PUMPKIN™
REAL-TIME SOFTWARE

# GSG-STELLARIS-KRVMDK-ARM
## Getting Started Guide

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

# *Building a Salvo Application for Stellaris™ Microcontrollers using Keil's ARM® RealView® Microcontroller Development Kit*

## Introduction

This Getting Started Guide explains how to use Keil's (http://www.keil.com/) ARM RealView Microcontroller Development Kit to create a multitasking Salvo application for Luminary Micro's Stellaris™ Cortex-M3-based microcontrollers.

We will show you how to build the standard Salvo demo application `tut5`. A complete project to build `tut5` is included in every Salvo distribution.

Building your own applications will be substantially similar.

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 Keil ARM RealView MDK tools. With the included µVision IDE you will be able to run and debug this application on real hardware (e.g. a Stellaris Development Kit).

## Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications for Stellaris™

*created by Andrew E. Kalman on May 15, 2006 updated on May 17, 2006*

MCUs using Keil's ARM RealView Microntroller Development Kit:

*Salvo User Manual*
*Salvo Compiler Reference Manual*
*RM-STELLARIS-ARMRV*

## 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 (we'll use `\Pumpkin\Salvo\Examples\ARM\Luminary_LM3S1XX\Luminary_DK-LM3S1XX\RVMDK\Tut\Tut5\Lite`) and enter a name for the project (we'll use `tut5`) in the File Name field:

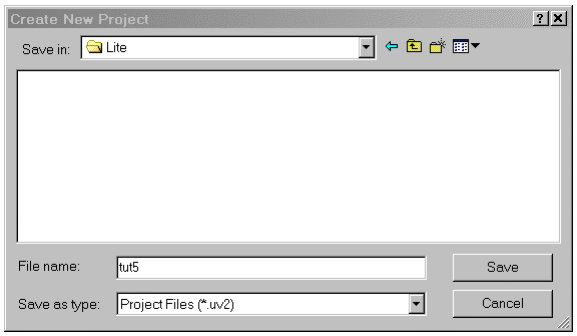


**Figure 1: Creating the New Project**

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

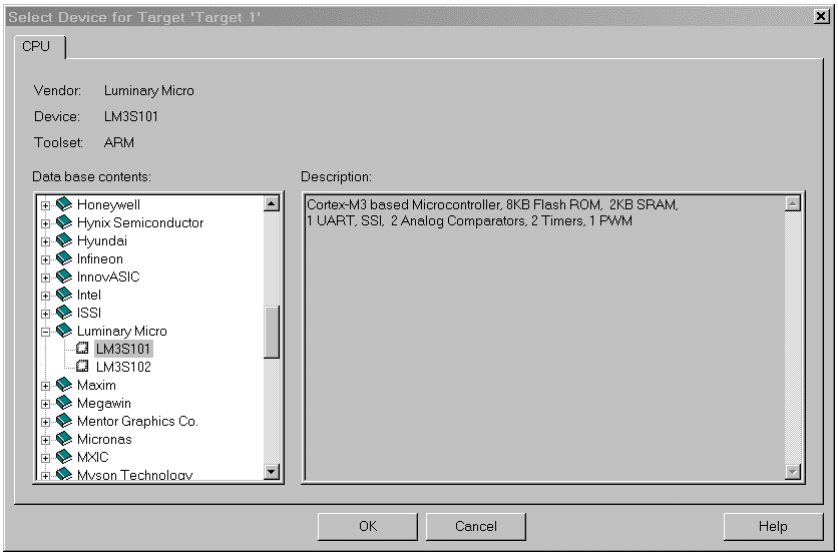


**Figure 2: µVision Device Selection Window with Luminary Micro LM3S101 Selected**

Select LM3S101 and click on OK to continue. You'll be prompted to copy and add target-specific startup code to the project. Select Yes to continue:


**Figure 3: Confirming Addition of Startup Code to Project**

The file `Startup.s` will be added to the project.

