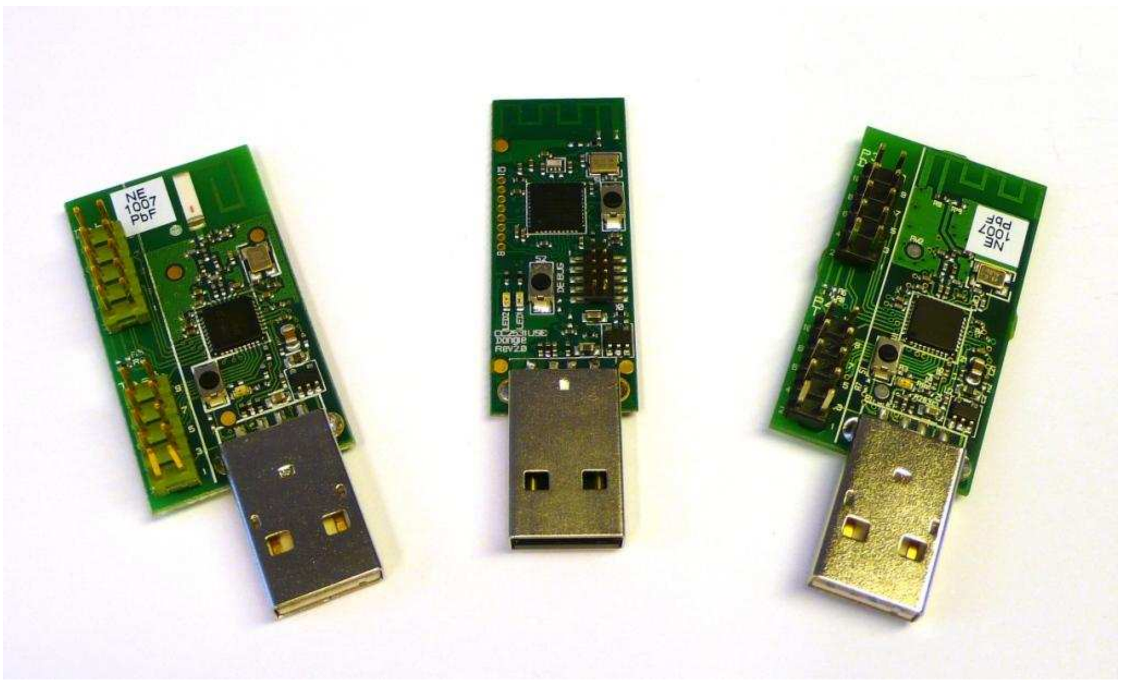
CC USB Software Examples User's Guide



swru222

Table of Contents

1 Introduction

This document describes the software examples for the CC1111, CC2511 and CC2531 RF USB dongles. The Human Interface Device (HID) application emulates mouse and keyboard using a SmartRF05EB or SmartRF04EB. The RF Modem example shows how the USB CDC class can be implemented to provide serial port emulation across USB (Virtual COM-port). Both examples use the MRFI protocol for radio communication. This document the software referred to in [5].

2 About this Manual

This manual covers the USB software examples and USB Firmware Library for the Texas Instruments Low-Power RF USB Dongles for the chipsets CC1111, CC2511 and CC2531.

3 Acronyms

ACM	-	Abstract Control Model
API	-	Application Programming Interface
CCA	-	Clear Channel Available
CDC	-	Communication Device Class
DCE	-	Distributed Computing Environment
DTE	-	Data Terminal Equipment
FCS	-	Frame Check Sequence
HAL	-	Hardware Abstraction Layer
HID	-	Human Interface Device
ICE	-	In Circuit Emulator
IO	-	Input/Output
PAN	-	Personal Area Network
MRFI	-	Minimal RF Interface
RF	-	Radio Frequency
RSSI	-	Received Signal Strength Indicator
SFD	-	Start of Frame Delimiter
SoC	-	System on Chip
USB	-	Universal Serial Bus

4 Getting Started

4.1 Preparations

The USB firmware described in this document may be debugged using any of the following In-Circuit Emulators:

- CC Debugger
- SmartRF05EB
- SmartRF04EB with SOC_DEM debug connector [1]

Please install SmartRF Studio **before** connecting the ICE to a PC. SmartRF Studio is a PC application for Windows that helps you find and adjust the radio register settings. Please refer to [3] for instructions on how to download and install SmartRF Studio. Installing SmartRF Studio prior to connecting the debugger ensures that all the relevant Windows drivers are installed.

Please see the relevant USB Hardware User's Guide ([7] or [8]) for detailed instructions on how to interconnect the PC, the ICE debugger and the target dongle.

When the drivers are installed and the debugger connected, the USB dongle can be programmed and debugged using IAR Embedded Workbench. Programming can also be done using the SmartRF Flash programmer [4].

4.2 General Guidelines

When writing and debugging USB firmware one should remember that when Windows (or another OS) detects a USB device it will assume that it works in accordance with the USB specification. During debugging this might not be the case. Breakpoints may stop the execution of code on the USB Dongle and firmware under development may not be completed or error free. This may lead to the PC disconnecting the USB device and turning off or toggling its power supply. In some cases Windows (or the OS in use) may crash or misbehave. Hence breakpoints should be used with caution.

Obviously inserting and using breakpoints in the firmware during debugging will be extremely useful to pinpoint bugs and follow code execution. And they can and should be used. But they will also stop the code execution on the target and may cause the device to act in conflict with the USB specification.

If the firmware contain bugs that make the PC disconnect the USB Dongle it may be necessary to disconnect the USB cable, and power the USB Dongle from the SmartRF04EB/SmartRF05EB in order to reprogram it. Please refer to the relevant **USB Hardware User's Guide** ([7] or [8]) for instructions on how to reprogram the dongle without it being powered by the PC.

4.3 Known Problems

When using the debug interface through the level-converter on the SmartRF04EB it might become unstable. This problem is more likely to occur at low voltages or with crosstalk in the cable at high interface speed.

Using the SOC_DEM instead of the P14 "SoC Debug/Flash" connector on SmartRF04EB is more stable when the voltage differs on the SmartRF04EB and the target board.

If problems are still experienced when using SOC_DEM, make sure that the 10 pin flat cable used to connect the RF USB Dongle to the SmartRF04EB is as short as possible.

Next, it might be necessary to reduce the debug interface speed in order to achieve stable operation.
In the IAR workbench do the following:

- Highlight the project name by clicking on it in the "Files" window (see Figure 1).
- Go to "Project" > "Options…"
- Select the Category "Texas Instruments"
- Click on the "Target" tab.

Check the “Reduce interface speed” option (see Figure 2).

Highlight the project name

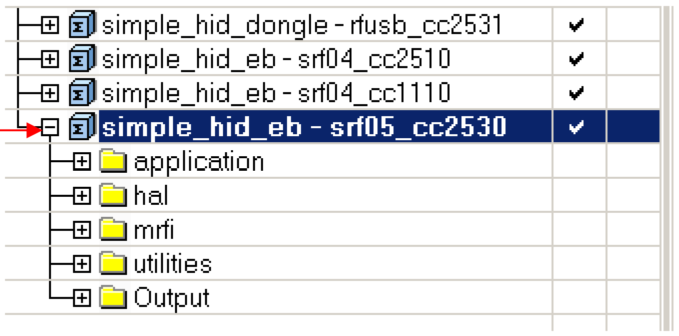
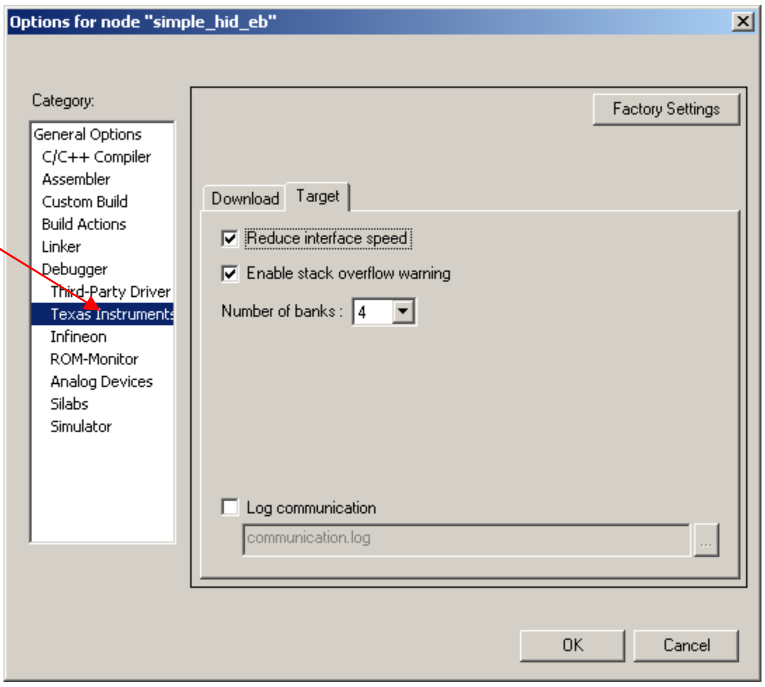

simple_hid_dongle - rfusb_cc2531
simple_hid_eb - srf04_cc2510
simple_hid_eb - srf04_cc1110
simple_hid_eb - srf05_cc2530
application
hal
mrfi
utilities
Output

Figure 1. Selecting Project

Check the “Reduce interface speed” option.



