

Porting of RF Blinking LED Software Example to CC2420 - MSP430

By J. Hønsi

1 Keywords

- *MSP430*
- *MSP430F1611*
- *IEEE 802.15.4*
- *CC2420*
- *RF Blinking LED Software Example*
- *CC2420EM*
- *CC2420DB*
- *IAR Embedded Workbench*
- *GNU*

2 Introduction

This application example demonstrates the basic use of Texas Instruments' **CC2420** ZigBee-ready RF transceiver together with the MSP430F1611 MCU from Texas Instruments [1].

The example shows how to interconnect the MSP430F1611 and the **CC2420** for RF transmission and reception, using a subset of the IEEE 802.15.4 protocol.

The software handles MCU configuration, transceiver configuration and basic data flow. The software is an adapted version of the RF Blinking LED software example provided with the CC2420DBK Demonstration Board Kit (see [3] and [5]).

The hardware used in the example consists of an MSP-FET430 Development Tool from Texas Instruments equipped with an MSP430F1611 MCU.

This kit is connected to the **CC2420** hosted on the CC2400EB/CC2420EM boards from Texas Instruments. An auxiliary CC2420DB should be used to provide another IEEE 802.15.4 node to test the RF communication against.

The software, which is available for download from Texas Instruments' web pages, is written to be compatible with the IAR C/C++ compiler and the MSP430-GCC compiler from GNU.

Table of Contents

1	KEYWORDS	1
2	INTRODUCTION	1
3	DESCRIPTION	3
4	HARDWARE	3
4.1	CC2400EB EVALUATION BOARD	4
4.2	CONNECTING MSP-TS430PM64 AND CC2400EB	6
4.3	IEEE 802.15.4 PACKET SNIFFER	8
4.4	SPI BUS COMMUNICATION	9
5	SOFTWARE	13
5.1	TOOLS	13
5.1.1	<i>IAR Studio</i>	14
5.1.2	<i>GNU Tools</i>	15
5.2	LIBRARIES AND EXAMPLES	15
5.2.1	<i>Hardware Definition Files</i>	15
5.2.2	<i>Hardware Abstraction Library (HAL)</i>	16
5.2.3	<i>Basic RF Library</i>	16
5.2.4	<i>Software Example</i>	16
5.2.5	<i>Running the example</i>	17
5.3	SOURCE FILES	17
5.4	MODIFYING THE EXAMPLE	18
5.4.1	<i>Adding another LED</i>	18
5.4.2	<i>Adding a Header File</i>	18
5.4.3	<i>Adding a Source File</i>	19
5.4.4	<i>Using a Different Compiler</i>	19
6	PROTOCOL	20
6.1	DATA PACKET DESCRIPTION	20
6.2	PACKET FORMAT	21
7	REFERENCES	22
8	GENERAL INFORMATION	23
8.1	DOCUMENT HISTORY	23

3 Description

The application described in this document demonstrates RF communication between two microcontroller systems using the **CC2420** RF transceiver and a subset of the IEEE 802.15.4 protocol. The application described here uses an MSP430F1611 microcontroller from Texas Instruments connected to a CC2420 RF Transceiver from Texas Instruments. The two IC's are interfaced through a CC2400EB Evaluation Board and the CC2420EM Evaluation Module.

As a standalone set-up, only basic initialisation and RF transmission can be demonstrated, so in order to test reception it is necessary to set up a peer node. For this purpose it is recommended to use the **CC2420DB Demonstration Board** with the Blinking LED Software Example (pre-programmed on the board). Using this set-up, there is no need to modify the source code provided with this application note.

In cases where there is a need to look at RF packets, the Texas Instruments Packet Sniffer [6] can be used. This requires an extra CC2400EB/CC2420EM pair.

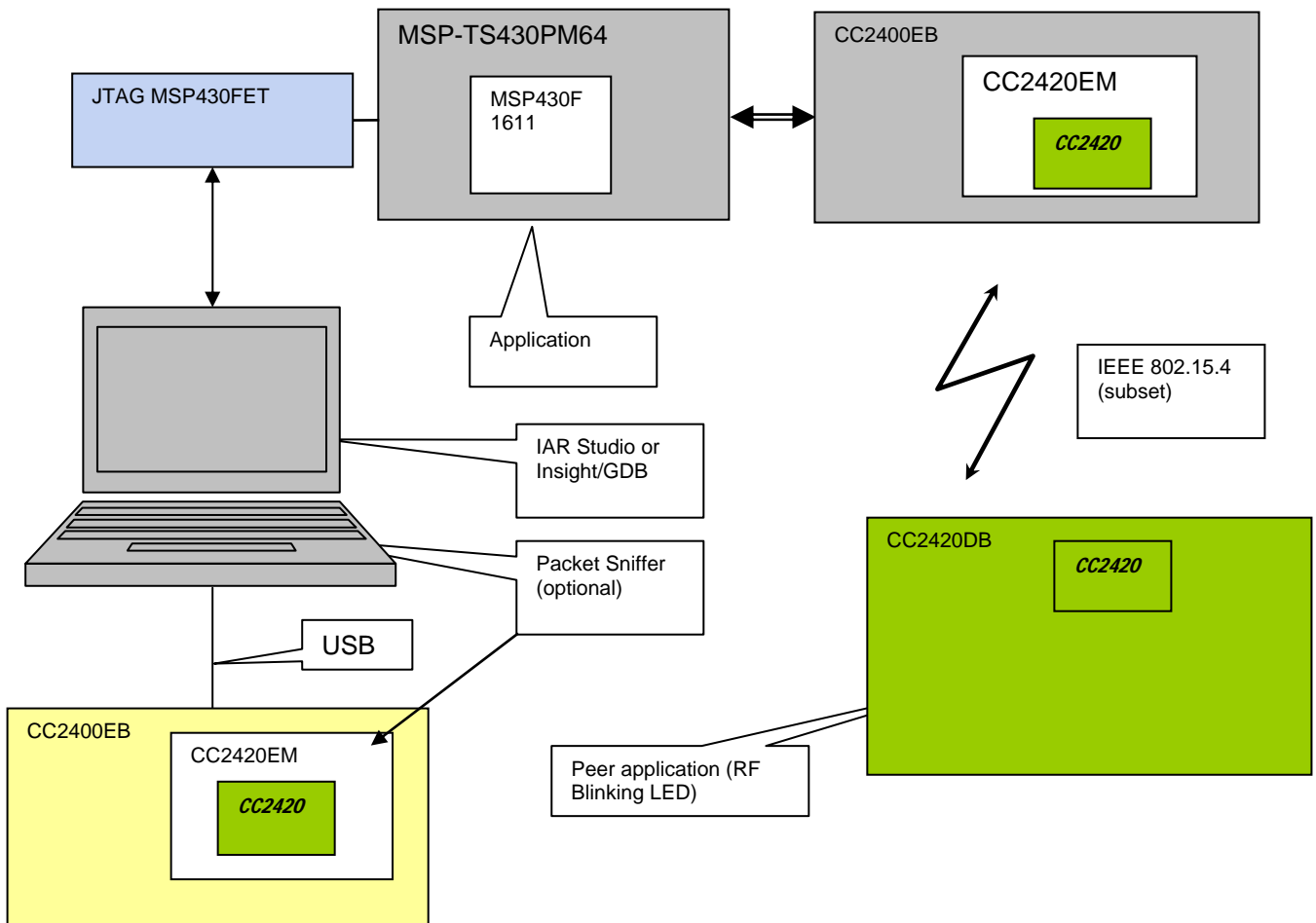


Figure 1. Application Note AN033 Overview

4 Hardware

The MSP430F1611 microcontroller from Texas Instruments performs all the control functions in the application. The microcontroller runs on the internal crystal. It interfaces to the **CC2420** using General Purpose IO ports, and the built-in SPI interface.

Application Note DN033

The MSP430F1611¹ is a 16-bit controller with 48+256 kB of Flash Memory and 10kB of RAM. The FLASH is programmed via the JTAG interface.

