Application Note

Boot Loader for Z8 Encore! XP® F642x Series MCU

AN016406-1207

Abstract

This Application Note describes a Boot Loader application of minimal footprint for Zilog's Z8 Encore! XP® microcontrollers (MCU) on-chip Flash. This Boot Loader application is loaded using Zilog's ZDS II IDE and provides the functionality to program an Intel HEX16 format file to the Z8 Encore! XP MCU's Flash memory using the RS-232 port.

The source code associated with this reference design is available under Z8 Encore! XP Applications Code Library in Application Sample Libraries on www.zilog.com.

Z8 Encore! XP® Flash MCU Overview

Z8 Encore! XP products are based on Zilog's eZ8™ CPU and introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows 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.

Featuring eZ8 CPU, the new Z8 Encore! XP MCUs combine 20 MHz core with Flash memory, linear-register SRAM, and an extensive array of on-chip peripherals. These peripherals make the Z8 Encore! XP suitable for a variety of applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Discussion

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

Boot Loader Features

A Boot Loader is a program which permanently resides in the non-volatile memory section of a target processor, and is the first piece of code to be executed at Power-On Reset (POR).

The functional characteristics of a Boot Loader are as follows:

- The Reset Vector Address points to the Start Address of the Boot Loader code.
- The Boot Loader polls on a port input for a predetermined logic level and 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 the 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 receives data through a COM port to program the user code into the Flash memory.
- The Boot Loader performs error checking on the received code using the checksum method.
- It issues commands to the Flash Controller to program the code into the 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 following section describes various methods of invoking the Boot Loader application into the Flash loading mode.

Invoking the Boot Loader into Flash Loading Mode

The methods for invoking the Boot Loader into the Flash loading mode are explained below:

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

Polling a GPIO Pin—In this method a dedicated general-purpose I/O pin is used, which is always pulled HIGH. On 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 the new code into the Flash memory.

Polling the Serial Port—The polling serial port method uses a serial port (the on-chip UART) to invoke the Boot Loader into the Flash loading mode. On 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 is invoked into the Flash-loading mode. When no character is received within the specified time period, the Boot Loader transfers program control to the existing user application program which 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), other than at Reset. When an interrupt occurs, the interrupt handler branches to the Boot Loader which is invoked into the Flash loading mode.

Note: *In Z8 Encore! XP Boot Loader application, invoking the Boot Loader in the Flash loading mode is achieved either by polling a GPIO pin or by polling the serial port.*

Developing the Z8 Encore! XP Boot Loader Application

The eZ8 CPU-based Z8 Encore! XP MCUs can write to their own Program Memory space. The Z8 Encore! XP MCUs feature an on-chip Flash Controller which erases and programs the on-chip Flash memory.

The Boot Loader either loads a user code into the Flash memory or transfers control to the user code for execution.

The Boot Loader uses the Z8 Encore! XP MCU's UART and Flash Controller on-chip peripherals to function.

UART—The UART0 device is used to communicate with the PC's HyperTerminal; it is initialized to the 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 inadvertent programming or erase.

Before performing either a program or 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 Flash with system clock frequencies ranging from 32 kHz to 20 MHz.

For more details on the on-chip Flash memory and Flash Controller, refer to *Z8 Encore! XP® 64K Series Flash Microcontrollers Product Specification (PS0199)*.

Software Implementation

The Z8 Encore! XP Boot Loader application offers the following functionality:

- The Boot Loader supports higher baud rate (up to 57600 bps). At high baud rates, the delay for writing into Flash is overcome by using the Clear To Send (CTS) pin of the HyperTerminal. After a line of record is read into the RAM memory, the CTS signal is asserted. This inhibits the HyperTerminal from sending any data. After the data is written into the Flash memory, the CTS signal is deasserted and the HyperTerminal resumes sending data.

