CC2530 Software Examples

User's Guide

Table of contents

1 Introduction

This document describes software examples for the CC2530 System-on-Chip solution for IEEE 802.15.4/ZigBee. It also describes the necessary hardware and software to run the examples, and how to get started. The software examples are designed to run on the CC2530EM mounted on SmartRF05EB.

Section 3 of this document describes necessary prerequisites and how to get started with the code examples. Section 4 describes how to run each of the application examples. The software library that the code examples are built upon is described in section 5. The latter section also gives an API reference and describes the functionality of the software library. Hex files for each of the example applications are provided. IAR EW8051 Full version is needed for building the source code.

2 Abbreviations

API	-	Application Programming Interface
CBC-MAC	-	Cipher Block Chaining Message Authentication Code
CCM	-	Counter with CBC-MAC (mode of operation)
CCM*	-	Extension of CCM
FCS	-	Frame Check Sequence
HAL	-	Hardware Abstraction Layer
IO	-	Input/Output
MIC	-	Message Integrity Code
MPDU	-	MAC Protocol Data Unit
PAN	-	Personal Area Network
PER	-	Packet Error Rate
RF	-	Radio Frequency
RSSI	-	Received Signal Strength Indicator
SFD	-	Start of Frame Delimiter

3 Using the software

This section describes the necessary hardware and software, and how to get started with the application examples for CC2530.

3.1 Prerequisites

To successfully download and run the software described in this document, the following material is needed:

- 2 x SmartRF05 EB rev. 1.7
- 2 x CC2530EM boards with appropriate antennas
- IAR Embedded Workbench for 8051 versions 7.51* (Full version)
- 4 AA batteries

*The software is tested with version 7.51, but later versions might also work.

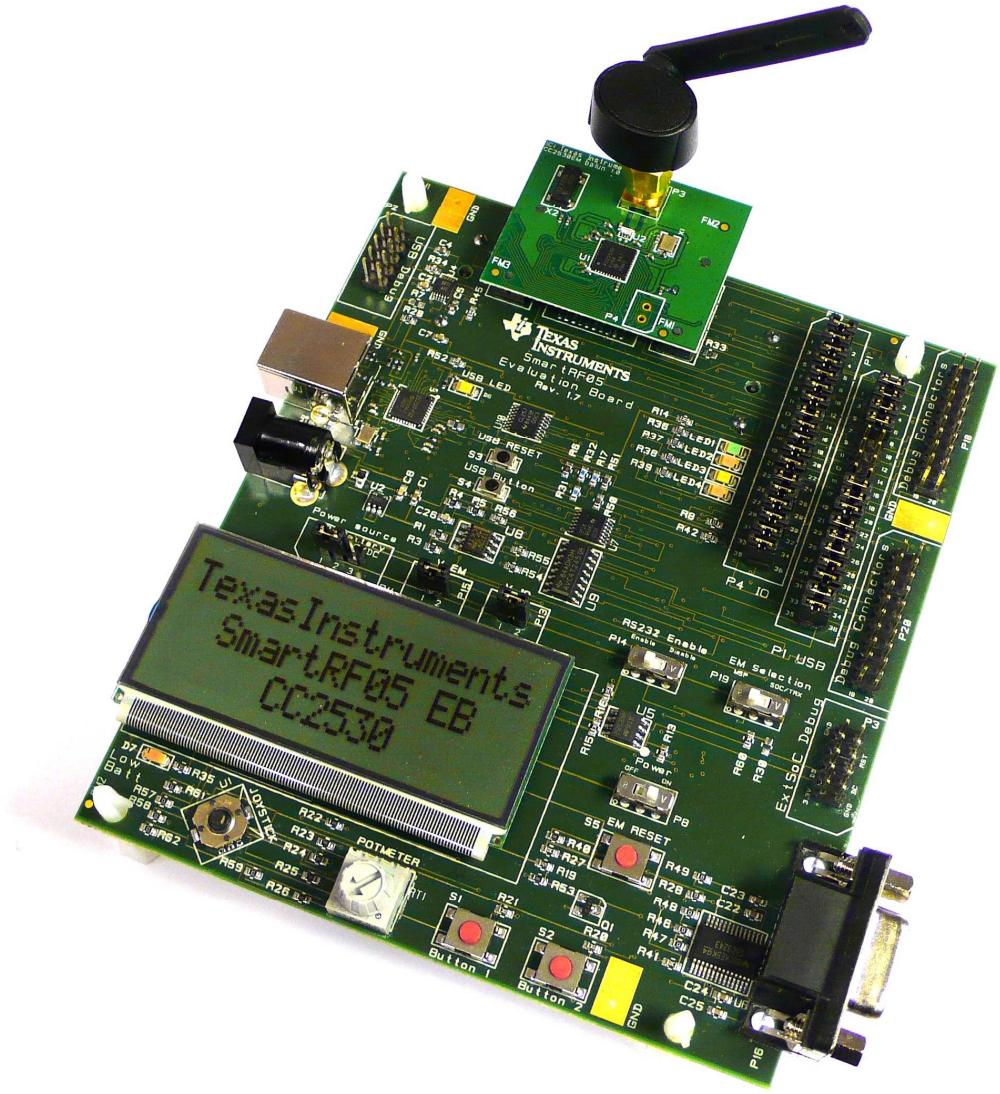

Figure 1 SmartRF05EB with CC2530EM

3.2 Getting started

The following sections describe hardware and software setup, how to program the board and how to run example code from the IAR debugger. A description of how to operate each software example is found in section 4 of this guide.