The microcontroller can be programmed using a JTAG module, MSP430FET, available from TI. For further details, consult the documentation from Texas Instruments. The JTAG module is connected to the parallel port on the PC used for development, and integrates with IAR Studio and the Insight/Proxy-GDB tools from GNU.

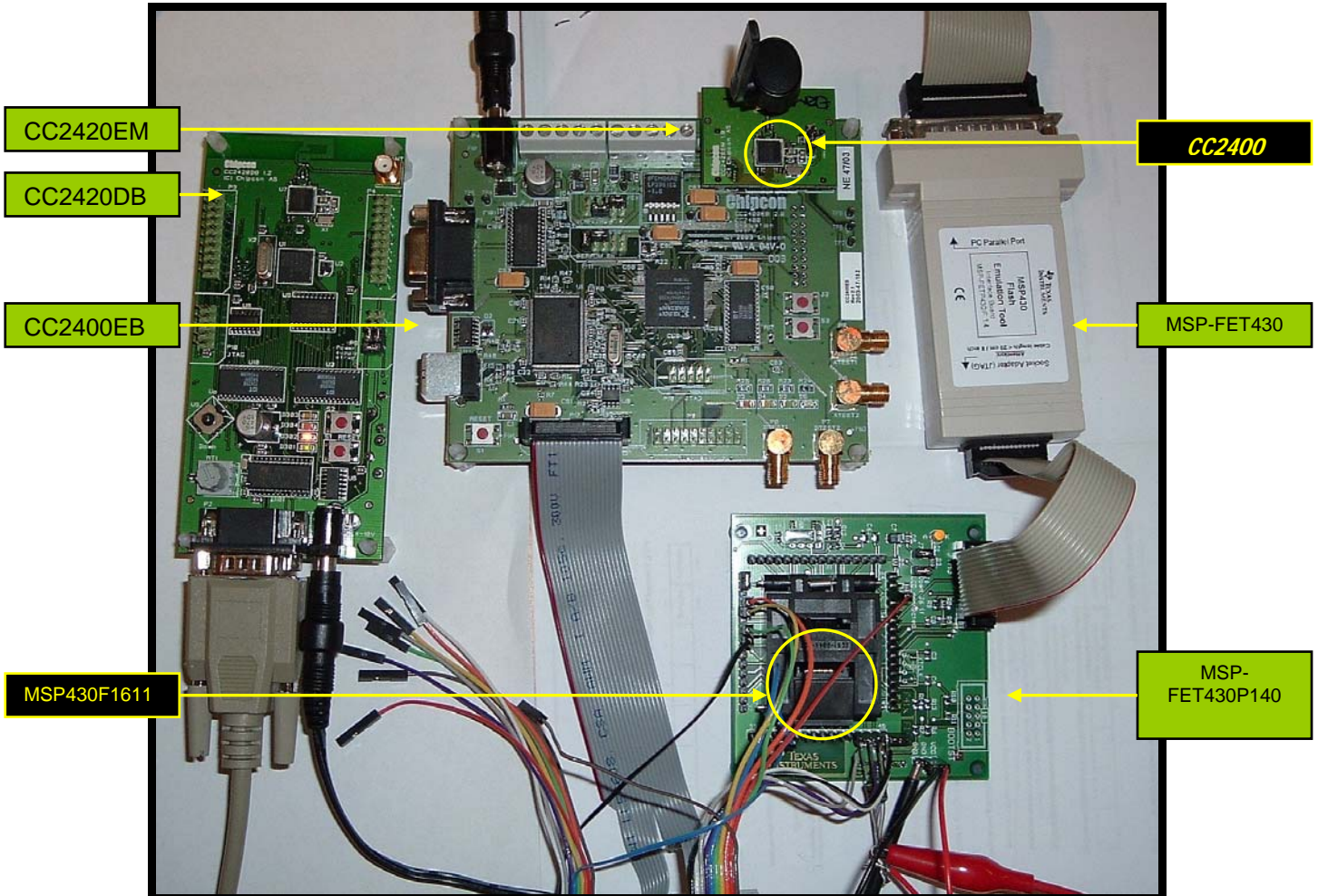


Figure 2. Application Note AN033 Hardware Overview

4.1 CC2400EB Evaluation Board

The CC2400EB Evaluation Board, which is used as a motherboard for the CC2420EM in this example, interfaces the MSP430F1611 and the *CC2420* radio transceiver.

¹ This controller has processing and memory capacity, which exceeds the requirement of the blinking LED software example, so this software could execute on smaller microcontrollers

Application Note DN033

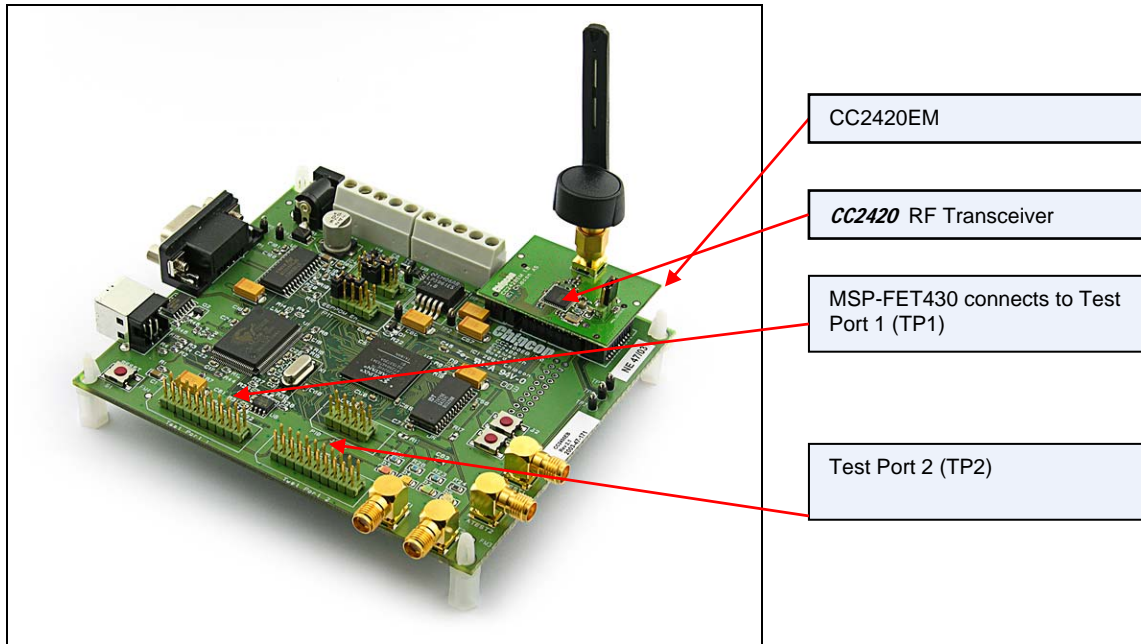


Figure 3. CC2400EB Evaluation Board with CC2420EM Mounted

Connecting the MSP-FET430 development kit to the CC2400EB through Test Port 1 is sufficient for performing prototyping for this example. The FPGA must then be programmed using the *SmartRF® Studio* “Load FPGA Configuration” function at startup. The “fpga_cc2420_uc_prototyping_1_0.bin” FPGA file is downloadable from the Texas Instruments website. All LEDs will be turned off after programming the FPGA. The FPGA will give the microcontroller access to all *CC2420* digital pins through Test Port 1. The signals are duplicated on Test Port 2.

