary, that is, address `200h` and above. The application code area occupies the address range `4000h` to the 64 K boundary at `FFFFh`. The Flash address range `38h–1FFh` is assumed to be reserved by the Flash Loader. See Appendix A—Project Settings on page 10 for Flash Loader project settings and Appendix B—User Application Code Generation on page 13 for user code relocation.

Metrics

The Flash Boot Loader requires about 4 KB of Flash and program space and under 1 KB of RAM. The code is written in C for easy maintenance and portability, or for integration into another method of Flash programming.

Flash Loader Code Features

The program (Flash) and data (RAM) sizes for each layer are computed based on the following ZDS II settings:

- Large memory model
- Disable optimizations
- Generate no debug information
- Store constant variables in ROM
- Alias checking turned ON
- Use intrinsics
- Use a dynamic model

Table 2 lists the sizes of the Flash Loader code layers.

Table 2. Flash Loader Code Layers

Description	Size in Bytes
Flash Loader application code	4800
Main program + User Interface + Standard Libraries	11,200

As shown in Table 2, the user interface and the standard libraries occupy most of the code space. However, this code space can be reduced if the user chooses to write a slimmed-down version of the standard libraries.

When excluding the standard libraries and the user interface, the Flash Loader requires around 4 KB of Flash and 1 KB of RAM. This reduction can be further optimized by observing the small memory model and minimizing the code size using via ZDS II settings.

Testing

The Flash Loader program discussed in this application note was tested successfully with the Z8 Encore! XP Z8F642 MCU Development Kit (Z8F64200100KIT). ZDS II for Z8 Encore! family of microcontrollers is used to build the Flash Loader. With this setup, IHX file downloads can be efficiently executed.

Equipment Used

- Z8 Encore! XP Z8F642 MCU Development Kit (Z8F64200100KIT)
- PC with Hyper Terminal set to 9600 baud, 8-bit, no parity, 1 stop bit

Procedure

Follow the steps below to test the Flash Loader application.

1. Download and extract the `AN0118-SC01.zip` file.
2. Build the development environment (ZDS II, Z8 Encore! XP Evaluation board, Target Interface Module, and serial cable connecting the ZDS II port on the host to the target debug board).
3. Load the Flash Loader code into Flash memory on the Z8F642 device.
4. Disconnect ZDS II and the Target Interface Module from the Evaluation board.
5. On the second PC (or on the same PC if it has 2 COM ports), launch and configure the HyperTerminal program to reflect settings of 9600 bps, 8-N-1, and no flow control.
6. Connect a serial cable between the HyperTerminal port of the second PC (or the second COM port of the first PC) and the Console port

(UART0) of the Z8 Encore! XP® Evaluation Board.

7. Press the space key on the second PC and press the **Reset** button on the Evaluation board to see the Boot Loader prompt in HyperTerminal.

Figure 1 displays the initial HyperTerminal screen.

