PUMPKIN™
REAL-TIME SOFTWARE

AN-19
Application Note

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

Building a Salvo Application with ImageCraft's ICC11 Development Tools

Introduction

This Application Note explains how to use ImageCraft's (http://www.imagecraft.com/) ICC11 Development Tools to create a multitasking Salvo application for Motorola (http://www.motorola.com/) M68HC11 microcontrollers.

We will show you how to build the Salvo application contained in `\salvo\ex\ex1\main.c` for a M68HC11 using ICC11 v6.03. 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 ICC11 Embedded Tools. Familiarize yourself with the ICC11 IDE.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with ImageCraft's ICC11 Development Tools:

Salvo User Manual
Salvo Compiler Reference Manual RM-ICC11

Creating and Configuring a New Project

Create a new ICC11 project under Project → New. Navigate to your working directory (in this case we've chosen `c:\temp`) and create a project named `myex1.prj`:

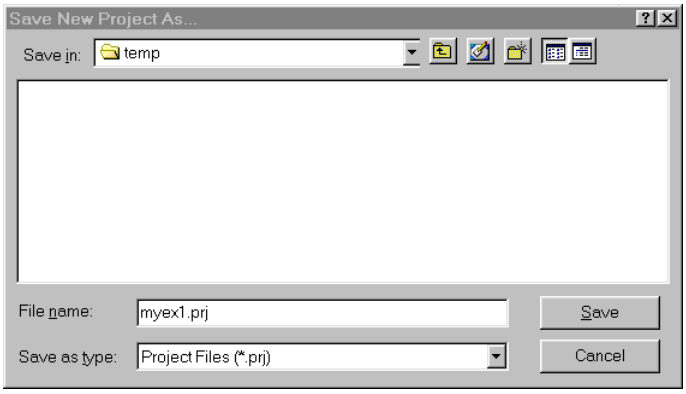

Figure 1: Creating the New Project

Click Save to continue. The ICC11 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 Sources
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:

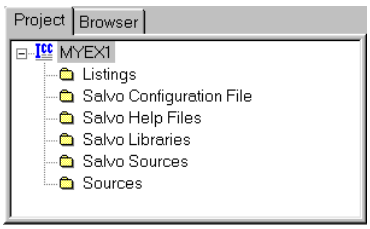

Figure 3: Project Manager Window with Folders

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:

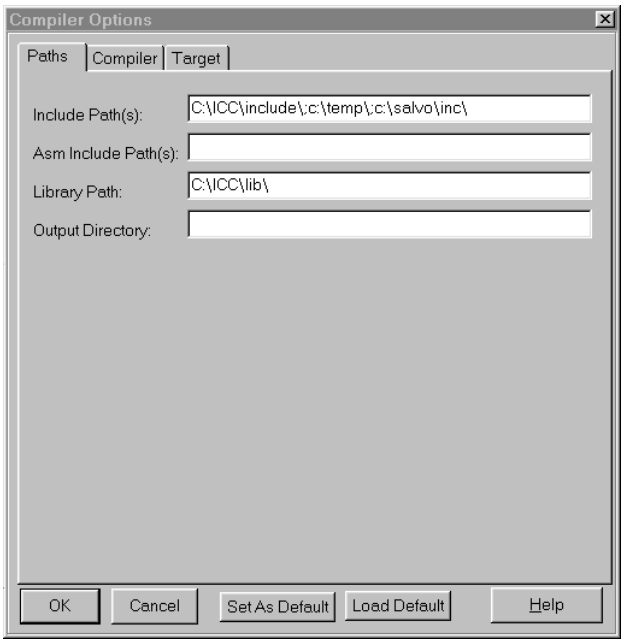

Figure 4: ICC11 Settings – Project Include Paths

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):