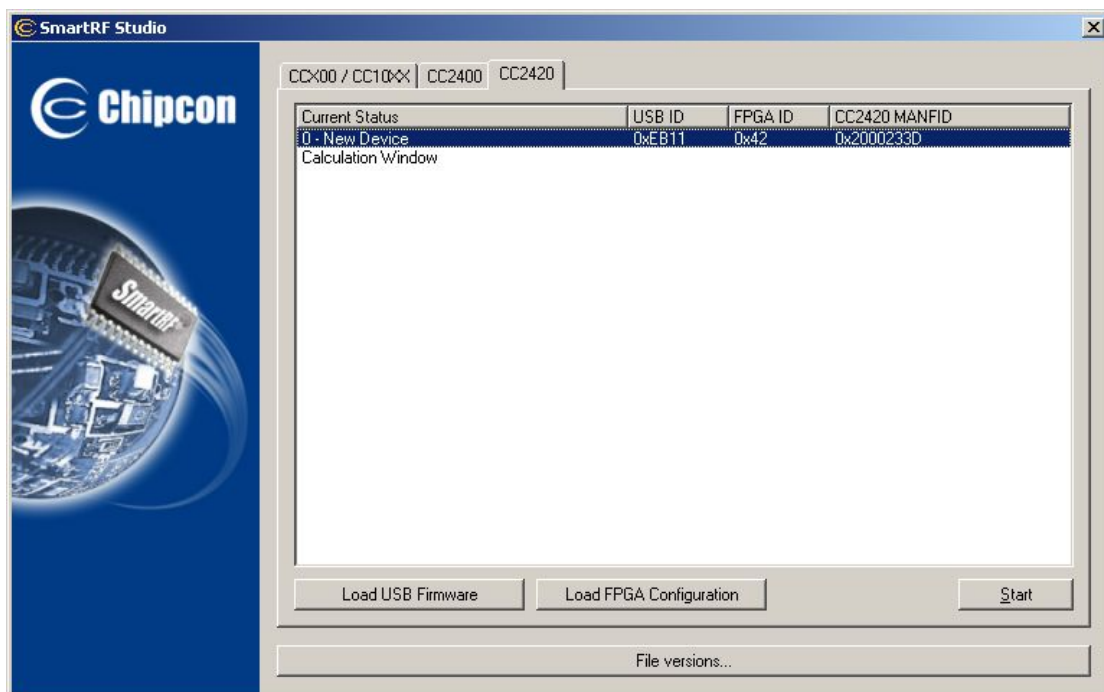


Figure 4. Loading FPGA Configuration for Prototyping, Step 1

Application Note DN033

Procedure for loading FPGA for prototyping:

1. Start SmartRF® studio
2. Connect the CC2400EB to the PC via USB
3. Observe that the LEDs on CC2400EB start blinking
4. Select the **CC2420** tab
5. Wait for “New Device” to appear
6. Select “New device” and click “Load FPGA Configuration” button
7. Select the file “fpga_cc2420_uc_prototyping_1_1.bin”, and click OK

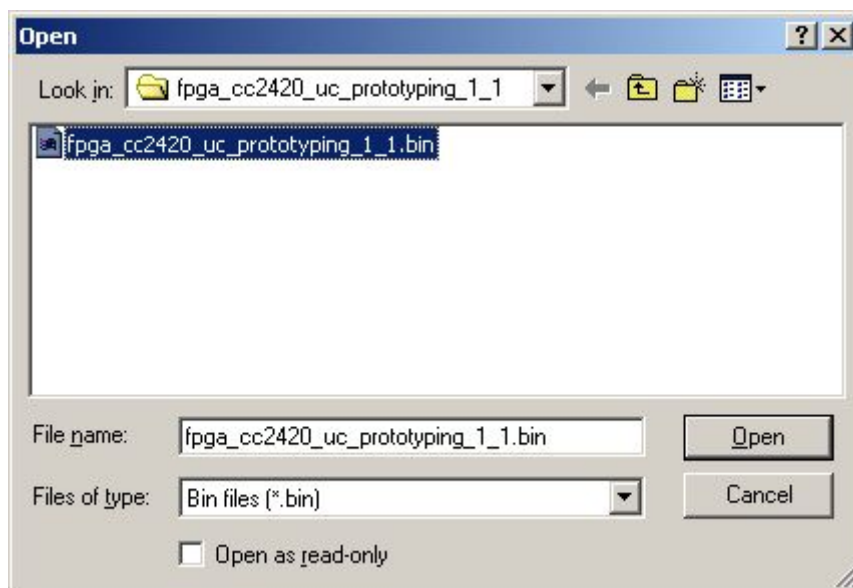


Figure 5. Download FPGA for Prototyping, Step 2

The LEDs on the CC2400EB should now stop blinking and the board is ready to be used for prototyping.

4.2 Connecting MSP-TS430PM64 and CC2400EB

The MCU and the RF transceiver communicate via the SPI bus and some auxiliary signals as described in Table 1.

Application Note DN033

CC2420 Signal	CC2400EB Pin# (TP1)	Direction	MCU Signal	MCU Pin#
FIFOP	4	TO uC	GPIO P2.4	24
SFD	8	TO uC	GPIO P4.0	36
CSn ²	12	From uC	GPIO P5.4	48
SCLK ²	13	From uC	GPIO P5.3	47
SI ²	14	From uC	GPIO P5.1	45
SO ²	15	TO uC	GPIO P5.2	46
CCA	16	TO uC	GPIO P4.1	37
FIFO	17	TO uC	GPIO P2.3	23
RESETn	18	From uC	GPIO P2.1	21
VREG_EN	19	From uC	GPIO P2.0	20
GND	20		GND	62

Table 1. MSP430F1611 and CC2420 Inter-connection

The signals are described in the *CC2420* Data Sheet [2].

