MSP430 Advanced Technical Conference 2006

Hands-On: Implementing an RF link with MSP430 and CC1100

Keith Quiring
MSP430 Applications Engineer
Texas Instruments

Overview

- Introduction
- Target Hardware
- Library File Organization
- Lab Activities

What is the Library?

- Register read/write functions (MSP430 to CC1100/CC2500)
  - Read register
  - Write register
  - Read burst
  - Write burst
  - Status read
  - Command strobe write
  - CC1100/2500 reset
- CC1100/CC2500 are targeted at general-purpose ISM-band apps at 315/433/868/915 MHz and 2.4 GHz respectively
- Library based on *CC1100/CC2500 Examples and Libraries* from Chipcon
- SPI functions only; no protocol functions
- Demo application project included

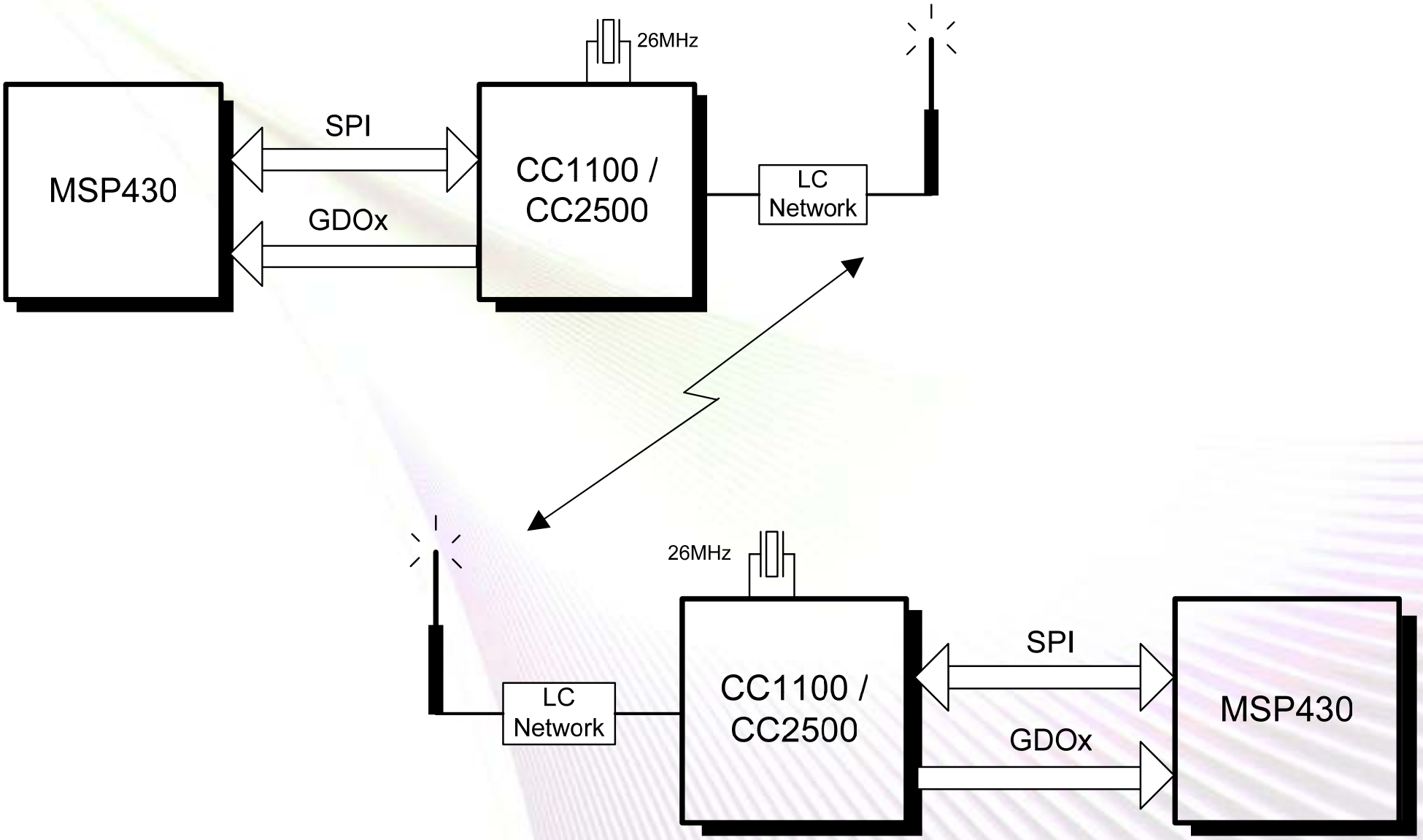
What is the Library?

- Works with any SPI-capable MSP430 interface
  - **USART0**
  - **USART1**
  - **USCI_A0**
  - **USCI_A1**
  - **USCI_B0**
  - **USCI_B1**
  - **USI**
  - **Bit-bang I/Os**
- Hardware abstraction assists porting between MSP430 devices
- Not tested for other Chipcon devices
- Tested with MCLK between 1-8MHz and SMCLK dividers of /1 and /8

