MOTOROLA
intelligence everywhere™

digital dna™

56800E Coding Guidlines for Small Data Memory Model and Large Data Memory Model

56F8300
Hybrid Controller

CG56800E
Rev. 0, 06/2004

MOTOROLA.COM/SEMICONDUCTORS

Contents

About This Document

Audience v
Suggested Reading v
Conventions vi
Definitions, Acronyms, and Abbreviations vii
References. vii

1 Introduction

1.1 Switching Between Models 1-1

2 Coding Guidelines

2.1 Type Casting Pointers. 1-1
2.2 Conversion of 56800 Instructions to 56800E Instructions 1-1
2.3 LoadRx, StoreRx, and TestRx Macros 1-1
2.4 Push and Pop Instructions. 1-2
2.5 Structure Offsets 1-3

List of Examples

Code Example 2-1 Use Macros to Convert DSP56800 or DSP56800E Assembly Code . . . 1-2
Code Example 2-2 Converting 56800 Assembly Code Using Push and Pop #1. 1-2
Code Example 2-3 Converting 56800 Assembly Code using Push and Pop #2 1-3
Code Example 2-4 Convert C structure. 1-3
Code Example 2-5 Convert Assembly Code Structure 1-4

About This Document

This manual describes some coding guidelines in order to generate valide code for both a Small Data Memory model (SDM) and a Large Data Memory model (LDM) target for a 56800E application.

Audience

This document targets software developers who have implemented software applications with 56800 devices and are converting to the 56800E family of devices.

Suggested Reading

We recommend that you have a copy of the following references:

- *DSP56800 to DSP56800E Porting Guide,* DSP56800ERG/D
- *DSP56800 Family Manual,* DSP56800FM/AD
- *DSP56800E Reference Manual*, DSP56800ERM/AD
- *Inside CodeWarrior: Core Tools*, Metrowerks Corp.

Conventions

This document uses the following notational conventions:

Typeface, Symbol or Term	Meaning	Examples
`Courier Monospaced Type`	Code examples	`//Process command for line flash`
Italic	Directory names, project names, calls, functions, statements, procedures, routines, arguments, file names, applications, variables, directives, code snippets in text	...and contains these core directories: *applications* contains applications software... ...CodeWarrior project, *3des.mcp* is... ...the *pConfig* argument.... ...defined in the C header file, *aec.h*....
Bold	Reference sources, paths, emphasis	...refer to the **Targeting DSP56F80x Platform** manual.... ...see: **C:\Program Files\Motorola\help\tutori-als**
Blue Text	Linkable on-line	...refer to Chapter 7, License....
Number	Any number is considered a positive value, unless pre-ceded by a minus symbol to signify a negative value	3V -10 DES^{-1}
ALL CAPITAL LETTERS	# defines/ defined constants	# define INCLUDE_STACK_CHECK
Brackets [...]	Function keys	...by pressing function key [F7]
Quotation marks, “...”	Returned messages	...the message, “Test Passed” is displayed.... ...if unsuccessful for any reason, it will return “NULL”...

Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document. As this template develops, this list will be generated from the document. As we develop more group resources, these acronyms will be easily defined from a common acronym dictionary. Please note that while the acronyms are in solid caps, terms in the definition should be initial capped ONLY IF they are trademarked names or proper nouns.

DSP	Digital Signal Processor or Digital Signal Processing
LDM	Large Data Memory
SDM	Small Data Memory

References

The following sources were referenced to produce this book:

1. *DSP56800 to DSP56800E Porting Guide,* DSP56800ERG/D
2. *DSP56800 Family Manual,* DSP56800FM/AD
3. *DSP56800E Reference Manual*, DSP56800ERM/AD
4. *Inside CodeWarrior: Core Tools*, Metrowerks Corp.

1 Introduction

1.1 Switching Between Models

A Small Data Memory model (SDM) and a Large Data Memory model (LDM) are available for the 56800E family of hybrid controllers. The SDM has a maximum address size of 16 bits for both data and program addresses. This allows for smaller, compact code size, since an address can be stored in one 16-bit word. The LDM has a maximum address size of 19 bits for program addresses and 21 bits for data addresses. This allows an application to use more memory, but it requires two 16-bit words of memory to store each address.

In order to switch between the different models during development, specific coding practices should be followed to make this as transparent as possible. The remainder of this document describes these practices.

2 Coding Guidelines

This section explains how the user's source code should be developed in order to generate code for both the SDM and LDM.

2.1 Type Casting Pointers

When developing software for the SDM, do not type cast pointers to be 16 bits (i.e., unsigned int). This will work for the SDM, but if you switch to the LDM, it may not work, because your pointer size may be greater than 16 bits.

2.2 Conversion of 56800 Instructions to 56800E Instructions

When the 56800E was developed, one requirement was the ability to use all of the 56800 instructions to easily upgrade from the 56800 to the 56800E. This approach works fine for the SDM; however, since the 56800 instructions are for a 16-bit DSP with 16-bit pointers, this approach is no longer valid when using the LDM.

When using the LDM, the "*allow legacy instructions*" *Language Settings: M56800E Assembler* option must be turned off and the user must convert all 56800 instructions to 56800E instructions. Please see the **DSP56800 to DSP56800E Porting Guide** to learn more.

