FEC Implementation

By Robin Hoel

Keywords

- *CC1100*
- *CC1101*
- *CC1110*
- *CC1111*
- *CC1150*
- *CC2500*
- *CC2510*
- *CC2511*
- *CC2550*
- *FEC*
- *Viterbi*
- *Trellis*

1 Introduction

This document gives an overview of the FEC implementation in the CC1100, CC1101, CC1110, CC1111, CC1150, CC2500, CC2510, CC2511, and CC2550.

Table of Contents

2 Abbreviations

FEC	Forward Error Coding
FSM	Finite State Machine

3 What is FEC?

Forward Error Correction (FEC) is a technique that allows the receiver to correct a certain amount of errors in the received message. This is achieved by letting a FEC encoder add redundancy to the data message at the transmitter according to certain prescribed rules. The FEC decoder at the receiver uses the knowledge of these rules to identify and, if possible, correct any errors that have appeared. Broadly speaking there are two main classes of FEC: linear block codes (BCH, Reed-Solomon, etc) and convolutional codes.

An (*n*,*k*) linear block encoder takes *k*-bit block of message data and appends *n*-*k* redundant bits algebraically related to the *k* message bits, producing a *n*-bit code block. There are 2^k valid code words, which is far less than the 2^n possible code words, and a good linear block code is one in which the *minimum distance* d_{min}, the minimum number of bits that must be changed to go from any one code word to any other code word, is maximized. In order to be able to correct *e* erroneous bits we have that $d_{min} > 2e$, i.e. after *e* erroneous bits the correct code word is still the one with the smallest distance to the received code word. The dimensionless ratio *r* = *k*/*n* is called the code rate.

A convolutional encoder is fundamentally a finite state machine with a *k*-bit input and *n*-bit output, *n*>*k*, and an internal *M*-bit memory. An important parameter of the convolutional encoder is its *constraint length L* = *M* + 1 which specifies over how many consecutive *n*-bit output periods a *k*-bit input value affects the output. The FSM is such that any given message sequence input results in a coded output sequence which maximizes the minimum distance to what would be generated for any other input message sequence. Convolutional decoding is usually performed by the Viterbi algorithm, which, conceptually, compares the received sequence to the encoded version of all possible encoder input sequences and keeps tab of how close of a match each is. Periodically, the Viterbi algorithm, tracks back through its memory and outputs part of the input sequence, which when encoded is the closest match to the received code sequence.

4 How is FEC Implemented?

The CC1100/CC1101/CC1110/CC1111/CC1150/CC2500/CC2510/CC2511/CC2550 all have a rate *r* = 1/2 convolutional, non-recursive encoder with constraint length *L* = 4 (*M* = *3)*, implemented as shown in Figure 1.

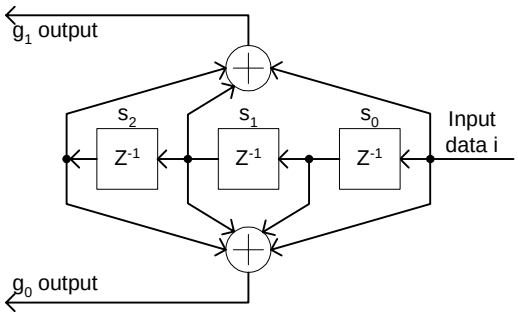

Figure 1. Implementation of Convolutional Encoder

Each input bit is encoded into two output bits, thus doubling the amount of data that must be transmitted. If the same radio data rate is used, error-free reception with a lower signal strength is possible – thus effectively the range of the radio has been increased or the power consumption can be decreased for a fixed range. If the same raw data rate is required, the radio data rate must be doubled either by doubling the modulation rate or going from a 2-ary to a 4-ary modulation format. Obviously, this will increase the number of bit errors in the received coded sequence, but the error-correction in the decoder ensures that the decoded message sequence contains less erroneous bits than if the message sequence had been transmitted without coding.

