Application Note

# Boot Loader for the Z51F6412 MCU

AN037701-0215

## Abstract

This application note discusses how to create a boot loader program for the Z51F6412 microcontroller, a member of Zilog's Z8051 Family of Microcontrollers. The boot loader is developed using the Keil µVision 4 IDE and provides the functionality to program an Intel Hex-format file to the Z51F6412 MCU's Flash memory through the UART.

**Note:** The source code file associated with this application note, AN0377-SC01, is available free for download from the Zilog website. To use files contained in this .zip file, install Zilog's Z8051 Software and Documentation (Product ID SD00023), which is available free for download from the Zilog Store. This source code can be compiled using Keil µVision 4 and has been tested using Zilog's Z51F6412 Development Kit. For this source code to work properly with our other Z8051 MCUs, you may be required to modify the code supplied with this application note.

## Features

The key features of this application include:

- Usable user application code space of up to 64,256 bytes
- Ability to load user application code via the RS232 serial interface

## Discussion

A *boot loader* is typically a program that permanently resides in the nonvolatile memory section of a target processor, and is the first set of code to execute at Power-On Reset (POR). Atypical boot loader features the following functional characteristics:

- The reset address of the target CPU points to the start address of the boot loader code.
- The boot loader reads the UART port for a certain time, waiting for a specific character input to be received. This character triggers the boot loader to enter Flash loading mode.
- In the absence of this character input, the boot loader code branches to the existing user application program. When the boot loader is in Flash loading mode, it typically receives data through a COM port to program the user code into Flash memory.

- The boot loader performs error checking on the received data, using the checksum method.
- The boot loader issues a command to the Flash controller to program the data into Flash memory.
- The boot loader checks the destination of the user code to prevent any inadvertent programming of the user code into its own memory space.

## Z51F6412 MCU Flash Overview

Zilog's Z51F6412 MCU features 64KB of Flash memory, which is divided into 1,024 pages; each page consists of 64 bytes. Flash memory is read or written one byte or page at a time. Figure 1 shows the Flash memory map of the Z51F6412 MCU.

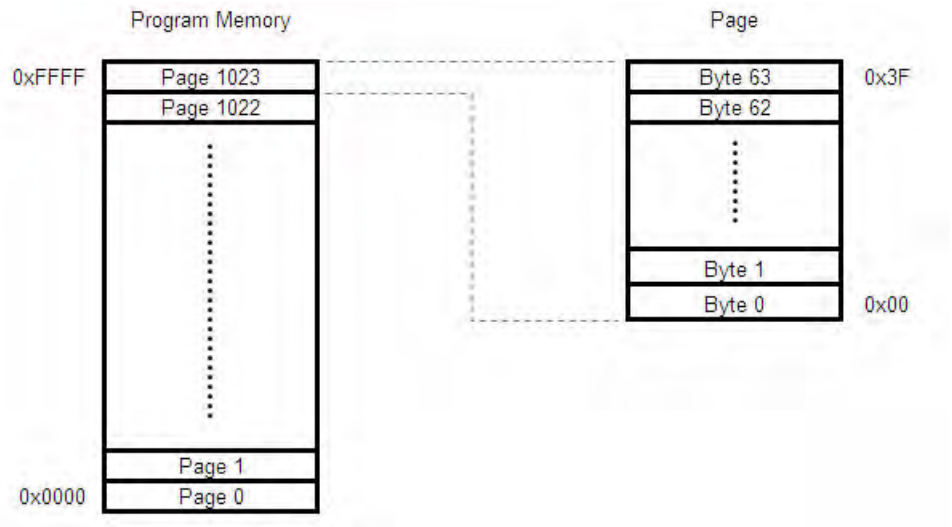
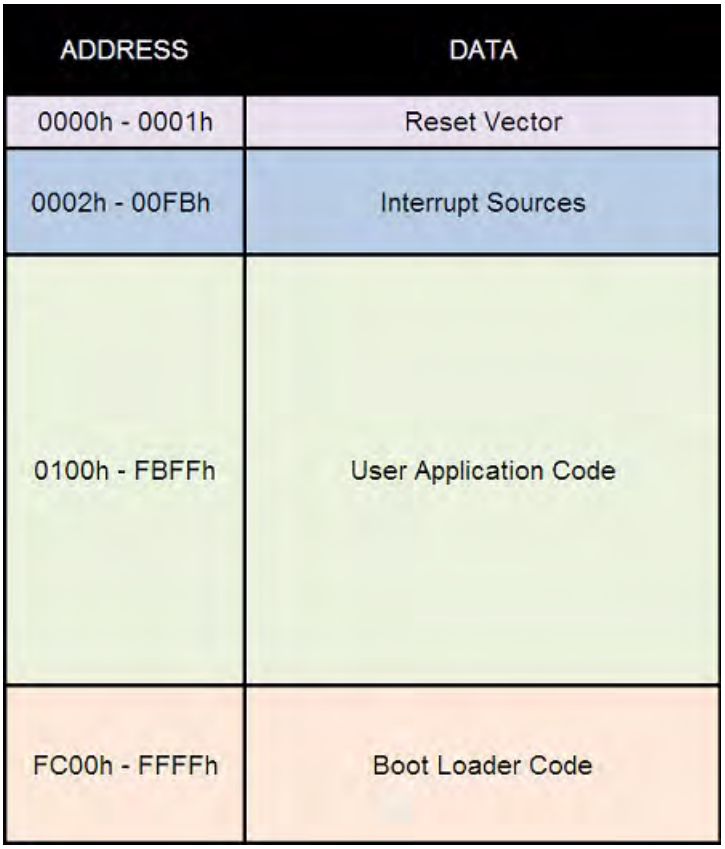