8. Press the H key to verify all of the items in the Flash Loader help menu, displayed in Figure 2.

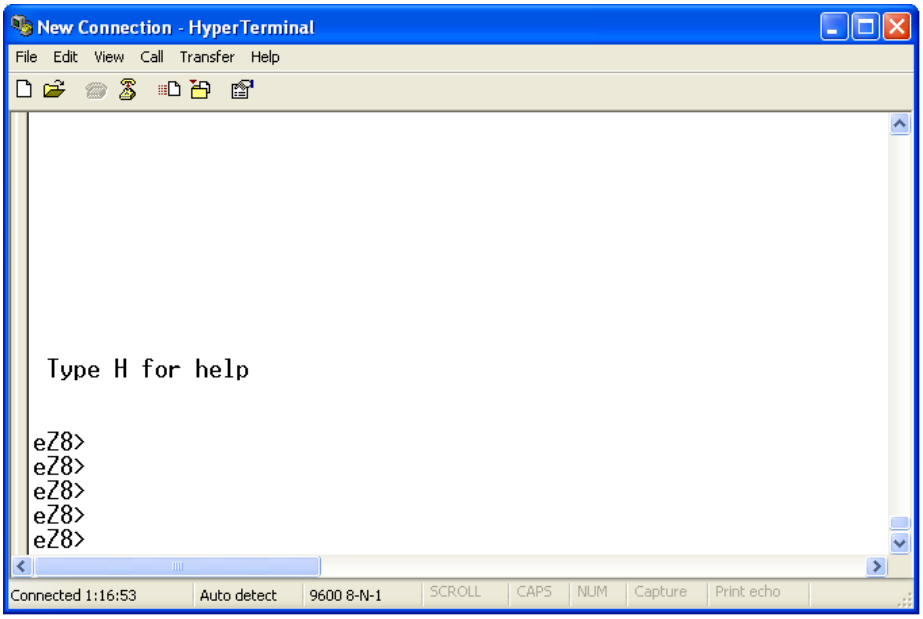

Figure 1. Initial Flash Loader Screen

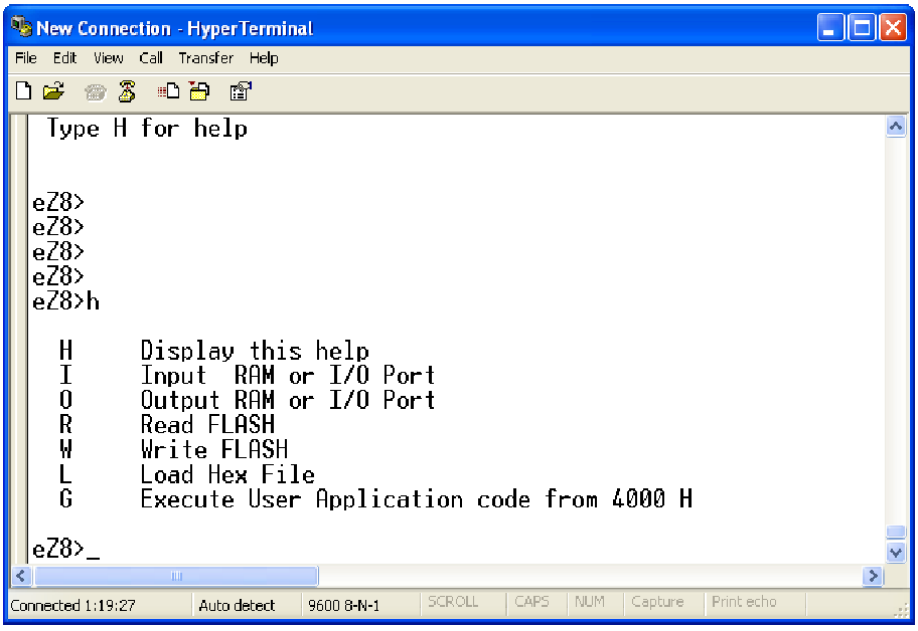

Figure 2. Flash Loader Screen Displays Help Menu

9. Examine the RAM and Flash Read/Write settings in Figure 3, Figure 4, and Figure 5. Ensure that Writes occur only in the application code space (4000h–FFFFh).

