RM-ICCAVR

Reference Manual

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

# *Salvo Compiler Reference Manual – ImageCraft ICCAVR*

## Introduction

This manual is intended for Salvo 4 users who are targeting Atmel (http://www.atmel.com/) AVR® and MegaAVR™ microcontrollers[1] with ImageCraft's (http://www.imagecraft.com/) ICCAVR C compiler v7.14 or later.

**Note** Users of ICCAVR v6 and earlier should use Salvo v3.x, which is now deprecated.

## Related Documents

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

- *Salvo User Manual*

## Example Projects

Example Salvo projects for use with ImageCraft's ICCAVR C compiler and the ImageCraft IDE can be found in the:

```
\Pumpkin\Salvo\Example\AVR\
```

directories of every Salvo for Atmel AVR and MegaAVR distribution.

**Tip** These example projects can be easily modified for any AVR or MegaAVR device.

## Features

Table 1 illustrates important features of Salvo's port to ImageCraft's ICCAVR C compiler.

| General | |
|---|---|
| Abbreviated as | `ICCAVR` |
| Available distributions | Salvo Lite, LE & Pro for Atmel AVR and MegaAVR |
| Supported targets | entire AVR and MegaAVR family |
| Header file(s) | `salvoporticcavr.h` |
| Other target-specific file(s) | `salvoporticcavr.s`, `salvoporticcatmega.s`, `salvoporticcatm256.s` |
| **salvocfg.h** | |
| Compiler auto-detected? | yes[2] |
| Include target-specific header file in `salvocfg.h`? | yes |
| **libraries** | |
| Located in | `Lib\ICCAVR-v7` (for v7.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 254 8-bit bytes |
| **Memory & Registers** | |
| Internal and external RAM supported? | yes, via `-bsalvoram:0xstart.0xend` |
| `R20`..`R23` used? | no |
| **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 ImageCraft's ICCAVR C compiler**

## Libraries

### Nomenclature

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

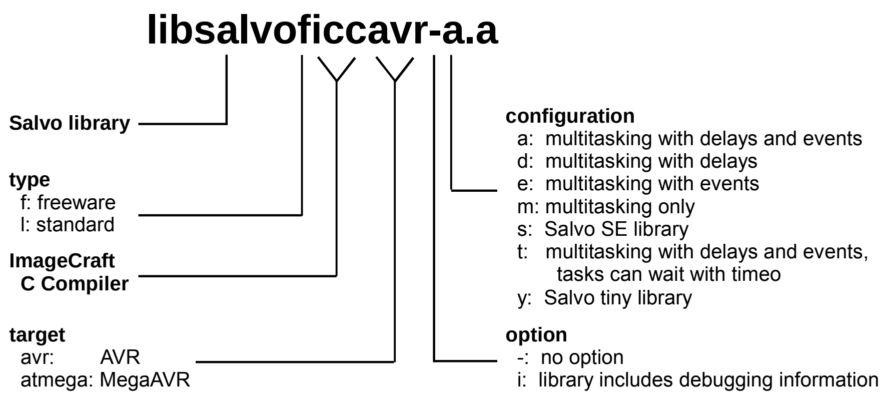

**Figure 1: Salvo Library Nomenclature – ImageCraft's ICCAVR C Compiler**

## 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

Each library is intended for one or more specific processors. Table 2 lists the correct library for each AVR and MegaAVR processor.

| Target Code | Processor(s) |
|---|---|
| `avr:` | AVR and megaAVR devices with 8KB or less program memory (e.g. AT90S8515). These libraries use the basic AVR instruction set. |
| `atmega:` | AVR and megaAVR devices with more than 8KB and less than 256KB of program memory (e.g. ATmega16). These libraries use the megaAVR instruction set. |
| `atm128:` | AVR and megaAVR devices with more than 8KB and less than 256KB of program memory (e.g. ATmega1280). These libraries use the enhanced AVR instruction set. |
| `atm256:` | megaAVR devices with 256KB or more of program memory (e.g. ATmega2561). These libraries use the extended AVR instruction set. |

**Table 2: Processors for Salvo libraries – ImageCraft's ICCAVR C compiler**

**Note** The target code for an unlisted processor will match that used by ImageCraft's ICCAVR C compiler for standard libraries, etc.

