PUMPKIN™
REAL-TIME SOFTWARE

AN-24
Application Note

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

Building a Salvo Application with ImageCraft's ICCAVR Development Tools

Introduction

This Application Note explains how to use ImageCraft's (http://www.imagecraft.com/) ICCAVR Development Tools to create a multitasking Salvo application for Atmel's (http://www.atmel.com/) AVR and MegaAVR MCUs.

We will show you how to build the Salvo application contained in `\salvo\ex\ex1\main.c` for an AT90S8515 using ICCAVR v6.28c and AVRStudio® v4.06. 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 the ImageCraft ICCAVR Embedded Tools. Familiarize yourself with the ICCAVR IDE.

Related Documents

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

Salvo User Manual
Salvo Compiler Reference Manual RM-ICCAVR

Creating and Configuring a New Project

Create a new ICCAVR project under Project → New. Navigate to your working directory (in this case we've chosen `c:\temp`) and create a project named `myex1.prj`:

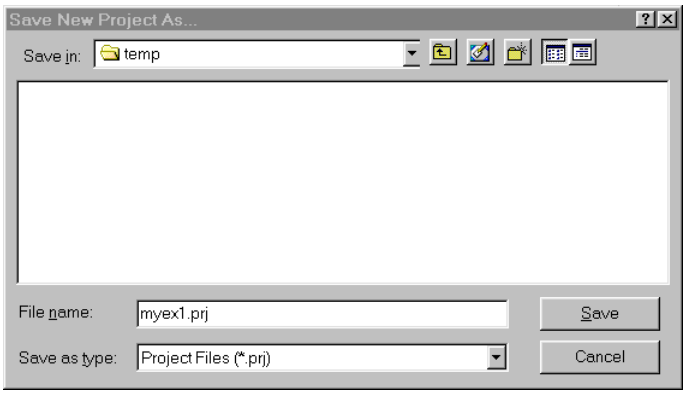

Figure 1: Creating the New Project

Click Save to continue. The ICCAVR IDE will automatically save the project whenever you close it.

In order to manage your project effectively, we recommend that you create a set of folders for your project. They are:

Listings
Salvo Configuration File
Salvo Help Files
Salvo Libraries
Salvo Source Files
Sources

For each folder,[1] choose Add Folder… by right-clicking in the Project window, enter the desired name under Folder Name and click OK.

Figure 2: Creating a Group

When finished, your Project Manager window should look like this:

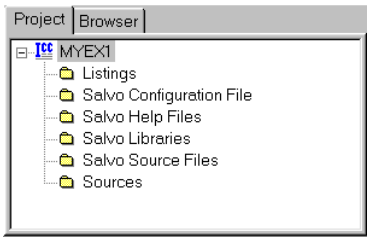

Figure 3: Project Manager Window with Folders

Now let's setup the project's options for Salvo's pathnames, etc. Open the Compiler Options window by selecting Project → Options… → Paths. Add the project's own include path and `\salvo\inc\`,[2] separated by semicolons:

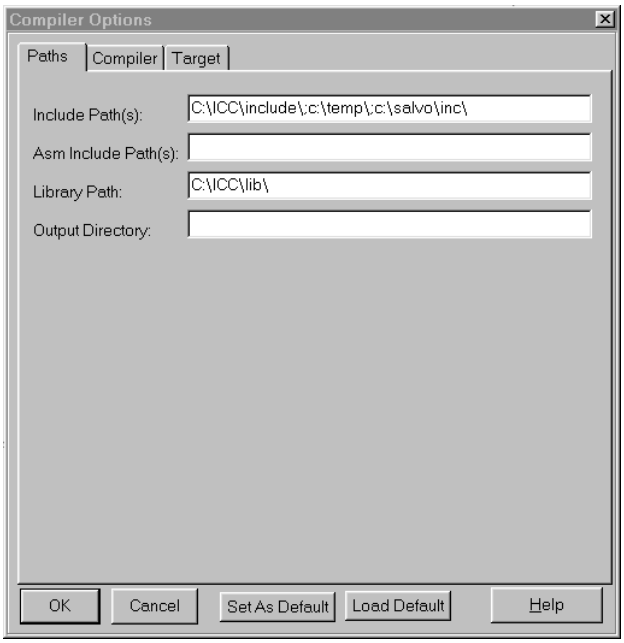

Figure 4: ICCAVR Settings – Project Include Paths

Next, define any symbols[3] you may need for your project in the Compiler Options window by selecting Compiler and entering the symbols under Macro Define(s):

