Universal Boot Loader for SOC-8051 by USB-MSD Developer's Guide

Document Number: SWRU316

Texas Instruments, Inc.
San Diego, California USA

Version	Description	Date
0.1	Create document for review.	10/18/2011
1.0	Add document number, update title page	01/31/2012
2.0	Updated file names and added more detailed instructions	09/13/2013

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

This document is a Developer's Guide for the Universal Boot Loader (UBL) for the SOC-8051 using the USB transport and the MSD class.

1.1 Purpose

The purpose of this document is to enable a developer who is working with a sample application from an LPRF protocol stack that is targeted for an SOC-8051 with a USB peripheral to be able to effectively prepare for and use the Universal Boot Loader which uses the USB transport by MSD protocol.

1.2 Definitions, Abbreviations, Acronyms

Term	Definition
API	Application Programming Interface
BL	Boot-Loader – the boot loader code.
BLE	The protocol stack implemented by TI-LPRF for Bluetooth Low Energy.
DL	Down-Loaded – an RC candidate that has been downloaded (probably by OAD).
ISR	Interrupt Service Routine – code that the CPU physically jumps to upon receiving any enabled interrupt. Thus the execution of this code interrupts the normal execution of background code.
IVEC	Interrupt Vectors – the table of address range where the CPU physically vectors for ISR's.
LLD	Low Level Design
LPRF	Low Power RF – a business unit within TI.
MSD	Mass Storage Device – a USB class for NV-storage capable devices.
NV	Non-volatile storage
OSAL	Operating System Abstraction Layer
OA	Over-the-Air – an RF transmission that can be seen with a packet sniffer.
OAD	Over-the-Air Download – a public or proprietary process of transmitting a candidate RC image over-the-air to a device with a universal boot loader.
PC	Personal Computer
RC	Run-Code – the flash image of a customer's application that gets loaded and run.
RemoTI	The protocol stack implemented by TI-LPRF for RF4CE & its profiles.
RF	Radio Frequency
RF4CE	RF for Consumer Electronics – a network protocol specified by ZigBee®.
RNP	Remote Network Processor – implementation of RPC for control and status from the RemoTI stack.
RPC	Remote Procedure Call – the exportation of an API to another processor.
SOC	System-on-a-Chip (for example the CC2531 or CC2540).
TI	Texas Instruments Incorporated
UBL	Universal Boot Loader – a design that supports any transport or protocol stack.
USB	Universal Synchronous Bus – a serial protocol specified by the USB-IF, Inc.
ZStack	The protocol stack implemented by TI-LPRF for ZigBee & its profiles.

1.3 References

[1] Z-Stack Developer's Guide (SWRA176)

1.4 Known Issues

The USB is currently not working with Linux or Ubuntu. Although it enumerates as a sc'x' device, Nautilus does not browse it and 'ls /dev/sd*' hangs when it reaches the device.

2. UBL by USB-MSD Overview

The UBL will enumerate on the USB as a MSD device, appearing to the Windows USB Host as a generic flash drive. The UBL fakes a FAT-12 file system to the USB host; but it will only ever support two files. The first file is the configuration file and must conform to the 8.3 filename constraint with the file extension being exactly ".CFG". An example of a valid 8.3 filename for the configuration file is the default configuration which the UBL will use if another one is not used to override it: DEFAULT.CFG. The second and last file that is supported is the RC-image file which can have any extension (although .bin would be the most typical) and a long filename is supported, so this would be a valid filename: GenericApp-ZC_UBL-MSD.bin.

The contents of the configuration file must be exactly a meta-data structure as described within this document and defined in the ubl_app.h module. The contents of the configuration file cannot be read from the UBL, as even the default configuration could contain protected information like the security key. If no RC-image exists, the current configuration file can simply be overwritten by dragging-and-dropping the new configuration file (i.e. the old configuration file does not need to be, and in this scenario cannot be, deleted first). If an RC-image does exist, first delete the configuration file and then drag-and-drop the new configuration file.

The content of the configuration file is the meta-data that is used when performing operations such as mass-erase of the current RC-image and the downloading of the new RC-image. As described below, the meta-data contains a bit-mask that prevents the erasing of pages that should be preserved, such as an NV memory area, so it is imperative that the default configuration is correct or that the correct configuration be established before making any operations on the RC-image.

When an RC-image exists on the generic flash drive, it must first be deleted before copying the new image to boot load. Note that anytime the RC-image has a non-erased security key, then the DEFAULT.CFG must always be overwritten with a configuration file with a valid security key. Otherwise, when the RC-image is "deleted", the UBL will not enforce the "ERASE-Enable" bit mask but rather it will erase all flash pages. This is protection from the downloading of a Trojan horse program that could read-out any otherwise "read-protected" pages, such as the NV pages area.

2.1 FAT file system Theory of Operation

In order to simplify the code size and memory requirements of the UBL as much as possible, the file system only supports up to two files as described above. To further simplify the UBL, the configuration file is always forced into the first FAT sector and the image into the remaining sectors. Even if the image loaded is not the full 244 KB available, the UBL will fake the FAT such that the generic flash drive appears to be full with 2 KB being used by the configuration file and 244 KB by the RC-image.

