RemoTI Network Processor Developer's Guide

Document Number: SWRU223D

TABLE OF CONTENTS

Acronyms and Definitions

ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
ARC	Advance Remote Control
bps	bits per second
CCM	Counter with CBC-MAC (CCM), a mode of operation for cryptographic block ciphers
CDC	Communications Device Class
CERC	Consumer Electronics Remote Control, name of a profile of Zigbee RF4CE (now ZRC)
DCE	Data Circuit-terminating Equipment
DMA	Direct Memory Access
DTE	Data Terminal Equipment
GPIO	General Purpose Input Output
HAL	Hardware Abstraction Layer
I2C	Inter-Integrated Circuit
IAR	IAR Systems, a software development tool vendor
IDATA	Internal Data memory
IEEE	Institute of Electrical & Electronics Engineers, Inc.
IO	Input Output
ISR	Interrupt Service Routine
LED	Light Emitting Diode
NIB	Network Information Base
NV	Non-Volatile, or Non-volatile memory
NWK	Network
OSAL	Operating System Abstraction Layer
RXD	Receive Data line
SOP	Start Of Packet
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TI	Texas Instruments Incorporated
TXD	Transmit Data line
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
XDATA	eXternal Data memory
Zigbee RF4CE	An 802.15.4 based remote control protocol standard
ZID	Zigbee Input Device, a name for a ZigBee RF4CE profile
ZRC	Zigbee Remote Control, a name for a ZigBee RF4CE profile

1 References

[1] RemoTI Developer's Guide, SWRU198, *http://www.ti.com/lit/swru198*
[2] RemoTI API, SWRA268, *can be found in the installation folder of RemoTI*
[3] HAL Drivers API, SWRA193, *can be found in the installation folder of RemoTI*
[4] OSAL API, SWRA194, *can be found in the installation folder of RemoTI*
[5] CC253X System-on-Chip Solution for 2.4-GHz IEEE 802.15.4/ZigBee/RF4CE User's Guide, SWRU191, *http://www.ti.com/lit/swru191*
[6] RemoTI Sample Applications User's Guide, SWRU201, *can be found in the installation folder of RemoTI*
[7] RemoTI Network Processor Interface Specification, SWRA271, *can be found in the installation folder of RemoTI*
[8] RemoTI Host Processor Sample Application and Porting Guide, SWRA259, *http://www.ti.com/lit/swra259,*

2 Introduction

2.1 Purpose

This document explains the RemoTI network processor application and topics related to customizing the application to add custom command set.

2.2 Scope

This document describes concepts and settings for the Texas Instruments RemoTI Release with respect to network processor development.

As to the general concept of Zigbee RF4CE and RemoTI architecture, please refer to [1].

3 RemoTI Network Processor Application

The RemoTI development kit includes the network processor application. This chapter describes the features of the network processor application and the organization of the source code and project files.

3.1 Features

The following summarize the features of the RemoTI network processor application:

- Compliance with Zigbee RF4CE network layer specification, either as a controller node or as a target node, which is dynamically configurable
- Zigbee RF4CE network layer security
- 115200bps baud rate UART connectivity with two pin configuration (TXD, RXD)
    - 9600bps, 19200bps, 38400bps and 57600bps alternative baud rate for UART connection with code modification instruction. Note that host processor emulation tools (PC tools) included in the development kit does not support the alternative baud rates.
- Configurable to SPI (up to 4 Mhz) or I2C connectivity (up to 400 Kbps). Note that host processor emulation tools (PC tools) included in the development kit does not support SPI nor I2C connection.
- Wakeup on UART protocol
- Optional serial boot loader downloading demo
- Full speed USB CDC support for CC2531 USB dongle

3.2 Build configurations

RemoTI network processor application project is located in the Projects\RemoTI\RNP\CC253x folder of the RemoTI software installation. There are 3 workspaces defined in this folder, please select the one corresponding to your target device type (CC2530, CC2533 or CC2531).

When you open the workspace file (rnp_cc2533.eww as an example), you could select different project configurations as in figure 1.

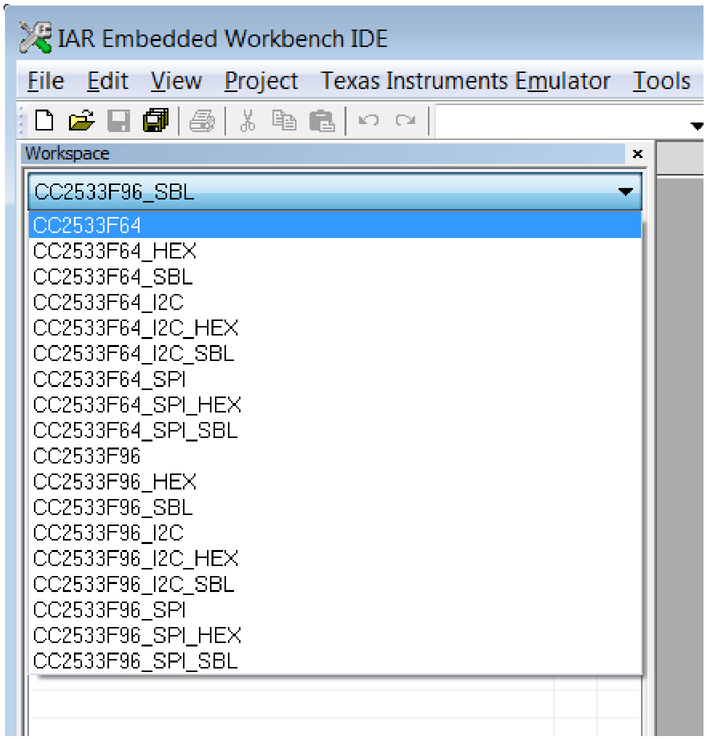

Figure 1 – IAR project configuration selection

Each configuration is explained in the following Table 1:

Table 1 – Project configurations

Configuration	Description
CC2533F64	Configuration for CC2533F64 part, UART interface.
CC2533F64_HEX	Configuration for CC2533F64 part, UART interface for serial boot loader – HEX file
CC2533F64_SBL	Configuration for CC2533F64 part, UART interface for serial boot loader downloading
CC2533F64_I2C	Configuration for CC2533F64 part, I2C interface.
CC2533F64_I2C_HEX	Configuration for CC2533F64 part, I2C interface for serial boot loader – HEX file
CC2533F64_I2C_SBL	Configuration for CC2533F64 part, I2C interface for serial boot loader downloading
CC2533F64_SPI	Configuration for CC2533F64 part, SPI interface.

Configuration	Description
CC2533F64_SPI_HEX	Configuration for CC2533F64 part, SPI interface for serial boot loader – HEX file
CC2533F64_SPI_SBL	Configuration for CC2533F64 part, SPI interface for serial boot loader downloading
CC2533F96	Configuration for CC2533F96 part, UART interface.
CC2533F96_HEX	Configuration for CC2533F96 part, UART interface for serial boot loader – HEX file
CC2533F96_SBL	Configuration for CC2533F96 part, UART interface for serial boot loader downloading
CC2533F96_I2C	Configuration for CC2533F96 part, I2C interface.
CC2533F96_I2C_HEX	Configuration for CC2533F96 part, I2C interface for serial boot loader – HEX file
CC2533F96_I2C_SBL	Configuration for CC2533F96 part, I2C interface for serial boot loader downloading
CC2533F96_SPI	Configuration for CC2533F96 part, SPI interface.
CC2533F96_SPI_HEX	Configuration for CC2533F96 part, SPI interface for serial boot loader – HEX file
CC2533F96_SPI_SBL	Configuration for CC2533F96 part, SPI interface for serial boot loader downloading

Configurations in rnp_cc2530.eww workspace

Configuration	Description
CC2530F128	Configuration for CC2530F128 part, UART interface
CC2530F128_HEX	Configuration for CC2530F128 part UART interface for serial boot loader – HEX file
CC2530F128_SBL	Configuration for CC2530F128 part UART interface for serial boot loader
CC2530F128_SPI	Configuration for CC2530F128 part, SPI interface

Configuration	Description
CC2530F128_SPI_HEX	Configuration for CC2530F128 part SPI interface for serial boot loader – HEX file
CC2530F128_SPI_SBL	Configuration for CC2530F128 part SPI interface for serial boot loader
CC2530F256	Configuration for CC2530F256 part, UART interface
CC2530F256_HEX	Configuration for CC2530F256 part UART interface for serial boot loader – HEX file
CC2530F256_SBL	Configuration for CC2530F256 part UART interface for serial boot loader
CC2530F256_CC2591	Configuration for CC2530F256 part UART iinterface with the CC2591 RF frontend chip
CC2530F256_SPI	Configuration for CC2530F256 part, SPI interface
CC2530F256_SPI_HEX	Configuration for CC2530F256 part SPI interface for serial boot loader – HEX file
CC2530F256_SPI_SBL	Configuration for CC2530F256 part SPI interface for serial boot loader

Configurations in rnp_cc2531.eww workspace

Configuration	Description
CC2531F256	Configuration for CC2531F256 USB dongle platform – CDC interface
CC2531F256_HEX	Configuration for CC2531F256 USB dongle platform hex file generation – CDC interface
CC2531F256_SB	Configuration for CC2531F256 image for serial boot loader downloading – CDC interface

The project option settings for each configuration, such as defined symbols (also known as compile flags) preprocessor, are set to work for the particular configuration.

Table 2 explains compile flags (preprocessor defined symbols) used in the project configurations.

Table 2 – Compile flags

Compile Flag	Description
POWER_SAVING	When defined, power-saving modes are enabled. Without the compile flag, CC2530 PM2 and PM3 are not exercised. The compile flag affects HAL sleep module, OSAL power management module, RemoTI application framework (RTI) and network processor module. Power-saving modes are not compatible with CC2531 USB dongle and hence this compile flag must not be added to configurations for CC2531.
HAL_BOARD_CC2530RB	Platform board selection for RemoTI network processor project. This compile flag selects RemoTI Target Board as the hardware platform. This compile flag is defined by default for all CC2530 configurations, unless HAL_BOARD_CC2530EB_REV13, HAL_BOARD_CC2530EB_REV17 or HAL_BOARD_CC2530EB_REV18 is defined.
HAL_BOARD_CC2530EB_REV13	Platform board selection for RemoTI network processor project. This compile flag selects SmartRF05 revision 1.3 board with CC2530EM as the hardware platform.
HAL_BOARD_CC2530EB_REV17	Platform board selection for RemoTI network processor project. This compile flag selects SmartRF05 revision 1.7 board with CC2530EM as the hardware platform.
HAL_BOARD_CC2530EB_REV18	Platform board selection for RemoTI network processor project. This compile flag selects SmartRF05 revision 1.8 board with CC2530EM as the hardware platform.
HAL_KEY	HAL key module feature flag. RNP does not include HAL key module and hence this compile flag has to be set to FALSE either in the hal_board_cfg.h file as default or in preprocessor definition setting of a project configuration. hal_board_cfg.h file for CC2531 set this compile flag to TRUE since another project which shares this file uses HAL key module. Hence, the compile flag is set to FALSE in the preprocessor definition setting for the CC2531F256

Compile Flag	Description
	configuration.
HAL_PA_LNA	The compile flag is for the board configuration of a CC2591 frontend chip connected to the CC2530.
HAL_PA_LNA_CC2590	This compile flag is reserved for the board configuration of a CC2590 frontend chip connected to the CC2530. This configuration is not supported with this release.
CC2530F128	Non-volatile memory configuration selection for CC2530F128. Note that default configuration of non-volatile memory is for CC2530F256 part without definition of either CC2530F64 or CC2530F128 compile option.
CC2533F64	Non-volatile memory configuration selection for CC2533F64.
CC2533F96	Non-volatile memory configuration selection for CC2533F96.
FEATURE_TEST_MODE	RTI test mode API functions shall be enabled with this compile flag. The compile flag affects RTI and RTI surrogate.
FEATURE_CONTROLLER_ONLY	This compile flag, when defined, reduces RTI code size when RTI is compiled for remote controller functionality only.
FEATURE_SBL	This compile flag, when defined, enables serial boot loading feature in RTI surrogate for RemoTI network processor. Note that defining this compile flag alone does not enable serial boot loader for the whole RemoTI network processor (RNP) sample application. Choose a proper configuration (with SB tag) in the provided project to enable the serial boot loading feature. Such a configuration includes this compile flag definition in the project settings. This flag will enable a build of an image binary suitable for Serial Bootloader downloading.
FEATURE_HEX	This compile flag, when defined, enables serial boot loading feature in RTI surrogate for RemoTI network processor. This flag will enable a build of an image which will include the NV area. Note that defining this compile flag alone does not enable