**Figure 1. Z51F6412 Flash Memory Map**

## Z51F6412 MCU Boot Loader Features

The boot loader program operates in the following sequence:

1. Upon receiving a specific character from the serial port within a specified period of time (i.e., two seconds), Flash Loading Mode is invoked. When Flash loading completes, the boot loader program transfers control to the user application, which then begins to execute. The address range of the user application code ranges from `100h` to `FBFFh`, as shown in Figure 2.



| ADDRESS | DATA |
|---|---|
| 0000h - 0001h | Reset Vector |
| 0002h - 00FBh | Interrupt Sources |
| 0100h - FBFFh | User Application Code |
| FC00h - FFFFh | Boot Loader Code |

**Figure 2. The Z51F6412 MCU's Flash Memory Address**

2. The boot loader program erases the reset vector, interrupt vectors, and the user application code area, which is the portion of memory in which the boot loader resides.
3. The user application code is received via the RS232 port of a serial terminal emulation program such as HyperTerminal. The boot loader calculates and verifies a checksum to detect errors, if any. If the loaded hexadecimal file contains checksum errors, the program displays an error message.
4. The boot loader program loads the Intel Hex-formatted data into Flash memory one byte at a time, and displays a progress indicator in HyperTerminal.

## Software Implementation

Generally, a boot loader's only function is to download a hexadecimal file to the MCU's Flash memory. This application is designed to program the Z51F6412 MCU via its UART block, which is an alternative to using Zilog's OCD. The advantage of using the UART is that users can update the firmware via the RS232 serial interface.

Figure 3 shows the typical flow of the boot loader execution.

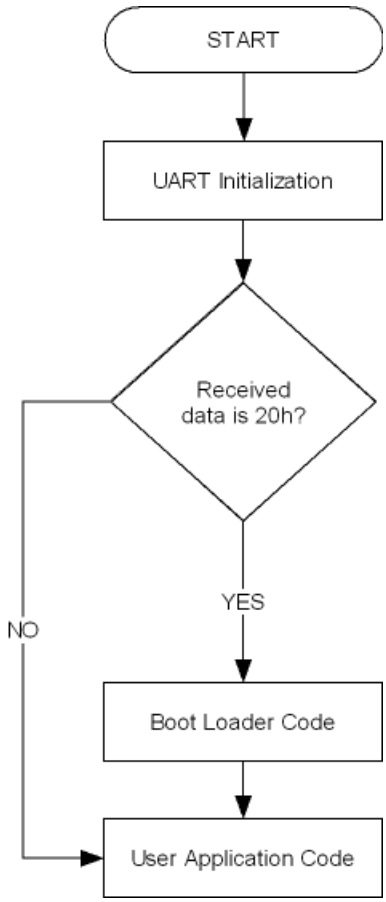



**Figure 3. Main Flow Diagram of the Boot Loader**

UART0 communication parameters are set to the following values in HyperTerminal (or a similar terminal emulation program):

