
RAW DATA MODE WITH EZRADIOPRO®

1. Introduction

This application note describes raw data mode configuration for the Si443x which allows for the reception of any packet structure without the need to follow the recommended packet structure in the data sheet. A typical recommended packet structure is a preamble (01010...) sequence followed by a synchronization word and payload data.

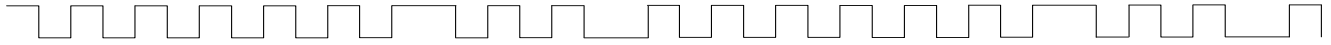
This application note covers raw mode for GFSK and OOK modulations with less than 10 kbps data rates. In raw mode, additional software algorithms (deglitching) are utilized on the MCU to achieve optimal performance. These algorithms are also described and presented. The results for various data rates and conditions are also presented and summarized for reference.

To receive a copy of the raw mode reference software contact customer support. If the application parameters are outside the stated scope, contact the application support for more details.

TX Data Clock



1. Short Preamble + Sync + Data



2. Manchester Encoded Preamble + Sync + Data



3. Long 1s + Sync + Manchester Data



4. Long 0 + Manchester Encoded (Sync + Data)



5. Long 1 + Sync + Random Data

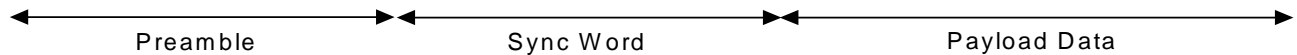


Figure 1. Non-Typical Data Formats

Figure 1 demonstrates a few non-standard data patterns or protocols. Typically, the first few bytes in these protocols are preamble or synchronization bytes and do not follow a conventional (010101...) preamble pattern. For these cases, the chip should be configured into raw data mode.

In raw data mode, the receiver does not search for a fixed preamble pattern. The receiver will track the incoming data as seen at the receiver input. Without deglitching the raw data mode, sensitivity is approximately 10 dB worse than the standard but with deglitching running on the host MCU the sensitivity is only a few dB's less than the standard mode.

2. Raw Data Mode Overview

In this mode, the receiver is configured to 5 to 10 times the actual data rate, therefore the data is oversampled. This allows for faster acquisition as well as tracking of non-standard preamble or synchronization patterns. Since the received data is sampled by a 5 to 10x higher sampling clock, glitches may be present as the signal level approaches the sensitivity limit. The glitches can easily be filtered by a microcontroller algorithm. In certain cases, the raw data mode can be optimized for a glitch-free operation at a fixed data rate.

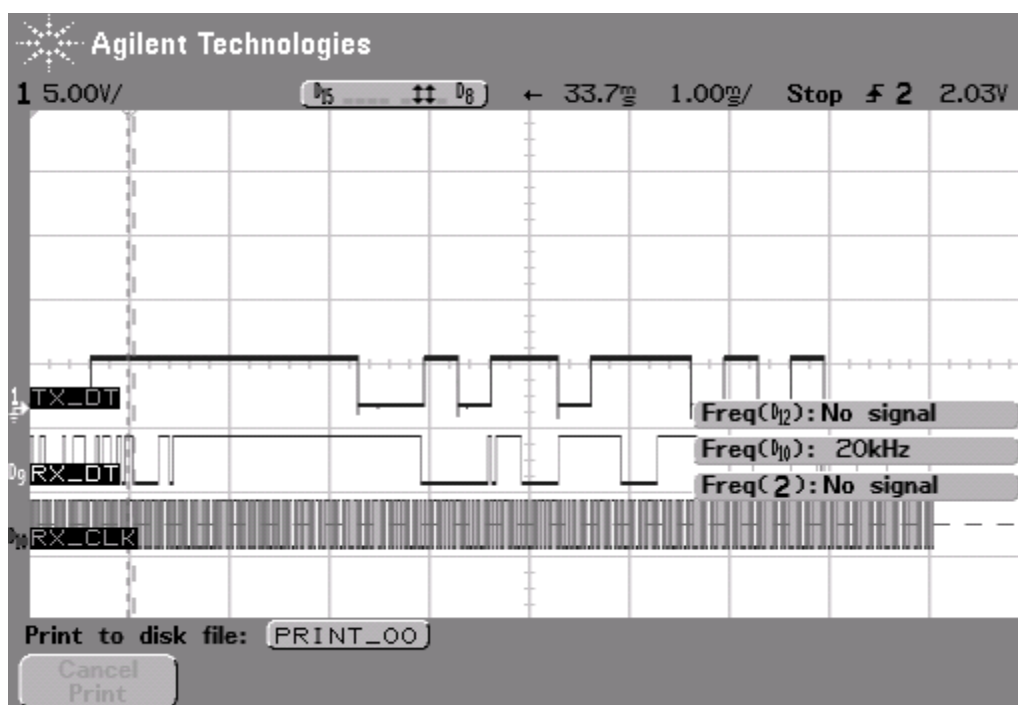


Figure 2. Raw Data Mode with Higher Sampling Clock

As shown in Figure 2, a glitch can occur because of the higher bit clock rate. The glitches will be the width of the oversampled clock. A deglitching algorithm can easily be run on the microcontroller to remove the glitches. Reference code for deglitching is provided in "3.2. Deglitching Algorithm" on page 6.

2.1. Raw Data Mode Modem Settings

Different modem settings are needed to cover different ranges of applications parameters in terms of modulation (GFSK, OOK), data rate (<10K DR), deviation (<75K) and crystal tolerance (<50 ppm). One of the settings below should be selected based on the application parameters.

Table 1. Raw Data Mode Settings Summary

Setting No.	Modulation	Setting Name	Comment
1	GFSK/FSK	Setting 1	DR<4.8 kbps, narrow deviation
2	GFSK/FSK	Setting 2	DR>4.8 kbps, wide deviation
3	GFSK/FSK	Setting 3	DR<4.8 kbps, wide deviation
4	GFSK/FSK	Setting 4	DR>4.8 kbps, narrow deviation
5	GFSK/FSK	Setting 5	ALL DR, wide deviation
6	OOK	Setting 6	DR<1 kbps
7	OOK	Setting 7	DR<4 kbps
8	OOK	Setting 8	DR<10 kbps

Detailed measurement results to evaluate tradeoffs between the settings are summarized in Table 3 and Table 4 and presented in more detail in the appendix.

2.2. Raw Data Mode Packet Handler Settings

In raw data mode, the Si443x is set in a mode such that the standard packet mode is deactivated. To achieve this, the following register settings are required:

- Set preamble detection threshold to maximum.
- Enable all Header comparisons.
- Enable multi packet mode.
- Enable CRC check.

These settings are done to avoid a possible condition where the receiver identifies some noise pattern as a preamble and receives an incorrect packet as a standard packet.

In the reference code the `user_setup.h` file will define the correct modem parameters based on the application parameters for the raw mode initialization.

3. Raw Data Mode Implementation

Figure 3 shows the raw data mode algorithm implementation details. The RX Data and RX_Bit_Clk are provided to the microcontroller which runs the raw data mode algorithm.

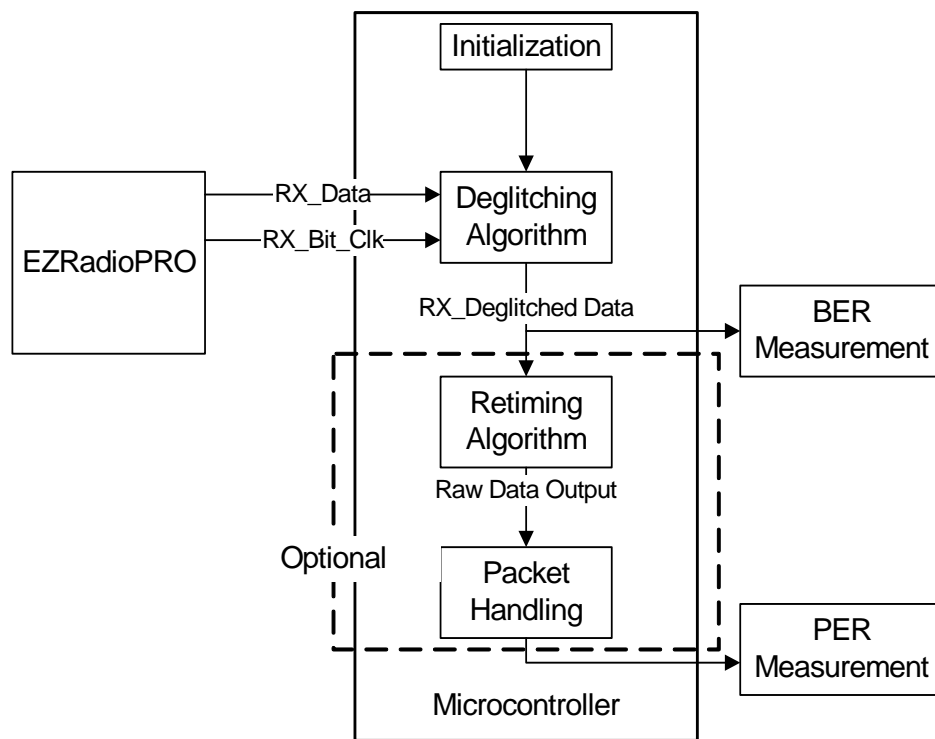


Figure 3. Raw Data Mode Implementation

As shown in Figure 3, the algorithm is divided in 4 parts:

- Initialization—Sets the receiver in a raw data mode configuration.
- Deglitching Algorithm—Removes glitches on the RX Data.
- Retiming Algorithm—Estimates the no. of raw data bits and extracts them in a buffer.
- Packet Handling—Detects Synchronization Word, processes and validates the Payload and data is available in payload data buffer.

The output of the deglitching algorithm is used for BER measurements. The RX_Clk for bit sampling is generated using an external clock source. A simple packet error rate calculation algorithm is implemented using the packet handler algorithm on the microcontroller. This is used for PER measurements.

Because of the modular implementation of the software algorithm, the user can easily select/deselect the required blocks. For example, if the deglitching block is the only relevant block for the application, the user can disable the retiming and packet handling blocks.

3.1. Initialization

This section of the code configures the receiver for raw data mode.

```
void Radio_Init ( unsigned int setting )
{
    unsigned char ItStatus1, ItStatus2;

    //read interrupt status registers to clear the interrupt flags and release NIRQ pin
    ItStatus1 = SpiReadRegister(0x03);
    ItStatus2 = SpiReadRegister(0x04);

    //SW reset
    SpiWriteRegister(0x07, 0x80);

    //wait for chip ready interrupt from the radio (while the nIRQ pin is high)
    while ( NIRQ == 1);

    //read interrupt status registers to clear the interrupt flags and release NIRQ pin
    ItStatus1 = SpiReadRegister(0x03);
    ItStatus2 = SpiReadRegister(0x04);

    //set the physical parameters

    //set Crystal Oscillator Load Capacitance register
    SpiWriteRegister(0x09, CRYSTAL_OSC_LOAD_CAP);

    //set the center frequency to 870.8 MHz
    SpiWriteRegister(0x75, FREQ_BAND_SELECT);
    SpiWriteRegister(0x76, NOM_CARRIER_FREQ_1);
    SpiWriteRegister(0x77, NOM_CARRIER_FREQ_0);

    // packet handler settings
    SpiWriteRegister(0x35, 0xF8);
    SpiWriteRegister(0x71, 0x63);

    // set to avoid false packet detection
    SpiWriteRegister(0x30, 0x84);
    SpiWriteRegister(0x32, 0x0F);
    SpiWriteRegister(0x33, 0x77);
    SpiWriteRegister(0x08, 0x10);

#ifdef Setting2
```

```
// Modem Parameters
SpiWriteRegister(0x56, 0x40);
SpiWriteRegister(0x1C, 0xAE); // 153.3 KHz BW
SpiWriteRegister(0x1D, 0x68); .
SpiWriteRegister(0x1E, 0x0A);
SpiWriteRegister(0x20, 0x3E);
SpiWriteRegister(0x21, 0x02);
SpiWriteRegister(0x22, 0x0C);
SpiWriteRegister(0x23, 0x49);
SpiWriteRegister(0x24, 0x04);
SpiWriteRegister(0x25, 0x60);
SpiWriteRegister(0x2A, 0xB0);
SpiWriteRegister(0x71, 0x63);
#endif

// packet handler settings
SpiWriteRegister(0x35, 0xF8);
SpiWriteRegister(0x71, 0x63);

// set GPIOs
SpiWriteRegister(0x0b, 0xD4);
SpiWriteRegister(0x0c, 0xCF);
SpiWriteRegister(0x51, 0x36);

// Set Non default registers
SpiWriteRegister(0x69, 0x60);

// Turn Receiver ON
SpiWriteRegister(0x07, 0x04);
```

3.2. Deglitching Algorithm

The main function of the deglitching algorithm is to remove any glitches that occur in the RX Data due to the higher data rate bit-clock. The glitches are one or two clock cycles wide and the example algorithm will filter out these narrow glitches.

There are multiple ways to implement the deglitching algorithm. Given below are two possible options that were implemented and tested in the software. Any one of the algorithms can be chosen for actual implementation depending on the application requirements. The first algorithm is a general solution. Using the second algorithm slightly better sensitivity can be achieved and the code space is slightly lower but the window size parameter will need to be optimized for each application.

3.2.1. Deglitching Algorithm 1: Adaptive Filtering

As shown in Figure 2 on page 3, an oversampled bit clock from the RF chip is connected to microcontroller input. This clock is used to sample the RX_Data and remove any glitches that are present on the received data. The RX_DATA is the actual received data from the chip.

The glitch filtering is done in 2 phases. Output of the first filter phase is given to the next phase.

In each phase, five samples of data are stored and based on logical analyses, glitches that are only one or two bit-clock wide are filtered.

A decision matrix is given in front of the code that is used to decide the current algorithm output.

```

if ( RX_DATA == Sample_2 )
{
    if ( RX_DATA == Sample_3 )
    {
        ValidData_1 = Sample_3;
    }
    else
    {
        if ( RX_DATA == Sample_4 )
        {

```

Sample No.	5	4	3	2	1
Sample Value	x	x	a	a	a
ValidData1 o/p			a		
where a = 0/1, b = ~a, x = don't care					

Sample No.	5	4	3	2	1
Sample Value	a	a	b	a	a
ValidData1 o/p			a		
where a = 0/1, b = ~a, x = don't care					

```

        if ( RX_DATA == Sample_5 )
        {
            ValidData_1 = Sample_5;
            Sample_3 = Sample_5;
        }

```

Sample No.	5	4	3	2	1
Sample Value	b	a	b	a	a
ValidData1 o/p			a		
where a = 0/1, b = ~a, x = don't care					

```

    else
    {
        ValidData_1 = Sample_4;
        Sample_3 = Sample_4;
    }
}
else
{

```

Sample No.	5	4	3	2	1
Sample Value	a	b	b	a	a
ValidData1 o/p			a		
where a = 0/1, b = ~a, x = don't care					

```

        if ( RX_DATA == Sample_5 )
        {
            ValidData_1 = Sample_5;
            Sample_3 = Sample_5;
            Sample_4 = Sample_5;
        }
    else
    {
        ValidData_1 = Sample_5;
    }
}

```

```

    }
}
else
{

```

Sample No.	5	4	3	2	1
Sample Value	b	b	b	a	a
ValidData1 o/p			b		
where a = 0/1, b = ~a, x = don't care					

```

if ( RX_DATA == Sample_3 )
{
    if ( RX_DATA == Sample_4 )
    {
        ValidData_1 = Sample_4;
        Sample_2 = Sample_4;
    }
    else
    {
        ValidData_1 = Sample_5;
        Sample_3 = Sample_5;
    }
}
else
{
    if ( RX_DATA == Sample_4 )
    {

```

Sample No.	5	4	3	2	1
Sample Value	x	a	a	b	a
ValidData1 o/p			a		
where a = 0/1, b = ~a, x = don't care					

Sample No.	5	4	3	2	1
Sample Value	a/b	b		b	a
ValidData1 o/p			a/b		
where a = 0/1, b = ~a, x = don't care					

Sample No.	5	4	3	2	1
Sample Value	a	a	b	b	a
ValidData1 o/p			b		
where a = 0/1, b = ~a, x = don't care					

```

        if ( RX_DATA == Sample_5 )
        {
            ValidData_1 = Sample_3;
        }
        else
        {
            ValidData_1 = Sample_5;
            Sample_4 = Sample_3;
        }
    }
}
else
{
    ValidData_1 = Sample_3;
}
}
}

```

Sample No.	5	4	3	2	1
Sample Value	b	a	b	b	a
ValidData1 o/p			b		
where a = 0/1, b = ~a, x = don't care					

else

```

{
    ValidData_1 = Sample_3;
}
}
}

```

Sample No.	5	4	3	2	1
Sample Value	x	b	b	b	a
ValidData1 o/p			b		
where a = 0/1, b = ~a, x = don't care					

```

// Register the Data Samples
Sample_7 = Sample_6; Sample_6 = Sample_5; Sample_5 = Sample_4; Sample_4 = Sample_3;
Sample_3 = Sample_2; Sample_2 = RX_DATA;

```


The filtering is done twice since at an rf input closer to sensitivity, there can be a lot of glitches. This needs additional filtering. The code for the second filter is same as the one shown above.

```
DelayedSample_4 = DelayedSample_3; DelayedSample_3 = DelayedSample_2;
DelayedSample_2 = DelayedSample_1; DelayedSample_1 = ValidData_1;
```

The output of the deglitching algorithm is a signal “databit.” This is given to the Retiming algorithm

```
databit = ValidData_2;
```

Deglitched data output available at LED3. The inverted deglitched data output is available at LED4.

```
LED3 = databit; LED4 = ~databit;
```

3.2.2. Deglitching Algorithm 2: Median Filtering

This algorithm is typically used to filter spikes. In this algorithm, the elements are sorted and the element in the middle of the window is selected as the output. Since there are only two logical states (0 or 1) it is enough to count the number of 1s in a window. If this number is greater than the half size of the window, the output will be logic 1, otherwise a logic 0.

For example:

01011 → sorting → 00111 → the median (element in the middle) is 1, so the output is 1. The number of the 1s is three, which is greater than the half size of the window (2.5), so the output is 1.

