TASKING_®

CROSS-LINKING SMARTCODE AND TRICORE VX V6.3R1 BINARIES INTO A SINGLE ABSOLUTE LOCATED FILE

APPLICATION NOTE





THIS APPLICATION NOTE WILL FOCUS ON CROSS-LINKING SMARTCODE BINARIES AND TRICORE VX V6.3R1 BINARIES INTO A SINGLE ABSOLUTE LOCATED FILE.

INTRODUCTION

Cross-linking is an important tool in the embedded software engineer's toolbox. Today's embedded safety systems include a myriad of performance, safety, and security requirements. These systems frequently rely on legacy binaries that have a proven track record when it comes to safety and reliability.

The TASKING TriCore development tools provide the capability to easily integrate relocatable object files (suffix .o) from legacy code built with older versions of the toolset.

The recently released SmartCode toolset which was designed to support the third generation of the AURIX[™] microcontroller (TC4x) accepts and links object files built with v6.3r1 of the TriCore VX-Toolset.

The following application note will show two simple examples of how to do this. Please refer to the following illustration for a high-level view of the process.



WHAT IS CROSS-LINKING?

Cross-Linking is a feature of the TASKING TriCore toolset that allows you to re-use object files built with previous versions of the toolset and link them into application code that is being developed with a newer version of the TriCore toolset. For this application note, SmartCode v10.1r1 is newer than TriCore VX v6.3r1.

WHY IS SOFTWARE RE-USE IMPORTANT?

Critical software components with a proven safety record are often re-used due their history of operating in the field without incident. The re-use of these legacy object files and binaries can save time and money since they have already undergone a costly validation and verification procedure.







Other notable reasons for using legacy software components:

- SW architects with detailed information on the algorithm IP may no longer be with the company
- Source code may not be available for re-compilation
- OEM safety case may require usage of specific SW components
- Third-party validated safety compliant SW components like MCAL and/or AUTOSAR specify compiler version and toolset options in their release notes.

WHY DO WE WANT TO USE THE NEWEST VERSION OF THE TOOLSET FOR OUR APPLICATION DEVELOPMENT?

The evolution of compiler toolsets like TASKING's SmartCode are continually introducing new features to increase the performance, safety, and efficiency of your embedded applications.

Examples of how new toolset features can improve software development:

- New application features can benefit from performance gains and improvements available with the latest compiler technology
- Latest compiler optimizations can improve performance and reduce footprint size allowing your application to fit into a more cost-effective device
- The improved performance and reduced application footprint size can help achieve any OEM specified overhead requirements.

WHY DO WE WANT TO CROSS-LINK?

Let us assume that the option to cross-link is not available. The developer of a new software component would have the following options:

- 1. The version of the compiler used for building the software component would have to be used for the rest of their software
- 2. All future software development with this component would be restricted to this version of the toolset
- 3. Any other applications wanting to use the software component would have to be ported to this version of the toolset.

Clearly this is undesirable. Thankfully, this is not the case. Software developers using the TASKING TriCore toolsets who need to use legacy object code built with previous versions of the TriCore toolset can cross-link with a newer version of the toolset. This provides the developer the following advantages:

- Seamlessly adding legacy object file(s) to their current build
- Being able to use certified third-party software (I.e., MCAL) that was developed with a previous toolset version.
- Commercial advantages -> avoid re-writing, porting, testing, or validating legacy object files

I think it is clear, why we want to cross-link, but should we? Is it safe? Why shouldn't we fear linking object files from different compiler versions? After all, calling conventions may have changed, data layouts may be different and who knows what other surprises lurk in the dark recesses of a micro-controller's memory map?





HOW DOES TASKING ENSURE THAT CROSS-LINKING IS SAFE?

All TriCore compiler versions adhere to the Infineon Embedded Application Binary Interface (EABI) specification. The EABI is a set of interface standards that writers of compilers, assemblers and linker/locators must use when creating compliant tools for the TriCore architecture. For example, characters must be stored on byte boundaries, short integers must be twobyte aligned and any data types with a size of four bytes or larger must be four-byte aligned.

Prior to specifying that two compiler versions are suitable for cross-linking, TASKING methodically compares the relevant aspects of the two compiler versions, such as data sizes and alignments, and tests the compatibility of the calling conventions. An additional set of tests has been generated to cover all possible cases; passing these tests means that the two compiler versions are cross-link compatible.

Note: The TriCore architecture has requirements related to the alignment of certain data types, structs and struct members. The following points should be considered:

- Alignment restrictions of the TriCore architecture
- Alignment requirements of the Infineon EABI
- Alignment options provided by the C compiler
- Alignment options provided by the LSL linker script language

Note: For additional details on the Infineon EABI in relation to the TASKING toolset, please refer to the application note "Alignment Requirements – Restrictions for the TriCore Architecture" located on the 'resources' tab of the TASKING website.

WHAT ISSUES CAN CAUSE CROSS LINKING ERRORS?

With the EABI, Infineon defines the calling conventions that are used; however, compiler vendors may offer alternative settings to achieve a more compact data placement. For example, the TASKING TriCore toolset allows the user to change the section/ data alignment to increase data density and reduce unnecessary alignment gaps. In addition, TASKING has introduced some compiler flags to enable or disable specific deviations from the standard EABI.

IMPORTANT: Please note, that some of the following cross-linking points no longer apply to SmartCode since it was designed to always be EABI compliant. (I.e., SmartCode does not support the --eabi option to disable EABI compliance.) However, these points are applicable, when trying to cross-link between two previous versions of the TriCore VX-toolset (e.g, v5.0r2 and v6.2r2).

To ensure cross-linking is successful:

- Ensure all object files are compiled with the same settings for these flags.
- Ensure all objects use the same size for the double type
- Ensure alignment options also match.





Potential issues with cross linking:

- EABI Compliance deviations within TriCore VX Toolset
 - Main two EABI settings affecting cross-linking are half-word-align AND treat double as float
 - Base type alignment may be an issue with bit-field alignment. Bit-field alignment issues can lead to errors when trying to access a register bit.
 - ISO C supports '0' size bit-field. This can lead to errors with Base type alignment. This topic is beyond the scope of this application note.
 - The -no-clear option violates the EABI. The specification declares that non-initialized global data must be zero'ed in startup code (I.e.,data needs to be cleared before used). However, Applications using battery backed RAM, need to know the content of RAM after a re-start. The -no-clear option ensures that data in battery backed RAM is retained and not overwritten by the initialization.
 - Word aligned structures and unions larger than or equal to 64-bit can cause compatibility issues with SmartCode. TASKING included for double word copy instructions (Not EABI compliant).
- Potential issues with cross-linking may arise when using the compiler options `--integer-enumeration' (always uses 32-bit integers for enumeration) and `--signed-bitfields'. Refer to Section 7.5 of the TASKING VX-toolset for TriCore v6.3r1 for more information.

CROSS-LINK A SIMPLE PROJECT BUILT IN TRICORE VX AND SMARTCODE

Create a simple TriCore project with a library cross-linked to SmartCode.