- 4800 baud
- No parity
- 8 data bits
- 1 stop bit

The program enters the boot loader code when the keyboard space bar (i.e., ASCII code `20h`) and the MCU's reset button are simultaneously pressed. The boot loader code downloads the hexadecimal file to the MCU's Flash memory. The program then jumps to the start address of the user application code, which starts executing from Flash memory.

## Boot Loader Code

The boot loader code is responsible for reading the hexadecimal file coming from the UART, and writing it to Flash memory. The hexadecimal file to be programmed into the user application code memory space is located in the 100h–FBFFh address range. The boot loader program resides in the remaining portion of Flash memory, FC00h–FFFFh. When the boot loader code starts, the HyperTerminal window displays a message indicating boot loader initialization.

The boot loader code operation depends on the following sequence of functions, which is also shown in workflow format in Figure 4 on page 6.

1. Flash memory erasure, in which Flash memory is erased within the address range 0000h–FBFFh. This range contains the user's application code and the start address of the boot loader code. Flash memory is erased so that new data can be written to Flash memory.
2. A boot loader address rewrite, in which the boot loader code start address bytes FCh and 00h are written to Flash addresses 0000h and 0001h, respectively.
3. HyperTerminal displays a message indicating that the MCU is ready to accept the application code.
4. The MCU writes the application code one byte at a time to Flash memory. After the data is completely written to Flash memory, HyperTerminal displays a message indicating that the application code hex file is successfully downloaded to the MCU.
5. Finally, the program counter shifts to the user application code starting address (FC00h) to implement the downloaded application code.

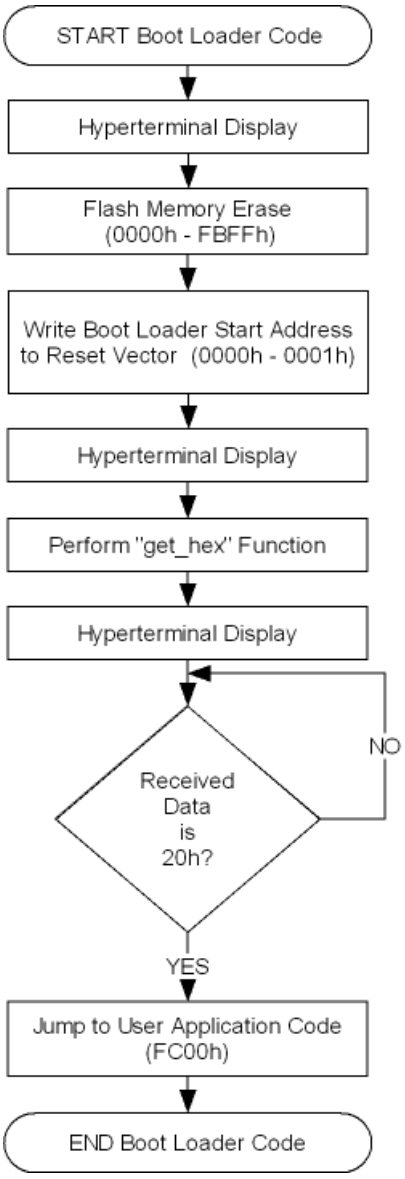



**Figure 4. Boot Loader Code Flow**

## get_hex Function

The `get_hex` function shown in Figure 5 on page 8 is responsible for reading the hex file and storing it in Flash memory pertinent to the following sequence.

1. The received data is checked. If the received character is a colon (:), the starting line of the hex file is indicated.

