RemoTI Basic Remote Developer's Guide

Document Number: SWRU224C

TABLE OF CONTENTS

Acronyms and Definitions

ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
ZRC	New abbreviation for 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
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. This software runs on the same hardware as for the Advanced Remote sample application. The only difference is that the sensor fusion software, and ZID, is no longer supported.

3.1 Features

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

- Compliance with ZigBee RF4CE ZRC profile, including push button pairing and ZRC user command generation
- Compliance with ZigBee RF4CE network layer specification, playing the controller node role
- ZigBee RF4CE network layer security
- 8 x 8 key matrix scanning
- Four target device selection keys
- Maximum ten pairing entries
- Buzzer feedback for pairing
- Latency and PER test mode
- Optional over the air downloading demo

3.2 Build configurations

The RemoTI Basic Remote Sample application project is located in the Projects\RemoTI\BasicRemote\CC2533RC folder.
When you open the workspace file (rsa_cc2533.eww), you can select the different project configurations which are explained in Table 1.

Configuration	Description
CC2533F64	Configuration for CC2533F64 part
CC2533F64-Hex	Hex file generation configuration for CC2533F64 part
CC2533F96	Configuration for CC2533F96 part

Configuration	Description
CC2533F96-Hex	Hex file generation configuration for CC2533F96 part
CC2533F96_OAD	Configuration for CC2533F96 part including over the air downloading feature

Table 1 – Project configurations

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. For a detailed description of these and other compile flags (preprocessor defined symbols), please refer to the chapter on “Compile Options” in [6]. In addition to 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 2 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.

File name	Description
Application	
rsa_basic.c	Legacy basic remote application code implemented on top of RemoTI application framework. Used with the legacy Basic Remote platform.
rsa_basic_arc.c	Key basic remote application code implemented on top of RemoTI application framework. Used with the Advanced Remote platform.
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
nvmigr_nv2snv.c	NV migration utility when moving from OSAL NV system to OSAL Simple NV system (for serial boot loading or for over the air downloading)
nvmigr_nv2snvw.c	NV migration utility when moving from OSAL NV system to OSAL Simple NV API over OSAL NV system (for serial boot loading or for

File name	Description
	over the air downloading)
CLIB	
chipcon_cstartup.s51	Assembly routines to override default C libraries for banked code
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	CC2533 flash device driver
hal_key.c	Key matrix driver
hal_buzzer.c	Buzzer driver
hal_sleep.c	Sleep mode (PM1, PM2, PM3) control implementation
hal_startup.c	Low level initialization code
Libraries	
rcnctrl-CC2530-banked.lib	RemoTI network layer library built for banked code model. This library will be selected for F96 configurations. Note that this library is optimized for controller nodes only.

File name	Description
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	
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_snv.c	OSAL Simplified Non-Volatile memory manager. This has better code size optimization than OSAL_Nv.c module.
osal_snv_wrapper.c	OSAL Simplified Non-Volatile memory manager interface wrapper module to be used with OSAL_Nv.c module
OSAL_Timers.c	OSAL timer implementation
RTI	
rti.c	RemoTI application framework implementation
rti_testmode.c	RemoTI test mode API function implementation

Table 2 – Project files

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 1 illustrates an architectural view.

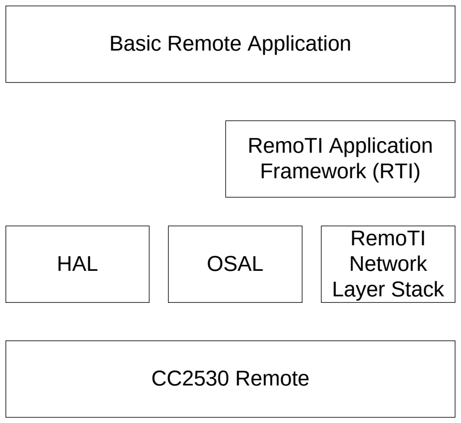

Basic Remote Application
RemoTI Application Framework (RTI)
HAL
OSAL
RemoTI Network Layer Stack
CC2530 Remote

Figure 1 – Basic Remote Software Architecture

4. Key matrix scanning

Key matrix scanning is implemented in hal_key.c file. Note that the file "Components\hal\target\CC2533ARC_RTM\hal_key.c" is specific to the CC2533 advanced remote reference platform so it should not be confused with other hal_key.c implementations such as Components\hal\target\CC2530EB_NPI\hal_key.c.

4.1 Scanning scheme

The key matrix is attached to peripheral IO pins in the CC2533 advanced remote reference platform, as shown in Figure 2:

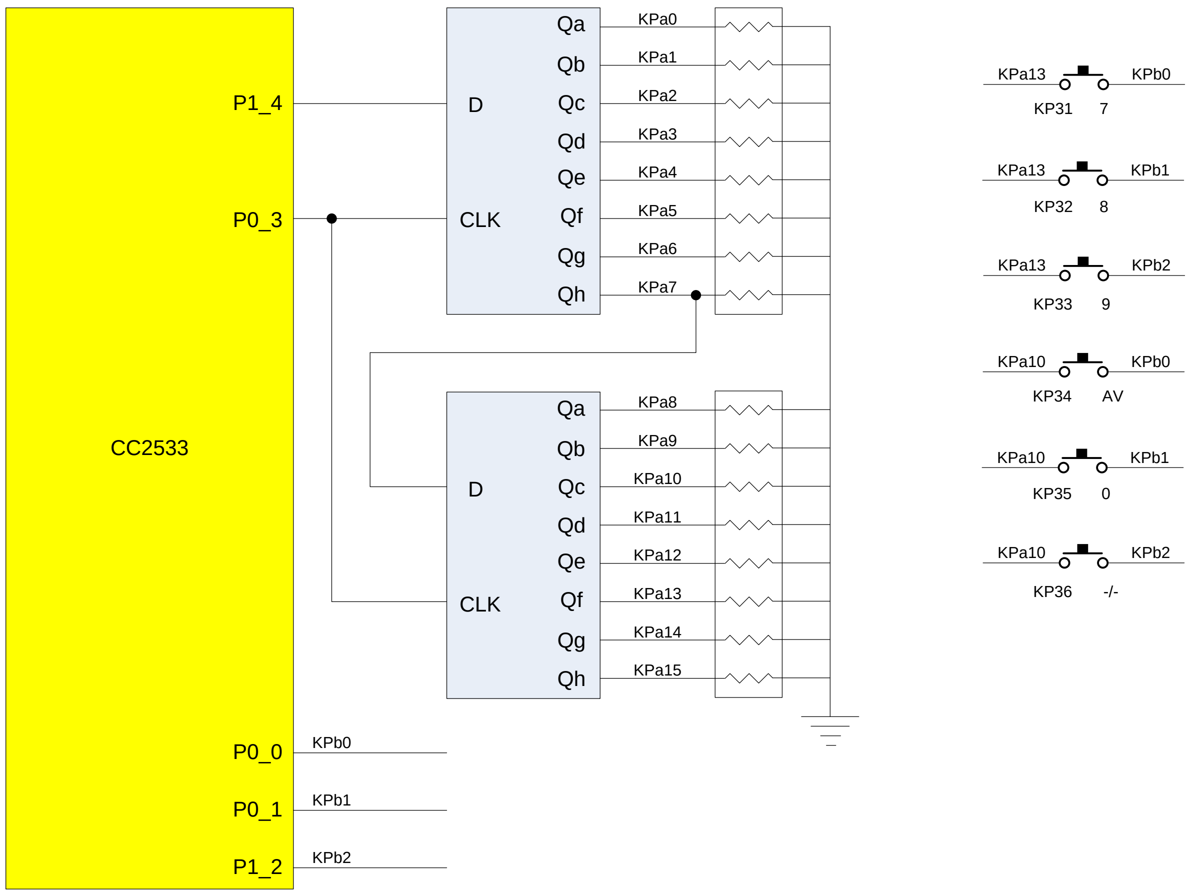


Figure 2 – Key switch matrix circuitry

In the advanced remote reference platform, GPIOs are used to control the key switch matrix, which is implemented via a combination of GPIO pins and a shift register array. 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 3 illustrates the flow of key scanning state changes.

Key scanning itself is performed by shifting a ‘1’ through the shift register by setting P1_4 to a ‘1’, and then repeatedly writing a ‘1’ and then ‘0’ to P0_3 to clock the shift register, with P1_4 set to ‘0’ (so that a single ‘1’ value is clocked in). Each time the shift register is clocked, column pins KPb0, KPb1, and KPb2 are read. For instance, in Figure 2, when KP31 is pressed, the GPIO associated with KPb0 will read a ‘1’ when the shift register is clocked enough times for the ‘1’ to reach KPa13. 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 the CC2533 advanced remote (CC2533ARC_RTM) platform.

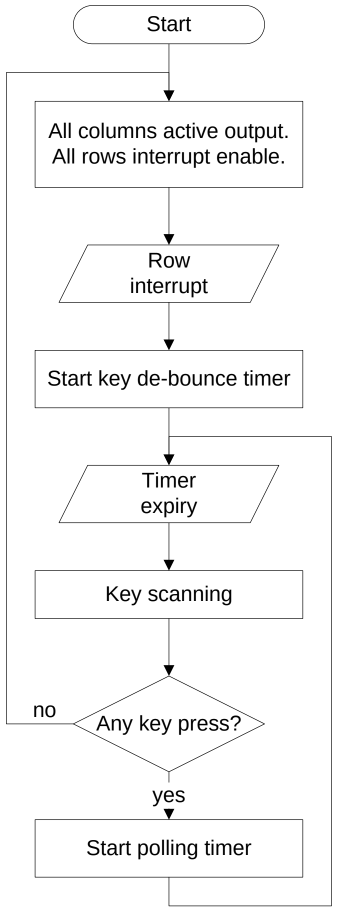


Figure 3 – Key scanning state flow diagram

4.2 Configuration

In the beginning of the hal_key.c module, certain constants are defined with arbitrary choice. Table 3 lists configurable constant values.