Convolutional coding works best if the erroneous bits are evenly (or at least randomly) spaced throughout the received coded sequence. Unfortunately, due to the bursty nature of many radio interference sources and the characteristics of the demodulator, it is more likely that erroneous bits will clump together. To combat this problem, so-called *interleaving* of the coded data is performed after encoding in the transmitter and *de-interleaving* before decoding in the receiver. The purpose of interleaving is to make sure that adjacent symbols in the coded sequence are spaced out in the transmitted sequence, so that any clumps of bit errors in the received sequence are spread out more uniformly by the de-interleaver, letting the decoder work under optimum conditions. Our chips employ a 4x4 matrix interleaver with 2 bits (one encoder output symbol) per cell.

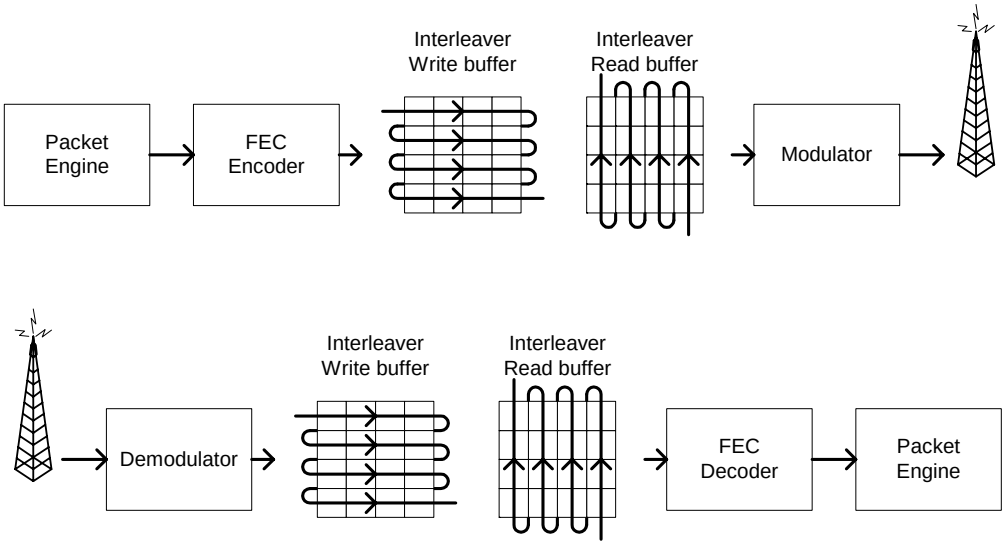

Figure 2. FEC and Interleaving

The decoder in the chip implements a Viterbi algorithm that works on 8-bit soft-decision values from the demodulator. The Viterbi algorithm conceptually compares the received coded sequence with the encoded sequences resulting from encoding of all possible input message sequences, calculates a deviation value for each and then selects the most likely one (the one with the lowest deviation).

A section of the so-called *trellis* shown in Figure 3 is useful in understanding how the Viterbi algorithm works. The circles at the left and right represent the possible values of the encoder state at any given time and the lines between them represent possible transitions from any one state to another. The numbers written along the transition lines are, the input bit to the encoder and the resulting output bits from the encoder, respectively.

In practice, the Viterbi algorithm manages to explore all possible input sequences to the encoder by keeping track of only a finite number of paths through the trellis (corresponding to certain sequences of input bits). Specifically, the Viterbi algorithm only keeps track of the most probable of all paths that end in each of the 2^M encoder states, and the accumulated deviation or cost of that path.

For each input symbol, all possible encoder output symbols (00, 01, 10, and 11 in Figure 3) are compared against the received symbol and a transition cost is calculated. The appropriate transition cost is added to the accumulated path cost of each path that terminates in the source state on the left in the figure. It can be seen that there are two transitions into each destination state on the right in the figure. For each destination state the incoming transition with the lowest accumulated path cost is selected (the survivor path) and the other one thrown away – nothing is lost as all future paths that go through this state at this point in

the trellis would do the same selection. Thus the number of paths that the Viterbi Algorithm tracks is always constant and the optimal path is always one of them.