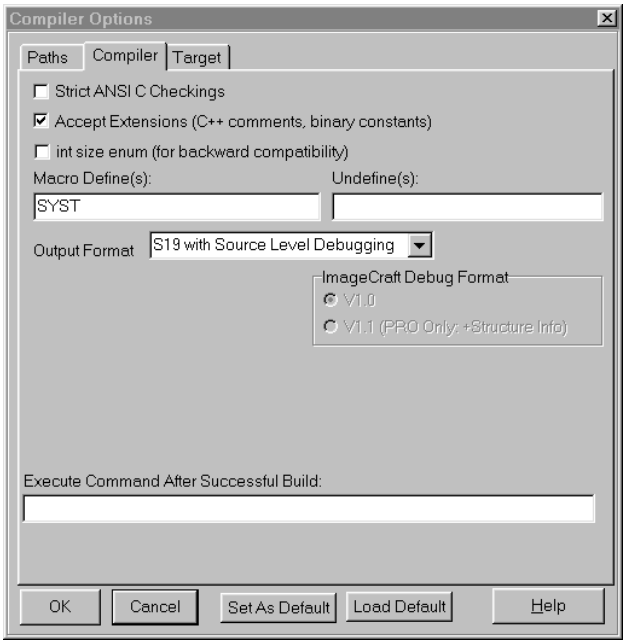

Figure 5: ICC11 Options – Project Compiler Settings

The S19 with Source Level Debugging output format is recommended over the simple Motorola S19 output format because the former will enable you to browse the project's source code in the IDE's Browser.

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

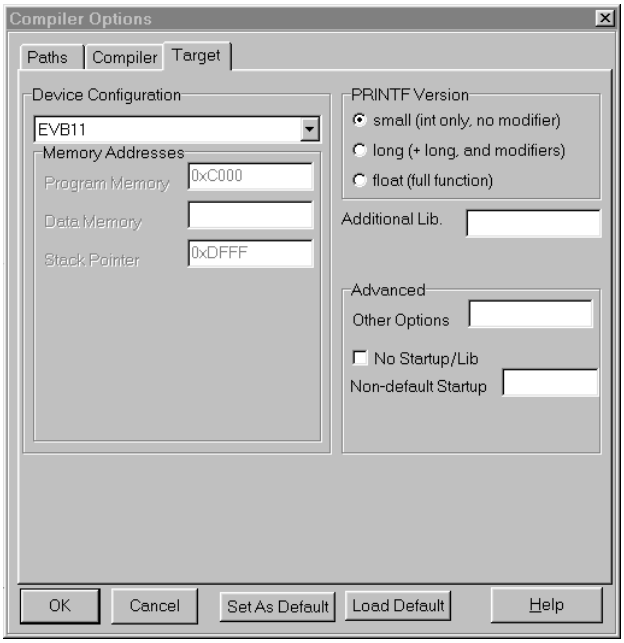

Figure 6: ICC11 Options – Project Target Settings

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:

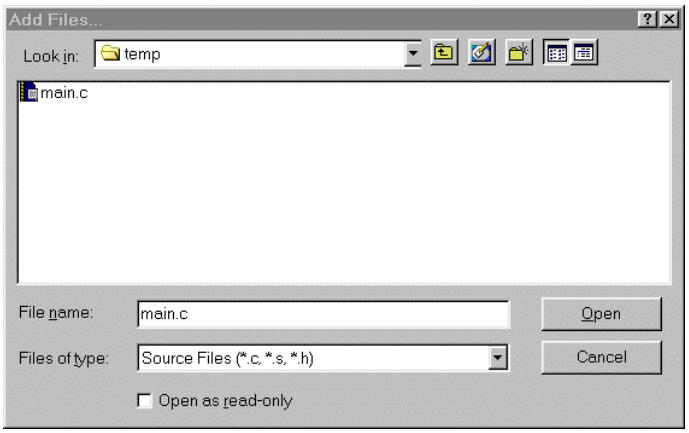

Figure 7: Project Files Window

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

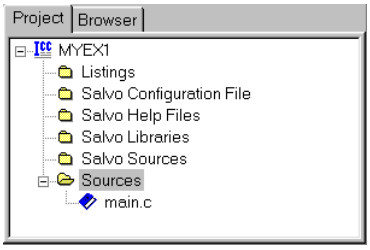

Figure 8: Project Manager Window with Project-Specific Source Files

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 M68HC11 for use with ICC11 is `libsficc11-a.a`.[4] Select Project → Options… → Target, and under Additional Lib. enter `sficc11-a`:

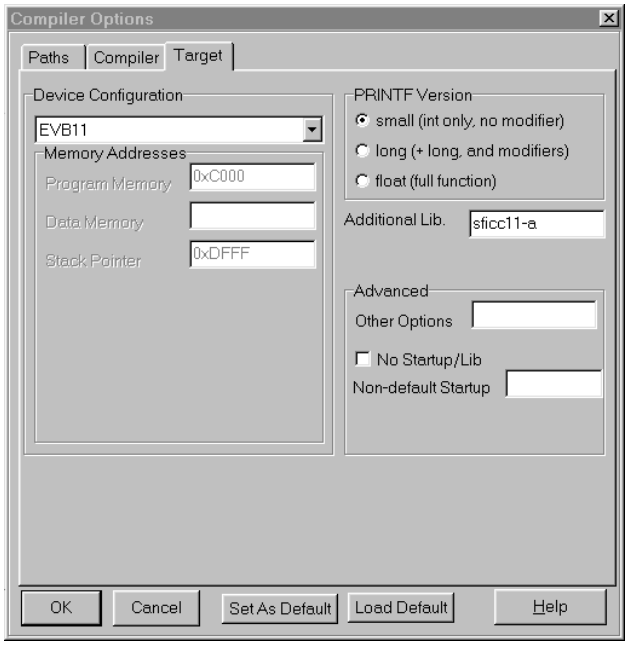

Figure 9: Adding the Library to the Project

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-ICC11*.

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:

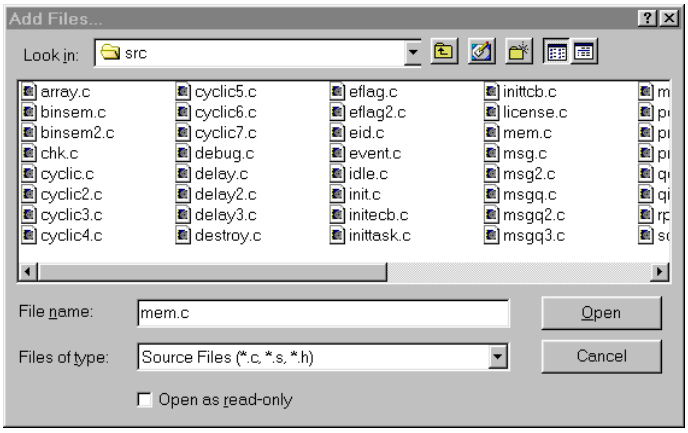

Figure 10: Add Files … Window

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 OSCOMPILER          OSIMAGECRAFT
#define OSTARGET            OSM68HC11
#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:

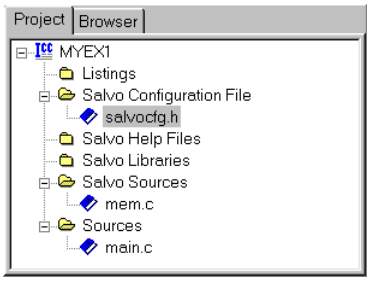

Figure 11: Project Manager Window for Library Build

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             porticc11.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:

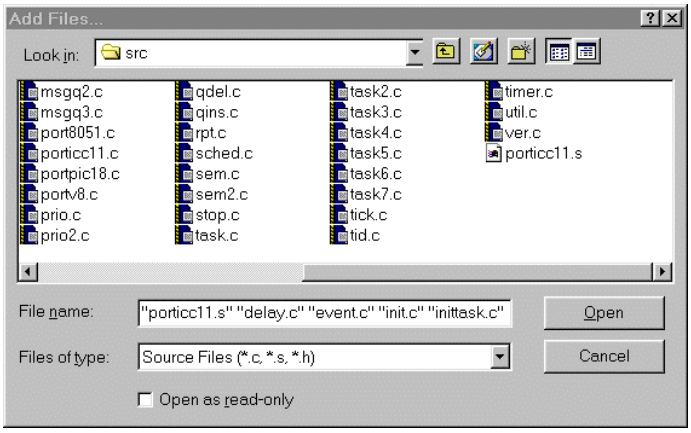

Figure 12: Adding Salvo Source Files to the Project

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 OSCOMPILER                  OSIMAGECRAFT
#define OSTARGET                    OSM68HC11
#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:

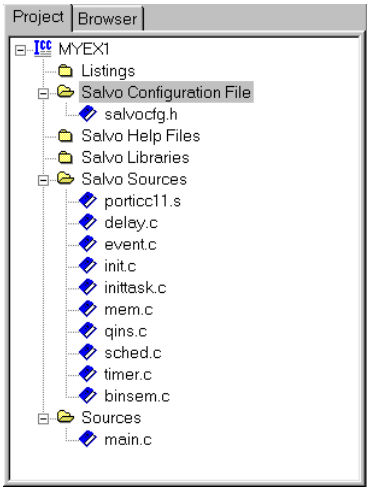

Figure 13: Complete Project Manager Window for a Source-Code Build

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 <hc11.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`).[6]

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 ICC11 command lines:

```
C:\ICC\BIN\imakew -f myex1.mak
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l   D:\salvo\src\porticc11.s
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  D:\salvo\src\delay.c
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  D:\salvo\src\event.c
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  D:\salvo\src\init.c
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  D:\salvo\src\inittask.c
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  D:\salvo\src\mem.c
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  D:\salvo\src\qins.c
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  D:\salvo\src\sched.c
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  D:\salvo\src\timer.c
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  D:\salvo\src\binsem.c
    icc11w -c -IC:\ICC\include\ -Ic:\temp\ -Ic:\salvo\inc\ -e
-DSYST  -l  C:\temp\main.c
    icc11w -o myex1 -LC:\ICC\lib\ -m -btext:0xC000
-dinit_sp:0xDFFF -fmots19 @myex1.lk  -lsficc11-a
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:[7]

```
Area                                Addr   Size   Decimal Bytes (Attributes)
--------------------------------   ----   ----   ------- ----- ------------
                             text   C000   0729 =   1833. bytes (rel,con)

      Addr  Global Symbol
     -----  --------------------------------
      C000  __start
      C02D  _exit
      C02F  _OSDispatch
      C079  _OSCtxSw
      C0C6  _OSDelay
      C123  _OSWaitEvent
      C1AB  _OSInit
      C1DA  _OSCreateTask
      C260  _OSInitPrioTask
      C2A4  _OSInsPrioQ
      C3B7  _OSSched
      C4F5  _OSTimer
      C515  _OSCreateBinSem
      C546  _OSWaitBinSem
      C57B  _OSSignalBinSem
      C5F3  _Task1
      C607  _Task2
      C623  _Task3
      C63D  _OSIdlingHook
      C63E  _main
      C6AD  _intVector
      C6C1  __enterb
      C6E8  __movspb
      C70A  _setbaud
      C722  __HC11Setup
      C729  __text_end

Area                                Addr   Size   Decimal Bytes (Attributes)
--------------------------------   ----   ----   ------- ----- ------------
                              bss   C729   0027 =     39. bytes (rel,con)

      Addr  Global Symbol
     -----  --------------------------------
      C729  __bss_start
      C729  _OSframeP
      C72B  _OSglStat
      C72C  _OSdelayQP
      C72E  _OSsigQoutP
      C730  _OSsigQinP
      C732  _OSecbArea
      C737  _OSeligQP
      C739  _OStcbArea
      C74E  _OScTcbP
      C750  __bss_end

