Application Report
SWRA428A – October 2013

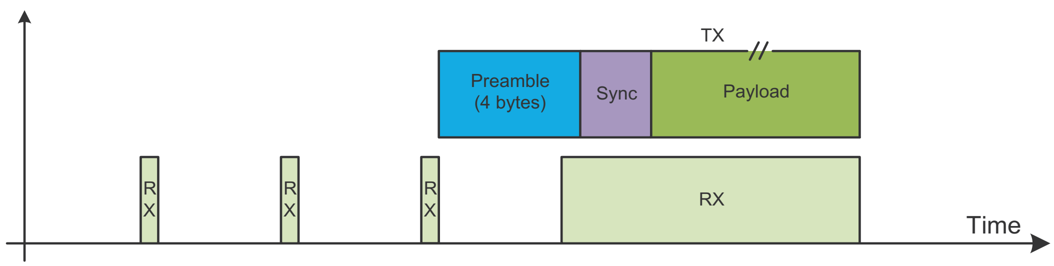
CC112x/CC120x RX Sniff Mode

Siri Johnsrud

ABSTRACT

This application report shows how the RX Sniff Mode can be configured for different TX configurations and discusses how average power consumption can be estimated for a receiver implementing this mode. You are expected to have basic knowledge on how the enhanced Wake on Radio (eWOR) and the RX Sniff Mode are implemented in TI's Performance Line Sub-1 GHz RF family of devices [1], [2].

Project collateral associated with this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/swra428.



Contents

1 Introduction 2
2 Notation 3
3 Configuring the Radio for RX Sniff Mode 3
4 Code Examples 11
5 Conclusion 26
6 References 26
Appendix A Revision History 27

List of Figures

1 Event 0 and Event 1 4
2 MCU Controlled RX Timeout 7
3 Total RX Consumption in a Noise Free Environment 8
4 Total RX Current Consumption in Presence of Noise 9
5 Typical Settings (CC1120, 1.2 kbps, 50 kHz RX filter BW) 11
6 Code Configurations 13
7 CS Response Time (CC1120, 1.2 kbps, 50 kHz RX filter BW) 18
8 Current Profile (CC1120, 1.2 kbps, 50 kHz RX filter BW) 18
9 Average Current Consumption (CC1120, 1.2 kbps, 50 kHz RX Filter BW, RX config. 1) 19
10 CC112x_RX_Sniff_Mode.xlsx Dashboard (RX config. 1) 19
11 CC112x_RX_Sniff_Mode.xlsx Dashboard (RX config. 2) 20
12 4 Bytes Sync Word vs. 11 Bits Sync Word 20
13 CC112x_RX_Sniff_Mode.xlsx Dashboard (RX config. 3) 20
14 Typical Settings (CC1200, 38.4 kbps, ETSI standard) 21

SmartRF is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

15 CC120x_RX_Sniff_Mode.xlsx Dashboard (3 bytes preamble) 23
16 CC120x_RX_Sniff_Mode.xlsx Dashboard (24 bytes preamble).................... 23
17 Average Current Consumption (CC1200, 38.4 kbps, ETSI standard, 24 bytes preamble) 25

List of Tables

1 Abbreviations 2
2 WOR_CFG1 - eWOR Configuration Reg. 1 3
3 Register Fields Used by RX Sniff Mode 3
4 Resolution and Max t_{Event0} for Different Values of WOR_RES 8
5 Register Fields to be Changed When Running RX Sniff Mode (RX config. 1) 15
6 SPNU428A Revisions 27

1 Introduction

The RX Sniff Mode is a novel feature enabled by the TI Performance Line WaveMatch technology. The receiver only needs 4 bits of preamble for settling, as opposed to legacy receivers that often need 3 - 4 bytes. In a typical RF protocol where several preamble bytes are transmitted, TI Performance Line can autonomously duty cycle the receiver while waiting for a packet. Therefore, the RX Sniff Mode can be used fully transparent to you, while offering greatly reduced average current without sacrificing RF performance. The RX Sniff Mode is enabled by using the eWOR timer together with the RX termination based on carrier sense (CS) or preamble quality threshold (PQT).

Table 1. Abbreviations

AGC	Automatic Gain Control
CRC	Cyclic Redundancy Check
CS	Carrier Sense
eWOR	Enhanced Wake on Radio
FIFO	First-In-First-Out
GPIO	General-purpose input/output
OSC	Oscillator
PQT	Preamble Quality Threshold
RC	Resistor-Capacitor
RSSI	Received Signal Strength Indicator
RX	Receive Mode

2 Notation

Throughout this document, mapped register values are used. m(`REGISTER_NAME.REGISTER_FIELD`) equals the value in the description field that matches the bit pattern. If a register is not part of a m()-construct, it means that the bit pattern value is used. After a reset, m(`WOR_CFG1.EVENT1`) = 4 while `WOR_CFG1.EVENT1 = 0`.

Table 2. WOR_CFG1 - eWOR Configuration Reg. 1

Bit No	Name	Reset	R/W	Description	
2:0	EVENT1	0x00	R/W	Event 1 timeout	
				EVENT1	WOR_EVENT1
				000	4
				001	6
				010	8
				011	12
				100	16
				101	24
				110	32
				111	48

3 Configuring the Radio for RX Sniff Mode

This section explains all the different parameters and register fields involved when configuring the radio for the RX Sniff Mode. Table 3 shows the different register fields discussed and in which section they are covered.

Table 3. Register Fields Used by RX Sniff Mode

Register Name	Register Field	Section
WOR_CFG1	WOR_RES	Section 3.2 and Section 3.8
	WOR_MODE	Section 3.10 and Section 3.11
	EVENT1	Section 3.3
WOR_CFG0	DIV_256HZ_EN	Section 3.12
	EVENT2_CFG	Section 3.1 and Section 3.4
	RC_MODE	Section 3.5
	RC_PD	Section 3.5
WOR_EVENT0_MSB	EVENT0_15_8	Section 3.2 and Section 3.7
WOR_EVENT0_LSB	EVENT0_7_0	Section 3.2 and Section 3.7
SETTLING_CFG	FS_AUTOCAL	Section 3.6
RFEND_CFG1	RXOFF_MODE	Section 3.10 and Section 3.11
	RX_TIME	Section 3.7
	RX_TIME_QUAL	Section 3.7
RFEND_CFG0	TERM_ON_BAD_PACKET	Section 3.10, Section 3.10.2, and Section 3.11
	ANT_DIV_RX_TERM_CFG	Section 3.9, Section 3.9.1, and Section 3.9.2
AGC_CS_THR	AGC_CS_THRESHOLD	Section 3.9.1
AGC_CFG1	AGC_WIN_SIZE	Section 3.9.1
	AGC_SETTLE_WAIT	Section 3.9.1
AGC_CFG0	RSSI_VALID_COUNT	Section 3.9.1

Table 3. Register Fields Used by RX Sniff Mode (continued)

Register Name	Register Field	Section
PREAMBLE_CFG0	PQT_EN	Section 3.9.2
	PQT	Section 3.9.2
FIFO_CFG	CRC_AUTOFLUSH	Section 3.10.1
PKT_CFG1	CRC_CFG	Section 3.10.1
PKT_CFG0	LENGTH_CONFIG	Section 3.10.2
PKT_LEN	PACKET_LENGTH	Section 3.10.2
IOCFGx	GPIOx_CFG	Section 3.1 and Section 3.10
MDMCFG1	CARRIER_SENSE_GATE	Section 3.2
SYNC_CFGx [(1)]	PQT_GATING_EN	Section 3.2

(1) PQT_GATING_EN is in the SYNC_CFG1 register for CC112x and SYNC_CFG0 register for CC120x.

3.1 Event 0, Event 1 and Event 2

The eWOR timer has three events: Event 0, Event 1, and Event 2. The relationship between Event 0 and Event 1 is shown in Figure 1. Event 2 is not used in the RX Sniff Mode (see Section 3.4) and is, therefore, not shown in the figure.

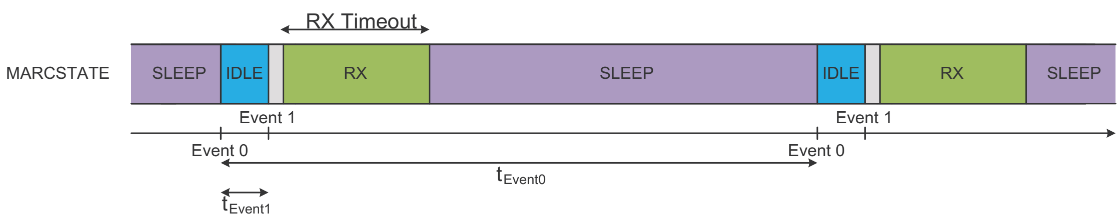


Figure 1. Event 0 and Event 1

All three events can be monitored on the general-purpose input/output (GPIO) pins by setting IOCFGx.GPIOx_CFG = WOR_EVENT0/1/2 (54/55/56). If IOCFGx.GPIOx_CFG = WOR_EVENT2 (56), WOR_CFG0.EVENT2_CFG must be ≠ 00_b. Event 2 is not shown in Figure 1 as it will not be used in most RX Sniff Mode applications. For more details, see Section 3.4.

3.2 Event 0 and t_{Event0}

The RX Sniff Mode Event 0 is the event used when the crystal oscillator and the digital regulator are turned on (when the radio is in SLEEP state). The time between two Event 0's are called t_{Event0} and there are several factors to take into account when determining this time. It is recommended that both MDMCFG1.CARRIER_SENSE_GATE and SYNC_CFGx.PQT_GATING_EN = 0 when using the RX Sniff Mode to simplify how t_{event0} should be determined.