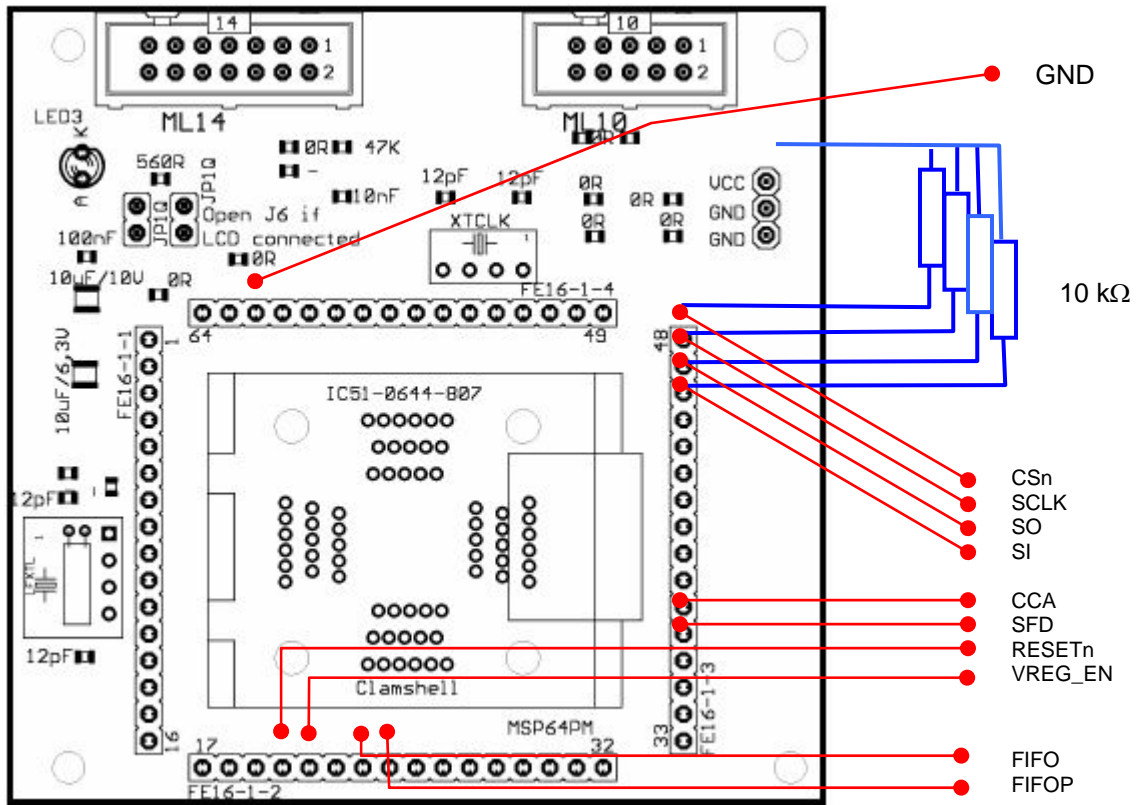


Figure 6. MSP430PM64 Target Socket Module and Connections to CC2400EB

² Pull-up to Vcc (3.0 V) 10 kΩ

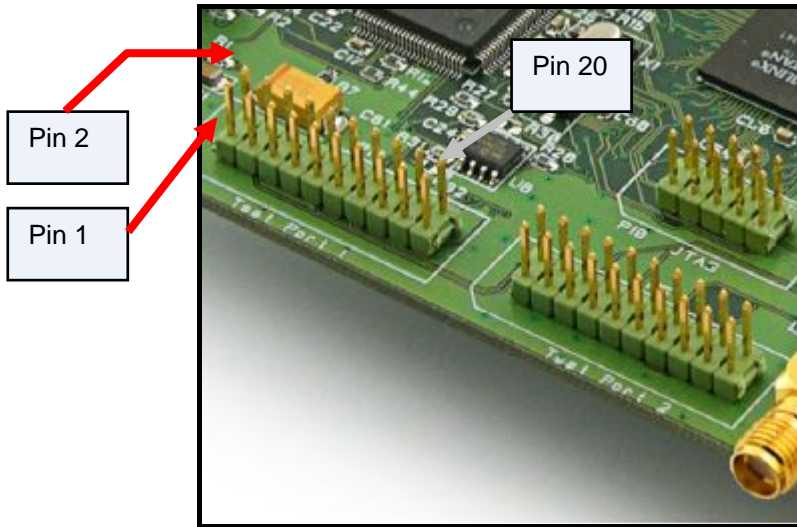


Figure 7. CC2400EB Test Port 1

4.3 IEEE 802.15.4 Packet Sniffer

The IEEE 802.15.4 Packet Sniffer from Texas Instruments may optionally be used to view the RF data traffic between the nodes used in the example. This tool can be downloaded from Texas Instruments' web site, and also requires a CC2400EB with a CC2420EM mounted.

Connect a CC2400EB/CC2420EM to your PC via the provided USB-cable and apply power. Start the packet sniffer and wait for the CC2400EB to appear, select as shown in Figure 8 and then choose channel 0x1a (2480 MHz). F5 and F6 start and stop data acquisition. Channel 26 is the IEEE 802.15.4 RF channel used in this example.

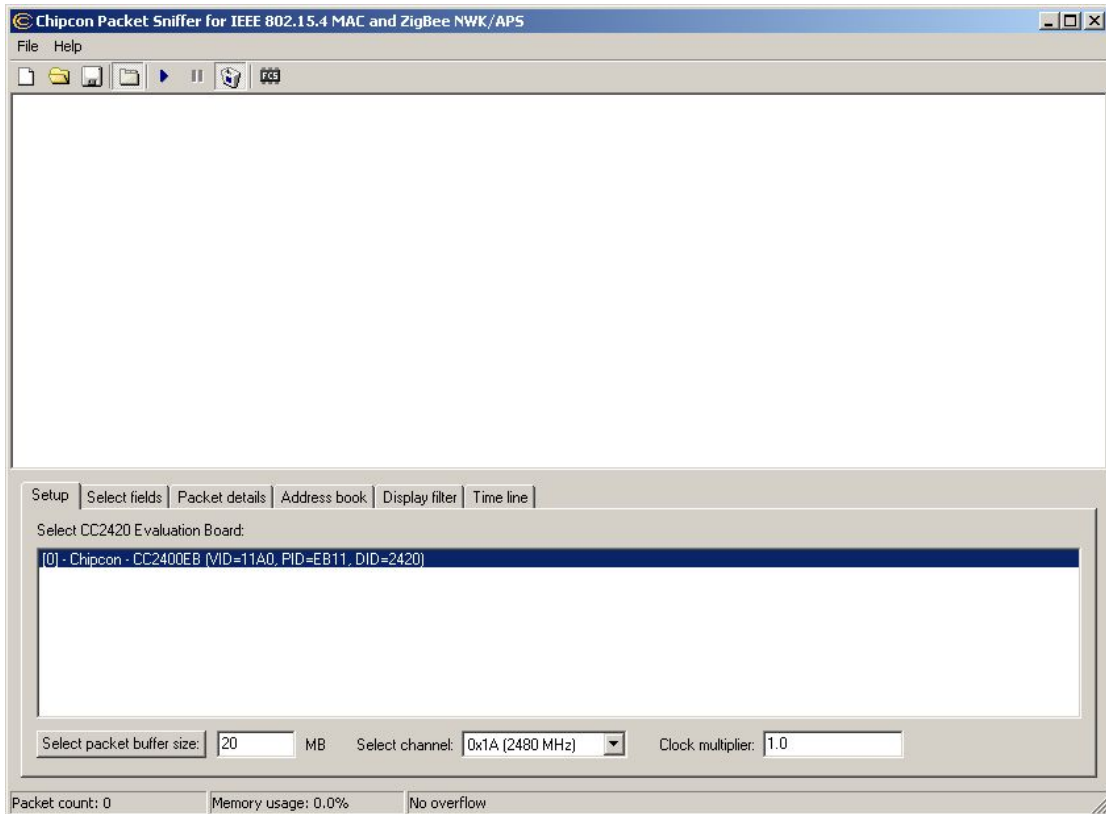


Figure 8. Packet Sniffer Start-up

Application Note DN033

NB! The sniffer can not use the same CC2400EB board as the MSP430.

The screenshot shows the 'Chipcon Packet Sniffer for IEEE 802.15.4 MAC and ZigBee NWK/APS' window. The main display area contains a list of captured packets. Each packet entry includes a time stamp, length, frame control field details (Type, Sec, Pnd, Ack, req, Intra, PAN), sequence number, destination PAN and address, source address, MAC payload (highlighted in yellow), LQI, and FCS.

Time (us)	Length	Frame control field	Sequence number	Dest. PAN	Dest. Address	Source Address	MAC payload	LQI	FCS
+0 =0	21	Type: DATA, Sec: 0, Pnd: 0, Ack: 1, req: 1, Intra: 1, PAN: PAN	0x15	0x2420	0x1234	0x5678	1A 00 00 00 FF FF FF FF FF FF	156	OK
+1061 =1061	5	Type: ACK, Sec: 0, Pnd: 0, Ack: 0, req: 0, Intra: 0, PAN: PAN	0x15					152	OK
+4135810 =4136871	21	Type: DATA, Sec: 0, Pnd: 0, Ack: 1, req: 1, Intra: 1, PAN: PAN	0x18	0x2420	0x5678	0x1234	B4 CC CC CC CC CC CC CC CC CC	148	OK
+1060 =4137931	5	Type: ACK, Sec: 0, Pnd: 0, Ack: 0, req: 0, Intra: 0, PAN: PAN	0x18					156	OK
+50882 =4188813	21	Type: DATA, Sec: 0, Pnd: 0, Ack: 1, req: 1, Intra: 1, PAN: PAN	0x19	0x2420	0x5678	0x1234	B4 CC CC CC CC CC CC CC CC CC	148	OK

Below the packet list is a 'Setup' section with tabs for 'Select fields', 'Packet details', 'Address book', 'Display filter', and 'Time line'. The 'Address book' tab is active, showing a table of nodes:

Node name	PAN id	Short address	IEEE extended address
CC2420DB	0x2420	0x1234	0xFFFFFFFFFFFFFFFF
MSP430	0x2420	0x5678	0xFFFFFFFFFFFFFFFF

At the bottom of the window, status information is displayed: Packet count: 8, Memory usage: 0.0%, No overflow.

Figure 9. Packet Sniffer Display

4.4 SPI Bus Communication

The SPI bus is used to transfer data between the MSP430 and **CC2420**. These snapshots demonstrate Strobe, Read and Write accesses to **CC2420** registers.

