eZ80Acclaim!® MCUs

Flash Library APIs

Reference Manual

RM001305-0317

www.zilog.com

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

 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concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded.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.

Revision History

Each instance in the Revision History table below reflects a change to this document from its previous version.

Date	Revision Level	Description	Page No
Mar 2017	05	Complete re-write for ZDS-Acclaim! 5.3.0 and RZK/ZTP 2.5.0 release.	All
Sep 2010	04	Updated the copyright date and logos.	All
Apr 2006	03	Added Registered trademark for eZ80 and eZ80Acclaim!	All
Mar 2004	02	Deleted 'Preliminary' from the footers and on the title page.	All
Oct 2003	01	Original issue.	All

Table of Contents

Introduction

The eZ80Acclaim! and eZ80Acclaim Plus! families of microcontrollers and microprocessors can be used to access parallel NOR-Flash devices for storing data and/or executing code. To modify the contents of these external Flash memories, it is necessary to issue a sequence of individual commands instead of simply changing the contents of the target memory location(s) as done with RAM. Developers can choose to implement their own routines to manipulate external Flash, or use the Application Programming Interface (API) of the Zilog Flash Library (ZFL) to eliminate this task.

ZFL is designed to be used with Flash devices that support the Common Flash Memory Interface (CFI) Specification. The library can also be used with Flash devices that are not CFI-compliant if the application provides information regarding the device's geometry. For the eZ80F9x series of devices, the ZFL API also includes support for programming internal Flash and the Flash information page.

External Flash Overview

A Flash device that is erased will have all bits in all bytes within Flash set to 1. Programming Flash involves changing one or more bits from 1 to 0. Once a bit has been programmed (i.e. set to 0), it must be erased to set it back to 1. Flash devices typically do not allow individual bits or bytes to be erased. Instead, the device implements a command that can be used to erase the entire Flash (which can take tens of seconds, depending on the size of Flash), or a smaller unit of Flash referred to as an erase block, a sector, or a page. This document uses the term erase block to refer to the smallest unit of Flash that can be erased.

Due to the way Flash devices are erased, modifying a single byte of data within a programmed section of Flash involves copying the entire block to RAM, modifying the byte(s) of interest, erasing the corresponding

block in Flash, and the reprogramming the erased block. This process requires the programmer to create a lookup table containing the location of each erase block in external Flash to determine which block(s) must be updated. The lookup table is derived from information in the datasheet of the applicable Flash device. To support multiple Flash devices, it is necessary to create multiple lookup tables.

When multiple different Flash devices need to be supported, applications typically use the manufacturer and device ID codes of these Flash devices to select the appropriate lookup table containing the location of all erase blocks. However, this is often insufficient as some Flash vendors have device identification codes that can be one or more bytes long, but are not necessarily located in contiguous addresses within Flash, further complicating the process of selecting the appropriate lookup table.

The process of trying to obtain the Flash device's manufacturer and device ID codes is a non-trivial task because different Flash devices from different vendors use different commands to force the Flash memory into a mode of operation where these values can be obtained. Therefore, the Flash driver must first assume it knows the command set implemented by the external Flash device before it can obtain the manufacturer and device ID codes to identify the Flash device and hence, its implemented command set.

Common Flash Memory Interface

The Zilog Flash Library can significantly simplify the process of obtaining the Flash device's erase block and command set parameters if the device supports the Common Flash Memory Interface (CFI). Flash devices that are CFI-compliant implement a special *CFI query* mode of operation. When the device is in CFI Query mode, the host is able to obtain information about the size and location of every erase block within Flash, eliminating the need for the programmer to manually create a lookup table. CFI Query mode also allows the host to determine the com-

mand set that the Flash device implements to erase and program blocks of Flash.

All CFI-compliant devices use the same 1-cycle command sequence to place Flash in CFI query mode[1] regardless of Flash manufacturer or supported command set. Doing so allows ZFL to determine the device's geometry and command set in a consistent manner for all CFI-compliant devices without the use of lookup tables or assuming the device implements a particular command set.

Note: [1]There are exceptions. Some manufactures use multi-cycle command sequences to enter CFI Query mode contrary to the CFI specification. Such devices are not recognized as CFI-compliant by the ZFL and can only be supported using an application-provided lookup table.

Zilog Flash Library Limitations

Due to the extremely large number of external Flash devices, it is impossible to exhaustively test ZFL to ensure compatibility with every single device. That said, ZFL has been tested with each of the following Zilog development kits and modules utilizing the specified external Flash device.

Table 1. Development Kits Tested with Zilog Flash Library

eZ80 Development Kit/ Module	External Flash Drive	Notes
eZ80F91x150MODG	Spansion S29GL064N	CFI-compliant device
eZ80F910300KITG	Spansion S29GL064N	CFI-compliant device
eZ80L925148MODG	Spansion S29GL064N	CFI-compliant device

It is expected that ZFL will support other CFI-compliant devices not listed in the previous table subject to the following limitations:

- ZFL will only recognize a device as CFI-compliant if it can be placed in CFI Query mode using the 1-cycle command sequence described in the *Common Flash Memory Interface (CFI) Specification, Release 2.0 December 1, 2001* document.
  - Non-CFI compliant devices are only supported using an application-provided lookup table as described in the `XFL_Init` API.
- ZFL currently only supports the following command sets (as defined in the CFL specification):
  - AMD/Fujitsu Standard Command Set (identification code `0x0002`).
  - Intel Standard Command Set (code `0x0003`).
- ZFL and the eZ80Acclaim! and eZ80Acclaim Plus! families of devices only support 8-bit Flash devices (CFI driver interface code `0x0000`) and 16-bit Flash devices that can be configured to operate in 8-bit mode (CFI driver interface code `0x0002`).
- ZFL assumes that any Flash device that supports the Intel Standard command set lists the device's erase blocks regions from lowest physical Flash address to highest physical address such that the first erase block pertains to Flash offset `0x000000`.
- ZFL only supports AMD-compatible top-boot Flash devices if the CFI Query information includes a version 1.1 (or later) primary vendor extension table with the Top/Bottom Boot Sector Flag set to `0x03` (top boot).
  - AMD-compatible Flash devices typically list the geometry table as if the device were a bottom-boot device, even if the device is actually a top-boot device. Therefore, unless the AMD primary vendor extension table is present and indicates the device is a top-boot device, the ZFL driver will assume the device is bottom-

boot and will not automatically reverse the order of the erase blocks.

If a Flash device does not meet these requirements, the ZFL driver will not be able to determine the device's geometry or implemented command set without an application-provided lookup table (see the `XFL_Init` API for more information).

Zilog Flash Library Application Programming Interface

This section describes the Zilog Flash Library Application Programming Interface (API). For more information, click the API of interest in the table below. API routines prefixed with `XFL_` pertain to external Flash devices. API routines prefixed with `IFL_` pertain to internal Flash and are only available when using the Zilog Flash Library for the eZ80F9x series of microprocessors. API routines prefixed with `ZFL_` are applicable to both the internal and external Flash library routines. API routines prefixed with XFLD pertain to external Flash devices on select Zilog development kits.

Table 2. Zilog Flash Library API Routines

Zilog Flash Library API	Description
IFL_Erase	Erases all pages of internal Flash and optionally the information page
IFL_ErasePages	Erases one or more page of internal Flash
IFL_EraseInfoPage	Erases the entire information page
IFL_GetPage	Obtains the page number corresponding to an address within internal Flash
IFL_Init	Initializes the eZ80F9x Internal Flash Library
IFL_IsAddrValid	Determines if specified address range resides within internal Flash
IFL_IsInfoPageAddrValid	Determines if specified address range resides within the Information Page
IFL_PageErase	Erases the page of internal Flash containing the specified address

Table 2. Zilog Flash Library API Routines (Continued)

Zilog Flash Library API	Description
IFL_Program	Programs one or more bytes in internal Flash
IFL_ProgramInfoPage	Programs one or more bytes in the information page
IFL_Read	Reads one or more bytes of data from internal Flash
IFL_ReadInfoPage	Reads one or more bytes of data from the information page
XFL_BlockErase	Erases a single block of external Flash
XFL_EraseBlocks	Erases one or more contiguous blocks of external Flash
XFL_EraseDevice	Erase the entire external Flash device
XFL_GetDeviceInfo	Obtains basic information about the external Flash device
XFL_GetGeometry	Obtains information about the size of all erase blocks in the external Flash device
ZFL_GetVersion	Returns the Zilog Flash library version number
XFL_Init	Initializes the external Flash Library
XFL_Program	Programs one or more byte of data in the external Flash device
XFL_Read	Reads one or more bytes of data from internal Flash
XFL_ReadCFI	Reads CFI Query response data from the external Flash device

Table 2. Zilog Flash Library API Routines (Continued)

Zilog Flash Library API	Description
XFL_ResetEraseFlags	Sets the state of all erase block flags to the not-erased state
XFLD_Erase	Erases a block of external Flash on select Zilog development kits
XFLD_Program	Programs one or more bytes of external Flash on select Zilog development kits
XFLD_Query	Obtains external Flash manufacturer and device identification codes on select Zilog development kits
XFLD_Read	Copies one or more bytes of data from external Flash device to RAM memory buffer. Implemented as a macro that invokes the ZFL_Read API
ZFL_Read	Copies one or more bytes of data from external Flash device to RAM memory buffer

IFL_Init

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_Init
(
   UINT8          FlashProtect
);
```

Parameters:

FlashProtect: Value written to the eZ80F9x Flash write/erase protection register.

Return Value:

ZFL_ERR_SUCCESS is returned if no error occurs.

Description:

The `IFL_Init` routine must be called before any other ZFL API pertaining to internal Flash for proper operation of the library. This routine initializes the eZ80F9x internal Flash frequency divider assuming the target processor is operating at the highest supported system clock frequency.

The Flash write/erase protection register is initialized with the value of the FlashProtect parameter. Each bit in the FlashProtect register specifies whether the corresponding block of Flash should be protected from accidental write and/or erase operations. Calls made to the `IFL_Erase`, `IFL_ErasePage`, and `IFL_Program` APIs will fail if the targeted page(s) reside within a protection-block whose corresponding bit is set in the Flash write/erase protection register. For more information, please refer to the appropriate eZ80F9x product specification.

IFL_IsAddrValid

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_IsAddrValid
(
   HANDLE               hAddr,
   UINT24               Len
```

```
);
```

Parameters:

hAddr: 24-bit address to validate.

