# RM-HTARCLITE

## Reference Manual

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

# *Salvo Compiler Reference Manual – HI-TECH ARClite-C*

The RTOS that runs in tiny places.™

*created by Andrew E. Kalman on Feb 23, 2004 updated on Feb 23, 2004*

## Introduction

This manual is intended for Salvo users who are targeting ARC International's (http://www.arc.com) ARClite microRISC synthesizable 8-bit RISC core general-purpose microprocessor with HI-TECH Software's (http://www.htsoft.com/) ARClite-C compiler.

**Note** The ARClite microRISC was originally developed and marketed by *VAutomation* as the *V8 µ-RISC 8-bit synthesizable core*. At the time that Salvo was ported to this processor, the HI-TECH compiler[1] was called *V8C*, and HI-TECH's IDE for the V8 was called *HPDV8*. Currently, the naming convention used within the Salvo for ARClite microRISC distribution contains references to ARClite and V8.

## Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with HI-TECH's ARClite-C C compiler:

*Salvo User Manual*
*Application Note AN-30*

## Example Projects

Example Salvo projects for use with HI-TECH's ARClite-C C compiler and the HI-TIDE HPDV8 IDE can be found in the:

```
\salvo\tut\tu1\sysl
\salvo\tut\tu2\sysl
\salvo\tut\tu3\sysl
\salvo\tut\tu4\sysl
\salvo\tut\tu5\sysl
\salvo\tut\tu6\sysl
```

directories of every Salvo for ARClite microRISC distribution.

## Features

Table 1 illustrates important features of Salvo's port to HI-TECH's ARClite-C C compiler.

| general | |
|---|---|
| available distributions | Salvo Lite, LE & Pro for ARClite microRISC |
| supported targets | ARClite microRISC core |
| header file(s) | `portv8c.h` |
| other target-specific file(s) | `portv8c.as` |
| project subdirectory name(s) | `SYSL` |
| **salvocfg.h** | |
| compiler auto-detected? | yes[2] |
| **libraries** | |
| `salvo\lib` subdirectory | `htv8c` |
| **context switching** | |
| method | function-based via `OSDispatch()` & `OSCtxSw()` |
| `_OSLabel()` required? | yes |
| size of auto variables and function parameters in tasks | total size must not exceed 255 8-bit bytes |
| **interrupts** | |
| controlled via | `I` bit (`PSR[3]`)[3] |
| disabled in | critical sections, `OSDisptach()` and `OSCtxSw()` |
| interrupt status preserved in critical sections? | no |
| method used | interrupts disabled on entry and enabled on exit of critical sections |
| nesting limit | no nesting permitted |
| alternate methods possible? | yes[4] |
| **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 HI-TECH's ARClite-C C Compiler**

# Libraries

## Nomenclature

The Salvo libraries for HI-TECH's ARClite-C C compiler follow the naming convention shown in Figure 1.

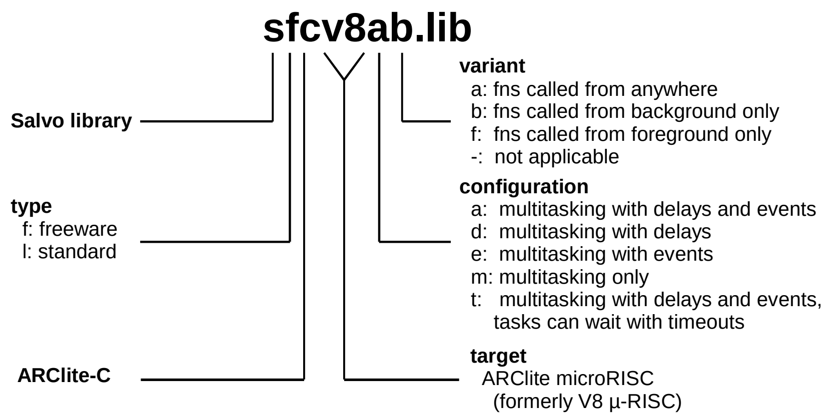


**Figure 1: Salvo Library Nomenclature – HI-TECH's ARClite-C C Compiler**

## 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

No target-specific identifiers are required.

