RemoTI Basic Remote Developer's Guide

Document Number: SWRU224A

TABLE OF CONTENTS

1. REFERENCES 4

2. INTRODUCTION 5

2.1 PURPOSE 5
2.2 SCOPE 5

3. REMOTI BASIC REMOTE SAMPLE APPLICATION 5

3.1 FEATURES 5
3.2 BUILD CONFIGURATIONS 5
3.3 FILES 8
3.4 ARCHITECTURE 10

4. KEY MATRIX SCANNING 11

4.1 SCANNING SCHEME 11
4.2 CONFIGURATION 12

5. KEY CODE TO COMMAND MAPPING 13

6. LED CONTROL 15

7. OSAL TASK SYNCHRONIZATION 16

8. FLASH PAGE MAP AND MEMORY MAP 19

9. STACK AND HEAP 23

10. NETWORK LAYER INTERFACE 28

11. PAIRING 29

12. TARGET DEVICE SELECTION 30

13. NON-VOLATILE MEMORY 32

14. IEEE ADDRESS 32

15. IR SIGNAL GENERATION 35

16. NETWORK LAYER CONFIGURATION 39

17. OVER THE AIR DOWNLOAD 40

17.1 OVERVIEW OF THE OVER THE AIR DOWNLOAD DEMO 40
17.2 OVER THE AIR DOWNLOAD COMMAND PACKET FORMATS 41
17.2.1 Poll protocol frame format 41
17.2.2 General Over the Air Download protocol frame format 41
17.2.3 Over the Air Download protocol command identifiers 41
17.2.4 Status Request command 42
17.2.5 Start Request command 42
17.2.6 Data Request command 42
17.2.7 Enable Request command 43
17.2.8 Cancel Request command 43
17.2.9 Status Response command 43
17.2.10 Start Response command 43
17.2.11 Data Response command 44
17.2.12 Enable Response command 44
17.2.13 Cancel Response command 45
17.3 OVER THE AIR DOWNLOAD COMMAND FLOW SEQUENCE 45
17.4 BASIC REMOTE CONTROLLER SAMPLE APPLICATION CONFIGURATION 46
17.5 RETENTION OF NON-VOLATILE MEMORY DATA 47
17.6 USE OF IMAGE IDENTIFIER 48
17.7 LOCK BIT PAGE 48

18. LATENCY TEST MODE 48

19. DMA, PERIPHERAL IO AND TIMERS 50
20. RF FRONTEND CHIP CONNECTION 50
21. GENERAL INFORMATION 52
21.1 DOCUMENT HISTORY 52
22. ADDRESS INFORMATION 52
23. TI WORLDWIDE TECHNICAL SUPPORT 52

Acronyms and Definitions

ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
CERC	Consumer Electronics for Remote Control, a name for a Zigbee RF4CE profile
CCM	Counter with CBC-MAC (CCM), a mode of operation for cryptographic block ciphers
DMA	Direct Memory Access
HAL	Hardware Abstraction Layer
IAR	IAR Systems, a software development tool vendor
IDATA	Internal Data memory
IEEE	Institute of Electrical & Electronics Engineers, Inc.
IO	Input Output
IR	Infra-red
LED	Light Emitting Diode
NIB	Network Information Base
NSDU	Network layer Service Data Unit
NV	Non-Volatile, or Non-volatile memory
NWK	Network
OAD	Over the Air Download
OSAL	Operating System Abstraction Layer
PAN	Personal Area Network
PER	Packet Error Rate
SRAM	Static Random Access Memory
TI	Texas Instruments Incorporated
XDATA	eXternal Data memory
Zigbee RF4CE	An 802.15.4 based remote control protocol standard

1. References

[1] RemoTI Developer’s Guide, SWRU198
[2] RemoTI API, SWRA268
[3] HAL Drivers API, SWRA193
[4] OSAL API, SWRA194
[5] CC253X System-on-Chip Solution for 2.4-GHz IEEE 802.15.4/ZigBee/RF4CE User’s Guide, SWRU191
[6] RemoTI Sample Applications User’s Guide, SWRU201

2. Introduction

2.1 Purpose

This document explains the basic remote controller sample application and focuses on areas of potential customization.

2.2 Scope

This document describes concepts and settings for the Texas Instruments RemoTI software release with respect to basic remote application development. The general concept of Zigbee RF4CE and RemoTI architecture is described in [1].

3. RemoTI Basic Remote Sample Application

The RemoTI development kit includes the basic remote sample application. This chapter describes the features of the sample application and the organization of the sample source code and project files.

3.1 Features

The following summarize the features of RemoTI Basic Remote Sample Application:

- Compliance with Zigbee RF4CE CERC profile, including push button pairing and CERC user command generation
- Compliance with Zigbee RF4CE network layer specification, playing the controller node role
- Zigbee RF4CE network layer security
- 7 x 8 key matrix scanning
- Four target device selection keys
- Maximum ten pairing entries
- LED feedback for pairing and CERC command transmission
- Latency and PER test mode
- Optional over the air downloading demo
- Optional IR signal generation demo with subset of keys

3.2 Build configurations

The RemoTI Basic Remote Sample application project is located in the Projects\RemoTI\BasicRemote\CC2530RC folder.
When you open the workspace file (rsa_cc2530.eww), you can select different project configurations as in Figure 1.

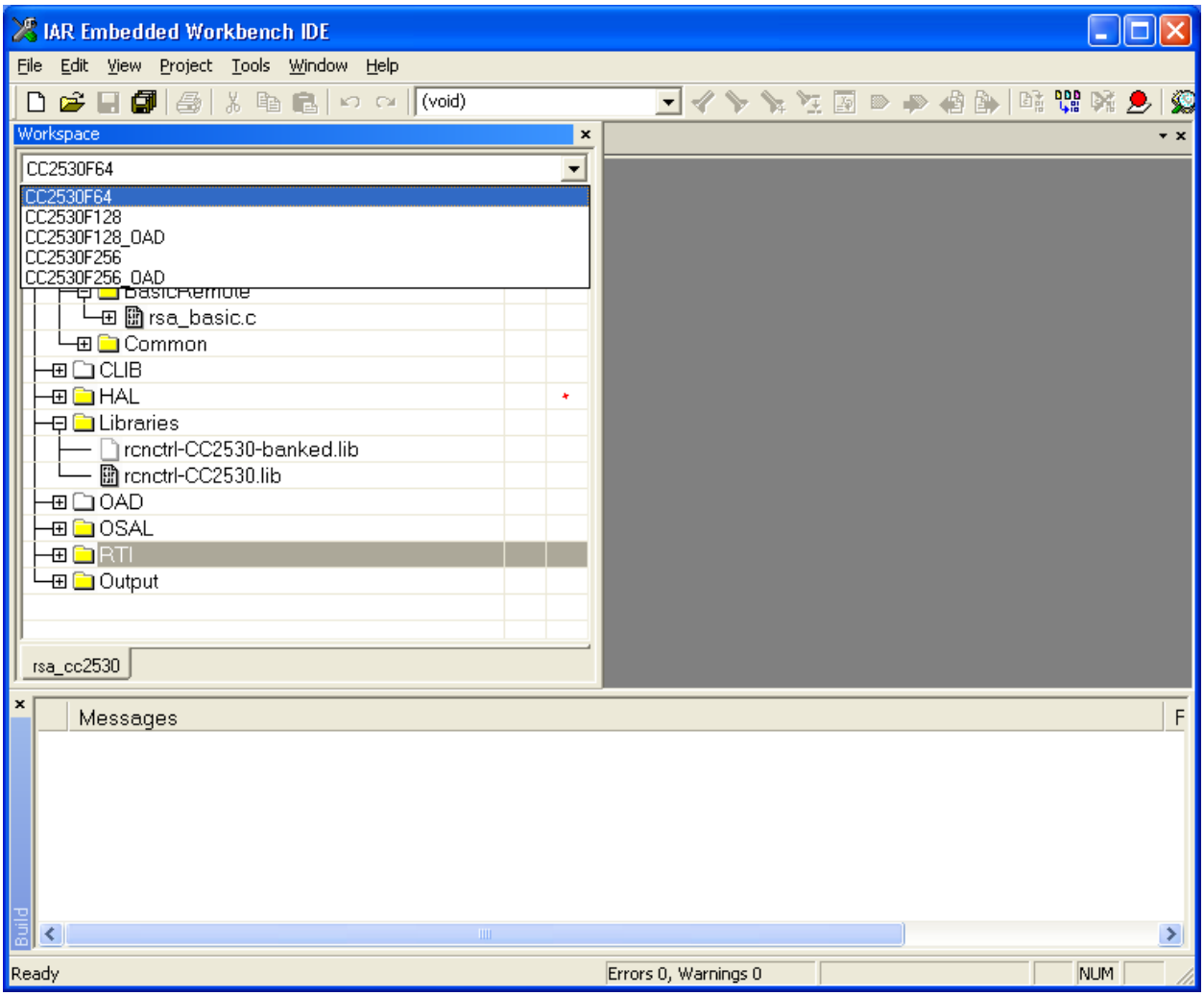

Figure 1 – Project configuration selection from IAR

Each configuration is explained in Table 1.

Table 1 – Project configurations

Configuration	Description
CC2530F64	Configuration for CC2530F64 part.
CC2530F128	Configuration for CC2530F128 part
CC2530F128_OAD	Configuration for CC2530F128 part including over the air downloading feature
CC2530F256	Configuration for CC2530F256 part
CC2530F256_OAD	Configuration for CC2530F256 part including over the air downloading feature

Note that IR signal generation feature is not included in any of the above configurations. Details about IR signal generation demo is described in chapter 15. The project option settings for each configuration, such as defined symbols (also known as compile flags) for 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.
CC2530F64	Non-volatile memory configuration selection for CC2530F64
CC2530F128	Non-volatile memory configuration selection for CC2530F128
CC2530F256OAD	Non-volatile memory configuration selection for CC2530F256 when over the air downloading is enabled. Note that without CC2530F64, CC2530F128 or CC2530F256OAD, default non-volatile memory configuration is set for CC2530F256 without over the air downloading feature. A notable difference between the default non-volatile memory configuration (i.e. configuration for CC2530F256) and the one set by CC2530F256OAD compile flag is the size of non-volatile memory pages.
OAD_IMAGE_ID=*value*	Image identifier value for application image used by over-the-air download can be set with this compile flag value. For example, OAD_IMAGE_ID=0x25300002 will set image identifier value to be 0x25300002.
OAD_KEEP_NV_PAGES	This compile flag, when defined, suppresses adding non-volatile memory pages (OSAL NV module) into linker code. When FEATURE_OAD is defined, this compile flag also has to be defined if OSAL NV is used.
FEATURE_CONTROLLER_ONLY	This compile flag, when defined, reduces RTI code size when RTI is compiled for remote controller functionality only.
FEATURE_OAD	This compile flag, when defined, enables over the air

Compile Flag	Description
	download feature in basic remote controller application code. Note that defining this compile flag alone does not enable over the air download for the basic remote controller sample application. Choose the proper configuration (with OAD tag) in the provided project to enable over the air download feature. Such configurations include this compile flag definition in the project settings.
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.

Besides the compile flags, other settings such as code model were also set to fit the configuration. For instance, CC2530F64 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 that appears in the IAR workspace window. Note that there are more files than those listed in the table, such as C header files that define constants and function prototypes and also note that workspace project itself does not list all header files referenced by the C files.

Table 3 – Project files

File name	Description
Application	
rsa_basic.c	Key basic remote application code implemented on top of RemoTI application framework
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.
rsa_main.c	C main routine implementation which calls all necessary initialization and then call OSAL
rsa_osal.c	OSAL task definition and initialization
CLIB	
chipcon_cstartup.s51	Assembly routines to override default C libraries for banked code

File name	Description
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_board_cfg.h	RemoTI basic remote hardware specific configuration parameters and macros used by HAL. Application also frequently uses board definition literal (HAL_BOARD_CC2530RC) 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_flash.c	Flash device driver
hal_irgen.c	IR signal generation driver
hal_key.c	Key matrix driver
hal_led.c	LED driver
hal_sleep.c	Sleep mode (PM1, PM2, PM3) control implementation
Libraries	
rcnctrl-CC2530-banked.lib	RemoTI network layer library built for banked code model. This library will be selected for F128 and F256 configurations. Note that this library is optimized for controller nodes only.
rcnctrl-CC2530.lib	RemoTI network layer library built for near code model. This library will be selected for F64 configuration. Note that this library is optimized for controller nodes only.
OAD	

File name	Description
oad_appflash.c	Implementation of flash driver abstraction for demo over the air download engine. This implementation is specifically intended for application image which has hal_flash.c as flash device driver.
oad_client.c	Demo over the air download engine
oad_crc.c	CRC calculation implementation use by the over the air download engine
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_Nv.c	OSAL non-volatile memory manager
OSAL_PwrMgr.c	OSAL power management scheme implementation
OSAL_Timers.c	OSAL timer implementation
RTI	
rti.c	RemoTI application framework implementation
rti_testmode.c	RemoTI test mode API function implementation

3.4 Architecture

Basic remote sample application software uses services of HAL, OSAL and RemoTI application framework. RemoTI application framework abstracts RemoTI network layer stack software by providing simplified API. RemoTI application framework is provided as source code and is itself part of application from RF4CE network layer perspective. HAL, OSAL and RemoTI stack interacts with one another. Figure 2 illustrates an architectural view.

