PUMPKIN™
REAL-TIME SOFTWARE
750 Naples Street • San Francisco, CA 94112 • (415) 584-6360 • http://www.pumpkininc.com

RM-IAR430
Reference Manual

Salvo Compiler Reference Manual – IAR MSP430 C

Salvo™
The RTOS that runs in tiny places.™

created by Andrew E. Kalman on Mar 21, 2003 updated on Apr 29, 2008

Introduction

This manual is intended for Salvo users who are targeting TI's MSP430 ultra-low-power single-chip microcontroller with IAR's (http://www.iar.com/) Embedded Workbench for MSP430.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with IAR's MSP430 C compiler:

- *Salvo User Manual*

Example Projects

Example Salvo projects for use with IAR's MSP430 C compiler and the Embedded Workbench IDE can be found in the:

```
\Pumpkin\Salvo\Example\MSP430\MSP430x1xx
\Pumpkin\Salvo\Example\MSP430\MSP430x4xx
```

directories of every Salvo for TI's MSP430 distribution.

Tip These example projects can be easily modified for any MSP430 device.

Features

Table 1 illustrates important features of Salvo's port to IAR's MSP430 C compiler.

General	
Abbreviated as	`IAR430`
Available distributions	Salvo Lite, LE & Pro for TI's MSP430
Supported targets	all MSP430 and MSP430X devices
Header file(s)	`salvoportiar430.h`
Other target-specific file(s)	`salvoportiar430.s43`
salvocfg.h	
Compiler auto-detected?	yes[1]
Include target-specific header file in `salvocfg.h`?	yes, use `#include <msp430.h>`[2]
Libraries	
Located in	`Lib\IAR430-v1` (for v1.x compilers) `Lib\IAR430-v2` (for v2.x compilers) `Lib\IAR430-v3` (for v3.x compilers) `Lib\IAR430-v4` (for v4.x compilers)
Context Switching	
Method	function-based via `OSDispatch()` & `OSCtxSw()`
Labels required?	no
Size of auto variables and function parameters in tasks	total size must not exceed 255 8-bit bytes
Interrupts	
Interrupt latency in context switcher	0 cycles
Interrupts in critical sections controlled via	user hooks
Default behavior in critical sections	see example user hooks
Debugging	
Source-level debugging with Pro library builds?	yes
Compiler	
Bitfield packing support?	no
printf() / %p support?	yes / yes
va_arg() support?	yes

Table 1: Features of Salvo port to IAR's MSP430 C compiler

Libraries

Nomenclature

The Salvo libraries for IAR's MSP430 C compiler C compiler follow the naming convention shown in Figure 1.

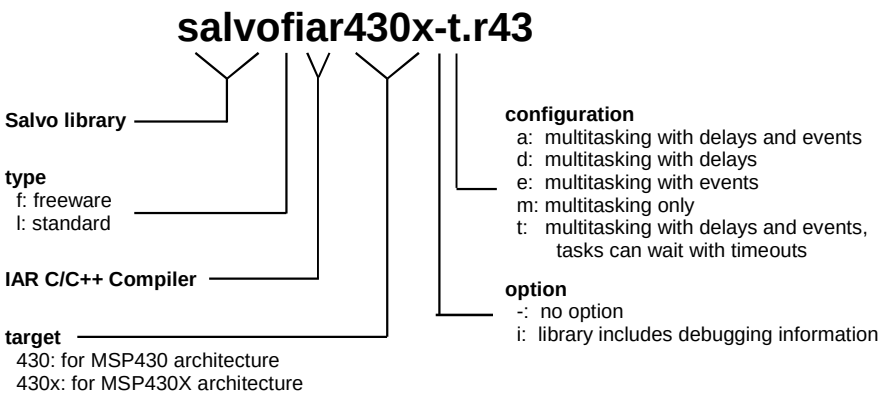

Figure 1: Salvo library nomenclature – IAR's MSP430 C compiler

Note Each successive version of IAR's MSP430 C/C++ compiler *uses different library formats.* Therefore independent sets of Salvo libraries are available for each compiler version – see *libraries* in Table 1, above.

Type

Salvo Lite distributions contain *freeware* libraries. All other Salvo distributions contain *standard* libraries. See the *Libraries* chapter of the *Salvo User Manual* for more information on library types.

Target

When targeting the MSP430X (e.g. MSP430FG4619) and wishing to use MSP430X extensions (i.e. 20-bit address space, etc.), 430x Salvo libraries must be selected. 430 Salvo libraries should be used on the MSP430 (e.g. MSP430F1611).

