

AN1200.01
Application Note

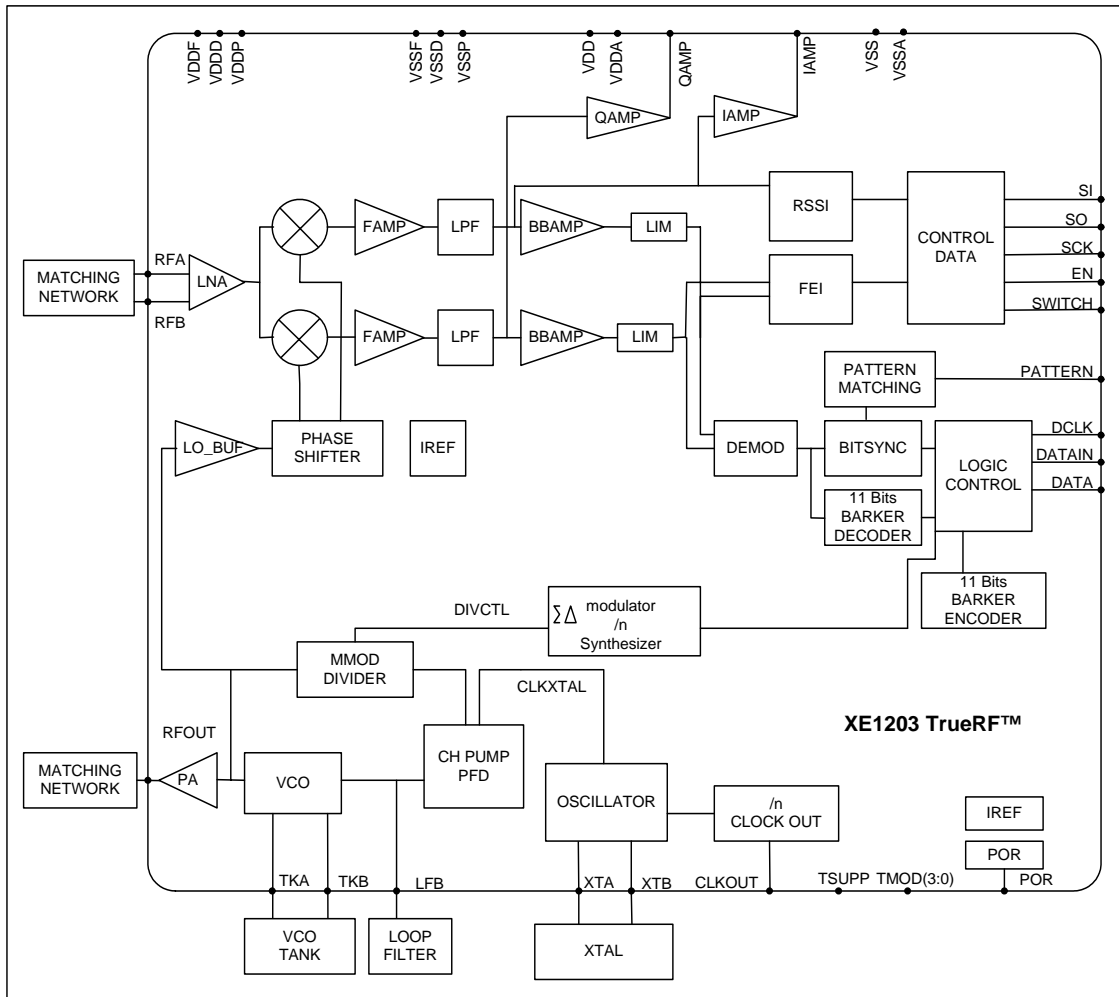
XE1200 External RSSI

Table of Contents

1	Introduction	3
2	RSSI algorithm	4
3	External RSSI usage	5
4	Hardware connections	7
5	Software implementation	7
5.1	XE1203Driver.c	8
5.2	Initialisation.c	9
5.3	Uart.c	9
5.4	Measure Rssi.c	10
5.5	Considerations for the MeasureRssi implementation	10
6	Performances and trade-off	13

1 INTRODUCTION

The purpose of this application note is to develop a high dynamic and good accuracy external RSSI for the XE1203F. This RSSI is made using I (IAMP) and Q (QAMP) analog signals. These signals are at the output of the low pass filter in the XE1203F Rx path:



The RSSI algorithm will evaluate the power on I or Q signal. It represents the received signal's power. It can be implemented on any of the XE1200 transceivers that have I and Q outputs.

In a first section, we will explain the algorithm, its hardware and software implementation and finally present its performances and trade-off. This algorithm is developed in C in order to be easily portable.

2 RSSI ALGORITHM

The RF received signal's power can be evaluated at the I and Q level. Knowing the gain in the Rx path and the power on I or Q, it is possible to determine the received signal's power.

The RSSI algorithm will evaluate the power on the I signal. This algorithm can be divided in two parts: First, the estimation of the DC voltage on I and then the estimation of the RSSI.