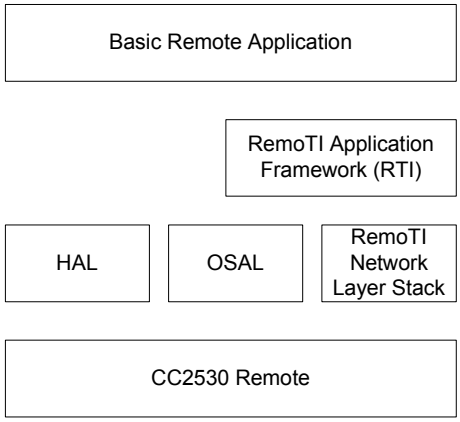

Figure 2 – Basic Remote Software Architecture

4. Key matrix scanning

Key matrix scanning is implemented in hal_key.c file. Note that the Components\hal\target\CC2530RC\hal_key.c file is specific to the CC2530 basic remote reference platform and do not confuse this file with other hal_key.c implementations such as Components\hal\target\CC2530EB_NPI\hal_key.c file.

4.1 Scanning scheme

Key matrix is attached to peripheral IO pins in CC2530 basic remote reference platform, which looks like Figure 3.

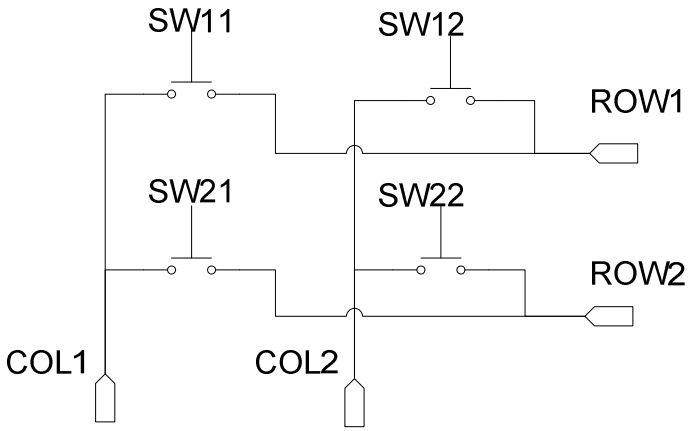

Figure 3 – Key switch matrix circuitry

In the basic remote reference platform, a single IO port is dedicated for either a row or a column of the key switch matrix. Initially, all row IO port pins are configured as pull-up input pins with falling edge interrupt setup, and all column port pins are configured to output low so that any key press triggers an interrupt with the IO port associated with row pins.

Once the interrupt is triggered, the de-bounce timer is triggered to ignore de-bouncing. On expiry of the de-bounce timer, key scanning is performed periodically till no key press is detected. Periodical key scanning is used so that a continuous key depress generates repeated key press events. Figure 4 illustrates the flow of key scanning state changes.

Key scanning itself is performed by asserting individual column IO port pins (active low) one by one and reading row IO pin values. For instance, in Figure 3, when SW12 is pressed, ROW1 input pin reads low when COL2 is set to low. At the end of key scanning, a configured callback function is called to pass the scanned key code.

Key scan state flows and scanning itself are implemented in hal_key.c module specific to CC2530 basic remote (CC2530RC) platform.

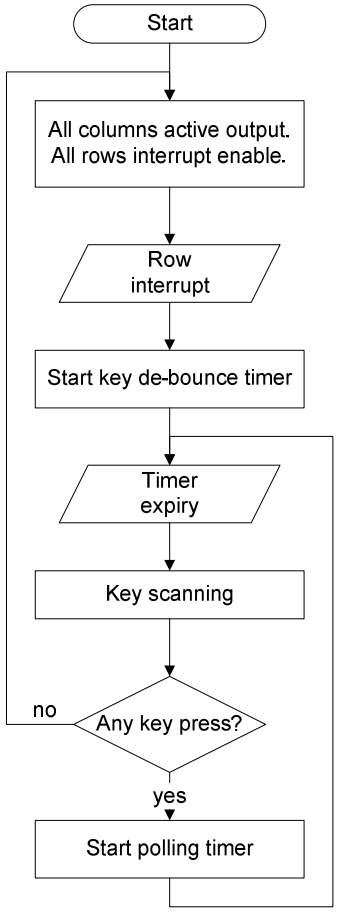

Figure 4 – Key scanning state flow diagram

4.2 Configuration

In the beginning of the hal_key.c module, certain constants are defined with arbitrary choice. Table 4 lists configurable constant values.

Table 4 – HAL key module configurable constants

Constant Name	Default Value	Description
HAL_KEY_DEBOUNCE_VALUE	25	Key de-bounce timer duration in milliseconds. Default value is set to 25 milliseconds. The value should be adjusted to fit the hardware.
HAL_KEY_POLLING_VALUE	100	This constant has no effect. Note that this value does not determine polling timer value used in interrupt driven key scanning scheme as explained in this chapter. Polling timer value for interrupt driven key scanning scheme is hard coded to 50 milliseconds to conform to CERC profile specification. To change the hard-coded timer value, change the third argument of osal_start_timerEx(Hal_TaskID, HAL_KEY_EVENT, 50) call in HalKeyPoll() function.
HAL_KEY_COL_BITS	0xFD	Bitmap of column IO port pins in use. '*0b11111101*' as default specifies that pin 1 is not used for key matrix.
HAL_KEY_ROW_BITS	0xFF	Bitmap of row IO port pins in use. '*0b11111111*' as default specifies that pin 0 to pin 7 all of them are in use.
HAL_KEY_ROW_PULLDOWN	FALSE	Whether row IO port mode should be configured as pull down mode. Setting of pull up or pull down mode depends on the hardware platform design.

Note that there are other constants defined in hal_key.c module. However, other configuration constants such as using port 1 for column and port 0 for row cannot be modified without potentially changing other parts of the codes or at least testing and verifying the change.

5. Key code to command mapping

The key scan code returned from key scanning is mapped to an appropriate CERC command in rsa_basic.c module. The key scan code is passed to this module via RSA_KeyCback() function. The callback function is configured in the main() routine in the rsa_main.c module.

The key callback function has two arguments: *keys* for key scan code and *state* for shift state. The hal_key.c module currently does not implement any shift state and hence *state* argument is not used.

The RSA_KeyCback() function does more than mapping a key code to a CERC command and generating a CERC command packet. It handles key presses differently for test mode, select targets device upon certain special keys, triggers pairing upon pair key press and optionally could generate IR signals as well.

The description of such special keys can be found in [6] and the source code is self explanatory. This chapter focuses on how to designate a key for a CERC command or a special function implemented in the rsa_basic.c module.

The key code map to command is defined in the rsaKeyMap array in the rsa_basic.c file. The array index is the key code. The key code definition depends on HAL key module implementation of a particular platform. For CC2530 remote, hal_key.c module generates the key code in the format specified in 5.

Table 5 – Key code format

Bit 0-2	Bit 3-5	Bit 6-7
Column number	Row number	Reserved

For example, index 0 (0b00 000 000) entry is a map for key row 0, column 0. Index 11 (0b00 0001 011) entry is a map for key row 1, column 3.

The array contains values that identify either CERC user control pressed command or a special command. Table 6 shows the value range.

Table 6 – Key map values

Value Range	Description
RTI_CERC_SELECT (0x00) – RTI_CERC_DATA (0x76)	RC command code of CERC user control pressed command frame. Note that only single byte commands (i.e. no RC payload) are supported for the values within this range.
0x77 – 0x7F	Reserved
RSA_MEDIA_SEL (0x80) – 0xAF	CERC user control pressed command for Select Media Function. RC command payload shall be set to the value - 0x80. For instance, value 0x92 indicates Select Media Function with payload value (0x92-0x80) = 0x12.
RSA_ACT_PAIR (0xB0)	Trigger pairing
0xB1	Reserved
RSA_ACT_TOGGLE_TGT (0xB2)	Target device toggle

Value Range	Description
RSA_ACT_TEST_MODE (0xB3)	Test mode toggle
RSA_ACT_POLL (0xB4)	Poll server (this map is valid only when over the air downloading feature is in use).
0xB5 – 0xBF	Reserved
0xC0 – 0xFE	Target device type select (device type = value - 0xBE, for instance 0xC0 corresponds to 0xC0-0xBE = 0x02, Television device type)
RTI_CERC_RESERVED_1 (0xFF)	No action

6. LED control

A CC2530 reference remote has two set of LEDs that can be controlled independently. One set of LEDs are called activity LEDs and the other set is called backlight LEDs. In hal_board_cfg.h file, the activity LEDs are mapped to LED2 while the backlight LEDs are mapped to LED1 of HAL LED module.

Table 7 shows configurable constants in hal_board_cfg.h that were set to fit the CC2530 basic remote.

Table 7 – LED configuration constants

Constant Name	Value	Description
LED1_BV	BV(0)	Bitmap corresponding to the port pin for LED1, used for direction register settings, etc.
LED1_SBIT	P2_0	Port register (single bit access) to control LED1
LED1_DDR	P2DIR	Port direction register for LED1
LED1_POLARITY	ACTIVE_LOW	Polarity of LED1
LED2_BV	BV(1)	Bitmap corresponding to the port pin for LED2, used for direction register settings, etc.
LED2_SBIT	P1_1	Port register (single bit access) to control LED2
LED2_DDR	P1DIR	Port direction register for LED2
LED2_POLARITY	ACTIVE_LOW	Polarity of LED2

The basic remote controller sample application uses activity LEDs to indicate pairing in progress. That is, when remote is performing pairing procedure, activity LED will be turned on.

Background LEDs will be turned on while transmitting CERC user control packets. Background LEDs are also used for various feedback in the test mode (see [6]). The code that controls LEDs reside in the rsa_basic.c module and the LED control is done through HAL LED API functions. Refer to HAL Drivers API (SWRA193) document for details on API.

Note that the activity LED control port pin is also used for IR diode connection for the CC2530 remote hardware platform. Hence, when IR demo code is enabled, activity LEDs control code is commented out.

7. OSAL task synchronization

TI OSAL is a very thin software framework where tasks can be defined with their own entry point and messages and events are exchanged between tasks. OSAL tasks do not employ any real context switching. Rather, all tasks defined in OSAL run in the same thread context as opposed to interrupt thread context.

See [4] for OSAL features and interfaces.

The *main*() routine in rsa_main.c file as quoted below illustrates typical startup sequence of the runtime system thread using OSAL.

```
int main(void)
{
  /* Initialize hardware */
  HAL_BOARD_INIT();

  /* Initialze the HAL driver */
  HalDriverInit();

  /* Initialize NV system */
  osal_nv_init(NULL);

  /* Initialize MAC */
  MAC_InitRf4ce();

  /* Initialize the operating system */
  osal_init_system();

  /* Enable interrupts */
  HAL_ENABLE_INTERRUPTS();

  /* Setup Keyboard callback */
  HalKeyConfig(RSA_KEY_INT_ENABLED, RSA_KeyCback);

  /* Start OSAL */
  osal_start_system(); // No Return from here

  return 0;
}
```


osal_start_system() routine loops forever updating system timer tick and checking signaled events and queued messages invoking each user tasks when necessary as illustrated in Figure 5.

Note that task priorities do not work as preemptive context switching as all tasks are running in the same context. However the priorities affect the order of event handling. Each task has a 16 bit event flag variable and OSAL checks the event flags from the highest priority task down to the lowest priority task and calls the task entry function when an event is signaled to the task. Upon completion of the entry function, OSAL checks the event flags from the highest priority task again. Hence, if a new event is signaled to a task with higher priority than another task which already had a signaled event, during any execution (either of an OSAL task or an interrupt service routine), this new event is handled before the already signaled event. The task priorities are determined by the order of task initialization function calls from within *osalInitTasks*() function implementation and the order of task entry functions in *tasksArr* array variable. See [4] for details.