Options for node "simple_hid_eb"

Category: Texas Instruments

Target:
- [✓] Reduce interface speed
- [✓] Enable stack overflow warning
- Number of banks : 4
- [ ] Log communication
- communication.log

Figure 2. Reduce Interface Speed

5 USB Application Examples

This section describes the application examples available for the USB dongles. These application examples focus on how to use the USB part of the respective device. The examples are available both as source code and ready-to-upload .hex files from [5]. There are two USB application examples.

Application example name	Description	USB class demonstrated
"simple_hid"	Wireless mouse / keyboard	HID (Human Interface Device)
"rf_modem"	Wireless USB to serial port converter	CDC-ACM (Communication Device Class, with Abstract Control Model subclass)

Table 1. Overview of application examples

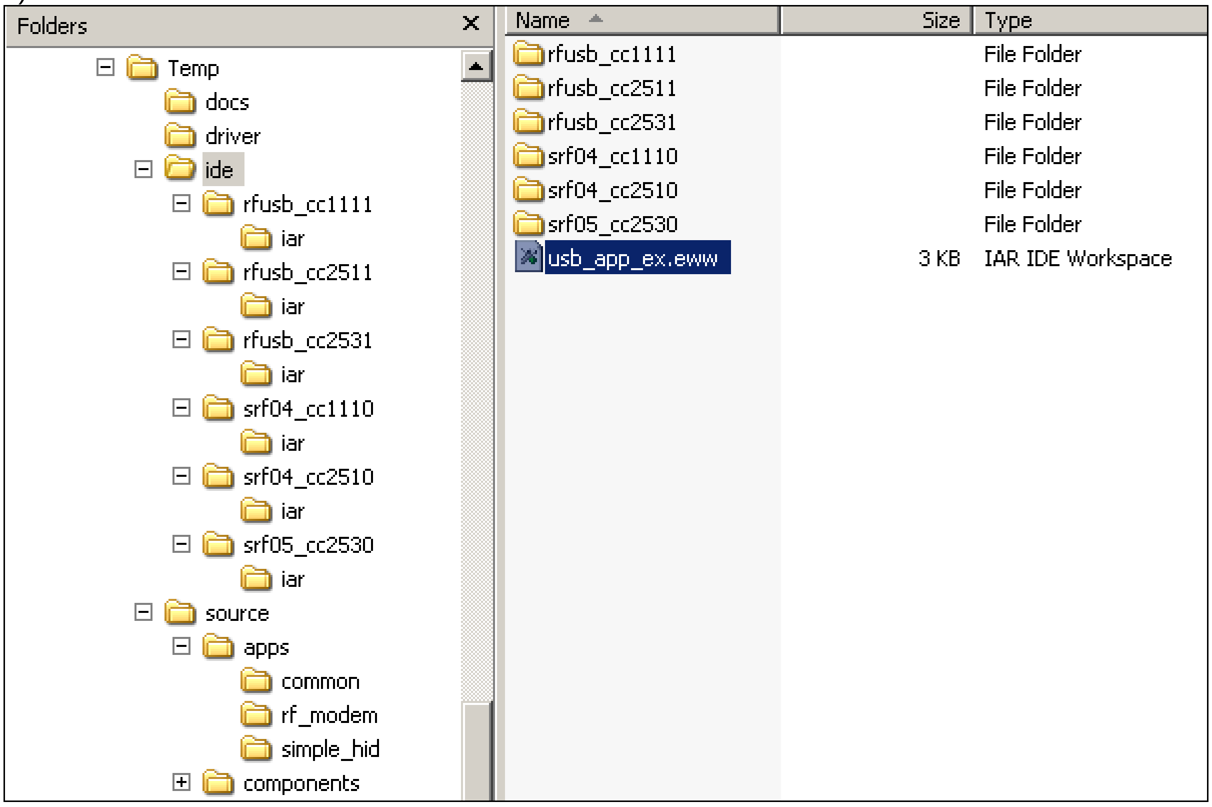
Working together with the USB examples are peer applications running on CC1110EM, CC2510EM and CC2530EM respectively. The CC1110EM and CC2510EM must be plugged into a SmartRF04EB. The CC2530EM must be plugged into a SmartRF05EB.

5.1 Building and Running the Examples

Note. The IAR projects contained in this package requires IAR EW8051 v7.51 or above. If a newer version than v7.51 is used, IAR will automatically convert the project files to the correct format.

First of all make sure that the dongle is connected to the ICE as described the "Programming" section of the relevant **USB Dongle User's Guide** (please see [7] for programming the CC1111/CC2511), or [8] for programming the CC2531).

When building the examples from source with IAR EW8051, make sure you select the project corresponding to the target you are developing your software for. Please see Figure 5 for an overview of the projects found the in the USB application example package. The IAR workspace is located in the sub-folder 'ide' just below the root of the installation. The file is named 'usb_app_ex.eww' (Figure 3).



Name	Size	Type
rfusb_cc1111		File Folder
rfusb_cc2511		File Folder
rfusb_cc2531		File Folder
srf04_cc1110		File Folder
srf04_cc2510		File Folder
srf05_cc2530		File Folder
usb_app_ex.eww	3 KB	IAR IDE Workspace

Figure 3. Directory Structure

After opening the IAR workspace all the examples can be built in one operation by hitting the F8 key. Downloading to the target can be done using the keyboard shortcut ‘CTRL+D’, alternatively ‘Project->Debug’ as shown in Figure 4.

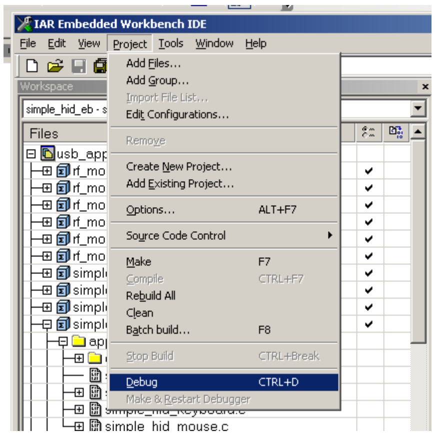
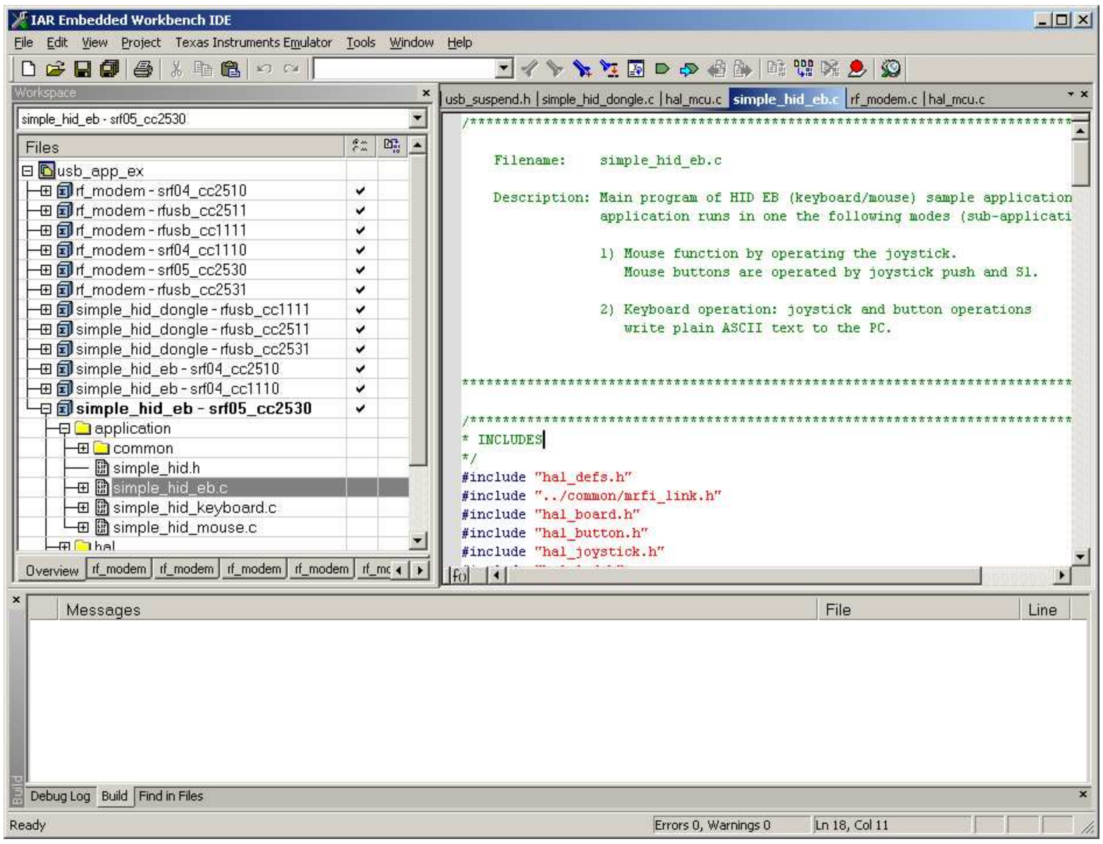