3.2.1 Set up Hardware and Software

Follow these steps to configure the hardware and software needed:

1. Install IAR Embedded Workbench for 8051 and the patch to enable support for CC2530.
2. Save the CC2530 Software Examples zip file and unzip this file.
3. Attach the CC2530EM board to the SmartRF05EB.
4. Connect the SmartRF05EB to the PC with a USB cable.
5. Make sure the EM selection switch (P19 on SmartRF05EB) is placed in position *SoC/TRX.*

3.2.2 Program the board with IAR

6. Open IAR Embedded Workbench
7. Open the workspace file **CC2530_SW_examples.eww** with IAR. This file is found in the folder **ide** under the folder where the CC2530 Software Examples was unzipped. Figure 2 shows the IAR EW with the workspace opened.
8. Each application has its own project tab in the IAR workspace viewer. Select the project to be compiled in the workspace viewer of IAR. See section 4 for a description of each application example.
9. Select **Project->Rebuild All**. This will perform a full rebuild on the selected project.
10. Select **Project->Debug.** IAR will now establish a connection with the CC2530 and program the application. The debugger will be started, halting the target at main().
11. Start the application by selecting **Debug -> Go**.
12. The board can be reset by selecting **Debug -> Reset**.
13. The debugger can be stopped by selecting **Debug -> Stop Debugging**.
14. The unit can now be operated independently from the debugger by disconnecting the USB cable and using the AA batteries as power source. Cycle power with the power switch on the SmartRF05EB.
15. Repeat the steps 9 to 12 to program additional boards.

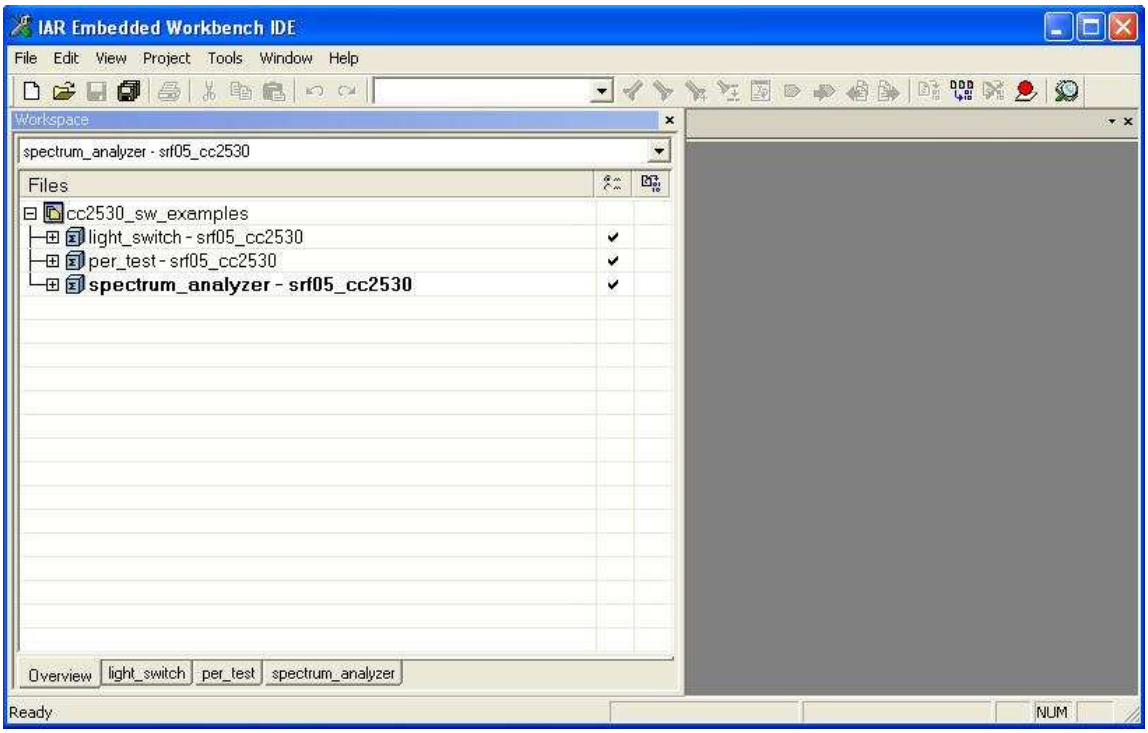

Figure 2 IAR EW

3.2.3 Alternative: Download hex files with the Flash Programmer

It is also possible to program the boards with the Texas Instruments Flash Programmer as an alternative to IAR. Follow these steps after connecting the USB cable in section 3.2.1.

5. Install the Texas Instruments Flash Programmer. The TI Flash programmer tool is found under the 'Tools and software' section on the CC2530 web site.
6. Open the Texas Instruments Flash Programmer, and choose the System-on-Chip tab. The connected device is shown in the list as in Figure 3.
7. In the Flash image field browse to the correct hex file.
8. Make sure the 'Erase, program and verify' action is checked.
9. Push the 'Perform actions' button to program the device.