## 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.

## Variant

When using HI-TECH's ARClite-C C compiler, the Salvo source code must be properly configured via the appropriate configuration parameters. The Salvo libraries for HI-TECH's ARClite-C C compiler are provided in different variants as shown in Table 2.

If your application does not call any Salvo services from within interrupts, use the *b* variant. If you wish to these services exclusively from within interrupts, use the *f* variant. If you wish to do this from both inside and outside of interrupts, use the *a* variant. In each case, you must call the services that you use from the correct place in your application, or either the linker will generate an error or your application will fail during runtime.

| variant code | description |
|---|---|
| `a / OSA:` | Applicable services can be called from *a*nywhere, i.e. from the foreground and the background, simultaneously. |
| `b / OSB:` | Applicable services may only be called from the *b*ackground (default). |
| `f / OSF:` | Applicable services may only be called from the *f*oreground. |

**Table 2: Variants for Salvo Libraries – HI-TECH's ARClite-C C Compiler**

See the `OSCALL_OSXYZ` configuration parameters for more information on calling Salvo services from interrupts.

See *Special Considerations*, below, for more information on using library variants.

## Build Settings

Salvo's libraries for HI-TECH's ARClite-C 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 3.

| compiled limits | |
|---|---|
| max. number of tasks | 3 |
| max. number of events | 5 |
| max. number of event flags | 1 |
| max. number of message queues | 1 |
| **target-specific settings** | |
| delay sizes | 8 bits |
| watchdog timer | not affected |
| system tick counter | available, 32 bits |

**Table 3: Build Settings and Overrides for Salvo Libraries for HI-TECH's ARClite-C C Compiler**

**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

There are 30 Salvo libraries for HI-TECH's ARClite-C C compiler. Each Salvo for ARClite microRISC distribution contains the Salvo libraries of the lesser distributions beneath it.

## salvocfg.h Examples

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

**Note** When overriding the default number of tasks, events, etc. in a Salvo library build, `OSTASKS` and `OSEVENTS` (respectively) *must also be defined* in the project's `salvocfg.h`. If left undefined, the default values (see Table 3) will be used.

### Salvo Lite Library Build

```
#define OSUSE_LIBRARY              TRUE
#define OSLIBRARY_TYPE             OSF
#define OSLIBRARY_CONFIG           OSA
#define OSLIBRARY_VARIANT          OSB
```

**Listing 1: Example salvocfg.h for Library Build Using sfcv8ab.lib**

### Salvo LE & Pro Library Build

```
#define OSUSE_LIBRARY              TRUE
#define OSLIBRARY_CONFIG           OST
#define OSLIBRARY_VARIANT          OSA
```

**Listing 2: Example salvocfg.h for Library Build Using slcv8ta.lib**

### Salvo Pro Source-Code Build

```
#define OSENABLE_IDLING_HOOK       TRUE
#define OSENABLE_SEMAPHORES        TRUE
#define OSEVENTS                   1
#define OSTASKS                    3
```

**Listing 3: Example salvocfg.h for Source-Code Build**

# Performance

## Memory Usage

| tutorial memory usage[5] | total ROM[6] | total RAM[7] |
|---|---|---|
| `tu1lite` | 383 | 31 |
| `tu2lite` | 538 | 32 |
| `tu3lite` | 579 | 34 |
| `tu4lite` | 1183 | 48 |
| `tu5lite` | 1828 | 72 |
| `tu6lite` | 1994 | 72 |
| `tu6pro` | 1819 | 68 |

**Table 4: ROM and RAM requirements for Salvo Applications built with HI-TECH's ARClite-C C Compiler**

# Special Considerations

## Stack Issues

For architectural reasons, HI-TECH's ARClite-C C compiler does not pass parameters on the stack. Nor does it allocate memory for auto (local) variables on the stack. Instead, it employs a *static overlay* model. This has advantages in speed and memory utilization, but it precludes recursion and has other impacts.

## Multiple Callgraph Issues

By default, it is expected that Salvo services will only be called from the background / main loop / task level. This is the default configuration for source-code builds. *b*-variant libraries allow service calls only from the background level. Should you wish to call certain services from the foreground / interrupt level, you will need to set `OSCALL_OSXYZ` configuration options for source-code builds or use a different library (see Table 2) for library builds.

