AN-28
Application Note

Building a Salvo Application with GNU's avr-gcc C Compiler, WinAVR and AVRStudio

Introduction

This Application Note explains how to use GNU's avr-gcc C compiler, WinAVR and with Atmel®'s AVRStudio (all are available[1] through http://www.avrfreaks.net/) to create a multitasking Salvo application on Atmel AVR and MegaAVR devices.

We will show you how to build the example program located in `/salvo/ex/ex1/main.c` for an Atmel AT90S8515 using WinAVR 20030913 and AVRStudio v4.07. For more information on how to write a Salvo application, please see the *Salvo User Manual*.

Before You Begin

If you have not already done so, install WinAVR and AVRStudio (WinAVR includes avr-gcc). You will also need a `bash` shell (or a functionally identical Linux-like command-line environment) in order to drive the makefile system. One is included with WinAVR.[2]

Related Documents

The following Salvo documents should be used in conjunction with this Application Note when building Salvo applications with GNU's avr-gcc C compiler and AVRStudio:

Salvo User Manual
Salvo Compiler Reference Manual RM-GCCAVR

created by Andrew E. Kalman on Oct 19, 2003 updated on Oct 20, 2003

Additionally, the following documents (available at http://www.avrfreaks.net/ and other locations) should be also used in conjunction with this Application Note:

Downloading, Installing and Configuring WinAVR

Toolsets

avr-gcc is a command-line-driven C compiler. WinAVR is a collection of command-line and Windows-based tools, some of which serve as a front-end to avr-gcc, etc. AVRStudio is a Windows graphical IDE.

Since these toolsets can be configured in a multitude of different ways, this Application Note will focus on configuring the WinAVR-based makefile system for use with Salvo, and building applications from the command line.

Creating and Configuring a New Project

Select a directory where your project will reside (e.g. `c:\temp`). You will place all of your project-specific files here.

The makefile

Each Salvo for Atmel AVR and MegaAVR distribution contains example makefiles for the supplied projects. These makefiles are derived from those commonly used with other avr-gcc tools, e.g. WinAVR. Should you wish to make your own makefile, or want to use a newer one (e.g. for the latest WinAVR release), you can easily modify an existing makefile to fully support Salvo.

The makefile should be stored in your project directory, e.g. `c:/temp/makefile`.

Tip It's much easier to start with an existing, "known-good" makefile than to try to create your own from scratch. Therefore we recommend that most users build an existing Salvo project first, verify that it builds successfully, and only then consider editing other makefiles as part of creating their own Salvo projects.

This section will examine those portions of the project's makefile that are specific to Salvo.

Note Each distribution also includes some additional files that are part of the Salvo makefile system for the avr-gcc C compiler. Their operation is normally hidden from the user.

Salvo: Project-Specific makefile Symbols

Normally, you need only define or modify six makefile symbols in order to successfully build a Salvo application with the avr-gcc compiler. They are shown in Listing 1, and are normally at the beginning of the makefile.

```
########################################################
# Salvo Options
# This section, with its six (6) defined symbols, is all
#  that a Salvo user will normally need to modify when
#  configuring a Salvo project for use with avr-gcc
#

# Specify Salvo install directory, eg c:/salvo
SALVO_DIR = c:/salvo


# Specify Salvo project build type. This must match
#  your project's salvocfg.h configuration file.
# Normal options are:
#         MAKE_WITH_FREE_LIB
#         MAKE_WITH_STD_LIB
#         MAKE_WITH_SOURCE
SALVO_BUILD_TYPE = MAKE_WITH_FREE_LIB


# Specify any extra Salvo definitions (often used in projects that
#  are part of the standard distribution)
SALVO_EXTRA_DEFS =


# Specify any other include paths the project requires
SALVO_EXTRA_INCS = c:/temp


# Specify any other source files that the project requires
SALVO_EXTRA_SRCS =


# Salvo Source Code Files (for source code build ONLY, from Salvo Pro).
#  Add whichever Salvo source code files (e.g. sched, timer, etc) your
#  Salvo Pro source-code build requires. Filename only -- no full paths,
#  no extensions.
# N.B. Do not add mem -- it's already included in every Salvo build.
SALVO_SRCS = binsem event init inittask qins sched
```

Listing 1: Salvo makefile Preamble with User-configurable Definitions

SALVO_DIR

`SALVO_DIR` tells the makefile system where Salvo is installed (usually `c:/salvo`).

