freescale semiconductor

Technical Note (8 & 16-bits) TN 234

Porting Applications from Cosmic to CodeWarrior

Introduction

This application node describes how to port HC08 and HC12 applications written for Cosmic compiler to CodeWarrior. Strict ANSI-C/C++ code can be ported without any modifications. Non ANSI-C keywords and pragmas are different or have different semantics. See chapters below for details. Code written in assembler or inline assembler has to be changed the way that CodeWarrior's calling conventions and the C - Assembly interface rules are fulfilled.

Related Documents

CodeWarrior Compiler Manual, section Appendix, chapter Migration Hints

Conceptual Differences between the two Compilers

1. The code produced by CodeWarrior is fully re-entrant, even for long and floating point arithmetic. Semaphores to avoid reentrancy and/or code saving library workspace (c_reg, c_lreg …) in interrupt handlers should be removed. CodeWarrior always allocates local variables on stack, or if option –or is active and if variable is suitable for register allocation, in registers. CodeWarrior does not offer the possibility to allocate local variables in not initialized shared data sections like Cosmic does.
2. CodeWarrior supports code and/or data banking when applicable to the target CPU (Banking not available on HC08 CPU).
3. The Cosmic Compiler distributes data into initialized and not initialized data segments automatically. CodeWarrior respects what the programmer specifies in his link parameter file.

Porting non ANSI-C Keywords

Semantics of the non ANSI-C Keywords

Keyword	Cosmic	CodeWarrior
tiny	Objects: the object has a 8 bit address Pointers: @tiny char *p; 8 bit large pointer char * @tiny p; pointer has 8 bit address	n.a.
near	Objects: the object has a 16 bit address Pointers: @near char *p; 16 bit large pointer char * @near p; pointer has 16 bit address Functions: near calling convention	HC08: char *__near p; 8 bit large pointer HC12: char *__near p; 16 bit large pointer Functions: near calling convention
far	Objects: the object has a 24 bit address Pointers: @far char *p; 24 bit large pointer char * @far p; pointer has 24 bit address Functions: far calling convention	HC08: char *__far p; 16 bit large pointer HC12: char *__far p; 24 bit large pointer Functions: far calling convention
eeprom	Objects are allocated in EEPROM area. Writing accesses are performed by a derivative specific runtime routine	n.a. (This reflect state of CodeWarrior software today. This may be implemented in a future release of the tool.)
interrupt	Function declared with interrupt keyword returns from an interrupt	Function declared with interrupt keyword returns from an interrupt. Interrupt number can be specified optionally.
_Bool	Objects declared with _Bool type are 1 bit large	n.a. (use ANSI-C bitfields)

asm	Enclose your inline assembly code with
	#asm
	<assembly code>
	#endasm
		Enclose your inline assembly code with
		__asm {
		<assembly code>
		}

Remark for tiny/near/far usage together with pointers:

Cosmic: Putting a @tiny/@near/@far on the left hand side of a pointer star symbol specifies the *size of a pointer*, e.g. @tiny char *p. Putting a @tiny/@near/@far on the right hand side of a pointer star symbol specifies *where the pointer is allocated*, e.g. char *@tiny p.

CodeWarrior: near or far can only be placed at the right hand side of a pointer and it specifies the *size of the pointer*.

Porting HC08 Code

The following table gives a CodeWarrior counterpart for every Cosmic keyword:

Cosmic	CodeWarrior	Remarks for CodeWarrior
`@tiny`	`@"zseg_name"`	#pragma DATA_SEG SHORT "zseg_name" has to be defined once in a compilation unit before using @"zseg_name"
`@near`	`int *__near p;`	near only allowed for pointers (8 bit wide pointer in small memory model)
`@far`	`int *__far p`	far only allowed for pointers (16 bit wide pointer in tiny memory model)
`@eeprom`	`n.a.`	EEPROM allocation is not supported
`@interrupt`	`__interrupt <number>`	Optional <number> to specify interrupt vector number
`@nostack`	`n.a.`	Shared local memory not available
`_Bool`	`n.a.`	Bit variables are not supported
`#asm … your assembly code … #endasm`	`__asm { … your assembly code … }`	CW support both syntax