Anytime that a file is deleted from or copied to the UBL generic flash drive, the drive will re-enumerate on the USB so that the "compacted" FAT can be displayed. This is necessary for enforcing the order of the configuration and RC-image as well as for reducing the RAM needed to fake the filenames.

3. UBL-Enabled Sample Applications

Any LPRF Sample Application can be enabled for use with the UBL by implementing the few simple changes which follow.

3.1 Typical Memory Map

The CC2531F256 and CC2540F256 contain 128 2048-byte flash pages. The last page, page 127, cannot be erased or written to programmatically (unless running via the debugger in debug mode) and thereby cannot be part of an RC-image that is UBL-enabled. This last page contains the lock bits and usually many other configuration data, such as a unique H/W or IEEE address of the device. The UBL occupies the first five flash pages.

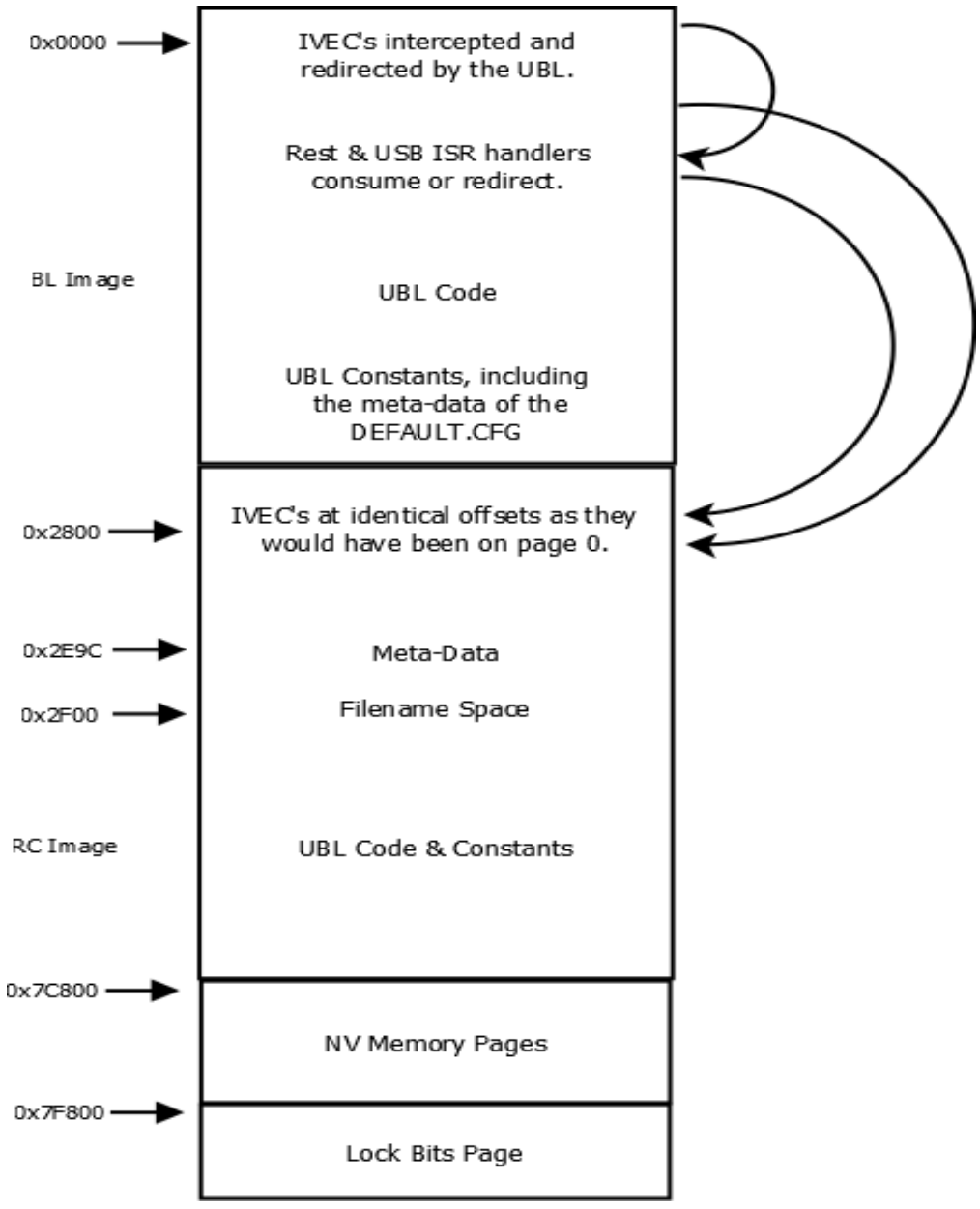

Figure 1: Typical Memory Map

3.2 Meta-Data

The meta-data has been padded so that fields are aligned on the flash word size of 4 bytes. Note that the CHK-MD is included, but not used.

