Application Report
SLAA140A–February 2002–Revised August 2018

Mixing C and Assembler With MSP430™ MCUs

Stefan Schauer *MSP430*

ABSTRACT

This application report describes how C and assembler code can be used together in an application that uses an MSP430™ microcontroller (MCU). The combination of C and assembler benefits the designer, because it can provide the power of a high-level language and the speed, efficiency, and low-level control of assembler.

Source code for the examples that are described in this application report can be downloaded from www.ti.com/lit/zip/slaa140.

Contents

Trademarks

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Definition of the C-Compiler for Passing Variables Between Functions

1.1 Stack Frames and Parameter Passing

Each function call creates a stack frame (see Figure 1).

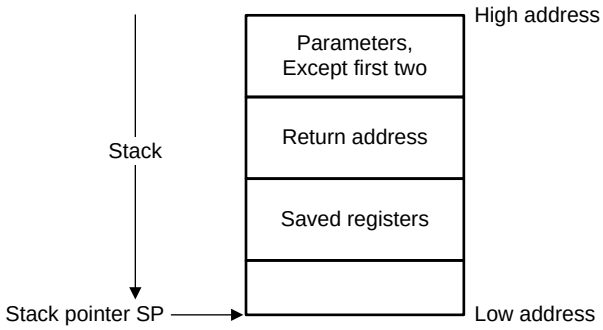

Figure 1. Parameter Passing From C

The parameters of a called function are passed to an assembler routine in a right-to-left order. The two leftmost parameters are passed in registers unless they are defined as a struct or union type, in which case they are also passed on the stack. The remaining parameters are always passed on the stack. See the following example of a call:

```
f(w,x,y,z)
```

1.2 Calling Convention – Register Use With the IAR C-Compiler

The compiler uses two groups of processor registers.

- The scratch registers R12 to R15 are used for parameter passing and are not normally preserved across the call. If R8 to R11 are used to hold a 64-bit parameter, these registers are also not preserved.
- The other general-purpose registers, R4 to R11, are mainly used for register variables and temporary results and must be preserved across a call. Within C this is handled automatically.

NOTE: The *–ur45* or *--lock_r4* and *--lock_r5* option prevents the compiler from using registers R4 and R5.

Table 1. Location of Passed Parameters

Argument	8-Bit or 16-Bit Type	32-Bit Type	64-Bit Type	Struct or Union
4th (z)	R15	On the stack	On the stack	On the stack
3th (y)	R14	On the stack	On the stack	On the stack
2nd (x)	R13	R15:R14	R11:R10:R8:R8	On the stack
1st (w)	R12	R12:R13	R15:R14:R13:R12	On the stack
Result	R12	R12:R13	R15:R14:R13:R12	On the stack

NOTE: IAR also support an older calling convention (version 1). For details, refer to the IAR C Compiler documentation

1.3 Calling Convention – Register Use With the CCS C-Compiler

The compiler uses two groups of processor registers.

- The scratch registers R11 to R15 are used for parameter passing and are not normally preserved across the call.
- The other general-purpose registers, R4 to R10, are mainly used for register variables and temporary results and must be preserved across a call. Within C, this is handled automatically.

NOTE: The *--global_register=r4* and *--global_register=r5* options prevent the compiler from using registers R4 and R5, respectively.

Table 2. Location of Passed Parameters

Argument	8-Bit or 16-Bit Type	32-Bit Type	64-Bit Type	Struct or Union
4th (z)	R15	On the stack	On the stack	Depends on size
3th (y)	R14	On the stack	On the stack	Depends on size
2nd (x)	R13	R15:R14	On the stack	Depends on size
1st (w)	R12	R12:R13	R15:R14:R13:R12	Depends on size
Result	R12	R12:R13	R15:R14:R13:R12	Depends on size

Also see the *Function Structure and Calling Conventions* section of MSP430 Optimizing C/C++ Compiler.

1.4 Interrupt Functions

Interrupt functions written in C automatically preserve the scratch registers, SR (status register), and the following registers:

IAR: R4 to R11

CCS: R4 to R10

The status register is saved as part of the interrupt calling process. Any registers used by the routine are then saved using `push Rxx` instructions. On exit, these registers are recovered using `pop Rxx` instructions, and the RETI instruction is used to reload the status register and return from the interrupt.

Functions written in assembler must explicitly manage these saves and restores.