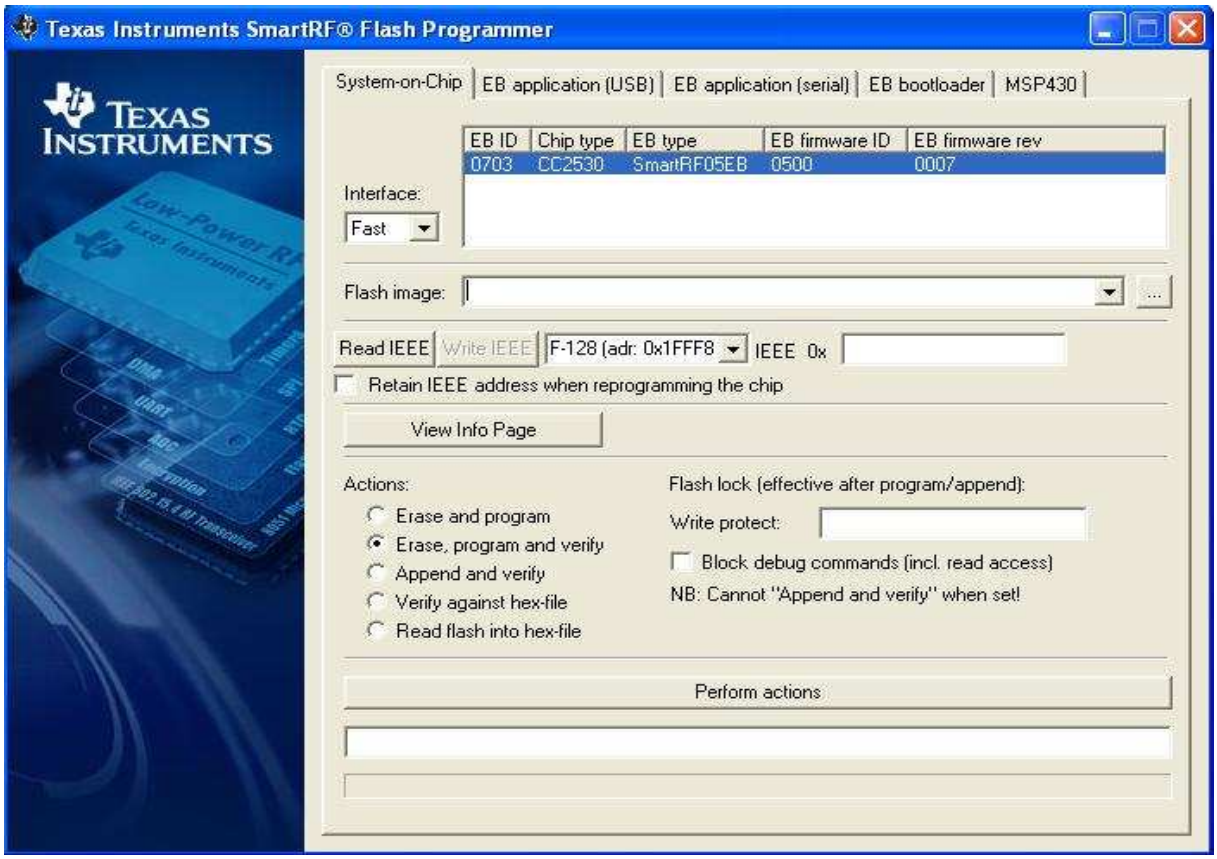

Figure 3 Flash programmer

4 Application Examples

The following application examples are included in this software package:

“light_switch”	Wireless light/switch application. One node is configured as a light controller, and the other node as a wireless light switch.
“PER_test”	Packet Error Rate test application.
“spectrum_analyzer”	This application use the LCD on the SmartRF05EB to display the RSSI values of all IEEE 802.15.4 defined channels.

Details about how to run the different application examples can be found in the following sections. Section 3.2 describes how to program the applications on the target.

4.1 Light/Switch application

This application example requires 2 nodes programmed with the ‘light_switch’ project.

The example implements a wireless light switch application. One of the nodes is configured as a light controller, while the other node is configured as a light switch.

The following steps must be done to use the light/switch application:

1. Reset both boards by cycling power.
2. Press Button 1 to enter the application menu
3. Choose device mode. The menu is navigated by moving the joystick right or left. Choose device mode ‘Switch’ on one of the nodes, and ‘Light’ on the other node. Confirm the choices by pressing Button 1.
4. The light switch application example is now ready. LED1 on the ‘Light’ node can now be toggled by pushing joystick down on the ‘Switch’.

The data sent out from the switch device can be observed with a Texas Instruments packet sniffer* configured on channel 25 – 2475 Mhz.

**The Texas Instruments packet sniffer application can be downloaded from: http://focus.ti.com/docs/toolsw/folders/print/packet-sniffer.html. The following hardware can be used for the packet sniffer: SmartRF05EB/CC2530EM, CC2531 USB dongle, CC2430DB, SmartRF04EB/CC2430EM or SmartRF05EB/CC2520EM.*

It is also possible to build the Light/Switch application with the CCM security feature included. This will enable CCM authentication and encryption on each packet. In order to use the CCM security feature the compile option SECURITY_CCM must be set in the project file. This can be done in IAR EW by selecting Project and Options. Navigate to the C/C++ Compiler and Preprocessor tab and set SECURITY_CCM as one of the defined symbols (i.e. change xSECURITY_CCM to SECURITY_CCM). See also Figure 4. The CCM security feature is also described in section 5.2 in this guide.

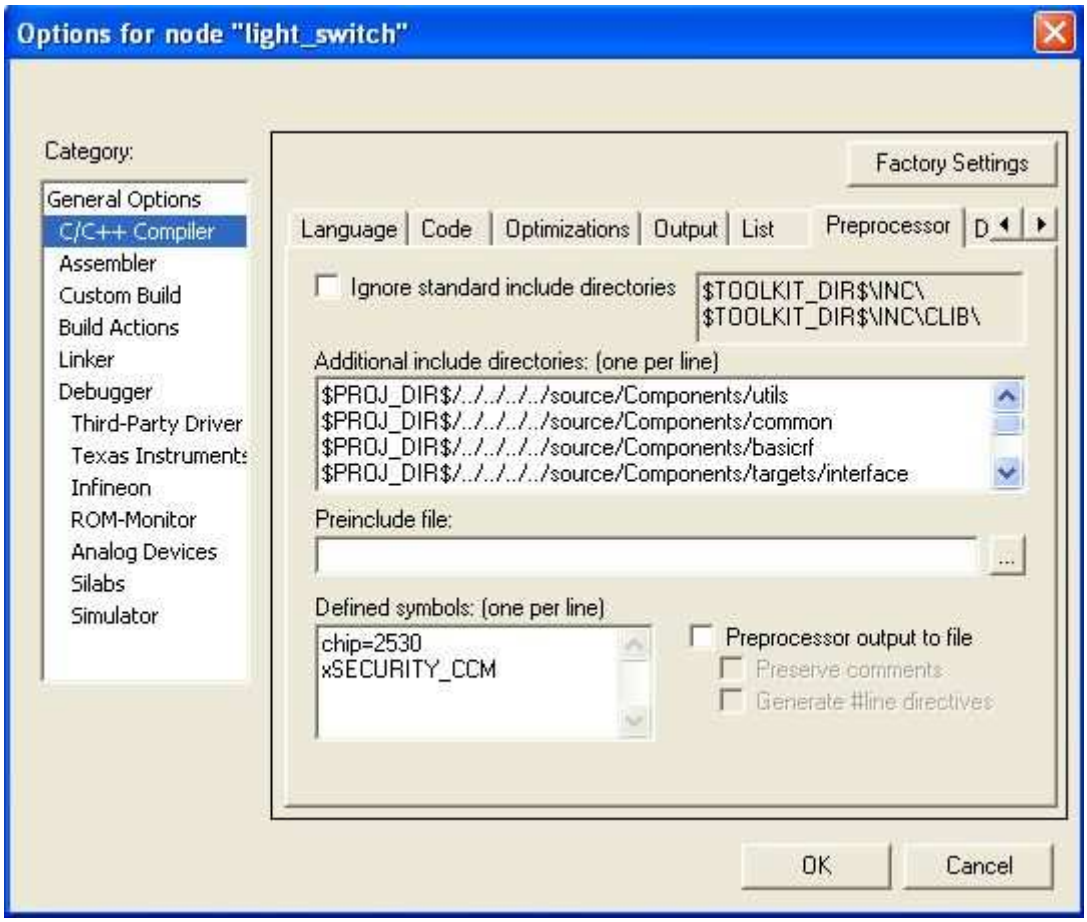

Figure 4 Compile option for CCM security

4.2 Packet Error Rate tester application

This application example requires 2 nodes for operation. Select the project 'per_test'* in the IAR workspace viewer and program both nodes.