Offset	Len	Name	Description
0	2	CRC-RC	CRC of the RC image, not including the meta-data or any pages that are not enabled for writing.
2	2	CRC-SHDW	Shadow of the CRC-RC which is written to non-0xFF only by the UBL.
4	2	CHECK-MD	Checksum of this meta-data, not including the CRC-RC or the CRC-SHDW.
6	2	DELAY-JUMP	Milliseconds to delay waiting for an indication (by GPIO, if enabled, and/or other indication received via the transport bus) to force boot loader mode.
8	32	SEC-KEY	256-bit security key that must be used to unlock the UBL in order to change the allow access bit arrays: Erase-Enable, Write-Enable, and Read-Lock.
40	16	ERASE-Enable	Page erase enable bit-mask of all flash pages.
56	16	WRITE-Enable	Page write enable bit-mask of all flash pages.
72	16	READ-Lock	Page read lock bit-mask of all flash pages.
88	1	CFG-DISCS	A byte array of configuration discrete bits.
89	1	GPIO-Port	A configuration byte to specify the Port and Pin of the GPIO control.
90	2	CFG-Spare	Spare configuration bytes for padding.
92	1	CNTDN-Forced	Count down the number of hard resets to force boot loader mode.
93	3	PAD1	Un-used bytes to pad to a 4-byte boundary for multiple writes to CNTDN.
96	1	CNTDN-SecKey	Count down attempted cfg-file hacks with a bad SEC-KEY.
97	3	PAD2	Un-used bytes to pad to a 4-byte boundary for multiple writes to CNTDN.

Figure 2: SOC-8051 USB-MSD specific meta-data structure.

3.2.1 CRC-RC

The CRC-RC is the CRC or FCS calculated over the RC image, excluding the meta-data and any pages not enabled for writing, by applying the calculation specified by the corresponding LLD. This FCS is generated and placed in the RC-image by the IAR linker. A CRC-RC value of all 0xFF's indicates to the UBL that no valid code image is present. If the CRC-RC does not equal the CRC-SHDW, or if the CRC-RC equals 0x0000 or 0xFFFF, then the BL mode is forced.

3.2.2 CRC-SHDW

This is a simple shadow of the CRC-RC, written by the UBL after it receives the last page of the RC-image. The RC image creation should set this value to all 0xFF's. The USB MSD data packets are received as verified valid data, and the UBL reads back and compares each flash page, page-by-page during the file copy, so that after the last page reads back good, the entire image is known good.

3.2.3 CRC-MD

This would be the checksum of the meta-data, not including the CRC-RC or the CRC-SHDW or itself, but it is not used by this UBL.

3.2.4 DELAY-JUMP

The milliseconds to delay waiting for an indication (by GPIO, if enabled, and/or an indication received via the transport bus) to force boot loader mode when a valid RC image is present.

3.2.5 SEC-KEY

If the current RC-image has a security key that is not all 0xFF's, then a meta-data structure with a matching security key must be transmitted to the BL via the transport in order to unlock the BL for erasing and writing to any internal flash, or for reading all pages (not just those enabled by the read lock bits). This blocks any unintended access to a network that could be illicitly obtained by downloading a Trojan horse program to a device that has successfully joined. The Trojan Horse code would read the Network Security keys and counters from the NV pages, which would have been locked with the READ-LOCK bits. If this key is all 0xFF's, then an image with any security key will be allowed to be downloaded or instantiated.

3.2.6 ERASE-Enable

Page erase enable bits as 16 consecutive bytes. A set bit indicates that the UBL shall allow erasing and re-writing of the physical page that corresponds to it and include the corresponding page in any mass-erase. This is a way to preserve areas such as the NV pages during a boot load.

3.2.7 WRITE-Enable

Page write enable bits as 16 consecutive bytes. A set bit indicates that the UBL shall allow writing of the physical page that corresponds to it. This is a way to preserve areas such as the NV pages during a boot load. Bits corresponding to pages belonging to the BL-image are disregarded.

3.2.8 READ-LOCK

Page read-lock bits as 16 consecutive bytes. A set bit indicates that the UBL shall not allow reading of any data from the physical page that corresponds to it. This is a way to prevent the reading of the Network Security keys and counters from the NV pages.

3.2.9 CFG-DISCS

This is a byte array of configuration bytes and discrete bits.

Byte	Bit	Name	Description
0	0	~GPIO-Use	Cleared indicates that the UBL shall decode the BL-GPIO pin, polarity, and pull and use it to determine whether the boot loader mode is being forced or not.
0	1	GPIO-Polarity	Set indicates that the BL-GPIO pin indication to force boot loader mode is active high; otherwise it is active low.
0	2	GPIO-Pull/Tri-state	Set indicates to pull-up/down the BL-GPIO pin; otherwise tri-state it.
0	3	GPIO-Pull-Up/Dn	Set indicates to pull-up (otherwise pull-down) the BL-GPIO pin if the BL-GPIO-Pull/Tri-state pin is set.
0	4	~CRC-Calc	Not applicable to this UBL – leave as set.
0	5	~DL-Use	Not applicable to this UBL – leave as set.
0	6	Spare-0_6	Reserved for future use and shall be set.
0	7	Spare-0_7	Reserved for future use and shall be set.
1	0-7	GPIO-Port/Pin	GPIO Port & Pin to use if ~GPIO-Use is cleared. Port-A, Bit-1 would be 0X8 + 1 = 1; Port-B, Bit-3 would be encoded as 1X8 + 3 = 11, and so forth.
2-3	All	SPARE	Spare configuration bytes reserved for future use which shall be set to 0xFF.