## C/C++ Options

Now let's setup the project's options for Salvo's pathnames, etc. Choose Project → Options for Target 'Target 1' → C/C++ and define any symbols you may need for your project in the Preprocessor Symbols → Define area. In the Include Paths, add a path to the current directory (`.\`) – Salvo needs this to find its project-specific configuration file `salvocfg.h` (see below). Next, add the path to Salvo's include directory (`C:\Pumpkin\Salvo\Inc` is the default location). Lastly, add any other include paths your project may require (e.g. to find board-specific header files – in this example, `..\..\..\..`):

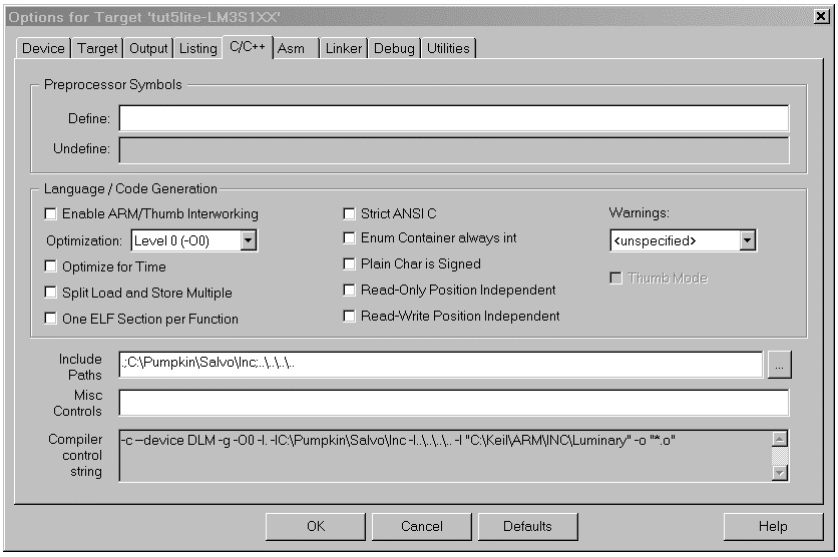


**Figure 4: C/C++ Options for Target**

## Assembler Options

If your project's assembly code requires any defined symbols, add them under the Asm tab:[1]

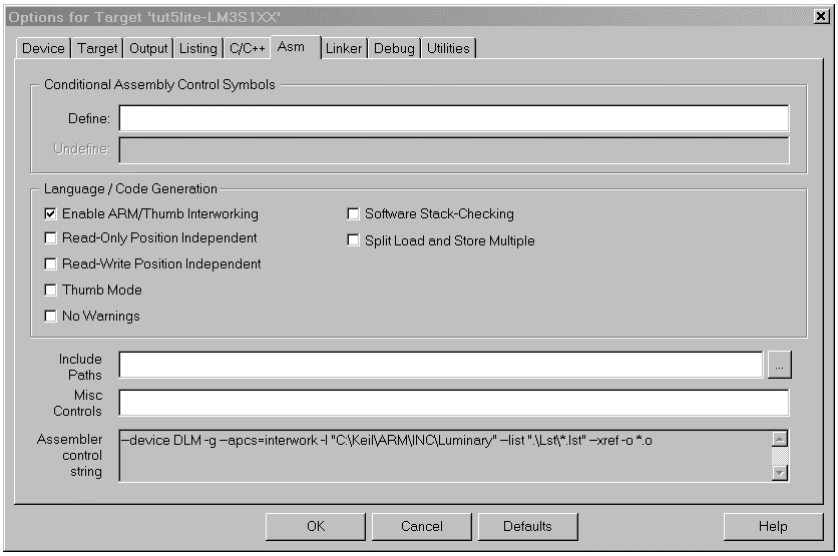


**Figure 5: Asm Options for Target**

## Linker Options

Under the Linker tab be sure to select Use Memory Layout from Target Dialog if the checkbox is available.[2] Add `--entry Reset_Handler` to the Misc controls box to ensure proper startup at runtime:

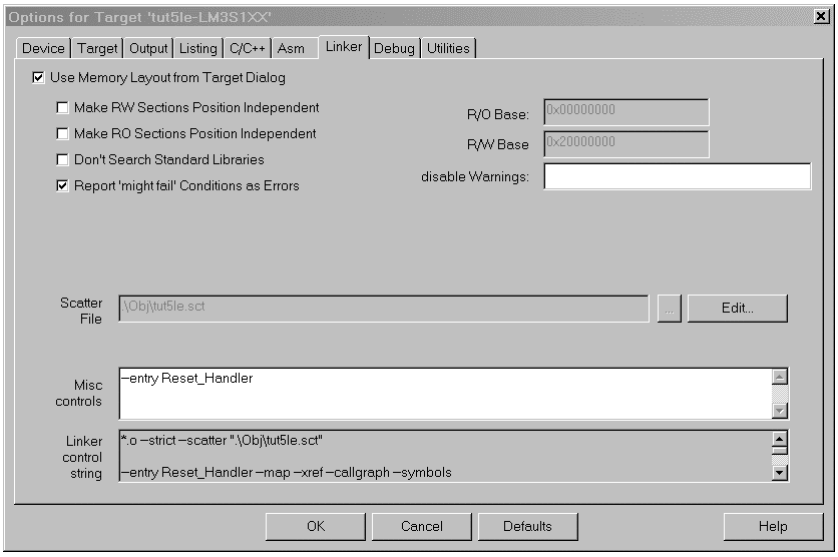


**Figure 6: Linker Options for Target**

## Debug Options

You'll need to select the appropriate debugger. Keil's ULINK debugger provides the ability to debug on real hardware over a JTAG port. Under the Debug tab select the ULINK Cortex-M3 Debugger from the drop-down list and ensure it's selected via the Use radio button:

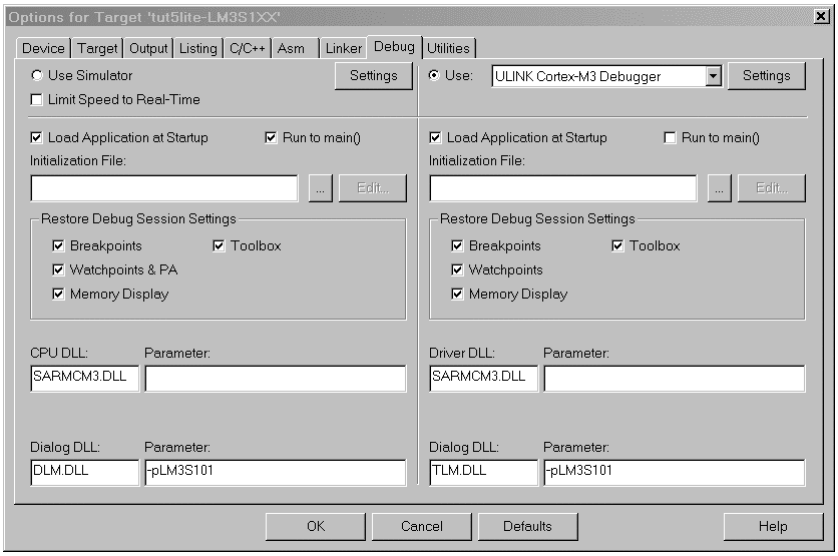


**Figure 7: Debug Options for Target**

## Utilities Options

Under the Utilities tab select Use Target Driver for Flash Programming pull-down. Selecting Update Target before Debugging will streamline your debugging sessions:

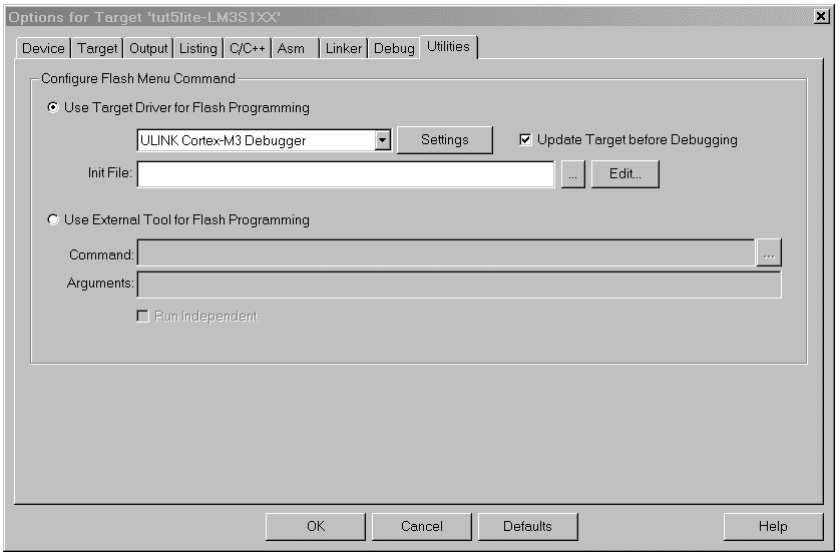


**Figure 8: Utilities Options for Target**

Click on OK to finish configuring your project.