2.3 *LoadRx*, *StoreRx*, and *TestRx* Macros

When loading, storing, or testing pointers in Assembly, you must specify if it is a word or a long. For the SDM, pointers are a word and for the LDM, pointers are a long.

Depending on the model, defined macros in *portasm.h* ease switching from SDM to LDM and vice versa. These assembly macros (*LoadRx*, *StoreRx*, and *TestRx*) are defined based on which model is used, so they load, store, and test the correct size of the pointer.

Code Example 2-1. Use Macros to Convert DSP56800 or DSP56800E Assembly Code

56800 Assembly code:

```
move   X:(R2+Offset_pCircBuffer),R0 ; where pCircBuffer contains a pointer
move   R0,X:(R2+Offset_pCircBuffer)
tstw   X:(R2+Offset_pCircBuffer)
```

or 56800E Assembly code (SDM):

```
moveu.w X:(R2+Offset_pCircBuffer),R0 ; where pCircBuffer contains a pointer
move.w R0,X:(R2+Offset_pCircBuffer)
tst.w  X:(R2+Offset_pCircBuffer)
```

or 56800E Assembly code (LDM):

```
move.l X:(R2+Offset_pCircBuffer),R0 ; where pCircBuffer contains a pointer
move.l R0,X:(R2+Offset_pCircBuffer)
tst.l  X:(R2+Offset_pCircBuffer)
```

can be converted to:

```
LoadRx X:(R2+Offset_pCircBuffer),R0 ; where pCircBuffer contains a pointer
StoreRx R0,X:(R2+Offset_pCircBuffer)
TestRx X:(R2+Offset_pCircBuffer)
```

2.4 *Push* and *Pop* Instructions

The 56800 has instructions for *push* and *pop* that save and restore 16-bit address registers. The *push* and *pop* instructions should not be used for the 56800E, since these instructions temporarily even the stack. This could cause alignment access problems for long data types, since long data types must be on odd addresses.

If 56800 Assembly code is being ported to the 56800E and it contains *push* or *pop* instructions, this code should be rewritten so it does not use these instructions.

Code Example 2-2. Converting 56800 Assembly Code Using *Push* and *Pop* #1

56800 Assembly code:

```
push R2
...
pop R2
```

can be converted to:

```
adda #2,sp
move.l R2,x:(sp)
...
move.l x:(sp)-,R2
```

Code Example 2-3. Converting 56800 Assembly Code using *Push* and *Pop* #2

DSP56800 Assembly code:

```
push Y0
push Y1
...
pop Y1
pop Y0
```

can be converted to:

```
adda #2,sp
move.w Y0,x:(sp)
move.w Y1,x:(sp-1)
...
move.w x:(sp-1),Y1
move.w x:(sp),Y0
suba #2,sp
```

2.5 Structure Offsets

If your assembly code accesses elements of a structure defined in C, offsets to the elements of the structure must be defined in Assembly. In *portasm.h*, a *PTR_SIZE* macro has been created that defines the size of a pointer (1 for SDM and 2 for LDM). To seamlessly switch from SDM to LDM and vice versa, assembly code that defines offsets to these elements should be relative to the previous element and *PTR_SIZE* should be used when specifying the size of a pointer.

Also, structure elements that are longs should be the first elements in the structure, because CodeWarrior requires that long values start on an odd address. Structure elements that are pointers should follow because they will be longs for LDM. By having all of the longs and pointers first, the start of the structure will be on an odd address and there will be no gaps in the structure. If longs or pointers were placed in the middle of the structure, then gaps would be inserted to guarantee an odd address. Because Assembly code is written to assume no gaps, this would cause a problem.

Code Example 2-4. Convert C structure

If C structure is defined as follows:

```
struct C_Structure {
      bool   bStatus;
      int *  pCircBuffer;
      int    Size;
      UWord32 LongWord;
};
```

then you can change the C structure to the following:

```
struct C_Structure {
      UWord32 LongWord;
      int *  pCircBuffer;
      bool   bStatus;
      int    Size;
};
```

Code Example 2-5. Convert Assembly Code Structure

If Assembly code defines these offsets to each element as follows:

```
Offset_bStatus      equ 0
Offset_pCircBuffer  equ 1
Offset_size         equ 2
Offset_LongWord     equ 3
```

then you can change the Assembly code to the following:

```
Offset_LongWord     equ 0
Offset_pCircBuffer  equ Offset_LongWord+2
Offset_bStatus      equ Offset_pCircBuffer+PTR_SIZE
Offset_size         equ Offset_bStatus+1
```

Index

D

DSP vii
    Digital Signal Processor vii

L

LDM vii
    Large Data Memory Model vii
LoadRx 1-1
long 1-1
longs 1-3

M

M56800E Assembler 1-1

P

pointers 1-3
pop 1-2
portasm.h 1-1, 1-3
PTR_SIZE macro 1-3
push 1-2

S

SDM vii
    Small Data Memory Model vii
StoreRx 1-1

T

TestRx 1-1

W

word 1-1

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

MOTOROLA

CG56800E