1. First step is to create a 'hello world' project in TriCore VX.

New C/C++ Project		^
/C++ Project		\diamond
Create a new C/C++ project for the TASKING VX-toolset for friCore	_	4
Project name: HelloTriCoreVXToolset_v63r1		
✓ Use default location		
.ocation: C:/Users/roger.smith/workspace_ctc_v6.3r1_PPU	В	rowse
Hello World C++ Project TASKING TriCore Position Independent Module TASKING TriCore Library TASKING TriCore MIL Library		

Figure 2: VX-Toolset for TriCore, New C/C++ Project dialog box





2. Choose an AURIX variant for the project. In this example, we use the TC39xB

New C/C++ Project	- 🗆 X
TriCore Project Settings	
Set options to create a TriCore project	
Processor selection	
V AURIX 2G Family	Expand All
TC33x	
TC33xEXT	Expand Selected
	Collapse All
TC37x	
TC38x	
TC39x	
✓ TC39xB	
User defined IriCore 1.3	
Multi-core configuration	
Use configuration: All cores \checkmark	
Actions	
Add startup file(s) to the project	
Add linker script file to the project	
Include debugger synchronization utility	
(?) < Back Next > Finish	Cancel

Figure 3: VX-Toolset for TriCore, New C/C++ Project Micro-controller selection dialog box

3. Adjust the project properties to include the following:

- Adjust the Allocation >> Threshold for putting data in _near to 0
- Set generate symbolic information to `default'
- Generate ascii .map file



Figure 4: VX-Toolset for TriCore, New C/C++ Project Settings - memory allocation







4. Modify the Hello_TriCoreVXToolset_v63r1 as shown:

```
#include <stdio.h>
extern int libfunc1(void);
extern int libfunc2(void);
extern int libfunc3(void);
extern int libl, lib2, lib3;
int sumext;
int main_6_3(void)
{
   printf( "Hello world\n" );
   printf("This was built with TriCore VX Toolset v63rl\n");
   sumext=lib1+lib2+lib3;
   printf("TriCore lib sumext= %d\n",sumext);
   return(1);
}
#if 1
void main(void) {
   main_6_3();
}
#endif
```

Figure 5: VX-Toolset for TriCore, Hello_TriCoreVXToolset_v63r1 code modification

5. Create a new TASKING TriCore Library and select the AURIX family variant

Vew C/C++ Project	New C/C++ Project TriCore Project Settings Set options to create a TriCore project	
Project name: janexliba ✓ Use default location Location: C:/Users/roger.smith/workspace_ctc_v6.3r1_PPU Browse Project type: ✓ TASKING TriCore Application ● Empty Project ● Hello World C + Project > TASKING TriCore Position Independent Module ✓ TASKING TriCore Library ● Empty Project > TASKING TriCore Library ● TASKING TriCore MIL Library	Processor selection AURIX 2G Family C33x TC33x TC33xEXT TC35x TC35x TC35x TC37x TC38x TC39x TC39x User defined TriCore 1.3 User defined TriCore 1.6 User defined TriCore 1.6 V	Expand All Expand Select Collapse All
? < Back Next > Finish Cancel	Multi-core configuration Use configuration: All cores ~	Cancel

Figure 6: VX-Toolset for TriCore, New Library and micro-controller dialog boxes







6. Create `3' new source files for the library. Right click on the library name in your workspace, select new, select source file. The three new source files should be named libfunc1.c, libfunc2.c and libfunc3.c

Source File Create a new s	source file.		C
Source folder:	janexliba		Browse
Source file:	libfunc1.c		
Template:	Default C source template	~	Configure

Figure 7: VX-Toolset for TriCore, New Library Source File dialog box

7. Define the contents of the library source files as shown:

```
⊕ * libfunc2.c
⊕ * libfuncl.c.
                                        #include <stdio.h>
 #include <stdio.h>
                                        int lib2=2;
 int libl=1;
int libfunc1(void) {
                                       int libfunc2(void) {
     printf("libfuncl tricore\n");
                                            printf("libfunc2 tricore\n");
                                            return(lib2);
     return(libl);
 }
                                         }
⊕ * libfuncl.c.
 #include <stdio.h>
 int libl=1;
int libfunc1(void) {
     printf("libfuncl tricore\n");
     return(libl);
 }
```

Figure 8: VX-Toolset for TriCore, New Library source code





8. Set your library to the active project and build. The console.log and screenshot of project workspace is shown.

TASKING C/C++ - Hello_TriCoreVXToolset_v63r1/Hello_TriCoreVXToolset_v63r1.c - TriCore Eclipse IDE v6.3r1 File Edit Source Refactor Navigate Search Project Debug Window Help

📬 🕶 🔚 🕼 i+ 📸 = 😂 = 🗳 = 🚱 = 🔑 🗟 📾 🕺	梦 ▼ 🅭 🖋 ▼ ⊿ 🗏 🗊 🖞 ▼ 🖗 ▼ 🖏 ▼ 🗘 ▼
En C/C++ Projects ⊠ <> -> @ □ \$ マ □ □	ibfunc1.c
> 🌮 Hello_TriCoreVXToolset_v63r1	Hello_TriCoreVXToolset_v63r1.c.
🗸 🖾 janexlib [Active - Debug]	
> III: Archives	<pre>#include <stdio.h></stdio.h></pre>
> 🗊 Includes	
> 🔁 Debug	extern int libfunc1 (void);
> c libfunc1.c	extern int libfunc2(void);
> c libfunc2.c	extern int libfunc3(void);
> 💽 libfunc3.c	
DConfig	extern int libl, lib2,lib3;
MConfig	int sumext;
Config	⊖ int main 6 3(void)

📳 Problems 📮 Console 🛛 🔲 Properties CDT Build Console [janexlib] "C:\\Program Files\\TASKING\\TriCore v6.3rl\\ctc\\bin\\amk" -j12 all -a Compiling libfuncl.c Compiling libfunc2.c Compiling libfunc3.c + "C:\Program Files\TASKING\TriCore v6.3rl\ctc\bin\ctc" -H sfr/regtc39xb.sfr --dep-file + "C:\Program Files\TASKING\TriCore v6.3rl\ctc\bin\astc" -D__CPU_=tc39xb -D__CPU_TC39x + "C:\Program Files\TASKING\TriCore v6.3rl\ctc\bin\ctc" -H sfr/regtc39xb.sfr --dep-file + "C:\Program Files\TASKING\TriCore v6.3rl\ctc\bin\astc" -D_CPU_=tc39xb -D_CPU_TC39x + "C:\Program Files\TASKING\TriCore v6.3rl\ctc\bin\ctc" -H sfr/regtc39xb.sfr --dep-file + "C:\Program Files\TASKING\TriCore v6.3rl\ctc\bin\astc" -D__CPU_=tc39xb -D__CPU_TC39X Archiving to janexlib.a creating janexlib.a a - libfuncl.o a - libfunc2.o a - libfunc3.o Time consumed: 661 ms **** End of build ****

Figure 9: VX-Toolset for TriCore, New Library workspace view and build results





9. Copy the janexlib.a, from the "janexlib >> debug folder" -> denoted by green to the "Hello_TriCoreVXToolset_v63r1 main project folder" denoted by light blue.