- Termination based on CS (see Section 3.9.1)

  Before the radio can detect a sync word, a minimum of 4-bit preamble is needed for AGC settling (including frequency offset compensation). This means that the maximum time between two Event 0's can be calculated as shown in Equation 1 when RX is terminated based on CS.

$$t_{Event0}[\text{Desired}] = \frac{\text{\# of Preamble Bits - 4}}{\text{Data Rate [bps]}}\,[\text{s}] \qquad (1)$$

- Termination based on PQT (see Section 3.9.2)

  If termination based on PQT is used, the radio needs a maximum of 10 bits of preamble to be sure that a preamble is detected (AGC settling and frequency offset compensation is included). The preamble quality estimator uses an 8-bit wide correlation filter to detect a preamble and 2 extra bits might be necessary to align the transmitter and receiver. In addition, the radio needs some time, T0, before starting to look for the preamble. For more information on how to calculate T0, see the device-specific user's guides, [1] and [2]. This means that the max time between two Event 0's can be calculated as shown in Equation 2 when RX is terminated based on PQT. Equation 2 is not valid when using OOK, feedback to PLL, or when `TOC_LIMIT ≠ 0`.

$$t_{Event0}[\text{Desired}] = \frac{\text{\# of Preamble Bits - 10}}{\text{Data Rate [bps]}} - T0\,[\text{s}] \tag{2}$$

t_{Event0} is given by the `WOR_CFG1.WOR_RES`, `WOR_EVENT0_MSB`, and `WOR_EVENT0_LSB` registers together with the frequency of the low-power RC oscillator as shown in Equation 2. For more details on `WOR_RES`, see Section 3.8.

$$t_{Event0}[\text{Programmed}] = \frac{1}{f_{RCOSC}} \cdot EVENT0 \cdot 2^{5 \cdot WOR\_RES}\,[\text{s}] \tag{3}$$

The RC oscillator has a tolerance of 0.1% after calibration (see [3] or [4]). This means that when programming t_{event0}, Equation 4 should be used.

$$t_{Event0}[\text{Programmed}] = \frac{100}{100.1} \cdot t_{Event0}[\text{Desired}] \tag{4}$$

3.3 Event 1 and t_{Event1}

t_{Event1} is the time between Event 0 and Event 1. If t_{Event1} is larger than the crystal start-up time, an `SRX` strobe is issued on this event. Setting t_{Event1} larger than the crystal start-up time is useful in applications where the transmitter and receiver are in sync and one needs to put the radio in RX mode at a known time. In a typical RX Sniff Mode application, t_{Event1} should be set shorter than the crystal start-up time. In these cases, the `SRX` strobe will be issued as soon as the crystal is stable (`CHIP_RDYn` is asserted). This way the radio enters RX mode as fast as possible, reducing the power consumption. t_{Event1} can be calculated as shown in Equation 5.

$$t_{Event1} = \frac{1}{f_{RCOSC}} \cdot m(WOR\_CFG1.WOR\_EVENT1)\,[\text{s}] \tag{5}$$

3.4 Event 2 and t_{Event2}

At Event 2, the radio can autonomously be taken out of SLEEP mode to perform RC oscillator calibration and improve the accuracy of the eWOR timer. The time between two Event 2's are called t_{Event2} and is given by Equation 6.

$$t_{Event2} = \frac{2^{m(WOR\_CFG0.WOR\_EVENT2)}}{f_{RCOSC}}\,[\text{s}] \tag{6}$$

When enabling calibration at Event 2 by setting `WOR_CFG0.WOR_EVENT2 ≠ 0`, t_{Event0} must be greater than t_{Event2} [1], [2]. Using the RX Sniff Mode does in most cases mean that t_{Event0} is much smaller than the minimum t_{Event2} (~1 s when f_{RCOSC} = 32 kHz and ~0.82 s when f_{RCOSC} = 40 kHz), therefore, `WOR_EVENT2` should in most RX Sniff Mode applications be set to 0.

3.5 RC Oscillator

To run the RX Sniff Mode, the internal RC oscillator must be enabled by setting `WOR_CFG0.RC_PD = 0`. In order to keep the frequency as accurate as possible, the RC oscillator needs to be calibrated. How often the RC oscillator should be calibrated is controlled through the `WOR_CFG0.RC_MODE` register field.

In an RX Sniff Mode application, the radio typically wakes up several times every second; it is recommended to do an initial calibration at start-up and then turn off the RC oscillator calibration (`RC_MODE = 0`). Re-calibration can then be initialized by the MCU.

The function found in Example 1 can be used to run a single RX oscillator calibration. In Example 1, cc112x should be replaced with cc120x in the function calls if the radio is a CC120X, and the register prefixes should be CC120X instead of CC112X.

Example 1. RC Oscillator Calibration (CC112x)

```
/*******************************************************************************
* @fn          calibrateRcOsc
*
* @brief       Calibrates the RC oscillator used for the eWOR timer. When this
*              function is called, WOR_CFG0.RC_PD must be 0
*
* @param       none
*
* @return      none
*/
static void calibrateRCOsc (void) {
  uint8 temp;

  // Read current register value
  cc112xSpiReadReg(CC112X_WOR_CFG0, &temp,1);

  // Mask register bit fields and write new values
  temp = (temp & 0xF9) | (0x02 << 1);

  // Write new register value
  cc112xSpiWriteReg(CC112X_WOR_CFG0, &temp,1);

  // Strobe IDLE to calibrate the RCOSC
  trxSpiCmdStrobe(CC112X_SIDLE);

  // Disable RC calibration
  temp = (temp & 0xF9) | (0x00 << 1);
  cc112xSpiWriteReg(CC112X_WOR_CFG0, &temp, 1);
}
```

3.6 Frequency Synthesizer Calibration

The internal on-chip FS characteristics varies with temperature and supply voltage changes as well as the desired operating frequency and must be calibrated regularly. Calibration can be done automatically when going to or from active states (RX and TX) by setting `SETTLING_CFG.FS_AUTOCAL ≠ 1`. Since the radio goes from IDLE to RX (and back to IDLE) several times a second when using the RX Sniff Mode, it is recommended to disable auto calibration `FS_AUTOCAL = 0` and do a manual calibration instead (when needed) to reduce current consumption. A manual calibration is performed by issuing an `SCAL` strobe command (for CC112x, see [7]).

3.7 RX Timeout

When using the RX Sniff Mode, RX is terminated when there is no carrier on the air or when no preamble is present, depending on the RFEND_CFG0.ANT_DIV_RX_TERM setting. For more details, see Section 3.8. Assume the radio wakes up in the beginning of the preamble and detects that a signal is present (CS or PQT). This means that ideally it should stay awake for a minimum time given by the preamble length plus the sync word length to be able to detect a sync word. The wakeup period, t_{Event0}, is programmed to be 4 bits or T0 + 10 bits shorter than the preamble length (see Section 3.2). It is not possible to program the RX timeout to be larger than t_{Event0}, therefore, the RX timeout must be disabled when running the RX Sniff Mode (RFEND_CFG1.RX_TIME = 111b).

The RX_TIME_QUAL bit in the RFEND_CFG1 register determines what happens when the RX timer expires. Since the RX Sniff Mode does not implement the RX timeout, this bit is don't care.

In a noisy environment, the MCU can be programmed to wake up if the radio detects a signal (CARRIER_SENSE or PQT_REACHED asserted). It can then start a timer before going back to sleep and wake up when the timer expires. If no sync is found, the MCU should put the radio back in SLEEP Mode by issuing an SIDLE strobe followed by an SWOR strobe (see Figure 2). Note that the radio needs to reach the IDLE state before the SWOR strobe can be issued.

Case A shows the format of the desired packet while case B shows how the MCU forces the radio back to SLEEP after a timeout equal to the preamble + sync length of the desired packet. In Case A, RX is terminated automatically after a packet is received (RFEND_CFG1.RXOFF_MODE = 0).

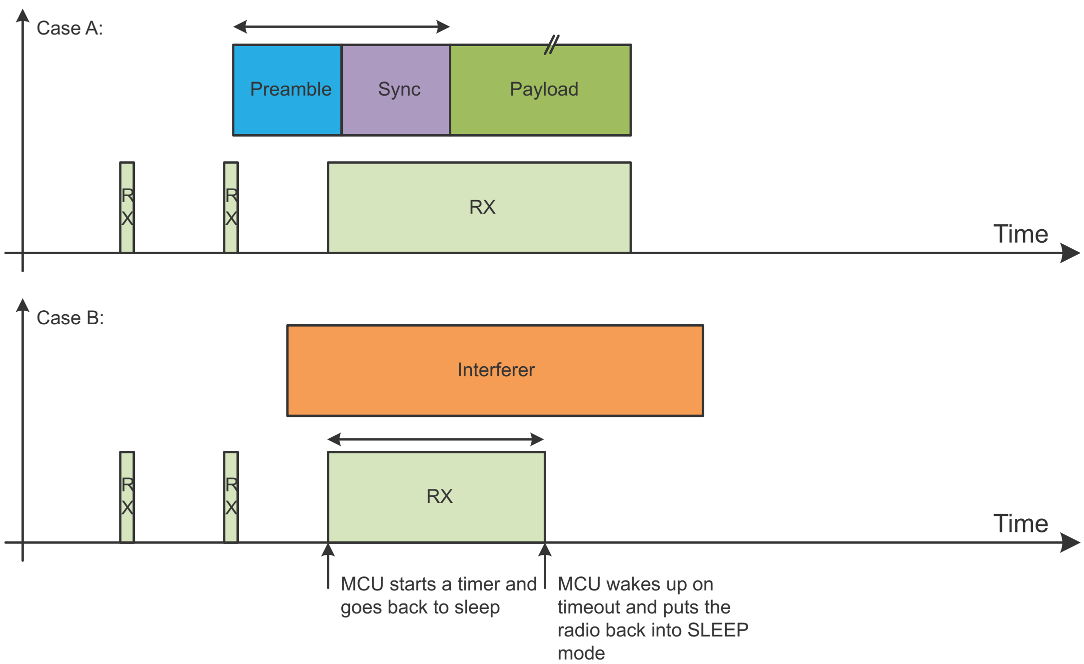


Figure 2. MCU Controlled RX Timeout

3.8 eWOR Timer Resolution

`WOR_CFG1.WOR_RES` is used to configure the eWOR timer resolution and the resolution of t_{Event0} (see Section 3.2).

