PUMPKIN™
REAL-TIME SOFTWARE

AN-21
Application Note

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

Building a Salvo Application with TI's Code Composer Studio 'C2000

Introduction

This Application Note explains how to use TI's (http://www.ti.com/) Code Composer (CC) and Code Composer Studio (CCS) Development Tools to create a multitasking Salvo application for TI's TMS320C2000 DSPs.

We will show you how to build the Salvo application contained in `\salvo\ex\ex1\main.c` for a TMS320C2812 using CCS 'C2000 v2.0. For more information on how to write a Salvo application, please see the *Salvo User Manual*.

Note Since the two compilers and their IDEs are essentially identical, Code Composer Studio will be used throughout this manual to refer to both products. Where necessary, differences will be identified.

Before You Begin

If you have not already done so, install CCS 'C2000. You will need to run the CCS Setup program to properly set the System Configuration.[1] Familiarize yourself with CCS. More information on TI's TMS320C2000 family of DSPs and the associated tools is available at http://www.ti.com/.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with TI's Code Composer Studio Development Tools:

Salvo User Manual
Salvo Compiler Reference Manual RM-CCS2000

Creating and Configuring a New Project

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

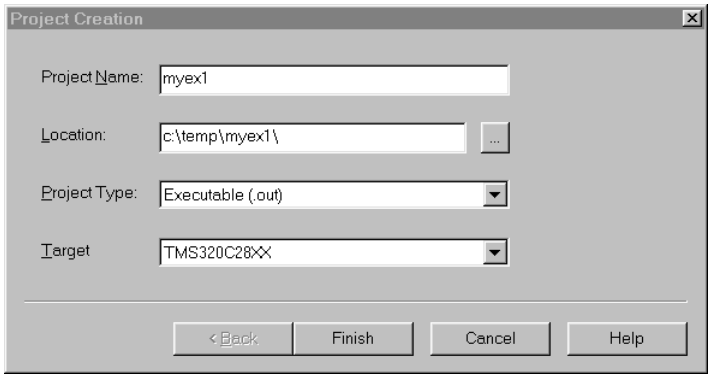

Figure 1: Creating the New Project

Choose the appropriate Project Type and Target and click Finish to continue. Your project window should look like this:

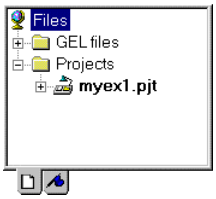

Figure 2: Project Window with Folders

Now let's setup the project's options for Salvo's pathnames, etc. Access the project's compiler options by selecting Project → Build Options… and the Compiler tab. Under Category select Preprocessor and add the project's own include path and `\salvo\inc\` to Include Search Path. Also, define any symbols[2] you may need for your project in Define Symbols. Use semicolons to separate entries.

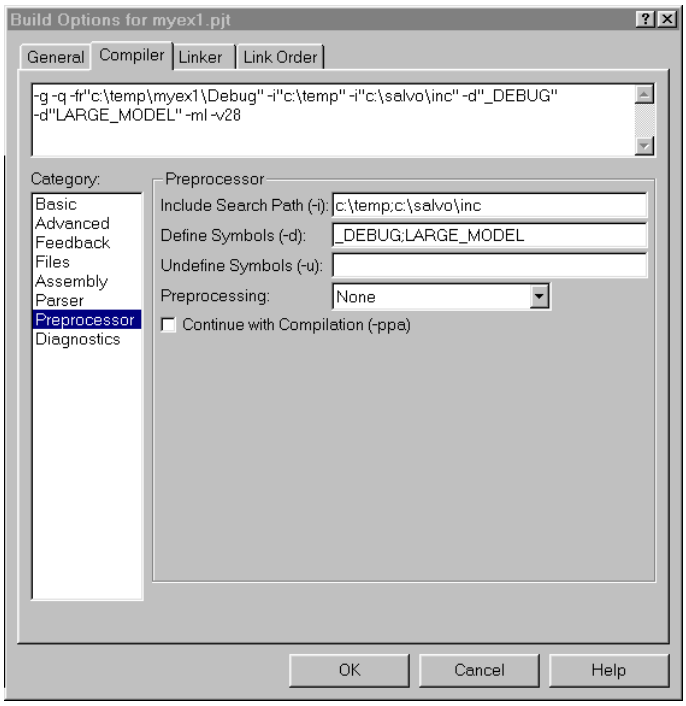

Figure 3: CCS 'C2000 Build Options – Include Search Path and Define Symbols

If you wish to generate a map file, select the Linker tab, select Basic under Category and enter the map file name in Map Filename:

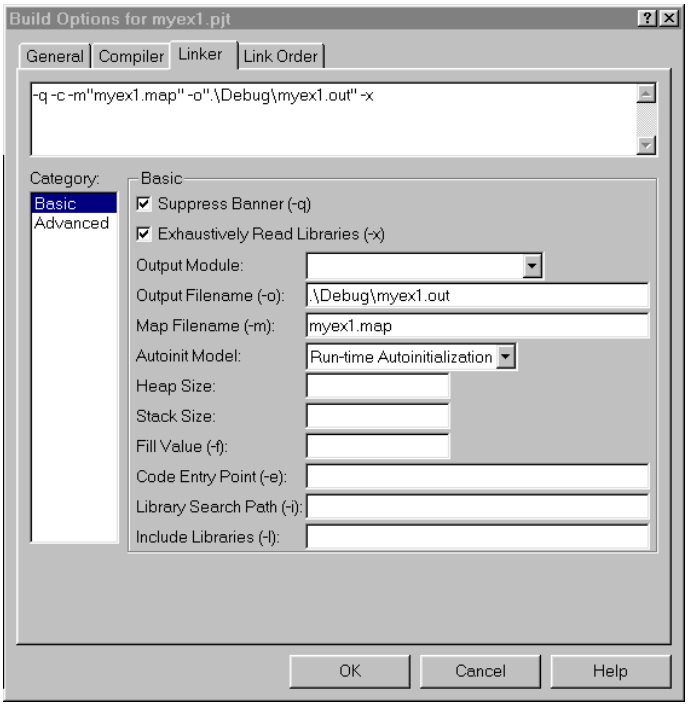

Figure 4: CCS 'C2000 Build Options – Map Filename

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. Select Project → Add Files to Project, choose Files of type: C Source Files (*.c, *.cc), navigate to your project's directory and select your `main.c`. Your Add Files to Project window should look like this:

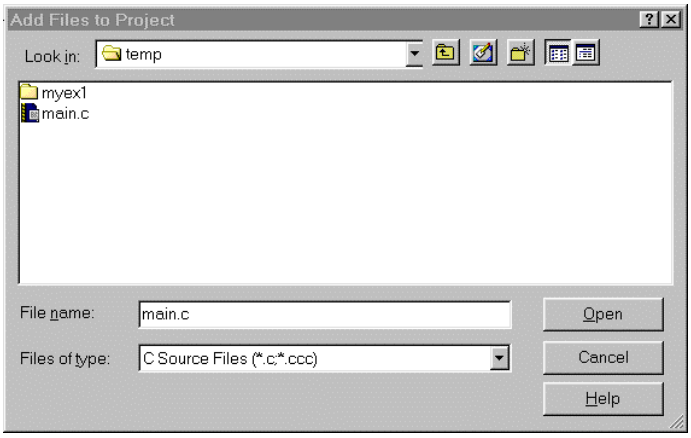

Figure 5: Add Files to Project Window

When finished, click Open, and your project window should look like this after expanding the project's folders:

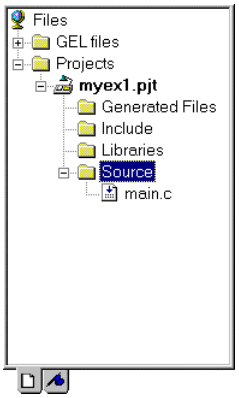

Figure 6: Project Manager Window with Project-Specific Source Files

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 TMS320C2812 for use with CCS 'C2000 is `sftic28xl-a.lib`.[3] Select Project → Add Files to Project, choose Files of type: Object and Library Files (*.o*, *.l*), navigate to `\salvo\lib` and select the library:

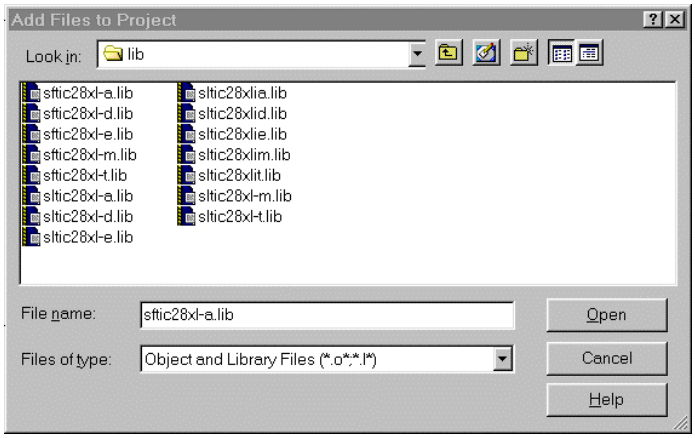

Figure 7: Adding the Library to the Project

Click Open 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-CCS2000*.

Adding Salvo's mem.c

Salvo library builds also require Salvo's `mem.c` source file as part of each project. Select Project → Add Files to Project, choose Files of type: C Source Files (*.c, *.cc), navigate to `\salvo\src` and select `mem.c`. Your Add Files to Project window should look like this:

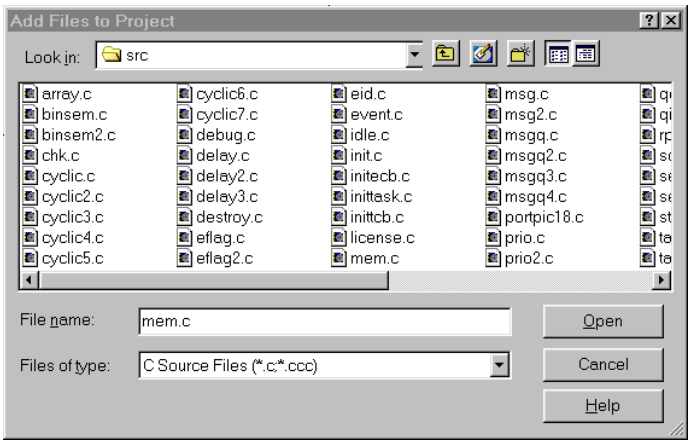

Figure 8: Add Files to Project Window

The salvocfg.h Header File

You will also need a `salvocfg.h` file for this project. To use the library selected in Figure 7, 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`.