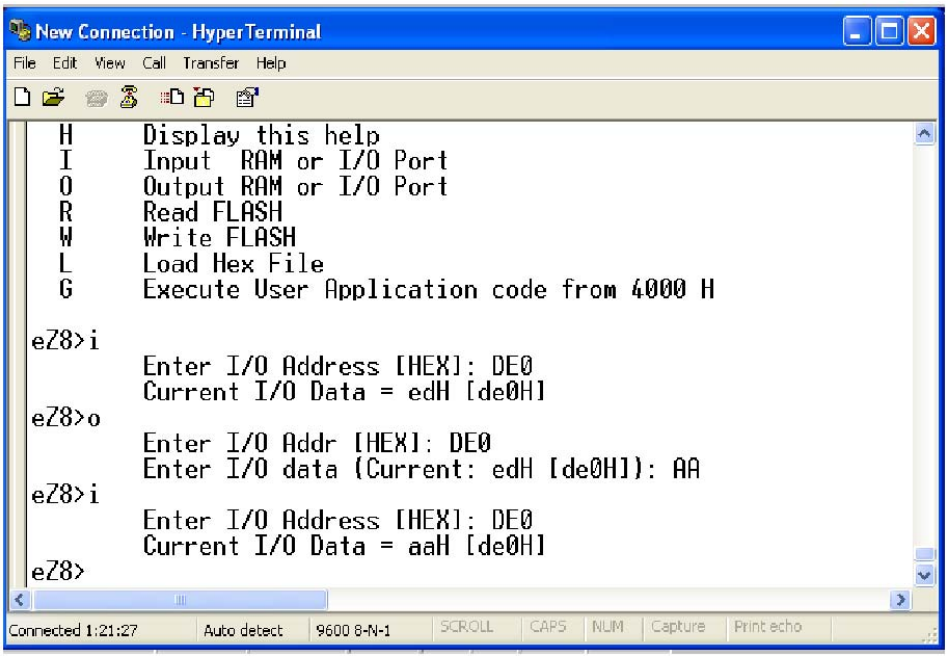

Figure 3. Flash Loader Screen Displays RAM Read/Write Capability

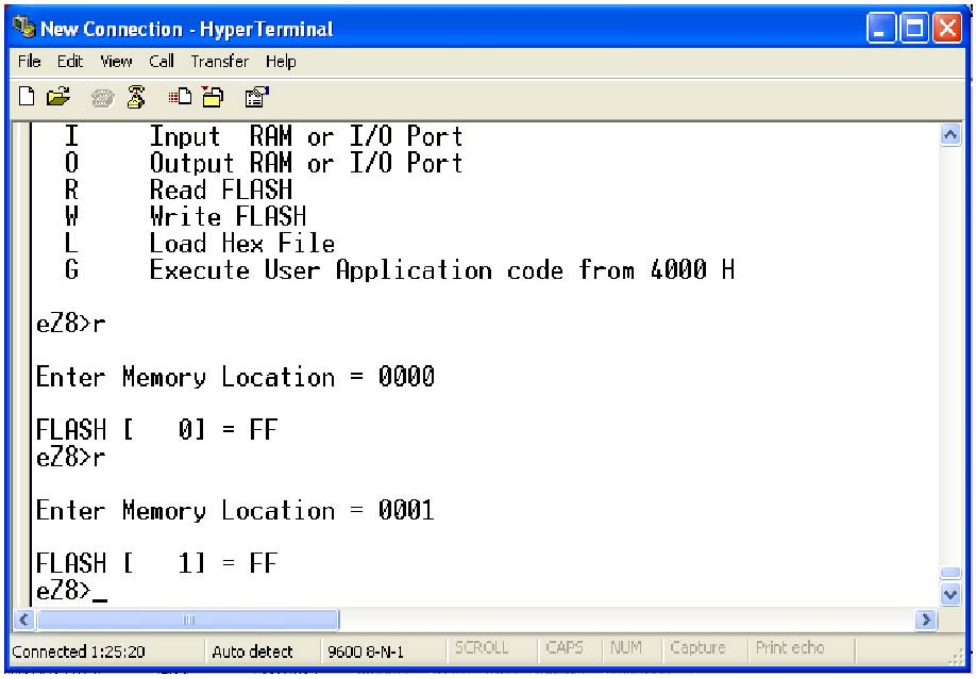

Figure 4. Flash Loader Screen Displays Flash Read Capability

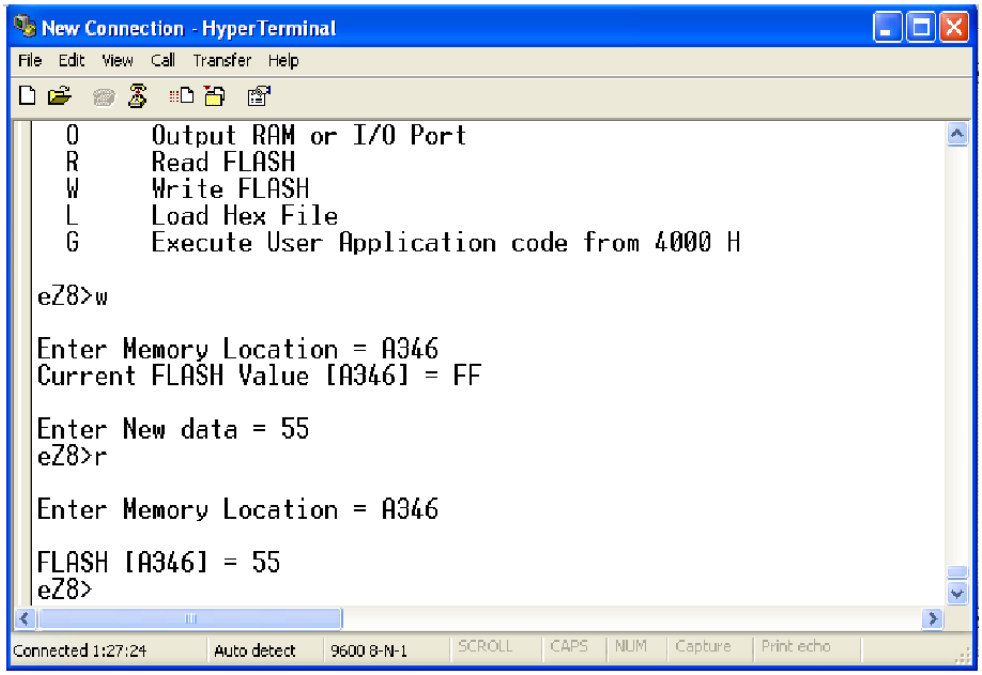

Figure 5. Flash Loader Screen Displays Flash Write Capability

Downloading Application Code Using the Flash Loader Utility

Press the L key to load the hex file, as displayed in Figure 6. To generate application code, see Appendix B—User Application Code Generation on page 13.

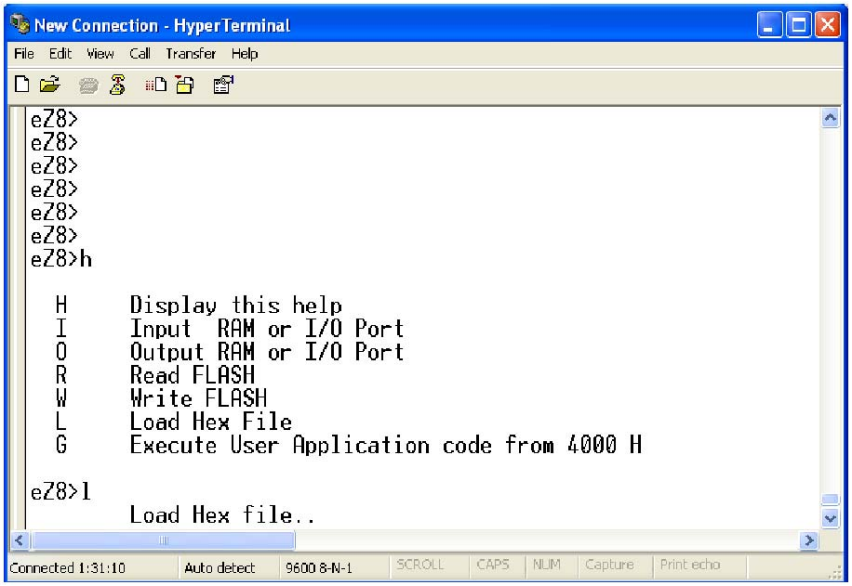

Figure 6. Flash Loader Screen Displays Start of HEX File Download

As displayed in Figure 7, choose the **Send Text File** option from the **Transfer** menu to download the hex file.