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	Polling timer value used in non-interrupt driven key scanning scheme. The polling timer value for interrupt driven key scanning scheme is hard coded to 50 milliseconds to conform to ZRC profile specification. To change the hard-coded timer value, change

Constant Name	Default Value	Description
		the third argument of osal_start_timerEx(Hal_TaskID, HAL_KEY_EVENT, 50) call in HalKeyPoll() function.

Table 3 – HAL key module configurable constants

Note that there are other constants defined in the 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 code 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 ZID or ZRC command in the rsa_point.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 utilizes the rsaKeyMap array to determine whether a key corresponds to a ZID keyboard key or a ZRC key. In addition to mapping the key to a ZID or ZRC command, it also handles key presses differently for test mode, selects the target device with certain special keys, and triggers pairing upon pair key press.

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 ZID command, ZRC command, or a special function implemented in the rsa_point.c module.

The key code map to command is defined in the rsaKeyMap array in the rsa_point.c file. The array index is the key code. The key code definition depends on the HAL key module implementation of a particular platform. For the CC2533 advanced remote, hal_key.c module generates the key code in the format specified in Table 4:

Bit 0-3	Bit 4-5	Bit 6-7
Column number	Row number	Reserved

Table 4 – Key code format

For example, index 0 (0b00 000 000) entry is a map for key row 0, column 0. Index 19 (0b00 01 0011) entry is a map for key row 1, column 3.

The array contains values that identify which profile the command is associated with, and the command (a ZRC command or special command). Table 5 shows the value range.

Value Range	Profile	Description
RTI_ZRC_SELECT to RTI_ZRC_F4_YELLOW	ZRC	RC command code of ZRC User Control commands.
RSA_ACT_PAIR (0xB0)	RTI	Trigger pairing
RSA_ACT_UNPAIR (0xB1)	RTI	Trigger un-pairing
RSA_ACT_TOGGLE_TGT (0xB2)	RTI	Toggle target
RSA_ACT_TEST_MODE (0xB3)	RTI	Test mode toggle
RSA_ACT_POLL (0xB4)	RTI	Send OAD poll command, and enter OAD mode

Table 5 – Key map values

The following table maps the above mentioned keys to the buttons on the advanced remote:

Advanced Remote Button	Description
Red button on bottom row	Trigger pairing
Green button on bottom row	Test mode toggle
Yellow button on bottom row	OAD poll, trigger OAD
TV	Toggle target
Movea (i.e. middle mouse) button	No function
Left / right mouse button	No function
0 – 9 keys	ZRC Numerical 0-9
OK key	ZRC Select
Left / right / up / down keys	ZRC left / right / up/ down.

Table 6 – Advanced Remote Button Descriptions

6. 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();

  /* Initialize 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 4.

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 the OSAL task loop 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.

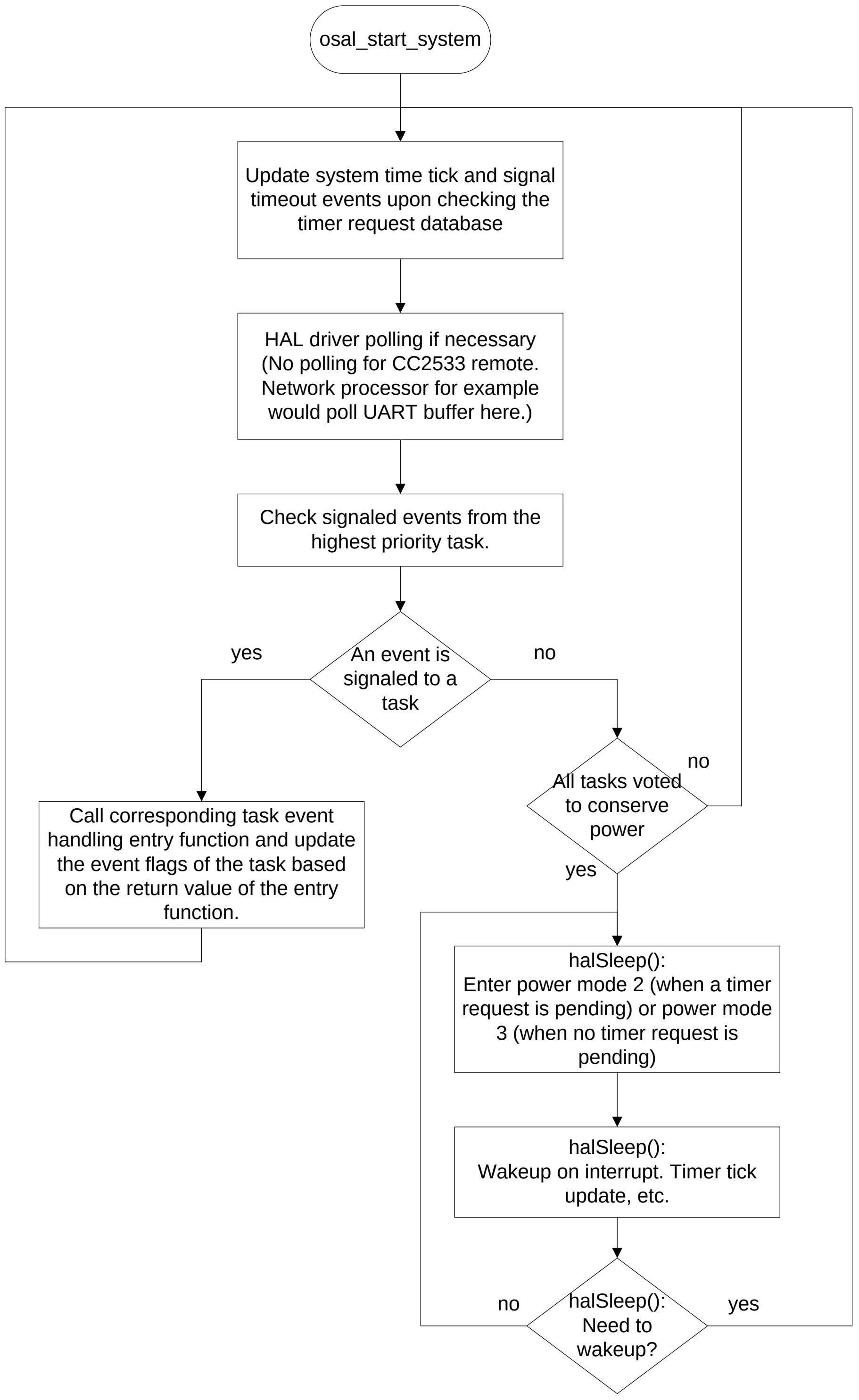

osal_start_system
Update system time tick and signal timeout events upon checking the timer request database
HAL driver polling if necessary (No polling for CC2533 remote. Network processor for example would poll UART buffer here.)
Check signaled events from the highest priority task.
yes
An event is signaled to a task
no
Call corresponding task event handling entry function and update the event flags of the task based on the return value of the entry function.
no
All tasks voted to conserve power
yes
halSleep(): Enter power mode 2 (when a timer request is pending) or power mode 3 (when no timer request is pending)
halSleep(): Wakeup on interrupt. Timer tick update, etc.
no
halSleep(): Need to wakeup?
yes

Figure 4 – 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 15. 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.

7. Flash page map and memory map

Each configuration of basic remote controller sample application project has a unique flash page map. Figure 5 illustrates two distinctive flash page maps used by basic remote sample application. One flash page is 1024 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 15.

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 13. The remainder of this last flash page can be used for additional code if the code fills up the reset of the space.

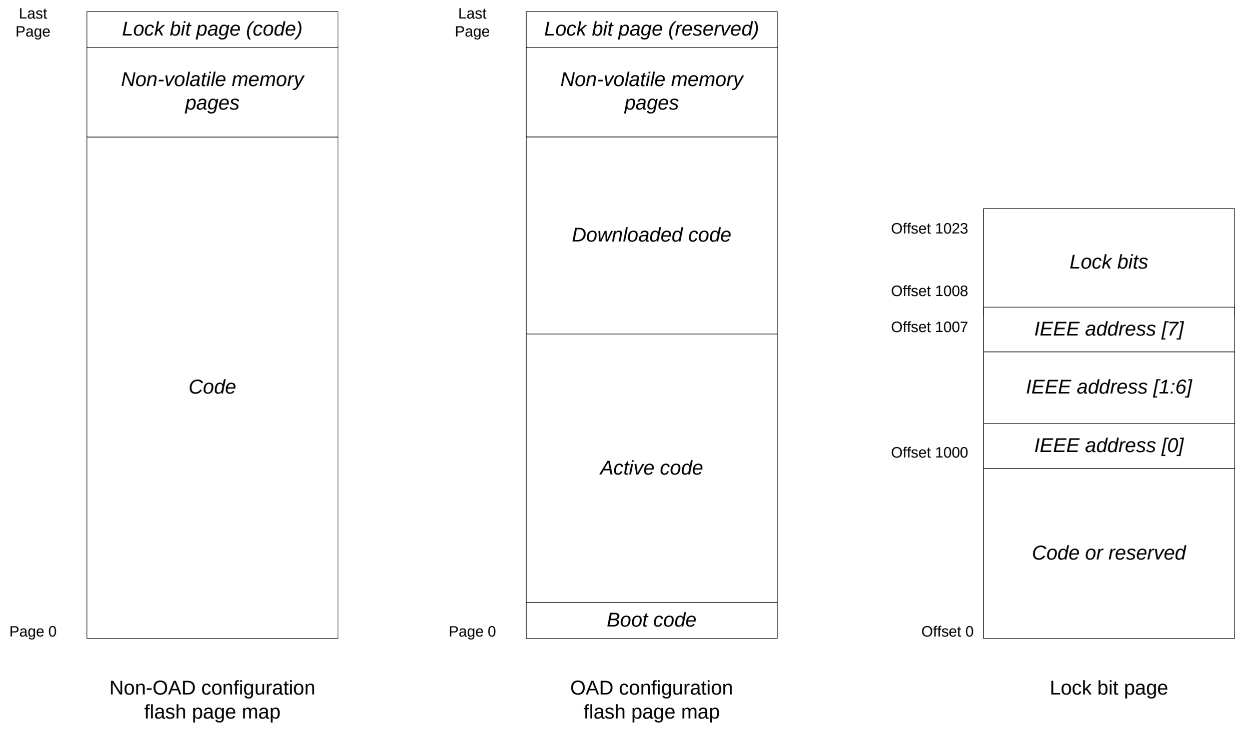


Figure 5 – Flash page map

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 in Table 7.

