Application Note

Boot Loader for the eZ80F91 MCU

AN017404-0608

Abstract

This Application Note describes a Boot Loader program of minimal footprint for the eZ80F91 microcontroller unit's (MCU) on-chip Flash. The Boot Loader is loaded using Zilog's ZDS II—IDE and provides the functionality to program an Intel HEX 32 format file to the eZ80F91 MCU's Flash memory, using the RS-232 port.

Note: *The source code file associated with this Application Note is* `AN0174-SC01.zip`, *and is available for download at* www.zilog.com.

eZ80Acclaim*Plus!*™ Flash MCU Overview

eZ80Acclaim*Plus!*™ on-chip Flash MCUs are an exceptional value for designing high performance, 8-bit MCU-based systems. The eZ80Acclaim*Plus!* product line comprises of eZ80F91 MCU and eZ80Acclaim!® product line comprises of eZ80F92 and eZ80F93 MCU. With speeds upto 50 MHz and on-chip Ethernet MAC (on eZ80F91 only), you have the performance necessary to execute complex applications quickly and efficiently. Combining Flash and SRAM, eZ80Acclaim*Plus!* devices provide the memory required to implement communication protocol stacks and achieve flexibility when performing in-system updates of application firmware.

The eZ80Acclaim*Plus!* Flash MCU operates in full 24-bit linear mode addressing 16 MB without a Memory Management Unit. Additionally, support for the Z80®-compatible mode allows you to execute Z80/Z180 legacy code within multiple 64 KB memory blocks with minimum modifications. With an external bus supporting eZ80®, Z80, Intel, and Motorola bus modes and a rich set of serial communications peripherals, you have several options when interfacing to external devices. Some of the many applications suitable for eZ80Acclaim*Plus!* devices include vending machines, point-of-sale terminals, security systems, automation, communications, industrial control and facility monitoring, and remote control.

Discussion

This section describes a typical Boot Loader and elaborates methods for invoking the Boot Loader into the Flash loading mode.

A Typical Boot Loader

A typical Boot Loader is a program that permanently resides in the non-volatile memory section of a target processor, and is the first set of code to execute at Power-On Reset (POR).

The functional characteristics of a typical Boot Loader are as follows:

- The Reset Address of the target CPU points to the Start Address of the Boot Loader code.
- The Boot Loader polls on a port input for a predetermined logic level or polls on the UART port to receive a specific character.
- When a predetermined logic is detected on the polled port pin or a specific character input is received on the polled UART port, the Boot Loader is invoked into the Flash loading mode to program new user code into the Flash memory. In absence of any such indications, the Boot Loader code branches to the existing user application program, which begins to execute.

- When the Boot Loader is in the Flash loading mode, it typically receives data through a COM port to program the user-data into Flash memory.
- The Boot Loader performs error checking on the received data, using the checksum method.
- It issues commands to the Flash Controller to program the data into Flash memory.
- It checks the destination address of the user code to prevent any inadvertent programming of the user code into its own memory space.

The next section discusses various ways of invoking the Boot Loader application into the Flash loading mode.

Invoking the Boot Loader into Flash Loading Mode

There are several ways to invoke the Boot Loader into the Flash loading mode. The following methods are briefly described in this section:

- Polling a GPIO Pin
- Polling the Serial Port
- Using an Interrupt

Polling a GPIO Pin—This is a simple method. Here, a dedicated general purpose I/O pin is used, which is always pulled High. Upon Reset, the Boot Loader polls the GPIO pin for a Low state. If a Low state is detected, the Flash-loading mode is invoked and the Boot Loader is ready to program new code into Flash memory.

Polling the Serial Port—The polling serial port method involves using the serial port to invoke the Boot Loader into the Flash loading mode. Upon Reset, the device continuously polls the serial port for a particular character. When the character is received within the specified period of time, the Boot Loader program is invoked into the Flash loading mode. When no character is received within the specified time period, the Boot Loader program transfers program control to the existing user application program that begins to execute.