Application Note DN033

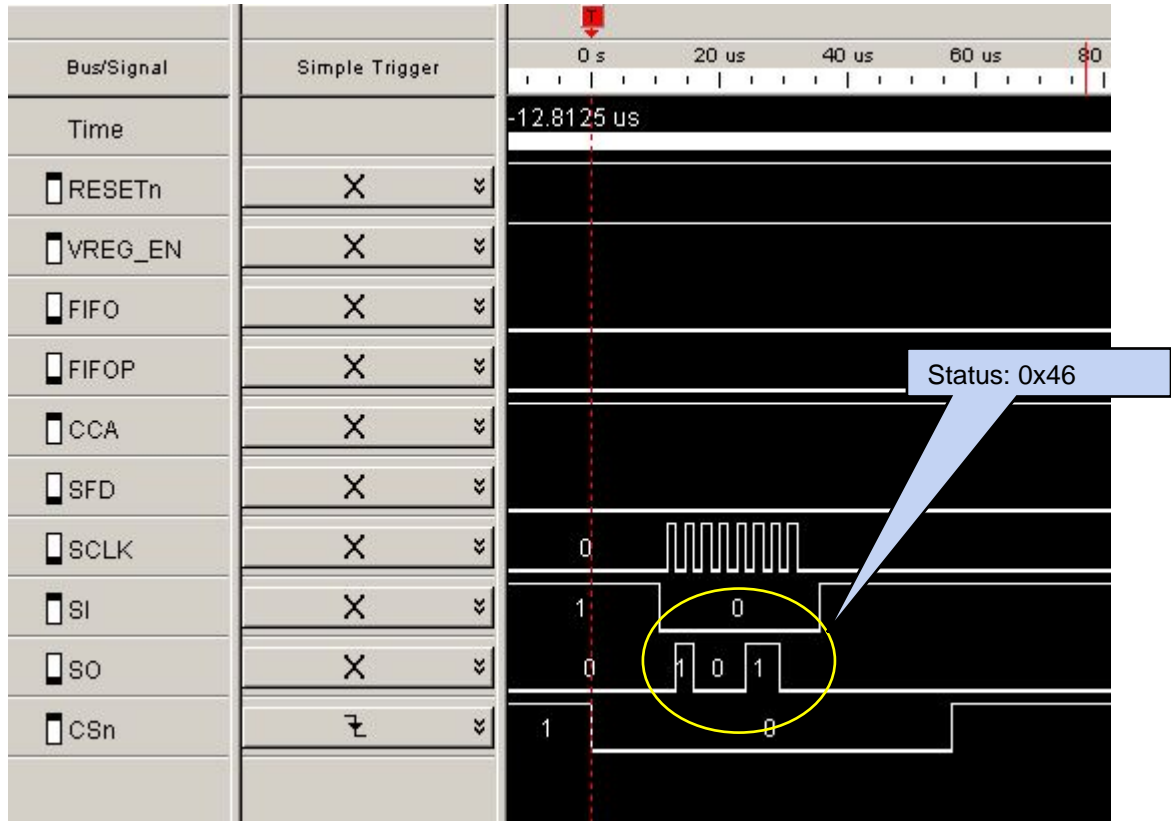


Figure 10, CC2420 Command Strobe

An example of an access cycle over the SPI-bus is shown in Figure 10. It is a *CC2420* SNOP³ operation (address 0). This is an example of a STROBE command with no data involved. The only output from the *CC2420* is the STATUS byte. The status byte is always presented on the SO pin when the command address is received on the SI pin. Data are clocked on the rising edge of SCLK.

³ SNOP: CC2420 Strobe No Operation Command.

Application Note DN033

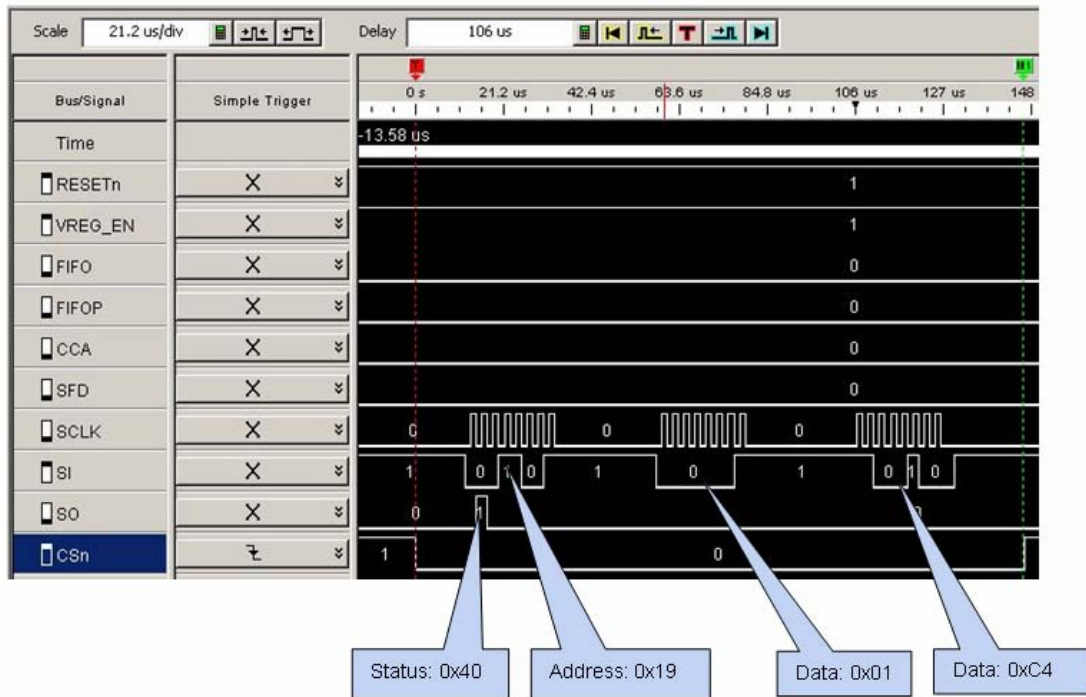


Figure 11. SPI Command Write

Figure 11 shows a command write to the *CC2420* over the SPI bus. The code effectuating this cycle is

```
#define CC2420_SECCTRL0          0x19
FASTSPI_SETREG(CC2420_SECCTRL0, 0x01C4); // Turn off "Security enable"
```