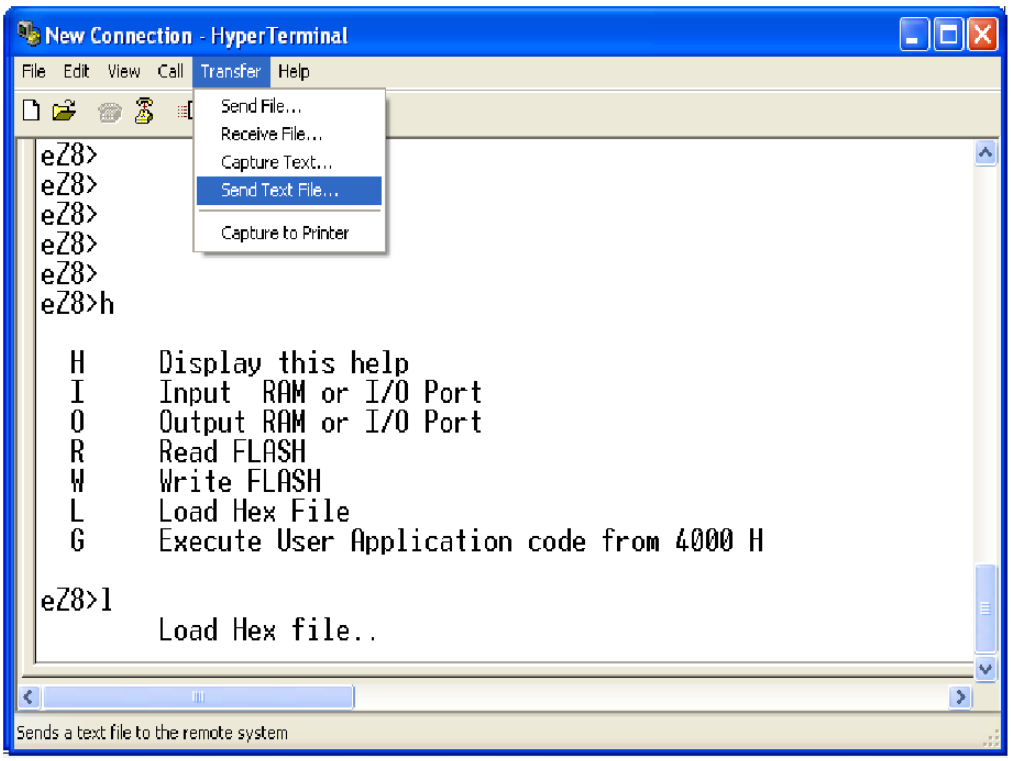

Figure 7. Transferring HEX File Using Hyper Terminal

Figure 8 displays that the download of the hex file is complete.

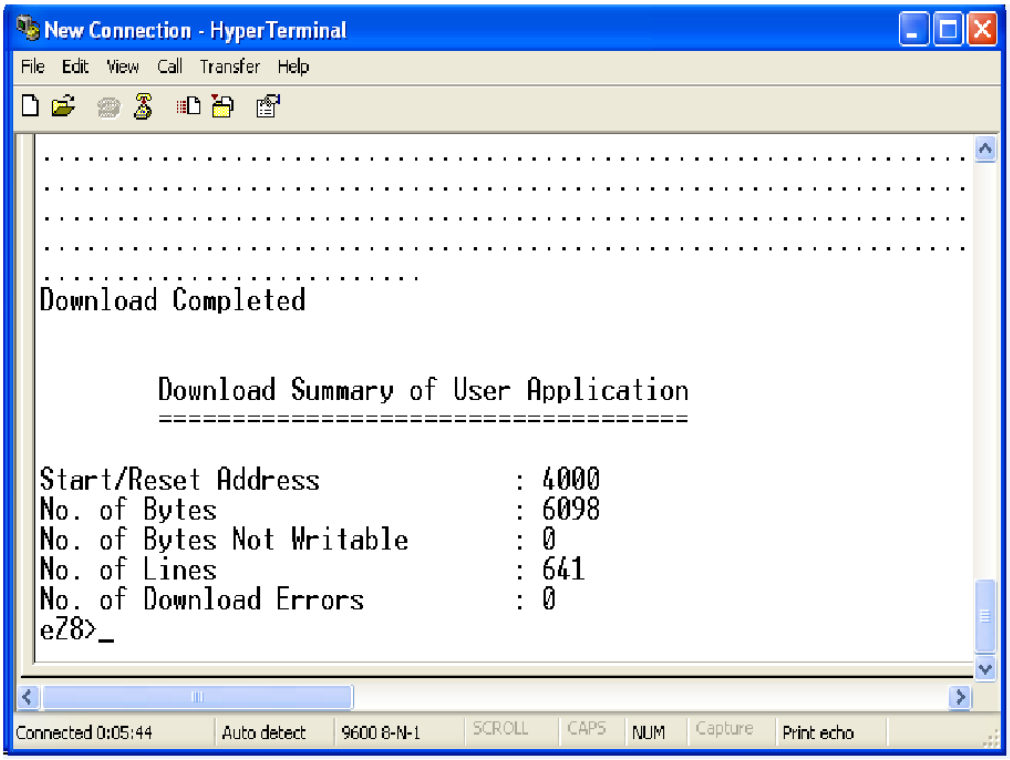

Figure 8. Flash Loader Screen Displays HEX File Download Completion Summary

Summary

The implementation of Flash Loader demonstrates the self-programming capability of the Z8 Encore! XP MCU. This code is not optimized for size, but offers a starting point for further Flash Loader optimization. Additionally, the user retains the ability to modify, use, and improve the Flash Loader utility for application-development situations in which the method of Flash Read/Write exists in the application code itself.

References

The documents associated with Z8 Encore! XP® MCU available at www.zilog.com are provided below:

- eZ8 CPU User Manual (UM0128)
- Z8 Encore! XP F64XX Series Product Specification (PS0199)
- Z8 Encore! XP F64XX Series Development Kit User Manual (UM0151)
- Zilog Developer Studio II—Z8 Encore! XP User Manual (UM0130)

Appendix A—Project Settings

After connecting a serial cable from the ZDS II COM port to the Target Interface Module (On-Chip Debug port) of the Z8 Encore! XP® Evaluation board, configure the project settings for compiling and loading to Flash memory using ZDS II. The sequence of screen shots in this appendix provide assistance with this configuration task.

Figure 9 through Figure 13 displays the ZDS II C-Compiler, linker, and target settings for the Flash Loader project.