Porting HC12 Code

The following table gives a CodeWarrior counterpart for every Cosmic keyword:

Cosmic	CodeWarrior	Remarks for CodeWarrior
`@tiny`	`@"zseg_name"`	#pragma DATA_SEG __SHORT_SEG "zseg_name" has to be defined once in a compilation unit before using @"zseg_name"

`@near`	`@"nseg_name"`
	
	`__near void func(void)`
	`int *__near p;`
		#pragma DATA_SEG __NEAR_SEG "nseg_name" has to be defined once in a compilation unit before using @"nseg_name"
		
		near only allowed for function definitions and 16 bit pointers (in large memory model)

`@far`	`@"fseg_name"`
	
	`__far void func(void)`
	`int *__far p;`
		#pragma DATA_SEG __FAR_SEG "fseg_name" has to be defined once in a compilation unit before using @"fseg_name"
		
		far only allowed for function definitions (HC12 only) and 24 bit pointers (in small/ banked memory model)

`@eeprom`	`n.a.`	EEPROM allocation is not supported in CW
`@interrupt`	`__interrupt <number>`	Optional <number> to specify interrupt vector number in CW
`@nostack`	`n.a.`	Shared local memory not available
`_Bool`	`n.a.`	Bit variables are not supported in CW

`#asm`
`… your assembly code …`
`#endasm`
	`__asm {`
	`… your assembly code …`
	`}`
		CW supports #asm …. #endasm too

Non ANSI-C Keyword Usage Examples

HC08

The following table gives CodeWarrior counterparts for Cosmic example declarations:

Cosmic keyword	Cosmic example	CodeWarrior example
@tiny	@tiny int i; int @tiny i; int * @tiny ptr; @tiny int * ptr	#pragma DATA_SEG __SHORT_SEG "zseg_name" #pragma DATA_SEG DEFAULT … int i @ "zseg_name"; int *ptr @ "zseg_name"; int *__near ptr;
@near	@near int i; int @near i;	int i ;
@far	@far int i; int @far i; int * @far ptr;	#pragma DATA_SEG __FAR_SEG "fseg_name" #pragma DATA_SEG DEFAULT … int i @ "fseg_name"; int * ptr @ "fseg_name";
@interrupt	@interrupt void inthandler(void) { …}	__interrupt void inthandler(void) { …} __interrupt 6 void int6handler(void) { …}

#asm
…
#endasm
	#asm
	nop
	#endasm
		__asm {
		nop
		}
		
		#asm
		nop
		#endasm

HC12

The following table gives CodeWarrior counterparts for Cosmic example declarations:

Cosmic keyword	Cosmic example	CodeWarrior example
@tiny	@tiny int i; int @tiny i;	#pragma DATA_SEG __SHORT_SEG "zseg_name" #pragma DATA_SEG DEFAULT … int i @ "zseg_name";
@near	@near int i; int @near i; int * @near ptr; @near int my_fun(void);	#pragma DATA_SEG __NEAR_SEG "nseg_name" #pragma DATA_SEG DEFAULT … int i @ "nseg_name"; int * ptr @"nseg_name"; __near int my_fun(void);
@far	@far int i; int @far i; int * @far ptr; @far int *ptr; @far int my_fun(void);	#pragma DATA_SEG __FAR_SEG "fseg_name" #pragma DATA_SEG DEFAULT … int i @ "fseg_name"; int * ptr @ "fseg_name"; int * __far ptr; __far int my_fun(void);
@interrupt	@interrupt void inthandler(void) { …}	__interrupt void inthandler(void) { …} __interrupt 6 void int6handler(void) { …}

#asm
…
#endasm
	#asm
	nop
	#endasm
		__asm {
		nop
		}
		
		#asm
		nop
		#endasm

Important Pragmas

Cosmic	CodeWarrior
`#pragma space [] @tiny`	`#pragma DATA_SEG __SHORT_SEG _ZEROPAGE`
`#pragma space []`	`#pragma DATA_SEG DEFAULT`

Calling Conventions and Parameter Passing

The calling conventions and parameter passing are significantly different. Be careful when you port (inline-) assembler code referring to parameters or return values. The next 2 tables show how return values and parameters are passed for HC08 and for HC12. HCS08 is not described.