00100 → sorting → 00001 → the median is 0, the output is 0. The number of 1s is 1, which is less than 2.5 → output is 0.

In the algorithm, a running sum of 1's is maintained. The algorithm is updated every RX_Clk edge. The bitmask variable is used to set a mask for the MSB bit of the shift register. For example, with windowsize=5, the bitmask is 10000 (binary). Then it is checked if the MSB is 1 or 0. When the data is shifted, the MSB will leave the window. If the MSB was 1, then the number of 1s (which is stored in the count variable) is decreased.

```
if (shiftreg & bitmask)
    count--;
```

Here we shift the shift register:

```
shiftreg <= 1;
```

If the incoming data is 1, the LSB of the shift register is set to 1. Also the count is increased by 1. If the RX_DATA is 0, shift register is unaltered i.e. 0 enters the shift register.

```
if (RX_DATA)
{
    shiftreg++;
    count++;
}
```

The algorithm also allows for a selection of window sizes from 1 to 15. Since a 16-bit variable is used for data storage, windows masks same as the size of the window are used to get rid of the bits that are shifted out of the window. For example, if the window size is 5, the window mask is 11111 (binary).

```
shiftreg &= windowmask;
```

If the number of 1s (count) is greater than the half size of the window (bitlimit = windowsize / 2) the output will be 1, otherwise 0.

```
if ( count > bitlimit )
    databit = 1;
else
    databit = 0;
```

The received data is made available on the LEDs (LED3 – RX_Data, LED4 – inverted RX_Data).

```
LED3 = databit;
LED4 = ~(databit);
```

3.2.3. Deglitching Algorithm Output Interface

The deglitched bit value is added to the data bit buffer:

```
DataBitBuffer[fBufferSelect][pointer] = databit;  
Increase the pointer of the data bit buffer  
pointer++;
```

In the software implementation, a double buffer is used for the DataBitBuffer. The deglitching algorithm fills the first row of the DataBitBuffer array. When the first row is full, a flag is set to indicate to the retiming algorithm that there is data available to process. While the retiming program processes the first row of the data, the deglitching part fills the second row. When it is done, the algorithms switch again. Now, the first row will be filled by the deglitching algorithm and the second processed by the retiming algorithm.

fBitBufferFull flag shows that one row is filled by the deglitching, fBufferSelect flag shows which row is filled.

```
if ( pointer > 31 )  
{  
    pointer = 0;  
    fLastBufferSelect = fBufferSelect;  
    if(fBufferSelect == 0)  
    {  
        fBufferSelect = 1;  
    }  
    else  
    {  
        fBufferSelect = 0;  
    }  
    fBitBufferFull = 1;  
}
```

3.3. Retiming Algorithm

The retiming algorithm will estimate the no. of raw data bits between two data transitions. It will also account for any jitter that is on the RX Data. DataBitBuffer, which is the output of the deglitching algorithm, is an input to the retiming part of the algorithm.

In the software implementation, the data rate is specified in user_setup.h and the RX_Clk is specified in the radio.c file. The no. of samples per bit is calculated by dividing the data rate by the RX_Clk. During the measurement, the no. of RX_Clk cycles is counted between each data transition. Based on the counter value the number of raw bits received (e.g., two consecutive ones or three consecutive zeros etc.) are determined.

```

/*+++++
+
+ FUNCTION NAME:void  BitsRetiming(U8  *  InputBuffer,  U8  NmbrOfInputBits,  U16
SamplePerBit, U8 * OutputBuffer, U8 * NmbrOfOutputBits)
+
+ DESCRIPTION:      Retime the deglitched bits
+
+ INPUT:            InputBuffer - input buffer, the deglitched bits
+                   NmbrOfInputBits - the number of the deglitched bit
+                   SamplePerBit- sample per bits
+                   OutputBuffer - output buffer, the retimed bits
+                   NmbrOfOutputBits - the number of retimed bits
+
+ RETURN:
+
+ NOTES:
+
+ +++++*/
void  BitsRetiming(U8  *InputBuffer,  U8  NmbrOfInputBits,  U16  SamplePerBit,  U8
*OutputBuffer, U8 *NmbrOfOutputBits)
{
    U8 i,j;
    U8 temp1, temp2;
    U16 DBitLimit;

    i = 0;
    DBitLimit = ( (U16)SamplePerBit << 3 );
    *NmbrOfOutputBits = 0;

    while ( i < NmbrOfInputBits )                //go through all of the input bits
    {
        if ( ( *( InputBuffer + i ) == BitValue ) && ( DeglitchedBitCnt <
DBitLimit ) ) //if the next bit is the same
        {
            DeglitchedBitCnt++;                //increase the deglitched bit counter
        }

        else                                //if the next bit isn't the same
        {

            DeglitchedBitCnt <= 2;
            temp1 = (U8)( DeglitchedBitCnt / SamplePerBit );
            //estimate the number of bits
            temp2 = (U8)( DeglitchedBitCnt % SamplePerBit );

            for ( j = 0; j < temp1; j++ )
            {

```

```
        *OutputBuffer++ = BitValue; //save the retimed bits
        *NmbrOfOutputBits += 1;
    }

    if ( temp2 > ( SamplePerBit >> 1 ) )
    {

        *OutputBuffer++ = BitValue;//save the retimed bit
        *NmbrOfOutputBits += 1;
    }

    BitValue = *( InputBuffer + i );//save the new bit value
    DeglitchedBitCnt = 1;

}

i++;

}

}
```

The output of the retiming algorithm is stored in OutputBuffer. This is the actual received data.

3.4. Packet Handling

The output of the retiming algorithm is an input parameter to the packet handling algorithm. The packet handler performs typical packet handling functionalities such as synchronization word detection and payload validation. The synchronization word, packet length etc. are specified in user_setup.h file. For the packet error rate evaluation an expected packet data is also specified in the user_setup.h file. Once the synchronization word is detected Sync_Detect signal (LED1) is set to 1. The Sync_Detect will remain hi till the end of the packet (based on the Payload_Length). If the packet was received correctly, Pk_Valid signal (LED2) will be set to 1. The Pk_Valid signal will again go low after a fixed delay.

```

/*+++++
+
+ FUNCTION NAME:void PacketHandler(U8 * InputBuffer, U8 NmbrOfInputBits)
+
+ DESCRIPTION:      This function process the retimed data, finds the sync word and
gives back the payload
+
+ INPUT:              InputBuffer - the retimed bits
+                      NmbrOfInputBits - the retimed bit number
+
+ RETURN:
+
+ NOTES:              the process starts always with the MSB bit
+
+++++*/

void PacketHandler(U8 * InputBuffer, U8 NmbrOfInputBits)
{
    U8 i;
    static U8 TempByte, BitCnt, ByteCnt, synctemp;
    static U32 SyncWordTempByte;
    U32 SyncWordCheckByte;

    i = 0;
    SyncWordCheckByte = SYNCWORD_MASK;
    SyncWordCheckByte = SYNCWORD;

    while(i< NmbrOfInputBits)
    {

        //check for synch word
        if(fSyncWordFound == 0)//if the start of the sync word hasn't found yet
        {

            //save bits into bytes
            SyncWordTempByte <= 1;
            if(*(InputBuffer+i) == 1)
            {
                SyncWordTempByte |= 0x00000001;
            }
        }
    }
}

```

```
        }
        else
        {
            SyncWordTempByte &= SYNCWORD_MASK;
        }

        if(SyncWordTempByte == SyncWordCheckByte)
        { //synchron word found

            fSyncWordFound = 1;
            BitCnt = 0;
            ByteCnt = 0;

        }
    }
    else
    {

        //save bits into bytes
        TempByte <= 1;
        if(*(InputBuffer+i) == 1)
        {
            TempByte |= 0x01;
        }
        else
        {
            TempByte &= 0xFE;
        }

        if( ++BitCnt == 8 )
        { //save the next byte into the buffer
            PayloadBuffer[ByteCnt++] = TempByte;
            BitCnt = 0;
            //check whether the number of specified bytes already
received or not

            if(ByteCnt == PAYLOAD_BYTE_LENGTH)
            { //packet is received

                fPacketReceived = 1;
                fSyncWordFound = 0;

            }
        }

        i++;
    }
}
1
```

3.5. Microcontroller Resource Utilization

The software algorithm is implemented using Silicon Labs C8051F930 microcontroller. The system clock speed is set to 24.5 MHz. The algorithm uses 1 external interrupt. The RX_Clk from the chip is connected to the interrupt pin of the microcontroller.

The RAM (data + xdata) & Flash (code) memory utilization is summarized in Table 2.

Table 2. Microcontroller Memory Utilization

Sr. No	Mode	Data	Xdata	Code
1	Deglitch 1 Only	11	6	884
2	Deglitch 2 Only	9.1	9	627
3	Deglitch 1 with packet handler	11.5	154	2546
4	Deglitch 2 with packet handler	9.6	157	2266

4. Steps for Setting Raw Data Mode

4.1. Select appropriate setting

Appropriate settings should be selected based on modulation, data rate, deviation, and crystal ppm. Refer to Table 3 on page 18 to select the appropriate settings for a given application.

4.2. Update user_setup.h file

All the user selectable parameters are included in user_setup.h file. This file needs to be updated according to application settings.

```
=====
// Crystal Oscillator Load Capacitance register: set the appropriate value to tune the
30MHz crystal
#define CRYSTAL_OSC_LOAD_CAP0xD5// address: 0x09

// Carrier frequency setting registers: set the desires receiving frequency here
// Frequency Band Select register
#define FREQ_BAND_SELECT0x73// address: 0x75
// Nominal Carrier Frequency 1 register
#define NOM_CARRIER_FREQ_10x86// address: 0x76
// Nominal Carrier Frequency 0 register
#define NOM_CARRIER_FREQ_00xF7// address: 0x77

// uncomment the appropriate modem setting; see the appropriate setting according to
table 3,4
#define SETTING_1
:
//#define SETTING_8

// define here the deglitch method to be used: 1 or 2
#define DEGLITCH_METHOD2

// if deglitch method #2 was chosen then define the size of the glitch filtering window
#define WINDOWSIZE15// size of the glitch filtering window

// if you do not want to use the packet handler then comment out the next line
#define PACKET_HANDLER

// if the packet handler is enabled specify here the details of the transmitted data

// datarate
#define DATARATE(1000)// datarate of the received data

// synchronization word properties
// Please make sure that the size of SYNCWORD = SYNCWORD_BIT_LENGTH
#define SYNCWORD0xAA2DD4//sync word pattern max 4 bytes
#define SYNCWORD_BIT_LENGTH24//length of the sync word in bits, max 32

// payload properties
```



```
// Please make sure that the size of PAYLOAD_CONTENT = PAYLOAD_BYTE_LENGTH
#define PAYLOAD_CONTENT { 0x01, 0x23, 0x45, 0x67, 0x89 }// the content of the payload
#define PAYLOAD_BYTE_LENGTH5// length of the payload in bytes
```

5. Raw Data Mode Results

5.1. Measurement Setup

Raw data mode measurements are done based on two criteria:

1. 0.1% BER: In this measurement, a PN9 signal is generated using a RF generator. The output of the generator is connected to the receiver input. Sensitivity is measured by using the BER option of the signal generator.
2. 20% Packet Error Rate: In this measurement, a fix packet is programmed in the RF generator. The output of the generator is connected to the receiver input. Sensitivity is measured by calculating the percentage of packets received by the receiver.

5.2. FSK/GFSK Results Summary

Table 3. Raw Data Mode GFSK Setting Selection Summary

Data Rate	Dev.	Setting # (Table 1)	Raw Mode Sensitivity		Deglitch2	Setting # (Table 1)	Raw Mode Sensitivity		Deglitch2
			Deglitch1	Deglitch2	Window		Deglitch1	Deglitch2	Window
kpbs	kHz	<20ppm Crystal	dBm	dBm	Size	20–50 ppm Crystal	dBm	dBm	Size
1	2	1	–102.5	–107.8	15	3	–89	–91	13
	5	1	–113.5	–116.8	15	3	–105.7	–109.7	15
	10	3	–110.5	–113	12	3	–110.5	–113	12
	30	3	–110	–110.5	15	3	–110	–110.5	15
	45	5	–103	–104.2	15	5	–102	–104.2	15
	75	5	–102.6	–105.1	15	5	–102.6	–105.1	15
2.4	2	1	–110.7	–112.6	9	3	–104	–105.7	7
	5	1	–115.7	–116.8	9	3	–109.2	–111	7
	10	3	–111.1	–109.6	12	3	–111.1	–109.6	12
	30	3	–111.8	–110.8	12	3	–111.8	–110.8	12
	45	5	–102.3	–105.3	15	5	–102.3	–105.3	15
	75	5	–102.7	–105.2	15	5	–102.7	–105.2	15
4.8	2	1	–104.5	–107.7	5	3	–99	–102.5	3
	5	1	–112.6	–114.1	5	3	–107	–108	3
	10	3	–108.5	–110	3	3	–108.5	–110	3
	30	2	–108.6	–109.2	11	2	–108.6	–109.2	11
	45	2	–106.1	–107.6	10	2	–106.1	–107.6	10
	75	5	–102.5	–105.1	15	5	–102.5	–105.1	15
9.6	2	4	–80	–87.7	5	4	–80	–87.7	5
	5	4	–98.2	–101.7	5	4	–98.2	–101.7	5
	10	4	–107	–107	5	4	–107	–107	5
	30	2	–106	–108.3	5	2	–106	–108.3	5
	45	2	–106.1	–105	7	2	–106.1	–105	7
	75	5	–102.3	–104.1	11	5	–102.3	–104.1	11
10	2	4	–80	–84.5	5	4	–80	–84.5	5
	5	4	–98.6	–102	5	4	–98.6	–102	5
	10	4	–104.6	–107	5	4	–104.6	–107	5
	30	2	–105.5	–107.7	5	4	–105.5	–107.7	5
	45	2	–105.8	–105.7	7	4	–105.8	–105.7	7
	75	5	–102.5	–104	9	5	–102.5	–104	9

5.2.1. OOK Results Summary

Table 4. Raw Data Mode OOK Setting Selection Summary

Data Rate	Setting No	Raw Mode Sensitivity		Deglitch2
		Deglitch1	Deglitch2	Window
kbps	<50 ppm Crystal	dBm	dBm	Size
1	6	-109	-109	12
2.4	7	-102	-103	14
4	7	-103	-104	8
4.8	8	-106	-107	10
9.6	8	-104	-105	5
10	8	-104	-105	4

5.2.2. AFC & BW Results Summary

Table 5. Raw Data Mode AFC and Filter BW Summary

Setting	Modulation	Setting Name	Filter BW			Approx. AFC Tolerance
No.			Min	Max	(with current settings)	(with current settings)
			kHz	kHz	kHz	(±)kHz
1	GFSK/FSK	Setting 1	9.5	26	26	14
2	GFSK/FSK	Setting 2	57	153	153	150
3	GFSK/FSK	Setting 3	38	102	102	50
4	GFSK/FSK	Setting 4	57	153	153	90
5	GFSK/FSK	Setting 5	225	621	621	70
6	OOK	Setting 6	75	204	204	100
7	OOK	Setting 7	225	621	621	200
8	OOK	Setting 8	75	204	204	100

6. Standard Mode and Raw Data Mode Comparison

6.1. Sensitivity

Table 6. Standard Data Mode vs. Raw Data Mode Sensitivity Summary

Data		Sensitivity	Sensitivity	Sensitivity
Rate	Deviation	Standard Mode	Deglintch1	No Deglintch
kHz	kHz	dBm	dBm	dBm
1	2	-116	-107.8	-80
	5	-116	-116.8	-105.4
	10	-113	-113	-102.9
	30	-116	-112	-107.8
	45	-110	-104.2	-95.6
	75	-111	-105.1	-98.6
2.4	2	-114	-112.6	-100.2
	5	-114	-116.8	-110.3
	10	-111	-111.1	-107.5
	30	-114	-111.8	-108.2
	45	-107	-105.3	-98.9
	75	-109	-105.2	-99.4
4.8	2	-112	-107.7	-102.2
	5	-115	-114.1	-111
	10	-109	-108.2	-107.7
	30	-110	-109.2	-99.5
	45	-111	-107.6	-100
	75	-107	-105.1	-99.5
9.6	2	-98	-87.7	-80
	5	-105	-101.7	-93.1
	10	-108	-107	-107.5
	30	-109	-108.3	-94
	45	-110	-106.1	-102.7
	75	-105	-104.1	-99
10	2	-98	-84.5	-80
	5	-106	-102	-95.1
	10	-109	-107	-97.3
	30	-109	-107.7	-99.5
	45	-109	-105.8	-102
	75	-105	-104	-99.3

As shown in Table 6, deglitch algorithm implementation improved the sensitivity of the receiver by ~0–10 dB as compared to the direct RX_Data output of the receiver. With the deglitch algorithm, the raw data mode typically shows ~0–6 dB degradation in sensitivity as compared to the sensitivity obtained in standard data mode with similar bandwidth.