Using an Interrupt—In this method, an interrupt is used. This interrupt allows a device to invoke the Boot Loader through an interrupt service routine (ISR), at times other than at Reset. When an interrupt occurs, the interrupt handler branches to the Boot Loader that is invoked into the Flash loading mode.

Developing the eZ80F91 Boot Loader Application

The eZ80 CPU-based eZ80F91 MCU can write to its own Program Memory space. The eZ80F91 MCU features an on-chip Flash Controller that erases and programs the on-chip Flash memory.

The Boot Loader program uses the following eZ80F91 MCU on-chip peripherals to function.

UART—The UART0 is used to communicate with the PC's HyperTerminal; it is initialized to required baud rate by writing the appropriate value to the UART baud rate registers.

Flash Controller—The Flash Controller provides the appropriate Flash controls and timing for byte programming, Page Erase, and Mass Erase of the Flash memory. The Flash Controller contains a protection mechanism, via the Flash Control register (FCTL), to prevent accidental programming or erase.

Before performing either a program/erase operation on the Flash memory, you must configure the Flash Frequency High and Low Byte registers. The Flash Frequency registers allow programming and erasing of the Flash with system clock frequencies ranging from 154 kHz to 50 MHz.

For complete details on the on-chip Flash memory and the Flash Controller, refer to *eZ80F91 MCU Product Specification (PS0192)* and *eZ80AcclaimPlus! ASSP eZ80F91 Product Specification (PS0270)*, available for download at www.zilog.com.

Software Implementation

The eZ80F91 Boot Loader program offers the following functionality:

- The Boot Loader program can be invoked into the Flash loading mode either by polling a specific GPIO pin immediately upon POR, or polling the serial port for a specific character within a specified period of time. When neither of these operations are performed, the Boot Loader program transfers the control to the user application that begins to execute.
- The Boot Loader program selectively erases the Flash memory before programming user code; the area where the Boot Loader code resides is left unchanged.
- The Boot Loader program receives the user application code through the RS-232 port (HyperTerminal).
- It calculates and verifies the checksum for error detection.
- The Boot Loader program loads the data (in the Intel HEX 32 format) to the Flash memory, one line at a time.

Note: *A brief description of the* Intel HEX File Format *is provided in* Appendix B—Intel Hex File Format *on page 11.*

- The Boot Loader program provides a progress indicator on the HyperTerminal to indicate the status while the data is loaded into Flash, and displays *COMPLETED* on the HyperTerminal after the programming is completed.
- The Boot Loader program protects its own memory space by preventing the user code from being programmed into the area occupied by the Boot Loader.

The next few sections discuss the Boot Loader's functionality in detail.

Integrating the Boot Loader Code and the User Code

The Boot Loader code starts from location 03F800H. It is the first set of code that executes after a Reset. The reset location at 000000H, 000001H, 000002H, and 000003H contains the jump instructions to jump to starting address of the Boot Loader program, which resides at 03F800H.

Note: *The user application code must always start from location* 000100H.

Figure 1 displays the memory map for the Boot Loader in eZ80F91 MCU.