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                 porttic28x.asm
event.c                 qins.c
idle.c                  sched.c
init.c                  timer.c
inittask.c
```

Select Project → Add Files to Project, choose Files of type: C Source Files (*.c, *.cc), navigate to `\salvo\src` and select[4] the `*.c` files listed above. Your Add Files to Project window should look like this:

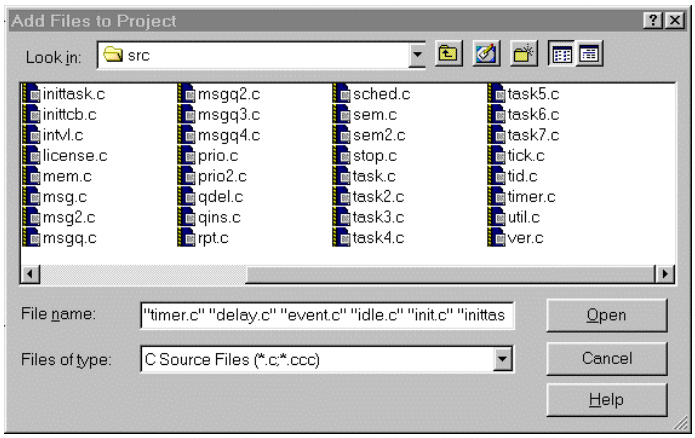

Figure 9: Adding Salvo Source Files to the Project

Click Open when finished. Repeat with Files of type: set to Asm Source Files (*.a*, *.s*) and add the `*.asm` files listed above to your project.

Your project window should now look like this:

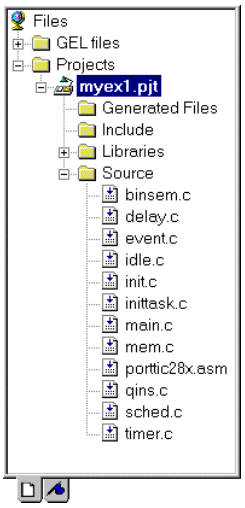

Figure 10: Project Window for a Source Code Build

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 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`.

Building the Project

Each CCS 'C2000 project requires a linker command file. Add it to your project now using Project → Add Files to Project, choose Files of type: Linker Command File (*.cmd)), navigate to and then select the linker command file you've created for your project:

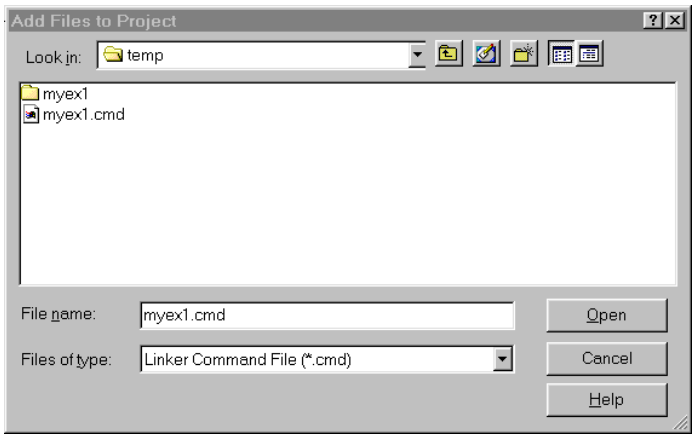

Figure 11: Adding the Configuration File to the Project

You will also need to add a runtime library (`*.lib`) and perhaps also interrupt vectors (`*.asm`) to your project.[5] Add them now. Once you project has all its files, the project window will look like this:

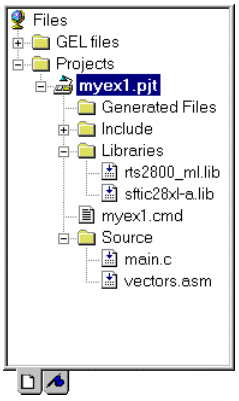

Figure 12: Complete Project Window for a Library Build

Save your project via Project → Save.[6] With everything in place, you can now build the project with Project → Make or Project → Rebuild All. The IDE's build results window will reflect the `cl2000` command lines:

```
-----------------------------  myex1.pjt - Debug  -------------
----------------
"C:\PROGRAM FILES\TIEVAL2\CC\BIN\cl2000" -g -q
-fr"c:/temp/myex1/Debug" -i"c:/temp" -i"c:/salvo/inc"
-d"_DEBUG" -d"LARGE_MODEL" -d"SYSW" -d"MAKE_WITH_FREE_LIB" -ml
-v28 -@"myex1/Debug.lkf" "main.c"
[main.c]

"C:\PROGRAM FILES\TIEVAL2\CC\BIN\cl2000" -g -q
-fr"c:/temp/myex1/Debug" -i"c:/temp" -i"c:/salvo/inc" -
d"_DEBUG" -d"LARGE_MODEL" -d"SYSW" -d"MAKE_WITH_FREE_LIB" -ml
-v28 -@"myex1/Debug.lkf" "vectors.asm"
<vectors.asm>

"C:\PROGRAM FILES\TIEVAL2\CC\BIN\cl2000" -@"Debug.lkf"
<Linking>

Build Complete,
  0 Errors, 0 Warnings, 0 Remarks.
```

Listing 3: Build Results for A Successful Library Build

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

```
******************************************************************************
      TMS320C2000 COFF Linker  PC Version 3.01
******************************************************************************
>> Linked Wed Jan 22 15:49:54 2003

