

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

1	Introduction	2
2	Prerequisites.....	3
3	Hardware Setup.....	4
4	Software Installation Instructions	4
5	Build the Software Example	13
6	Load Binary Image to Target	15
7	Debug the Software Example	21
8	References	27
Appendix A	Makefile.....	28
Appendix B	Linker and Startup Files	31

List of Figures

1	Eclipse - Install New Software (Windows).....	5
2	Installing CDT	6
3	Installation Details	7
4	Eclipse - Install New Software (Linux)	8
5	Installing CDT (Linux)	8
6	GNU Toolchain (Windows)	9
7	Installing MinGW (Windows).....	10
8	Installing MinGW (Windows).....	10
9	UniFlash Installation	11
10	UniFlash Installation	12
11	Import Example Project Into IDE	13
12	Build Project	14
13	Clean Project	14
14	Connected Devices	15
15	Flash Programming Configuration	16
16	Flash Programming Configuration	17
17	Configuring the Flash Programmer Tool (Linux)	18
18	Load the Image to Target (Windows).....	19
19	Load the Image to Target (Linux)	20
20	Launching the GDB Server (Windows).....	21
21	Launching the GDB Server (Linux)	22

22	GDB Hardware Debugging – Main Tab	23
23	GDB Hardware Debugging – Debugger Tab	24
24	GDB Hardware Debugging – Startup Tab	25
25	Running the Software Example From Debugger.....	26

Trademarks

SimpleLink, Code Composer Studio, SmartRF are trademarks of Texas Instruments.

ARM, Cortex are registered trademarks of ARM Limited.

Linux Mint is a trademark of Linux Mark Institute.

Linux is a registered trademark of Linux Torvalds in the U.S. and other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries, or both.

All other trademarks are the property of their respective owners.

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](#)).

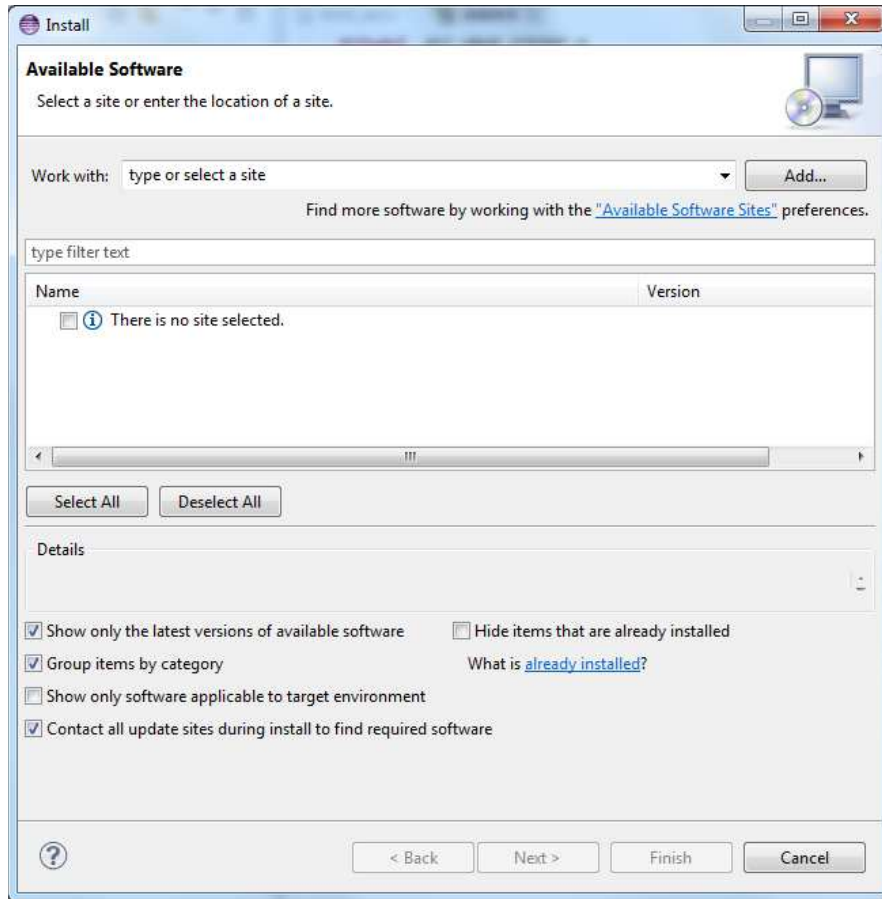


Figure 1. Eclipse - Install New Software (Windows)

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](#)).

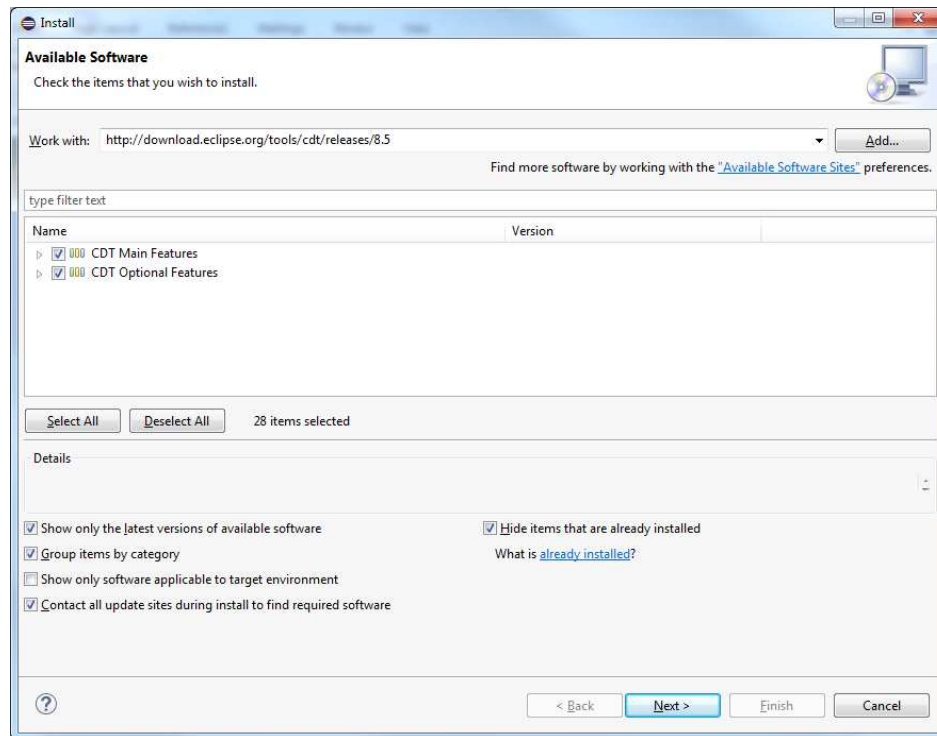


Figure 2. Installing CDT

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

- 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](#)).