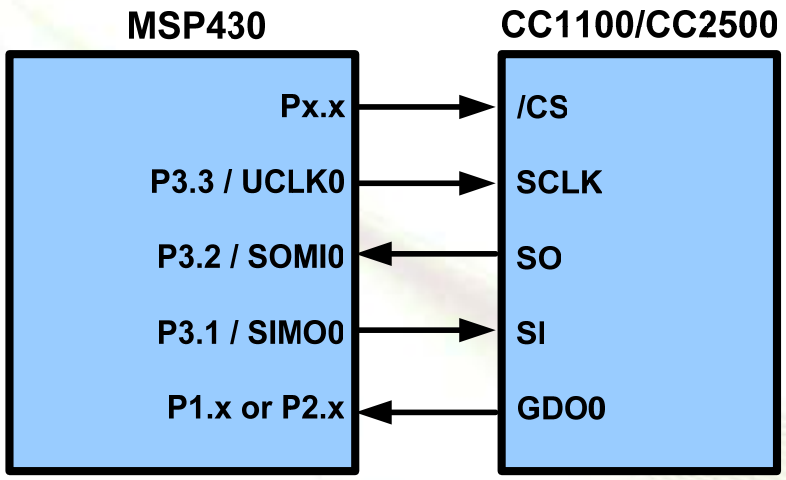
Overview

- Introduction
- Target Hardware
- Library File Organization
- Lab Activities

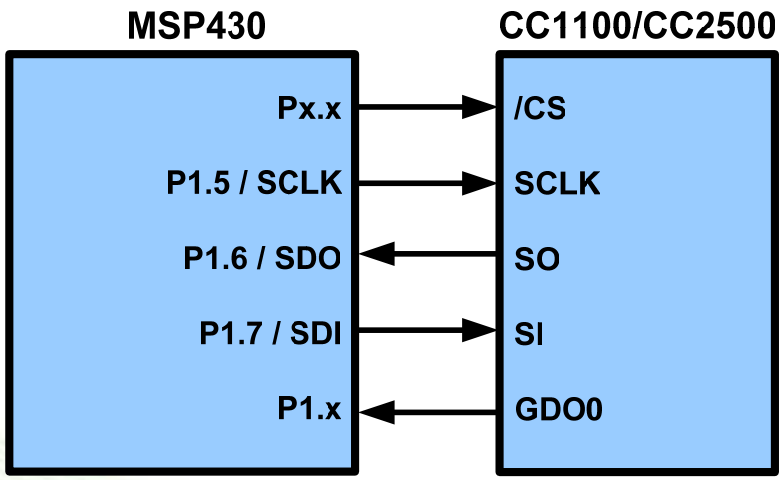
Target Hardware



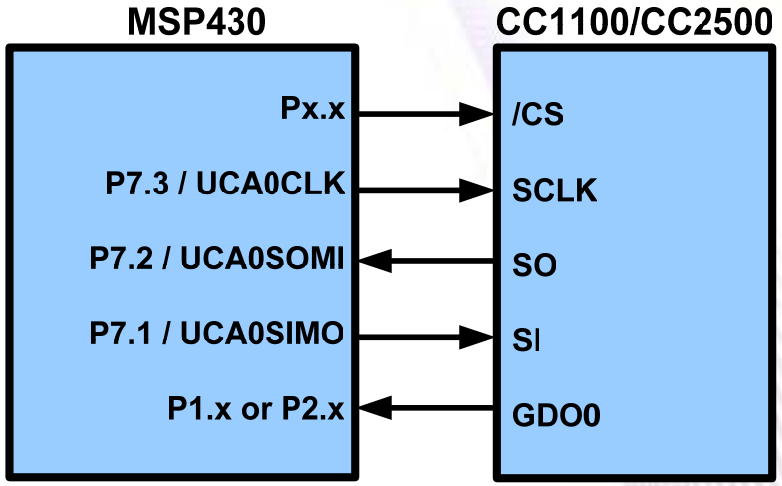
MSP430 SPI Interfaces



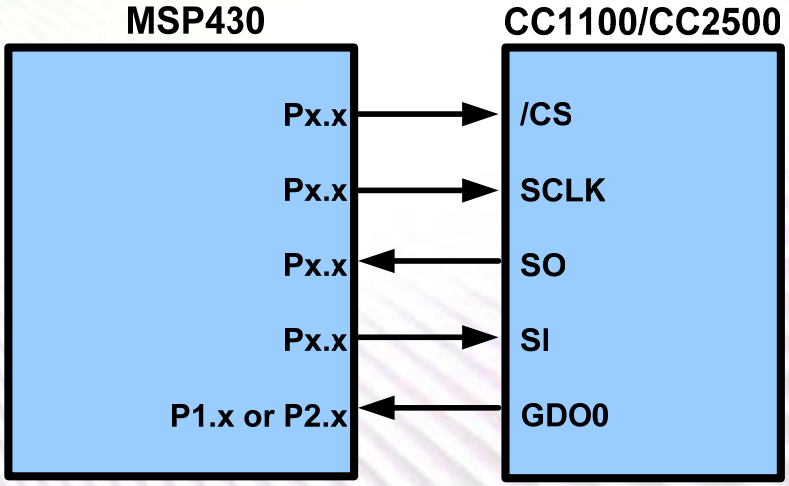
USART (USART0 shown)



USI



USCI (USCI_A0 shown)