Figure 4. Launching the Debugger


IAR Embedded Workbench IDE
File Edit View Project Texas Instruments Emulator Tools Window Help
Workspace
simple_hid_eb - srf05_cc2530
Files
usb_app_ex
rf_modem - srf04_cc2510
rf_modem - rfusb_cc2511
rf_modem - rfusb_cc1111
rf_modem - srf04_cc1110
rf_modem - srf05_cc2530
rf_modem - rfusb_cc2531
simple_hid_dongle - rfusb_cc1111
simple_hid_dongle - rfusb_cc2511
simple_hid_dongle - rfusb_cc2531
simple_hid_eb - srf04_cc2510
simple_hid_eb - srf04_cc1110
simple_hid_eb - srf05_cc2530
application
common
simple_hid.h
simple_hid_eb.c
simple_hid_keyboard.c
simple_hid_mouse.c
hal
Overview rf_modem rf_modem rf_modem rf_modem
usb_suspend.h simple_hid_dongle.c hal_mcu.c simple_hid_eb.c rf_modem.c hal_mcu.c
Messages File Line
Debug Log Build Find in Files
Ready
Errors 0, Warnings 0
Ln 18, Col 11

```
/*************************************************************************

    Filename:    simple_hid_eb.c

    Description: Main program of HID EB (keyboard/mouse) sample application
                 application runs in one the following modes (sub-applicati

                 1) Mouse function by operating the joystick.
                    Mouse buttons are operated by joystick push and S1.

                 2) Keyboard operation: joystick and button operations
                    write plain ASCII text to the PC.

**************************************************************************

/*************************************************************************
* INCLUDES
*/
#include "hal_defs.h"
#include "../common/mrfi_link.h"
#include "hal_board.h"
#include "hal_button.h"
#include "hal_joystick.h"
```

Figure 5. usb_app_ex.eww Workspace

Alternatively the generated HEX-file can be downloaded to the target using the SmartRF Flash programmer.

5.2 RF Protocol

Both examples use MRFI as the basic protocol for RF communication. On top of MRFI there is a minimal network layer which adds retransmission and acknowledge. The application layer uses two frame types: DATA and ACK. The latter is used to indicate to the sender that the receiver is busy.
The RF channel is fixed to MRFI logical channel ‘0’, defined as shown in Table 2. Please refer to the document *SimpliciTI Channel Table Information.pdf* contained in the SimpliciTI installer for details on the channel allocation [21].

RF Device	Channel	Frequency [MHz]
CC1111	20	906
CC2511	3	2425.7
CC2531	15	2425

Table 2. MRFI Channel '0' Frequencies

5.2.1 MRFI Frame Format

The MRFI frame format is identical to SimpliciTI apart from the NWK payload which is this example has been replaced by a two-byte header and the application payload. The MRFI/SimpliciTI header varies slightly between IEEE 802.15.4 compliant devices (CC2531) and CC1111/CC2511, please refer to [11] for details.

Frame Header	Sequence Number	Frame Type	Payload	Frame Check
11 bytes	1 byte	1 byte	0 - 44 bytes	1 byte

Table 3. Frame format

5.2.2 MRFI API

MRFI is the lowest software layer of the SimpliciTI protocol and is used in these application examples *without* the rest of SimpliciTI. It implements a common API for all supported RF devices, hiding the implementation details from the application. It is a pure radio interface, not to be confused with a more generic HAL. This section is limited to a brief overview of the API, as this document focuses on the applications and the USB Firmware Library. Only the API calls which are used by the examples are mentioned here. For a complete overview of the MRFI API, please refer to the MRFI interface header, **mrfi.h** (located in *source/components/simpliciTI/mrfi*) and the various implementations found in **mrfi_radio.c** (*source/components/simpliciTI/mrfi/radios/familyXX).* The relevant sub-folders are *family6 (CC2530/31)* and *family2 (CC1110/1111/2510/2511).*
All references to MRFI are contained in a single source file which is common to both examples (*source/apps/common/mrfi_link.c).*

void MRFI_Init(void);

This function must be called before any other MRFI calls.

uint8_t MRFI_Transmit(mrfiPacket_t *, uint8_t);

This call supplies a pointer to a mrfiPacket_t object whose frame element has been populated with the payload. The implementation populating the payload portion of the frame should use the MRFI macro (MRFI_P_PAYLOAD(p) defined in `mrfi.h`) to obtain a pointer to the place in the frame in which the payload information should be written. This allows MRFI to protect the preamble portion of the frame into which MRFI itself populates radio-specific information. This implies that the mrfiPacket_t pointer must point to a valid object even if there is no payload at all.
The txType parameter specifies whether transmit should occur immediately and unconditionally or CCA should be applied.

Valid values are:

- MRFI_TX_TYPE_FORCED
- MRFI_TX_TYPE_CCA

The address fields and the length field must also be set before the transmit call is invoked. These fields are not strictly part of the network layer encapsulation but it is that layer's responsibility to populate those fields. MRFI supplies 3 macros to populate these fields and they should be used. The 'p' formal parameter is (mrfiPacket_t *). The 'x' parameter is the length to be set in when using that macro.

- MRFI_P_SRC_ADDR(p)
- MRFI_P_DST_ADDR(p)
- MRFI_SET_PAYLOAD_LEN(p,x)

This routine returns a value indicating status of the transmit attempt.

- MRFI_TX_RESULT_SUCCESS
- MRFI_TX_RESULT_FAILED

Transmit may fail if CCA is requested and the channel is not clear after a number of retries.
All symbols referenced here are defined in `mrfi.h`.

void MRFI_Receive(mrfiPacket_t *);

This function copies a received frame into user space from the MRFI layer. The parameter is a pointer to an area in user space into which the frame can be copied.
This routine should be called as a result of having MRFI_RxCompleteISR() invoked. At that time there is guaranteed to be a valid frame available. The MRFI payload may be maximum 53 bytes for the CC1111/CC2511 and maximum 117 bytes with the CC2531.

void MRFI_RxCompleteISR(void);

User code must implement this method. When MRFI has obtained a frame from the radio and validated it MRFI will invoke this function. This is done within the receive ISR context.
At the very least the function should invoke MRFI_Receive() so that the new, valid frame is copied into user space and queued for processing. If the ISR thread is released before this copy is done it is possible that a second frame can be received and the first will be lost.

void MRFI_RxOn(void);

This function sets the radio to receive mode. Currently it is used only at startup by the *mrfi_link* layer. When the radio wakes up it is in the state IDLE. This call puts it in the RECEIVE state.

void MRFI_SetLogicalChannel(uint8_t);

This function sets the logical channel to the value specified as the argument. MRFI keeps a table of actual values to be written to the radio to accomplish the actual channel change. The argument is not the channel value itself but a logical channel number that is mapped to a tabled value actually sent to the radio. This table is instantiated at build time.
The maximum allowed logical channel can be determined by subtracting one from the number of logical channels. This is available from the following definition in `mrfi.h` :

- MRFI_NUM_LOGICAL_CHANS

The discipline to accomplish the channel change, such as perhaps changing the radio through the correct state transitions, is owned by MRFI and is not a concern of the caller.

uint8_t MRFI_SetRxAddrFilter(uint8_t *);

This function sets the address to be used as a filter value. Depending on the radio not all the bytes may be used as part of the native hardware filtering capabilities. Some of the actual filtering implementation may be in software. MRFI hides this partition and the caller does not know how the actual filtering is accomplished.

void MRFI_EnableRxAddrFilter(void);

When this call is made MRFI will subsequently filter out all received packets that do not match the address previously set. Broadcast packets are always received regardless of filter setting.

void MRFI_Sleep(void);

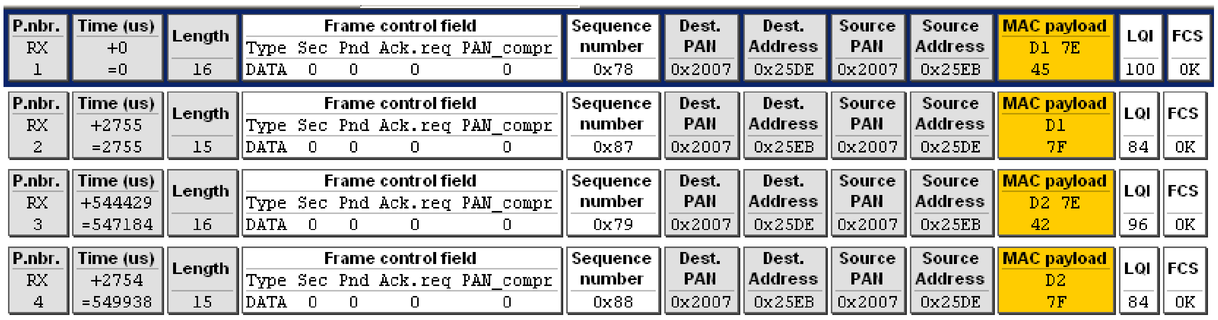
This call puts the radio into the sleep state. The call is synchronous in the sense that when it returns the radio in SLEEP state. To ensure reliable operation, this function must never be called from interrupt context.

void MRFI_WakeUp(void);

This call takes the radio out of SLEEP state into IDLE state. The call is synchronous in the sense that when it returns the radio in IDLE state.

5.2.3 Packet Sniffer Capture

The Texas Instruments Packet Sniffer is an indispensable tool when developing RF based applications. Using it for sniffing on IEEE 802.15.4 compliant devices like the CC2531 is straightforward as no configuration file is required. Simply select Channel 15 (0xF) and start. Sniffing on the CC1111 and CC2511 can be done by selecting the SimpliciTI protocol and selecting logical channel '0'.