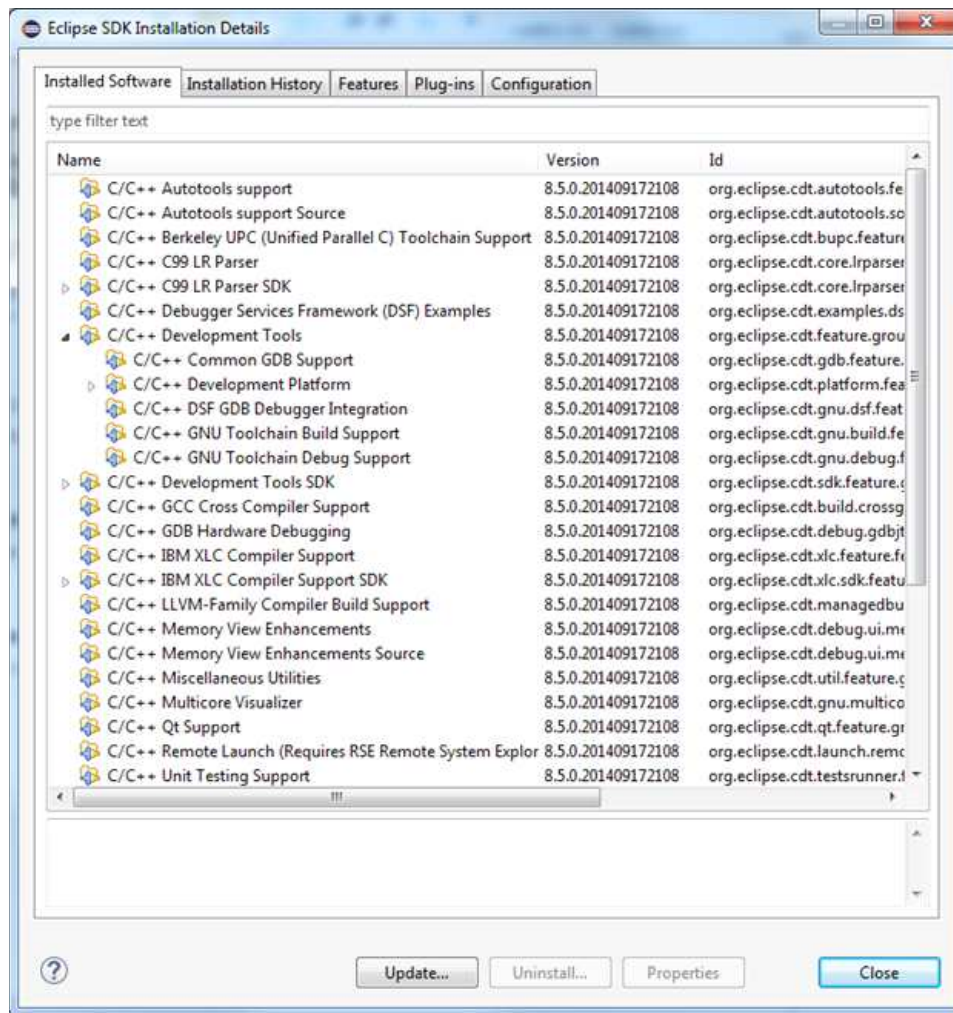


Figure 3. Installation Details

4.3 Eclipse IDE (Linux)

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

- Download Eclipse from <http://eclipse.org/downloads/index.php>. Choose the installer for your Linux platform.
- Extract the download package.
- 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
```

- Verify the installed versions of the package under Help → About Eclipse SDK.
- 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](#)).

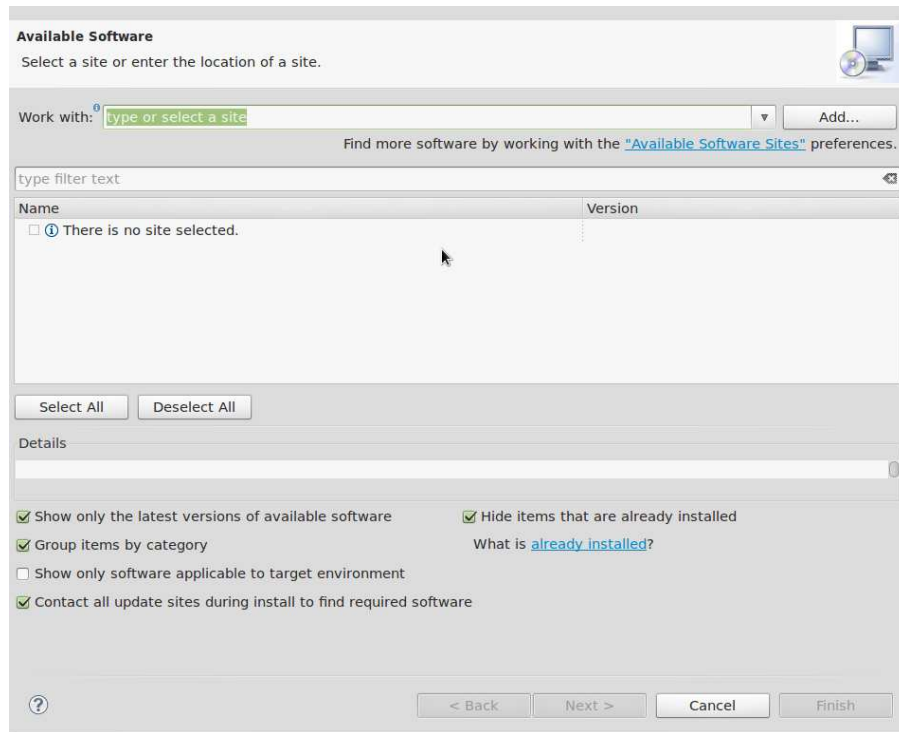


Figure 4. Eclipse - Install New Software (Linux)