Figure 3: Meta Data CFG-DISCS.

3.2.10 CNTDN-Forced

This is a byte of discrete bits to count down by clearing bit-by-bit. If non-zero, clear one set bit on every power-on reset (not on a watch-dog reset, for example). Do not clear the last bit - this will force boot loader mode on all subsequent power-on resets until a new image is boot loaded. On a closed system that is not easily re-programmable, any new, un-tested code build should be downloaded with one or more bits set in the CNTDN in order to prove that the code does not lock-up in a way such that the boot loader can never be entered to download the bug fix. So this would act as a fail-safe from permanently losing a device due to a bug that locks out the boot loader mode. If the new code was locking up such that the boot loader could not be entered, the user would simply cycle power to the device until the boot loader mode was forced. After proving a code build, another build will of course be downloaded with the only change being that the CNTDN field is set to zero. Note that the BL can only count down the number of times that a flash word can be re-written between page erases.

3.2.11 CNTDN-SecKey

This is a byte of discrete bits to count down attempts to load a configuration file with a bad security key. If non-zero, clear one set bit on every failed attempt. Do not clear the last bit – just erase all flash pages not including the UBL image. This will neutralize any attempt to gain unauthorized access to pages with a read lock by directly reading them or downloading a Trojan horse RC-image that will.

3.3 The UBL_APP Module

The ubl_app.c is included in the Application project when FEATURE_UBL_MSD is added to the preprocessor define list (see section 3.6.1). It is linked in through OnBoard.c. Note that the structure of the meta-data defined in ubl_app.h cannot be changed, as it is locked into the UBL image. The meta-data configurations provided are recommended for the target boards specified by the pre-compiler defines and are for the CC2531DK-Dongle and CC2540DK-Dongle or for the CC2531-NanoDongle and CC2540-NanoDongle. In order for the UBL to correctly jump to the application, the UBL project is built with knowledge of the checksum value crcRC by using the preprocessor define UBL_BUILD. The Application remains unaware that this value is included in the meta-data because inside the Application's linker configuration file the checksum is treated as a value at an adjacent address to address of the meta-data. Customer hardware should define another set of meta-data with a new pre-compiler definition, following the example provided. Note that the existing meta-data examples could also be changed to suit the goals of the final use case of the Sample Application being developed.

3.4 IAR Project Linker

The Sample Application Linker Options need to be changed as described below. Access this configuration by choosing Project→Options→Linker.

3.4.1 Config

In the “Config” tab of the Linker Options, check “Override default” and add linker file “cc254x_f256_ubl_msd.xcl”. Under “Search paths” include the relative path “$PROJ_DIR$\..\..\common\cc2540”. It may be advantageous to copy the example cc254x_f256_ubl_msd.xcl from its original location to the IAR project area of the Application being modified. In this way the linker file can be modified for the exact memory layout and number of NV pages being used, etc.