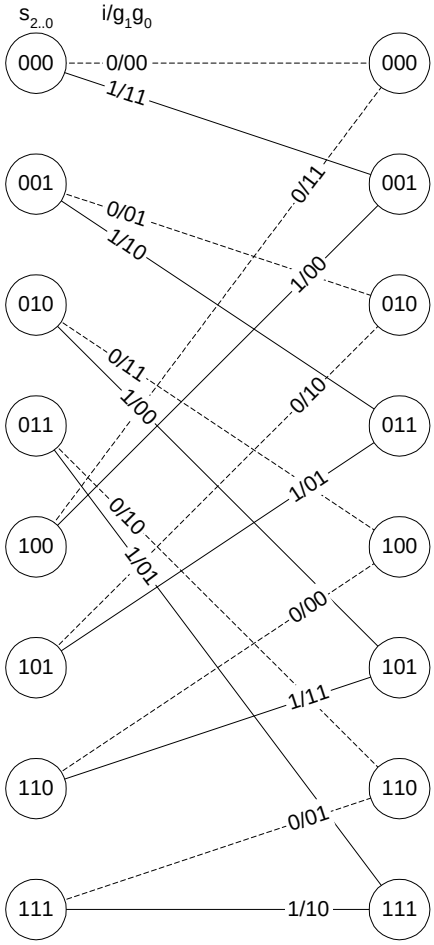

Figure 3. Trellis Diagram

In order to provide the end of the transmitted data, which do not get the benefit of being evaluated over a full trellis path, with a protection equal to that of the rest of the transmitted data, so-called *trellis termination* is necessary. Terminating the trellis means to transmit extra data that brings the convolutional encoder to a known state (usually all zero) so that the decoder doesn't need to make a decision on the most-probable trellis path with limited history. At the end of the transmitted data it is also necessary to fill up the last interleaver buffer with something so that a full interleaver block can be transmitted.

Our FEC implementation appends "00001011_b" to the data input to the encoder/interleaver when an odd number of data bytes are transmitted and "00001011 00001011_b" when an even num ber of data bytes are transmitted. The first three zeros of these sequences are used to terminate the trellis and the rest are used to fill up the last interleaver block. (The reason that not all zeros are transmitted is to ensure that there are some symbol transitions in the output of the interleaver to facilitate clock recovery.)

5 How Many Bit Errors Can FEC Correct?

The convolutional code (optionally) employed in our chips has a maximum free distance of d_{free}= 6 bits. This means that changing any one bit in the message sequence will change at least 6 bits in the coded output sequence. Correspondingly, at least three erroneous bits are required in the received coded sequence before any other message sequence than the correct one is equally likely or more likely.

The ability to correct two bits in the entire coded sequence may not sound like much, but of course this is not the whole story. Normally, when there are no bit errors in the received sequence, the correct path through the trellis will have a much lower cost than all other possible paths and these alternative paths will thus quickly die out. This situation is illustrated in Figure 4 (for a different convolutional code than that employed in the CC11xx/CC25xx having fewer states for illustrative clarity):

