Application Note

How to Make eZ80® Code Execute Faster with Copy To RAM

AN015103-1208

Abstract

This Application Note discusses how Zilog's eZ80® code can execute faster by executing from RAM instead of Flash Memory, and be able to store the eZ80 code in Flash. With Zilog Development Studio II (ZDS II) code development tools, a few simple changes to the project settings and the project link file are required for speedy code execution. The `leddemo` project that is included with the eZ80L92 device, which uses the ZDS II toolset for eZ80 microprocessor unit (MPU), is used in this Application Note as an example. However, the information in this document is also applicable to the ZDS II toolset for eZ80Acclaim!® microcontrollers unit (MCU).

Note: *The source code file associated with this application note,* `AN0151-SC01.zip` *is available for download at* www.zilog.com.

eZ80 Overview

eZ80 has revolutionized the communication industry. It executes Zilog's Z80® code four times faster at the same clock speed of traditional Z80s and can operate at frequencies up to 50 MHz. Unlike most 8-bit microprocessors, which can only address 64 KB, the eZ80 can address 16 MB without a Memory Management Unit (MMU).

Designed with over 25 years of experience with the Z80, this microprocessor is best suited for embedded internet appliances, industrial control, automation, web management, modem controller, electronic games, and personal digital assistant (PDA) applications.

Two popular versions of the eZ80 offered by Zilog® are the eZ80 General-Purpose MPU (eZ80L92) and the eZ80Acclaim! Family of Flash MCUs, which includes the eZ80F91, eZ80F92, and eZ80F93 devices.

eZ80L92 Features

The key features of eZ80L92 include:

- Single-cycle instruction fetch, high-performance, pipelined eZ80 CPU core
- Low power features including SLEEP mode, HALT mode, and selective peripheral power-down control
- Two Universal Asynchronous Receiver/Transmitters (UART) with independent baud rate generators
- Serial Peripheral Interface (SPI) with independent clock rate generator
- Inter-Integrated Circuit (I^2C) with independent clock rate generator
- Infrared Data Association (IrDA)-compliant infrared encoder/decoder
- New DMA-like eZ80 instructions for efficient block data transfer
- Glueless external peripheral interface with four Chip Selects, individual Wait State generators, and an external $\overline{WAIT}$ input pin—supports Intel- and Motorola-style buses
- Fixed-priority vectored interrupts (both internal and external) and interrupt controller
- Real-time clock (RTC) with on-chip 32 kHz oscillator, selectable 50/60 Hz input, and separate V_{DD} pin for battery backup

Copyright ©2008 by Zilog®, Inc. All rights reserved.
www.zilog.com

- Six 16-bit Counter/Timers with prescalers and direct input/output drive
- Watchdog Timer (WDT)
- 24 bits of General-Purpose Input/Output (GPIO)
- JTAG and ZDI debug interfaces
- 100-pin LQFP package
- 3.0—3.6 V supply voltage with 5 V tolerant inputs

eZ80Acclaim! Overview

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

The eZ80Acclaim! Flash MCU can operate in full 24-bit linear mode addressing 16 MB without a MMU. Additionally, support for the Z80-compatible mode allows Z80/Z180 legacy code execution within multiple 64 KB memory blocks with minimum modification. With an external bus supporting eZ80, Z80, Intel, and Motorola bus modes and a rich set of serial communications peripherals, you have several options when interfacing to external devices.

Some of the many applications suitable for eZ80Acclaim! devices include vending machines, point-of-sale (POS) terminals, security systems, automation, communications, industrial control and facility monitoring, and remote control.

eZ80Acclaim! Features

The key features of eZ8Acclaim! include:

- Single-cycle instruction fetch, high-performance, pipelined eZ80 CPU core
- 10/100 BaseT Ethernet Media Access Controller with Media-Independent Interface (MII)
- 256 KB Flash Memory and 16 KB SRAM (8 KB user and 8 KB EMAC)
- Low power features including SLEEP mode, HALT mode, and selective peripheral power-down control
- Two UARTs with independent baud rate generators
- SPI with independent clock rate generator
- I^2C with independent clock rate generator
- IrDA-compliant infrared encoder/decoder
- Glueless external peripheral interface with four Chip Selects, individual Wait State generators, and an external $\overline{WAIT}$ input pin—supports Z80-, Intel-, and Motorola-style buses
- Fixed-priority vectored interrupts (both internal and external) and interrupt controller
- RTC with on-chip 32 kHz oscillator, selectable 50/60 Hz input, and separate V_{DD} pin for battery backup
- Four 16-bit Counter/Timers with prescalers and direct input/output drive
- WDT with internal oscillator clocking option
- 32 bits of GPIO
- On-Chip Instrumentation (OCI™) and ZDI debug interfaces
- IEEE 1149.1-compliant JTAG
- 144-pin LQFP package
- 3.0—3.6 V supply voltage with 5 V tolerant inputs

Discussion

When programming with the eZ80L92 device or the eZ80Acclaim! devices, the code is debugged in RAM and Flash using the ZDS II development tool. The execution of code is slow in Flash. For faster execution, copy the code from Flash to RAM and execute it from RAM. This Application Note describes how to perform this task using ZDS II. The example described within this document uses the ZDS II development toolset for the eZ80 that is contained in the eZ80L92 Development Kit.

In the ZDS II development environment, there are project settings that allow you to select the type of configuration that the code stores and executes. These configurations include:

- Standard configuration
- All RAM configuration
- Custom configuration
- Copy to RAM configuration

These four configurations are described in the *Zilog Developer Studio II—eZ80Acclaim! User Manual (UM0144)* and *How Code and Data are Placed in Memory Using ZDS II Application Note (AN0299)*.

The Copy to RAM configuration is described in this Application Note. The objective is to:

1. Program Flash Memory with a hex file containing your application code and data.
2. On startup, direct the application program to copy the remainder of the program from Flash into RAM.
3. Continue executing from RAM.

The end result is increased code execution speed.

The code that executes upon reset (initiated by a Power-On Reset or the pushbutton on the eZ80 Development Platform) is the Boot Loader. The Boot Loader is part of the external Flash Loader utility that is packaged with the eZ80 development kits. This external Flash Loader programs Flash Memory on the eZ80 Development Platform starting at a location in Flash that the Boot Loader jumps to after reset. The typical flow of the Copy to RAM and Boot Loader functions is displayed in Figure 1.

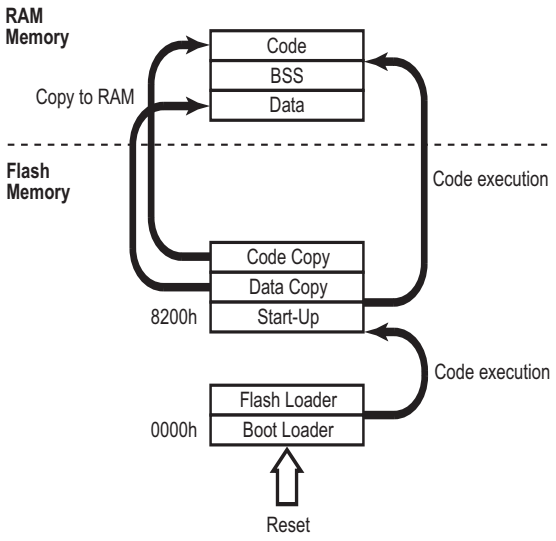

Figure 1. The Copy to RAM and Boot Loader Execution Flow

Figure 1 displays the Start-up code, Data Copy, and Code Copy programmed into Flash by the Flash Loader. Pushing the reset button on the eZ80 Development Platform starts the Boot Loader. Pressing the space bar on your keyboard while pushing the reset button causes the Boot Loader to jump to the Flash Loader utility. When the Flash Loader starts executing, it waits for data on serial port 0. Data received on this port is programmed into Flash starting at location `08200h`. Alternatively, Flash Memory can be programmed using the ZDS II internal Flash Loader, which is the tested utility described in this Application Note.