Len: 24-bit count of the number addresses (starting with `hAddr`) to include in the address range to be validated by this API.

Return Value:

ZFL_ERR_SUCCESS is returned if the address range specified by the `hAddr` and `Len` parameters resides within eZ80F9x internal Flash.

ZFL_ERR_ADDRESS is returned if any portion of the address range specified by the `hAddr` and `Len` parameters does not reside within eZ80F9x internal Flash.

Description:

This routine is used to test whether the entire address range corresponding to the `hAddr` and `Len` parameters, `hAddr` to (`hAddr` + `Len` -1), resides within eZ80F9x internal Flash.

IFL_IsInfoPageAddrValid

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_IsInfoPageAddrValid
(
   HANDLE     hAddr,
  UINT24     Len
);
```

Parameters:

hAddr: 24-bit address to validate.

Len: 24-bit count of the number addresses (starting with hAddr) to include in the address range to be validated by this API.

Return Value:

ZFL_ERR_SUCCESS is returned if the address range specified by the `hAddr` and `Len` parameters resides within the eZ80F9x Flash information page.

ZFL_ERR_ADDRESS is returned if any portion of the address range specified by the `hAddr` and `Len` parameters does not reside within the eZ80F9x Flash information page.

Description:

This routine is used to test whether the entire address range corresponding to the hAddr and Len parameters, hAddr to (hAddr + Len -1), resides within the eZ80F9x Flash information page.

IFL_GetPage

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
UINT8
IFL_GetPage
(
   HANDLE      hAddr
);
```

Parameters:

hAddr: 24-bit internal Flash address to be converted to a page number.

Return Value:

Page Number: 8-bit value indicating the 0-based page number containing the address corresponding to the hAddr parameter.

Description:

The `IFL_GetPage` API is used to obtain the page number corresponding to an address in eZ80F9x internal Flash. This routine performs no validation of the hAddr parameter and will return meaningless values if the hAddr parameter does not reside within internal Flash. The `IFL_IsAddrValid` API should be used to determine if the target address is within the address space of internal Flash before this API is called, if the validity of the address is unknown.

The range of values returned by this API is dependent upon which eZ80F9x series Zilog Flash Library is linked with the application. There are 128 pages of internal Flash on the eZ80F91 and eZ80F92 microcontroller. Consequently, this API will return a value between 0 and 127 when running on either microcontroller. The eZ80F93 has 64 pages of internal Flash and therefore, this API will return a value between 0 and 63 when executed on an

eZ80F93 device. Each page of Flash on the eZ80F91 is 2 KB in size while Flash pages on the eZ80F92 and eZ80F93 MCUs are each 1 KB in size.

IFL_Erase

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_Erase
(
   BOOL     EraseIP
);
```

Parameters:

EraseIP: Boolean parameter that specifies whether the information page should also be erased when erasing all pages of eZ80F9x internal Flash.

Return Value:

ZFL_ERR_SUCCESS is returned if all pages of internal Flash (and optionally the information page) are successfully erased.

ZFL_ERR_ERASE is returned if one or more bits in the Flash write/erase protection register is set.

Description:

The `IFL_Erase` API erases all pages of eZ80F9x internal Flash memory, resetting all bits to the non-programmed state (binary value of 1). If the `EraseIP` parameter is non-zero (TRUE), then, in addition to erasing internal Flash, the information page is also erased.

The erase operation will fail if the write-protect pin (nWP) available on the eZ80F91 is asserted, or if any of the 8 protection-blocks is in the write-protect

state (i.e. the corresponding bit in the Flash write/erase protection register is set).

IFL_ErasePages

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_ErasePage
(
   HANDLE   hAddr,
    UINT8    NumPages
);
```

Parameters:

hAddr: Address within eZ80F9x internal Flash of the first page to be erased.

NumPages: Specifies the number of pages of internal Flash to be erased starting with the page corresponding to `hAddr`.

Return Value:

ZFL_ERR_SUCCESS is returned if the specified number of pages of internal Flash are successfully erased.

ZFL_ERR_ADDRESS is returned if the address passed in the hAddr parameter does not reside within eZ80F9x internal Flash or an attempt is made to erase a page beyond the end of internal Flash.

ZFL_ERR_ERASE is returned if one or more pages of Flash could not be erased.

Description:

The `IFL_ErasePages` API erases a contiguous subset of the pages within eZ80F9x internal Flash. The `hAddr` parameter specifies an address within the first page of internal Flash to be erased and the `NumPages` parameter specifies the total number of contiguous pages to erase, starting with the page containing `hAddr`.

The erase operation will fail if an attempt is made to erase a page that does not reside within internal Flash, or an attempt is made to erase a page that is within a protection-block whose write-protect bit is set, or if an attempt is made to erase a page within the eZ80F91 boot-block (first 32 KB of internal Flash on the eZ80F91) and the write protect pin (nWP) is asserted.

IFL_EraseInfoPage

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_EraseInfoPage
(
   void
);
```

Parameters:

None

Return Value:

ZFL_ERR_SUCCESS is returned if the information page is successfully erased.

Description:

The `IFL_EraseInfoPage` API erases both rows of the eZ80F9x information page setting all bits to 1. After this API has been called, the `IFL_ProgramInfoPage` API can be called to program bytes within the information page.

IFL_PageErase

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_ErasePage
(
 HANDLE   hAddr
);
```

Parameters:

hAddr: Address within eZ80F9x internal Flash of the page to be erased.

Return Value:

ZFL_ERR_SUCCESS is returned if the internal Flash page containing hAddr is successfully erased.

ZFL_ERR_ADDRESS is returned if the address passed in the `hAddr` parameter does not reside within eZ80F9x internal Flash.

ZFL_ERR_ERASE is returned if the target page of Flash could not be erased.

Description:

The `IFL_PageErase` API is used to erase the page of Flash containing the specified target address (`hAddr`). The `hAddr` parameter does not have to be aligned to the start of a page of eZ80F9x internal Flash.

The erase operation will fail if an attempt is made to erase a page that does not reside within internal Flash, or an attempt is made to erase a page that is within a protection-block whose write-protect bit is set, or if an attempt is made to erase a page within the eZ80F91 boot-block (first 32 KB of internal Flash on the eZ80F91) and the write protect pin (nWP) is asserted.

IFL_Program

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_Program
(
   HANDLE   hDst,
   HANDLE   hSrc,
   UINT24   Len
);
```

Parameters:

hDst: Address of first byte of internal Flash to be programmed.

hSrc: References memory buffer containing the data to be programmed into internal Flash.

Len: Number of bytes of data to be programmed into internal Flash.

Return Value:

ZFL_ERR_SUCCESS is returned if all bytes of data were successfully programmed into ez80F9x internal Flash.

ZFL_ERR_ADDRESS is returned if an attempt is made to program a memory location that does not reside within internal Flash.

ZFL_ERR_WRITE is returned if an attempt is made to program a byte of data within a protected block.

ZFL_ERR_VERIFY is returned if the value programmed into internal Flash does not match the corresponding value in the array referenced by `hSrc`.

Description:

The `IFL_Program` API is used to program one or more contiguous bytes of eZ80F9x internal Flash. Once a bit of Flash has been programmed (set to 0), it cannot be set to 1 using the `IFL_Program` API. Instead, the page containing the programmed bit(s) needs to be erased (all bits set to 1) using either the `IFL_Erase` or `IFL_ErasePage` API.

The programming operation will fail if any of the addresses targeted for programming are not within the address space of eZ80F9x internal Flash, or if the page containing the target addresses is write-protected. When a bit in the flash write/erase protection register is set, the corresponding set of 16 internal Flash pages is write-protected and this API cannot be used to program any memory locations within the protected pages. Refer to the IFL_Init API description for more information.

If the eZ80F91 write protect pin (nWP) is asserted, it is not possible to call this API to program any locations within the first 32KB of internal Flash regardless of whether bit 0 in the Flash write/erase protection register is set.

IFL_ProgramInfoPage

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_ProgramInfoPage
(
   HANDLE   hDst,
   HANDLE   hSrc,
   UINT24   Len
);
```

Parameters:

hDst: Offset of the first byte in the eZ80F9x information page to be programmed.

hSrc: References memory buffer containing the data to be programmed into the information page.