SALVO_BUILD_TYPE

`SALVO_BUILD_TYPE` tells the makefile system what kind of build this is. The allowable values for the various Salvo distributions are listed below:

SALVO_BUILD_TYPE	
Salvo Lite	`MAKE_WITH_FREE_LIB`
Salvo LE	`MAKE_WITH_FREE_LIB`, `MAKE_WITH_STD_LIB`
Salvo Pro	`MAKE_WITH_FREE_LIB`, `MAKE_WITH_STD_LIB`, `MAKE_WITH_SOURCE`

Table 1: SALVO_BUILD_TYPE Allowable Values

`MAKE_WITH_FREE_LIB` tells the makefile system to link to one of Salvo's freeware libraries when building the application. `MAKE_WITH_FREE_LIB` tells it to use a standard library. And `MAKE_WITH_SOURCE` tells it to build the application directly from the Salvo source files.

SALVO_EXTRA_DEFS

You can define extra, project-specific symbols via the `SALVO_EXTRA_DEFS` symbol.

SALVO_EXTRA_INCS

You can specify additional directories to be searched (e.g. for header files) via the `SALVO_EXTRA_INCS` symbol.

SALVO_EXTRA_SRCS

You can specify additional source files to be added to the project via the `SALVO_EXTRA_SRCS` symbol.

SALVO_SRCS

When doing a Salvo source-code build with Salvo Pro, you must define `SALVO_SRCS` to be all of the Salvo source files (without pathnames or extensions) required for the project.

Note `salvo/tut/tu6/sysy/makefile` illustrates the use of `SALVO_EXTRA_DEFS`, `SALVO_EXTRA_INCS`, `SALVO_EXTRA_SRCS` and `SALVO_SRCS`.

Salvo: Additional makefile Paths and Settings

Salvo variables must be integrated into the existing makefile variables, as shown below in Listing 2. `CFLAGS` and `ASFLAGS` are assumed to be the names of the makefile variables that hold the flags to add to the compiler and assembler command line. If the makefile you are using doesn't use this convention, you must make the necessary changes.

```
########################################################
# Brings Salvo code together
# N.B. Normally, there is no need to make changes to
#  this area!

# Salvo include directory
SALVO_DIR_INC = $(SALVO_DIR)/inc


# Salvo source directory
SALVO_DIR_SRC = $(SALVO_DIR)/src


# Salvo library directory
SALVO_DIR_LIB = $(SALVO_DIR)/lib/gccavr


# SYSY is the GNU GCC AVR Compiler -- used by Salvo projects
#  supplied in the distribution.
SALVO_SYS = SYSY


# Command-line additional symbols and include paths for Salvo
#  and user source files
SALVO_ADDS = -D $(SALVO_SYS) \
  -D $(SALVO_BUILD_TYPE)=1 \
  $(patsubst %,-D %,$(SALVO_EXTRA_DEFS)) \
  $(patsubst %,-I %,$(SALVO_EXTRA_INCS)) \
  -I $(SALVO_DIR_INC)


# Add in other (user) source files for this project
SRC += $(patsubst %,%,$(SALVO_EXTRA_SRCS))


# Salvo's mem.c must be compiled with every Salvo project
SRC += $(SALVO_DIR_SRC)/mem.c


# If we're doing a Salvo Pro source-code build, then add in
#  the Salvo source files the user has specified, as well
#  as portgccavr.S.
# Pro source-code builds don't use Salvo libraries -- all
#  others do. Library name is found in an included makefile
#  listed in the targets area
ifeq ($(SALVO_BUILD_TYPE), MAKE_WITH_SOURCE)

SRC += $(patsubst %,$(SALVO_DIR_SRC)/%.c,$(SALVO_SRCS))
ASRC += $(SALVO_DIR_SRC)/portgccavr.S
SALVO_USELIBS = false

else

SALVO_USELIBS = true
SALVO_ADDS += -L $(SALVO_DIR_LIB)

endif


# Compiler and Assembler get same extra flags
CFLAGS += $(SALVO_ADDS)
ASFLAGS += $(SALVO_ADDS)
```