The algorithm will be the following:

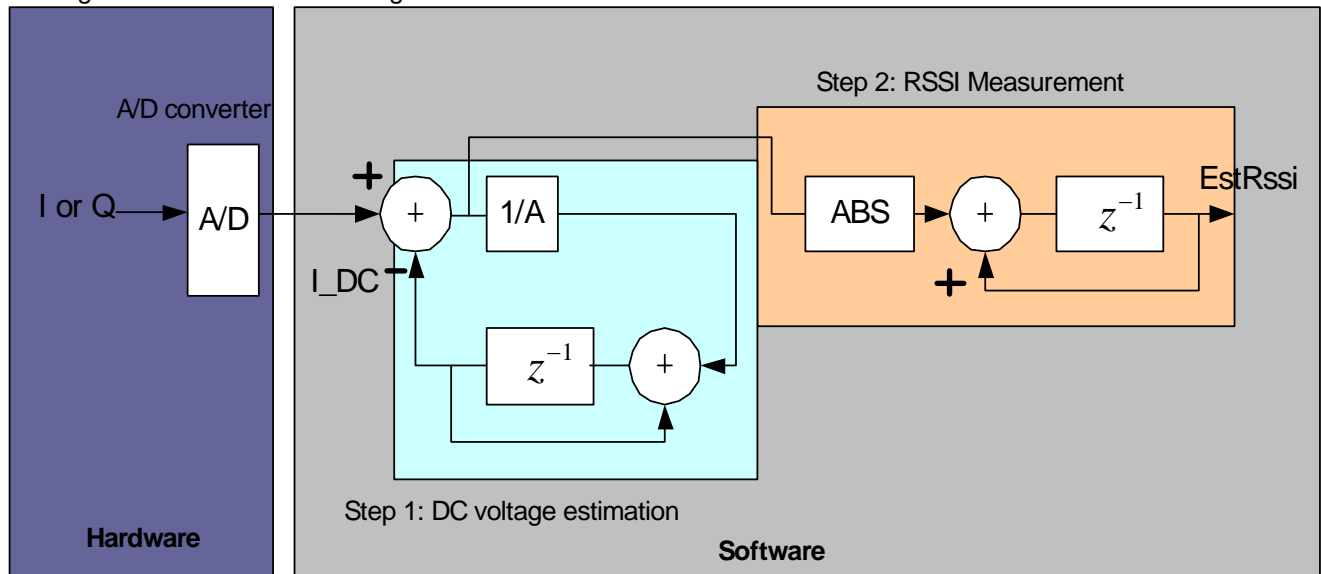


Figure 1.

EstRssi relates to the level of the signal at the antenna, so depending on the application; one can calculate the real power of the signal by applying the formula or works in relative.

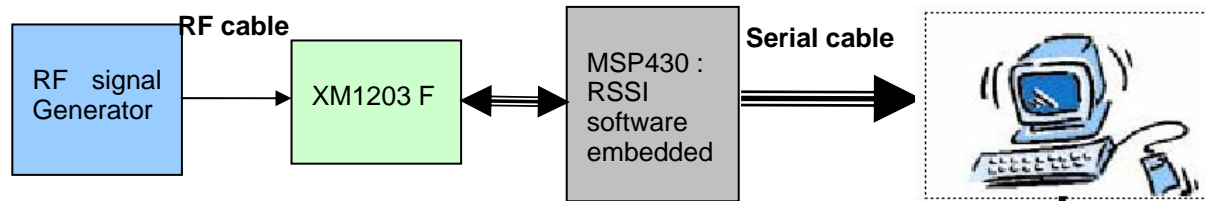
To calculate the power of the received signal in dBm, first it is necessary to calculate the power of EstRssi with a 50Ω load, to subtract the gain of the Rx path, to transform dB to dBm and then to take into account that the algorithm introduces a constant of 3.9. The formula is the following:

Formula1:

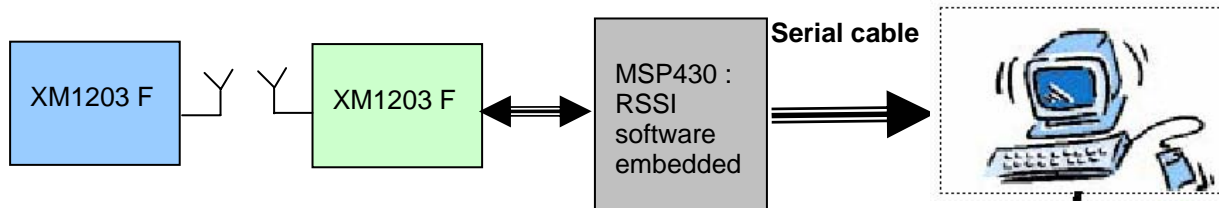
$$P_{RealDbm} = 10 * \log(P_{EstRssi}) - G + 30 + 3.9 = 10 * \log\left(\frac{EstRssi^2}{50}\right) - G + 30 + 3.9$$

3 EXTERNAL RSSI USAGE

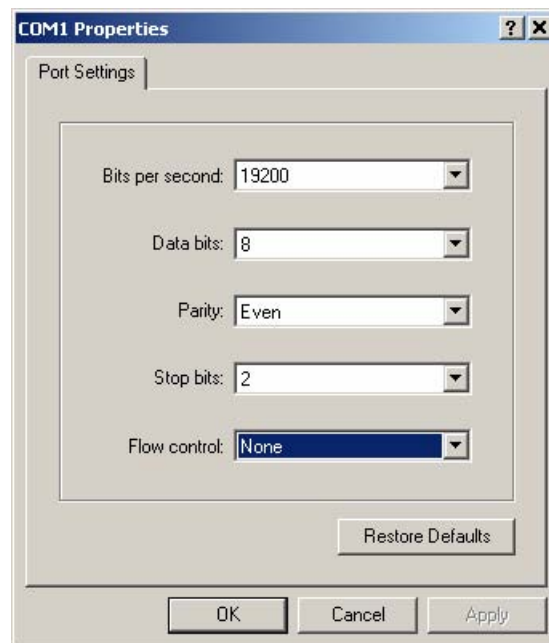
This application can be easily set up. It only requires an RF signal generator or a XM1203F to transmit a RF signal, a XM1203F to receive this signal, a MSP430, a serial cable and a PC to receive the data.