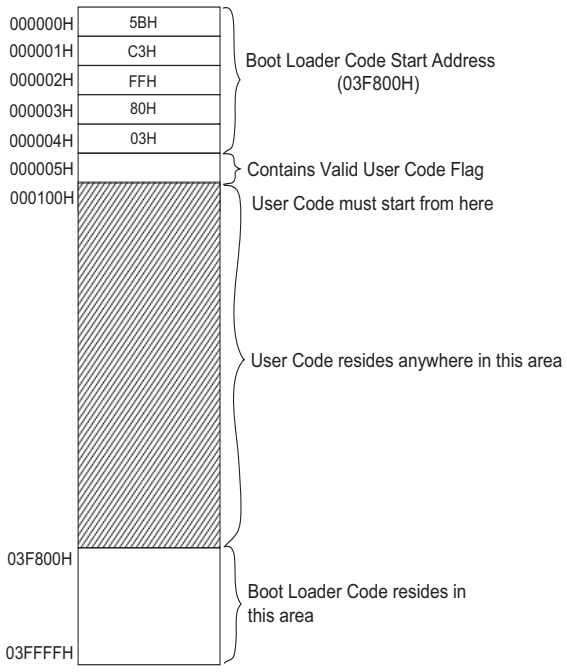

Figure 1. Memory Map for Boot Loader in Internal Flash Memory of the eZ80F91 MCU

Functioning of the Boot Loader

Immediately after Reset, the Boot Loader program initializes the UART0 to 9600 bps and initializes the specified GPIO pin (port A) as an input for polling.

The Boot Loader polls UART0 for a special character for a specified period of time. If a logic Low is detected at the GPIO pin, or if a special character is received, the Flash loading mode is invoked and the Boot Loader is ready to receive the data from the HyperTerminal through the RS-232 port.

Figure 2 displays the switch connection to a GPIO pin. For this implementation, pressing the switch SW1 invokes the Flash loading mode. When SW1 is pressed, the Boot Loader detects a logic Low at Port A pin 0, and the Flash loading mode is invoked. The *LOAD HEX FILE* message is displayed on the HyperTerminal screen.

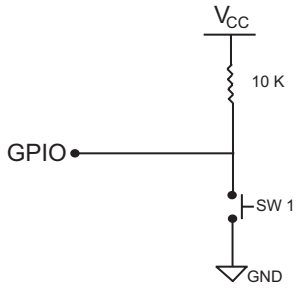

Figure 2. Switch Connection to a GPIO Pin

When the space bar is pressed during POR, the Boot Loader, polling on the UART0, detects this special character and is invoked into the Flash loading mode.

When the Boot Loader is invoked into the Flash loading mode, it initializes the frequency of programming the Flash memory by writing to the appropriate Flash Control registers.

When a special character is not detected at UART0 or a logic High is detected at the specified GPIO pin, the Boot Loader program branches out to the user application program. The user application program is first checked to see if it is valid before control is transferred to it and it begins to execute. For more information on checking the validity of the user application code, see Error Handling on page 5.

Receiving New User Code and Decoding the Hex File

When the Boot Loader is ready to receive the data from the HyperTerminal, it displays *LOAD HEX FILE* on the HyperTerminal. The Boot Loader initializes the global variable `offset_address` to `00H`. The Boot Loader code waits in a loop until a colon (:) is received. The colon indicates the start of a new line in the Intel HEX file format (see Appendix B—Intel Hex File Format on page 11).

If a colon is not received, then the program gets into a continuous loop by calling the `get_line()` routine, and waits until a colon is received. When a colon is received, the program calls `get_hex_byte()` routine, which returns a byte of data.

The `get_hex_byte()` routine in turn calls `receive_char()` routine, which receives ASCII characters from the HyperTerminal through UART0. The `get_hex_byte()` routine reads two consecutive ASCII characters, converts each character into Hex nibbles and concatenates the two to form a Hex byte.

The first byte received after the colon indicates the number of bytes of data in that line. This data length value is stored in the `byte_count` variable.

Note: *Each nibble is sent as an ASCII character from the HyperTerminal. Two consecutive nibbles are concatenated to form a Hex byte.*

The `get_hex_byte()` routine is called to receive the starting address of the data byte in that line. First, the higher address byte is received and stored in the `high_address_byte` variable and next, the lower address byte is received and stored in the `low_address_byte` variable.