Table 7 – NV configuration constants

Constant Name	Description	CC2533F64	CC2533F96
HAL_NV_PAGE_END	Last OSAL NV page plus one	31 (62 for CC2533F64 not configured for 2K virtual page size)	47 (94 for CC2533F96 not configured for 2K virtual page size)
HAL_NV_PAGE_CNT	Number of OSAL NV pages	2	2

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, change hal_board_cfg.h file as follows:

```
...
#define HAL_NV_PAGE_END              HAL_FLASH_IEEE_PAGE
#define HAL_NV_PAGE_CNT              4
#define HAL_NV_PAGE_BEG             (HAL_NV_PAGE_END-HAL_NV_PAGE_CNT)
...
```

The linker command file can be located from project option pop up window. For instance after selecting CC2533F96 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 6.

In the linker command file, find -D_FLASH_BEG 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
...
```

If you want four pages for non-volatile memory instead, the non-volatile memory page should be located at 22nd page of the last bank (32 - 2 - 8), and the address should be 0x28000 + (0x400 * (22 - 1)) = 0x2D800. 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_FLASH_BEG=0x2D800
...
```

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

CC2533F64 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.

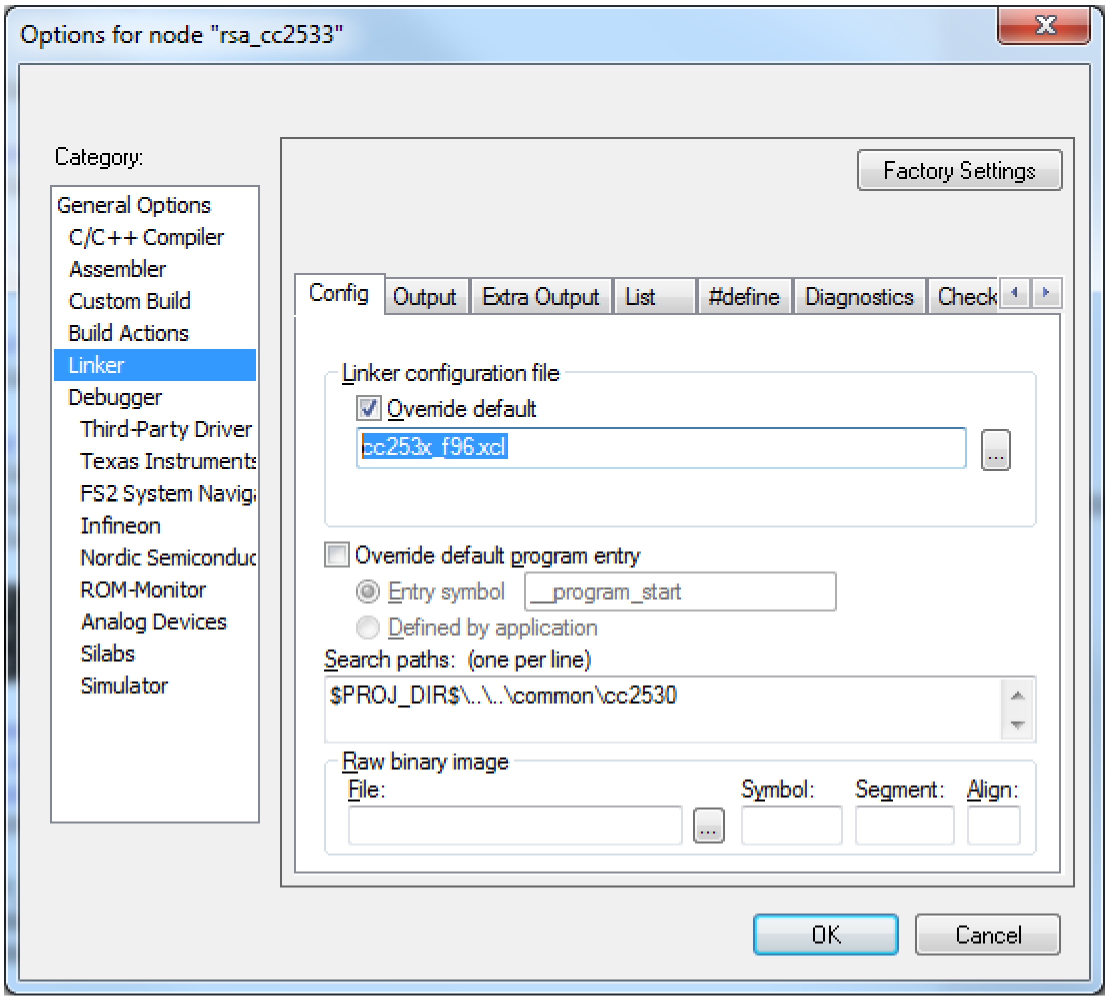
The CC2533F96 configuration use banked code model and bank area is dynamically mapped to flash bank (comprised of 16 pages) in use. Code address space is represented in virtual code address. Virtual address for code bank is listed in Table 8.

Table 8 – Code bank virtual address

Code Bank	Bank 0	Bank 1	Bank 2
Address Range	0x00000 – 0x07FFF	0x18000 – 0x1FFFF	0x28000 – 0x2FFFF

Bank 0 is constantly mapped to common area (0x0000 – 0x7FFF) and the other banks are mapped to bank area (0x8000 – 0xFFFF) dynamically.

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.



Linker configuration file

[✓] Override default

cc253x_f96.xcl

[ ] Override default program entry

Entry symbol __program_start

Defined by application

Search paths: (one per line)

$PROJ_DIR$\..\..\common\cc2530

Raw binary image

File: Symbol: Segment: Align:

Figure 6 – Basic remote linker configuration option window

The CC2533 has a physical flash page size of 1024. 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 needs to 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:

```
        //efine HAL_FLASH_PAGE_SIZE        1024
        #define HAL_FLASH_PAGE_SIZE        2048
```

If building for non-OAD configuration, reduce the available space for OSAL NV by changing _FLASH_BEG and _FLASH_END here for the F64:

```
      // Address range for HAL_FLASH_PAGE_SIZE == 1024
      //-D_FLASH_BEG=0xF400
      //-D_FLASH_END=0xFBFF
      //
      // Address range for HAL_FLASH_PAGE_SIZE == 2048
      -D_FLASH_BEG=0xE800
      -D_FLASH_END=0xF7FF
```

Or here for the F96:

```
      // Address range for HAL_FLASH_PAGE_SIZE == 1024
      //-D_FLASH_BEG=0x2F400
      //-D_FLASH_END=0x2FBFF
      //
      // Address range for HAL_FLASH_PAGE_SIZE == 2048
      -D_FLASH_BEG=0x2E800
      -D_FLASH_END=0x2F7FF
```

If building for OAD, increase the available code space in the linker file by changing _CODE_END here for the F64:

```
      //_CODE_END=0xF3FF  // Address range for HAL_FLASH_PAGE_SIZE == 1024
      -D_CODE_END=0xE7FF  // Address range for HAL_FLASH_PAGE_SIZE == 2048
```

Or here for the F96:

```
      //_CODE_END=0xBBFF  // Address range for HAL_FLASH_PAGE_SIZE == 1024
      -D_CODE_END=0xB7FF  // Address range for HAL_FLASH_PAGE_SIZE == 2048
```

The space allocated for NV does not have to be a segment for OAD configuration. This is because the lock-bit page is considered reserved for OAD configuration, and never considered part of code space. Hence for OAD there is no potential conflict for NV and code space. This is also why the _nvBuf constant, defined in osal_snv.c, is not required for an OAD configuration.

