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

Salvo Compiler Reference Manual – AVR-GCC

Salvo™
The RTOS that runs in tiny places.™

Introduction

This manual is intended for Salvo users who are targeting Atmel (http://www.atmel.com/) AVR® and MegaAVR™ microcontrollers[1] with GNU's AVR-GCC C/C++ compiler for AVR.

Note It is assumed that the IDE used is Atmel's AVR Studio[1], commonly installed as part of the WinAVR[2] tool suite.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with the AVR-GCC C/C++ compiler:

- *Salvo User Manual*

Example Projects

Example Salvo projects for use with the AVR-GCC C/C++ compiler 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 the AVR-GCC C/C++ compiler.

1 Available directly from Atmel's website, http://www.atmel.com/.
2 "Windows for AVR", available at http://winavr.sourceforge.net/.

General	
Abbreviated as	GCCAVR
Available distributions	Salvo Lite, LE & Pro for Atmel AVR and MegaAVR
Supported targets	entire AVR and MegaAVR family
Header file(s)	salvoportgccavr.h
Other target-specific file(s)	salvoportgccavr.S, salvoportgccavr256.S
salvocfg.h	
Compiler auto-detected?	yes[2]
Include target-specific header file in salvocfg.h?	yes
Libraries	
Located in	Lib\GCCAVR
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 / yes (see section on external memory)
Interrupts	
Interrupt latency in context switcher	2 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 GNU's AVR-GCC C/C++ compiler

Libraries

Nomenclature

The Salvo libraries for the AVR-GCC C/C++ compiler follow the naming convention shown in Figure 1.

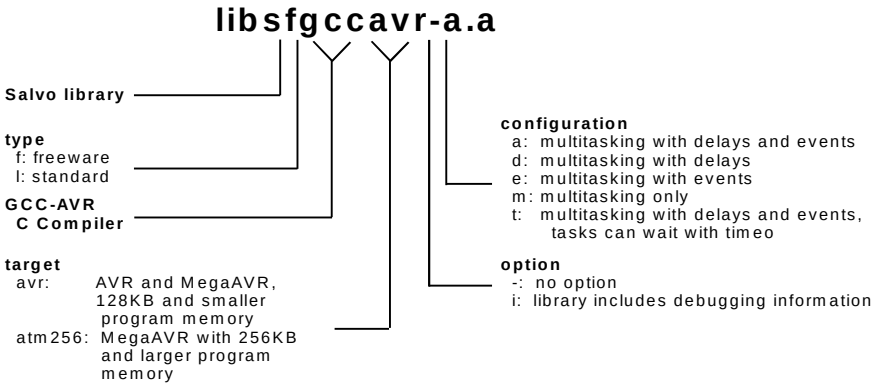

Figure 1: Salvo library nomenclature – GNU's AVR-GCC C/C++ C compiler

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

Each library is intended for one or more specific processors with internal SRAM. Table 1 lists the correct library for each AVR and MegaAVR processor.

Target Code	Processor(s)
`avr:`	AVR and megaAVR devices with up to 128KB of program memory (e.g. AT90S8515, ATmega16, ATmega128). These libraries all use the basic 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 1: Processors for Salvo libraries GNU's AVR-GCC C/C++ compiler

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 AVR-GCC C/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 AVRStudio IDE. To use these libraries, simply select one that includes the debugging information (e.g. `libsalvolgccavrit.a`) instead of one without (e.g. `libsalvolgccavr-t.a`) in your 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 the AVR-GCC 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 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 GNU's AVR-GCC C/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 22 Salvo libraries for the AVR-GCC C/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

The source file `salvoportgccavr.S` is needed for Salvo Pro source-code builds for targets with up to 128KB program memory space.

The source file `salvoportgccavr256.S` is needed for Salvo Pro source-code builds for targets with 256KB and greater program memory space.