The `get_hex_byte()` routine is called again to receive the record type. If the record type is `00H`, then the data bytes available in one line and equal to the data length are read and stored in an array called `data_array` variable. If the record type is `01H`, it

indicates the end of the file, and the program jumps to the `file_complete()` routine. If the record type is `04H`, then the address field in that record is discarded and the two bytes of data that are received are treated as the offset address. The upper byte received is always `00H` and hence only the least-significant byte (LSB) is moved to the `offset_address` variable. The effective three byte address is calculated by concatenating the following variables:

```
offset_address + high_address_byte + low_address_byte.
```

The checksum is calculated for the data bytes received after the colon, which includes the checksum value. This calculated checksum value is sent as the last data in that line. Thus, the calculated checksum value for all the received bytes, in a line, must equal zero if the data is valid. If the calculated checksum is equal to zero then the received data is written to the Flash memory. If the calculated checksum is not equal to zero, then the program jumps to `display_error()` routine and waits for a Reset. In case of an error, *CHECKSUM ERROR* is displayed on the HyperTerminal.

Programming Data into Flash Memory

To program the user code into the eZ80F91 MCU's Flash memory, the entire Flash memory must be erased. The Flash memory on the eZ80F91 MCU is divided into pages that are blocks of consecutive Flash addresses. A *page erase* operation erases all of the bytes on a page, while a *mass erase* operation erases data from all of the pages.

The CPU remains idle after issuing an erase command to the Flash Controller and resumes execution when the erase action is complete. The instance of erasing a page or a number of pages is termed as an erase cycle.

The Flash Controller routines, that the Boot Loader uses to program the user code into Flash are called in the following order:

1. `unlock_flash()` unlocks the Flash memory for writing.
2. `erase_flash()` erases the entire Flash memory.
3. `write_to_flash()` writes the received data to Flash memory.
4. `lock_flash()` locks the Flash memory.

For procedures to unlock, program, erase, and lock the Flash memory, refer to *eZ80F91 MCU Product Specifications (PS0192)*, available for download at www.zilog.com.

The Boot Loader writes the received data into the Flash memory at the address specified in the Hex file, using the `write_to_flash()` routine. As the data is written to the appropriate address, a progress indicator, in the form of dots (...), is printed on the HyperTerminal. Each dot represents a line of data programmed into Flash.

When the data from the line is completely programmed, the subsequent lines are received and programmed until the line indicating the end of the file is received. When the end of file indication is received, a success indication, *COMPLETED* is printed on the HyperTerminal. You must press the Reset switch for the downloaded user code to execute.

Error Handling

The error handling measures implemented for the eZ80F91 Boot Loader are explained in this section.

Error in Incoming Data—A checksum is calculated as the data bytes are received. This calculated checksum includes the last byte of data received, which is the checksum value for the data received. Thus, the calculated checksum value for all the received bytes, in a line must equal zero if the data is valid. If the calculated checksum value is not zero, an error condition *CHECKSUM ERROR* is displayed and the Boot Loader goes into an infinite loop. To return the program to normal operation, a Reset is performed.

Error due to Received Address Overlapping Boot Loader Address—The address received with each line of the Hex file is compared to the Boot Loader's memory area. The Boot Loader checks the address range in every line received. If the user code address overlaps with the Boot Loader memory area, the Boot Loader stops writing to Flash and displays *ADDRESS OVERLAP* on the HyperTerminal. The Boot Loader gets into a continuous loop, waiting for a Reset.

Error in Validity of User Code—The Boot Loader program must ensure that it does not execute faulty user code because the execution of such code might lead to unpredictable results. Faulty user code may be generated due to a variety of reasons, such as an error in communication, peripheral failure, or power failure, which may occur while the Boot Loader receives the user code.

The absolute location, `000005H`, is used for storing the user code valid flag. When the Boot Loader completes successfully loading to the Flash memory, `00H` is written to the location `000005H` by the `file_complete()` routine.

