Design Note DN508

Frequency Scanning using CC430Fx, CC110x, and CC111xFx

By Siri Johnsrud

Keywords

- *CC430Fx*
- *CC1100*
- *CC1100E*
- *CC1101*
- *CC1110Fx*
- *CC1111Fx*
- *Frequency Scanning*
- *VCO*

1 Introduction

The purpose of this design note is to show the necessary steps to successfully scan through a frequency band covering *n* numbers of channels, and find the strongest signal in the band.

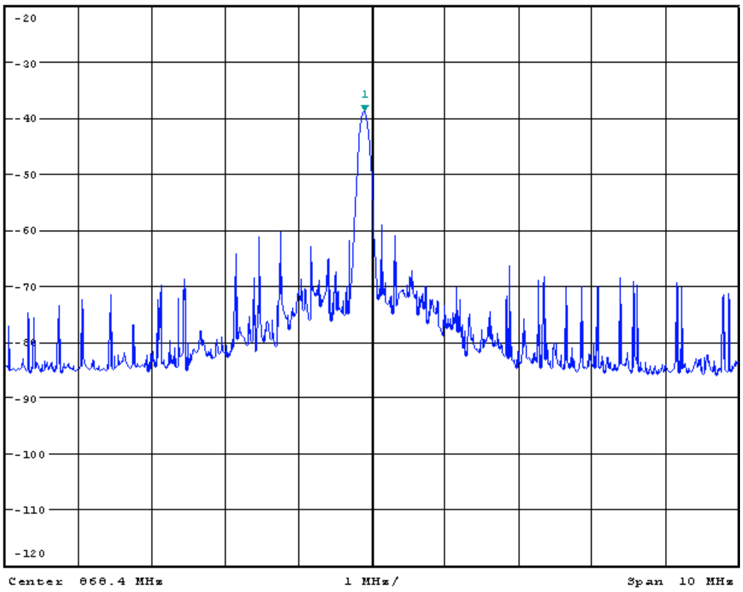

Table of Contents

2 Abbreviations

CS	Carrier Sense
kHz	Kilo Hertz
MHz	Mega Hertz
RSSI	Received Signal Strength Indicator
RF	Radio Frequency
RX	Receive
µs	Micro Seconds
VCO	Voltage Controlled Oscillator

3 Example

3.1 Assumptions

Assume one wants to scan the complete frequency band ranging from 779 - 928 MHz in steps of 200 kHz (This band is not covered by the CC1100E [3], which only supports the frequency bands ranging from 470 - 510 MHz and from 950 - 960 MHz). The easiest way to accomplish this is to select a base frequency for the frequency synthesizer (FREQ2, FREQ1, and FREQ0) and then use the CHANNR register to change frequency between channels. The channel spacing is given by Equation 1.

$$\Delta f_{CHANNEL} = \frac{f_{XOSC}}{2^{18}} \cdot (256 + CHANSPC\_M) \cdot 2^{CHANSPC\_E}$$

Equation 1. Channel Spacing

To achieve a channel spacing of ~200 kHz, MDMCFG1.CHANSPC_E must be set to 2 and MDMCFG0.CHANSPC_M to 248. The channel spacing will then be 199.951172 kHz. There will be 746 channels that need to be scanned in the entire band ((928 MHz – 779 MHz)/200 kHz + 1 = 746) and since the CHANNR register is only 8 bits wide, one will have to use three different base frequencies to cover it. SmartRF® Studio [6] is used to find the correct base frequencies and corresponding register settings. Please see Table 1 for details.

Sub Band #	Frequency Range		CHANNR	Base Frequency		
	Base (Start) Frequency [MHz]	Stop Frequency [MHz]		FREQ2	FREQ1	FREQ0
0	779.009766	829.997314	0 - 255	0x1D	0xF6	0x40
1	830.196869	881.184418	0 - 255	0x1F	0xEE	0x3F
2	881.384369	927.972992	0 - 233	0x21	0xE6	0x3F

Table 1. Base Frequencies and Channel Numbers

When using SmartRF Studio [6] to generate register settings one will see that the TEST0 register is frequency dependent. In the band covered here, TEST0 should be 0x0B for frequencies from 861 MHz and below, and 0x09 for frequencies above 861 MHz. This means that sub band #0 should always use TEST0 = 0x0B, sub band #2 should always use TEST0 = 0x09, while sub band #1 should use 0x0B for channel 0 - 154 and 0x09 for channel 155 - 255.