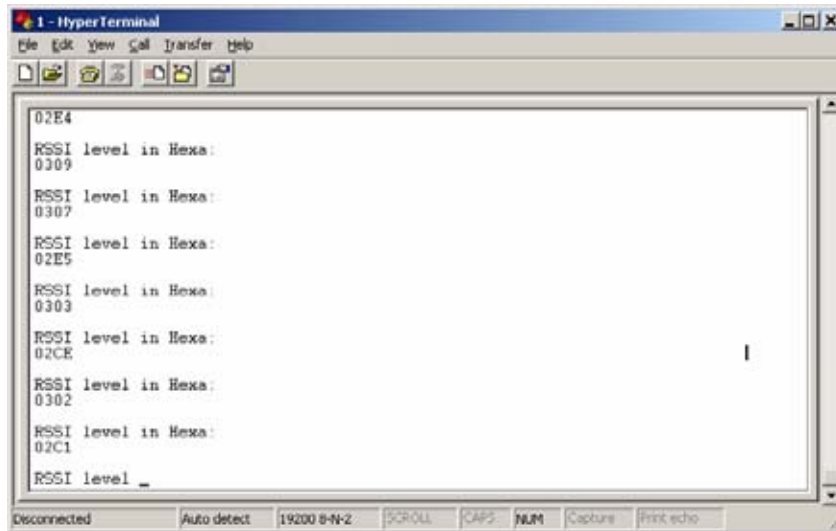
Or



The configuration of the UART is the following:



On the PC, the received data will be in the following format:



The return value is related to the 12 bit ADC. To get the received signal power in dBm, we should apply the formula 1 from section 2.

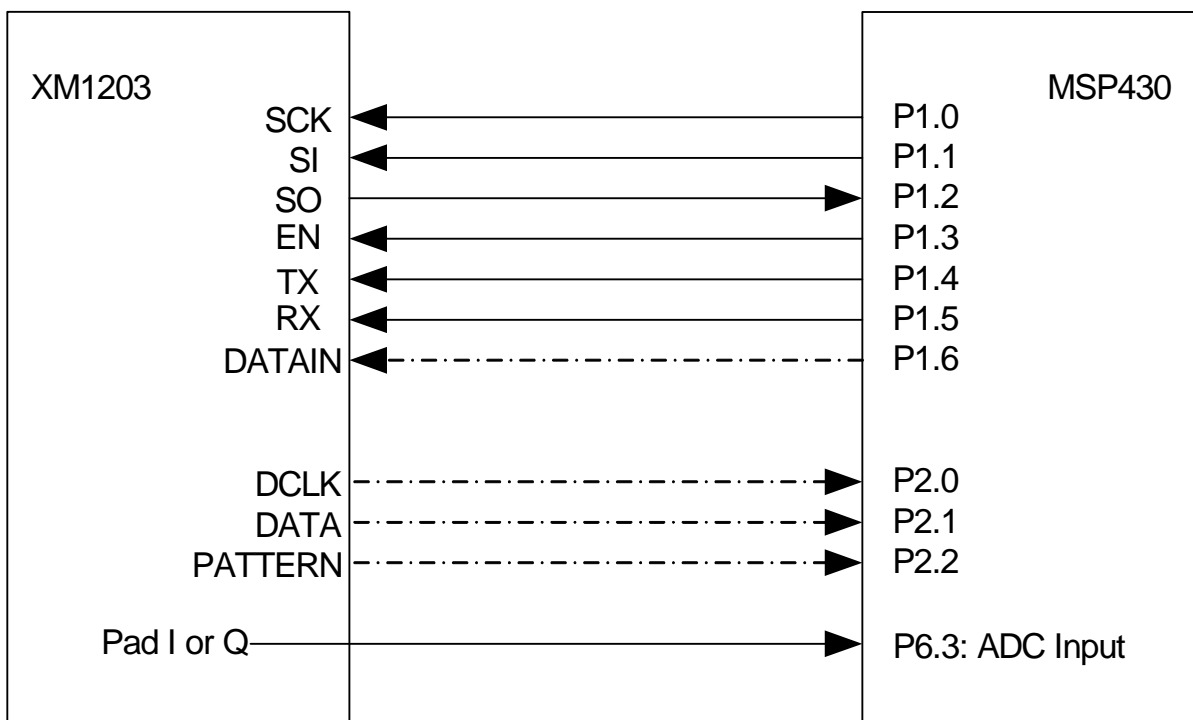
So for a return value of 0x0303, it corresponds to $x = 303 * 2.5 / 255 = 0.47$ V and if we apply the formula

$$P_{RealDbm} = 10 * \log\left(\frac{EstRssi^2}{50}\right) - G + 30 + 3.9, \text{ we find that the level is } -73.6 \text{ dBm for } G = 84 \text{ dB.}$$

4 HARDWARE CONNECTIONS

This algorithm is developed in C and has been implemented on a MSP430. It can be easily ported to another microcontroller that has an ADC.

Connections between the XM1203 and the MSP430:



—— Required for the RSSI application

- - - - Optional

5 SOFTWARE IMPLEMENTATION

For the software implementation, the main function calls sub functions for the initialization of the MSP430, the initialization of the XM1203F, to set the XM1203F into receive mode, to calculate the RSSI and to send the result to the UART.

The sub functions related to the XM1203F are located in the file XE1203Driver.c. The ones related to the initialization of the MSP430 are in the file Initialisation.c. Uart.c contains the sub functions to handle the UART and MeasureRssi.c to measure the RSSI.

5.1 XE1203DRIVER.C

The file XE1203Driver.c includes functions to drive the XE1203. They have the same prototypes as the ones developed for the API TN8000.18 "XE8000 driving XE1200 transceivers". The same conventions and global variables are used.