**The PER test application is also preprogrammed on both of the CC2530EM boards found in the CC2530DK. The CC2530DK Quick Start Guide describes how to run the preprogrammed PER test out of the box.*

The packet error rate test application sets up a one-way RF link between two nodes. One board will operate as a transmitter and the other board will operate as a receiver. The transmitter node must be configured with the output power to use and the number of packets to transmit as part of the PER test (burst size). During a PER test the receiver node will display the number of received packets, the RSSI level (signal strength) and PER.

The user configurable parameters for the test can be seen in Table 1. These parameters are set using a menu on the LCD during initialization. The menu is navigated with the joystick (see arrows in the display) and the settings are confirmed by pressing Button 1.

Parameter	Settings
Channel	11 – 26 (2405 – 2480 MHz)
Operating Mode	Receiver, Transmitter
TX Output Power	-3 dBm, 0 dBm, 4 dBm
Burst Size	1K, 10K, 100K, 1M
Packet rate	100, 50, 20 or 10 packets per second

Table 1 User configurable parameters

Configure one node as **receiver** and the other node as **transmitter** by following the steps below:

PER Test Receiver configuration:
Perform the following steps to configure the receiver node:

1. Reset the board by cycling power.
2. Press Button 1 to enter the application menu.
3. Select a channel between 11 and 26. Navigate the menu by pressing joystick left or right, and confirm the selection by pressing Button 1. Note the channel, since it will also be used for the Transmitter node.
4. Select operating mode 'Receiver' and confirm with Button 1.
5. The Receiver node is now ready for operation, displaying 'Receiver Ready'.

PER Test Transmitter configuration
Perform the following steps to configure the transmitter node:

1. Reset the board by cycling power.
2. Press Button 1 to enter the application menu.
3. Select the same channel as for the Receiver node. Navigate the menu by pressing joystick left or right, and confirm the selection by pressing Button 1.
4. Select operating mode 'Transmitter' and confirm with Button 1.
5. Select TX output power by navigating the joystick. Confirm with Button 1.
6. Select the number of packets, either 1000, 10K, 100K or 1M. Press Button 1 to confirm.
7. Select packet rate; 100, 50, 20 or 10 packet per second. Press Button 1 to confirm.
8. Push the joystick down to start a PER test. The number of packets specified by burst size will be sent to the Receiver node. Packet Error Rate, RSSI, and number of packets received are displayed on the Receiver's LCD panel.
9. The PER test can be stopped by pressing joystick down again.

Calculation of PER and RSSI
In order to obtain the statistics during the PER test, the receiver maintains the following variables. The variable rxStats is of type perRxStats_t as defined in per_test.h.

rxStats.expectedSeqNum	The expected sequence number for the next packet that should arrive. This is equivalent to the number of received packets+lost packets +1.
rxStats.rssiSum	This is the sum of the RSSI level of the last 32 packets.
rxStats.rcvdPkts	The number of correctly received packets as part of the PER test.
rxStats.lostPkts	The number of packets that have been lost.

Lost packets are detected through a jump in the sequence number. If the received packet has a higher sequence number than *rxStats.expectedSeqNum* the packets in between are calculated as lost. This implies that a series of lost packets will not be detected until a subsequent packet has been received correctly. Packets with errors are considered as lost.

The PER value per thousand packets is calculated by the formula:

PER = 1000* *rxStats.lostPkts*/ (*rxStats. lostPkts+ rxStats. rcvdPkts)*
(for rxStats. rcvdPkts>=1)

The RSSI value is fetched from the first byte following the payload in the received packet. This value is in signed 2's complement and must be corrected with an RSSI offset to get the absolute RSSI value (in dBm). This offset is specified in the CC2530 datasheet. The converted RSSI values of the last 32 (defined by RSSI_AVG_WINDOW_SIZE in per_test.h) received packets are stored in a ring buffer, implementing a moving average filter.

The average RSSI of the 32 last received packets is presented on the LCD of the receiver node during a PER test.

Reset statistics in display
The values of PER, RSSI and number of received packets in the receiver's display can be reset during a PER test by pressing button 1.

4.3 Spectrum Analyzer application

This application example requires one SmartRF05EB/CC2530EM as it merely measures and presents the RSSI values of all IEEE 802.15.4 defined channels in the 2.4 GHz frequency band. The application displays the RSSI values of all channels from 11 (2405 MHz) to 26 (2480 MHz).

The application starts up in **bar graph mode**. In this mode only the bar graphs of the 16 channels are shown. The application displays values in the range -120 dBm to -10 dBm. **Text mode** adds a textual display of the channel number and the measured value for one specific channel whilst still displaying the bar graphs of all channels, albeit with reduced resolution. The user may toggle between the display modes by moving the *joystick up***.** In text mode the channel is selected by moving the joystick left or right.

5 Software Library Reference

This section describes the software libraries the application examples are built upon.

5.1 Software architecture

The design of the software in this package is based on the layered architecture as depicted in Figure 5 below.