P.nbr.	Time (us)	Length	Frame control field Type	Sec	Pnd	Ack.req	PAN_compr	Sequence number	Dest. PAN	Dest. Address	Source PAN	Source Address	MAC payload	LQI	FCS
RX 1	+0 =0	16	DATA	0	0	0	0	0x78	0x2007	0x25DE	0x2007	0x25EB	D1 7E 45	100	OK
RX 2	+2755 =2755	15	DATA	0	0	0	0	0x87	0x2007	0x25EB	0x2007	0x25DE	D1 7F	84	OK
RX 3	+544429 =547184	16	DATA	0	0	0	0	0x79	0x2007	0x25DE	0x2007	0x25EB	D2 7E 42	96	OK
RX 4	+2754 =549938	15	DATA	0	0	0	0	0x88	0x2007	0x25EB	0x2007	0x25DE	D2 7F	84	OK

Figure 6. Screenshot of the Texas Instruments General Packet Sniffer

5.3 USB HID Example ("simple_hid")

This application demonstrates the use of the USB Dongle as a USB composite HID (Human Interface Device), with both keyboard and mouse.

This application example uses the USB framework found in the USB firmware library [5] to configure the USB part of the device and to handle all USB standard requests. In addition, the Class requests required by the HID device class are implemented in a separate file. The radio communication between SmartRF board/EM and the dongle is carried out using MRFI.
Please see [9] or more information about the HID class.

Note. The application has been tested on Windows XP SP2, Windows XP SP3, Windows Vista SP1 and GNU/Linux (Ubuntu 9.04). Being a standard HID-device it should work on other operating systems as well.

5.3.1 Software Components

The executables for the applications are located as follows:

simple_hid_dongle.hex

The file is located in **ide\rfusb_cc1111\iar**, **ide\rfusb_cc2511\iar** or **ide\rfusb_cc2531\iar** depending on the target device.

simple_hid_eb.hex

The file is located in **ide\srf04_cc1110\iar**, **ide\srf04_cc2511\iar** or **ide\srf05_cc2531\iar** depending on the target device.

5.3.2 Installing the Application, CC1111/CC2511

- Attach the CC1110EM/CC2510EM to the SmartRF04EB and program it with the flash image *simple_hid_eb.hex*, using IAR Embedded Workbench or the SmartRF Flash Programmer. (See [1] for more information).
- Connect the CC1111/CC2511 USB dongle to the other SmartRF04EB through the SOC_DEM and program it with the flash image *simple_hid_dongle.hex*, using IAR Embedded Workbench or the SmartRF Flash Programmer. (See [1] for more information).

5.3.3 Installing the Application, CC2531

- Attach the CC2530EM to the SmartRF05EB and program it with the flash image *simple_hid_eb.hex*, using IAR Embedded Workbench or the SmartRF Flash Programmer. (See [1] for more information).
- Connect the the CC2531 USB dongle to the other SmartRF05EB and program it with the flash image *simple_hid_dongle.hex*, using IAR Embedded Workbench or the SmartRF Flash Programmer. (See [1] for more information). It is also possible to use the CC Debugger for this purpose.

5.3.4 Running the Application

Insert USB Dongle into a free USB port. The firmware will identify the USB Dongle as a composite HID with a keyboard and mouse. On most operating systems no extra driver is necessary; the OS will automatically load a standard driver for HID devices.
Select the operating mode "Mouse" or "Keyboard" by operating the joystick left/right and confirm the selection using the 'S1' button, see Figure 7.

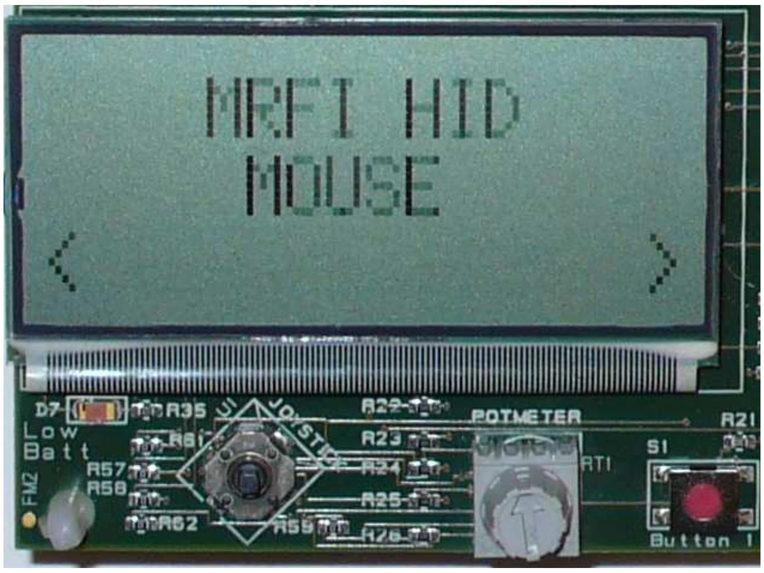


Figure 7. Selecting Mouse/Keyboard Mode, SmartRF05EB.

Mouse Mode

In mouse mode the joystick on the SmartRF04EB/SmartRF05EB can be used as a mouse. Push the joystick up/down/left/right to emulate mouse movement. The button “S1” emulates the right mouse button, pushing the joystick emulates the left mouse button.

Keyboard Mode

In keyboard mode, the joystick operations and button hits transmit characters to the PC according to Table 4.

EB Operation	Transmitted Character
Joystick up	U
Joystick down	D
Joystick left	L
Joystick right	R
Joystick press	P
Button S1 press	B

Table 4. Transmitted Characters in Keyboard Mode

In order to observe the output from the HID example in keyboard mode it is recommended to open an editor like Notepad and make sure that the editor window has focus. Please observe that when pressing the joystick or S1 button, the letter will be transmitted as the button/joystick is released.

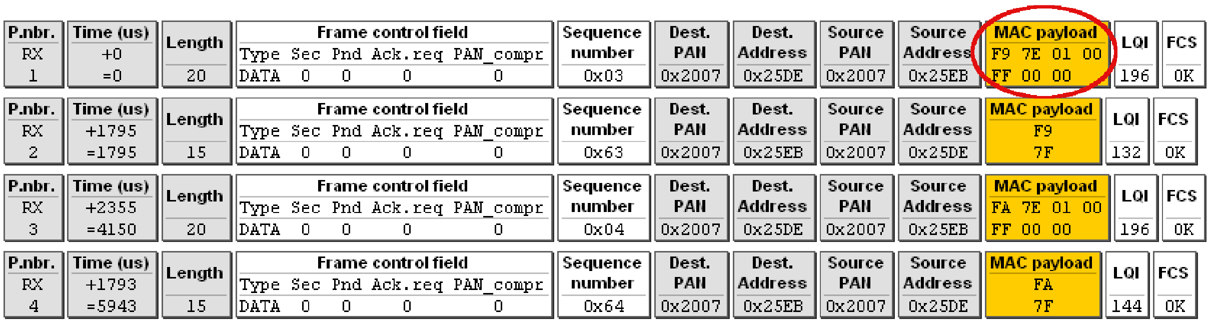
5.3.5 Packet Sniffer Capture

Figure 8 shows the radio traffic when operating the joystick left in mouse mode. The first byte of the MAC payload is the sequence number. The second byte is the frame type, where 7E identifies a data frame and 7F an ACK frame. The third byte is the in this case single byte payload. Frame 1 and 3 in the figure are data packets with the payload 45 and 42 respectively, i.e. the letters E and B. The source address of the EB is 25EB and the source address of the dongle is 25DE.

The mouse mode payload is decoded as shown in Table 5. Figure 8 gives an example of a mouse frame. Byte 4 of the MAC payload is decoded as 0xFF (-1) which means mouse left movement. Byte 2 identifies the frame as a mouse packet. There is no button activity (byte 3), and no activity on the Y (byte 5) and Z (byte 6) axis. The frames with 2-byte payloads are application ACK frames.

MAC Payload Byte	Description
1	Sequence number
2	Frame type (DATA)
3	Mouse DATA ID (fixed 1)
4	Button activity
5	Mouse X movement
5	Mouse Y movement
6	Mouse Z movement (Scroll wheel)

Table 5. Mouse Mode Payload



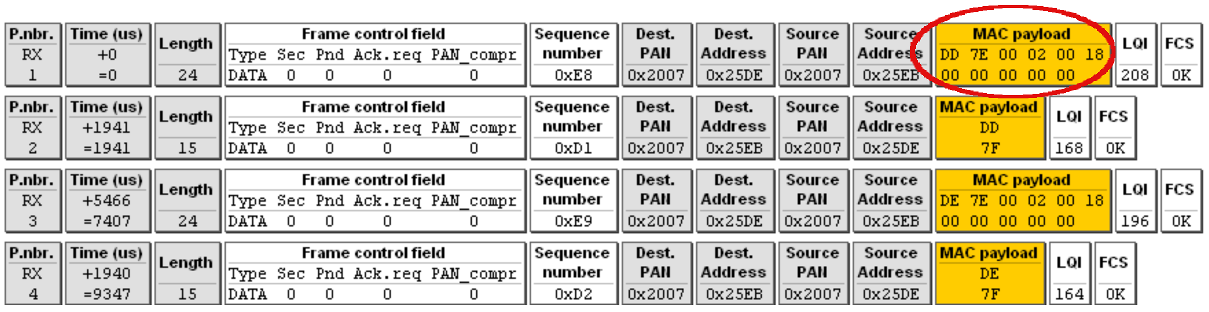
P.nbr. RX	Time (us)	Length	Frame control field Type	Sec	Pnd	Ack.req	PAN_compr	Sequence number	Dest. PAN	Dest. Address	Source PAN	Source Address	MAC payload	LQI	FCS
1	+0 =0	20	DATA	0	0	0	0	0x03	0x2007	0x25DE	0x2007	0x25EB	F9 7E 01 00 FF 00 00	196	OK
2	+1795 =1795	15	DATA	0	0	0	0	0x63	0x2007	0x25EB	0x2007	0x25DE	F9 7F	132	OK
3	+2355 =4150	20	DATA	0	0	0	0	0x04	0x2007	0x25DE	0x2007	0x25EB	FA 7E 01 00 FF 00 00	196	OK
4	+1793 =5943	15	DATA	0	0	0	0	0x64	0x2007	0x25EB	0x2007	0x25DE	FA 7F	144	OK

Figure 8. Mouse Left Movement

The keyboard mode payload is decoded as shown in Table 6. An example of a keyboard frame is shown in Figure 9. Byte 4 of the MAC payload has the value 0x02 which means that the “SHIFT LEFT” button is pressed. This is hard-coded in the application for the purpose of this example. Byte 6 is 0x18 which corresponds to the letter ‘u’. The capitalization is achieved through the key modifier, bit 1.