Application Note DN033

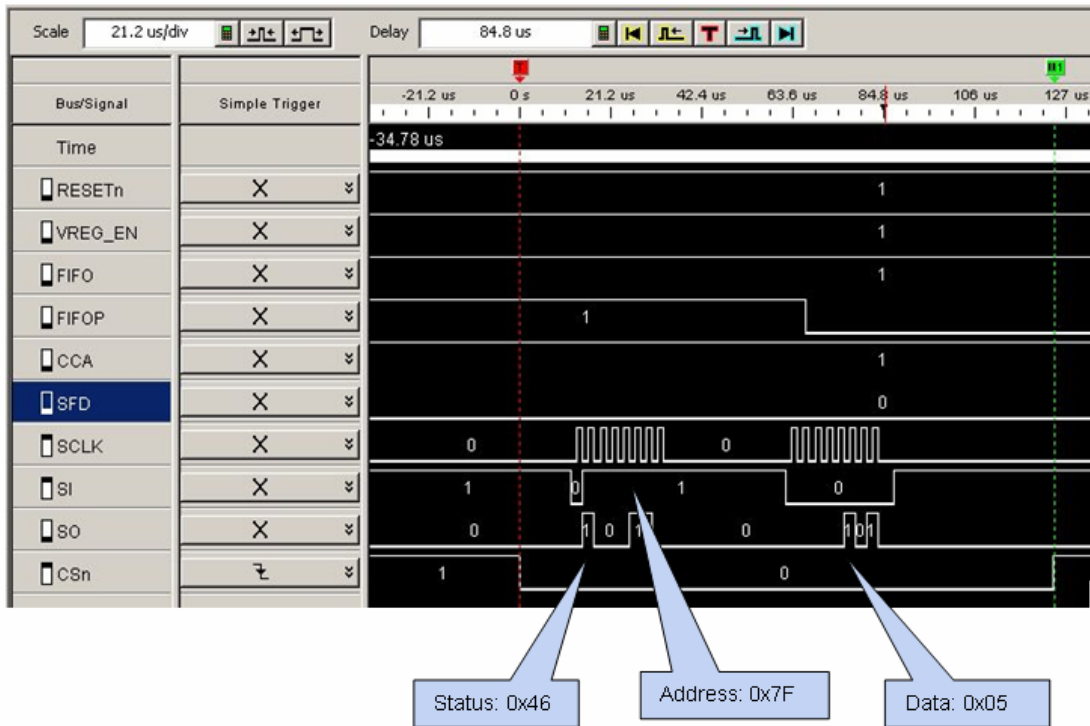


Figure 12. SPI FIFO Read Byte

Figure 12 shows a FIFO byte read from the *CC2420* over the SPI bus. The code effectuating this cycle is

```
BYTE length;  
FASTSPI_READ_FIFO_BYTE(length);
```

The message received here is an ACK, hence the short length.

5 Software

The software developed for the MSP430F1611 microcontroller is written for the IAR MSP43016 C-compiler and the GNU MSP430-GCC compiler.

Configuration of the **CC2420** is performed using general I/O pins and the MSP430's SPI interface. Basic initialisation of the **CC2420** is done by asserting the RESET pin active low for at least 1 μ s. The VREG_EN signal must also be set for the transceiver to operate. After these two conditions are met, the MCU can start operating the RF transceiver over the SPI bus. Before any further action can take place, the **CC2420** oscillator must be allowed time to stabilize. Bit 6 of the status byte holds the state of the oscillator.

The main program initialises the MCU and the RF transceiver, and sends data to the peer application periodically. The LED blinks at a rate determined by the contents of the last message received from the peer application on the CC2420DB board.

The highest priority software task is performed by the interrupt handler, which is triggered by low-high transitions on the FIFOP signal coming from the **CC2420**. Such a transition indicates that the number of bytes in the receive FIFO exceeds the programmed threshold.

The interrupt handler receives incoming messages from the peer application and adjusts the blinking period of the LED according to the contents of byte 0 in the payload.

5.1 Tools

Several tools are required to run the example in the application note:

- IAR Studio [9] with support for MSP430FET JTAG Flash Emulation Tool [4] ... or GNU MSP430-GCC and Insight/GDB Debugger (freeware alternative to IAR Studio)
- Texas Instruments SmartRF[®] Studio [7]
- Texas Instruments Packet Sniffer [6] (if needed for debugging)

Application Note DN033

5.1.1 IAR Studio

The source files are organized as projects in the manner known from IDEs like Microsoft Visual Studio. There is no need to maintain make files. The IAR Studio project file is included in the distribution of this application note.

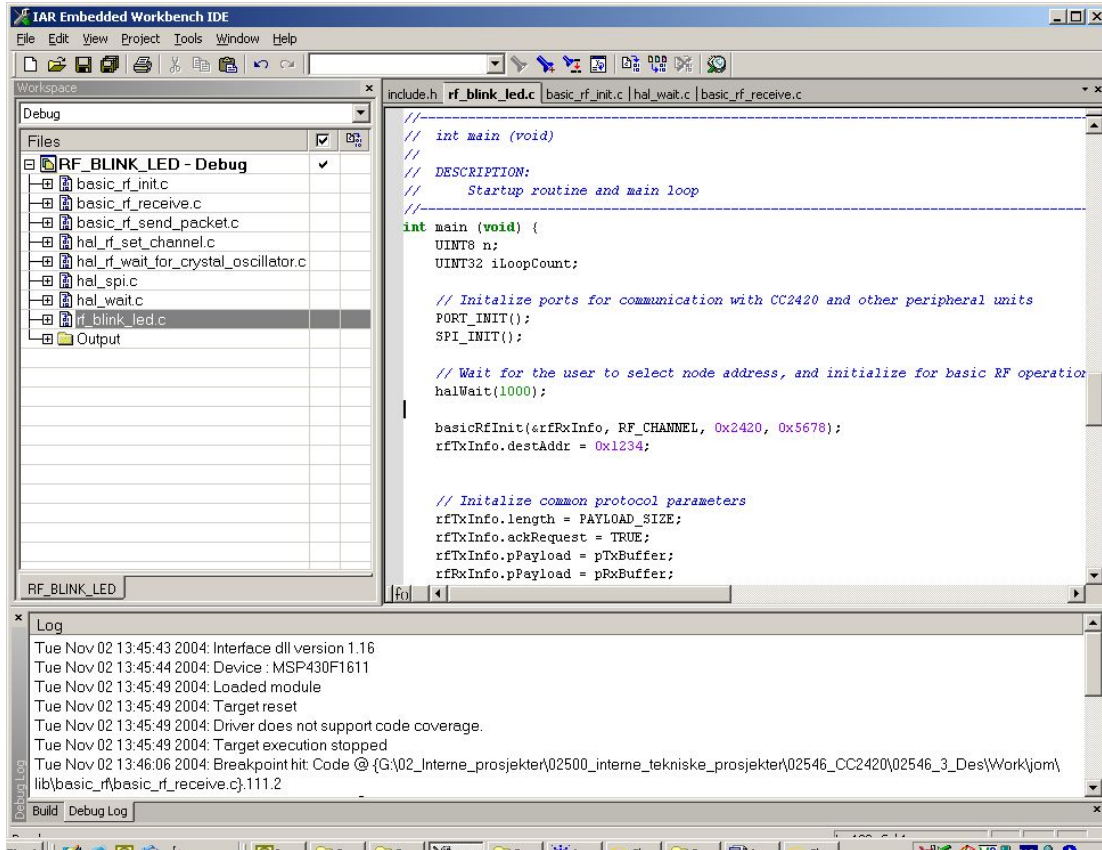


Figure 13 - IAR Studio

Please make sure that the *FET Debugger* option is selected in the Project->Tools->Options menu. Please also make sure that the MSP430F1611 is selected as target device.

The best way of getting started is to open the 'RF_BLINK_LED.eww' IAR workspace.

Application Note DN033

5.1.2 GNU Tools

Generating code using GNU tools freeware make use of a standard makefile and Cygwin underlying (UNIX) commands. Running 'make clean' and 'make all' will generate an ELF file, which can be downloaded and debugged using GDB.

1. Start msp430-gdbproxy. This makes the printer port available to GDB across TCP/IP.
2. Start GDB or Insight to download and debug the application. The file that must be downloaded is *rf_blink_led.elf*.

Insight is a graphical front end to GDB. Either tool must be configured to connect to whichever TCP/IP port the GDB-proxy uses. The hostname to connect to is *localhost* or alternatively IP-address 127.0.0.1 (assuming you run GDB/Insight on the same machine as GDB-PROXY).