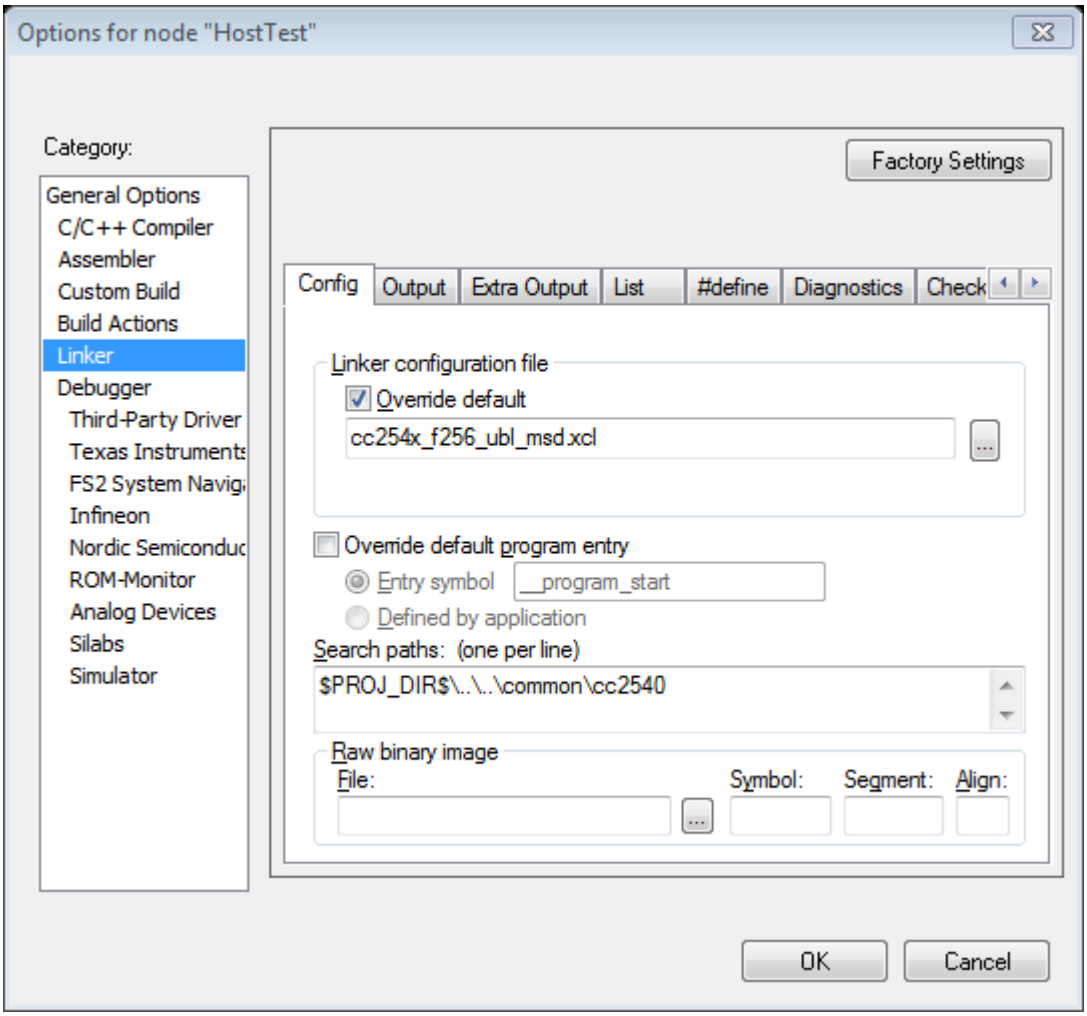

Figure 4: Linker Config.

3.4.2 Output

In the “Output” tab of the Linker Options, check the box to “Allow C-SPY-specific extra output file” as shown:

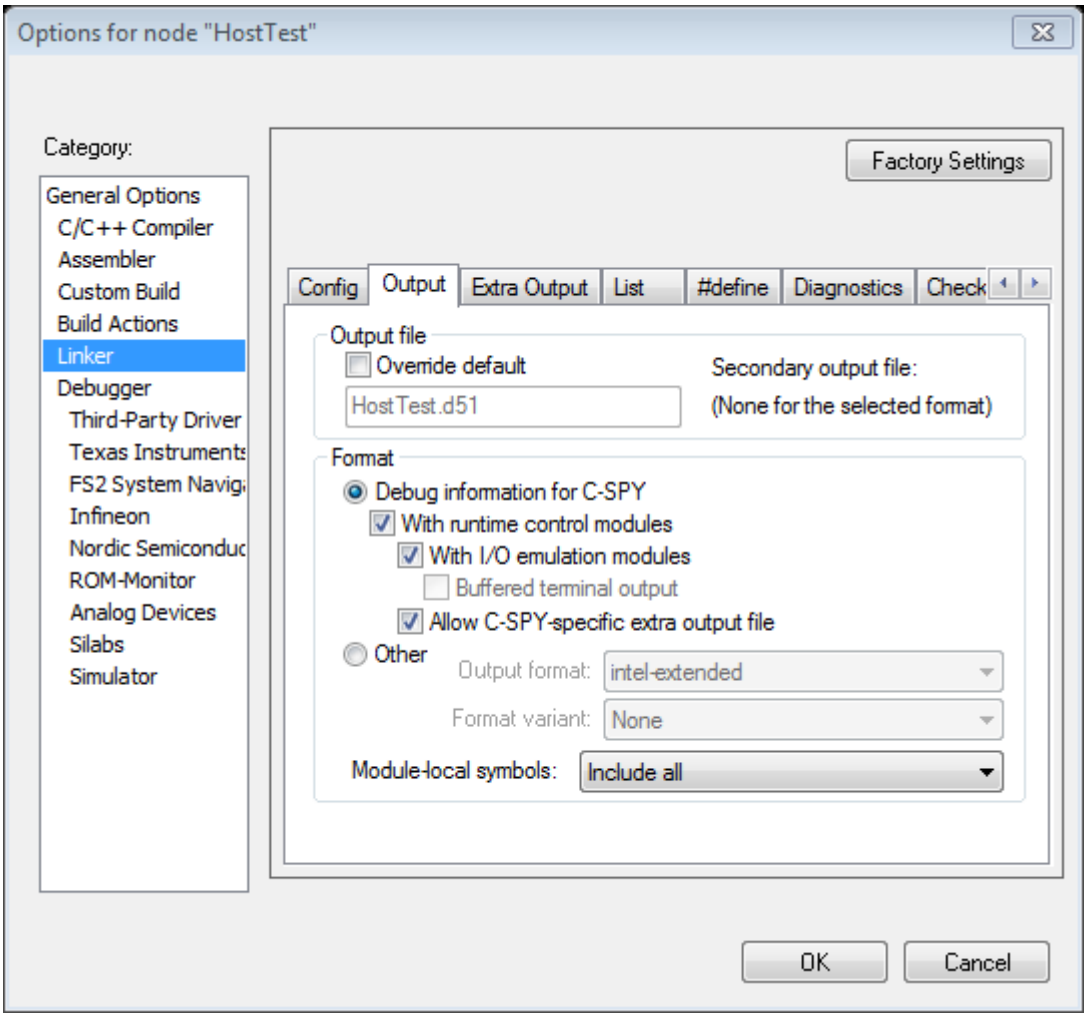

Figure 5: Linker Output.