Listing 2: Additional Salvo Makefile Paths and Settings

Salvo: Automatically Determining the Correct Library

The Salvo makefile system is extended via an additional makefile in order to automatically specify the correct Salvo library when doing a library build. The `getsalvolibrary` target will figure out what the library the user has selected in the project's `salvocfg.h` file. The code to do this is shown in Listing 3, and should be included in sometime after the `all` target.

```
# Automatically figures out what Salvo library to include
# This file sets two variables: one is SALVO_LIB which will
# have the add to the LDFLAGS variable, such as -lsfgccavr-d
# and also adds to LDFLAGS the proper name (essentially adds
# SALVO_LIB to LDFLAGS). If you are missing this file for
# some reason either download it from the Pumpkin Inc website
# or manually add the proper library name to LDFLAGS
# N.B. Normally, there is no need to make changes to
#  this area!
include $(SALVO_DIR_SRC)/make/makefile_autolibs
```

Listing 3: Including makefile_autolibs

Salvo: Providing Feedback on the Build Process (Optional)

The next step is purely optional, as it is used to provide user feedback. It reports which library has been auto-detected for use in Salvo, and Listing 4 should be included as a makefile target sometime after the previous target.

Note: the `@echo` lines have tabs before them, not just four or five spaces. Only use tabs for lines that are supposed to be executed in makefiles.

```
# Display Salvo options back to use to make sure they got them
# right...
# N.B. Normally, there is no need to make changes to
#  this area!
salvoecho :
          @echo ""
          @echo ""
          @echo "***** Salvo options set up in this Makefile *****"
ifeq ($(SALVO_BUILD_TYPE), MAKE_WITH_SOURCE)
          @echo "Building from source"
  else
          @echo "Library to be included is $(subst -l,lib,$(SALVO_LIB)).a"
endif
          @echo "**************************************************"
          @echo ""
```

Listing 4: Optional Build-Time Feedback

Salvo: Modifying the Targets

The final step is to make sure the makefile knows about these new targets. First, modify the `all` target to include your new targets. For example if previously the `all` target looks like Listing 5, then after modifying it might look like Listing 6. The targets `getsalvolibrary` and `salvoecho` are added. If you did not include the code in Listing 4 then do not include the `salvoecho` target.

```
# Default target.
all: begin gccversion sizebefore \
    $(TARGET).elf $(TARGET).hex $(TARGET).eep \
    $(TARGET).lss sizeafter finished end
```

Listing 5: all Target before Modification

```
# Default target.
all: begin gccversion sizebefore getsalvolibrary salvoecho\
    $(TARGET).elf $(TARGET).hex $(TARGET).eep \
    $(TARGET).lss sizeafter finished end
```

Listing 6: all Target after Modification

Finally, go to the end of the makefile where you will find a `.PHONY` listing, these are targets that don't affect the build process. If you have included the `salvoecho` target (Listing 4) then list it here, as shown in Listing 7.

```
# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion \
coff extcoff clean clean_list program salvoecho
```

Listing 7: .PHONY Target after Modification

Other Makefile Settings

At this point you should have added all of the requisite Salvo-centric symbols, rules, etc. to the project's makefile. Any other settings (e.g. the `$(TARGET)` symbol) are not specific to Salvo *per se*, and follow the usual rules for makefiles. Please consult the WinAVR guide, etc. for more information on using makefiles with the avr-gcc compiler.

Adding Salvo-specific Files to the Project

Now it's time to add any additional Salvo files your project needs. Salvo applications can be built by linking to precompiled Salvo libraries, or with the Salvo source code files as nodes in your project.

Adding a Library

For a *library build*, Salvo's makefile system automatically figures out the appropriate Salvo library. Therefore there is no need to explicitly identify a library for your project.

You can find more information on Salvo libraries in the *Salvo User Manual* and in the *Salvo Compiler Reference Manual RM-GCCAVR*.

The salvocfg.h Header File

You will also need a `salvocfg.h` file for this project. To use a typical library (e.g. `libsfgccavr-a.a`), your `salvocfg.h` should contain only:

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

Listing 8: salvocfg.h for a Library Build

Create this file and save it in your project directory, e.g. `c:/temp/salvocfg.h`.

Proceed to *Building the Project*, below.