MAC Payload Byte	Description
1	Sequence number
2	Frame type (here DATA)
3	Keyboard data ID (fixed 0)
4	Key modifiers
5	Reserved (fixed 0)
6 - 11	Key values

Table 6. Keyboard Mode Payload



P.nbr. RX	Time (us)	Length	Frame control field Type	Sec	Pnd	Ack.req	PAN_compr	Sequence number	Dest. PAN	Dest. Address	Source PAN	Source Address	MAC payload	LQI	FCS
1	+0 =0	24	DATA	0	0	0	0	0xE8	0x2007	0x25DE	0x2007	0x25EB	DD 7E 00 02 00 18 00 00 00 00 00	208	OK
2	+1941 =1941	15	DATA	0	0	0	0	0xD1	0x2007	0x25EB	0x2007	0x25DE	DD 7F	168	OK
3	+5466 =7407	24	DATA	0	0	0	0	0xE9	0x2007	0x25DE	0x2007	0x25EB	DE 7E 00 02 00 18 00 00 00 00 00	196	OK
4	+1940 =9347	15	DATA	0	0	0	0	0xD2	0x2007	0x25EB	0x2007	0x25DE	DE 7F	164	OK

Figure 9. Operating Joystick UP (letter ‘U’)

5.4 USB RF Modem Example ("rf_modem")

This application example is a wireless USB to serial port (RS232) converter. It creates a virtual serial port on the host operating system.

Data sent to this virtual serial port will be transferred to the USB device over the PC's USB hub, then over the air to the CC1110/CC2510/CC2530EM and finally to a PC serial port via the SmartRF04EB/SmartRF05EB.

Hardware flow control for RTS is implemented. CTS is not supported directly by the CDC driver but is indirectly implemented by stopping the processing of USB requests when the RF modem is busy with processing the RF traffic. The radio link has CRC check and automatic retransmission for error free operation.

Note. The application has been tested on Windows XP SP2, Windows XP SP3, Windows Vista SP1 and GNU/Linux (Ubuntu 9.04). For other operating systems a suitable drivers may have to be found or created.

This application example uses the USB framework found in the USB firmware library to configure the USB Dongle and to handle all USB standard requests. In addition, the class requests required by the CDC device class are implemented in a separate file.

The radio communication between CC2530EM and the USB Dongle is done using the MRFI protocol Please see [10] for more information about the CDC class.

5.4.1 Software Components

The executables for the applications are located as follows:

rf_modem.hex

The file is located in **ide\rfusb_cc1111\iar**, **ide\rfusb_cc2511\iar**, **ide\rfusb_cc2531\iar, ide\srf04_cc1110\iar**, **ide\srf04_cc2511\iar** or **ide\srf05_cc2531\iar** depending on the intended target device.

In addition, a setup information file (**usb_cdc_driver_cc*xxxx*.inf**) will be needed on Windows operating systems to associate the device with the operating system's driver for USB CDC-ACM class (*usbser.sys*). There are three files, one per USB dongle, located in the **.\driver** folder.

Note. The 64-bit edition of Microsoft Windows Vista only accepts digitally signed driver installation files. This driver files in this software package are not signed and will not be accepted. They can however be used on 32-bit edition of Windows Vista.

5.4.2 Installing the Application, CC1111/CC2511

- Attach the CC1110EM/CC2510EM to the SmartRF04EB and program it with the flash image **rf_modem.hex**, using IAR Embedded Workbench or the SmartRF Flash Programmer. (See [1] for more information).
- Connect the USB Dongle to the other SmartRF04EB through the SOC_DEM and program it with the flash image **rf_modem.hex**, using IAR Embedded Workbench or the SmartRF Flash Programmer. (See [1] for more information).

5.4.3 Installing the Application, CC2531

- Attach CC2531EM to the SmartRF05EB and program it with the flash image **rf_modem.hex**, using IAR Embedded Workbench or the SmartRF Flash Programmer. (See [1] for more information).

- Connect the CC2531 USB Dongle to the other SmartRF05EB and program it with the flash image **rf_modem.hex**, using IAR Embedded Workbench or the SmartRF Flash Programmer. (See [1] for more information). It is also possible to use the CC Debugger for this purpose.

5.4.4 Running the USB Application

- Disconnect the programmed USB Dongle from the SmartRFEB/Programmer and plug it into a free USB port on the PC.
- If using Microsoft Windows, the operating system will ask for a driver. Point to the file ".\driver" folder containing the "usb_cdc_driver_cc*xxxx*.inf" setup information files.

When the USB-dongle is plugged in first time it will appear in Windows XP as in Figure 10.

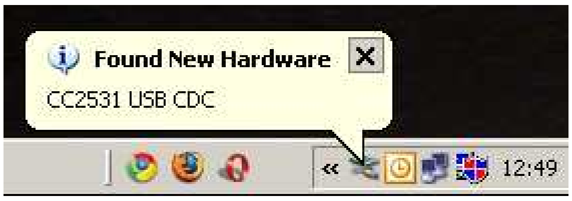
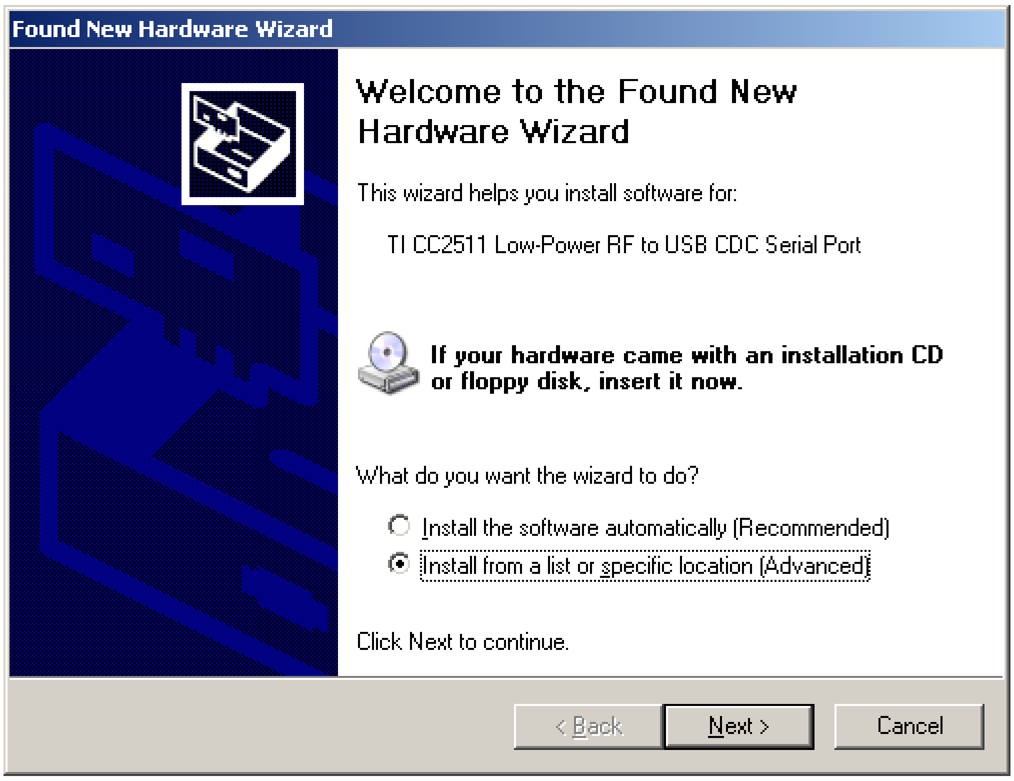

Found New Hardware
CC2531 USB CDC

Figure 10. Plugging in the USB Dongle First Time

The Windows installation Wizard will now start and guide the user through the driver installation procedure. Please select "**Install from a list or specific location**" as shown in Figure 11.


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

TI CC2511 Low-Power RF to USB CDC Serial Port

If your hardware came with an installation CD or floppy disk, insert it now.