For the different data types, the same conventions are used but adapted to the MSP430 compiler:

Standard type name	Defined type name	Comment
unsigned char	_U8	8 bit quantity
signed char	_S8	8 bit signed quantity
unsigned short	_U16	16 bit quantity
short	_S16	16 bit signed quantity
long	_S32	32 bit signed quantity
unsigned long	_U32	32 bit quantity
double	_F32	32 bit float quantity

To configure the XE1203F (baud rate, frequency deviation, mode A or B), the variable RegistersCfg[] is used (see chap 5.4 "Initializing transceiver parameters" from TN8000.18 [2]).

Most of the functions in the XE1203Driver.c file of this application can be found in the XE1203Driver.c file of TN8000.18 [2]. They are here implemented on a MSP430 using the same syntax. Utility functions have been specially developed. For further information see chap. 6 "API Functions" of TN8000.18 [2].

Configuration functions:

InitRFChip : This routine initializes the RFChip registers, Using Pre Initialized variable
SetRFMode : Sets the XE1203F operating mode (Sleep, Receiver, Transmitter)
WriteRegister : Writes the register value at the given address on the XE1203F
ReadRegister : Reads the register value at the given address on the XE1203F

Communication functions:

SendRfFrame : Sends a RF frame
ReceiveRfFrame : Receives a RF frame
SendByte : Send a data of 8 bits to the transceiver LSB first
ReceiveByte : Receives a data of 8 bits from the transceiver LSB first

Utility functions:

Wait : This routine uses the timer A to create a delay using the ACLK clock
InvertByte : Inverts a byte. MSB -> LSB, LSB -> MSB

Functions developed specifically for the MSP430:

WaitSCK : This routine use a time out loop to have SCK lower than 1 MHz
TxInterruptOn : Initializes the timers and the interruptions related to the TX routines
TxInterruptOff : Disables the timers and the interruptions related to the TX routines
RxInterruptOn : Initializes the timers and the interrupt related to the RX routines
RxInterruptOff : Disables the timers and the interrupt related to the RX routines

5.2 INITIALISATION.C

The file Initalisation.c contains all the functions used to initialize the MSP430. The MSP430 will be clocked by an 8 MHz external quartz.

InitMicro : Initializes the MicroController peripherals, it calls the following sub functions:

InitPort1 : Initializes the Port 1 as an I/O and set the pins as outputs

InitPort2 : Initializes the Port 2 as an I/O and set the pins as inputs

InitPort3 : Initializes the Port 3 and set P3.4 for the UART

InitPort6 : Initializes the Port 6 and set P6.3 = A3 = Input used by the ADC

InitClockModule : Initializes the clock module. MCLK = LFXT1 = 8MHz and ACLK = LFXT1 = 8MHz, SMCLK=DCOCLK (Res=0, DCO=0 -> freq=74.1 KHz)

InitTimerA : Initializes the Timer A

InitTimerB : Initializes the Timer B

InitADC : Initializes the ADC. The internal reference voltage 2.5 V is used for Vr+ and Vr- = AVss. Continuous sampling and conversion mode is programmed.

ADC12CLK = ACLK = 8 MHz, tsample = 4* ADC12CLK, tconvert = 13* ADC12CLK

InitUART: Initialize the UART0 to 8 bits data, 1 stop bit, even parity, Baud Rate = 19200 bps

5.3 UART.C

This contains functions to transmit data to the UART.

Hex2Char: Converts an hexadecimal number

SendChar: Sends a character to the UART

SendString: Send a string on UART Tx using SendChar function

SendCRLF: Sends carriage return and line feed characters

TransmitRssiUart: It transmits the result of the RSSI to the UART

5.4 MEASURE RSSI.C

The file MeasureRssi.c contains the functions to measure the RSSI.

MeasureRssi: It first estimates the DC on I and calculates EstRssi

MeasureRssi function: is a function that is in the fileMeasureRssi.c

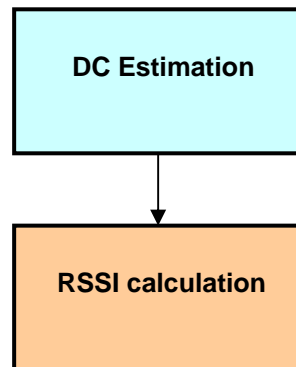
5.5 CONSIDERATIONS FOR THE MEASURERSSI IMPLEMENTATION

With the 12 bit MSP430 ADC, it takes 17 clock cycles to sample and convert a signal. So the maximum sampling frequency is 470 KHz. Between two samplings, it is necessary to process them. To estimate the DC or to calculate the RSSI, it requires more than 17 clock cycles. So the real sampling frequency is fixed by the time execution of the code which is 250 KHz in the two processes. The goal is to get the maximum sampling frequency in order to deal with higher frequency deviation. When a sample is ready, a flag is activated, the sample is processed and we wait for the next sample.

