Application Report
SWRA446–February 2015

Using GCC/GDB With SimpleLink™ CC26xx/CC13xx

Ida Onshus

ABSTRACT

This application report describes how to set up an open source software-based development environment for the SimpleLink CC26xx and CC13xx wireless MCUs using Eclipse as integrated development environment (IDE).

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/swra446.

Contents

List of Figures

Trademarks

1 Introduction

The SimpleLink CC26xx and CC13xx wireless MCUs are optimized for ultra-low power, while providing fast and capable MCU systems to enable short processing times and high integration. The combination of an up-to 48-MHz ARM® Cortex®-M3 processing core, Flash memory, and a wide selection of peripherals makes the CC26xx/CC13xx device family ideal for single-chip implementation or network processor implementations of lower power RF nodes. For more information about the CC26xx/CC13xx family, see [1].

This document describes which software packages are necessary, and provides installation and configuration instructions for each. It also describes how to use this tool chain setup to debug a software example for a SimpleLink CC26xx device.

Section 2 and Section 3 list the software packages and hardware required for this tool chain setup. Installation procedures for the software packages are provided in Section 4. The next sections describe how to use the tool chain to build, program and debug a software example. Section 5 describes the process of building the project. Section 6 describes how to load the binary image to target and Section 7 describes how to debug the program. Appendix A gives a more detailed description of the makefile used to build the example project, while Appendix B gives a description of the linker and startup file used in the example project.

This setup has been tested on a Windows® 7 and on a Linux Mint™ machine.

Table 1. Abbreviations

CCS	Code Composer Studio™
CDT	C/C++ Development Tooling
EB	Evaluation Board
EM	Evaluation Module
GCC	The GNU Compiler Collection
GDB	The GNU Project Debugger
IDE	Integrated Development Environment
JRE	Java Runtime Environment

2 Prerequisites

2.1 Platforms

This tools setup has been tested on Windows 7 and on Linux Mint 12. Most of the setup instructions will be identical for these platforms. It is noted in the text in the cases where the instructions differ on Linux®.

2.2 Hardware

The following hardware pieces are required:

- SmartRF06EB
- A SimpleLink CC26xxEM
- USB cable

The SmartRF06EB has an integrated XDS100v3 emulator. For more information about the hardware, see [2].

2.3 Software

This section describes the software packages that are required for going through this application report. Tools components, other than the below packages, might be used as well, for example, newer versions of the tools that have become available after this document was written. The below packages and versions are tested together during the writing of this document.

- An IDE. For example, Eclipse and the following components:
  - Java Runtime Environment – JRE
    http://java.com/en/download/index.jsp
  - Eclipse Luna 4.4.1
    http://archive.eclipse.org/eclipse/downloads/drops4/R-4.4.1-201409250400/
  - CDT (C/C++ Development Tooling) version 8.5.0 for Eclipse Luna
    http://www.eclipse.org/cdt/downloads.php
- A GCC Compiler. For example:
  - GNU Tools for ARM Embedded Processors 4.8
    https://launchpad.net/gcc-arm-embedded
- Build tools for Windows. For example:
  - MinGW
    http://www.mingw.org/
- A GDB agent and a Flash programmer tool. For example:
  - For Windows: XDS100v3 EMUpack + GDB agent
    http://processors.wiki.ti.com/index.php/XDS_Emulation_Software_Package
  - For Windows: SmartRF™ Flash Programmer 2
    http://www.ti.com/tool/flash-programmer
  - For Linux: CCS Uniflash Standalone Flash Tool
    http://www.ti.com/tool/uniflash
- Software example:
  - Blink LED software example for GCC (packaged with this document)

Download and installation instructions for each software package are provided in Section 4.

3 Hardware Setup

For instructions on how to configure the hardware, see [2].

4 Software Installation Instructions

4.1 Java Runtime Environment

The Eclipse IDE is dependent on the Java Runtime Environment (JRE) being installed on the machine. Note that both JRE and Eclipse must be downloaded for the same platform (the 64-bit version of Eclipse requires the 64-bit version of JRE). Make sure JRE is installed before performing installation of the Eclipse IDE. If not, it can be downloaded from http://java.com/en/download/index.jsp.

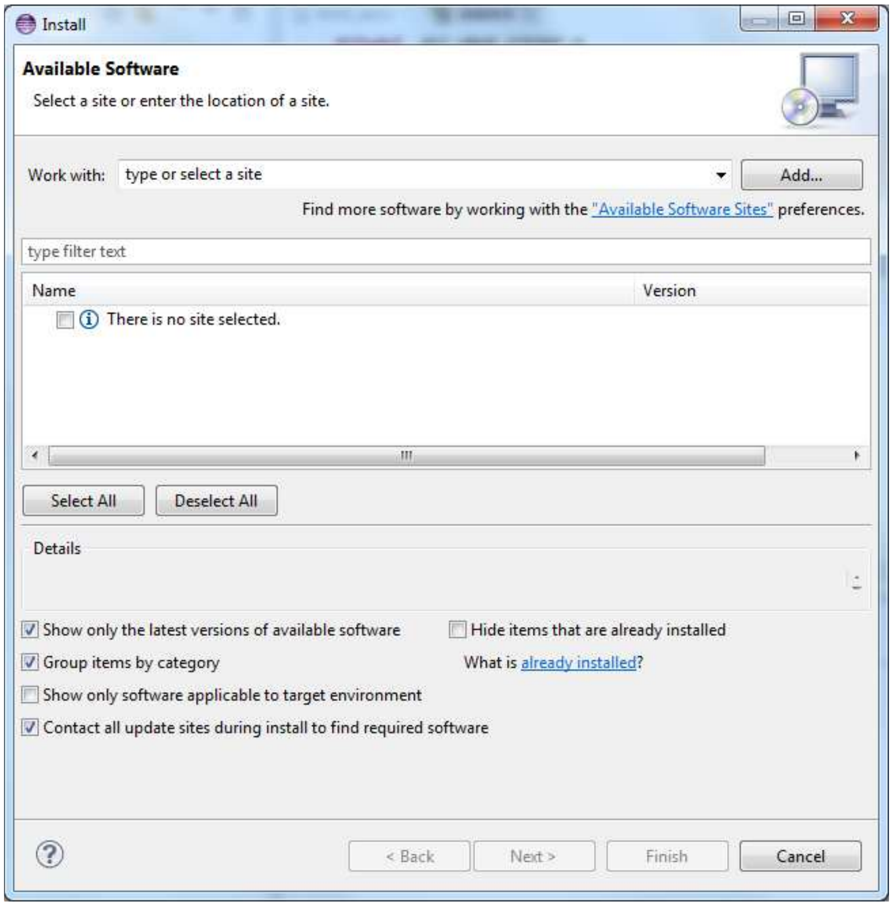
4.2 Eclipse IDE (Windows)

Follow these instructions to install Eclipse and CDT on a Windows platform.

1. Download Eclipse from http://eclipse.org/downloads/index.php.
2. Unzip the downloaded package.
3. The Eclipse IDE can now be started by running the eclipse.exe file. If you are using Windows 7, make sure that Eclipse is run as administrator to allow for the new installations. Eclipse will ask for a folder to use for workspace location when it is started. If Eclipse reports that it cannot find the Java Runtime Environment, make sure that the installation path is added to your environment variables. This is done by going to Control Panel → System and Security → System → Advanced system settings. Append the path of the bin folder in the JRE installation to the PATH variable.
4. Verify the installed versions of the package under Help → About Eclipse SDK.
5. Verify that you use the correct version of the Java Runtime Environment. The version can be found under Window → Preferences → Java → Installed JREs.

   The following instructions describe how to install the CDT plugins:

6. Navigate to Help → Install New Software (see Figure 1).



Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☐ There is no site selected.	

- ✓ Show only the latest versions of available software
- ☐ Hide items that are already installed
- ✓ Group items by category
- What is already installed?
- ☐ Show only software applicable to target environment
- ✓ Contact all update sites during install to find required software

Figure 1. Eclipse - Install New Software (Windows)

7. Type http://download.eclipse.org/tools/cdt/releases/8.5/ (or the address of the software repository for another version of CDT) in the field called 'Work with:'.

8. Select both ‘CDT Main Features’ and ‘CDT Optional Features’ to install (see Figure 2).

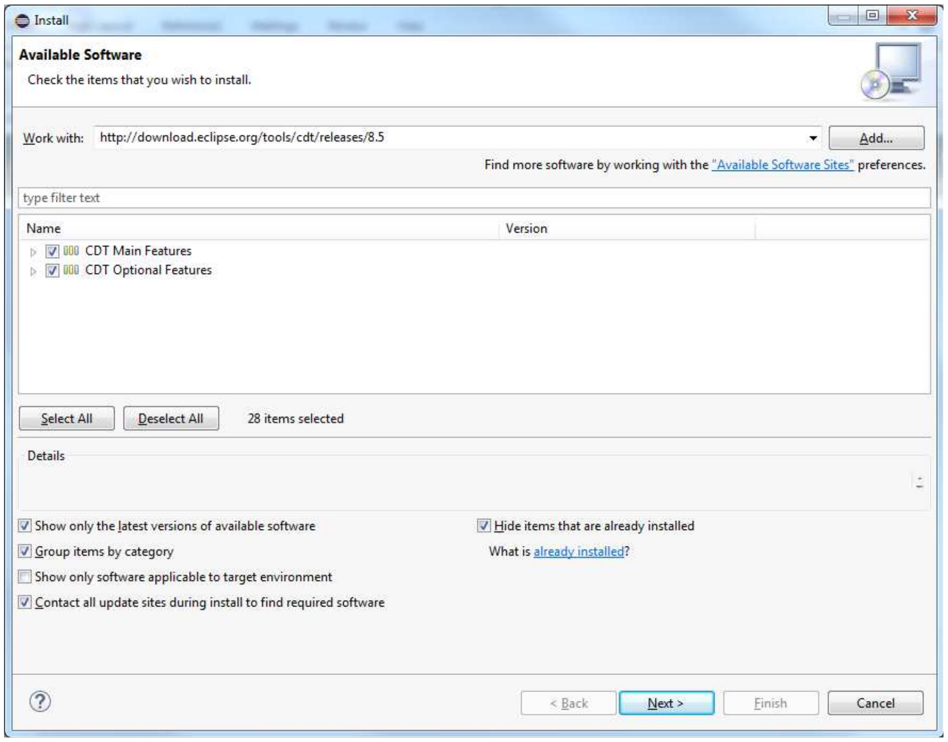
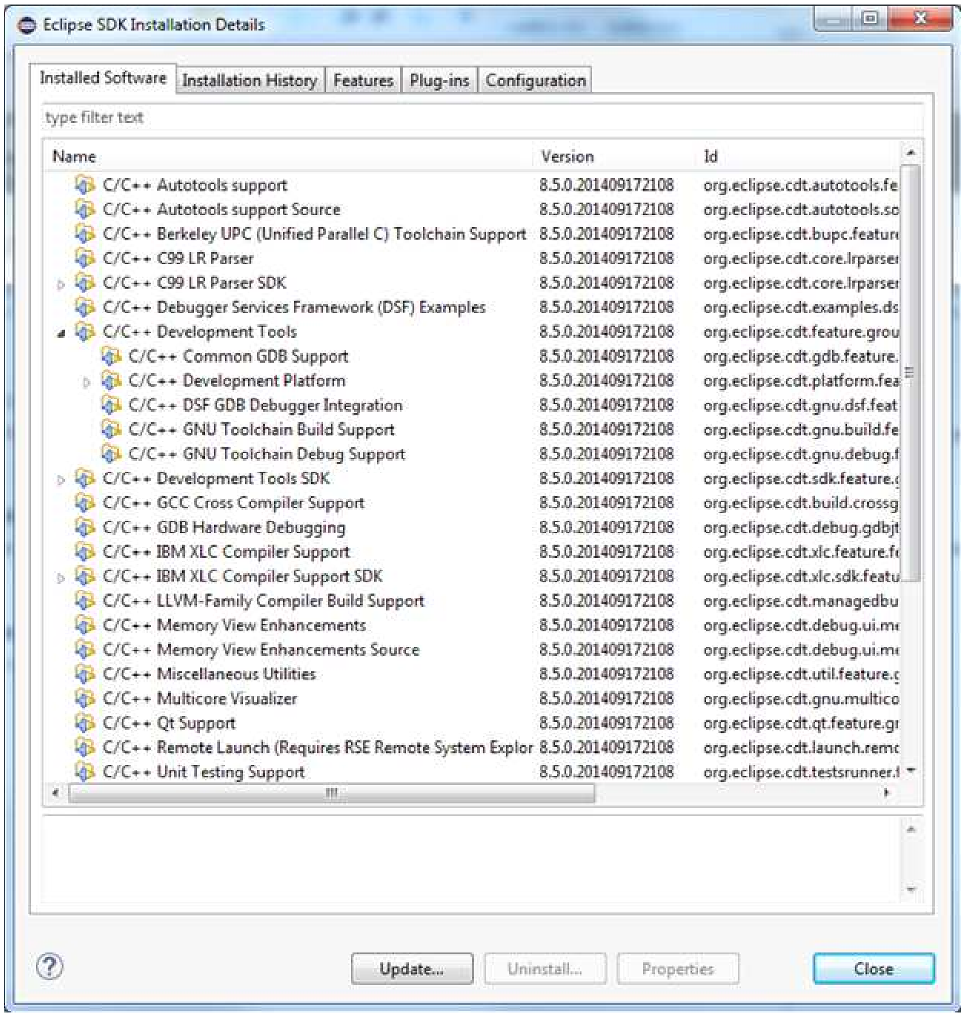