Len: Number of bytes of data to be programmed into the information page.

Return Value:

ZFL_ERR_SUCCESS is returned if all bytes of data were successfully programmed into the eZ80F91 Flash information page.

ZFL_ERR_ADDRESS is returned if an attempt is made to program a memory location that does not reside within the information page.

Description:

The `IFL_ProgramInfoPage` API is used to program contiguous bytes of data in the eZ80F9x Flash information page. Once a bit of Flash in the information page has been programmed (set to 0), it cannot be set to 1

using the `IFL_ProgramInfoPage` API. Instead, the entire information page must be erased using either the `IFL_Erase` or `IFL_EraseInfoPage` API.

IFL_Read

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_Read
(
   HANDLE   hDst,
   HANDLE   hSrc,
   UINT24   Len
);
```

Parameters:

hDst: References memory buffer into which data read from internal Flash is copied.

hSrc: Address of the first byte of data to be read from internal Flash.

Len: Number of bytes of data to read.

Return value:

ZFL_ERR_SUCCESS is returned if the data is successfully read from internal Flash.

ZFL_ERR_ADDRESS is returned if an attempt is made to read a memory location that does not reside within Internal Flash.

Description:

The `IFL_Read` API can be used to read multiple contiguous bytes of data from eZ80F9x internal Flash. Applications that do not require the Flash Library to validate the source address range can call the `memcpy` API to read data from internal Flash instead of using this API.

IFL_ReadInfoPage

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
IFL_Read
(
   HANDLE   hDst,
   HANDLE   hSrc,
   UINT24   Len
);
```

Parameters:

hDst: References memory buffer into which data read from the information page is copied.

hSrc: Offset of the first byte of data in the information page to be read.

Len: Number of bytes of data to read from the information page.

Return value:

ZFL_ERR_SUCCESS is returned if the information page data is successfully read.

ZFL_ERR_ADDRESS is returned if an attempt is made to read a memory location that does not reside within the information page.

Description:

The `IFL_ReadInfoPage` API can be used to read multiple contiguous bytes of data from the eZ80F9x Flash information page. Application programs typically use the information page to store program parameters independently of the application image. Data can be programmed into the information page using the `IFL_ProgramInfoPAge` API.

XFL_Init

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFL_Init
(
  UINT8          ChipSelect,
  NON_CFI_DEV      * pNonCFI
);
```

Parameters:

ChipSelect: Specifies the chip select of the external Flash device. Typically, a value of 0 is used to indicate Chip Select 0 (CS0).

pNonCFI (input): Optional parameter that specifies the set of non-CFI-compatible Flash devices the XML should support. If only CFI-compliant devices (subject to the limitations described in Zilog Flash Library Limitations) will be used, set this parameter to `NULLPTR` to indicate that support for non-CFI-compliant Flash devices is not required.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_INVALID_PARAMETER is returned if the specified chip select is not configured for a memory-mapped device or is not enabled.

ZFL_ERR_UNSUPPORTED_CMDSET indicates that the device is CFI-compliant, but implements a command set that is not supported by ZFL.

ZFL_ERR_UNSUPPORTED_DEVICE indicates the external Flash device is not CFI-compliant, and is either not on the list of non-CFI-compliant devices referenced by the `pNonCFI` parameter, or does not implement a supported command set.

ZFL_ERR_TOO_MANY_ERASE_BLOCKS is returned if the external Flash contains more than `MAX_EB_STATUS_BYTES`*8 erase-blocks. Only applicable when `MAX_EB_STATUS_BYTES` is non-zero.

ZFL_ERR_FAILURE is returned if ZFL is unable to read the device's manufacturer and device ID codes.

Description:

The `XFL_Init` API should be called before calling any other XFL API for proper operation of the External Flash Library.

Typically, the target HW platform is designed such that CS0 is connected to the external Flash device. Therefore, in most situations, the `ChipSelect` parameter should be set to 0. However, if there is a secondary Flash device on the target HW platform, the value of the `ChipSelect` parameter should match the target chip select signal. The chip select parameter must always be between 0 and 3 since the eZ80Acclaim! and eZ80Acclaim Plus microcontrollers have only four chip select signals.

By default, ZFL can only be used with one external Flash device at a time. After completing all Flash operations on chip select `x` (`CSx`), the `XFL_Init` API can be called again to initiate Flash operations on chip select `y` (`CSy`). If it is necessary to use ZFL with multiple Flash devices simultaneously, refer to the Supporting Flash Devices on Multiple Chip Selects section for more information.

Subject to the limitations described in the Zilog Flash Library Limitations section, the Zilog Flash Library will support most CFI-compliant AMD– and Intel-compatible Flash devices. However, older designs that use non-CFI-compliant devices cannot be used with the ZFL driver unless that application provides information about the device's geometry in the list referenced by the `pNonCFI` parameter. The devices referenced by the `pNonCFI` parameter are also required to implement one of the command sets supported by ZFL as determined by entries in the `CmdSetTable`. For more information on supporting non-CFI Flash devices, refer to the description of the NON_CFI_DEV Structure and the Adding Additional Command Sets advanced topic.

If the Flash device is successfully initialized, this API will return a value of `ZFL_ERR_SUCCESS`, allowing other external Flash API functions to be called. If this API does not return a value of `ZFL_ERR_SUCCESS`, the application should not attempt to call any other external Flash API.

ZFL_GetVersion

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
UINT16
ZFL_GetVersion
(
   void
);
```

Parameters:

None

Return value:

Major | Minor: The Zilog Flash Library version number as a 16-bit unsigned integer.

Description:

This API is used to obtain the 16-bit version number of the Zilog Flash Library API. The upper byte of the version number is the major version number and the lower byte contains the minor version number. As an example, a return value of `0x0200` indicates version 2.0 of the Zilog Flash Library.

XFL_GetDeviceInfo

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8

XFL_GetDeviceInfo

(

   XFL_DEVICE_INFO * pDev

);
```

Parameters:

pDev (output): Pointer to an `XFL_DEVICE_INFO` data structure the library initializes with information about the underlying Flash device and chip select.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_INVALID_PARAMETER is returned if the specified chip select is not configured for a memory-mapped device or is not enabled, or if the `pDev` parameter is 0.

Description:

This API is used to obtain information about the external Flash device, such as the size (in bytes) of Flash, the manufacturer and device ID codes, and the command set implemented by the external Flash controller. For more information about all of the parameters obtained from this API, refer to the description of the XFL_DEVICE_INFO Structure.

XFL_GetGeometry

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFL_GetGeometry
(
  XFL_DEVICE_INFO   * pDevInfo,
  UINT8             * pNumRegions,
  CFI_REGION        * pRegions
);
```

Parameters:

pDevInfo (output): Pointer to an XFL_DEVICE_INFO data structure the library initializes with information about the underlying Flash device and chip select.

pNumRegions (input, output): On input, indicates the maximum number of (uninitialized) entries in the array of `CFI_REGION` structures referenced by the `pRegions` parameter. On output, indicates the number of erase block regions in the external Flash device.

pRegions (output): Upon successful return, the array referenced by the `pRegions` parameter contains information about each erase block region in the external Flash device.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_INVALID_PARAMETER is returned if the `XFL_Init` API was not called, or did not succeed, or if either of the `pNumRegions` or `pRegions` parameters is 0.

Description:

This API is used to obtain information about each erase block region in the external Flash device. Each region contains one or more erase blocks that are the smallest unit of Flash that can be erased using the `XFL_EraseBlocks` API.

A CFI-compliant Flash device can contain up to 255 regions requiring a maximum of 255 entries in the array referenced by the `pRegions` parameter. However, Flash devices typically only contain a few regions, so it is not necessary for the caller to allocate a maximum sized buffer to store the theoretical maximum number of erase block regions in external Flash. Consequently, the 8-bit value referenced by the `pNumRegions` parameter on input to this API should be set to the maximum number of `CFI_REGION` data structures that can be stored in the array referenced by the `pRegions` parameter.

The actual number of `CFI_REGION` data structures ZFL stores in the array will be the smaller of the actual number of erase block regions in external Flash, and the maximum size of the `pRegions` array as determined by the `pNumRegions` parameter on input. Upon return, the 8-bit value referenced by the `pNumRegions` output parameter will contain the actual number of erase block regions in the external Flash device, regardless of how many entries were written to the `pRegions` array. For more information, refer to the description of the CFI_REGION Structure.

XFL_BlockErase

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFL_BlockErase
(
   HANDLE   hAddr
);
```

Parameters:

hAddr: Arbitrary pointer referencing any memory location within the block of external Flash to be erased.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_INVALID_PARAMETER is returned if the `XFL_Init` API was not called, or did not complete successfully.

ZFL_ERR_ADDRESS is returned if the `hAddr` parameter is not located within the external Flash device.

ZFL_ERR_ERASE is returned if one or more blocks of Flash are not successfully erased.

Description:

This API is used to erase a single block of external Flash. `hAddr` is an arbitrary pointer to any memory location within the address space assigned to the external Flash device, as determined by the setting of corresponding chip select upper and lower bound registers. It is not necessary for `hAddr` to reference the first memory location within the block to be erased. The external Flash controller will erase the entire (erase) block in which `hAddr` resides.

ZFL disables interrupts while the external Flash device is erased. This is done to prevent software failure that can occur if an interrupt is processed while the external Flash device is not in read-array mode.

XFL_EraseDevice

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFL_EraseDevice
(
   void
);
```

Parameters:

None

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_INVALID_PARAMETER is returned if the `XFL_Init` API was not called, or did not complete successfully.