Note: *Connect the T2IN pin of the Serial Driver IC (of UART) to port pin PF1. Connect the corresponding T2OUT pin to Pin 8 of the UART console, which is in turn connected to the CTS pin of the HyperTerminal.*

- The Boot Loader can be invoked in the Flash loading mode either by polling a specific GPIO pin immediately on POR, or by polling the serial port for a specific character within a specified period of time. When neither of these operations are performed, the Boot Loader transfers the control to the user application which begins to execute.
- The Boot Loader selectively erases the Flash memory before programming user code; the area where the Boot Loader code resides is left unchanged.
- It receives the user application code through the RS-232 port (HyperTerminal).
- The Boot Loader also calculates and verifies the checksum for error detection.
- It loads the data (in the Intel HEX 16 format) to the Flash memory, one line at a time. For more details on Intel HEX file format, see Appendix B—Intel HEX File Format on page 11.
- After loading a line of data into RAM, the Boot Loader indicates the HyperTerminal to stop sending data. After the data is written into Flash, the Boot Loader indicates the HyperTerminal to resume sending data.
- It provides a progress indicator in the HyperTerminal application to indicate the status while the data is loaded into Flash, and displays success in the HyperTerminal application after the programming is completed. If an address overlap or a checksum error occurs then an error message is displayed.
- The Boot Loader protects its memory space by preventing the user code from being programmed into the area occupied by the Boot Loader. The Boot Loader allows the user application code to start from any page other than from the Boot Loader area.

The following sections discuss the Boot Loader application in detail.

Integrating the Boot Loader Code and the User Code

The Boot Loader code starts from location FC00H and ends at FFFFH. It is the first piece of code that executes after a Reset. The reset vector is at locations 0002H and 0003H. The reset vector location contains the starting address of the Boot Loader code; that is, 0002H contains FCH and 0003H contains 00H. Therefore, after a Reset, the processor jumps from the reset vector location to location FC00H.

Figure 1 displays the memory map of the Boot Loader for the Z8F642x MCU.

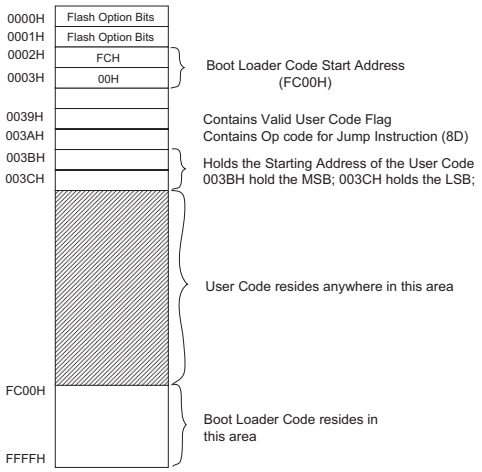


Figure 1. Memory Map for Boot Loader in Flash Memory (Z8F642x MCUs)

Executing the User Code

Immediately after Reset, the Boot Loader sets the Port A pins 4 and 5 as the UART0 receive and transmit pins, respectively. The UART0 peripheral is initialized to 9600 bps (higher baud rate is supported). The Boot Loader also initializes the specified GPIO pin as an input pin for polling.

The Boot Loader polls the specified GPIO pin for a logic LOW. When the Boot Loader detects a logic High, it polls the serial port for a special character. When a logic Low or a special character is detected at this pin, the Flash loading mode is invoked and the Boot Loader is ready to receive the data from the HyperTerminal application.

Figure 2 displays the switch connection to a GPIO pin. For this implementation, press switch SW1 to the ON position to invoke the Flash loading mode.

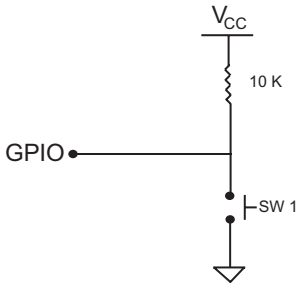


Figure 2. Switch Connection to a GPIO Pin

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 the Boot Loader does not detect a logic LOW at the specified GPIO pin, or the special character is not present, it checks if the user application is valid before the control is transferred to the application code, which then starts executing. For more information on checking the validity of the user application code, see Error Handling on page 6.