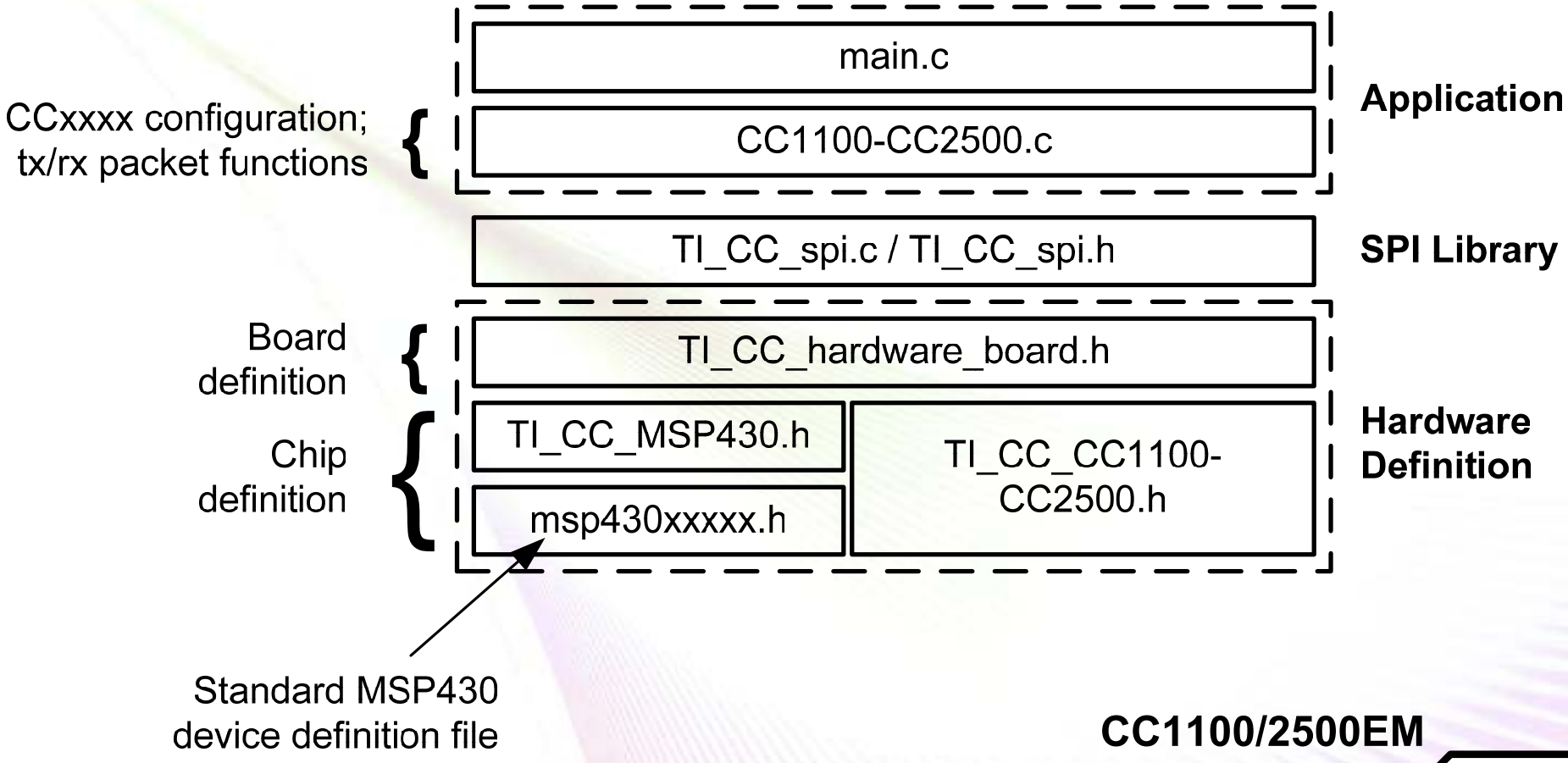


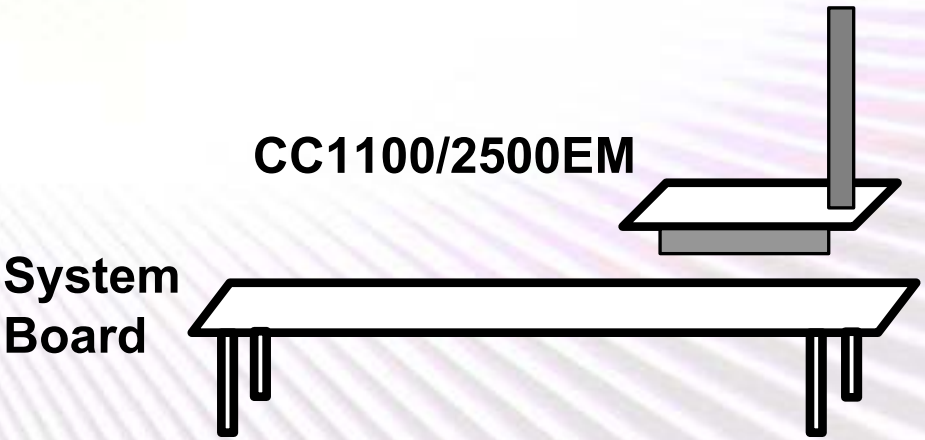
BIT-BANG

Overview

- Introduction
- Target Hardware
- Library File Organization
- Lab Activities

Demo Application Stack

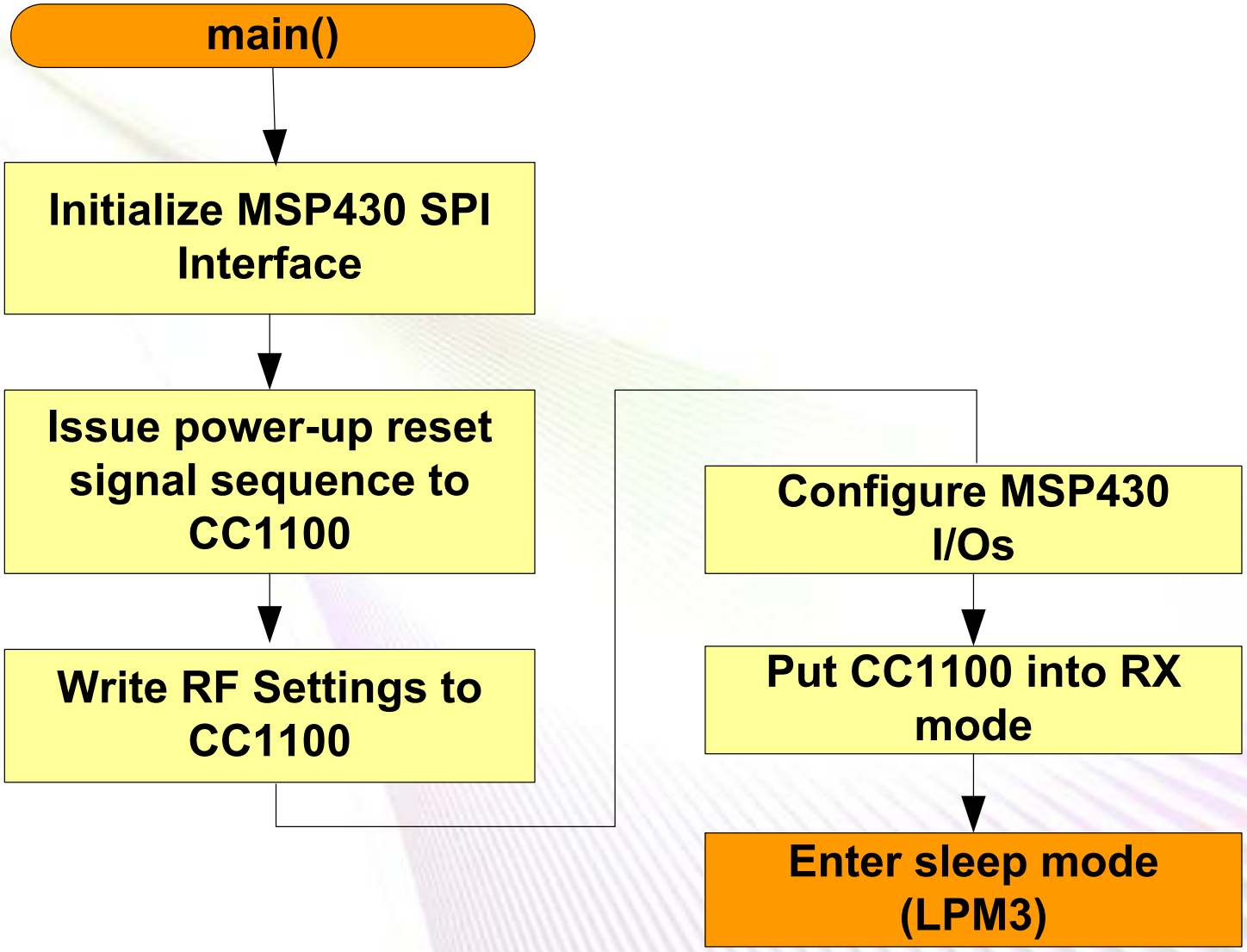




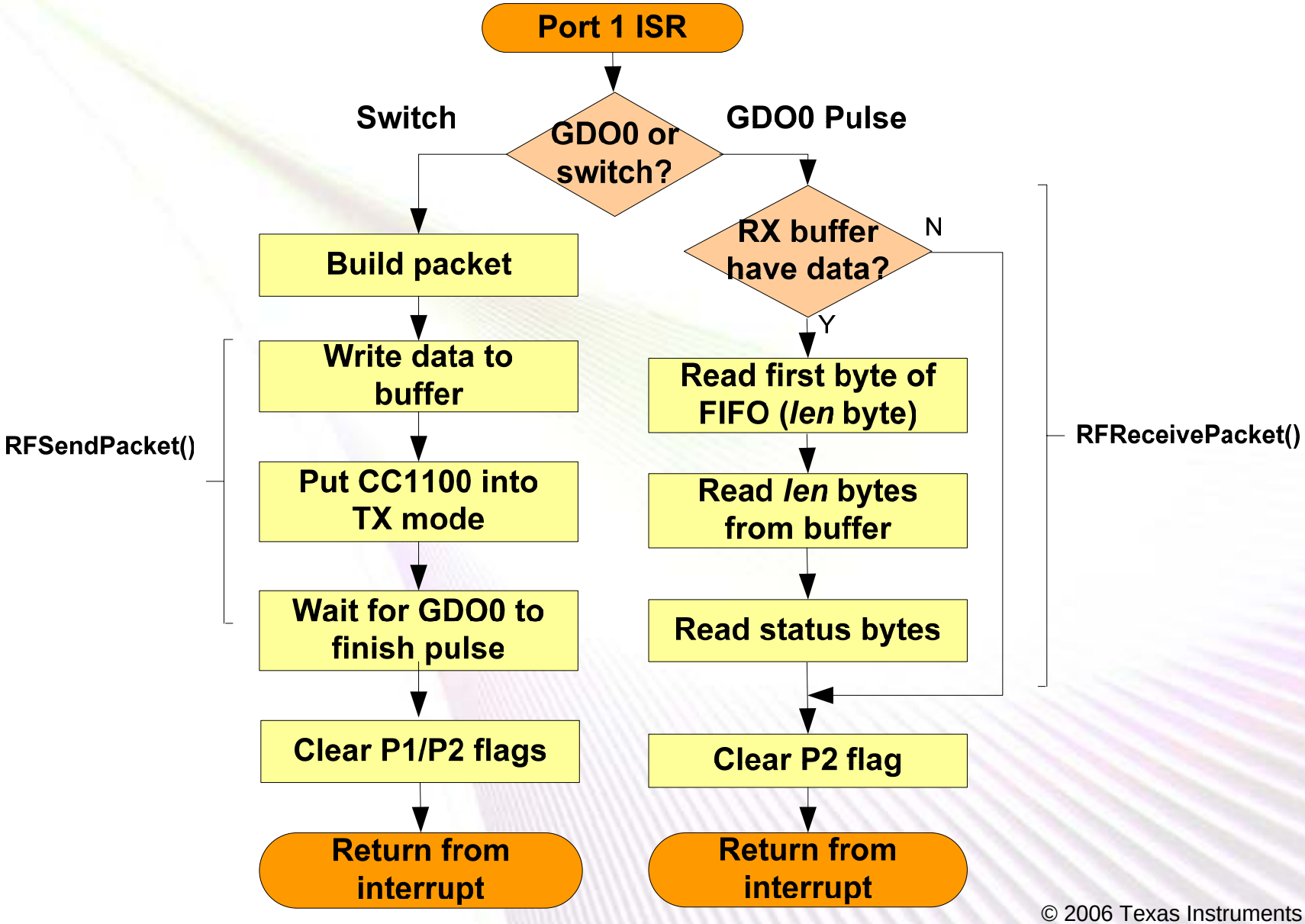
File Organization

TI_CC_CC1100-CC2500.h	Definitions specific to the CC1100/2500 devices
TI_CC_MSP430.h	Definitions specific to the MSP430 device
TI_CC_hardware_board.h	Definitions specific to the board (connections between MSP430 and CCxxxx)
TI_CC_spi.h	Function declarations for hal_spi.c
TI_CC_spi.c	Functions for accessing CC1100/CC2500 registers via SPI from MSP430