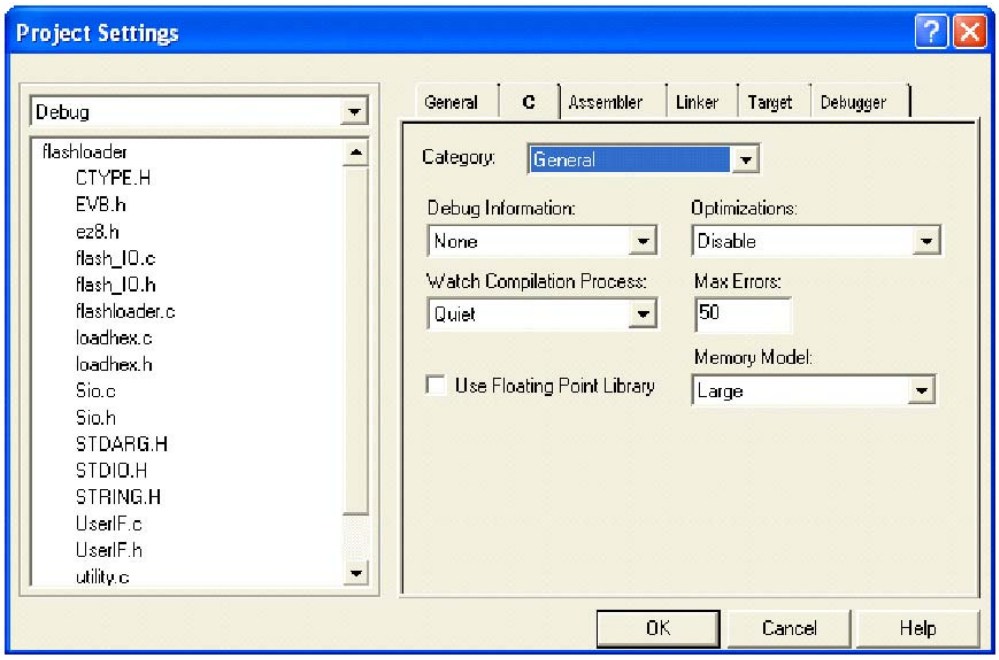

Figure 9. ZDS II C-Compiler General Category Settings

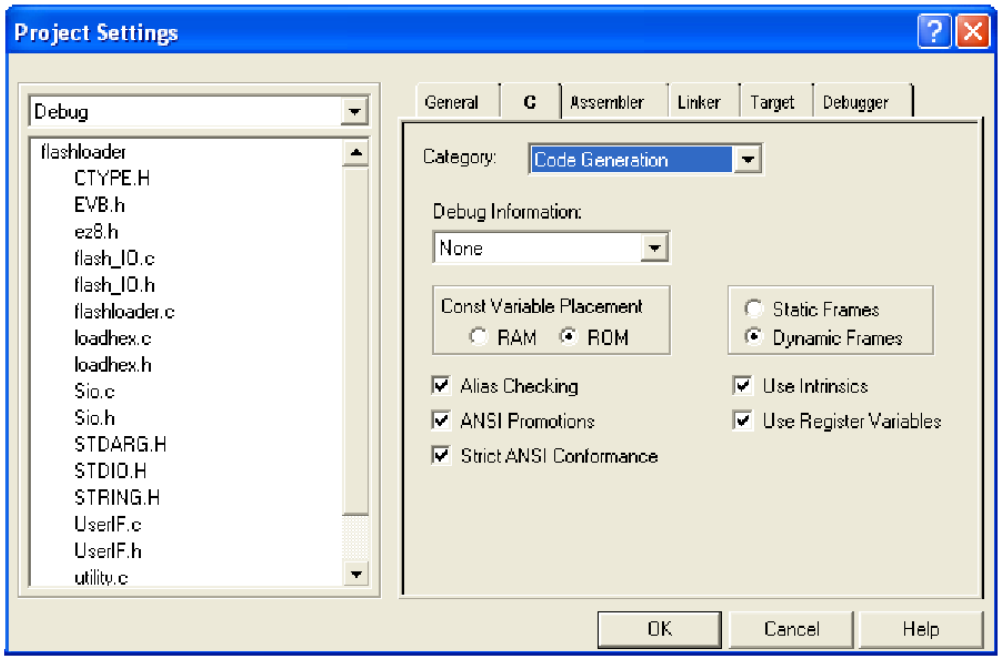

Figure 10. ZDS II C-Compiler Code Generation Settings

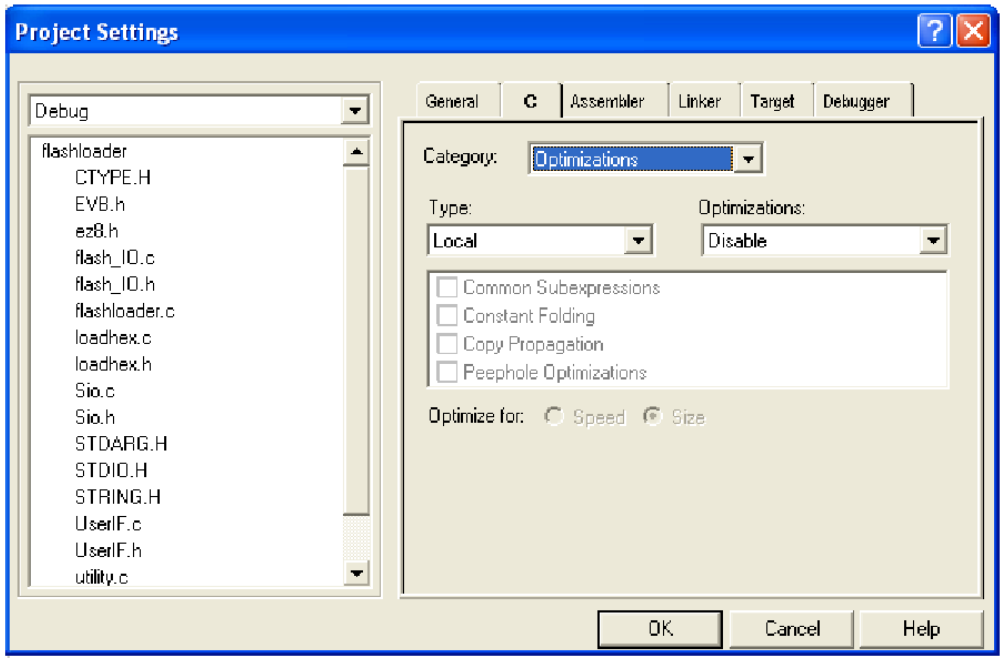

Figure 11. ZDS II C-Compiler Optimizations Settings

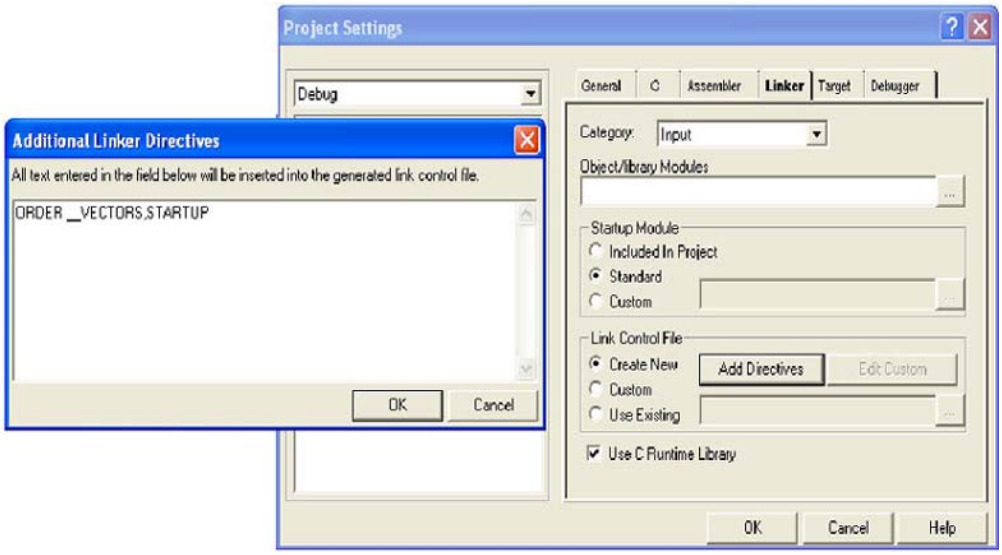

Figure 12. Input Directive Settings

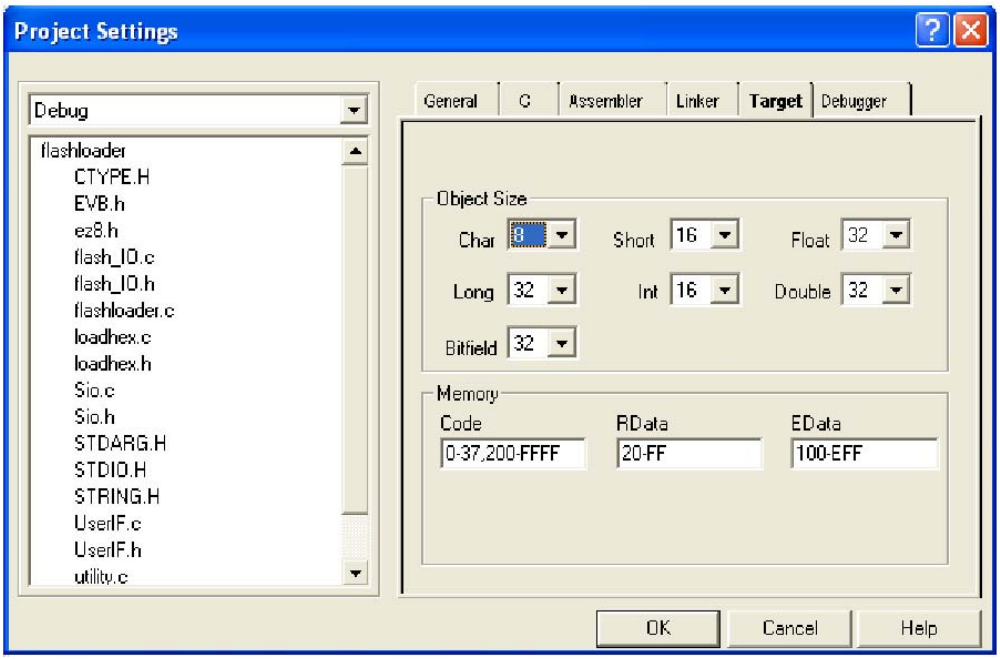

Figure 13. ZDS II Target Settings for Flash Loader

Appendix B—User Application Code Generation

This section discusses code relocation and the Intel HEX file format.

Code Relocation

Upon defining certain settings in ZDS II, application code can be relocated to the user application area starting at address location 4000h. Figure 14 displays the **Project Settings** dialog box, with the **Target** tab selected. In the **Code** field of the **Memory** pane, reserve the address range 0-37h for the Reset and Interrupt vectors.