Receiving New User Code and Decoding the HEX File

When the Boot Loader is ready to receive the data from HyperTerminal, it displays *Load Hex File* in the HyperTerminal application.

The Boot Loader waits in a loop until a colon (:) is received. A colon indicates the start of a new line, see Appendix B—Intel HEX File Format on page 11. If the colon character is not received, 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 application 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 application. The 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 then 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.

The checksum is calculated for the data bytes received after the colon, including the checksum which is sent as the last data in that line. The calculated checksum value for all the received bytes in a line must be equal to 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 the display_chksum_error() routine and displays *Checksum Error* on the HyperTerminal. After this, the program waits for a Reset.

Programming Data into Flash Memory

To program the user code data into the Z8 Encore! XP MCU's Flash memory, the entire Flash memory must be erased. The Flash memory on the Z8 Encore! XP MCU is divided into pages which are blocks of consecutive Flash addresses. A Page Erase operation erases all bytes on a page, while a Mass Erase operation erases data from all 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 is termed as an erase cycle.

The Flash Controller routines used by the Boot Loader to program the user data into Flash is 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 *Z8 Encore! XP® 64K Series Flash Microcontrollers Product Specification (PS0199)*.

The Boot Loader writes the received data into the Flash memory at the received address using the write_to_flash() routine. When the data is written to the appropriate address, a progress indicator in the form of dots (.) is printed in the HyperTerminal application. 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 message, *Completed* is displayed on the HyperTerminal. Press the Reset switch on the application board to execute the downloaded user code.

Error Handling

The error handling measures implemented for the Z8 Encore! XP Boot Loader are explained in this section.

Error in Incoming Data—A checksum is calculated as 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 be equal to 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 with the Boot Loader's memory area. The user code can start at any location but must not occupy the Boot Loader memory space. The Boot Loader checks the address range in every line received. If the user code address overlaps with the Boot Loader memory area then the Boot Loader stops writing to Flash and displays *Address Overlap* on HyperTerminal. Then the Boot Loader is in continuous loop waiting for Reset.

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

An absolute location, `0039H`, is used for programming the user code valid bit. When the Flash loader completes successful loading to the Flash memory, `00H` is written to the location `0039H` by the `file_complete()` routine.

The Boot Loader reads the data at the absolute location `0039H`, before transferring the control for user program execution. If the value at location `0039H` is `00H`, the control is passed to the user program. If the value is not `00H`, (indicating faulty user code) the Boot Loader cannot execute the user code and is trapped in an endless loop to avoid 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 the Z8 Encore! XP Boot Loader application.

Table 1. Software Metrics for Boot Loader on Z8 Encore! XP F642x MCUs

Description	Memory Location		Code Size	Remark
Startup code for Boot Loader	FC00H	FC09H	10 bytes	1000 bytes in the last two pages of Flash memory
Boot Loader code	FC0AH	FFE7H	990 bytes	
Valid user program flag location. Holds 00h if valid code is present	0039H	----------	1 byte	4 bytes of memory is reserved for Boot Loader in Page 0 of Flash memory
Location of user code where Boot Loader jumps to start executing the code	003AH	----------	1 byte	
User code start addresses are stored at locations	003BH	003CH	2 bytes	

After the .hex file is loaded into the Flash memory, data memory area used by the Boot Loader becomes free and is available for the user code.

Note: *The user code can use all the Flash pages except the last two pages and some locations on the first page (page 0). Locations* 0039H *to* 003CH *are reserved for the Boot Loader.*

Downloading the Boot Loader Application

Follow the steps below to download the Boot Loader to the Z8 Encore! XP F642x MCU using Zilog's ZDS II IDE:

1. Install the Z8 Encore! XP Applications Library available in the applications library on www.zilog.com.
2. Launch ZDS II for Z8 Encore!, and open the Z8 Encore! F642x Boot Loader.zdsproj file located in the source folder.
3. Go to **Project → Settings**. The **Project Settings** dialog box appears.
4. Click **Linker** tab. Select **Address Spaces** in the **Category** field. Enter the following information:

   RData:20–FF (for the small memory model)

   ROM:0–3, 39–3C, FC00–FFFF

   EData:100–EFF
5. Build the project and download the .lod output file onto the Z8 Encore! XP MCU using ZDS II. For more details, refer to *Zilog Developer Studio II (ZDSII)—Z8 Encore! User Manual (UM0130)*.