3.4.3 Extra Output

In the “Extra Output” tab of the Linker Options, check the box to “Generate extra output file” and choose the “simple-code” output format. You may optionally check “Override default” if you wish to rename the the output file, as done here:

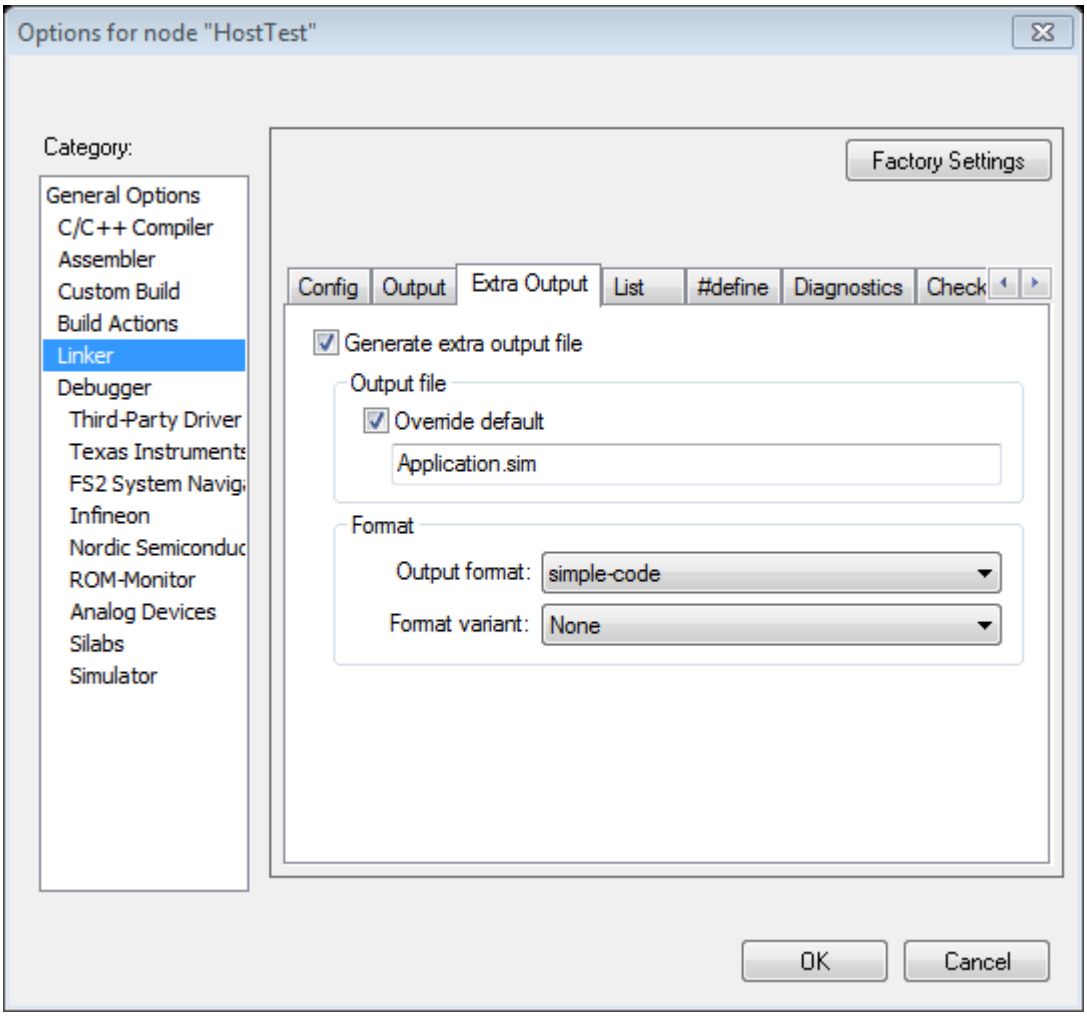

Figure 6: Linker Output.

3.5 IAR Project Build Actions

The simple-code extra output configured to be produced in the previous section is not the pure, monolithic binary image that is required for the UBL. So a post-processing build action needs to be added to strip out the extra information.

3.5.1 Post-build command line

On the “Post-build command line” of the Build Actions, paste the following 3 lines as one line with each part separated by a space. Of course the specific paths will be edited according to the stack and project with which you are working. This example is using the HostTest project running in the USB configuration.