ZFL_ERR_ERASE is returned if one or more blocks of Flash are not successfully erased.

ZFL_ERR_FAILURE is returned for Flash devices implementing the Intel standard command set with the default build of the Flash Library.

Description:

This API sets every bit in every erasable block of external Flash in the non-programmed state (binary 1). Depending on the storage capacity of Flash, the erase operation could take tens of seconds (or more) to complete. The `XFL_EraseDevice` API does not return to the caller until all blocks of Flash have been successfully erased, or until the operation is aborted due to an error.

ZFL disables interrupts while the external Flash device is erased. This is done to prevent software failure that can occur if an interrupt is processed while the external Flash device is not in read-array mode. Be aware the if external Flash is erased and Flash was mapped to address `0x0`, then interrupts that occur on non-eZ80F91 devices (including NMI and/or reset vectors) could cause the system to fail to operate as expected. Use caution when calling this API since all application code residing in external Flash will get destroyed as the Flash is erased and can no longer be executed by the eZ80 CPU.

Note: Flash devices that implement the Intel standard command set do not include a command capable of erasing the entire Flash device in one operation. If the Zilog Flash Library is built with erase-flag processing support (as described in the Enabling Erase-Flag Processing section), ZFL will

erase the Intel Flash device by issuing multiple commands to erase one block of Flash at a time. However, the default build of ZFL does not include support for erase-block processing and therefore this API will return `ZFL_ERR_FAILURE` when attempting to erase Flash devices implementing the Intel standard command set with the default build of ZFL.

XFL_EraseBlocks

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFL_EraseBlocks
(
   HANDLE   hAddr,
   UINT24   NumBlocks
);
```

Parameters:

hAddr: Arbitrary pointer referencing the first (or only) erase block to be erased.

NumBlocks: The number of consecutive (erase) blocks to erase beginning with the erase block referenced by `hAddr`.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_INVALID_PARAMETER is returned if the `XFL_Init` API was not called, or did not complete successfully.

ZFL_ERR_ADDRESS is returned if the `hAddr` parameter is not located within the external Flash device or one of the blocks implied by the `NumBlocks` parameter lies outside the address spaces of the external Flash device.

ZFL_ERR_ERASE is returned if one or more blocks of Flash are not successfully erased.

Description:

This API is not included in the default build of the Zilog Flash Library. The `XFL_EraseBlocks` API is only available if the library was compiled with the `MAX_EB_STATUS_BYTES` macro set to a non-zero value to enable erase-flag processing as described in the Modifying the Maximum Number of Erase Blocks section.

The `XFL_EraseBlocks` API is used to erase a subset of the total number of erase blocks in external Flash. `hAddr` is an arbitrary pointer to any memory location within the address space assigned to the external Flash device, as determined by the setting of corresponding chip select upper and lower bound registers. It is not necessary for `hAddr` to reference the first memory location of the first erase block to be erased. The external Flash controller will erase the entire (erase) block in which `hAddr` resides.

If the `NumBlocks` parameter is 0, this API does nothing. If `NumBlocks` is non-zero, it specifies the number of consecutive erase blocks, beginning with the block referenced by `hAddr` to be erased. It is not necessary for the consecutive erase blocks to reside in the same erase block region (i.e. it is not necessary for all erase blocks targeted by the command to have the same size). As each block is erased, the ZFL automatically advances `hAddr` to the start of the next block and attempts to erase that block. If at

any point during the process of erasing all `NumBlocks`, `hAddr` references an erase block not located within the address space of the underlying chip select, then the erase operation is aborted and a return code of `FL_ERR_ADDRESS` is returned.

ZFL disables interrupts while the external Flash device is erased. This is done to prevent software failure that can occur if an interrupt is processed while the external Flash device is not in read-array mode.

XFL_Program

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8

XFL_Program

(

   HANDLE  hDst,

   HANDLE  hSrc,

   UINT24  Len

);
```

Parameters:

hDst: Arbitrary pointer referencing the first (or only) byte in Flash to be programmed.

hSrc: Arbitrary pointer referencing an area of memory containing the data value(s) to be programmed into Flash.

Len: The number of bytes of data from the buffer referenced by `hSrc` to be programmed into external Flash starting at the memory location referenced by `hDst`.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_INVALID_PARAMETER is returned if the `XFL_Init` API was not called, or did not complete successfully.

ZFL_ERR_ADDRESS is returned if the memory location referenced by the `hDst` parameter is not located within the external Flash device, or one of the addresses implied by the `Len` parameter lies outside the address spaces of the external Flash device.

ZFL_ERR_WRITE is returned if a failure occurs while programming any of the locations in external Flash targeted by this command.

Description:

This API is used to change one or more bits in one or more consecutive bytes of external Flash from the erased state (binary 1) to the programmed state (binary 0). This API cannot be used to change the state of any programmed bits to the erased state. Erasing bits of external Flash can only be accomplished using the `XFL_EraseDevice`, `XFL_BlockErase`, or `XFL_EraseBlocks` API.

The Zilog Flash Library disables interrupts while programming the external Flash device to prevent application interrupt handlers from attempting to access the Flash device while it is not in read-array mode. It is only possible to execute code from within external Flash or read application data when the Flash is in read-array mode.

XFL_Read

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8

XFL_Read

(

   HANDLE   hDst,

   HANDLE   hSrc,

   UINT24   Len

);
```

Parameters:

hDst: References memory buffer into which data read from external Flash is copied.

hSrc: Address of the first byte of data to be read from external Flash.

Len: Number of bytes of data to read.

Return value:

ZFL_ERR_SUCCESS is returned if the data is successfully read from internal Flash.

ZFL_ERR_ADDRESS is returned if an attempt is made to read a memory location that does not reside within Internal Flash.

ZFL_ERR_INVALID_PARAMETER is returned if the `XFL_Init` API was not called, or did not complete successfully.

Description:

The `XFL_Read` API can be used to read multiple contiguous bytes of data from external Flash. Applications that do not require the Flash Library to validate the source address range can call the `memcpy` API to read data from external Flash instead of using this API.

XFL_ResetEraseFlags

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFL_ResetEraseFlags
(
   void
);
```

Parameters:

None

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_INVALID_PARAMETER is returned if the `XFL_Init` API was not called, or did not complete successfully.

Description:

This API is not included in the default build of the Zilog Flash Library. The `XFL_ResetEraseFlags` API is only available if the library was compiled with the `MAX_EB_STATUS_BYTES` macro set to a non-zero value to enable erase-flag processing as described in the Modifying the Maximum Number of Erase Blocks section.

When the ZFL has been compiled to support erase-flag processing, the `XFL_ResetEraseFlags` API is used to reset all erase-flags to the non-erased state (binary value of 0). This API does not erase any blocks of external Flash. For more information, refer to the Enabling Erase-Flag Processing section.

XFL_ReadCFI

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFL_ReadCFI
(
   UINT24  Offset
   HANDLE  hDst,
   UINT16  Len
);
```

Parameters:

Offset: Specifies the offset of the first byte of data to be read from the CFI Query table.

hDst: Arbitrary pointer that references a buffer in RAM into which data read from the CFI Query table is written.

Len: Number of bytes of data to read from the CFI Query table (starting at the specified Offset) and written into the memory buffer referenced by `hDst`.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_INVALID_PARAMETER is returned if the `XFL_Init` API was not called, or did not complete successfully or if the external Flash device is not CFI-compliant.

Description:

This API is used to read data from the CFI query table. Flash devices that are not CFI-compliant typically do not contain a CFI query table that can be read using this API. Therefore, an error is returned if this API is called to read CFI query table data from a device that is not CFI compliant.

The default starting offset of the CFI query table is `CFI_OFS_QRY_ID` (`0x010`) as defined in `CFI.h`. Applications can optionally use this command to obtain the standard low-level CFI information for the external Flash device such as programming voltage minimum and maximum values or vendor-specific information contained in an optional vendor-specified extended query table by supplying the appropriate `Offset` and `Len` parameters.

The Zilog Flash Library disables interrupts while reading data from the CFI query table to prevent application interrupt service routines from attempting to access the external Flash while it is in CFI query mode of operation. Program code can only execute from external Flash when Flash is in read-array mode.

XFLD_Erase

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFLD_Read
```

```
(
   HANDLE   hAddr
);
```

Parameters:

hAddr: Arbitrary pointer referencing any memory location within the block of external Flash to be erased.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_ERASE is returned if the block of Flash containing hAddr is not erased.

Description:

This API is used to erase a single block of external Flash. `hAddr` is an arbitrary pointer to any memory location within the address space assigned to the external Flash device (as determined by the setting of corresponding chip select upper and lower bound registers). It is not necessary for `hAddr` to reference the first memory location within the block to be erased. The external Flash controller will erase the entire (erase) block in which `hAddr` resides.