6	C/C	++	Projects 🛛 🗘 🗘 🗘 🗘 🗘
~	ø	He	o_TriCoreVXToolset_v63r1 [Active - Debug]
	>	10	Binaries
	>	Ы	ncludes
	>	e	Debug
	>	.c	cstart_tc1.c
	>	.h	cstart_tcl.h
	>	.0	cstart_tc2.c
	?	.h	cstart_tc2.n
	2		cstart_tc3.c
	2		cstart_tc3.n
	1		cstart_tc4.b
	(estart_te5.c
	5	h	cstart_tc5.h
	5		cstart_cost
	5	h	cstart.h
	>	G	Hello_TriCoreVXToolset_v63r1.c
	>	.c	sync_on_halt.c
		B	DConfig
			Hello_TriCoreVXToolset_v63r1.launch
		Ś	Hello_TriCoreVXToolset_v63r1.lsl
		010	anexlib.a
			MConfig
			OConfig
	~	010	Sim.out
~	6	jan	xlib
	>	明	Archives
	2	El	Debug
	*	0	libfunc1 o - [TriCore/le]
		(libfunc2.o - [TriCore/le]
		ŝ	libfunc3.o - [TriCore/le]
		>	🔲 janexlib.a
			console.log
			S libfunc1.src
			S libfunc2.src
			S libfunc3.src
			🚡 makefile
			🚡 subdir.mk
	>	.c	libfunc1.c
	>	.c	ibfunc2.c
	>	.0	libtunc3.c
			DConfig
		E	Jeonig

Figure 10: VX-Toolset for TriCore, New C/C++ Project and Library workspace view





10. In the project property settings, add the library, janexlib.a as shown:



Figure 11: VX-Toolset for TriCore, Add Library to Project dialog box

11. Set Hello_TriCoreVXToolset_v63r1 project active, build and debug using the TriCore Simulator. From the debugger, hit `restart' and `resume'



Figure 12: VX-Toolset for TriCore, Program output

12. Create a SmartCode project. Follow the same procedure as above except name the project 'SmartCode_helloworld' and choose the TC49x (This is currently the only TC4x option available).





13. Modify the SmartCode helloworld.c as shown:

🎯 workspace_smartcode_v10.1r1 - SmartCode_helloworld/SmartCode_helloworld.c - SmartCode Eclipse IDE v10.1r1 File Edit Source Refactor Navigate Search Project Debug Window Help 😁 🕶 📓 💼 📄 💷 💷 💷 🔊 🗈 💿 📭 🗁 🖅 👉 🤝 👘 🚳 🌾 😑 🗖 💽 SmartCode_helloworld.c 🗙 C/C++ Projects X ⊕ * SmartCode_helloworld.c. > 😂 cross_linking_example #include <stdio.h> SmartCode_helloworld [Active - Debug] > 🕌 Binaries extern int main_6_3(void); > 🔊 Includes > 🗁 Debug ⊖ int main(void) > cstart_tc1.c { > h cstart_tc1.h printf("Hello world\n");
printf("Created in SmartCode v101r1\n"); > cstart_tc2.c main_6_3(); > h cstart_tc2.h > cstart_tc3.c } > h cstart_tc3.h > cstart_tc4.c > h cstart_tc4.h > cstart_tc5.c > h cstart_tc5.h > cstart_tc6.c > h cstart_tc6.h > cstart.c > h cstart.h > C SmartCode_helloworld.c

Figure 13: SmartCode, New Project workspace view

14. One option is to modify the TriCore Project since the TASKING Linker will not be able to create the SmartCode .elf file with 127 main functions defined. Modify the TriCore source file as shown.

```
Hello TriCoreVXToolset v63rl.c.
 #include <stdio.h>
 extern int libfunc1(void);
 extern int libfunc2(void);
 extern int libfunc3(void);
 extern int libl, lib2,lib3;
 int sumext;
⊖ int main_6_3(void)
 {
     printf( "Hello world\n" );
     printf("This was built with TriCore VX Toolset v63rl\n");
     sumext=lib1+lib2+lib3;
     printf("TriCore lib sumext= %d\n",sumext);
     return(1);
 }
 #if 0
 void main(void) {
     main_6_3();
 #endif
```

Figure 14: VX-Toolset for TriCore code modification







- 15. Re-build the TriCore Project. The linker error is due to the lack of a defined main().
 - For cross-linking, copy `Hello_TriCoreVXToolset_v63r1.o' and the janexlib.a to the SmartCode Project folder. Don't forget to add the **janexlib.a library reference** as shown in **step 10**.



Figure 15: SmartCode project workspace view after VX-Toolset for TriCore ".o" and Library ".a" files are added

16. Build the SmartCode Project. You will receive the following warnings since the TriCore project is based on the TC39xB(TC1V1.6.2) and the SmartCode project is for the TC49x(TC1V1.8)



Figure 16: SmartCode Project Build Results







17. Run the SmartCode project with the simulator. (Hardware is not available)

■ Memory → FSS #1 - SmartCode_helloworld × Hello world Created in SmartCode v101r1 Hello world This was built with TriCore VX Toolset v63r1 TriCore lib sumext= 6

Figure 17: SmartCode and VX-Toolset for TriCore cross-linked program output

You have successfully cross-linked a TriCore VX Toolset project with a library -AND- a SmartCode Project.

This project was very simple and didn't introduce any of the potential stumbling blocks that will be shown in the next example.

DEMONSTRATE CHALLENGES WHEN CROSS-LINKING TRICORE VX AND SMARTCODE

When cross-linking TriCore VX v6.3r1 and SmartCode binaries, special attention needs to be paid with the following.

- Float Point Model: The TriCore TC1.6.x has an integrated Floating-Point Unit (FPU) that supports single precision floating point operations. The TriCore VX toolset offers the compiler option 'treat double as float' which significantly improves performance if true double precision is not required. The SmartCode toolset was developed to support the TC4x which has a double precision FPU. In this use-case, there is the possibility to have a floating-point mismatch. (Double Precision -> 8bytes, Singe Precision -> 4bytes)
- Memory Alignment: The TriCore VX Toolset can specify the alignment of data types in memory. For example, with the --eabi option, halfword memory alignment is possible. The SmartCode toolset had this feature removed and certain data types are always EABI compliant (I.e, int's are always full word aligned, no exceptions).

Note: With 'Global Type Checking' enabled, the toolset will give an error with any memory misalignments.

Please reference *project 220223_cross_linking_example*.

1. Copy the project file to a folder and unzip.

I I I I I Cross File Home S	sLink_TriCoreSm	artCode				_	□ × ^ (?
Pin to Quick Copy Paraccess	Cut Copy pa ste Paste sh pard	ath hortcut Move Copy to • Copy to • Orga	Delete Rename	New folder New	Properties • Open	open - Select a dit Select a listory Invert s Select	election
$\leftrightarrow \rightarrow \uparrow \uparrow$	CrossLink_TriC	CoreSmartCode	~	Q 5	Search CrossLi	nk_TriCoreSmartCod	e
📌 Quick access	^	Name		Date mod	ified	Туре	Size
		SmartCode		2/23/2022	9:43 AM	File folder	
Documents	TriCore			2/24/2022	3:02 PM	File folder	
Downloads Pictures	*	220223_cross_linking_	example.zip	2/23/2022	9:51 AM	Compressed (zipp	66 KB

Figure 18: Cross-link example project 220223







2. Import the SmartCode Project. Select File >> Import.

(Enabling 'copy projects into workspace' will leave original project files untouched.)