2. All ASCII characters are converted to the Intel Hex file format. ASCII characters A–F (41h–46h) are converted to the numbers 10–15 (0Ah–0Fh) while ASCII characters 0–9 (30h–39h) are converted to the numbers 0–9.
3. The first byte indicates the amount of data in a line; this amount is stored as a value in R3.
4. The second and third bytes indicate the MSB and LSB of the memory address. These bytes are stored in R1 and R2, respectively.
5. The fourth byte indicates the record byte of the data. The record byte is used to determine whether the data should be stored at a normal address, at an extended address, or at an end-of-file address.
   - Normal addressing is represented by the value 00h, while extended addressing is indicated by the value 04h.
   - End-of-file (EOF) addressing is represented by 01h. If end-of-file addressing is detected, the function defaults to the return command.
6. The bytes in the fifth to (N–1) range indicate the data to be stored in Flash memory.
7. The final byte indicates a checksum, which is used to check for errors during communication. The checksum byte must be equal to the two's complement of the total value of the 1st byte to the (N–1) byte. Failure to satisfy this condition will result in program termination. The checksum is stored in Register B.

Figure 5 depicts the flow of the `get_hex` function.

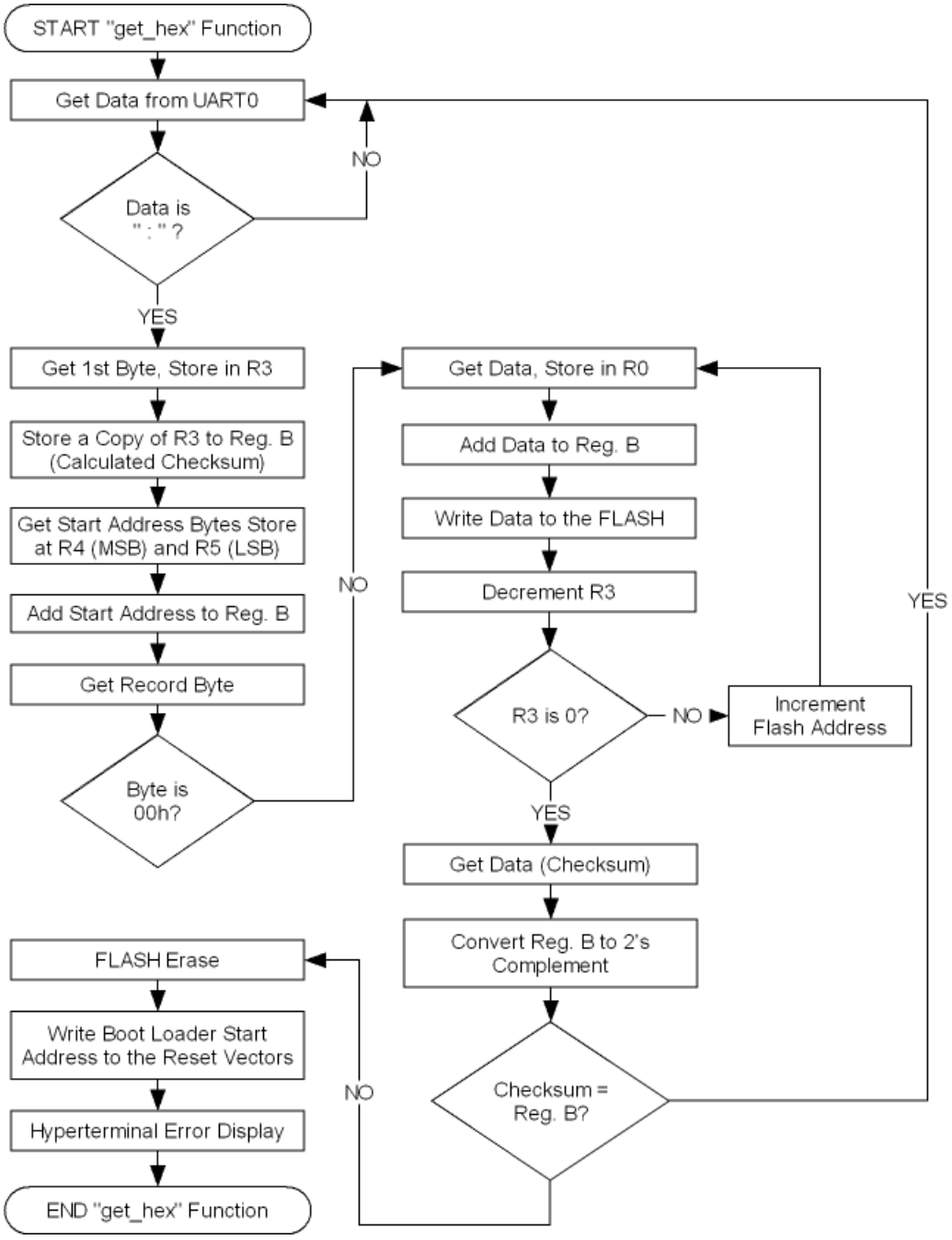



**Figure 5. Flow Diagram of the get_hex Function**

## Flash Erase and Flash Write Operations

Flash memory is the most important element in this application because the user application code and the boot loader code reside in this location. Two Flash operations are used in this application: Flash Erase and Flash Write. Prior to executing these two operations, a command sequence must be performed to activate Flash Write or Flash Erase Mode. The following sequence of events must be performed.

1. Write AAh to F555h.
2. Write 55h to FAAAh
3. Write A5h to F555h

After the command sequence is executed successfully, Flash memory is ready to perform an Erase/Write operation. Flow diagrams of Flash page erase and Flash write operations are shown in Figures 6 and 7, respectively.