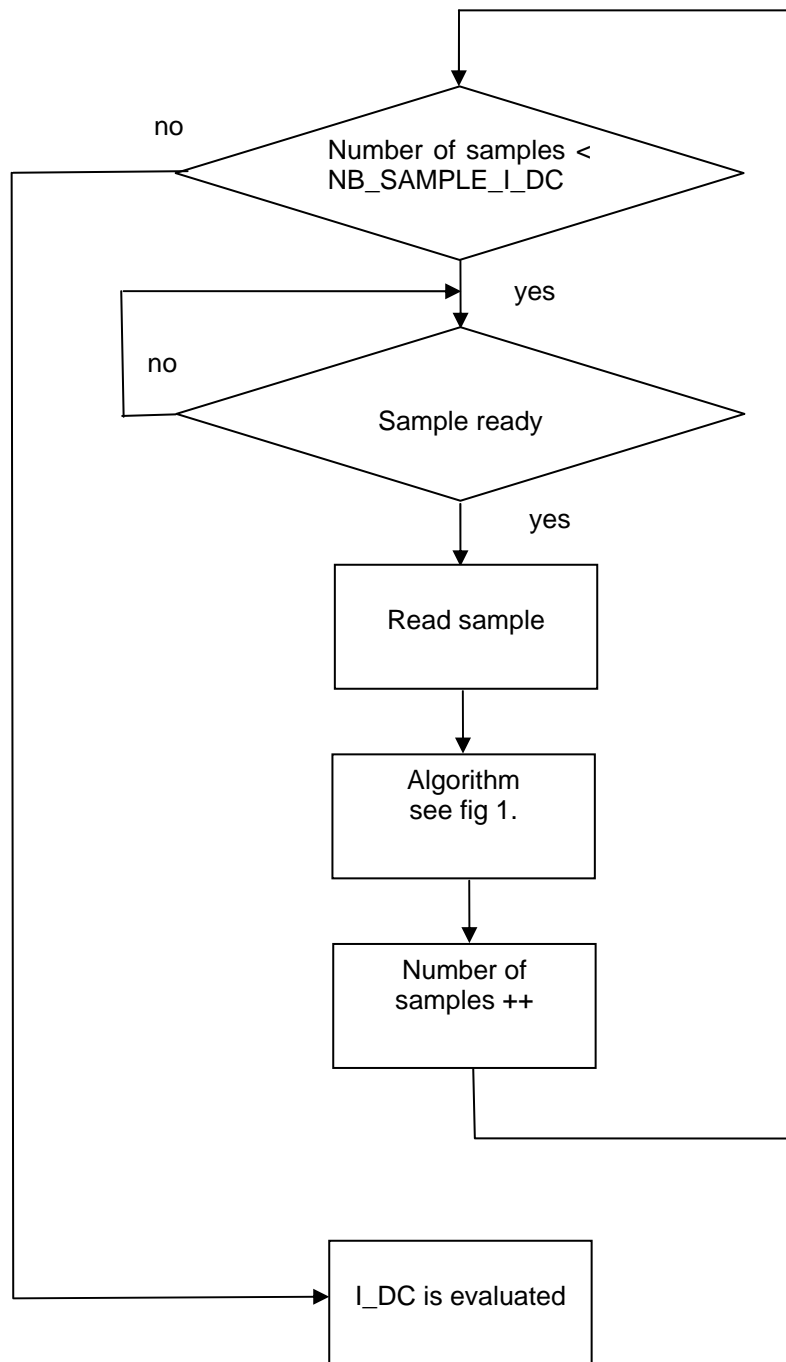
In order to fill the Shannon theory and avoid overlapping, the sampling frequency will be at least two times the frequency of I or Q. The maximum sampling frequency obtained is 250 KHz, so the maximum frequency deviation that can be used is 125 KHz. When changing the frequency deviation, the variable `FREQ_DEV` located in `MeasureRssi.h` should be updated and set to the new frequency deviation. The variable `FREQ_SAMPLE` is fixed to 250000.

This code is optimized for high frequency deviation; the goal has been to set as less instructions and cases as possible.

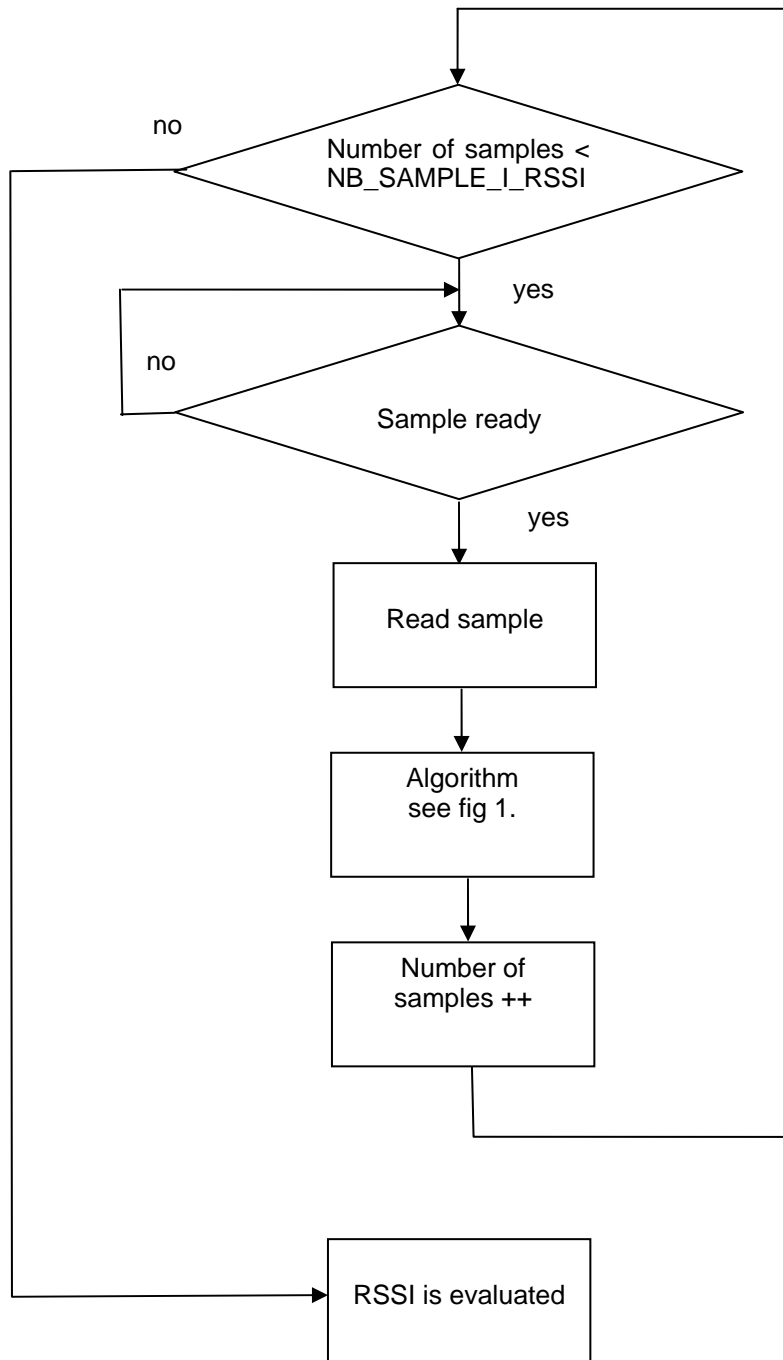
The measurement of the RSSI will be done in two steps as followed:



DC Estimation



EstRssi Calculation



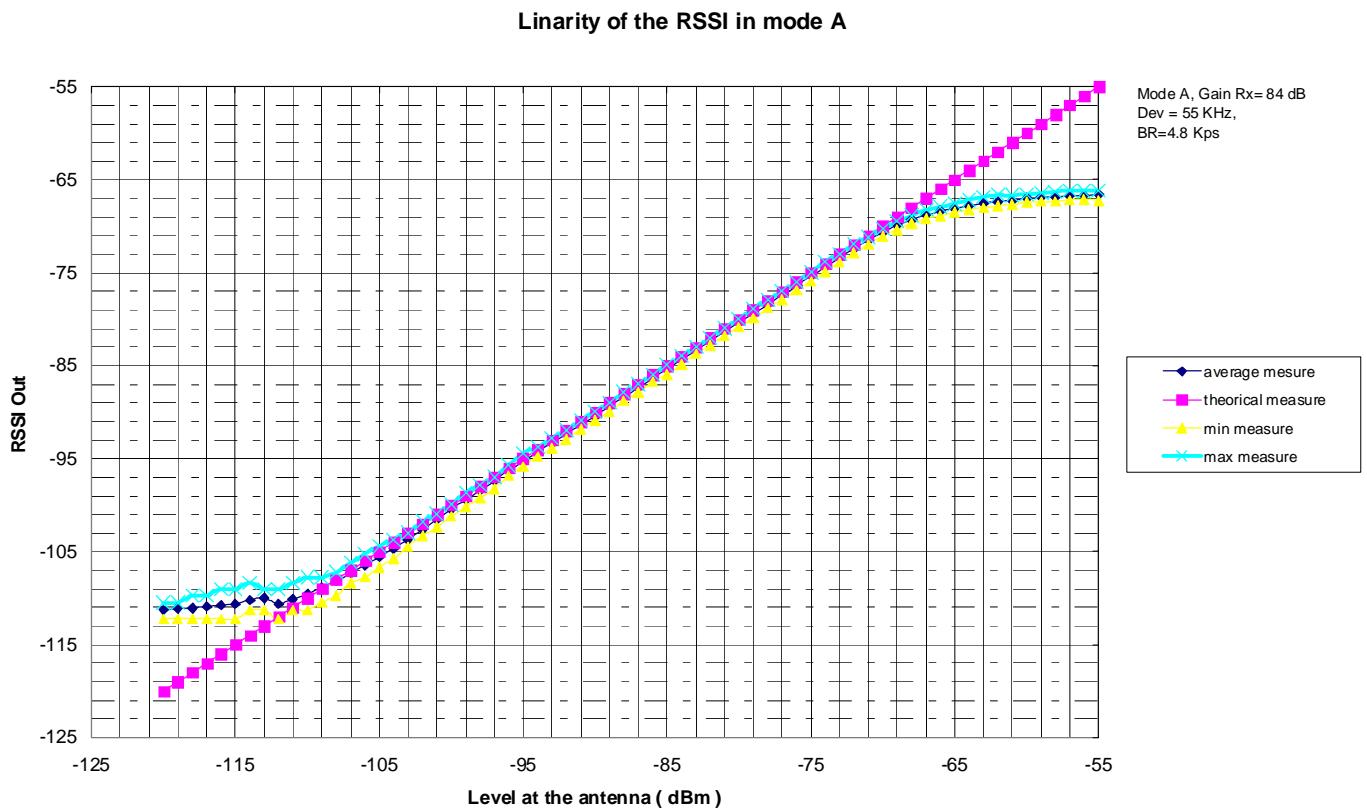
6 PERFORMANCES AND TRADE-OFF

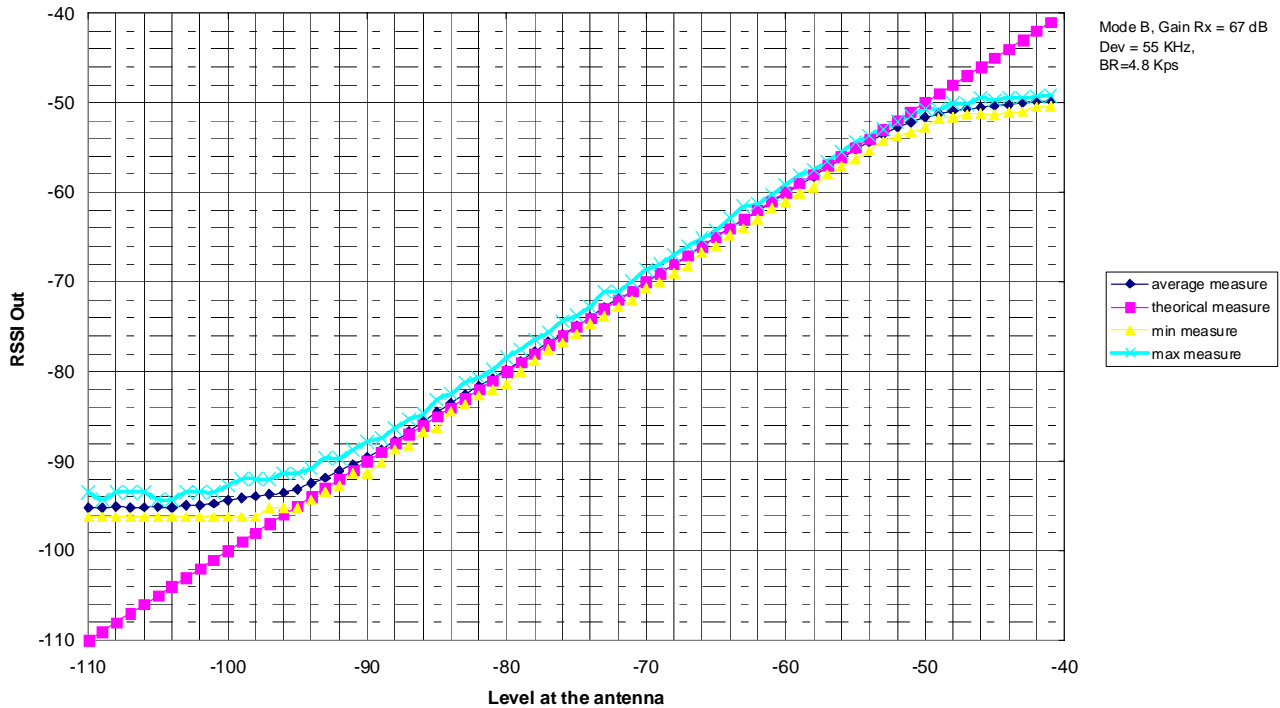
6.1 PERFORMANCES

In order to get a greater dynamic, we use mode A and mode B of the XE1203F. In mode A, the gain of the XE1203F is the greatest, it is about 84 dB. In mode B, the linearity of the receiver is increased and the gain is about 67 dB. The RSSI dynamic obtained is 60 dB, going from -111 dBm to -51 dBm. The standard deviation obtained is:

- 0.3 dB from -51 dBm to -70 dBm in mode B
- 0.2 dB from -70 dBm to -95 dBm in mode A
- 0.3 dB to 0.6 dB from -95 dBm to -111 dBm in mode A

Following is the linearity of the RSSI in mode A and mode B:



Linearity of the RSSI in mode B


6.2 TRADE-OFF

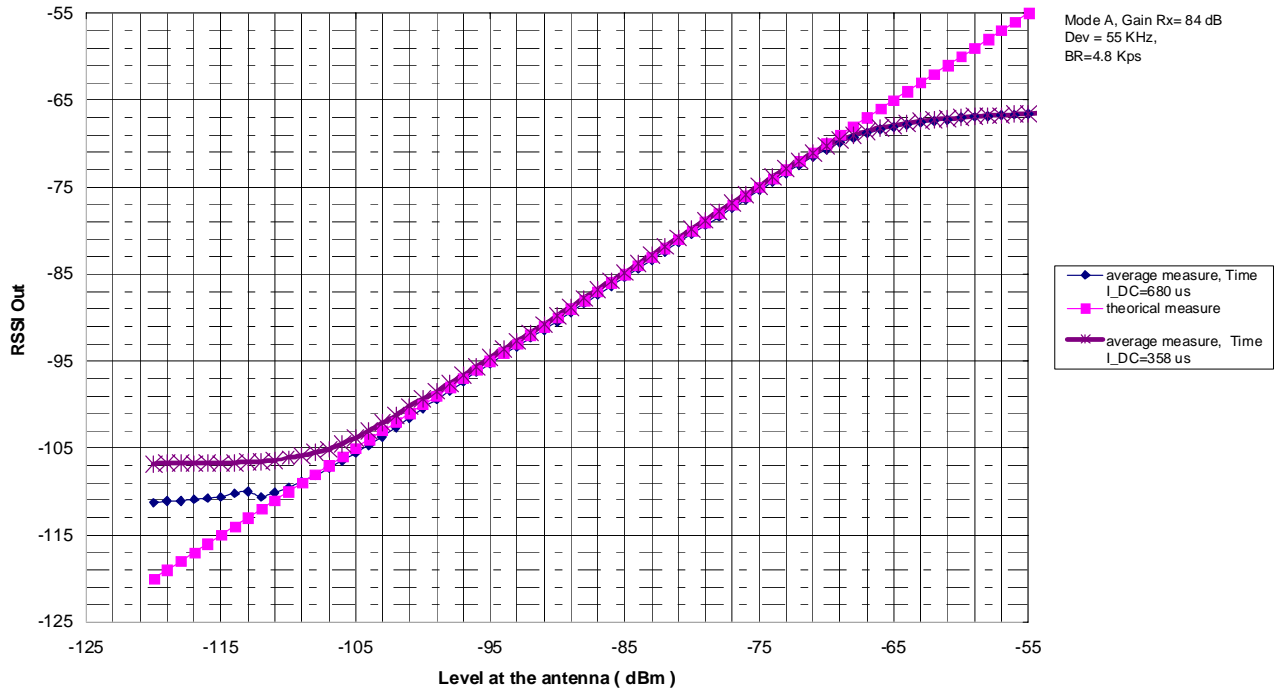
To get this dynamic, this accuracy and this linearity, it takes first 680 μ s to estimate the DC and 500 μ s to calculate EstRssi. This time can be optimized by changing some parameters in the DC estimation or in the EstRssi calculation.