Downloading User Code Using Boot Loader

This section describes how to download a user application code to the Z8 Encore! XP MCU's Flash memory using the Boot Loader.

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

- Set the proper memory segments in the code area for the Z8 Encore! XP MCU to avoid overlapping the Boot Loader code with the user code.
  For example, when using ZDS II to generate the user HEX file, the settings under **Project → Settings → Linker→ Category→ Address Spaces** must be:

  ROM: 0–38,3D–FBFF (the memory locations in the ranges 39–3C and
  FC00–FFFF are used by the Boot Loader)
- The Intel HEX file must contain a maximum of 16 bytes of data per record.
  For example, when using ZDS II to generate the user HEX file, enter the following settings under **Project → Settings → Linker:**
  - Select **Output** for the **Category** text field.
  - Select **Intel Hex16 - Records** in the **Executable Formats** to set the maximum bytes per Hex File Line to 16.

The UART port on the Z8 Encore! XP MCU must be connected to the COM port on the PC.

Follow the steps below to download the user code to Z8 Encore! XP MCU's Flash, using the Boot Loader:

1. Launch the HyperTerminal application on the PC with the following settings: 9600 bps, 8 data bits, no parity, one start bit, and no flow control. Follow the steps below to invoke the Boot Loader into the Flash loading mode:

a) Press the **Reset** button on the target board while holding down the SW1 switch (so as to connect the specified GPIO pin to a logic LOW level).

b) When the Boot Loader displays *Load Hex File* in the HyperTerminal application, release the SW1 switch. The Boot Loader is now ready to receive the user data from the HyperTerminal.

OR

a) Press the **Reset** button on the target board while holding down the space bar on the PC.

b) When the Boot Loader displays *Load Hex File* in the HyperTerminal application, release the space bar. The Boot Loader is now ready to receive the data from the HyperTerminal application.

2. Select **Transfer** → **Send Text File** from the HyperTerminal application menu bar. The **Send File** dialog box appears.
3. In the **Filename** field, specify the path for the user HEX file (in the Intel HEX file format) using the **Browse** button.
4. Click **Open** to send the file and close the **Send File** dialog box. The HyperTerminal application sends the user Intel HEX file.
5. Observe the progress indicator in the form of dots (.) which indicates that Flash loading is in process. When programming is completed successfully, the progress indicator displays *Completed*. If an error occurs, an error message is displayed.
6. Press Reset on the target board to execute the user code.

Summary

The Boot Loader application described in this Application Note is confined to the last two pages of the Flash memory, occupying 1000 bytes of memory. This Boot Loader facilitates field updates to the application/user program through the RS-232 port. In this way, products can be conveniently upgraded for features or functionality when required by upgrading the firmware at the site.