After Flash Memory is programmed, push the reset button a second time without pressing the space

bar. The Boot Loader again starts execution. This time, it jumps to location 8200h, where the start-up code resides. This start-up code performs the following sequence:

1. Initializes the required processor registers.
2. Copies the Data Copy section in ROM to the Data section in RAM.
3. Creates the BSS section in RAM.
4. Copies the Copy Code section in ROM to the Code section in RAM.
5. Jumps to the first line of code in the code section.

Note: *The start-up code is not copied. Additionally, the Code section is the main portion of code that runs after startup.*

In this Application Note, the eZ80L92 device is used to explain an example of how to use the Copy to RAM feature. The leddemozdsFlash.pro project file that is packaged with ZDS II version ez80_4.4.2 explains how to change the project such that the code can be copied from Flash to RAM and executed. This file is located in the ZDSII_eZ80_4.4.2/samples/ez80L92_LedDemo/src directory. The ZDS II build generates a hex file that can be programmed into Flash by either the external Flash Loader or the Flash Loader internal to ZDS II.

The leddemozdsFlash project uses the Custom configuration and a start-up module that is included in the project. The start-up module is named l92_extFlashboot.s. Two other source files are used in the project: ledDemo.c and ledInit.c. The project is configured to access the header files in the include directory.

The steps in Table 1 describe how to make a new project that copies the Code section from Flash and execute it from RAM. It is considered that you have successfully built, downloaded, and run the leddemozdsFlash project, or you are at least familiar with the procedure.

Table 1. How to Copy Code from Flash and Execute in RAM

Step	Comment
1. Create a new directory and name it **ramcopy**.	Create **ramcopy** as a subdirectory within the **ez80L92_LedDemo** directory, and at the same level as **src**.
2. Copy the following files from the **src** directory to the **ramcopy** directory: • **leddemozdsFlash.pro** • **ledDemo.c** • **ledInit.c** • **l92_Flashboot.s**	All the project settings in **leddemozdsFlash.pro** are also copied. The path set in the **Project Settings** dialog box identifies the header files that are referenced in the include directory.
3. Change the name of the project file **leddemozdsFlash.pro** to **leddemoRamCopy.pro**.	Files are generated with the new project filename.
4. Start ZDS II_eZ80_4.4.2.	Ensure that you use this version of ZDS II for eZ80 or a more recent version.
5. Open the **leddemoRamCopy** project.	
6. Click on **Linker** tab in the **Project Settings** dialog box. Select the **General** category and change the output file name from **leddemozdsFlash** to **leddemoRamCopy**.	Output files are generated with the new filename.

Table 1. How to Copy Code from Flash and Execute in RAM (Continued)

