Freescale Semiconductor, Inc.
Application Note

Document Number: AN5070
Rev. 0, 07/2015

Implementing Data Whitening and CRC Verification in Software in Kinetis KW01 Microcontrollers

1 Introduction

When establishing communication with third party products there are cases when the data whitening and CRC algorithms available in hardware are not compatible between devices. To overcome this limitation, this application note provides a software-based implementation.

The use case presented, contains a Packet Error Rate (PER) application using the Kinetis KW01 microcontrollers, where IBM data whitening and CRC calculations are implemented in software to comply with the hardware algorithms of third party products.

Contents

2 Abstract

This application note describes a use case in which the KW01 microcontroller interacts with another KW01 MCU (emulating a third party device such as TI's CC1110) by implementing data whitening and CRC computations in software.

Tools required to run this application are the IAR Embedded Workbench for ARM® v7.10 (or newer) and the Freescale Tower System using the TWR-RF development board.

3 Principle of data whitening

In an RF system, transmitted data is grouped into packets. These packets may contain long sequences of 0's and 1's which can introduce a DC bias into the transmitted signal. A radio signal with a DC bias will have a non-uniform power distribution over the occupied channel bandwidth. A DC bias will also introduce data dependencies during normal operation of the demodulator. However, it is optimal for the transmitted data to be random and DC free.

As will be discussed, CCITT data whitening processes the data packets byte-per-byte, whereas IBM data whitening processes the data packets bit-per-bit. The data is whitened using a random sequence during transmit and de-whitened during receive using the same sequence.

NOTE

Two techniques are available in the packet handler: Manchester encoding and data whitening, however, for the purpose of this application note we will focus only in the data whitening implementation available in the Kinetis KW01 devices.

- Only one of the two methods should be enabled at a time.
- For more details on the Manchester encoding option please refer to the MKW01xxRM reference manual (See Section 10, “References.”)

Data whitening is only used when the user’s data has a high correlation of long strings of 0’s and 1’s. If the data is already random then whitening is not required. For example, when a random source generates the transmit data such that the whitened data produces the longer strings of 0’s and 1’s, then it is not required to randomize an already random sequence.

4 Hardware implementation in KW01 MCUs

KW01 microcontrollers support data whitening and CRC verification within the hardware. Understanding the register programming is a pre-requisite for the software implementation presented later. This section provides the CRC register and CCITT data whitening register settings.

4.1 CRC verification and register settings

A cyclic redundancy check (CRC) is often required to confirm the validity of the data received.

Figure 1 provides the register setting to enable the CRC verification. The CRC verification is enabled by setting the *CrcOn* bit in the *RegPacketConfig1(0x37)* register. This function evaluates the integrity of the transmitted signal.

RegPacketConfig1 (0x37)	7	PacketFormat	rw	0	Defines the packet format used: 0 → Fixed length 1 → Variable length
	6-5	DcFree	rw	00	Defines DC-free encoding/decoding performed: 00 → None (Off) 01 → Manchester 10 → Whitening 11 → reserved
	4	CrcOn	rw	1	Enables CRC calculation/check (TX/RX): 0 → Off 1 → On

Figure 1. *RegPacketConfig1* (*CrcOn* field) register

- During transmit—A two-byte CRC checksum is calculated on the payload of the packet and appended to the end of the message.
- During receive—the checksum is calculated on the received payload and compared with the two checksum bytes received. The result of the comparison is stored in bit *CrcOk*, in the transceiver's register *RegIrqFlags2(0x28)*. See Figure 2.

RegIrqFlags2 (0x28)	7	FifoFull	r	0	Set when FIFO is full (i.e. contains 66 bytes), else cleared.
	6	FifoNotEmpty	r	0	Set when FIFO contains at least one byte, else cleared
	5	FifoLevel	r	0	Set when the number of bytes in the FIFO strictly exceeds *FifoThreshold*, else cleared.
	4	FifoOverrun	rwc	0	Set when FIFO overrun occurs. (except in Sleep mode) Flag(s) and FIFO are cleared when this bit is set. The FIFO then becomes immediately available for the next transmission / reception.
	3	PacketSent	r	0	Set in TX when the complete packet has been sent. Cleared when exiting TX.
	2	PayloadReady	r	0	Set in RX when the payload is ready (i.e. last byte received and CRC, if enabled and *CrcAutoClearOff* is cleared, is Ok). Cleared when FIFO is empty.
	1	CrcOk	r	0	Set in RX when the CRC of the payload is Ok. Cleared when FIFO is empty.
	0	LowBat	rwc	-	Set when the battery voltage drops below the Low Battery threshold. Cleared only when set by the user.