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](#)).

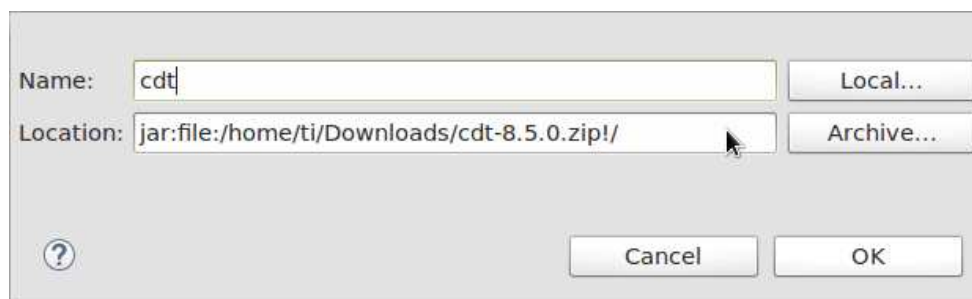


Figure 5. Installing CDT (Linux)

9. Follow step 3 in [Section 4.2](#). These final steps for Windows and Linux are equal.

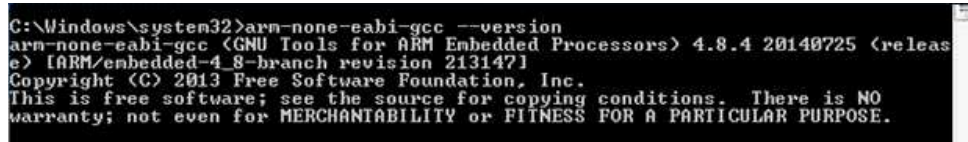
4.4 GNU Toolchain (Windows)

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 (release)
 [ARM/embedded-4.8-branch revision 2131471]
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:

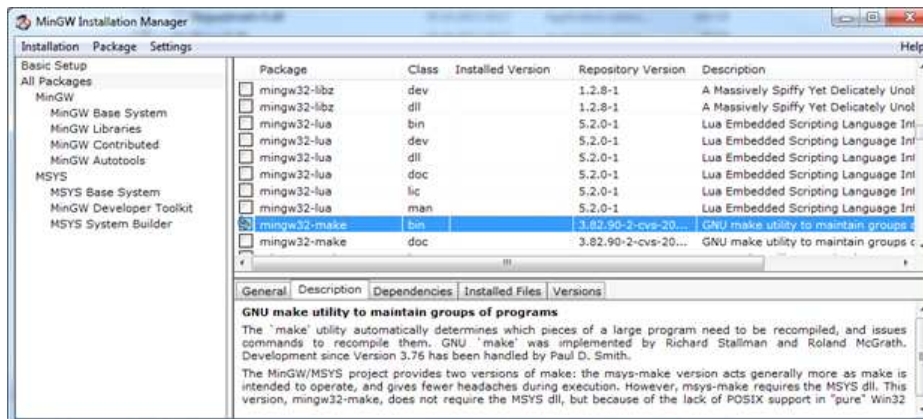
```
$ arm-none-eabi-gcc -v
```

More information about this package can be found in [\[3\]](#).