	🗑 Import – 🗆 🗙
	Import Projects Select a directory to search for existing Eclipse projects.
🗑 Import — 🗆 🗙	Select root directory: C:\WIP\In_Process\CrossLink_TriCoreSmart(Browse Select archive file: Browse Dmiactr:
Select Create new projects from an archive file or directory.	Cross_linking_example (C:\WIP\In_Process\CrossLink_TriCoreSmart Select All Deselect All
Select an import wizard: type filter text Image: Select an import wizard: Image: Select an import w	Refresh Refresh Options Search for nested projects Copy projects into workspace Close newly imported projects upon completion Hide projects that already exist in the workspace Working sets Add project to working sets Working sets Select
(?) < Back Next > Finish Cancel	(?) < Back Next > Finish Cancel

Figure 19: SmartCode, Import Project dialog boxes

3. In the TriCore Folder, rename gen.txt -> gen.bat

I Image: Second state File Home	ore Share View						- 🗆	×
Pin to Quick Copy Pa access	Cut Marken Copy p iste	hortcut	Copy to *	New folder	Properties	Gpen → Edit History	Select all Select none	
Clipb	oard		Organize	New	Ope	n	Select	
← → ~ ↑ 📘	> crosslink_ex	> TriCore	~	Q 5	Search TriCore	e		
	^	Name	^	Date modif	ied	Туре	Size	
📌 Quick access								
Documents	*	file_2.c		2/23/2022 9	:42 AM	CFile		1 KB
		file_2_complian	nt_settings.o	2/23/2022 9	:41 AM	O File		6 KB
Downloads	*	file_2_runtime	problems.o	2/23/2022 9	:41 AM	O File		6 KB
Pictures	*	gen.txt		2/23/2022 9	:40 AM	Text Docu	ment	1 KB
Cesktop								
In December								

Figure 20: VX-Toolset for TriCore, ,.o' file directory view



The **gen.bat** file contains the command line invocation calls for the TriCore VX Toolset. This batch file will generate two separate object files. The `file_2_compliant_settings.o' is built with the correct EABI and floating-point double settings. The `file_2_runtime_problems.o' is built with `treat double as float' and `half-word' alignment as supported by the TriCore Toolset.

📓 C:\WIP\In_Process\CrossLink_TinCoreSmartCode\TirCore\gen.bat - Notepad++					
le Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?					
) 🖶 🖶 🗞 🕼 🕼 🗶 🕊 👘 👘 🗩 🕊 🛤 🀅 🔍 🔍 🔄 🔤 🖬 📜 🗰 🚱 💷 🐨 🖉 💷 🐨 🖉 💷 🐨 🖉					
enbat [3]					
1 cctc file 2.c -tcore=tc1.6.2 -t -c -g -vglobal-type-checking -o file 2_runtime_problems.o					
2 cctc file_2.c -tcore=tc1.6.2 -t -c -g -veabi=Hfp-model=Fglobal-type-checking -o file_2_compliant_settings.o					
3					

Figure 21: Command Line invocation for VX-Toolset for TriCore ,.o' files

• Open a command prompt (cmd/k) and set the path to include the TriCore VX Toolset v6.3r1 ctc\bin folder. A simple batch file is shown below.

set	temppath=%path%		
set	path=C:\program	files\tasking\TriCore	v6.3r1\ctc\bin; <pre>%temppath%</pre>
Figur	e 22: VX-Toolset for T	ricore, batch file for setting p	ath environment variable

 \bullet From the command prompt, type <code>cctc -help</code>, to show the control program options.

C:\Windows\System32\cmd.exe	
eabi= <flag>,</flag>	diagnostic messages using the specified format: html, rtf or text (default) control EABI compliancy (default: bcfhnsw)
<pre>b/B +/-bitfield-align c/C +/-char-bitfield f/F +/-float h/H +/-half-word-align n/N +/-no-clear s/S +/-structure-return w/W +/-word-struct-align</pre>	<pre>lowercase/+ enables, uppercase/- disables: base type alignment for 0-size bit-field access char bit-field as a byte allow treating 'double' as 'float' allow half-word alignment allow the use of optionno-clear allow structure-return optimization word align structures and unions larger than or equal to 64-bit</pre>
eabi-compliant	the code generated will adher to the EABI
error-file error-limit= <number> exceptions force-c force-c++ force-munch format=<type></type></number>	redirect all diagnostic messages set maximum number of errors (default: 42) enable C++ exception handling treat C++ files as C files force C++ compilation and linking force invocation of C++ muncher set linker output format [inex[spec]
fp-model[= <flag>,] 0 1 2 3</flag>	<pre>floating-point model (default: cflnrSTz) alias forfp-model=CFLNRStZ (strict) alias forfp-model=cFLNRSTZ (precise) alias forfp-model=cFlnrSTz (fast-dp) alias forfp-model=cflnrSTz (fast-sp)</pre>
c/C +/-contract f/F +/-float l/L +/-fastlib n/N +/-nonan r/R +/-rewrite s/S +/-soft t/T +/-tran	<pre>lowercase/+ enables, uppercase/- disables: allow expression contraction treat 'double' as 'float' allow less precise library functions allow optimizations to ignore NaN/Inf allow expression rewriting use software floating point library support trapping on excentions</pre>
z/Z +/-negzero global-type-checking	ignore sign of -0.0 enable global type checking

Figure 23: VX-Toolset for Tricore, build program ,EABI' related options

--eabi = H -> Removes half-word align option.
 --fp-model = F -> Removes treat double as a float.

Note: The TriCore EABI default is: bcfhnsw (The `f' treats `double as a float' -and- the `h' allows half-word alignment.)







• Run the batch file to generate updated object files

C1Windows\System32\cmd.exe	- 0	\times
4 Dir(s) 221,457,010,688 bytes free		^
C:\WIP\In_Process\CrossLink_TriCoreSmartCode>cd tricore		
C:\WIP\In_Process\CrossLink_TriCoreSmartCode\TriCore≻dir Volume in drive C is OS Volume Serial Number is A4CF-2722		
Directory of C:\WIP\In_Process\CrossLink_TriCoreSmartCode\TriCore		
02/24/2022 03:02 PM <dir> . 02/24/2022 03:02 PM <dir> . 02/23/2022 09:42 AM 236 file_2.c 02/24/2022 09:42 AM 6,924 file_2.src 02/24/2022 03:00 PM 6,456 file_2_compliant_settings.o 02/24/2022 03:00 PM 6,440 file_2_runtime_problems.o 02/23/2022 09:40 AM 219 gen.bat 5 File(s) 20,275 bytes 2 Dir(s) 221,457,027,072 bytes free</dir></dir>		
C:\WIP\In_Process\CrossLink_TriCoreSmartCode\TriCore>gen		
C:\WIP\In_Process\CrossLink_TriCoreSmartCode\TriCore>cctc file_2.c -tcore=tc1.6.2 -t -c -g -vglobal-type-check _runtime_problems.o + "C:\program files\tasking\TriCore v6.3r1\ctc\bin\ctc" -DCPU_=userdef162 -DCPU_USERDEF162core=tc1.6.2f tglobal-type-checking -g2 -o file_2.src file_2.c + "C:\program files\tasking\TriCore v6.3r1\ctc\bin\astc" -DCPU_=userdef162 -DCPU_USERDEF162core=tc1.6.2 -g runtime_problems.o file_2.src	ing -o f p-model= sl -o fi	ile_2 +floa le_2_
C:\WIP\In_Process\CrossLink_TriCoreSmartCode\TriCore>cctc file_2.c -tcore=tc1.6.2 -t -c -g -veabi=Hfp-model ype-checking -o file_2_compliant_settings.o + "C:\program files\tasking\TriCore v6.3r1\tct\bin\ctc"fp-model=F -D_CPU_=userdef162 -D_CPU_USERDEF162cor abi=Hglobal-type-checking -g2 -o file_2.src file_2.c + "C:\program files\tasking\TriCore v6.3r1\ctc\bin\astc" -D_CPU_=userdef162 -D_CPU_USERDEF162core=tc1.6.2 -g compliant_settings.o file_2.src	=Fglo e=tc1.6. sl -o fi	bal-t 2e le_2_
C:\WIP\In_Process\CrossLink_TriCoreSmartCode\TriCore>		

Figure 24: VX-Toolset for TriCore, build results generating new ,.o' files

• Copy the `file_2_runtime_problems.o' to the SmartCode project, build the project and launch the debugger. Note the linker warning message regarding the different TriCore Versions in the console window.