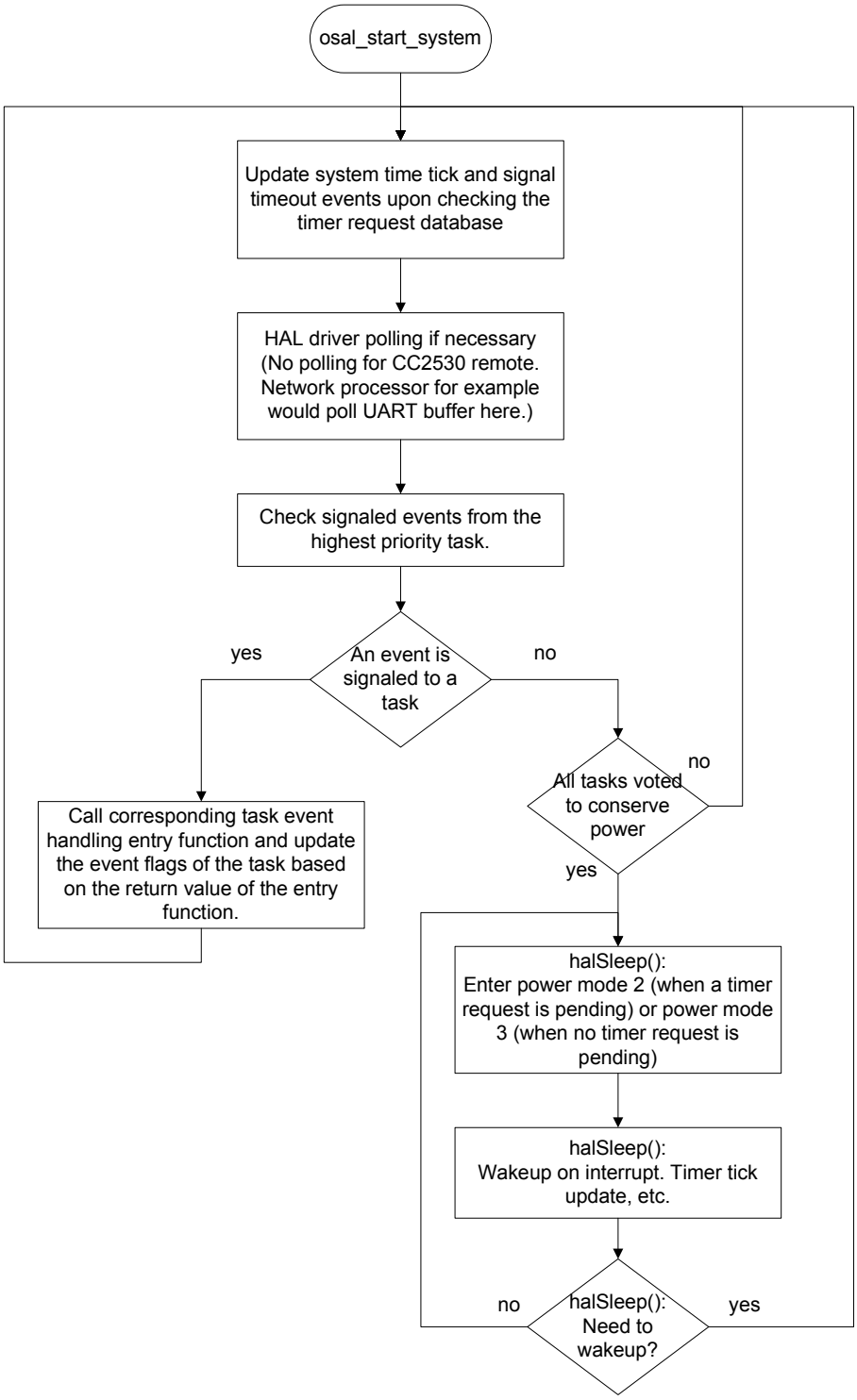

Figure 5 – OSAL loop: User thread flow

Since all tasks are running in the same thread context, there is no need for inter-task synchronization. However, OSAL provides task synchronization API functions in order to synchronize between interrupt service routines and the OSAL tasks. Such API functions can also be used to delay certain initialization routines till after all other initialization such as peripheral driver initialization is complete and OSAL loop, i.e., *osal_start_system*() is entered.

Two events are used in the rsa_basic.c module in all configurations besides the other two events used only by over the air downloading. Over the air downloading is explained in chapter 17. The two events used for all configurations are RSA_EVT_INIT and RSA_EVT_RANDOM_BACKOFF_TIMER.

An RSA_EVT_INIT event is triggered from within the RSA_Init() function. The RSA_Init() function is called when OSAL initializes all its tasks in the osalInitTasks() function of the rsa_osal.c module. The event is handled in RSA_ProcessEvent() function and performs certain initialization. The reason some initialization is performed in the event handler in this way instead of all done inside the RSA_Init() function is that certain initialization has to take place after all drivers and tasks are initialized. Triggering an event upon the task initialization routine (RSA_Init) and performing some initialization upon event handling ensures that the latter initialization happens after OSAL completes all driver and task initialization.

The other event RSA_EVT_RANDOM_BACKOFF_TIMER is a timer event also triggered from the application task itself (from within rsaRRTSendData() function). This event is used when the remote is in the test mode.

Communication among application task, RemoTI application framework (RTI) task and network layer is through direct function call and callback functions. Such function calls are made from within user thread context and hence no complex synchronization need be considered.

HAL key module callback function, RSA_KeyCback() function is also called from user thread context and not from interrupt thread context.

In summary, OSAL events are used in basic remote controller sample application in order to postpone certain actions till after OSAL event handler loop is running or for timer events. OSAL events are typically used for such purposes. OSAL messages are used more frequently between tasks because the messaging intuitively involves transfer of ownership of certain data structure. Basic remote controller sample application itself does not use OSAL messages but direct function calls between the application task, the application framework and the network layer.

8. Flash page map and memory map

Each configuration of basic remote controller sample application project has a unique flash page map. Figure 6 illustrates two distinctive flash page maps used by basic remote sample application. One flash page is 2048 bytes as specified in [5].

For over the air downloading feature enabled configuration, the boot loader code occupies the bottom page and the top page is reserved just for the lock bits and certain information such as commissioned IEEE address. The other code space is split into active code space and downloaded code space. The details of the over-the-air downloading feature enabled configuration are explained in chapter 17.

Without the over-the-air downloading feature, the code starts from 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) and commissioned IEEE address (8 bytes, prior to lock bits). IEEE address is explained more in chapter 14. The remainder of this last flash page can be used for additional code if the code fills up the reset of the space.

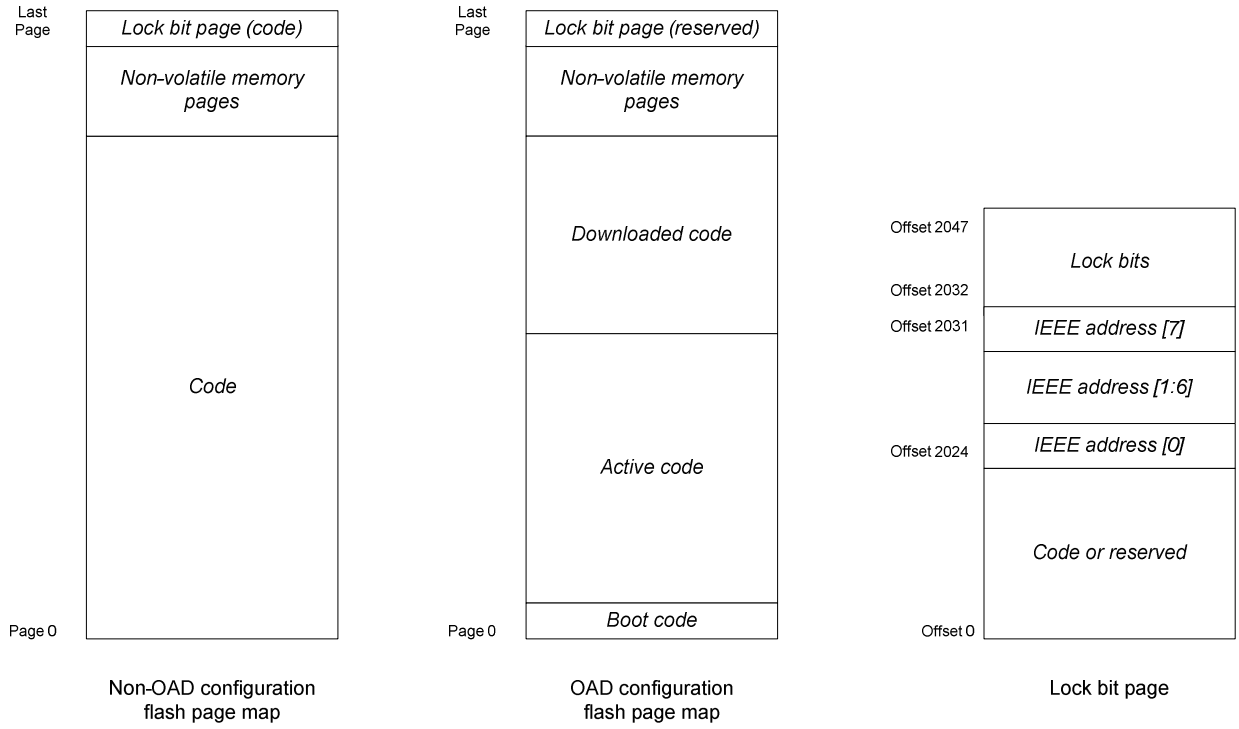

Figure 6 – Flash page map

Number of pages used for OSAL non-volatile memory system is defined in hal_board_cfg.h file. The configurable constants and their values are listed Table 8.

Table 8 – NV configuration constants

Constant Name	Description	CC2530F64 value	CC2530F128	CC2530F256
HAL_NV_PAGE_END	Last OSAL NV page plus one	31	63	127
HAL_NV_PAGE_CNT	Number of OSAL NV pages	2	2	6

In order to change the number of pages used for the non-volatile memory system, both hal_board_cfg.h file and linker command file have to be updated. In hal_board_cfg.h file, change the HAL_NV_PAGE_CNT definition. For instance, if you wish to use 4 flash pages and OSAL NV pages for CC2530F128 part, change hal_board_cfg.h file as follows:

```
...
#elif defined CC2530F128
#define HAL_FLASH_LOCK_BITS      16
#define HAL_NV_PAGE_END          63
#define HAL_NV_PAGE_CNT          4
...
```

The linker command file can be located from project option pop up window. For instance after selecting CC2530F128 configuration, select Project -> Options menu. In project option pop up window, select linker category and Config tab. Linker command file name and path is displayed as Figure 7.

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 CC2530F128 configuration has the following lines:

```
...
-D_ZIGNV_ADDRESS_SPACE_START=0x3E800
...
```

If you want four pages for non-volatile memory instead, the non-volatile memory page should be located at 12th page of the last bank (16 - 1 - 3), and the address should be 0x38000 + (0x800 * (12 - 1)) = 0x3D800. See further below in this section, for flash pages per bank and address ranges. The linker command file in this case has to be updated as follows:

```
...
-D_ZIGNV_ADDRESS_SPACE_START=0x3D800
...
```

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

CC2530F64 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.

CC2530F128 configuration and CC2530F256 configuration 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 9.

Table 9 – Code bank virtual address

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.

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 7.