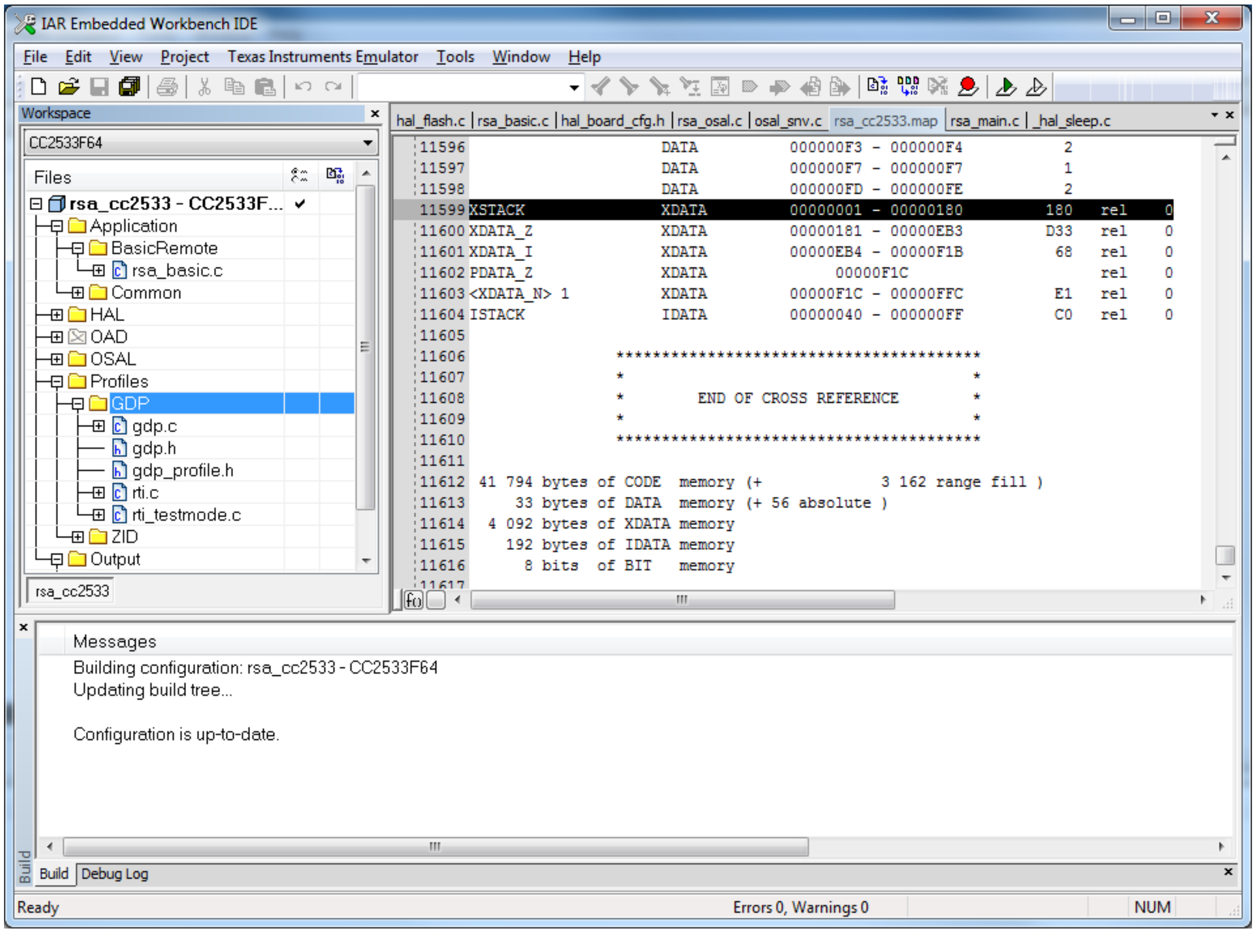
8. Stack and Heap

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). CC2533 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 8.10, RemoTI CC2533 development kit 1.3 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 7.



```
                             DATA          000000F3 - 000000F4        2
                             DATA          000000F7 - 000000F7        1
                             DATA          000000FD - 000000FE        2
XSTACK                       XDATA         00000001 - 00000180      180    rel     0
XDATA_Z                      XDATA         00000181 - 00000EB3      D33    rel     0
XDATA_I                      XDATA         00000EB4 - 00000F1B       68    rel     0
PDATA_Z                      XDATA              00000F1C                   rel     0
<XDATA_N> 1                  XDATA         00000F1C - 00000FFC       E1    rel     0
ISTACK                       IDATA         00000040 - 000000FF       C0    rel     0

                   *****************************************
                   *                                       *
                   *        END OF CROSS REFERENCE         *
                   *                                       *
                   *****************************************

 41 794 bytes of CODE  memory (+             3 162 range fill )
     33 bytes of DATA  memory (+ 56 absolute )
  4 092 bytes of XDATA memory
    192 bytes of IDATA memory
      8 bits  of BIT   memory
```

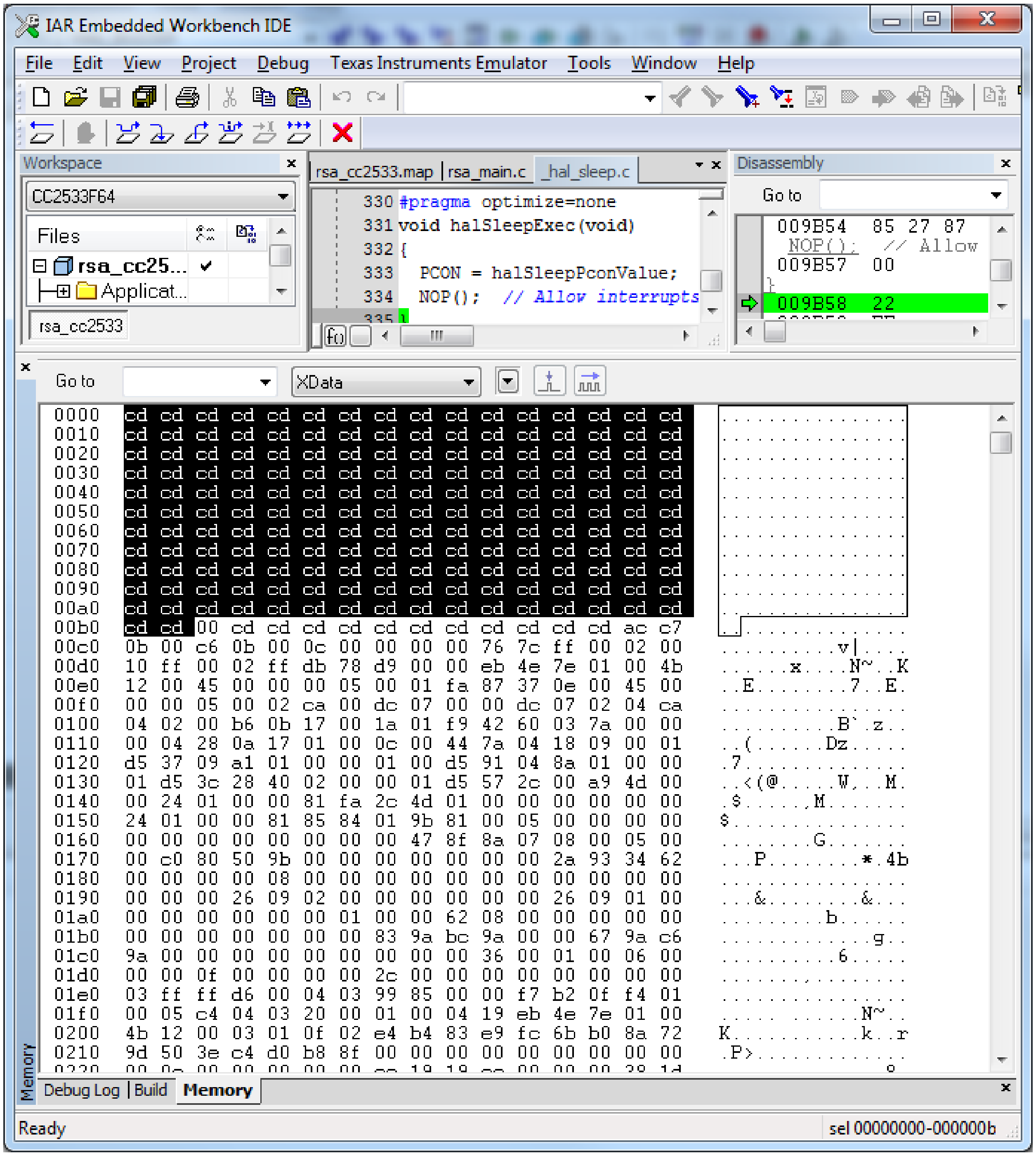
```
Messages
Building configuration: rsa_cc2533 - CC2533F64
Updating build tree...

Configuration is up-to-date.
```