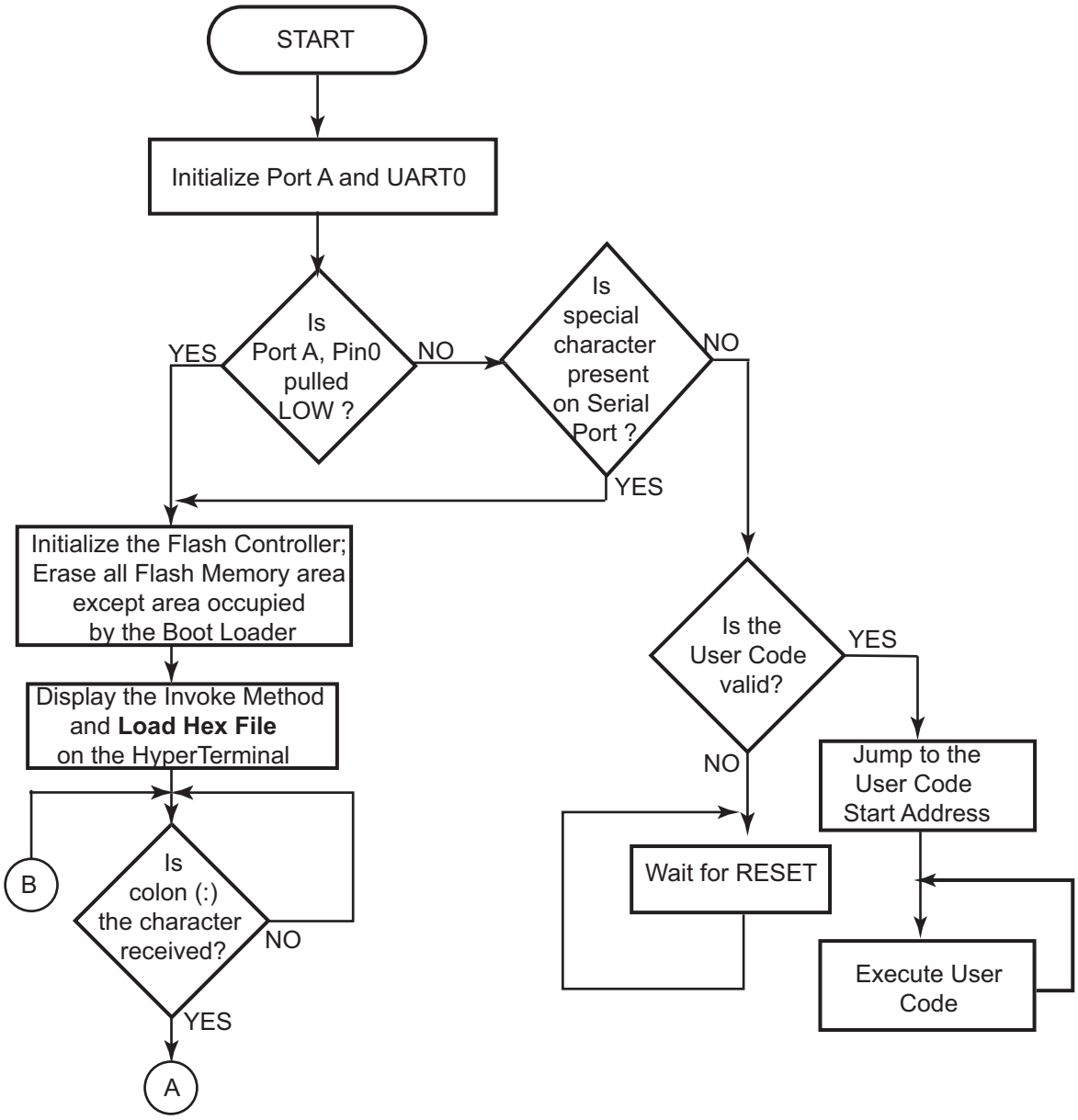
References

The document associated with Z8 Encore! XP products available for download at www.zilog.com are listed below.

- eZ8™ CPU Core User Manual (UM0128)
- Z8 Encore! XP® 64K Series Flash Microcontrollers Product Specification (PS0199)
- Z8 Encore! XP® 64K Series Flash Microcontroller Development Kit User Manual (UM0151)
- Zilog Developer Studio II (ZDS II)—Z8 Encore! User Manual (UM0130)

Appendix A—Flowcharts

The flowcharts for Boot Loader is displayed in Figure 3.