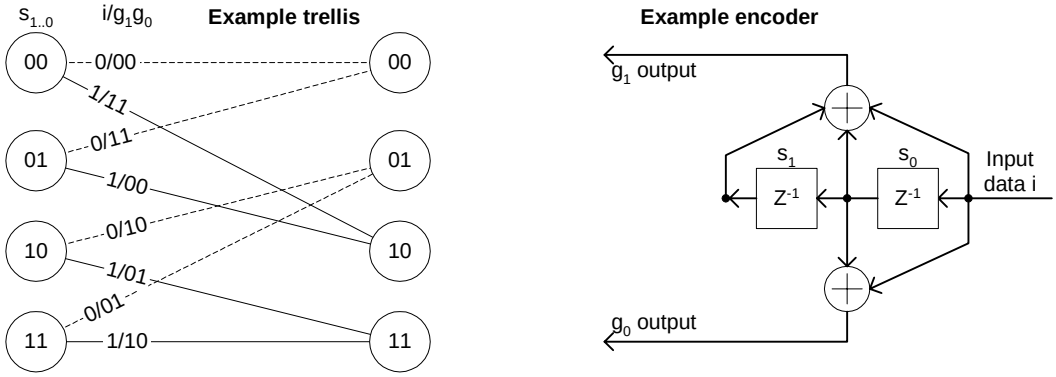

Figure 4. Example Trellis and Example Encoder

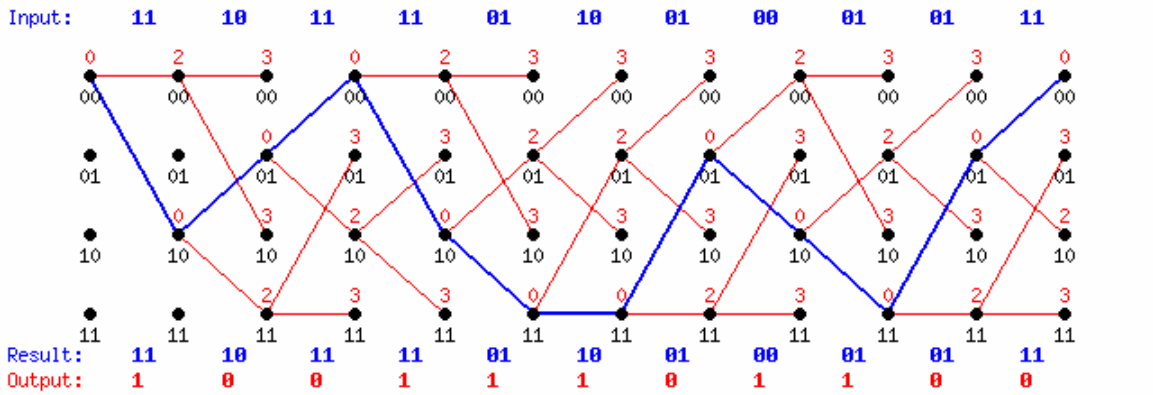

Figure 5. Trellis 1

The numbers above each state node is the cost in erroneous bits in the received sequence (this assumes a binary symmetric channel and hard decoding). It can be seen that most alternative paths quickly disappear since their cost become prohibitively high. If we introduce an error in the received sequence (input) as shown below we see that the cost of the alternative path(s) are much closer to the cost of the correct path and are thus longer-lived.