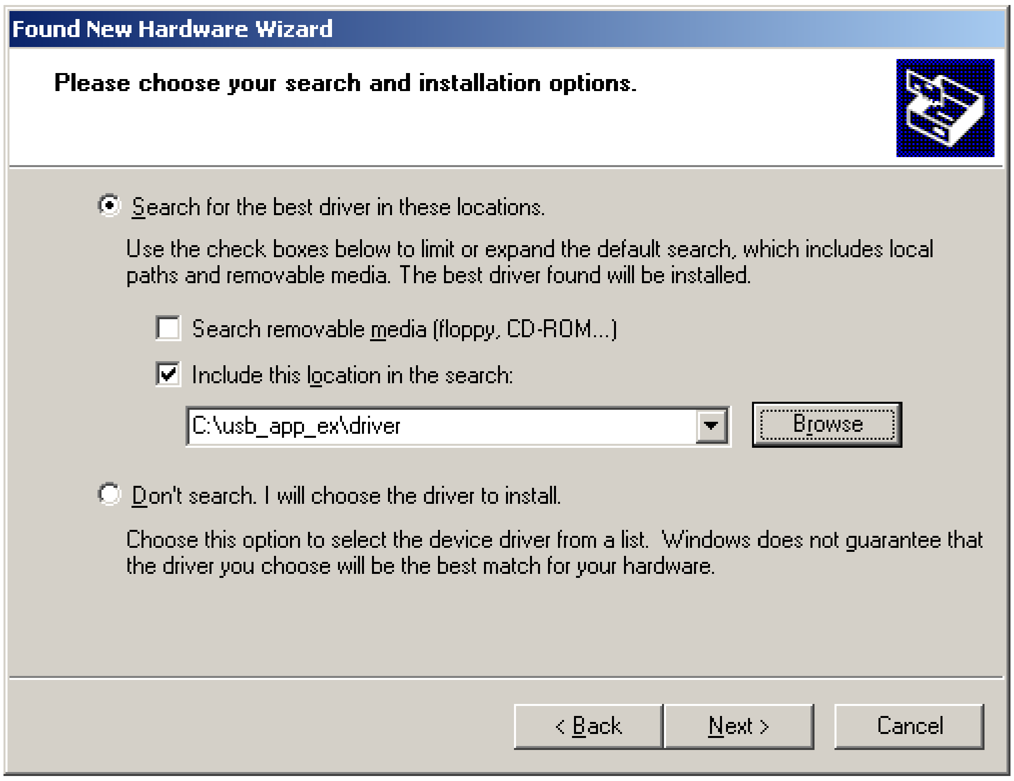
What do you want the wizard to do?

○ Install the software automatically (Recommended)
● Install from a list or specific location (Advanced)

Click Next to continue.

Figure 11. Driver Installation, Step 1

Click '**Next'** to advance to the next step. Select "**Search for the best driver in these locations"** as shown in Figure 12. Make sure that the wizard searches the **driver** directory one level below the installation root. It is also possible to install the driver manually. This is quicker but the user runs the risk of installing the wrong driver.


Found New Hardware Wizard

Please choose your search and installation options.

(●) Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

[ ] Search removable media (floppy, CD-ROM...)

[✓] Include this location in the search:

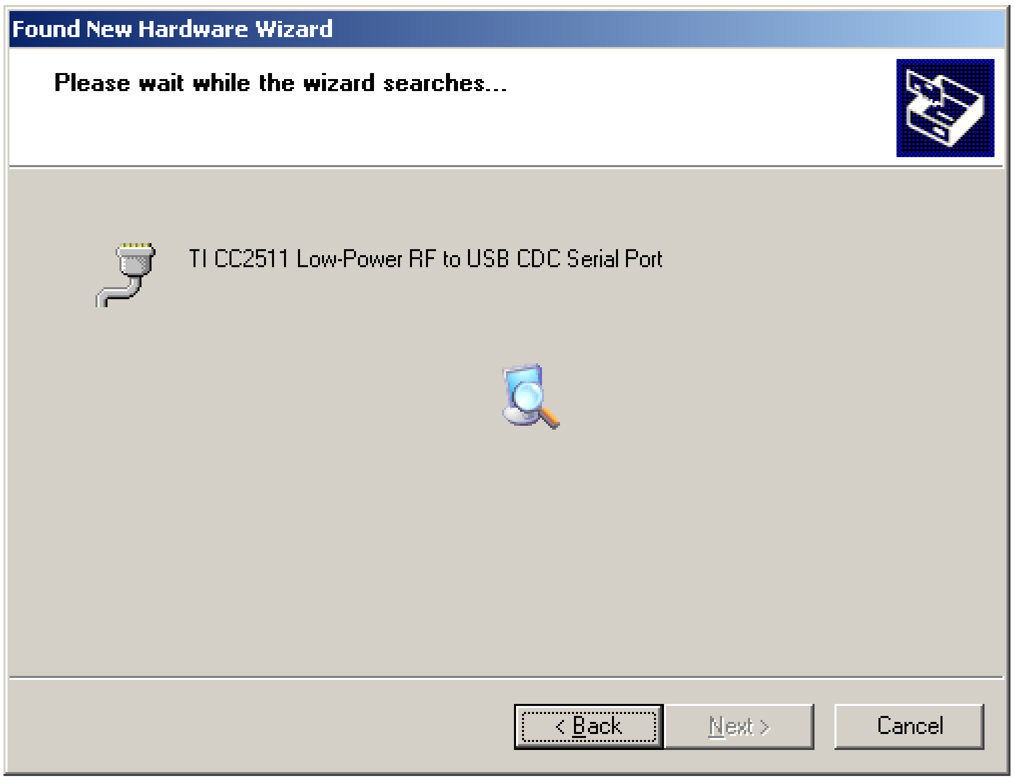
C:\usb_app_ex\driver

( ) Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Figure 12. Driver Installation, Step 2

Click **Next** to advance to the next step. Windows will now search for the driver which may take some time (see Figure 13).

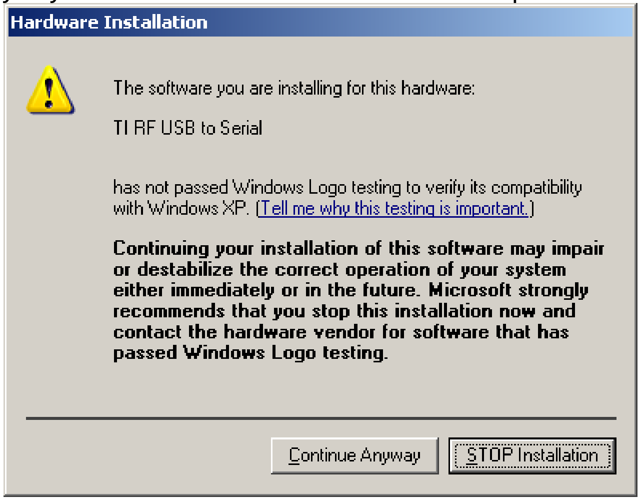

Found New Hardware Wizard

Please wait while the wizard searches...

TI CC2511 Low-Power RF to USB CDC Serial Port

Figure 13. Driver Installation, Step 3

You are now likely to encounter the warning dialog shown in Figure 14. If that's the case you can safely click 'Continue anyway' and the driver installation will be complete.



Hardware Installation

The software you are installing for this hardware:

TI RF USB to Serial

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

Figure 14. Driver Signature Warning

When Windows have finished installing the device, a new serial port will show in the device manager in Windows. See Figure 15.

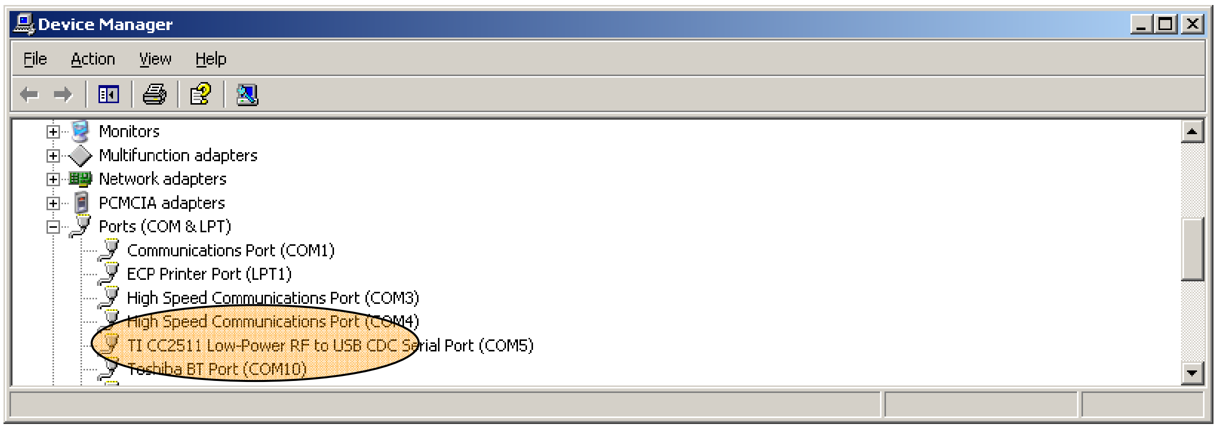


Figure 15. Virtual COM Port in Windows Device Manager

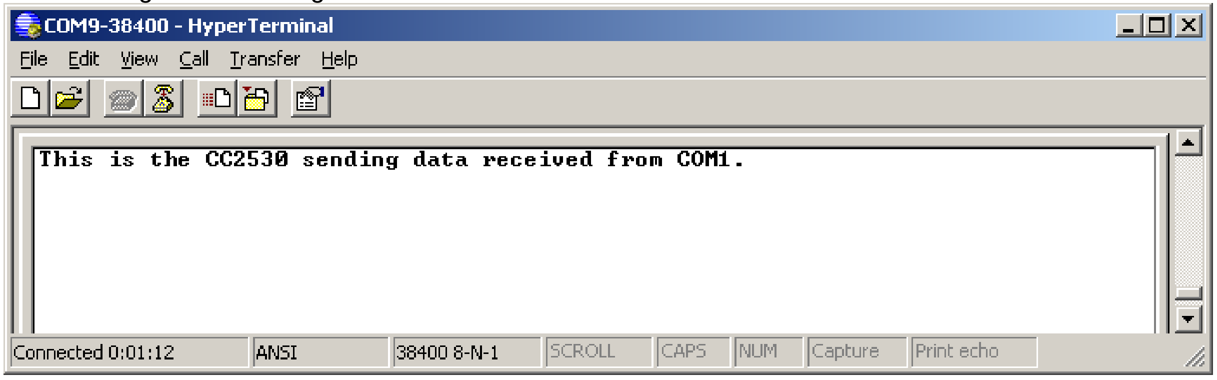
Note. The port number assigned to the USB Virtual Serial port is arbitrary. You may change it to an alternative port number as follows: Right click on the port in the device manager and choose "Properties" > "Port settings" > "Advanced" to change.