Figure 2. *RegIrqFlags2* (*CrcOk* flag) register

By default, when the CRC verification fails then the FIFO is automatically cleared and no interrupt is generated. This filtering function can be disabled via *CrcAutoClearOff* bit and in this case, even if CRC fails, the FIFO is not cleared and only the *PayloadReady* interrupt goes high. Please note that in both cases, the two CRC checksum bytes are removed by the packet handler and only the payload is made available in the FIFO.

RegPacketConfig1 (0x37)	7	PacketFormat	rw	0	Defines the packet format used: 0 → Fixed length 1 → Variable length
	6-5	DcFree	rw	00	Defines DC-free encoding/decoding performed: 00 → None (Off) 01 → Manchester 10 → Whitening 11 → reserved
	4	CrcOn	rw	1	Enables CRC calculation/check (TX/RX): 0 → Off 1 → On
	3	CrcAutoClearOff	rw	0	Defines the behavior of the packet handler when CRC check fails: 0 → Clear FIFO and restart new packet reception. No *PayloadReady* interrupt issued. 1 → Do not clear FIFO. *PayloadReady* interrupt issued.

Figure 3. *RegPacketConfig1* (*CrcAutoClearOff* bit) register

4.2 CCITT data whitening register settings

The KW01 microcontrollers use CCITT[1] data whitening. To enable whitening or de-whitening the user must set the field *DcFree=10* in the transceiver's register *RegPacketConfig1(0x37).*

RegPacketConfig1 (0x37)	7	PacketFormat	rw	0	Defines the packet format used: 0 → Fixed length 1 → Variable length
	6-5	DcFree	rw	00	Defines DC-free encoding/decoding performed: 00 → None (Off) 01 → Manchester 10 → Whitening 11 → reserved

Figure 4. *RegPacketConfig1* (*DcFree* field) register

5 Hardware implementation for application

When communicating with another KW01 device or with a device from another vendor that supports the same data whitening structure in hardware as the KW01 device (CCITT data whitening) the whitening and de-whitening is transparent to the user who must only configure the transceiver registers in hardware to be compliant between devices.

The problem occurs when the devices are not compatible in hardware with the different data whitening structures. For example, one device supports CCITT data whitening (KW01 devices) and a third party device supports IBM data whitening (an alternate whitening technique). In this scenario, a software implementation on one of the devices is the only way to enable compatibility and to whiten or de-whiten the data for successful communication between devices.

5.1 IBM data whitening method

This section describes the IBM data whitening process and provides code and code examples.

1. Consultative Committee for International Telephony (CCITT) is now part of the International Telecommunications Union-Telecommunication Standardization Sector (ITU-T),

5.1.1 IBM data whitening structure

As explained earlier, CCITT data whitening processes data packets byte-per-byte, whereas IBM data whitening processes the data packet bit-per-bit as shown in Figure 5.

The process to whiten the data is similar but the result of the whitening sequence is different. The user must ensure that the same whitening algorithm is used in the devices to be compliant and establish communication.

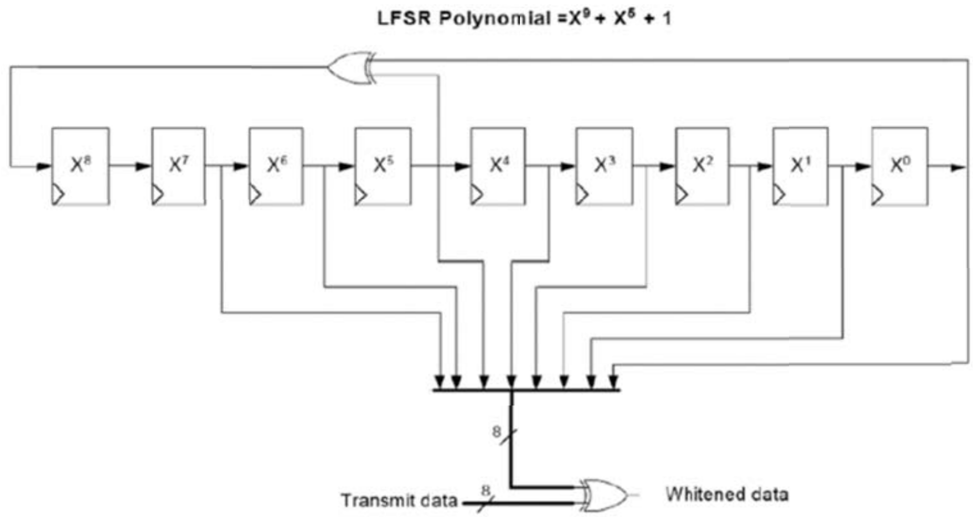

Figure 5. IBM data whitening polynomial

5.1.2 IBM data whitening example

Whitening key:

The initial value of the whitening key is set to all ones (1 1111 1111), this is 0xFF plus a ninth bit that we will call the Most Significant Bit (MSB).

