PUMPKIN™
REAL-TIME SOFTWARE

RM-IARARM
Reference Manual

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

Salvo Compiler Reference Manual – IAR Embedded Workbench for ARM

Salvo™
The RTOS that runs in tiny places.™

created by Andrew E. Kalman on Sep 10, 2007 updated on Jul 6, 2011

Introduction

This manual is intended for Salvo users who are targeting ARM ARM7TDMI single-chip microcontrollers with IAR's (http://www.iar.com/) Embedded Workbench for ARM (EWARM) development suite.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with IAR Embedded Workbench for ARM:

- *Salvo User Manual*

Example Projects

Example Salvo projects for use with IAR Embedded Workbench for ARM can be found in the:

```
\Pumpkin\Salvo\Example\ARM
```

directories of every Salvo for ARM® distribution.

Features

Table 1 illustrates important features of Salvo's port to IAR Embedded Workbench for ARM.

General	
Abbreviated as	IARARM
Available distributions	Salvo Lite, LE & Pro for ARM®
Supported targets	all ARM7TDMI- and Thumb-based devices
Header file(s)	`salvoportiararm.h`
Other target-specific file(s)	`salvoportiararm.s79`, `salvohook_interrupt_IRQ.c`
salvocfg.h	
Compiler auto-detected?	yes[1]
Include target-specific header file in `salvocfg.h`?	recommended
Libraries	
Located in	`Lib\IARARM-v5` (for EWARM v5) `Lib\IARARM-v6` (for EWARM v6)
Behavior of user hooks in libraries	do nothing (dummy functions)
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 65,535 8-bit bytes
Interrupts	
Interrupt latency in context switcher	0 cycles
Interrupts in critical sections controlled via	`OSDisableHook()`, `OSEnableHook()`, `OSRestoreHook()`, `OSSaveHook()`
Interrupt status preserved in critical sections?	optional, via appropriate user functions
Method used in critical sections	see example user functions
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 Embedded Workbench for ARM

Libraries

Nomenclature

The Salvo libraries for IAR Embedded Workbench for ARM follow the naming convention shown in Figure 1.

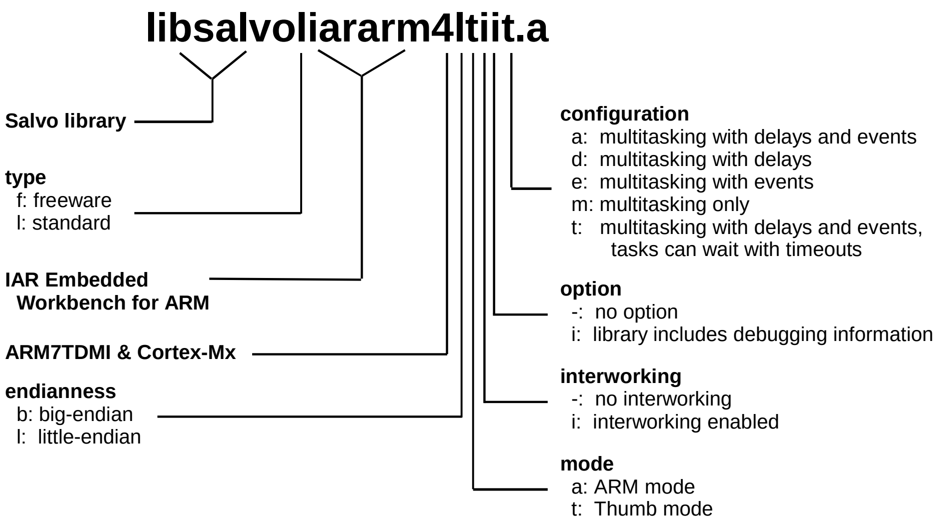

Figure 1: Salvo library nomenclature – IAR Embedded Workbench for ARM

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

Since the CPU instruction set is common to all target architectures based on the ARM7TDMI core, all ARM7TDMI targets use the same Salvo libraries.

Note There is no target-specific dependence apart from whether the target operates in little- or big-endian mode.

Option

Salvo Pro users can select between two sets of libraries – standard libraries, and standard libraries incorporating source-level debugging information. 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 C-SPY. To use these libraries, simply select one that includes the debugging information (e.g. `libsalvoliararm4ltiit.a`) instead of one without (e.g. `libsalvoliararm4lti-t.a`) 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 Embedded Workbench for ARM 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.