OUTPUT FILE NAME:   <./Debug/myex1.out>
ENTRY POINT SYMBOL: "_c_int00"  address: 003d8019


MEMORY CONFIGURATION

                  name            origin    length      used    attr    fill
         ----------------------  --------  ---------  --------  ----  --------
PAGE  0: PROG                    003d8000   00020000  0000033c  R
         BOOT                    003ff000   00000fc0  00000000  R
         RESET                   003fffc0   00000002  00000002  R
         VECTORS                 003fffc2   0000003e  0000001e  R

PAGE  1: M0RAM                   00000000   00000400  00000000  RW
         M1RAM                   00000400   00000400  00000400  RW
         L0L1RAM                 00008000   00002000  00000000  RW
         H0RAM                   003f8000   00002000  000000e3  RW


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length      input sections
--------  ----  ----------  ----------   ----------------
.reset     0    003fffc0    00000002
                  003fffc0    00000002     rts2800_ml.lib : boot.obj (.reset)

vectors    0    003fffc2    0000001e
                  003fffc2    0000001e     vectors.obj (vectors)

.pinit     0    003d8000    00000000

.cinit     0    003d8000    00000019
                  003d8000    0000000e     rts2800_ml.lib : exit.obj (.cinit)
                  003d800e    0000000a                    : _lock.obj (.cinit)
                  003d8018    00000001     --HOLE-- [fill = 0000]

.text      0    003d8019    00000323
                  003d8019    00000044     rts2800_ml.lib : boot.obj (.text)
                  003d805d    00000000     sftic28xl-a.lib : mem.obj (.text)
                  003d805d    0000004a     rts2800_ml.lib : exit.obj (.text)
                  003d80a7    00000009                    : _lock.obj (.text)
                  003d80b0    0000005c     main.obj (.text)
                  003d810c    00000014     vectors.obj (.text)
                  003d8120    00000067     sftic28xl-a.lib : binsem.obj (.text)
                  003d8187    0000002e                     : delay.obj (.text)
                  003d81b5    0000001e                     : event.obj (.text)
```

```
                  003d81d3    00000018                   : init.obj (.text)
                  003d81eb    00000043                   : inittask.obj (.text)
                  003d822e    0000003e                   : qins.obj (.text)
                  003d826c    00000084                   : sched.obj (.text)
                  003d82f0    0000000e                   : timer.obj (.text)
                  003d82fe    0000003d                   : porttic28x.obj (.text)
                  003d833b    00000001                   : idle.obj (.text)

.const      1     00008000    00000000     UNINITIALIZED

.bss        1     00008000    00000000     UNINITIALIZED
                    00008000    00000000     rts2800_ml.lib : boot.obj (.bss)
                    00008000    00000000     sftic28xl-a.lib : idle.obj (.bss)
                    00008000    00000000                     : porttic28x.obj (.bss)
                    00008000    00000000                     : timer.obj (.bss)
                    00008000    00000000                     : sched.obj (.bss)
                    00008000    00000000                     : qins.obj (.bss)
                    00008000    00000000                     : mem.obj (.bss)
                    00008000    00000000                     : inittask.obj (.bss)
                    00008000    00000000                     : init.obj (.bss)
                    00008000    00000000                     : event.obj (.bss)
                    00008000    00000000                     : delay.obj (.bss)
                    00008000    00000000                     : binsem.obj (.bss)
                    00008000    00000000     vectors.obj (.bss)
                    00008000    00000000     main.obj (.bss)
                    00008000    00000000     rts2800_ml.lib : _lock.obj (.bss)
                    00008000    00000000                    : exit.obj (.bss)

.stack      1     00000400    00000400     UNINITIALIZED
                    00000400    00000000     rts2800_ml.lib : boot.obj (.stack)

.sysmem     1     00000000    00000000     UNINITIALIZED

.ebss       1     003f8000    000000e3     UNINITIALIZED
                    003f8000    00000080     rts2800_ml.lib : exit.obj (.ebss)
                    003f8080    0000005e     sftic28xl-a.lib : mem.obj (.ebss)
                    003f80de    00000004     rts2800_ml.lib : _lock.obj (.ebss)
                    003f80e2    00000001     main.obj (.ebss)