Option

Salvo Pro users can select between two sets of libraries – standard libraries, and standard libraries incorporating source-level debugging information.[3] The latter have been built with the appropriate command-line options. This adds source-level debugging information to the libraries, making them ideal for source-level debugging and stepping in the C-SPY debugger. To use these libraries, simply select one that includes the debugging information (e.g. `salvoliar430it.r43`) instead of one without (e.g. `salvoliar430-t.r43`) in your Embedded Workbench project.

Configuration

Different library configurations are provided for different Salvo distributions and to enable the user to minimize the Salvo kernel's footprint. See the *Libraries* chapter of the *Salvo User Manual* for more information on library configurations.

Build Settings

Salvo's libraries for IAR's MSP430 C compiler are built using the default settings outlined in the *Libraries* chapter of the *Salvo User Manual*. Target-specific settings and overrides are listed in Table 2.

Target-specific Settings	
Delay sizes	8 bits
Idling hook	dummy, can be overridden
Interrupt hook	disables then restores GIE bit, can be overridden
Watchdog hook	clears WDT without other changes, can be overridden
System tick counter	available, 32 bits
Task priorities	enabled

Table 2: Build settings and overrides for Salvo libraries for IAR's MSP430 C compiler

Note Salvo Lite libraries have precompiled limits for the number of supported tasks, events, etc. Salvo LE and Pro libraries have no such limits. See the *Libraries* chapter of the *Salvo User Manual* for more information.

Available Libraries

Salvo Lite for TI's MSP430 contains a single freeware library. Salvo LE for TI's MSP430 adds standard libraries in all configurations. Salvo Pro for TI's MSP430 adds standard libraries in all configurations with debugging information included. Each Salvo for TI's MSP430 distribution contains the Salvo libraries of the lesser distributions beneath it.

Target-Specific Salvo Source Files

The source file `salvoportiar430.s43` is needed for Salvo Pro source-code builds. It automatically detects whether the target is part of the MSP430 or MSP430X architecture.

salvocfg.h Examples

Below are examples of `salvocfg.h` project configuration files for various different Salvo distributions and the MSP430F1612.

Salvo Lite Library Build

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSF
#define OSLIBRARY_CONFIG        OST
#define OSTASKS                 2
#define OSEVENTS                4
#define OSEVENT_FLAGS           0
#define OSMESSAGE_QUEUES        1
```

Listing 1: Example salvocfg.h for library build using salvofiar430-t.r43

Salvo LE & Pro Library Build

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSL
#define OSLIBRARY_CONFIG        OST
#define OSTASKS                 7
#define OSEVENTS                13
#define OSEVENT_FLAGS           3
#define OSMESSAGE_QUEUES        2
```

Listing 2: Example salvocfg.h for library build using salvoliar430-t.r43 or salvoliar430it.r43

Salvo Pro Source-Code Build

```
#define OSENABLE_IDLING_HOOK    TRUE
#define OSENABLE_SEMAPHORES     TRUE
#define OSTASKS                 9
#define OSEVENTS                17
#define OSEVENT_FLAGS           2
#define OSMESSAGE_QUEUES        4
```

Listing 3: Example salvocfg.h for source-code build

Performance

Interrupt Latencies

Since Salvo's context switcher for IAR's MSP430 C compiler does not need to control interrupts, Salvo applications can easily be created with zero total interrupt latency for interrupts of interest.

In a properly-configured application, only those interrupts that call Salvo services will (by necessity) experience interrupt latency from Salvo's operations. Users must ensure that these interrupt sources are disabled (and re-enabled) via the user interrupt hooks.

Disabling and re-enabling interrupts globally in the user interrupt hooks (i.e., the default user interrupt hook behavior) is of course permitted, but will result in non-zero interrupt latencies for all interrupt sources, even those that do not call Salvo services. See the target-specific source files of this distribution for examples.

Memory Usage

Example Application[4]	Program Memory Usage[5]	Data Memory Usage[6]
`\MSP430x1xx\…\tut5lite`	1486	61
`\MSP430x1xx\…\tut5le`	1448	61
`\MSP430x1xx\…\tut5pro`	1658	61

Table 3: ROM and RAM requirements for Salvo applications built with IAR's MSP430 C compiler

User Hooks

Overriding Default Hooks

In library builds, users can define new hook functions in their projects and the linker will choose the user function(s) over the default function(s) contained in the Salvo library.

In source-code builds, users can remove the default hook file(s) from the project and substitute their own hook functions.

Idling

The default idling hook in `salvohook_idle.c` is a dummy function, as shown below.

```
void OSIdlingHook ( void )
{
  ;
}
```

Listing 4: Default Salvo idling hook for IAR's MSP430 C compiler

Users can replace it (e.g. with a directive to put the FM430 to sleep) by building their own version with their application.

