PUMPKIN™
REAL-TIME SOFTWARE

GSG-STELLARIS-GCC-CM3

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 GCC for Cortex-M3

Introduction

This Getting Started Guide gives a brief overview of how to create a multitasking Salvo application for Luminary Micro's Stellaris™ Cortex-M3-based microcontrollers using GNU ARM tools for Cortex-M3.

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

The examples provided with the Salvo for Stellaris distribution are meant to run on a Luminary Micro Stellaris Development Board. Therefore, this guide will be most useful if are using a Stellaris Development Kit.

In order to build with the GNU toolchain, you must install the CodeSourcery Sourcery G++ for Stellaris Family tools. Furthermore, if you are using a Stellaris Development Board, then you must also install the software library, DriverLib, and the FTDI USB drivers which allows direct JTAG debug via a USB cable. All of the steps necessary for this are contained in the "GNU Tools QUICKSTART" guide, found on the CD included with the Stellaris Development Kit.

created by Andrew E. Kalman on May 16, 2006 updated on May 17, 2006

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications for Stellaris™ MCUs using GCC ARM tools for Cortex-M3:

Salvo User Manual
Salvo Compiler Reference Manual
RM-STELLARIS-GCCARM

The following documents found on the Luminary Micro Development Kit CD provide useful information:

GNU Tools QUICKSTART

Creating and Configuring a New Project

The build environment is driven by a Makefile. The Salvo distribution contains an example project with a Makefile, and you should start with the example Makefile for your own project. In the Salvo distribution, you can find the example Makefile here:

```
C:\Pumpkin\Salvo\Examples\ARM\Luminary_LM3S1XX\
   Luminary_DK-LM3S1XX\GCCARM\Tut\Tut5\Lite
```

To start, create your own project directory. In this guide, we will use a project directory called `MyProj`, as shown below:

```
C:\Pumpkin\Salvo\Examples\ARM\Luminary_LM3S1XX\
   Luminary_DK-LM3S1XX\GCCARM\Tut\Tut5\MyProj
```

Copy the Makefile from the Lite directory into this directory, then view the Makefile with an editor. Look at the following macros defined in the Makefile:

```
#
# Define locations where the DriverLib headers and library can be found
# Change this as needed on your system.
#
DRIVERLIB_ROOT=C:/DriverLib
DRIVERLIB_HWINC=${DRIVERLIB_ROOT}
DRIVERLIB_INC=${DRIVERLIB_ROOT}/src
DRIVERLIB_LIB=${DRIVERLIB_ROOT}/src/gcc/libdriver.a

# salvo library
SALVOLIB=../../../../../../../../Lib/GCCARM/libsalvofgccarmcm3t.a
```

The macro `DRIVERLIB_ROOT` points to the location where the Stellaris software driver library, DriverLib, was installed and built. If it is in another location other than that shown in the Makefile, then edit the Makefile and change the path to DriverLib. The remaining `DRIVERLIB_...` macros do not need to be changed unless you rearranged the tree.

If you do not want to use DriverLib at all, then set all the DRIVERLIB_ macros to be empty (ex: DRIVERLIB_ROOT=)
The macro SALVOLIB points to the location of the Salvo library that will be linked with the application. If you are using a different Salvo library, then this macro should be edited.

Adding your Source File(s) to the Project

Refer to the following macros in the Makefile:

```
VPATH=${RTOS_SOURCE_DIR}:${RTOS_SOURCE_DIR}/GCCARM:${COMPILER_SOURCE_DIR
}:${DEMO_SOURCE_DIR}:${PLATFORM_SRC_DIR}

OBJ= startup.o \
     tut5.o \
     timer.o \
     pdc.o \
     salvohook_interrupt_cm3.o \
     salvomem.o
```

The VPATH macro lists all the locations where source files may be found. Change this if you locate your source files in different locations from the example.

The OBJ macro lists all the object files that are to be generated from source files. Add or remove files as appropriate for your project. In this example, the files listed are the following:

tut5, timer, pdc, startup – these are part of the application
salvomem – required for a Salvo project
salvohook_interrupt_cm3 – target specific code

The following line in the Makefile provides the name of the application executable. It can be changed to the name of your application.

```
tut5.axf: ${OBJ}
```

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.

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.

Building the Project

In this example we will build from the command line using a shell. It is possible to integrate Makefile building with an IDE such as Eclipse, but that is not covered in this guide. The shell is a *nix-style shell such as `bash` or `sh`. You should already have this installed as part of the installation and building of DriverLib (see Luminary Micro GNU Tools QUICKSTART).

Open a shell window and navigate to the project directory, in this case:

```
C:\Pumpkin\Salvo\Examples\ARM\Luminary_LM3S1XX\
   Luminary_DK-LM3S1XX\GCCARM\Tut\Tut5\MyProj
```

With everything in place, you can now build the project using

```
$ make clean
```

followed by

```
$ make
```

This will result in

```
$ make clean
$ make
  CC    ../../../startup.c
  CC    ../tut5.c
  CC    ../timer.c
  CC    ../../../../pdc.c
  CC    ../../../../../../../../Src/GCCARM/salvohook_interrupt_cm3.c
  CC    ../../../../../../../../Src/salvomem.c
  LD    tut5.axf
$
```

Figure 1: Build Results

Testing the Application

You can test and debug this application using the built-in FTDI-based USB connection on the Stellaris demo boards. Consult the Luminary Micro Development Kit CD, and tools documentation for more information.

The following steps assume you have installed the FTDI USB drivers and have the Development Board plugged in to a USB cable from your computer (refer to the Luminary Micro GNU Tools QUICKSTART for instructions on setting up the board).

Start the gdb debugger using the following command:

```
# arm-stellaris-eabi-gdb tut5.axf
```

The debugger will start and show the following messages:

```
GNU gdb 6.4.50.20060226-cvs
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "--host=i686-mingw32
--target=arm-stellaris-eabi"...
(gdb)
```

Connect to the target with the following command:

```
(gdb) target extended-remote | armswd –s 2 –f lmi.dll
  stdio
```

(note that the above command can be replaced with the shortcut “connect-r” if you installed .gdbinit as mentioned in the Luminary Micro GNU Tools QUICKSTART)

You should see a message about the target being halted. There may be other messages as well.

To load the program into flash memory in the chip:

```
(gdb) load
```

You should see messages similar to the following:

```
(gdb) load
Loading section .text, size 0x1a98 lma 0x0
Loading section .data, size 0x4 lma 0x1a98
Start address 0x49, load size 6812
Transfer rate: 18165 bits/sec, 296 bytes/write.
(gdb)
```

Once the program has been loaded into flash, you can start it from the debugger by typing “run”. GDB will ask if you would like to restart the application, answer ‘y’. The application will start and then stop at the Reset entry point.

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: C:\Pumpkin\Salvo\Examples\ARM\Luminary_LM3S1XX\
Luminary_DK-LM3S1XX\GCCARM\Tut\Tut5\MyProj/tut5.axf
Peripherals in SoC have been reset
Processor was reset: PC/SP loaded from 0x0.
Stopped at entry point

Program received signal SIGTRAP, Trace/breakpoint trap.
0x00000048 in ResetISR ()
(gdb)
```

Type “continue” to start the program running. You should see the application running on the Development Board. Messages should appear on the LCD panel, and LEDs should be blinking.

Example Projects

Example Salvo projects for use with CodeSourcery Sourcery G++ for Stellaris Family tools can be found in the:

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

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.