PUMPKIN™
REAL-TIME SOFTWARE

AN-13
Application Note

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

Building a Salvo Application with Keil's C51 C Compiler and µVision IDE

Introduction

This Application Note explains how to use Keil's (http://www.keil.com/) C51 compiler and µVision IDE to create a multitasking Salvo application for the 8051 family of microcontrollers.

We will show you how to build the Salvo application contained in `\salvo\ex\ex1\main.c` for a generic 8051 microcontroller using the Keil tools. 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 C51 and µVision tools. With the µVision IDE you will be able to run and debug this application in the simulator or on real hardware (if available).

Creating and Configuring a New Project

Create a new µVision project using Project > New Project. In the Create New Project window, navigate to your working directory (in this case we've chosen `c:\temp`) and enter a name for the project (we'll use `myex1`) in the File Name field:

created by Andrew E. Kalman on Jul 27, 2002 updated on Jul 27, 2002

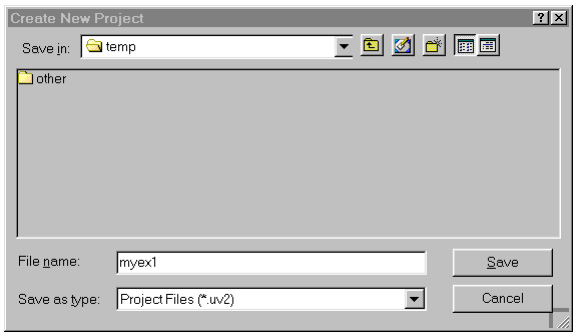

Figure 1: Creating the New Project

Click on Save to continue. The Select Devices for Target 'Target 1' window appears. Under the CPU tab select and expand Generic:

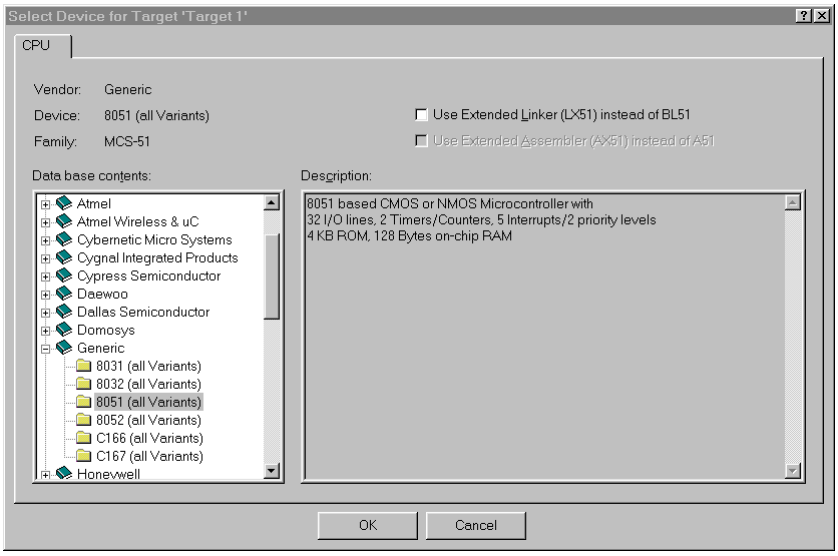

Figure 2: µVision Device Selection Window with Generic 8051 Selected

Select 8051 (all Variants) and click on OK to continue.

Now let's setup the project's options for Salvo's pathnames, etc. Choose Project > Options for Target 'Target 1' > C51 and define any symbols you may need for your project in the Preprocessor Symbols > Define area.[1] In the Include Paths, add `\salvo\inc`:

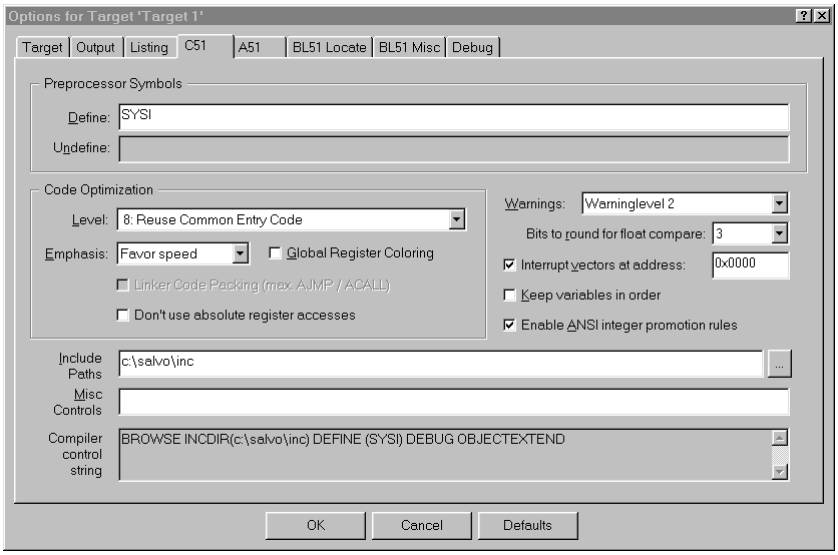

Figure 3: C51 Options for Target

Click on OK to finish configuring your project.