Compile Flag	Description
	serial boot loader for the whole RemoTI network processor (RNP) sample application. Choose a proper configuration (with HEX tag) in the provided project to enable the serial boot loading feature. Such a configuration includes this compile flag definition in the project settings. All the configurations with the _HEX postfix will invoke a build of the proper configuration of the Serial Bootloader project in the pre build option. The post build option will invoke a post build tool which will combine the Serial Bootloader, built with the pre build option, with the image built by the _HEX configuration.
HAL_UART	This flag is used when UART serial port is used. This is the default setting if no other serial interface is used
HAL_I2C	This flag is used when I2C serial port is used. This must be defined when using I2C interface
NPI_I2C_SRDY_ACTIVE_LOW	This flag is used when I2C serial port is used. This must be defined when using I2C interface
NPI_I2C_MRDY_ACTIVE_LOW NPI_I2C_MRDY_ACTIVE_HIGH	This flag is used when I2C serial port is used. This must be defined when using I2C interface
HAL_SPI	This flag is used when SPI serial port is used. This must be defined when using SPI interface
FEATURE_ZID_CLD	This flag needs to be defined if you want to use RNP as a ZID class device
FEATURE_ZID_ADA	This flag needs to be defined if you want to use the RNP as a ZID HID adapter
GENERIC=__generic	This compile flag shall always be defined as GENERIC=__generic to be compatible with the RemoTI library files. The compile flag was devised to add IAR specific compiler keyword to certain function parameters.
SPI_CONFIG_ON_PORT1	This flag needs to be defined if you want SRDY and MRDY to be mapped on Port 1: SRDY: P1.2 MRDY: P1.3 If not defined SPI bus SRDY and MRDY are mapped on port

Compile Flag	Description
	0: SRDY: P0.4 MRDY: P0.3 Not that SPI bus is always mapped on Port 1, from P1.4 to P1.7. (see [5] for more details)
I2C_CONFIG_ON_PORT1	This flag needs to be defined if you want SRDY and MRDY to be mapped on Port 1 (P1.2 for SRDY, P1.3 for MRDY). If not defined, SRDY and MRDY are mapped on port 0 (P0.6 for SRDY, P0.7 for MRDY). Note that I2C bus is always mapped on Port2, from P2.2 to P2.3.
INT_HEAP_LEN	Value assign to this flag will specify the size of the Heap.

Besides the compile flags, other settings such as code model were also set to fit the configuration. For instance, CC2533F64 configuration uses near code model while other configurations use banked code model.

3.3 Files

C source files and library files are explained in Table 3 in the order they appear in the IAR workspace window. Note that there are more files than those listed in the table, such as C header files that defines constants and function prototypes and that even workspace project does not list all header files referenced by the C files.

Note that certain driver modules are included although they are not actually used, just for potential use of the drivers by custom application.

Table 3 – Project files

File name	Description
Application	
mac_rffrontend.c	RF frontend chip (either CC2591 or CC2590) connection configuration module. This file is default set to fit the connection in a CC2530-CC2591EM 2.0 board.
np_main.c	Application main entry routine. This module initializes OSAL tasks.
rcn_config.c	Network layer configuration file. The file contains global variables with initial values which are used as configuration parameters by RemoTI network layer. The configuration parameters are explained

File name	Description
	in chapter 11.
HAL	
hal_assert.c	HAL assertion library
hal_drivers.c	Entry point for congregation of HAL drivers, such as initialization for all HAL drivers, HAL task, as an OSAL task, entry point (event handler) and polling entry point.
hal_rpc.h	Remote procedure call enumerations
hal_adc.c	ADC device driver
hal_aes.c	AES device driver
hal_batmon.c	Battery monitor service driver
hal_board_cfg.h	RemoTI hardware platform specific configuration parameters and macros used by HAL. Application may also use board definition literal (HAL_BOARD_CC2530RB) and HAL feature flags (HAL_KEY, HAL_LED, etc).
hal_ccm.c	CCM implementation using AES device driver
hal_dma.c	DMA device driver
hal_i2c.c	I2C peripheral interface driver
hal_flash.c	CC253x flash device driver
hal_led.c	LED driver (not in use)
hal_sleep.c	Sleep mode (PM1, PM2, PM3) control implementation
hal_spi.c	SPI device driver
hal_startup.c	Low level init.
hal_uart.c	UART device driver
hal_vddmon.c	VDD monitor driver
HAL / USB CDC class specific modules	

File name	Description
usb_cdc_hooks.c	hook functions for various USB request processing, specific to USB CDC class
usb_firmware_library_config.c	USB library configuration
usb_RemoTICdcDescriptor.s51	USB descriptors specific to RemoTI USB network processor dongle
HAL / USB generic firmware library for CC2531	
usb_board_cfg.h	Collection of macros abstracting the hardware details for USB control.
usb_interrupt.c	USB interrupt initialization routine and USB interrupt service routine
usb_suspend.c	USB suspend mode related subroutines
usb_descriptor_parser.c	Parser for USB descriptor structures
usb_framework.c	Main interface routines for USB generic library
usb_standard_request.c	Handlers for USB standard requests
Libraries	
rcnsuper-CC253x-banked.lib	RemoTI network layer library built for banked code model. This library will be selected for F96 configuration.
rcnsuper-CC253x.lib	RemoTI network layer library built for near code model. This library will be selected for F64 configuration.
OSAL	
OSAL.c	OSAL implementation for messaging and main event handling loop
OSAL_Clock.c	OSAL clock tick implementation
OSAL_Memory.c	OSAL heap implementation
OSAL_PwrMgr.c	OSAL power management scheme implementation
OSAL_Timers.c	OSAL timer implementation
osal_snv.c	OSAL Simplified Non-Volatile memory manager. This has better code size optimization than OSAL_Nv.c module.

File name	Description
OSAL_Math.s51	Optimized assembly code for mathematical function (division)
RPC	
npi.c	Network processor interface module. This module includes either npi_uart.c, npi_i2c.c or npi_spi.c file depending on configuration.
rcns.c	RemoTI network layer surrogate module. This module is called by RTI surrogate module and it serializes and de-serializes RemoTI network layer function call interfaces (absent in the CC2533 configuration).
rtis_np.c	RemoTI application framework (RTI) surrogate module. This module handles network processor interface packets and serializes and de-serializes RemoTI application framework (RTI) function call interfaces.
PROFILES\GDP	
gdp.c	GDP profile implementation layer
rti.c	RemoTI application framework implementation
rti_testmode.c	RemoTI test mode API function implementation
PROFILES\ZID	
zid_common.c	ZID profile implementation common (adaptor and class-device) co-layer
zid_class_device.c	ZID profile implementation class-device co-layer
zid_adaptor.c	ZID profile implementation adaptor co-layer
SerialBoot	
sb_target.c	Image preamble definition for use by serial boot loader, and serial boot loading command packet handler.

3.4 Architecture

This section briefly explains the interactions and relationship among the modules represented by files described in previous section. See [7] for the architectural description of the network processor as a whole.

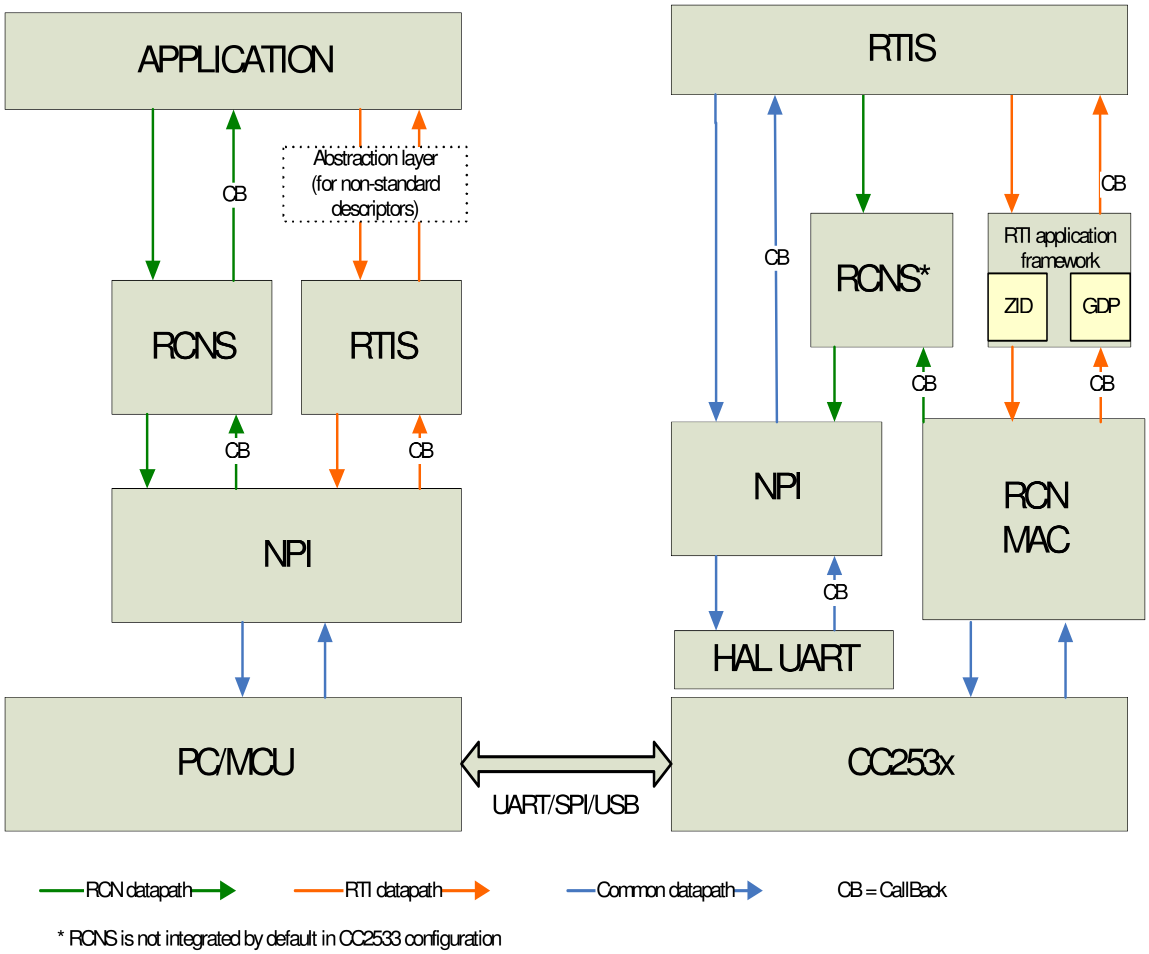

Figure 2 – RemoTI network processor component architecture

Figure 2 illustrates inter-module interactions within RemoTI network processor on the right hand. On the left hand is the inter-module interactions of the PC tools as emulating host processor.
Each network processor module acronym is explained below:

- RCN – RemoTI network layer
- RCNS – RemoTI network layer surrogate
- RTI – RemoTI application framework
- RTIS – RemoTI application framework surrogate
- NPI – Network Processor Interface
- MAC – Medium Access Control layer (it is part of RemoTI network layer library in the file list)
- HAL UART – Hardware Abstraction Layer UART driver
- GDP – Generic Device Profile implementation co-layer
- ZID – Zigbee Input Device profile implementation co-layer
- Thin abstraction layer, needed only in case of ZID non standard descriptors, to access ZID proxy non-standard descriptor attributes through RTI. This abstraction layer implements fragmentation and re-composition of ZID proxy attributes through separate RTI API calls

Either RTI or RCN interface is selected dynamically using different path among the modules.

The following modules have their own OSAL tasks (which is different from the generic meaning of OS tasks: The OSAL tasks do not own its own thread context. It is merely an entity leveraging OSAL event, messaging and power management mechanism):

- MAC module
- RCN module
- RTI module
- ZID module tasks (separate task for ZID class device and ZID adaptor)
- NPI module
- HAL module
- Application main entry module

Amongst the modules above, application main entry module does not use any OSAL services. However it creates an OSAL task for any custom extension, following usual sample application template.