The Virtual COM-port is now ready and can be used as any standard serial port. Note that the control line CTS (Clear To Send) will not be visible in terminal emulators or analysis tools which shows the state of this line – the CDC class does not make use of this feature. The green LED on the dongle is lit only when the PC asserts the RTS signal, in HyperTerminal this is equivalent to connecting.

Note. Make sure that the PC application (e.g. HyperTerminal) is disconnected before physically unplugging and inserting the USB dongle. Leaving HyperTerminal connected may cause the USB dongle to fail during enumeration due to being flooded by requests by a PC application which thinks it is still connected.

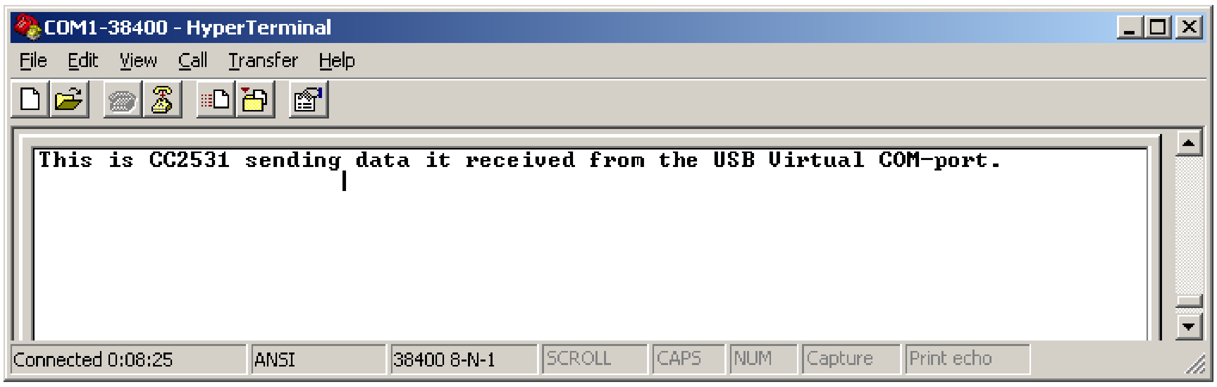
5.4.5 Running the EB Application

Connect the CC2510/CC1100EM to the SmartRF04EB and power it OR connect the CC2530EM to a SmartRF05EB and apply power. Select "Device 1" by pushing button S1 – LCD will then display "Device 1 ready". Connect an RS232 NULL-modem cable between a COM-port on the PC and the serial port on the SmartRF04B/SmartRF05EB. Connect one HyperTerminal window to the Virtual COM port, and another HyperTerminal window to the COM port connected to the evaluation board. The port settings should be 38400 baud, 8 bits, 1 stop bit, no parity. Hardware handshaking must be enabled. You should observe text written in one HyperTerminal window appearing in the other as shown in Figure 16 and Figure 17.



```
This is the CC2530 sending data received from COM1.
```

Figure 16. CC2531 Receiving Data

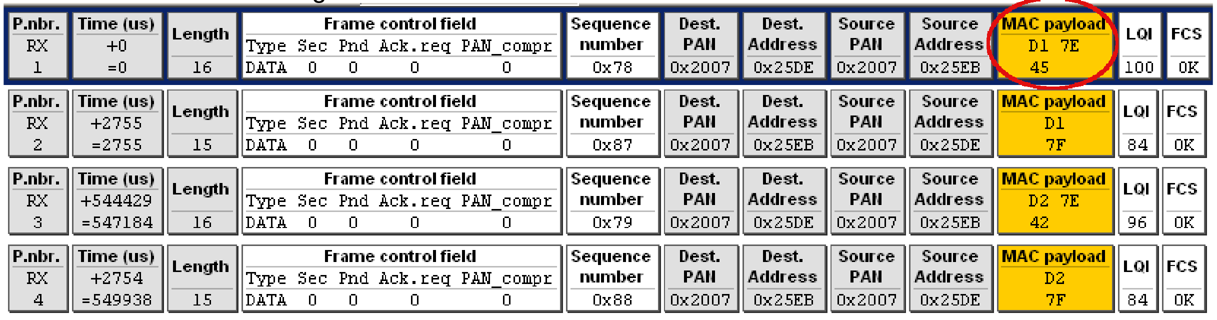


```
This is CC2531 sending data it received from the USB Virtual COM-port.
```

Figure 17. CC2530 Receiving Data

5.4.6 *Packet Sniffer Capture*

When typing single characters in the HyperTerm window the slow input speed means that each packet will only contain one byte of payload. Figure 18 shows the radio traffic when typing the letters 'EB' in the terminal window connected to COM1. The first byte of the MAC payload is the sequence number. The second byte is the frame type, where 7E identifies a data frame and 7F an ACK frame. The third byte is the in this case single byte payload. Frame 1 and 3 in the figure are data packets with the payload 45 and 42 respectively, i.e. the letters E and B. The source address of the EB is 25EB and the source address of the dongle is 25DE.



P.nbr. RX	Time (us)	Length	Frame control field Type	Sec	Pnd	Ack.req	PAN_compr	Sequence number	Dest. PAN	Dest. Address	Source PAN	Source Address	MAC payload	LQI	FCS
1	+0 =0	16	DATA	0	0	0	0	0x78	0x2007	0x25DE	0x2007	0x25EB	D1 7E 45	100	OK
2	+2755 =2755	15	DATA	0	0	0	0	0x87	0x2007	0x25EB	0x2007	0x25DE	D1 7F	84	OK
3	+544429 =547184	16	DATA	0	0	0	0	0x79	0x2007	0x25DE	0x2007	0x25EB	D2 7E 42	96	OK
4	+2754 =549938	15	DATA	0	0	0	0	0x88	0x2007	0x25EB	0x2007	0x25DE	D2 7F	84	OK

Figure 18. Typing 'EB' at COM1

5.5 USB Firmware Library

5.5.1 *Structure*

Texas Instruments LPRF provides a USB Firmware Library which contains all the basic USB firmware needed for application development. The firmware library handles the interface to the USB chip, handles endpoints, traffic flow, descriptor parsing and configuration. It also performs USB standard request processing. The USB Firmware Library consists of firmware common to all the three USB supported USB devices as well as some device specific low-level code. The USB Firmware Library does NOT contain any USB class specific components. It supports both vendor-specified device classes and standardized device classes such as HID and USB audio, with any number of configurations, interfaces, and alternate settings. The following features are provided:

- Initialization of the USB peripheral unit, and response to reset-signaling on the bus.
- Automated response to almost all standard requests, based on data from the USB descriptor set (the SYNCH_FRAME and SET_DESCRIPTOR requests must be handled by the application).
- Automated setup of endpoint control registers, depending on the selected configuration and interface alternate settings.
- All endpoint types are supported: Setup, bulk, interrupt and isochronous.
- An interface for responding to class and vendor requests. Both automated and manual data transfer is supported.
- Endpoint access via a simple set of macros and functions.
- Support for suspend mode and remote wake-up.
- A function hook- and event-based interface allows for the framework to run from main, the USB interrupt, or a combination of both.

The library is generic. However, it does require some knowledge of the USB specification, including the concepts of configurations, interfaces and endpoints, different endpoint types, standard, vendor and class requests, and USB descriptor sets. Figure 19 shows how the USB firmware library is organized.