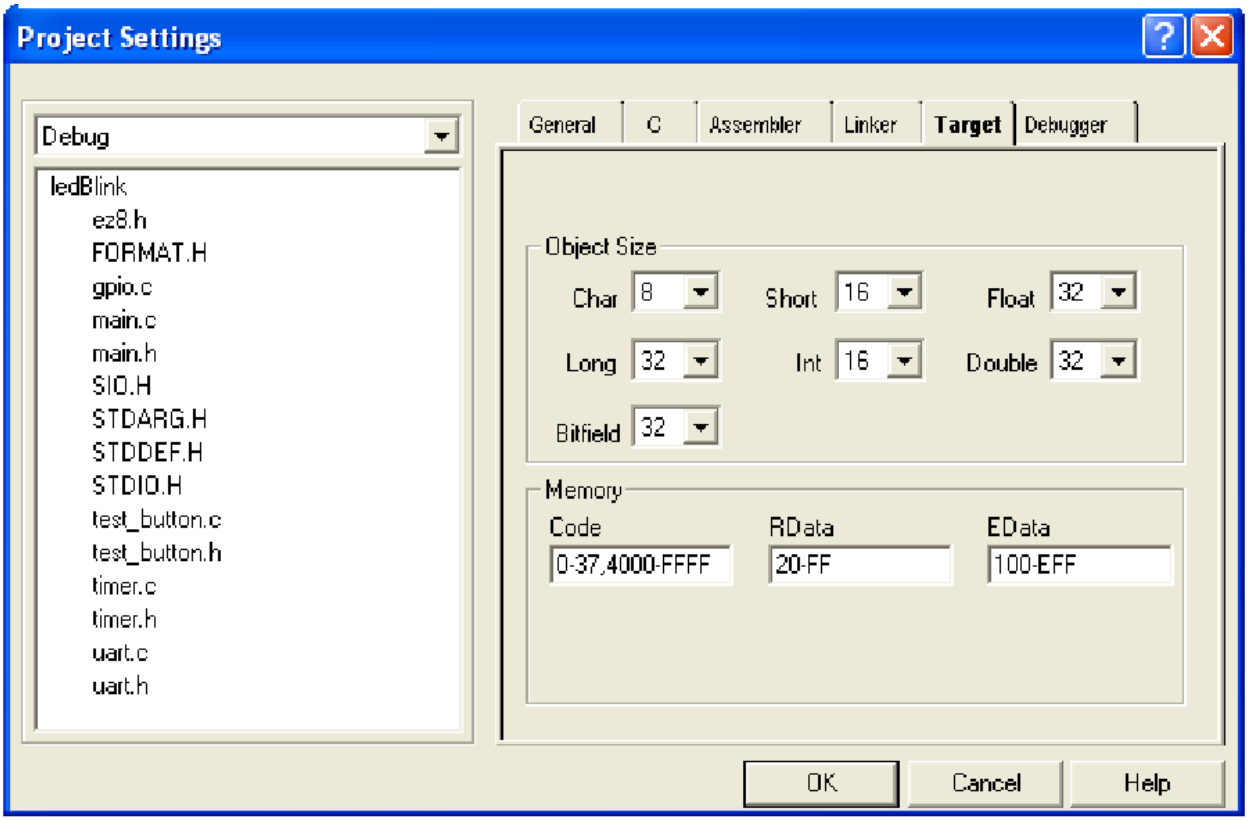

Figure 14. ZDS II Initial Target Settings

Figure 15 displays the **Project Settings** dialog box, with the **Linker** tab selected and a **Category** selection of **Input**. Clicking the **Add Directives** button opens the **Additional Linker Directives** dialog box, into which is added the directive `ORDER_VECTORS,STARTUP`.

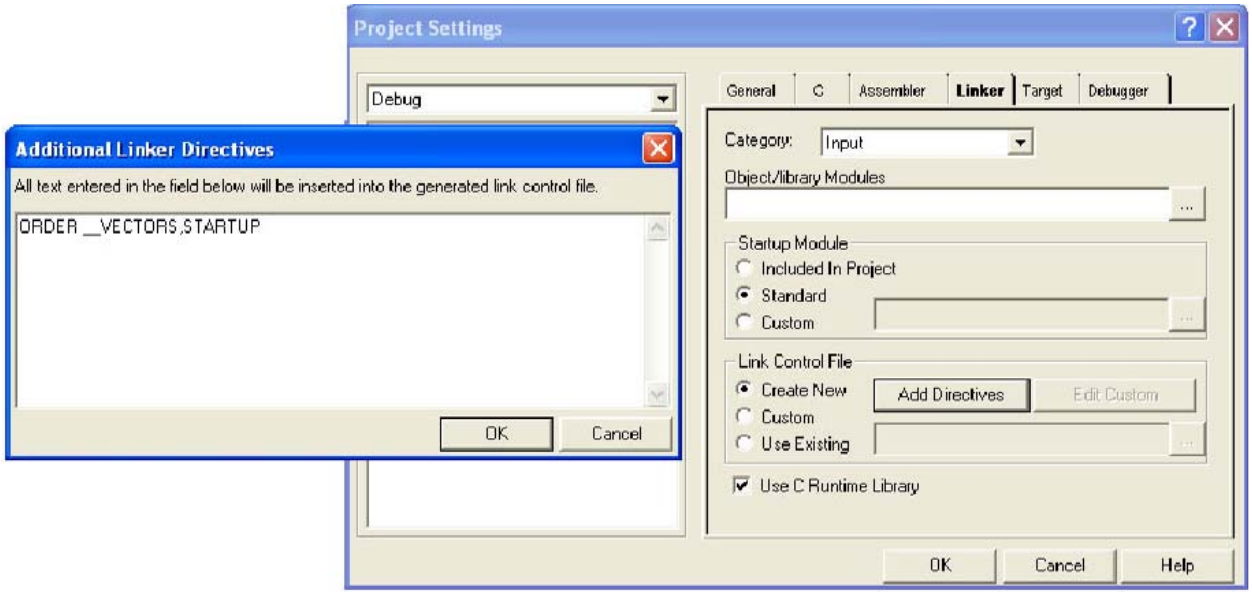

Figure 15. Input Directive Settings

Figure 16 displays the **Project Settings** dialog box, with the **Linker** tab selected and a **Category** selection of **Output**. To generate a hex file of the user application with ZDS II, check the box for hex code generation. The hex file is generated after the application build.