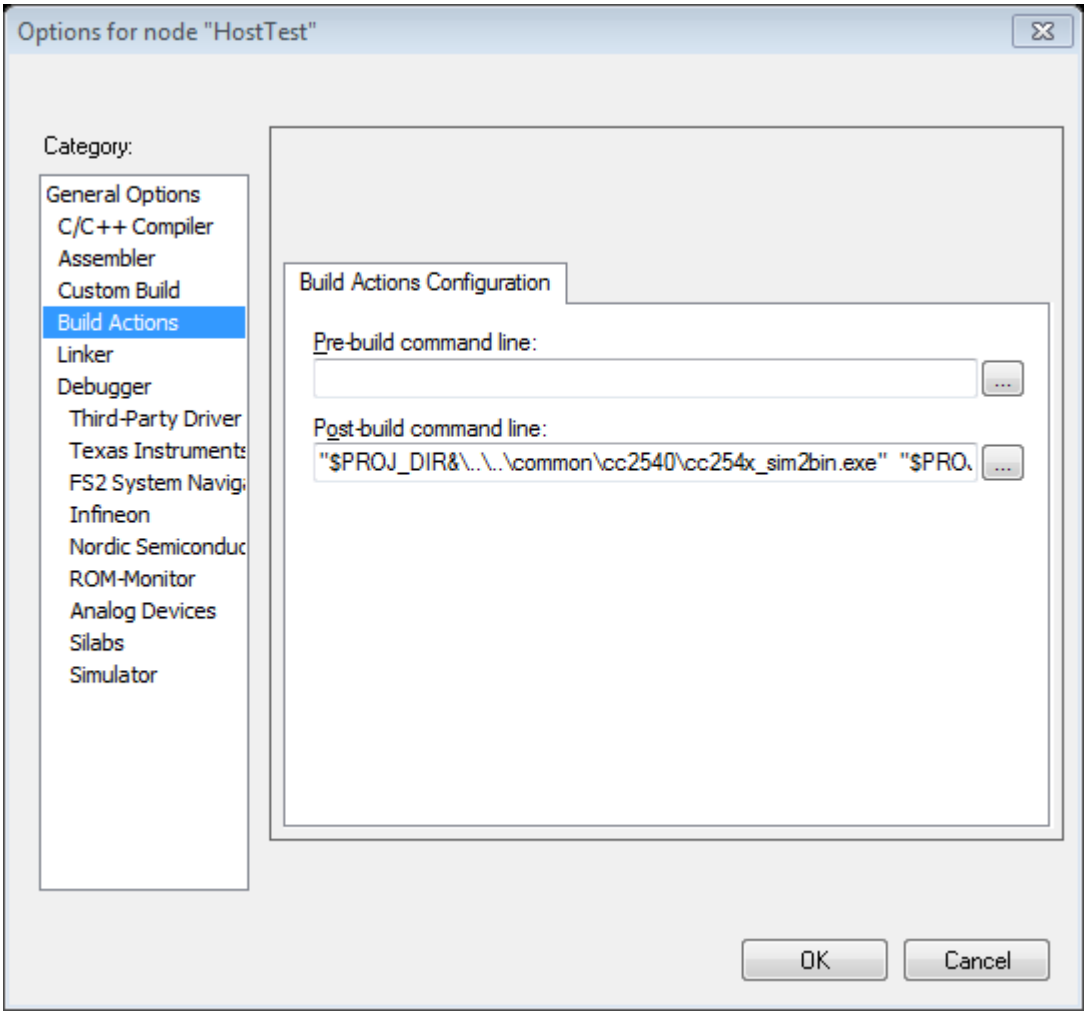

Figure 7: Post-build command line.

```
"$PROJ_DIR$\..\..\common\cc2540\cc254x_sim2bin.exe"
"$PROJ_DIR$\CC2540USB\Exe\Application.sim"
"$PROJ_DIR$\CC2540USB\Exe\Application_UBL-MSD.bin"
```

The first string is an executable which converts a “.sim” file – the second string – into a “.bin” file – the third string.

3.6 IAR Project C/C++ Compiler

Additional changes must be included for UBL functions to be called from the Application.

3.6.1 Preprocessor

In the “Preprocessor” tab of the Linker Options, add defines as necessary for the stack and project with which you are working. The Sample Application must have a call to appForceBoot() in order to force IAR to include the data structures in the ubl_app.c module. Such a call is crucial as well to be able to set the “DELAY-JUMP” to zero and have fast power-ups that start running the application immediately without delaying in BL mode.