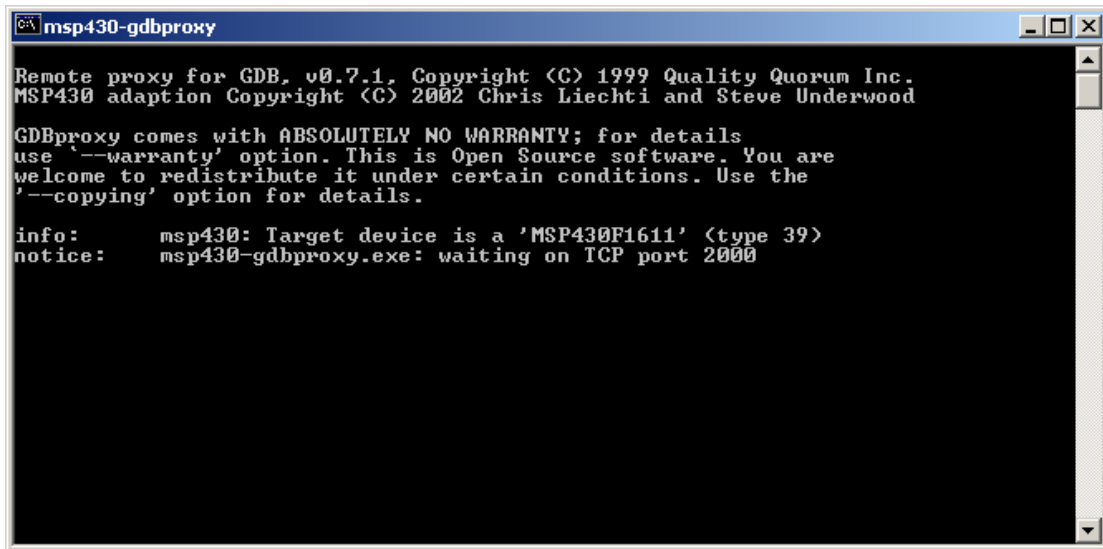


Figure 14. Screenshot of GDB-Proxy

5.2 Libraries and Examples

The *CC2420* libraries and examples include a variety of source files to ease and support the program development. Besides the standard C libraries, the source/support files are divided into 4 main groups: Hardware Definition Files, Hardware Abstraction Library, Basic RF library and finally application example.

	Application Example (source code)	Rf_blink_led
Standard C Libraries	Basic RF Library	Rf_basic_send_packet etc
	Hardware Abstraction Library	SPI etc.
	Hardware Definition Files	MSP430 and <i>CC2420</i> register definitions, etc.

Figure 15. Library Stack

5.2.1 Hardware Definition Files

The hardware definition files (include files) define the hardware registers in the MCU and the *CC2420*. They also include useful macros for the CC2420DB, and all definitions generally support the C language.

Application Note DN033

5.2.2 Hardware Abstraction Library (HAL)

To support quick and easy program development Texas Instruments provides a library of macros and functions that simplifies hardware access on the **CC2420**. These are located in the Hardware Abstraction Library (HAL) and implement a hardware abstraction interface for the user program. As a result the user program can access the microcontroller peripherals, etc. via function/macro calls, without specific knowledge about the hardware details.

5.2.3 Basic RF Library

The "Basic RF" library contains simple functions for packet transmission and reception with Texas Instruments's **CC2420**. The intention of this library is mainly to demonstrate how the **CC2420** is operated, and not to provide a complete and full functional packet protocol.

The protocol uses IEEE 802.15.4 MAC compliant data and acknowledgment packets, however it contains only a small subset of the IEEE 802.15.4 standard:

- Association, scanning and beacons are not implemented
- No defined coordinator/device roles (peer-to-peer, all nodes are equal)
- Waits for the channel to become ready, but does not check CCA twice (IEEE 802.15.4 CSMA-CA)
- Does not retransmit packets
- Can not communicate with other networks (using a different PAN identifier)
- Short IEEE 802.15.4 addresses only

5.2.4 Software Example

The software used in this application note is a modified version of an RF Blinking LED software example originally made for the CC2420DB board from Texas Instruments. Both programs available for download from Texas Instruments's WEB site. The original *RF Blinking LED software* example is described in [3].

The MSP-FET430P140 has only one LED and no pushbutton or joystick, so the software in this application note has reduced functionality.

The program demonstrates the use of the **CC2420** libraries, including the basic RF library. The "Basic RF" library can thus not be used to communicate with compliant IEEE 802.15.4 networks.

The MSP430 running this application will establish a point-to-point RF link on channel 26, using the following node address:

- PAN ID: 0x2420 (both nodes)
- Short address: 0x5678

After initializing the MCU, MSP-FET430P140 hardware and the **CC2420**, the program enters a loop where it toggles the LED each 50th cycle, and then sends a message to the CC2420DB.

The main loop cycle time is fixed at startup, but will later vary according to the value received from the CC2420DB (potentiometer value, byte 0 of payload). The LED blinks fast when messages are transmitted. The unused part of the 10-byte payload is initialized to 0xFF when the program is built with the IAR compiler and 0xEE when using the GNU compiler.

The peer program running on the CC2420DB uses the Short Address 0x1234, and works as follows:

Data packets containing a 10-byte payload will be transmitted when the potentiometer is turned, or the joystick centre button is held down. The first byte of the payload contains the potentiometer value, which is used to control the LED blinking frequency on the receiving MSP-FET430P140. The other bytes are random (never initialised).

The program on the CC2420DB uses the following LED indicators:

Application Note DN033

- Red: Transmission failed (acknowledgment not received)
- Yellow: Transmission OK (acknowledgment received)
- Orange: Remote controlled dimmer
- Green: Packet received

5.2.5 Running the example

After downloading the application to FLASH via either debug tool, the application will start after powering on the MSP-FET430P140. The LED should blink to indicate that the program has entered the main loop.

1. Power on a CC2420DB (make sure it has a RF Blinking LED example programmed in the FLASH).
2. Press down the centre joystick to start communication
3. Tweak the potentiometer to send data across to the MSP430 application. If the yellow LED blinks, the transmission was successful, otherwise the red LED will blink.
4. Adjusting the potentiometer further, you should observe a change in the blinking frequency of the yellow LED the MSP-FET430P140 board.

5.3 Source Files

The software consists of C-code and header files. An overview of the files is provided in the table below.

File Name	Short description	Group
Rf_blink_led.c	Main program source file	Application
Basic_rf_init.c	Initializations of the CC2420 for transmit and receive operations.	Basic RF lib
Basic_rf_send.c	Transmit routine for RF packets	Basic RF lib
Basic_rf_receive.c	Interrupt service routine for receiving RF packets	Basic RF lib
Hal_rf_set_channel.c	Derive frequency programming from channel number	CC2420 HAL
Hal_rf_wait_for_crystal_oscillator.c	Wait for CC2420 crystal oscillator to stabilize.	CC2420 HAL
Hal_spi.c	Initialization of SPI communication	MSP430HAL
Hal_wait.c	Delay	MSP430HAL
Basic_rf.h	Common definitions for BASIC RF library	Basic RF lib
Hal.h	SPI access macros for MSP430/CC2420	MSP430HAL
Hal_CC2420.h	CC2420 specific definitions	CC2420 HAL
Hal_MSP430FET.h	MSP430FET specific definitions	MSP430HAL
Include.h	Common include file; contains common definitions and includes files specific to compiler and MCU type	
Rf_blink.eww	Workspace file for IAR Studio	
Rf_blink.ewp	Project file for IAR Studio	
makefile	GNU makefile for msp430-gcc	
Make_all.bat	NT command script to generate the executable (GNU)	
Make_clean.bat	NT command script to remove executable and intermediate files	

Table 2. Software File Overview

Application Note DN033

All other files referenced are standard ANSI C library files and should be provided by the compiler vendor.

Extract from the GNU makefile:

```
# List C source files here. (C dependencies are automatically generated.)
SRC = rf_blink_led.c
SRC += $(EXTRALIBDIRS)/hal/msp430/hal_wait.c
SRC += $(EXTRALIBDIRS)/hal/msp430/hal_spi.c
SRC += $(EXTRALIBDIRS)/hal/hal_rf_set_channel.c
SRC += $(EXTRALIBDIRS)/hal/hal_rf_wait_for_crystal_oscillator.c
SRC += $(EXTRALIBDIRS)/basic_rf/basic_rf_init.c
SRC += $(EXTRALIBDIRS)/basic_rf/basic_rf_send_packet.c
SRC += $(EXTRALIBDIRS)/basic_rf/basic_rf_receive.c
```

5.4 Modifying the Example

Changes to the application would normally be limited to `rf_blink_led.c`, and will not require any changes to the structure of either the IAR Studio project nor the GNU make file.