1.1.1 Blocking

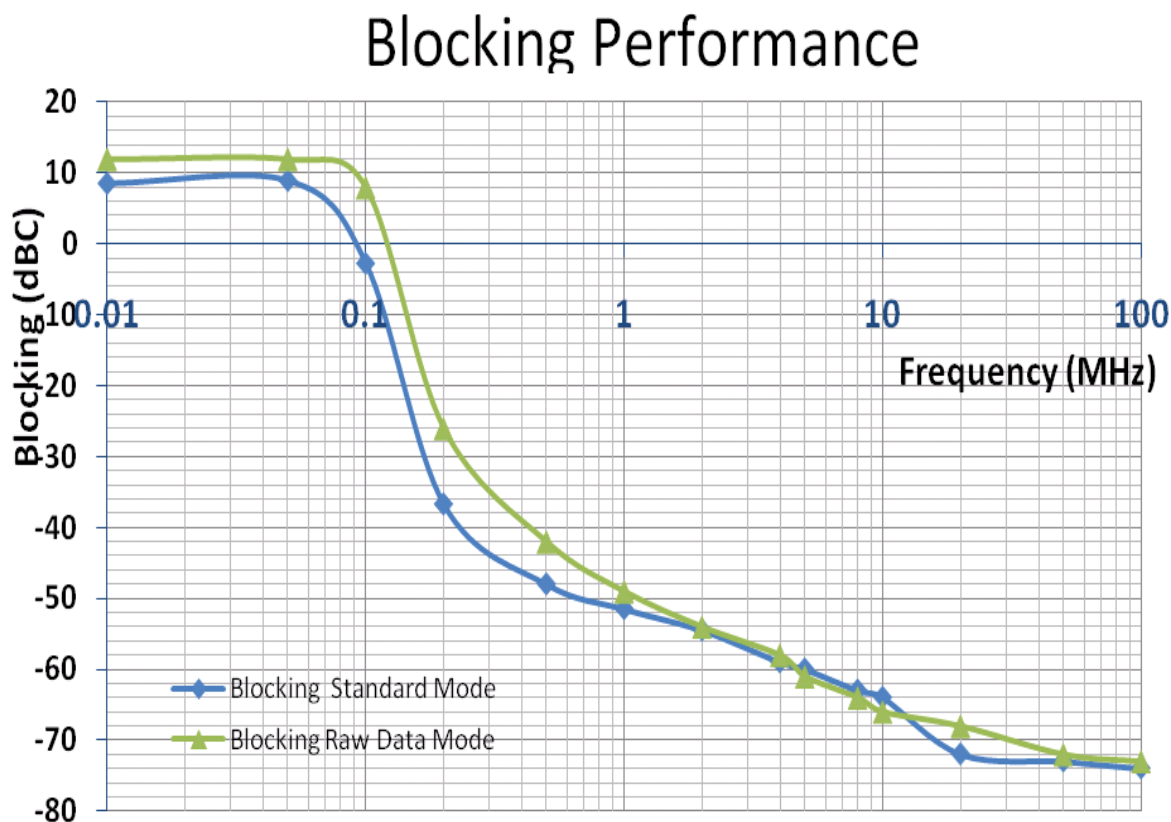


Figure 4. Standard Data Mode vs. Raw Data Mode Blocking Performance

As shown in Figure 4, the raw data mode typically show ~0–10 dB lower blocking performance. This is because in raw data mode, all in-band signals are considered as desired signals. In the standard data mode, the desired signal has a fixed packet format. (i.e., it starts with “01010...” pattern). This allows more optimization to be done in standard packet mode.

6.1.1. Acquisition Time

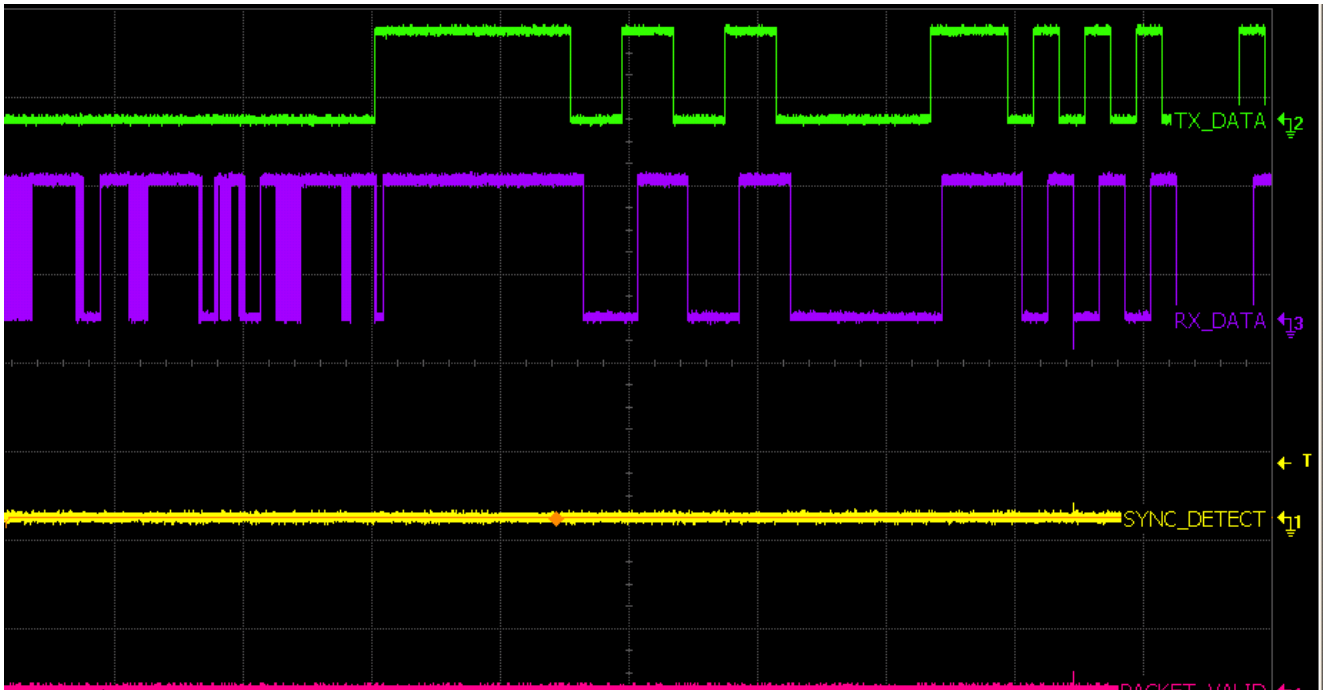


Figure 5. Raw Data Mode Provides Very Fast Acquisition (within a few bits)

As shown in Figure 5, the top trace is a transmitted packet. A noise is observed on the RX_Data before the actual packet arrives. Thus, the raw data mode provides a very fast acquisition of the received data. The received data is tracked usually within 1 to 2 bytes. Please note that there is some group delay because of the group delay within the Si443x as well as group delay due to microcontroller algorithm.

6.1.2. Packet Handler Usage

Following are the limitations when using a packet handler in raw data mode as compared to typical data modes supported by the chip.

- The receiver does not use internal packet identification such as preamble detection, sync detection etc. Hence, these signals as well as the corresponding interrupts will not be available.
- Since the receiver outputs an oversampled data, internal packet handler cannot be used. Hence, the data cannot be stored in the internal FIFO of the chip. Please note that in order to use the internal FIFO it is mandatory to detect a preamble and a correct synchronization word.
- CRC algorithm cannot be used for packet validation since the internal packet handler is not used. Customers may implement their own CRC algorithms in the software.

7. Conclusion

- EZRadioPRO can receive all different data patterns ranging from Manchester encoded to completely random (PN9) data patterns. There is no specific requirement of a particular packet structure.
- Data rates up to 10 kbps are supported by the settings provided in this document.
- There is typically ~0 to 6 dB degradation in sensitivity when using raw data mode.
- Raw data mode supports very fast acquisition of the received signal.
- In raw data mode the on chip packet handler, CRC algorithm, preamble detection, synchronization word detection, FIFOs, interrupts etc. cannot be used. However, the software algorithm is provided to detect the synchronization word and process the payload.

The following method should be followed for optimum reception:

1. Select the appropriate settings described in Tables 3 and 4.
2. Update User_Settings.h file according to the application parameters.
3. Compile and download the source code. (Compiler info: Keil C51 Compiler V8.08, Keil A51 Macro Assembler V8.00d, Keil BL51 Banked Linker/Locater V6.05)
4. Measure the performance.

Modem Settings

GFSK Settings

```
///GFSK Modem Settings
#ifdef Setting1
// Modem Parameters
SpiWriteRegister(0x56, 0x40);
SpiWriteRegister(0x1C, 0x3E); // 25.7 KHz BW
SpiWriteRegister(0x1D, 0x58);
SpiWriteRegister(0x1E, 0x0A);
SpiWriteRegister(0x20, 0x0D);
SpiWriteRegister(0x21, 0x09);
SpiWriteRegister(0x22, 0xD4);
SpiWriteRegister(0x23, 0x95);
SpiWriteRegister(0x24, 0x27);
SpiWriteRegister(0x25, 0x54);
SpiWriteRegister(0x2A, 0x11);
SpiWriteRegister(0x71, 0x63);
#endif
#ifdef Setting2
// Modem Parameters
SpiWriteRegister(0x56, 0x40);
SpiWriteRegister(0x1C, 0xAE); // 153.3 KHz BW
SpiWriteRegister(0x1D, 0x68); .
SpiWriteRegister(0x1E, 0x0A);
SpiWriteRegister(0x20, 0x3E);
SpiWriteRegister(0x21, 0x02);
SpiWriteRegister(0x22, 0x0C);
SpiWriteRegister(0x23, 0x49);
SpiWriteRegister(0x24, 0x04);
SpiWriteRegister(0x25, 0x60);
SpiWriteRegister(0x2A, 0xB0);
SpiWriteRegister(0x71, 0x63);
#endif
#ifdef Setting3
// Modem Parameters
SpiWriteRegister(0x56, 0x40);
SpiWriteRegister(0x1C, 0x1E); // 102.2 KHz BW
SpiWriteRegister(0x1D, 0x58);
SpiWriteRegister(0x1E, 0x1A);
SpiWriteRegister(0x1F, 0x03);
SpiWriteRegister(0x20, 0x68);
SpiWriteRegister(0x21, 0x01);
SpiWriteRegister(0x22, 0x3A);
SpiWriteRegister(0x23, 0x92);
SpiWriteRegister(0x24, 0x00);
```



```
SpiWriteRegister(0x25, 0xCB);
SpiWriteRegister(0x2A, 0x61);
SpiWriteRegister(0x71, 0x63);
#endif
#ifdef Setting4
// Modem Parameters
SpiWriteRegister(0x56, 0x40);
SpiWriteRegister(0x1C, 0xAE); // 153.3 KHz BW
SpiWriteRegister(0x1D, 0x58);
SpiWriteRegister(0x1E, 0x1A);
SpiWriteRegister(0x20, 0x3E);
SpiWriteRegister(0x21, 0x02);
SpiWriteRegister(0x22, 0x0C);
SpiWriteRegister(0x23, 0x49);
SpiWriteRegister(0x24, 0x04);
SpiWriteRegister(0x25, 0x60);
SpiWriteRegister(0x2A, 0x61);
SpiWriteRegister(0x71, 0x63);
#endif
#ifdef Setting5
// Modem Parameters
SpiWriteRegister(0x56, 0x40);
SpiWriteRegister(0x1C, 0x8E); // 620.7 KHz BW
SpiWriteRegister(0x1D, 0x68);
SpiWriteRegister(0x1E, 0x0A);
SpiWriteRegister(0x20, 0x68);
SpiWriteRegister(0x21, 0x01);
SpiWriteRegister(0x22, 0x3A);
SpiWriteRegister(0x23, 0x92);
SpiWriteRegister(0x24, 0x04);
SpiWriteRegister(0x25, 0xB9);
SpiWriteRegister(0x2A, 0xB0);
SpiWriteRegister(0x71, 0x63);
#endif
```

AN463

OOK Settings

```
//// OOK Modem Settings
#ifdef Setting6
// Modem Parameters
SpiWriteRegister(0x1F, 0x00);
SpiWriteRegister(0x1C, 0x4E); // 204.3 KHz BW
SpiWriteRegister(0x20, 0x7D);
SpiWriteRegister(0x21, 0xE3);
SpiWriteRegister(0x22, 0x86);
SpiWriteRegister(0x23, 0x25);
SpiWriteRegister(0x24, 0x01);
SpiWriteRegister(0x25, 0x08);
SpiWriteRegister(0x2C, 0x18);
SpiWriteRegister(0x2D, 0xD0);
SpiWriteRegister(0x2E, 0x2E);
SpiWriteRegister(0x71, 0x69);
SpiWriteRegister(0x70, 0x0E);
#endif

#ifdef Setting7
// Modem Parameters
SpiWriteRegister(0x1F, 0x00);
SpiWriteRegister(0x1C, 0xC7); // 412.1 KHz BW
SpiWriteRegister(0x20, 0x7D);
SpiWriteRegister(0x21, 0x63);
SpiWriteRegister(0x22, 0x86);
SpiWriteRegister(0x23, 0x25);
SpiWriteRegister(0x24, 0x01);
SpiWriteRegister(0x25, 0x08);
SpiWriteRegister(0x2C, 0x18);
SpiWriteRegister(0x2D, 0xD0);
SpiWriteRegister(0x2E, 0x2E);
SpiWriteRegister(0x71, 0x69);
SpiWriteRegister(0x70, 0x0E);
#endif

#ifdef Setting8
SpiWriteRegister(0x1F, 0x00);
SpiWriteRegister(0x1C, 0x2E); // 204.3 KHz BW
SpiWriteRegister(0x20, 0x7D);
SpiWriteRegister(0x21, 0x63);
SpiWriteRegister(0x22, 0x86);
SpiWriteRegister(0x23, 0x25);
SpiWriteRegister(0x24, 0x01);
SpiWriteRegister(0x25, 0x08);
SpiWriteRegister(0x2C, 0x18);
SpiWriteRegister(0x2D, 0xD0);
SpiWriteRegister(0x2E, 0x2E);
SpiWriteRegister(0x71, 0x69);
    SpiWriteRegister(0x70, 0x0E);
#endif
```

GFSK Measurement Results

Setting 1 Results

Test Setup: 4432-T-B1 B 868 split antenna test card.

RF Signal: GFSK, 2K4 data rate, 5K deviation

Measurement Criterion: BER

Deglitch Method: 1

Receiver BW: 25.7 kHz

RF Sensitivity

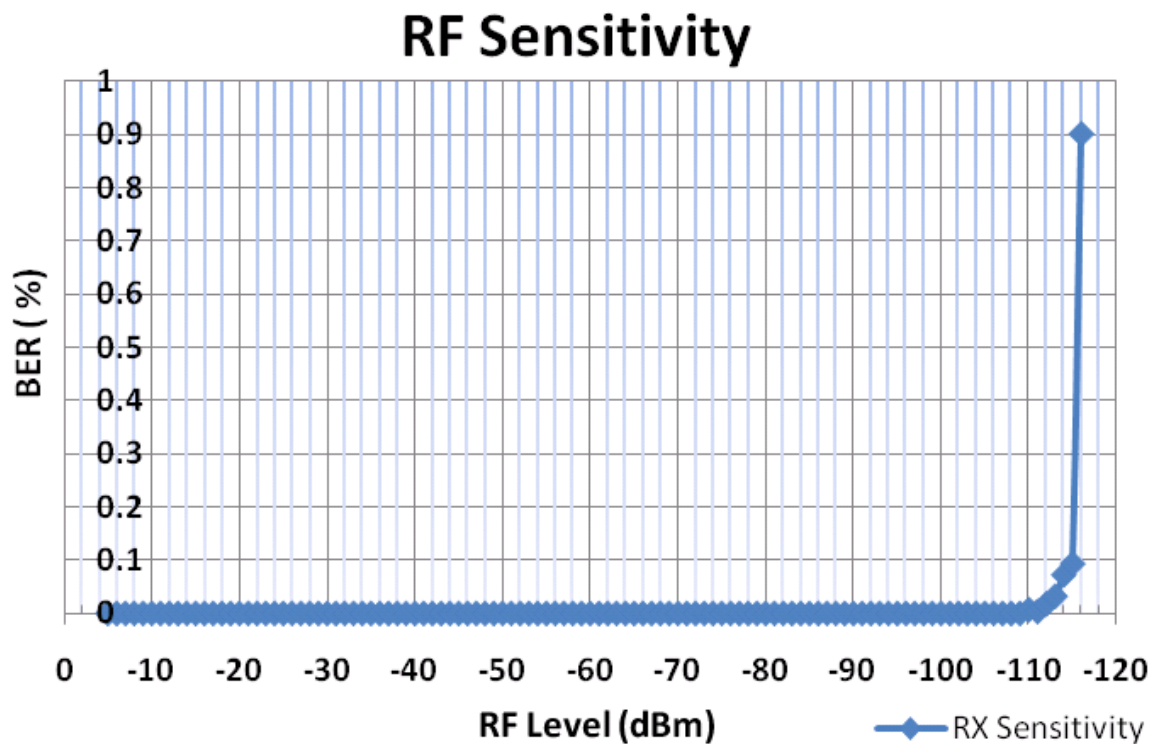


Figure 6. RF Sensitivity with Setting 1 and Deglitch Algorithm 1

Receiver Sensitivity @0.1% BER = -115.7 dBm

Bucket Curve

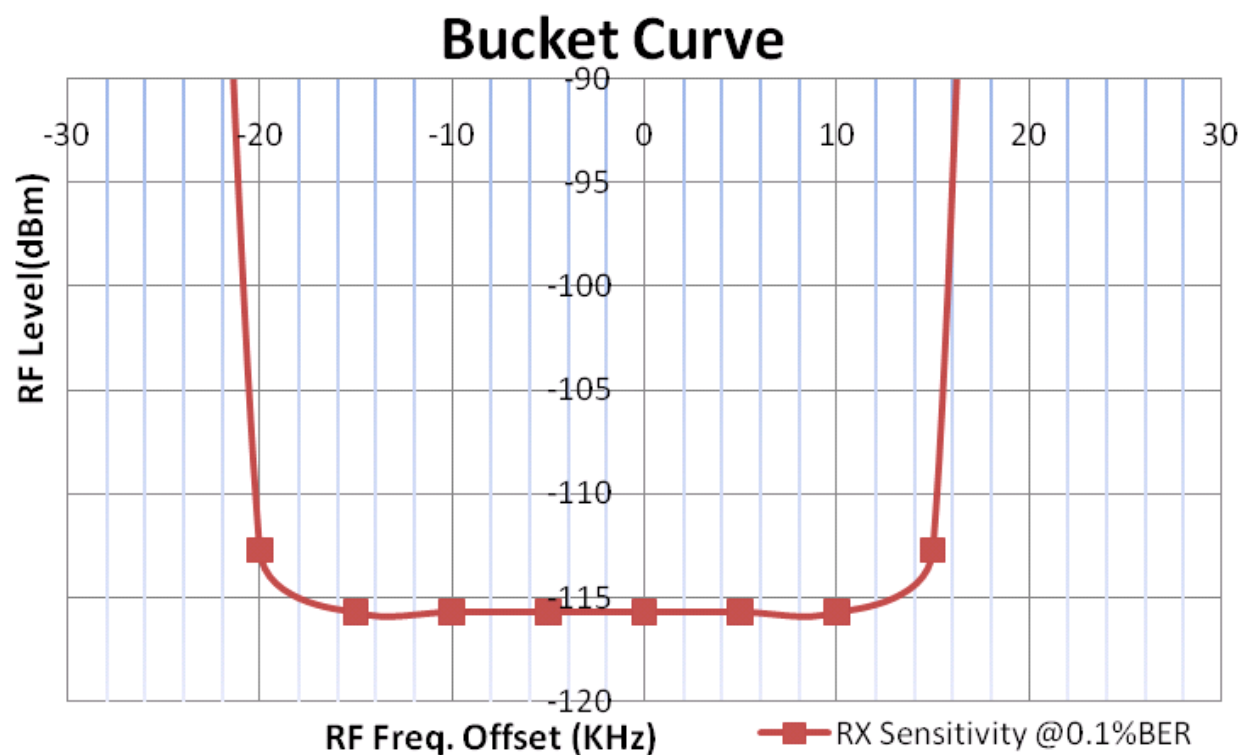


Figure 7. Frequency Error Tolerance with Setting 1 and Deglitch Algorithm 1

Receiver Frequency Error Tolerance @0.1% BER = $\sim \pm 14$ kHz.