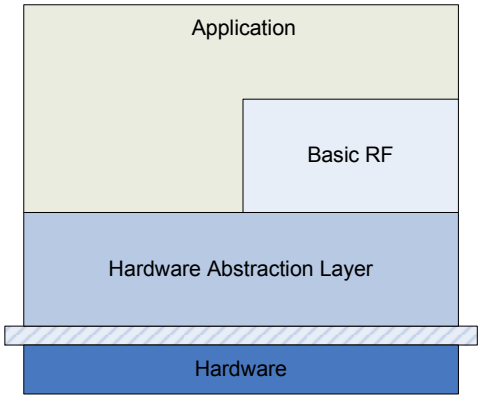

Figure 5 SW architecture

The software implementation consists of the following modules:

- **Application** layer. This software package contains several applications examples with access to Basic RF and HAL.
- **Basic RF**. This layer offers a simple protocol for transmission and reception using a two-way RF link.
- **Hardware Abstraction Layer**. Contains functionality for access to the radio and onboard peripherals modules like LCD, UART, joystick, buttons, timers etc.

A detailed description of the Basic RF protocol is found in section 5.2. The Hardware Abstraction Layer is described in section 5.3.

5.1.1 Software folder structure

The software and documentation in this package is organized in the folder structure shown in Figure 6. The documentation is found in the *docs* folder. The workspace file is found in the *ide* folder. Source code for the different applications can be found in the folder *source/Apps*. The *components* folder includes source code for the different components used by the applications. The HAL and Basic RF source code components are found under the *components* folder.

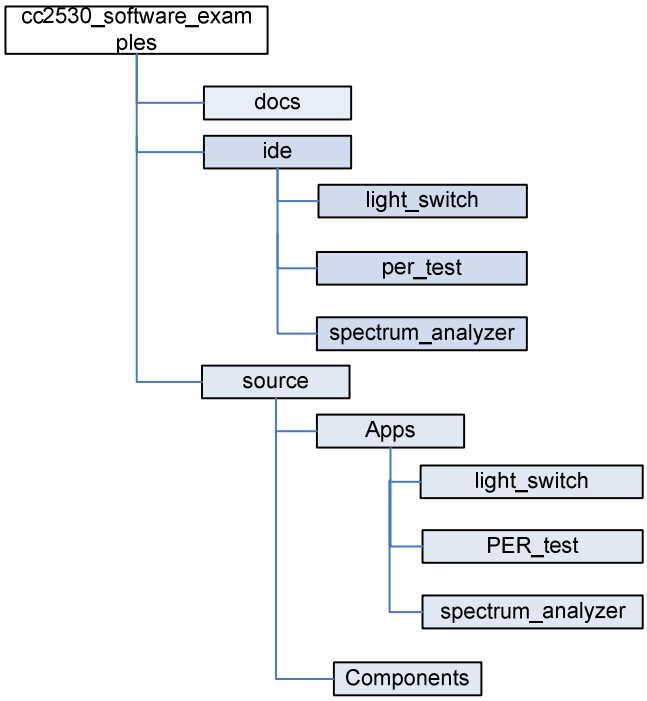

Figure 6 Software folder structure

5.2 Basic RF

The Basic RF layer offers a simple protocol for transmission and reception using a two-way RF link. The Basic RF protocol offers service for packet transmission and reception. It also offers secure communication by use of CCM-64 authentication and encryption/decryption of packets. The security features of Basic RF can be compiled in by defining the compile switch SECURITY_CCM in the project file. The compile time inclusion of security features is done to save code space for the applications where security features are not needed.

The protocol uses IEEE 802.15.4 MAC compliant data and acknowledgment packets. However it does not offer a full MAC layer, only a simple data link layer for communication between two nodes. See also section 5.2.5 for limitations of Basic RF.

Basic RF contains only a small subset of the 802.15.4 standard:

- Association, scanning or beacons are not implemented
- No defined coordinator/device roles (peer-to-peer, all nodes are equal)
- No packet retransmission. This must be taken care of by the layer above Basic RF.

5.2.1 Basic RF frame format

Octets: 1	2	1	2	2	2	5	Variable	2
Length Byte	Frame Control	Sequence number	Dest. PAN ID	Dest. Address	Source Address	Aux.Sec. Header	Frame payload	FCS

Figure 7 Basic RF frame format

The frame format of the Basic RF protocol is shown in Figure 7. The first byte is a length byte. The length byte itself is not counted in the length. The frame control field is set according to the IEEE Std. 802.15.4-2006. Please refer to section 7.2 in the IEEE Std. 802.15.4-2006 [1].

The sequence number is an 8 bit value starting on 0 for the first packet transmitted after initialization. The values of the destination PAN ID, destination address and source address fields are configured by the application as part of the basic RF initialization. Please refer to the Basic RF API reference section 5.2.3 for further information.

The auxiliary security header is only included in the frames when the security features of Basic RF is used i.e when the compile option SECURITY_CCM is set in the project file. This field is 5 bytes long and consists of a security control byte that defines the level of protection applied to this frame, and frame counter. The Basic RF protocol supports only one security mode: ENC-MIC-64 i.e. encryption and authentication with 64 bits Message Integrity Code. For this mode the Security Control field is set to 0x06 according to IEEE Std. 802.15.4-2006 [1]. The frame counter field of the auxiliary security header is 4 bytes long and is used for replay protection of the frame. The value of the frame counter field is set to 0 on initialization and incremented for each transmitted packet.

The frame payload is variable in length and consists of data sent from the layer above Basic RF. The maximum length of this field is 103 Bytes.

The Frame Check Sequence field (FCS) is 2 bytes long. This field is automatically appended by the radio chip, and is not taken care of by the Basic RF layer. When a frame is received the first byte of FCS is replaced with the RSSI value in the RX FIFO on the radio.

5.2.2 Basic RF usage instructions

Startup

1. Make sure that the board peripherals and radio interface is initialized i.e. halBoardInit() must have been called first.
2. Create a basicRfCfg_t structure, and initialize its members. If the security features of Basic RF are used, the higher layer is responsible for allocating and assigning the 16 bytes key.
3. Call basicRfInit() to initialize the packet protocol.

Transmission:

1. Create a buffer with the payload to send. Maximum payload size for Basic RF is 103 Bytes.
2. Call basicRfSendPacket(). Check the return value.

Reception:

1. Perform polling by calling basicRfPacketIsReady() to check if a new packet is ready to be received by the higher layer.
2. Call basicRfReceive() to receive the packet by higher layer. The caller is responsible for allocating a buffer large enough for the packet and 2 Bytes buffer space for the RSSI value.

By calling basicRfReceiveOn() the radio receiver is kept on all the time. This is done for nodes that need to be able to receive packets at any time. The drawback is a higher current consumption.