The Boot Loader reads the data at the absolute location `000005H`, before transferring the control for user program execution. If the value at location `000005H` is `00H`, then the control is passed to the user program. If the value is not `00H`, (indicating faulty user code) the Boot Loader does not transfer control to execute the user code; instead it is trapped in an endless loop to avoid causing any unintended operation. To return the program to its normal operation, a hardware Reset is performed.

Software Metrics

Table 1 lists the software metrics for eZ80F91 Boot Loader application.

Table 1. Software Metrics for Boot Loader on eZ80F91 MCU

	Memory Location		Code Size	Remark
Total Boot Loader code	03F800H	03FFFFH	941 bytes	~2 KB in the last page of Flash memory
Valid user program flag location, holds 00H if valid code is present	000005H	----------		
User code start addresses are stored at locations	000100H	----------		

When the Hex file is completely loaded into the Flash memory, the data memory area used by the Boot Loader is freed and is available for the user code.

Note: *The user code can use all the Flash pages except the last page and some locations on the first page.*

Downloading the Boot Loader Program

This section discusses how to download the Boot Loader program to the eZ80F91 MCU using Zilog's ZDS II IDE.

Procedure to Download the Boot Loader Program

Follow the steps to download the Boot Loader program to the eZ80F91 MCU:

1. Download the AN0174-SC01.zip file from www.zilog.com. Extract the file contents to a folder.
2. Launch ZDS II for eZ80Acclaim!, and open the *.zdsproj file located in the folder where the AN0174-SC01.zip file was extracted.
3. Build the project. The *.hex file is obtained.
4. To download the resulting *.hex file to the eZ80F91 MCU using ZDS II, follow the instructions given below:
   a. Select **Flash Loader** from **Tools** menu. The **Flash Loader Processor (eZ80F91)** dialog box is displayed.
   b. Type the path and the Hex file name to be flashed, in the **File** field or click **Browse** button to navigate to the Hex file.
   c. Select **Internal** in the **Flash options** area.
   d. Check **Erase Flash Before Burning** option and click **Program** button.
   e. The HEX file is then downloaded into Internal Flash memory.
   f. Select **Close** to close the dialog box when the download is complete.

Note: *Ensure that the **Internal Base Offset** option is set to 0.*

For more details, refer to *ZDS II—eZ80Acclaim! User Manual (UM0144)*, available for download at www.zilog.com.

Downloading User Code using the Boot Loader Program

This section describes how to download a user application code to the eZ80F91 MCU's Flash memory using the Boot Loader program.

The user code is built and the resultant *.hex file is used for loading into the Flash memory. While generating the Intel HEX File from the user code, the following points must be considered:

- The user code must start from 000100H. This is accomplished by setting ORIGIN as 000100 for the user code segment.
- Ensure to set the proper memory segments in the code area for eZ80F91 MCU, to avoid overlapping the Boot Loader code with the user code.

  As an example, using ZDS II to generate the user *.hex file, the settings under **Project → Settings → Linker → Address Space** must be: ROM area: 000006 - 037FFF
- The Intel HEX File must contain a maximum of 32 bytes of data per record. To achieve this, set the appropriate linker directive to 32.

  As an example, using ZDS II to generate the user HEX file, enter the following settings under **Project → Settings → Linker:**
  a. Select **Input** for the **Category** text field.
  b. Click **Add Directives** in the **Link Control File** pane. The **Additional Linker Directives** dialog box opens.
  c. Enter MAXHEXLEN = 32 in the **Linker Directives** text field. This generates a maximum of 32 bytes of data per record in the HEX file.

Notes: *1. MAXHEXLEN = 16 generates 16 bytes of data per record.*

2. While relocating the interrupt vector, make sure to relocate it into Internal Flash only.

The UART port on the eZ80F91 MCU must be connected to the COM port on the PC.

Follow the steps given below to download the user code to eZ80F91 MCU's Flash, using the Boot Loader program:

1. Launch the **HyperTerminal** on the PC with the following settings: 9600 bps, 8 data bits, No parity, one start bit, and no flow control.