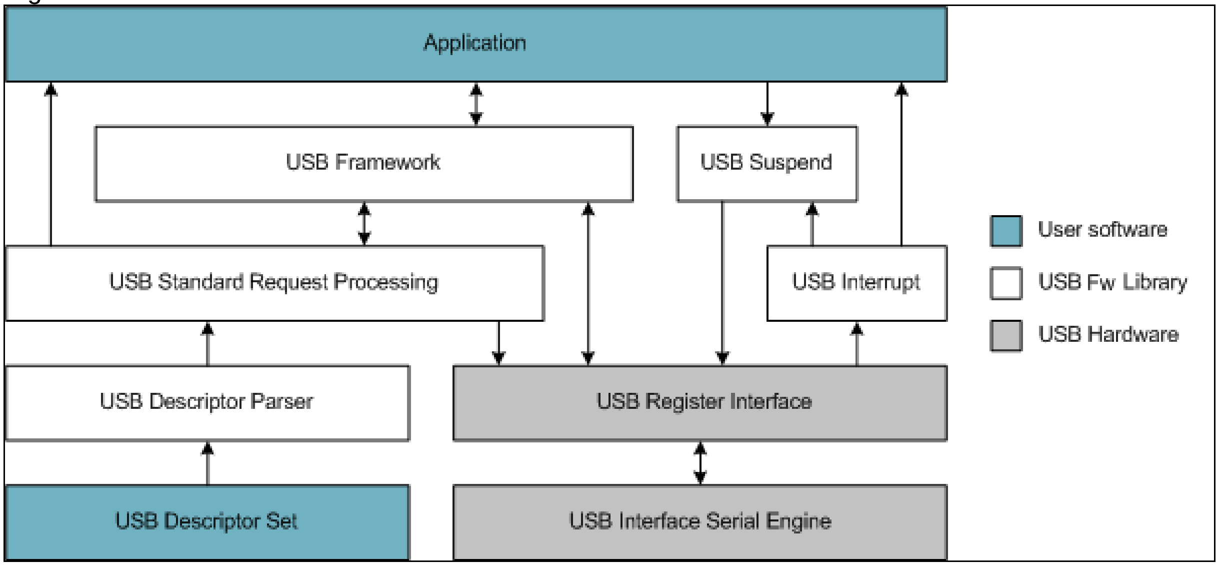


Figure 19. USB Firmware Library Structure

As shown in Figure 19 the USB library is divided into five modules:
The USB Firmware Library contains all USB status information and functions for initialization, device reset handling, and transfers on endpoint 0. It also contains basic macros and functions for endpoint control and FIFO access. Vendor and class requests are passed up to the application, and standard requests further down for internal processing.
The USB Standard Requests (usbsr) module contains automated processing functions for almost all standard requests. The automated handling and endpoint configuration relies on the USB descriptor set and two look-up tables. Requests that are not supported (SYNCH_FRAME and SET_DESCRIPTOR) or requests that are class specific are passed up to the application. The application is also notified upon important events, such as change in endpoint status or interface alternate settings.
The USB Descriptor Parser (usbdp) module provides a mechanism for locating standard-formatted USB descriptors. The user must provide a USB descriptor set, and two simple lookup tables - one for locating other descriptor formats, and another for setting up endpoint double-buffering. The necessary

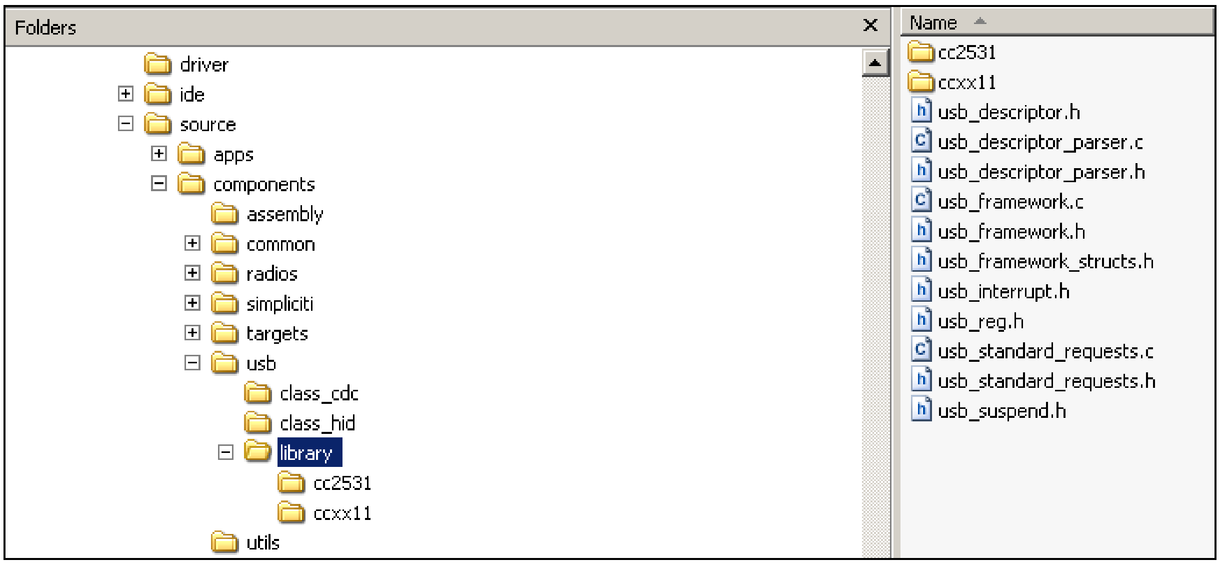
constant and structure definitions, and guidelines for writing compatible descriptor sets, are found in the USB Descriptor module.

The USB Suspend (usbsusp) module provides easy to use, fully automated support for USB suspend, USB resume, and USB remote wakeup functionality.

To implement USB suspend and USB resume the user only need to add a short piece of code to the main loop. USB remote wakeup is performed by a single function call.

5.6 USB Source Code Organization

Figure 20 shows how the source code of the USB firmware library is organized. The common code is kept in the 'library' directory and the device dependent code is place in sub-folders, one for each device. Note that the code for the CC1111 and CC2511 is identical; the device dependent code is shared in the folder CCxx11.



```
Folders
        driver
      ⊞ ide
      ⊟ source
        ⊞ apps
        ⊟ components
            assembly
          ⊞ common
          ⊞ radios
          ⊞ simpliciti
          ⊞ targets
          ⊟ usb
              class_cdc
              class_hid
            ⊟ library
                cc2531
                ccxx11
            utils
```

```
Name
cc2531
ccxx11
usb_descriptor.h
usb_descriptor_parser.c
usb_descriptor_parser.h
usb_framework.c
usb_framework.h
usb_framework_structs.h
usb_interrupt.h
usb_reg.h
usb_standard_requests.c
usb_standard_requests.h
usb_suspend.h
```

Figure 20. USB Firmware Library Directory Structure

The complete documentation of USB Firmware Library is available as a compressed HTML file, **usb_firmware_library.chm** found in ".\docs" folder.

5.6.1 Adapting the USB Application

Although the USB Firmware Library will work as is for most applications it may in some cases be desirable to reorganize the way interrupts/events are handled. In the default configuration handling USB standard requests, USB class requests and USB data transfer is done in the main loop to keep interrupt processing time as brief as possible. The interrupt handler simply schedules events and passes these on to the main loop for processing. Simple round-robin scheduling is thus implicitly determined by the order in which the event processing is done. For more sophisticated pre-emptive schemes the choice is either to do more of the processing in an interrupt context or devise a method for pre-emptive scheduling in the main loop - it is beyond the scope of this document to describe in detail how either can be achieved. However, this section will give a brief outline on how to achieve immediate processing of time-critical code sections. Note that there is no need to modify the USB Firmware Library; the desired effect is achieved by moving code from the main loop to the USB interrupt hook.

The USB Firmware Library requires the application to provide a hook, *usbirqHookProcessEvents(),* for events which needs immediate processing in an interrupt context. The applications described in this document do all their USB processing in the main context, this in effect gives the radio traffic priority over the USB traffic. Hence *usbirqHookProcessEvents()* is just a stub in both applications. However, it is straightforward to move code from the main loop to the interrupt hook.

In the HID example the main loop processing is found in the file **usb/class_hid/usb_hid.c**. All the processing in done in the main loop, i.e. standard requests, vendor requests, class requests and data transfer. For the HID example this makes perfect sense as the data rates are very low. The hooks found in **usb/class_hid/usb_hid_hooks.c.** These are just stubs for this application. The stubs *must* be declared as the USB firmware library assumes that they are present.

In the CDC example the main loop processing is found in the file **usb/class_cdc/usb_cdc.c**. All the processing is done in the main loop, i.e. standard requests, vendor requests, class requests and data transfer. The hooks are found in **usb/class_cdc/usb_cdc_hooks.c.** Most of these are stubs, only the hooks for setting and reading control lines and port settings are implemented. Note that these are still processed by the main loop, the reason for implementing this functionality as hooks is that it is application dependent.

5.7 USB Descriptors

One of the challenges when developing USB firmware is to understand and work with USB Descriptors. Details can be found in the help file **usb_firmware_library.chm** found in the *docs* directory of this software distribution [5]. The USB class definitions ([9] and [10]) contain further guidelines on writing USB descriptor. USB developers should also visit **http://www.usb.org/developers** for more resources.

6 References

[1] CC1110-CC1111DK CC2510-CC2511DK Development Kit User Manual (swru134)

[2] CC2530DK Development Kit User Manual (swru208)

[3] Texas Instruments SmartRF Studio (swrc046)

[4] Texas Instruments SmartRF Flash Programmer (swrc044)

[5] CC USB Firmware Library and Examples (swrc088)

[6] Texas Instruments Packet Sniffer (swrc045)

[7] CC1111/CC2511 USB Hardware User's Manual (swrc082)

[8] CC2531 USB Hardware User's Manual (swrc221)

[9] Device Class Definition for HID 1.11, US-IF Inc., 2001 (http://www.usb.org/developers/devclass_docs/HID1_11.pdf)

[10] USB Class Definitions for Communication Devices 1.1, USB-IF Inc., 1999 (http://www.usb.org/developers/devclass_docs/usbcdc11.pdf)

[11] SimpliciTI Installer (simpliciti.html)

7 General Information

7.1 Document History

Revision	Date	Description/Changes
SWRU222	2009.05.08	Initial release