The Least Significant Bit (LSB) or X^0 is XOR-ed with the value of the fifth bit (X^4) to generate the new MSB (refer to the MSB of line 2 in Table 1), then the whitening key is shifted one position to the right. This process counts as 1 loop. The same process must be completed eight times and the result will be the New Whitening Key + the MSB.

Table 1. Whitening key process

MSB (X^8)	X^7	X^6	X^5	X^4	X^3	X^2	X^1	X^0	Counter	Hex Value	—
1	1	1	1	1	1	1	1	1	—	0xFF	Start Key
0	1	1	1	1	1	1	1	1	1	—	—
0	0	1	1	1	1	1	1	1	2	—	—
0	0	0	1	1	1	1	1	1	3	—	—
0	0	0	0	1	1	1	1	1	4	—	—
1	0	0	0	0	1	1	1	1	5	—	—

Table 1. Whitening key process (continued)

MSB (X^8)	X^7	X^6	X^5	X^4	X^3	X^2	X^1	X^0	Counter	Hex Value	—
1	1	0	0	0	0	1	1	1	6	—	—
1	1	1	0	0	0	0	1	1	7	—	—
1	1	1	1	0	0	0	0	1	8	0xE1	New key

When this process is followed it will provide the following whitening keys.

Table 2. IBM whitening keys

Byte Number	Whitening Key
0	0xFF
1	0xE1
2	0x1D
3	0x9A

Let's assume that we have a four byte payload that we want to whiten as shown in Table 3.

Table 3. Example 1 raw data (un-whitened)

Address	Data
0	0x11
1	0x22
2	0x33
3	0x44

The payload (Table 3) XOR-ed with the whitening keys (Table 2) will produce the whitened data that will be transmitted.

Table 4. Example IBM whitened data results

Byte Number	Whitening Key	Data	Whitened Data (XOR-ed)	De-Whitened Data
0	0xFF	0x11	0xEE	0x11
1	0xE1	0x22	0xC3	0x22
2	0x1D	0x33	0x2E	0x33
3	0x9A	0x44	0xDE	0x44

5.1.3 IBM data whitening software implementation

The IBM data whitening software implementation is presented in Figure 6. This implementation can be used to either whiten or de-whiten the data.

```c
#include "Datawhitening.h"

static uint8_t WhiteningKeyMSB = 0x01;                                   //Global variables so the value is kept after starting the de-whitening process
static uint8_t WhiteningKeyLSB = 0xFF;                                   //Init values for the LFSR, at the start of a whitening or a de-whitening process

/****************************************************************************
*buffer is a char pointer indicating the data to be whiten / de-whiten
* buffersize is the number of char to be whiten / de-whiten
* >> The whitened / de-whitened data are directly placed into the pointer
****************************************************************************/

void RadioComputeWhitening( uint8_t *buffer, uint16_t bufferSize )       //IBM Whitening
{
  uint8_t i = 0;
  uint16_t j = 0;
  uint8_t WhiteningKeyMSBPrevious = 0;                                   //9th bit of the LFSR

  for( j = 0; j < bufferSize; j++ )                                      //byte counter
  {
      buffer[j] ^= WhiteningKeyLSB;                                      //XOR between the data and the whitening key
      for( i = 0; i < 8; i++ )                                           //8-bit shift between each byte
      {
          WhiteningKeyMSBPrevious = WhiteningKeyMSB;
          WhiteningKeyMSB = ( WhiteningKeyLSB & 0x01 ) ^ ( ( WhiteningKeyLSB >> 5 ) & 0x01 );
          WhiteningKeyLSB= ( ( WhiteningKeyLSB >> 1 ) & 0xFF ) | ( ( WhiteningKeyMSBPrevious << 7 ) & 0x80 );
      }
  }
```

Figure 6. IBM data whitening in software

5.2 CCITT data whitening method

This section describes the CCITT data whitening process and provides code and code examples.

5.2.1 CCITT data whitening structure

CCITT data whitening is available in hardware for the KW01 microcontrollers, however, when a third party device is required to communicate with the KW01 microcontrollers this algorithm must be implemented in software.

The data whitening process is built around a 9-bit Linear Feedback Shift Register (LFSR) used to generate a random sequence. The payload and 2-byte CRC checksum are then XORed with this random sequence as shown in Figure 7. The data is de-whitened on the receiver side by XORing with the same random sequence. This setup limits the number of consecutive 0's or 1's to nine.