Note: *If the record length is more than 16 bytes, change the baud rate to 2400 bps.*

2. The Boot Loader can be invoked into the Flash loading mode by two methods: polling on a GPIO pin and polling the serial port.
   a. Press **Reset** button on the development board while holding down the **SW1** switch (to obtain a logic Low at the GPIO pin, PA0).
   b. Observe that the Boot Loader displays *LOAD HEX FILE* on the HyperTerminal. The Boot Loader is now ready to receive the user data from the HyperTerminal.

   Or

   a. Press **Reset** button on the development board while holding down the space bar on the PC. A special character is detected at UART0.
   b. Observe that the Boot Loader displays *LOAD HEX FILE* on the HyperTerminal. Release the space bar. The Boot Loader is now ready to receive the data from the HyperTerminal.
3. Select **Transfer → Send Text File** from the HyperTerminal menu bar. The **Send File** dialog box is displayed.
4. In the **Filename** text field, specify the path for the user Hex file (in the Intel HEX file format) using the **Browse** button.
5. Click **Send** to send the file and close the **Send File** dialog box. The HyperTerminal sends the user Intel HEX file.
6. Observe the progress indicator in the form of dots (...), indicating that Flash loading is in process. When programming is completed successfully, the progress indicator displays *COMPLETED*. If there is a checksum mismatch, *CHECKSUM ERROR* is displayed. If an error due to address overlap occurs, then *ADDRESS OVERLAP* is displayed.
7. Release the switch **SW1** to obtain a logic High at the GPIO pin, PA0.
8. Press **Reset** on the development board to execute the user code.

Observations

An Intel HEX file of size 18 KB was programmed into the eZ80F91 MCU's Flash using the Boot Loader program. This Intel HEX file, with a minimum record size of 32 bytes and transferred at 9600 bps, takes approximately 90 s to be programmed into the Flash memory. The same file transferred at 19200 bps, takes approximately 60 s.

Summary

The Boot Loader program described in this Application Note is confined to the last page of the Flash memory, occupying 941 bytes of memory. The Boot Loader program is invoked into the Flash loading mode either by generating a logic Low at a GPIO pin or by detecting a special character at the serial port. This Boot Loader program facilitates field updates to the application/user program through the RS-232 port. Thus, products can be conveniently upgraded for features or functionality as and when required by upgrading the firmware at the site.

References

The documents associated with eZ80®, eZ80Acclaim!®, and eZ80Acclaim*Plus!*™ family of products are listed below:

- eZ80F91 MCU Product Specification (PS0192)
- eZ80F91 Development Kit User Manual (UM0142)
- eZ80 CPU User Manual (UM0077)
- eZ80AcclaimPlus! ASSP eZ80F91 Product Specification (PS0270)
- Zilog Developer Studio II—eZ80Acclaim! User Manual (UM0144)

Appendix A—Flowcharts

This Appendix displays the Flowchart for Boot Loader (eZ80F91) in Figure 3.