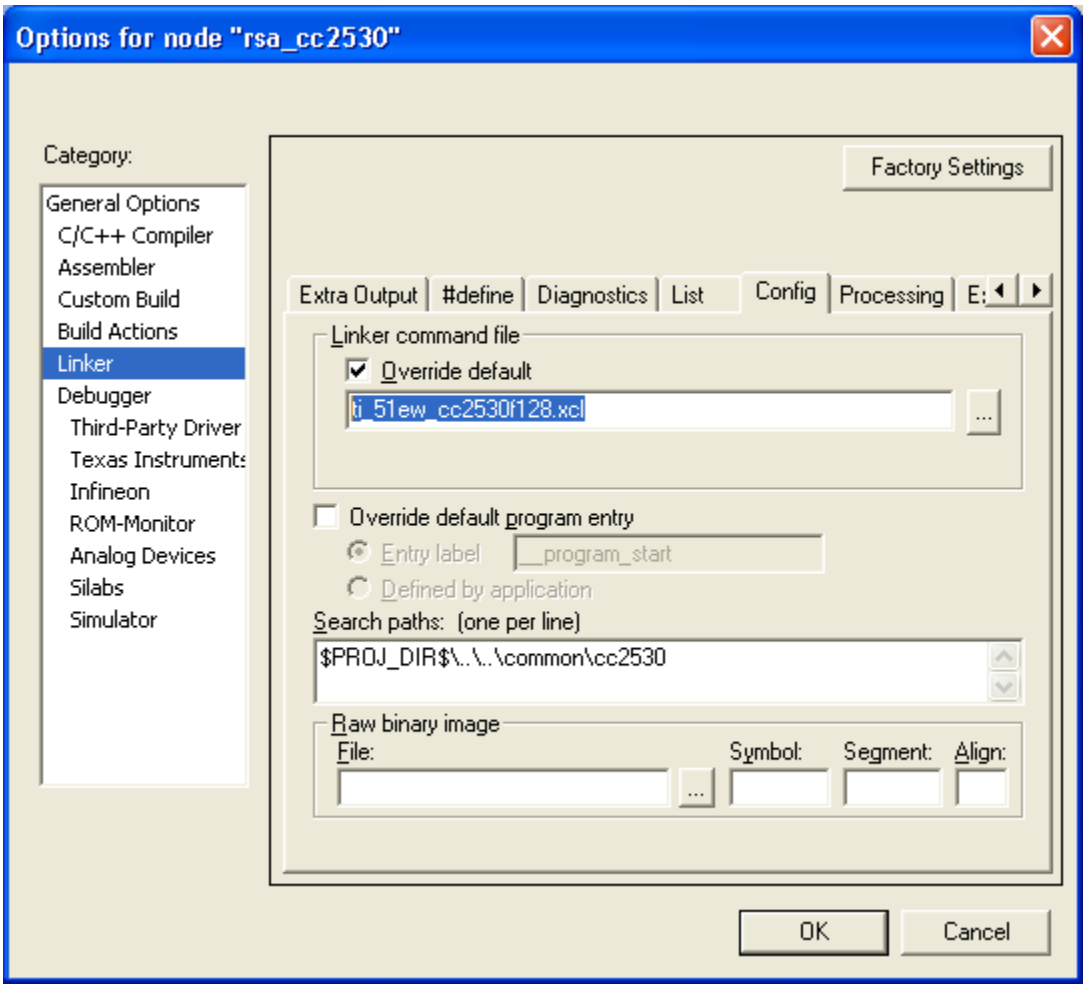

Figure 7 – Basic remote linker configuration option window

9. Stack and Heap

The 8051 micro-controller uses a variety of data memory access methods, among them, most distinctively, one for internal data memory with 8 bit address space (IDATA) and for external data memory with 16 bit address space (XDATA). CC2530 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 the compiled code uses IDATA and XDATA for stack is highly dependent on the compiler itself.

With IAR 8051 compiler version 7.51A, RemoTI CC2530 development kit 1.0 basic remote sample application uses about 207 bytes of XDATA stack and 53 bytes of IDATA stack. However, such stack usage could change with even slight modification of 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 the RemoTI CC2530 development kit 1.0 basic remote. The stack size settings can be adjusted after profiling the stack usage with the final application code, by browsing stack memory space through a 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 basic remote CC2530F64 build, as can be found from a generated map file as in Figure 8.

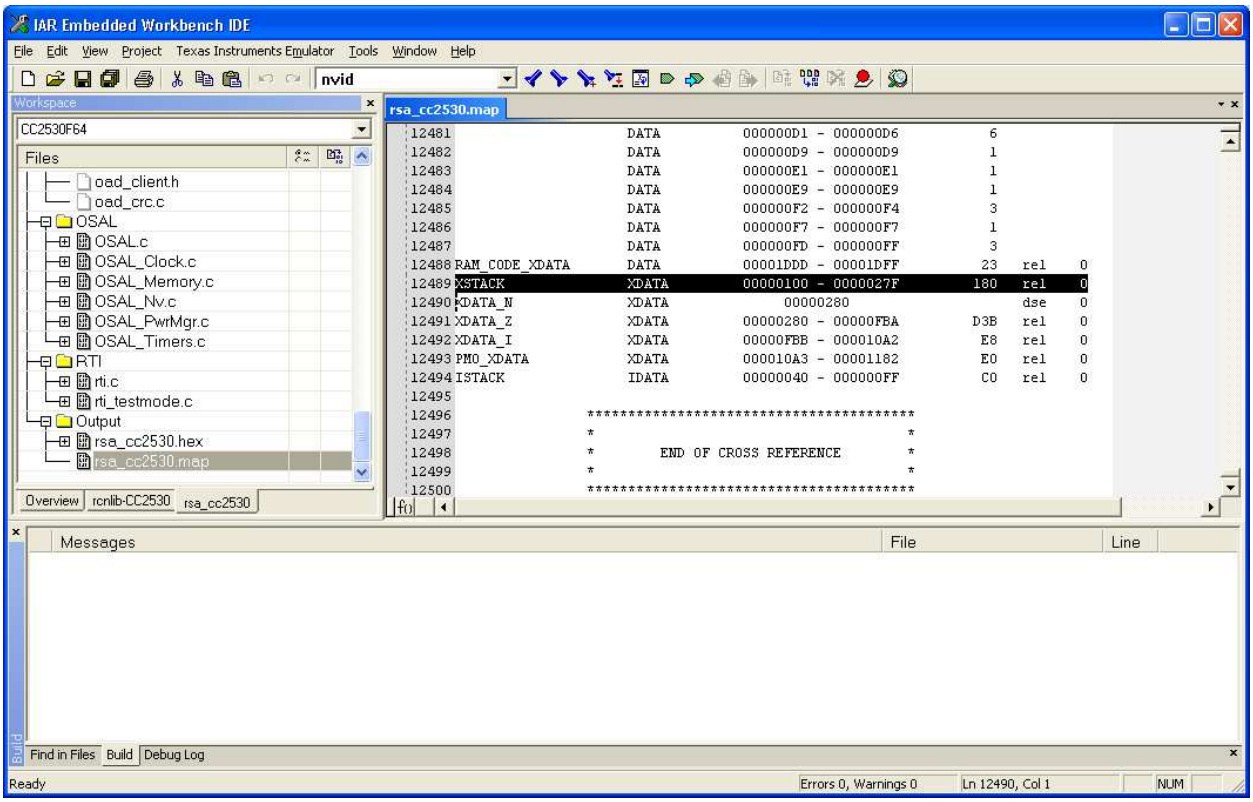

Figure 8 – Finding stack location

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 used. In Figure 9, XDATA stack was used down to 0x1DF, which makes the stack depth in this use case to be 0x27F – 0x1DF + 1 = 161 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).

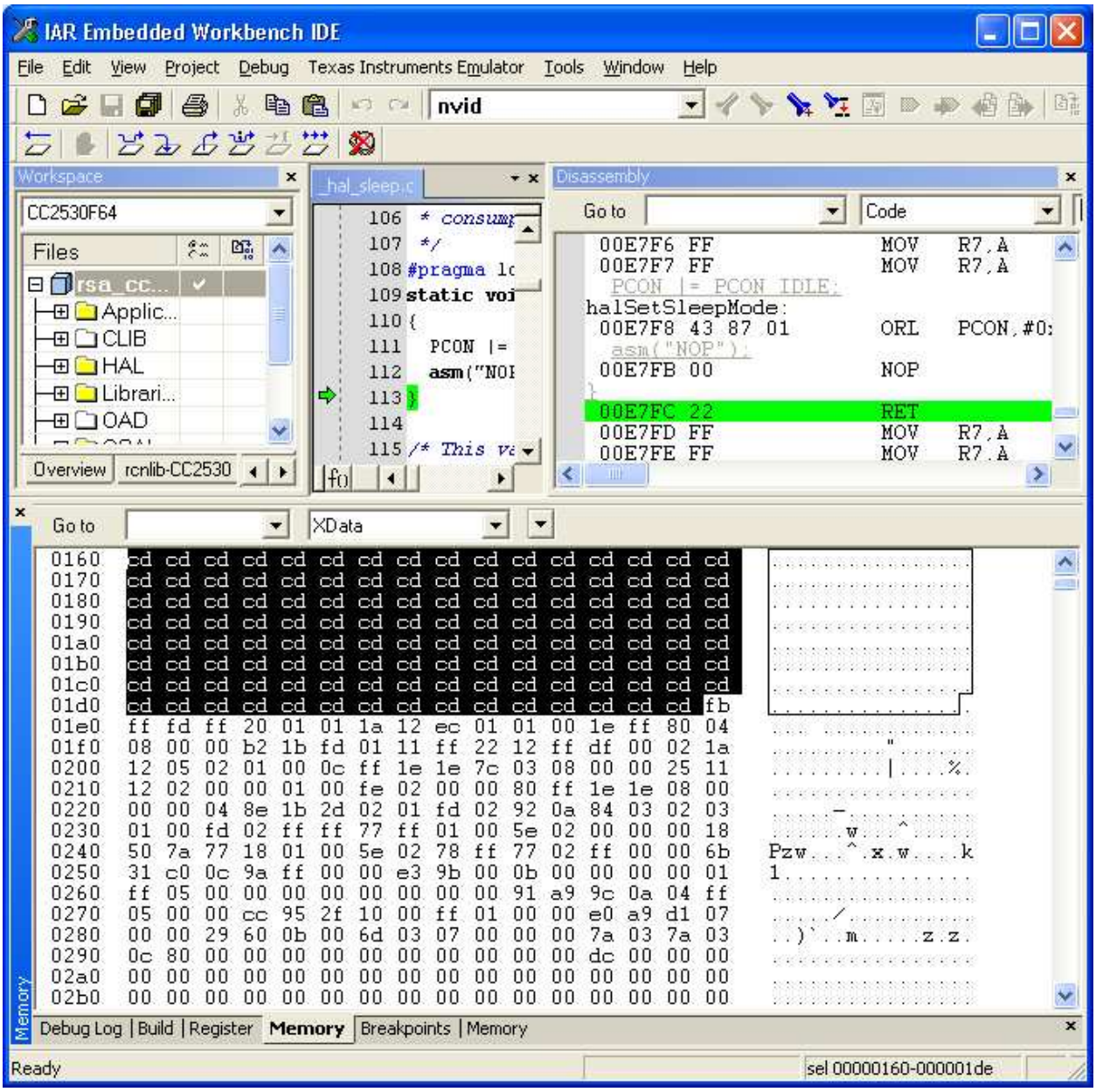

Figure 9 – XDATA Stack Profiling

RemoTI software uses heap through OSAL memory management module. Application could also use heap memory through OSAL memory management module but the basic remote sample application as is does not.

Heap usage varies 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 product. Heap size can be adjusted by defining INT_HEAP_LEN in preprocessor definition of the project with the desired number of bytes as its value. The default value is 2048 (bytes).

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 inside the osal_mem_init() function of the OSAL_Memory.c module. At the beginning of the function, add a 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 used, the heap space is initialized with OSALMEM_INIT value instead of 0xCD in the above code. OSALMEM_PROFILE compile flag brings in more code than the heap initialization, which is not explained in this document.

With the new image, after running the use case with maximum heap usage, break the debugger and check the *_theHeap* memory space. The address range of *_theHeap* variable can be found from map file.