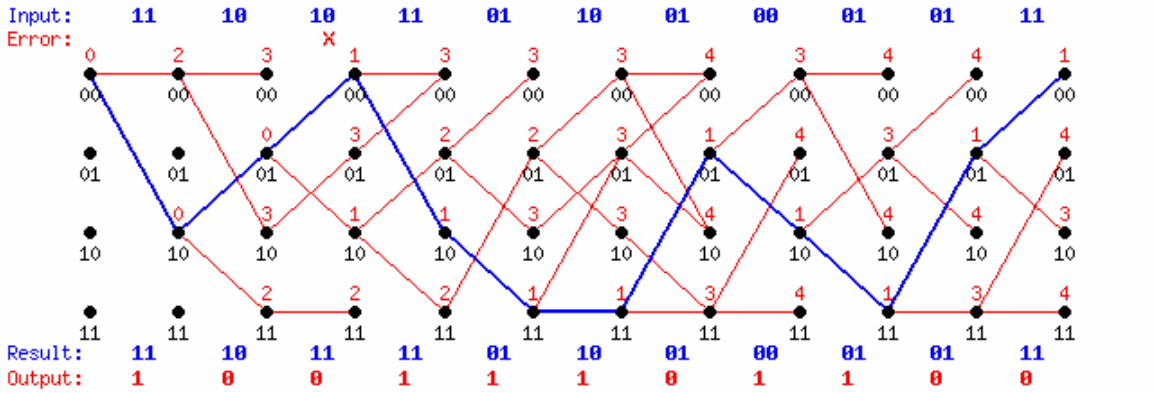

Figure 6. Trellis 2

If we were to introduce another bit error in the received sequence close to the first one, this effect would be even more pronounced:

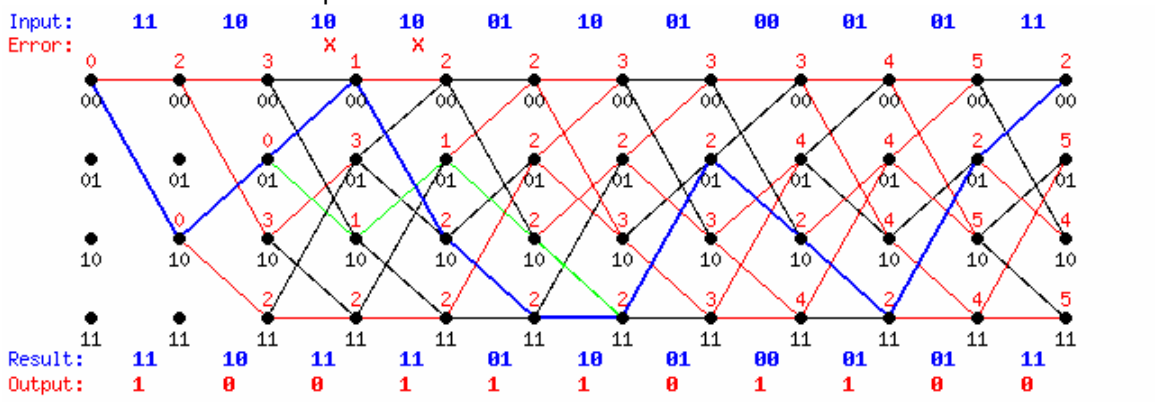

Figure 7. Trellis 3

We see that this time there exists an alternative path that originates in state 10 at the point of the first error (marked in green) which for time afterwards has the same (or even lower) cost as the correct path. The culled (non-surviving) transitions are also shown in black to illustrate at which point the correct path and the alternative path merge. The convolutional encoder employed in CCxx00 can tolerate one additional bit error in the received sequence for the life-span of such an alternative path (from when the two paths split at the first bit error until they meet again at the same state sometime later). If a third bit error was to occur during this time, the alternative path instead of the optimum path might be the survivor path upon merge. We introduce a third error after the two paths merge to demonstrate the principle:

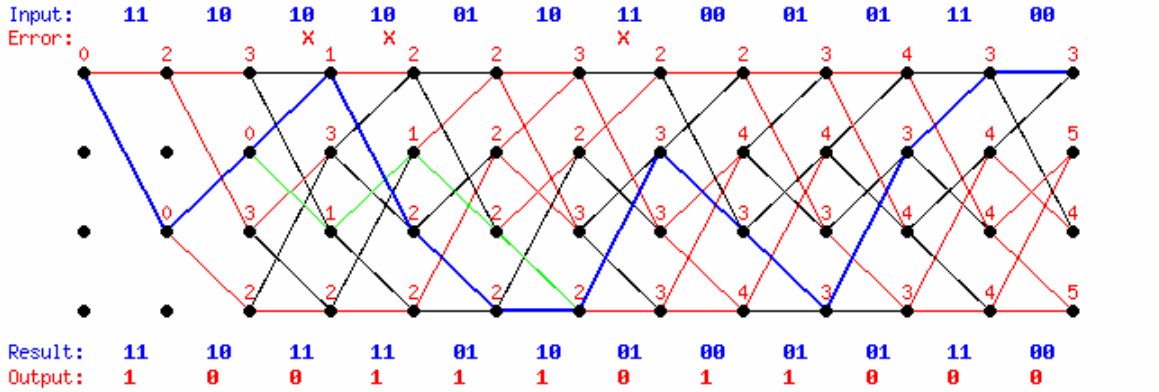

Figure 8. Trellis 4

The exact life-span of each such alternative path is dependant on the input data and the state in which the alternative and correct paths split. As a rule of thumb one could say that they usually merge again within 3*L* (constraint lengths). The interleaver will help in distributing clumps of erroneous bits, which often occur in real-world received data, further part. Due to the inability to precisely predict how many erroneous bits can be corrected by a convolutional coder, the figure of merit usually associated with a convolutional code is its *asymptotic coding gain*, i.e. the reduction of the SNR of the received signal that yields an equivalent BER as the un-coded case. This can be used to increase range or decrease power in the transmitter. For a binary-input AWGN channel (relevant for 2-ary modulation formats on the CCxx00) the asymptotic coding gain is:

$$G_a = 10\log_{10}\left(d_{free}r\right)\,\mathrm{dB}\,,$$

where d_{free} is the free distance of the code and r is the code rate. The used code (d_{free}= 6, r = 1/2) has an asymptotic coding gain of 4.8 dB, although the achievable gain is considerable less for binary modulation formats (perhaps 2-3 dB).

6 FEC Implementation

```
//---------------------------------------------------------------------------
UINT16 culCalcCRC(BYTE crcData, UINT16 crcReg) {
    UINT8 i;
    for (i = 0; i < 8; i++) {
        if (((crcReg & 0x8000) >> 8) ^ (crcData & 0x80))
            crcReg = (crcReg << 1) ^ 0x8005;
        else
            crcReg = (crcReg << 1);
        crcData <<= 1;
    }
    return crcReg;
}// culCalcCRC
//---------------------------------------------------------------------------
// Variables
UINT16 xdata fecEncodeTable[] = {
    0, 3, 1, 2,
    3, 0, 2, 1,
    3, 0, 2, 1,
    0, 3, 1, 2
};
UINT16 input[260];
UINT16 i, j, val, fecReg, fecOutput;
UINT32 intOutput;
UINT16 fec[520];
UINT16 interleaved[520];
UINT16 inputNum = 0, fecNum;
UINT16 checksum;
UINT16 length;
//---------------------------------------------------------------------------
//Example code
length = 3;
input[0] = length;
input[1] = 1;
input[2] = 2;
input[3] = 3;

inputNum = length + 1;

printf("Input: [%5d bytes]\n", inputNum);
for (i = 0; i < inputNum; i++)
    printf("%02X%s", input[i], (i % 8 == 7) ? "\n" : (i % 2 == 1) ? "  " : " ");
printf("\n\n");

// Generate CRC
checksum = 0xFFFF;  //Init value for CRC calculation
for(i = 0; i <= input[0]; i++)
    checksum = culCalcCRC(input[i], checksum);
input[inputNum++] = checksum >> 8;     // CRC1
input[inputNum++] = checksum & 0x00FF; // CRC0

printf("Appended CRC: [%5d bytes]\n", inputNum);
for (i = 0; i < inputNum; i++)
    printf("%02X%s", input[i], (i % 8 == 7) ? "\n" : (i % 2 == 1) ? "  " : " ");
printf("\n\n");

// Append Trellis Terminator
input[inputNum] = 0x0B;
input[inputNum + 1] = 0x0B;

fecNum = 2*((inputNum / 2) + 1);

printf("Appended Trellis terminator: [%5d bytes]\n", fecNum);
for (i = 0; i < fecNum; i++)
    printf("%02X%s", input[i], (i % 8 == 7) ? "\n" : (i % 2 == 1) ? "  " : " ");
printf("\n\n");
```

```
// FEC encode
fecReg = 0;
for (i = 0; i < fecNum; i++) {
    fecReg = (fecReg & 0x700) | (input[i] & 0xFF);
    fecOutput = 0;
    for (j = 0; j < 8; j++) {
        fecOutput = (fecOutput << 2) | fecEncodeTable[fecReg >> 7];
        fecReg = (fecReg << 1) & 0x7FF;
    }
    fec[i * 2] = fecOutput >> 8;
    fec[i * 2 + 1] = fecOutput & 0xFF;
}

printf("FEC encoder output: [%5d bytes]\n", fecNum * 2);
for (i = 0; i < fecNum * 2; i++)
    printf("%02X%s", fec[i],  (i % 16 == 15) ? "\n" : (i % 4 == 3) ? "  " : " ");
printf("\n\n");

// Perform interleaving
for (i = 0; i < fecNum * 2; i += 4) {
    intOutput = 0;
    for (j = 0; j < 4*4; j++)
        intOutput =
       (intOutput << 2) | ((fec[i +(~j & 0x03)] >> (2 * ((j & 0x0C) >> 2))) & 0x03);

    interleaved[i]      = (intOutput >> 24) & 0xFF;
    interleaved[i + 1]  = (intOutput >> 16) & 0xFF;
    interleaved[i + 2]  = (intOutput >> 8)  & 0xFF;
    interleaved[i + 3]  = (intOutput >> 0)  & 0xFF;
}

printf("Interleaver output: [%5d bytes]\n", fecNum * 2);
for (i = 0; i < fecNum * 2; i++)
    printf("%02X%s", interleaved[i],  (i % 16 == 15) ? "\n" : (i % 4 == 3) ? "  " : " ");
printf("\n\n");
```