RF Blocking Performance

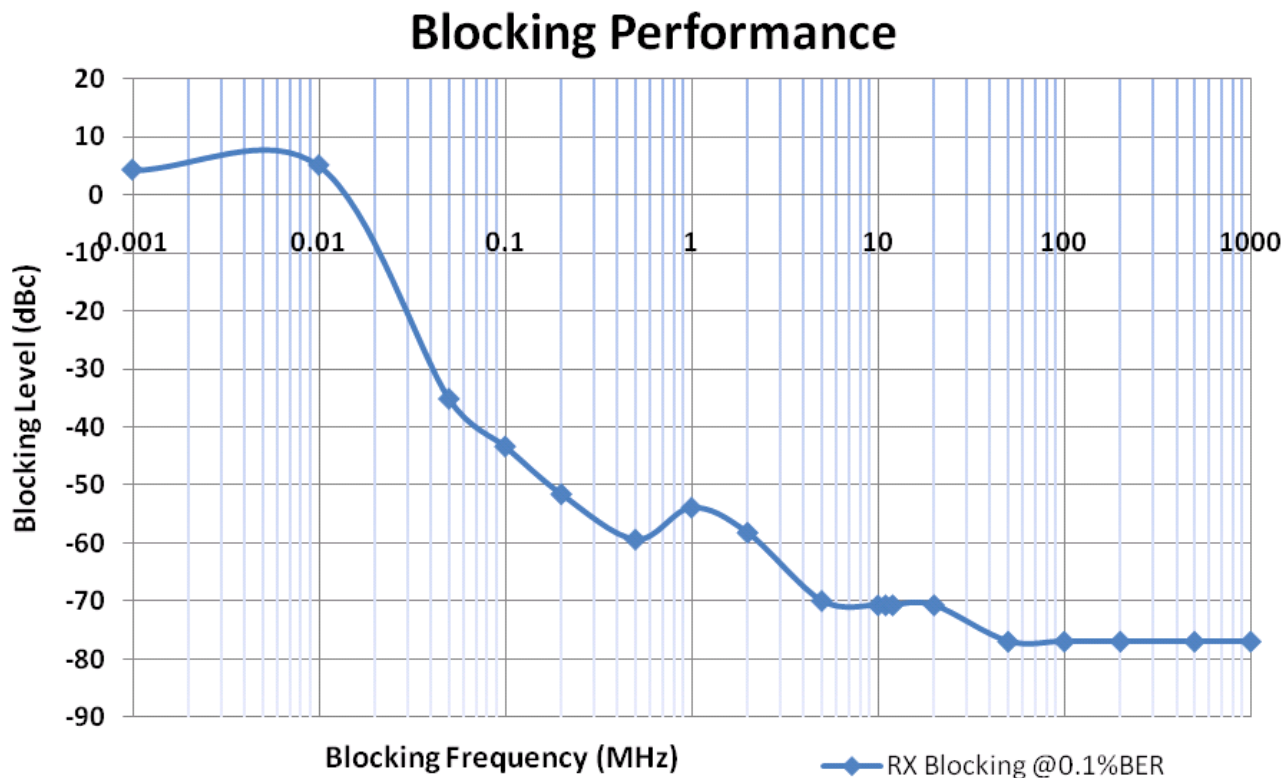


Figure 8. Blocking Performance with Setting 1 and Deglitch Algorithm 1

Receiver Blocking with 0.1% BER @ 1 MHz = -53 dBc

Receiver Blocking with 0.1% BER @ 5 MHz = -68 dBc

Receiver Blocking with 0.1% BER @ 8 MHz = -70 dBc

Setting 2 Results

Test Setup: 4432-T-B1 B 868 split antenna test card.

RF Signal: GFSK, 4K8 data rate, 30K deviation

Measurement Criterion: BER

Deglitch Method: 1

Receiver BW: 153.3 kHz

RF Sensitivity

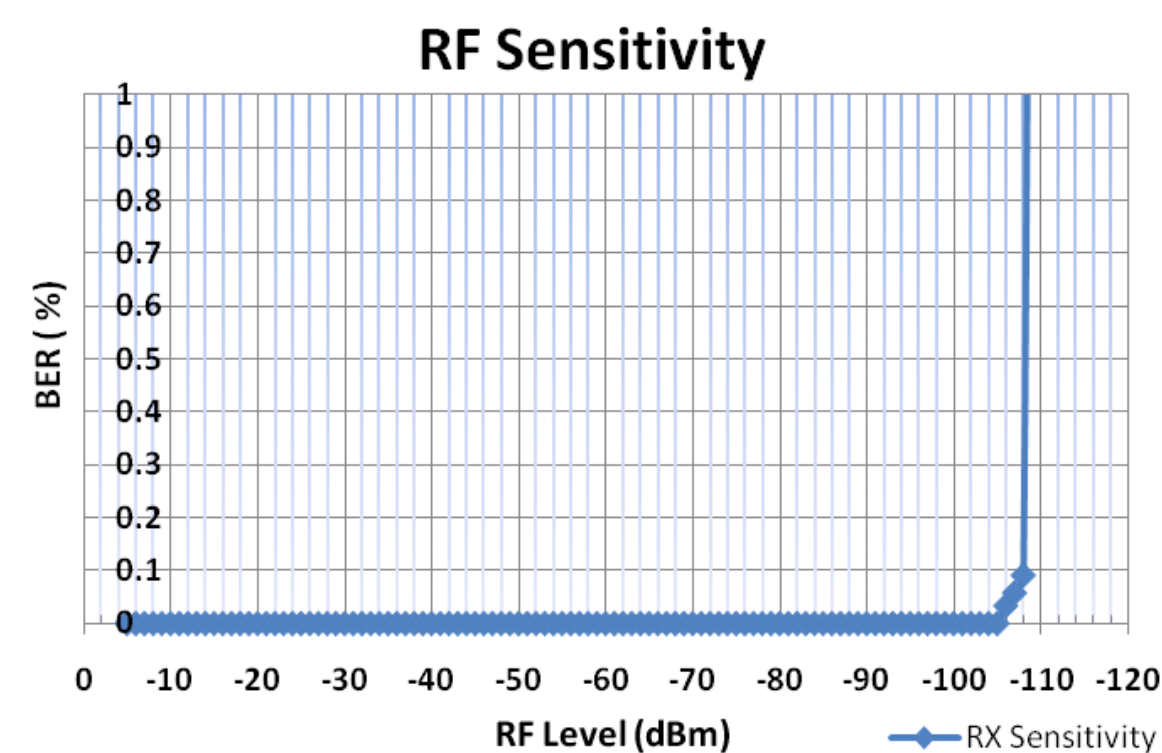


Figure 9. RF Sensitivity with Setting 2 and Deglitch Algorithm 1

Receiver Sensitivity @0.1% BER = -108.5 dBm.

Bucket Curve

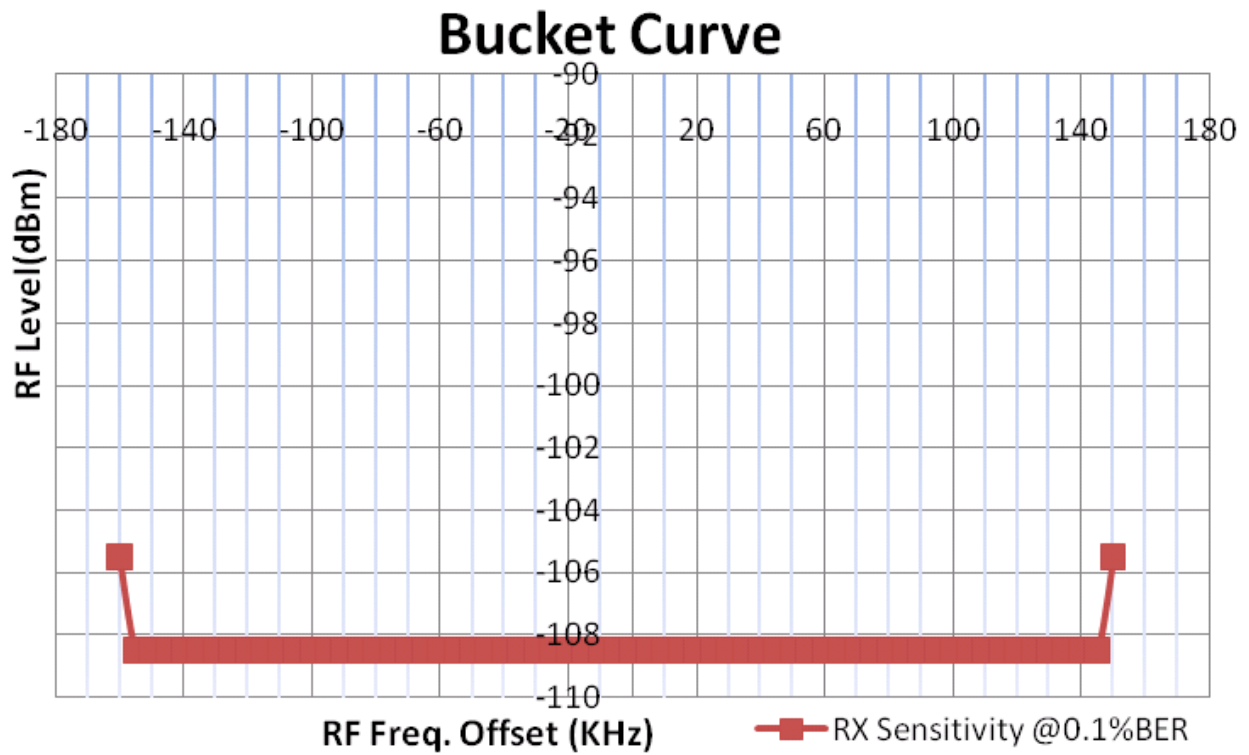


Figure 10. Frequency Error Tolerance with Setting 2 and Deglitch Algorithm 1

Receiver Frequency Error Tolerance @0.1% BER = $\sim \pm 150$ kHz.

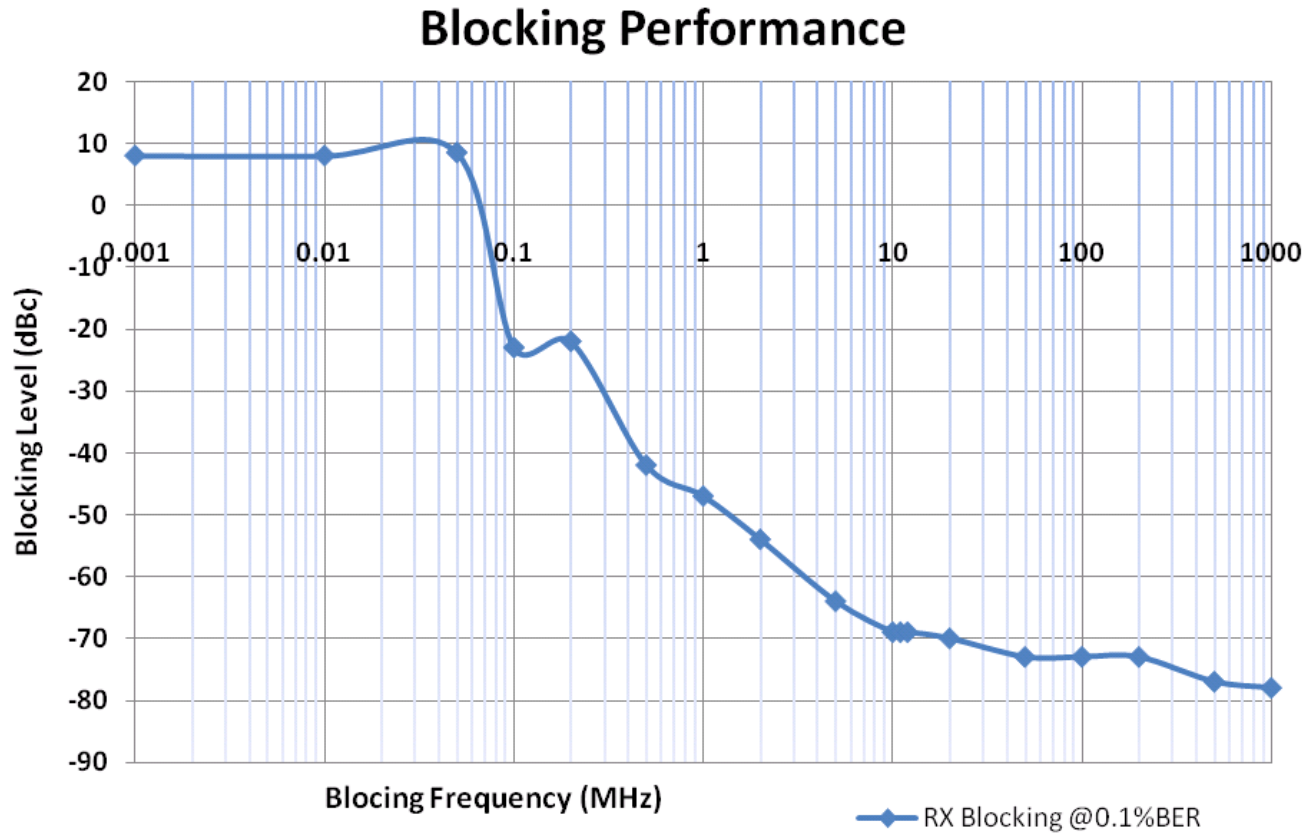


Figure 11. Blocking Performance with Setting 2 and Deglitch Algorithm 1

Receiver Blocking with 0.1% BER @ 1 MHz = -48 dBc.

Receiver Blocking with 0.1% BER @ 5 MHz = -63 dBc.

Receiver Blocking with 0.1% BER @ 8 MHz = -68 dBc.

Setting 3 Results

Test Setup: 4432-T-B1 B 868 split antenna test card.

RF Signal: GFSK, 2K4 data rate, 10K deviation

Measurement Criterion: BER

Deglitch Method: 1

Receiver BW: 102.2 kHz

RF Sensitivity

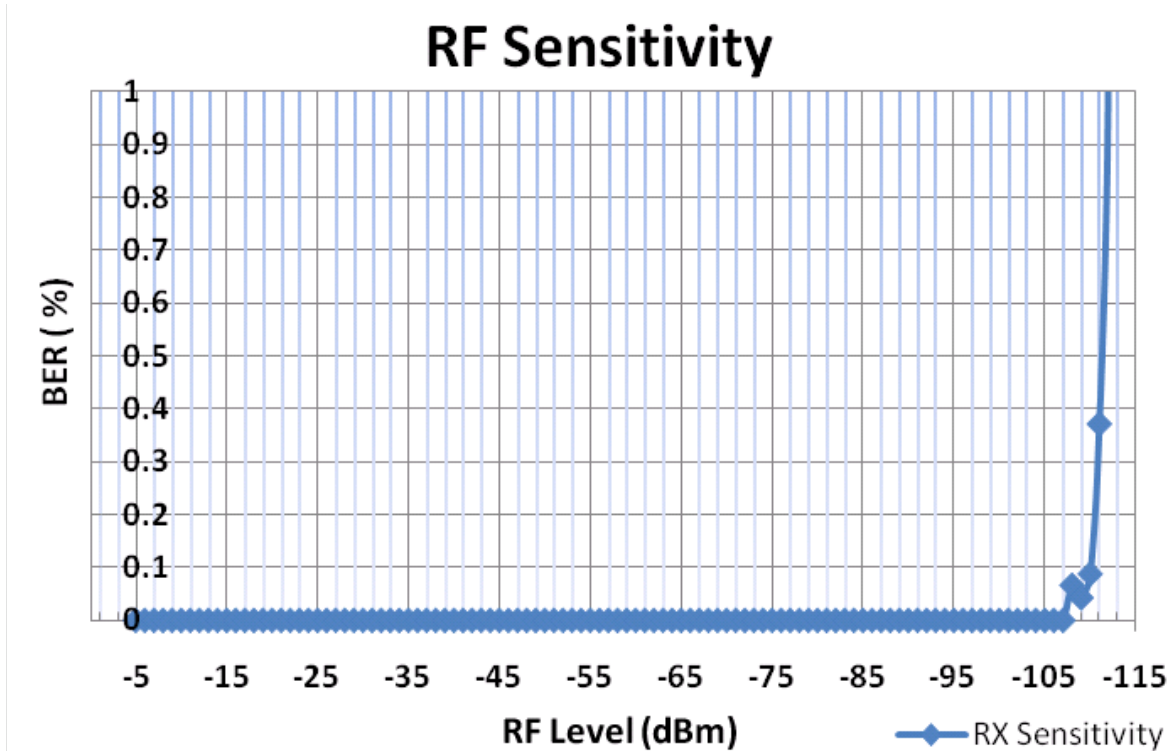


Figure 12. RF Sensitivity with Setting 3 and Deglitch Algorithm 1

Receiver Sensitivity @0.1% BER = -111 dBm.