Step	Comment
7. Using ZDS II, open the **l92_Flashboot.s** file and remove or comment out the following lines of code: • Define .invect, space=ROM, org=%0000 • Segment .invect • jp.lil (%8200)	Removing these instructions from the start-up code in **l92_Flashboot.s** removes the instructions that cause the linker to place the **jump to 08200h** instruction at location **0000h** for resets. Flash Memory should already contain the Flash Loader boot code at this location and should not be overwritten if it is to be used.
8. In the **General** category of the **Linker** tab of the **Project Settings** dialog box, click on **Link Configuration** and select **Copy to RAM**. Click **OK**. A pop-up dialog is displayed, which asks if you want to use the **Standard Startup Module**. Click **Yes**.	This step sets the configuration to **Copy to RAM**. To place the **Copy to RAM** linker commands into the Link Control File, the **Standard Startup Module** must also be selected. (With this warning, ZDS II selects the **Standard Startup Module** for you.)
9. Click **OK** in the **Project Settings** dialog box. Another A pop-up dialog is displayed, which asks if you want to rebuild the files. Click **Yes**.	A Link Control File named **leddemoRamCopy** is created. This file requires further modification, as described in the next step.
10. Using ZDS II, open the **leddemoRamCopy.lnk** link control file and make the following changes near the end of the file: • Remove the line containing **…\lib\startup.obj** • Remove the line **locate .startup at 8200h**	**leddemoRamCopy.lnk** can show up as a shortcut file with no extension. Performing this step modifies the reference to the start-up code in the Standard Start-up Module to reference the start-up code in **l92_Flashboot.s**.
11. Perform a **Save As...** for this file and name it **ledemoRamCopy2.lnk**. Select this file from the **Use Existing** text box in the **Input Category** field of the **Linker** tab of the **Project Settings** dialog box.	The link file **leddemoRamCopy2.lnk** is now used in the build. Renaming the file also protects the link control file from being overwritten if **Create New** is accidentally selected.
12. In ZDS II, open the **Standard Startup Module** file **startup.asm** and the **l92_Flashboot.s** file included in the project.	The **startup.asm** file is located in the ZDS II directory **…src\rtl\common**.
13. Copy the **Copy to RAM** code from **startup.asm** and paste it into the **l92_Flashboot.s** file. This code starts after **label _c_data_done** and ends with **label _copy_code_to_ram_done**.	This code segment should be located after the copy of initialized data code, in both files, to provide the **Copy to RAM** code in the **l92_Flashboot.s** start-up module.
14. Change the selection from **Standard** to **Included in Project** for the Startup Module in the **Input Category** field of the **Linker** tab in the **Project Settings** dialog box.	Everything is now changed to use the start-up code included in the project from the start-up code provided by the **Standard Startup Module**.
15. Now click **Build** or **Build All** to build the **leddemoRamCopy** project.	The **leddemoRamCopy.hex** file is generated and can be programmed into Flash Memory using either the external Flash Loader or the ZDS II internal Flash Loader.
16. To program Flash Memory using the ZDS II internal Flash Loader, start by selecting the **ZDS II Flash Loader command** from the **Tools** menu to bring up the **Flash Loader Processor** dialog box.	It is considered that the eZ80 Development Platform is set up and the **leddemozdsFlash** project has been successfully demonstrated.

Table 1. How to Copy Code from Flash and Execute in RAM (Continued)

Step	Comment
17. In the **External Flash Devices** pane, select the **Micron MT 28F008B3 Non-Prot 0x8000** entry and click the **ERASE** button. A pop-up dialog is displayed, which asks if you want to erase **Flash 8000 to FFFFF**. Click **Yes**.	The user portion of Flash Memory is erased and not the Flash Loader/Boot Loader portion. Be patient, because the erase can take a few seconds.
18. Using the browser in the **File** pane, find and select the **leddemoRamCopy.hex** file.	
19. Click **Burn and Verify** in the **Flash Controls** pane.	Flash Memory is programmed with the **leddemo** code that includes **Copy to RAM**.
20. After Flash memory is programmed, push the reset button on the eZ80 Development Platform.	Notice the difference in the speed of the LED flashing between the **leddemozdsFlash** code and the **leddemoRamCopy** code.

The **leddemoRamCopy** project included in this applications note, already contains the modifications performed in Table 1. The project is located in the .zip file `RamCopy.zip`. After this file is unzipped, place the `ramcopy` folder in the following directory path.

```
c:\Program Files\Zilog\
ZDSII_eZ80_4.4.2\samples\ez80L92_L
edDemo\
```

The *include* file for this project should already reside within the `include` folder located in the same path. If the `ramcopy` folder already exists from completing the steps in Table 1, rename the existing folder. Using ZDSII_eZ80_4.4.2, build and Flash-load the project. Press the **Reset** button on the eZ80 Development Platform to run the program.

Summary

This Application Note discusses how a few changes to an existing project that had formerly stored and executed code from Flash Memory could be made into a project in which the code is stored in Flash and executed from RAM. The resulting increase in execution speed was easily observed in the rate of the LED display sequence. Achieving faster code execution involved creating a new link file by modifying the existing file and making changes to the startup source code file.

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

©2008 by Zilog, Inc. All rights reserved. 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.

Z80, eZ80, and eZ80Acclaim! are registered trademarks of Zilog, Inc. All other product or service names are the property of their respective owners.