Adding your Source File(s) to the Project

Now it's time to add files to your project. Choose Project > Targets, Groups, Files > Groups / Add Files, select Source Group 1 under Available Groups, click on Add Files to Group…, navigate to your project's directory, select your `main.c` and click on Add. Your project window should now look like this:

Figure 4: µVision Project Window with your Source File(s)

Click on Close after you are adding source files to your project.

Creating Groups for Salvo Files

For legibility and organizational purposes, we recommend you add additional groups to your project to hold Salvo files. They are:

Salvo Configuration File
Salvo Libraries
Salvo Sources

Add these groups now using Project > Targets, Groups, Files > Groups / Add Files > Group to Add. When done, your project window should look like this:

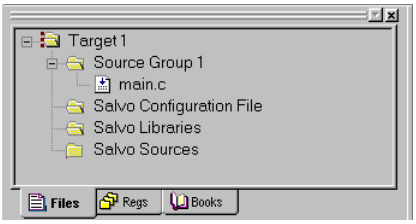

Figure 5: µVision Project Window with your Source File(s) and Salvo Groups

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 C51 compiler is `sfc51sdab.lib`.[2] Choose Project > Targets, Groups, Files > Groups / Add Files, select Salvo Libraries under Available Groups, click on Add Files to Group…, choose Library file (*.lib) under Files of type, navigate to the `\salvo\lib` directory, and select `sfc51sdab.lib`:

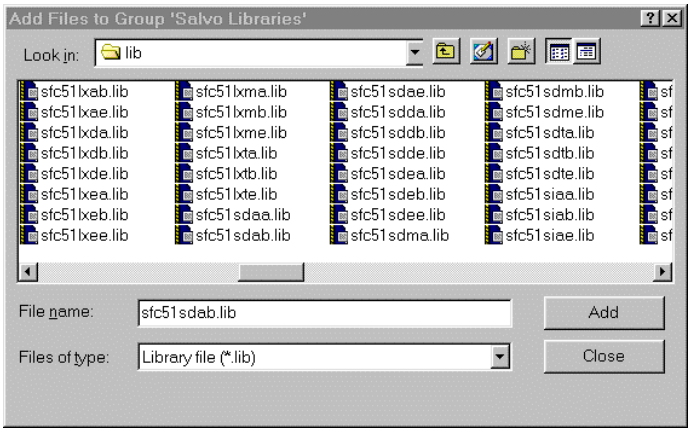

Figure 6: Adding the Library to the Project

Click on Add, then on Close when you are finished. You can find more information on Salvo libraries in the *Salvo User Manual*.

Your project window should now look like this:

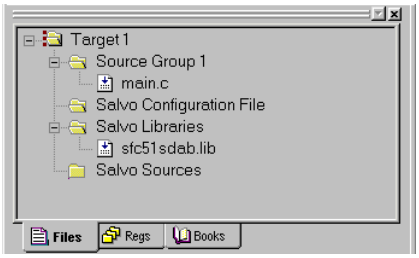

Figure 7: Vision Project Window for Library Build

The salvocfg.h Header File

You will also need a `salvocfg.h` file for this project. To use the library selected in Figure 6, your `salvocfg.h` should contain only:

```
#define OSUSE_LIBRARY          TRUE
#define OSLIBRARY_TYPE         OSF
#define OSLIBRARY_GLOBALS      OSD
#define OSLIBRARY_CONFIG       OSA
#define OSLIBRARY_VARIANT      OSB
```

Listing 1: salvocfg.h for a Library Build

Create this file and save it in your project directory, e.g. `c:\temp\salvocfg.h`. We also recommend adding it to the project's Salvo Configuration File group using Project > Targets, Groups, Files > Groups / Add Files, etc.

Note To add a header file (`*.h`) to a Group, in the Get Filetype window you must specify that the file is of type Text Document file for it to be accepted.