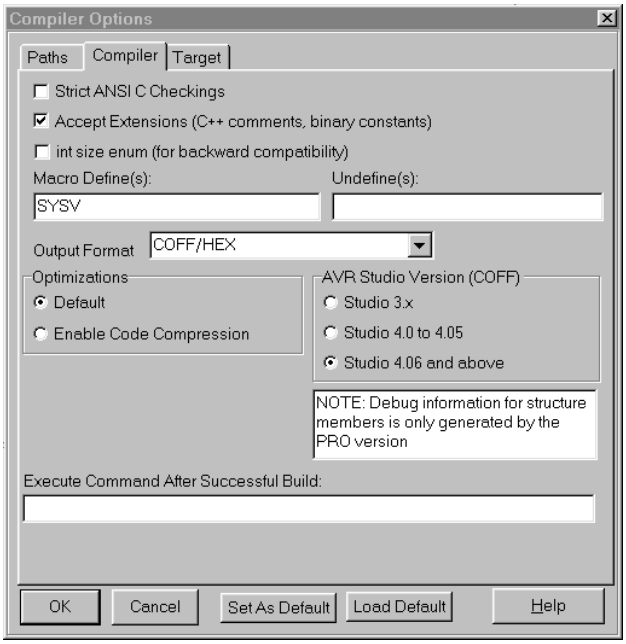

Figure 5: ICCAVR Options – Project Compiler Settings

Be sure to select the COFF format that's appropriate for your version of AVRStudio as shown in Figure 5. If you have ICCAVR Professional, you can select Optimizations → Enable Code Compression.

Lastly, in the Compiler Options window under Target, select the appropriate Device Configuration:

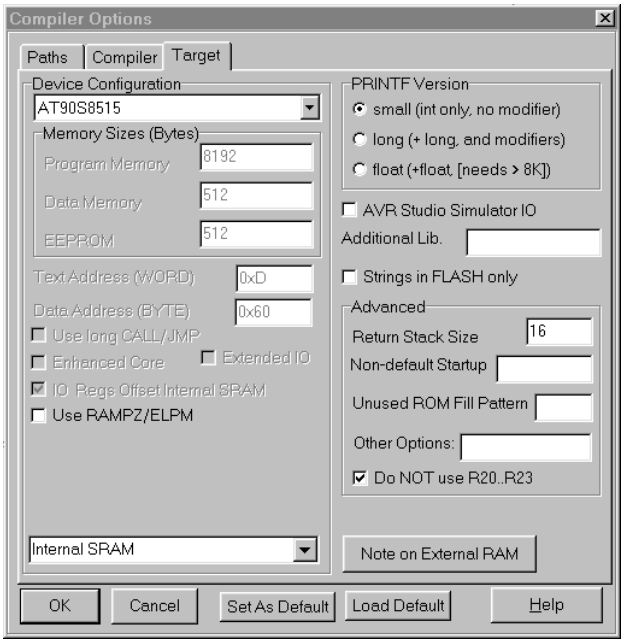

Figure 6: ICCAVR Options – Project Target Settings

Depending on the complexity of your application, you may need to increase the Return Stack Size under Advanced. You can select Do NOT use R20.R23 or leave it unselected – Salvo is compatible with both settings.

Click OK to finish setting your project's options.