By calling basicRfReceiveOff() the radio receiver is turned off.

5.2.3 Basic RF API reference

Include files

basic_rf.h
basic_rf_security.h

Compile time configuration

In order to use the security features of basic RF the following compile option must be set in the project files:
SECURITY_CCM

Defining this compile switch enables the security features of Basic RF. All outgoing packets will be authenticated and encrypted with ENC-MIC-64 CCM*. Likewise it will be assumed that all incoming packets are authenticated and encrypted the same way.

When this compile flag is set the higher layer is responsible for allocation of a 16 bytes key for security operations.

Data structures

The following data structure is used for Basic RF configuration:

```
typedef struct {
  uint16 myAddr;
  uint16 panId;
  uint8 channel;
  uint8 ackRequest;
  #ifdef SECURITY_CCM
  uint8* securityKey;
  uint8* securityNonce;
  #endif
} basicRfCfg_t;
```

uint16 myAddr – 16-bit short address (This node's address)
uint16 panId – PAN ID (ID of the Personal Area Network this node is operating on)
uint8 channel – RF Channel (must be set between 11 and 26)
uint8 ackRequest – Set true to request acknowledgement from destination
uint8* securityKey – Pointer to the security key buffer allocated by the caller
uint8* securityNonce – Pointer to the security nonce buffer. This is not used by the caller.

The securityKey and securityNonce members of the structured are only compiled in if the compile switch SECURITY_CCM is enabled.

Functions

void basicRfInit(basicRfCfg_t* pRfConfig)

Initialise basic RF datastructures. Sets channel, short address and PAN ID in the chip and configures interrupt on packet reception. The board peripherals and radio interface must be called before this function with the function halBoardInit().

uint8 basicRfSendPacket(uint16 destAddr, uint8* pPayload, uint8 length)

Send packet to the given destination short address. Returns TRUE if packet was sent successfully, and FAILED otherwise. If ackRequest is TRUE the return value of this function will only be TRUE if an acknowledgment is received from the destination.

uint8 basicRfPacketIsReady(void)

Returns TRUE if a received packet is ready to be retrieved by higher layer.

int8 basicRfGetRssi(void)

Returns the RSSI value of the last received packet

uint8 basicRfReceive(uint8* pRxData, uint8 len, int16* pRssi)

Retrieve packet from basic RF layer. The caller is responsible for allocating buffer space for data and the RSSI value.

void basicRfReceiveOn(void)

Turn on receiver on radio. After calling this function the radio is kept on until basicRfReceiveOff is called.

void basicRfReceiveOff(void)

Turn off receiver on radio, and keep it off unless for transmitting a packet with Clear Channel Assessment.

Security Interface

void basicRfSecurityInit(basicRfCfg_t* pConfig)

Initialises security key and nonces

5.2.4 Basic RF operation

This section will describe how the Basic RF and the HAL operate during initialization, packet transmission and reception. This section assumes that the radio transceiver is a CC2530.

Initialization

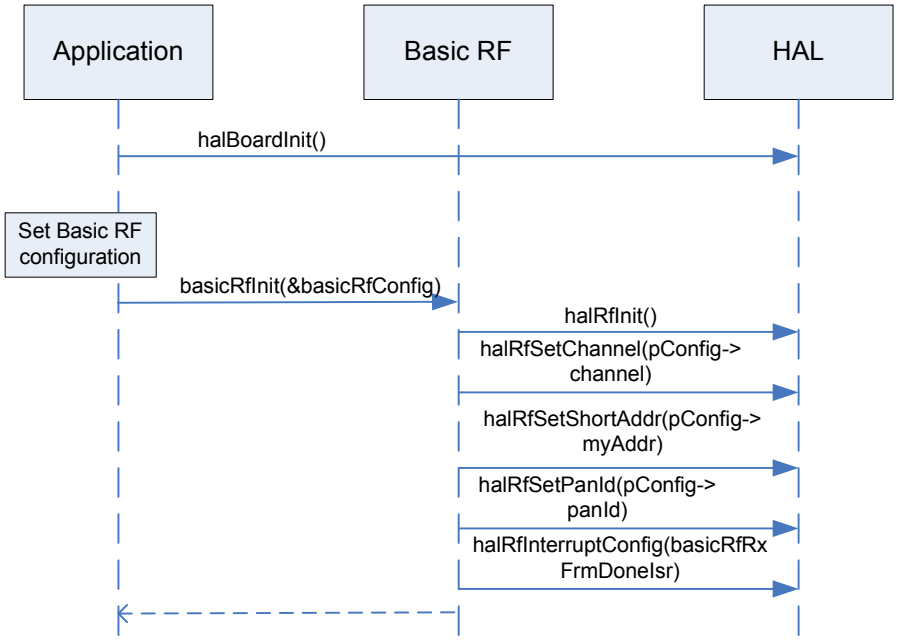

Figure 8 Initialization

Figure 8 illustrates the sequence of calls during initialization of Basic RF and the HAL. The application is responsible for calling halBoardInit() to initialize the hardware peripherals and configure IO ports.

The application must then initialize an instance of the basicRfCfg_t struct (see section 5.2.3).The application will then call basicRfInit() with the address to the instance of this struct as parameter. The basicRfInit() function calls the function halRFInit() which configures the radio with recommended register settings and enabled the receive interrupt. In addition the basicRfInit() function sets up the channel, short address and PAN ID to the CC2530, and register the interrupt service routine to handle the received packet interrupt from the radio.

Packet transmission

Figure 9 illustrates the sequence of function calls for a packet transmission scenario with Basic RF. In this scenario the security features of Basic RF are disabled.