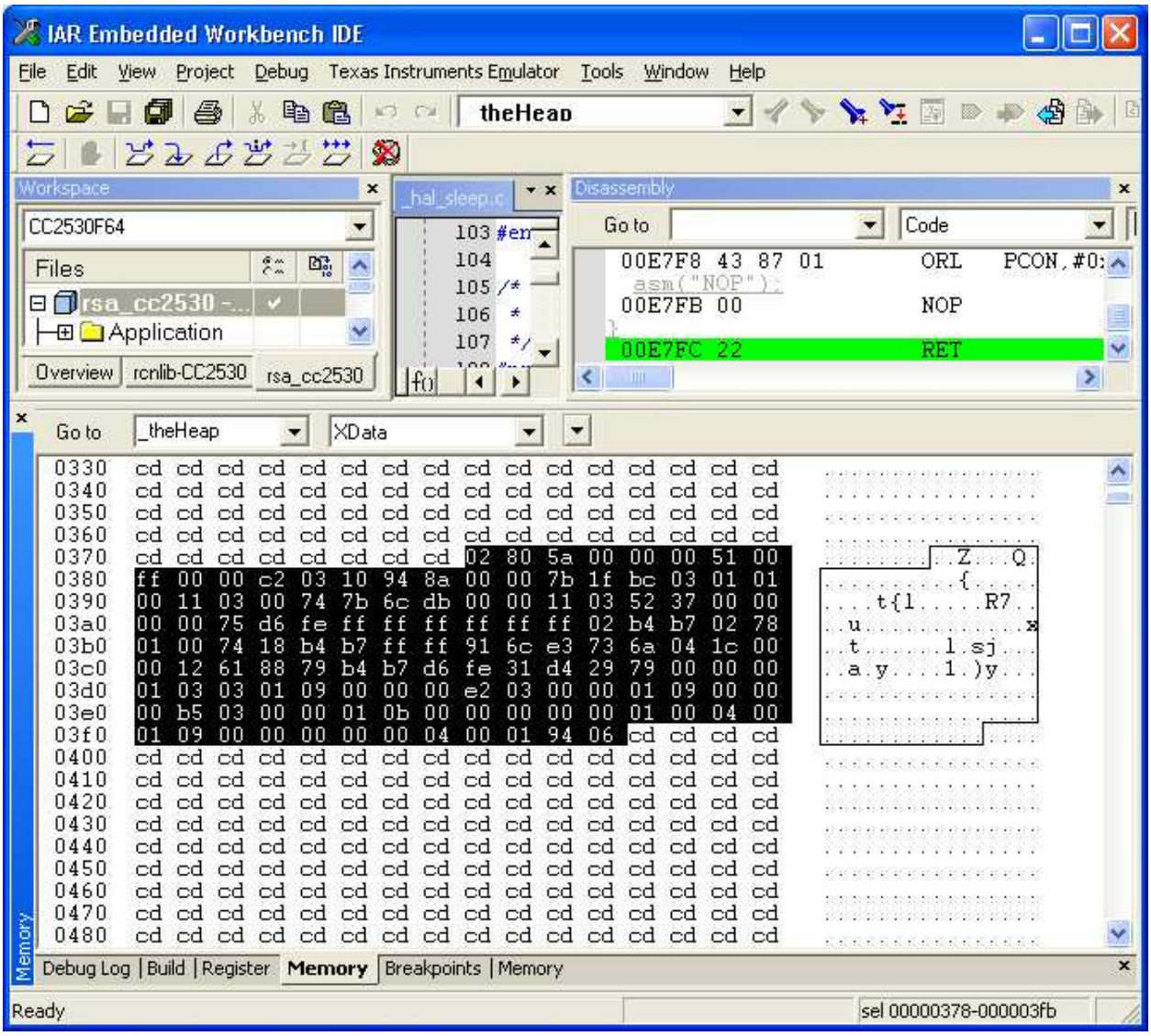

Figure 10 – Heap usage profiling

If the *_theHeap* variable occupies 0x290 to 0xA8F address space for example, search from 0xA8F down to bottom for any foot print of memory usage. In Figure 10, 0x3fb is the highest address of memory space that was used in the heap space. That amounts to 0x3fb - 0x290 + 1 = 364 bytes of heap usage.

Once heap size is profiled, the heap size can be adjusted by adding INT_HEAP_LEN definition as compile option. For instance adding INT_HEAP_LEN=1024 to defined symbols window of Preprocessor tab of C/C++ Compiler category, adjusts heap size to 1,024 bytes.

10. Network layer interface

The basic remote sample application uses the RemoTI application framework interface instead of directly accessing network layer interface. RemoTI application framework interface and RemoTI network layer interface are described in [2].

The RemoTI application framework module, rti.c, uses RemoTI network layer API. The basic remote controller sample application uses either rcnctrl-CC2530.lib or rcnctrl-CC2530-banked.lib. The selection is based on whether the desired code model is the near code model or the banked code model. Both network layer libraries implement a subset of full network layer features. In order to use the full features of network layer, rcnsuper-CC2530.lib or rcnsuper-CC2530-banked.lib should be used instead. Note that using rcnsuper-CC2530xxxx.lib increases code size. Table 10 shows difference of features supported by the two sets of libraries.

Table 10 – Network layer library features

Network layer features	rcnctrl	rcnsuper
Discovery originator	√	√
Discovery recipient		√
Pairing originator	√	√
Pairing recipient		√
Un-pair originator	√	√
Un-pair recipient	√	√
Frequency agility (specific to target base channel change based on interference)		√
Power saving mode (duty cycling active period of receiver on)		√
Receiver control (NLME-RX-ENABLE)	√	√
Auto discovery		√
Security	√	√
Controller node capability	√	√
Target node capability		√
NSDU originator	√	√
NSDU recipient	√	√

RTI has a feature compile flag, *FEATURE_CONTROLLER_ONLY* to reduce code size when used just for a controller. For basic remote controller sample application, this feature flag is set in project options by default.

Note that the basic remote controller sample application incorporated a state machine to ensure that network layer interface is triggered in acceptable state. For instance, when calling *RTI_SendDataReq*() which eventually triggers *RCN_NldeDataReq*() call, application changes its state (*rsaState* variable) to *RSA_STATE_NDATA* state, so that another key press event does not trigger *RCN_NldeDataReq*() again till the state changes back to *RSA_STATE_READY* upon *RTI_ReceiveDataInd*() called from *RCN_CbackEvent*() for NLDE-DATA.confirm event. Note that RemoTI network layer does not queue data request and hence back to back data requests without waiting for confirmation in between would fail.

11. Pairing

The basic remote controller sample application triggers pairing simply by calling *RTI_PairReq*() upon pairing key press event. RTI, in turn, performs discovery and pairing in sequence.

When RTI performs discovery, it specifies CERC profile and searches for any device type. To change this behavior, modify the *rcnNlmeDiscoveryReq_t* structure build into *rtiReqRspPrim.prim.discoveryReq* buffer in *RTI_PairReq*() function. Note that discovery related NIB attributes are set up to comply with CERC profile in *rtiResetSA*() function.

RTI filters the discovered node descriptor by looking at the device type of the node descriptor. The device type list of a node descriptor is compared against supported target device type list set by RTI configuration parameter (*RTI_CP_ITEM_NODE_SUPPORTED_TGT_TYPES*). Supported target types can be set up to six entries. In order to change this behavior, modify *rtiOnNlmeDiscoveredEvent*() function. In order to simply increase maximum number of supported target types, change *RTI_MAX_NUM_SUPPORTED_TGT_TYPES* constant in rti.h file and rebuild RTI module, rti.c file.

Once discovery completes with acceptable node descriptor, *rcnNlmePairReq_t* structure is built and *RCN_NlmePairReq*() function is called to move on with network layer pairing procedure. See *rtiOnNlmeDiscoverCnf*() function for details.

When pairing successfully completes, the application sets destination target (*rsaDestIndex*, used for destination of any CERC command till next change) with the newly paired entry in *RTI_PairCnf*() callback function. This RTI callback function is triggered from *RCN_CbackEvent*() callback corresponding to NLME-PAIR.confim event.

Once pairing table is full, the basic remote sample application won't be able to pair with a new device any longer. Depressing any key that is mapped to CERC Stop user control command, for five seconds, triggers RTI level state clearing. Refer to [2] for the effect of state attribute clearing. In short, application will restart with NIB including pairing table reset to default values. In order to change such usage of a CERC Stop key, modify *RSA_KeyCback*() function.

Basic remote controller sample application does not handle any cancellation of on-going pairing procedure. Cancellation of on-going pairing by the user is not recommended as the pairing progress

unknown to the user. In case it is required for certain use cases, cancellation can be done by resetting the system (e.g., by calling *RTI_SwResetReq*() function).

12. Target device selection

As described in chapter 11, newly paired target is selected as destination device. When there are multiple devices already paired and application wants to allow multiple pairing, a mechanism to switch back to another paired device is necessary. CC2530 remote platform has four keys right below power key, labeled as *TV*, *DVD*, *STB* and *AUX*. Basic remote controller sample application uses these keys as target designator keys to use them to switch to another paired entry as destination. See chapter 5 for configuring key code to target device type select command mapping.

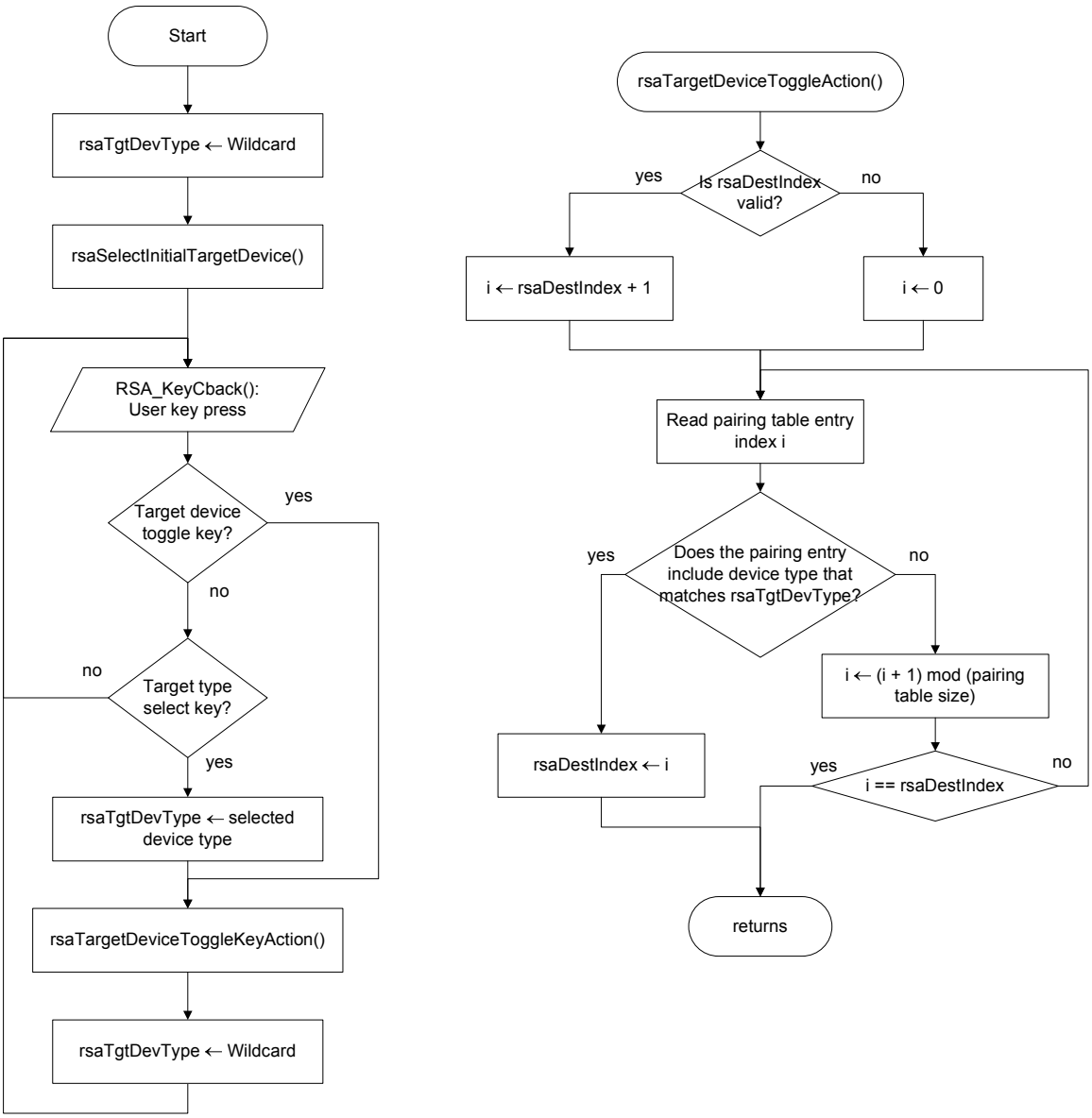

Figure 11 – Destination Target Selection Algorithm

Figure 11 illustrates algorithm used for selecting the destination target device. *rsaSelectInitialTargetDevice*() simply goes through pairing table and finds the first valid entry and select the entry as destination device.

The difference between target device toggle key and target type select key is that target device toggle key triggers selection of any next valid pairing entry as destination while target type select key triggers selection of a next valid pairing entry that includes the designated device type in its device type list.

Note that the initial target device selection algorithm can be improved, for example, by storing *rsaDestIndex* value into OSAL NV memory and restoring it in *rsaSelectInitialTargetDevice*() function.

13. Non-volatile memory

The basic remote sample application does not use its own OSAL NV item but application can be modified to use its own NV items if necessary. See [4] for API details.

When application uses its own OSAL NV items, the NV item identifiers must not conflict with the ones used by network layer and RemoTI application framework (RTI module). Table 11 shows the NV item identifiers reserved by network layer and RemoTI application framework.

Table 11 – NV item identifier ranges

Reserved NV item identifier range	Description
0x0000	NULL identifier
0x0001 – 0x01FF	Reserved for future usage
0x0200 – 0x0300	Reserved for RemoTI network layer
0x0301 – 0x0400	Reserved for RemoTI application framework (Search for *RTI_NVID_xxx* constants in *rti.c* file to find exact identifiers used)
0x0401 – 0x0FF	Application
0x1000 – 0xFFFF	Reserved

14. IEEE address

CC2530 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.