2 Requirements of Assembler Routines to Support Calls From C

An assembler routine that is to be called from C must:

- Conform to the calling convention described above.
- Have a PUBLIC entry-point label.
- Be declared as external before any call, to allow type checking and optional promotion of parameters, as in `extern int foo()` or `extern int foo(int i, int j)`.

2.1 Local Storage Allocation

If the routine needs local storage, the routine allocates the storage in the following ways:

- On the hardware stack
- In static workspace (if the routine is not required to be simultaneously reusable (re-entrant))

IAR: Functions can always use R12 to R15 without saving them and R6 to R11 if they are pushed before use. R4 and R5 must not be used for ROM monitor compatible code. If the C code is compiled with the *-ur45* option, but the application is not to run in the ROM monitor, then it is possible to use R4 and R5 in the assembler routine without saving them, because the C code never uses them.

CCS: Functions can always use R12 to R15 without saving them and R6 to R11 if they are pushed before use. R4 and R5 must not be used for ROM monitor compatible code.

2.2 Interrupt Functions

The calling convention cannot be used for interrupt functions, because the interrupt may occur during the calling of a foreground function. The requirements for an interrupt function routine are different from those of a normal function routine as follows:

- The routine must preserve all used registers, including scratch registers R12 to R15.
- The routine must exit using RETI.
- The routine must treat all flags (Carry, Negative, Zero, and Overflow) of the Status register as undefined.

2.3 Define Interrupt Vectors

As an alternative to defining a C interrupt function in assembly language as described above, the user can assemble an interrupt routine and install it directly in the interrupt vector.

The interrupt vectors are located in the INTVEC segment.

3 Combining C and Assembler Functions

3.1 General Basics

The mechanics to combine C and assembler functions are fairly straightforward. Basically C code that is in .c files imports labels exported by the assembler files using the extern keyword. Assembler codes within .s43 or .asm files export labels to the C code using the PUBLIC keyword. Assembler code import labels exported by C code using the EXTERN keyword. No keyword is required to export C labels to assembler code.

When the .c and .s43 or .asm files are written, they must be added to the project before the project is built. See the .c, .s43, .asm, and project (.ewp or .project) example files included with this application report. See the CCS and IAR documentation for a more complete description of this process.

3.2 Call Assembler Functions Without Parameters to Pass

If no parameters are passed between the C code and the assembler function, a simple call instruction can be used. See the Example 1 code.

3.3 Call Assembler Functions With Parameters to Pass

To pass parameters from C to the assembler function, the parameters must be located as described in the Stack Frames and Parameter Passing section.

The Example 2 code shows how parameters are passed between the C main program and an assembler function.

3.4 Define an Interrupt Service Routine in Assembler

To optimize the interrupt service routine for speed, use assembler. See the Interrupt Functions section for the requirements of an assembler interrupt service routine.

The Example 4 code shows a watchdog interrupt service routine written in assembler that is used from a C program.

3.5 Define an Interrupt Service Routine for Special Interrupts

Some modules (for example, Timer_A, Timer_B, and ADC12) use a special combination of hardware and software to detect the source of an individual interrupt. For additional information, see the family user's guide:

- MSP430x1xx Family User's Guide
- MSP430x4xx Family User's Guide
- MSP430F5xx and MSP430F6xx Family User's Guide
- MSP430FR4xx and MSP430FR2xx Family User's Guide
- MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx Family User's Guide

The interrupt service routine for these modules sometimes needs to be written in assembler. The Example 5 code shows how this can be done.

3.6 Call C Functions from Assembler

An application can also call a C function from an assembler routine. The same restrictions described in Section 1 apply.

In the Example 3 code, the assembler program calls the C function `rand()`.

3.7 Exit Low-Power Mode While in the Interrupt Service Routine

During an interrupt service routine, the status register and the return address are stored on the stack (see the family user's guide for more detailed information). To set the CPU to active mode after returning from the interrupt service routine, modify the value of the status register on the stack (specifically, clear the bits that specify the low-power mode). From C, it is not possible to directly access the stack pointer, but the intrinsic function `__bic_SR_register_on_exit(bits)` can modify the status register that is saved on the stack.

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from February 28, 2002 to August 7, 2018	Page
• Editorial and formatting updates throughout document; clarified features that apply to CCS or IAR	1
• Removed former Section 4 *Building Libraries*, Section 5 *Using Watch Windows With Assembler Variables*, and Appendix of code listings	5

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265