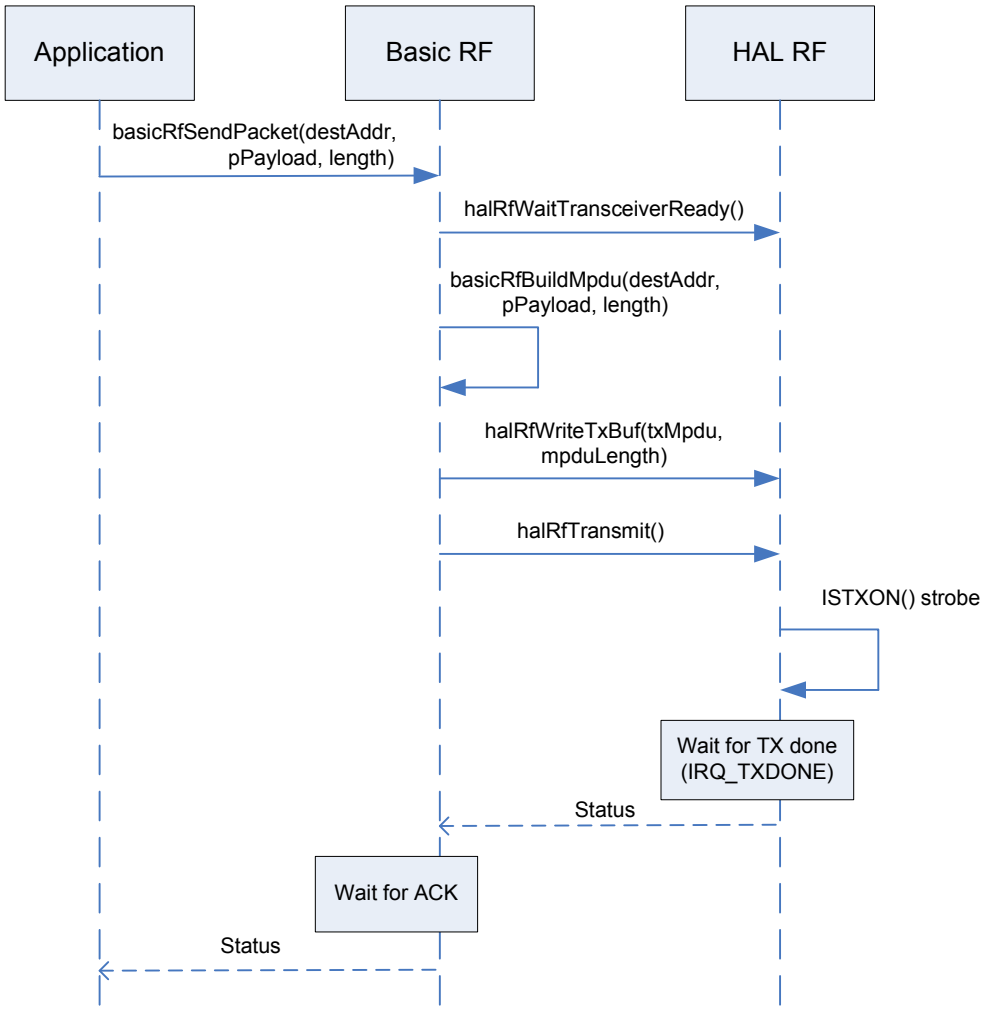

Figure 9 Packet transmission

With reference to Figure 9 these are the steps that are performed during a Basic RF packet transmission with acknowledgement request.

1. The application prepares the payload to be sent and calls the function basicRfSend() with the 16 bit destination address, a reference to the payload buffer and the length of the payload buffer as arguments.
2. Basic RF waits for the radio to be idle by calling halRfWaitTransceiverReady(). This function checks that the SFD is not active.
3. Basic RF calls the function basicRfBuildMpdu(). Basic RF keeps an internal buffer for the outgoing MPDU. It will first build the header with the correct address and header information,

and then it will copy the payload from the application over to the remaining part of the internal buffer.

4. Basic RF will then call halRfWriteTxBuf() to write the MPDU to the CC2530 TX FIFO buffer.
5. The function halRfTransmit() is called to transmit a packet with on the air. This function will issue the command strobe ISTXON() to send the packet in the TX FIFO.
6. The function halRfTransmit() will wait until the packet transmission is finished (IRQ_TXDONE flag is set).
7. Basic RF will then wait for receiving the acknowledgement packet in the function basicRfSend().
8. If the ACK is successfully received within a predefined waiting time, basicRfSend() returns with status SUCCESS to the application.

Packet reception

The sequence of function calls for a packet reception scenario with security features disabled is illustrated with Figure 10.