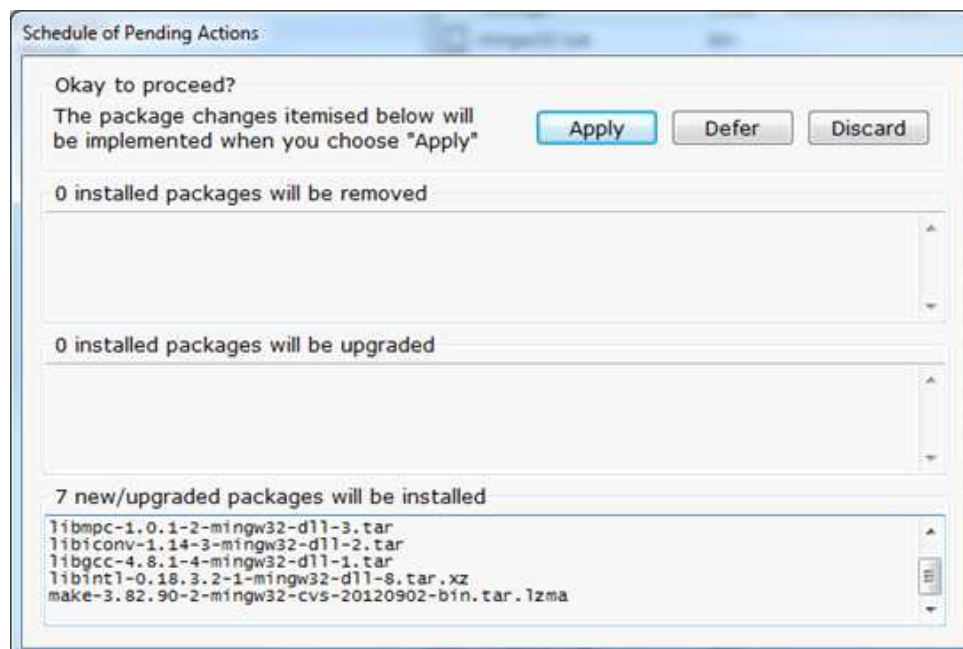
4.6 Build Tools for Windows

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'.


Figure 7. Installing MinGW (Windows)

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


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](#)).

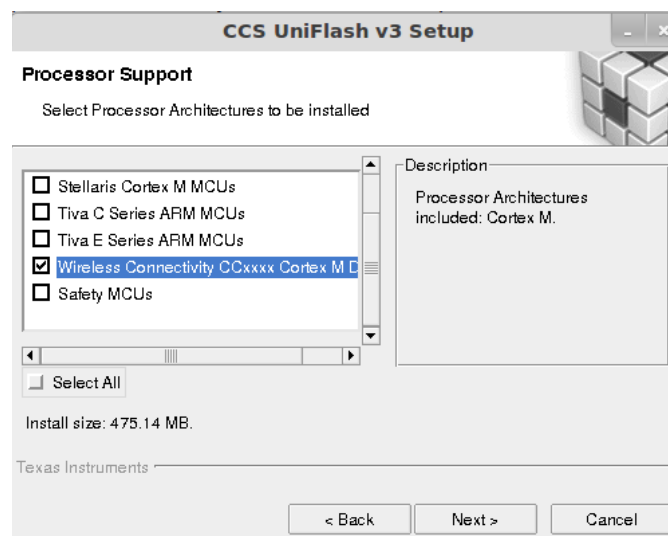


Figure 9. UniFlash Installation

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

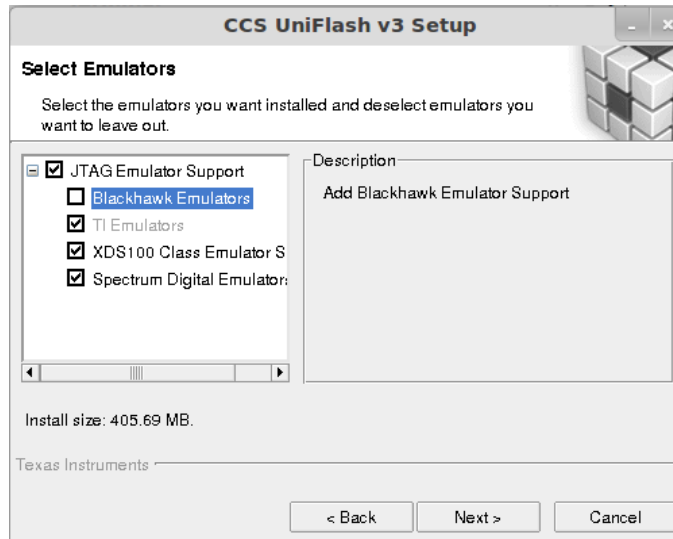


Figure 10. UniFlash Installation

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.

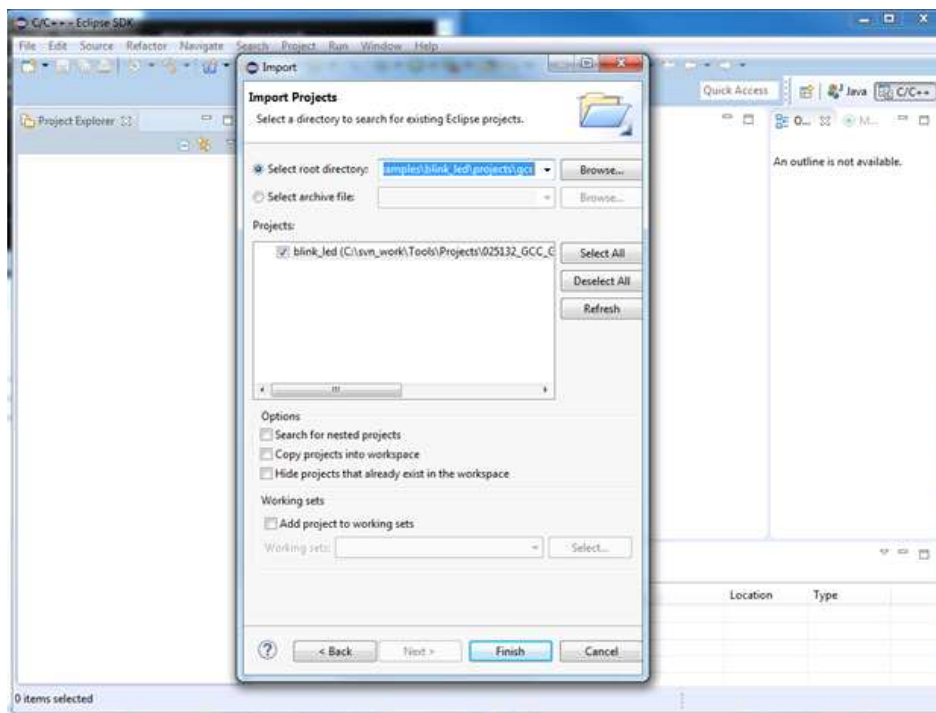