Table 4. Resolution and Max t_{Event0} for Different Values of `WOR_RES`

	f_{RCOSC} = 32 kHz		f_{RCOSC} = 40 kHz	
WOR_RES	**Resolution**	**Max t_{Event0}**	**Resolution**	**Max t_{Event0}**
0	31.25 µs	2.048 s	25 µs	1.638 s
1	1 ms	65.536 s	0.8 ms	52.429 s
2	32 ms	2097.152 s	25.6 ms	1677.722 s
3	1.024 s	67108.864 s	819.2 ms	53687.091 s

3.9 RX Termination

When implementing the RX Sniff Mode, the radio should terminate RX as fast as possible if there is no signal on the air to minimize the current consumption. The radio can terminate the RX mode in lack of a carrier (`RFEND_CFG0.ANT_DIV_RX_TERM_CFG` = 1) or in lack of preamble (`RFEND_CFG0.ANT_DIV_RX_TERM_CFG` = 100_b). Detecting a carrier takes less time compared to detecting a preamble and t_{Event0} is shorter when RX termination is based on PQT compared to CS. Which RX termination to use (CS or PQT) depends on the system requirements and the environment the system is operating in. Consider the scenario shown in Figure 3. When the radio never triggers on noise, termination on CS gives the lowest total RX current. Figure 4 shows a scenario where noise is present in the channel and terminating on PQT gives the lowest RX current even if detecting a preamble takes longer than detecting a carrier. A third case is also shown in Figure 4. Here the MCU terminates RX after a timeout equal to the length of the preamble and sync word (the sync word is in this case half the length of the preamble). As seen from the figure, this leads to an even lower current consumption on the radio, but the MCU will draw some more current compared to case A and case B.

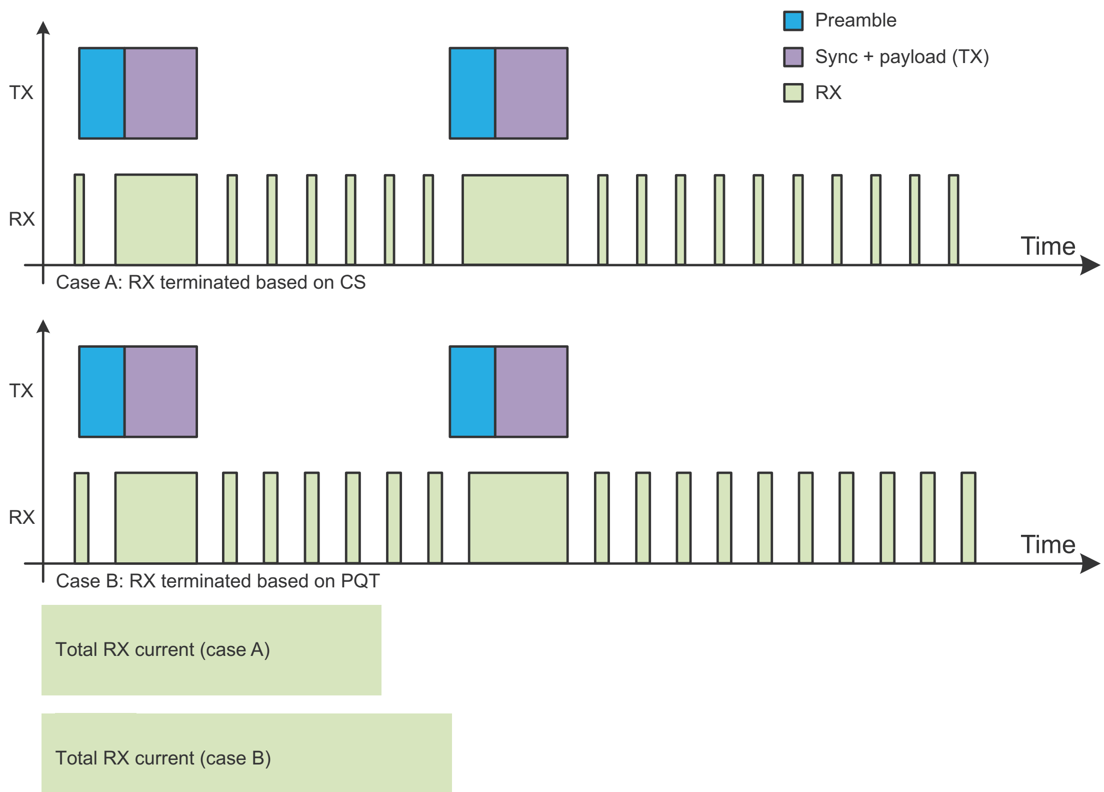


Figure 3. Total RX Consumption in a Noise Free Environment

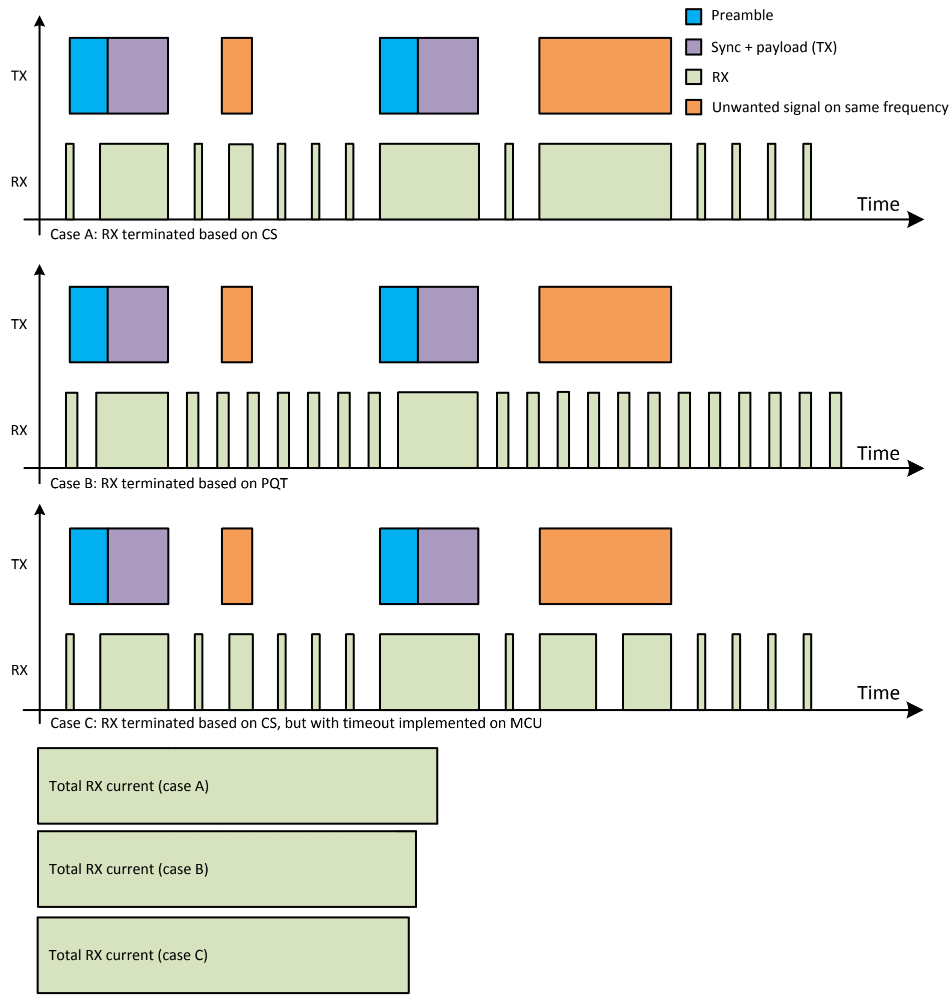


Figure 4. Total RX Current Consumption in Presence of Noise

3.9.1 Termination Based on CS

When termination based on CS is implemented (`RFEND_CFG0.ANT_DIV_RX_TERM_CFG = 1`), the CS threshold has to be set to a proper value. The CS threshold is programmed through the `AGC_CS_THRESHOLD` register field found in the `AGC_CS_THR` register. `AGC_CS_THRESHOLD` is a two's complement number with 1 dB resolution and is given by Equation 7. For details regarding the RSSI offset, see the device-specific user's guides [1], [2].

CS Threshold = AGC_CS_THR + RSSI Offset (7)

Setting the optimal threshold is a trade-off between sensitivity and current consumption. The CS threshold determines the sensitivity limit of the application as only packets with a signal strength above the threshold will be received. The CS threshold should be set as low as possible to achieve good sensitivity (close to the sensitive limit given in the radio's data sheet [3], [4], [5], [6]. However, setting the threshold too low causes more wake-ups due to noise and interference, and the current consumption will increase.

The carrier sense response time is the time it takes from when the radio enters RX mode until it can determine if there is a signal on the air or not. There are several registers that determines the CS response time and they are all discussed in the *RSSI* section of the user's guides [1], [2]. Two Excel sheets are available for calculating the CS response time for both CC112x and CC120x [9], [10]. It is recommended to use typical settings from SmartRF™ Studio [8] when implementing the RX Sniff Mode and then set `AGC_CFG1.AGC_SETTLE_WAIT`, `AGC_CFG1.AGC_WIN_SIZE`, and `AGC_CFG0.RSSI_VALID_COUNT to 0`. When doing this, there will be a trade-off between CS response time and accuracy. Note that changing registers from the recommended settings to reduce the CS response time might reduce the sensitivity, so testing must be done to assure satisfying RF performance.

3.9.2 Termination Based on PQT

When termination based on PQT is implemented (`RFEND_CFG0.ANT_DIV_RX_TERM_CFG = 100`$_b$), the preamble detector must be enabled by setting `PREAMBLE_CFG0.PQT_EN = 1`. The preamble threshold is configured with the register field `PREAMBLE_CFG0.PQT`. The PQT response time is the time it takes from the radio enters RX mode until it can determine if there is a valid preamble on the air or not. For more info on PQT, see the *Preamble Detection* section in the device-specific user's guide [1], [2].

3.10 Termination Due to Bad Packets

What happens after a bad packet has been received is determined by the `TERM_ON_BAD_PACKET` bit in the `RFEND_CFG0` register (in this context, a bad packet is a packet with the wrong length, wrong address, or with CRC error). When `TERM_ON_BAD_PACKET = 0,` the radio stays in RX regardless of the `RFEND_CFG1.RXOFF_MODE` when a bad packet has been discarded. When `TERM_ON_BAD_PACKET = 1`, the radio will go back to SLEEP after rejecting a bad packet given that `WOR_CFG1.WOR_MODE = 0 or 1.` For more details, see Section 3.11. When a good packet is received, the radio will enter the state indicated by the `RFEND_CFG1.RXOFF_MODE` setting.

When making a low power system, it is not just the radio that should minimize its current consumption. The MCU running the application should also reduce its current consumption as much as possible. One thing that can be done is to only wake up the MCU when there is a good packet in the RX FIFO and let the radio go back to SLEEP automatically otherwise. This means that `TERM_ON_BAD_PACKET` should be 1.