Adding Salvo Source Files

If you have Salvo Pro, you can do a *source code build* instead of a library build. The application in `/salvo/ex/ex1/main.c` contains calls to the following Salvo user services:

```
OS_Delay()           OSInit()
OS_WaitBinSem()      OSSignalBinSem()
OSCreateBinSem()     OSSched()
OSCreateTask()       OSTimer()
OSEi()
```

You must add the Salvo source files that contain these user services, as well as those that contain internal Salvo services, to your project. The *Reference* chapter of the *Salvo User Manual* lists the source file for each user service. Internal services are in other Salvo source files. For this project, the complete list is:

```
binsem.c                inittask.c
delay.c                 mem.c
event.c                 qins.c
idle.c                  sched.c
init.c                  timer.c
```

Salvo's `mem.c` module is automatically added to every project via the makefile system. Therefore you must edit the `SALVO_SRCS` symbol in the project's makefile to read:

```
SALVO_SRCS = binsem delay event idle init inittask qins sched timer
```

Listing 9: Salvo Source Files for a Source Code Build

The salvocfg.h Header File

You will also need a `salvocfg.h` file for this project. Configuration files for source code builds are quite different from those for library builds (see Listing 8, above). For a source code build, the `salvocfg.h` for this project contains only:

```
#define OSBYTES_OF_DELAYS           1
#define OSENABLE_IDLING_HOOK        TRUE
#define OSENABLE_BINARY_SEMAPHORES  TRUE
#define OSEVENTS                    1
#define OSTASKS                     3
```

Listing 10: salvocfg.h for a Source Code Build

Create this file and save it in your project directory, e.g. `c:/temp/salvocfg.h`.

Building the Project

With everything in place, you can now build the project by invoking the makefile from the project's directory:

```
user@domain /cygdrive/c/temp
$ make
```

Listing 11: Building the Project (Default Target)

The `stdout` window will reflect the avr-gcc command lines:

```
set -e; avr-gcc -MM -mmcu=at90s8515 -I. -g -Os -funsigned-char
-funsigned-bitfields -fpack-struct -fshort-enums -Wall
-Wstrict-prototypes -Wa,-ahlms=c:/salvo/src/mem.lst  -D SYSY -D
MAKE_WITH_FREE_LIB=1  -I c:/temp -I c:/salvo/inc -L
c:/salvo/lib/gccavr c:/salvo/src/mem.c \
| sed 's,\(.*\)\.o[ :]*,\1.o \1.d : ,g' > c:/salvo/src/mem.d; \
[ -s c:/salvo/src/mem.d ] || rm -f c:/salvo/src/mem.d
set -e; avr-gcc -MM -mmcu=at90s8515 -I. -g -Os -funsigned-char
-funsigned-bitfields -fpack-struct -fshort-enums -Wall
-Wstrict-prototypes -Wa,-ahlms=main.lst  -D SYSY -D
MAKE_WITH_FREE_LIB=1  -I c:/temp -I c:/salvo/inc -L
c:/salvo/lib/gccavr main.c \
| sed 's,\(.*\)\.o[ :]*,\1.o \1.d : ,g' > main.d; \
[ -s main.d ] || rm -f main.d

-------- begin --------
avr-gcc --version
avr-gcc.exe (GCC) 3.3 20030421 (prerelease)
Copyright (C) 2002 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

rm -rf c:/salvo/src/make/salvofindlib.o
avr-gcc -c -w c:/salvo/src/make/salvofindlib.c -I .  -D SYSY -D
MAKE_WITH_FREE_LIB=1  -I c:/temp -I c:/salvo/inc -L
c:/salvo/lib/gccavr -o c:/salvo/src/make/salvofindlib.o


***** Salvo options set up in this Makefile *****
Library to be included is libsfgccavr-a.a
*************************************************

avr-gcc -c -mmcu=at90s8515 -I. -g -Os -funsigned-char
-funsigned-bitfields -fpack-struct -fshort-enums -Wall
-Wstrict-prototypes -Wa,-ahlms=main.lst  -D SYSY -D
MAKE_WITH_FREE_LIB=1  -I c:/temp -I c:/salvo/inc -L
c:/salvo/lib/gccavr main.c -o main.o
avr-gcc -c -mmcu=at90s8515 -I. -g -Os -funsigned-char
-funsigned-bitfields -fpack-struct -fshort-enums -Wall
-Wstrict-prototypes -Wa,-ahlms=c:/salvo/src/mem.lst  -D SYSY -D
MAKE_WITH_FREE_LIB=1  -I c:/temp -I c:/salvo/inc -L
c:/salvo/lib/gccavr c:/salvo/src/mem.c -o c:/salvo/src/mem.o
avr-gcc -mmcu=at90s8515 -I. -g -Os -funsigned-char
-funsigned-bitfields -fpack-struct -fshort-enums -Wall
-Wstrict-prototypes -Wa,-ahlms=main.o  -D SYSY -D
MAKE_WITH_FREE_LIB=1  -I c:/temp -I c:/salvo/inc -L
c:/salvo/lib/gccavr main.o  c:/salvo/src/mem.o   --output
main.elf -Wl,-Map=main.map,--cref  -lm -lsfgccavr-a
avr-objcopy -O ihex -R .eeprom main.elf main.hex
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load"
\
--change-section-lma .eeprom=0 -O ihex main.elf main.eep
avr-objdump -h -S main.elf > main.lss
Size after:
main.elf  :
section    size      addr
.text      1554         0
.data         0   8388704
.bss         45   8388704
.noinit       0   8388749
.eeprom       0   8454144
.stab      2760         0
.stabstr   4040         0
Total      8399


Errors: none
-------- end --------
```