Adding your Source File(s) to the Project

Now it's time to add files to your project. In the Project Manager window, select the Sources folder, right-click to choose Add Files…, choose Files of type: Source Files (*.c, *.s, *.h), navigate to your project's directory, select your `main.c` and click Open. Your Add Files… window should look like this:

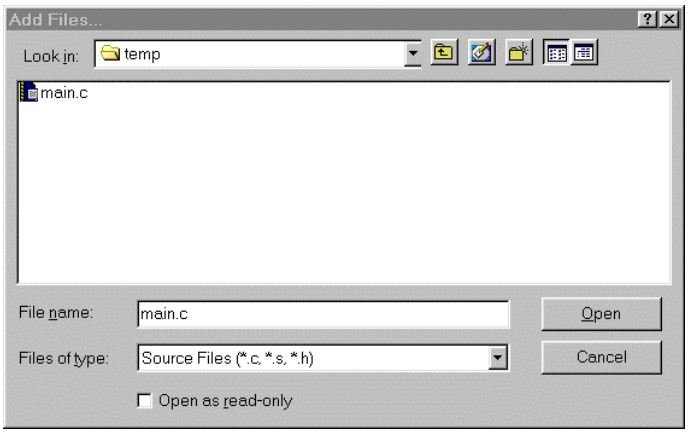

Figure 7: Project Files Window

When finished, your Project Manager window should look like this:

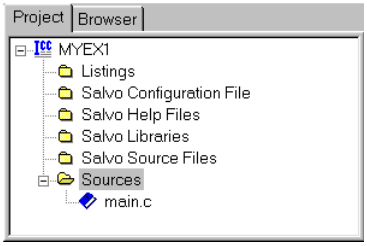

Figure 8: Project Manager Window with Project-Specific Source Files

Adding Salvo-specific Files to the Project

Now it's time to add the 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*, a fully-featured Salvo freeware library for the AT90S8515 for use with ICCAVR is `libsficcavr-a.a`.[4] Select Project → Options… → Target, and under Additional Lib. enter `sficcavr-a`:

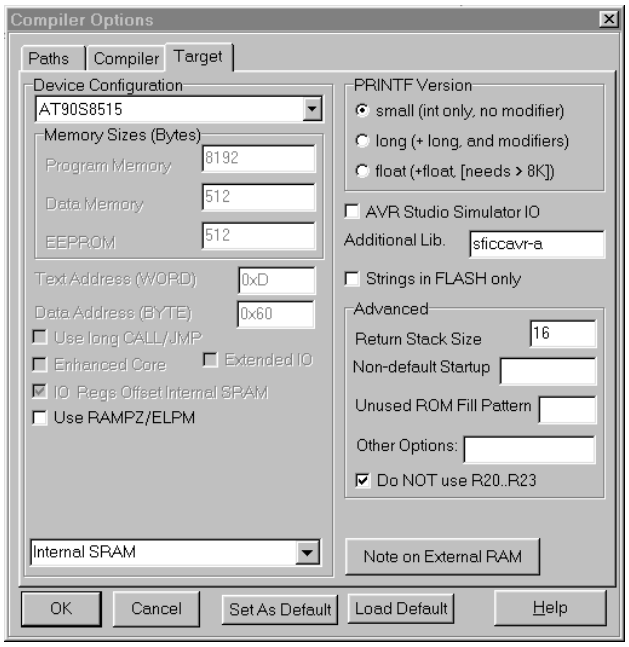

Figure 9: Adding the Library to the Project

Click OK when you are finished. You can find more information on Salvo libraries in the *Salvo User Manual* and in the *Salvo Compiler Reference Manual RM-ICCAVR*.

Adding Salvo's mem.c

Salvo library builds also require Salvo's `mem.c` source file as part of each project. In the Project Manager window, select the Salvo Sources folder, right-click to choose Add Files…, choose Files of type: Source Files (*.c, *.s, *.h), navigate to `\salvo\src`, select `mem.c` and click Open. Your Add Files… window should look like this:

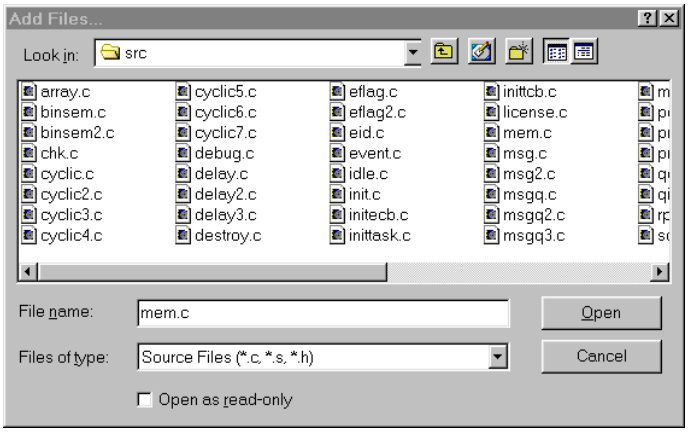

Figure 10: Add Files … Window

The salvocfg.h Header File

You will also need a `salvocfg.h` file for this project. To use the library selected in Figure 9, your `salvocfg.h` should contain only:

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

Listing 1: salvocfg.h for a Library Build

Create this file and save it in your project directory, e.g. `c:\temp\salvocfg.h`. For convenience, add it to your project's Salvo Configuration File folder:

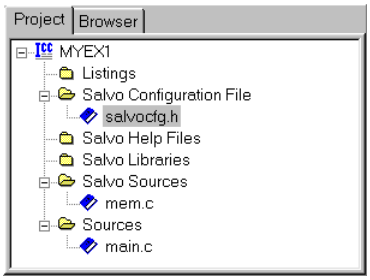

Figure 11: Project Manager Window for Library Build

Proceed to *Building the Project*, below.

Adding Salvo Source Files

If you have a Salvo distribution that contains source files, 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                mem.c
delay.c                 porticcavr.s
event.c                 qins.c
idle.c                     sched.c
init.c                  timer.c
inittask.c
```

In the Project Manager window, select the Salvo Sources folder, right-click to choose Add Files…, choose Files of type: Source Files (*.c, *.s, *.h), navigate to the `\salvo\src` directory and select[5] the `*.c` files listed above. Your Add Files… window should look like this:

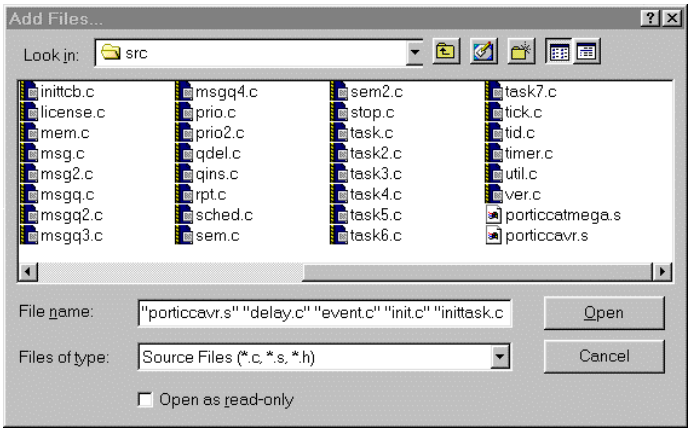

Figure 12: Adding Salvo Source Files to the Project

Click Open when finished.

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 1, 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 2: salvocfg.h for a Source Code Build

Create this file and save it in your project directory, e.g. `c:\temp\salvocfg.h`. For convenience, add it to your project's Salvo Configuration File folder:

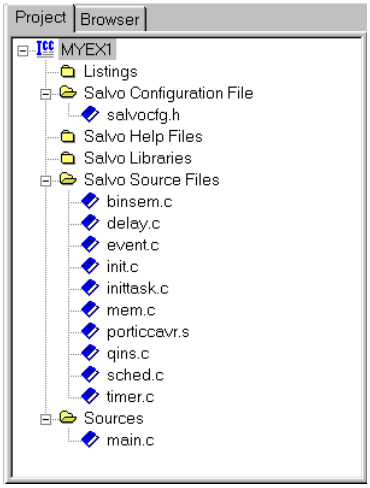

Figure 13: Complete Project Manager Window for a Source-Code Build

Tip The advantage of placing the various project files in the groups shown above is that you can quickly navigate to them and open them for viewing, editing, etc.

Building the Project

For a successful compile, your project must also include a header file (e.g. `#include <io8515v.h>`) for the particular chip you are using. Normally, this is included in each of your source files (e.g. `main.c`), or in a header file that's included in each of your source files (e.g. `main.h`).

With everything in place, you can now build the project using Project → Make Project or Project → Rebuild All. The IDE's status window will reflect the ICCAVR command lines:

```
C:\ICC\BIN\imakew -f myex1.mak
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\binsem.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\delay.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\event.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\init.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\idle.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\inittask.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\mem.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  -Wa-g C:\salvo\src\porticcavr.s
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\qins.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\sched.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\salvo\src\timer.c
    iccavr -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYSV  -l -g -Wa-W -Wf-r20_23  C:\temp\main.c
    iccavr -o myex1 -LC:\ICC\lib\ -g -Wl-W
-bfunc_lit:0x1a.0x2000 -dram_end:0x25f -bdata:0x60.0x25f
-dhwstk_size:16 -beeprom:1.512 -fihx_coff -S2 @myex1.lk
-lsficcavr-a -lcavrgr
Device 22% full.
Done.
```

Listing 3: Build Results for A Successful Source-Code Build

The map (`*.mp`) file located in the project's directory contains address, symbol and other useful information:[6]

```
Area                                Addr   Size   Decimal Bytes (Attributes)
--------------------------------    ----   ----   ------- ----- ------------
                         func_lit   001A   000C =     12. bytes (rel,con,rom)

      Addr  Global Symbol
     -----  --------------------------------
      001A  __func_lit_start
      0026  __func_lit_end

Area                                Addr   Size   Decimal Bytes (Attributes)
--------------------------------    ----   ----   ------- ----- ------------
                             text   0026   070E =   1806. bytes (rel,con,rom)

      Addr  Global Symbol
     -----  --------------------------------
      0013  __start
      0013  __text_start
      0032  _exit
      0033  _OSCreateBinSem
      0047  _OSWaitBinSem
      0063  _OSSignalBinSem
      00B5  _OSDelay
      00D7  _OSWaitEvent
      0110  _OSInit
      0129  _OSCreateTask
      015C  _OSInitPrioTask
      0171  _OSDispatch
      0182  _OSCtxSw
      01A0  _OSInsPrioQ
      022B  _OSSched
      02EA  _OSTimer
      02F6  _Task1
      02FF  _Task1a
      0301  _Task2
      0305  _Task2a
      0310  _Task3
      0315  _Task3a
      031E  _main
      034B  _IntVector
      034F  pop_gset3x
      0350  popx
      035B  pop_gset4x
      035D  push_gset3x
      0360  push_gset4x
      0363  xicall
```

```
       036C  push_lset
       0383  pop_lset
       039A  __text_end

Area                              Addr   Size   Decimal Bytes (Attributes)
--------------------------------  ----   ----   ------- ----- ------------
                           vector   0000   000A =     10. bytes (abs,ovr,rom)

Area                              Addr   Size   Decimal Bytes (Attributes)
--------------------------------  ----   ----   ------- ----- ------------
                         salvoram   0060   0027 =     39. bytes (rel,con,ram)

       Addr  Global Symbol
      -----  --------------------------------
       0060  _OScTcbP
       0062  _OStcbArea
       0077  _OSeligQP
       0079  _OSecbArea
       007E  _OSsigQinP
       0080  _OSsigQoutP
       0082  _OSdelayQP
       0084  _OSlostTicks
       0085  _OSframeP
       0087  __salvoram_end

Files Linked      [ module(s) ]

C:\ICC\lib\crtAVR.o [ crtavr.s ]
binsem.o  [ binsem.c ]
delay.o   [ delay.c ]
event.o   [ event.c ]
idle.o    [ idle.c ]
init.o    [ init.c ]
inittask.o     [ inittask.c ]
mem.o     [ mem.c ]
porticcavr.o   [ porticcavr.s ]
qins.o    [ qins.c ]
sched.o   [ sched.c ]
timer.o   [ timer.c ]
main.o    [ main.c ]
<library> [ gpop3x.s, gpop4x.s, gpush3x.s, gpush4x.s, icall.s, lpush.s ]