Install
Available Software
Check the items that you wish to install.
Work with:
http://download.eclipse.org/tools/cdt/releases/8.5
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
CDT Main Features
CDT Optional Features
Select All
Deselect All
28 items selected
Details
Show only the latest versions of available software
Group items by category
Show only software applicable to target environment
Contact all update sites during install to find required software
Hide items that are already installed
What is already installed?
< Back
Next >
Finish
Cancel

Figure 2. Installing CDT

9. Click ‘Next’ twice and accept the licence agreements. Click ‘Finish’ to install the CDT package.
10. When finished, click Yes to restart Eclipse.

11. Verify that the CDT components are correctly installed. Navigate to Help → About Eclipse SDK and click 'Installation Details'. Make sure that the C/C++ Development Platform and the C/C++ GDB Hardware Debugging are both installed (see Figure 3).



Name	Version	Id
C/C++ Autotools support	8.5.0.201409172108	org.eclipse.cdt.autotools.fe
C/C++ Autotools support Source	8.5.0.201409172108	org.eclipse.cdt.autotools.so
C/C++ Berkeley UPC (Unified Parallel C) Toolchain Support	8.5.0.201409172108	org.eclipse.cdt.bupc.featur
C/C++ C99 LR Parser	8.5.0.201409172108	org.eclipse.cdt.core.lrparser
C/C++ C99 LR Parser SDK	8.5.0.201409172108	org.eclipse.cdt.core.lrparser
C/C++ Debugger Services Framework (DSF) Examples	8.5.0.201409172108	org.eclipse.cdt.examples.ds
C/C++ Development Tools	8.5.0.201409172108	org.eclipse.cdt.feature.grou
C/C++ Common GDB Support	8.5.0.201409172108	org.eclipse.cdt.gdb.feature.
C/C++ Development Platform	8.5.0.201409172108	org.eclipse.cdt.platform.fea
C/C++ DSF GDB Debugger Integration	8.5.0.201409172108	org.eclipse.cdt.gnu.dsf.feat
C/C++ GNU Toolchain Build Support	8.5.0.201409172108	org.eclipse.cdt.gnu.build.fe
C/C++ GNU Toolchain Debug Support	8.5.0.201409172108	org.eclipse.cdt.gnu.debug.f
C/C++ Development Tools SDK	8.5.0.201409172108	org.eclipse.cdt.sdk.feature.
C/C++ GCC Cross Compiler Support	8.5.0.201409172108	org.eclipse.cdt.build.crossg
C/C++ GDB Hardware Debugging	8.5.0.201409172108	org.eclipse.cdt.debug.gdbjt
C/C++ IBM XLC Compiler Support	8.5.0.201409172108	org.eclipse.cdt.xlc.feature.f
C/C++ IBM XLC Compiler Support SDK	8.5.0.201409172108	org.eclipse.cdt.xlc.sdk.featu
C/C++ LLVM-Family Compiler Build Support	8.5.0.201409172108	org.eclipse.cdt.managedbu
C/C++ Memory View Enhancements	8.5.0.201409172108	org.eclipse.cdt.debug.ui.me
C/C++ Memory View Enhancements Source	8.5.0.201409172108	org.eclipse.cdt.debug.ui.me
C/C++ Miscellaneous Utilities	8.5.0.201409172108	org.eclipse.cdt.util.feature.
C/C++ Multicore Visualizer	8.5.0.201409172108	org.eclipse.cdt.gnu.multico
C/C++ Qt Support	8.5.0.201409172108	org.eclipse.cdt.qt.feature.gr
C/C++ Remote Launch (Requires RSE Remote System Explor	8.5.0.201409172108	org.eclipse.cdt.launch.remc
C/C++ Unit Testing Support	8.5.0.201409172108	org.eclipse.cdt.testsrunner.

Figure 3. Installation Details

4.3 *Eclipse IDE (Linux)*

Follow these instructions to install Eclipse and CDT on a Linux platform.

1. Download Eclipse from http://eclipse.org/downloads/index.php. Choose the installer for your Linux platform.
2. Extract the download package.
3. The Eclipse IDE can now be started by running the binary file eclipse. Make sure that Eclipse is run with administrator privileges to allow for the new installations. Eclipse will ask for a folder to use for workspace location when it is started. If Eclipse reports that it cannot find the Java Runtime Environment, open the eclipse.ini file and add the following two lines direct under the 'openFile' statement:

```
-vm
<path to your JRE installation>/bin/java
```

4. Verify the installed versions of the package under Help → About Eclipse SDK.
5. Verify that you use the correct version of the Java Runtime Environment. This can be checked under Window → Preferences → Java → Installed JREs.

The following instructions describe how to install the CDT plugins:

6. Navigate to Help → Install New Software (see Figure 4).

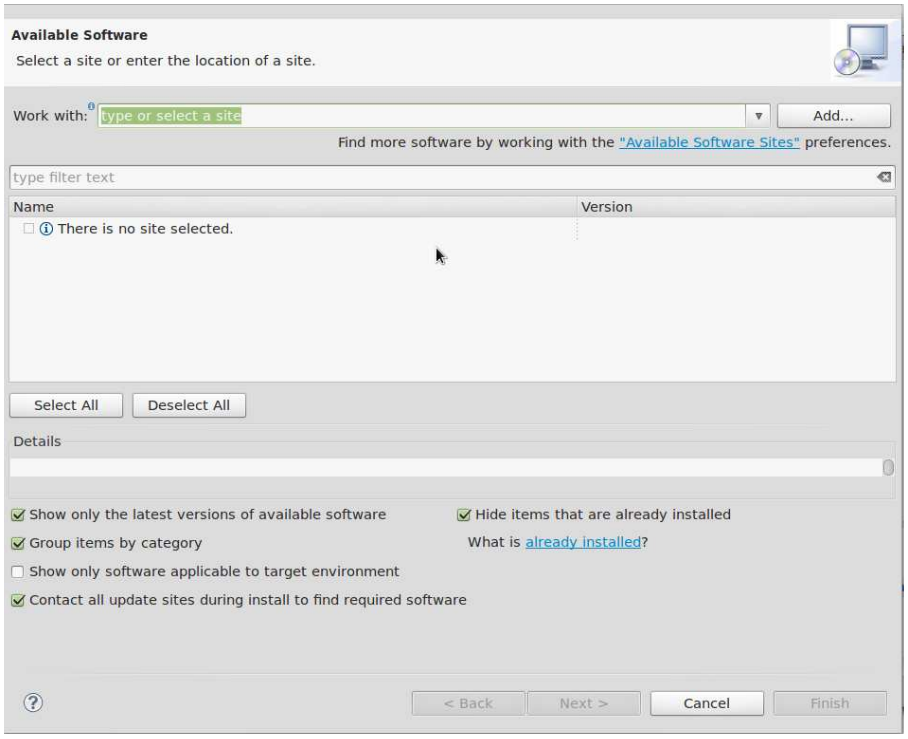

Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
There is no site selected.
Select All
Deselect All
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Cancel
Finish

Figure 4. Eclipse - Install New Software (Linux)

7. Type http://download.eclipse.org/tools/cdt/releases/8.5/ (or the address of the software repository for another version of CDT) in the field called 'Work with:'.
8. If there is a problem with finding the repository site, this might be because of incorrect proxy settings on the system. These can be configured in the Eclipse IDE under Window → Preferences → General → Network Connections.

   An alternative is to download the CDT manually from http://www.eclipse.org/cdt/downloads.php. Choose the archive file called cdt-x.x.x.zip. Select Add → Archive… and browse to your downloaded archive file. Name it CDT and click OK (see Figure 5).

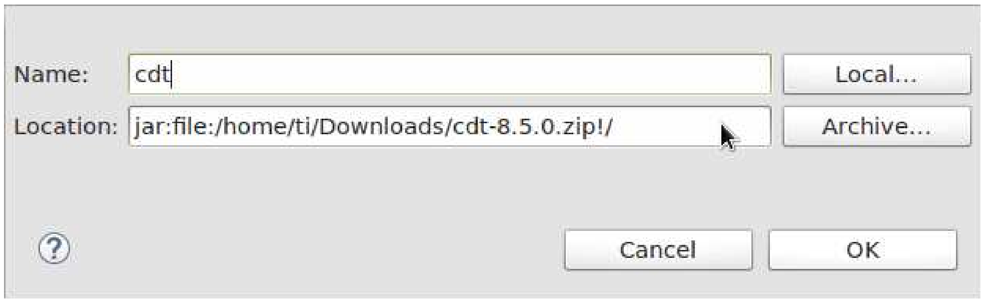

Name: cdt
Location: jar:file:/home/ti/Downloads/cdt-8.5.0.zip!/

Figure 5. Installing CDT (Linux)

9. Follow step 3 in Section 4.2. These final steps for Windows and Linux are equal.

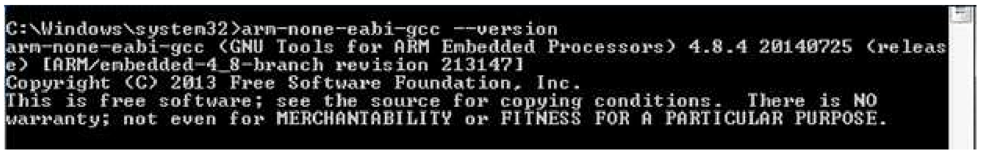
4.4 GNU Toolchain (Windows)

GNU Tools for ARM Embedded Processors is a free package that includes the GNU compiler and linker tools as well as the GNU debugger. Perform the following steps for Windows installation:

1. Download GNU Tools for ARM Embedded Processors. The package can be found on this site: https://launchpad.net/gcc-arm-embedded.
2. Run the installer and follow the instructions. In the last step, make sure to select the checkbox 'Add path to environment variable'.
3. Verify from the command prompt that the package is installed is updated correctly. This can be done by clicking Start → Run… . Type in cmd and click OK. In the command window, write:

   ```
   arm-none-eabi-gcc --version
   ```

   The expected output is shown in Figure 6. If the arm-none-eabi-gcc program is not found, check that the PATH variable is correctly set.



```
C:\Windows\system32>arm-none-eabi-gcc --version
arm-none-eabi-gcc (GNU Tools for ARM Embedded Processors) 4.8.4 20140725 (releas
e) [ARM/embedded-4_8-branch revision 213147]
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Figure 6. GNU Toolchain (Windows)

More information about this package can be found in [3].

4.5 GNU Toolchain (Linux)

Perform the following steps for the Linux installation of GNU Tools for ARM Embedded Processors:

1. Download GNU Tools for ARM Embedded Processors. The package can be found on this site : https://launchpad.net/gcc-arm-embedded.
2. Unpack the tarball to the install directory, with the following command:

   ```
   $ cd <install_dir> && tar xjf gcc-arm-none-eabi-*-yyyymmdd-
   linux.tar.bz2
   ```

3. Add the installation path to the PATH variable with the following command:

   ```
   $ export PATH=$PATH:<install_dir>/gcc-arm-none-eabi-*/bin
   ```

4. Check that the correct version is installed with the command:

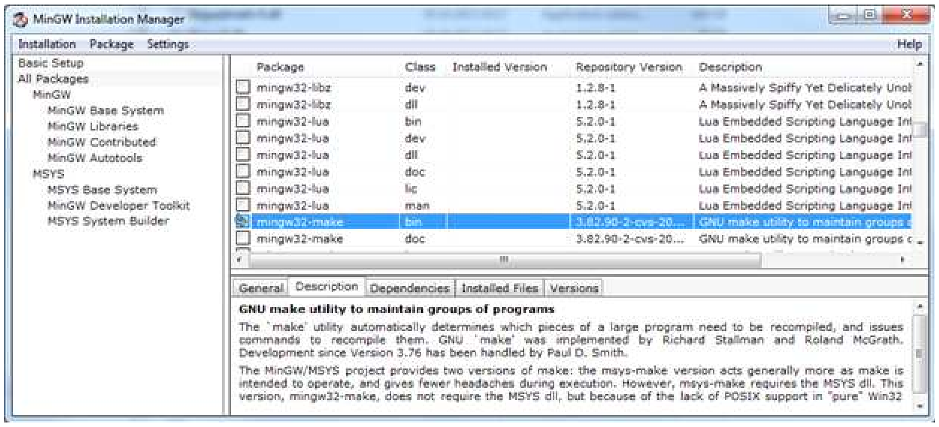
   ```
   $ arm-none-eabi-gcc –v
   ```

   More information about this package can be found in [3].

4.6 Build Tools for Windows

If your development platform is Windows, an additional command line program, ‘make’, is required by the Eclipse external builder. One choice is to download the MinGW tool. Perform the following steps for installation:

1. Download the MinGW setup tool from http://www.mingw.org/.
2. Run the Setup. Choose default installation directory.
3. Click Continue when the setup is finished to open the installation manager.
4. In the Installation Manager, mark ‘All Packages’ in the left panel. Click on the checkboxes for the packages called mingw32-make and mingw32-gcc and select ‘mark for installation’.



Package	Class	Installed Version	Repository Version	Description
mingw32-libz	dev		1.2.8-1	A Massively Spiffy Yet Delicately Unol
mingw32-libz	dll		1.2.8-1	A Massively Spiffy Yet Delicately Unol
mingw32-lua	bin		5.2.0-1	Lua Embedded Scripting Language Inl
mingw32-lua	dev		5.2.0-1	Lua Embedded Scripting Language Inl
mingw32-lua	dll		5.2.0-1	Lua Embedded Scripting Language Inl
mingw32-lua	doc		5.2.0-1	Lua Embedded Scripting Language Inl
mingw32-lua	lic		5.2.0-1	Lua Embedded Scripting Language Inl
mingw32-lua	man		5.2.0-1	Lua Embedded Scripting Language Inl
mingw32-make	bin		3.82.90-2-cvs-20...	GNU make utility to maintain groups o
mingw32-make	doc		3.82.90-2-cvs-20...	GNU make utility to maintain groups c

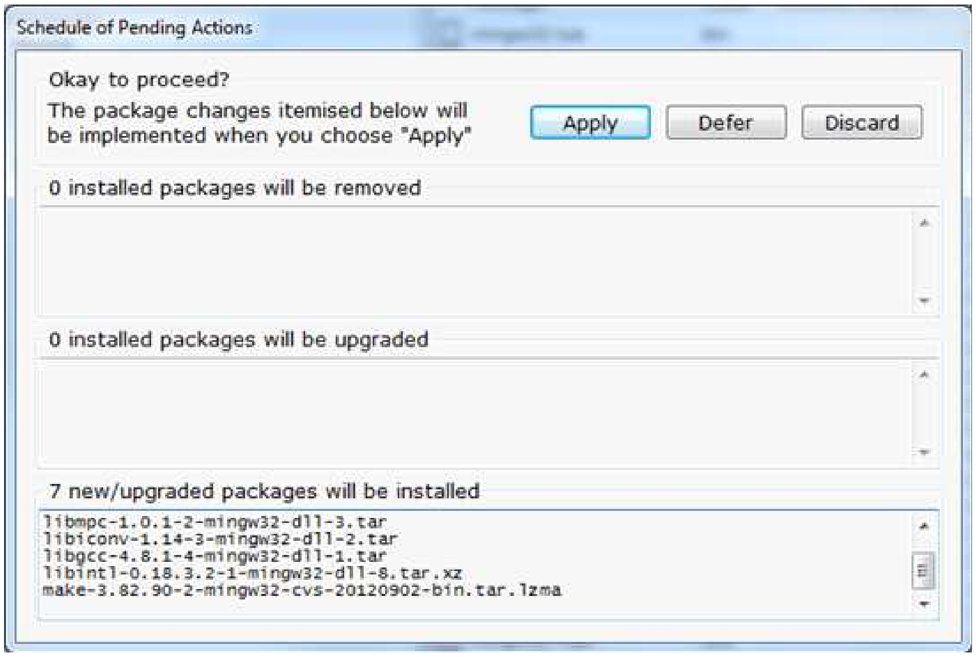
GNU make utility to maintain groups of programs

The `make' utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them. GNU `make' was implemented by Richard Stallman and Roland McGrath. Development since Version 3.76 has been handled by Paul D. Smith.

The MinGW/MSYS project provides two versions of make: the msys-make version acts generally more as make is intended to operate, and gives fewer headaches during execution. However, msys-make requires the MSYS dll. This version, mingw32-make, does not require the MSYS dll, but because of the lack of POSIX support in "pure" Win32

Figure 7. Installing MinGW (Windows)

5. Choose Installation → Apply Changes and click the 'Apply' button.



Okay to proceed?

The package changes itemised below will be implemented when you choose "Apply"

0 installed packages will be removed

0 installed packages will be upgraded

7 new/upgraded packages will be installed