4 Baud Rate

RemoTI network processor was tested with 115200bps baud rate. However, hal_uart.c module supports 9600bps, 19200bps, 38400bps and 57600bps as well and it is possible to modify network processor to work with one of these baud rates. Note that use of lower baud rate has to be compatible with the use cases. For instance, the average throughput over the UART must not be higher than selected baud rate and heap size has to be adjusted depending on how long peak data rate that is higher than selected baud rate lasts in the worst case. When data rate of network processor interface packets towards host processor is higher than that of UART transport, the data packets occupies heap space until they can be processed out to UART transport layer. Hence, the longer and the higher the peak data rate is, the more heap memory is used. The default heap size (1024 bytes) is set to fit Target Emulator use case (test mode operation from a single remote or CERC packets from multiple remotes). For more stressful use case (reception of more lasting contested traffic from multiple remotes), start with bigger heap size (for instance, 2048 bytes) at the beginning of development and optimize heap size down. See chapter 9 for method to profile and adjust heap size.

In order to switch baud rate, change the NPI_UART_BAUD_RATE macro value in Projects\RemoTI\common\cc2530\npi_uart.c file.

The default setting is as follows:

```
#define NPI_UART_BAUD_RATE   HAL_UART_BR_115200
```

To change the baud rate to 57600bps for instance, change the line as follows:

```
#define NPI_UART_BAUD_RATE   HAL_UART_BR_57600
```

Note that the host processor UART baud rate has to be modified to match the changed baud rate of network processor.

Note that SPI baud rate is controlled by host processor, for host processor SPI is the master and the network processor SPI is the slave.

CC2531 uses USB as the underlying physical connection with the host processor and supports USB CDC interface. USB CDC interface baud rate is selectable from the USB host.

5 UART wake up mechanism

Network processor supports two-pin configuration for UART (RXD and TXD).

Hence, there is no difference whether network processor is DTE or DCE. When a PC is used as the host processor in use with the RemoTI Target Board provided with the development kit, the PC assumes the role of DTE, but the concept of DTE/DCE ends within PC application and USB virtual serial port driver.

CC2530 does not support wakeup on UART receive line activity and hence network processor software dynamically configures UART receive pin as a generic IO pin to wake up on UART activity when the network processor is in sleep mode. In such a mechanism as this, more than one character may be lost till UART is fully functional exiting sleep mode. Hence, a handshake mechanism is added.

When waking up a network processor a null character (0x00) is sent to network processor and network processor responds with a null character (0x00) when it has set up UART.

Figure 3 and Figure 4 illustrate the sleep sequence and the wake up sequence each.

Note that the rti.c module changes its power management state to conserve power state during UART sleep sequence but it does not change its power management state back to hold power state during UART wakeup sequence. RTI power management state in network processor is used only upon boot up of the software till the first RTI_EnableSleepReq() and afterwards NPI power management state represents the latest sleep or wakeup command from host processor. The reason for having RTI power management state is to re-use the same RTI module implementation that is used by a CC2530 standalone application and also to have the consistent RTI API behavior.

Network processor did not incorporate any host processor wakeup mechanism as it cannot assume capabilities of a host processor. For host processors that could wake up its UART block on UART receive line activity within one character duration, the wakeup mechanism could be simply adding a preamble null character preceding a transmit UART frame from network processor. For other host processors that require its own wakeup handshaking, the npi_uart.c module has to be modified to add such handshaking.

Note that UART wakeup mechanism does not apply to CC2531. CC2531 emulates the same interface as UART over USB CDC interface but the device does not operate in the power-saving mode when RTI_EnableSleepReq() is called.

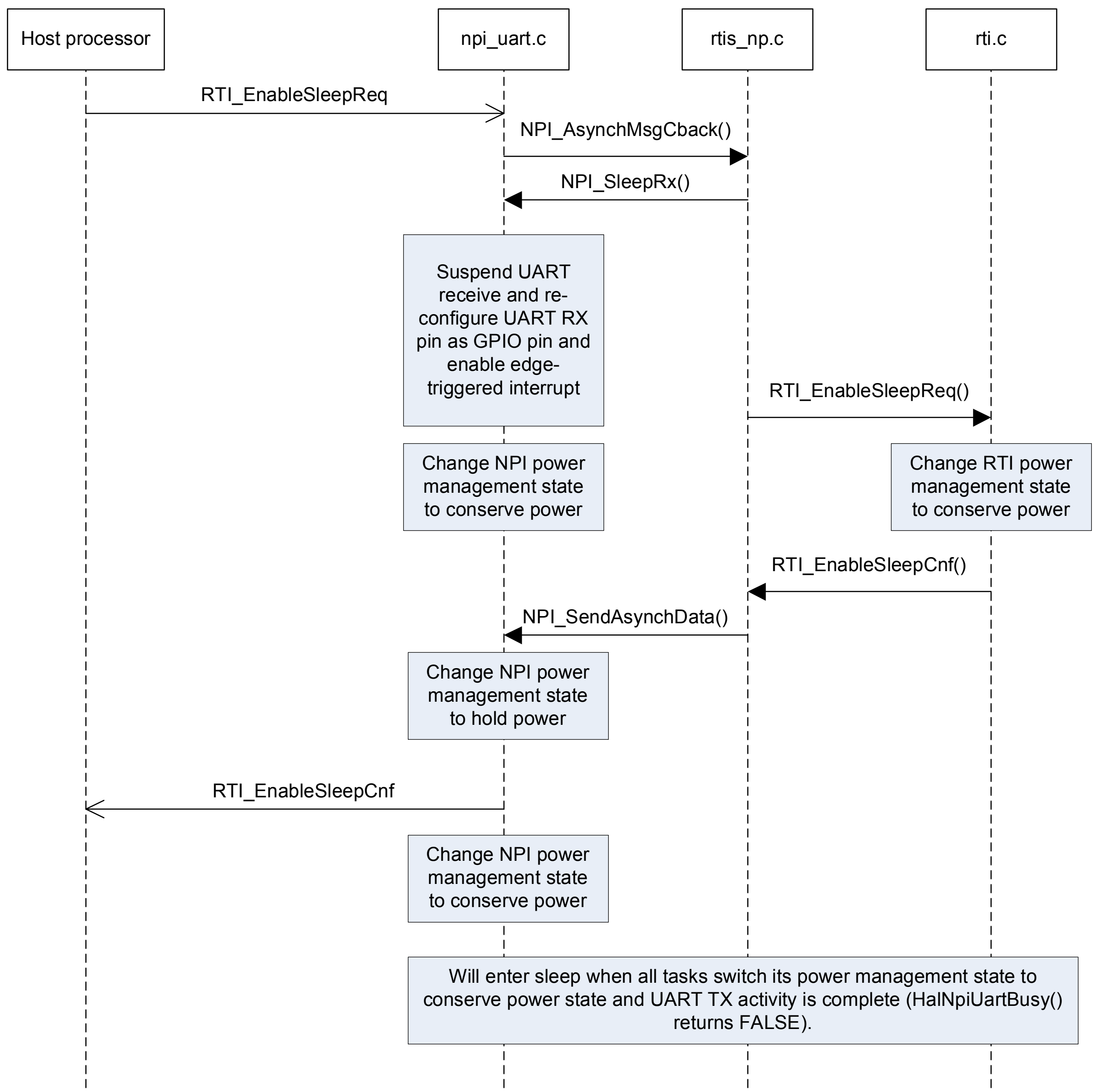

Figure 3 – UART sleep sequence

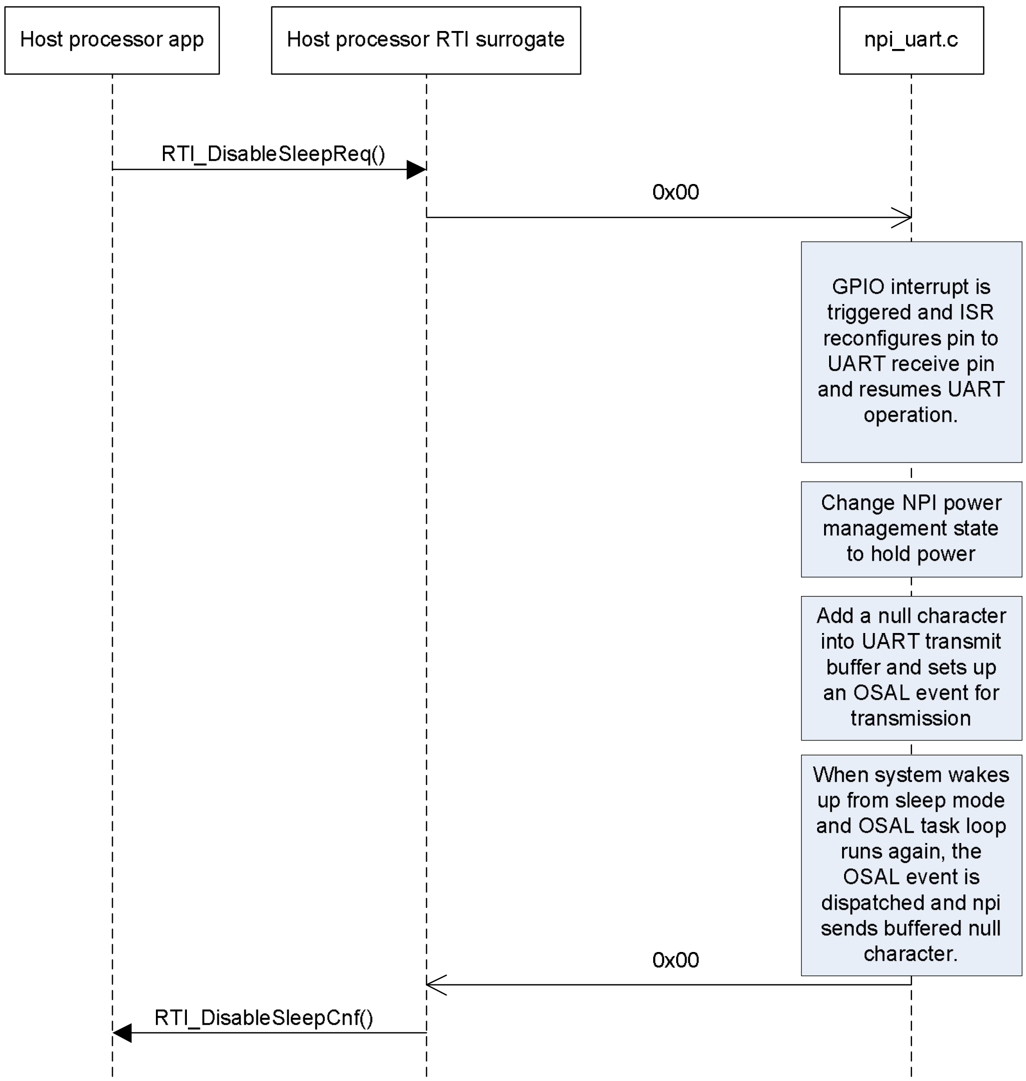

Figure 4 – UART wakeup sequence

6 Adding New Network Processor Interface Commands

[7] specifies all supported network processor interface commands. This chapter provides instructions on how to add additional commands which are not listed in [7].

Additional commands have to comply with the packet format specified in [7]. All network processor interface commands are forwarded to either NPI_AsynchMsgCback() function or NPI_SyncMsgCback() function depending on whether the command is an asynchronous request or a synchronous request (See

[7] for the definition of asynchronous request and synchronous request). Both functions are defined in the rtis_np.c file.

A new command can be added using its own subsystem identifier and command identifier. NPI_AsynchMsgCback() function and NPI_SyncMsgCback() function simply decodes subsystem identifier and command identifier and performs proper actions accordingly.

Currently used subsystem identifier values are listed in hal_rpc.h file. The following is an example. Look at the hal_rpc.h file in the released software to find the correct subsystem identifier values in use.

```
// RPC Command Field Subsystem
#define RPC_SYS_RES0        0
#define RPC_SYS_SYS         1
#define RPC_SYS_MAC         2
#define RPC_SYS_NWK         3
#define RPC_SYS_AF          4
#define RPC_SYS_ZDO         5
#define RPC_SYS_SAPI        6    // Simple API
#define RPC_SYS_UTIL        7
#define RPC_SYS_DBG         8
#define RPC_SYS_APP         9
#define RPC_SYS_RCAF        10   // Remote Control Application Framework
#define RPC_SYS_RCN         11   // Remote Control Network Layer
#define RPC_SYS_RCN_CLIENT  12   // Remote Control Network Layer Client
#define RPC_SYS_BOOT        13   // Serial Bootloader
#define RPC_SYS_MAX         14   // Maximum value, must be last
```

Note that NPI_SyncMsgCback() function has to store the response message into the same buffer where the request command is stored when the function is called. The buffer is de-referenced using the pMsg pointer argument.