Listing 12: Build Results for A Successful Library Build

The build directory can be "cleaned" prior to a build using:

```
user@domain /cygdrive/c/temp
$ make clean
```

Listing 13: Cleaning the Project Directory

This is especially useful when switching between library and source-code builds.

Testing the Application

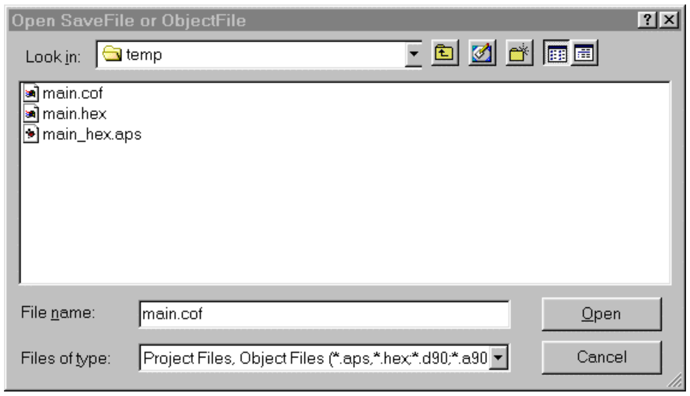
You can test and debug this application with full source code integration in AVRStudio. But first, after making the application, you must generate an appropriately debug-enabled output file, via:

```
user@domain /cygdrive/c/temp
$ make extcoff
```

Listing 14: Building for Symbol Debugging with AVRstudio v4.07

This will generate an extended COFF-format file for AVRStudio.

Launch AVRStudio. When prompted, open the COFF (`.cof`) file you just created:


Open SaveFile or ObjectFile

Look in: temp

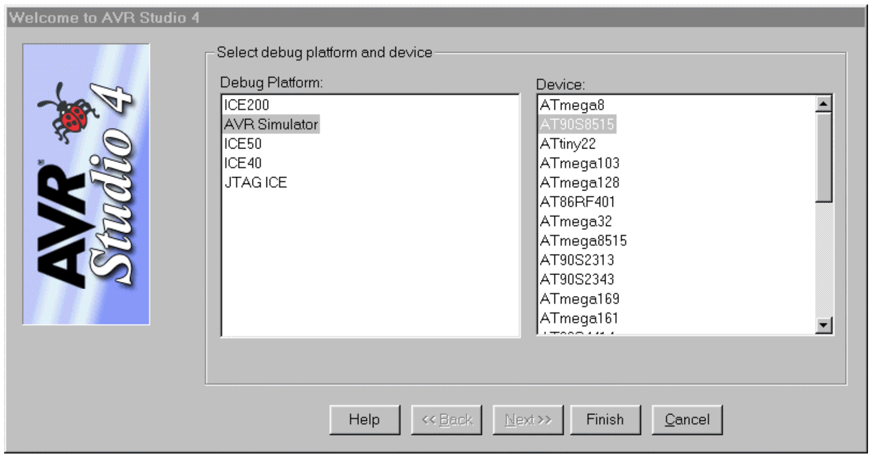
main.cof
main.hex
main_hex.aps

File name: main.cof

Files of type: Project Files, Object Files (*.aps,*.hex;*.d90;*.a90

Figure 1: Opening the COFF File for Symbolic Debugging

Click Open. Under Debug Platform select AVR Simulator, and under Device select AT90S8515:



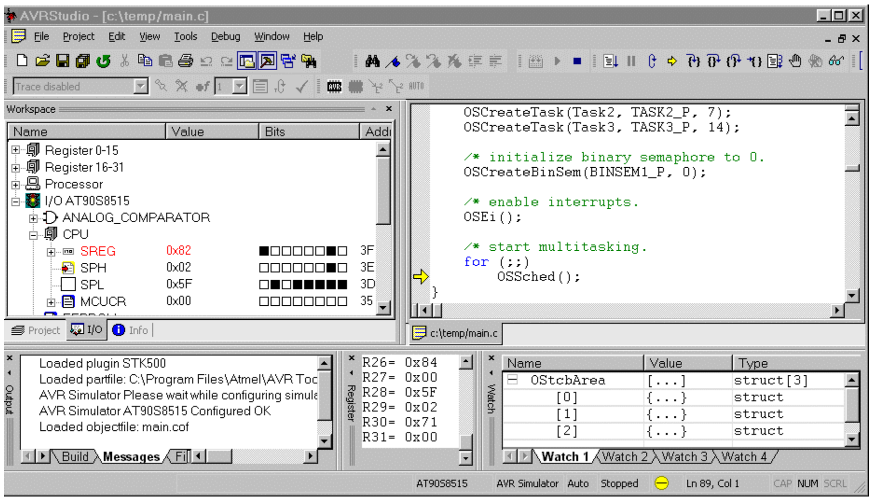
Select debug platform and device

Debug Platform: ICE200, AVR Simulator (selected), ICE50, ICE40, JTAG ICE

Device: ATmega8, AT90S8515 (selected), ATtiny22, ATmega103, ATmega128, AT86RF401, ATmega32, ATmega8515, AT90S2313, AT90S2343, ATmega169, ATmega161

Figure 2: Selecting the ToolSuite in the Project Wizard

Click Finish. AVR Studio will load the `.cof` object file and position the runtime / debugging cursor at the start of the project's `main()`. You can then step through each source-code module in the project:



```
Name              Value   Bits       Addr
Register 0-15
Register 16-31
Processor
I/O AT90S8515
  ANALOG_COMPARATOR
  CPU
    SREG          0x82    ■□□□□□■□   3F
    SPH           0x02    □□□□□□■□   3E
    SPL           0x5F    □■□■■■■■   3D
    MCUCR         0x00    □□□□□□□□   35
```

```
    OSCreateTask(Task2, TASK2_P, 7);
    OSCreateTask(Task3, TASK3_P, 14);

    /* initialize binary semaphore to 0.
    OSCreateBinSem(BINSEM1_P, 0);

    /* enable interrupts.
    OSEi();

    /* start multitasking.
    for (;;)
        OSSched();
}
```

```
Loaded plugin STK500
Loaded partfile: C:\Program Files\Atmel\AVR Toc
AVR Simulator Please wait while configuring simula
AVR Simulator AT90S8515 Configured OK
Loaded objectfile: main.cof
```

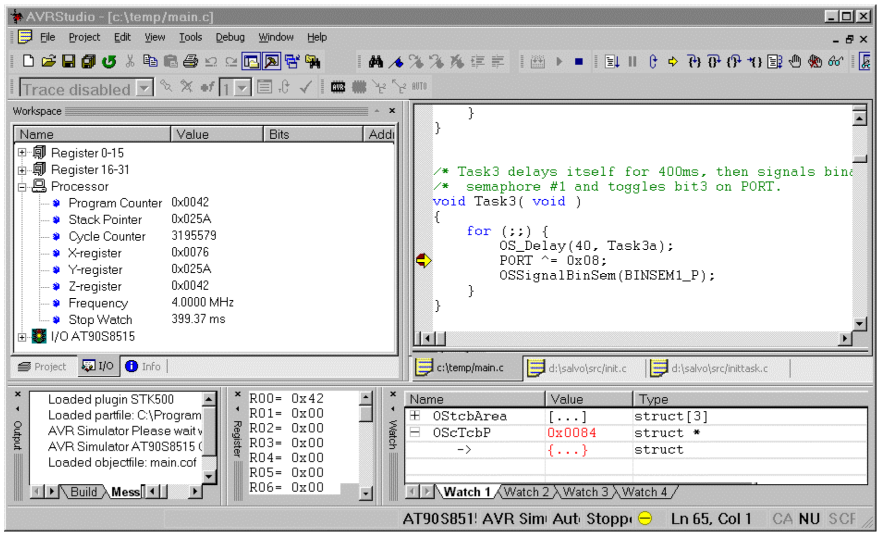
```
R26=  0x84
R27=  0x00
R28=  0x5F
R29=  0x02
R30=  0x71
R31=  0x00
```