From *Variant*, above, we find that the `f`-variant libraries allow you to call event-reading and –signaling services from the foreground. Similarly, the `a`-variant libraries allow you to call the applicable services from anywhere in your code.

## The interrupt_level Pragma

When using the a-variant libraries, each instance of an applicable service in use must be called from the foreground, i.e. from an interrupt. Also, ARClite-C's `interrupt_level` pragma must be set to 0 and placed immediately ahead of the application's interrupt routine, like this:

```
#pragma interrupt_level 0[8]
void interrupt IntVector( void )
{
  OSStartTask(TASK_P);
}
```

**Listing 4: Setting the HI-TECH ARClite-C interrupt_level Pragma for an ISR when Using a-variant Libraries**

ARClite-C requires this in order to manage the parameter overlay areas for functions located on multiple call graphs.

**Note** This pragma has no effect if there aren't any functions located on multiple call graphs. Therefore it's OK to add it to any application compiled with ARClite-C.

## Example: Foreground Signaling of One Event Type

In a library build, if you were to move a call to `OSSignalBinSem()` from a Salvo task (i.e. from the background) to an interrupt handler (i.e. to the foreground) without changing the library variant, you'd find that the application crashes from a stack overflow almost immediately. This is because the default interrupt control[9] in `OSSignalBinSem()` is incompatible with being placed inside an interrupt. To circumvent this, you must change `OSLIBRARY_VARIANT` to `OSF` and link an `f`-variant library (e.g. `sfcv8af.lib` — note the `f` for *foreground* in the variant field) in order to properly support event service calls in the foreground.

## Example: Foreground and Background Signaling of One Event Type

If we call `OSSignalBinSem()` from a task and from within an interrupt handler without addressing the callgraph issues, the compiler issues an error message:

```
Error[ ] file : function _OSSignalBinSem appears
in multiple call graphs: rooted at intlevel0 and
_main Exit status = 1
```

To resolve this, add the `interrupt_level 0` pragma to your interrupt handler (see Listing 4, above) and use the a-variant library after setting `OSLIBRARY_TYPE` to `OSA`.

## OSProtect() and OSUnprotect()

HI-TECH's ARClite-C C compiler requires that when a function is contained in multiple callgraphs, interrupts must be disabled "around" that function to prevent corruption of parameters and/or return values.[10] Therefore you must call `OSProtect()` immediately before and `OSUnprotect()` immediately after all background instances of every Salvo service that is called from both the background and foreground levels, e.g.:

```
void TaskN ( void )
{
  …
  OSProtect();
  OSSignalBinSem(SEM_P);
  OSUnprotect();
  …
}

#pragma interrupt_level 0
void interrupt IntVector( void )
{
  OSSignalBinSem(SEM_P);
}
```

**Tip** Wrapping `OSProtect()`, the affected Salvo service and `OSUnprotect()` within another function can make your code more legible. The wrapper may only be called from mainline code – i.e. it can only have a single callgraph. A wrapper function might look like this:

```
OSSignalBinSem_Wrapper(OStypeEcbP ecbP)
{
  OSProtect();
  OSSignalBinSem(ecbP);
  OSUnprotect();
}
```

and a wrapper macro might look like this:

```
#define OSSignalBinSem_Wrapper(ecbP) \
    do { OSProtect(); \
         OSSignalBinSem(ecbP); \
         OSUnprotect(); \
       } while (0)
```

### Example: Mixed Signaling of Multiple Event Types

The library variants affect all event services equally – that is, an `f`-variant library expects all applicable event services to be called from the foreground, i.e. from within interrupts. If you wish to call some services from the background, and others from the foreground, you'll have to use the `a`-variant library, as explained above.