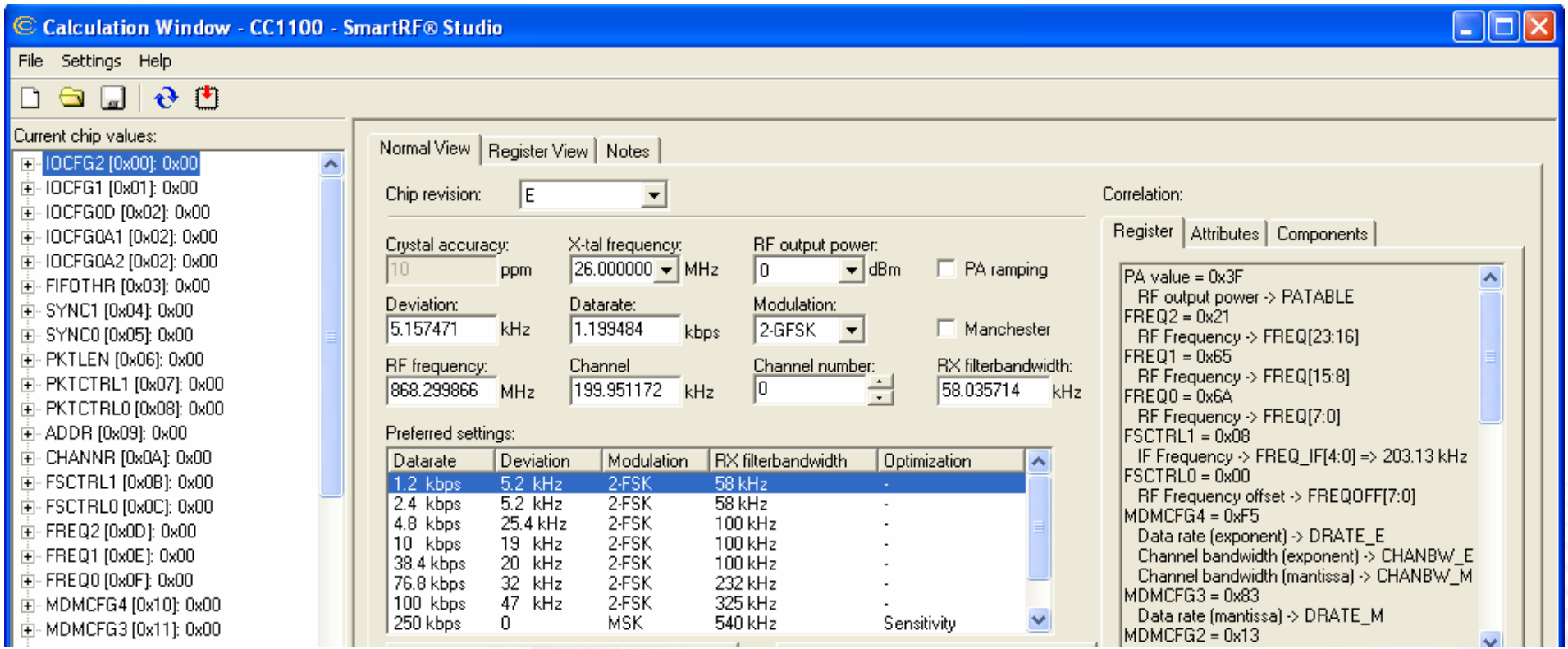
Demo Code Flowchart: main()



Demo Code Flowchart: Port_1 ISR



CC1100 Register Settings



- **Chipcon's SmartRF Studio software can assist in generating register contents**

Overview

- Introduction
- Target Hardware
- Library File Organization
- Lab Activities

Lab Activities and Resources

- **What you're going to do**
  - Phase 1: Configure library for ATC board
  - Phase 2: Write function to receive packet from administrator
  - Phase 3: Write function to send packet to front screen

- **Resources**
  - Exercise code (on your CD)
  - Solution code (on your CD)
  - MSP430FG4619 datasheet extract (at your table)
  - CC1100 datasheet extract (at your table)
  - ATC board schematic (at your table)

Phase 1: Configuration of Library

- **The library hardware definition files are currently configured for different hardware. Required changes:**
  1. **Reference the appropriate device *.h file (msp430xG46x.h)**
  2. **Configure appropriate pins for SPI interface**
  3. **Configure non-SPI connections**

TI_CC_CC1100-CC2500.h	Definitions specific to the CC1100/2500 devices
→ TI_CC_MSP430.h	Definitions specific to the MSP430 device
→ TI_CC_hardware_board.h	Definitions specific to the board (connections between MSP430 and CCxxxx)
TI_CC_spi.h	Function declarations for hal_spi.c
TI_CC_spi.c	Functions for accessing CC1100/CC2500 registers via SPI from MSP430

Files in blue need to be modified

Configuration of Library: Select Intf

TI_CC_hardware_board.h

```
#define TI_CC_GDO0_PxOUT          P2OUT
#define TI_CC_GDO0_PxIN           P2IN
#define TI_CC_GDO0_PxDIR          P2DIR
#define TI_CC_GDO0_PxIE           P2IE
#define TI_CC_GDO0_PxIES          P2IES
#define TI_CC_GDO0_PxIFG          P2IFG
#define TI_CC_GDO0_PIN            0x10
.
.
.
//-------------------------------------
// SPI port selections.  Select which
// port will be used for interface to
// CCxxxx
//-------------------------------------
#define TI_CC_RF_SER_INTF  TI_CC_SER_INTF_USART1
```

- Configure which interface used for SPI
- Defined options are listed at bottom of TI_CC_msp430.h