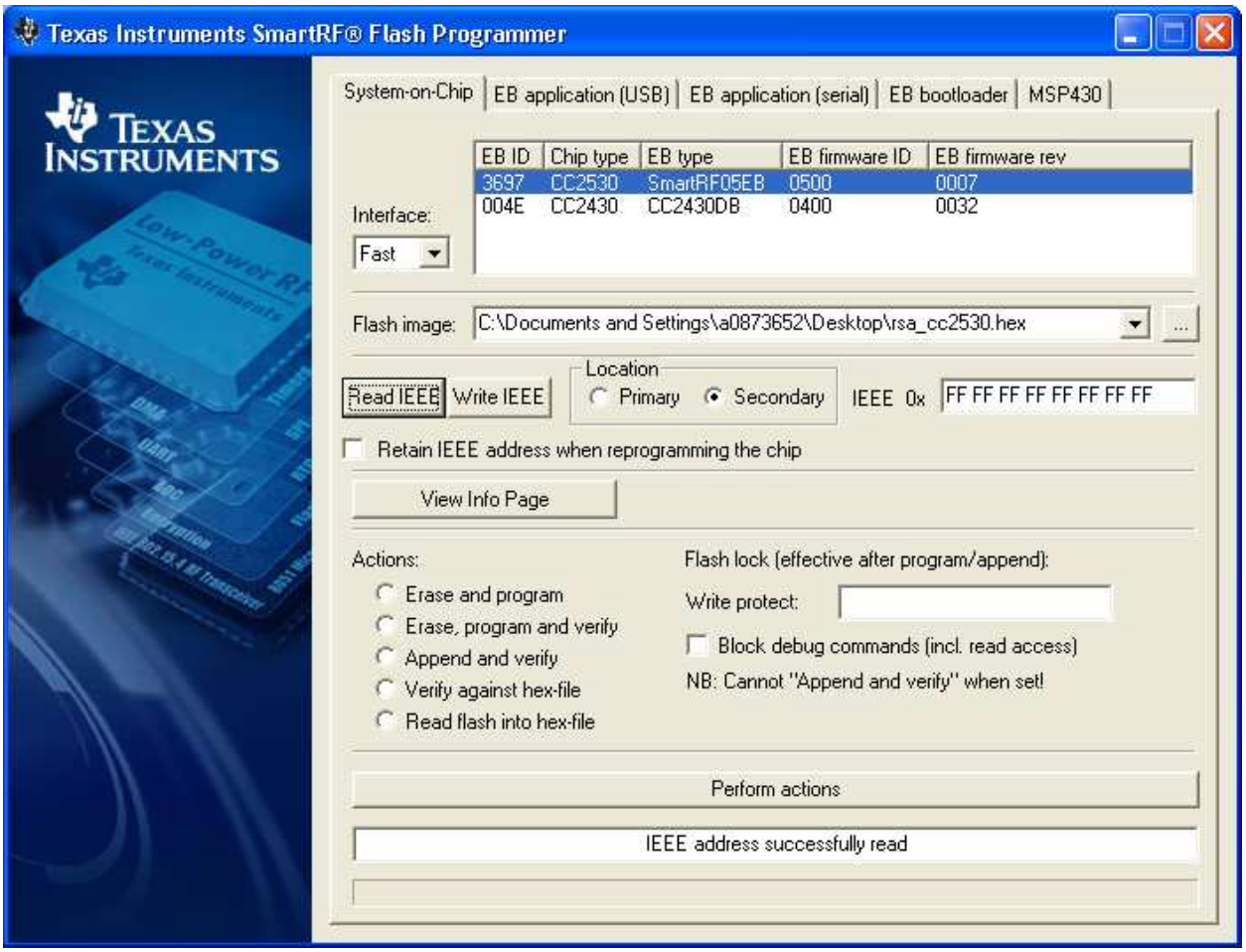

Figure 12 – SmartRF programmer

Hence, with RemoTI application framework, the hierarchy of IEEE address upon CC2530 reset is as follows:

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

Figure 13 illustrates the flow chart of selecting the network layer IEEE address, during startup of a device.

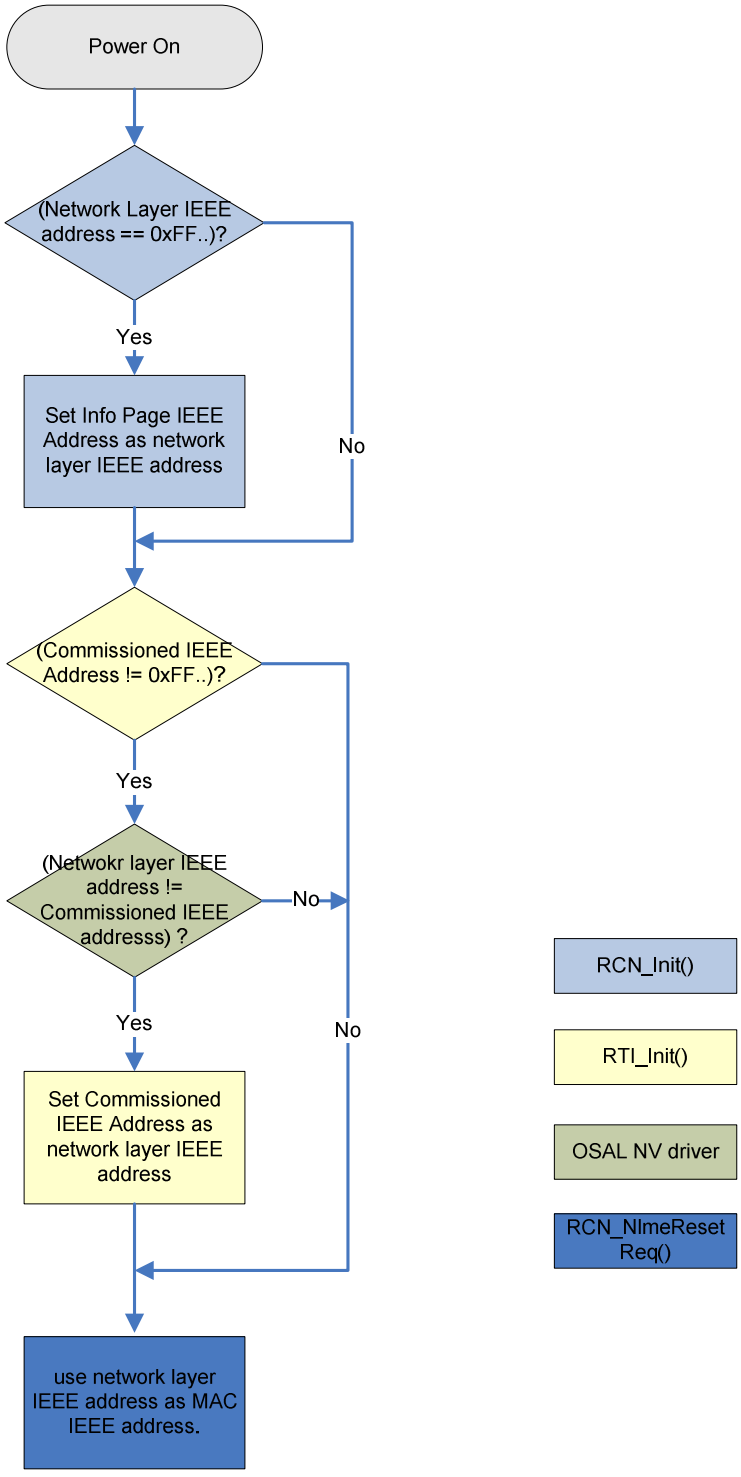

Figure 13 – IEEE address selection flow during startup

15. IR signal generation

The basic remote sample application includes IR signal generation driver as a demonstration of the CC2530 capability. Note that there are a variety of IR signal formats and signal generation driver has to be optimized in terms of its memory usage and code (data table) size to suit the signal format.

The included IR signal generation driver (hal_irgen.c and hal_irgen.h) is optimized for IR signal formats that have all carrier on and off time duration derived as a relatively small multiples of a certain time duration. For example, Manchester coding would use the same time unit for carrier on and carrier off duration to compose a bit.

Some other IR signals transmit carrier for certain duration and turns off carrier for two different durations to discriminate bit value 0 and 1. Both carrier-off durations are usually a certain multiple of the carrier on duration. If both are small multiples of the carrier on duration, sample IR generation driver included in the package is suitable for the particular signal generation.

The sample driver uses a single DMA channel and there isn't any software interaction while an entire command signal is being generated, ensuring correct signal format regardless of interrupt latency.

The IR generation driver outputs the signal to peripheral IO port 1 pin 1. The hal_irgen.c module has to be modified in order to use a different port pin.

The sample IR generation driver (hal_irgen.c, hal_irgen.h) uses a single timing unit as a basis for all timing generation. The common factor timing duration is determined by HAL_IRGEN_TICKSPD, HAL_IRGEN_BIT_TIMING_PRESCALER and HAL_IRGEN_BIT_TIMING_TICKS compile flags. Compile flags follow in this chapter.

Bit 0 and bit 1 signal formats are then defined as a sequence of carrier on and carrier off durations as multiples of the aforementioned common factor duration.

The driver also supports generically adding preamble signal that precedes all bits for a single command. Preamble is also defined using carrier on and carrier off durations as multiples of the common factor duration.

Table 12 lists compile flags used for configuring the timing and signal formats.

Table 12 – HAL IR generation driver compile flags

Compile Flag	Description
HAL_IRGEN	Set to TRUE in order to include HAL IR signal generation driver
HAL_IRGEN_ACTIVE_HIGH	Set to TRUE when active state should be asserted high on port output, and set to FALSE otherwise
HAL_IRGEN_TICKSPD	Set to HAL_IRGEN_TICKSPD_xxxHZ macro for configuring tick speed

Compile Flag	Description
HAL_IRGEN_CARRIER_PRESCALER	Set to HAL_IRGEN_CARRIER_PRESCALER_DIVxxx macro for configuring prescaler for carrier PWM generation
HAL_IRGEN_CARRIER_DUTY_CYCLE	Set to timer tick counter comparator value for duty cycle duration setup for carrier PWM
HAL_IRGEN_CARRIER_PULSE_WIDTH	Set to timer tick counter comparator value for pulse width duration of carrier PWM. For non-modulated IR signal generation, this macro should be set to 0xff
HAL_IRGEN_BIT_TIMING_PRESCALER	Set to HAL_IRGEN_BIT_TIMING_PRESCALER_DIVxxx macro for configuring prescaler for bit information common factor duration
HAL_IRGEN_BIT_TIMING_TICKS	Set to timer tick counter value for bit information common factor duration
HAL_IRGEN_BIT_0_PRE_CARRIER_DUR	Set to value of a multiple of common factor duration, corresponding to carrier non-transmission duration for bit value 0 preceding carrier transmission
HAL_IRGEN_BIT_1_PRE_CARRIER_DUR	Set to value of a multiple of common factor duration, corresponding to carrier non-transmission duration for bit value 1 preceding carrier transmission
HAL_IRGEN_BIT_0_CARRIER_DUR	Set to value of a multiple of common factor duration, corresponding to carrier transmission duration for bit value 0
HAL_IRGEN_BIT_1_CARRIER_DUR	Set to value of a multiple of common factor duration, corresponding to carrier transmission duration for bit value 1
HAL_IRGEN_BIT_0_POST_CARRIER_DUR	Set to value of a multiple of common factor duration, corresponding to carrier non-transmission duration for bit value 0 following carrier transmission
HAL_IRGEN_BIT_1_POST_CARRIER_DUR	Set to value of a multiple of common factor duration, corresponding to carrier non-transmission duration for bit value 1 following carrier transmission

Compile Flag	Description
HAL_IRGEN_PREAMBLE_PAUSE_1_DUR	Set to value of a multiple of common factor duration, corresponding to the carrier non-transmission preceding the first carrier transmission for a preamble
HAL_IRGEN_PREAMBLE_CARRIER_1_DUR	Set to value of a multiple of common factor duration, corresponding to the first carrier transmission for a preamble
HAL_IRGEN_PREAMBLE_PAUSE_2_DUR	Set to value of a multiple of common factor duration, corresponding to the carrier non-transmission preceding the second carrier transmission for a preamble
HAL_IRGEN_PREAMBLE_CARRIER_2_DUR	Set to value of a multiple of common factor duration, corresponding to the second carrier transmission for a preamble
HAL_IRGEN_PREAMBLE_PAUSE_3_DUR	Set to value of a multiple of common factor duration, corresponding to carrier non transmission following the second carrier transmission of a preamble, and preceding data bit transmission
HAL_IRGEN_DOUBLE_LENGTH_BIT_IDX	Set to the index (from the least significant bit location as index 0) of bit location of a command which has to extend to double the length of a normal bit signal. The value should be set to HAL_IRGEN_CMD_LENGTH if no bit should have such an exception. Example usage of this compile flag is the trailing bit location of RC6 signal format.

Also, compile flags HAL_IRGEN_RC5_CONFIG and HAL_IRGEN_RC6_CONFIG define all the above compile flag parameters for RC5 and RC6 signal formats.

Note that none of the default project configurations enable HAL_IRGEN option.

Figure 14 shows example preamble format which precedes all command data signal format.