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                 port8051.c
event.c                 qins.c
init.c                  sched.c
inittask.c              timer.c
```

To add these files to your project, choose Project > Targets, Groups, Files > Groups / Add Files, select Salvo Sources under Available Groups, click on Add Files to Group…, choose C source file (*.c) under Files of type, navigate to the `\salvo\src` directory, select[3] the `*.c` files listed above, and click on Add:

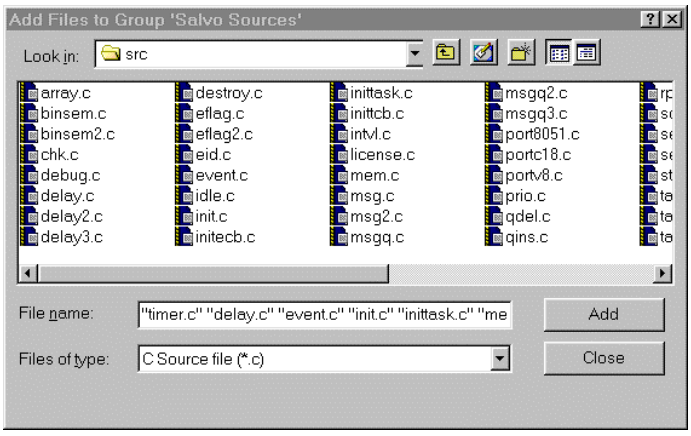

Figure 8: Adding Salvo Source Files to the Project

Click on Close when finished. Your project window should now look like this:

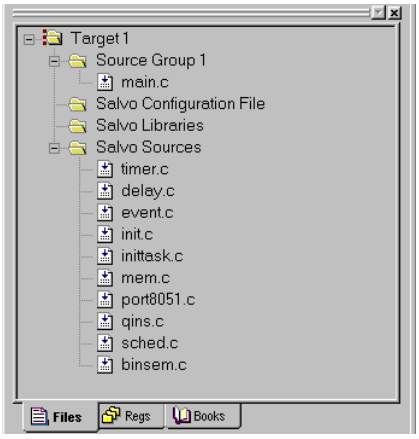

Figure 9: Project Window for a Source Code Build

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
#define OSLOC_ALL                   data
```

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`. We also recommend adding it to the project's Salvo Configuration File group using Project > Targets, Groups, Files > Groups / Add Files, etc.

Note To add a header file (`*.h`) to a Group, in the Get Filetype window you must specify that the file is of type Text Document file for it to be accepted.

Building the Project

For a successful compile, your project must also include a header file (e.g. `reg51.h`) for the particular chip you are using. We recommend that you include it in your project's `salvocfg.h`. For a library build, your `salvocfg.h` would look like this:

```
#include <reg51.h>

#define OSUSE_LIBRARY         TRUE
#define OSLIBRARY_TYPE        OSF
#define OSLIBRARY_GLOBALS     OSD
#define OSLIBRARY_CONFIG      OSA
#define OSLIBRARY_VARIANT     OSB
```

Listing 3: salvocfg.h with Target-specific Header File

With everything in place, you can now build the project using Project > Build Target or Project > Rebuild all target files. The build results can be seen in the Build window:

```
Building targer 'Target 1'
compiling timer.c…
compiling delay.c…
compiling event.c…
compiling init.c…
compiling inittask.c…
compiling mem.c…
compiling port8051.c…
compiling qins.c…
compiling sched.c…
compiling binsem.c…
linking…
Program Size: data=46.0 xdata=0 code=1028
"myex1" – 0 Error(s), 0 Warning(s).
```

Listing 4: Source Code Build Results

This example uses a total of 46 bytes of RAM in the `data` space, and 1028 bytes of ROM in the `code` space.

Note The library-based projects supplied in the Salvo for 8051 family distributions use `\salvo\src\mem.c` and the appropriate entries in the `salvocfg.h` header file to minimize the RAM usage. See the *Libraries* chapter in the *Salvo User Manual* for more information.

Testing the Application

You can test and debug this application using the µVision simulator or real hardware. You launch the debugger after a successful build by choosing Debug > Start/Stop Debug Session.

You can use all of the IDE's supported features when debugging and testing Salvo applications. This includes breakpoints, profiling, watch windows, tracing, etc.