Bucket Curve

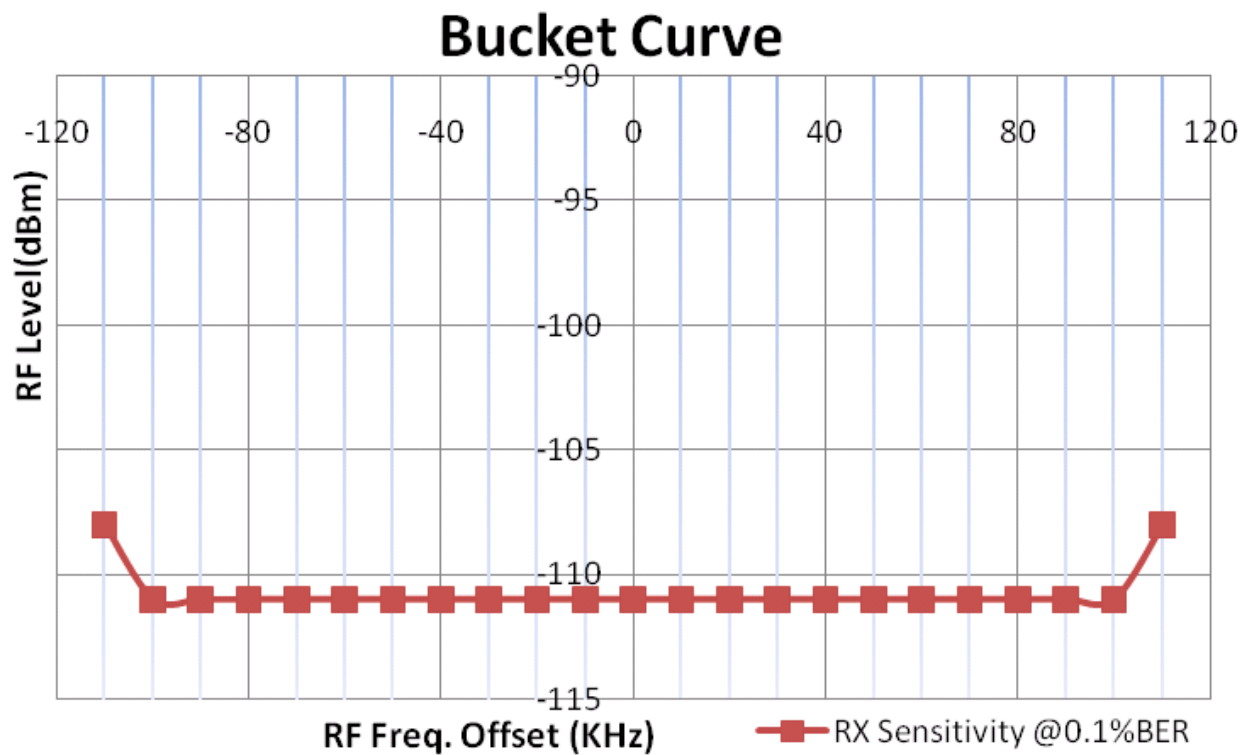


Figure 13. Frequency Error Tolerance with Setting 3 and Deglitch Algorithm 1

Receiver Frequency Error Tolerance @0.1% BER = ~±50 kHz.

RF Blocking Performance

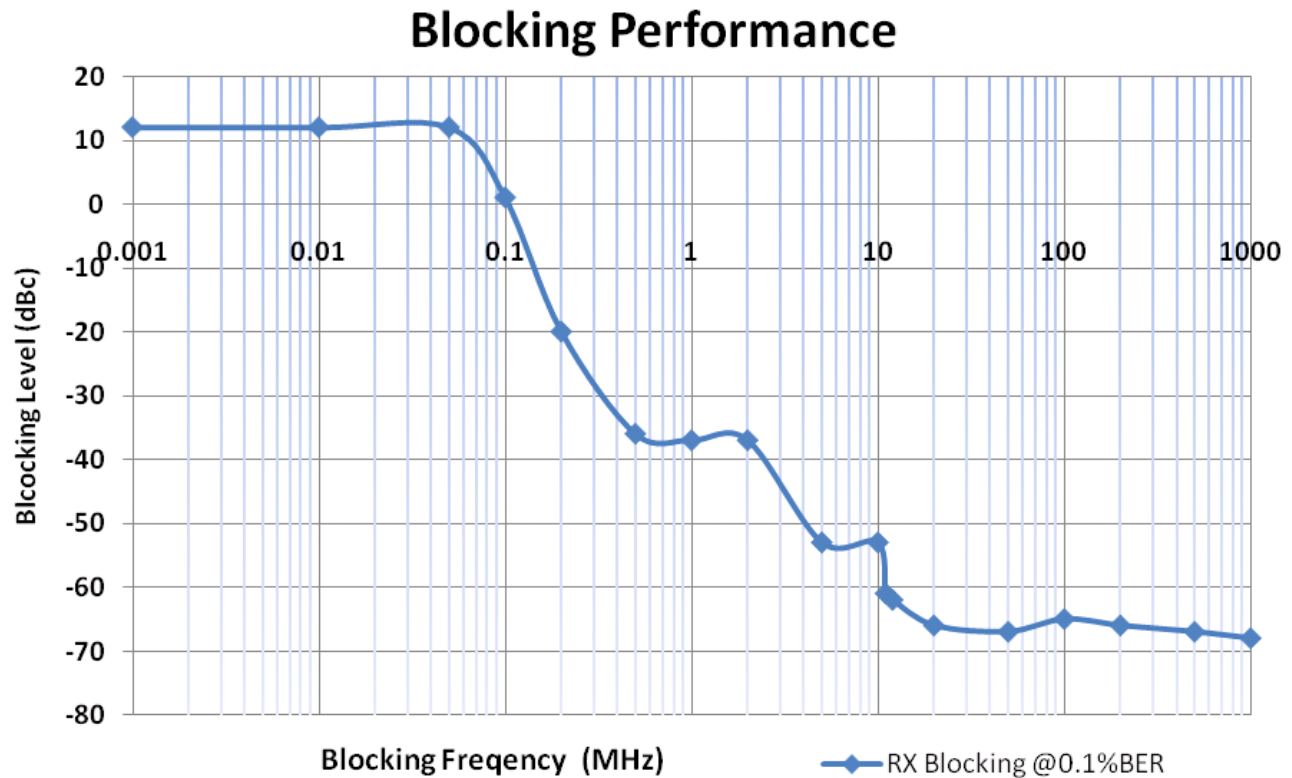


Figure 14. Blocking Performance with Setting 3 and Deglitch Algorithm 1

Receiver Blocking with 0.1% BER @1 MHz = -38 dBc.

Receiver Blocking with 0.1% BER @5 MHz = -52 dBc.

Receiver Blocking with 0.1% BER @8 MHz = -53 dBc.

Setting 4 Results

Test Setup: 4432-T-B1 B 868 split antenna test card.

RF Signal: GFSK, 10K data rate, 5K deviation

Measurement Criterion: BER

Deglitch Method: 1

Receiver BW: 153.3 kHz

RF Sensitivity

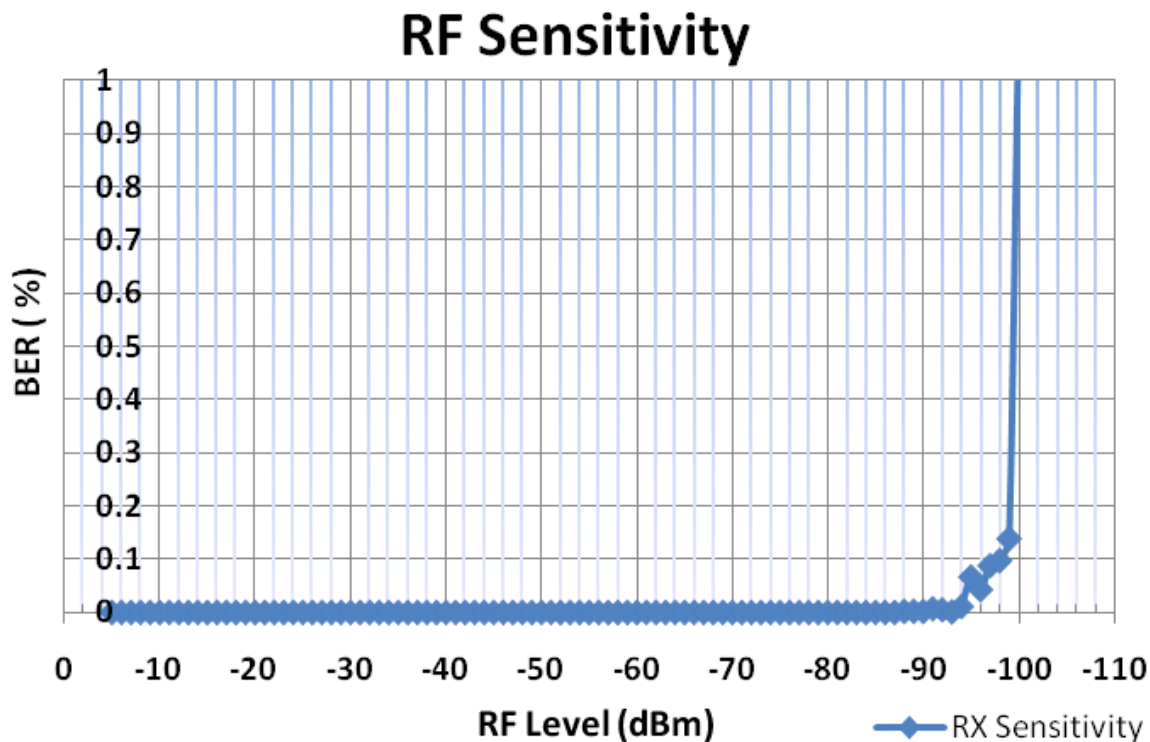


Figure 15. RF Sensitivity with Setting 4 and Deglitch Algorithm 1

Receiver Sensitivity @0.1% BER = -99 dBm.

Bucket Curve

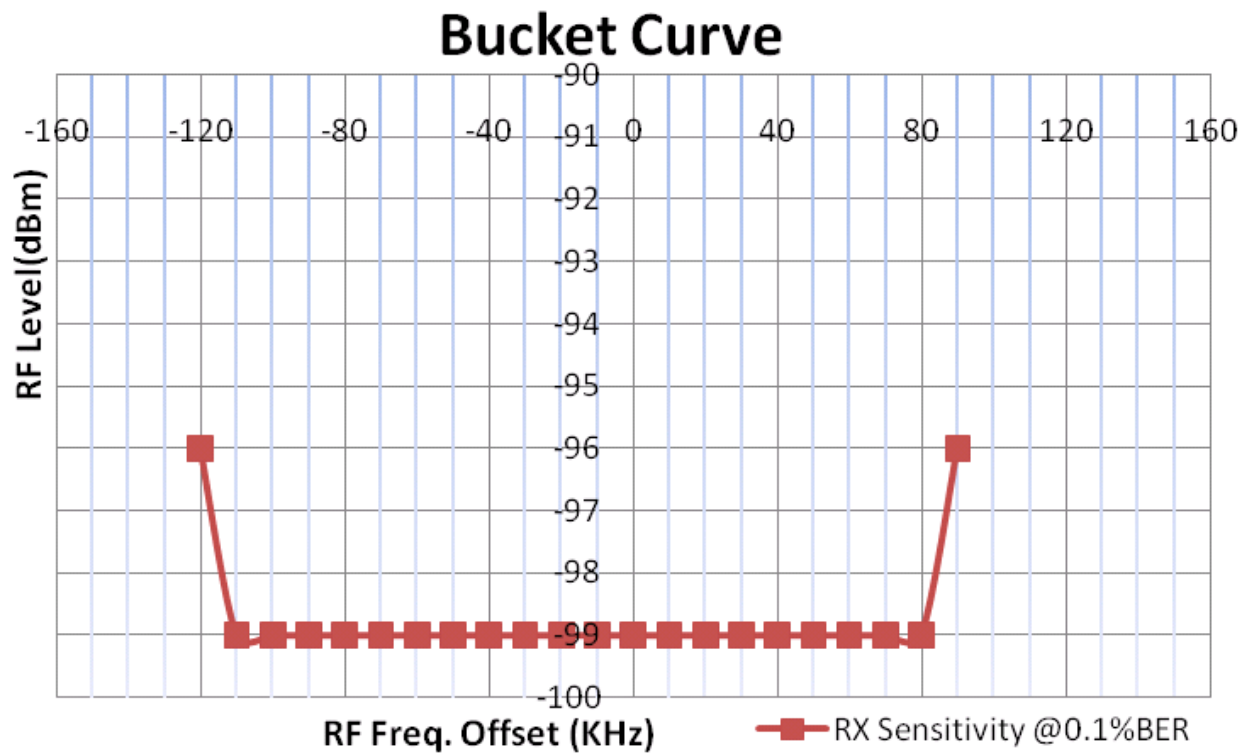


Figure 16. Frequency Error Tolerance with Setting 4 and Deglitch Algorithm 1

Receiver Frequency Error Tolerance @0.1% BER = $\sim \pm 90$ kHz.

RF Blocking Performance

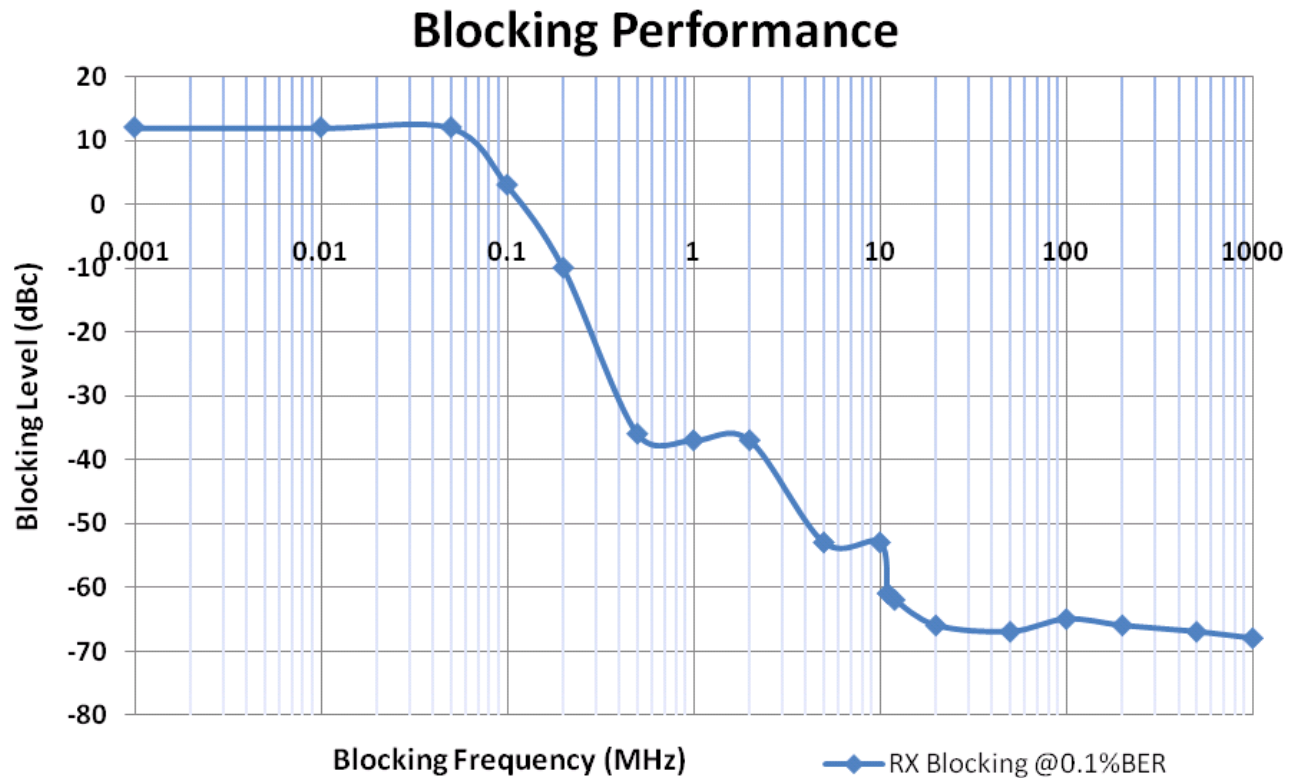


Figure 17. Blocking Performance with Setting 4 and Deglitch Algorithm 1

Receiver Blocking with 0.1% BER @ 1 MHz = -38 dBc.

Receiver Blocking with 0.1% BER @ 5 MHz = -52 dBc.

Receiver Blocking with 0.1% BER @ 8 MHz = -53 dBc.

Setting 5 Results

Test Setup: 4432-T-B1 B 868 split antenna test card.

RF Signal: GFSK, 9K6 data rate, 75K deviation

Measurement Criterion: BER

Deglitch Method: 1

Receiver BW: 620.7 kHz

RF Sensitivity

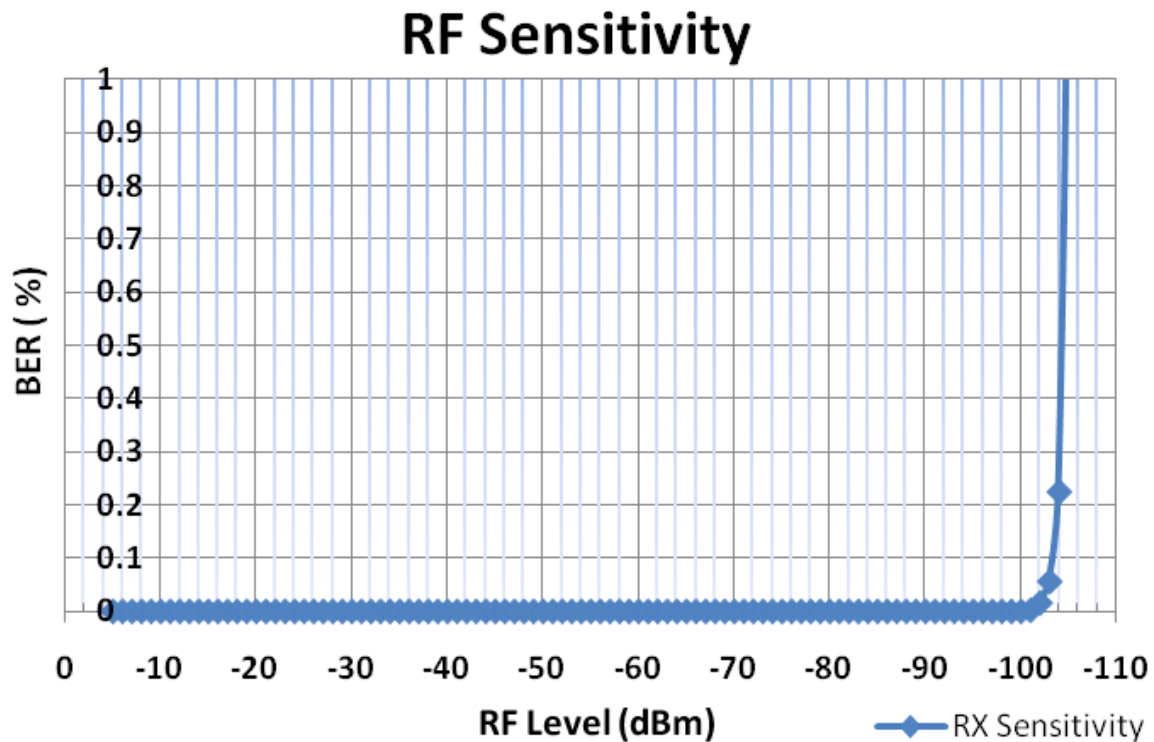


Figure 18. RF Sensitivity with Setting 5 and Deglitch Algorithm 1

Receiver Sensitivity @0.1% BER = -104 dBm.