Name	Value	Type
OStcbArea	[...]	struct[3]
[0]	{...}	struct
[1]	{...}	struct
[2]	{...}	struct

AT90S8515 AVR Simulator Auto Stopped Ln 89, Col 1 CAP NUM SCRL

Figure 3: Source-Level Debugging in AVRStudio's Simulator

For example, to measure a delay period in the simulator, expand the Processor tab in the Workspace window so that the Stop Watch feature is visible. After a successful build, open the project's `main.c` (i.e. `/temp/main.c`), set a breakpoint on the `PORTB ^= 0x08;` line of `Task3()`, and select Debug → Run. After a while, program execution will stop at the breakpoint in `Task3()`. Now zero the stopwatch in the Stop Watch window by double-clicking on it, and right-click to select Stop Watch: show as milliseconds. Select Debug → Run again, and wait until execution stops. The Stopwatch window now displays an elapsed time of 400ms (40 times 10ms, the `Timer0`-driven system tick rate in this application for a 4MHz clock).


AVRStudio - [c:\temp/main.c]
File Project Edit View Tools Debug Window Help
Trace disabled
Workspace
Project I/O Info
c:\temp/main.c d:\salvo\src/init.c d:\salvo\src/inittask.c
Output Register Watch
Build Mess
Watch 1 Watch 2 Watch 3 Watch 4
AT90S851! AVR Sim Aut Stopp Ln 65, Col 1 CA NU SCF

Name	Value	Bits	Addi
Register 0-15			
Register 16-31			
Processor			
Program Counter	0x0042		
Stack Pointer	0x025A		
Cycle Counter	3195579		
X-register	0x0076		
Y-register	0x025A		
Z-register	0x0042		
Frequency	4.0000 MHz		
Stop Watch	399.37 ms		
I/O AT90S8515			

```
        }
}


/* Task3 delays itself for 400ms, then signals bina
/*  semaphore #1 and toggles bit3 on PORT.
void Task3( void )
{
    for (;;) {
        OS_Delay(40, Task3a);
        PORT ^= 0x08;
        OSSignalBinSem(BINSEM1_P);
    }
}
```

```
Loaded plugin STK500
Loaded partfile: C:\Program
AVR Simulator Please wait v
AVR Simulator AT90S8515 (
Loaded objectfile: main.cof
```

```
R00= 0x42
R01= 0x00
R02= 0x00
R03= 0x00
R04= 0x00
R05= 0x00
R06= 0x00
```

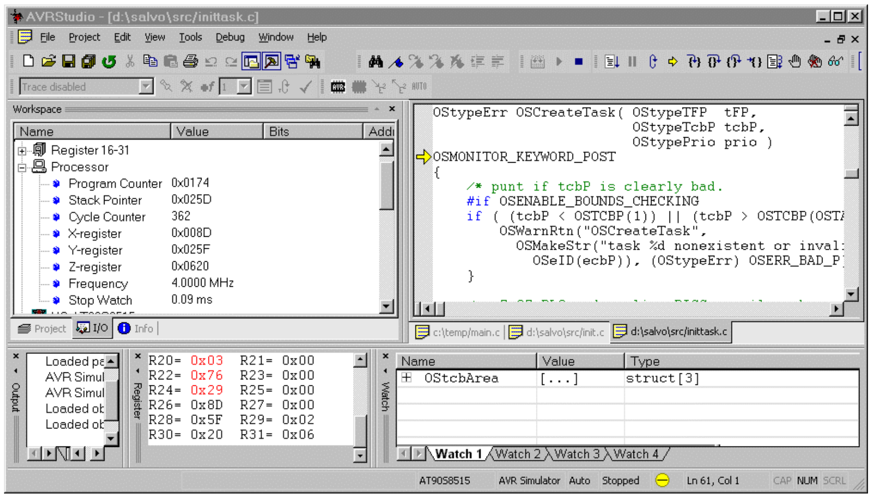
Name	Value	Type
OStcbArea	[...]	struct[3]
OScTcbP	0x0084	struct *
->	{...}	struct

Figure 4: Measuring 400ms of Task Delay in the Simulator via a Breakpoint

Note The 630 microseconds (i.e. 400.00ms – 399.37ms) that are "short" in the Stop Watch window of Figure 4 are due to the way the hardware timer was initialized in this application – the actual timer period is 1/(4.000MHz/1024/(38+1)) = 9.984ms, and 40 * 9.984ms = 399.36ms. See the *Salvo User Manual* for more information on the system timer.