3.2 Pseudo Code

In short, what one should do is to go through all cannels in the band. For each channel, the radio should enter RX mode and if CS is asserted, the RSSI value should be read and stored together with the corresponding frequency. When all channels have been scanned, the frequency with the highest RSSI value is selected. In the following pseudo code (see Figure 1 and Figure 2), it is assumed that the frequency synthesizer is calibrated every time when going from IDLE to RX state (MCSM0.FS_AUTOCAL = 01_b).

```c
#define NUMBER_OF_SUB_BANDS   3

// Variables used to calculate RSSI
UINT8 rssi_dec;
INT16 rssi_dBm;
UINT8 rssi_offset[NUMBER_OF_SUB_BANDS] = {77, 77, 77};

// Freq. Band      Range                      Channel
// 0               779.009766 - 829.997314    0 - 255   All 0x0B
// 1               830.196869 - 881.184418    0 - 255   <- 154 = 0x0B, 155 -> = 0x09
// 2               881.384369 - 927.972992    0 - 233   All 0x09

INT16 rssiTable[256];
UINT16 channelNumber[256];
UINT8 carrierSenseCounter = 0; // Counter used to keep track on how many time CS has been asserted in one sub band
                               //(i.e. how many RSSI values are stored for each band)

// Stop Channel in each of the sub bands
UINT8 lastChannel[NUMBER_OF_SUB_BANDS]          = { 255,   255,   233};

// Channel number for each of the sub bands where one should change from TEST0 = 0x0B to TEST0 = 0x09
UINT16 limitTest0Reg[NUMBER_OF_SUB_BANDS]       = { 256,   155,   0  };

// Initialized to a value lower than the RSSI threshold
INT16 highRSSI[NUMBER_OF_SUB_BANDS]             = { -150, -150, -150};

// Initialized to a value greater than the highest channel number
UINT16 selectedChannel[NUMBER_OF_SUB_BANDS]     = { 300,   300,   300};

                                                // {FREQ2, FREQ1, FREQ0}
UINT8 freqSettings[NUMBER_OF_SUB_BANDS][3]      = {{0x1D,  0xF6,  0x40},
                                                   {0x1F,  0xEE,  0x3F},
                                                   {0x21,  0xE6,  0x3F}};

UINT8 activeBand;     // After the scanFreqBands() function has run, this variable will contain the sub band where
                      // the strongest signal was found
UINT16 activeChannel; // After the scanFreqBands() function has run, this variable will contain the channel number
                      // where the strongest signal was found
UINT8 calCounter = 0; // This variable is only used when running the code shown in Figure 3
```