Configuration of Library: Define Board

TI_CC_hardware_board.h

```
#define TI_CC_SW_PxIN          P1IN
#define TI_CC_SW_PxIE          P1IE
#define TI_CC_SW_PxIES         P1IES
#define TI_CC_SW_PxIFG         P1IFG
#define TI_CC_SW1              0x01
.
.
#define TI_CC_GDO0_PxOUT       P1OUT
#define TI_CC_GDO0_PxIN        P1IN
#define TI_CC_GDO0_PxDIR       P1DIR
#define TI_CC_GDO0_PxIE        P1IE
#define TI_CC_GDO0_PxIES       P1IES
#define TI_CC_GDO0_PxIFG       P1IFG
#define TI_CC_GDO0_PIN         0x04
.
.
#define TI_CC_CSn_PxOUT        P4OUT
#define TI_CC_CSn_PxDIR        P4DIR
#define TI_CC_CSn_PIN          0x04
```

- Define connections from ATC board:
  - **Switch**
  - **GDO0**
  - **/CS**

Configuration of Library: Select MSP430

TI_CC_msp430.h

```
#include "msp430xG46x.h"                  // Adjust according to the
                                          // MSP430 device being used.
// SPI port definitions                   // Adjust for chosen intf,
#define TI_CC_SPI_USART0_PxSEL  P3SEL    // according to the pin
#define TI_CC_SPI_USART0_PxDIR  P3DIR    // assignments in the
#define TI_CC_SPI_USART0_PxIN   P3IN     // chosen MSP430 datasheet.
#define TI_CC_SPI_USART0_SIMO   0x02
#define TI_CC_SPI_USART0_SOMI   0x04
#define TI_CC_SPI_USART0_UCLK   0x08

#define TI_CC_SPI_USCIA0_PxSEL  P3SEL
#define TI_CC_SPI_USCIA0_PxDIR  P3DIR
#define TI_CC_SPI_USCIA0_PxIN   P3IN
.
.
```

- Assign device-specific standard definition file
- Check the \430\inc directory within IAR's \Program Files listing

Configuration of Library: Define MSP430

TI_CC_msp430.h

```
#include "msp430xG46x.h"                  // Adjust according to the
                                          // MSP430 device being used.
// SPI port definitions                   // Adjust for chosen intf,
#define TI_CC_SPI_USART1_PxSEL   P4SEL // according to the pin
#define TI_CC_SPI_USART1_PxDIR   P4DIR // assignments in the
#define TI_CC_SPI_USART1_PxIN    P4IN  // chosen MSP430 datasheet.
#define TI_CC_SPI_USART1_SIMO    0x08
#define TI_CC_SPI_USART1_SOMI    0x10
#define TI_CC_SPI_USART1_UCLK    0x20

#define TI_CC_SPI_USCIA0_PxSEL   P3SEL
#define TI_CC_SPI_USCIA0_PxDIR   P3DIR
#define TI_CC_SPI_USCIA0_PxIN    P3IN
#define TI_CC_SPI_USCIA0_SIMO    0x10
.
.
```

- Define which MSP430 pins used for SPI functions for interface being used, referencing the MSP430 datasheet

Phase 1: Did you Pass?

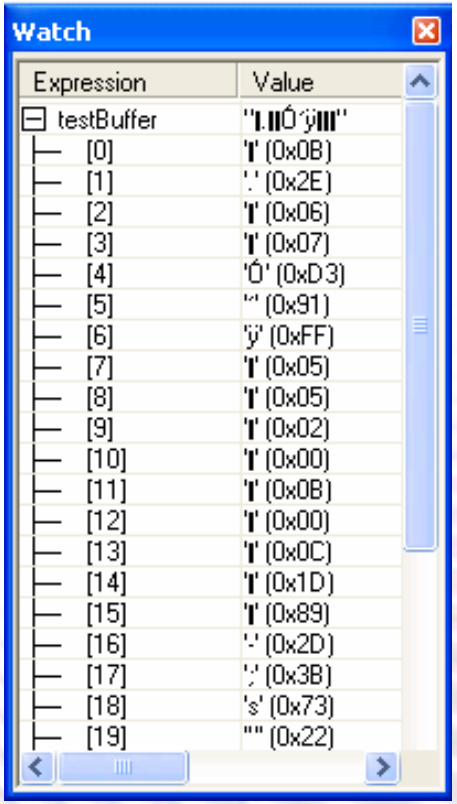
- This code reads back the data written in function writeRFSettings()
- Set breakpoint at TI_CC_SPIReadBurstReg() and step through code
- Set a watch on testBuffer[] and expand it in the “watch view”

```
// *** PHASE 1 TEST CODE ***
// To test the SPI connection, set a breakpoint to observe
// testBuffer after the burst read is executed.  The register
// settings should match the ones programmed in writeRFSettings()
  char testBuffer[0x3A];
  TI_CC_SPIReadBurstReg(TI_CCxxx0_IOCFG2, testBuffer, 0x3A);
  _NOP();
```

Phase 1: Did you Pass?

- Compare values to the ones programmed in writeRFSettings() in cc1100-cc2500.c
- Note that writeRFSettings skips some addresses

```
void writeRFSettings(void)
{
  TI_CC_SPIWriteReg(TI_CCxxx0_IOCFG2,    0x0B);
  TI_CC_SPIWriteReg(TI_CCxxx0_IOCFG0,    0x06);
  TI_CC_SPIWriteReg(TI_CCxxx0_PKTLEN,    0xFF);
  TI_CC_SPIWriteReg(TI_CCxxx0_PKTCTRL1,  0x05);
  TI_CC_SPIWriteReg(TI_CCxxx0_PKTCTRL0,  0x05);
  TI_CC_SPIWriteReg(TI_CCxxx0_ADDR,      0x02);
  TI_CC_SPIWriteReg(TI_CCxxx0_CHANNR,    0x00);
  TI_CC_SPIWriteReg(TI_CCxxx0_FSCTRL1,   0x0B);
  TI_CC_SPIWriteReg(TI_CCxxx0_FSCTRL0,   0x00);
  .
  .
```