5.4.1 Adding another LED

In this case the file `hal_MSP430FET.h` would have to be modified. Assuming we want to connect an extra LED at Port 1.1, the following modifications to the code would be necessary:

In `hal_MSP430FET.h`, introduce the following lines:

```
#define GLED          1 // P1.1 - Output: Green LED
```

In the macro `PORT_INIT`:

```
P1DIR |= BM(YLED) | BM(GLED) | BM(BTN_SUPPLY); \
```

In the LED section:

```
#define SET_GLED()      ( P1OUT |=  BM(GLED) )
#define CLR_GLED()     ( P1OUT &= ~BM(GLED) )
#define TOGGLE_GLED()  ( P1OUT ^=  BM(GLED) )
```

5.4.2 Adding a Header File

Both in IAR Studio and in MSP430-GCC, inclusion of header files is taken care of; the tool automatically generates dependencies. However, the new header file must be placed in a folder which is already in the include path. Finally a line would have to be added to `include.h`:

```
#include "myInclude.h"
```

Application Note DN033

5.4.3 Adding a Source File

GNU: add the following line to the makefile:

```
SRC += myfile.c
```

IAR: Click Project->AddFiles, and select the source file you want to include.

NB! All source files use the *master inclusion file (include.h)*, this MUST be the first line of code in a source file.

5.4.4 Using a Different Compiler

Compiler switches identify each compiler at the time of build. This example has been generated for the MSP430 using GNU msp430-gcc and IAR C/C++ compiler. The code also compiles for the Atmel ATmega128L using GNU avr-gcc.

There are two places in the code where compiler switches are needed:

1. In include.h, all references to compiler-dependent header files are determined by using compiler switches.
2. In basic_rf_receive.h, declaration of the Interrupt Service Routine for receive FIFO readout depends on the compiler.

Application Note DN033

Compiler	Predefined Symbol for Identification
MSP430-GCC	__MSP430__
IAR C/C++ compiler for MSP430	__ICC430__
AVR-GCC for ATmega128	__AVR_ARCH__

Table 3. Compiler switches

6 Protocol

6.1 Data Packet Description

The frame formats used are data and acknowledge frames according to the IEEE 802.15.4 standard. The Frame Control Field (FCF) is fixed, the address information field uses short addresses only and the payload data is inserted by the application.

The preamble, Start of Frame Delimiter, the FCS is generated and inserted in the package by *CC2420*. The data and acknowledges frames are shown below.

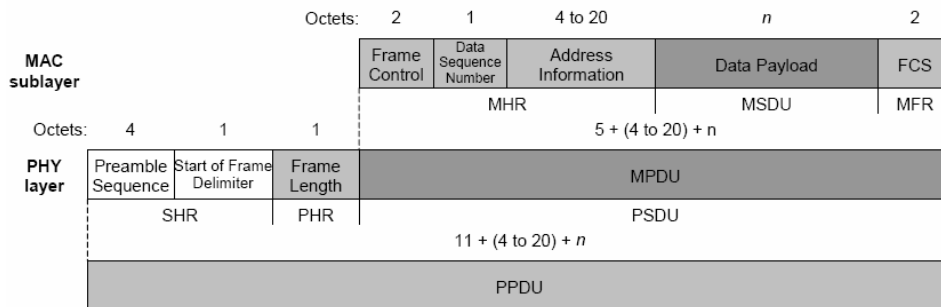


Figure 16: IEEE 802.15.4 Data Frame

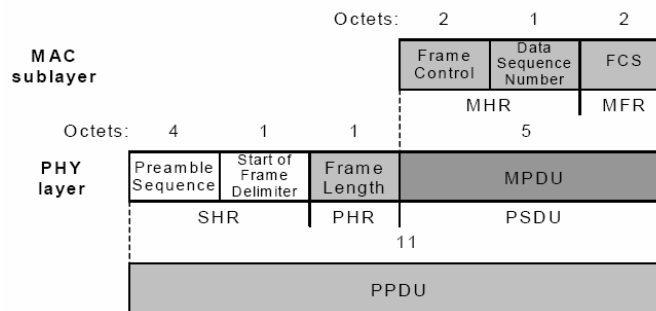


Figure 17: IEEE 802.15.4 Acknowledge Frame

Application Note DN033

6.2 Packet Format

The figures below show examples of the data flow between the MSP430/CC2420 node and the CC2420DB node. The examples are snapshots from the Texas Instruments IEEE 802.15.4 packet sniffer.

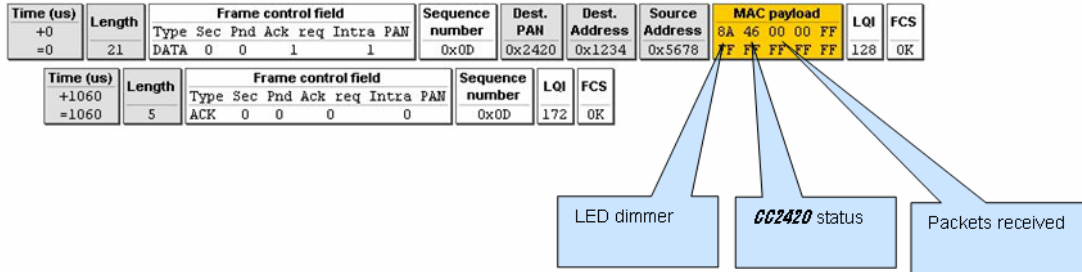


Figure 18. Data Transmitted from MSP430/CC2420 to CC2420DB

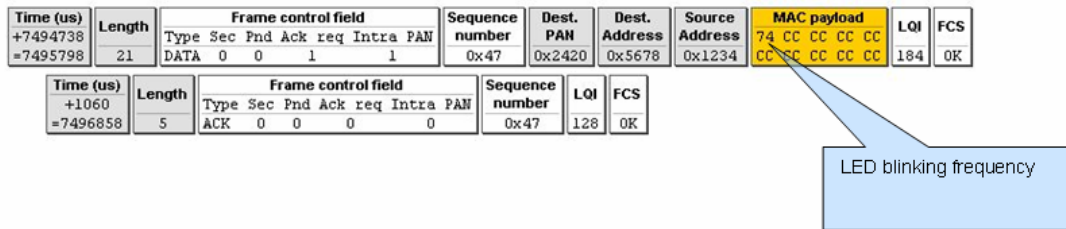


Figure 19. Data Received by MSP430/CC2420 from CC2420DB

Application Note DN033

7 References

- [1] MSP430 x15x, MSP430x16x, MSP430x161x Mixed Signal Microcontroller Data Sheet ([slas368.pdf](#))
- [2] 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver ([CC2420.pdf](#))
- [3] CC2420DBK User Manual ([swru043.pdf](#))
- [4] MSP430 Flash Emulation Tool (FET) User's Guide ([slau138](#))
- [5] CC2420DBK Examples Release ([swrc023.zip](#))
- [6] Texas Instruments Packet Sniffer ([swrc045.zip](#))
- [7] SmartRF® Studio ([swrc046.zip](#))
- [8] FPGA CC2420 uC Prototyping ([swrc025.zip](#))
- [9] IAR Embedded Workbench Kickstart Version ([slac050.zip](#))

Application Note DN033

8 General Information

8.1 Document History

Revision	Date	Description/Changes
1.0	2004-11-16	Initial release.
SWRA059	2004-11-17	Minor corrections.
SWRA059A	2008-02-05	Fixed IAR include problem. Updated headers and removed references to Chipcon. Updated Section 7 and removed the Download Section. Minor cosmetic changes

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

Products

Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
RF/IF and ZigBee® Solutions	www.ti.com/lprf

Applications

Audio	www.ti.com/audio
Automotive	www.ti.com/automotive
Broadband	www.ti.com/broadband
Digital Control	www.ti.com/digitalcontrol
Medical	www.ti.com/medical
Military	www.ti.com/military
Optical Networking	www.ti.com/opticalnetwork
Security	www.ti.com/security
Telephony	www.ti.com/telephony
Video & Imaging	www.ti.com/video
Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2008, Texas Instruments Incorporated