ZFL disables interrupts while the external Flash device is erased. This is done to prevent software failure that can occur if an interrupt is processed while the external Flash device is not in read-array mode.

XFLD_Program

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFLD_Program
```

```
(
   HANDLE  hDst,
   HANDLE  hSrc,
   UINT24  Len
);
```

Parameters:

hDst: Arbitrary pointer referencing the first (or only) byte in Flash to be programmed.

hSrc: Arbitrary pointer referencing an area of memory containing the data value(s) to be programmed into Flash.

Len: The number of bytes of data from the buffer referenced by `hSrc` to be programmed into external Flash starting at the memory location referenced by `hDst`.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_VERIFY is returned if an Intel Flash device was programmed without error but one or more bytes read back from external Flash did not match the corresponding value in the buffer referenced by `hSrc`.

ZFL_ERR_WRITE is returned if a failure occurs while programming any of the locations in external Flash targeted by this command.

Description:

This API is used to change one or more bits in one or more consecutive bytes of external Flash from the erased state (binary 1) to the programmed state (binary 0). This API cannot be used to change the state of any programmed bits to the erased state.

The Zilog Flash Library disables interrupts while programming the external Flash device to prevent application interrupt handlers from attempting to access the Flash device while it is not in read-array mode. It is only possible to execute code from within external Flash or read application data when the Flash is in read-array mode.

XFLD_Query

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
UINT16
XFLD_Query
(
   HANDLE   hAddr
);
```

Parameters:

hAddr: Arbitrary pointer referencing any memory location within the address space of the external Flash device to be queried.

Return value:

Man_ID | Dev_ID is returned if no error occurs.

Description:

This API is used to obtain the 8-bit manufacturer identification code (`Man_ID`) and 8-bit device identification code (`Dev_ID`) of the external Flash device containing the `hAddr` parameter.

ZFL disables interrupts while the external Flash device is queried. This is done to prevent software failure that can occur if an interrupt is processed while the external Flash device is not in read-array mode.

XFLD_Read

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
XFLD_Read
(
   HANDLE   hDst,
   HANDLE   hSrc,
   UINT24   Len
);
```

Parameters:

hDst: References memory buffer into which data read from external Flash is copied.

hSrc: Address of the first byte of data to be read from external Flash.

Len: Number of bytes of data to read.

Return value:

ZFL_ERR_SUCCESS is always returned.

Description:

The XFLD_Read API can be used to read multiple contiguous bytes of data external Flash. Internally this routine calls the C run-time-library `memcpy` API to read data from external Flash.

ZFL_Read

Header File:

```
#include "FlashLib.h"
```

Prototype:

```
INT8
ZFL_Read
(
   HANDLE   hDst,
   HANDLE   hSrc,
   UINT24   Len
);
```

Parameters:

hDst: References memory buffer into which data read from external Flash is copied.

hSrc: Address of the first byte of data to be read from external Flash.

Len: Number of bytes of data to read.

Return value:

ZFL_ERR_SUCCESS is always returned.

Description:

The ZFL_Read API can be used to read multiple contiguous bytes of data external Flash. Internally this routine calls the C run-time-library `memcpy` API to read data from external Flash.

Advanced Topics

ZFL Code Segment Names

The default build of the Zilog Flash Library uses a mix of standard and non standard segment names to distinguish between segments that may require additional linker directives to be added to the project settings based on build configuration. The ZFL code segments are described in this section.

CODE

The default ZFL code segment is named CODE and contains routines that do not directly erase or program external Flash, or otherwise place external Flash into a mode that does not support normal read-access to Flash. On the eZ80F91, the default ZFL code segment also contains routines that manipulate the internal Flash information page.

The name of the ZFL default code segment can be changed to `ZFL_CODE` by uncommenting the `ZFL_CODE_SEG` definition in `FlashLib.h` and rebuilding the appropriate Zilog Flash Library as described in Building the Zilog Flash Library. In this instance, applications will typically need to include a linker directive to control the placement of the `ZFL_CODE` code segment. This step is typically not required when the default ZFL code segment is named `CODE`.

ZFL_nEXT_Flash

ZFL routines that directly program or erase external Flash and ZFL routines that place the external Flash in a state which does not support normal read access are grouped in a code segment named `ZFL_nEXT_Flash`. Because it is not possible to execute code from external Flash unless external Flash is in normal read-access mode, the `ZFL_nEXT_Flash` segment must not be located in external Flash. Applications that manipulate external Flash and are built using the Flash or Copy to RAM build config-

uration typically require additional linker directives to control the placement of the ZFL_nEXT_Flash code segment.

ZFL_nINT_Flash

On the eZ80F92 and eZ80F93 microcontrollers, ZFL routines that manipulate the internal Flash information page are grouped in a code segment named ZFL_nINT_Flash. Because it is not possible to execute code from eZ80F92/eZ80F93 internal Flash that manipulates the information page, this code segment must not be located within eZ80F92/eZ80F93 internal Flash. Applications targeting the eZ80F92/eZ80F93 using the Flash or Copy to RAM build configurations will typically need to include a linker directive to control the placement of the ZFL_nINT_Flash code segment.

On the eZ80F91, it is possible to access the Flash information page from code executing in internal Flash. Therefore, the only ZFL routine contained in the ZFL_nINT_Flash code segment in the eZ80F91 Flash library is the IFL_Erase Flash API that erases all of internal Flash (and optionally the Flash information page) which requires that the IFL_Erase routine must not be located in internal Flash.

The IFL_Erase API is always located in the ZFL_nINT_Flash code section regardless of which eZ80F9x Flash library is linked to the application. Therefore, applications that call the IFL_Erase API may require additional linker directives to ensure the ZFL_nINT_Flash code segment is not located within internal Flash.

Linking the Zilog Flash Library

Applications that link the Zilog Flash Library can be built using one of the standard build configurations supported in the ZDS IDE – RAM, Flash, or Copy to RAM. Depending on which build configuration is used, it may be necessary to add linker directives to the project settings to locate non-standard ZFL code segments into select areas of memory.

RAM Build Configuration

When the RAM build configuration is used, all ZFL routines execute from RAM. Typically, there are no restrictions on where ZFL routines are located within RAM (for example, internal versus external RAM) and it is not necessary to add linker directives to the project settings when the RAM build configuration is used.

Flash Build Configuration

When the Flash build configuration is used, all ZFL routines execute from Flash and applications that need to erase/program external Flash or access the eZ80F92/eZ80F93 information page using the ZFL API will typically need to include directives to control the placement of one or more of the ZFL code segments. Typically, the `ZFL_nEXT_Flash` code segment will need to be located in RAM or internal Flash (not applicable to eZ80L92-based applications) and the `ZFL_nINT_Flash` code segment will need to be located in RAM or external Flash (only required with projects targeting the eZ80F92/eZ80F93).

The following procedure can be used with projects configured to use the Flash build configuration to add the necessary linker directives to locate select ZFL routines in internal Flash (eZ80F9x) or RAM (eZ80L92 or eZ80F9x).

1. From the **Project** menu, select **Settings** and click **Commands** under the Linker heading on the left side of the **Project Settings** dialog.
2. On the right side of the **Project Settings** dialog, ensure that the **Always Generate From Settings** radio button is enabled and that the **Additional Directives** checkbox is selected.
3. Click the **Edit…** button. Select between one of the following options (a through d) depending on which eZ80 device is being targeted.

   a) For eZ80F91, enter the following directive to locate ZFL routines that program or erase external Flash in internal Flash:

   ```
   RANGE ZFL_nEXT_Flash $0: $03FFFF
   ```

b) For eZ80F92, enter the following directives to locate ZFL routines that program or erase external Flash in internal Flash and ZFL routines that manipulate the Flash information page in the first 1 MB of external Flash between `0x100000` and `0x1FFFFF`:

```
RANGE ZFL_nEXT_Flash $0: $01FFFF
RANGE ZFL_nINT_Flash $100000: $1FFFFF
```

c) For eZ80F93, enter the following directives to locate ZFL routines that program or erase external Flash in internal Flash and ZFL routines that manipulate the Flash information page in the first 1 MB of external Flash between `0x100000` and `0x1FFFFF`:

```
RANGE ZFL_nEXT_Flash $0: $00FFFF
RANGE ZFL_nINT_Flash $100000: $1FFFFF
```

d) For eZ80L92, enter the following directive to locate ZFL routines that program or erase external Flash in RAM:

```
CHANGE ZFL_nEXT_Flash is DATA
```

4. After adding the necessary linker directives, click **OK** to close the Additional Linker Directives dialog and then click **OK** again to close the Project Settings dialog.
5. If prompted to rebuild the project, click **Yes**; otherwise, click **Rebuild All** or the Rebuild All icon in the **Build** menu.

Copy to RAM build Configuration

When the Copy to RAM build configuration is used, the ZDS default CODE segment is copied from external and/or internal Flash to RAM at startup allowing routines in the CODE segment to execute from RAM. However, ZFL routines that are not located in the default CODE segment (such as those routines in the `ZFL_nEXT_Flash`, `ZFL_nINT_Flash`, and if used, the `ZFL_CODE` segments) will remain resident in Flash unless the project settings are modified to include linker directives that explicitly relocate non-default ZFL code segments into RAM.

The following procedure can be used with projects configured to use the Copy to RAM build configuration to add the necessary linker directives to locate select ZFL code segments in RAM.

1. From the **Project** menu, select **Settings** and click **Commands** under the Linker heading on the left side of the **Project Settings** dialog.
2. On the right side of the Project Settings dialog, ensure that the **Always Generate From Settings** radio button is enabled and that the **Additional Directives** checkbox is checked.
3. Click the **Edit…** button. Select between one of the following options (a through d), depending on which eZ80 device is being targeted.

   a) For eZ80F91, enter the following directive to locate ZFL routines that program or erase external Flash in RAM:

   ```
   CHANGE ZFL_nEXT_Flash is CODE
   ```

   b) For eZ80F92 or eZ80F93, enter the following directives to locate ZFL routines that program or erase external Flash, and routines manipulate the Flash information page in RAM:

   ```
   CHANGE ZFL_nEXT_Flash is CODE
   CHANGE ZFL_nINT_Flash is CODE
   ```

   c) For eZ80L92, enter the following directive to locate ZFL routines that program or erase external Flash in RAM:

   ```
   CHANGE ZFL_nEXT_Flash is CODE
   ```

4. After adding the necessary linker directives, click **OK** to close the Additional Linker Directives dialog and then click **OK** again to close the Project Settings dialog.
5. If prompted to rebuild the project click **Yes**; otherwise, click **Rebuild All** or the Rebuild All icon in the **Build** menu.

Building the Zilog Flash Library

Most users will not need to rebuild the Zilog Flash Library. The library is typically only rebuilt to add a low-level Flash command set (as described in the Adding Additional Command Sets section), or to enable the use of erase-flags (Enabling Erase-Flag Processing), or possibly to enable simultaneous support for multiple Flash devices (Supporting Flash Devices on Multiple Chip Selects).

The Zilog Flash Library (ZFL) is built using the Zilog Development Studio II (ZDS II) integrated development environment (IDE) for the eZ80® family of microprocessors and microcontrollers. To rebuild the ZFL, use the following procedure:

1. Launch the ZDS II – eZ80Accalim! 5.3.0 (or later version) IDE.
2. Navigate to the `<ZDS II Install directory>\applications\FlashLibrary` folder and open the appropriate `eZ80F9x_Flash.zdsproj` or `eZ80_Flash.zdsproj` project file depending on the target platform.
3. From the **Build** configuration pull-down, select either the **Debug** or **Release** build, as shown in Figure 1.

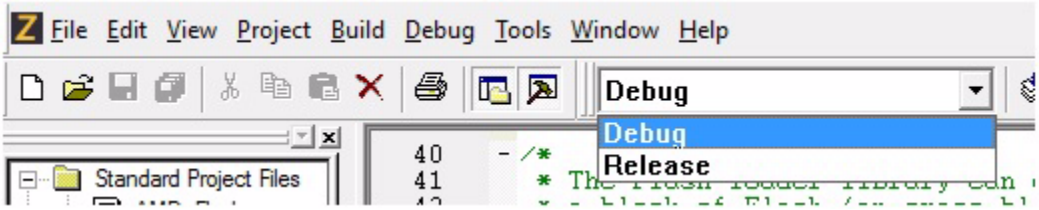

Figure 1. Select Build Configuration

The Debug build includes debug information that can be used to single step library API routines when linked with an application

program. The Release build does not contain debug information, but is otherwise identical to the Debug library build.

4. From the **Build** menu, select **Rebuild All** or click the Rebuild All icon to regenerate the library.
5. Copy the `FlashLibrary\Debug\ez80F9x_FlashD.lib` (or `eZ80_FlashD.lib` for eZ80L92) or `FlashLibrary\Release\eZ80F9x_Flash.lib` (or `eZ80_Flash.lib` for eZ80L92), depending on whether the Debug or Release build configuration was selected in step 3, to a folder where your application expects to find ZFL. For the `FlashLoader_App` sample project, the library file should be copied to the `<ZDS II Install directory>\applications\Lib` folder.

After rebuilding the appropriate Flash Library, rebuild all applications that use ZFL to ensure any changes made to the library source code are included in each application.

Enabling Erase-Flag Processing

The Zilog Flash Library can optionally be compiled to use a 1-bit flag, called an erase-flag, to track whether or not a block of external Flash has been erased. After an erasable block of external Flash has been erased (via the `XFL_BlockErase`, `XFL_EraseBlocks`, or `XFL_EraseDevice` APIs), the associated erase-flag(s) is placed in the *erased* state (i.e. the erase flag is set to binary 1). After the `XFL_Init` or `XFL_ResetEraseFlags` API routines are called, all erase flags are placed in the *not-erased* state (i.e. all erase flags are reset to binary 0).

When the `XFL_Program` API is called and erase flag processing has been enabled, the Flash library will automatically erase all blocks of Flash targeted by the program command if the block's erase flags is in the *not-erased* state. If the Flash blocks are successfully erased, the associated erase flags are placed in the *erased* state. The `XFL_Program` API will not

automatically erase a block of Flash targeted by the program command if its erase flag is in the *erased* state (or if the use of erase flag processing has not been enabled). This ensures that an erase block is erased the first time the `XFL_Program` API is called to program one or more bytes of data in that erase block; however, it prevents the block from being erased on subsequent calls to the `XFL_Program` API targeting that same erase block.

The concept of erase flags was introduced in version 1.0 of the Zilog Flash Library (the *status* byte of the `MEMORY_T` data structure was used as a 1-bit flag indicating whether the associated block of external Flash had been erased). The use of erase flags can eliminate the need for the application programmer to know the location of each erasable block in external Flash in some circumstances. However, including erase flag processing support in the ZFL increases the size of the library and therefore the size of the user application. Also, the use of erase flags does not eliminate the need for the application to decide when an erasable block of external Flash needs to be erased and whether or not a subset of the data previously programmed into the block needs to be saved and programmed into the block again after it has been erased. These operations require the application programmer to know the location of the underlying block(s) of Flash reducing the utility of erase-flags. For these reasons, version 2.0 (and later) of the ZFL does not include support for erase flags in the default build of the library.

The value of the `MAX_EB_STATUS_BYTES` macro (defined in `.\Flash-Library\XFL_Internal.h`) determines whether ZFL includes support for processing erase flags. By default, the value of `MAX_EB_STATUS_BYTES` is 0, which disables the use of erase flags. To include support for erase flag processing, it is necessary to change the value of the `MAX_EB_STATUS_BYTES` to a non-zero value as described in the Modifying the Maximum Number of Erase Blocks section.

Modifying the Maximum Number of Regions

The Zilog Flash Library is able to support external Flash devices with up to `CFI_MAX_REGIONS` regions. Each region contains one or more blocks or erasable Flash called an erase block. An erase block is the smallest unit of external Flash that can be erased. Each region of Flash contains erase blocks of identical size but adjacent regions contain erase blocks of different sizes.

Typical Flash devices only contain a few erase blocks; however, the CFI specification allows for a Flash device to contain up to 255 erase blocks. To reduce the size of the data structure ZFL uses to store information about each region, the default implementation of the Flash library supports Flash devices with up to `CFI_MAX_REGIONS`.

If the Zilog Flash Library is used with a CFI-compliant Flash device that contains more than `CFI_MAX_REGIONS` regions, the `XFL_Init` API will return a status of `ZFL_ERR_TOO_MANY_REGIONS`, indicating that the library needs to be rebuilt after increasing the value of the `CFI_MAX_REGIONS` macro definition using the procedure below:

1. Launch the ZDS II – eZ80Accalim! 5.3.0 (or later version) IDE.
2. Navigate to the `<ZDS II Install directory>\applications\FlashLibrary` folder and open the appropriate `eZ80F9x_Flash.zdsproj` or `eZ80_Flash.zdsproj` project file, depending on the target platform.
3. In the **Workspace** window on the left side of the IDE, expand the **External Dependencies** section and double-click the `CFI.h` header file.
4. Press `CTRL + F` or select **Find** in the **Edit** menu and type `CFI_MAX_REGIONS`.
5. Increase or decrease the value of the macro definition as appropriate. The value of the `CFI_MAX_REGIONS` macro definition should be at least 1 and should be less than or equal to 255.

Modifying the Maximum Number of Erase Blocks

The Zilog Flash Library is able to support Flash devices with up to MAX_EB_STATUS_BYTES * 8 erase blocks. The number of erase block status bytes (MAX_EB_STATUS_BYTES) determines the maximum number of erase flags the library supports. The default build of ZFL sets the value of the MAX_EB_STATUS_BYTES macro to 0, as defined in .\FlashLibrary\XFL_Internal.h, disabling the use of erase flags as described in the Enabling Erase-Flag Processing section. To enable erase flag processing, set the value of MAX_EB_STATUS_BYTES to be at least 1/8th of the total number of erasable blocks within the external Flash device and always round up to the next even byte. For example, if the external Flash contains 9 erase blocks, then MAX_EB_STATUS_BYTES should be set to 2 to enable erase flag processing.

Typical Flash devices contain between a few dozen erase blocks for smaller Flash devices, and up to a few hundred erase blocks for very large Flash devices. However, it is theoretically possible for a CFI-compliant Flash device to contain up to 255 regions with each region containing up to 65536 erase blocks. Such a Flash device would have an enormous storage capacity and would require 16,711,680 erase flags requiring ZFL to be rebuilt with MAX_EB_STATUS_BYTES set to 2,088,960, requiring over 2 MB of external RAM.

If the Zilog Flash Library is used with a CFI-compliant Flash device that contains more than MAX_EB_STATUS_BYTES *8 erase blocks, the XFL_Init API will return a status of ZFL_ERR_TOO_MANY_ERASE_BLOCKS, indicating that the library needs to be rebuilt after increasing the value of the MAX_EB_STATUS_BYTES macro definition using the following procedure:

1. Launch the ZDS II – eZ80Accalim! 5.3.0 (or later version) IDE.
2. Navigate to the <ZDS II Installation folder>\applications\FlashLibrary folder and open the appropriate eZ80F9x_Flash.zdsproj or eZ80_Flash.zdsproj project file, depending on the target platform.

3. In the Workspace window on the left side of the IDE, expand the **External Dependencies** section and double click the `XFL_Internal.h` header file.
4. Press `CTRL+ F` or select **Find** from the **Edit** menu and type `MAX_EB_STATUS_BYTES`.
5. Increase or decrease the value of the macro definition as appropriate. The value of the `MAX_EB_STATUS_BYTES` macro definition should be at least (Max Erase Blocks +7) /8, where Max Erase Blocks is determined by the target external Flash device. A value of 0 is used to prevent ZFL from automatically erasing a block of Flash the first time the `XFL_Program` API is called to program one or more bytes in the erase block.

Note: If the value of the `MAX_EB_STATUS_BYTES` macro is 0 when the Zilog Flash Library is rebuilt, the library will not include the `XFL_ResetEraseFlags` or `XFL_EraseBlocks` API routines. These API functions are only available if ZFL is compiled to support erase-flag processing.
A further consequence of setting the `MAX_EB_STATUS_BYTES` to 0 is that the `XFL_EraseDevice` API will return `ZFL_ERR_FAILURE` if an attempt is made to erase an external Flash device that implements the Intel standard command set. This occurs because the Intel standard command set does not actually include a command to erase the entire device. If `MAX_EB_STATUS_BYTES` is non-zero, ZFL calls the `XFL_EraseBlocks` API to erase all blocks in the external Flash.

6. Proceed to step 4 in the Building the Zilog Flash Library section.

Supporting Flash Devices on Multiple Chip Selects

The default build of the Zilog Flash Library can support only one external Flash device connected to one of the eZ80's chip select signals (`CS0` to `CS3`). Typically, the system is designed such that `CS0` is connected to the only external Flash device in the system. However, some systems could have multiple external Flash devices connected to other chip selects.

Since the default build of ZFL only supports a single external Flash device, applications that need to access multiple external Flash devices must call the XFL_Init API each time the application switches between Flash devices. For example, to erase the Flash devices on CS0 and CS2 and then program a block of Flash on CS0, the application would have to use issue the following commands:

```
XFL_Init(0, 0);          // Initialize ZFL for use with CS0
XFL_EraseDevice();       // Erase CS0 Flash
XFL_Init(2, 0);          // Initialize ZFL for use with CS2
XFL_EraseDevice();       // Erase CS2 Flash
XFL_Init(0, 0);          // Initialize ZFL for use with CS0
XFL_Program              // Program CS0 Flash (parameters not shown)
```

Alternatively, if the application needs to frequently switch between multiple external Flash devices, the ZFL library will have to be rebuilt, increasing the maximum number of Flash devices that can be supported using the following procedure:

1. Launch the ZDS II – eZ80Accalim! 5.3.0 (or later version) IDE.
2. Navigate to the <`ZDS II Install directory`> `\applications\FlashLibrary` folder and open the appropriate ZFL project file.
3. In the **Workspace** window on the left side of the IDE, expand the **External Dependencies** section and double-click the `FlashLib.h` header file.

4. Press CTRL + F or select **Find** in the **Edit** menu and type XFL_MAX_CHIP_SELECT_NUM.
5. Increase or decrease the value of the macro definition as appropriate. The value of the XFL_MAX_CHIP_SELECT_NUM macro definition must be equal to 1 + (the highest external Flash chip select number). For example, if CS0 and CS2 are connected to Flash devices, the value of the XFL_MAX_CHIP_SELECT_NUM macro should be 3.
6. Proceed to step 4 in the Building the Zilog Flash Library section.

Note: If ZFL is modified to maintain the state of multiple external Flash devices using the procedure above, the first parameter of all ZFL API functions must be the chip select number of the target Flash device. For example, if the value of XFL_MAX_CHIP_SELECT_NUM is 1 and only chip select 2 is connected to external Flash, the XFL_Init API should only be called with the ChipSelect parameter set to 2 but the XFL_Program API does not have to specify which chip select is targeted by the API because ZFL is only maintaining state about one external chip select (CS2). However, if both CS2 and CS3 are connected to an external Flash device, XFL_MAX_CHIP_SELECT_NUM should be set to 4 and the XFL_Init API needs to be called twice using ChipSelect parameters of 2 and 3 to identify which chip select is targeted by the Init call. In this instance, the first parameter passed to the XFL_Program API must be the chip select number of the external Flash device to be programmed.

The listing below shows the ZFL function prototypes used when the size of XFL_MAX_CHIP_SELECT_NUM is greater than 1.

ZFL API functions that require a ChipSelect parameter when XFL_MAX_CHIP_SELECT_NUM is not 1 have the parameter shown in red below. When XFL_MAX_CHIP_SELECT_NUM is 1, the ChipSelect

parameter must not be included in the call to the corresponding API function call.

```
extern INT8 FL_GetDeviceInfo( UINT8 ChipSelect,
                              FL_DEVICE_INFO * pDev );
extern INT8 FL_GetGeometry( UINT8 ChipSelect,
                            FL_DEVICE_INFO * pDev,
                            UINT8 * pNumRegions,
                            CFI_REGION * pRegions );
extern INT8 FL_EraseDevice( UINT8 ChipSelect );
extern INT8 FL_BlockErase( UINT8 ChipSelect,
             HANDLE hAddr );
extern INT8 FL_Program( UINT8 ChipSelect,
                        HANDLE hDst,
                        HANDLE hSrc,
                        UINT24 Len );
extern INT8 FL_Read( UINT8 ChipSelect,
                     HANDLE hDst,
                     HANDLE hSrc,
                     UINT24 Len );
extern INT8 FL_ReadCFI( UINT8 ChipSelect,
                        UINT24 Offset,
                        HANDLE hDst,
                        UINT16 Len );
extern INT8 FL_EraseBlocks( UINT8 ChipSelect,
                            HANDLE hAddr,
                            UINT24 NumBlocks );
```

```
extern INT8 FL_ResetEraseFlags( UINT8 ChipSelect );
```

External Flash Low-Level (Expert) CFI API

Developers with strong C programming skills that have experience with external parallel (NOR) Flash devices can utilize the External Flash Low-Level CFI API. This low-level API contains a single library function and allows the developer to access routines to program and erase external Flash using function pointers. The low-level API can only be used with CFI-compliant Flash devices.

CFI_Query

Header File:

```
#include "XFL_Internal.h"
```

Prototype:

```
INT8
CFI_Query
(
   FLASH_DEV_INFO * pDev
);
```

Parameters:

pDev: References a `FLASH_DEV_INFO` structure with the `pDev → Public.pBaseAddress` structure member referencing the first memory location in the target external Flash device. No other member of the structure needs to be initialized prior to calling this API.

Return value:

ZFL_ERR_SUCCESS is returned if no error occurs.

ZFL_ERR_UNSUPPORTED_CMDSET indicates that the device is CFI-compliant, but implements a command set that is not supported by the ZFL driver.

ZFL_ERR_UNSUPPORTED_DEVICE indicates the external Flash device is not CFI-compliant, or does not implement a supported command set.

Description:

This function reads the CFI Query table in a CFI-compliant external Flash device. If the CFI-compliant external Flash implements a supported command set, the `fpQuery`, `fpErase`, `fpEraseBlock`, and `fpProgram` function pointers in the structure referenced by `pDev` are initialized. After the function pointers are initialized, they can be de-referenced to program and erase the underlying Flash device.

Type definitions for the function pointers are contained in the `<ZDS II Install directory>\applications\FlashLibrary\Src\XFL_Internal.h` header file, as shown below.