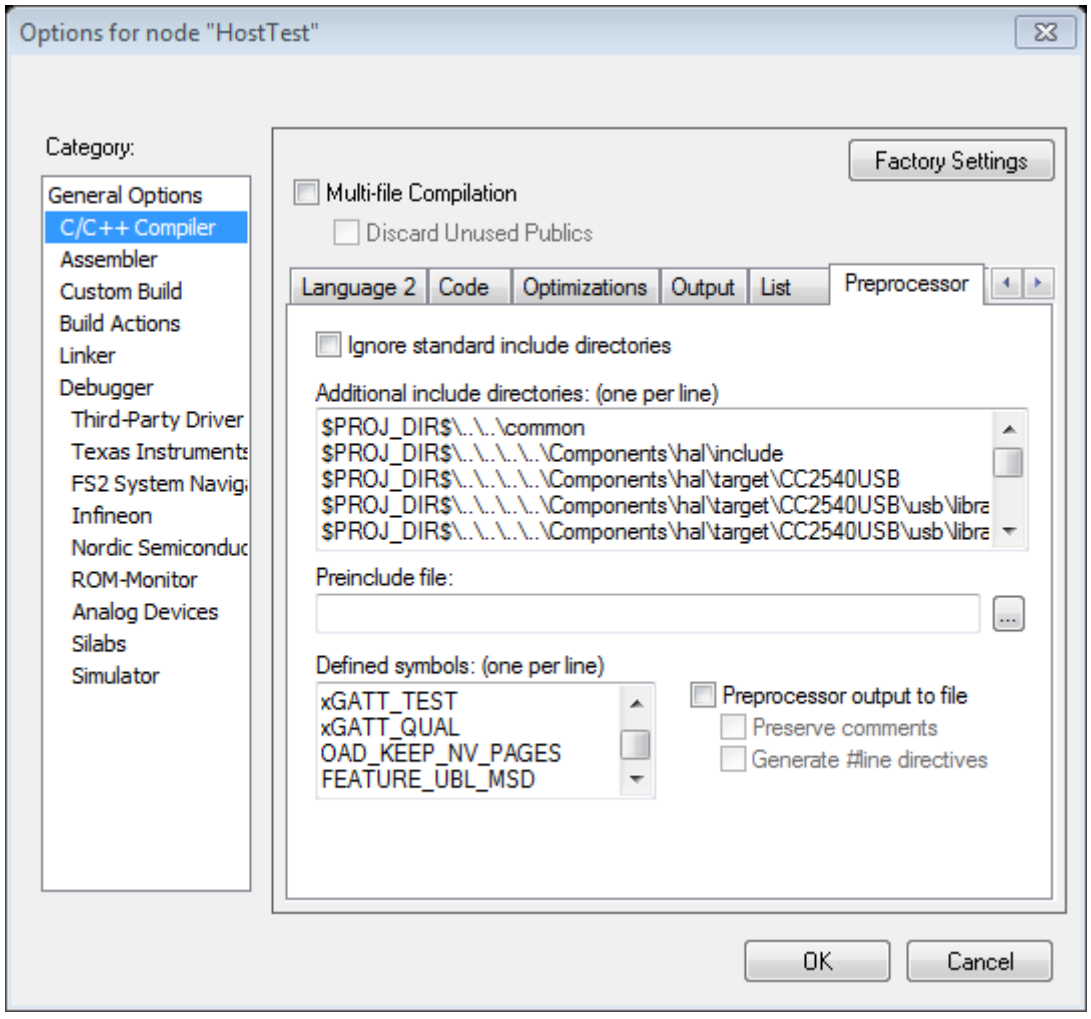

Figure 8: Defined symbols.

Add the bottom two “Defined symbols”: OAD_KEEP_NV_PAGES and FEATURE_UBL_MSD. The goal of the symbols added will be to enable inclusion of the UBL meta-data and filename data structures and to exclude from the RC-image things like the NV memory page area.

3.7 Synchronizing HAL Board Configuration, Linker Control & Meta-Data

There are a few inter-dependencies between the HAL board configuration, the linker control file, and the meta-data that cannot be automatically enforced with the IAR toolset. So, as an example of what to look for, consider the example of a project that uses the last 2 flash pages before the lock-bits page as the area for NV storage. As a general rule of thumb, it will be preferred to save this NV information across boot loads, so it must be excluded from the RC-image that will be boot loaded. Since the lock-bits page is the last available flash page and this UBL is for F256K parts, that page is 127. So the pages to preserve across boot loads for NV storage are 125-126 which correspond to logical addresses 0x7E800 – 0x7F7FF. Thus you should find in a HAL board configuration file such as hal_board_cfg.h in several stacks, an area so reserved such as this:

```
#define HAL_NV_PAGE_END             126
#define HAL_NV_PAGE_CNT             2
```

UBL uses the 5 preceding pages, 120-124. Thus the linker file for the UBL includes corresponding address range, 0x7C000 – 0x7F7FF. The following should be found in “ubl.xcl”:

```
//
// CODE Banks - Taking pages 120-124
//
-D_BANK7_BEG=0x7C000        // First byte address of page 120
-D_BANK7_END=0x7E7FF        // Last byte address of page 124
```

As the above range is reserved for the UBL code, the Application’s linker file, cc254x_f256_ubl_msd.xcl, should define its last code bank as follows:

```
-D_BANK7_BEG=0x78000
// Skip UBL pages 120-124; NV pages 125-126; Lock-bits page 127.
-D_BANK7_END=0x7BFFF
```

And remember to also exclude these corresponding memory areas from the checksum calculation by making sure the following is found at the very bottom of the Application’s linker file:

```
//
// Skip SBL, CRC & SHDW, and NV pages when calculating the CRC.
//
-J2,crc16,=_CODE_BEG-2E9B,3000-7BFFF
```

Finally, prevent the UBL from erasing or writing to these pages by clearing the corresponding bits in the ERASE/WRITE-Enable bit arrays in ubl_app.c:

```
{ // Disable erase of the NV: flash pages 121-126.
 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x81
},
{ // Disable write of the NV: flash pages 121-126.
 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x81
},
```


It is optional to prevent reading out these NV pages, but it is recommended to lock out the reading of NV since it will store sensitive information like security keys:

```
{ // Disable read of all flash pages to protect code from reverse-compilers & NV from access.
 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
},
```

If you do lock out the reading of NV or the entire flash area (the UBL can never be read out), you can always re-gain access to reading any RC-image page (including the NV pages) by replacing the DEFAULT.CFG file with a configuration file with a valid security key.

Note that a set of sample configuration binary files were checked into SVN in a sub-directory of where this document resides.

3.8 Combining UBL and Application

Download UBL project to your hardware and it should appear as an attached USB MSB device. Compile the application Application and look for the ".bin" file it output. Copy this file into the USB MSB directory and reset or power cycle and UBL will jump to the start of the Application code and begin execution.