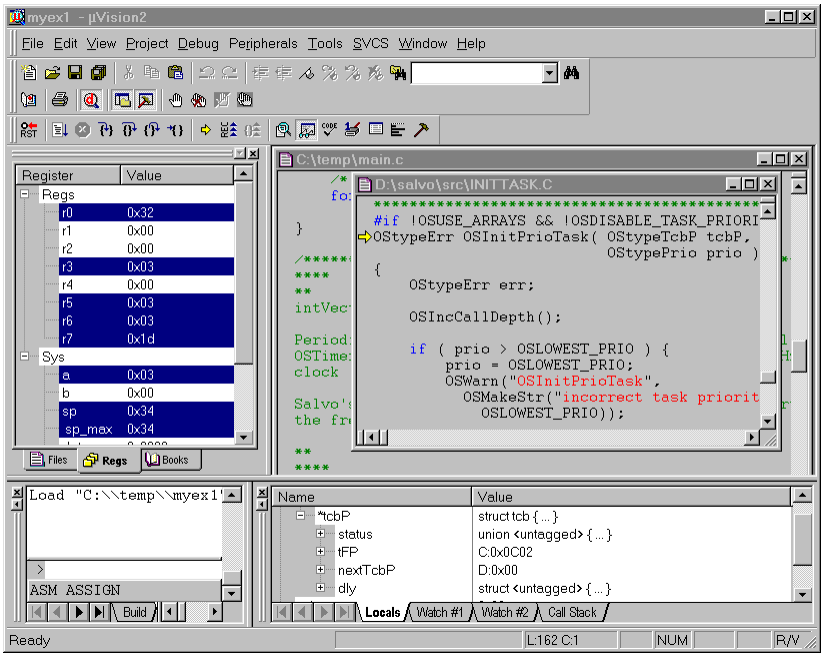

Figure 10: Testing a Salvo Application in µVision Debugger

Note µVision supports debugging at the source code level. Only applications built from the Salvo source code enable you to step through Salvo services (e.g. `OSCreateBinSem()`) at the source code level. Regardless of how you build your Salvo application, you can always step through your own C and assembly code in the IDE / debugger.

Troubleshooting

C51 Error: can't open file 'salvo.h'

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

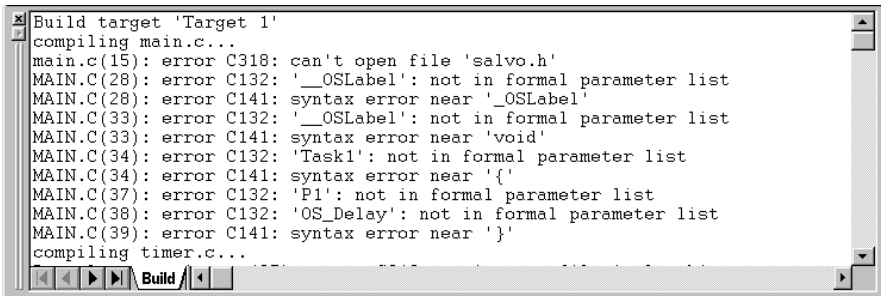

Figure 11: Compiler Error due to Missing \salvo\inc Include Path

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 create a `salvocfg.h` header file in the project's own directory, the compiler will generate errors like these:

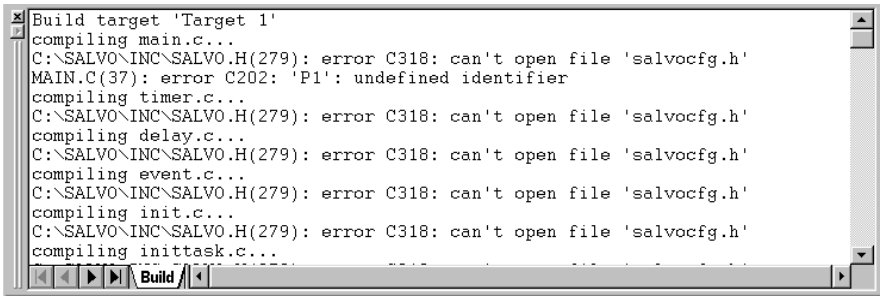

Figure 12: Compiler Error due to Missing salvocfg.h

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 See Window Upon Opening Project

If you can't see a particular window after opening an µVision project that's part of a Salvo distribution, it may be because your display's resolution is less than that used to create the project. Select Window > Tile Horizontal to make all open windows visible.

Example Projects

Example projects for the C51 compiler and µVision IDE are found in the `\salvo\tut\tu1-6\sysi` directories. The include path for each of these projects includes `\salvo\tut\tu1\sysi`, and each project defines the `SYSI` symbol.

Complete projects using Salvo source code are contained in the project files `\salvo\tut\tu1-6\sysi\tu1-6pro.Uv2`.

Complete projects using Salvo standard libraries are contained in the project files `\salvo\tut\tu1-6\sysi\tu1-6le. Uv2`. These projects also define the `MAKE_WITH_STD_LIB` symbol.

Complete projects using Salvo freeware libraries are contained in the project files `\salvo\tut\tu1-6\sysi\tu1-6lite. Uv2`. These projects also define the `MAKE_WITH_FREE_LIB` symbol.

[1] This Salvo project supports a wide variety of targets and compilers. For use with µVision and the C51 compiler, it requires the `SYSI` 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.

[2] This Salvo Lite library contains all of Salvo's basic functionality. The corresponding Salvo LE and Pro library is slc51sdab.lib.

[3] You can Ctrl-select multiple files at once.