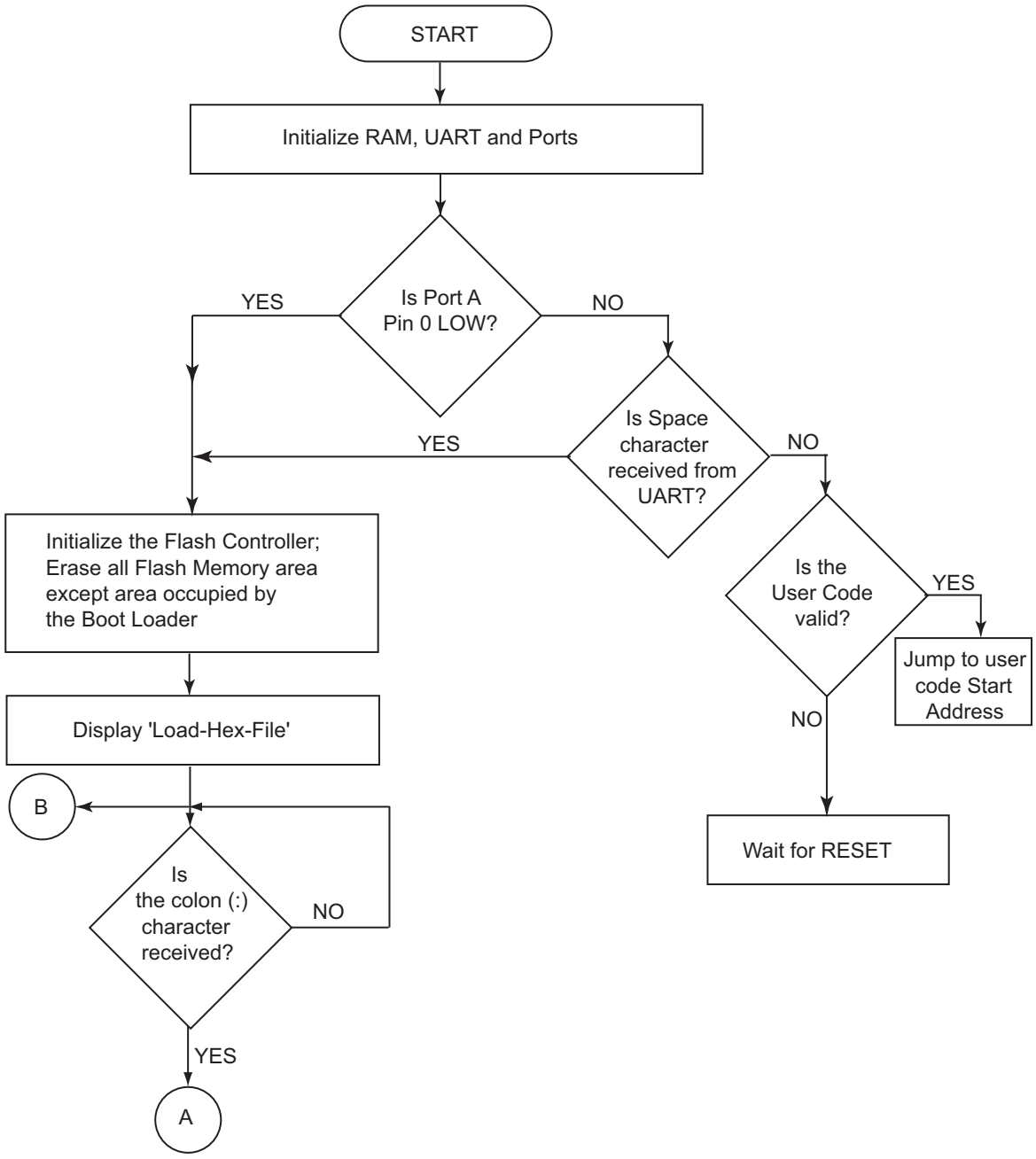

Figure 3. Flowchart for Boot Loader (eZ80F91 MCU)