Running this code will give the following result (all data in hexadecimal base):

Input: [4 bytes]
03 01 02 03

Appended CRC: [6 bytes]
03 01 02 03 30 3A

Appended Trellis terminator: [8 bytes]
03 01 02 03 30 3A 0B 0B

FEC encoder output: [16 bytes]
00 0E 8C 03 7C 0D F0 0E 82 8C 0E 5E F0 D1 8C D1

Interleaver output: [16 bytes]
C8 3C 00 20 84 CF 33 31 A2 FC 40 4A 44 30 47 EF

To test this FEC encoder program one can transmit its output data from one device (with FEC disabled) and recover the original input from a receiving device with FEC enabled. It is also possible to deliberately insert errors in the transmitted sequence to experiment with the error correcting abilities of the code.

For relevant register settings in TX and RX, see table 1 and table 2 respectively.

Transmitter	Comments
PKTCTRL0.LENGTH_CONFIG = 0	Fixed Packet Length
PKTCTRL0.CRC_EN = 0	Disable CRC
MDMCFG1.FEC_EN = 0	Disable FEC
PKTLEN = 0x10	Packet length = 16
TXFIFO = 0xC8, 0x3C, 0x00, 0x20 , 0x84, 0xCF, 0x33, 0x31, 0xA2, 0xFC, 0x40, 0x4A, 0x44, 0x30, 0x47, 0xEF	The output from the interleaver

Table 1. TX Settings

Receiver	Comments
PKTCTRL0.LENGTH_CONFIG = 1	Variable Packet Length
PKTCTRL0.CRC_EN = 1	Enable CRC
MDMCFG1.FEC_EN = 1	Enable FEC
RXFIFO = 0x03, 0x01, 0x02, 0x03	The received packet

Table 2. RX Settings

7 General Information

7.1 Document History

Revision	Date	Description/Changes
SWRA113A	2007.10.22	Removed logo from header. Added CC1101 and CC1111
SWRA113	2006.07.31	Initial release.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

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

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

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

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

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

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