B ApplicationNote - cross_linking_example/file_1.c - SmartCode Eclipse	DE v10.1r1		- D X
File Edit Source Refactor Navigate Search Project Debug V	indow Help		
📑 • 📰 🐚 📾 📸 • 🚳 • 🗭 • 🚱 • 🥕 🔛 🛍 🖗 🗞	券・1 😕 🚀 +1 🖳 100 🗐 👖 11 🔄 + 初・や ご ク・ウ・ 📑		Q 🔡 🖽
🚾 C/C++ Projects X 🔅 🗢 🗟 📔 🖶 😫 🗖 🗖	le file_1.c X	- 0	BE Outline X
1 0.0000 100000 10000	<pre>Name Target Name Target</pre>	Als usir cols usir is to ti is to ti is using byte gr is using byte gr is using is using is using	C IS III C Outline X I Status S Course X I Status S Course X I Status S Course X I S Status I S Status I S Status I S Status I S Status I S Status I S Status I S Status I S S
8.9	+ "C:\Program Files\TASKING\SmartCode v10.1r1\tcc\bin\tcc or cross_linking_example.elf -d.,/cross_linking_example. Itc wir5: [core mpervtc] linking file "file_z_runtime_problems.o" with different core architecture "TCIVI.6.2" in 1 Time consumed: 1360 ms "*** End of build ****	lsl -DCP ink task w	PU_=tc49x -D_PROC_TC49Xcore=mpe:vtc with core architecture "TCIV1.8"
	\$		~









Note: When you add an object file to an eclipse project folder, it is treated as an input file, not an intermediate file. Eclipse adds to linker invocation. No need to do a refresh.

B ApplicationNote - cross_linking_example/cstart.c - SmartCode Ec	lipse IDE v10.1r1	na oo iyo da aharada dadada kayada		al la kink in	na chata a san		and de Récalidades com da	- 🗆 X
File Edit Source Refactor Navigate Search Project Debug	g Window Help							
📑 • 🔚 🐚 📾 🕸 • 🥭 🔗 • 🖳 🔌 🌬 🕹 🕨 I	0 . 19 3. 3	.e i+ = 🗷 👪 !+ 🗳	41 图 - 有	• 🏷 🕫 🔶 •	⇔ - 🛃			Q 🔡 🖪 🅸
🎋 Debug 🗙 📃	1 🔆 🕴 🗖 🗍	(x)= Variables × 💁 Breakpo	oints 🙀 Expr	essions	- 8	1919 Registers ×	ł	🗇 🏘 🖻 📑 🛃 🕴 🗖 🗖
✓ ☆ cross_linking_example [TASKING C/C++ Debugger]				10	🕩 🔮 🗶 🙀 🕴	Name	Value	Description ^
✓		Name	Type		Value	> MA ADC		ADC
✓ m Thread [TC49x] (Suspended)		(x)= 'file 1 c'::dvar 2	double		3.4	> MA ASCLIN		ASCLIN
_START() at cstart.c:210 0x80000000		S (= 'file 1 c'rs 1	struct		{ c 1 = 85 'U' i 1 = 30'	S > M CBCU_SBCU_TBC	U_CSBC	CBCU_SBCU_TBCU_
		(x)= 'file 2.c'::dvar 1	float		7.8	> 👬 CBS		CBS
		> (= 'file 2.c'::s 2	struct		{ c 1 = 170 'a': i 1 = 28	> 👬 CLOCK		CLOCK
		(x)= 'file 2.c'::sizeof value s 2	2 unsigned int		6	> M GPR		GPR
						> M CSFR		CSFR Y
		<			>	<		>
<pre>b file_1.c C TC49x.frame[0] D cstart.c X b init sp();</pre>		<		Disassembly >	>	Enter loca	ition here 🗸 🛛 👌	
<pre> pragma optimize restore #pragma section code restore /* initialize stackpointer /* </pre>	******		•		0004: msub 0008: .HALF 0008: .HALF 0006: add 0002: st.h 0012: add.a 0014: ld.bu 0016: add.a 0018: nop	d3,d3,d3,d3,d+0xcd 0x400b d15,d9,d+0x7 [a4]0x14,d0 a0,a9 d0,[a0] a0,a9		•
Console X Jasks		📑 🛃 🐼 📑 🖬 🕇 📑	• • •	Memory R F	SS #1 - cross_linking_ex	ample ×		
Debug[cross_linking_example] Debug Instrument Module: tsimi6p_e Starting Debugger Launching configuration: cross_linking_example Loading 'C:\WIP\In_Process\CrossLink_TriCoreSmartCod	e\SmartCode\crc	oss_linking_example\Debug\	cross_1					~
<			>	< Miller Fit	e	27.1.142		>
				Writable	Smart II	nsert 27:1:142	5	

Figure 26: Cross-linked SmartCode and TriCore project shown in SmartCode debug perspective





NON-COMPLIANT PORTION OF THE EXAMPLE

The TriCore source file, **file_2.c** is shown below:

```
Etypedef struct {
    unsigned char c_1;
    unsigned int i_1;
} STR;
STR s_2 = { .c_1=0xAA, .i_1=0x1122334
unsigned int sizeof_value_s_2 = sizec
double dvar_1;
void init_double_tricore(void)
```

During the build process the TriCore toolset ignores the `double' definition of dvar_1 in the c source and creates `float'. (Based on command line settings)

 Looking at the 	TriCore code in Debugger (`treat	double	as	float'	enabled)
------------------------------------	----------------------------	--------	--------	----	--------	----------

⊖ void init_doub] {	le_tricore(v	void)
dvar 1 = 7.	8;	
}		
	dvar_1	= 7.8;
00000080049922:	ld.w	d15,.1.cnt (0x8000000
00000080049926:	st.w	dvar_1 (0x9000001c),d:
	}	

The Disassembly view shows ld.w and st.w (load and store word)

• Referring to the .map file,

 Link Result

 [in] File
 | [in] Section
 | [in] Size (MAU) | [out] Offset | [out] Section
 | [out] Size (MAU) |

Chip	Group	Section	Size (MAU)	Space addr	Chip addr	Alignment
mpe:pflash00		.zrodata.file_111.cnt (191)	0×0000008	0x80000004	0x00000004	0x00000004
mpe:pflash00		.zrodata.file 21.cnt (240)	0x00000004	0x8000000c	0x000000c	0x00000002
mpe:dlmucpu0	i i	.zdata.file_1.s_1 (197)	0x00000008	0x90000000	0x0	0x00000004
mpe:dlmucpu0	1 1	.zdata.file 2.s 2 (242)	0x00000006	0x9000008	0x0000008	0x00000002
mpe:dlmucpu0	i i	.zdata.file_2.sizeof_value_s_2 (243)	0x00000004	0x9000000e	0x0000000e	0x00000002
mpe:dlmucpu0	1	.zbss.file 1.dvar 2 (196)	0x00000008	0x90000014	0x00000014	0x00000004
mpe:dlmucpu0		.zbss.file_2.dvar_1 (244)	0x00000004	0x9000001c	0x0000001c	0x00000002

dvar_1 is 4 bytes (Float), half-word aligned and located at address 0x9000 001c







The variable and memory windows are shown:

				·		
Aonitors	🕂 🗶 🔆	0x9000001c <hex></hex>	0x900001c:0x9000	001C < Floating Poin	🗙 🕂 New Renderi	ngs
♦ 0x900001c		0x000000009000001C	7.80000E0	0.00000E0	0.00000E0	0.00000E0
		0x000000009000002C	0.00000E0	0.00000E0	0.00000E0	0.00000E0
		0x000000009000003C	0.00000E0	0.00000E0	0.00000E0	0.00000E0
		1				
A)= Variables × Se Breakpoints	😚 Expressi	ons) 🅅 🚽		r 🗙 🍇 8 🗖 🗖	1	
Nama	Ture		Value	··· •• •• · · · · · · · · · · · · · · ·		
Name	iype		value			
(x)= 'file_1.c'::dvar_2	double		3.4			
> 🥭 'file_1.c'::s_1	struct		{ c_1 = 85 'U'; i_	1 = 305419896; }		
(x)= 'file_2.c'::dvar_1	float		7.8			
> 🥭 'file_2.c'::s_2	struct		{ c_1 = 170 'a'; i	1 = 287454020; }		
(x)= 'file_2.c'::sizeof_value_s_2	unsigned	int	6			
				~		
				~		

Figure 27: SmartCode, Debug Perspective showing global variables

The debugger correctly displays dvar_1 as type *float* with a value of **7.8**. The problem will be highlighted below when we print out the variables.

file_1.c is built with SmartCode. (Treats 'double' as a 'double' and type int is word-aligned)







- The Check_double() function uses a pointer to the variable &dvar_1 (defined in TriCore) allowing you to `cast' a double value to a float value.
- The function init_double_smartcode(), defines a constant value of 3.4 which is shown as ld.d and st.d (load and store double) in the disassembly view. (8 bytes for the value 3.4)

void init_double_smartcode(void)

{ dvar_2 = 3.4;		
}		
48	dvar 2	2 = 3.4;
000000008004990c:	ld.d	d0/d1,0x80000004
0000000080049910: 49	st.d }	dvar_2 (0x90000014),d0/d1

- The function init_double_tricore() was already discussed.

- The printf() calls display: dvar_1, pointer to dvar_1 and dvar_2.

	<pre>printf("The expected printf("Printing the</pre>	double value of <u>dvar_</u> float value of <u>dvar_</u> 1	is 7.8. The value read is instead of the double value	; : %e\n\n", dvar_1); ue : %f\n\n", *fp_1);
}	<pre>printf("The expected</pre>	value of <u>dvar</u> 2 is 3.4	4. The value read is	: %e\n\n", dvar_2);
The	expected double valu	ue of dvar 1 is 7.8.	The value read is : 5.38	5808e-315

Printing the float value of dvar_1 instead of the double value	:	7.800000
The expected value of dvar_2 is 3.4. The value read is	:	3.400000e+00

• The first printf call displays dvar1 with the `%e' conversion character. Per the TASKING SmartCode user manual, this represents `type double'.

Character	Printed as
d, i	int, signed decimal
0	int, unsigned octal
x, X	int, unsigned hexadecimal in lowercase or uppercase respectively
u	int, unsigned decimal
С	int, single character (converted to unsigned char)
s	char *, the characters from the string are printed until a NULL character is found. When the given precision is met before, printing will also stop
f, F	double
e, E	double
g, G	double
a, A	double
n	int *, the number of characters written so far is written into the argument. This should be a pointer to an integer in default memory. No value is printed.
p	pointer
r, Ir	fract,lfract
R, IR	accum,laccum
%	No argument is converted, a '%' is printed.

printf conversion characters

Figure 28: SmartCode, printf data format options





From the disassembly view, notice printf() loads a double. [printf() gets 8bytes, but there are only 4 bytes of data!]

40	printf(The expected double value of dvar_1 is 7.8. The value read	is :	%e\n\n", dvar_1);
0000000080049850:	ld.d	d0/d1,dvar_1 (0x9000001c)		
000000080049854:	st.d	[sp]0x0,d0/d1		
0000000080049858:	movh.a	a4,#0x8005		
000000008004985c:	lea	a4,[a4]-0x648d		
000000080049860:	call	printf (0x80049a30)		

Let's take a closer look at the math. As previously mentioned, the memory address of dvar_1 is 0x9000 001c.

University × Spr F33	#1 - cross_linking_ex	ampie						-00	~0	4 M 🗋
Monitors	🕂 🗙 🖗	0x9000001c:0x9000001C	<h< th=""><th>× 0x</th><th>900000</th><th>1c <sigr< th=""><th>ned Inte</th><th>ger></th><th>0x90</th><th>00001c <flo< th=""></flo<></th></sigr<></th></h<>	× 0x	900000	1c <sigr< th=""><th>ned Inte</th><th>ger></th><th>0x90</th><th>00001c <flo< th=""></flo<></th></sigr<>	ned Inte	ger>	0x90	00001c <flo< th=""></flo<>
Ox90000000		Address	0	1	2	3	4	5	6	7
Ox9000001c		000000090000018	33	33	ØB	40	9A	99	F9	40
0x90000014		000000090000020	00	00	00	00	00	00	00	00
		000000090000028	00	00	00	00	00	00	00	00
		000000090000030	00	00	00	00	00	00	00	00
		00000009000038	00	00	00	00	00	00	00	00
		000000090000040	00	00	00	00	00	00	00	00
		000000090000048	00	00	00	00	00	00	00	00
		000000090000050	00	00	00	00	00	00	00	00

Figure 29: SmartCode, Debug Perspective memory watch

The hex integer value (32-bit) of 0x9000 001c = 40F9 999A

Converting this value to Float (32-bit) and Double (64-bit) ->

	HEX		4			0)			F			9				9				9			1	9			1	4			Т	Т	Γ	Г	Γ	Γ			Т	T	Τ		Π	Π	П		T		Τ	Т	T	Т	T	Τ	Т	П	Т	
	Binary	0	1 (o o	0	0	0	0	1 1	1	1	1	0	0	1	1	0	O	1	1 (0	0	1	1	0	1	1	0	1	0																													
																																														П													
	Sign	0																																												Ц													
	Exponent		1 (0 C	0	0	0	0	1					\downarrow		\downarrow		\downarrow		\downarrow			\downarrow	\perp								\perp												Ц	\square	Ц		\perp		\downarrow		\perp		\perp			Ľ		
	Mantissa								1	1	1	1	0	0	1	1	0	D	1:	1 (0	0	1	1 0	0	1	1	0	1	0														\square		Ц		_							_				
Convert to Float			Ц													\downarrow		\downarrow		\downarrow			\downarrow	\perp								\perp												Ц	\square	Ц		\perp		\downarrow		\perp		\perp			Ľ		
	Sign	+	Ц													1		1		1			1																					Ц		Ц						\downarrow		\downarrow			\square		
	Exponent		1x	27	+1)	20	= 1	129)																							\perp												Ц		Ц						\downarrow		\downarrow			\square		
	Mantissa								1	x2	++	1x	2-2	+1	x2	-3	1)	(2	·6+	+	1)	2	21	1	95	00		00	47	68														Ц		Ц				\downarrow		\downarrow		\downarrow	_		Ľ		
																																\perp												Ц		Ц								\downarrow					
							+1	*	2 ⁽¹	29-1	127) *	1.9	95	00	00	004	47	68	37	71	58	=	7.8	3				0.000					1																									
																																														Π													
	Binary	0	0	0 0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	D	0	0	0		0	0	0	0	0	0	0 1	L	0	0	0	0	0	1	1	1 1	1	0	0	1	1	0	0	1	1	0	0	1	1 (0	0 1	1	0	1	0
	Sign	0																																												\square													
	Exponent		0	0 0	0	0	0	0	D	0	0																																																
												0	0	0	0	0	0	0	0	D	0	0			0	0	0	0	0	0	0	L	0	0	0	0	0	1	1	1 1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	2 1	1	0	1	0
Convert to Double																																																											
	Sign	+																																																									
	Exponent			0	- 1	023	3 =	-1	02	3	_																																																
	Mantissa		Ц													_		_		_		_	_	_	_	_	_	_			0	.0	00	00	04	84	10	014	74	5	_	_	_		_	_								_	_	_	_	_	
					+1 * 2 ⁽⁻¹⁰²³⁾ * 0.0000004841014745 = 5.385807680000000e-315																																																						

Figure 30: Procedure for converting INT to FLOAT and DOUBLE





- The second printf() displays the contents of a pointer. With type casting we reference the address of a double but show it as a float. Thus, the correct result.
- The third printf() shows the value in SmartCode which is always a double. Per the .map file, the address of dvar_2 is 0x9000 0014.