The following code is an example of how a network processor interface command is processed.

```
void NPI_SynchMsgCback( npiMsgData_t *pMsg )
{
  if (pMsg->subSys == RPC_SYS_RCAF)
  {
    switch( pMsg->cmdId )
    {
    // read item request
    case RTIS_CMD_ID_RTI_READ_ITEM:
      // confirm message length has to be set up
      // and that before pMsg->pData[1] is overwritten.
      pMsg->len = 1 + pMsg->pData[1];

      // unpack itemId and len data and send to RTI to read config interface
      // using input buffer as the reply buffer
      // Note: the status is stored in the first word of the payload
      // Note: the subsystem Id and command Id remain the same, so we only
      //       need return to complete the synchronous call
      pMsg->pData[0] =  (rStatus_t)RTI_ReadItem( pMsg->pData[0], pMsg->pData[1],
&pMsg->pData[1] );
      break;
    /* Other case statements follow */
    }
  } /* if (pMsg->subSys == RPC_SYS_RCAF) */
```



```
  /* Other if statement may follow for other subsystems */
}
```

7 UART vs. SPI vs. I2C

The network processor SW comes with a set of configurations defined as in 3.2 which specify the interface being used. Table 4 compares UART, SPI and I2C.

Table 4 – UART vs. SPI vs I2C comparison

	UART	I2C	SPI
Highest peak baud rate	115200bps	400 Kbps	4Mbps
Number of signal pin connections (including network processor reset line)	3	5	7
Host processor portability (See [8])	Generic UART drivers are available for most of the platforms (e.g. PC has serial port driver out of the box). Network processor interface and RTI surrogate have to be ported.	Custom I2C driver has to be written per platform, as proprietary MRDY and SRDY line control is necessary. Network processor interface and RTI surrogate have to be ported.	Custom SPI driver has to be written per platform, as proprietary MRDY and SRDY line control is necessary. Network processor interface and RTI surrogate have to be ported.

8 Flash page map and memory map

Each configuration of network processor has a unique flash page map. Figure 5 illustrates two distinctive flash page maps used by network processor. One flash page is 2048 bytes for CC2530/2531 and 1024 for CC2533 as specified in [5]. OSAL used a system of virtual page, one virtual page is 2 physical page for CC2533.

For serial boot loading feature enabled configuration, the boot loader code occupies the first virtual flash page (page 0). The last virtual flash page is reserved for the flash lock bits and the commissioned IEEE address. OSAL non-volatile memory pages occupy configurable number of pages from the second last page down. Between the NV pages and the first page is the code space. The remainder of the last flash page cannot be used for code space because this page cannot be updated during serial boot loader execution. The details of serial boot loading feature enabled configuration are explained in chapter 12. Find more information about the last flash page and flash lock bits in [5].

Without serial boot loading feature, the code starts at the first page (lowest address page) up. OSAL non-volatile memory pages occupy configurable number of pages from the second last page down. The last flash page includes lock bits (last 16 bytes. See [5] for details) and commissioned IEEE address (8 bytes, prior to lock bits). IEEE address is explained more in chapter 10. The remainder of this last flash page can be used for additional code if the code fills up the rest of the space.

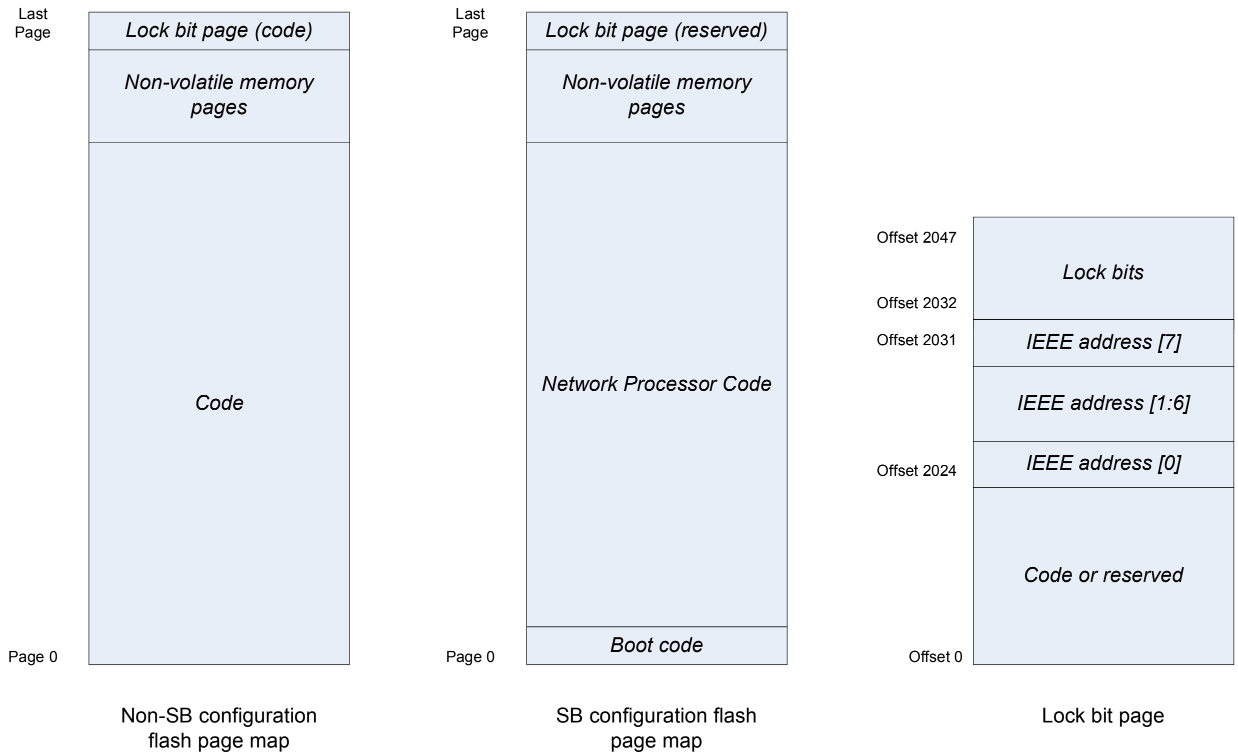

Figure 5 – Flash page map

The number of virtual pages used for OSAL non-volatile memory system is defined in hal_board_cfg.h file. The constants and their values are listed in Table 5. This application is using the osal_snv.c, Simple NV, driver. There can only be 2 virtual pages defined with this driver.

Constant Name	Description	CC2533F64	CC2533F96
HAL_NV_PAGE_END	HAL_FLASH_IEEE_PAGE (i.e. relative to lock bits page)	31	47
HAL_NV_PAGE_CNT	Number of OSAL NV pages	2	2

Table 5 – NV configuration constants

The linker command file can be located from project option pop up window. For instance after selecting the CC2533F96 configuration, select Project -> Options menu. In project option pop up window, select linker category and Config tab. The linker command file name and path is displayed as in **Error! Reference source not found.**.

In the linker command file, find _ZIGNV_ADDRESS_SPACE_START definition and change the starting address to match the number of pages defined. For instance, the default linker command file for CC2533F96 configuration has the following lines:

```
...
-D_FLASH_BEG=0x2E800          // Address range for HAL_FLASH_PAGE_SIZE == 2048
-D_FLASH_END=0x2F7FF          // Address range for HAL_FLASH_PAGE_SIZE == 2048
-P(CODE)ZIGNV_ADDRESS_SPACE=_FLASH_BEG-_FLASH_END
...
```

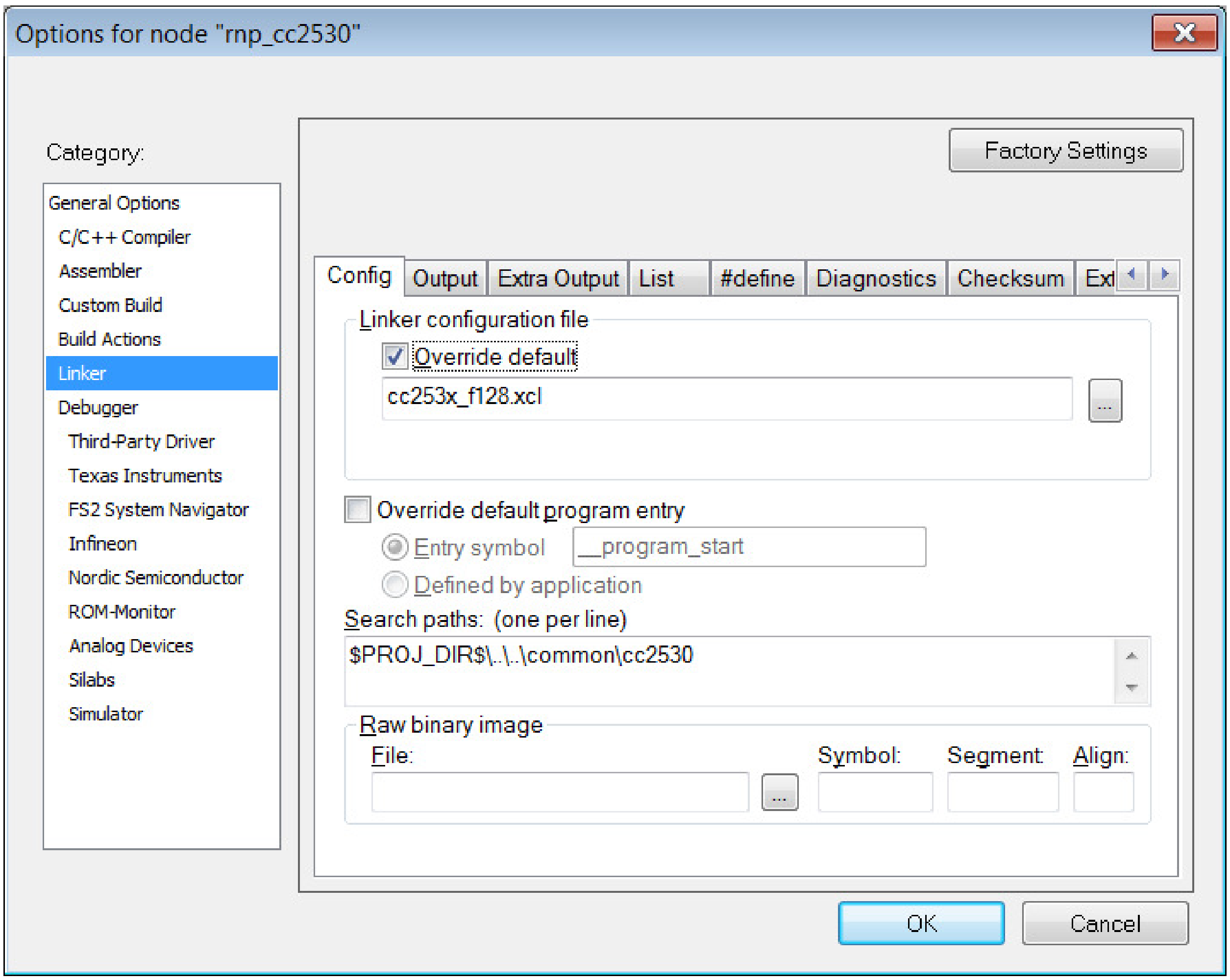

Figure 6 – Locating linker command file

XDATA memory map and CODE memory space are described in [5].

CC2530/33F64 configuration uses near code model and bank area is always occupied with the same code, non-volatile memory pages and lock bit pages content as in flash page map.

The CC2533F96, CC2530F128, CC2530F256, and CC2531F256 configurations use banked code model and bank area is dynamically mapped to flash bank (comprised of 16 pages) in use. Code address space is represented in virtual code address. Virtual address for code bank is listed in Table 6.

Table 6 – Virtual address of banked code

Code Bank	Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
Address Range	0x00000 – 0x07FFF	0x18000 – 0x1FFFF	0x28000 – 0x2FFFF	0x38000 – 0x3FFFF	0x48000 – 0x4FFFF	0x58000 – 0x5FFFF	0x68000 – 0x6FFFF	0x78000 – 0x7FFFF

Bank 0 is constantly mapped to common area (0x0000 – 0x7FFF) and the other banks are mapped to bank area (0x8000 – 0xFFFF) dynamically. CC2530F128 has up to bank 3. Bank 4 to bank 7 applies only to CC2530F256 and CC2531F256.