Since the radio only wants to wake up the MCU when there is a good packet in the RX FIFO, the `PKT_CRC_OK` should be used to wake up the MCU (`IOCFGx.GPIOx_CFG = PKT_CRC_OK` (19)). On the CC112x, the GPIO2 pin is the only GPIO pin that will be 0 also in SLEEP state when configured as PKT_CRC_OK, while on the CC120x, both GPIO0 and GPIO2 will be low in SLEEP state.

3.10.1 CRC Filtering

Since the radio is configured to go back to SLEEP if a packet is discarded (see Section 3.10), CRC filtering needs to be enabled. This is done by setting `FIFO_CFG.CRC_AUTOFLUSH = 1`. When this bit is set, it is important that the CRC check is enabled (`PKT_CFG1.CRC_CFG` $\neq$ `0`).

3.10.2 Maximum Packet Length Filtering

If the application uses variable packet length (`PKT_CFG0.LENGTH_CONFIG = 1`), maximum packet length filtering must be enabled. Maximum packet length filtering keeps the radio from entering RXFIFO_OVERFLOW state and also makes sure that packets that are too big for the RX FIFO are discarded so that the radio goes back to SLEEP (since `RFEND_CFG0.TERM_ON_BAD_PACKET = 1`).

The maximum allowed packet length must allow the complete packet, including optional status bytes, room in the RX FIFO. If `PKT_CFG1.APPEND_STATUS = 0`, max packet length is 127 and if `APPEND_STATUS = 1` max packet length is 125. The maximum allowed packet length is written to the `PKT_LEN.PACKET_LENGTH` register field.

In an application where the radio does not expect packets longer than 20 bytes (length byte + payload), `PACKET_LENGTH` should be set to 19 even if there are room for longer packet in the RX FIFO.

3.11 eWOR Mode

There are five different eWOR modes to select from, but only two that can be used if the radio should return to SLEEP automatically on the reception of a bad packet, as discussed in Section 3.10. These two are Feedback Mode (`WOR_CFG1.WOR_MODE = 0`) and Normal Mode (`WOR_CFG1.WOR_MODE = 1`). For both modes, the radio enters the state indicated by the `RFEND_CFG1.RXOFF_MODE` setting when a good packet is received, and enter SLEEP if a bad packet is received (since `RFEND_CFG0.TERM_ON_BAD_PACKET_EN` = 1). Using Feedback Mode, the radio enters IDLE instead of SLEEP if 16 bad packets are received in a row. For more details, see [1] and [2].

3.12 eWOR Clock Division

Setting `WOR_CFG0.DIV_256HZ_EN = 1` lowers the current consumption in SLEEP mode, but when this bit is set, the radio should not be woken from SLEEP by pulling CSn low. The setting of this bit depends on the application's need to manually wake up the radio.

4 Code Examples

There is one code example for the CC1120 and one for the CC1200 to supplement this application report. Both code examples can be downloaded from the web [11], [12].

4.1 CC1120 Code Example

The CC1120 code example uses the 1.2 kbps, 50 kHz RX filter BW typical settings from SmartRF Studio [8], version 1.13.0 as a starting point (see Figure 5).

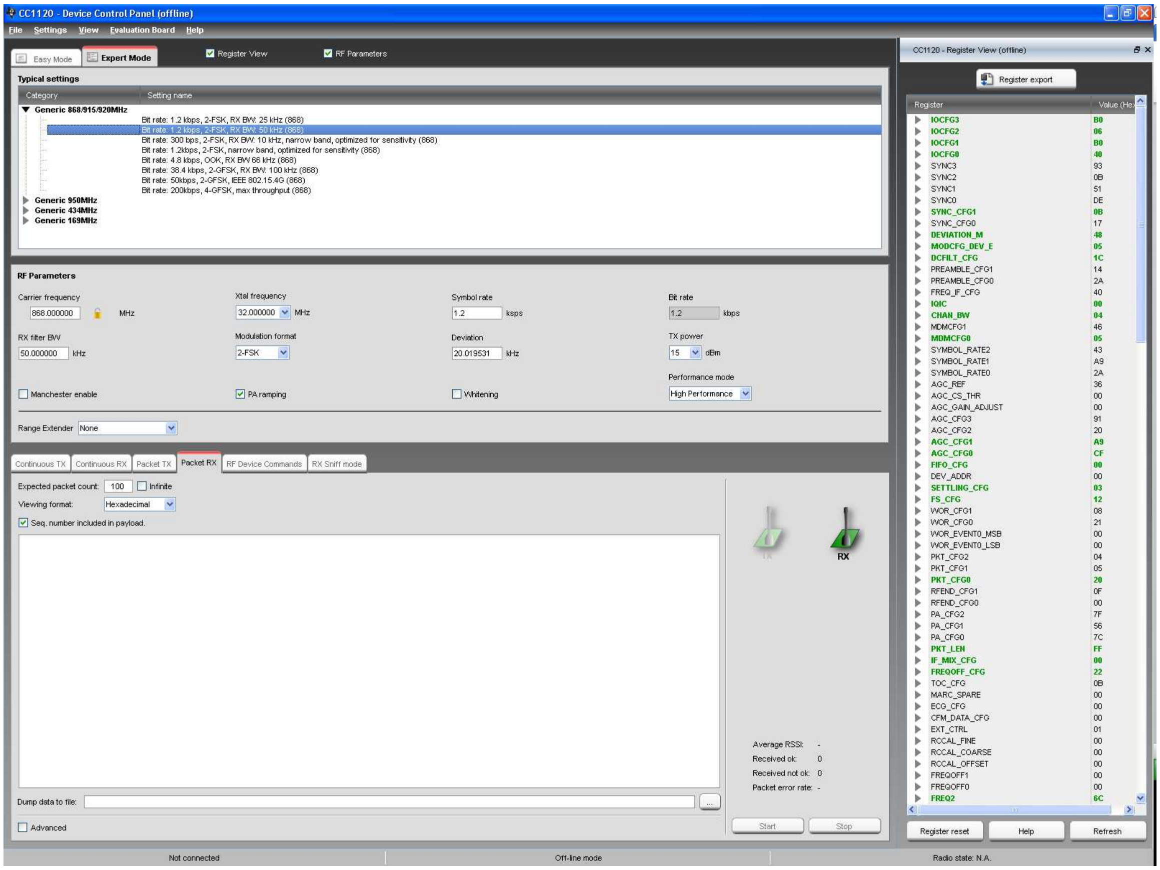


Figure 5. Typical Settings (CC1120, 1.2 kbps, 50 kHz RX filter BW)

The register settings obtained by using the code export feature is shown in Example 2.

Example 2. Code Export (CC1120, 1.2 kbps, 50 kHz RX filter BW)

```
// RX filter BW = 50.000000
// Address config = No address check
// Packet length = 255
// Symbol rate = 1.2
// PA ramping = true
// Performance mode = High Performance
// Carrier frequency = 868.000000
// Bit rate = 1.2
// Packet bit length = 0
// Whitening = false
// Manchester enable = false
// Modulation format = 2-FSK
// Packet length mode = Variable
// Device address = 0
// TX power = 15
// Deviation = 20.019531
// Rf settings for CC1120
static const registerSetting_t preferredSettings[] = {
  {CC112X_IOCFG3,        0xB0},
  {CC112X_IOCFG2,        0x06},
  {CC112X_IOCFG1,        0xB0},
  {CC112X_IOCFG0,        0x40},
  {CC112X_SYNC_CFG1,     0x0B},
  {CC112X_DEVIATION_M,   0x48},
  {CC112X_MODCFG_DEV_E,  0x05},
  {CC112X_DCFILT_CFG,    0x1C},
  {CC112X_IQIC,          0x00},
  {CC112X_CHAN_BW,       0x04},
  {CC112X_MDMCFG0,       0x05},
  {CC112X_AGC_CFG1,      0xA9},
  {CC112X_AGC_CFG0,      0xCF}
  {CC112X_FIFO_CFG,      0x00},
  {CC112X_SETTLING_CFG,  0x03},
  {CC112X_FS_CFG,        0x12},
  {CC112X_PKT_CFG0,      0x20},
  {CC112X_PKT_LEN,       0xFF},
  {CC112X_IF_MIX_CFG,    0x00},
  {CC112X_FREQOFF_CFG    0x22},
  {CC112X_FREQ2,         0x6C},
  {CC112X_FREQ1,         0x80},
  {CC112X_FS_DIG1,       0x00},
  {CC112X_FS_DIG0,       0x5F},
  {CC112X_FS_CAL1,       0x40},
  {CC112X_FS_CAL0,       0x0E},
  {CC112X_FS_DIVTWO,     0x03},
  {CC112X_FS_DSM0,       0x33},
  {CC112X_FS_DVC0,       0x17},
  {CC112X_FS_PFD,        0x50},
  {CC112X_FS_PRE,        0x6E},
  {CC112X_FS_REG_DIV_CML 0x14},
  {CC112X_FS_SPARE,      0xAC},
  {CC112X_FS_VCO0,       0xB4},
  {CC112X_XOSC5,         0x0E},
  {CC112X_XOSC1,         0x03},
};
```

Three different configurations (see Figure 6) are available for this example and they are discussed in the following sections (Section 4.1.1, Section 4.1.2, and Section 4.1.3).

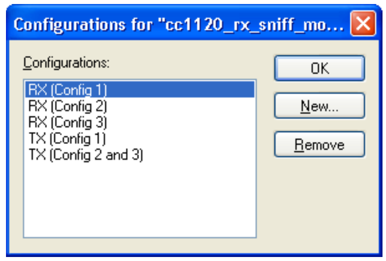


Figure 6. Code Configurations

4.1.1 RX Config. 1

In this configuration, three preamble bytes are used since this is the preamble configuration given by SmartRf Studio (see Figure 5).

- **t_{Event0} Configuration**

  Equation 8 shows t_{Event0} as maximum 16.67 ms.

$$t_{Event0} = \frac{\text{\# of Preamble Bits - 4}}{\text{Data Rate [bps]}} = \frac{3 \cdot 8 - 4}{1200} = 16.67 \cdot 10^{-3}\ \text{s} \tag{8}$$

  This means that `WOR_RES` should be 0 (see Table 4).

  Solving Equation 9 with respect to EVENT0, you get EVENT0 to be 532.80.

$$EVENT0 = t_{Event0}[\text{Programmed}] \cdot f_{RCOSC} = \frac{100}{100.1} \cdot t_{Event0}[\text{Desired}] \cdot f_{RCOSC} = \frac{100}{100.1} \cdot 16.67 \cdot 10^{-3} \cdot 32 \cdot 10^{-3} = 532.80 \tag{9}$$

  EVENT0 = 532.80 means that `EVENT0_15_8` = `0x02` and `EVENT0_7_0` = `0x14`.