Note Never re-name `portgccavrXXX.S` to `portgccavrXXX.s`, as the makefile treats these as two different files. The `.S` file is a user ASM code, whereas the `.s` file is an intermediate file generated by `AVR-GCC` that can be deleted.

salvocfg.h Examples

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

Salvo Lite Library Build

```
#include <avr/io.h>
#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 libsalvofgccavr-t.a

Salvo LE & Pro Library Build

```
#include <avr/io.h>
#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 libsalvolgccavr-t.a

Salvo Pro Source-Code Build

```
#include <avr/io.h>
#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

Memory Usage

tutorial memory usage[4]	total ROM[5]	total RAM[6]
\AVR\AT90S8515\…\tut5lite	2472	58
\AVR\AT90S8515\…\tut5le	2448	58
\AVR\AT90S8515\…\tut5pro	3610	57

Table 3: ROM and RAM requirements for Salvo applications built with GNU's AVR-GCC C/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 GNU's AVR-GCC C/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 uint8_t sreg;

void OSDisableHook(void)
{
  uint8_t sreg_local;


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


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

Listing 5: Default Salvo interrupt hooks GNU's AVR-GCC C/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 )
{
```

```
    wdt_reset();
}
```

Listing 6: Default Salvo watchdog hook for GNU's AVR-GCC C/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

Incompatible Optimizations

All compiler optimizations (`-O1`, `-O2` & `-O3`) are currently incompatible with Salvo code and Salvo tasks. Users should build their projects with optimizations off (`-O0`).

Other code (e.g. user functions, non-Salvo libraries, etc.) can be used with compiler optimizations without issues.

A solution to allow compiler optimizations to be used with Salvo code and Salvo tasks is currently under development.

Register R1

The `avr-libc` library which is an integral part of AVR-GCC assumes register `R1` will always have a `0` value, and Salvo does not modify this register.

Special Considerations

ATmega2560/2561 (256KB and greater program memory)

GNU's AVR-GCC C/C++ compiler does not yet fully support function pointers that span the 256KB or greater program memory space. Salvo supports these parts by considering only the low-order 16 bits of each task function pointer (*tFP*). Therefore Salvo tasks can only be located in the lower 128KB of program memory space.

When building a Salvo application for 256KB or greater program memory space from Salvo libraries or Salvo source code, be sure

to use the library or source code modules appropriate for the larger program memory model.

Stack Issues

The AVR-GCC compiler uses the same stack for return addresses and for local storage. The `Y` pointer (`R29:R28`) is used as a frame pointer.

Compared to a non-Salvo, non-multitasking application with similar call trees, the corresponding Salvo application will require an additional 4 bytes (i.e. one return addresses) in the hardware stack[9].

External SRAM

Salvo's global objects[10] can be placed in internal or external RAM. The placement of data objects is controlled by linker options, and takes the format `-Wl,-Tdata=0x800000+start` for relocationg the program `data` section (ie: variable storage) where the 0x800000+start means hex address with an 0x800000 offset. There is no end location, it is your responsibility as a programmer to not use more RAM than you have. For example:

```
AVR-GCC … -Wl,-Tdata=0x800260 …
```

specifies that the `data` program area start at `0x260` (the end of internal SRAM). This could be added to your makefile with the rest of the linker flags (normally called `LFLAGS`), and in fact your makefile may already be set up for working with external SRAM. If you do this you also have to make sure the processor hardware is set up to use external RAM. It is best to do this as early as possible, in fact the absolute best way is to make a file called `xram.S`, and fill it with something like this:

```
;; begin xram.S

#include <avr/io.h>

        .section .init1,"ax",@progbits

        ldi r16,(1<<SRE) | (1<<SRW)
        out _SFR_IO_ADDR(MCUCR),r16

;; end xram.S
```

This would work for the AT90S8515, enabling the external XRAM with one wait state. The file should be added to your project. Note

that the code executed in that file takes place very early on, before anything like the stack has been set up.

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.

Credits & Acknowledgements

Colin O'Flynn originally wrote the Salvo context switcher in `salvoportgccavr.S`, created the Salvo project makefile system for the Salvo v3 port to AVR-GCC, and wrote much of the documentation surrounding the Salvo v3 port to GNU's AVR-GCC compiler. Colin is active in the AVR community and is the author of various AVR-centric material to be found at the popular AVR Freaks (http://www.avrfreaks.net/) website.

---

1 tinyAVR devices are not supported because of their lack of RAM.

2 This is done automatically through the `__GNUC__` and `__AVR__` symbols defined by the compiler.

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

4 Salvo 4.1.2 with WinAVR release 20071221rc1.

5 `.text` + `.data` + `.bootloader` sections, in bytes, with optimization setting of `-O0`.

6 `.data` + `.bss` + `.noinit` sections, in bytes, with optimization setting of -O0.

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 Salvo Pro application can reduce this by 2 bytes (one return address) by inlining `OSSched()`.

10 E.g. task control blocks, queue pointers, counters, etc.