```
libmpc-1.0.1-2-mingw32-dll-3.tar
libiconv-1.14-3-mingw32-dll-2.tar
libgcc-4.8.1-4-mingw32-dll-1.tar
libintl-0.18.3.2-1-mingw32-dll-8.tar.xz
make-3.82.90-2-mingw32-cvs-20120902-bin.tar.lzma
```

Figure 8. Installing MinGW (Windows)

6. Finally, add the installation path (default: C:\MinGW\bin) to your environment variables. This is done by going to Control Panel → System and Security → System → Advanced system settings. Append the installation path to the PATH variable.

4.7 TI Emupack and GDB Server (Windows)

The following two steps are necessary only for Windows:

1. Download the XDS Emulation Software Package from the following location: http://processors.wiki.ti.com/index.php/XDS_Emulation_Software_Package.

   This includes drivers for the XDS emulators and the GDB server.
2. Run the installer as an administrator. Select the typical installation and default installation directory.

4.8 Flash Programmer (Windows)

The following two steps describe the installation of the Flash programmer tool for Windows:

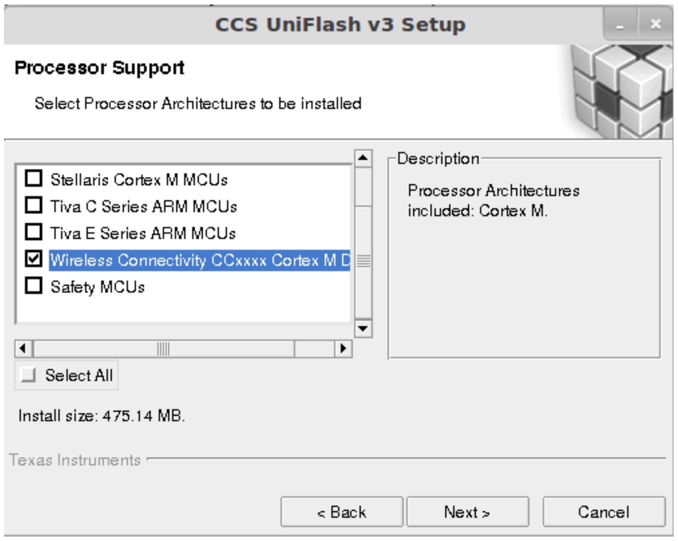
1. Download SmartRF Flash Programmer 2 from the following location: http://www.ti.com/tool/flash-programmer.
2. Unzip the downloaded file and run the installer.

4.9 Flash Programmer (Linux)

The following three steps describe the installation of the Flash programmer tool for Linux:

1. Download CCS Uniflash Flash programmer for Linux from the following location: http://www.ti.com/tool/uniflash.

   CCS Uniflash includes the TI Emupack and GDB Server.
2. Run the installer (uniflash_setup_x.x.x.xxxxx.bin). Select custom installation and make sure that support for Wireless Connectivity devices is installed (see Figure 9).



- [ ] Stellaris Cortex M MCUs
- [ ] Tiva C Series ARM MCUs
- [ ] Tiva E Series ARM MCUs
- [x] Wireless Connectivity CCxxxx Cortex M D
- [ ] Safety MCUs

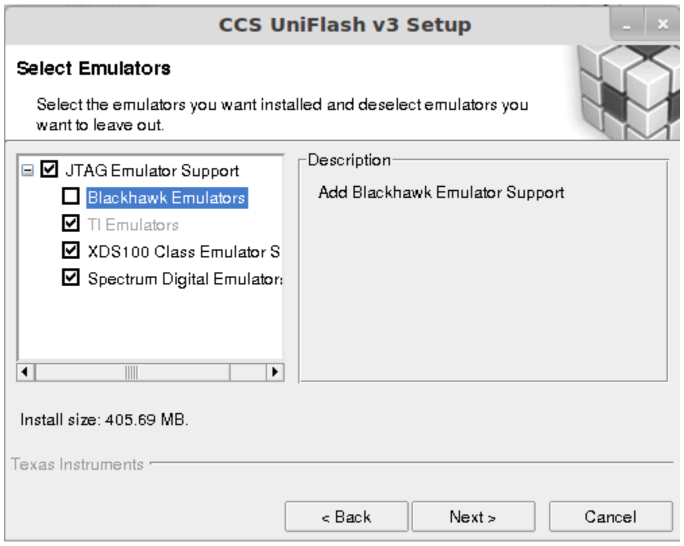
Description

Processor Architectures included: Cortex M.

Install size: 475.14 MB.

Figure 9. UniFlash Installation

3. Make sure TI emulators, XDS100 class and Spectrum Digital emulator support are installed (see Figure 10).



Select Emulators

Select the emulators you want installed and deselect emulators you want to leave out.

- [✓] JTAG Emulator Support
  - [ ] Blackhawk Emulators
  - [✓] TI Emulators
  - [✓] XDS100 Class Emulator S
  - [✓] Spectrum Digital Emulator:

Description

Add Blackhawk Emulator Support

Install size: 405.69 MB.

Texas Instruments

Figure 10. UniFlash Installation

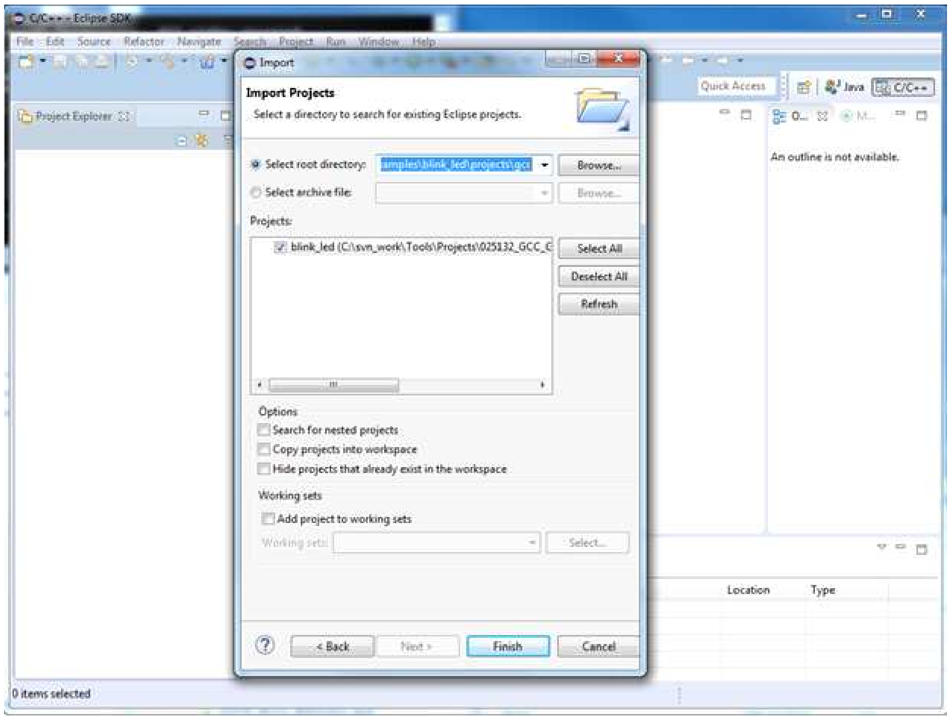
4. If the installed version of TI EmuPack is older than ti_emupack_5.1.635.0, open UniFlash, click Help → Check for Updates, and install the latest version. In the older version, there was a problem using the GDB agent on Linux.

5 Build the Software Example

This section describes how to compile and build the example project in Eclipse. The example can also easily be compiled and built without Eclipse by running the makefile rule ‘all’. The makefile used to build the project is described in more details in Appendix A.

5.1 Import Example Project Into the IDE

1. Open the Eclipse IDE. If the Welcome screen is shown, click on the Workbench icon to open the workbench. Then, click Window → Open Perspective → Other… and select the C/C++ perspective.
2. Navigate to File → Import…
3. Select ‘General’ and ‘Existing Projects into Workspace.’ Click 'Next'.
4. Choose Select root directory, click Browse and navigate to the directory of the example project. Select the project. Make sure that the option 'Copy projects into workspace' is not checked (see Figure 11). Then click Finish.



Import Projects

Select a directory to search for existing Eclipse projects.

(●) Select root directory: samples\blink_led\projects\gcc

( ) Select archive file:

Projects:

[✓] blink_led (C:\svn_work\Tools\Projects\025132_GCC_G

Options

[ ] Search for nested projects

[ ] Copy projects into workspace

[ ] Hide projects that already exist in the workspace

Working sets

[ ] Add project to working sets

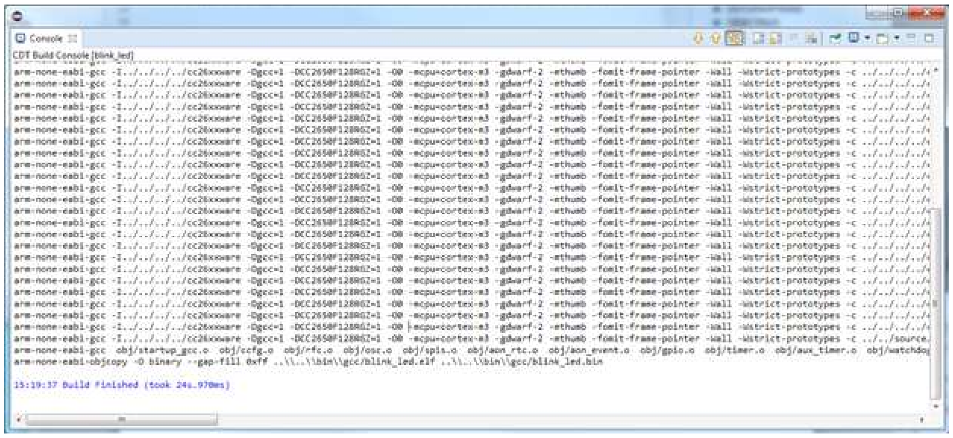
Working sets:

Figure 11. Import Example Project Into IDE

5.2 Build the Software Example

1. On Windows, specify the make tool you are using. Navigate to Project → Properties → C/C++ build. Under the tab 'Builder Settings', uncheck 'Use default build command' and type in the build command you are using. If you are using the MinGW tools, the command will be mingw32-make.
2. If you have a device that is different from the CC2650F128RGZ device, open the file called 'makedefs' in the project directory and change the variable called CHIP_ID to the ID that suits your device.
3. Right click the project and select 'Build Project'. This will create the binary files blink_led.bin and blink_led.elf.

   On Windows, the console output from the build command should be as shown in Figure 12.

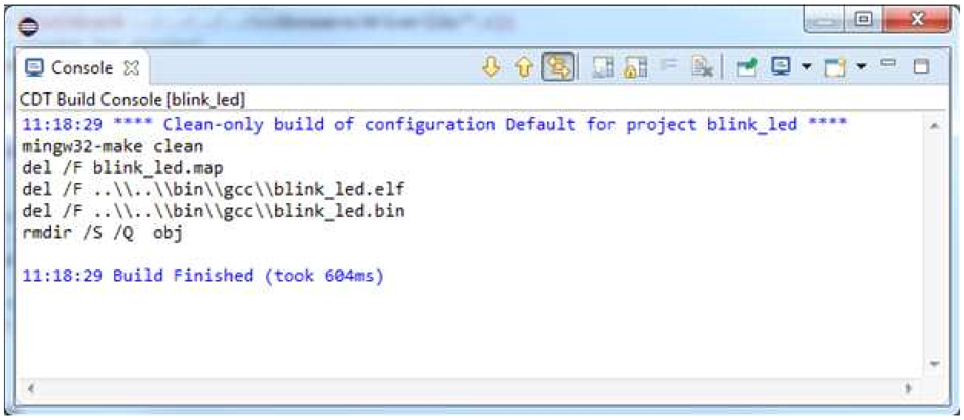


```
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../../../
arm-none-eabi-gcc -I../../../../cc26xxware -Dgcc=1 -DCC2650F128RGZ=1 -O0 -mcpu=cortex-m3 -gdwarf-2 -mthumb -fomit-frame-pointer -Wall -Wstrict-prototypes -c ../../source
arm-none-eabi-gcc  obj/startup_gcc.o  obj/ccfg.o  obj/rfc.o  obj/osc.o  obj/spis.o  obj/aon_rtc.o  obj/aon_event.o  obj/gpio.o  obj/timer.o  obj/aux_timer.o  obj/watchdog
arm-none-eabi-objcopy -O binary --gap-fill 0xff ..\\..\\bin\\gcc/blink_led.elf ..\\..\\bin\\gcc/blink_led.bin

15:19:37 Build Finished (took 24s.970ms)
```

Figure 12. Build Project

NOTE: On Linux, if the build fails and arm-none-eabi-gcc is not found, you must edit the file called *makedefs*. Modify the CC and OBJCOPY variables by adding the full path to arm-none-eabi-gcc and to arm-none-eabi-objcopy. These tools are both found in the bin folder under where GNU Tools for ARM Embedded Processors was installed.

4. Right click blink_led and select 'Clean Project' if you want to clean the project. On Windows, the console output from the build command should be as shown in Figure 13.



```
11:18:29 **** Clean-only build of configuration Default for project blink_led ****
mingw32-make clean
del /F blink_led.map
del /F ..\\..\\bin\\gcc\\blink_led.elf
del /F ..\\..\\bin\\gcc\\blink_led.bin
rmdir /S /Q  obj

11:18:29 Build Finished (took 604ms)
```

Figure 13. Clean Project

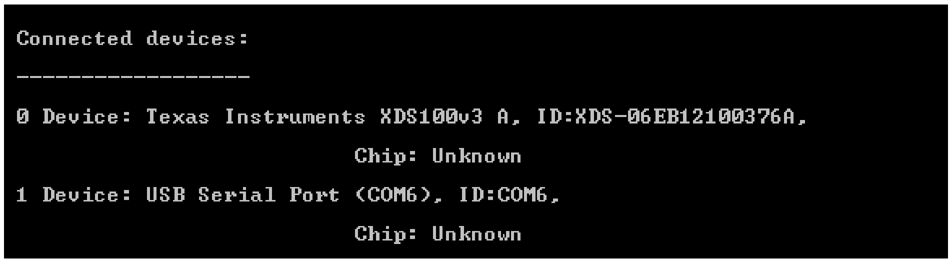
6 Load Binary Image to Target

This section describes how to load the binary image created when building the software example, to the target. Both the .bin file and the .elf file can be loaded to target, but if your Flash programmer tool support .elf files, this will be faster.

6.1 Configure Flash Programmer Tool (Windows)

This section describes how to configure the SmartRF Flash Programmer tool for the Eclipse environment on Windows. The Flash programmer tool is used to load a generated binary image to the target before debugging.

1. Start SmartRF Flash Programmer 2 GUI application (installation is described in Section 4.8).
2. Start command line by clicking the tools symbol in the top right corner of the GUI.
3. Make sure the hardware is connected on the USB and type `srfprog –ls all` in the command line to detect the connected hardware. If there is one SmartRF EB board connected on USB the output will be similar to the output shown in Figure 14.



```
Connected devices:

------------------

0 Device: Texas Instruments XDS100v3 A, ID:XDS-06EB12100376A,

                         Chip: Unknown

1 Device: USB Serial Port (COM6), ID:COM6,

                         Chip: Unknown
```

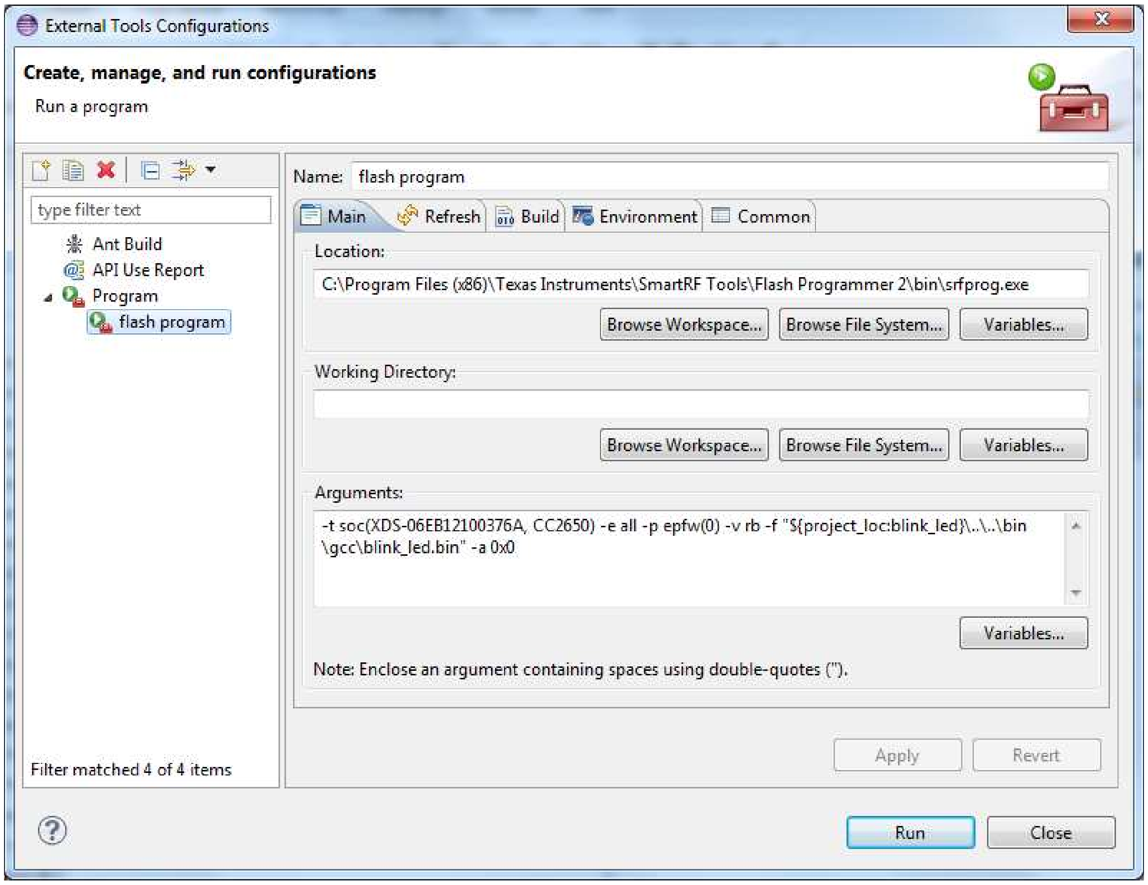
Figure 14. Connected Devices

4. Write down the ID of the detected EB board (in the above example it is XDS-06EB12100376A).
5. In Eclipse, navigate to Run → External Tools → External Tools Configurations.
6. Right click on 'Program' in the left side panel and select 'New' to create a new Tools configuration.
7. Fill in the fields 'Name', 'Location' and 'Arguments' as shown in Figure 15.
   a. Location should be set to the directory where the SmartRF Flash programmer 2 was installed. The location below is correct when default installation directory is used.
   b. In the Arguments field, type in the following code:

```
-t soc(XDS-06EB12100376A, CC2650) -e all -p epfw(0) -v rb -f
"${project_loc:blink_led}\..\..\bin\gcc\blink_led.bin" -a 0x0
```

   Make sure to use the same XDS ID number as noted in step 4.

   What the command means is:
   -t: specified target
   -e all: Erase all pages
   -p epfw(0): program, but exclude pages filled with 0x00
   -v rb: verify using readback
   -f: file to program
   -a 0x0: start address is 0x00



Create, manage, and run configurations

Run a program

type filter text

Name: flash program

Location:

C:\Program Files (x86)\Texas Instruments\SmartRF Tools\Flash Programmer 2\bin\srfprog.exe

Working Directory:

Arguments:

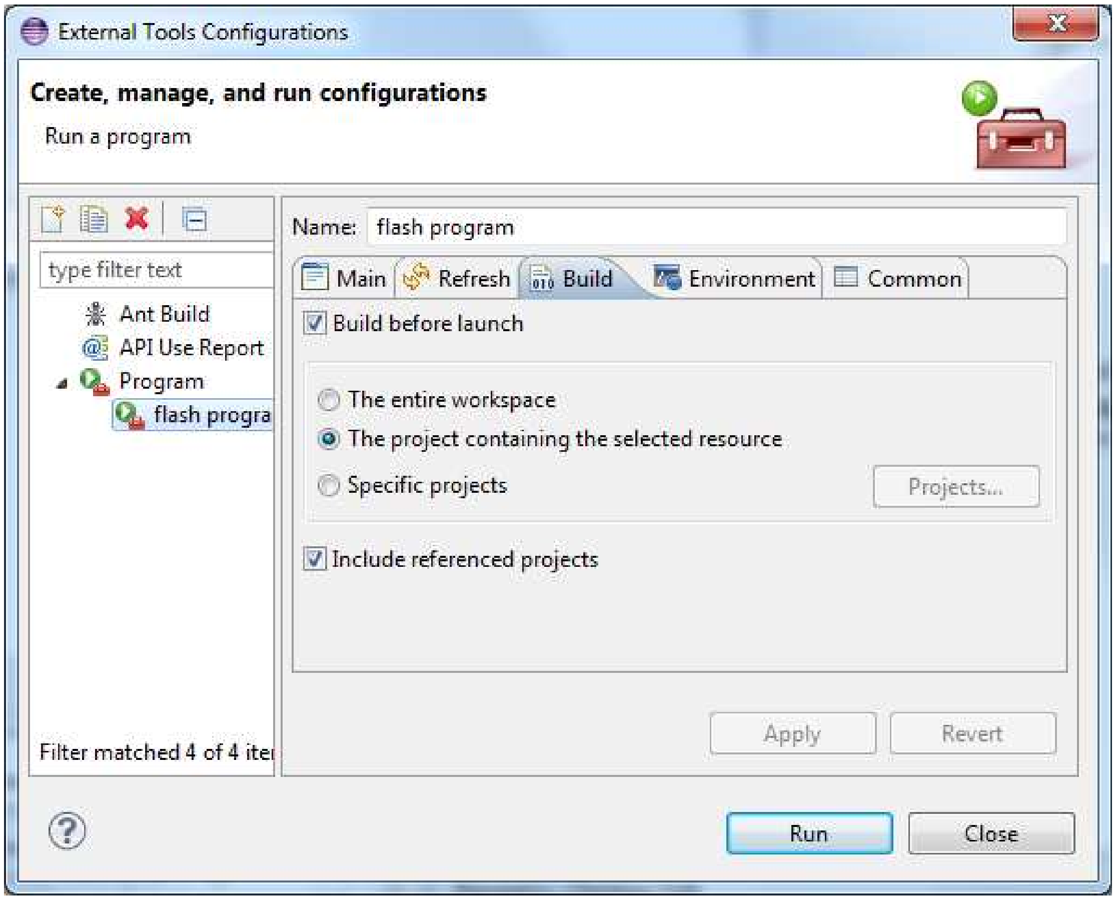
```
-t soc(XDS-06EB12100376A, CC2650) -e all -p epfw(0) -v rb -f "${project_loc:blink_led}\..\..\bin
\gcc\blink_led.bin" -a 0x0
```

Note: Enclose an argument containing spaces using double-quotes (").

Filter matched 4 of 4 items

Figure 15. Flash Programming Configuration

8. If you have several projects in your workspace, select the 'Build' tab, and choose to build 'The project containing the selected resource', instead of building the complete workspace (see Figure 16).



Name: flash program

[✓] Build before launch

- ○ The entire workspace
- ● The project containing the selected resource
- ○ Specific projects

[✓] Include referenced projects

Figure 16. Flash Programming Configuration

9. Press 'Apply'.
10. Each EB has its own ID, so if you change to another SmartRF EB board, repeat step 1 to 4 and change the ID used in the 'Arguments' field.

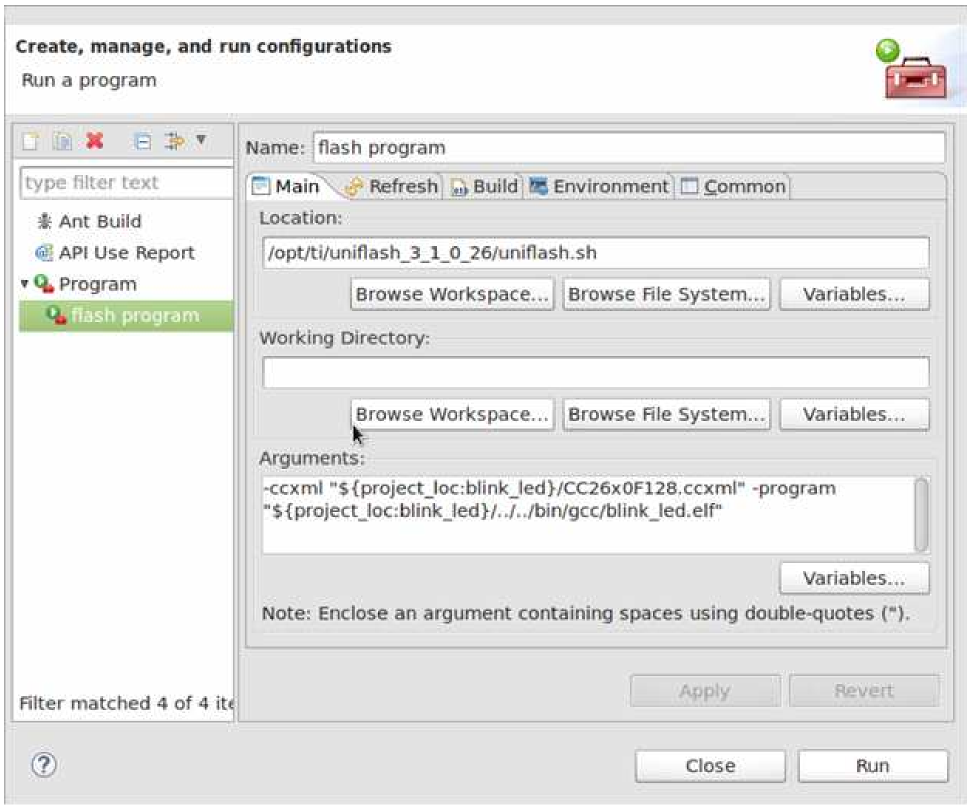
6.2 Configure Flash Programmer Tool (Linux)

This section describes how to configure the CCS UniFlash Flash programmer tool in the Eclipse environment for Linux.

1. Create a target configuration file for your device. Open CCS Uniflash. Select File → New Target Configuration.
2. Select Texas Instruments XDS100v3 USB Emulator in the ‘Connection Field. In the ‘Board or Device’ field, select the device you are working with. Then click OK.
3. Select File → Save Target Configuration and save your target configuration file in the project folder of the blink-led example. In this application report, it is assumed that the file has been named CC26x0F128.ccxml.
4. In Eclipse, Navigate to Run → External Tools → External Tools Configurations.
5. Right click on 'Program' in the left side panel and select 'New' to create a new Tools configuration.
6. Fill in the fields 'Name', 'Location' and 'Arguments' as shown in Figure 17.
   a. Location should be set to the directory where the CCS UniFlash was installed.
   b. In the 'Arguments' field, type in the following code:

```
-ccxml “${project_loc:blink_led}/CC26x0F128.ccxml” –program
“${project_loc:blink_led}/../../bin/gcc/blink_led.elf”
```

   blink_led.elf is the output of the project, created during build. CC26x0F128.ccxml is the target configuration file that was created by UniFlash.



Name: flash program

Location:
/opt/ti/uniflash_3_1_0_26/uniflash.sh

Working Directory:

Arguments:
```
-ccxml "${project_loc:blink_led}/CC26x0F128.ccxml" -program
"${project_loc:blink_led}/../../bin/gcc/blink_led.elf"
```

Note: Enclose an argument containing spaces using double-quotes (").

Filter matched 4 of 4 ite

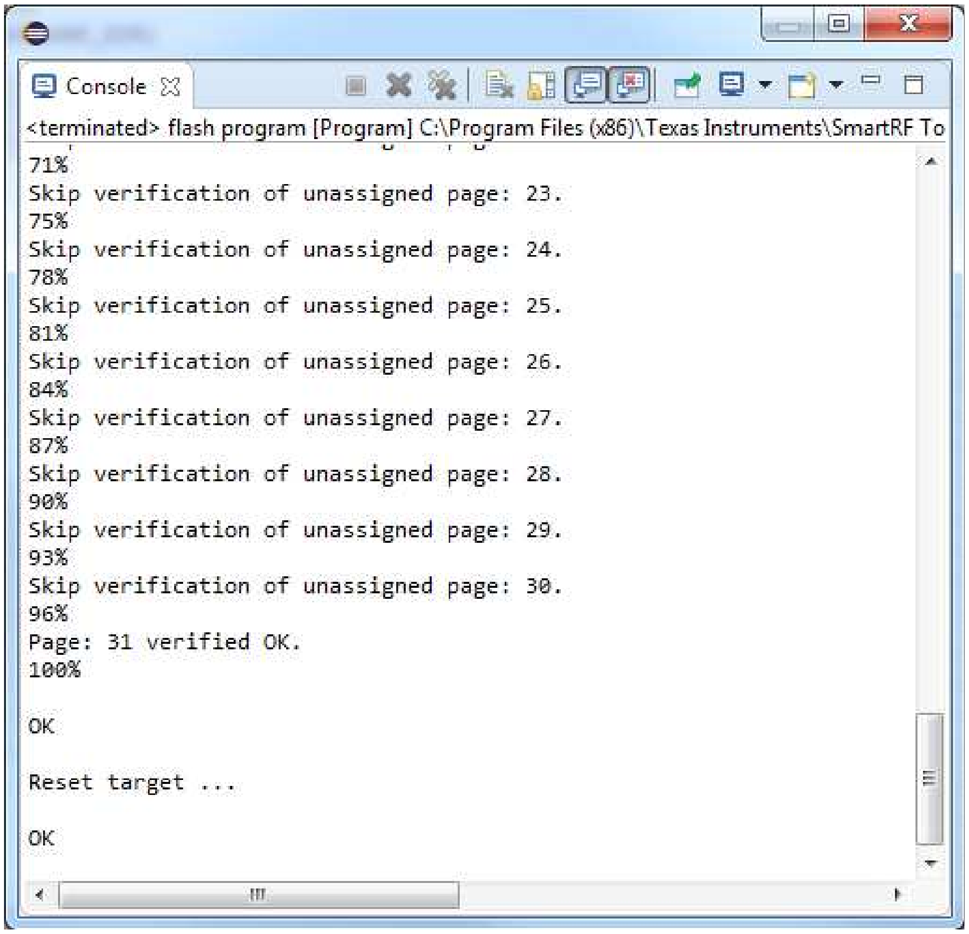
Figure 17. Configuring the Flash Programmer Tool (Linux)

7. If you have several projects in your work space, select the 'Build' tab, and choose to build 'The project containing the selected resource', instead of building the complete workspace.
8. Press 'Apply'.

6.3 Load the Image to Target (Windows)

For Windows, the binary image that was generated in the previous section will be loaded to the target using the SmartRF flash Programmer 2 tool.

Load the binary image to the target by running the Flash Programmer tool: Run → External Tools → External Tools Configurations ... . Select Flash program under 'Program' in the left side panel and click 'Run'. The console output should be similar to the output shown in Figure 18.



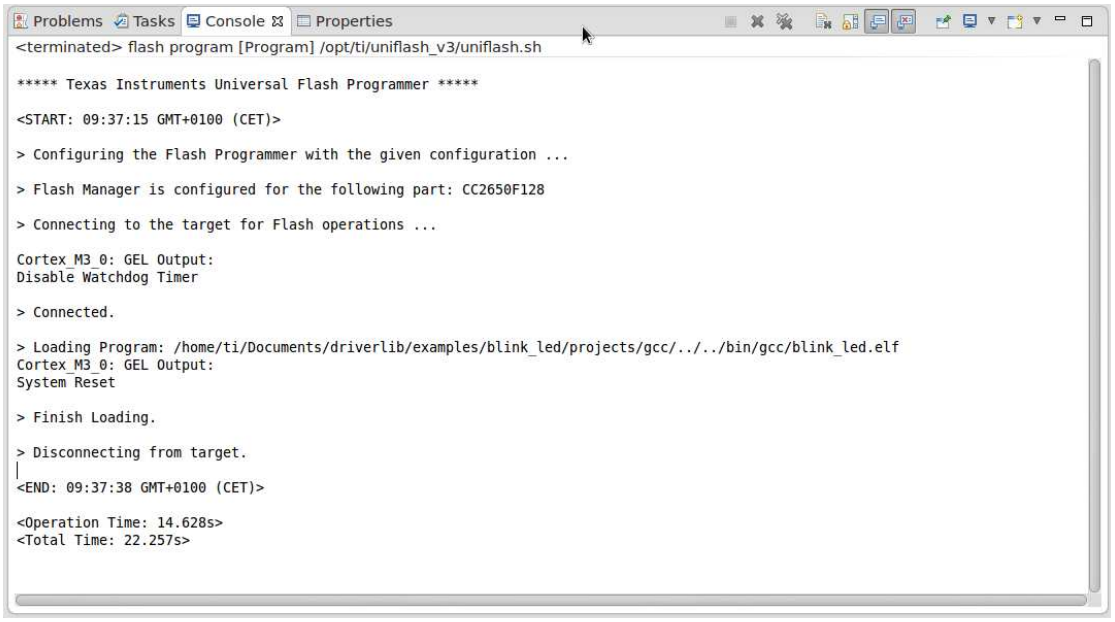
```
<terminated> flash program [Program] C:\Program Files (x86)\Texas Instruments\SmartRF To
71%
Skip verification of unassigned page: 23.
75%
Skip verification of unassigned page: 24.
78%
Skip verification of unassigned page: 25.
81%
Skip verification of unassigned page: 26.
84%
Skip verification of unassigned page: 27.
87%
Skip verification of unassigned page: 28.
90%
Skip verification of unassigned page: 29.
93%
Skip verification of unassigned page: 30.
96%
Page: 31 verified OK.
100%

OK

Reset target ...

OK
```

Figure 18. Load the Image to Target (Windows)

6.4 Load the Image to Target (Linux)

For Linux, the *.elf file that was generated in the previous section will be loaded to the target using the Uniflash Flash programming tool.

Load the binary image to the target by running the Flash Programmer tool: Run → External Tools → External Tools Configurations… . Select Flash program under 'Program' in the left side panel and click 'Run'. The console output should be similar to the output shown in Figure 19.



```
<terminated> flash program [Program] /opt/ti/uniflash_v3/uniflash.sh

***** Texas Instruments Universal Flash Programmer *****

<START: 09:37:15 GMT+0100 (CET)>

> Configuring the Flash Programmer with the given configuration ...

> Flash Manager is configured for the following part: CC2650F128

> Connecting to the target for Flash operations ...

Cortex_M3_0: GEL Output:
Disable Watchdog Timer

> Connected.

> Loading Program: /home/ti/Documents/driverlib/examples/blink_led/projects/gcc/../../bin/gcc/blink_led.elf
Cortex_M3_0: GEL Output:
System Reset

> Finish Loading.

> Disconnecting from target.

<END: 09:37:38 GMT+0100 (CET)>

<Operation Time: 14.628s>
<Total Time: 22.257s>
```

Figure 19. Load the Image to Target (Linux)

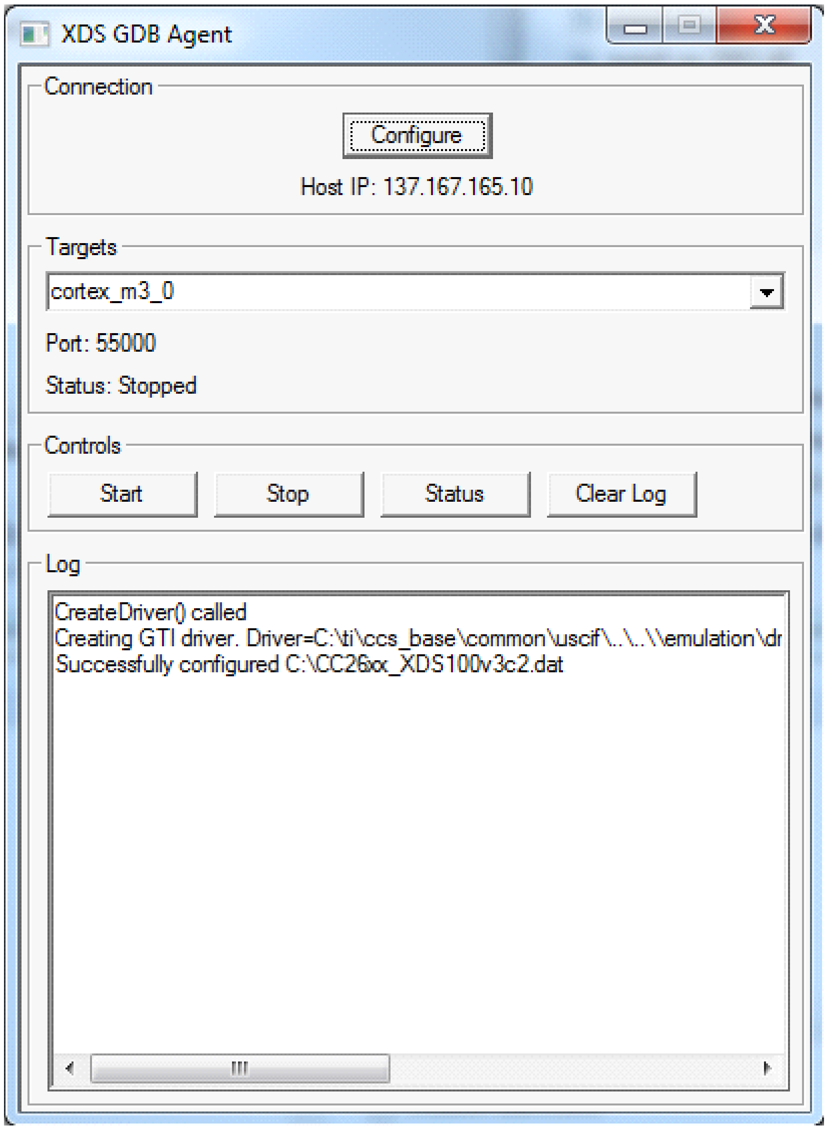
7 Debug the Software Example

This section describes how to configure the debugger and debug the software example using Eclipse and GNU Tools for ARM Embedded Processors.

7.1 Launch the GDB Server (Windows)

The GDB Server agent is found in the folder where the TI EmuPack is installed. If it is installed in default directory, it can be found in the folder C:\ti\ccs_base\common\uscif. Perform the following steps to launch and configure the GDB Server for the XDS100v3 emulator on Windows:

1. Launch the GDB Server agent: gdb_agent_gui.exe.
2. Press the 'Configure' button and select the board configuration *.dat file for your target. The *.dat file for this target is found together with the CC26xx software example. For Windows, use the file CC26xx_XDS100v3c2.dat.
3. Next, press the 'Start' button (see Figure 20). The GDB server agent is now started.



Host IP: 137.167.165.10

cortex_m3_0

Port: 55000

Status: Stopped