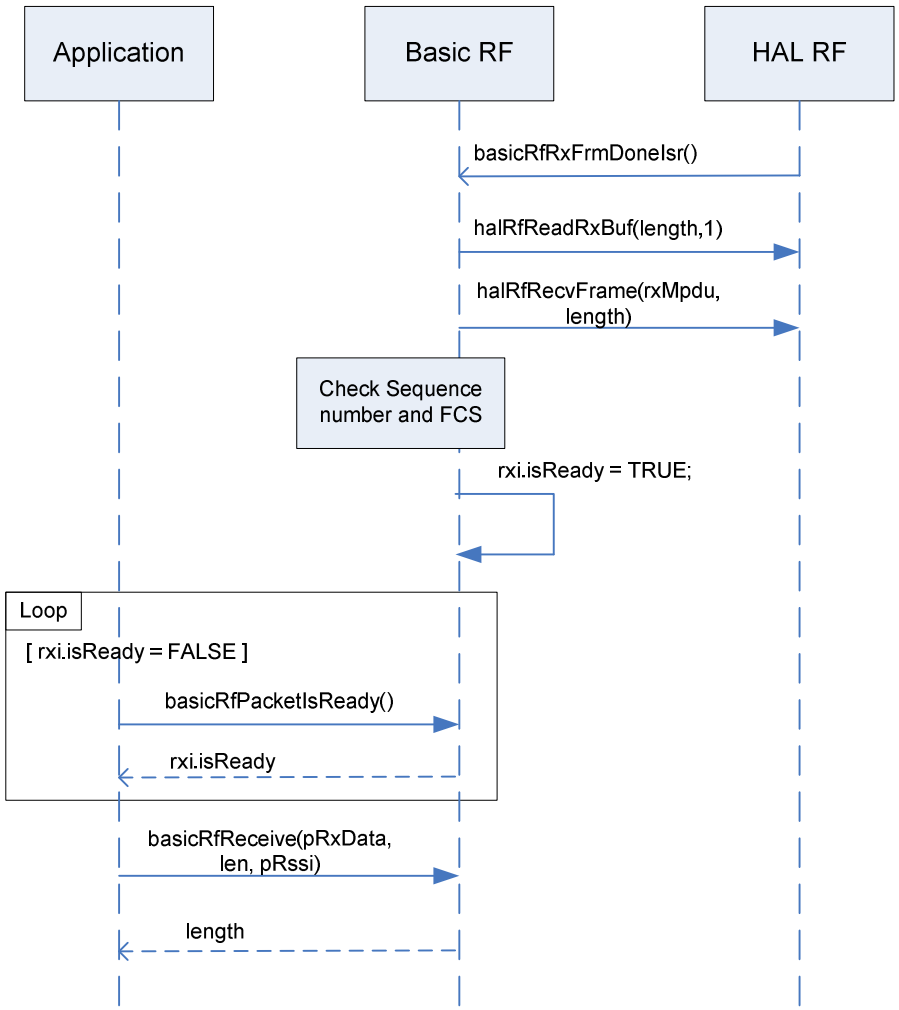

Figure 10 Packet reception

With reference to Figure 10 these are the steps that are performed during a packet reception with acknowledgement request.

1. When a new packet is completely received the RX packet done (RXPKTDONE) interrupt is issued and the rfIsr interrupt service routine function in the HAL_RF module is called. This leads to the basicRfRxFrmDoneIsr() interrupt service routine being invoked. See Figure 10.
2. The length of the received frame is read out from the first byte in the RX buffer on CC2530. This is done with the call halRfReadRxBuf(length, 1) in Figure 10.
3. The complete packet is read out from the RX buffer with the call halRfRecvFrame(rxMpdu, length). The incoming packet is stored in the internal data buffer rxMpdu.
4. The CC2530 automatically sends the ACK when AUTOACK is enabled and the incoming frame is accepted by the address recognition with the acknowledgement flag set and the CRC is correct. See the CC2530 datasheet for how AUTOACK is enabled [2].
5. The FCS field and the sequence number of the packet are checked. If they are as expected the rxi.isReady flag is set TRUE to indicate that a new packet is received.
6. The application will poll this flag in a loop by calling the function basicRfPacketIsReady().
7. When basicRfPacketIsReady() return TRUE the application calls the function basicRfReceive() to retrieve the payload and RSSI from the new incoming packet.
8. The function basicRfReceive() copies the payload over to the memory location pRxData in the argument list. The number of bytes actually copied is returned. The RSSI value in dBm is also copied over to the memory location pRssi in the argument list.

Note that this implementation uses RF interrupt for packet reception handling. DMA can also be used for packet reception handling which might be more efficient in some cases. In this case the DMA will be set up to trigger a memory transfer when a packet is received.

5.2.5 Limitations of Basic RF

Basic RF is only meant to serve as a simple example of how to use the chip. **It is not a complete protocol layer ready to be used in a commercial product.**

- Basic RF is not a complete data link or MAC layer protocol.
- Basic RF does not have full error handling support. As an example RX FIFO overflow handling is not implemented, and such an error will cause the software to stall.

It is recommended to use either TIMAC or SimplicTI instead of Basic RF for commercial product development.

SimpliciTI is a simple protocol aimed at small RF networks.

TIMAC is an IEEE 802.15.4 compliant MAC layer software implementation aimed for standardized solutions.

More info is found on the following web pages:
www.ti.com/simpliciti
www.ti.com/timac

5.3 Hardware Abstraction Layer

5.3.1 HAL RF API reference

Include files

hal_rf.h
hal_rf_security.h

Functions

uint8 halRfInit(void)

Powers up the radio, configures the radio with recommended register settings, enables autoack and configures the IO on the radio. This function must be called after halBoardInit().

uint8 halRfSetPower(uint8 power)

Set TX output power

uint8 halRfTransmit(void)

Transmit frame

void halRfSetGain(uint8 gainMode)

Set gain mode. This is only used if external LNA/PA is used.

uint8 halRfGetChipId(void)

return radio chip id register

uint8 halRfGetChipVer(void)

Return radio chip version register

uint8 halRfGetRandomByte(void)

Return random byte.

uint8 halRfGetRssiOffset(void)

Return RSSI offset for radio.

void halRfWriteTxBuf(uint8* data, uint8 length)

Write the number of bytes given by *length* from the memory location pointed to by the pointer *data* to the radio TX buffer.

void halRfReadRxBuf(uint8* data, uint8 length)

Read the number of bytes given by *length* from radio RX buffer to the memory location pointed to by the pointer *data*. The radio status byte is returned.

void halRfWaitTransceiverReady(void)

Wait until the transceiver is ready

void halRfReceiveOn(void)

Turn on receiver on radio.

void halRfReceiveOff(void)

Turn off receiver on radio.

void halRfDisableRxInterrupt(void)

Clear and disable RX interrupt.

void halRfEnableRxInterrupt(void)

Enable RX interrupt.

void halRfRxInterruptConfig(ISR_FUNC_PTR pf)

Configure RX interrupt, and setting the function to be called on interrupt.

void halRfSetChannel(uint8 channel)

Set RF channel. Channel must be in the range 11-26.

void halRfSetShortAddr(uint16 shortAddr)

Write 16 bit short address to the radio.

void halRfSetPanId(uint16 PanId)

Write 16 bit PAN ID to the radio.

Security Interface

void halRfSecurityInit(uint8* key, uint8* nonceRx, uint8* nonceTx)

Write 16 bit nonces and key to to the radio from the memory locations pointed to by *key*, *nonceRx* and *nonceTx*.

uint8 halRfReadRxBufSecure(uint8* pData, uint8 length, uint8 encrLength, uint8 authLength, uint8 m)

Read out RX buffer from radio with CCM authentication and decryption

void halRfWriteTxBufSecure(uint8* pData, uint8 length, uint8 encrLength, uint8 authLength, uint8 m)

Write to TX buffer in radio with CCM authentication and encryption.

void halRfIncNonceTx(void)

Increment the frame counter field of the nonce used for outgoing packets. Refer to IEEE Std. 802.15.4-2006 [1] for a description of this field.

References

[1] **IEEE Std. 802.15.4-2006, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)**

[2] **CC2530 datasheet**

Document History

Revision	Date	Description/Changes
-	2009.03.18	Initial version