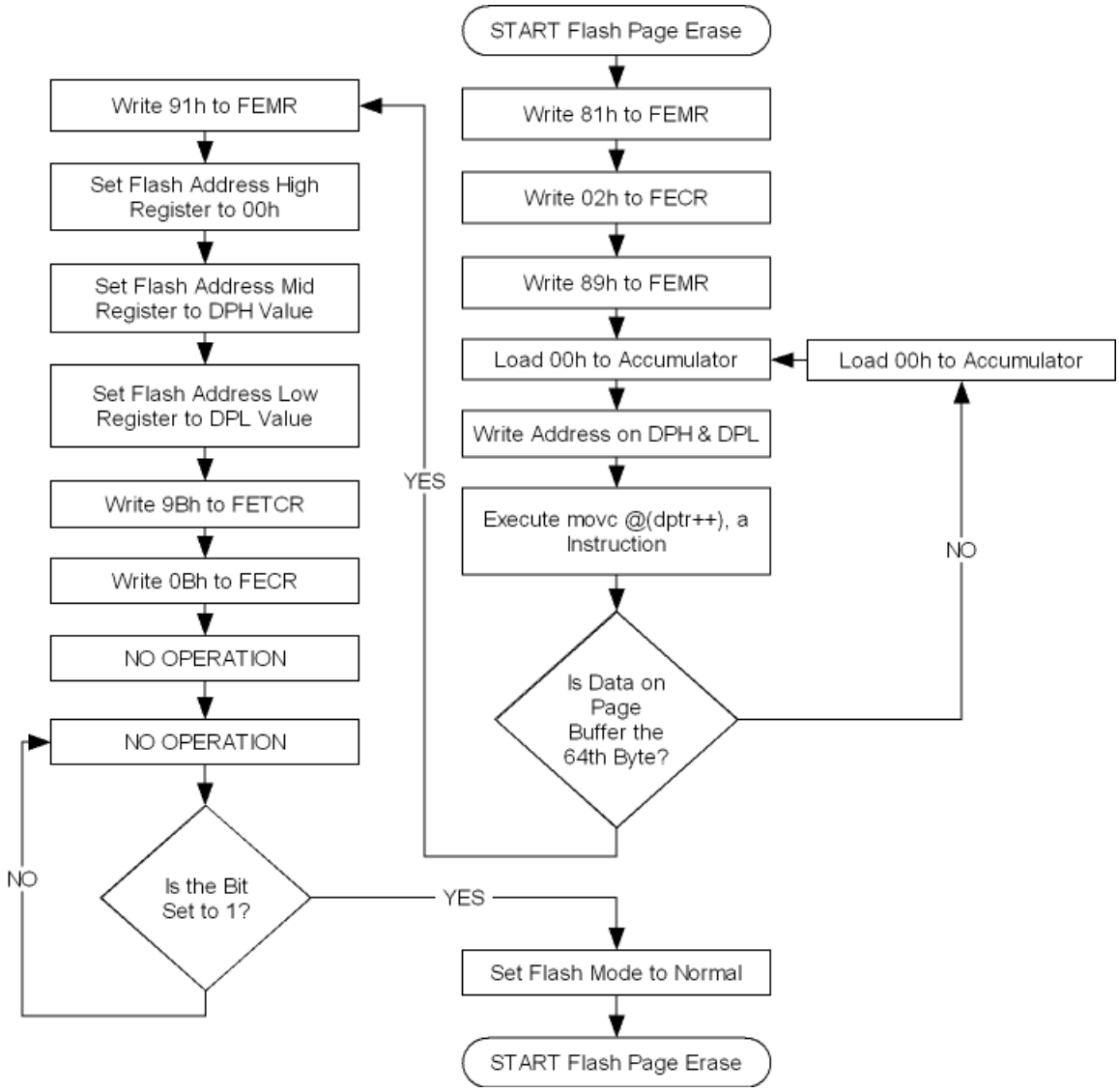



**Figure 6. Flash Page Erase Operation**

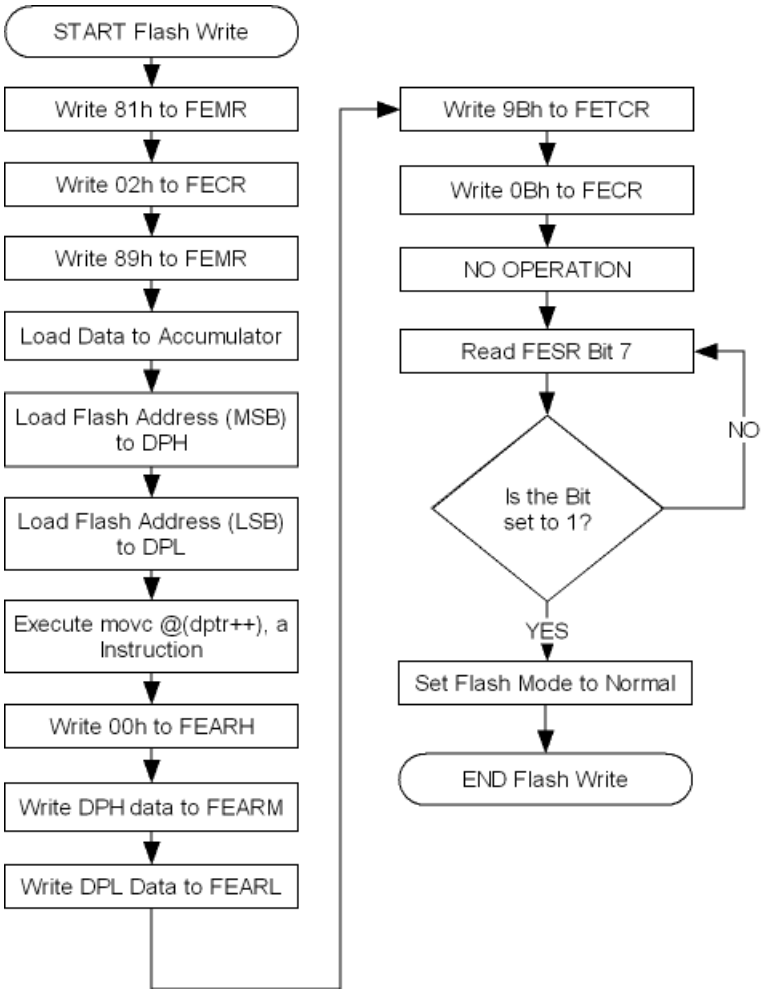



**Figure 7. Flash Write Operation**

# Equipment Used

This section provides a complete list of the hardware and software requirements for this application.

## Hardware

The only hardware used to develop this application is the Z51F6412 Development Kit.

### Software

The software tools used to develop this application include:

- Zilog Z8051 OCD 1.147
- Keil µVision 4 IDE
- AN0377-SC01.zip, which contains the project files and source code files.
- HyperTerminal or any equivalent communications and terminal emulation program.

## References

The documents associated with the Z51F6412 MCU are listed in Table 1. Each of these documents can be obtained from the Zilog website by clicking the link associated with its Document Number.