```
CreateDriver() called
Creating GTI driver. Driver=C:\ti\ccs_base\common\uscif\..\..\emulation\dr
Successfully configured C:\CC26xx_XDS100v3c2.dat
```

Figure 20. Launching the GDB Server (Windows)

7.2 Launch the GDB Server (Linux)

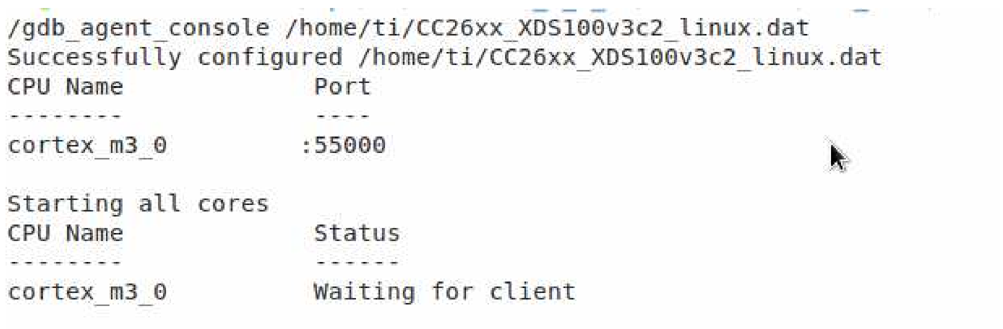
The GDB server agent is found in the following folder where UniFlash is installed: ccs_base\common\uscif. Perform the following steps to launch and configure the GDB Server for the XDS100v3 emulator on Linux:

1. Launch the GDB Server agent with the following command:

   ```
   gdb_agent_console <board-data-file>
   ```

   The board-data-file is the *.dat file found in the software example package: CC26xx_XDS100v3c2_linux.dat.

2. If launched successfully, the output shown in Figure 21 can be seen in the console.



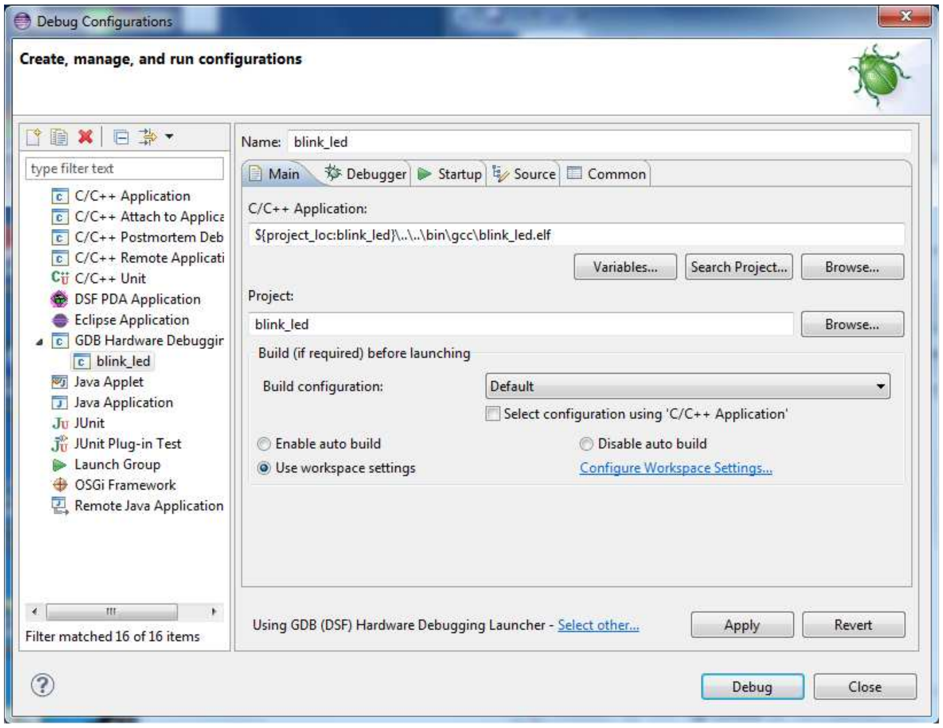
```
/gdb_agent_console /home/ti/CC26xx_XDS100v3c2_linux.dat
Successfully configured /home/ti/CC26xx_XDS100v3c2_linux.dat
CPU Name             Port
--------             ----
cortex_m3_0         :55000

Starting all cores
CPU Name             Status
--------             ------
cortex_m3_0          Waiting for client
```

Figure 21. Launching the GDB Server (Linux)

7.3 Configure Eclipse Debugger

This section describes how to configure the Eclipse debugger.

1. In the Project Explorer, right click blink_led and select Debug As → Debug Configurations… The Debug Configurations window will now be shown.
2. Right click 'GDB Hardware Debugging' and select 'New'.
3. Make sure that the settings in the 'Main' tab are configured as in Figure 22.



Name: blink_led

C/C++ Application:
${project_loc:blink_led}\..\..\bin\gcc\blink_led.elf

Project:
blink_led

Build (if required) before launching

Build configuration: Default

☐ Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

● Use workspace settings

Using GDB (DSF) Hardware Debugging Launcher

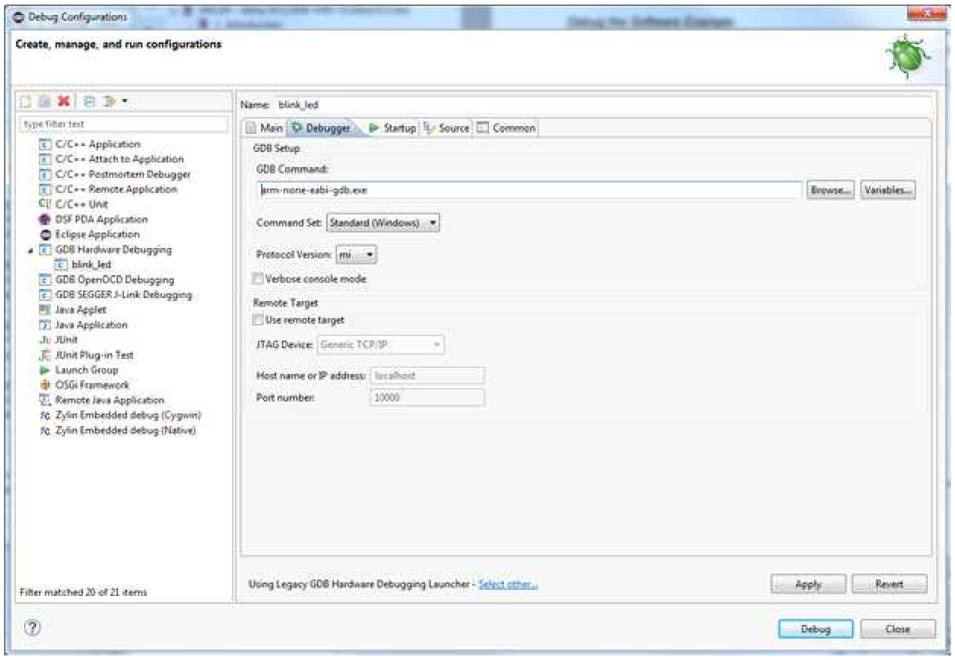
Figure 22. GDB Hardware Debugging – Main Tab

4. Select the 'Debugger' tab. Make sure the 'Use remote target' box is unchecked. In the GDB command field type in:
   a. Windows: arm-none-eabi-gdb.exe
   b. Linux: arm-none-eabi-gdb

NOTE: For Linux: In some cases an absolute path to gdb must be used for Eclipse to be able to find it. Use the full path to arm-none-eabi-gdb. This program is found in the bin folder where GNU Tools for Arm Embedded Processors is installed.

5. At the bottom of the window where the text 'Using GDB (DSF) Hardware Debugging Launcher' is shown, press 'Select other….Mark the checkbox 'Use configuration specific settings' and select the 'Legacy GDB Hardware Debugging Launcher'.

6. Select the 'Debugger' tab again. The settings should look like the ones shown in Figure 23.



Debug Configurations

Create, manage, and run configurations

Name: blink_led

Main | Debugger | Startup | Source | Common

GDB Setup

GDB Command: arm-none-eabi-gdb.exe

Command Set: Standard (Windows)

Protocol Version: mi

Verbose console mode: ☐

Remote Target

Use remote target: ☐

JTAG Device: Generic TCP/IP

Host name or IP address: localhost

Port number: 10000

Using Legacy GDB Hardware Debugging Launcher - Select other...

Filter matched 20 of 21 items

Figure 23. GDB Hardware Debugging – Debugger Tab

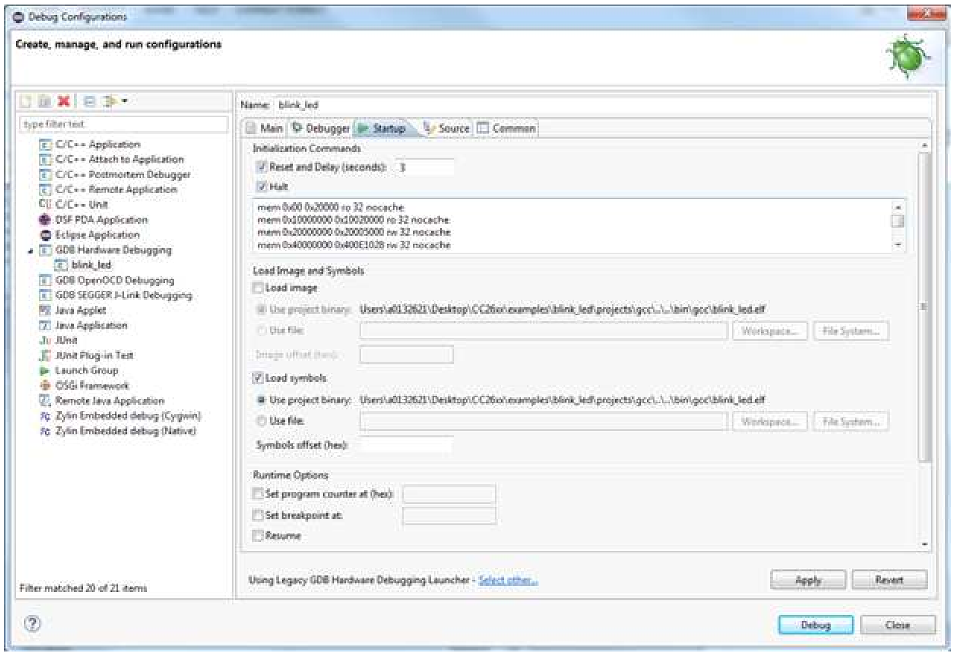
7. Select the ‘Startup’ tab and configure the same setting as shown in Figure 24. Make sure that Load image is unchecked, and that the following commands are added in the Initialization Commands window:

```
mem 0x00 0x20000 ro 32 nocache
mem 0x10000000 0x10020000 ro 32 nocache
mem 0x20000000 0x20005000 rw 32 nocache
mem 0x40000000 0x400E1028 rw 32 nocache
mem 0xE000E000 0xE000F000 rw 32 nocache
target remote localhost:55000
```

The mem command defines the specified memory regions with different attributes.

ro: read only
rw: read write
32: use 32 bit memory access
nocache: disable GDB from caching target memory

The target remote localhost:55000 will make the GDB to connect to the gdbserver on the local pc using port 55000.



Name: blink_led

Initialization Commands

[✓] Reset and Delay (seconds): 3

[✓] Halt