Such a bank set up is determined at link time and it is configured through linker configuration file. Linker configuration file can be found through project options in IAR (*Linker* category and then *Config* tab) as illustrated in Figure 6.

Figure 7 shows where to find near code model or banked code model setting in an IAR project options window.

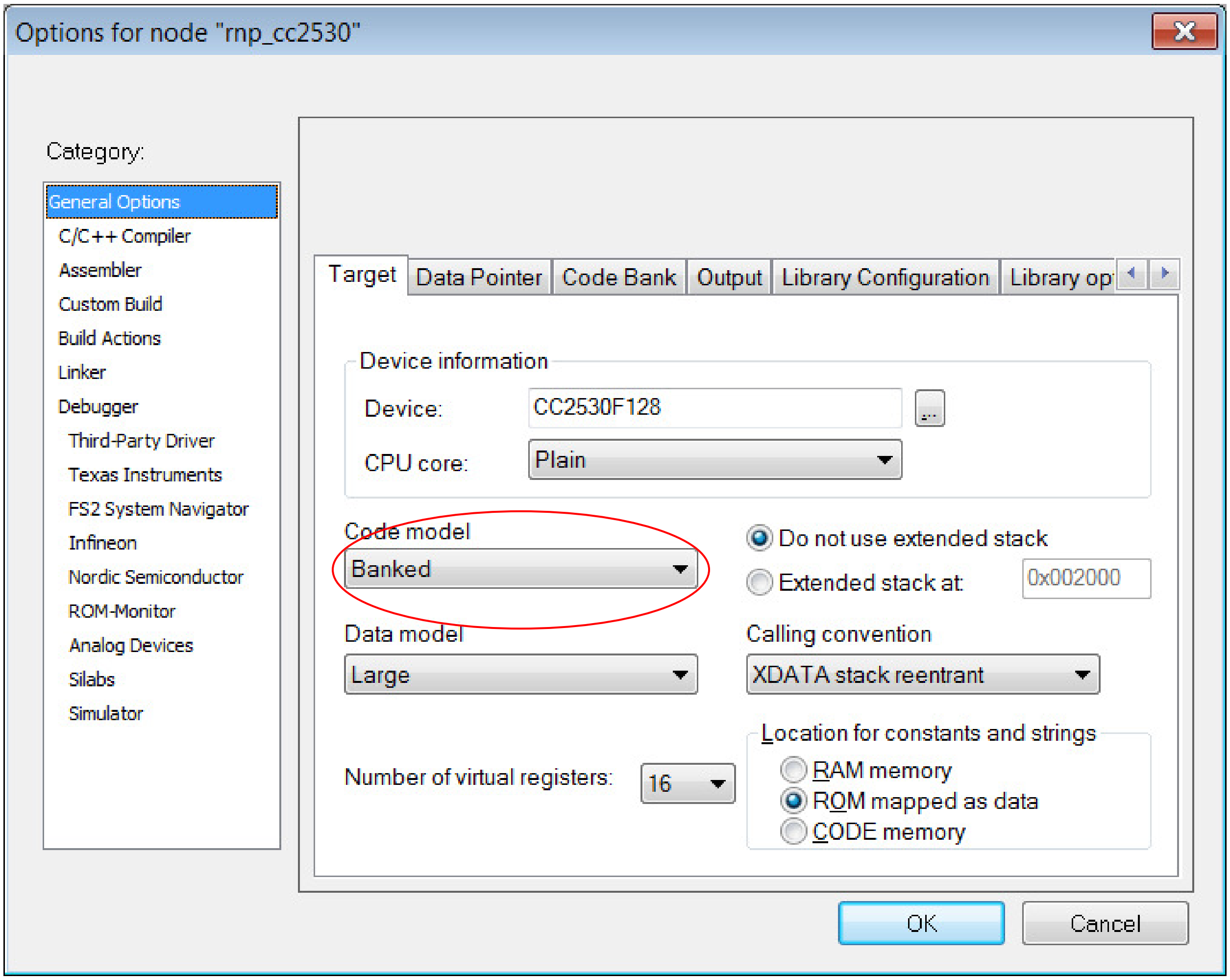

Figure 7 – Code model of a project

The CC2533 actually has a physical flash page size of 1024 vice 2048 on the CC2530/31 parts. The released, default configuration of OSAL NV is to combine two physical 1K pages into a virtual 2K page. If code space must be reclaimed from the OSAL NV module on the CC2533 part, it is possible to change the default setting from 2-2K pages to 2-1K pages. But if the OSAL NV page size is changed to be the physical page size of 1K, then writes to NV will cause more frequent page compactions and the lifetime endurance of the OSAL NV flash pages can be expected to be reduced. If such a reduction in lifetime can be tolerated and it is absolutely necessary to reclaim 2K of flash for code space, then one only need change a setting in the hal_board_cfg.h and corresponding linker .xcl file as follows.

In hal_board_cfg.h, change HAL_FLASH_PAGE_SIZE from 2048 to 1024 here:

```
#define HAL_FLASH_PAGE_PHYS         1024UL

//#define HAL_FLASH_PAGE_SIZE        (HAL_FLASH_PAGE_PHYS * 2)
define HAL_FLASH_PAGE_SIZE          HAL_FLASH_PAGE_PHYS
```


If building for SBL, increase the available code space in the linker file by changing _CODE_END from 0xE7FF to 0xEFFF here:

```
-D_CODE_END=0xE7FF
//-D_CODE_END=0xEFFF
```

Otherwise, reduce the available space for OSAL NV by changing _ZIGNV_ADDRESS_SPACE_START from 0xE800 to 0xF000 here:

```
//-D_FLASH_BEG=0xE800
-D_FLASH_BEG=0xF000
-D_FLASH_END=0xF7FF
-P(CODE)ZIGNV_ADDRESS_SPACE=_FLASH_BEG-_FLASH_END
```

9 Stack and Heap

8051 micro-controller uses a variety of data memory access methods. Generic data memory (i.e. not one specific for register access) are the internal data memory with 8 bit address space (IDATA) and the external data memory with 16 bit address space (XDATA). CC253x maps both memory address space to the same internal SRAM. See [5] for details. IAR compiler generates code to use stack from both IDATA and XDATA. How a compiled code uses IDATA and XDATA for stack is highly dependent on compiler itself.

With IAR 8051 compiler version 7.51A, RemoTI CC2530 development kit 1.0 network processor uses about 282 bytes of XDATA stack and 56 bytes of IDATA stack. However, the depth of the used stacks could change with even slight modification of the code as how compiler generates code to use stack is unpredictable.

Hence, 384 bytes of XDATA stack and 192 bytes of IDATA stack were reserved in project settings for RemoTI CC2530 development kit 1.0 network processor. Stack sizes can be adjusted after profiling the stack usage with the final application code, by browsing stack memory space through debugger.

For instance, XDATA stack is located between addresses 0x100 and 0x27F and IDATA stack is located between addresses 0x40 and 0xFF in case of RemoTI network processor CC2530F64 build, as could be found from generated map file.

IAR embedded workbench populates the value 0xCD to the entire XDATA stack and IDATA stack space when debugger resets CC253x.

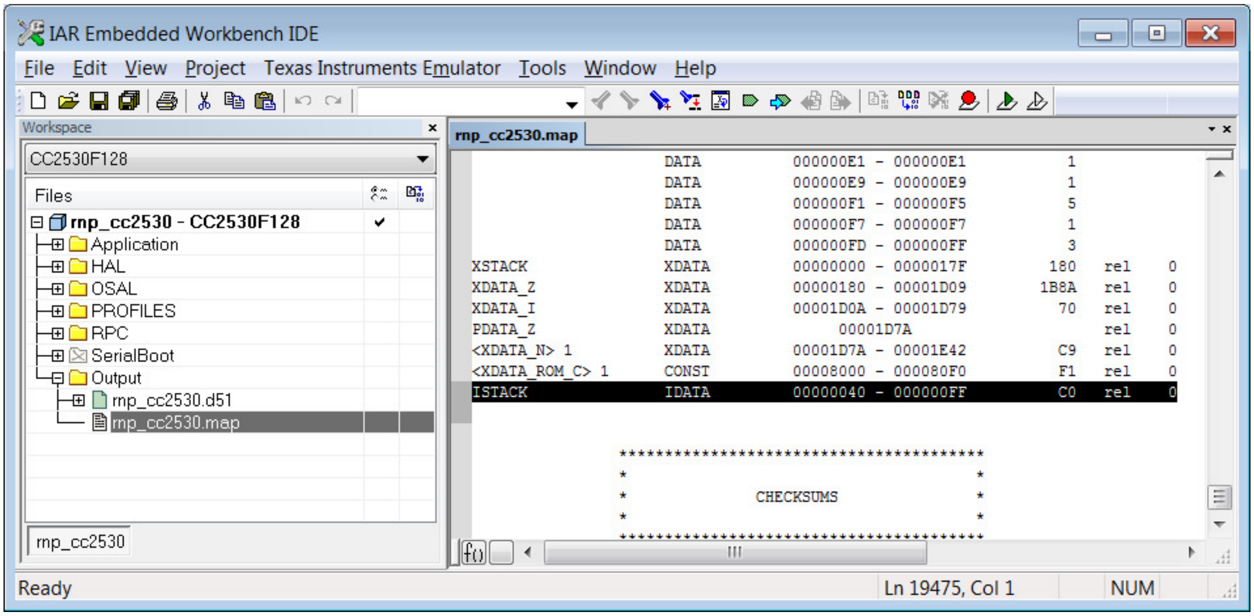

Figure 8 – Finding stack location

After running the application for the use cases picked for the deepest stack usage, the stack memory space can be browsed to determine how much stack was in use. In Figure 9, XDATA stack was used down to 0x59 (starting at 0x17F), which makes the stack depth in this use case to be 0x17F – 0x59 + 1 = 294 bytes.

IDATA stack usage can be profiled likewise. Just select IData to browse IData memory.

Once stack usage is profiled, the stack size can be adjusted from project settings (General Options category, Stack/Heap tab).

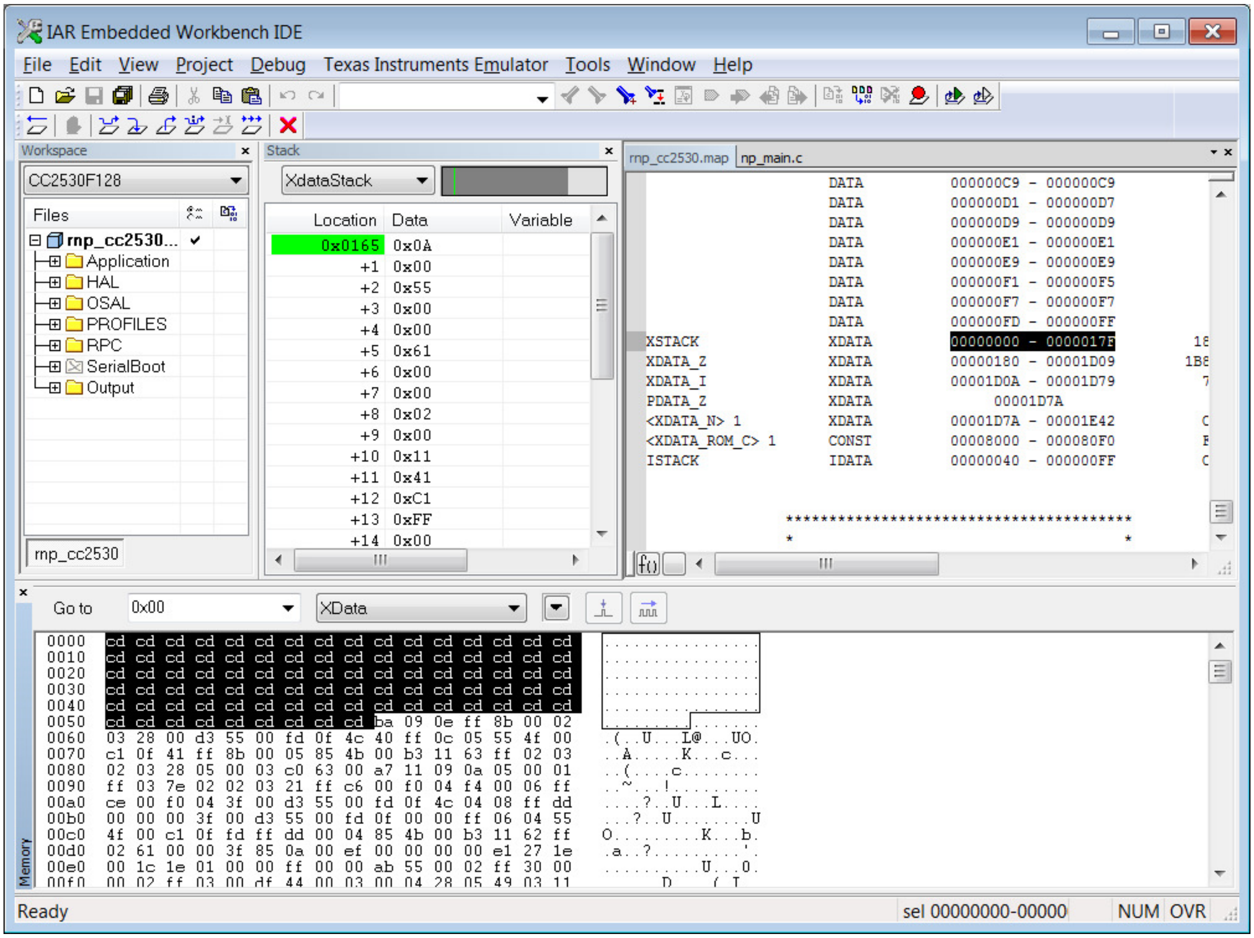

Figure 9 – XDATA Stack Profiling

RemoTI software uses heap through OSAL memory management module. The default heap size is in np_main.cfg file or by setting the compilation flag INT_HEAP_LEN. Heap usage varies drastically per use case even with the same software image. In other words, heap size has to be determined based on the supported use cases of the products. See chapter 4 for correlation with baud rate.