Compiled Limits	
Max. number of tasks	4
Max. number of events	8
Max. number of event flags	1
Max. number of message queues	1
Target-specific Settings	
Delay sizes	8 bits
Idling hook	enabled
Interrupt-enable bits during critical sections	controlled via user functions
System tick counter	available, 32 bits
Task priorities	enabled
Watchdog timer	controlled via user functions

Table 2: Build settings and overrides for Salvo libraries for IAR Embedded Workbench for ARM

Note The compiled limits for tasks, events, etc. in Salvo libraries can be overridden to be less (all Salvo distributions) or more (all Salvo distributions except Salvo Lite) than the library default. See the *Libraries* chapter of the *Salvo User Manual* for more information.

Available Libraries

Salvo Lite for ARM contains four freeware libraries in a single configuration. Salvo LE for ARM adds standard libraries in multiple configurations. Salvo Pro for ARM adds standard libraries in multiple configurations with debugging information included.

Each Salvo for ARM distribution contains the Salvo libraries of the lesser distributions beneath it. Additionally, Salvo Pro distributions contain makefiles for all possible library configurations.

Target-Specific Salvo Source Files

salvoportiararm.s79

The source file `salvoportiararm.s79` is required for Salvo Pro source-code builds.

ARM vs. Thumb Mode

By default, `salvoportiararm.s79` is assembled for Thumb mode. To assemble it for ARM mode, ensure that the symbol `MAKE_FOR_ARM` is defined during assembly, e.g. from the command line or via Embedded Workbench.

Big-Endian vs. Little-Endian

`salvoportiararm.s79` can be assembled for little-endian (assembler default) and big-endian (assembler command-line argument: `-e`) targets without any modifications.

Interworking

`salvoportiararm.s79` can be assembled independent of any interworking settings.

salvocfg.h Examples

Below are examples of `salvocfg.h` project configuration files for various different Salvo distributions and the ARM7TDMI core.

Salvo Lite Library Build

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSF
#define OSLIBRARY_CONFIG        OST
#define OSTASKS                 3
#define OSEVENTS                4
#define OSEVENT_FLAGS           0
#define OSMESSAGE_QUEUES        1
```

Listing 1: Example salvocfg.h for library build using libsalvofiararm4lti-t.a

Salvo LE & Pro Library Build

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSL
#define OSLIBRARY_CONFIG        OSA
#define OSTASKS                 7
#define OSEVENTS                11
#define OSEVENT_FLAGS           0
#define OSMESSAGE_QUEUES        4
```

Listing 2: Example salvocfg.h for library build using libsalvoliararm4lti-a.a or libsalvoliararm4ltiia.a

Salvo Pro Source-Code Build

```
#define OSEVENTS                   9
#define OSEVENT_FLAGS              1
#define OSMESSAGE_QUEUES           2
#define OSTASKS                    17

#define OSENABLE_BINARY_SEMAPHORES  TRUE
#define OSENABLE_IDLING_HOOK        TRUE
#define OSENABLE_TIMEOUTS           TRUE
#define OSBYTES_OF_DELAYS          1
#define OSBYTES_OF_TICKS           4
```

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

Performance

Interrupt Latencies

Since Salvo's context switcher for IAR Embedded Workbench for ARM 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 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

Examples of the total memory usage of actual Salvo-based applications are listed below.

Example Application[2]	Program Memory Usage[3]	Data Memory Usage[4]
`tut5lite (for ARM7TDMI core)`	1844	147
`tut5le (for ARM7TDMI core)`	1800	147
`tut5pro (for ARM7TDMI core)`	2156	143

Table 3: Program and data memory requirements for Salvo applications built with IAR Embedded Workbench for ARM

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 Embedded Workbench for ARM

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_IRQ.c` are shown below.

```
__istate_t s;

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

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

Listing 5: Default Salvo interrupt hooks for for IAR Embedded Workbench for ARM

These functions disable interrupts across Salvo's critical sections, and restore interrupts to their pre-critical-section values 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.

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 a timer, then the following functions to disable and re-enable the timer's interrupt enable bit are all that is required, e.g.:

```
void OSDisableHook(void) {
  //disable timer interrupts
}

void OSEnableHook(void) {
  //enable timer interrupts
}
```

Listing 6: Example of user interrupt hooks where only a single timer ISR calls Salvo services

With the hooks shown in Listing 6, only the timer interrupt is disabled during Salvo's critical sections. All other interrupts sources 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.c` is shown below.

```
void OSClrWDTHook(void) {
  ;
}
```

Listing 7: Default Salvo watchdog hook for IAR Embedded Workbench for ARM

The default hook does nothing because clearing the watchdog timer is implementation-dependent. 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.

1 This is done automatically through the `OSIAR_ICC` and `__TID__` symbols defined by the compiler.
2 Salvo 4.0.0.
3 In bytes. Entire `CODE` section.
4 In bytes. Entire `DATA_Z` section. This represents all of Salvo's objects. Does not include RAM allocated to the heap or stack. Salvo applications typically require the same (small) stack size as simple, non-multitasking applications.