  In addition, `MDMCFG1.CARRIER_SENSE_GATE` = `0` and `SYNC_CFG1.PQT_GATING` = `0` as discussed in Section 3.2.

- **t_{Event1} Configuration**

  According to the CC1120 data sheet [3], the start-up time of the crystal is typical 400 µs, therefore, `EVENT 1` should be less than 100_b. In the CC1120 RX Sniff Mode code example [11] `EVENT1 = 000`$_b$.

- **t_{Event2} Configuration**

  Since t_{Event0} is 16.67 ms, t_{Event0} is less than the minimum t_{Event2} (1 s) and `WOR_EVENT2_CFG` should be set to 0 (see Section 3.4).

- **RC Oscillator Configuration**

  `WOR_CFG0.RC_PD` and `WOR_CFG0.RC_MODE` both set to 0 as described in Section 3.5.

- **FS Calibration**

  `SETTLING_CFG.FS_AUTOCAL` = `0` as stated in Section 3.6.

- **RX Timeout Configuration**

RFEND_CFG1.RX_TIME = 111_b and RFEND_CFG1.RX_TIME_QUAL = x as described in Section 3.7 (RX_TIME_QUAL = 1 in the code example [11] since this is the default value of this bit). No timeout is implemented, meaning the radio will stay in RX as long as there is a signal on the air.

- **RX Termination Configuration**

  In the code example [11], termination based on carrier sense is implemented. This means that RFEND_CFG0.ANT_DIV_RX_TERM_CFG = 1. Because of this, PREAMBLE_CFG0.PQT_EN and PREAMBLE_CFG0.PQT are don't care and the default values are being used (PREAMBLE_CFG0.PQT_EN = 1 and PREAMBLE_CFG0.PQT = 1010_b)

  When ANT_DIV_RX_TERM_CFG = 1, the CS threshold must be set to something other than default. The sensitivity limit for the chosen RF settings is -114 dBm [3]. In the code example [11], CS threshold is set 1 dB higher, at -113 dBm, to avoid too many wake-ups due to noise.

  The RSSI offset is -102 dBm [1]. To get a CS threshold of -113 dBm, AGC_CS_THR must be set to 0xF5 (see Equation 10).

$$\begin{aligned} &\text{CS Threshold} = \text{AGC\_CS\_THR} + \text{RSSI Offset} \\ &\text{AGC\_CS\_THR} = \text{CS Threshold} - \text{RSSI Offset} = -113 - (-102) = -11 \end{aligned} \tag{10}$$

  To reduce the CS response time, AGC_CFG1.AGC_WIN_SIZE, AGC_CFG1.AGC_SETTLE_WAIT and AGC_CFG0.RSSI_VALID_COUNT are all set to 0.

- **"Bad Packet" Termination**

  To save as much power as possible, the radio is configured to return to SLEEP if a bad packet is received and to only wake up the MCU if a good packet is received, therefore, RFEND_CFG0.TERM_ON_BAD_PACKET = 1 and IOCFG2.GPIO2_CFG = 010011_b (see Section 3.10). (1)

  In addition, FIFO_CFG.CRC_AUTOFLUSH must be 1 and PKT_CFG1.CRC_CFG ≠ 0. In the code example [11], CRC_CFG = 01_b as this is the default value of this register field.

  RFEND_CFG1.RXOFF_MODE is set to 0 (IDLE) making the radio enter IDLE state when a good packet is received.

  Since the radio should only wake up the MCU when a good packet is received, the radio must be configured to avoid entering RX FIFO ERROR state. The settings given in Example 2 indicates variable packet length mode PKT_CFG0.LENGTH_CFG = 01_b and PKT_CFG1.APPEND_STATUS = 1, therefore, PKT_LEN should be set to 125 (0x7D) (see Section 3.10.2)

- **eWOR Mode**

  To make the receiver SW as simple as possible, Normal mode (WOR_CFG1.WOR_MODE = 1) is selected for this code example [11].

- **eWOR Clock Division**

  WOR_CFG0.DIV_256HZ_EN = 1 to make the current consumption as low as possible.

(1) IOCFG0.GPIO0_CFG = 000110b and is used by the TX to determine when a packet has been sent.

Table 5. Register Fields to be Changed When Running RX Sniff Mode (RX config. 1)

Register Name	Register Field	Value	Register Value
WOR_CFG1	WOR_RES	00_b	WOR_CFG1 = 0x08
	WOR_MODE	001_b	
	EVENT1	000_b	
WOR_CFG0	WOR_CFG_NOT_USED	00_b	WOR_CFG0 = 0x20
	DIV_256HZ_EN	1_b	
	EVENT2_CFG	00_b	
	RC_MODE	00_b	
	RC_PD	0_b	
WOR_EVENT0_MSB	EVENT0_15_8	0x02	WOR_EVENT0_MSB = 0x02
WOR_EVENT0_LSB	EVENT0_7_0	0x15	WOR_EVENT0_LSB = 0x15
SETTLING_CFG	SETTLING_CFG_NOT_USED	000_b	SETTLING_CFG = 0x03
	FS_AUTOCAL	00b	
	LOCK_TIME	01_b	
	FSREG_TIME	1_b	
RFEND_CFG1	RFEND_CFG1_NOT_USED	00_b	RFEND_CFG1 = 0x0F
	RXOFF_MODE	00_b	
	RX_TIME	111_b	
	RX_TIME_QUAL	1_b	
RFEND_CFG0	RFEND_CFG0_NOT_USED	0_b	RFEND_CFG0 = 0x09
	CAL_EN_WAKE_UP_EN	0_b	
	TXOFF_MODE	00_b	
	TERM_ON_BAD_PACKET	1_b	
	ANT_DIV_RX_TERM_CFG	001_b	
AGC_CS_THR	AGC_CS_THRESHOLD	0xF5	AGC_CS_THR = 0xF5
AGC_CFG1	AGC_SYNC_BEHAVIOUR	101_b	AGC_CFG1 = 0xA0
	AGC_WIN_SIZE	000_b	
	AGC_SETTLE_WAIT	00_b	
AGC_CFG0	AGC_HYST_LEVEL	11_b	AGC_CFG0 = 0xC3
	AGC_SLEWRATE_LIMIT	00_b	
	RSSI_VALID_COUNT	00_b	
	AGC_ASK_DECAY	11_b	
PREAMBLE_CFG0	PREAMBLE_CFG0_NOT_USED	00_b	PREAMBLE_CFG0 = 0x2A
	PQT_EN	1_b	
	PQT_VALID_TIMEOUT	0_b	
	PQT	1010_b	
FIFO_CFG	CRC_AUTOFLUSH	1_b	FIFO_CFG = 0x80
	FIFO_THR	0000000_b	

Table 5. Register Fields to be Changed When Running RX Sniff Mode (RX config. 1) (continued)

Register Name	Register Field	Value	Register Value
PKT_CFG1	PKT_CFG1_NOT_USED	0_b	PKT_CFG1 = 0x05
	WHITE_DATA	0_b	
	ADDR_CHECK_CFG	00_b	
	CRC_CFG	01_b	
	BYTE_SWAP_EN	0_b	
	APPEND_STATUS	1_b	
PKT_CFG0	PKT_CFG0_RESERVED7	0_b	PKT_CFG0 = 0x20
	LENGTH_CONFIG	01_b	
	PKT_BIT_LEN	000_b	
	UART_MODE_EN	0_b	
	UART_SWAP_EN	0_b	
PKT_LEN	PACKET_LENGTH	0x7D	PKT_LEN = 0x7D
IOCFG2	GPIO2_ATRAN	0_b	IOCFG2 = 0x13
	GPIO2_INV	0_b	
	GPIO2_CFG	010011_b	
MDMCFG1	CARRIER_SENSE_GATE	0	MDMCFG1 = 0x46
	FIFO_EN	1	
	MANCHESTER_EN	0	
	INVERT_DATA_EN	0	
	COLLISION_DETECT_EN	0	
	DVGA_GAIN	11b	
	SINGLE_ADC_EN	0	
SYNC_CFG1	SYNC_CFG1_RESERVED7	0	SYNC_CFG1 = 0x0B
	PQT_GATING_EN	0	
	SYNC_CFG1_RESERVED5	0	
	SYNC_THR	1011b	

Example 3. RX Sniff Mode Settings (CC1120, 1.2 kbps, 50 kHz RX filter BW, RX config. 1)

```
// RX filter BW = 50.000000
// Address config = No address check
// Packet length = 125
// Symbol rate = 1.2
// PA ramping = true
// Performance mode = High Performance
// Carrier frequency = 868.000000
// Bit rate = 1.2
// Packet bit length = 0
// Whitening = false
// Manchester enable = false
// Modulation format = 2-FSK
// Packet length mode = Variable
// Device address = 0
// TX power = 15
// Deviation = 20.019531
// Rf settings for CC1120
static const registerSetting_t preferredSettings[] = {
  {CC112X_IOCFG3,           0xB0},
  {CC112X_IOCFG2,           0x13},
  {CC112X_IOCFG1,           0xB0},
  {CC112X_IOCFG0,           0x06},
  {CC112X_SYNC_CFG1,        0x0B},
  {CC112X_DEVIATION_M,      0x48},
  {CC112X_MODCFG_DEV_E,     0x05},
  {CC112X_DCFILT_CFG,       0x1C},
  {CC112X_IQIC,             0x00},
  {CC112X_CHAN_BW,          0x04},
  {CC112X_MDMCFG0,          0x05},
  {CC112X_AGC_CS_THR,       0xF5},
  {CC112X_AGC_CFG1,         0xA0},
  {CC112X_SETTLING_CFG,     0x03},
  {CC112X_FS_CFG,           0x12},
  {CC112X_WOR_CFG0,         0x20},
  {CC112X_WOR_EVENT0_MSB,   0x02},
  {CC112X_WOR_EVENT0_LSB,   0x14},
  {CC112X_PKT_CFG0,         0x20},
  {CC112X_RFEND_CFG0,       0x09},
  {CC112X_PKT_LEN,          0x7D},
  {CC112X_IF_MIX_CFG,       0x00},
  {CC112X_FREQOFF_CFG,      0x22},
  {CC112X_FREQ2,            0x6C},
  {CC112X_FREQ1,            0x80},
  {CC112X_FS_DIG1,          0x00},
  {CC112X_FS_DIG0,          0x5F},
  {CC112X_FS_CAL1,          0x40},
  {CC112X_FS_CAL0,          0x0E},
  {CC112X_FS_DIVTWO,        0x03},
  {CC112X_FS_DSM0,          0x33},
  {CC112X_FS_DVC0,          0x17},
  {CC112X_FS_PFD,           0x50},
  {CC112X_FS_PRE,           0x6E},
  {CC112X_FS_REG_DIV_CML,   0x14},
  {CC112X_FS_SPARE,         0xAC},
  {CC112X_FS_VCO0,          0xB4},
  {CC112X_XOSC5,            0x0E},
  {CC112X_XOSC1,            0x03},
};
```

4.1.1.1 Measurements and Estimates (RX Config. 1)

Using the CC112x_RX_Sniff_Mode Excel sheet [9] to estimate the CS response time, you get ~540 µs (see Figure 7).