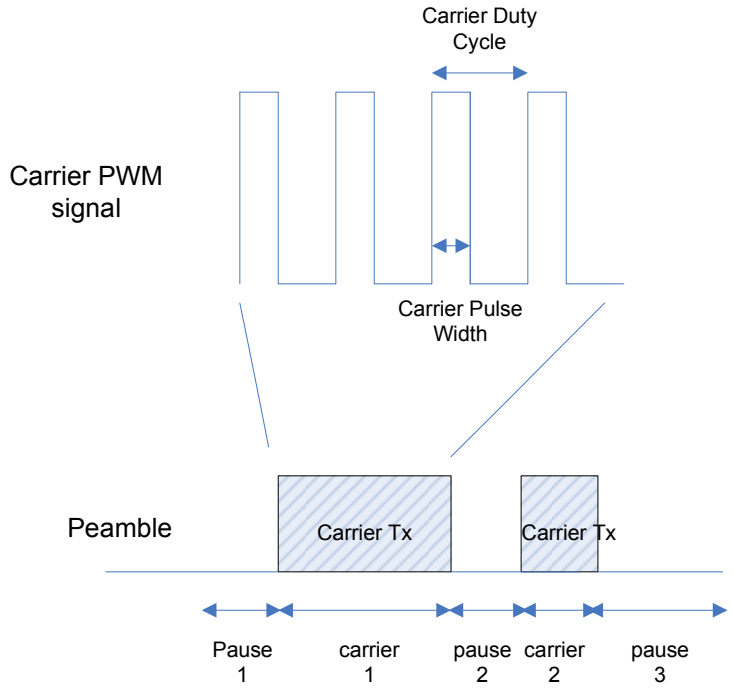

Figure 14 – IR preamble signal format

Figure 15 shows an example data signal format following preamble:

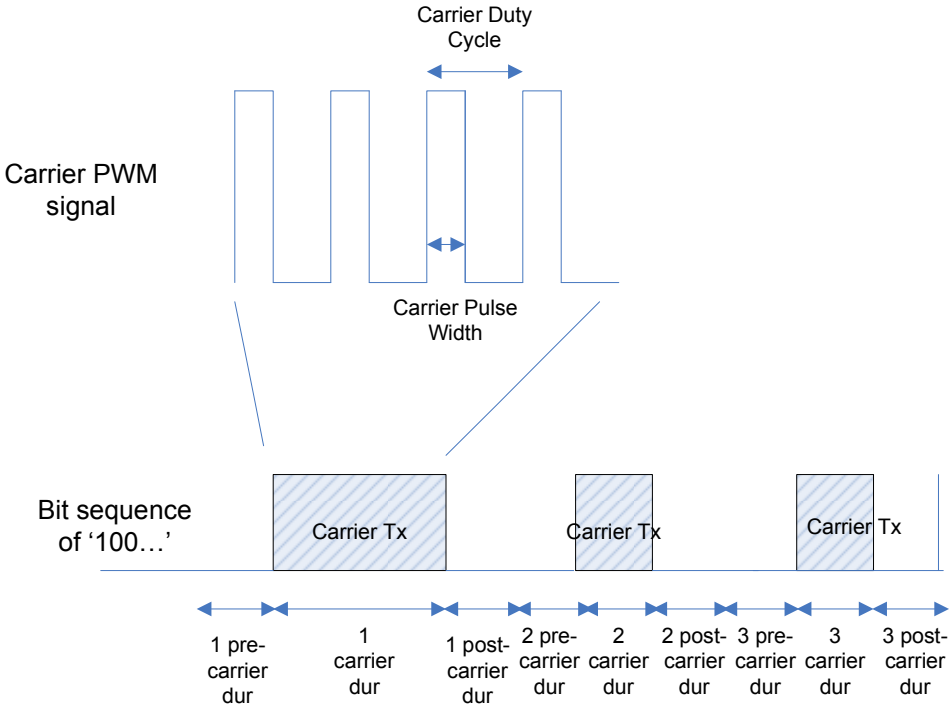

Figure 15 – IR bit signal format

16. Network layer configuration

The standard NIB attributes can be configured and updated at run time through *RTI_WriteItem*() function or *RCN_NlmeSetReq*() function in case the 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_WriteItem* or *RCN_NlmeSetReq*() are enumerated in rcn_attribs.h file. Note that several non-standard attributes are also provided.

Table 13 explains the non-standard attributes.

Table 13 – Non-standard network layer 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 14.

Attribute identifier	Description
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.

17. Over the air download

17.1 Overview of the over the air download demo

Over the air download is a feature that enables a RemoTI controller device to download its embedded software image from a RemoTI target device over the air. It is out of scope of this document how the target node gets a software image for a particular controller device.

Over the air download demo consists of a basic remote controller sample application which is built in OAD specific configuration, a network processor and OAD demo PC tool. See [6] for build, setup and execution instruction.

Over the air downloading demo is implemented using two way data exchange using RemoTI network layer service data unit transmission and reception feature. The network layer service data unit payload used for image downloading is Texas Instruments vendor specific data. TI vendor specific data format was designed to work with any profile identifier. **Note that TI vendor specific data format is solely defined by TI. Customers are allowed to use the format as is based on the governing product license agreement terms but they are not allowed to use a derived, modified or newly created data format and use the TI vendor identifier with such a format. Such use of TI vendor identifier with unauthorized format could break inter-operability with products that use TI vendor specific commands and further cause significant damage to TI business.**

See section 17.2 for the format of frames used for over the air download demo. The demo uses over the air download protocol frames and poll protocol frame.

A typical remote controller optimizes current consumption by turning off radio receiver whenever possible. Basic remote controller sample application incorporates the same logic. When idling, radio receiver is turned off and radio processor enters power mode 3 to save power consumption. As such is the case, a target device which has a new software image to download to a remote controller paired with that device does not know when the remote controller wakes up and is ready to receive packets from the target device, unless it is notified by the controller device. Hence, a poll command is used by the basic remote

controller sample application to notify a designated target that the remote has turned on its receiver for a short timing window. In the sample application, the poll command is transmitted upon user key input and to a currently designated target device as unicast transmission. In real product remote controllers, the triggering mechanism could vary such as periodic transmission, selecting target based on target device type, use of broadcast transmission, etc. Periodic transmission and choice of poll command recipient should be considered with power consumption and timing.

Once the poll command is received by a target device (in demo, OAD demo PC tool via connected network processor), the target device knows that it could start sending a data to the remote controller within a short period of time. In the OAD demo, the OAD demo PC tool sends OAD command packets and the basic remote enters OAD mode and leaves the receiver on till watchdog timer expires (note that this is application watchdog timer, different from CC2530 watchdog timer) or till OAD command directs remote to exit OAD mode. The target device controls the OAD downloading procedure and remote controller receives the command, performs the requested actions and sends back responses.

In the demo, OAD replaces entire image of basic remote controller including RemoTI network layer image. The only exception is a small boot code that resides in the first page of flash (See chapter 8). The basic remote controller stores the image to downloaded code area of the flash and resets itself to run the boot code which copies the downloaded image to the active image area and executes active image.

17.2 Over the air download command packet formats

17.2.1 Poll protocol frame format

Poll protocol shall a single frame type to be called Poll frame, and its format is illustrated in Figure 16. Protocol payload size shall be zero.

Bits : 8
Protocol Identifier

Figure 16 – Poll frame format

17.2.2 General Over the Air Download protocol frame format

General Over the Air Download protocol frame format is illustrated in Figure 17.
Over the Air Download protocol frames have 8 bit command identifier followed by command payload as protocol payload.

Bits : 8	8	Variable
Protocol Identifier	Command Identifier	Command Payload

Figure 17 – General Over the Air Download protocol frame Format

17.2.3 Over the Air Download protocol command identifiers

All valid command identifiers are listed in Table 14.

Table 14 – Over the Air Download Server protocol command Identifiers

Command identifier	Command
0x00	Status Request
0x01	Start Request
0x02	Data Request
0x03	Enable Request
0x04	Cancel Request
0x80	Status Response
0x81	Start Response
0x82	Data Response
0x83	Enable Response
0x84	Cancel Response

17.2.4 Status Request command

Status Request command has zero length command payload.

17.2.5 Start Request command

Start Request command payload format is illustrated in Figure 18.

Bits : 16
Image Size

Figure 18 – Start Request command frame format

17.2.6 Data Request command

Data Request command payload format is illustrated in Figure 19.

Bits : 16	16	512
Sequence Number	Address	Data

Figure 19 – Data Request command frame format

17.2.7 Enable Request command

Enable Request command payload format is illustrated in Figure 20.

Bits : 32	32
Image ID	CRC

Figure 20 – Enable Request command frame format

17.2.8 Cancel Request command

Cancel Request command has zero length command payload.

17.2.9 Status Response command

Status Response command payload format is illustrated in Figure 21.

Bits : 8	8	32	32	24
OAD State	Last Rx Sequence Number	Active Image ID	Secondary Image ID	Secondary Storage Size

Figure 21 – Status Response command frame format

Valid OAD State field values are listed in Table 15.

Table 15 – OAD State Field Values

Value	State
0x00	Idle
0x01	Download
0x02	Validate

17.2.10 Start Response command

Start Response command payload format is illustrated in Figure 22.

Bits : 8
Status

Figure 22 – Start Response command frame format

Valid Status field values are listed in Table 16.

Table 16 – OAD Response Status Field Value

Value	Status
0x00	Success
0x01	General Failure
0x02	Already Started
0x03	Not Started
0x04	Invalid Sequence Number
0x05	Download Incomplete
0x06	Bad CRC
0x07	Invalid File
0x08	No Response
0x09	Cancelled
0x0A	No Resources
0x0B	File System Error
0x0C	Image Too Big

17.2.11 Data Response command

Data Response command payload format is illustrated in Figure 23.

Bits : 8	16
Status	Sequence Number

Figure 23 – Data Response command frame format

Valid Status field values are listed in Table 16.

17.2.12 Enable Response command

Enable Response command payload format is illustrated in Figure 24.

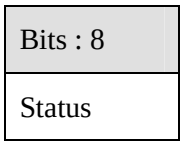

Figure 24 – Enable Response command frame format

Valid Status field values are listed in Table 16.

17.2.13 Cancel Response command

Cancel Response command payload format is illustrated in Figure 25.

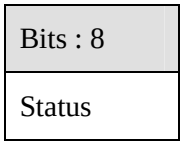

Figure 25 – Cancel Response command frame format

Valid Status field values are listed in Table 16.

17.3 Over the air download command flow sequence

Figure 26 illustrates flow of over the air download commands between a device which triggers over the air download and transmits a new image and a remote that receives the new image over the air.

The over the air download demo PC tool in the RemoTI development kit plays the role of image distributor device. The demo tool sends status request command prior to sending start request in order to update its window display of the image status before initiating download.

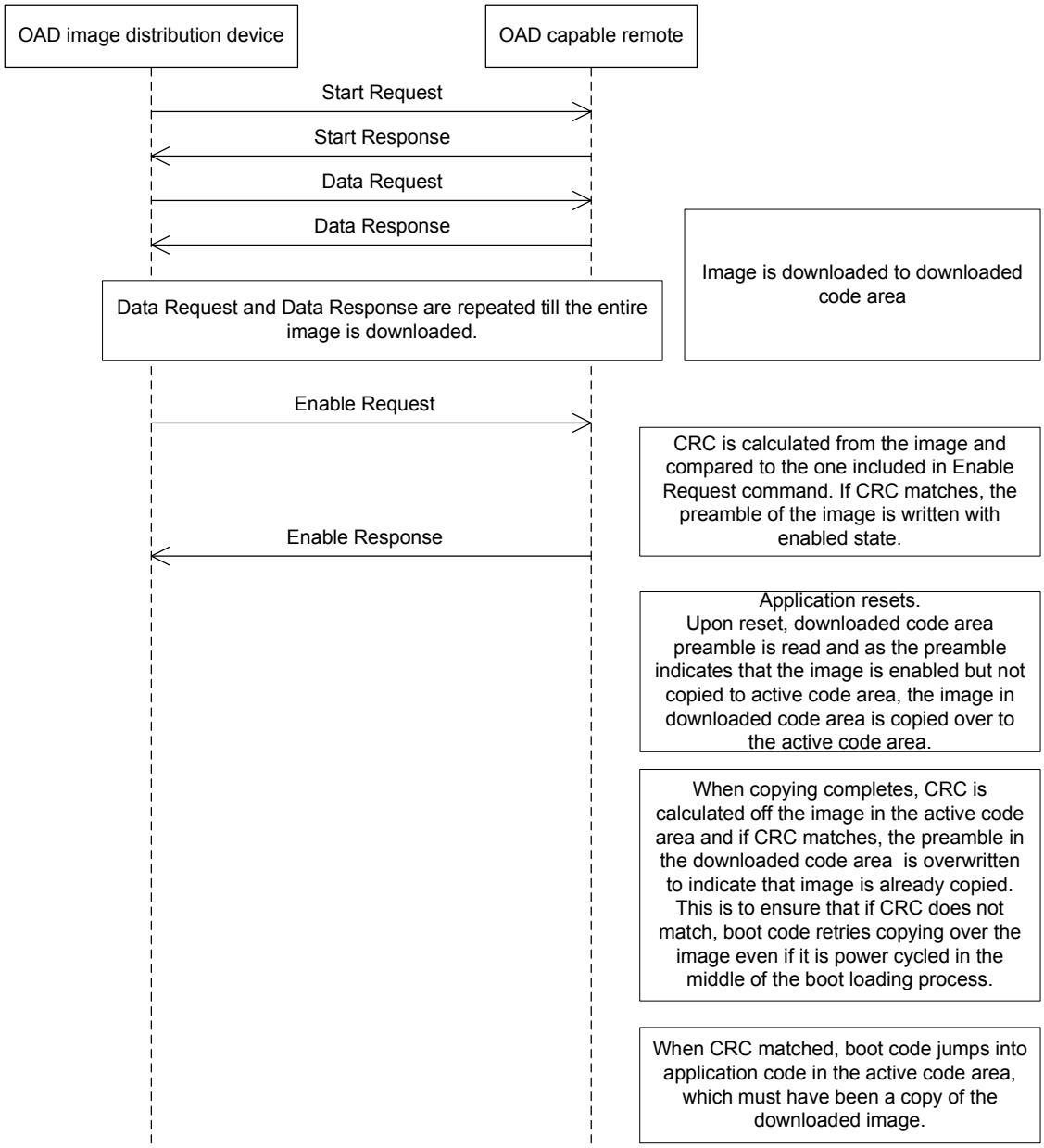

Figure 26 – OAD command flow

17.4 Basic remote controller sample application configuration

Basic remote controller sample application includes over the air download demo feature in certain configurations (CC2530F128_OAD and CC2530F256_OAD).