```
mem 0x00 0x20000 ro 32 nocache
mem 0x10000000 0x10020000 ro 32 nocache
mem 0x20000000 0x20005000 rw 32 nocache
mem 0x40000000 0x400E1028 rw 32 nocache
```

Load Image and Symbols

[ ] Load image

Use project binary: Users\a0132621\Desktop\CC26xx\examples\blink_led\projects\gcc\..\..\bin\gcc\blink_led.elf

[✓] Load symbols

● Use project binary: Users\a0132621\Desktop\CC26xx\examples\blink_led\projects\gcc\..\..\bin\gcc\blink_led.elf

○ Use file:

Symbols offset (hex):

Runtime Options

[ ] Set program counter at (hex):

[ ] Set breakpoint at:

[ ] Resume

Using Legacy GDB Hardware Debugging Launcher - Select other...

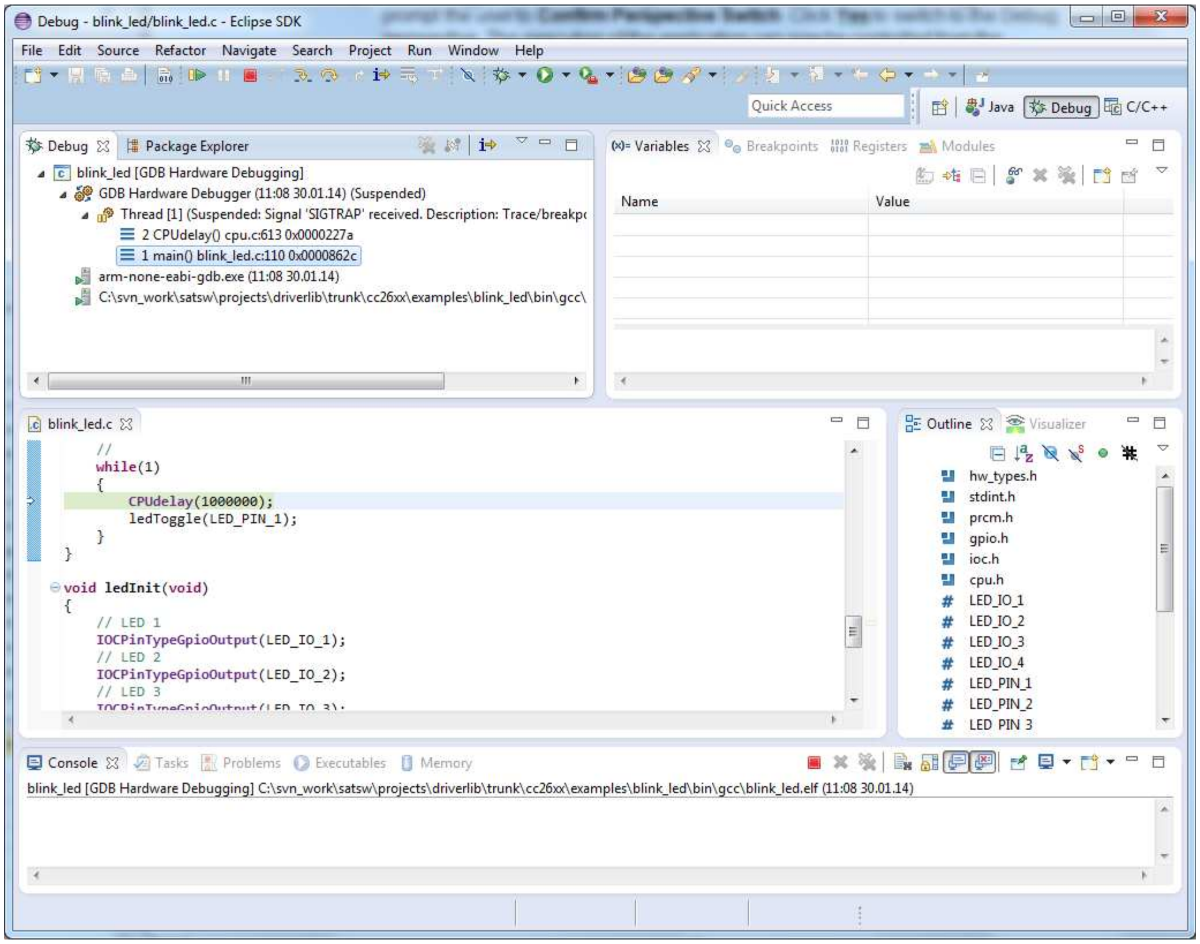
Figure 24. GDB Hardware Debugging – Startup Tab

8. Click 'Apply' and close the 'Debug Configurations' window.

7.4 Running the Software Example From Debugger

After the image is downloaded to the target, the debugger can be launched.

1. Select Run → Debug Configurations, and the Debug Configurations panel will be opened.
2. Make sure that the GDB server agent is started (as described in Section 7.1 for Windows and Section 7.2 for Linux)
3. Select the 'blink_led' configuration under 'GDB Hardware Debugging' and press the 'Debug' button.
4. The debugger connects to the target through the GDB server. The debugger prompts you to 'Confirm Perspective Switch'. Click 'Yes' to switch to the Debug perspective. The execution of the application can now be controlled from the debugger in Eclipse (press F8 (or the Resume button) to start running the application).



```
blink_led [GDB Hardware Debugging]
  GDB Hardware Debugger (11:08 30.01.14) (Suspended)
    Thread [1] (Suspended: Signal 'SIGTRAP' received. Description: Trace/breakpc
      2 CPUdelay() cpu.c:613 0x0000227a
      1 main() blink_led.c:110 0x0000862c
  arm-none-eabi-gdb.exe (11:08 30.01.14)
  C:\svn_work\satsw\projects\driverlib\trunk\cc26xx\examples\blink_led\bin\gcc\
```

```
    //
    while(1)
    {
        CPUdelay(1000000);
        ledToggle(LED_PIN_1);
    }
}

void ledInit(void)
{
    // LED 1
    IOCPinTypeGpioOutput(LED_IO_1);
    // LED 2
    IOCPinTypeGpioOutput(LED_IO_2);
    // LED 3
    IOCPinTypeGpioOutput(LED_IO_3);
```

```
hw_types.h
stdint.h
prcm.h
gpio.h
ioc.h
cpu.h
LED_IO_1
LED_IO_2
LED_IO_3
LED_IO_4
LED_PIN_1
LED_PIN_2
LED_PIN_3
```

```
blink_led [GDB Hardware Debugging] C:\svn_work\satsw\projects\driverlib\trunk\cc26xx\examples\blink_led\bin\gcc\blink_led.elf (11:08 30.01.14)
```

Figure 25. Running the Software Example From Debugger

8 References

1. *CC26xx Technical Reference Manual* (SWCU168)
2. *SmartRF06 Evaluation Board User's Guide* (SWRU321)
3. GNU Tools for ARM Embedded Processors home page: (https://launchpad.net/gcc-arm-embedded)

Makefile

The two files called *makedefs* and *makefile* are used when building the software example in this application report. The files are located in the folder examples/blink_led/projects/gcc/ in the software example package. This appendix gives a description of these files.

A.1 Makedefs

The file called 'makedefs' defines project independent variables, which are explained in this section. The file is included in the makefile.

CC	This variable should defines the compiler command for your GCC compiler
OBJCOPY	This variable should define the compiler command for copying a binary file.
CHIP_ID	This variable should define the device that is used.

If using Linux, the complete path to the above commands is required. 'makedefs' includes a section that can be commented out in order to solve this. You must then update the COMPILERPATH variable to your GCC compiler installation path.
The next variables are defined differently whether the makefile is run on Windows or Linux.

RMDIR	Command for deleting a directory.
RM	Command for deleting a file.
SLASH	This variable is need where windows require backslashes and Linux require forward slashes.

A.2 Makefile

This section describes the elements included in the makefile.

Table 2. Variables

vpath	Defines the list of directories that the make command should search. All directories containing source files for the project should be added to this variable. All directories are relative to the directory containing the makefile.
PROJECT	Contains the name of the project to build. The variable will be used to name the output files of the build
OUT_DIR	Defines the directory of where to place the generated output files.
OBJ_DIR	Defines the directory of where to place the generated object files.
SOURCE_FILES	This variable contains all the source files in the project. All CC26xx/CC13xx projects will require the startup files startup_gcc.c and ccfg.c. These files are again depended on several driverlib files. In this example, all driverlib files are included, by searching for every .c file in the driverlib source folder.
LINKERFILE	Defines the linker file for the project. The path is relative to the directory containing the makefile.
INCLUDES	Contains the include directories used by the compiler. The variable must include the source and inc folder in the driverlib package.
OBJGENOPTIONS	This variable contains options for the compiler when building object files. -Dgcc =1: Predefine gcc=1 -O0: Set optimization for compilation time -mcpu=cortex-m3: specifies the ARM processor to be cortex-M3 -gdwarf-2: Produce debugging information in DWARF version 2 format. -mthumb : Generate code for the Thumb instruction set -fomit-frame-pointer: Don't keep the frame pointer in a register for functions that don't need one -Wall: Enable all compiler's warning messages -Wstrict-prototypes: Warn if a function is declared or defined without specifying the argument types. -D$(CHIP_ID)=1: Predefine symbol for chip ID.
OUTGENOPTIONS	This variable contains options for the compiler when building output files -mcpu=cortex-m3: specifies the ARM processor to be cortex-M3 -nostartfiles: Do not use the standard system startup files when linking -T $(LINKERFILE): specifies the linker script. -Wl,: pass the subsequent options to the linker. -Map=$(PROJECT).map: print a link map to the file $(PROJECT).map --cref: Output a cross reference table --no-warn-mismatch: allow linking together input files that are mismatched.
OBJECTFILES	This variable contains the name of all object files that will be generated from the source files. The variable content is created by replacing the .c in the sourcefiles with .o.

Table 3. Rules

.PHONY : all clean	Tell make that the targets all and clean are not associated with any files.
clean :	Rule for cleaning up after the build. All output files and all object files are deleted.
all:	Rule for building the complete project.
$(OBJ_DIR):	This rule creates the directory to place object files if it doesn't exist.
$(OUT_DIR):	This rule creates the directory to place output files if it doesn't exist.
$(OBJECTFILES):	Rule to make sure the object directory is created before any object files.
$(PROJECT).elf $(PROJECT).bin:	Rule to make sure the output directory is created before any of the output files.
$(OBJ_DIR)/%.o:	Rule for building the object files. –o $@ places the output in a file with the same name as the target. -c means that the source file should be compiled, but not linked. $< sets the source file to be the first prerequisite.
$(PROJECT).elf:	Rule for building the .elf file. The option –o places the output in the directory passed as argument.
$(PROJECT).bin:	Rule for building the .bin file. $(OUT_DIR)/$< defines the input file. $< returns the first prerequisite. $(OUT_DIR)/$@ defines the output file. $@ returns the target --gap-fill 0xFF means that gaps between sections should be filled with 0xFF. -O binary creates an output file in binary format.

Linker and Startup Files

This appendix gives a description of the linker and startup files required when making software projects for CC26xx/CC13xx.

B.1 Linker File

The linker file cc26x0f128.lds is found in the folder cc26xxware/linker_files/ in the software example package.

The following variables are defined in the linker script.

Table 4. Variables Defined in the Linker Script

_estack	This variable is used in the startup file and defines the top of the stack to be at the end of SRAM.
_Min_Heap_Size	This variable defines required amount of heap. The linker file will report an error if there is not enough space for this amount of heap in the RAM.
_Min_Stack_Size	This variable defines required amount of stack. The linker file will report an error if there is not enough space for this amount of stack in the RAM.

The following commands are called in the linker script.

Table 5. Commands Defined in the Linker Script

ENTRY	Defines the entry point, i.e. the first instruction to execute in the program. The entry point is set to ResetISR, which is a function implemented in the startup file.
MEMORY	This command describes the location and size of blocks of memory in the target. ORIGIN defines the start address of the memory block. LENGTH defines the size of the memory block. Attributes can be placed in parenthesis after the memory block name. The attributes used in this linker file is 'R' Read-only section 'W' Write section 'X' Executable section

SECTIONS	This commend tells the linker how to map input sections into output sections and how to place the output sections in memory.
	This linker script contains the output sections .text, .data, .bss and .ccfg. In addition a section called _user_heap_stack is defined. This is to make sure that there is enough space in the SRAM region for the required amount of heap and stack.
	The below code defines the output section .text:
	`.text :`
	`{`
	`    _text = .;`
	`    KEEP(*(.vectors))`
	`    *(.text*)`
	`    *(.rodata*)`
	`    _etext = .;`
	`} > FLASH= 0`

Table 5. Commands Defined in the Linker Script (continued)

	_text = .; sets the symbol _text to the value of the location counter. The location counter has value 0 at the beginning of the SECTIONS command. Then follows the input sections that should be placed in this output section. KEEP(*(.vectors)) will keep the symbols *(.vectors) in the section even if symbols are not referenced. The '*' is a wildcard symbol. *(.text*)and *(.rodata*) means that all .text* and .rodata* input sections in all input files should be placed in this section. _etext = .; sets the symbol _etext to the value of the location counter. > FLASH= 0 assigns the section to the memory region FLASH, and fills unspecified regions of the memory with 0. In the definition of the .data section, the keyword AT(lma) is used. This keyword specifies the load address (lma) of the section.

B.2 Startup Files

The startup *files startup_gcc.c and ccfg.c* files are found in the 'cc26xxware/startup_files/' folder in the driverlib package.

ccfg.c specifies the Customer Configuration Area (CCA). The CCA is given the compiler attribute section in order to assign it to the .ccfg section of the linker script.

startup_gcc.cmd contains the implementation of the reset ISR, the default fault handlers and the vector table. The interrupt handler prototypes are declared with the attribute weak. This attribute lets functions get overridden if declared anywhere else, such as in the main program. The vector table is given the compiler attribute section in order to assign it to the .vectors section of the linker script. The first value of the vector table is a pointer to the top of the stack. This variable is also gotten from the linker script.

Make sure that the referred sections used in the startup files match the names used in the linker file.