	A	B
1	**Settings**	
2	f_{XOSC} [MHz]	32
3	DCFILT_CFG.DCFILT_BW	4
4	DCFILT_CFG.DCFILT_FREEZE_COEFF	0
5	MDMCFG1.CARRIER_SENSE_GATE	0
6	CHAN_BW.BB_CIC_DECFACT	4
7	CHAN_BW.ADC_CIC_DECFACT	0
8	Decimation factor	20
9	CHAN_BW.CHFILT_BYPASS	0
10	AGC_CFG1.AGC_WIN_SIZE	0
11	AGC_CFG1.SETTLE_WAIT	0
12	AGC_CFG0.RSSI_VALID_CNT	0
13	# of AGC_UPDATE pulses	1
14	**Results**	
15	D0 [us]	12,31 µs
16	D1 [us]	77,50 µs
17	D2 [us]	77,50 µs
18	D3 [us]	77,50 µs
19	D4 [us]	43,75 µs
20	D5 [us]	170,00 µs
21	D6 [us]	2,50 µs
22	T0 = D0+D1+D3+D5	337,31 µs
23	T1	120,00 µs
24	T2	41,44 µs
25	**CS response time (# of AGC_UPDATE pulses is known)**	**540,19 µs**
26	**Max CS Response Time**	**660,19 µs**

Figure 7. CS Response Time (CC1120, 1.2 kbps, 50 kHz RX filter BW)

Figure 8 shows the current profile when running the CC1120 RX Sniff Mode code . The RX time is measured to be 540 µs, the same as the estimated CS response time found using the Excel sheet [9] (see Figure 7).

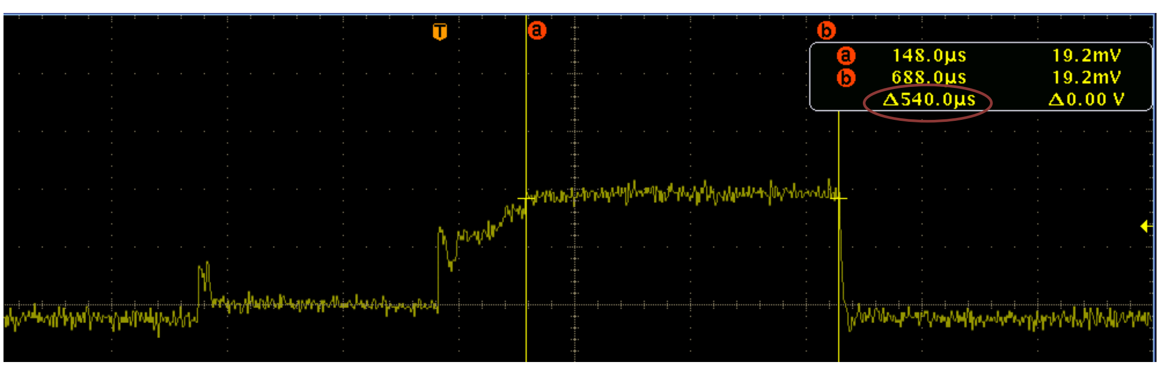


Figure 8. Current Profile (CC1120, 1.2 kbps, 50 kHz RX filter BW)

Figure 9 shows the measurement of the average current consumption when running the RX Sniff Mode (RX config. 1). The average current consumption (when there is no data on the air) is 0.905 mA (905 µV measured over a 1Ω resistor). This is almost the same as estimated in CC112x_RX_Sniff_Mode.xlsx [9] (see Figure 10).

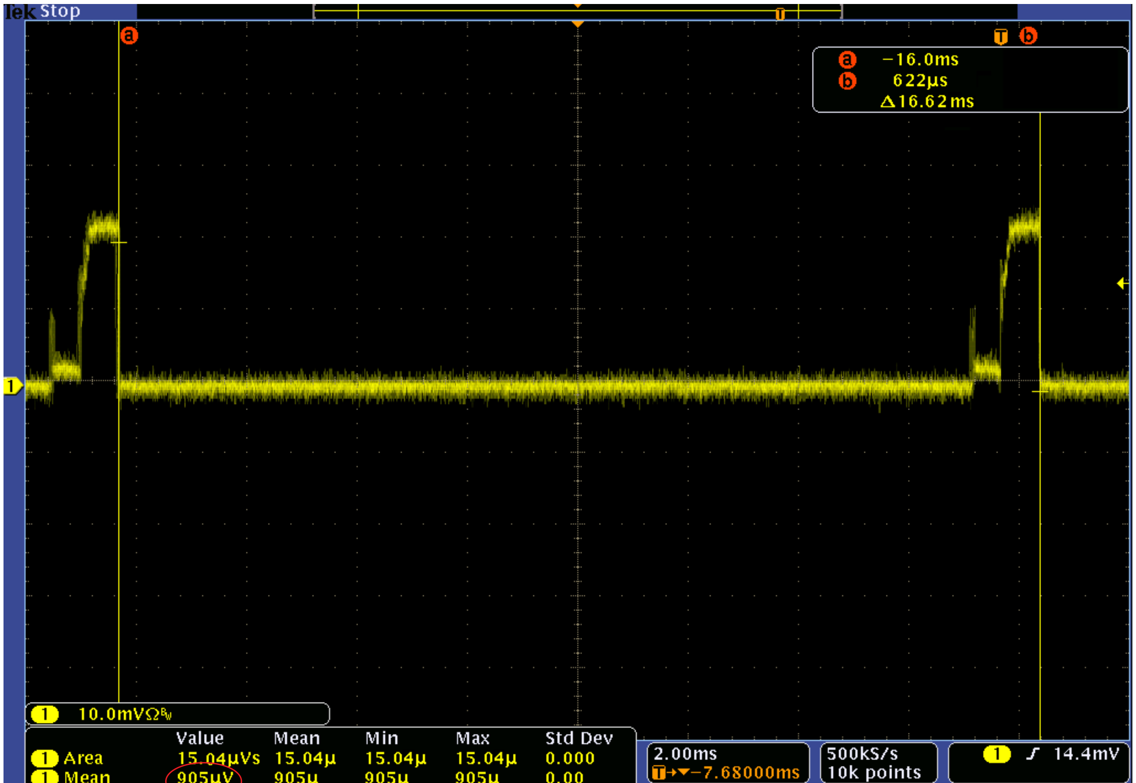


Figure 9. Average Current Consumption (CC1120, 1.2 kbps, 50 kHz RX Filter BW, RX config. 1)

1	Input Parameters:	
2	Number of Preamble Bytes	3
3	SYMBOL_RATE2	43
4	SYMBOL_RATE1	A9
5	SYMBOL_RATE0	2A
6	Data Rate [sps]	1200
7	f_{XOSC} [MHz]	32
8	f_{RCOCS} [kHz]	32
9	DCFILT_CFG	1C
10	MDMCFG1	46
11	CHAN_BW	04
12	AGC_CFG1	A0
13	AGC_CFG0	C3
14	PREAMBLE_CFG0	2A
15	Termination Based on:	CS
16		
17	Average Current Consumption	0,904 mA
18	t_{Event0}	16,63 ms
19	EVENT0 (hex)	0214

Figure 10. CC112x_RX_Sniff_Mode.xlsx Dashboard (RX config. 1)

4.1.2 RX Config. 2

Many existing protocols use 4 bytes of preamble and 4 bytes sync. When increasing the preamble from 3 (RX config. 1) to 4 bytes, the average current consumption is reduced from 0.904 mA to 0.646 mA as shown in Figure 11, as t_{Event0} can be increased from 16.63 ms to 23.28 ms.

1	Input Parameters:	
2	Number of Preamble Bytes	4
3	SYMBOL_RATE2	43
4	SYMBOL_RATE1	A9
5	SYMBOL_RATE0	2A
6	Data Rate [sps]	1200
7	f_{XOSC} [MHz]	32
8	f_{RCOCS} [kHz]	32
9	DCFILT_CFG	1C
10	MDMCFG1	46
11	CHAN_BW	04
12	AGC_CFG1	A0
13	AGC_CFG0	C3
14	PREAMBLE_CFG0	2A
15	Termination Based on:	CS
16		
17	Average Current Consumption	0,646 mA
18	t_{Event0}	23,28 ms
19	EVENT0 (hex)	02E9

Figure 11. CC112x_RX_Sniff_Mode.xlsx Dashboard (RX config. 2)

4.1.3 RX Config. 3

Assume a system using 4 bytes of preamble and a 4 bytes long sync word. The CC112x (and CC120x) can be configured to look for an 11 bits sync word. This means that the receiver can sleep for part of the sync word as well (see Figure 12), increasing t_{Event0} and reducing the current consumption.

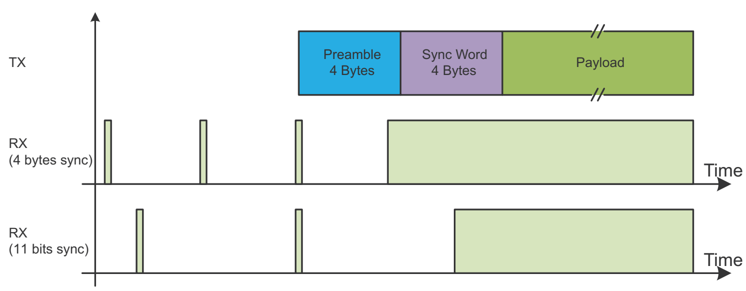


Figure 12. 4 Bytes Sync Word vs. 11 Bits Sync Word

In this case, the effective preamble is 4 bytes + (32 – 11 bits) = 53 bits = 6.625 bytes and the average current consumption is estimated to be 0.369 mA (see Figure 13).