## 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 ImageCraft's ICCAVR C compiler's `+g` command-line option. This adds source-level debugging information to the libraries, making them ideal for source-level debugging and stepping in the ICCAVR debugger. To use these libraries, simply select one that includes the debugging information (e.g. `libsalvoliccavrit.a`) instead of one without (e.g. `libsalvoliccavr-t.a`) in your ICCAVR 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 ImageCraft's ICCAVR 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.

| 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 3: Build settings and overrides for Salvo libraries for ImageCraft's ICCAVR 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

There are a total of 44 Salvo libraries for ImageCraft's ICCAVR C compiler. Each Salvo for Atmel AVR and MegaAVR distribution contains the Salvo libraries of the lesser distributions beneath it.

## Target-Specific Salvo Source Files

Depending on the target AVR, one of three different context-switcher files is required for Salvo Pro source-code builds, as shown in Table 4:

| Context-switcher Filename | Application |
|---|---|
| salvoporticcavr.s | AVRs with 8KB or less program memory. Uses `IJMP` and `RJMP` instructions. |
| salvoporticcatmega.s | AVRs with more than 8KB and up to 128KB program memory. Uses `IJMP` and `JMP` instructions. |
| salvoporticcatm256.s | AVRs with 256KB and more program memory. Uses `EIJMP`, `JMP` and `LPM` instructions.[3] |

**Table 4: Target-specific context-switcher files for ImageCraft's ICCAVR C compiler**

## salvocfg.h Examples

Below are examples of `salvocfg.h` project configuration files for different Salvo for Atmel AVR and MegaAVR distributions targeting a variety of AVR targets.

### 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 libsalvoficcavr-t.a**

### 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 libsalvoliccatmega-t.a or libsalvoliccatm256.a**

### 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**

### Interrupt Latencies

Since Salvo's context switcher for ImageCraft's ICCAVR 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] |
|---|---|---|
| `\AVR\AT90S8515\…\tut5lite` | 3206 | 57 |
| `\AVR\AT90S8515\…\tut5le` | 3148 | 57 |
| `\AVR\AT90S8515\…\tut5pro` | 2962 | 56 |

**Table 5: ROM and RAM requirements for Salvo applications built with ImageCraft's ICCAVR 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 ImageCraft's ICCAVR C compiler**

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

## Interrupt

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

```
static unsigned char sreg;

void OSDisableHook(void)
{
  unsigned char sreg_local;


  sreg_local = SREG;
  _CLI();
  sreg = sreg_local;
}


void OSEnableHook(void)
{
  SREG = sreg;
}
```

**Listing 5: Default Salvo interrupt hooks for ImageCraft's ICCAVR C compiler**

These functions clear the `GIE` bit (i.e. disable global interrupts) across Salvo's critical section, and restore the bit to its previous value thereafter. These hooks are suitable for *all* applications. These hooks work very well within Salvo services called from interrupts, as the GIE bit is automatically cleared upon entry to an interrupt. Therefore interrupts are *not* re-enabled at the end of a Salvo service that is called in an ISR. This avoids unnecessary interrupt nesting. The use of the auto variable `sreg_local` avoids issues that would affect the shared global `sreg` when a Salvo service is called from within an ISR.

**Note** 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.[8]

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

**Listing 6: Default Salvo watchdog hook for ImageCraft's ICCAVR 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 ImageCraft's ICCAVR C compiler 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 ImageCraft's ICCAVR C compiler that you are using. Table 6 lists the locations of Salvo libraries as a function of the ImageCraft's ICCAVR C compiler version.

| ICCAVR Version | Salvo Library Location |
|---|---|
| `7.x` | `Lib\ICCAVR-v7` |

**Table 6: Compiler versions, runtime models and Salvo library locations for ImageCraft's ICCAVR C compiler**

### Incompatible Optimizations

There are no known incompatibilities between ImageCraft's ICCAVR C compiler's optimizations (e.g. `-O8`, `-O16`, `-O24`) and Salvo.

## Special Considerations

### ATmega2560/2561 (256KB and greater program memory)

In order to support the larger-than-16-bit program memory space of the ATmega2560/2561 and similar megaAVRs, Salvo uses 4 bytes per function pointer instead of two.[9] This allows Salvo tasks to be located anywhere in program memory.

Salvo for AVR and megaAVR automatically detects whether you are building for the ATmega2560 or ATmega2561, and configures Salvo’s function pointer type accordingly. If you are using a chip other than the ATmega2560 or ATmega2561, define the symbol `OSAVR_256K` for every Salvo source module (including `salvomem.c` in Salvo library builds). This can be done by passing

```
–DOSAVR_256K
```

to the compiler (e.g. in the AVR Studio IDE), or by including

```
#define OAVR_256K 1
```

in the project’s `salvocfg.h`.