Figure 7 – 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 8, XDATA stack was used down to 0x0B1, which makes the stack depth in this use case to be 0x180 – 0x0B1 + 1 = 207 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).



```
#pragma optimize=none
void halSleepExec(void)
{
  PCON = halSleepPconValue;
  NOP();  // Allow interrupts
```

```
009B54   85 27 87
  NOP();    // Allow
009B57   00
}
009B58   22
```

```
0000  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
0010  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
0020  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
0030  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
0040  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
0050  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
0060  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
0070  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
0080  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
0090  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
00a0  cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd   ................
00b0  cd cd 00 cd cd cd cd cd cd cd cd cd cd cd ac c7   ................
00c0  0b 00 c6 0b 00 0c 00 00 00 00 76 7c ff 00 02 00   ..........v|....
00d0  10 ff 00 02 ff db 78 d9 00 00 eb 4e 7e 01 00 4b   ......x....N~..K
00e0  12 00 45 00 00 00 05 00 01 fa 87 37 0e 00 45 00   ..E........7..E.
00f0  00 00 05 00 02 ca 00 dc 07 00 00 dc 07 02 04 ca   ................
0100  04 02 00 b6 0b 17 00 1a 01 f9 42 60 03 7a 00 00   ..........B`.z..
0110  00 04 28 0a 17 01 00 0c 00 44 7a 04 18 09 00 01   ..(......Dz.....
0120  d5 37 09 a1 01 00 00 01 00 d5 91 04 8a 01 00 00   .7..............
0130  01 d5 3c 28 40 02 00 00 01 d5 57 2c 00 a9 4d 00   ..<(@.....W,..M.
0140  00 24 01 00 00 81 fa 2c 4d 01 00 00 00 00 00 00   .$.....,M.......
0150  24 01 00 00 81 85 84 01 9b 81 00 05 00 00 00 00   $...............
0160  00 00 00 00 00 00 00 00 47 8f 8a 07 08 00 05 00   ........G.......
0170  00 c0 80 50 9b 00 00 00 00 00 00 00 2a 93 34 62   ...P........*.4b
0180  00 00 00 00 08 00 00 00 00 00 00 00 00 00 00 00   ................
0190  00 00 00 26 09 02 00 00 00 00 00 00 26 09 01 00   ...&........&...
01a0  00 00 00 00 00 00 01 00 00 62 08 00 00 00 00 00   .........b......
01b0  00 00 00 00 00 00 00 83 9a bc 9a 00 00 67 9a c6   .............g..
01c0  9a 00 00 00 00 00 00 00 00 00 36 00 01 00 06 00   ..........6.....
01d0  00 00 0f 00 00 00 00 2c 00 00 00 00 00 00 00 00   .......,........
01e0  03 ff ff d6 00 04 03 99 85 00 00 f7 b2 0f f4 01   ................
01f0  00 05 c4 04 03 20 00 01 00 04 19 eb 4e 7e 01 00   ............N~..
0200  4b 12 00 03 01 0f 02 e4 b4 83 e9 fc 6b b0 8a 72   K...........k..r
0210  9d 50 3e c4 d0 b8 8f 00 00 00 00 00 00 00 00 00   .P>.............
```

Figure 8 – 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:

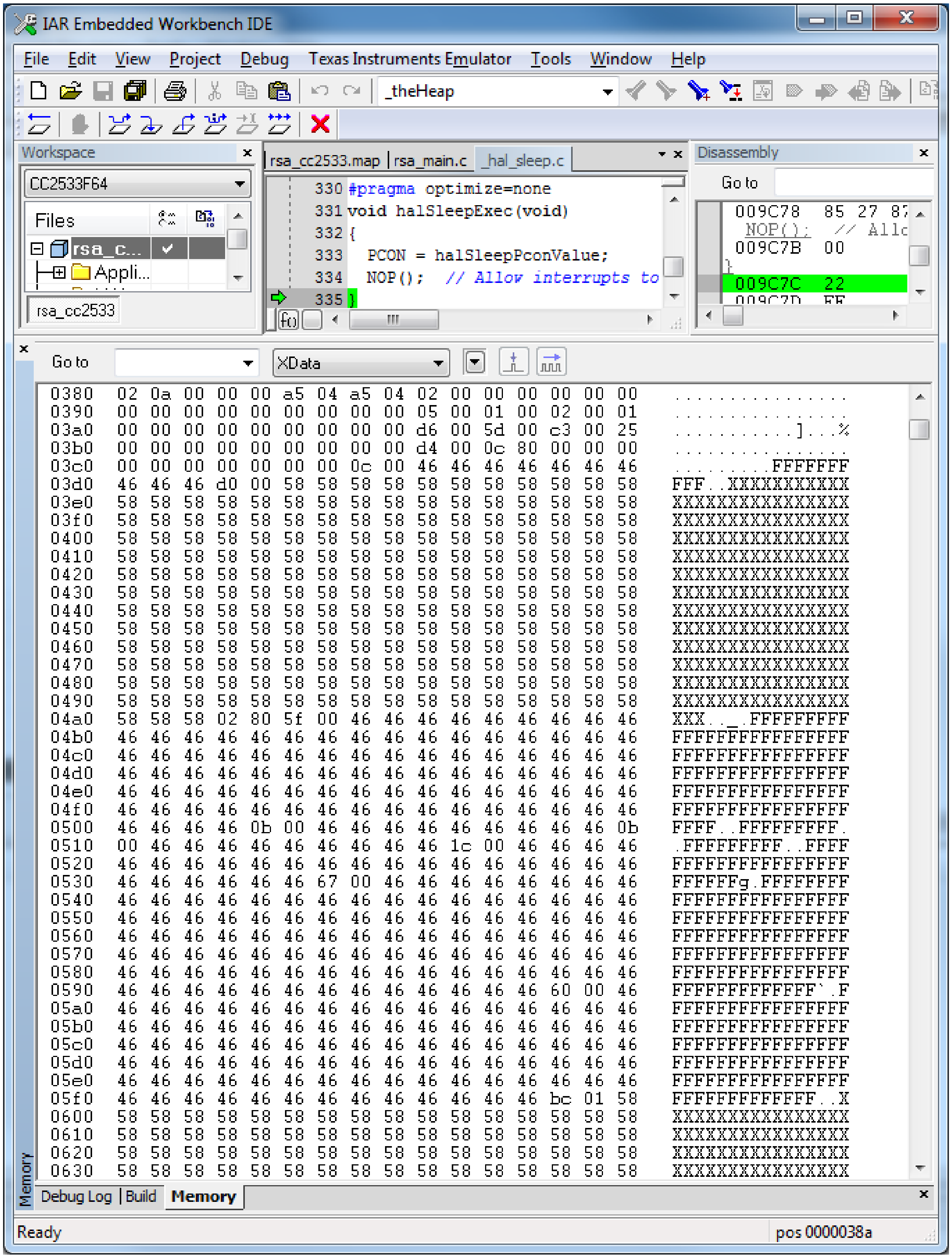
```
void osal_mem_init( void )
{
  osalMemHdr_t *tmp;

#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.

```
#if ( OSALMEM_PROFILER )
  #define OSALMEM_INIT   'X'
  #define OSALMEM_ALOC   'A'
  #define OSALMEM_REIN   'F'
#endif
```

OSALMEM_PROFILER compile flag brings in more code than the heap initialization, which is not explained in this document.

With OSALMEM_PROFILER defined, we build a 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.



```
#pragma optimize=none
void halSleepExec(void)
{
  PCON = halSleepPconValue;
  NOP();  // Allow interrupts to
}
```

```
009C78   85 27 87
 NOP();   // Allc
009C7B   00
}
009C7C   22
009C7D   FF
```

```
0380   02 0a 00 00 00 a5 04 a5 04 02 00 00 00 00 00 00   ................
0390   00 00 00 00 00 00 00 00 00 05 00 01 00 02 00 01   ................
03a0   00 00 00 00 00 00 00 00 00 d6 00 5d 00 c3 00 25   ...........]...%
03b0   00 00 00 00 00 00 00 00 00 d4 00 0c 80 00 00 00   ................
03c0   00 00 00 00 00 00 00 0c 00 46 46 46 46 46 46 46   .........FFFFFFF
03d0   46 46 46 d0 00 58 58 58 58 58 58 58 58 58 58 58   FFF..XXXXXXXXXXX
03e0   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
03f0   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0400   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0410   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0420   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0430   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0440   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0450   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0460   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0470   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0480   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0490   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
04a0   58 58 58 02 80 5f 00 46 46 46 46 46 46 46 46 46   XXX.._.FFFFFFFFF
04b0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
04c0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
04d0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
04e0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
04f0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
0500   46 46 46 46 0b 00 46 46 46 46 46 46 46 46 46 0b   FFFF..FFFFFFFFF.
0510   00 46 46 46 46 46 46 46 46 46 1c 00 46 46 46 46   .FFFFFFFFF..FFFF
0520   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
0530   46 46 46 46 46 46 67 00 46 46 46 46 46 46 46 46   FFFFFFg.FFFFFFFF
0540   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
0550   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
0560   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
0570   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
0580   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
0590   46 46 46 46 46 46 46 46 46 46 46 46 46 60 00 46   FFFFFFFFFFFFF`.F
05a0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
05b0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
05c0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
05d0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
05e0   46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46   FFFFFFFFFFFFFFFF
05f0   46 46 46 46 46 46 46 46 46 46 46 46 46 bc 01 58   FFFFFFFFFFFFF..X
0600   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0610   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0620   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
0630   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58   XXXXXXXXXXXXXXXX
```

pos 0000038a

Figure 9 – Heap usage profiling

If the *_theHeap* variable occupies 0x38A to 0xB89 address space for example, search from 0xB89 down to bottom for any foot print of memory usage. *_theHeap* is split in a space for small blocks and a space for large blocks. 232 bytes are allocated for small blocks, and the rest for large blocks. Look for