Figure 11. Import Example Project Into IDE

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

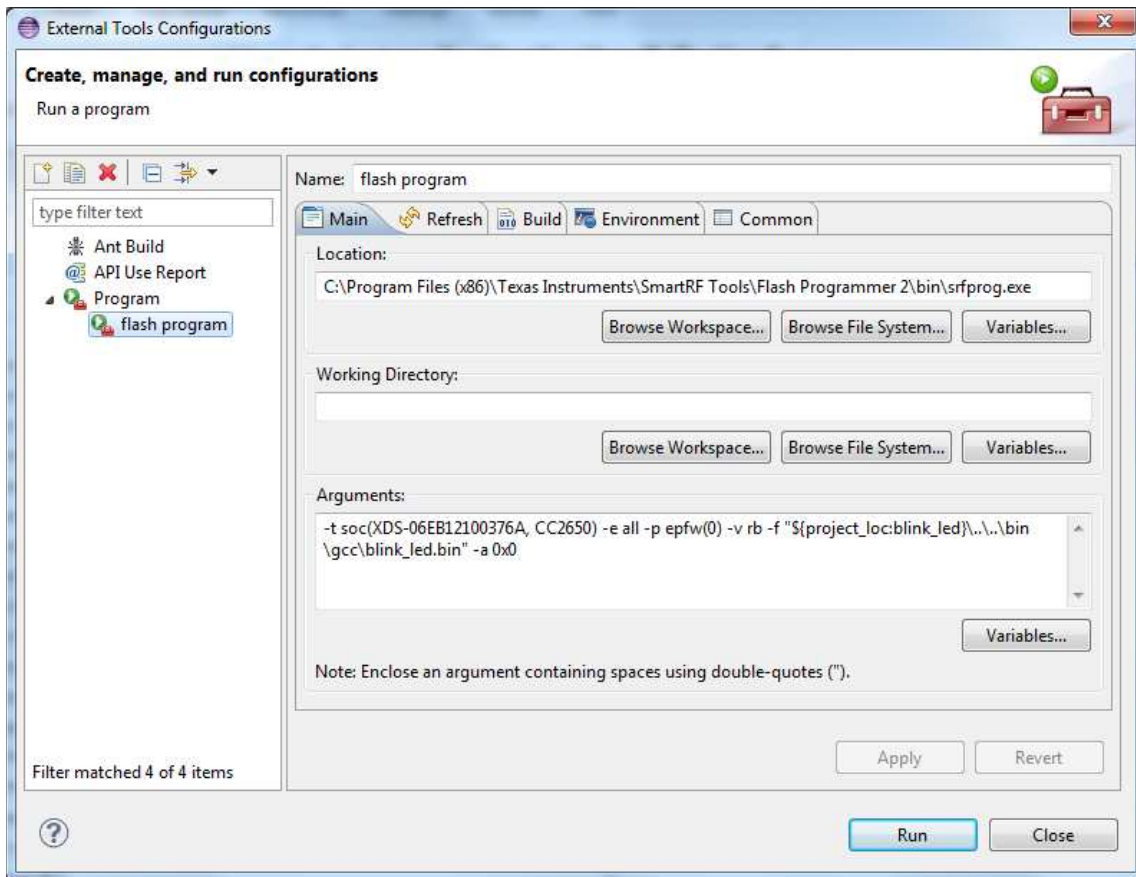


Figure 15. Flash Programming Configuration

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](#)).

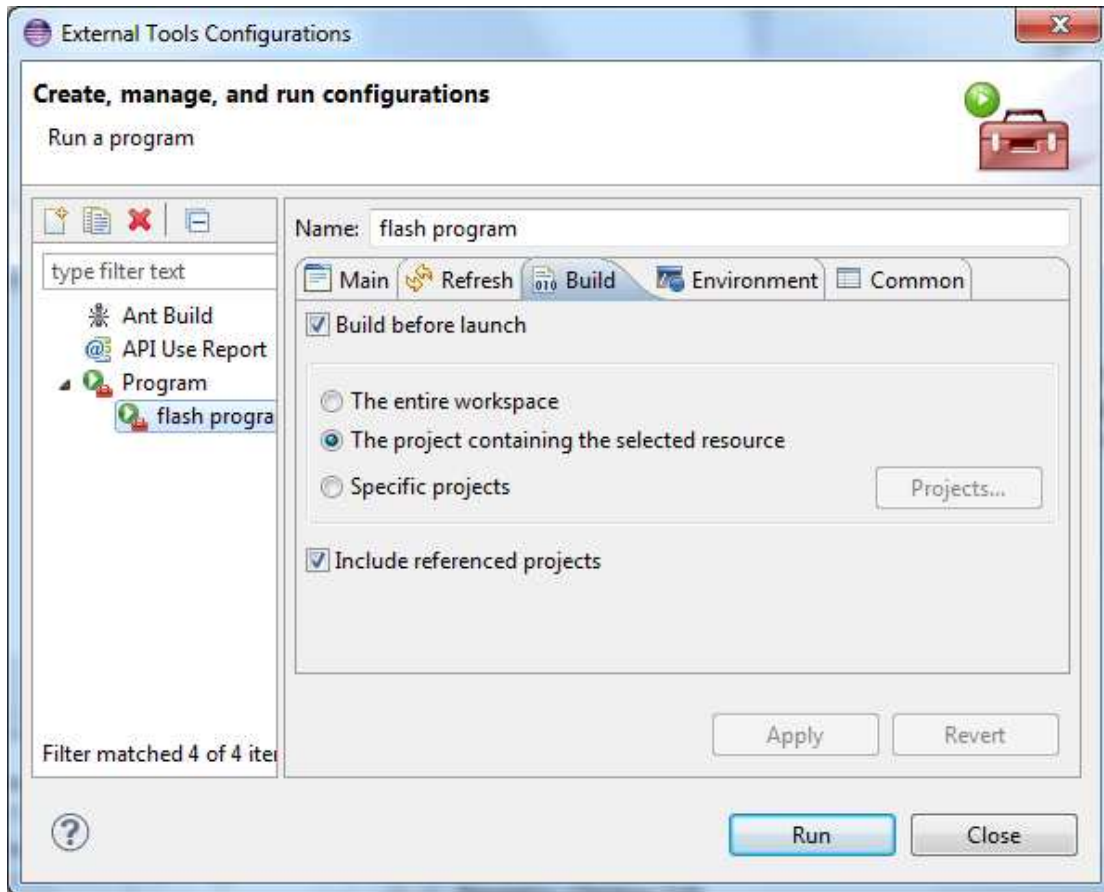


Figure 16. Flash Programming Configuration

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.

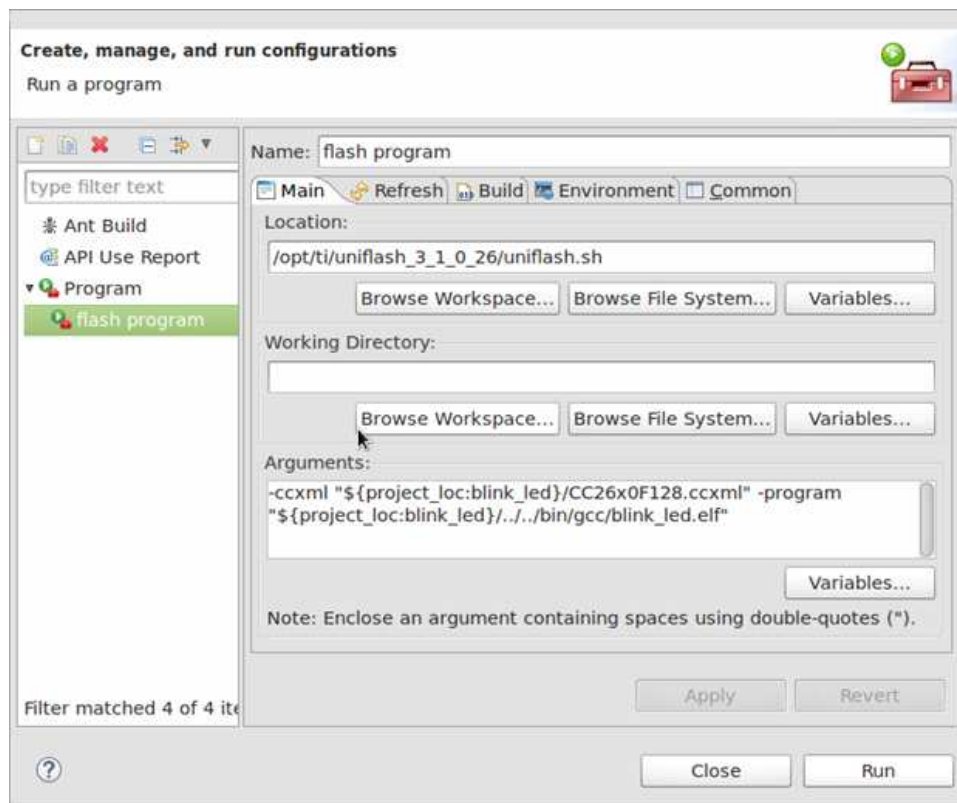


Figure 17. Configuring the Flash Programmer Tool (Linux)

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](#).

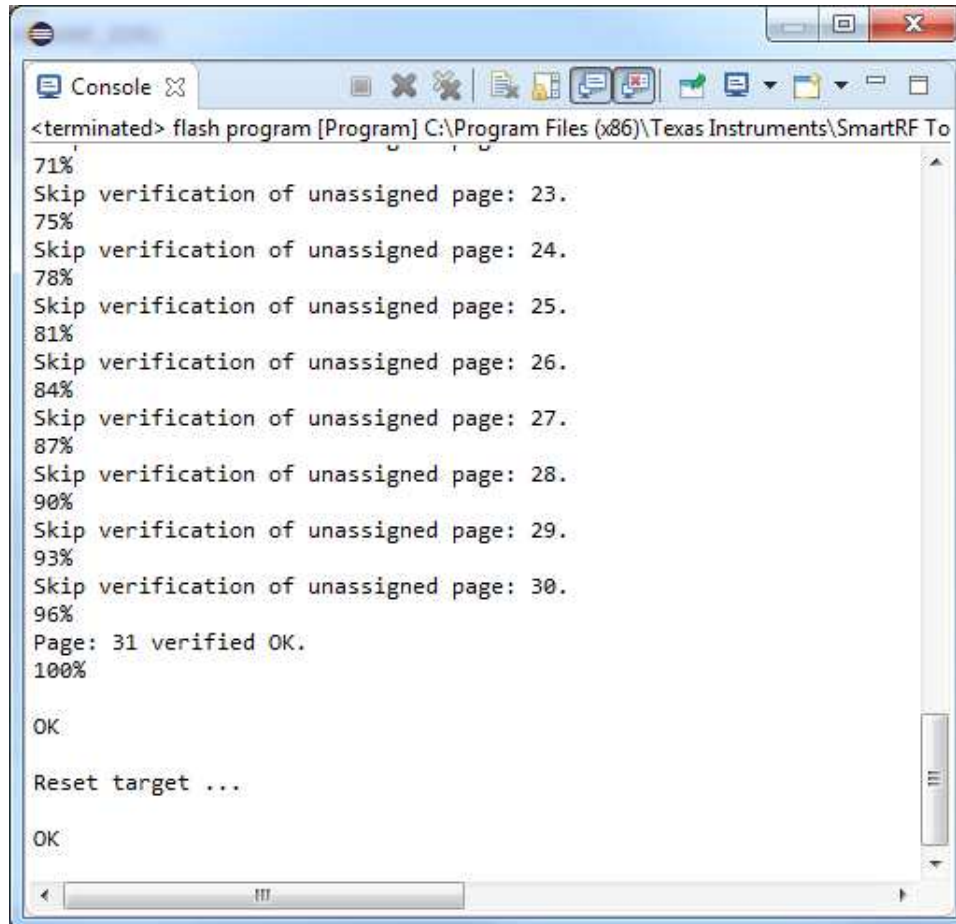