## Stack Issues

ImageCraft's ICCAVR C compiler uses two separate stacks – one for return addresses (the hardware stack, which uses `SP`) and one for parameter passing and local storage (the software stack, which uses `Y`).

Compared to a non-Salvo, non-multitasking application with similar call trees, the corresponding Salvo application will require an additional two return addresses (i.e. 4 bytes for typical AVRs and megaAVRs) in the hardware stack.[10]

The size of the hardware stack can be set in the ICCAVR IDE via Project → Options → Target → Advanced → Return Stack Size or on the `iccavr` linker command line, e.g.:

```
iccavr … -dhwstk_size:20 …
```

Applications using nested interrupts, floating points or `long`s will require a hardware stack larger than the default size – see ICCAVR Help for more information.

## External SRAM

Salvo's global objects[11] can be placed in internal or external RAM. In ImageCraft's ICCAVR IDE, the placement of objects (e.g. variables) in the `data` *program area* is controlled via Project → Options → Target → Device Configuration (Internal SRAM), etc. On the `iccavr` linker command line, placement of these objects is specified via –`bdata:start.end`, e.g.:

```
iccavr … -bdata:0x260.0xffff …
```

specifies that the `data` program area start at `0x260` (the end of internal SRAM) and extend to `0xFFFF` (the 64K boundary).

Salvo's global objects can be placed – as a group – anywhere in RAM (internal or external) by specifying the start and end addresses of the `salvoram` program area. This applies to source-code and library builds. For example, to place all of Salvo's global objects in a 256-byte block of external RAM just beneath the 32K boundary, use

```
iccavr … -bsalvoram:0x7F00.0x7FFF …
```

when linking your application.[12]

**Note** If you do not use the `–b` linker command-line argument, the `salvoram` program area will be located immediately after the `bss` program area in the `data` program area. Therefore it is only required if you wish to locate Salvo's global objects separately from your program's variables, etc. You can override the order of the program areas by using the `.area` assembler directive.

## Data Segments

The `RAMPD` register is normally used to access the entire data space on processors with more than 64K bytes data space. There are no provisions for accessing Salvo's global objects outside of the current data segment of 64K bytes.

## Code Compressor

Salvo is compatible with ImageCraft's ICCAVR Code Compressor[13] in both library- and source-code builds.

### Indirect Function Calls

Salvo handles indirect function calls directly through `IJMP` and `EIJMP` instructions, and not through Code Compressor's `xicall`. This is transparent to the user.

## Registers R20..R23

ICCAVR can be instructed to not use registers `R20..R23`. In practice, this has little effect on the Salvo code – it may result in a small speedup and smaller ROM size.

The Salvo libraries are built *without* using `R20..R23` so that control of these registers is left to the programmer.[14]

Salvo Pro users can control the use of these registers in a source-code build.

[1] tinyAVR devices are not supported because of their lack of RAM.
[2] This is done automatically through the `__IMAGECRAFT__` and `_AVR` symbols defined by the compiler.
[3] ICCAVR v7 places 3-byte address of function pointers into a literal table located within the first 64K of program memory. Therefore `LPM` is used (`ELPM` is not required).
[4] Salvo 4.1.2-rc0 with v7.15 compiler.
[5] In bytes. Entire application, including `func_lit`, `text` and `vector` areas.
[6] In bytes. Entire application, including `bss` area. Does not include RAM reserved for the return address (hardware, `SP`) stack, nor for the parameter passing and local storage (software, `Y`) stack.
[7] This hook is valid for all AVR and MegaAVR targets because the register and GIE bit locations are the same for all targets.
[8] This hook is valid for all AVR and MegaAVR targets because the watchdog control register is the same for all targets.
[9] Bits 16 through 21 are defined for the AVR architecture on these parts. Salvo does not use the uppermost / most-significant byte – it remains 0.
[10] Salvo Pro application can reduce this by one return address by inlining `OSSched()`.
[11] E.g. task control blocks, queue pointers, counters, etc.
[12] Failure to allocate enough RAM for the `salvoram` program area will result in an `area 'salvoram' not large enough` linker error.
[13] Code Compressor is included in ICCAVR Professional.
[14] The ICCAVR libraries that do not use R20..R23 have the `–gr` (“global register”) suffix.