1	Input Parameters:	
2	Number of Preamble Bytes	6,625
3	Data Rate [sps]	1200
4	f_{XOSC} [MHz]	32
5	f_{RCOCS} [kHz]	32
6	DCFILT_CFG	1C
7	MDMCFG1	46
8	CHAN_BW	4
9	AGC_CFG1	A0
10	AGC_CFG0	C3
11		
12	Average Current Consumption	0,369 mA
13	t_{Event0}	40,83 ms
14	EVENT0 (hex)	051A

Figure 13. CC112x_RX_Sniff_Mode.xlsx Dashboard (RX config. 3)

4.2 CC1200 Code Example

The CC1200 code example [12] uses the 38.4 kbps typical settings from the SmartRF Studio [8] version 1.13.0 as a starting point (see Figure 14). The register settings will be altered to achieve a 1.5 mA average current consumption.

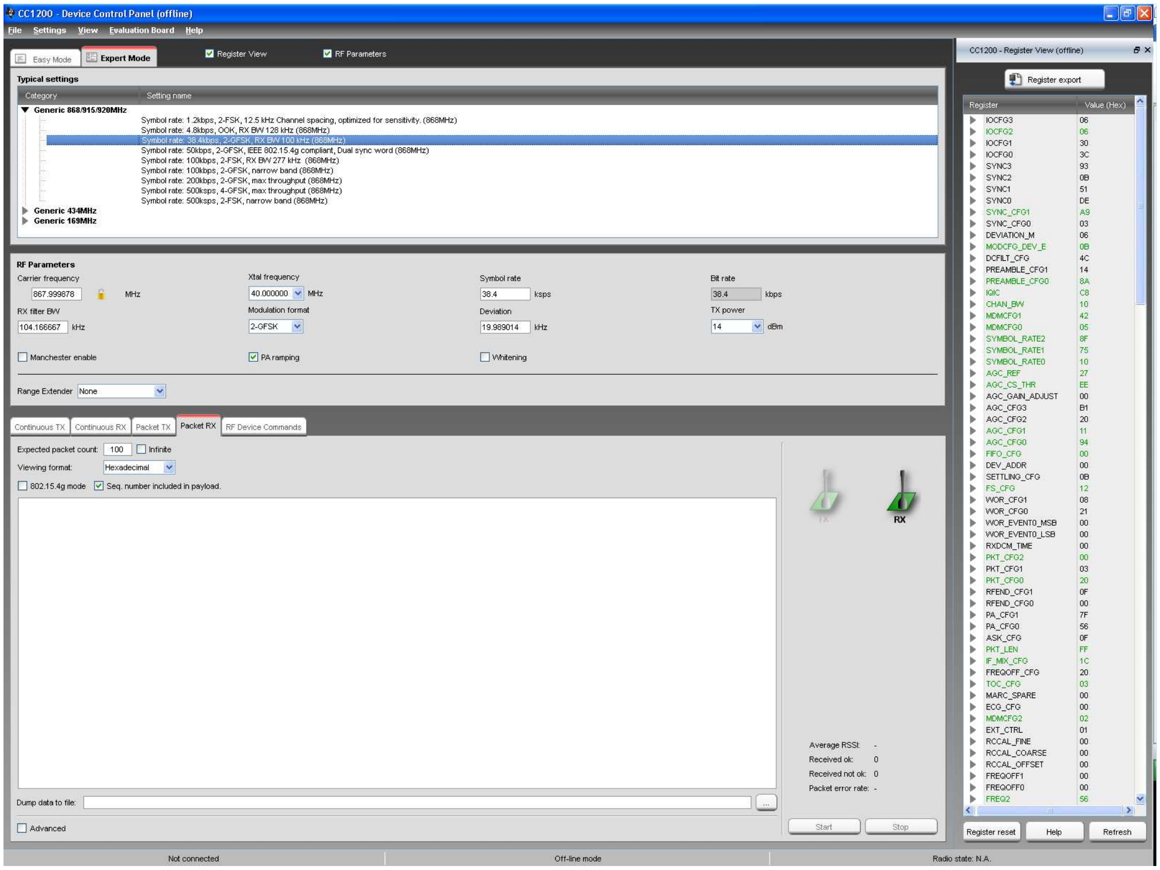


Figure 14. Typical Settings (CC1200, 38.4 kbps, ETSI standard)

The register settings obtained by using the code export feature is shown in Example 4.

Example 4. Code Export (CC1200, 38.4 kbps, ETSI standard)

```
// RX filter BW = 104.166667
// Address config = No address check
// Packet length = 255
// Symbol rate = 38.4
// Carrier frequency = 867.999878
// Bit rate = 38.4
// Packet bit length = 0
// Whitening = false
// Manchester enable = false
// Modulation format = 2-GFSK
// Packet length mode = Variable
// Device address = 0
// Deviation = 19.989014
// Rf settings for CC1200
static const registerSetting_t preferredSettings[] = {
  {CC120X_IOCFG2,           0x06},
  {CC120X_SYNC_CFG1,        0xA9},
  {CC120X_MODCFG_DEV_E,     0x0B},
  {CC120X_PREAMBLE_CFG0,    0x8A},
  {CC120X_IQIC,             0xC8},
  {CC120X_CHAN_BW,          0x10},
  {CC120X_MDMCFG1,          0x42},
  {CC120X_MDMCFG0,          0x05},
  {CC120X_SYMBOL_RATE2,     0x8F},
  {CC120X_SYMBOL_RATE1,     0x75},
  {CC120X_SYMBOL_RATE0,     0x10},
  {CC120X_AGC_REF,          0x27},
  {CC120X_AGC_CS_THR,       0xEE},
  {CC120X_AGC_CFG1,         0x11},
  {CC120X_AGC_CFG0,         0x94},
  {CC120X_FIFO_CFG,         0x00},
  {CC120X_FS_CFG,           0x12},
  {CC120X_PKT_CFG2,         0x00},
  {CC120X_PKT_CFG0,         0x20},
  {CC120X_PKT_LEN,          0xFF},
  {CC120X_IF_MIX_CFG,       0x1C},
  {CC120X_TOC_CFG,          0x03},
  {CC120X_MDMCFG2,          0x02},
  {CC120X_FREQ2,            0x56},
  {CC120X_FREQ1,            0xCC},
  {CC120X_FREQ0,            0xCC},
  {CC120X_IF_ADC1,          0xEE},
  {CC120X_IF_ADC0,          0x10},
  {CC120X_FS_DIG1,          0x07},
  {CC120X_FS_DIG0,          0xAF},
  {CC120X_FS_CAL1,          0x40},
  {CC120X_FS_CAL0,          0x0E},
  {CC120X_FS_DIVTWO,        0x03},
  {CC120X_FS_DSM0,          0x33},
  {CC120X_FS_DVC0,          0x17},
  {CC120X_FS_PFD,           0x00},
  {CC120X_FS_PRE,           0x6E},
  {CC120X_FS_REG_DIV_CML,   0x1C},
  {CC120X_FS_SPARE,         0xAC},
  {CC120X_FS_VCO0,          0xB5},
  {CC120X_IFAMP,            0x09},
  {CC120X_XOSC5,            0x0E},
  {CC120X_XOSC1,            0x03},
};
```

`AGC_WIN_SIZE`, `AGC_SETTLE_WAIT`, and `RSSI_VALID_COUNT` were all set to 0 (see Section 3.9.1) and FS auto calibration was turned off (`SETTLING_CFG.FS_AUTOCAL = 0`). The register values were input to the CC120x_RX_Sniff_Mode Excel sheet [10] as shown in Figure 15. Figure 15 also shows that the number of preamble bytes is too low to be able to run the RX Sniff Mode. (1)

1	Input Parameters:		
2	Number of Preamble Bytes	3	Number of preamble is too low
3	SYMBOL_RATE2	8F	
4	SYMBOL_RATE1	75	
5	SYMBOL_RATE0	10	
6	Data Rate [sps]	38400	
7	f_{XOSC} [MHz]	40	
8	f_{RCOSC} [kHz]	40	
9	MDMCFG2	2	
10	MDMCFG0	5	
11	IQIC	C8	
12	SYNC_CFG0	3	
13	CHAN_BW	10	
14	DCFILT_CFG	4C	
15	IF_MIX_CFG	1C	
16	AGC_CFG1	0	
17	AGC_CFG0	90	
18	PREAMBLE_CFG0	8A	
19	Termination Based on:	CS	
20			
21	Average Current Consumption	Error	
22	t_{Event0}	Error	
23	EVENT0 (hex)	Error	

Figure 15. CC120x_RX_Sniff_Mode.xlsx Dashboard (3 bytes preamble)

To be able to run the RX Sniff Mode and achieve an average current consumption of ~1.5 mA, the number of preamble bytes transmitted must be increased to 24 bytes, as shown in Figure 16. It can easily be seen that reducing the current on the RX side means increased current consumption on the TX side.

1	Input Parameters:	
2	Number of Preamble Bytes	24
3	SYMBOL_RATE2	8F
4	SYMBOL_RATE1	75
5	SYMBOL_RATE0	10
6	Data Rate [sps]	38400
7	f_{XOSC} [MHz]	40
8	f_{RCOSC} [kHz]	40
9	MDMCFG2	2
10	MDMCFG0	5
11	IQIC	C8
12	SYNC_CFG0	3
13	CHAN_BW	10
14	DCFILT_CFG	4C
15	IF_MIX_CFG	1C
16	AGC_CFG1	0
17	AGC_CFG0	90
18	PREAMBLE_CFG0	8A
19	Termination Based on:	CS
20		
21	Average Current Consumption	1,518 mA
22	t_{Event0}	4,88 ms
23	EVENT0 (hex)	00C3

Figure 16. CC120x_RX_Sniff_Mode.xlsx Dashboard (24 bytes preamble)

The sensitivity limit for the chosen RF settings is -110 dBm (see the CC1200 data sheet [6]). In the code example [12], the CS threshold is set 1 dB higher, at -109 dBm, to avoid too many wake-ups due to noise.