Bucket Curve

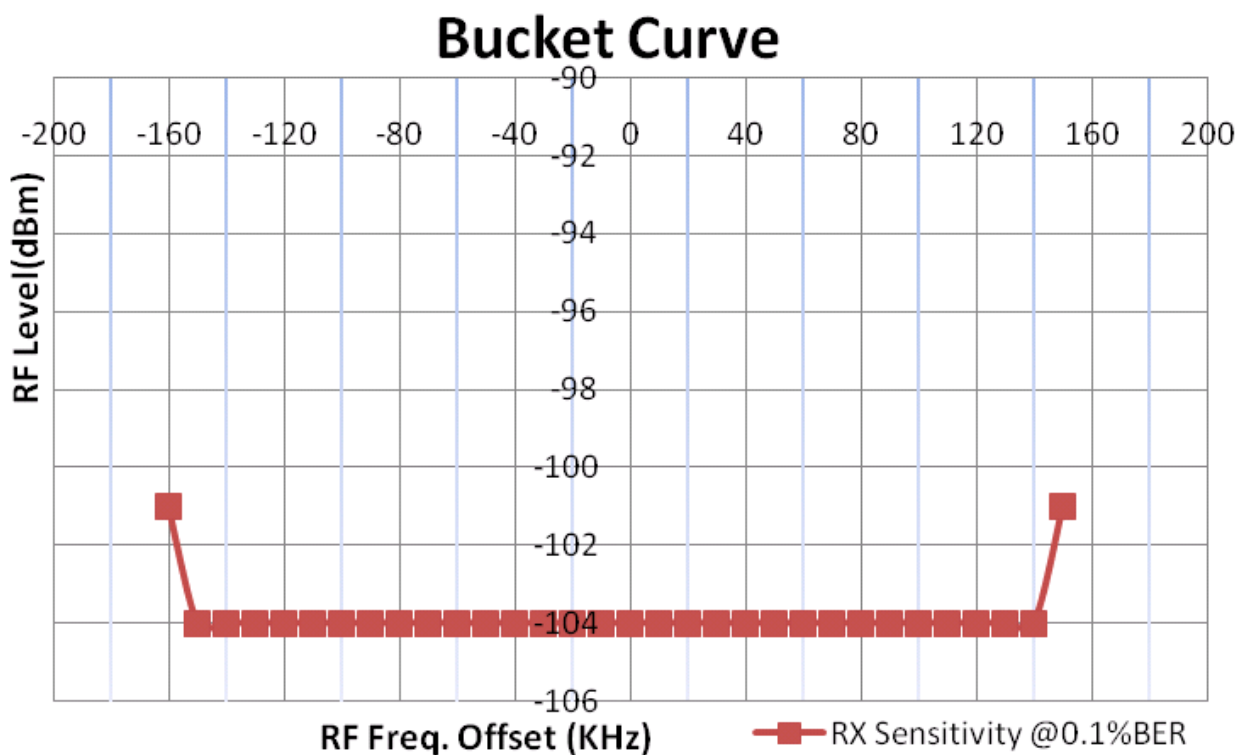


Figure 19. Frequency Error Tolerance with Setting 5 and Deglitch Algorithm 1

Receiver Frequency Error Tolerance @0.1% BER = $\sim \pm 150$ kHz.

RF Blocking Performance

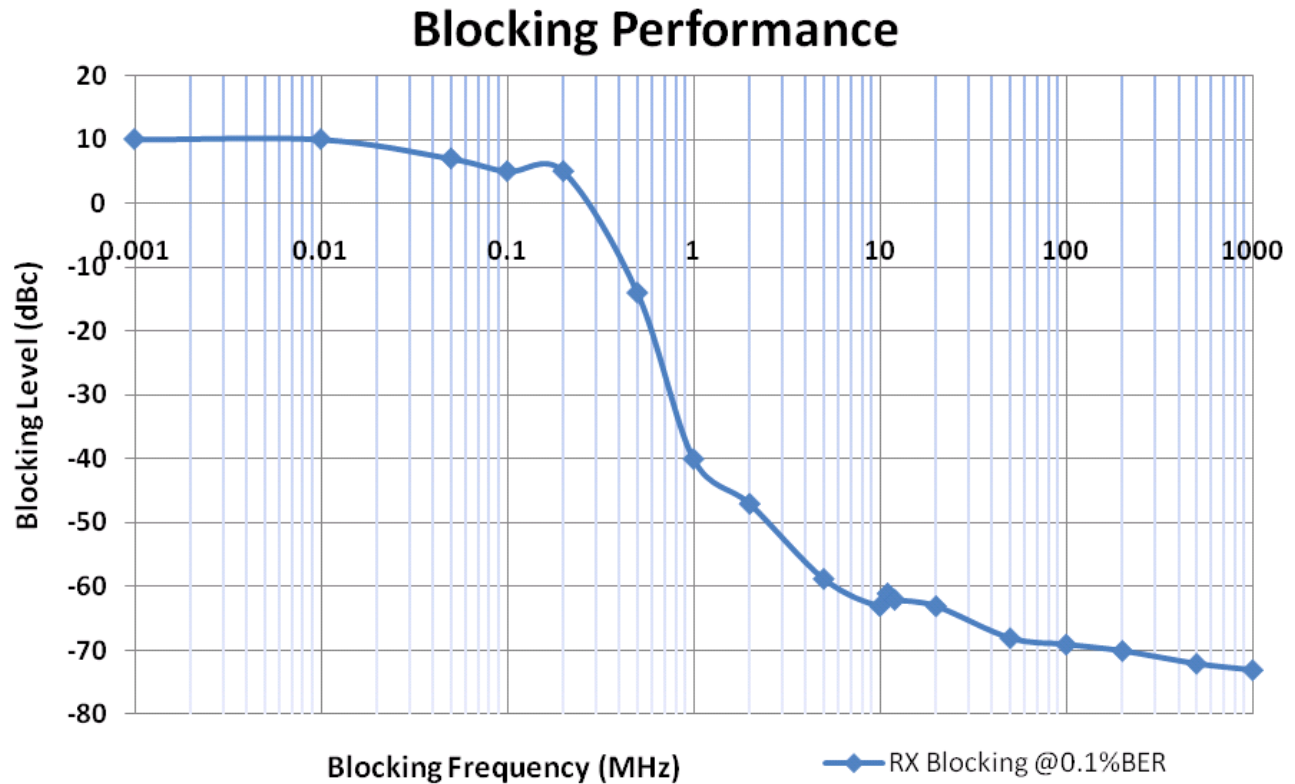


Figure 20. Blocking Performance with Setting 5 and Deglitch Algorithm 1

Receiver Blocking with 0.1% BER @1 MHz = -40 dBc

Receiver Blocking with 0.1% BER @5 MHz = -59 dBc

Receiver Blocking with 0.1% BER @8 MHz = -62 dBc

OOK Measurement Results

Setting 6 Results

Test Setup: 4432-T-B1 B 868 split antenna test card.

RF Signal: OOK (Pulse Modulation), 1K data rate

Measurement Criterion: BER

Deglitch Method: 1

Receiver BW: 204.3 kHz

RF Sensitivity

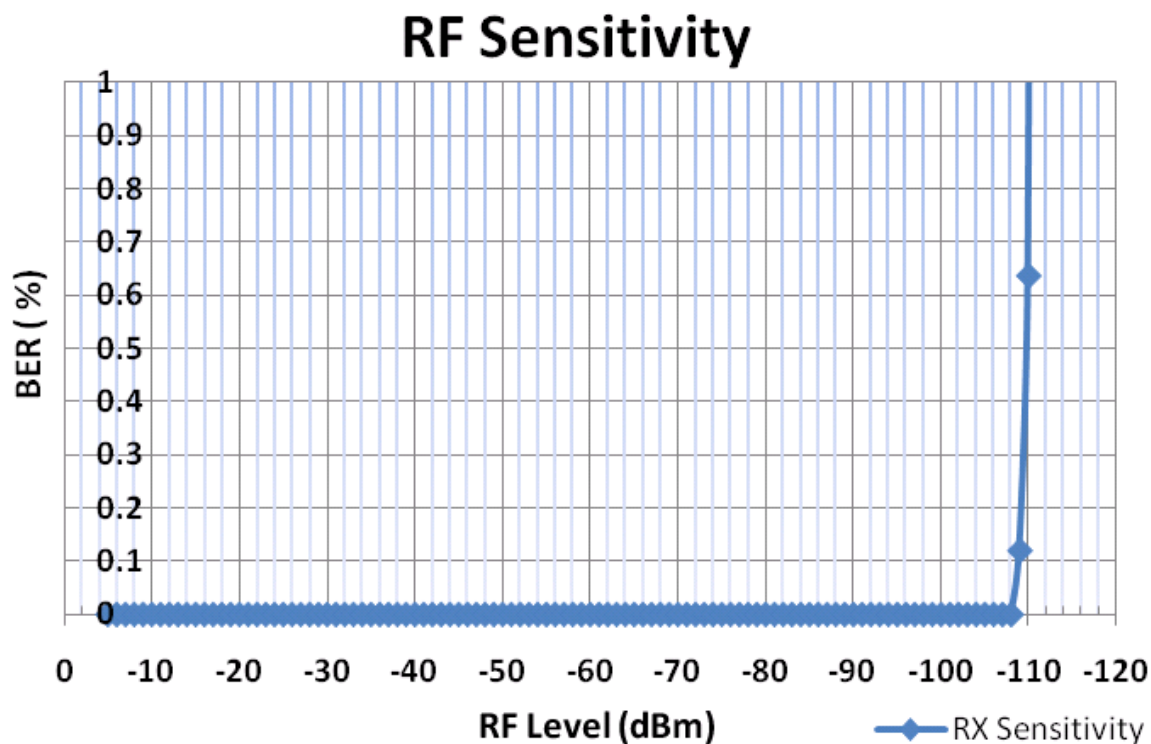


Figure 21. RF Sensitivity with Setting 6 and Deglitch Algorithm 1

Receiver Sensitivity @0.1% BER = -109 dBm.

Bucket Curve

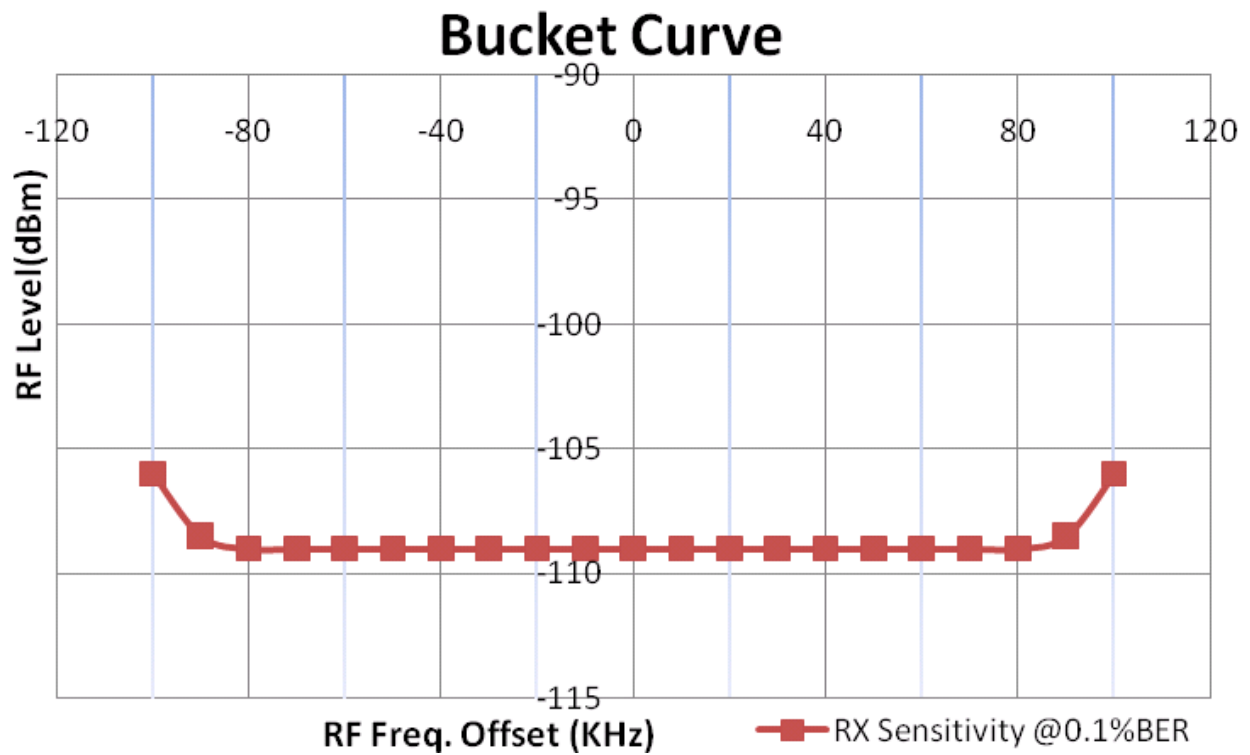


Figure 22. Frequency Error Tolerance with Setting 6 and Deglitch Algorithm 1

Receiver Frequency Error Tolerance @0.1% BER = $\sim \pm 100$ kHz.

RF Blocking Performance

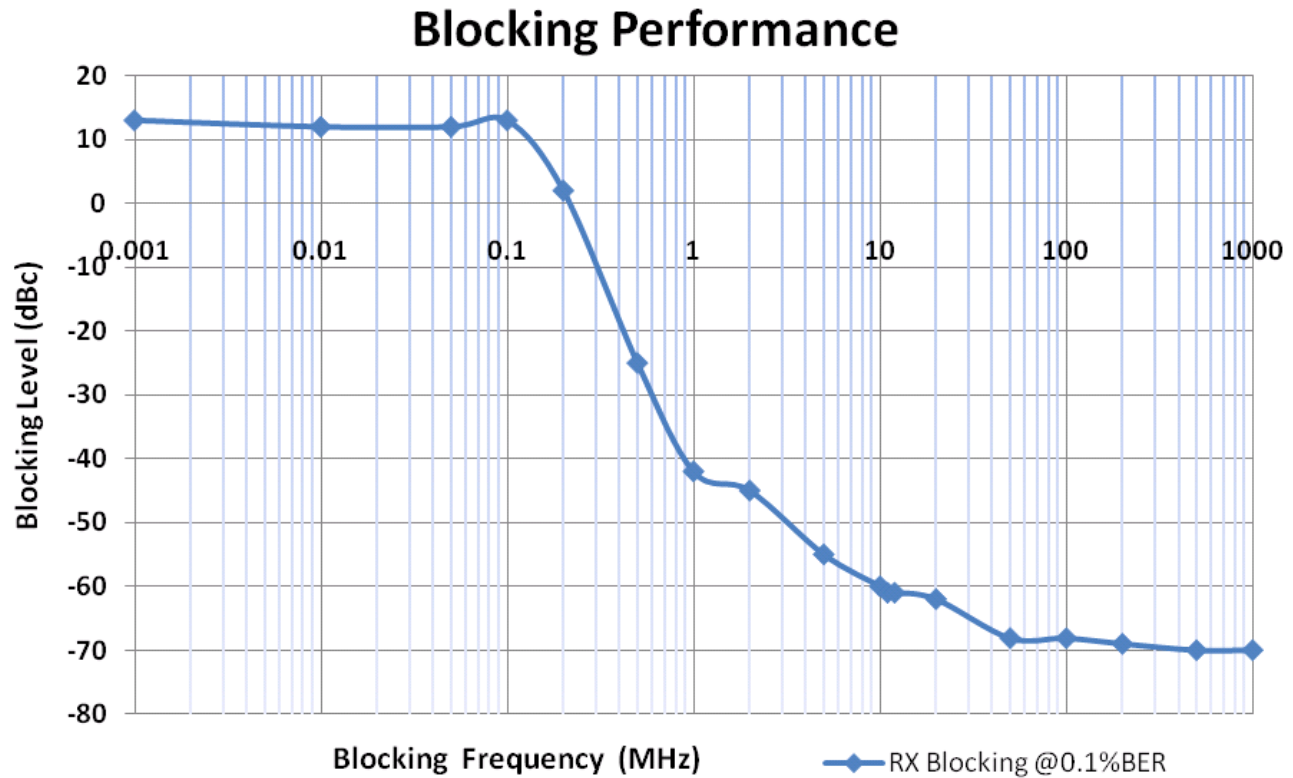


Figure 23. Blocking Performance with Setting 6 and Deglitch Algorithm 1

Receiver Blocking with 0.1% BER @1 MHz = -41 dBc.

Receiver Blocking with 0.1% BER @5 MHz = -55 dBc.

Receiver Blocking with 0.1% BER @8 MHz = -60 dBc .

Setting 7 Results

Test Setup: 4432-T-B1 B 868 split antenna test card.