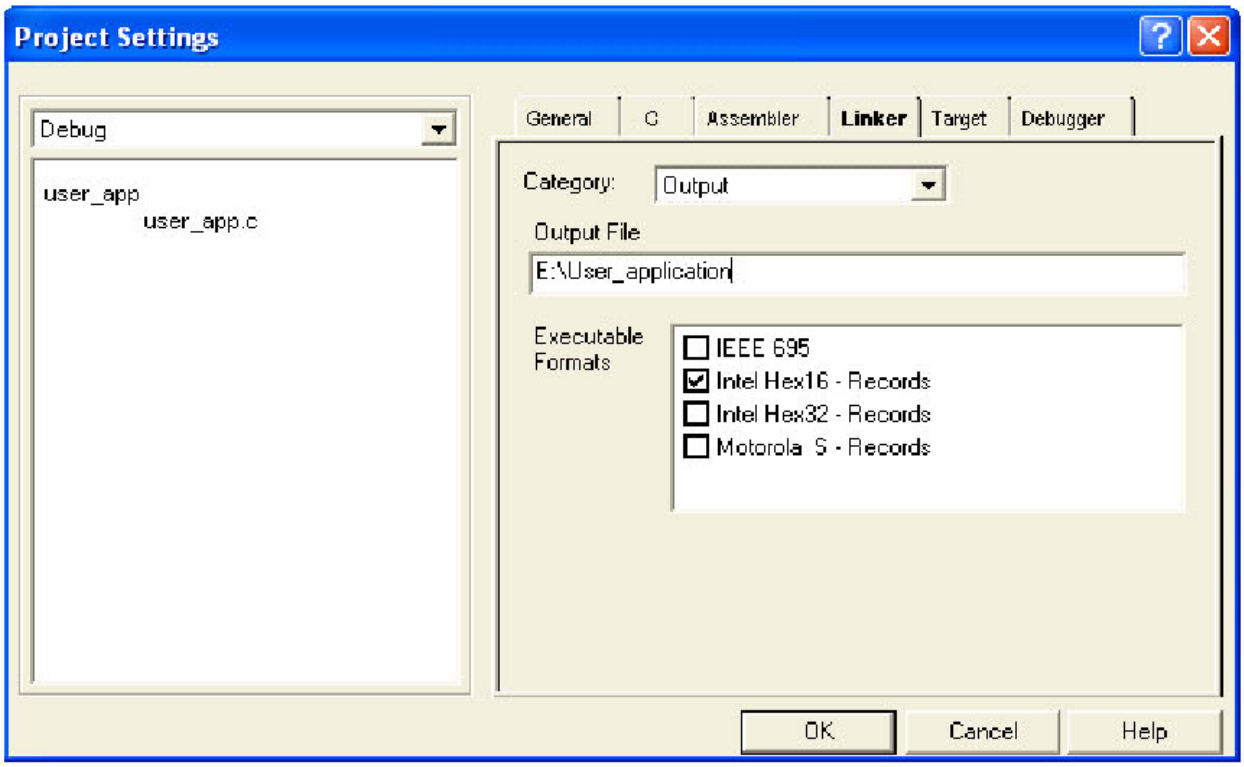

Figure 16. Generating a HEX File Using ZDS II

Intel Hex File

The Boot Loader expects Intel hex files to be loaded. The Intel Standard HEX file is one of the most commonly used formats. An Intel Standard HEX file is an ASCII file containing one record per line. Each line features the format shown in Table 3.

Table 3. Intel Standard Hex File Format

Position	Description
1	Record Marker: The first character of the line is always a colon (ASCII 0x3A) to identify the line as an Intel HEX file.
2–3	Record Length: This field contains the number of data bytes in the register represented as a 2-digit hexadecimal number. This is the total number of data bytes, not including the checksum byte nor the first 9 characters of the line.
4–7	Address: This field contains the address where the data should be loaded into the chip. This is a value from 0 to 65,535 represented as a 4-digit hexadecimal value.

Table 3. Intel Standard Hex File Format (Continued)

Position	Description
8–9	Record Type: This field indicates the type of record for this line. The possible values are: 00 = Register contains normal data. 01 = End of File.
10–?	Data Bytes: The following bytes are the actual data that are burned into the EPROM. The data is represented as 2-digit hexadecimal values.
Last 2 characters	Checksum: The last two characters of the line are a checksum for the line. The checksum value is calculated by taking the two's complement of the sum of all of the preceding data bytes, excluding the checksum byte itself and the colon at the beginning of the line.

The following text is an example hex file:

```
:10008000AF5F67F0602703E0322CFA92007780C361
:1000900089001C6B7EA7CA9200FE10D2AA00477D81
:0B00A00080FA92006F3600C3A00076CB
:00000001FF
```

Observe the top line, in which the first character (:) indicates the start of a record. The next two characters indicate the record length (in this case, `10h`). The next four characters represent the load address (in this case, `0080h`). The next two characters indicate the record type.

Following the above nine characters is the actual data. The last two characters are a checksum (sum of all bytes + checksum = 00).

Warning: DO NOT USE IN LIFE SUPPORT

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