```
#define SMALLBLKHEAP    232
```

In Figure 9, 0x5FF is the highest address of memory space that was used in the heap space, for large blocks, and 0x3D4 for small blocks. That amounts to 0x5FF - 0x4A3 + 1 = 348 bytes of heap usage in the large block space. For small blocks; 0x3DF – 0x38A = 85 bytes were used.

Once heap size is profiled, the heap size can be adjusted by changing the INT_HEAP_LEN definition in the compile option. For instance changing INT_HEAP_LEN=768 in defined symbols window of Preprocessor tab of C/C++ Compiler category, adjusts heap size to 768 bytes.

9. 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 9 shows difference of features supported by the two sets of libraries.

Table 9 – Network layer library features

Network layer features	rcnctrl	rcnsuper
Discovery originator	√	√
Discovery recipient		√
Pairing originator	√	√
Pairing recipient		√
Un-pair originator	√	√
Un-pair recipient	√	√

Network layer features	rcnctrl	rcnsuper
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.

10. 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 ZRC 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 ZRC 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 ZRC 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 ZRC 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 ZRC 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).

11. Target device selection

As described in chapter 10, 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.

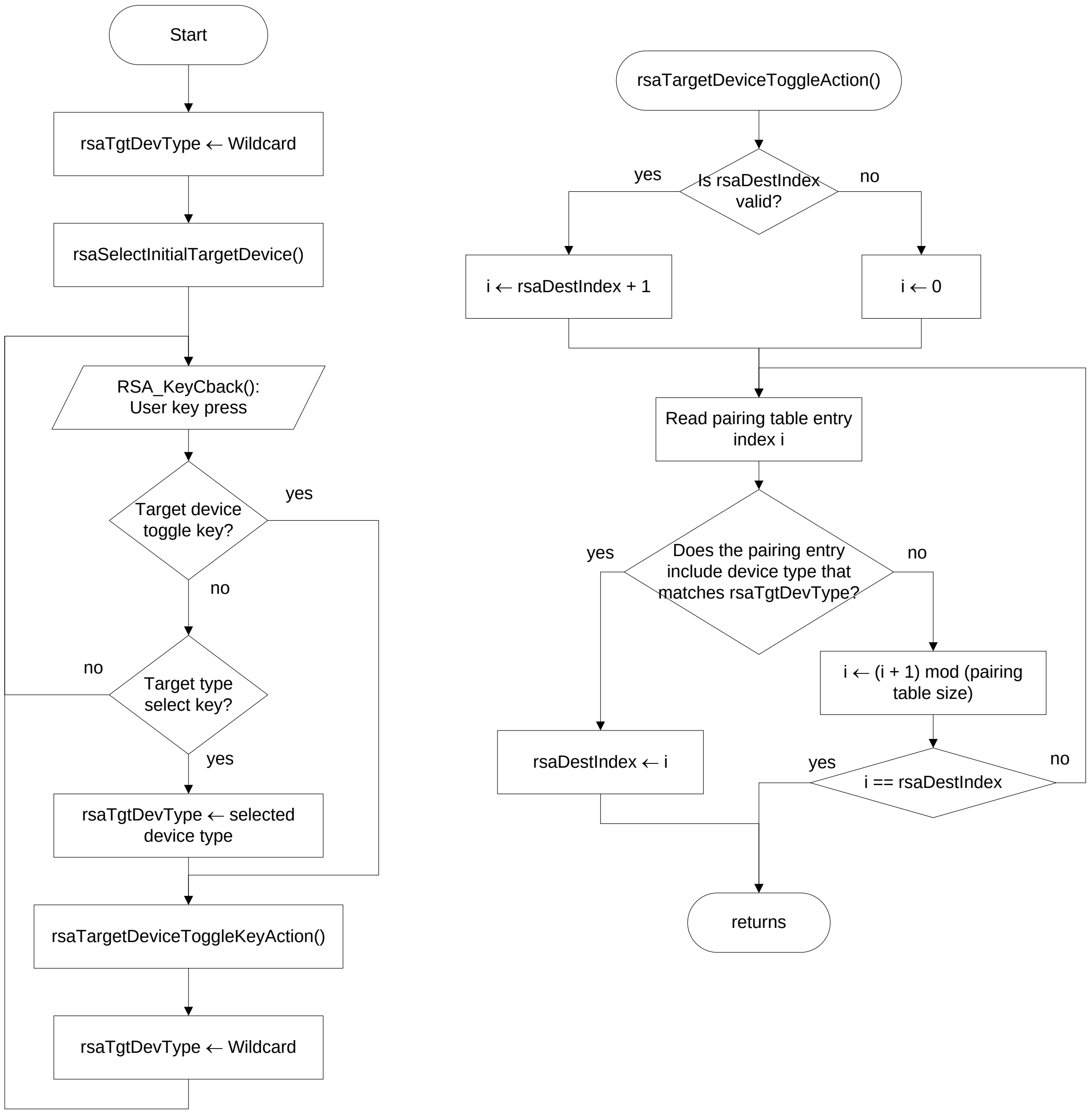

Start
rsaTgtDevType ← Wildcard
rsaSelectInitialTargetDevice()
RSA_KeyCback(): User key press
Target device toggle key?
yes
no
Target type select key?
no
yes
rsaTgtDevType ← selected device type
rsaTargetDeviceToggleKeyAction()
rsaTgtDevType ← Wildcard
rsaTargetDeviceToggleAction()
Is rsaDestIndex valid?
yes
no
i ← rsaDestIndex + 1
i ← 0
Read pairing table entry index i
Does the pairing entry include device type that matches rsaTgtDevType?
yes
no
i ← (i + 1) mod (pairing table size)
rsaDestIndex ← i
i == rsaDestIndex
yes
no
returns

Figure 10 – Destination Target Selection Algorithm

Figure 10 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.

12. 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 through Simple Non-Volatile Memory API if necessary. See [4] for API details. Note that RemoTI network layer uses OSAL Simple Non-Volatile Memory API instead of regular OSAL Non-Volatile Memory API and the Simple Non-Volatile Memory API provides potential for better code size optimization and it is supported for all configurations. Hence it is recommended that application also uses the same Simple Non-Volatile Memory API so that the non-volatile memory module dependency is universal, unless the application requires more items than can be supported by the Simple Non-Volatile Memory API.

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 10 shows the NV item identifiers reserved by network layer and RemoTI application framework.

Table 10 – NV item identifier ranges

Reserved NV item identifier range	Description
0x00	NULL identifier
0x01 – 0x6F	Reserved for RemoTI network layer
0x70 – 0x7F	Reserved for RemoTI application framework (Search for *RTI_NVID_xxx* constants in *rti.c* file to find exact identifiers used)
0x80 – 0xFE	Application
0xFF	Reserved

Please note that, as of RemoTI 1.2, the network layer migrated to Simple NV API, hence it is not compatible with RemoTI 1.1 NV image. That is, if a RemoTI-1.1 image is updated with RemoTI-1.2, or later, image without clearing the NV pages, RemoTI-1.2, or later, image will not work correctly.

To support the updated RemoTI-1.2, or later, image to clear RemoTI-1.1 NV pages, sample applications from releases later than, and including, RemoTI-1.2 looks for RemoTI-1.1 NV pages and erases them by default. Note that updated RemoTI-1.2, or later, image erases RemoTI-1.1 non-volatile memory and consequently, for instance, when an over-the-air download is performed to update a remote controller that used to have RemoTI 1.1 network layer, all the pairing information is cleared and the user has to pair the remote again.

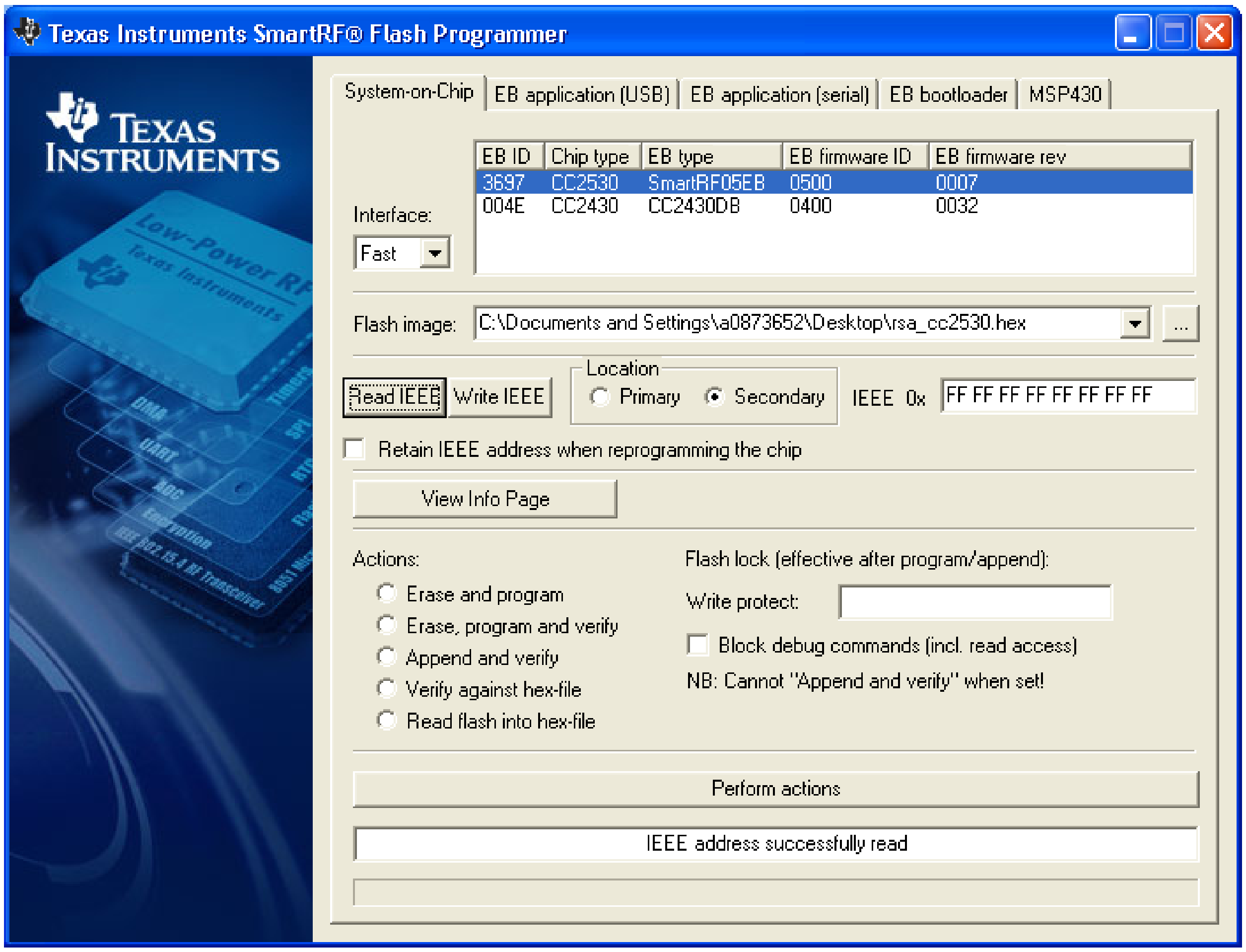
If you do not want such a behavior which increases RemoTI-1.2, or later, downloadable image size, exclude both “nvmigr_nv2snv.c” file and “nvmigr_nv2snvw.c” file from the build and change either “$PROJ_DIR$\..\..\COMMON\NV\NV2SNV” or “$PROJ_DIR$\..\..\COMMON\NV\NV2SNVW” of the Preprocessor additional include directories settings (of the project options) to