**Table 1. Z51F6412 MCU Documentation**

| Document ID | Description |
|---|---|
| PS0303 | Z51F6412 Product Specification |
| UM0259 | Z51F6412 Development Kit User Manual |

## Testing the Application

This section discusses a methodology for demonstrating this application and testing the software.

### Hardware Setup

Connect one end of the USB-to-USB mini cable to the Z51F6412 Development Board, and connect the other end to the PC's USB port. The LED indicator near the USB mini port of the Development Board will illuminate to indicate that power is supplied to the Board.

### Software Setup

To install, configure, and test the software for this application, observe the following procedure.

1. Download and install the Z8051 Software and Documentation files if you have not previously done so. These files are available free for download from the Downloadable Software category of the Zilog Store.
2. After the Z8051 software is installed, download the AN0377-SC01.zip file from the Zilog website and unzip it to the following path, which was created during the installation process in Step 1.

```
<Z8051 software installation folder>\samples
```

3. Open the project file `Z51F6412_Bootloader.uvproj` located in the following file-path, then build the application software by pressing F7 or selecting **Project → Build** from the menu bar.

   ```
   <Z8051 software installation folder>\samples\
   AN0377-SC01\keil\proj
   ```

4. A hex file is created at the conclusion of the build. Load this hex file to the MCU via Zilog's Z8051 OCD 1.147 software tool. The hex file can also be loaded using Keil by clicking the **Load** icon or selecting **Flash → Download** from the menu bar.

## Testing the Application

To test the operation of this application, observe the following procedure.

1. Load the `Z51F6412_Bootloader.hex` file by using either Keil µVision4 or the Z8051 OCD 1.147 software tool.
2. After loading the software to the MCU, disconnect the USB-to-USB mini connector from the Development Board, then disconnect the 10-pin connector of the OCD from the Development Board.
3. Apply power to the Development Board by reconnecting the USB-to-USB mini cable to the Development Board.
4. Configure HyperTerminal. In the application software, the MCU's UART is configured to communicate with 4800 baud, 8 bit data frame, no parity bits, and 1 stop bit. Essentially, configure the terminal emulation program to match the MCU's requirements.
5. To enter the boot loader code, reset the MCU and immediately press the space bar on the keyboard. HyperTerminal displays a message indicating that the MCU has entered the boot loader code, as shown in Figure 8.

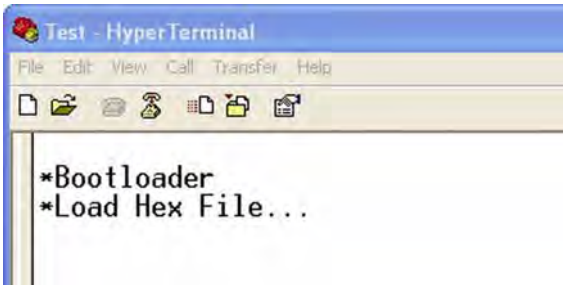



**Figure 8. HyperTerminal Display After Boot Loader Initialization**

**Note:** The text appears on the console screen 2–3 seconds after the space bar is pressed.

6. Send the hex file of the user application software to the MCU using Hyperterminal. On the menu bar, click **Transfer → Send Text File...**, then select the appropriate hex file to send to the MCU and click **Open** to begin writing of the hex codes to the MCU's Flash memory. A sample hex file can be found at the following location:

   `\AN0377-SC01\sample hex\led_blink.hex`

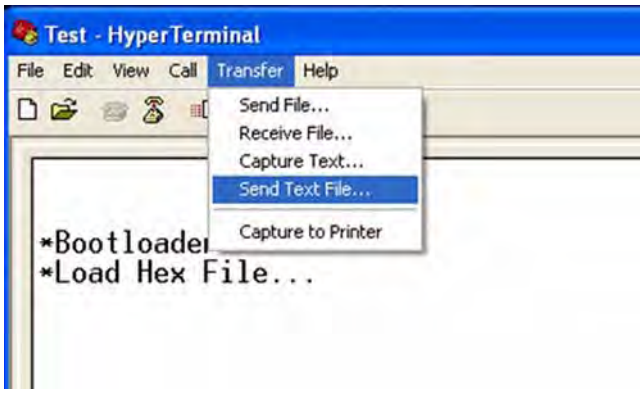



**Figure 9. Sending the Hex File via HyperTerminal**

**Note:** Ensure that the user application's memory layout is set correctly within the code range `0100h–FBFFh`.

7. Wait for the MCU to finish writing to Flash memory, after which HyperTerminal displays the message shown in Figure 10.