A complication arises when you need an `a`-variant library for a particular event type, and you also are using additional event types. In this case, each instance of an applicable event service in use *must be called from the foreground*. If it's not called from the foreground, the compiler issue this error message:

```
Error[ ] file : function _OSSignalBinSem is not
called from specified interrupt level
Exit status = 1
```

However, it need not be called from the background. If you have the "opposite" situation, e.g. you are using an `a`-variant library for one type of event and you need to call an event service for a different event type only from the background, one solution is to place the required foreground call inside an interrupt handler, with a conditional that prevents it from ever happening, e.g.:

```
#pragma interrupt_level 0
void interrupt IntVector( void )
{
  /* real code is here …          */
  …
  /* dummy to satisfy call graph. */
  if ( 0 )
  {
    OSSignalBinSem(OSECBP(1));
  }
}
```

This creates a call graph acceptable to HI-TECH's ARClite-C C compiler and allows a successful compile and execution. Interestingly, the optimizer will remove the call from the final application.

## Interrupt Control

In the original[11] V8 core specification, bit 3 in the Processor Status Register (`PSR[3]`, the `I` bit) selects a second bank of registers which are used only during interrupt processing. Additionally, interrupts are enabled when `I` is cleared (0), and disabled when `I` is set (1). Normally, when an interrupt is acknowledged and I is automatically set, the interrupt is processed with the alternate register set, and then the normal register set is restored upon returning from the interrupt. This implementation was designed to improve interrupt performance.

Unfortunately, this scheme causes problems with any background / task-level code that wishes to explicitly disable interrupts, e.g. for a critical code section that accesses global variables. That's because it *changes the register set in use*, which is at odds with the compiler's register allocation and use.

Most ARClite microRISC users change the default implementation to separate the register bank switching from the actions of the I bit. In these implementations, register bank switching requires an explicit action on the part of the programmer. The result is that interrupts can be explicitly disabled without switching the register bank:

> *"The only change made to the V8-microRISC CPU core is that PSR[4] is used select the alternate bank of registers instead of PSR[3] (I bit). Using a PSR[4] allows software to decide when to select the alternate register bank. Masking interrupts without switching register banks allows software to perform atomic operations without losing the contents of the registers. Interrupts can use the alternate register bank by simply executing a STP 4 opcode which is considerably faster than pushing all of the registers. Also, upon the execution of the RTI opcode at the end of the interrupt service routine, the PSR is reloaded with the value from the stack. This automatically selects the register bank that was in use before the interrupt was taken. Note that it is up to the software to very carefully manage which interrupts will use the alternate register bank by setting PSR[4] and which interrupts will simply push the required registers onto the stack. Note that PSR[4] cannot be used for any other purpose."* [12]

In order to be compatible with the original VAutomation Simulator, which implements register bank switching when the I bit is changed, the default definitions for Salvo's interrupt-controlling macros `OSDi()` and `OSEi()` do not explicitly manipulate the I bit. While appropriate for use in the simulator, it is unlikely that these definitions will work for real target hardware.

*When targeting real hardware, Salvo Pro users should add the following definitions to their* `salvocfg.h` *files and rebuild their projects, and also use them when building custom libraries*:

```
#define OSDi()                          di()
#define OSEi()                          ei()
```

where `di()` and `ei()` are defined by HI-TECH's ARClite-C C compiler to disable and enable interrupts (via setting and clearing the `I` bit), respectively.

---

1 The V8C compiler was a DOS (e.g. non-Win32) application, with certain runtime restrictions, e.g. support for only DOS-style 8.3 filenames and extensions.

2 This is done automatically through the `HI_TECH_C` and `_V8` symbols defined by the compiler.

3 See *Interrupt Control*.

4 The lack of an addressable stack severely limits the scope of alternate methods.

5 Salvo v3.2.4 with V8C v7.85PL1.

6 In bytes, as reported under total Program ROM.

7 In bytes, as reported under total Data RAM.

8 Salvo always uses level 0.

9 `OSSignalBinSem()`, like many other user services, disables interrupts on entry and (blindly) re-enables them on exit. The re-enabling of interrupts, if placed inside an ARClite microRISC interrupt routine, causes problems. `OSSignalBinSem()` in the `f`- and `a`-variant libraries control interrupts differently.

10 See ARClite-C manual for more information.

11 VAutomation, Inc., *V8-microRISC Synthesizable 8-bit RISC Microprocessor Core Reference Manual*, September 1999.

12 Correspondence with Salvo for V8 µRISC user, 2003.