Freescale Semiconductor
Application Note

Document Number: AN4414

CodeWarrior Build Tools Options for Optimal Performance on HCS08 Cores

1 Introduction

This document describes two sets of options and pragmas that can be used with the CodeWarrior tools to produce optimal code for the HCS08 cores. One set optimizes the size of the code; another set optimizes the execution speed. You can use the build tools options and pragmas described in this document for optimal performance, but the build tools settings must be set according to the application being developed.

For more information on the HCS08 Compiler, refer to CodeWarrior Development Studio for Microcontrollers V10.x HC(S)08 Build Tools Reference Manual by Freescale.

2 Optimization for Size

Several strategies can be used, simultaneously if the application allows, to optimize the size of the generated code:

- use specific compiler options
- choose a less expensive memory model

Contents

- place code and/or data in segments which require shorter addressing.

The following sections provide details on each of these approaches.

For additional hints on code size optimization, refer to topic Generating Compact Code in CodeWarrior Development Studio for Microcontrollers V10.x HC(S)08 Build Tools Reference Manual by Freescale.

2.1 Compiler Options

For compact code, pass the following options to the compiler:

```
-Os -Ous -Of
```

A specific set of options may reduce the code size for some functions, but increase it for other functions in the same compilation unit. In order for the compiler to dynamically configure options for each function, invoke it with option `-OdocF`. This option takes a list of option sets as input, and instructs the compiler to run with each option set and keep the one that produces the best code size for each function. For example: `-OdocF="-Or|-Cni -Cu|-Oc".`

> **NOTE** The compilation time multiplies by a factor equal to the number of option sets passed to the compiler.

2.2 Memory Models

The TINY memory model is the least expensive one - with the BANKED memory model being the most expensive. Use the TINY memory model if the application allows, that is, all data, including stack, fits into the zero page area (between 0 and 0xFF). Use the SMALL memory model if the application code and data fits into the 64KB address space.

To select TINY as the memory model, pass option `-Mt` to the compiler. To select SMALL as the memory model, invoke the compiler with option `-Ms`.

For more information on memory models, refer to topic Memory Models in CodeWarrior Development Studio for Microcontrollers V10.x HC(S)08 Build Tools Reference Manual by Freescale.

2.3 Compiler Pragmas for Data and Code Allocation

If the selected memory model is either SMALL or BANKED, use __SHORT_SEG data segments in order to allocate a subset of the application data on the zero page (0 to 0xFF). Objects allocated on this page are accessed using 8-bit addressing. To specify that a data segment is a __SHORT_SEG segment, use modifier __SHORT_SEG with either a CONST_SEG or a DATA_SEG pragma.

For example:

```
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
unsigned int var1;  /* 8b access */
#pragma DATA_SEG DEFAULT
```

```
unsigned int var2;  /* 16b access */
```

If the selected memory model is BANKED, use __NEAR_SEG code segments in order to mark a subset of the application functions as __near. Such functions are accessed using 16-bit addressing (instead of the default 24-bit addressing). To specify that a code segment is a __NEAR_SEG segment, use modifier __NEAR_SEG with a CODE_SEG pragma.

For example:

```
#pragma CODE_SEG __NEAR_SEG NON_BANKED
void foo() {}  /* JSR/RTS */
#pragma CODE_SEG DEFAULT
void bar() {} /* CALL/RTC */
```

For more information on data/code placement pragmas, refer to topic Pragma Details in CodeWarrior Development Studio for Microcontrollers V10.x HC(S)08 Build Tools Reference Manual by Freescale.

3 Optimization for Speed

In order to reduce the overall cycle count of the code, pass the following options to the compiler:

```
-Ot -Cu -Ous -Onf -Oi -OiLib
```

To further improve execution speed, configure the compiler to use shorter addressing:

-select the least expensive memory model that the application allows (refer to section 2.2 for more details)

-allocate some of the application data objects and functions in segments that require shorter than default addressing: 8-bit, instead of 16-bit, for data, with either the SMALL or the BANKED memory model; 16-bit, instead of 24-bit, for code, with the BANKED memory model (refer to section 2.3 for more details).

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

Document Number: AN4414

23 November 2011