When the OAD enabling configuration is in use, OAD files are compiled together (oad_appflash.c, oad_client.c, oad_crc.c. See section 3.3), proper OSAL non-volatile memory configuration is selected (CC2530F128 or CC2530F256OAD. See section 3.2) and OAD specific linker command file is selected.

Such a configuration also adds *FEATURE_OAD* compile flag definition, which enables OAD configuration specific key command map change and actions in rsa_basic.c module.

Basic remote controller sample application takes charge of the following tasks with related to OAD demo:

- Transmission of poll command (triggered by user key input) and receiver turn on for a short period.
- Forwarding received OAD protocol frames to OAD module (*OAD_ReceiveDataInd*() function in oad_client.c module)
- Blocking other user activities while OAD is active (use of *rsaState* variable. Whether OAD is active or not can be determined by return value of *OAD_ReceiveDataInd*() call).
- Leaving the radio receiver on while OAD is active and turning it off when exiting OAD mode.
- Managing application watchdog timer to monitor OAD activity.
- Resetting the system when requested by OAD module (as a return value of *OAD_ReceiveDataInd*() call).

See the code wrapped within *FEATURE_OAD* compile flag in rsa_basic.c file to see the code corresponding to the above tasks.

17.5 Retention of non-volatile memory data

The build configuration of the basic remote controller sample application projects creates a downloadable image of the entire application including RemoTI network layer code. The downloadable image does not include OSAL non-volatile memory data pages so that the non-volatile image data pages are retained.

However, this imposes a restriction that the new OSAL NV driver that is included in new image has to be backward compatible with the non-volatile memory data pages that previous image has created.

There are several ideas to address the issue of retaining non-volatile memory data content while upgrading OSAL NV driver to a non-compatible driver as follows. Note that these ideas are not implemented in the OAD demo and they are presented just for information purpose:

- New NV driver has built-in conversion routine, which convert old NV pages with the new one. This results in bigger code with the conversion code which is going to be used only once.
- Target downloads a special image that only has conversion logic and OAD download feature as application first. Once the special image completes execution, it should trigger downloading of a new image.
- Target device reads NV page content from controller device first and composes new NV pages images to be compatible with the new NV driver. It then attaches new NV pages into downloadable image and performs OAD with this combined application/NV data image. This has potential issue of updating target device not only with a new controller image but also with target device code that executes NV page update.

17.6 Use of image identifier

An image identifier is used to identify an image. The OAD PC tool will read the current image identifier after downloading the image.

OAD demo uses 16 most significant bits of image identifier to identify compatibility of image. That is, the most significant 16 bit of image identifier of the currently active image has to be equal to the image identifier of the new image to download. The target side, i.e. the OAD PC demo tool checks the match of the image identifiers.

Use of 16 most significant bits was arbitrary decision for the demo. Manufacturers have to decide how big their number space should be and if necessary, they could add a new field to image preamble. Image preamble is defined in oad.h file as preamble_t type. Note that OAD PC tool is fixed with the 16 bit use of 32 bit image identifier.

17.7 Lock bit page

When building application for over the air download, lock big page cannot be used to download code image as lock bit page itself cannot be overwritten during execution of code. Lock bit page can be updated only though debug interface.

18. Latency test mode

Basic remote controller sample application includes code for latency testing. Pressing a test mode key (mapped to RSA_ACT_TEST_MODE. See chapter 5) puts the application into test mode and key actions in this mode are handled differently. Radio receiver is also turned on during test mode. See *rsaToggleTestModeKeyAction*() function for the action triggered by the test mode key.

In test mode, the key commands are handled in *rsaRRTRunTest*() function.

A timer event *RSA_EVT_RANDOM_BACKOFF_TIMER* is used during the test mode. See *RSA_ProcessEvent*() function for the code that handles this event.

When test completes, rsaState variable changes its state to *RSA_STATE_TEST_COMPLETED* state and the only key accepted is a key mapped to CERC SELECT user control command, which triggers sending test report. See *RSA_KeyCback*() function and *rsaRRTSendReport*() function for details.

RTI_SendDataCnf() callback function also include logics for test mode.

Figure 27 illustrates latency test mode algorithm.

In order to remove latency test mode code to reduce code size, remove *rsaToggleTestModeKeyAction*() function, *rsaRRTRunTestMode*() function, *rsaRRTSendData*() function, *rsaRRTSendReport*() function and all references to the functions (for instance, remove RSA_STATE_TEST, RSA_STATE_TEST_COMPLETED state code from RTI_SendDataCnf() function).

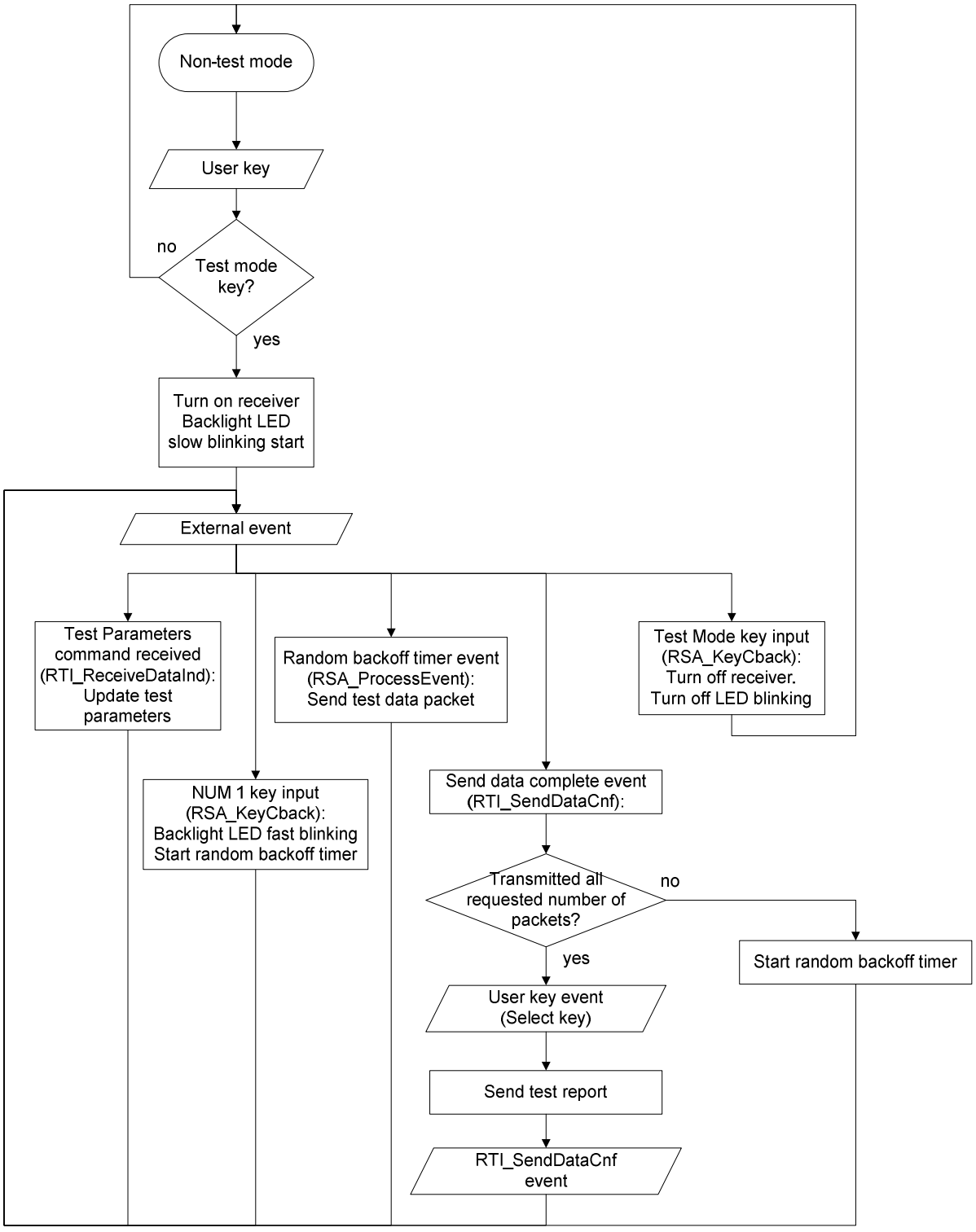

Figure 27 – Latency Test Mode Algorithm

19. DMA, peripheral IO and timers

Basic remote sample application uses the following resources:

- Peripheral IO pin P2_0 for LED
- Peripheral IO pin P1_1 for either LED or IR signal generation.
- Peripheral IO pins P1_0, P1_2, P1_3, P1_4, P1_5, P1_6, P1_7 and all P0 for key matrix.
- DMA channel 0 for non-volatile memory access
- DMA channel 1 for IR signal generation (if IR signal generation is enabled)
- Timer2 (MAC timer) and sleep timer.
- Optionally (with IR generation), timer 1 and timer 4.

20. RF frontend chip connection

Transmit power and receiver gain of CC2530 can be increased by adding an RF frontend chip such as CC2591. A remote sample application has to configure pin-to-pin connections, TX power register table and RSSI adjustment value table in order to support a new remote board design with a CC2591 added to a CC2530.

Both RemoTI stack library files do not support register value table, etc. for CC2591 frontend. Only the rcnctrl-CC2530-banked.lib includes such a table and hence the banked code model has to be used and either the CC2530F128 part or the CC2530F256 part is recommended.

The supported library requires PAEN pin and EN pin of CC2591 to be connected to P1_1 and P1_4 of CC2530 each. If HGM pin of CC2591 is connected to CC2530, the application must set this pin. The sample configuration is in MAC_RfFrontendSetup() function of the Components\mac\low_level\srf04\single_chip\mac_frontend.c file from the development kit installation path. The sample code assumes that the HGM pin of CC2591 is connected to P0_7 of CC2530. But the pin could be connected to Vdd or ground instead. The sample MAC_RfFrontendSetup() function assumes that the HGM pin of CC2591 is connected to P0_7 of CC2530 and P0_7 direction is set to output prior to the call to this function.

Any other GPIO pin configuration has to be consistent with the board design (i.e. not conflicting with those pins). The proper place to call MAC_RfFrontendSetup() function is together with other hardware initializations such as right after the HAL_BOARD_INIT() macro call in the main function of the basic remote sample application.

The MAC_RfFrontendSetup() function also calls MAC_SetRadioRegTable() to select power register value table and RSSI adjustment value table. MAC_SetRadioRegTable() function takes two arguments, TX power register value table index and RSSI adjustment value index. The table indices, i.e. arguments to MAC_SetRadioRegTable() function are dependent on the RemoTI stack library. The rcnctrl-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]).

21. General Information

21.1 Document History

Table 17 – Document History

Revision	Date	Description/Changes
1.0	2009-07-06	Initial release
swru224a	2009-09-18	Added the RF frontend chip connection chapter. Corrected figure 14 and figure 15 captions.

22. 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

23. 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, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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