Figure 1. Defines and Global Variables

```
void scanFreqBands(void) {
  UINT8 subBand;
  UINT8 i;
  UINT16 channel;

  // 1) Loop through all sub bands
  for (subBand = 0; subBand < NUMBER_OF_SUB_BANDS; subBand++) {

    // 1.1) Set the base freq. for the current sub band. The values for FREQ2, FREQ1, and FREQ0 can be found in
    //      freqSettings[subBand][n], where n = 0, 1, or 2

    // 1.2) Set TEST0 register = 0x0B

    // 1.3) Loop through all channels
    for (channel = 0; channel <= lastChannel[subBand]; channel++ ) {
      UINT8 pktStatus;

      // 1.3.1) Set CHANNR register = channel

      // 1.3.2) Change TEST0 register settings to 0x09 if freq is above 861 MHz
      if (channel == limitTest0Reg[subBand]) {

        // 1.3.2.1) Set TEST0 register = 0x09
      }

      // 1.3.3) Enter RX mode by issuing an SRX strobe command

      // 1.3.4) Wait for radio to enter RX state (can be done by polling the MARCSTATE register)

      // 1.3.5) Wait for RSSI to be valid (See DN505 [7] on how long to wait)

      // 1.3.6) Read the PKTSTATUS register while the radio is in RX state (store it in pktStatus)

      // 1.3.7) Enter IDLE state by issuing an SIDLE strobe command

      // 1.3.8) Check if CS is asserted (use the value obtained in 1.3.6)
      if (pktStatus & 0x40) {  // CS is asserted

        // 1.3.8.1) Read RSSI value and store it in rssi_dec

        // 1.3.8.2) Calculate RSSI in dBm (rssi_dBm)(offset value found in rssi_offset[subBand])

        // 1.3.8.3) Store the RSSI value and the corresponding channel number
        rssiTable[carrierSenseCounter] = rssi_dBm;
        channelNumber[carrierSenseCounter] = channel;
        carrierSenseCounter++;
      }
    } // End Channel Loop

    // 1.4) Before moving on to the next sub band, scan through the rssiTable to find the highest RSSI value. Store
    //      the RSSI value in highRSSI[subBand] and the corresponding channel number in selectedChannel[subBand]
    for (i = 0; i < carrierSenseCounter; i++) {
      if (rssiTable[i] > highRSSI[subBand]) {
        highRSSI[subBand] =  rssiTable[i];
        selectedChannel[subBand] = channelNumber[i];
      }
    }

    // 1.5) Reset carrierSenseCounter
    carrierSenseCounter = 0;
  } // End Band Loop

  // 2) When all sub bands have been scanned, find which sub band has the highest RSSI (Scan the highRSSI[subBand]
  //    table). Store the subBand (0, 1, or 2) and the corresponding channel in the global variables activeBand and
  //    activeChannel respectively
  {
    INT16 tempRssi = -150;
    for (subBand = 0; subBand < NUMBER_OF_SUB_BANDS; subBand++) {
      if (highRSSI[subBand] >= tempRssi) {
        tempRssi = highRSSI[subBand];
        activeChannel = selectedChannel[subBand];
        activeBand = subBand;
      }
    }
  }
}
```

Figure 2. Pseudo Code for scanFreqBand() when `MCSM0.FS_AUTOCAL = 01`$_b$

Going from IDLE to RX with calibration (`MCSM0.FS_AUTOCAL = 01`$_b$) takes ~800 µs while going from IDLE to RX without calibration (`MCSM0.FS_AUTOCAL = 00`$_b$) takes ~75 µs. The scanning time can therefore be reduced significantly by reducing the numbers of frequencies to calibrate. If calibrating for frequency x MHz, it is possible to enter RX on a frequency 1 MHz below x and 1 MHz above x without doing a new calibration (11 frequencies are covered by only one calibration when the channel spacing is ~200 kHz). Using this approach will complicate the code a bit, so the next code example (see Figure 3) simply shows how one can calibrate every 5th channel instead (calibrate for channel x, x + 1 MHz, x + 2 MHz etc.).

```
void scanFreqBands(void) {
  UINT8 subBand;
  UINT8 i;
  UINT16 channel;

  // 1) Loop through all sub bands
  for (subBand = 0; subBand < NUMBER_OF_SUB_BANDS; subBand++) {

    // 1.1) Set the base freq. for the current sub band. The values for FREQ2, FREQ1, and FREQ0 can be found in
    //      freqSettings[subBand][n], where n = 0, 1, or 2

    // 1.2) Set TEST0 register = 0x0B

    // 1.3) Reset Calibration Counter (calibration performed when counter is 0)
    calCounter = 0;

    // 1.4) Loop through all channels
    for (channel = 0; channel <= lastChannel[subBand]; channel++ ) {
      UINT8 pktStatus;

      // 1.4.1) Set CHANNR register = channel

      // 1.4.2) Change TEST0 register settings to 0x09 if freq is above 861 MHz. When TEST0 is changed to 0x09, it
      //        is important that FSCAL2 is set to 0x2A and that a new calibration is performed
      if (channel == limitTest0Reg[subBand]) {

        // 1.4.2.1) Set TEST0 register = 0x09

        // 1.4.2.2) Set FSCAL2 register = 0x2A

        // 1.4.2.3) Calibration is needed when TEST0 is changed
        calCounter = 0;
      }

      // 1.4.3) Calibrate for every 5th ch. + at start of every sub band and every time the TEST0 reg. is changed
      if (calCounter++ == 0) {

        // 1.4.3.1) Perform a manual calibration by issuing an SCAL strobe command
      }

      // 1.4.4)) Reset Calibration Counter (if calCounter = 5, we are 1 MHz away from the frequency where a
      //         calibration was performed)
      if (calCounter == 5) {

        // 1.4.4.1) Calibration is performed if calCounter = 0
        calCounter = 0;
      }

      // 1.4.5) Enter RX mode by issuing an SRX strobe command

      // 1.4.6) Wait for radio to enter RX state (can be done by polling the MARCSTATE register)

      // 1.4.7) Wait for RSSI to be valid (See DN505 [7] on how long to wait)

      // 1.4.8) Read the PKTSTATUS register while the radio is in RX state (store it in pktStatus)

      // 1.4.9) Enter IDLE state by issuing an SIDLE strobe command

      // 1.4.10) Check if CS is asserted (use the value obtained in 1.4.8)
      if (pktStatus & 0x40) {  // CS is asserted

        // 1.4.10.1) Read RSSI value and store it in rssi_dec

        // 1.4.10.2) Calculate RSSI in dBm (rssi_dBm)(offset value found in rssi_offset[subBand])

        // 1.4.10.3) Store the RSSI value and the corresponding channel number
        rssiTable[carrierSenseCounter] = rssi_dBm;
        channelNumber[carrierSenseCounter] = channel;
        carrierSenseCounter++;
      }
    } // End Channel Loop

    // 1.5) Before moving on to the next sub band, scan through the rssiTable to find the highest RSSI value. Store
    //      the RSSI value in highRSSI[subBand] and the corresponding channel number in selectedChannel[subBand]
    for (i = 0; i < carrierSenseCounter; i++) {
      if (rssiTable[i] > highRSSI[subBand]) {
        highRSSI[subBand] =  rssiTable[i];
        selectedChannel[subBand] = channelNumber[i];
      }
    }

    // 1.6) Reset carrierSenseCounter
    carrierSenseCounter = 0;
  } // End Band Loop

  // 2) When all sub bands have been scanned, find which sub band has the highest RSSI (Scan the highRSSI[subBand]
  //    table). Store the subBand (0, 1, or 2) and the corresponding channel in the global variables activeBand and
  //    activeChannel respectively
  {
    INT16 tempRssi = -150;
    for (subBand = 0; subBand < NUMBER_OF_SUB_BANDS; subBand++) {
      if (highRSSI[subBand] >= tempRssi) {
        tempRssi = highRSSI[subBand];
        activeChannel = selectedChannel[subBand];
        activeBand = subBand;
      }
    }
  }
```