Area                                Addr   Size   Decimal Bytes (Attributes)
--------------------------------   ----   ----   ------- ----- ------------
                           memory   0000   0100 =    256. bytes (abs,ovr)

Files Linked      [ module(s) ]

C:\ICC\lib\crt11.o  [  crt11.s ]
porticc11.o       [ porticc1 ]
delay.o           [  delay.c ]
event.o           [  event.c ]
init.o            [   init.c ]
inittask.o        [ inittask ]
mem.o             [    mem.c ]
qins.o            [   qins.c ]
sched.o           [  sched.c ]
timer.o           [  timer.c ]
binsem.o          [ binsem.c ]
main.o            [   main.c ]
<library>         [   byte.s, serial.s,  setup.s ]

User Global Definitions

init_sp = 0xdfff

User Base Address Definitions

text = 0xc000
```

Listing 4: Map File for a Source-Code Build

Note The ICC11 projects supplied in the Salvo for Motorola M68HCxx distributions contain additional help files in each project's Salvo Help Files group.

Using the Browser

By selecting the S19 with Source Level Debugging output format (see Figure 5), ICC11 will build debug information for your project that can be used by the IDE's Browser:

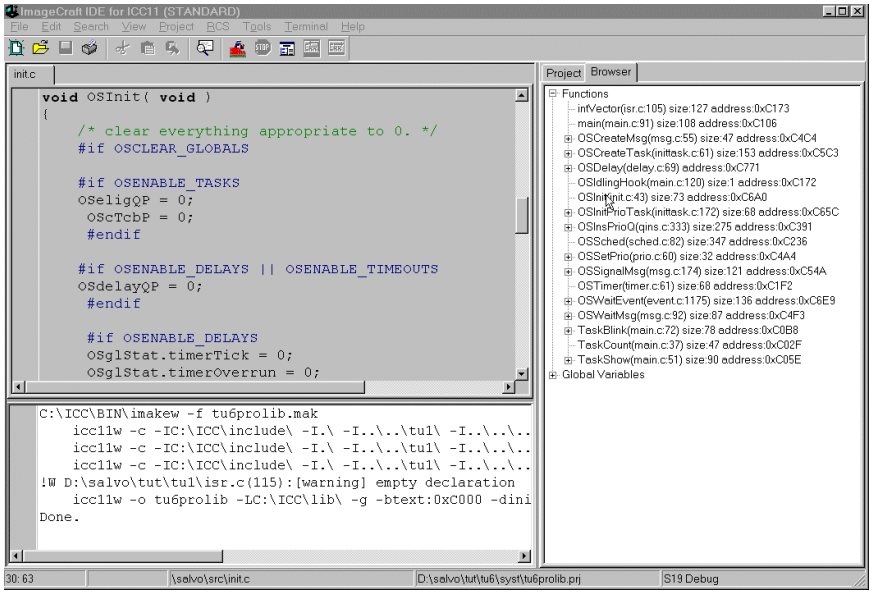

Figure 14: Browsing at the Source-Code Level

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

The ICC11 IDE does not provide an integrated debugger. It does have a built-in terminal for debugging with terminal-based development and debugging systems like the Motorola M68HC11 EVB, EVM and EVS. You can test and debug your application(s) using whatever debugging hardware is at your disposal.

To test a Salvo application with an M68HC11 EVB, build the application, then choose Terminal → Show Terminal Window. Click Open Com Port, click Browse…, navigate to the project's directory and choose the project's S19 file[8] (in this case, `myex1.s19`). In the EVB's BUFFALO[9] monitor program, enter `load t` [Enter]. In the Terminal window, click Download!. When finished, BUFFALO will display `done`. In BUFFALO, enter `rm` [Enter], `C000`[10] [Enter], and `go` [Enter]. Your Salvo application will start running.

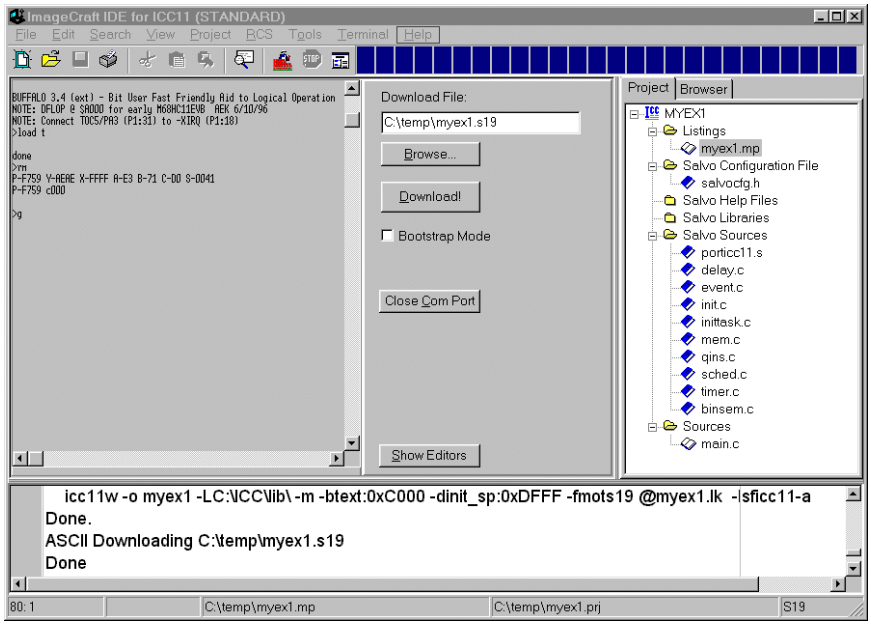

Figure 15: Testing a Salvo Application in ICC11

Tip When debugging with a monitor program, 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 ICC11 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 ICC11's output.