HC08

Topic	Cosmic	CodeWarrior
Return value	Return value is 1 byte large: passed in A Return value is 2 byte large: passed in X:A Everything else: passed on stack	Return value is 1 byte large: passed in A Return value is 2 byte large: passed in X:A Everything else: passed on stack
Parameters	1st parameter is 1 byte large: passed in A 1st parameter is 2 byte large: passed in X:A Everything else: passed on stack	Last parameter is 1 byte large: passed in X, and if 2nd last parameter is also 1 byte: passed in A Last parameter is 2 byte large: passed in X:A Everything else: passed on stack
Parameter passing order	Right to left	Left to right

HC12

Topic	Cosmic	CodeWarrior
Return value	Return value is 1 byte large: passed in B Return value is 2 byte large: passed in D Return value is 4 byte large: passed in X:D Everything else: passed on stack	Return value is 1 byte large: passed in B Return value is 2 byte large: passed in D Return value is 3 byte large: passed in B:X Return value is 4 byte large: passed in X:D Everything else: passed on stack
Parameters	1st parameter is 1 or 2 byte large: passed in D 1st parameter is 4 byte large: passed in X:D Everything else: passed on stack	Last parameter is 1 byte large: passed in B Last parameter is 2 byte large: passed in D Last parameter is 3 byte large: passed in B:X Last parameter is 4 byte large: passed in X:D Everything else: passed on stack
Parameter passing order	Right to left	Left to right

Interfacing C to Assembly

- The Cosmic assembler references external C objects by a leading underscore: XREF _cobj
- The Cosmic inline assembler references external C objects by a leading underscore: #asm lda _cobj #endasm
- The Cosmic C compiler references external assembler objects without underscore, but the assembler definition must have a leading underscore: XDEF _asmobj
- For CodeWarrior Assembler and Inline Assembler, there is no need of leading underscores for external definitions and references. This is easier, but register names cannot be used as object names.

Language	Cosmic	CodeWarrior

C	`extern int myasmobj;`
	`int myCobj;`
	`…`
	`myCobj = myasmobj;`
		`extern int myasmobj;`
		`int myCobj;`
		`…`
		`myCobj = myasmobj;`

Inline Assmbly	`#asm`
	`lda _myCobj`
	`sta _myasmobj`
	`#endasm`
		`#asm ;or`
		`__asm {`
		`lda myCobj`
		`sta myasmobj`
		`#endasm ;or }`

Assembly	`XDEF _myasmobj`
	`XREF _myCobj`
	`…`
	`lda _myCobj`
	`sta _myasmobj;`
		`XDEF myasmobj`
		`XREF myCobj`
		`…`
		`lda myCobj`
		`sta myasmobj;`

Assembly Pseudo Instructions (Aliases)

CodeWarrior does not support Cosmic assembler pseudo instructions, unless they are specified in a binary application interface.

HC12X

Cosmic	CodeWarrior
`lbsr fun`	`JSR fun,PCR`
`clrd`	`CLRA` `CLRB`
`lslw`	`ASLW`
`tstd`	`CPD #0`

lslx	ASLX
lsly	ASLY

Coding Example

Following HC08 example show how object allocation and pointer kinds are translated from Cosmic to CodeWarrior:

Cosmic	CodeWarrior

`@tiny char tch1,tch2;`
`@tiny char * ptch;`
`@tiny char * @tiny tptch;`
 	
`void foo(void) {`
`  ptch = &tch1;`
`  *ptch = 0;`
`  tptch = &tch2;`
`  *tptch = 0;`
`}`
	`#pragma DATA_SEG __SHORT_SEG Zeropage`
	`#pragma DATA_SEG DEFAULT`
	`…`
	`char tch1,tch2 @"Zeropage";`
	`char *__near ptch;`
	`char *__near tptch @"Zeropage";`
	
	`void foo(void) {`
	`  ptch = &tch1;`
	`  *ptch = 0;`
	`  tptch = &tch2;`
	`  *tptch = 0;`
	`}`