Application Note

ZDS II for eZ80Acclaim!: Explicit Code Memory Placement in C

AN033201-0711

Abstract

This application note illustrates a solution for placing code into the RAM, ROM or Flash memory spaces of the eZ80Acclaim! MCU using the general-purpose computer programming language known as C.

Note: The source code file associated with this application note, AN0332-SC01.zip, has been tested with ZDS II – eZ80Acclaim! version 5.1.1.

Introduction

In certain applications, particularly those that use more than 1 type of physical memory, code and variable placement is sometimes required to manage memory better. Other firmware developers use such placement tricks to gain control over memory allocation. The Assembly language does this very well, but the C Compiler only supports variable placement.

This paper aims to demonstrate a quick solution to placing code segments onto user-specified portions of memory when using the C language. This solution includes code segments for functions, variable declarations and nonvolatile variable initializations which are stored in RAM, ROM or Flash.

Code Memory Placement

One method for allocating code segments into specific portions of memory uses the `pragma asm` directive to insert assembly language syntax into a C source file. The `pragma asm` directive consists of the following syntax:

```
#pragma asm "<assembly line>"
```

As a general guideline, a segment name must be declared first with the following syntax:

```
#pragma asm "define <segment_name>, space=<space_id>,
org=<address>"
```

Another `pragma asm` directive is required to actually place the following routines into the appropriate address segment.

```
#pragma asm "segment <segment_name>"
```

All routines (within the same C file) following this directive will be allocated in subsequent addresses starting from the specified segment address. The default memory allocation scheme resumes at the next C file processed by the compiler, which starts at the first available address in the range specified in the Address Settings.

The sections that follow provide examples of code segment placement into ROM and RAM. Each example includes a portion of the map file produced to confirm that the code is actually placed into its proper memory locations.

Caution: When using the `pragma asm` directive, make sure that the assembly line to be inserted follows standard assembly language syntax. The compiler does not process the assembly line; instead, the assembly line is passed through the compiler to the assembler as-is. Therefore, no error checking occurs during the compile. Although the assembler reports whether an error has been encountered, it may be difficult to locate which portion of the C source file will need to be corrected.

Function Code Stored in Flash Memory or ROM

Flash memory is a nonvolatile type of memory in which both program code and data code are stored. It contains the flash option bits, reset vector address, interrupts and program code. Program code, such as the main function code, can be placed into any address within Flash. The function code named *factorial* is stored in a ROM space which starts at `0x3048`; sample code for this type of declaration is listed below.

```
//C code
#pragma asm "define application1, space=rom, org=%3048"
#pragma asm "segment application1"
unsigned int factorial_code(unsigned int x)
{
  if (x==1){
    return (1);
  }
  else{
    return (x*factorial_code(x-1));
  }
}

MAP FILE:
 Name                              Base       Top        Size
 ------------------------------------------------------------
 Segment: application_code1     C:003048   C:00307A   33h
```

Function Code Stored in RAM

The function code can also be stored in a specified RAM address; this address in RAM can be found in the map file. A function named *swap* is stored in a RAM space which starts at address `0xB7E200`; an example code segment for this type of declaration is listed below.

```
//C code
#pragma asm "define application_code2, space=ram, org=%B7E200"
#pragma asm "segment application_code2"
void swap(unsigned int a, unsigned int b)
{
  a = a+b;
  b = a-b;
  a = a-b;
}

MAP FILE:
 Name                            Base       Top        Size
 ------------------------------------------------------------
 Segment: application_code2      D:B7E200   D:B7E223   24h
```

Variable Memory Placement

Another method for allocating variables into specific portions of RAM memory differs from the code placement method, and is also much easier. The eZ80Acclaim! C Compiler provides a language extension for variable code placement: the `_At` directive, which consists of the following syntax:

```
  <variable declaration> _At <address, in hex>;
```

The sections that follow provide examples for placing a variable into the ROM space and include a portion of the map file produced to confirm that the variable is actually placed into its proper memory locations.

Variable Declaration in a Specified Address

Variables can be declared for a specific address space using the prefix `_At` and when specifying the address space. The variable name `specific_address_0xB7E400` is stored in RAM at the `0xB7E400` address location; sample code for this type of declaration is listed below.

```
//C code
unsigned int specific_address_0xB7E400 _At 0xB7E400; //is
                               //assigned at address 0xB7E400

MAP FILE:
 Name                            Base       Top        Size
 ------------------------------------------------------------
 Segment: ___specific_address_0xB7E400_s  D:B7E400   D:B7E402   3h
```

Map File

As a result of compiling the code, a portion of the map file labeled *space allocation* lists the variable and function names used in the project, along with the memory address to which these names are allocated.

From this section of the map file, and as is shown in the following code segment, the variables `specific_address_0xB7E400` and `function swap_code` are stored in RAM, while the function factorial code is stored in ROM. The code shows that the functions and variable names are allocated as expected within memory.

```
SPACE ALLOCATION:
=================
Space                Base        Top          Size
-------------------- ----------- ----------- ---------
RAM                    D:B7E000    D:B7E402     403h
ROM                    C:000000    C:00307A     307bh

SEGMENTS WITHIN THIS SPACE:
======================
RAM                  Type          Base       Top        Size
-------------------- ------------- ---------- ---------- ----
.IVECTS              normal data   D:B7E000   D:B7E0FF   100h
___specific_address  absolute data D:B7E400   D:B7E402     3h
application_code2    absolute data D:B7E200   D:B7E223    24h
DATA                 normal data   D:B7E100   D:B7E102     3h

ROM                  Type          Base       Top        Size
-------------------- ------------- ---------- ---------- ----
.RESET               normal data   C:000000   C:00006A    6bh
.STARTUP             normal data   C:00006B   C:00024F   1e5h
application_code1    absolute data C:003048   C:00307A    33h
CODE                 normal data   C:000250   C:0002A6    57h
DATA                *segment copy* C:0002A7   C:0002A9     3h
```

Configuration

The following tools were used to develop this application note.

- ZDS II – eZ80Acclaim! version 5.1.1
- eZ80F91 Development Kit (eZ80F910x00ZCOG)

References

The following documents describe the functional specifications and toolsets for the eZ80Acclaim! MCU. Each is available for download from the Zilog website.

eZ80F91 MCU Product Specification (PS0192)

eZ80 CPU User Manual (UM0077)

Zilog Developer Studio II – eZ80Acclaim! User Manual (UM0144)

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

 Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.