User Global Definitions

ram_end = 0x25f
hwstk_size = 0x10

User Base Address Definitions

func_lit = 0x1a
data = 0x60
eeprom:1.512
```

Listing 4: Map File for a Source-Code Build

Note The projects supplied in the Salvo for Atmel AVR and MegaAVR distributions contain additional help files in each project's Salvo Help Files group.

Using the Browser

By selecting the COFF/HEX output format (see Figure 5), ICCAVR will build debug information for your project that can be used by the IDE's Browser:

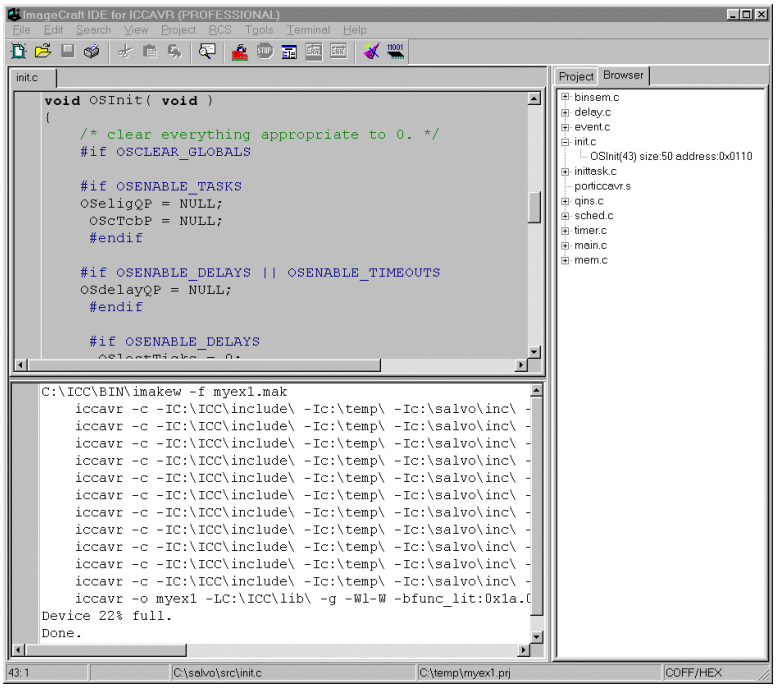

Figure 14: Browsing at the Source-Code Level

By double-clicking in the Browser on the function of interest, the source code that contains the function will be displayed in the Editors window. This works with any source code (project- or Salvo-specific), and also with the `i`-option Salvo libraries that include debugging information.

Testing the Application

AVRStudio Simulator

You can test and debug this application using the AVRStudio simulator. Launch AVRStudio, then select File → Open File…, navigate to your working directory, select the COFF output file `myex1.cof`, and click on Open.

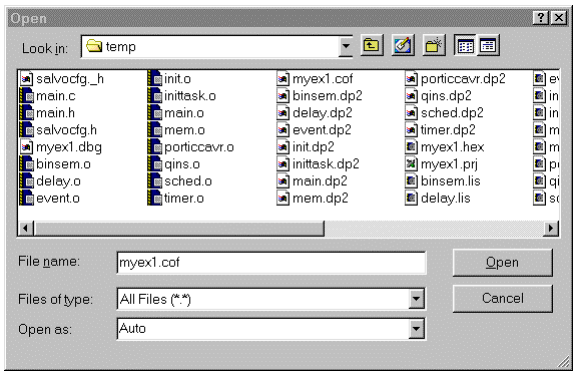

Figure 15: Opening the COFF File

You will then be asked to select a debug platform and device:

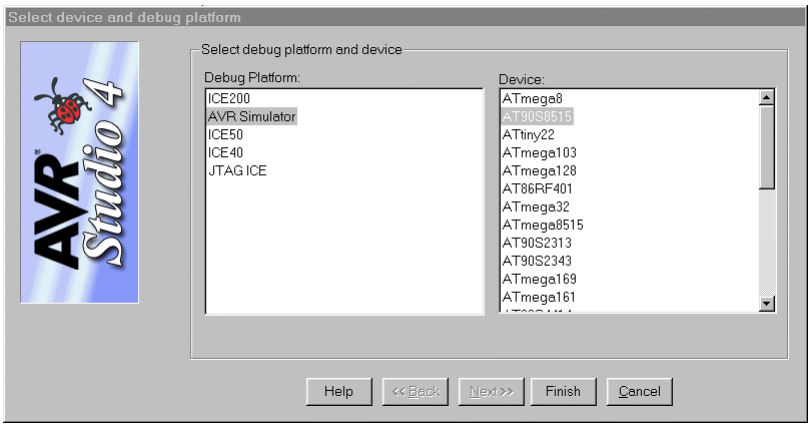

Figure 16: Selecting the Debug Platform and Device

After you've selected the appropriate device, click on Finish. AVRStudio will configure and load the simulator with your Salvo application.

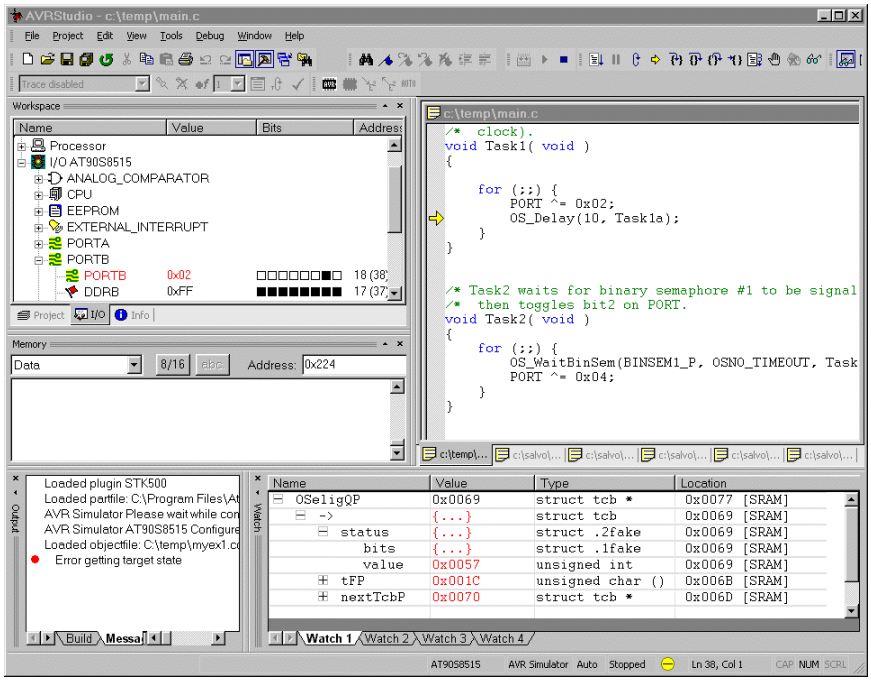

Figure 17: Running Your Application in the AVRStudio Simulator

