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

RM-GCCAVR
Reference Manual

Salvo Compiler Reference Manual – GNU avr-gcc

Salvo™
The RTOS that runs in tiny places.™

© Copyright 2003 Pumpkin, Inc. last updated on Oct 20, 2003
All trademarks mentioned herein are properties of their respective companies.

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 compiler.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with the GNU avr-gcc compiler:

Salvo User Manual
Application Note AN-28

Application Note AN-28 explains how to use makefiles to successfully build a Salvo application using GNU's avr-gcc compiler and related tools.

Example Projects

Example Salvo projects for use with GNU's avr-gcc C compiler can be found in the:

```
salvo/ex/ex1/sysy
salvo/tut/tu1/sysy
salvo/tut/tu2/sysy
salvo/tut/tu3/sysy
salvo/tut/tu4/sysy
salvo/tut/tu5/sysy
salvo/tut/tu6/sysy
```

directories of every Salvo for Atmel AVR and MegaAVR distribution.

Features

Table 1 illustrates important features of Salvo's port to GNU's avr-gcc C compiler.

general	
available distributions	Salvo Lite, LE & Pro for Atmel AVR and MegaAVR
supported targets	AVR and MegaAVR family
header file(s)	`portgccavr.h`
other target-specific file(s)	`portgccavr.S`
project subdirectory name(s)	SYSY
salvocfg.h	
target-specific header file required?	yes
compiler auto-detected?	yes[2]
context switching	
method	function-based via `OSDispatch()` & `OSCtxSw(label)`
`_OSLabel()` required?	yes
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)
register usage	`R0`, `R16`, `R17` used for temporary storage in `OSCtxSw()` routine. `R0`, `R18` used for temporary storage in `OSDispatch()` routine
interrupts	
controlled via	`I` bit
interrupt status preserved in critical sections?	yes
method used	saved to local variable at start of routine, restored at end
nesting limit	unlimited
alternate methods possible?	not recommended
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 Compiler

Libraries

Nomenclature

The Salvo libraries for GNU's avr-gcc C compiler follow the naming convention shown in Figure 1.

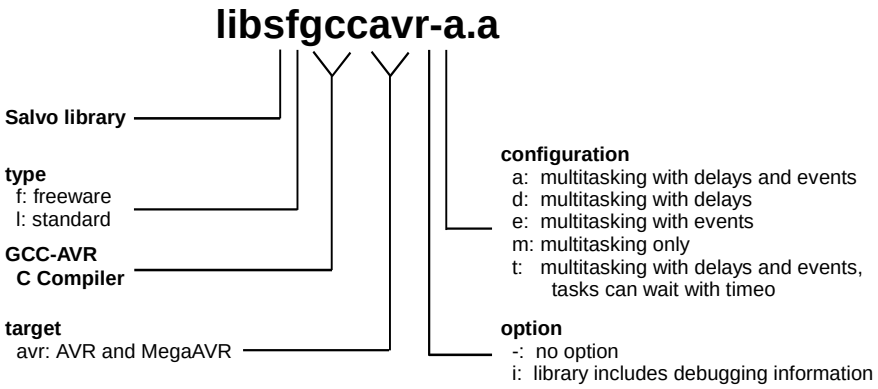

Figure 1: Salvo Library Nomenclature – GNU's avr-gcc C Compiler

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

One set of libraries will work with all of the AVR microcontrollers, provided they have the following features:

- Internal SRAM
- `ICALL` instruction

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 GNU's avr-gcc 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. `libslgccavrit.a`) instead of one without (e.g. `libslgccavr-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 GNU's avr-gcc 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.

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	cleared in `OSSched()`.
system tick counter	available, 32 bits

Table 2: Build Settings and Overrides for Salvo Libraries for GNU's avr-gcc 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 a total of 15 Salvo libraries for GNU's avr-gcc 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 `portgccavr.S` is needed for Salvo Pro source-code builds.

Note Never re-name `portgccavr.S` to `portgccavr.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.

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 2) will be used.

Salvo Lite Library Build

```
#include <avr/io.h>
#define OSUSE_LIBRARY            TRUE
#define OSLIBRARY_TYPE           OSF
#define OSLIBRARY_CONFIG         OSA
```

Listing 1: Example salvocfg.h for Library Build Using libsfgccavr-a.a

Salvo LE & Pro Library Build

```
#include <avr/io.h>
#define OSUSE_LIBRARY            TRUE
#define OSLIBRARY_TYPE           OSL
#define OSLIBRARY_CONFIG         OSA
```

Listing 4: Example salvocfg.h for Library Build Using libslgccavr-a.a

Salvo Pro Library Build with Source-Code Debugging

```
#include <avr/io.h>
#define OSUSE_LIBRARY            TRUE
#define OSLIBRARY_TYPE           OSL
#define OSLIBRARY_OPTION         OSI
#define OSLIBRARY_CONFIG         OSA
```

Listing 4: Example salvocfg.h for Library Build Using libslgccavria.a

Salvo Pro Source-Code Build

```
#include <avr/io.h>
#define OSENABLE_IDLING_HOOK     TRUE
#define OSENABLE_SEMAPHORES      TRUE
#define OSEVENTS                 1
#define OSTASKS                  3
```

Listing 5: Example salvocfg.h for Source-Code Build

Performance

Memory Usage

tutorial memory usage[4]	total ROM[5]	total RAM[6]
`tu1lite`	197	24
`tu2lite`	253	24
`tu3lite`	280	26
`tu4lite`	562	35
`tu5lite`	770	47
`tu6lite`	848	48
`tu6pro`[7]	760	44

Table 3: ROM and RAM requirements for Salvo Applications built with GNU's avr-gcc C Compiler

Special Considerations

Stack Issues

GNU's avr-gcc C compiler uses two separate stacks – one for return addresses (the hardware stack, which uses `SP`) and one for 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 4 bytes (i.e. two return addresses) in the hardware stack[8].

The hardware stack and the software stack are set to the same location. However the hardware stack grows downwards, and the software stack grows upwards.

External SRAM

Salvo's global objects[9] 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.

Optimizer

Salvo is compatible with GNU's avr-gcc code optimizer at all levels in both source and library builds.

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.

Library Locations

The Salvo installer places its libraries for avr-gcc in the `/salvo/lib/gccavr` folder. When linking to Salvo libraries, an additional library path must be specified in the link phase, e.g. via:

```
avr-gcc … -L c:/salvo/lib/gccavr …
```

This is normally done automatically as part of the Salvo makefile system for use with avr-gcc and WinAVR.

Credits & Acknowledgements

Colin O'Flynn wrote the Salvo context switcher in `portgccavr.S`, created the Salvo project makefile system, and wrote much of the documentation surrounding the Salvo 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 v3.2.0 with WinAVR release 20030424

5 In words with optimization setting of `-Os`. Note that avr-size returns sizes in bytes not words

6 In bytes, this is the `.bss` section of the `.elf` file.

7 Salvo Pro build differs slightly from Salvo Lite build due to configuration – see tutorial's `salvocfg.h`.

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

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