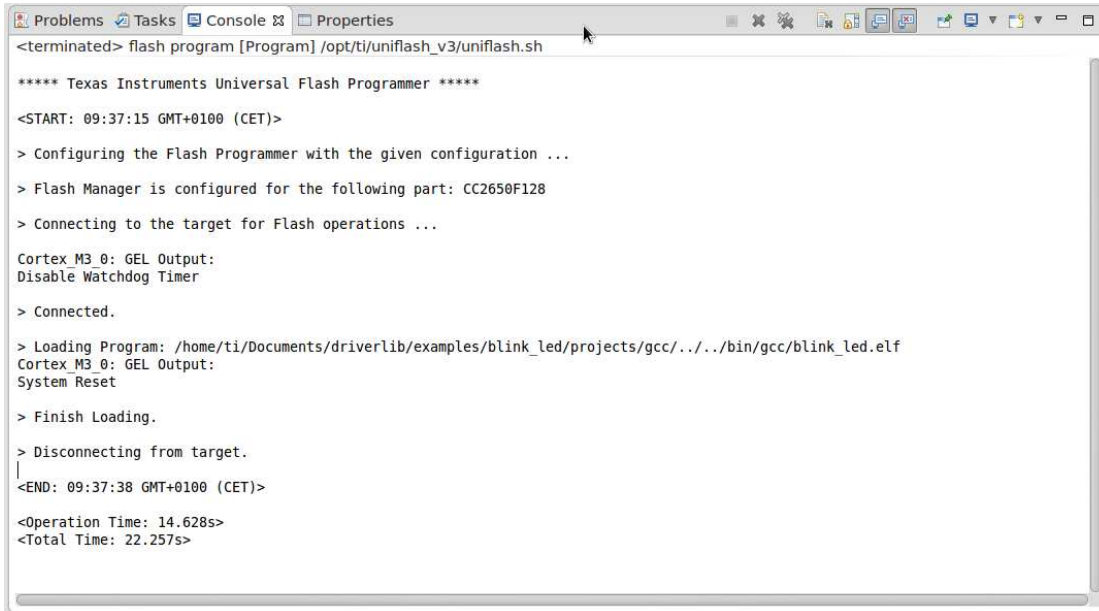


Figure 18. Load the Image to Target (Windows)

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.

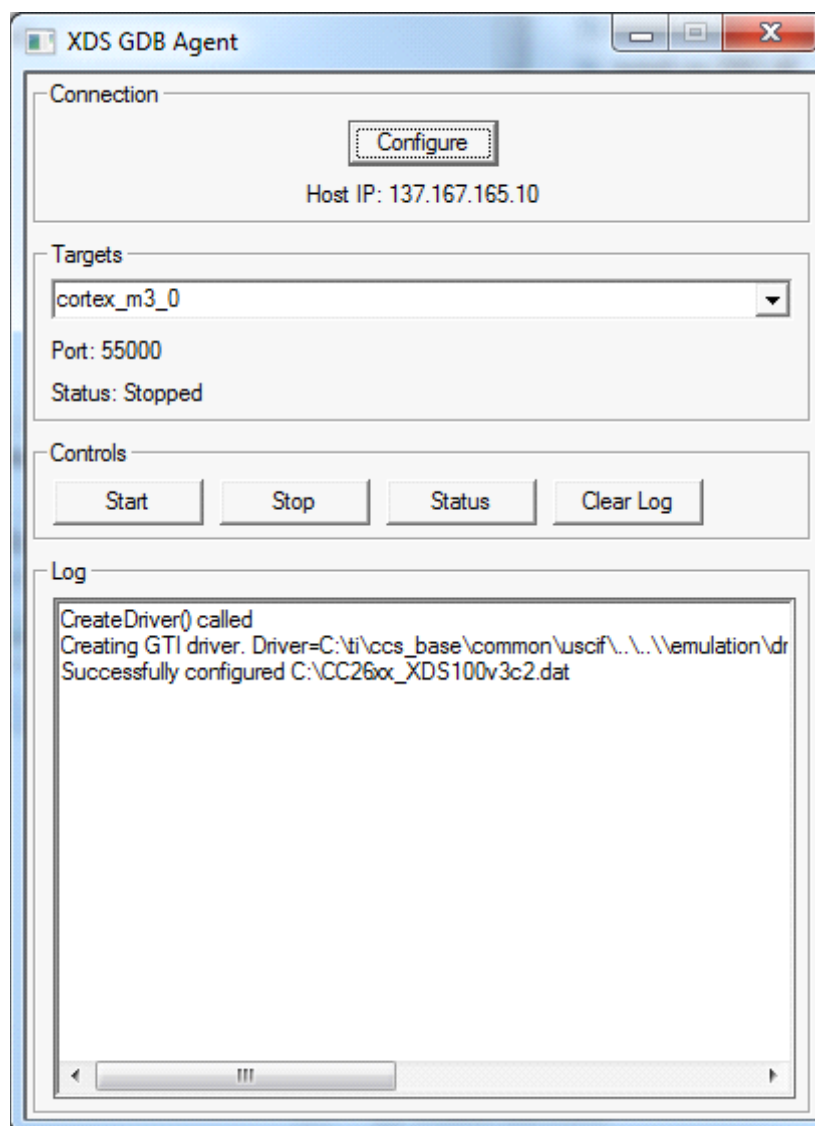


Figure 20. Launching the GDB Server (Windows)

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](#).

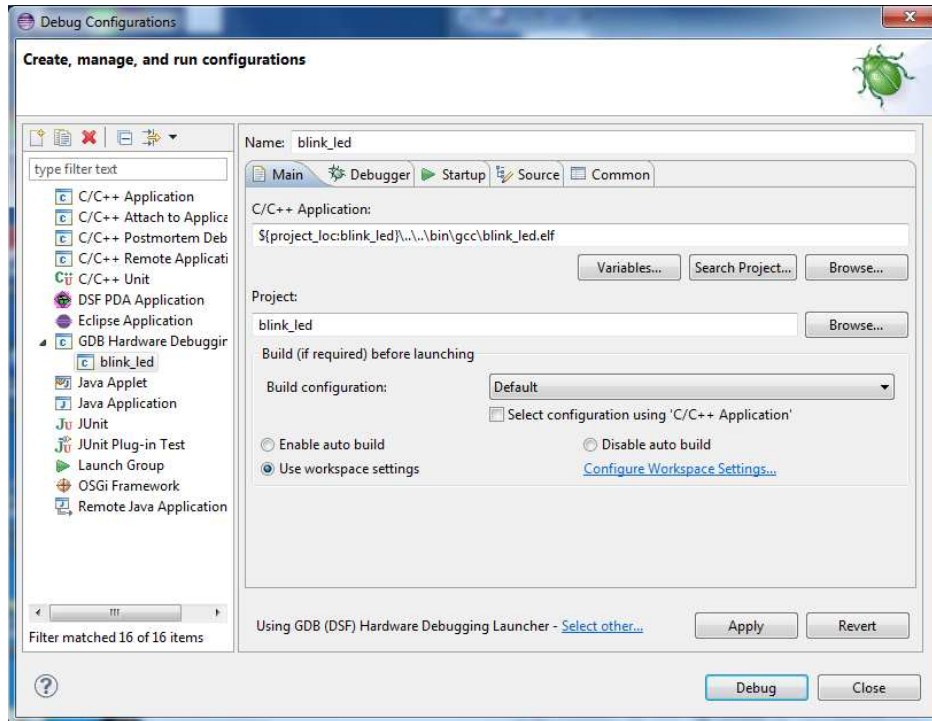


Figure 22. GDB Hardware Debugging – Main Tab

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](#).

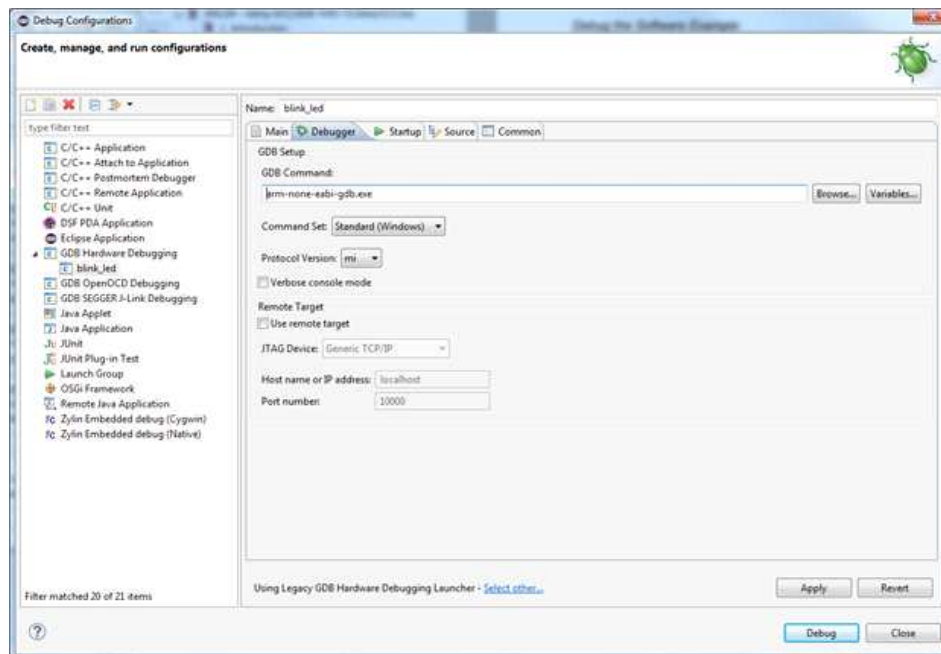


Figure 23. GDB Hardware Debugging – Debugger Tab

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.