Phase 2: Receiving a Packet

- CC1100 packets have an address field. The receiving CC1100 can filter according to address.
- There is a transmitter in this room, sending a set of packets every two seconds, each addressed to a table in the room. (See your table number.)
- Example: if there are 20 tables, the address set for each two-second sweep is {0x01, 0x02, 0x03,… 0x14}.
- Your job is to modify the code such that your CC1100 filters out all but your target address.
- Received packets are displayed via Hyperterminal.
- Tasks:
  - **Locate function writeRFSettings() and modify the address for your device (*TI_CCxxx0_ADDR*), such that it allows your packets to pass**
  - **Write function RFReceivePacket() that, when alerted by the CC1100 of a valid packet, reads the packet into the MSP430**

Phase 2: Modify Address

- Locate writeRFSettings (top of *cc1100-cc2500.c*)
- Convert your table # to hex and modify this function for your address
- Example: If you are table #11, set field to 0x0B

```
void writeRFSettings(void)
{
    TI_CC_SPIWriteReg(TI_CCxxx0_IOCFG2,   0x0B);
    TI_CC_SPIWriteReg(TI_CCxxx0_IOCFG0,   0x06);
    TI_CC_SPIWriteReg(TI_CCxxx0_PKTLEN,   0xFF);
    TI_CC_SPIWriteReg(TI_CCxxx0_PKTCTRL1, 0x05);
    TI_CC_SPIWriteReg(TI_CCxxx0_PKTCTRL0, 0x05);
    TI_CC_SPIWriteReg(TI_CCxxx0_ADDR,     0x0B);
    TI_CC_SPIWriteReg(TI_CCxxx0_CHANNR,   0x00);
    .
    .
```

Phase 2: Write RFReceivePacket()

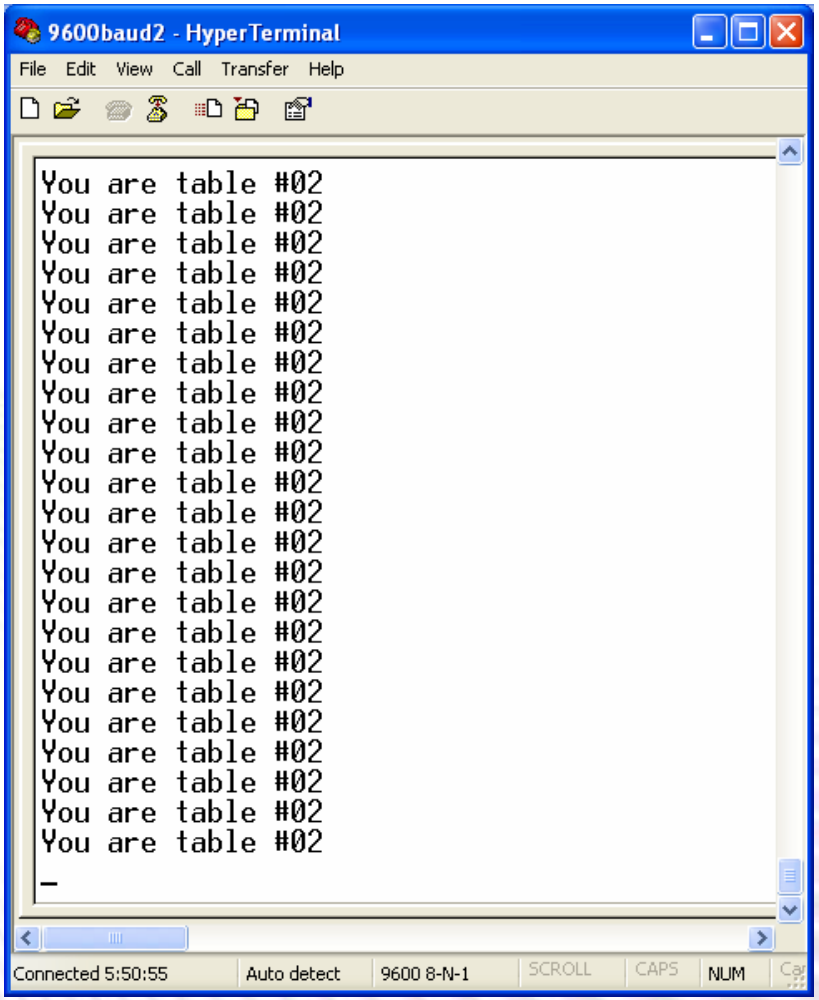
- Locate function RFReceivePacket (bottom of cc1100-cc2500.c)
- For context, locate and study PORT1_VECTOR in main(), from where it is called
- Fill in the blanks to finish RFReceivePacket
- Each blank is a SPI access, a call to the library
- Refer to file TI_CC_SPI.h for headers of the library functions
- Within RFReceivePacket(), there are directions and hints on how to solve

Phase 2: Write RFReceivePacket()

```
char RFReceivePacket(char *rxBuffer, char *length)
{
  char status[2];
  char pktLen;

  if ((TI_CC_SPIReadStatus(TI_CCxxx0_RXBYTES) & TI_CCxxx0_NUM_RXBYTES))
  {
    // read the first byte in the RX FIFO
    pktLen = TI_CC_SPI_____________(TI_CCxxx0_RXFIFO);

    // read the rest of the packet into rxBuffer
    TI_CC_SPI_____________(TI_CCxxx0_RXFIFO, rxBuffer, pktLen);
    *length = pktLen;

    // Read the two status bytes
    TI_CC_SPIReadBurstReg(TI_CCxxx0________, status, 2);

    return (char)(status[TI_CCxxx0_LQI_RX]&TI_CCxxx0_CRC_OK);
  }
  else return 0;
}
```

Phase 2: Did you Pass?