🚺 Me	mory 🗙 🗔 FSS	#1 - cross_linking_ex	ample						0,	Co 101	1010
Monito	rs	🕂 🗙 🖗	0x90000014:0x90000014	<hex i<="" th=""><th>ntege</th><th>X</th><th>0x90000</th><th>)14 < Sig</th><th>ned Int</th><th>eger></th><th>-</th></hex>	ntege	X	0x90000)14 < Sig	ned Int	eger>	-
4	0x9000000		Address	0	1	2	3	4	5	6	7
•	0x9000001c		0000000090000010	00	00	00	00	33	33	33	33
•	0x90000014		000000090000018	33	33	ØB	40	9A	99	F9	40
			0000000090000020	00	00	00	00	00	00	00	00
			000000000000000000000000000000000000000	aa	00	00	00	00	00	60	99

Figure 31: SmartCode, Debug Perspective memory watch

The hex integer value (64-bit) of 0x9000 0014 = 400B 3333 3333 3333

	HEX		4	í.		0			0			В			3			3				3			3			3			3			3			3	1			3			3	í.		1	3			3	
	Binary	0	1 0	0 0	0	0 0	0	0 0	0	0	1 0	1	1	0	0 1	1	0	0	1 1	10	0	1	1 (0 0	1	1	0	0 1	1	0	0 1	1	0	0 1	1	0	0	1 1	1 C	0) 1	1	0	0	1 :	L	0	1	1 (0 0) 1	1
							Π		Π	Τ									Τ			Π	Τ			П											Τ					Γ					Γ					
	Sign	0					Π		Π	Τ						Γ					Τ	Π	Τ			Π						Π						T	T	T	Τ	П			T		Γ	Π	T	Τ	Τ	
	Exponent		1 0	0 0	0	0 0	0	0 0	0	0						Γ						Π	Τ			Π													T			Π					Γ				Γ	
		Π					Π		Π	1	1 0	1	1	0	0 1	1	0	0	1 1	1 0	0	1	1 (0 0	1	1	0	0 1	1	0	0 1	1	0	0 1	1	0	0	1 1	1 0	0 0) 1	1	0	0	1 :	LC	0	1	1 (0 0) 1	1
Convert to Double							Π			Τ												Π	Т			Π											Τ										Γ				T	
	Sign	+					Π		Π	Τ						Γ					Τ	Π	Τ			Π												Τ	Τ			Π			T		Γ		T	Τ	Τ	
	Exponent			10	24	- 1	023	3 = 3	1	Т						Γ						Π	Τ			Π											Τ										Γ				Γ	
	Mantissa																	18	1.1	101	110	001	10	01	10	01	100	011	100	11	00:	110	01	10	01	100	01:	100	011	10	01	10	01	1			_					
	10 16																																																			
																							+	1*	2(1	1) *	1.7	7 =	3.4	1																					_	

Figure 32: Procedure for converting INT to DOUBLE

• The Check_struct_member() function shows how the struct member alignment affects the results:

```
void check_struct_member(void)
{
    printf("s_1.c_1 value is: 0x%x\n\n", s_1.c_1);
    printf("s_1.i_1 value is: 0x%x\n\n", s_1.i_1);
    printf("sizeof(s_1) is : %d\n\n", sizeof(s_1));
    printf("s_2.c_1 value is: 0x%x\n\n", s_2.c_1);
    printf("sizeof(s_2) is from SmartCode view : 0x%d\n\n",sizeof(s_2));
    printf("sizeof(s_2) is from TriCore view : 0x%d\n\n",sizeof_value_s_2);
}
```

Note: s_1 is built with SmartCode and s_2 is built with TriCore. The printf() formatter $%x \rightarrow$ hex display data (refer to table in user guide shown above)





The TriCore initialized struct members:

The TriCore initialized struct members:

```
cf file_1.c × cc cstart.c
    #include <stdio.h>
    typedef struct {
        unsigned char c_1;
        unsigned int i_1;
    } STR;
    STR s_1 = { .c_1=0x55, .i_1=0x12345678 };
    ///
```

Per the .map file, s_1 is located at 0x9000 0000 and s_2 is located at 0x9000 0008

Chip	Group Section	Size (MAU)	Space addr	Chip addr	Alignment
<pre>mpe:pflash00 mpe:nflash00</pre>	.zrodata.file_111.cnt (191)	0x00000008	0x80000004	0x00000004	0x00000004
<pre>mpe:dlmucpu0 mpe:dlmucpu0</pre>	.zdata.file_1.s_1 (197)	0x00000008	0x90000000	0x0	0x00000004
	.zdata.file_2.s_2 (242)	0x00000006	0x9000008	0x00000008	0x00000002
<pre>mpe:dlmucpu0 mpe:dlmucpu0 mpe:dlmucpu0</pre>	.zdata.file_2.sizeof_value_s_2 (243)	0x00000004	0x9000000e	0x0000000e	0x00000002
	.zbss.file_1.dvar_2 (196)	0x00000008	0x90000014	0x00000014	0x00000004
	.zbss.file_2.dvar_1 (244)	0x00000004	0x9000001c	0x0000001c	0x00000002

Looking at the memory location 0x9000 0000 and 0x9000 0008:

🚺 Memory 🗙 🐺 FSS	#1 - cross_linking_e	xample						0 ₀ ,	Co 101	1010
Monitors	🕂 🗙 🆗	0x9000008 : 0x9000008	<hex< th=""><th>Integer></th><th>×</th><th>New</th><th>Renderi</th><th>ngs</th><th></th><th></th></hex<>	Integer>	×	New	Renderi	ngs		
Ox9000008		Address	0	1	2	3	4	5	6	7
		00000008FFFFF8	00	00	00	00	00	00	00	00
		0000000090000000	55	00	00	00	78	56	34	12
		0000000090000008	AA	00	44	33	22	11	06	00
		000000090000010	00	00	00	00	33	33	33	33
		000000090000018	33	33	ØB	40	9A	99	F9	40

Figure 33: SmartCode, Debug Perspective memory watch







The output of check_struct_member():

```
s_1.c_1 value is: 0x55
s_1.i_1 value is: 0x12345678
sizeof(s_1) is : 8
s_2.c_1 value is: 0xaa
s_2.i_1 value is: 0x61122
sizeof(s_2) is from SmartCode view : 0x8
sizeof(s_2) is from TriCore view : 0x6
```

- SmartCode Results -> `s_1':
 - SmartCode is word aligned. Therefore, the char 0x55 takes 4 bytes of memory. (1 byte for the 0x55 and 3 gap bytes).
- The int takes an additional 4 bytes for the 78 56 34 12.
- The size of operator yields a correct response of 8.