## Groups

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

Header Files
Sources
Board Driver Library
Salvo Help Files
Salvo Configuration File
Salvo Source Files
Salvo Target-Specific Source Files
Salvo Library Files
Listings

For each group, choose Project → Components, Environment and Books, and under Project Components → Groups add and (re-)order the new group names[3], and select OK. When finished, your project window should look like this:

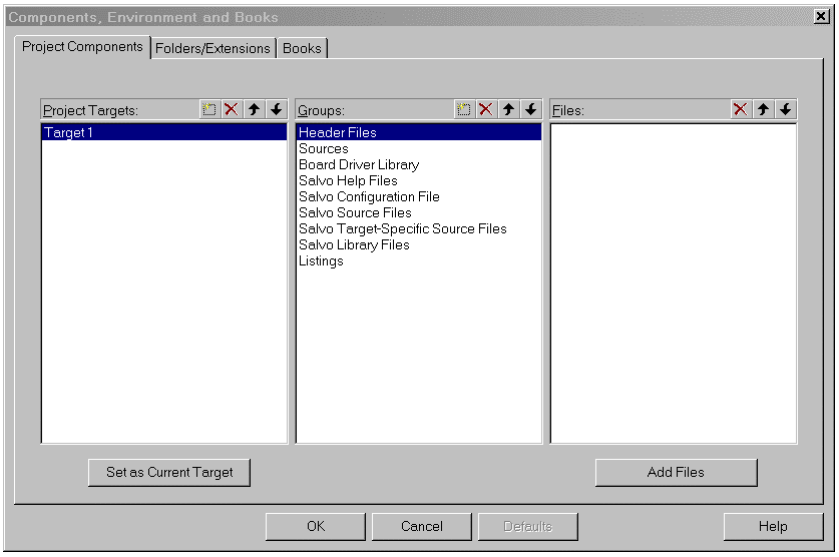


**Figure 9: Project Window with Groups**

## Compiler Selection

Lastly, you'll need to configure this project for use with Keil's ARM RealView C compiler. Choose Project → Components, Environment and Books, and under Folders/Extensions → Select ARM Development Tools select Use RealView Compiler:

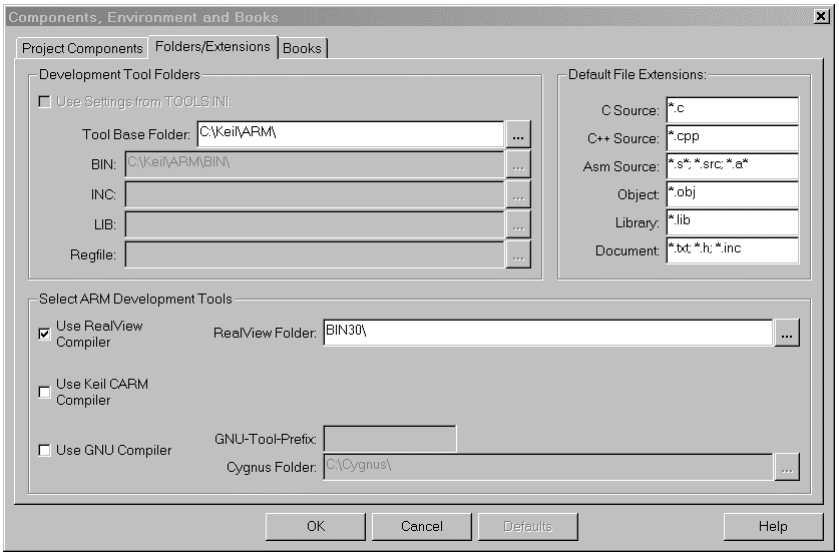


**Figure 10: Selecting the RealView C Compiler**

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 → Components, Environment and Books, and under Project Components → Groups select Sources. Click on Add Files, navigate to your project's directory, select the files your application requires, and Add, then Close. Your Project Files window should look like this:

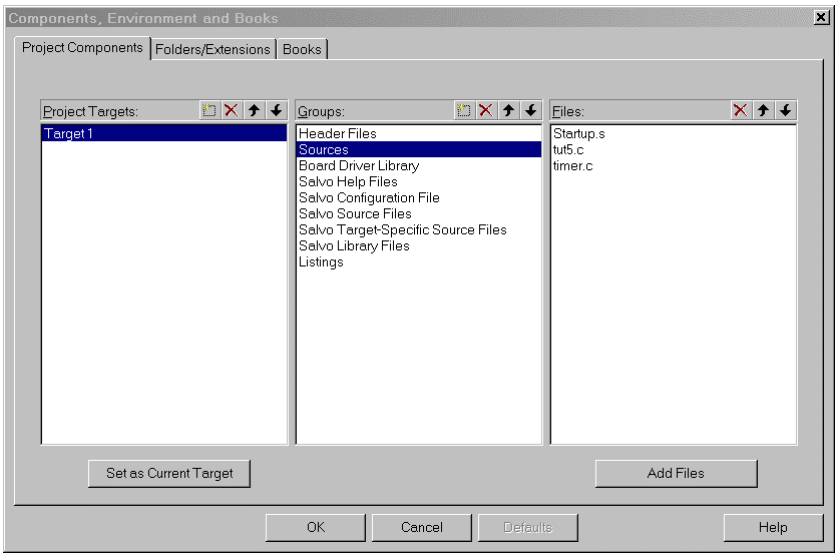


**Figure 11: Adding Source Files to the Project**

When finished, select OK.

**Note** In an attempt to minimize the unnecessary duplication of source files, those that are shared across multiple Salvo projects are often located in higher-level (parent) directories above the project directories. Therefore when adding source files to a group like the Sources group, you may need to navigate to multiple folders to select the desired files.

In the above example, the Sources files in Figure 11 are located in the parent directory of the project directory.

## 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 (Salvo Lite, LE and Pro), or with the Salvo source code files (Salvo Pro only) as nodes in your project.

## Adding a Salvo Library

For a *library build* – e.g. what you would do when evaluating Salvo via Salvo Lite – the Salvo freeware library for the Luminary Micro LM3Sxxx is `salvofarmrvcm3-t.lib`.[4] Choose Project → Components, Environment and Books, and under Project Components → Groups select Salvo Library Files. Click on Add Files, navigate to the `\Pumpkin\Salvo\Lib\ARMRV` directory, select `salvofarmrvcm3-t.lib` and Add, then Close. Your Project Files window should look like this:

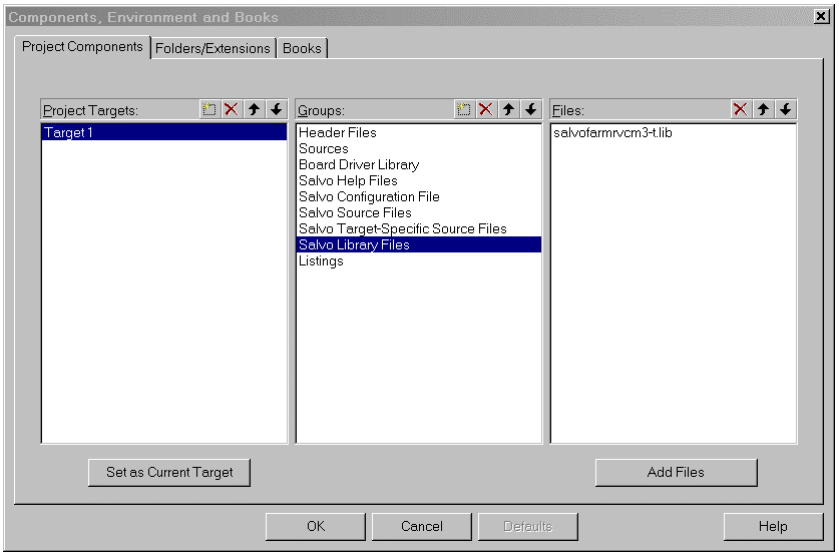


**Figure 12: Adding Salvo Libraries to the Project**

When finished, select OK.

---

**Note** When browsing to add files to a group via the Add Files button, use the Files of type setting to see files other than source (`*.c`) files.