- If you run your code and a packet displays on HyperTerminal, indicating YOUR table (not someone else's), you passed!
- If you display more than one table number, you are using address 0x00 or 0xFF, which are broadcast addresses



Phase 2: Write RFReceivePacket()

- **Answers**

```
char RFReceivePacket(char *rxBuffer, char *length)
{
  char status[2];
  char pktLen;

  if ((TI_CC_SPIReadStatus(TI_CCxxx0_RXBYTES) & TI_CCxxx0_NUM_RXBYTES))
  {
    // read the first byte in the RX FIFO
    pktLen = TI_CC_SPIReadReg(TI_CCxxx0_RXFIFO);

    // read the rest of the packet into rxBuffer
    TI_CC_SPIReadBurstReg(TI_CCxxx0_RXFIFO, rxBuffer, pktLen);
    *length = pktLen;

    // Read the two status bytes
    TI_CC_SPIReadBurstReg(TI_CCxxx0_RXFIFO, status, 2);

    return (char)(status[TI_CCxxx0_LQI_RX]&TI_CCxxx0_CRC_OK);
  }
  else return 0;
}
```

Phase 3: Transmitting a Packet

- **There is a receiver node in the front of room, with Hyperterminal output directed to the screen**
- **Starter code contains a function that transmits a packet when the button is pressed. Your job is to write code that builds the packet in the appropriate format**
- **When the packet is received, it will be displayed on the screen**
- **The receiver node is operating at 868MHz instead of 915MHz, and it responds to address 0x43**
- **Tasks:**
  - Go to cc1100-cc2550.c and modify the carrier freq setting to 868MHz
  - Go to the Port1 ISR (called when switch is pressed); write code to build the packet, with a data field “We are table xx!”

Phase 3: Change the Carrier Freq

- **Modify the carrier frequency from 915MHz to 868MHz**

```
#include "include.h"
#include "TI_CC_CC1100-CC2500.h"

#define TI_CC_RF_FREQ  868  // 315, 433, 868, 915, 2400
```

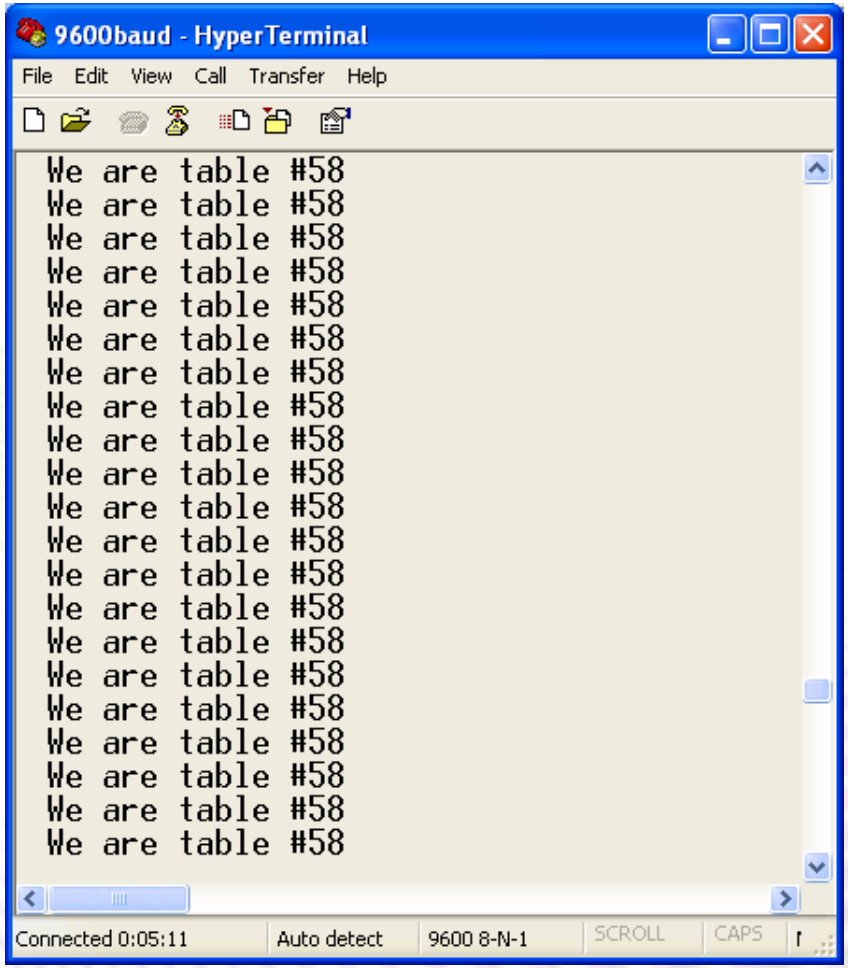
Phase 3: Build the Packet

- **Fill in the blanks with the correct values to build the packet**

```
__interrupt void port1_ISR (void)
{
  unsigned int i;
  const char kTableStr[] = "We are table #";

  if(P1IFG&TI_CC_SW1)                 // If interrupt comes from switch
  {
    txBuffer[0] = __;                 // Pkt length (not inc. len byte)
    txBuffer[1] = __;                 // Pkt address
    for(i=0;i<15;i++)                 // Copy the string
      txBuffer[2+i] = kTableStr[i];
    txBuffer[16] = ___;               // Table #, first digit
    txBuffer[17] = ___;               // Table #, second digit

    RFSendPacket(txBuffer, __);       // Send
}
```

Phase 3: Did you Pass?

- Your packet should display on the screen at the front of the room
- If packet formed incorrectly, it may be displayed in corrupted fashion or not displayed at all



Phase 3: Build the Packet

- Answers

```
__interrupt void port1_ISR (void)
{
  unsigned int i;
  const char kTableStr[] = "We are table #";

  if(P1IFG&TI_CC_SW1)                 // If interrupt comes from switch
  {
    txBuffer[0] = 0x17;               // Pkt length (not inc. len byte)
    txBuffer[1] = 0x43;               // Pkt address
    for(i=0;i<15;i++)                 // Copy the string
      txBuffer[2+i] = kTableStr[i];
    txBuffer[16] = '1';               // Table #12
    txBuffer[17] = '2';               //

    RFSendPacket(txBuffer, 18);       // Send
}
```

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 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.

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

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

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

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

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

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

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