I_DC Estimation:

The DC estimation is done on $NB_SAMPLE_I_DC = 25 * \frac{FREQ_SAMPLE}{FREQ_DEV}$ samples that is to say 25 periods of the frequency deviation.

For a frequency deviation of 55 KHz ($FREQ_DEV = 55000$), it takes 680 μ s to make the DC estimation but we can decrease this time by setting $NB_SAMPLE_I_DC = 10 * \frac{FREQ_SAMPLE}{FREQ_DEV}$. The DC estimation will be done on 10 periods of the frequency deviation.

These two examples have been implemented. In the second case, the DC is not estimated with the best accuracy. So, the lowest level of the RSSI is shifted and the dynamic decreased. The accuracy of the RSSI has remained the same.

Linearity of the RSSI versus the time to estimate I_DC


As a general rule, the time to estimate I_{DC} is:

$$Time_{I_{DC}} = 10^3 * \frac{NB_SAMPLE_I_DC}{200} + 140$$

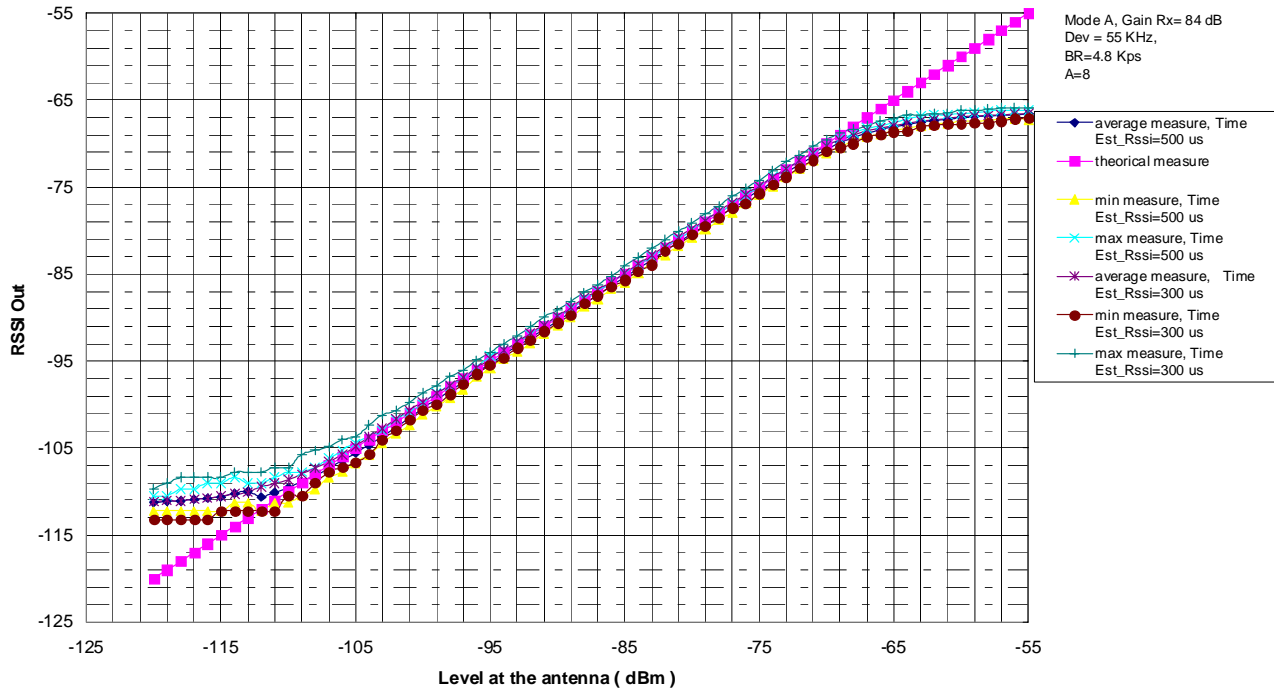
EstRssi calculation: Integration of $|I - I_{DC}|$

The EstRssi calculation is done on $NB_SAMPLE_RSSI = 25 * \frac{FREQ_SAMPLE}{FREQ_DEV}$ samples.

It takes 500 μs to make this calculation. As for the DC estimation, this time can be reduced by setting:

$$NB_SAMPLE_RSSI = 10 * \frac{FREQ_SAMPLE}{FREQ_DEV}$$

These two examples have also been implemented and we can see that the dynamic is the same but the accuracy has been degraded (standard deviation goes from 0.3 to 0.8 dB)

Linearity of the RSSI versus the time to integrate | I - I_DC |


As a general rule the time to calculate the EstRssi is:

$$Time_{EstRssi} = 10^3 * \frac{NB_SAMPLE_I_RSSI}{275} + 140$$

So depending on the application, we can decide either to improve the dynamic, the resolution or the time by changing the parameter NB_SAMPLE_I_DC or NB_SAMPLE_RSSI located in the file MeasureRssi.h.

So finally, we will only have to change the parameter FREQ_DEV which corresponds to the frequency deviation of the application and if necessary NB_SAMPLE_I_DC or NB_SAMPLE_RSSI in the file MeasureRssi.h.

REFERENCES

- [1] XE1203F datasheet
- [2] TN8000.18:XE8000 driving XE1200 transceivers standard API definitions
<http://www.semtech.com>

© Semtech 2005

All rights reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent or other industrial or intellectual property rights. Semtech. assumes no responsibility or liability whatsoever for any failure or unexpected operation resulting from misuse, neglect improper installation, repair or improper handling or unusual physical or electrical stress including, but not limited to, exposure to parameters beyond the specified maximum ratings or operation outside the specified range.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

Contact Information

Semtech Corporation
Wireless and Sensing Products Division
200 Flynn Road, Camarillo, CA 93012
Phone (805) 498-2111 Fax : (805) 498-3804