Salvo applications are fully functional in the AVRStudio simulator. You can set breakpoints at any place in your application, watch variables, count instruction cycles, override register flags, etc.

Tip When debugging with the AVRStudio simulator, the project's map (`*.mp`) file and listing (`*.lst`) files are very useful because they list the addresses of functions and variables in ROM and RAM. This information can be used in the monitor program to set breakpoints, display memory, better understand trace results, etc.

Note ICCAVR can create generate debugging info via the `-g` command-line option. Only applications built from the Salvo source code or a Salvo Pro library enable you to step through Salvo services (e.g. `OSCreateBinSem()`) at the source code level when using an external debugger. Regardless of how you build your Salvo application, you can always step through your own C and assembly code with ICCAVR's output.

STK500 Flash Microcontroller Starter Kit

Alternatively, you can download your project to the STK500 using Tools → STK500/… → STK500/…, Select Program → Flash → Use Current Simulator/Emulator FLASH Memory and click on Program.

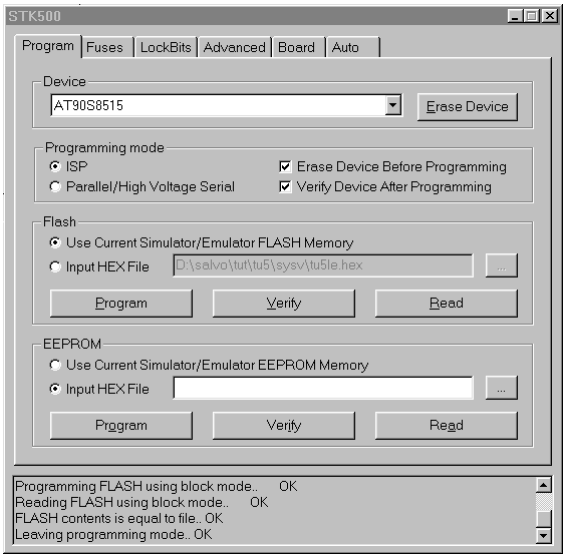

Figure 18: Downloading the Application to STK500 in AVRstudio

AVRStudio will first program the STK500's FLASH, then verify it, and finally leave the programming mode, enabling the Salvo application to run on the STK500.

Troubleshooting

Cannot find and/or read include file(s)

If you fail to add `\salvo\inc` to the project's include paths (see Figure 4) the compiler will generate an error like this one:

```
!E D:\salvo\src\event.c(30): Could not find
include file "salvo.h"
```

Figure 19: Compiler Error due to Missing \salvo\inc Include Path

By adding `\salvo\inc` to the project's include path, you enable the compiler to find the main Salvo header file `salvo.h`, as well as other included Salvo header files.

If you fail to add the project's own directory to the project's include paths (see Figure 4) the compiler will generate an error like this one:

```
!E c:/salvo/inc/salvo.h(292):
D:\salvo\src\delay.c(27): Could not find
include file "salvocfg.h"
```

Figure 20: Compiler Error due to Missing Project Include Path

By adding the project's own directory to the project's include path, you enable the compiler to find the project-specific header file `salvocfg.h`.

Odd Behavior After Changing Processor Type

Whenever you select or change the processor type in ICCAVR, you must ensure that you've made the corresponding changes in AVRStudio. Additionally, your Salvo build configuration must be appropriate for the AVR or MegaAVR target you've chosen.

Cannot Resolve Location of Salvo Source Files

The Salvo Pro libraries with embedded debug information (`i`-option) reference the salvo source files in their default location, `\salvo\src`. If you have placed these files in an alternate location and you want to use debugging information, you can edit the library files and change the pathnames that reference Salvo source files. An automated method (e.g. a perl script) is recommended.

Example Projects

Example projects for the ICCAVR Development Tools are found in the `\salvo\tut\tu1-6\sysv` directories. The include path for each of these projects includes `\salvo\tut\tu1\sysv`, and each project defines the `SYSV` symbol.

Complete projects using Salvo freeware libraries are contained in the project files `\salvo\tut\tu1-6\sysv\tu1-6lite.prj`. These projects also define the `MAKE_WITH_FREE_LIB` symbol.

Complete projects using Salvo standard libraries are contained in the project files `\salvo\tut\tu1-6\sysv\tu1-6le.prj`. These projects also define the `MAKE_WITH_STD_LIB` symbol.

Complete projects using Salvo standard libraries with embedded debugging information are contained in the project files `\salvo\tut\tu1-6\sysv\tu1-6prolib.prj`. These projects also define the `MAKE_WITH_STD_LIB` symbol.

Complete projects using Salvo source code are contained in the project files `\salvo\tut\tu1-6\sysv\tu1-6pro.prj`. These projects also define the `MAKE_WITH_SOURCE` symbol.

---

1 Since folders cannot be deleted, you should rename the default `Files`, `Headers` and `Documents` folders to `Listings`, `Salvo Configuration File` and `Salvo Help Files`, respectively.

2 ICCAVR also supports pathnames relative to the project's home directory. Using relative pathnames is recommended, as it makes a project much more portable.

3 This Salvo project supports a wide variety of targets and compilers. For use with ICCAVR Development Tools, it requires the `SYSV` defined symbol, as well as the symbols `MAKE_WITH_???_LIB` for library builds. When you write your own projects, you may not require any symbols.

4 This Salvo Lite library contains all of Salvo's basic functionality. The corresponding Salvo LE and Pro libraries are `libsliccavr-a.a` and `libsliccavria.a`, respectively.

5 You can Ctrl-select multiple files at once.

6 We recommend that you add the project's map file to your project's `Listings` group.