0x90000000 : 0x90000000	<hex i<="" th=""><th>nteger></th><th>×</th><th>New</th><th>Renderi</th><th>ngs</th><th></th><th></th></hex>	nteger>	×	New	Renderi	ngs		
Address	0	1	2	3	4	5	6	7
000000090000000	55	00	00	00	78	56	34	12

- SmartCode Results -> 's 2':
 - Per the TriCore build, the struct s_2 is half-word aligned. (memory is saved by reducing the number of alignment gap bytes).
 - The char c_1 starts on a word address. Since it is half-word aligned,

the character is followed with a 1-byte gap. The displayed OxAA is still correct.

0x9000008 : 0x9000008	<hex i<="" th=""><th>nteger></th><th>×</th><th>New</th><th>Renderi</th><th>ngs</th><th></th><th></th></hex>	nteger>	×	New	Renderi	ngs		
Address	0	1	2	3	4	5	6	7
00000009000008	AA	00	44	33	22	11	06	00

- For the TriCore *int*, the expected value is 44 33 22 11. However, the value printed is 00 06 11 22.
 - The int member can start on a half-word aligned address since this EABI violation has been enabled in the TriCore project.
 - The 11 22 are shown, but the 33 44 are missing. Why? SmartCode is accessing data not part of the TriCore struct.
- \cdot Based on how the project was built, the TriCore struct is 6-bytes.
- 2-bytes for the char (char + 1 gap) and 4-bytes for the int.
- SmartCode is expecting 8-bytes, so it loads byte7 and byte8 as shown in the purple box. SmartCode includes the 44 and 33 as part of the gap as shown in the green box. Thus, 00 06 11 22

0x9000008 : 0x9000008	<hex i<="" th=""><th>nteger></th><th>X</th><th>🐈 New I</th><th>Renderi</th><th>ngs</th><th></th><th></th></hex>	nteger>	X	🐈 New I	Renderi	ngs		
Address	0	1	2	3	4	5	6	7
0000000090000008	AA	00	44	33	22	11	06	00

· The TriCore sizeof operator reports the length of 6-bytes.







USE COMPLIANT VERSION

1. The SmartCode project will be re-built with `file_2_compliant_settings.o', for reference, the following command line innovation was used:

cctc file_2.c -t --core=tc1.6.2 -t -c -g -v --eabi=H --fp-model=F --global-type-checking -o file_2_compliant_settings.o

--eabi = H -> Removes half-word align option --fp-model = F -> Removes treat double as a float.

Please note in other use cases --EABI=W is needed to prevent word alignment of structs larger or equal than 64 bit to be SmartCode compliant. The --EABI=B is needed when 0-size bit fields are used and the struct in question is accessed by both TriCore v6.3r1 and SmartCode generated modules.

2. Delete `file_2_runtime_problems.o' and replace with `file_2_compliant_settings.o'



Figure 34: SmartCode, Project Workspace view with new TriCore ,.o' file





3. Build the new SmartCode project and launch the debugger



Figure 35: SmartCode, Debug Perspective with compliant version of Cross-linked program

Zooming in on the disassembly view, notice the load and store double with the TriCore defined variable: dvar_1.



Figure 36: SmartCode, Debug Perspective with compliant version of Cross-linked program, Disassembly view

The following excerpt from the .map file shows the memory location for the key symbols

dvar_1	0x9000001c	mpe:vtc:abs18
dvar_2	0x90000014	
s_1	0x90000000	
s_2	0x90000008	
sizeof value s 2	0x90000010	







The variable table shows the correct type and value for both dvar_1 and dvar_2.

×)= Variables 🗙 🏾 🍨 Br	eakpoints 🙀 Expressions	
Name	Туре	Value
(x)= 'file_1.c'::dvar_2	double	3.4
> 🏉 'file_1.c'::s_1	struct	{ c_1 = 85 'U'; i_1 = 305419896; }
(x)= 'file_2.c'::dvar_1	double	7.8
> 🍃 'file_2.c'::s_2	struct	{ c_1 = 170 'a'; i_1 = 287454020; }
(x)= 'file_2.c'::sizeof_valu	ue unsigned int	8

Figure 37: SmartCode, Debug Perspective variable view

The printf() shows the expected results. Please note that the pointer to a float value will show 0 since the value isn't a float value anymore but a double value. The variable table shows the correct type and value for both dvar_1 and dvar_2.

🚺 Memory 🗔 FSS #1 - cross_linking_example 🗙	
The expected double value of dvar_1 is 7.8. The value read is	: 7.800000e+00
Printing the float value of dvar_1 instead of the double value	: 0.000000
The expected value of dvar_2 is 3.4. The value read is	: 3.400000e+00
s_1.c_1 value is: 0x55	
s_1.i_1 value is: 0x12345678	
<pre>sizeof(s_1) is : 8</pre>	
s_2.c_1 value is: 0xaa	
s_2.i_1 value is: 0x11223344	
<pre>sizeof(s_2) is from SmartCode view : 0x8</pre>	
sizeof(s_2) is from TriCore view : 0x8	



TASKING_®

The TriCore variables dvar_1 and s_2 are located at 0x9000 001c and 0x9000 0008 respectively.

Monitors	🕂 🗙 🖗	0x90000008 : 0x90000008 <hex integer=""> 🗙 🜵 New Renderings</hex>								
♦ 0x9000008		Address	0	1	2	3	4	5	6	7
		000000009000008	AA	00	00	00	44	33	22	11
		0000000090000010	08	00	00	00	33	33	33	33
		000000090000018	33	33	ØB	40	33	33	33	33
		000000090000020	33	33	1F	40	00	00	00	00
		000000090000028	00	00	00	00	00	00	00	00
		000000090000030	00	00	00	00	00	00	00	00
		000000090000038	00	00	00	00	00	00	00	00

Figure 38: SmartCode, Debug Perspective memory view

- The red box shows the s_2.c_1 with correct size and number of gap bytes
- The violet box shows the correct 4 byte int
- \bullet The orange box shows the correct 8 bytes double 40 OB 33 33 33 33 33 33 33

SUMMARY

In summary, the TASKING TriCore toolset offers many competitive advantages including best-in-class code optimization performance based on their proprietary in-house Viper technology. Additional advantages include advanced multicore support, an integrated debugger, a highly configurable linker with versatile script language for optimal memory control, integrated MISRA C and CERT C static analysis tools, and integration into the popular Eclipse[™] platform (IDE).

One advantage that is frequently over-looked is the ability for newer TriCore toolset versions to cross-link legacy object files built with older versions of the toolset. This is a powerful feature which allows the user to easily integrate older trusted software components without the hassles of porting or re-validation.

When semiconductor vendors release new microcontroller's (new family or new variant of an existing family) there is the potential for differences between toolset versions with respect to memory layout. It is imperative that TASKING provide a set of cross-link guidelines to ensure that these differences between toolset versions do not result in erroneous program behavior.

With the introduction of SmartCode, the '--eabi' option was removed, to guarantee that SmartCode generated code is always EABI compliant with the exception of known EABI violations, reported on the TASKING issues portal.

When cross-linking SmartCode with previous versions of the TriCore toolset, users must take special care to ensure that their legacy object files were not built with 'treat double as a float', half-word alignment, base type alignment for 0-size bit field or word-struct-align toolset options. If these options were used, please re-compile the legacy software with the compliant toolset options if data will be exchanged between the TriCore and SmartCode objects.