In order to profile heap usage, some OSAL code has to be instrumented. Unlike stack memory space, heap memory space is not initialized with a certain pattern of data (0xCD). Hence, it is necessary to add code to initialize the heap memory space before the space is being used.

The best location is osal_mem_init() function in OSAL_Memory.c module.

At the beginning of the function, add memory initialization code as follows:

```
void osal_mem_init( void )
{
  osalMemHdr_t *tmp;

#if ( OSALMEM_PROFILER )
  osal_memset( theHeap, OSALMEM_INIT, MAXMEMHEAP );
#endif
```



```
// Add this code to initialize memory space
extern void *osal_memset( void *dest, uint8 value, int len );
osal_memset( theHeap, 0xCD, MAXMEMHEAP );
```

Note that the OSALMEM_PROFILER compile flag is also supported. When the compile flag is defined as TRUE, the heap space is initialized with OSALMEM_INIT value. OSALMEM_PROFILE compile flag brings in more code than the heap initialization, which is not explained in this document.

By default the OSALMEM_INIT value is ‘X’ (0x58). Value ‘F’ (0x46) describes an area that has been allocated and freed.

With the new image, after running the use case with maximum heap usage, break the debugger and check the *_theHeap* memory space.

Address range of *_theHeap* variable can be found from map file.

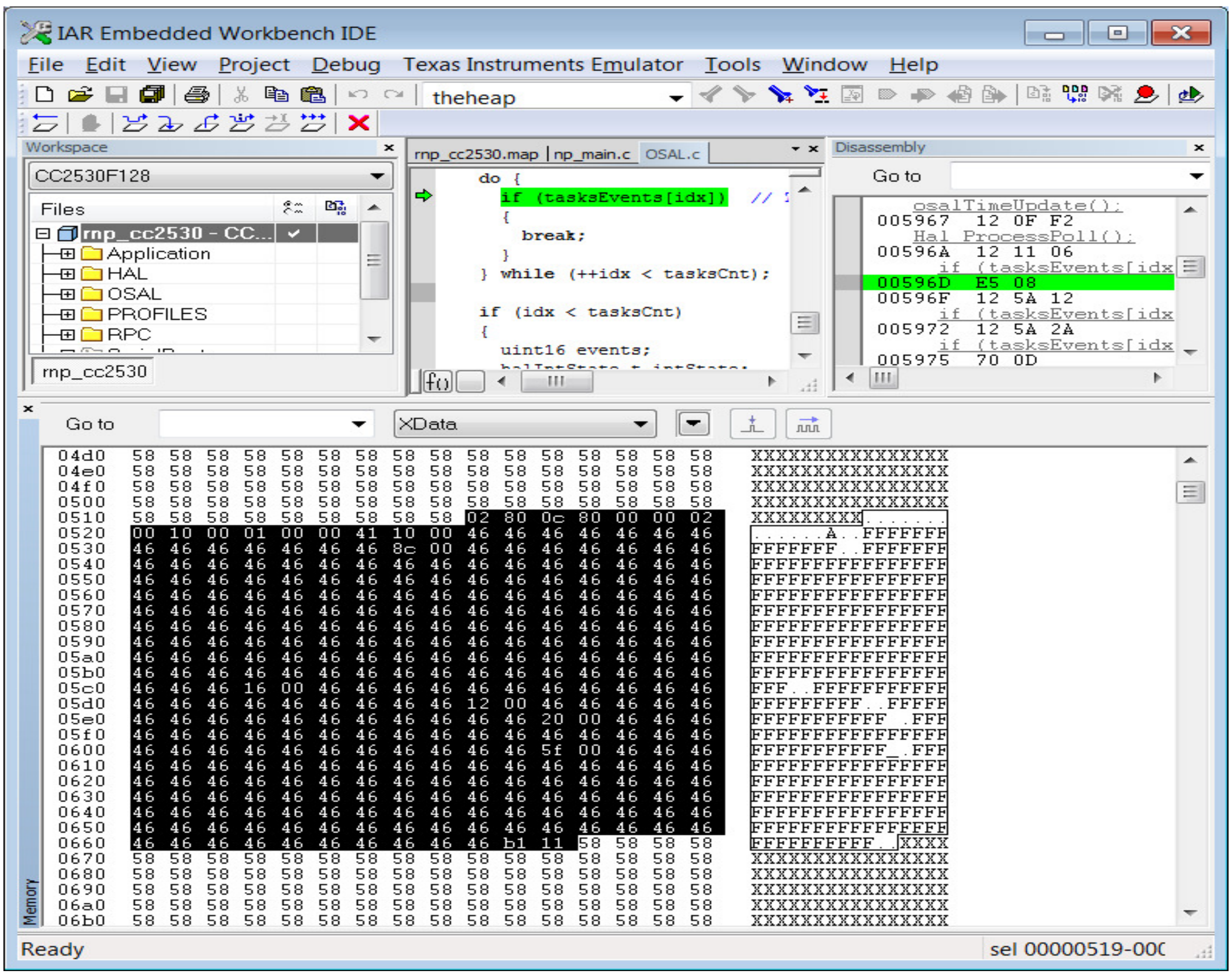

Figure 10 – Heap usage profiling

If the _theHeap variable occupies 0x431 to 0x181c address space for example, search from any foot print of memory usage. Note the heap is not used as the stack, memory can be allocated in several different places in the heap, and not always form the beginning of the Heap. You will need to go though the heap and count all value different than ‘F’.

In Figure 10, you can see memory used form 0x519 to 0x66B. That amounts to 0x66B - 0x519 + 1 = 338 bytes of heap usage.

Once heap size is profiled, the heap size can be adjusted by changing INT_HEAP_LEN definition as compile option in np_main.cfg file. For instance, replacing -DINT_HEAP_LEN=1024 with -DINT_HEAP_LEN=2048 in np_main.cfg file adjusts heap size to 2,048 bytes.

10 IEEE address

CC253x has its own IEEE address built into the chip (information page IEEE address). RemoTI network layer uses this IEEE address unless the IEEE address is overridden with a custom IEEE address by *RCN_NlmeSetReq*() call for *RCN_NIB_IEEE_ADDRESS* attribute. Once the IEEE address is overridden, network layer uses the custom IEEE address till this custom IEEE address is overwritten with another *RCN_NlmeSetReq*() call. If upper layer writes 0xFFFFFFFFFFFFFFFF as the custom IEEE address, network layer uses this null IEEE address till next power cycle. From next power cycle, network layer will start using the IEEE address built into the chip again.

RemoTI application framework, rti.c module, uses *RCN_NlmeSetReq()* to prioritize an IEEE address programmed to a specific last flash page location. See *rtiProgramIeeeAddr*() function for the source code. This function is called upon every system reset and the function reads the commissioned IEEE address in the special location and if it is valid (non-0xFFFFFFFFFFFFFFFF), this IEEE address is set to the network layer using *RCN_NlmeSetReq*() call. The special location is offset 0x7E8 of the last page stored in little-endian order, which neighbors lock bits which starts from offset 0x7F0. This is the location where SmartRF programmer will program the secondary IEEE address. The secondary location of IEEE address on the SmartRF Flash programmer window as in Figure 11 corresponds to the commissioned IEEE address while the primary location of IEEE address corresponds to the afore-mentioned information page IEEE address.

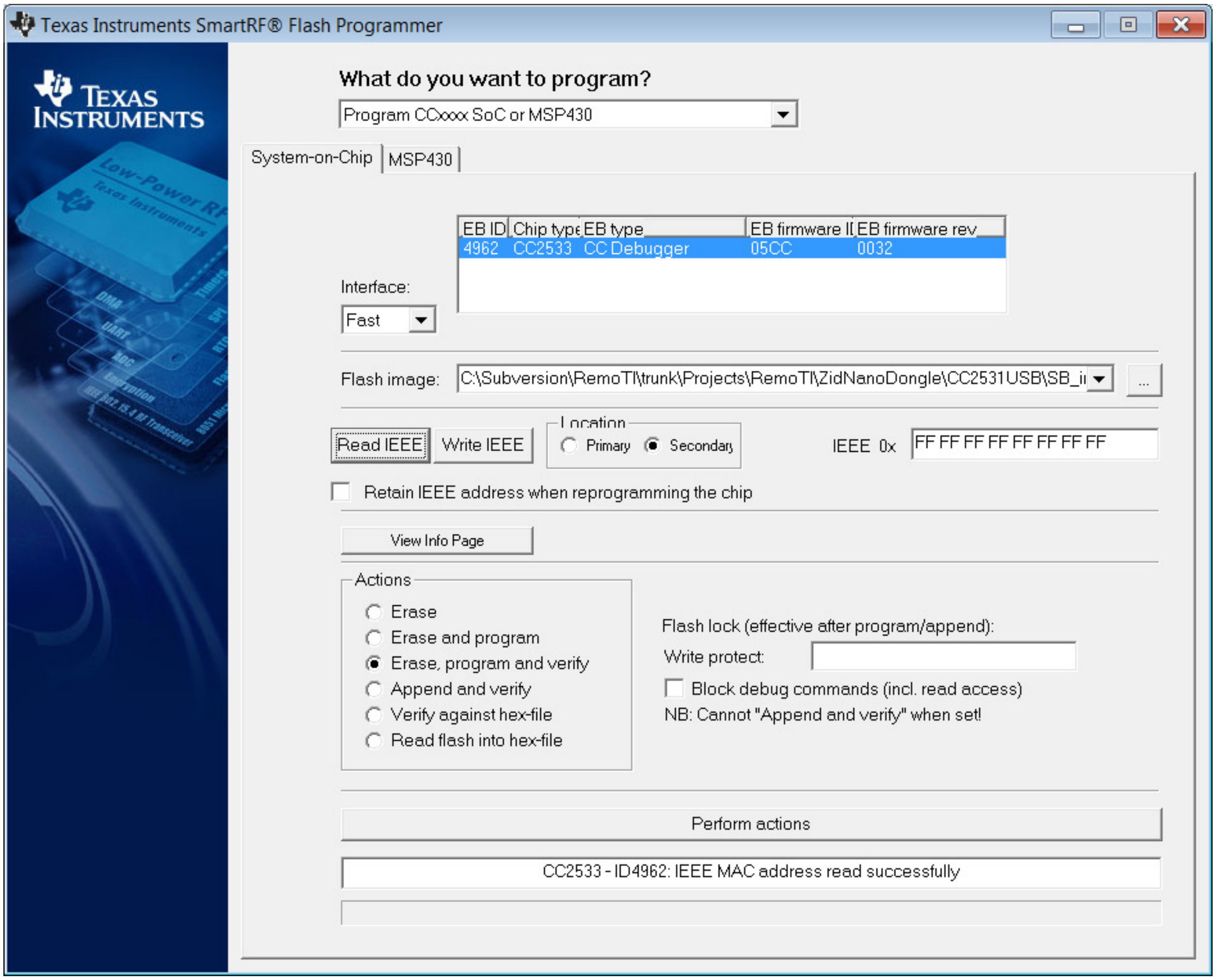

Figure 11 – SmartRF programmer

Hence, with RemoTI application framework, the hierarchy of IEEE address upon CC253x reset is as follows:

- If the commissioned IEEE address is valid, use the commissioned IEEE address
- Otherwise, use the information page IEEE address

Figure 12 illustrates the flow chart of selecting the network layer IEEE address, during startup of a device.