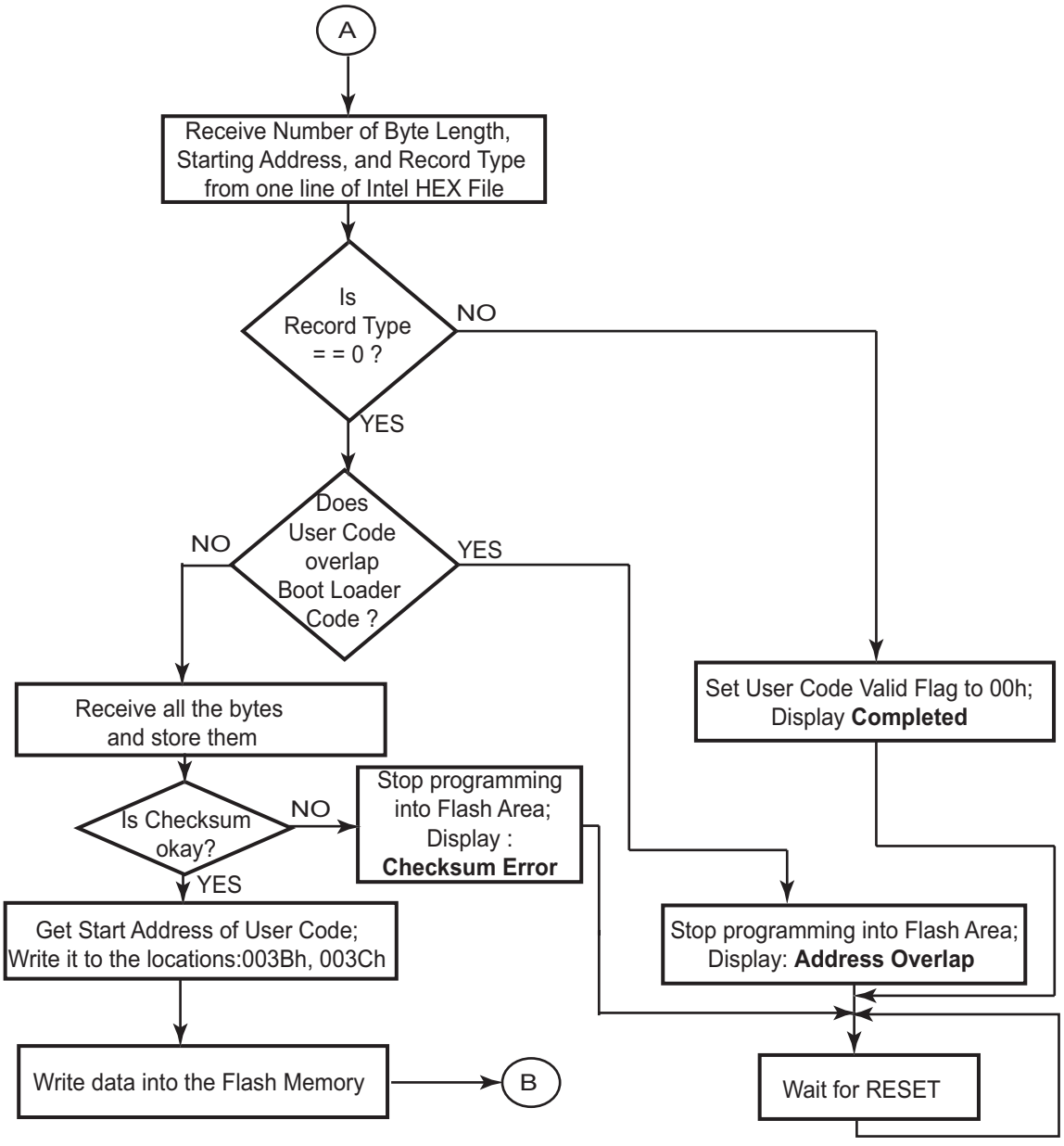


Figure 3. Flowchart for Boot Loader

Appendix B—Intel HEX File Format

The Boot Loader application programs a standard file format into Z8 Encore! XP® MCU's Flash memory. The Intel Standard HEX file is one of the popular and commonly used file formats. An Intel Standard HEX file is an ASCII file with one record per line. Table 3 lists the format for each file.

Table 3. 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 must be loaded into the chip. This is a value from 0 to 65,535 represented as a 4-digit hexadecimal number.
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 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 numbers.
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.

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