---

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

## Adding Salvo's salvomem.c

Every Salvo project requires Salvo's `salvomem.c` source file. Choose Project → Components, Environment and Books, and under Project Components → Groups select Salvo Source FIles. Click on Add Files, navigate to the `\Pumpkin\Salvo\Src` directory, select `salvomem.c` and Add, then Close. Your Project Components window should look like this:

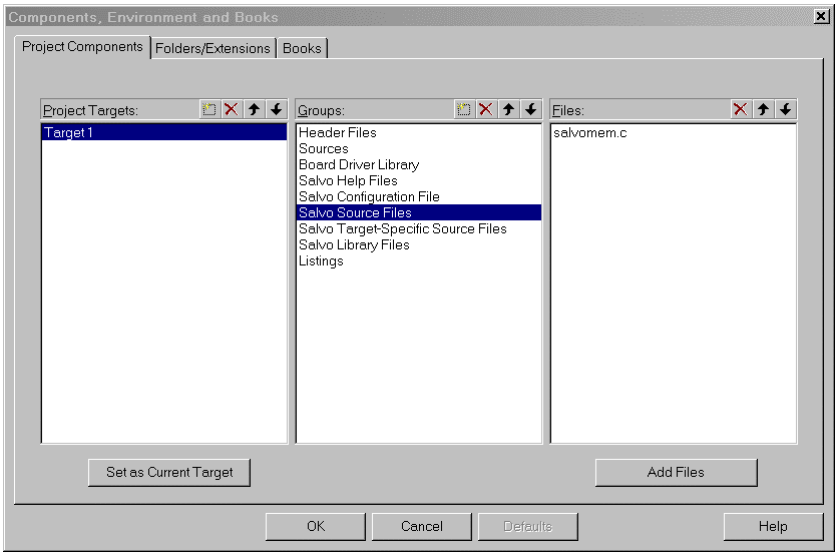


**Figure 13: Adding Salvo Source Files to the Project**

## Adding Target-specific Salvo Source Files

Additionally, your project will require target-specific source files. These include the files for user hooks to target-specific hardware. The files in `\Pumpkin\Salvo\Src\ARMRV` provide pre-defined user hooks and the Salvo context switcher[5] for Keil's ARM RealView compiler:

```
salvohook_interrupt_cm3.c
```

To add these files to your project, choose Project → Components, Environment and Books, and under Project Components → Groups select Salvo Target-Specific Source Files. Click on Add Files, navigate to the `\Pumpkin\Salvo\Src\ARMRV` directory, select the `*.c` files listed above and Add, then Close. Your Project Components window should look like this:

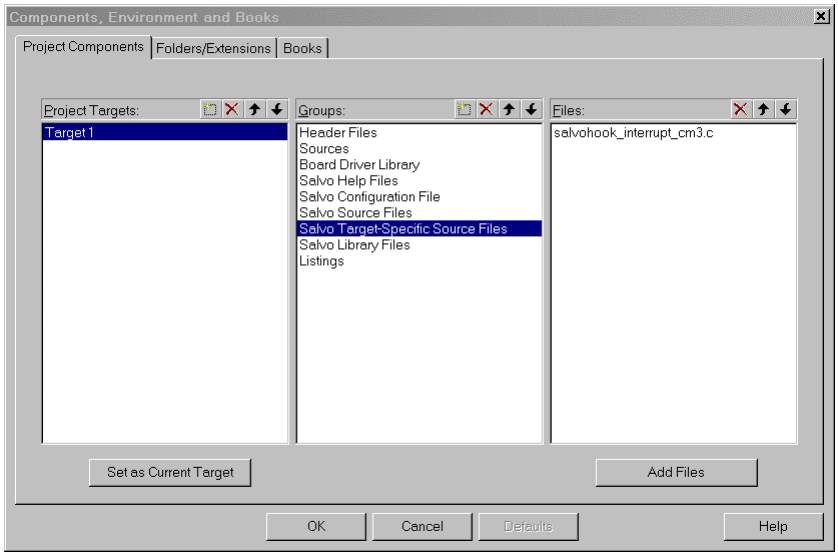


**Figure 14: Adding Salvo Target-specific Source Files to the Project**

When finished, select OK.

## The salvocfg.h Header File

A `salvocfg.h` header file is required for every Salvo project. You can create your own `salvocfg.h` or copy an existing one and modify it accordingly. Place it in the project's directory (part of the project's include paths – see *C/C++ Options*).