Interrupt

The default interrupt hooks in `salvohook_interrupt_IAR430_GIE.c` are shown below.[7]

```
static istate_t s;

void OSDisableHook(void)
{
  istate_t t = __get_interrupt_state();
  __disable_interrupt();
  s = t;
}


void OSEnableHook(void)
{
  __set_interrupt_state(s);
}
```

Listing 5: Default Salvo interrupt hooks for IAR's MSP430 C compiler

These functions clear the `GIE` bit (i.e. disable global interrupts) across Salvo's critical section, and restore the GIE bit to its pre-critical-section value thereafter. Interrupts are *not* re-enabled inside of ISRs with these functions, thereby avoiding unnecessary or unwanted interrupt nesting. Therefore these default interrupt hooks are suitable for *all* applications.

Note By saving the interrupt state to an auto variable, `OSDisableHook()` avoids the potential for the global variable `s` being overwritten should an interrupt that calls a Salvo service occur between when the interrupt status is read and when it is saved to `s`.[8]

For greater runtime performance, users can replace these functions with targeted interrupt-control functions by building their own version with their application. For example, if the Salvo service `OSTimer()` is the *only* Salvo service called from the foreground (i.e. interrupt) level, *and* it's called (only) via the TimerA0 ISR, then the following functions to disable and re-enable TimerA0's interrupt enable bit are all that is required, e.g.:

```
void OSDisableHook ( void )
{
  TACCTL0 &= ~CCIE;
}

void OSEnableHook ( void )
{
  TACCTL0 |= CCIE;
}
```

Listing 6: Example of user interrupt hooks where only TimerA0 ISR calls Salvo services

With the hooks shown in Listing 6, only the TimerA0 interrupt is disabled during Salvo's critical sections. All other interrupts sources (and the `GIE` bit) are untouched. This allows other interrupts that do not call Salvo services to proceed with *zero interrupt latency* due to Salvo.

Warning Not disabling all source of interrupts that call Salvo services during critical sections will cause the Salvo application to fail.

Watchdog

The default watchdog hook in `salvohook_wdt_IAR430_CLRWDT.c` is shown below.[9]

```
void OSClrWDTHook ( void )
{
  WDTCTL = (WDTCTL & 0x00FF) | WDTPW | WDTCNTCL;
}
```

Listing 7: Default Salvo watchdog hook for IAR's MSP430 C compiler

Users can replace it (e.g. with a dummy function – this would stop Salvo from clearing the watchdog timer and allow the user to clear it elsewhere) by building their own version with their application.

Compiler Issues

Runtime Models and Compatible Libraries

The runtime models used by Embedded Workbench for MSP430 have evolved over the years. When building an application with Salvo libraries, it's necessary to link to the libraries compatible with the version of Embedded Workbench for MSP430 that you

are using. Table 4 lists the locations of Salvo libraries as a function of the Embedded Workbench for MSP430 version.

Embedded Workbench for MSP430 Version	IAR C Compiler Version	IAR Runtime Model	Salvo Library Location
2.0	v1.x	1	Lib\IAR430-v1
3.0	v2.x	1	Lib\IAR430-v2
4.0	v3.x	2	Lib\IAR430-v3
5.0	v4.x	3	Lib\IAR430-v4

Table 4: Compiler versions, runtime models and Salvo library locations for IAR's MSP430 C compiler

Incompatible Optimizations

Code Motion

The code motion optimization (controlled via a checkbox in the Embedded Workbench IDE and/or disabled via the `-no_code_motion` command-line directive) in compiler version 2 and later may cause problems in Salvo applications when applied to the scheduler (contained in the `salvosched.c` module) and to Salvo tasks. Therefore code motion is disabled in Salvo libraries for IAR's MSP430 C compiler, and should be explicitly disabled by the user within Embedded Workbench for MSP430 projects.

Note Code motion may or may not be enabled with the optimization level (none, low, medium or high) and approach (balanced, speed or size) chosen by the user. *Be sure to explicitly disable code motion in your Embedded Workbench project.*

---

1 This is done automatically through the `__IAR_SYSTEMS_ICC__` and `__TID__` symbols defined by the compiler.

2 `msp430.h` for compiler version 3 and later. For earlier versions, use appropriate `ioxxx.h` file.

3 The Salvo libraries provided with Salvo Lite and LE do not contain C-SPY-compatible debugging information because this requires the inclusion of source file listings.

4 Salvo 4.1.0-rc0 with v3.42A compiler.

5 In bytes. Salvo code only.

6 In bytes. Salvo objects only.

7 This hook is valid for all MSP430 and MSP430X targets.

8 Thanks to Salvo user Dave Hohl for suggesting this method over one written in assembly.

9 This hook is valid for MSP430 and MSP430X targets because the watchdog control register is the same for all targets.