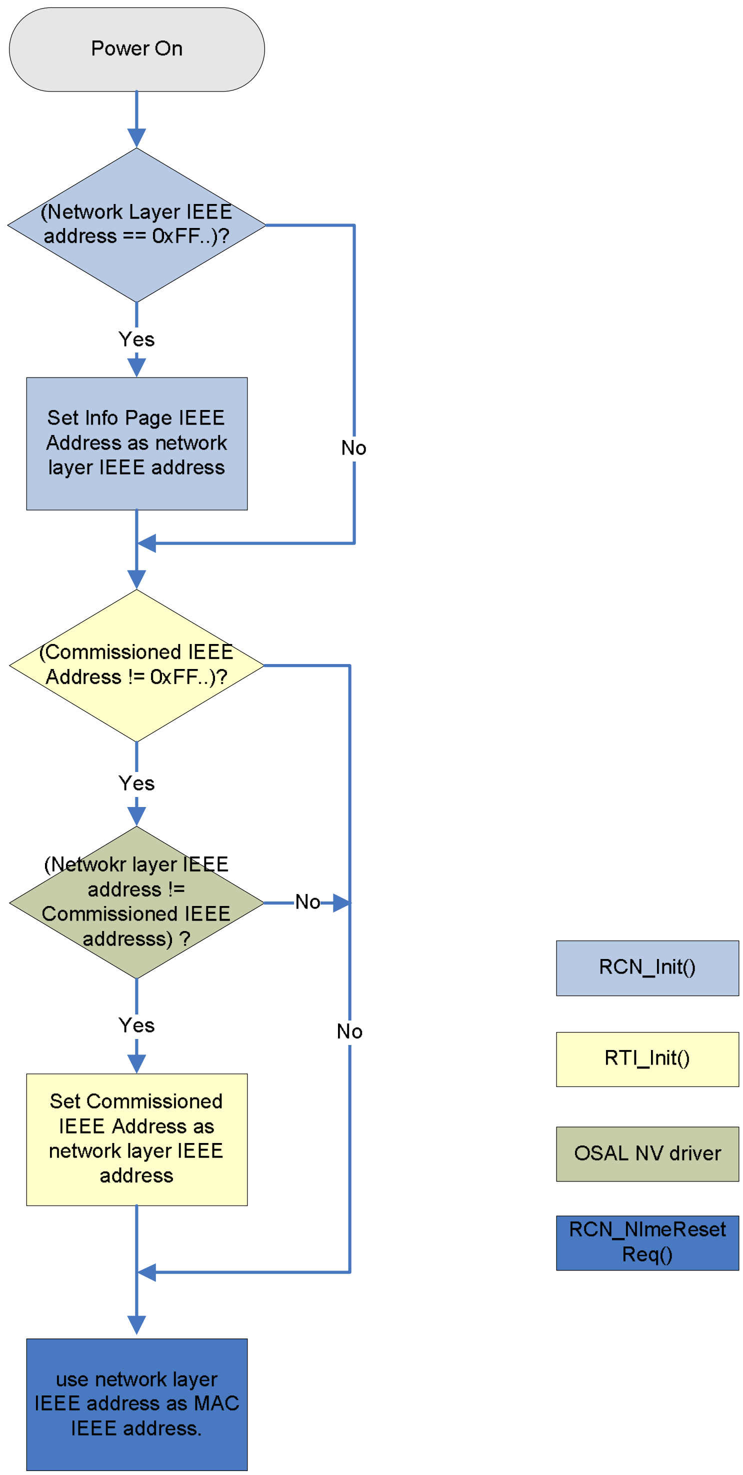

Figure 12 – IEEE address selection flow during startup

11 Network layer configuration

The standard NIB attributes can be configured and updated at run time through *RTI_WriteItemEx*() function or *RCN_NlmeSetReq*() function in case rti.c module is not used.

In rti.c module, *rtiResetSA*() function implementation shows example of *RCN_NlmeSetReq*() calls to set standard defined NIB attributes.

Network layer attributes that can be used with either *RTI_WriteItemEx* or *RCN_NlmeSetReq*() are enumerated in rcn_attribs.h file. Note that several non-standard attributes are also provided.

The following Table 7 explains non-standard attributes.

Table 7 – Network layer custom attributes

Attribute identifier	Description
RCN_NIB_NWK_NODE_CAPABILITIES	This attribute corresponds to standard constant nwkcNodeCapabilities. The value of this attribute should not change in product.
RCN_NIB_NWK_VENDOR_IDENTIFIER	This attribute corresponds to standard constant nwkcVendorIdentifier. The value of this attribute should not change in product.
RCN_NIB_NWK_VENDOR_STRING	This attribute corresponds to standard constant nwkcVendorString. The value of this attribute should not change in product.
RCN_NIB_STARTED	It is an attribute to indicate whether network layer has started ('1') or not ('0'). This attribute is useful for application to determine whether it has to perform cold boot procedure or warm boot procedure. RTI module (rti.c) uses this attribute to determine cold boot or warm boot procedure.
RCN_NIB_IEEE_ADDRESS	IEEE address attribute. By default, network layer will program IEEE address using chip IEEE addresss. Application can override chip IEEE address with this attribute. Note that RTI module (rti.c) writes into this attribute upon system reset. Application should consider conflict with RTI module when writing this attribute. See chapter 10.
RCN_NIB_AGILITY_ENABLE	Enable/disable frequency agility
RCN_NIB_TRANSMIT_POWER	Set transmission power level in dBm.

Note that other non-standard attributes such as RCN_NIB_PAN_ID and RCN_NIB_SHORT_ADDRESS are not configurable items. Those attribute values can be read for debug purpose.

Certain set of network layer implementation parameters can also be modified at build time by changing rcn_config.c file. The file is configured with default recommended values.

12 Serial Boot Loader

12.1 Overview of the serial boot loader demo

Serial boot loading is a feature that enables a RemoTI network processor device to download its embedded software image from a host processor through serial interface, such as UART and SPI. It is out of scope of this document how the host processor gets a software image for a particular network processor.

Serial boot loader demo consists of a network processor image which is built in serial boot loading enabled configuration, a serial boot loader programmed network processor device and serial boot loader demo PC tool. See [6] for build, setup and execution instructions.

Serial boot loader code resides at the bottom of the flash memory map as in Figure 5. Upon power cycle, serial boot loader decides whether to start serial boot loading or to jump to the downloaded image area. How the decision is made is implementation specific. In the UART serial boot loader demo code, serial boot loader makes a decision by validity of the downloaded image. If the image in the downloaded image area is not a valid image, the serial boot loader starts in serial boot loading mode and waits for commands from host processor. If the image in the downloaded image area is valid, the boot loader jumps to the valid image area. A network processor application which supports this UART serial boot loader mechanism has to clear the image preamble area and triggers a watchdog reset, as a result of processing a command from host processor to trigger serial boot loading.

Once in serial boot loading mode, the serial boot loader receives commands from host processor and executes them. Host processor is the intelligent part of the protocol. Host processor chooses image sector to download, reads back downloaded image area sector to verify the written image and authorize the use of the image, etc.

In the demo, host processor is emulated by a PC demo tool.

12.2 Serial boot loading commands

Serial boot loading command packets follow the same format as regular network processor interface commands. However, they are not exactly the same as serial boot loading commands are accepted only by the serial boot loader in serial boot loading mode and underlying transport mechanism could be different from the one used by network processor image. For instance, serial boot loader might be running 9600bps baud rate while network processor interface could be running 115200bps baud rate.

The serial boot loading command is always triggered by host processor first and then the serial boot loader of the network processor sends a respond command. Each command is described in the subsections.

12.2.1 Handshake Command

The Handshake is command ID 0x04. The handshake has no parameters. The handshake is sent by the host processor to determine if the boot loader is running on the network processor device.
Handshake Command

1 Byte	1 Byte	1 Byte	1 Byte
SOP	Len	Sys (13)	CMD (0x4)

The network processor boot loader responds with a 1 byte status code containing SB_SUCCESS.
Handshake Response

1 Byte	1 Byte	1 Byte	1 Byte	1 Bytes
SOP	Len	Sys (13)	CMD (0x84)	Status (0)

12.2.2 Write Command

The Write command is command ID 0x01. The write is sent by the host processor to write image portion to the flash on the network processor device. The write command has the following parameters:
Write Command:

1 Byte	1 Byte	1 Byte	1 Byte	2 Bytes	64 Bytes
SOP	Len	Sys (13)	CMD (0x1)	Address	Data

The address contains a word aligned address of the image. The network processor boot loader must add the base address of the network processor program area to the address. The network processor boot loader responds to the write command with the status of the operation.
Write Response

1 Byte	1 Byte	1 Byte	1 Byte	1 Bytes
SOP	Len	Sys (13)	CMD (0x81)	Status

12.2.3 Read Command

The Read command is command ID 0x02. The read command is sent by the host processor to read from the flash on the network processor. The read command has the following parameters:
Read Command:

1 Byte	1 Byte	1 Byte	1 Byte	2 Bytes
SOP	Len	Sys (13)	CMD (0x2)	Address

The address contains a word aligned address of the image. The network processor boot loader must add the base address of the network processor program area to the address. The network processor responds to the read command with the status of the operation, the address, and the data.
Write Response

1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	2 Bytes	64 Bytes
SOP	Len	Sys (13)	CMD (0x82)	Status	Address	Data

12.2.4 Enable Command

The Enable command is command ID 0x03. The enable command is sent by the host processor to indicate the image on the network processor is valid. When the network processor boot loader has received the enable, it writes SB_ENABLED_VALUE to the enabled parameter of the preamble in the application image. The boot loader uses the enabled parameter of the preamble at startup to determine if a valid image is present in the application memory space.

Enable Command

1 Byte	1 Byte	1 Byte	1 Byte
SOP	Len	Sys (13)	CMD (0x3)

The network processor boot loader responds with a 1 byte status code containing SB_SUCCESS.

Enable Response

1 Byte	1 Byte	1 Byte	1 Byte	1 Bytes
SOP	Len	Sys (13)	CMD (0x83)	Status (0)

12.3 Boot loading sequences

Figure 13 and Figure 14 illustrate boot loading sequences and application image downloading sequences performed during boot loading.

Note that validity of the image is determined by checking preamble of the application image area, which is updated only as the last transaction of image download sequence. If anything goes wrong during download of the image, such as power failure, the preamble of the image area is not updated and serial boot loader stays in boot loading mode waiting for boot loading command from host processor.

When the image is valid, the boot loader jumps to the image area.

Note that host processor reads back application image before enabling the image. The idea is to validate written image without having to impose CRC checking in the serial boot loader. It was done so to minimize code size of the serial boot loader. On the other hand, the time taken for serial downloading would take longer than using CRC validation mechanism since the entire image has to be read back.