The `salvocfg.h` for this project contains only:

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

#define OSEVENTS                1
#define OSEVENT_FLAGS           0
#define OSMESSAGE_QUEUES        0
#define OSTASKS                 4
```

**Listing 1: Example salvocfg.h for a Salvo Lite Library Build**

**Note** The settings above are for this particular example project. The settings for your projects will vary depending on which libraries you use, how many tasks and events are in your application, etc.

For your convenience, you'll want your project's `salvocfg.h` to be easily accessible. Choose Project → Components, Environment and Books, and under Project Components →

Groups select Salvo Configuration File. Click on Add Files, navigate to your project's directory, select `salvocfg.h` and Add, then Close. Your Project Components window should look like this:

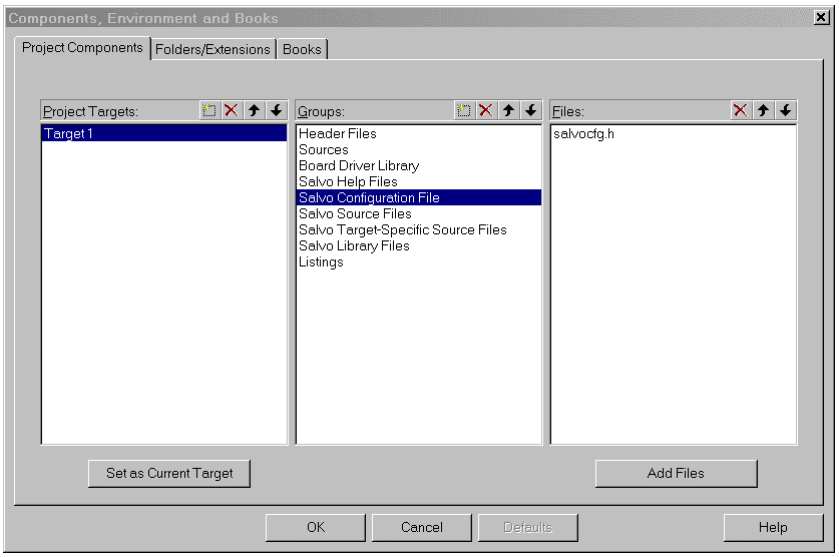


**Figure 15: Adding the Configuration File to the Project**

When finished, select OK.

Lastly, you'll need to add any board driver files your application requires to the Board Driver Library folder. These files are usually provided as part of Keil's ARM RealView MDK or with your target hardware. In this example, `pdc.c` and its header file `pdc.h` are provided by Luminary Micro for the Stellaris Development Kit and are located in the Salvo Examples tree several levels up from the project directory – hence the `..\..\..\..` include path (see *C/C++ Options*). The DriverLib.lib library is part of Keil's ARM RealView MDK installation.

Your project window should now look like this:

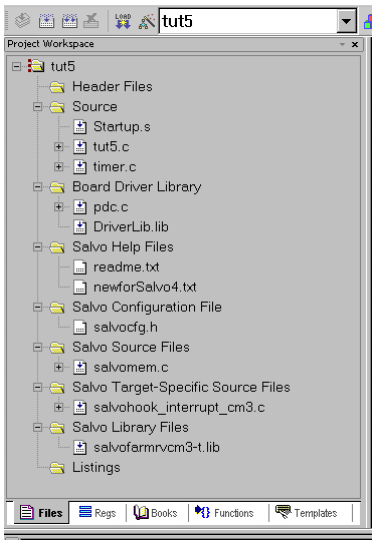


**Figure 16: Project Window for a Library Build**

**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 editing, etc. Additional often-used files like header or listing files can be added to their appropriate groups.

## Modifying the Startup.s file

The default startup code in `Startup.s` is populated primarily with default handlers that will trap errant vectors.

Nearly all Salvo applications will make use of Salvo's time services (e.g. `OS_Delay()`), normally via an IRQ handler. Therefore you must modify the vector table in `Startup.s` to invoke your application's interrupt handlers so that `OSTimer()` is called.

In this example, the IRQ handler (`SysTick_irq_handler()`, in the project's `timer.c`) is used to call Salvo's `OSTimer()` at a fixed rate of 100Hz. Therefore you must edit your project's `Startup.s` file to add the SysTick handler vector. E.g. replace

```
DCD     Default_Handler         ; PendSV Handler
DCD     Default_Handler         ; SysTick Handler
DCD     Default_Handler         ; GPIO Port A
```

with