RF Signal: OOK (Pulse Modulation), 2K4 data rate

Measurement Criterion: BER

Deglintch Method: 1

Receiver BW: 412.1 kHz

RF Sensitivity

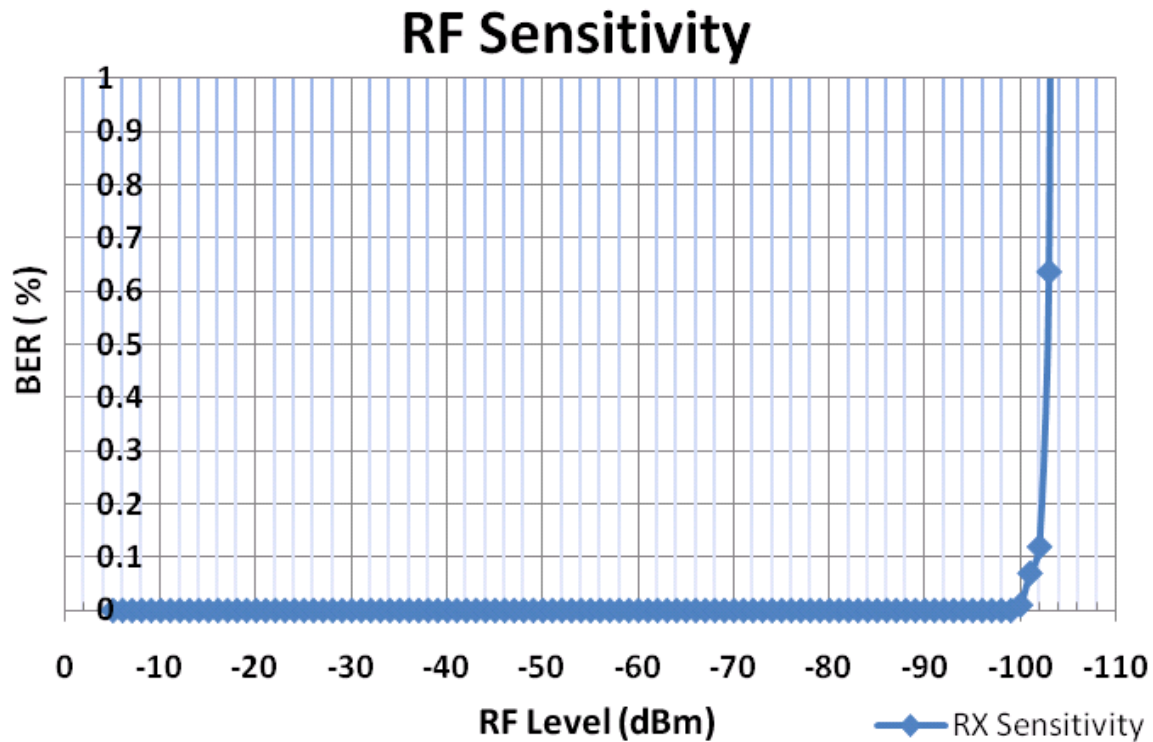


Figure 24. RF Sensitivity with Setting 7 and Deglitch Algorithm 1

Receiver Sensitivity @0.1% BER= -102 dBm.

Bucket Curve

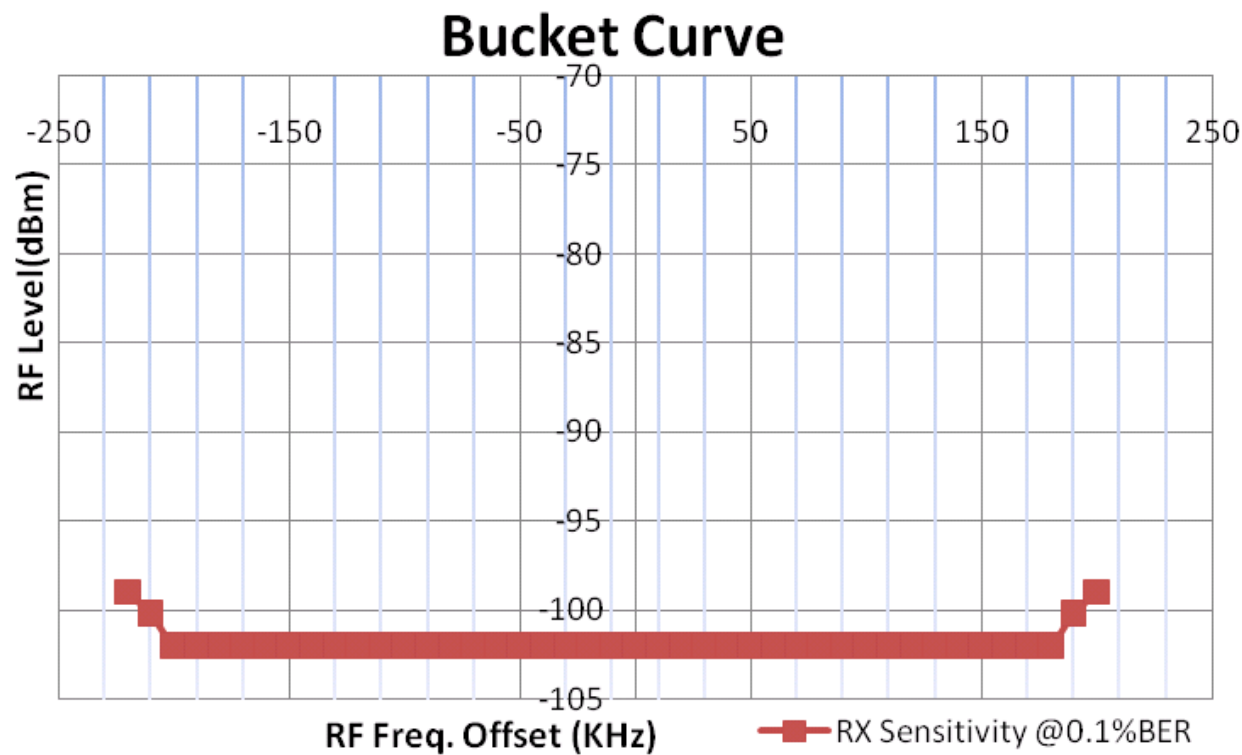


Figure 25. Frequency Error Tolerance with Setting 7 and Deglitch Algorithm 1

Receiver Frequency Error Tolerance @0.1% BER = $\sim \pm 200$ kHz.

RF Blocking Performance

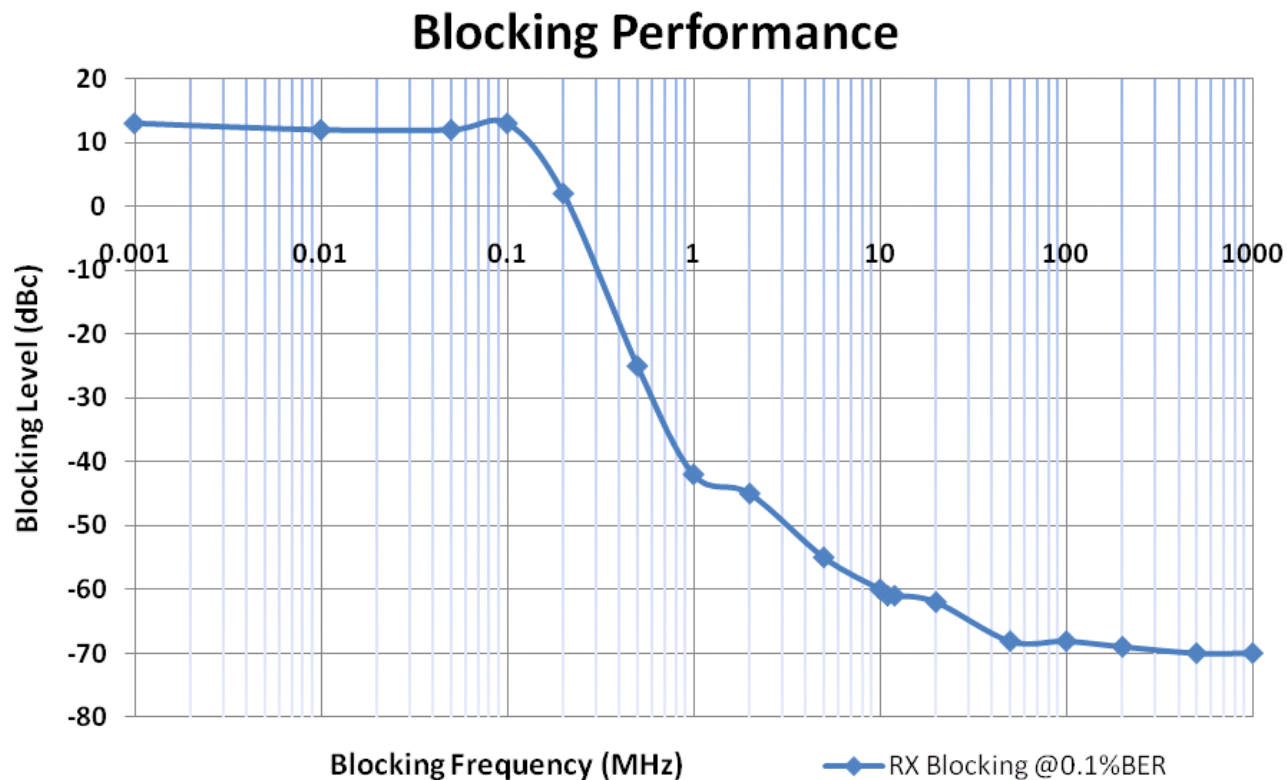


Figure 26. Blocking Performance with Setting 7 and Deglitch Algorithm 1

Receiver Blocking with 0.1% BER @ 1 MHz = -41 dBc.

Receiver Blocking with 0.1% BER @ 5 MHz = -55 dBc.

Receiver Blocking with 0.1% BER @ 8 MHz = -60 dBc.

Setting 8 Results

Test Setup: 4432-T-B1 B 868 split antenna test card.

RF Signal: OOK (Pulse modulation), 10K data rate

Measurement Criterion: BER

Deglitch Method: 1

Receiver BW: 204.3 kHz

RF Sensitivity

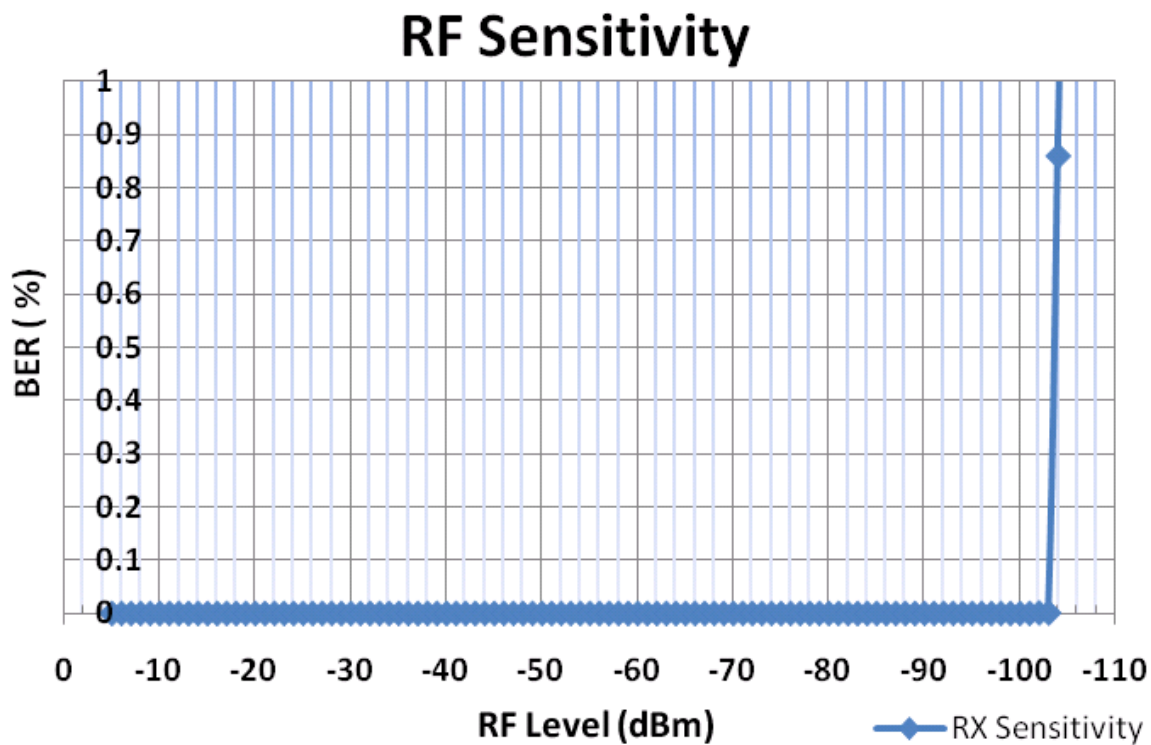


Figure 27. RF Sensitivity with Setting 8 and Deglitch Algorithm 1

Receiver Sensitivity @0.1% BER = -104 dBm.

Bucket Curve

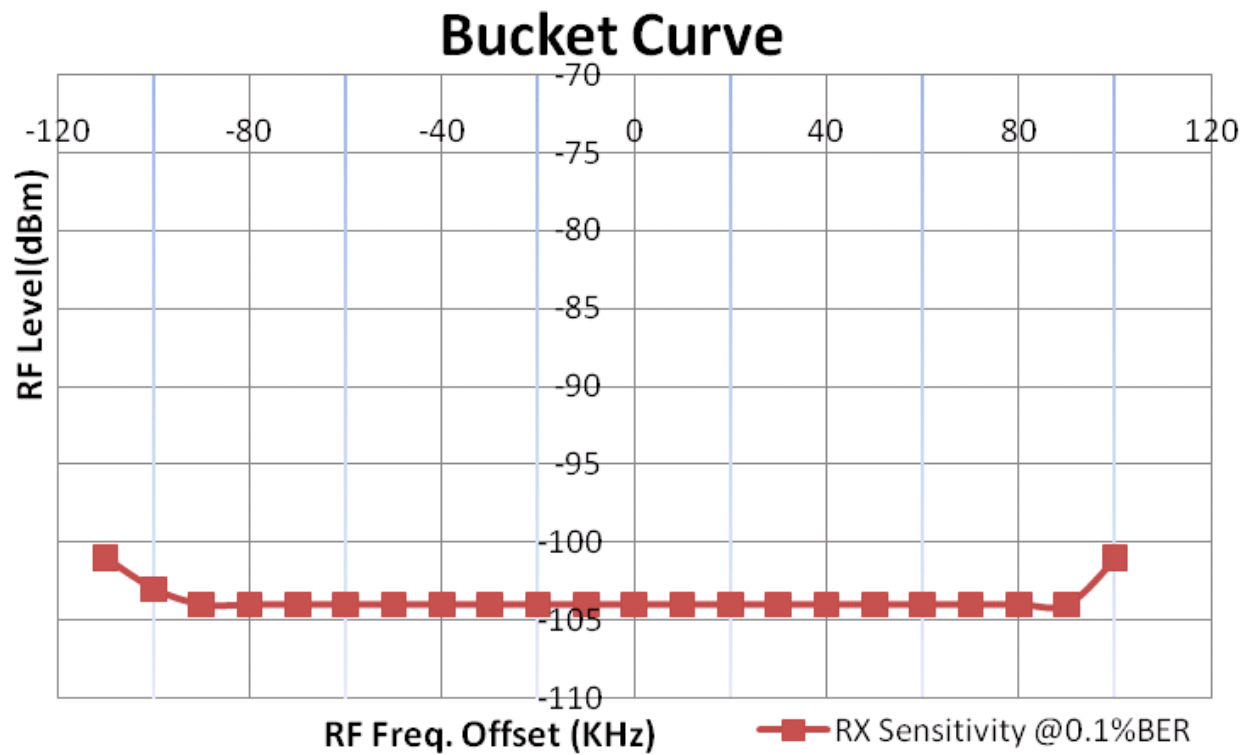


Figure 28. Frequency Error Tolerance with Setting 8 and Deglitch Algorithm 1

Receiver Frequency Error Tolerance @0.1% BER = $\sim \pm 100$ kHz.

RF Blocking Performance

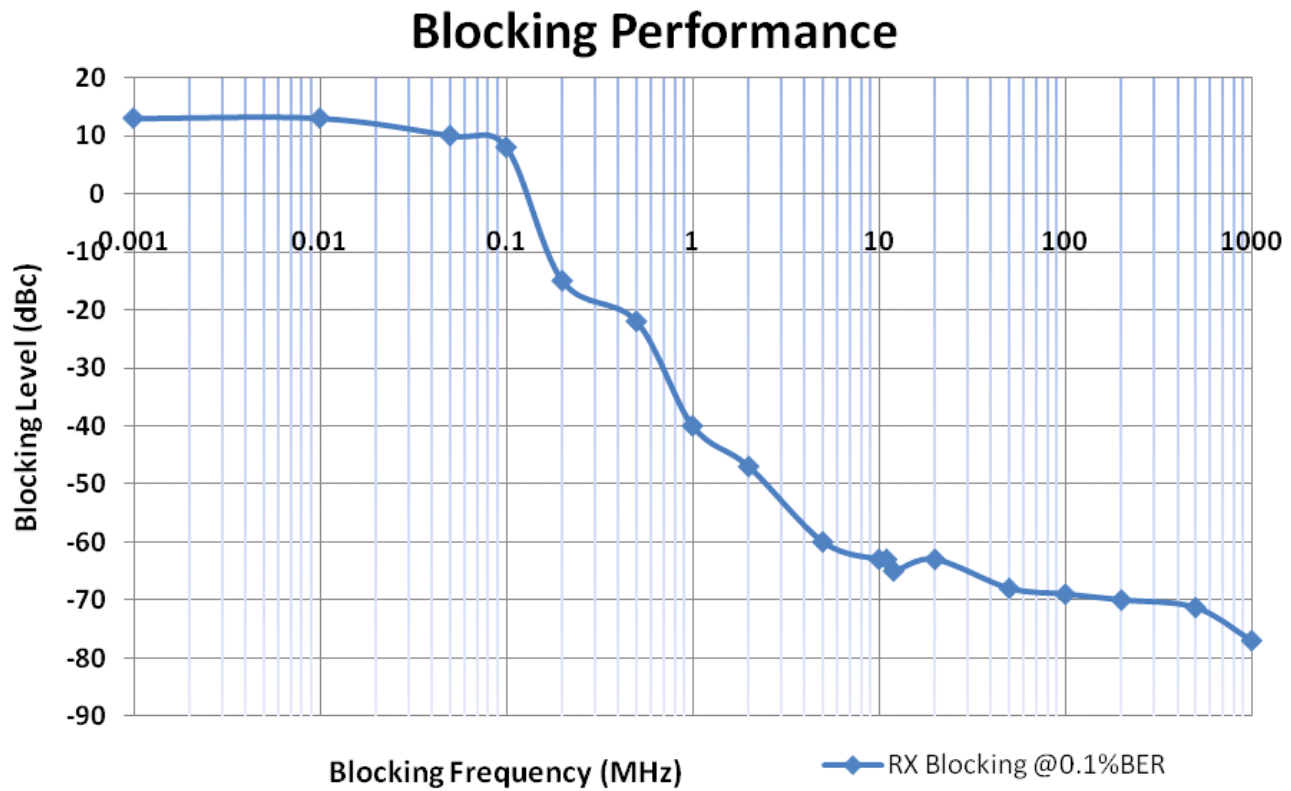


Figure 29. Blocking Performance with Setting 8 and Deglitch Algorithm 1

Receiver Blocking with 0.1% BER @ 1 MHz = -40 dBc.