```
/*
 * Flash Algorithm function pointer definitions
 */
typedef INT8 (* FP_FLASH_QUERY)
   ( struct FLASH_DEV_INFO_s * pDev );

typedef INT8 (* FP_FLASH_ERASE)
   ( struct FLASH_DEV_INFO_s * pDev );

typedef INT8 (* FP_FLASH_ERASE_BLOCK)
   ( struct FLASH_DEV_INFO_s * pDev, HANDLE hDst );
```

```
typedef INT8 (* FP_FLASH_PROGRAM)
   ( struct FLASH_DEV_INFO_s * pDev, HANDLE hDst,
     HANDLE hSrc, UINT8 Len );
```

The `fpQuery` function pointer is used to set the `Public.ManId` and `Public.DevId` members of the structure referenced by `pDev`. Some Flash command sets refer to the command used to obtain the manufacturer and device identification codes as the Autoselect, Read ID, or as SW ID command instead of a Query command as used in this document.

The `fpErase` function pointer is used to erase all erase blocks in the external Flash device. The `fpEraseBlock` function pointer is used to erase a single erase block in external Flash. The `fpProgram` function pointer is used to program one or more bytes in external Flash.

The default ZFL implementation of the Query, Erase, EraseBlock, and Program algorithms for the AMD and Intel standard command sets disables external interrupts while the external Flash is not in read-array mode to prevent system failure that could occur if the system attempted to activate an ISR located in external Flash.

Note: When an interrupt is processed on the eZ80F92, eZ80F93, eZ80L92, and eZ80190 devices, the first level interrupt table must reside within the first 64KB of memory. If the interrupt vector table is located in external Flash (a must for the eZ80L92 and eZ80190 processors), the eZ80 CPU will not be able to access the vector table while the external Flash is being erased, programmed, or queried.

Adding Additional Command Sets

The default build of the Zilog Flash Library contains support for the AMD and Intel standard command set. If additional command sets are required, use the following procedure.

1. Create routines to implement the function pointers described in the External Flash Low-Level (Expert) CFI API section. This file should be created in the `<ZDS II Install directory>\applications\FlashLibrary` folder.
2. Launch the ZDS II – eZ80Accalim! 5.3.0 (or later version) IDE.
3. Navigate to the `<ZDS II Install directory>\applications\FlashLibrary` folder and open the appropriate ZFL project file.
4. In the **Workspace** window on the left of the IDE, right-click **Standard Project Files** and select **Add File to Project….** In the **Add Files To Project** dialog, navigate to the folder containing the file created in Step 1.
5. Double-click the `XFL_CmdSet.c` file and add an entry to the `CmdSetTable` array for the newly created command set. Be sure to add the command set to the address-sensitive or address-insensitive sections as appropriate. For example, if the `XXX` command set is added, the new entry in the `CmdSetTable` array should look similar to the following:

```
{
        0x1234,
        XXX_QueryDevice,
        XXX_EraseDevice,
        XXX_EraseBlock,
        XXX_Program
    },
```

In the above array, `0x1234` represent the Vendor Command Set and Control Interface ID Code assigned in the CFI specification for the command set implemented in Step 1. The four function names listed in the `CmdSetTable` entry being created must match the function names used in Step 1.

6. Add Function prototypes for the command set created in Step 1 to the `XFL_Internal.h` header file.
7. Proceed to step 4 in the Building the Zilog Flash Library section.

External Flash Direct (XFLD) API

Applications that only need to interface with the external Flash device located on select Zilog development kits can use the Direct External Flash (XFLD) API to minimize the amount of library code added to the project. Because the ZFLD API provides only basic serves, the application must implement its own address management and error checking.

The XFLD API is described in Table 3.

Table 3. XFLD API Description

XFLS API	Description
XFLD_Erase	Erases a block of external Flash.
XFLD_Program	Programs one or more bytes of external Flash.
XFLD_Query	Obtain Flash manufacturer and device identification codes.
XFLD_Read	Copies one or more bytes of memory from the external Flash to a RAM memory buffer.

The XFLD API is implemented as a set of macros that reference AMD or Intel Flash algorithms based on the Zilog Development kit that the appli-

cation targets. The application must include a `DEV_KIT=` linker command that specifies a supported Zilog Development kit. For example, an application targeting the eZ80F910300KITG development kit would include the following linker directive to use the XFLD API:

```
DEV_KIT=eZ80F91_99C1322
```

Where `eZ80F91_99C1322` is the development kit ID defined in the `<ZDS II install directory>\applications\Inc\eZ80DevKit.h` header file. Note that the development kit ID is silkscreened on all Zilog development kits.

Based on the value of the DEV_KIT linker symbol, the `<ZDS II install directory>\applications\Inc\FlashLib.h` header file defines the low-level AMD or Intel Flash routine that is used to implement each XFLD API. For example, the `eZ80F91_99C1322` development kit uses the following mappings:

#define XFLD_Erase	AMD16_EraseBlock
#define XFLD_Program	AMD16_Program
#define XFLD_Query	AMD16_Query
#define XFLD_Read	ZFL_Read

Note: If the XFLD macros are mapped to an incompatible set of low-level Flash routines, the XFLD will fail to operate as expected (or at all).

Data Structures and Macros

This section describes the primary data structures and macros used by the Zilog Flash Library. For the most up-to-date information and descriptions, refer to the header files in the `<ZDS II Install directory>\applications\Inc` folder.

Basic Data Types

```
/*
 * Basic data types
 */
typedef unsigned char         UINT8;
typedef unsigned short int    UINT16;
typedef unsigned int          UINT24;
typedef unsigned long int     UINT32;

typedef signed char           INT8;
typedef signed short int      INT16;
typedef signed int            INT24;
typedef signed long int       INT32;

typedef void *                HANDLE;
#define NULLPTR               (void *) 0

typedef UINT8                 BOOL;
```

XFL_DEVICE_INFO Structure

```
typedef struct          XFL_DEVICE_INFO_S
{
   /*
    * External Flash chip select number (0..3)
    */
   UINT8                CSx;


   /*
   * If the external Flash device supports the
   * Common Flash memory Interface, the Is_CFI flag
   * is set to TRUE. In this instance, the Regions
   * array contains valid information about the
   * device. Otherwise, the Is_CFI flag is FALSE
   * and the Regions array will only contain
   * information about the (legacy) Flash device if
   * it was used on an old Zilog development kit.
   */
      BOOL                 Is_CFI;


   /*
   * Legacy Flash identification information.Before
   * Flash devices started including CFI support,
   * applications had to use the Manufacturer and
   * Device ID codes to index a static lookup table
   * to find the location of Flash erase blocks and
   * determine which Flash algorithms to use for
   * programming and erasing the device. When CFI-
   * compatible Flash devices are used, the device
   * geometry can be determined at run time without
   * the use of lookup tables.
   */
```

```
    UINT8                ManId;
    UINT8                DevId;
    /*
    * Flash devices typically implement a standard
    * set of algorithms to erase and program Flash.
    * The Zilog Flash Library can only be used if the
    * device implements a supported command set
    * (see FL_CmdSet_Conf.c for a list of supported
    * command sets and CFI.h for the
    * CMD_SET_xxx identifier codes).
    */
       UINT16              CmdSet;
    /*
    * The Base and End addresses of external Flash
    * are determined based on the chip select
    * settings while the Size is the maximum storage
    * capacity of the external Flash device in Bytes.
    * If the entire Flash is visible in the eZ80
    * address space then
    * (pEndAddr+1 - pBaseAddr) = Size.
    */
       UINT8             * pBaseAddr;
       UINT8             * pEndAddr;
       UINT32              Size;
} XFL_DEVICE_INFO;
```

CFI_REGION Structure

```
/*
 * CFI Geometry Information
 */
typedef struct          CFI_REGION_s
{
   /*
   * Each Flash region is composed of 1 or more
   * erase blocks of the same size. The NumBlocks
   * value is 1 less than the actual number of
   * blocks in the region (each region contains
   * between 1 and 65536 erase blocks). The size of
   * each erase block is determined by multiplying
   * BlockSize by 256 bytes. A BlockSize of 0
   * indicates each erase block is 128-bytes long.
   */
   UINT16               NumBlocks;
   UINT16               BlockSize;
} CFI_REGION;
```

NON_CFI_DEV Structure

```
typedef struct          NON_CFI_DEV
{
   UINT8                ManId;
   UINT8                DevId;
/*
    * The structure members used to describe the
    * non-CFI device geometry are a subset of the
```

```
     * geometry information in the CFI_GEOMETRY
     * structure in CFI.h.
     */
    UINT8                SizeExp;
    UINT8                NumRegions;
    CFI_REGION         * pRegions;
} NON_CFI_DEV;
```

ZFL Error Code Macros

```
#define ZFL_ERR_SUCCESS                0
#define ZFL_ERR_FAILURE               –1
#define ZFL_ERR_VPP                   –2
#define ZFL_ERR_WRITE                 –3
#define ZFL_ERR_ERASE                 –4
#define ZFL_ERR_SUSPEND               –5
#define ZFL_ERR_ADDRESS               –6
#define ZFL_ERR_VERIFY                –7
#define ZFL_ERR_UNSUPPORTED_CMD_SET   –8
#define ZFL_ERR_UNSUPPORTED_DEVICE    –9
#define ZFL_ERR_INVALID_PARAMETER     –10
#define ZFL_ERR_TOO_MANY_ERASE_BLOCKS –11
#define ZFL_ERR_TOO_MANY_REGIONS      –12
```

ZFL Library Version

```
#define ZFL_VERSION                    0x0200
```

Customer Support

For answers to technical questions about the product, documentation, or any other issues with Zilog's offerings, please visit Zilog's Knowledge Base at http://www.zilog.com/kb.

For any comments, detail technical questions, or reporting problems, please visit Zilog's Technical Support at http://support.zilog.com.