Stepping Through Salvo Source Code

If you have Salvo Pro and are doing a full source-code build, or a debug-enabled library build, you can also trace program execution through the Salvo source code. Select Debug → Reset, Debug → Remove Breakpoints, and set a breakpoint at the first call to `OSCreateTask()` in `main.c`. Select Debug → Run. Execution will stop in `main.c` at the call to `OSCreateTask()`. Now choose Debug → Step Into. The `/salvo/src/inittask.c` file window will open, and you can step through and observe the operation of `OSCreateTask()`.


AVRStudio - [d:\salvo\src/inittask.c]
File Project Edit View Tools Debug Window Help
Trace disabled
Workspace
Project I/O Info
c:\temp/main.c
d:\salvo\src/init.c
d:\salvo\src/inittask.c
Output
Register
Watch
Watch 1 Watch 2 Watch 3 Watch 4
AT90S8515 AVR Simulator Auto Stopped Ln 61, Col 1 CAP NUM SCRL

Name	Value	Bits	Addi
Register 16-31			
Processor			
Program Counter	0x0174		
Stack Pointer	0x025D		
Cycle Counter	362		
X-register	0x008D		
Y-register	0x025F		
Z-register	0x0620		
Frequency	4.0000 MHz		
Stop Watch	0.09 ms		

```
OStypeErr OSCreateTask( OStypeTFP  tFP,
                        OStypeTcbP tcbP,
                        OStypePrio prio )
OSMONITOR_KEYWORD_POST
{
    /* punt if tcbP is clearly bad.
    #if OSENABLE_BOUNDS_CHECKING
    if ( (tcbP < OSTCBP(1)) || (tcbP > OSTCBP(OSTA
      OSWarnRtn("OSCreateTask",
        OSMakeStr("task %d nonexistent or inval:
          OSeID(ecbP)), (OStypeErr) OSERR_BAD_P
    }
```

```
Loaded pa
AVR Simul
AVR Simul
Loaded ob
Loaded ob
```

```
R20= 0x03   R21= 0x00
R22= 0x76   R23= 0x00
R24= 0x29   R25= 0x00
R26= 0x8D   R27= 0x00
R28= 0x5F   R29= 0x02
R30= 0x20   R31= 0x06
```

Name	Value	Type
OStcbArea	[...]	struct[3]

Figure 5: Stepping Through Salvo Source Code

Troubleshooting

Errors Reported by make

The Salvo makefile system for avr-gcc requires only a properly defined `salvocfg.h` (see the *Salvo Compiler Reference Manual RM-GCCAVR* for more information) and the additions to the WinAVR-style makefile outlined above, in addition to a properly installed Salvo for Atmel AVR and MegaAVR distribution.

When reviewing make's output in case of a build error, ensure that:

- the Salvo path is set correctly (i.e. the `-I` and `-L` command-line arguments to avr-gcc point to a valid Salvo installation)
- the symbol `SYSY` is defined (i.e. `-D SYSY`)
- there is a `MAKE_WITH_XYZ` symbol defined
- for Salvo Pro source-code builds, all of the necessary Salvo source modules are listed in the `$(SALVO_SRCS)` makefile symbol

Tip All of these will be in place if you start with a makefile from an existing Salvo `SYSY` project.

Example Projects

Example projects for GNU's avr-gcc C compiler can be found in the `salvo/ex/ex1/sysy` and `salvo/tut/tu1-6/sysy` directories. A single makefile handles Salvo Lite, Salvo LE and Salvo Pro builds. Each makefile (and hence, each project) defines the `SYSY` symbol.

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] Additionally, WinAVR is available at http://winavr.sourceforge.net/.

[2] Another shell is the Cygwin (http://www.cygwin.com/) `bash` shell for Windows, and is the one used here.