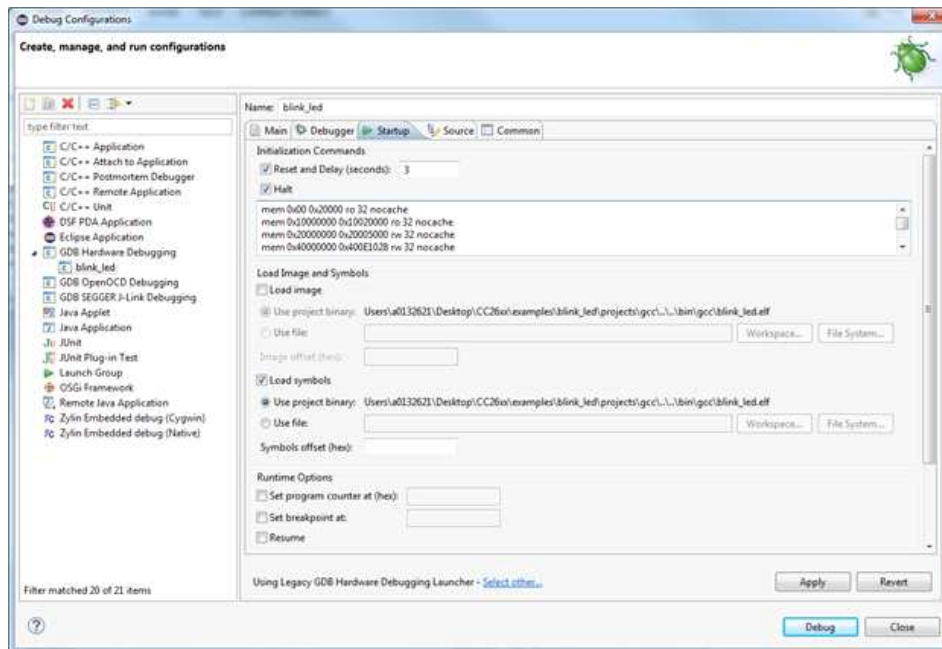


Figure 24. GDB Hardware Debugging – Startup Tab

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).

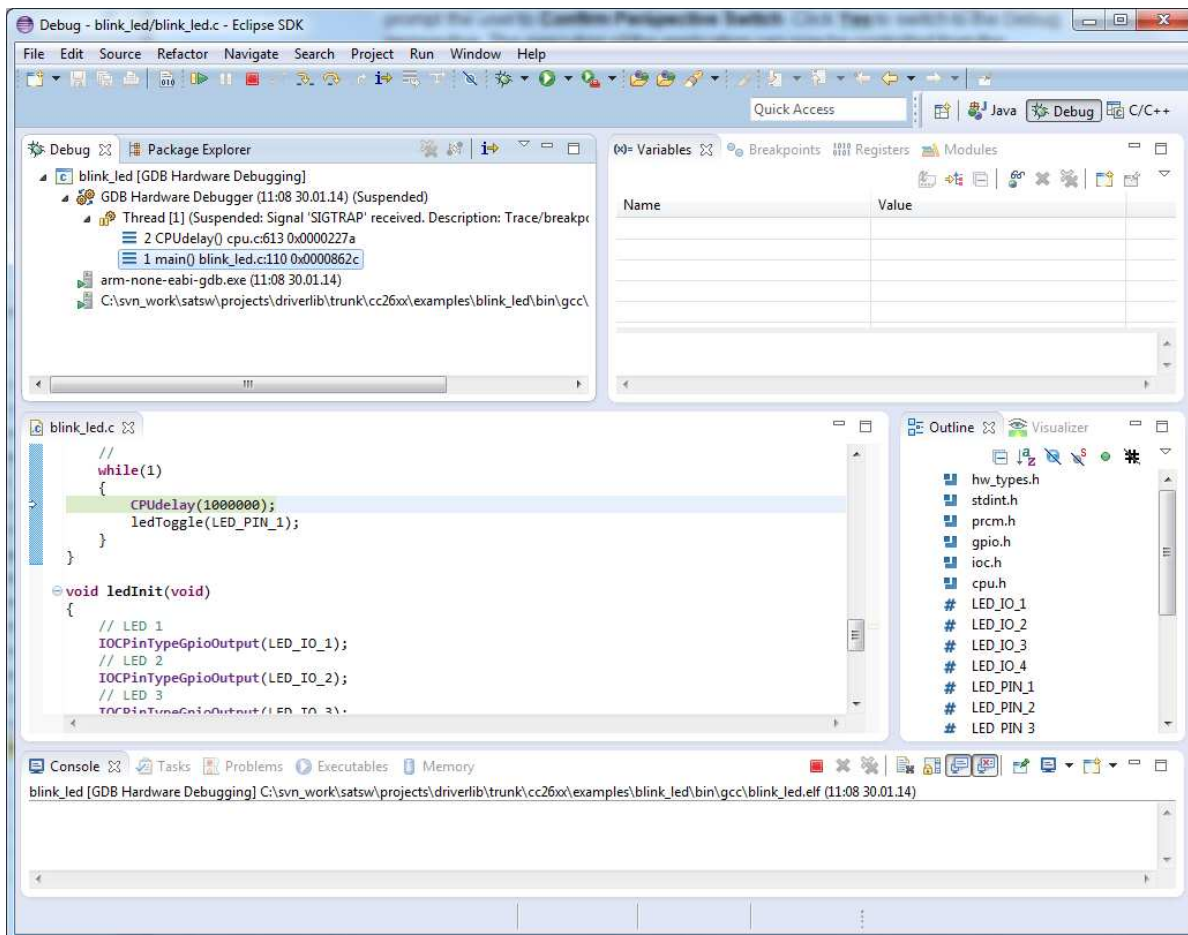


Figure 25. Running the Software Example From Debugger

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 cfg.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.lids 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 command 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: <pre>.text : { _text = .; KEEP(*(.vectors)) *(.text*) *(.rodata*) _etext = .; } > FLASH= 0</pre>

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

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

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 `.cfg` 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.

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ("TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products (<http://www.ti.com/sc/docs/stdterms.htm>), [evaluation modules](#), and [samples](http://www.ti.com/sc/docs/sampterm.htm) (<http://www.ti.com/sc/docs/sampterm.htm>).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2018, Texas Instruments Incorporated