```
DCD     Default_Handler         ; PendSV Handler
EXTERN  SysTick_irq_handler
DCD     SysTick_irq_handler     ; SysTick Handler
DCD     Default_Handler         ; GPIO Port A
```

in `Startup.s`. Repeat this procedure for every IRQ handler in your application.

## Building the Project

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 Output window:

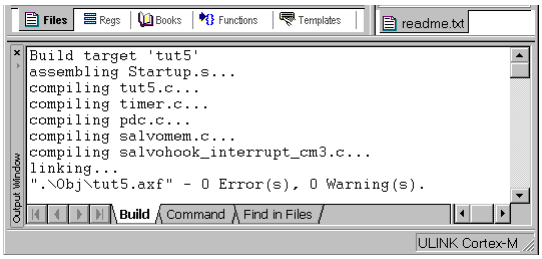


**Figure 17: Build Results for a Library Build**

**Note** The µVision projects supplied in the Salvo for Stellaris distributions contain additional help files in each project's Salvo Help Files group.

## Testing the Application

You can test and debug this application using the optional ULINK JTAG interface. After building the project, select Flash → Download. This will erase, program and then verify your application on the target.[6] Select Flash → Start/Stop Debugging Session. This will begin your debugging session, where you can run your application, set and clear breakpoints, watch variables, etc:

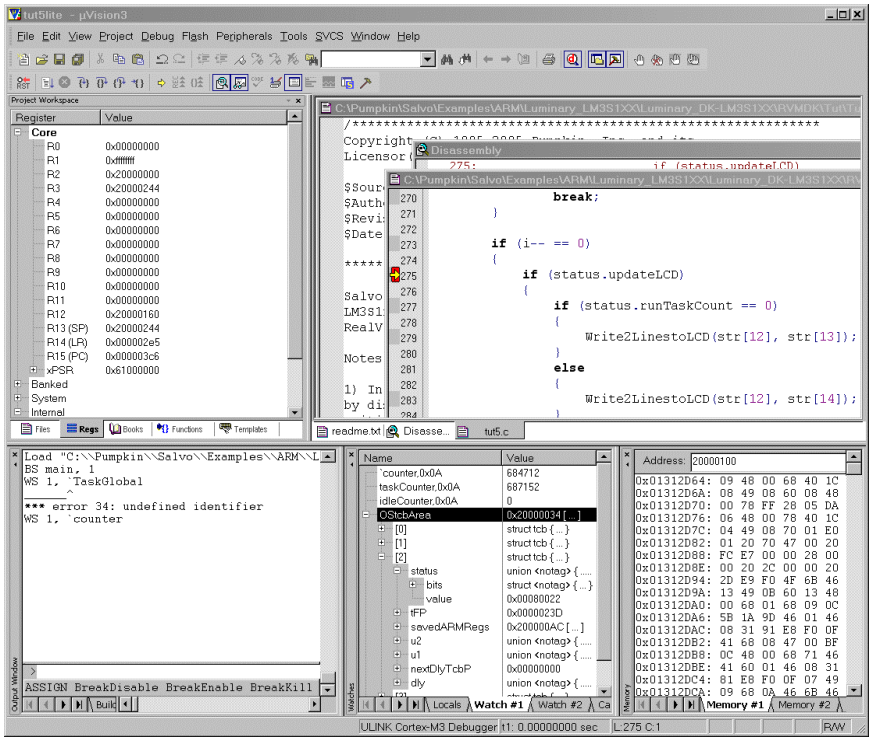

**Figure 18: Single-stepping with ULINK**

## Example Projects

Example Salvo projects for use with the Stellaris Development Board, Keil's ARM RealView compiler and Keil's µVision3 IDE can be found in the

```
C:\Pumpkin\Salvo\Examples\ARM\Luminary_LM3S1XX\
  Luminary_DK-LM3S1XX\RVMDK
```

directories of every Salvo for Stellaris family distribution. Salvo Lite and LE example projects are built using Salvo libraries. Salvo Pro example projects are built using Salvo libraries and the Salvo source code.

---

1 Enable Arm/Thumb interworking is a bug in µVision and will be fixed in a future release. Stellaris Cortex-M3 parts run only in Thumb2 mode and therefore have no need for an ARM interworking veneer.

2 This option is greyed out in the evaluation version of the Keil RealView MDK toolset.

3 Groups can be renamed in this window.

4 This Salvo Lite library contains all of Salvo's basic functionality. The corresponding Salvo LE and Pro libraries are salvolarmrvcm3[-|i]t.lib.

5 Salvo Pro only.

[6] Once your target is successfully programmed you can start program execution immediately without entering the µVision debugger, e.g. by pressing the Stellaris Development Kit's reset button.