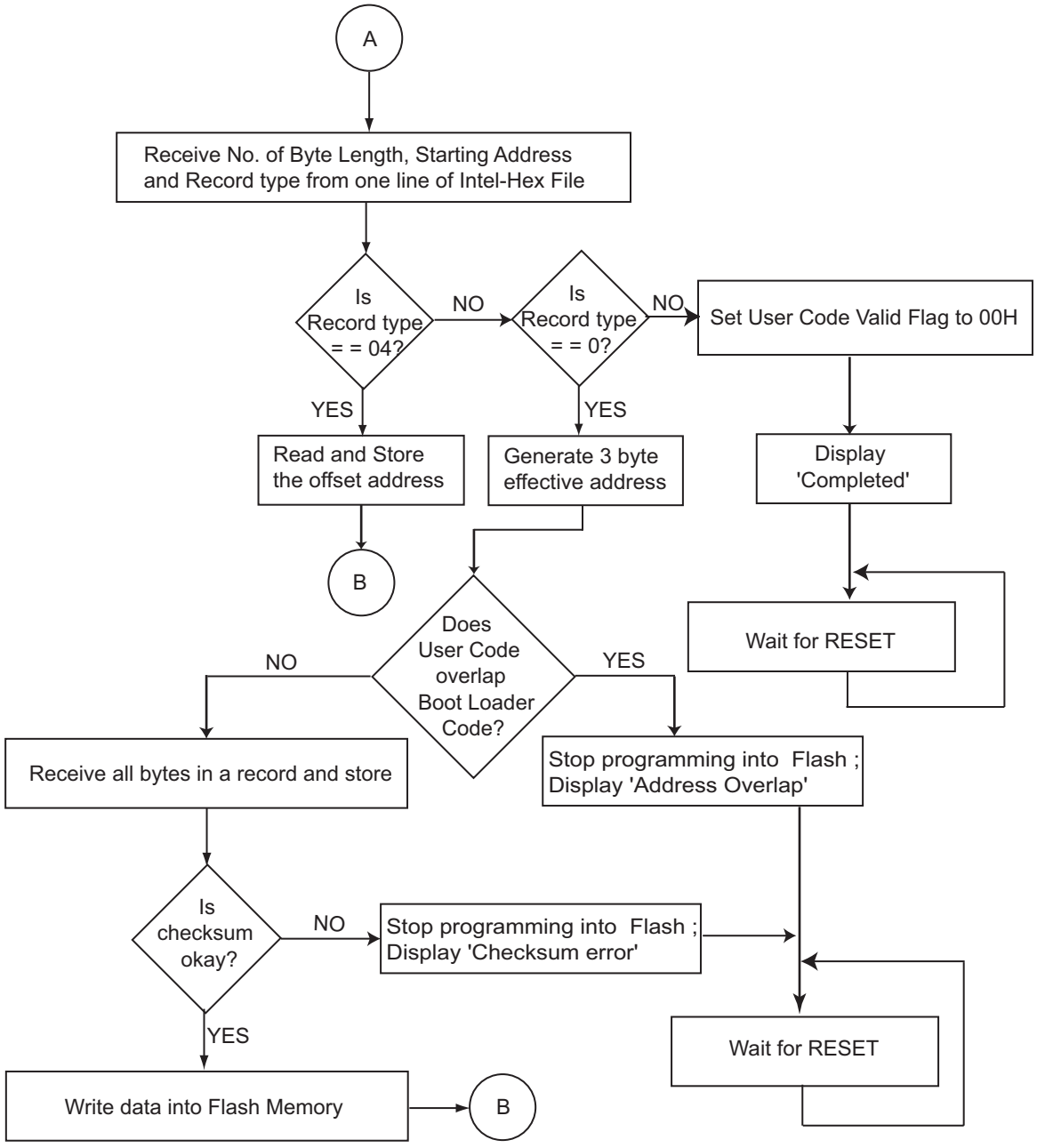

Figure 3. Flowchart for Boot Loader (eZ80F91 MCU) Contd.

Appendix B—Intel Hex File Format

The Boot Loader application can program a standard file format into eZ80F91 MCU's Flash memory. The Intel Standard HEX 32 format file is one of the popular and commonly used file formats. An Intel Standard HEX 32 format file is an ASCII file with one record per line. Table 2 lists the format for each line.

Table 2. Intel 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 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.
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. **02** = Extended linear address.
10 -?	**Data Bytes:** The following bytes are the actual data that will be 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 the checksum for the line. The checksum value is calculated by taking the two's complement of the sum of all the proceedings data bytes, excluding the checksum byte itself and the colon (:) at the beginning of the line.

Note: * Position1 is at the left.

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