Receiver Blocking with 0.1% BER @ 5 MHz = -60 dBc.

Receiver Blocking with 0.1% BER @ 8 MHz = -62 dBc.

Sample LoadBoard Scripts

Following is a sample script for receiving a 4K data rate, OOK signal at 915 MHz. The script can be easily updated for different modulation, data rate, deviation, BW, frequency etc.

```
#BATCHNAME Raw_Data_Mode_Script
# Set SDN Pin 20 = LOW
L6
# Set VDD: 1.8V=V54, 2.4V=V7E, 3.0V=V98, 3.3V=VA1, 3.6V=VA9
VA1
# Apply Software Reset
S2 8780
#Clear Interrupts
S2 0300
S2 0400
# Adjust Crystal for zero freq error (User may need to modify this value)
S2 89d5
# Set Desired Receive Frequency
# Use excel calculator to obtain frequency settings
S2 F575
S2 F6BB
S2 F780
# Set Packet Handler
# Set to avoid false packet detection for Raw Data Mode
S2 B5F8
S2 B084
S2 B20F
S2 B377
S2 8810
#Set GPIOs
S2 8BD4
S2 8CCF
S2 D136
# Set RX Modem Parameters
# Copy the appropriate Setting from the Raw Data Mode Application note here
S2 9CD7
s2 9D40
S2 A07f
S2 A1c3
S2 A288
S2 A325
S2 A401
S2 A508
S2 AC18
S2 ADD0
S2 AE2E
```

```
#Set Modulation (GFSK/FSK/OOK)
#OOK Modulation
S2 F169
S2 F00E
#Turn Receiver ON
S2 8704
```

BER Measurements

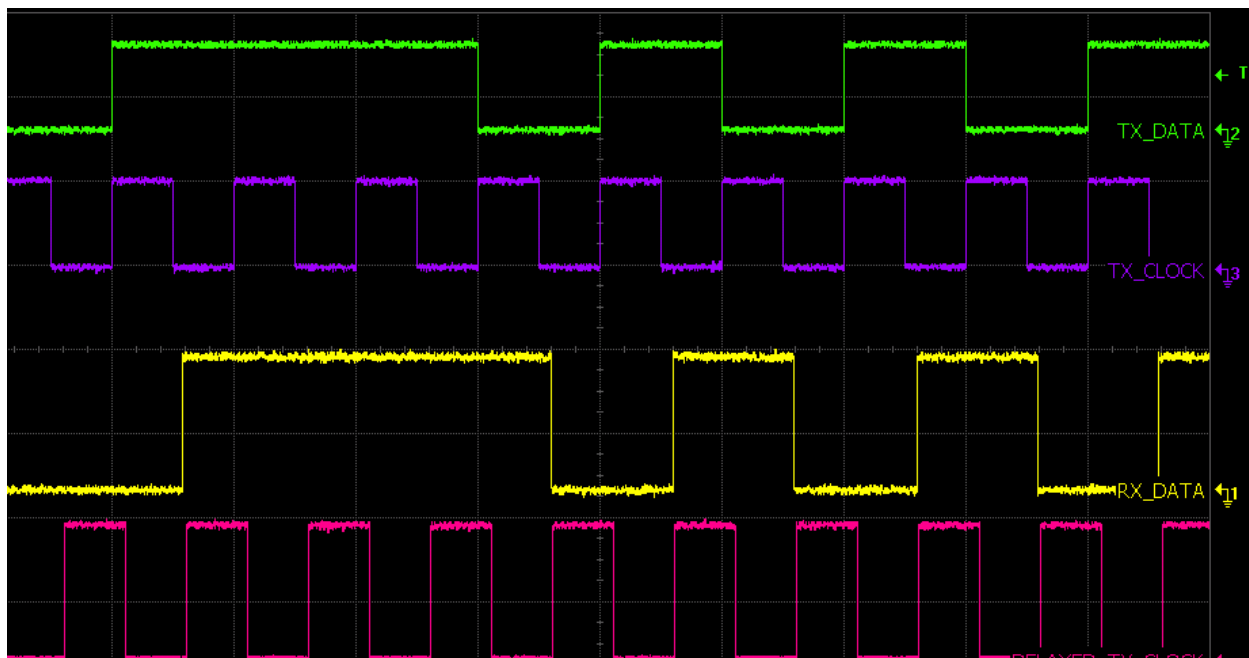


Figure 30. BER Measurements

Figure 30 shows the TX_Data (from the signal generator), TX_Clock (from the signal generator). The RX_Data is obtained from the RF chip or from the output of Deglitching algorithm. An external clock source such as a function generator or a signal generator clock or a delayed TX_Clock can be used for BER measurements. In order to get accurate BER measurement results the RX_Clock must be aligned with RX_Data such that the sampling edge of the clock is in the middle of the RX_Data.

PER Measurements

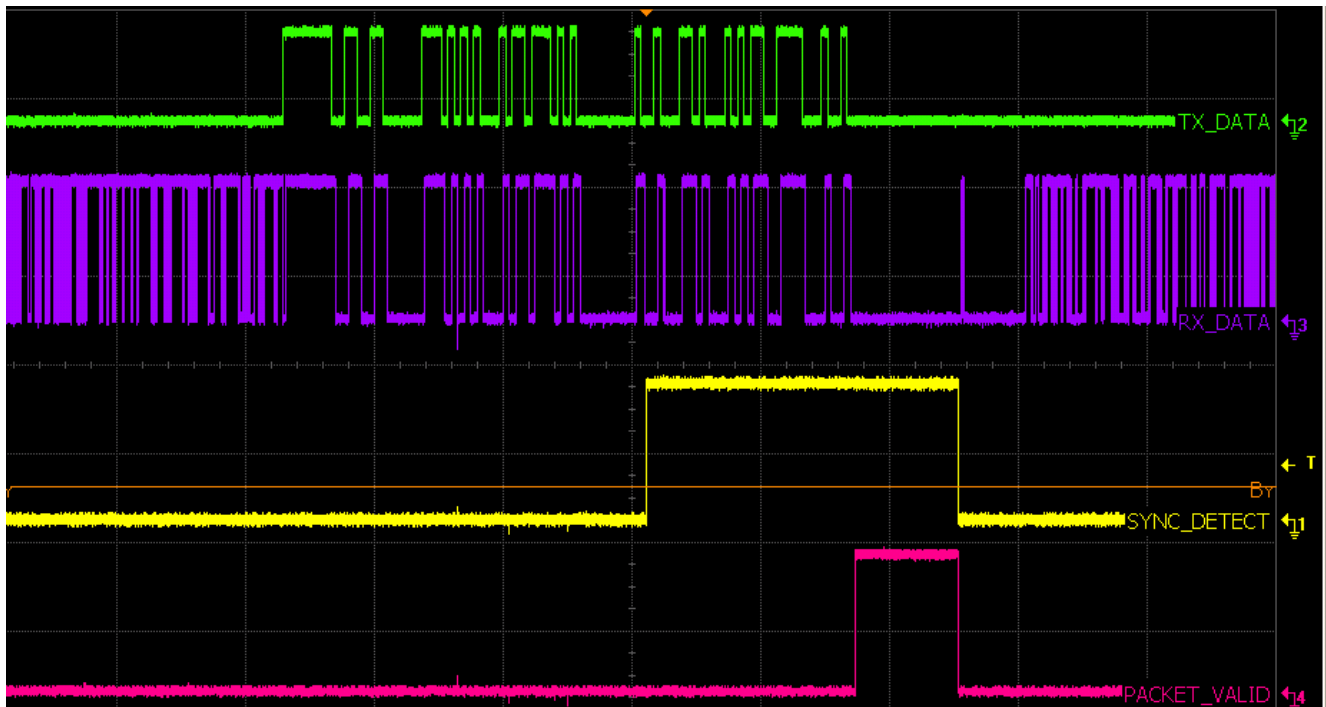


Figure 31. PER Measurements

Figure 30 shows a fixed packet is set on the TX generator. The packet used for the PER test is as follows:
 FF 33 03 AA 2D D4 01 23 45 67 89 00

Packets are sent in a continuous mode or a triggered mode. The Sync_Detect and Packet_Valid signals are monitored on the LED1 and LED2. The Received data is available on LED3, while the inverted received data is available on LED4.

Board connections

The deglitching software can be easily tested using the C8051F9xx Wireless Software Development Board (MSC-DBSB8). For the correct operation external wires must be connected between some pins on the development board.

On board version 1.1 two jumpers must be added:

1. Connect pin 4 of J8 to pin 8 (P0.7) of J1 (RX Data Clock).
2. Connect pin 2 of J8 to pin 3 (P2.2) of J3 (RX Data).

See the orange and light blue lines in Figure 32.

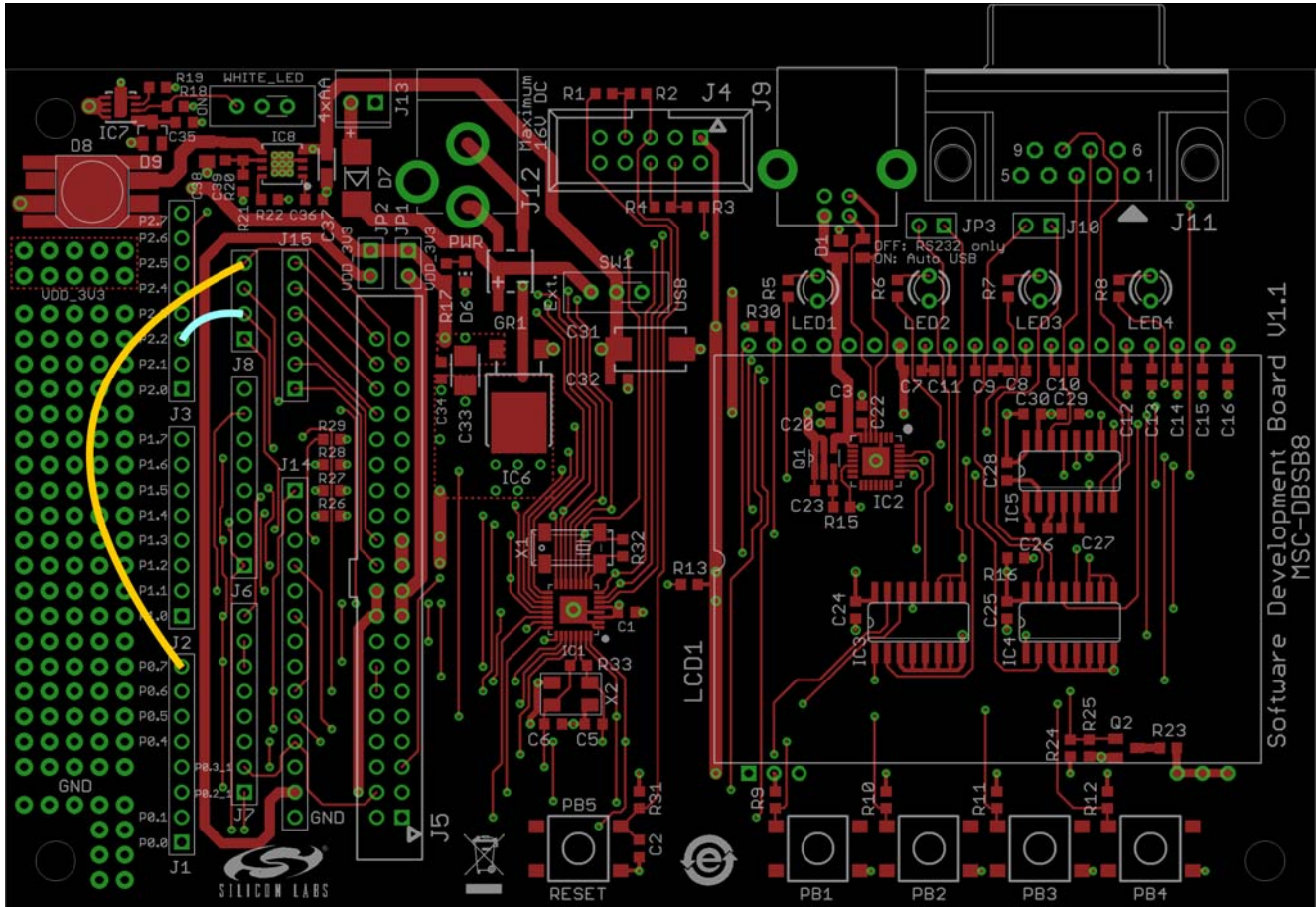


Figure 32. SDBD v1.1 Connections

On board version 1.2 the RX Data is already connected to the microcontroller, so only a single wire needs to be connected:

- Connect pin 4 of J8 to pin 8 (P0.7) of J1 (RX Data Clock)—the orange line on Figure 33.

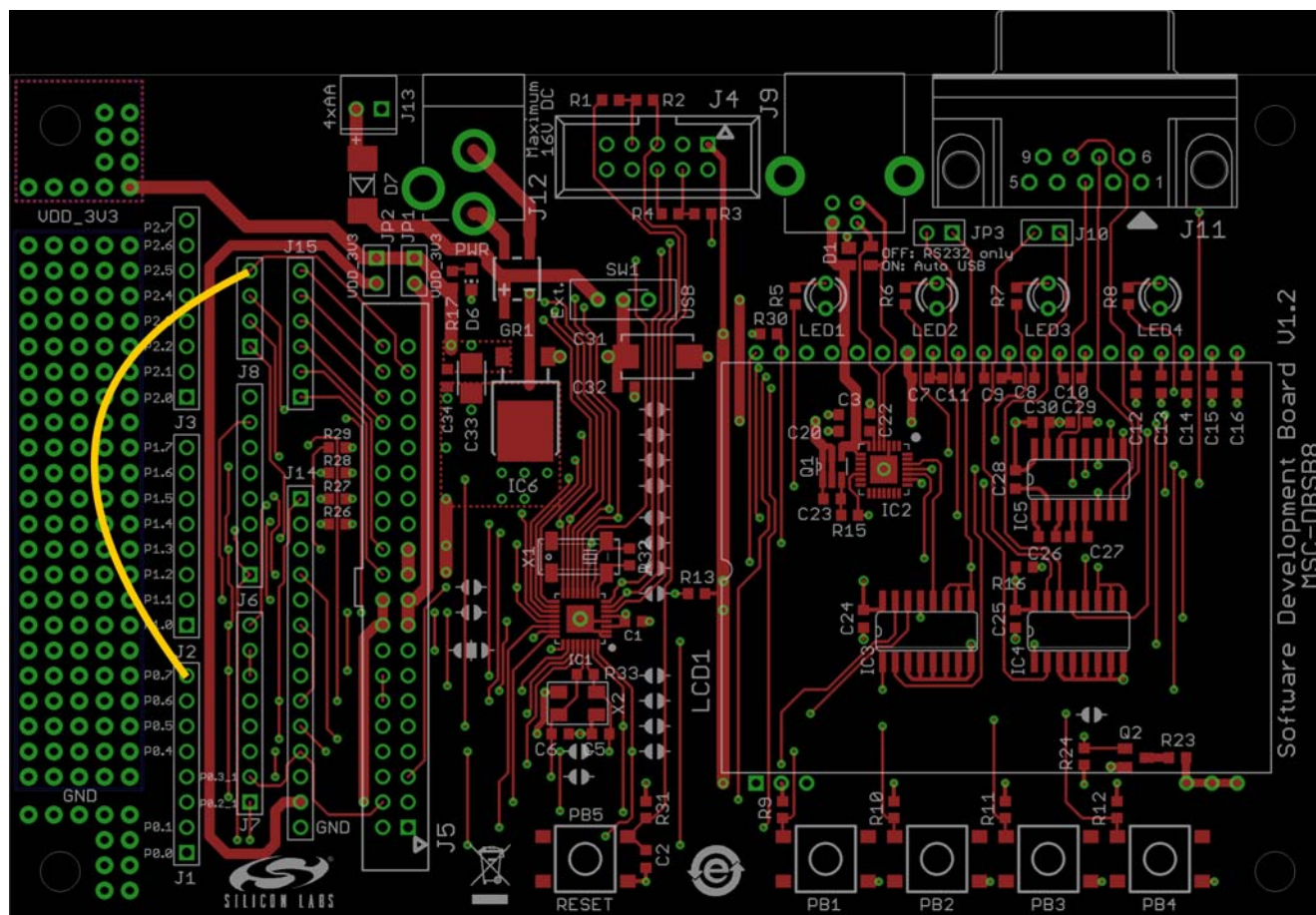


Figure 33. SDBD v1.2 Connections

The Sync_Detect signal is available on pin 5, the Packet_Valid on pin 6, the received data on pin 7 and the inverted received data on pin 8 of J2 connector. These signals can be monitored on the LEDs of the board as shown in Table 7.

Table 7. Output Signals

Signal name	Pin # of J2 connector	LED number
Sync_Detect	5	LED1
Packet_Valid	6	LED2
Received data	7	LED3
Inverted received data	8	LED4

Simplicity Studio

One-click access to MCU tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

www.silabs.com/simplicity



MCU Portfolio
www.silabs.com/mcu



SW/HW
www.silabs.com/simplicity



Quality
www.silabs.com/quality



Support and Community
community.silabs.com

Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

<http://www.silabs.com>