The RSSI offset is -81 dBm (see the CC120x user's guide [2]). To get a CS threshold of -109 dBm, AGC_CS_THR must be set to 0xE4 (use Equation 11).

$$\text{CS Threshold} = \text{AGC\_CS\_THR} + \text{RSSI Offset}$$
$$\text{AGC\_CS\_THR} = \text{CS Threshold} - \text{RSSI Offset} = -109 - (-81) = -28 \quad (11)$$

The rest of the relevant register fields were changed in the same manner as explained for the CC1120 example [11] (see Section 4.1) and the registers settings used by the CC1200 RX Sniff Mode code example [12] are shown in Example 5.

(1) A solution might be to use the RX Duty Cycle Mode as the RX duty cycle timer provides smaller timeouts on the time not spent in RX, but this application report does not go into details on this topic. The CC120x user's guide [2] provides more info on this mode.

Example 5. RX Sniff Mode Settings (CC1200, 38.4 kbps, ETSI standard)

```
// RX filter BW = 104.166667
// Address config = No address check
// Packet length = 125
// Symbol rate = 38.4
// Carrier frequency = 867.999878
// Bit rate = 38.4
// Packet bit length = 0
// Whitening = false
// Manchester enable = false
// Modulation format = 2-GFSK
// Packet length mode = Variable
// Device address = 0
// Deviation = 19.989014
// Rf settings for CC1200
static const registerSetting_t preferredSettings[] = {
  {CC120X_IOCFG2,            0x13},
  {CC120X_IOCFG0,            0x06},
  {CC120X_SYNC_CFG1,         0xA9},
  {CC120X_MODCFG_DEV_E,      0x0B},
  {CC120X_PREAMBLE_CFG1,     0x30},
  {CC120X_PREAMBLE_CFG0,     0x8A},
  {CC120X_IQIC,              0xC8},
  {CC120X_CHAN_BW,           0x10},
  {CC120X_MDMCFG1,           0x42},
  {CC120X_MDMCFG0,           0x05},
  {CC120X_SYMBOL_RATE2,      0x8F},
  {CC120X_SYMBOL_RATE1,      0x75},
  {CC120X_SYMBOL_RATE0,      0x10},
  {CC120X_AGC_REF,           0x27},
  {CC120X_AGC_CS_THR,        0xE4},
  {CC120X_AGC_CFG1,          0x00},
  {CC120X_AGC_CFG0,          0x90},
  {CC120X_SETTLING_CFG,      0x03},
  {CC120X_FS_CFG,            0x12},
  {CC120X_WOR_CFG0,          0x20},
  {CC120X_WOR_EVENT0_LSB,    0xC3},
  {CC120X_PKT_CFG2,          0x00},
  {CC120X_PKT_CFG0,          0x20},
  {CC120X_RFEND_CFG0,        0x09},
  {CC120X_PKT_LEN,           0x7D},
  {CC120X_IF_MIX_CFG,        0x1C},
  {CC120X_TOC_CFG,           0x03},
  {CC120X_MDMCFG2,           0x02},
  {CC120X_FREQ2,             0x56},
  {CC120X_FREQ1,             0xCC},
  {CC120X_FREQ0,             0xCC},
  {CC120X_IF_ADC1,           0xEE},
  {CC120X_IF_ADC0,           0x10},
  {CC120X_FS_DIG1,           0x07},
  {CC120X_FS_DIG0,           0xAF},
  {CC120X_FS_CAL1,           0x40},
  {CC120X_FS_CAL0,           0x0E},
  {CC120X_FS_DIVTWO,         0x03},
  {CC120X_FS_DSM0,           0x33},
  {CC120X_FS_DVC0,           0x17},
  {CC120X_FS_PFD,            0x00},
  {CC120X_FS_PRE,            0x6E},
  {CC120X_FS_REG_DIV_CML,    0x1C},
  {CC120X_FS_SPARE,          0xAC},
  {CC120X_FS_VCO0,           0xB5},
  {CC120X_IFAMP,             0x09},
  {CC120X_XOSC5,             0x0E},
  {CC120X_XOSC1,             0x03},
```

Example 5. RX Sniff Mode Settings (CC1200, 38.4 kbps, ETSI standard) (continued)

```
};
```

Figure 17 shows the current profile when running the CC1200 RX Sniff Mode code [12]. The average current consumption is measured to be 1.50 mA, very close to the estimated current consumption found using the 120x_RX_Sniff_Mode Excel sheet [10] (see Figure 16).

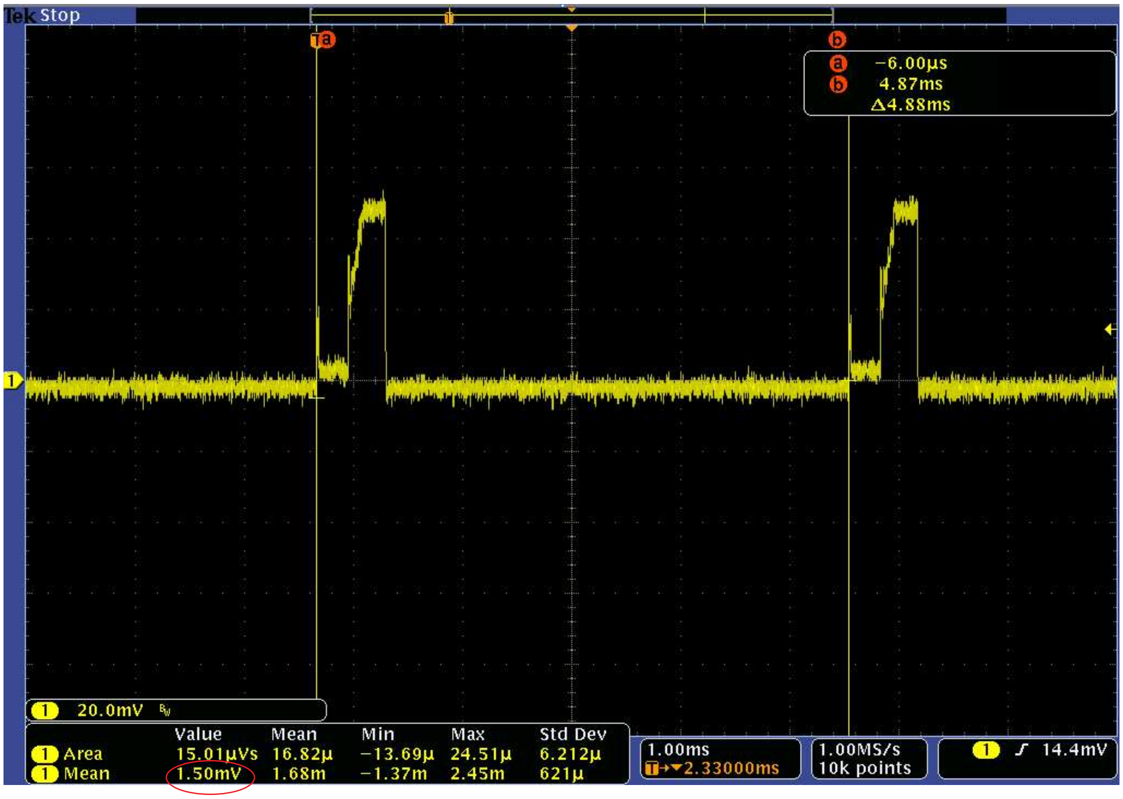


Figure 17. Average Current Consumption (CC1200, 38.4 kbps, ETSI standard, 24 bytes preamble)

5 Conclusion

This application report has shown which registers to configure when implementing the RX Sniff Mode and also provides two Excel sheets, which provide a good estimation of the average current consumption when using this mode. The code examples can be used as starting points when implementing the RX Sniff Mode in a real-life application.

From the examples one can see that the average current consumption decreases when the preamble length increases. For very long preamble sequences it is recommended to use SmartPreamble. This concept is described in details in [13].

6 References

1. *CC112X/CC1175 Low-Power High Performance Sub-1 GHz RF Transceivers/Transmitter User's Guide* (SWRU295)
2. *CC120x Low-Power High Performance Sub-1 GHz RF Transceivers User's Guide* (SWRU346)
3. *CC1120 High Performance RF Transceiver for Narrowband Systems Data Sheet* (SWRS112)
4. *CC1121 High Performance Low Power RF Transceiver Data Sheet* (SWRS111)
5. *CC1125 Ultra-High Performance RF Narrowband Transceiver Data Sheet* (SWRS120)
6. *CC1200 Low Power, High Performance RF Transceiver Data Sheet* (SWRS123)
7. *CC112x/CC1175 High Performance RF Transceiver/Transmitter for Narrowband Systems Errata Note* (SWRZ039)
8. *SmartRF Studio* (SWRC176)
9. *CC112x_RX_Sniff_Mode.xlsx* (SWRA428)
10. *CC120x_RX_Sniff_Mode.xlsx* (SWRA428)
11. *CC112x Software Examples* (SWRC253)
12. *CC120x Software Examples* (SWRC274)
13. *CC112x/CC120x RX Sniff Mode With Smart Preamble* (SWRA438)

Appendix A Revision History

This document has been revised from SWRA428 to SWRA428A because of the following technical change(s).

Table 6. SPNU428A Revisions

Location	Additions, Deletes, and Edits
Table 3	Added two registers (MDMCFG1 and SYNC_CFGx)
Section 3.2	Added Info on how to determine t_{Event0} when using PQT termination. Also added info on how the RC oscillator tolerance should be accounted for.
Section 3.7	Added info on t_{Event0} when PQT termination is enabled
Section 3.9	Added info saying that t_{Event0} is shorter when PQT termination is enabled compared to when using CS termination.
Figure 4	Added case C showing the current consumption for the radio when MCU is used to terminate RX (and CS termination is enabled on the radio).
Section 3.9.2	Added explanation of PQT response time and reference to user's guides.
Section 4.1	SmartRF Studio version changed from 1.12.0 to 1.13.0.
Section 4.1.1.1	New value for EVENT0 since the RC oscillator tolerance is accounted for in the equation.
Figure 5	Added `MDMCFG1` and `SYNC_CFG1`
Example 3	Changed value of `WOR_EVENT0_LSB`
Figure 9	Updated with new measurements
Figure 10	Updated due to new revision of excel sheet
Figure 11	Updated due to new revision of excel sheet
Section 4.1.2	Changes numbers in the text related to Figure 11
Figure 13	Updated with new measurements
Example 4	Changed value `FS_REG_DIV_CML`
Figure 15	Updated due to new revision of excel sheet
Figure 16	Updated due to new revision of excel sheet
Example 5	Changed value `FS_REG_DIV_CML`
Figure 17	Updated with new measurements
Section 5	Added reference to [13]

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