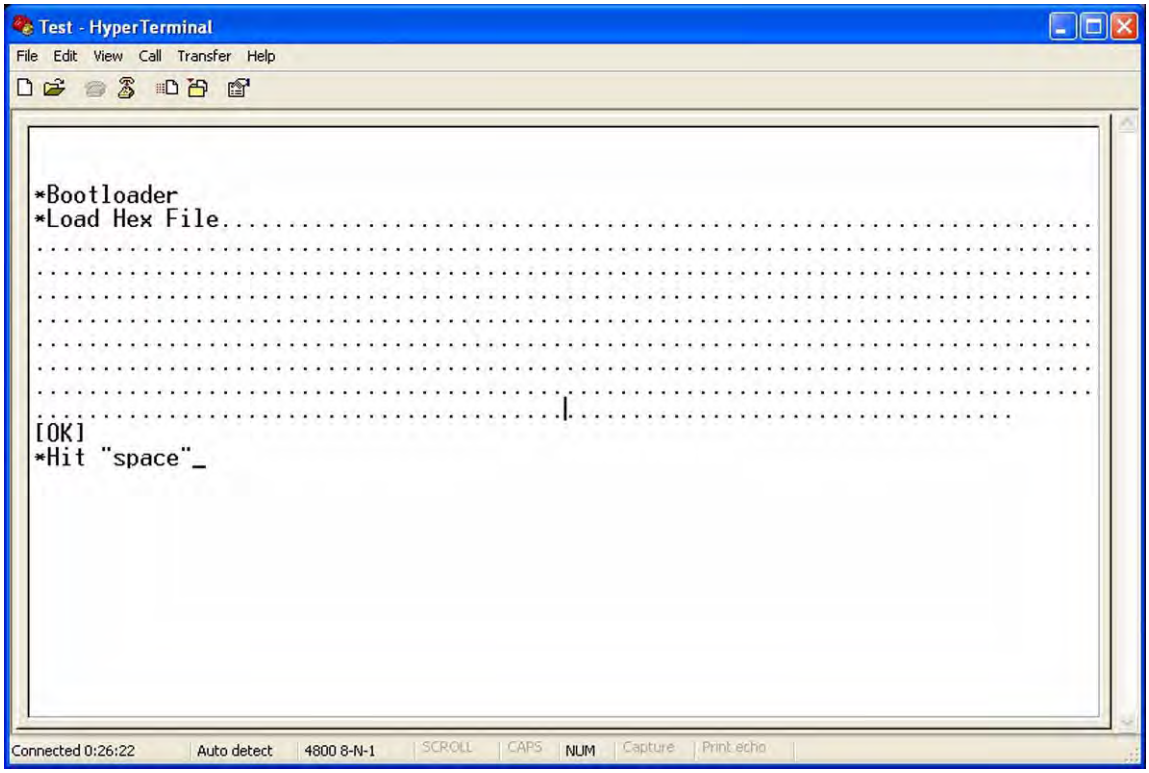



**Figure 10. Message Displayed in HyperTerminal Indicating Flash Write Operation Completed**

8. Press the space bar to run the application code.

# Results

Upon testing this application, the boot loader for the Z51F6412 MCU performs exactly as expected. When downloading the user application hex file, a baud rate of 4800 bps was found to be the optimum speed. Baud rates faster than 4800 can cause an unstable operation or malfunction.

# Summary

This boot loader program for the Z51F6412 MCU is designed to be used as a serial communication firmware download application, an alternative to Zilog's On-Chip Debug tool which communicates via a USB port. In this application, the hex file of the user code is sent to the MCU via HyperTerminal with a baud rate of 4800. Upon receiving the hex file, the MCU writes the received data to its own Flash memory. Testing of the application showed that the hex codes in the hex file were correctly written to the MCU's Flash memory and worked as expected.

# 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 at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/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.

**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.

Z8051 is a trademark or registered trademark of Zilog, Inc. All other product or service names are the property of their respective owners.