.econst     1     003f8000    00000000     UNINITIALIZED

.esysmem    1     003f8000    00000000     UNINITIALIZED

.data       1     00000000    00000000     UNINITIALIZED
                    00000000    00000000     rts2800_ml.lib : boot.obj (.data)
                    00000000    00000000     sftic28xl-a.lib : idle.obj (.data)
                    00000000    00000000                     : porttic28x.obj (.data)
                    00000000    00000000                     : timer.obj (.data)
                    00000000    00000000                     : sched.obj (.data)
                    00000000    00000000                     : qins.obj (.data)
                    00000000    00000000                     : mem.obj (.data)
                    00000000    00000000                     : inittask.obj (.data)
                    00000000    00000000                     : init.obj (.data)
                    00000000    00000000                     : event.obj (.data)
                    00000000    00000000                     : delay.obj (.data)
                    00000000    00000000                     : binsem.obj (.data)
                    00000000    00000000     vectors.obj (.data)
                    00000000    00000000     main.obj (.data)
                    00000000    00000000     rts2800_ml.lib : _lock.obj (.data)
                    00000000    00000000                    : exit.obj (.data)


GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name

[SNIP]

[65 symbols]
```

Listing 4: Map File for a Library Build

Note The CCS projects supplied in the Salvo for TI's TMS320C2000 DSPs distributions contain additional help files (`*.txt.c`) in each project's main directory.

Testing the Application

You can test and debug this application using the simulator or with actual hardware. Load the program via File → Load Program. The executable is normally a `*.out` file in the project's `Debug` subdirectory:

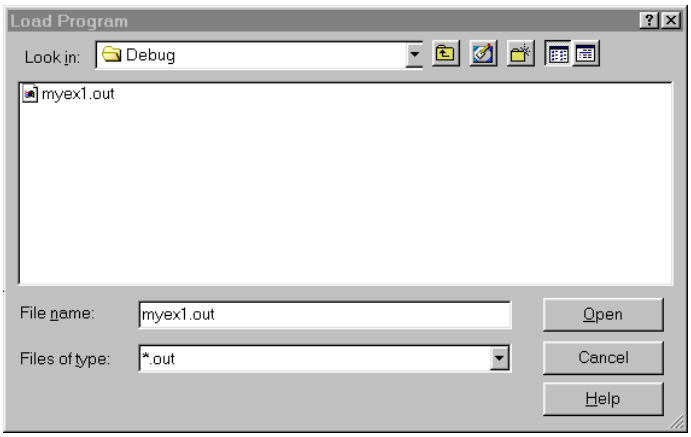

Figure 13: Loading the Program into Memory

Once the program is loaded, you can view source code in the integrated debugger, set breakpoints, run the profiler, watch variables, select Mixed Mode for simultaneous C- and assembly-language viewing, etc.

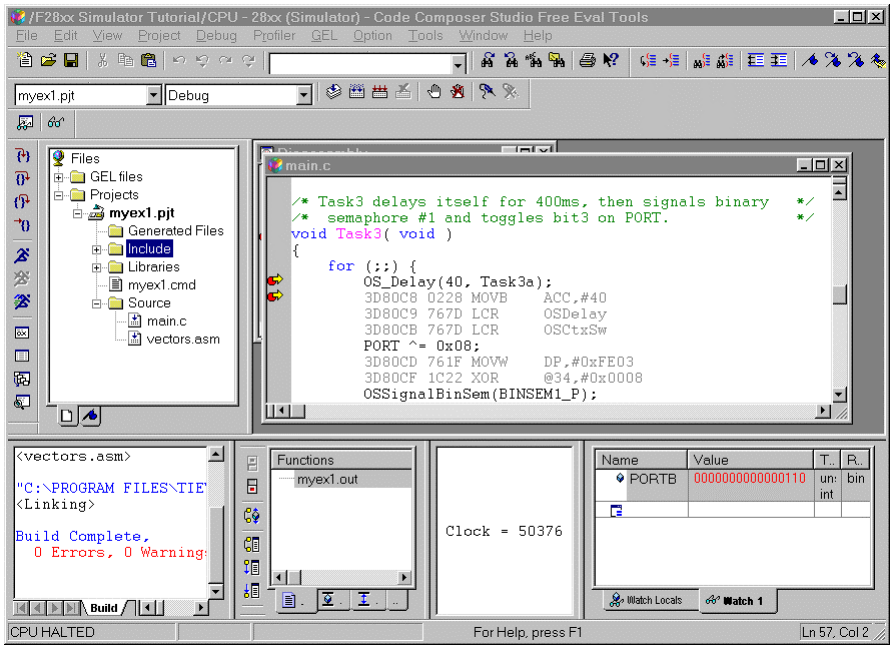

Figure 14: Testing a Salvo Application in the Simulator

Tip The project's map (`*.map`) file is very useful for debugging because it lists the addresses of functions and variables in memory. This information can be used in the monitor program to set breakpoints, display memory, better understand trace results, etc.

Note `cl2000` 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 in CCS.

Troubleshooting

Cannot find and/or read include file(s)

If you fail to add `\salvo\inc` to the project's include paths (see Figure 3) the compiler will generate an error like this one:

```
"main.c", line 15: fatal error: could not open
source file "salvo.h"
1 fatal error detected in the compilation of
"main.c".
Compilation terminated.
```

Figure 15: 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 3) the compiler will generate an error like this one:

```
"c:/salvo/inc/salvo.h", line 320: fatal error:
could not open source file "salvocfg.h"
1 fatal error detected in the compilation of
"main.c".
Compilation terminated.
```

Figure 16: 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`.

Application Crashes After Successful Build

CCS 'C2000 supports two memory models – small and large. Ensure that when using the large model (the default), all libraries in the project are compatible with the large model.

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 help CCS locate these files to enable source-level debugging. Note that this "corrective information" is not stored when the project is saved.

Example Projects

Code Composer 'C2000

Example projects for CC 'C2000 are found in the `\salvo\tut\tu1-6\sysaa` directories. The include path for each of these projects includes `\salvo\tut\tu1\sysaa`, and each project defines the `SYSAA` symbol.

Complete projects using Salvo freeware libraries are contained in the project files `\salvo\tut\tu1-6\sysaa\tu1-6lite.pjt`. 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\sysaa\tu1-6le.pjt`. 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\sysaa\tu1-6prolib.pjt`. 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\sysaa\tu1-6pro.pjt`. These projects also define the `MAKE_WITH_SOURCE` symbol.

Code Composer Studio 'C2000

Example projects for CCS 'C2000 are found in the `\salvo\tut\tu1-6\sysw` directories. The include path for each of these projects includes `\salvo\tut\tu1\sysw`, and each project defines the `SYSW` symbol.

Complete projects using Salvo freeware libraries are contained in the project files `\salvo\tut\tu1-6\sysw\tu1-6lite.pjt`. 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\sysw\tu1-6le.pjt`. 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\sysw\tu1-6prolib.pjt`. 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\sysw\tu1-6pro.pjt`. These projects also define the `MAKE_WITH_SOURCE` symbol.

---

1 Salvo's tutorial and example projects for CCS 'C2000 use the F28xx Simulator Tutorial System Configuration because it simulates both the CPU core and some peripherals, including interrupts and timers.

2 This Salvo project supports a wide variety of targets and compilers. For use with CCS 'C2000, it requires the `SYSW` 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.

3 This Salvo Lite library contains all of Salvo's basic functionality. The corresponding Salvo LE and Pro libraries are `sltic28xl-a.lib` and `sltic28xlia.a`, respectively.

4 You can Ctrl-select multiple files at once.

5 Since the project was created with the default options, including the large memory model, then a large-memory-model runtime library must be used.

6 CCS supports multiple projects in the project window, with just one active project. Project → Save Project will save only the active project.