Figure 3. Pseudo Code for scanFreqBand() when `MCSM0.FS_AUTOCAL = 00`$_b$

When `TEST0 = 0x0B` the VCO selection calibration stage is enabled (`TEST0.VCO_SEL_CAL_EN = 1`). In this case, the value of `FSCAL2.VCO_CORE_H_EN` does not matter and after a calibration `FSCAL2.VCO_CORE_H_EN` can be 1 or 0, depending on which VCO was selected. When `TEST0 = 0x09` the VCO selection calibration stage is disabled (`TEST0.VCO_SEL_CAL_EN = 0`) and the VCO is forced high or low depending on the `FSCAL2.VCO_CORE_H_EN` setting. Since `FSCAL2.VCO_CORE_H_EN` might have been 0 after the last calibration when `TEST0` was 0x09, it is important to change `FSCAL2.VCO_CORE_H_EN` to 1 when `TEST0` is changed to 0x09 since SmartRF Studio [6] recommends high VCO for all frequencies above 861 MHz.

4 References

[1] CC430 User's Guide (slau259.pdf)

[2] CC1100 Single-Chip Low Cost Low Power RF-Transceiver, Data sheet (cc1100.pdf)

[3] CC1100E Low-Power Sub-GHz RF Transceiver (470-510 MHz & 950-960 MHz) (CC1100E.pdf)

[4] CC1101 Single-Chip Low Cost Low Power RF-Transceiver, Data sheet (cc1101.pdf)

[5] CC1110Fx/CC1111Fx Low-Power Sub-1 GHz RF System-on-Chip (SoC) with MCU, Memory, Transceiver, and USB Controller (cc1110f32.pdf)

[6] SmartRF® Studio (swrc046.zip)

[7] DN505 RSSI Interpretation and Timing (swra114.pdf)

5 General Information

5.1 Document History

Revision	Date	Description/Changes
SWRA315	2010.01.22	Initial release.

IMPORTANT NOTICE