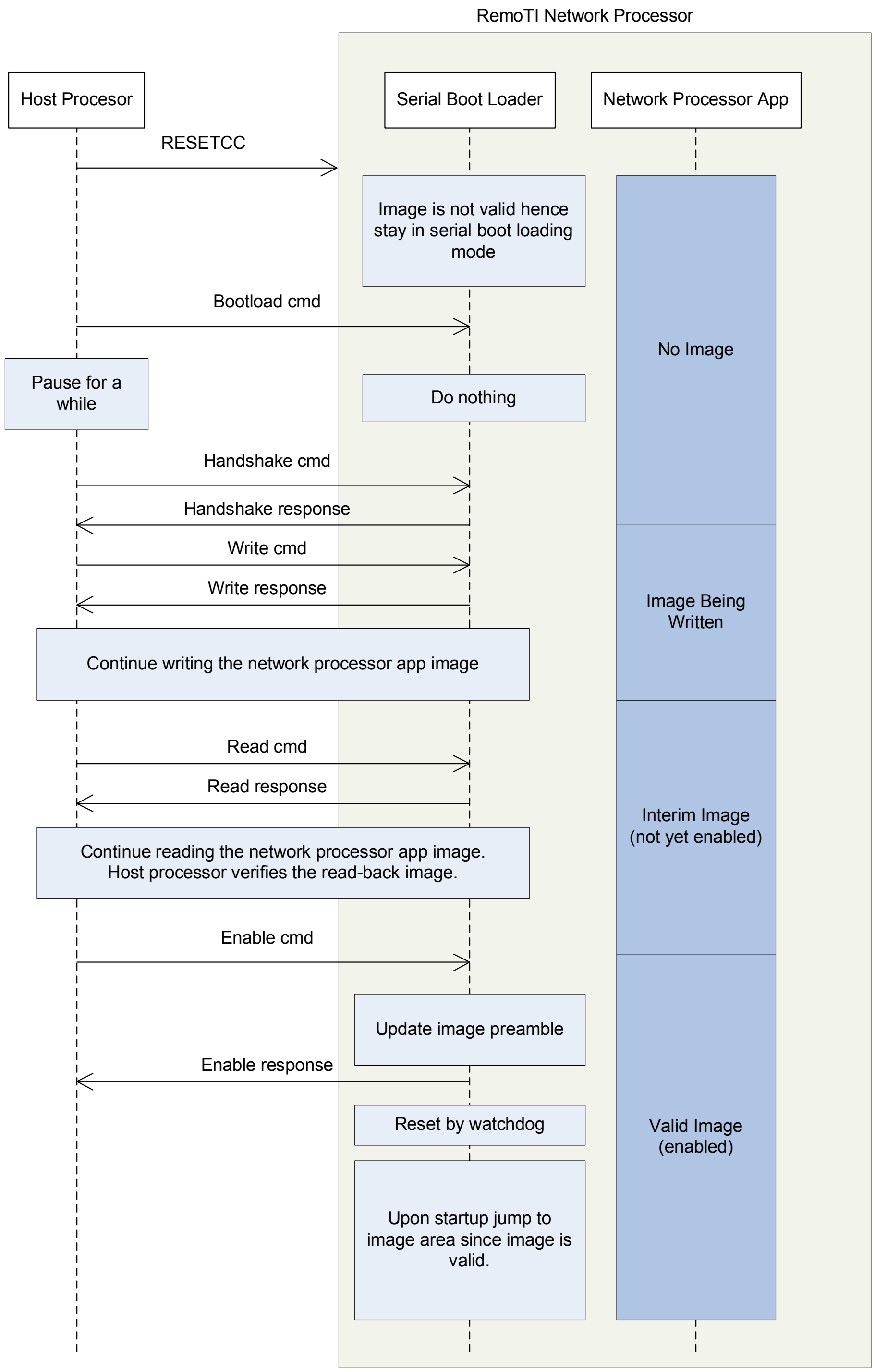

Figure 13 – Initial Application Image Download Sequence

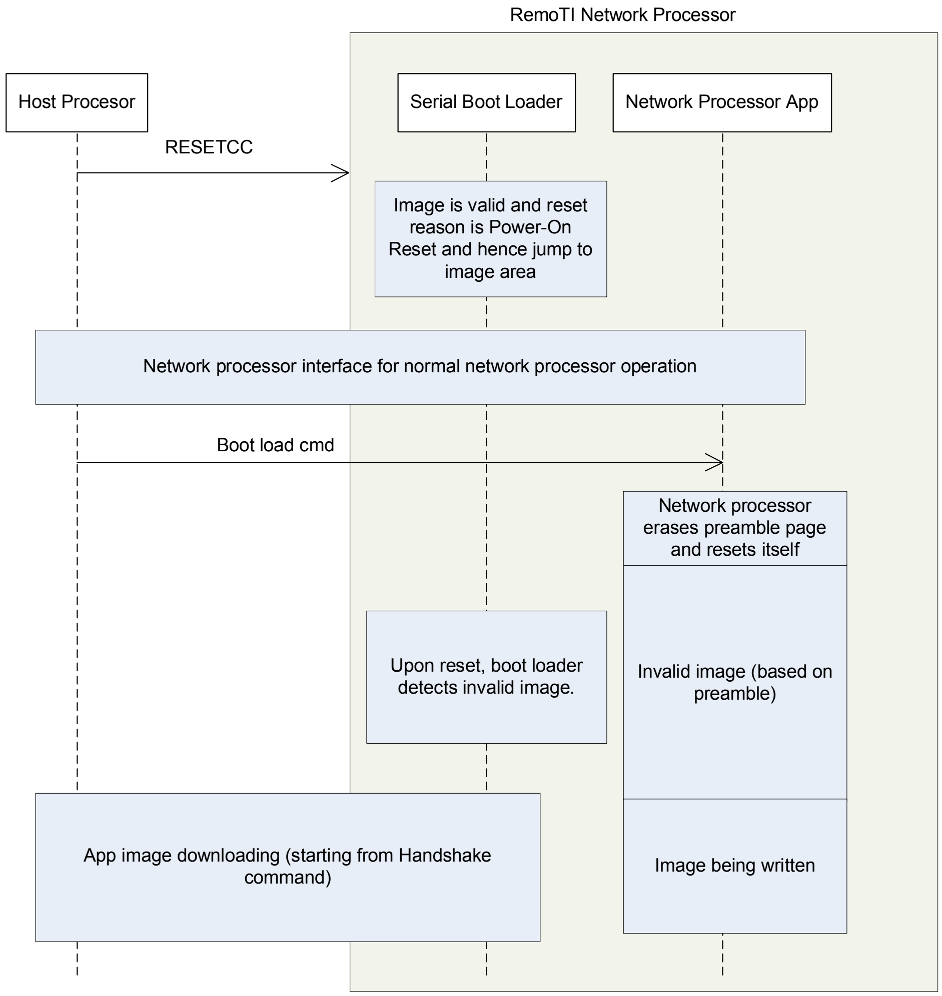

Figure 14 – Consequent Application Image Download Sequence

12.4 Network processor configuration for serial boot loading

RemoTI network processor includes serial boot loading demo feature. When the serial boot loading enabling configuration is in use, the sb_target.c module is compiled together and serial boot loading specific linker command file is selected. Serial boot loader configuration also adds *FEATURE_SERIAL_BOOT* compile flag definition, which enables handling of serial boot loader mode command which triggers software reset to serial boot loading mode from normal network processor operation.

12.5 Lock bit page

When building network processor image for serial boot loading, lock bit page cannot be used to store the downloaded code image as lock bit page itself cannot be overwritten during execution of code. Lock bit page can be updated only though debug interface. That is why the lock bit page usage is reserved in Figure 5.

13 DMA, peripheral IO and timers

RemoTI network processor uses the following resources:

- USART0 when configured for UART
- USART1 when configured for SPI
- Peripheral IO pins P0_2 and P0_3 when configured for UART
- Peripheral IO pins P0_3, P0_4, P1_4, P1_5, P1_6 and P1_7 when configured for SPI
- Peripheral IO pins P0_7, P1_1 and P1_4 when configured for CC2530-CC2591EM 2.0.
- DMA channel 0 for non-volatile memory access
- DMA channel 1 and 2 for AES encryption/decryption
- DMA channel 3 and 4 for UART or SPI
- Timer2 (MAC timer) and sleep timer
- USB controller for CC2531

Other peripheral IO might be set up by default (for instance, IO pin connected to LEDs on RemoTI Target Board platform) by HAL but they are not used by the network processor application and they are free to use by custom code.

14 RF frontend chip connection configuration

Transmit power and receiver gain of CC2530 can be increased by adding an RF frontend chip such as CC2591. The network processor project includes a configuration (CC2530F256+CC2591) for use of the RF frontend chip.

If the CC2591 is in use, the security feature of the RemoTI stack when used as a target node has to be disabled by setting RCN_NIB_NWK_NODE_CAPABILITIES attribute accordingly (See [2]). It is because a target node is required to transmit key seed command frames at maximum -15dBm but with use of CC2591, such low power transmission is not possible. The Target Emulator tool is not configured to disable security feature and hence cannot be used with a CC2530+CC2591 device while complying with the standard.

When the HAL_PA_LNA compile flag is defined, the network processor application is compiled to create a binary image for a CC2530-CC2591EM 2.0 board. The chip to chip connection is configured partially by the stack and partially by the MAC_RfFrontendSetup() function defined in the mac_rffrontend.c file. The PAEN pin and EN pin of CC2591 must be connected to P1_1 and P1_4 of CC2530 each just like it is done on the CC2530-CC2591 EM 2.0 board. MAC_RfFrontendSetup() function can be modified to customize HGM pin connection. On the CC2530-CC2591 EM 2.0 board, the pin is connected to P0_7 pin of CC2530 but in custom design it could be either grounded or connected to Vcc instead.

MAC_RfFrontendSetup() function not only configures the HGM pin connection but it also selects TX power register value table and RSSI value adjustment value table entry through a function call to MAC_SetRadioRegTable().MAC_SetRadioRegTable() function takes two arguments, TX power register value table index and RSSI adjustment value index. Note that the tables for CC2591 are included only in the rcnsuper-CC2530-banked.lib file.

rcnsuper-CC2530-banked.lib supports the following table indices.

Index Parameter	Table	Index
TX power register value table index (txPwrTblIdx)	CC2530 with no frontend	0
	CC2530 + CC2591	1
RSSI adjustment value index (rssiAdjIdx)	CC2530 with no frontend	0
	CC2530 + CC2591 in high gain mode	1
	CC2530 + CC2591 in low gain mode	2

Note that regardless of RF frontend selection, an application can set the transmit power level using the same Texas Instruments proprietary network layer attribute, RCN_NIB_TRANSMIT_POWER (See [2]).

15 General Information

15.1 Document History

Table 8 – Document History

Revision	Date	Description/Changes
1.0	2009-07-06	Initial release
swru223a	2009-09-18	New configurations for CC2531 dongle platform and CC2591 RF frontend were added. UART serial boot loading decision making algorithm was modified.
swru223b	2010-02-15	Section 3.2 added new configurations supported in RemoTI-1.2 release. Section 3.3 added new files in RemoTI-1.2 release
swru223c	2011-10-01	Updated for 1.3 release
Swru223d	2012-11-21	Updated for 1.3.1 release

16 Address Information

Texas Instruments Norway AS
Gaustadalléen 21
N-0349 Oslo
NORWAY
Tel: +47 22 95 85 44
Fax: +47 22 95 85 46
Web site: http://www.ti.com/lpw

17 TI Worldwide Technical Support

Internet

TI Semiconductor Product Information Center Home Page: support.ti.com

TI Semiconductor KnowledgeBase Home Page: support.ti.com/sc/knowledgebase

TI LPRF forum E2E community http://www.ti.com/lprf-forum

Product Information Centers

Americas

Phone: +1(972) 644-5580

Fax: +1(972) 927-6377

Internet/Email: support.ti.com/sc/pic/americas.htm

Europe, Middle East and Africa

Phone:

Belgium (English) +32 (0) 27 45 54 32

Finland (English) +358 (0) 9 25173948

France +33 (0) 1 30 70 11 64

Germany	+49 (0) 8161 80 33 11
Israel (English)	180 949 0107
Italy	800 79 11 37
Netherlands (English)	+31 (0) 546 87 95 45
Russia	+7 (0) 95 363 4824
Spain	+34 902 35 40 28
Sweden (English)	+46 (0) 8587 555 22
United Kingdom	+44 (0) 1604 66 33 99
Fax:	+49 (0) 8161 80 2045
Internet:	support.ti.com/sc/pic/euro.htm

Japan

Fax	International	+81-3-3344-5317
	Domestic	0120-81-0036
Internet/Email	International	support.ti.com/sc/pic/japan.htm
	Domestic	www.tij.co.jp/pic

Asia

Phone	International	+886-2-23786800
	Domestic	Toll-Free Number
	Australia	1-800-999-084
	China	800-820-8682
	Hong Kon	800-96-5941
	India	+91-80-51381665 (Toll)
	Indonesia	001-803-8861-1006
	Korea	080-551-2804
	Malaysia	1-800-80-3973
	New Zealand	0800-446-934
	Philippines	1-800-765-7404
	Singapore	800-886-1028
	Taiwan	0800-006800
	Thailand	001-800-886-0010
Fax		+886-2-2378-6808
Email		tiasia@ti.com or ti-china@ti.com
Internet		support.ti.com/sc/pic/asia.htm

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information 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.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265