“$PROJ_DIR$\..\..\COMMON\NV\NONMIGR”. See the project options for CC2533_SB configuration of RNP project for not having the RemoTI-1.1 NV image clearing. CC2533 serial downloading image was not included as part of RemoTI-1.1 and hence the RemoTI-1.2, or later, CC2533 serial downloading image is a good example of an image without RemoTI-1.1 NV page clearing.

13. IEEE address

CC2533 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.



EB ID	Chip type	EB type	EB firmware ID	EB firmware rev
3697	CC2530	SmartRF05EB	0500	0007
004E	CC2430	CC2430DB	0400	0032

Interface: Fast

Flash image: C:\Documents and Settings\a0873652\Desktop\rsa_cc2530.hex

Location: Primary [✓ Secondary]

IEEE 0x FF FF FF FF FF FF FF FF

Retain IEEE address when reprogramming the chip

Actions:
- Erase and program
- Erase, program and verify
- Append and verify
- Verify against hex-file
- Read flash into hex-file

Flash lock (effective after program/append):

Write protect:

Block debug commands (incl. read access)

NB: Cannot "Append and verify" when set!

IEEE address successfully read

Figure 11 – 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 12 illustrates the flow chart of selecting the network layer IEEE address, during startup of a device.

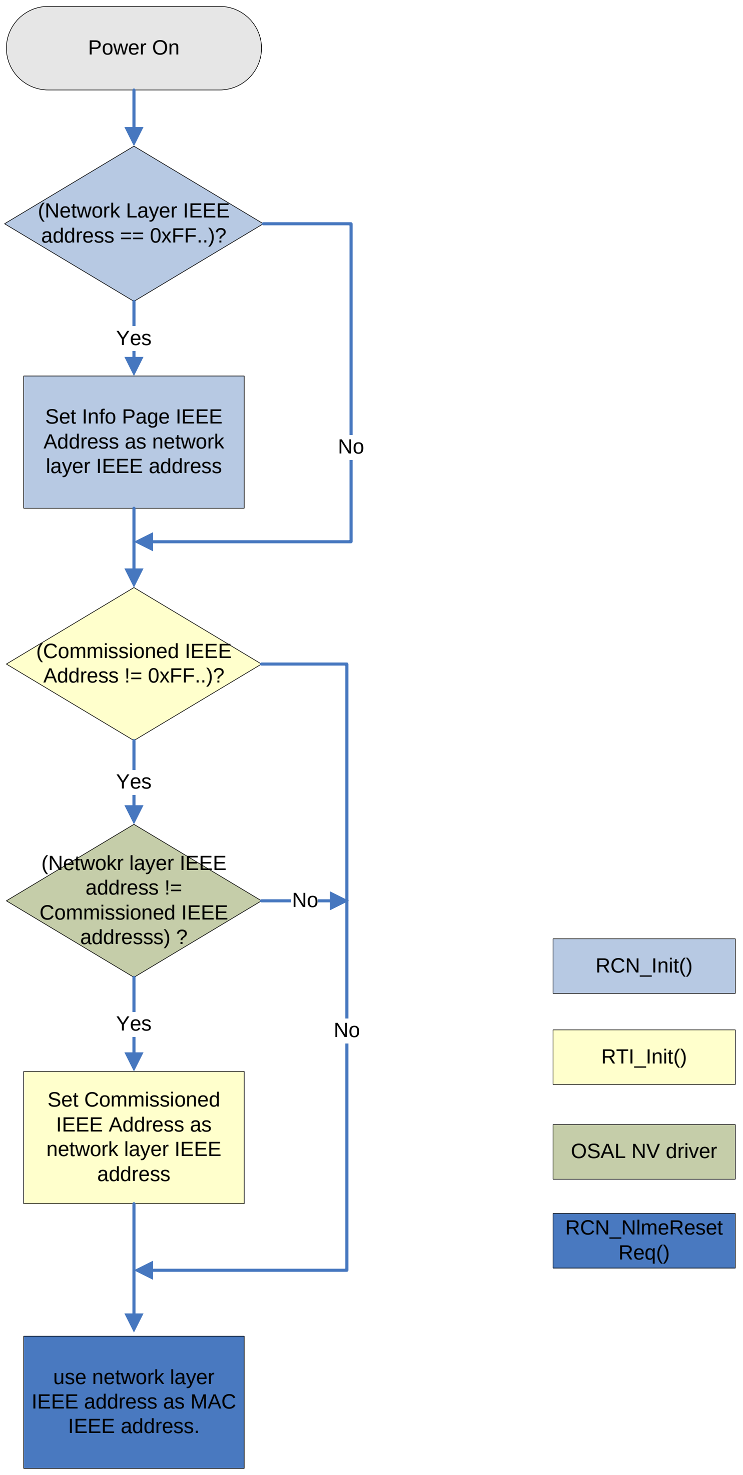

Power On
(Network Layer IEEE address == 0xFF..)?
Yes
No
Set Info Page IEEE Address as network layer IEEE address
(Commissioned IEEE Address != 0xFF..)?
Yes
No
(Netwokr layer IEEE address != Commissioned IEEE addresss) ?
No
Yes
Set Commissioned IEEE Address as network layer IEEE address
use network layer IEEE address as MAC IEEE address.
RCN_Init()
RTI_Init()
OSAL NV driver
RCN_NlmeReset Req()

Figure 12 – IEEE address selection flow during startup

14. 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 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_WriteItemEx* or *RCN_NlmeSetReq*() are enumerated in rcn_attribs.h file. Note that several non-standard attributes are also provided.

Table 11 explains the non-standard attributes.

Table 11 – 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 address. 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 13.
RCN_NIB_AGILITY_ENABLE	Enable/disable frequency agility

Attribute identifier	Description
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.

15. Over the air download

15.1 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 instructions.

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 15.2 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 7). 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.

15.2 Over the air download command packet formats

15.2.1 Poll protocol frame format

Poll protocol shall a single frame type to be called Poll frame, and its format is illustrated in Figure 13. Protocol payload size shall be zero.

Bits : 8
Protocol Identifier

Figure 13 – Poll frame format

15.2.2 General Over the Air Download protocol frame format

General Over the Air Download protocol frame format is illustrated in Figure 14.
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 14 – General Over the Air Download protocol frame Format

15.2.3 Over the Air Download protocol command identifiers

All valid command identifiers are listed in Table 12.

Table 12 – 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

15.2.4 Status Request command

Status Request command has zero length command payload.

15.2.5 Start Request command

Start Request command payload format is illustrated in Figure 15.

Bits : 16
Image Size

Figure 15 – Start Request command frame format

15.2.6 Data Request command

Data Request command payload format is illustrated in Figure 16.

Bits : 16	16	512
Sequence Number	Address	Data

Figure 16 – Data Request command frame format

15.2.7 Enable Request command

Enable Request command payload format is illustrated in Figure 17.

Bits : 32	32
Image ID	CRC

Figure 17 – Enable Request command frame format

15.2.8 Cancel Request command

Cancel Request command has zero length command payload.

15.2.9 Status Response command

Status Response command payload format is illustrated in Figure 18.

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

Figure 18 – Status Response command frame format

Valid OAD State field values are listed in Table 13.

Table 13 – OAD State Field Values

Value	State
0x00	Idle
0x01	Download
0x02	Validate

15.2.10 Start Response command

Start Response command payload format is illustrated in Figure 19.

Bits : 8
Status