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 16: 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 17: 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`.

Cannot Trace (M68HC11 EVB with BUFFALO)

First, make sure you do not have a breakpoint set to the instruction you are trying to trace. This will happen if you set a breakpoint and then trace after reaching that breakpoint. Remove the breakpoint before tracing.

Also, because of the way that BUFFALO traces instructions, you will have problems tracing if you have periodic interrupts enabled. For example, the M68HC11's RTI interrupt – which you might use as the periodic interrupt that calls `OSTimer()` – will prevent you from tracing properly. Therefore you should disable interrupts before tracing, and re-enable them thereafter (i.e. just before using the GO or PROCEED commands). Use BUFFALO's RM command to set/clear the CCR's `I` bit (bit 4 of 7) to disable/enable interrupts, respectively.

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 ICC11 Development Tools are found in the `\salvo\tut\tu1-6\syst` directories. The include path for each of these projects includes `\salvo\tut\tu1\syst`, and each project defines the `SYST` symbol.

Complete projects using Salvo freeware libraries are contained in the project files `\salvo\tut\tu1-6\syst\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\syst\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\syst\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\syst\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 ICC11 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 ICC11 Development Tools, it requires the `SYST` defined symbol, as well as the symbols `MAKE_WITH_FREE_LIB` or `MAKE_WITH_STD_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 `libslicc11-a.a` and `libslicc11ia.a`, respectively.

5 You can Ctrl-select multiple files at once.

6 Since ICC11 has only a single header file – `hc11.h` – and since it is included via `salvo.h`, technically it's unnecessary to include it in any source files that include `salvo.h`.

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

8 Motorola S-record, a hex file format.

9 Bit User Fast Friendly Aid to Logical Operation, the default monitor program from Motorola on the EVB, EVM and EVS.

10 `C000h` is the default location for the start of User RAM in the EVB.