Figure 19 – Start Response command frame format

Valid Status field values are listed in Table 14.

Table 14 – 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

15.2.11 Data Response command

Data Response command payload format is illustrated in Figure 20.

Bits : 8	16
Status	Sequence Number

Figure 20 – Data Response command frame format

Valid Status field values are listed in Table 14.

15.2.12 Enable Response command

Enable Response command payload format is illustrated in Figure 21.

Bits : 8
Status

Figure 21 – Enable Response command frame format

Valid Status field values are listed in Table 14.

15.2.13 Cancel Response command

Cancel Response command payload format is illustrated in Figure 22.

Bits : 8
Status

Figure 22 – Cancel Response command frame format

Valid Status field values are listed in Table 14.

15.3 Over the air download command flow sequence

Figure 23 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.

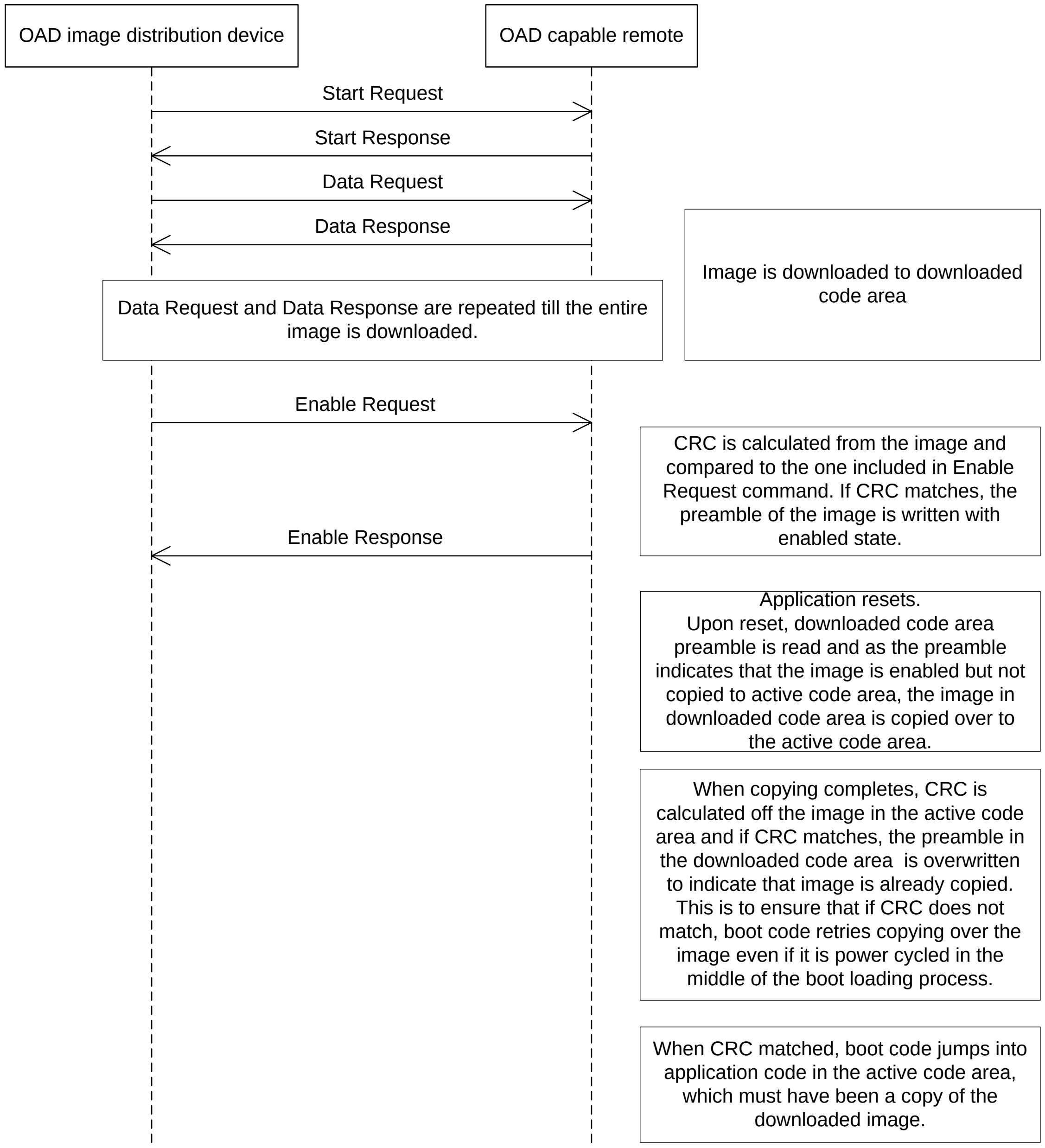

OAD image distribution device
OAD capable remote
Start Request
Start Response
Data Request
Data Response
Data Request and Data Response are repeated till the entire image is downloaded.
Image is downloaded to downloaded code area
Enable Request
CRC is calculated from the image and compared to the one included in Enable Request command. If CRC matches, the preamble of the image is written with enabled state.
Enable Response
Application resets. Upon reset, downloaded code area preamble is read and as the preamble indicates that the image is enabled but not copied to active code area, the image in downloaded code area is copied over to the active code area.
When copying completes, CRC is calculated off the image in the active code area and if CRC matches, the preamble in the downloaded code area is overwritten to indicate that image is already copied. This is to ensure that if CRC does not match, boot code retries copying over the image even if it is power cycled in the middle of the boot loading process.
When CRC matched, boot code jumps into application code in the active code area, which must have been a copy of the downloaded image.

Figure 23 – OAD command flow

15.4 Basic remote controller sample application configuration

Basic remote controller sample application includes over the air download demo feature in the CC2533F96_OAD configuration only.

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 (CC2533F96OAD. 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.

15.5 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.

15.6 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.

15.7 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.

16. 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 ZRC 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 24 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).

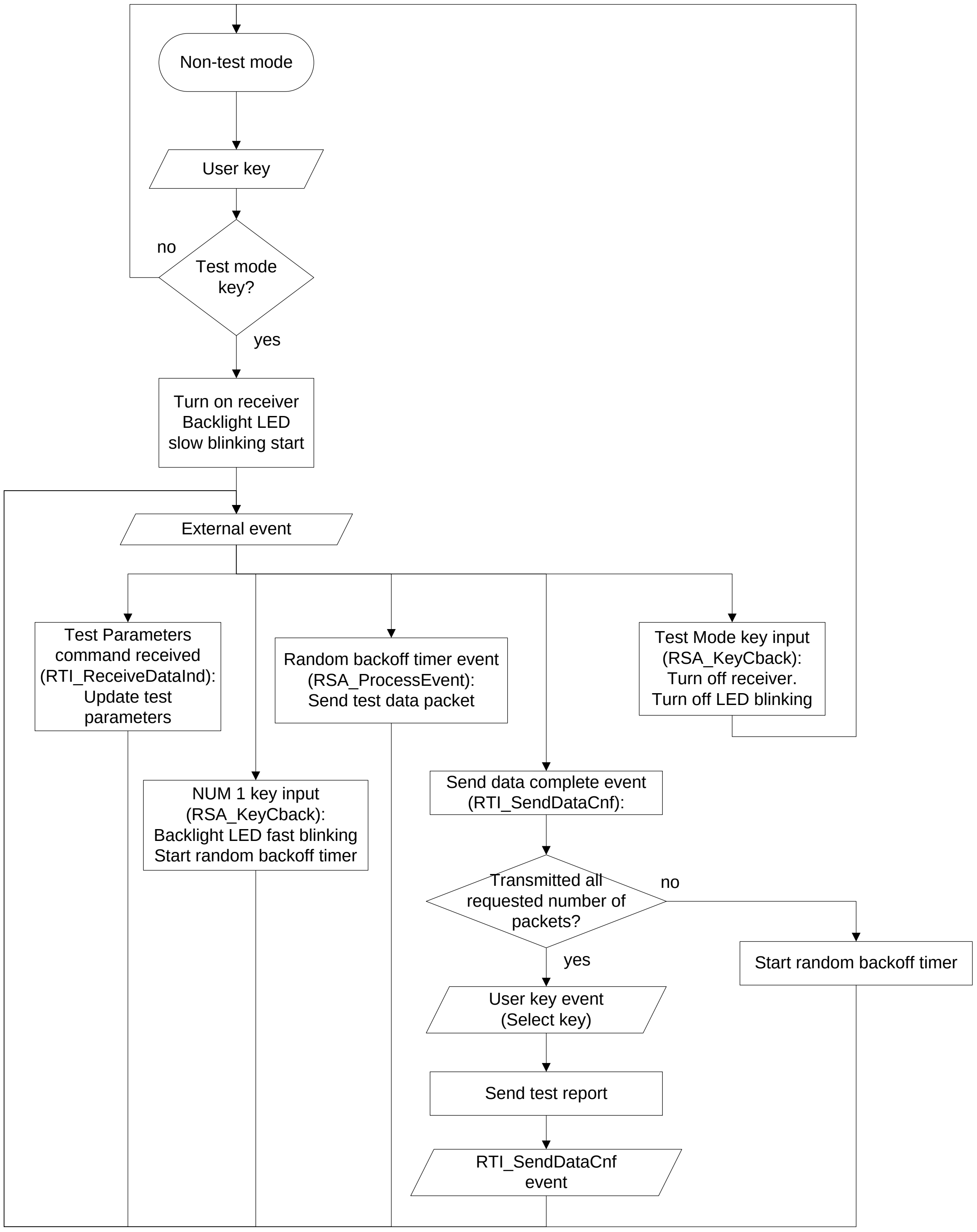

Non-test mode
User key
Test mode key?
no
yes
Turn on receiver Backlight LED slow blinking start
External event
Test Parameters command received (RTI_ReceiveDataInd): Update test parameters
Random backoff timer event (RSA_ProcessEvent): Send test data packet
Test Mode key input (RSA_KeyCback): Turn off receiver. Turn off LED blinking
NUM 1 key input (RSA_KeyCback): Backlight LED fast blinking Start random backoff timer
Send data complete event (RTI_SendDataCnf):
Transmitted all requested number of packets?
no
yes
Start random backoff timer
User key event (Select key)
Send test report
RTI_SendDataCnf event

Figure 24 – Latency Test Mode Algorithm

17. DMA, peripheral IO and timers

Basic remote sample application uses the following resources:

- Peripheral IO pin P1_0 for the buzzer
- Peripheral IO pins P0_0, P0_1, P0_2, P0_3, P0_4, and P1_4 for the key matrix.
- DMA channel 0 for non-volatile memory access
- Timer2 (MAC timer) and sleep timer.
- Timer 4 for buzzer operation

18. General Information

18.1 Document History

Table 15 – 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.
swru224b	2010-07-07	CC2533 related project configurations and files and Simplified Non-Volatile Memory management module were added to sections 3.2 and 3.3. Chapter 13 was modified to reflect stack migration to Simple Non-Volatile Memory management API.
swru224c	2012-05-09	Ported Basic Remote project to the Advanced Remote HW

19. 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

20. TI Worldwide Technical Support

Internet

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

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