Payload whitening or de-whitening is thus made transparent for the user, who continues to provide and retrieve NRZ data to and from the FIFO.

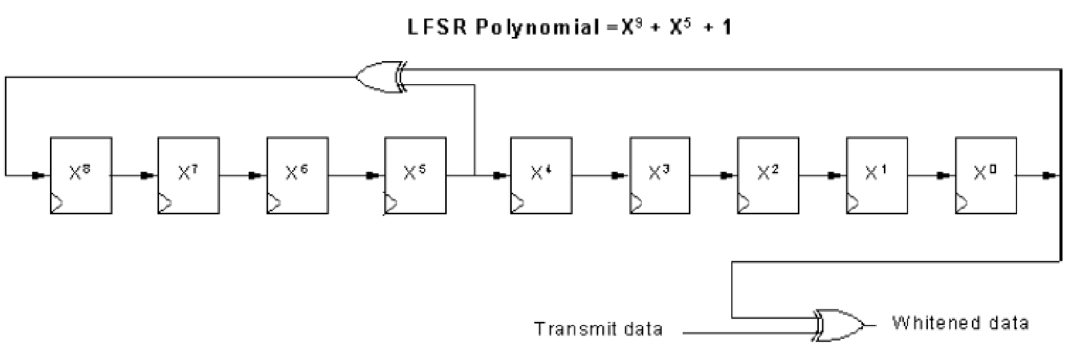

Figure 7. Data whitening polynomial available in hardware (CCITT whitening)

5.2.2 CCITT data whitening example

Whitening key

The LFSR polynomial is the same polynomial as for IBM data whitening ($X^9 + x^5 + 1$), but the whitening process is executed by XORing the LSB at the output of the LFSR with the MSB of the data as shown in Figure 7.

The initial value of the IBM data whitening key is set to all ones (1 1111 1111), this is 0xFF plus a ninth bit (MSB).

When this process is followed, then it will provide the data whitening keys provided in Table 5.

Table 5. CCITT whitening keys

Byte Number	Whitening Key
0	0xFF
1	0x87
2	0xB8
3	0x59

Let's assume that we have the same four-byte payload (as shown in Table 3) that we want to whiten using CCITT data whitening. That provides the whitened data by XOR-ing the CCITT data whitening keys with the data given in Table 6.

Table 6. Example CCITT data whitened results

Byte Number	Whitening Key	Data	Whitened Data (XOR-ed)	De-Whitened Data
0	0xFF	0x11	0xEE	0x11
1	0x87	0x22	0xA5	0x22
2	0xB8	0x33	0x8B	0x33
3	0x59	0x44	0x1D	0x44

5.2.3 CCITT data whitening software implementation

The CCITT data whitening software implementation consists of shifting the LFSR for every new bit of data and to XOR the LSB of the last flip-flop (X^0) with the MSB of the incoming data. This implementation can be used to either whiten or de-whiten the data.

```
void RadioComputeWhitening_CCIT( uint8_t *buffer, uint16_t bufferSize )        //CCIT Whitening
{
uint8_t i = 0;
uint16_t j = 0;
uint8_t WhiteningKeyMSBPrevious = 0;
uint8_t revertedWhiteningKeyLSB = 0;
revertedWhiteningKeyLSB = WhiteningKeyLSB;                                    //WhiteningKeyLSB is 0xFF at init
  for( j = 0; j < bufferSize - 1; j++ )
  {
    buffer[j] ^= revertedWhiteningKeyLSB;
    for( i = 0; i < 8; i++ )
    {
      WhiteningKeyMSBPrevious = WhiteningKeyMSB;
      WhiteningKeyMSB = ( WhiteningKeyLSB & 0x01 ) ^ ( ( WhiteningKeyLSB >> 5 ) & 0x01 );
      WhiteningKeyLSB = ( ( ( WhiteningKeyMSBPrevious << 7 ) & 0x80 | ( WhiteningKeyLSB >> 1 ) & 0xFF ) );
    }
    revertedWhiteningKeyLSB = (WhiteningKeyLSB & 0xF0) >> 4 | (WhiteningKeyLSB & 0x0F) << 4;
    revertedWhiteningKeyLSB = (revertedWhiteningKeyLSB & 0xCC) >> 2 | (revertedWhiteningKeyLSB & 0x33) << 2;
    revertedWhiteningKeyLSB = (revertedWhiteningKeyLSB & 0xAA) >> 1 | (revertedWhiteningKeyLSB & 0x55) << 1;
  }
}
```

Figure 8. CCITT data whitening in software

5.3 CRC validation in hardware

In combination with data whitening, it is common to have a CRC validation at the end of the payload to verify the integrity of the transmission signal—that is, to validate the data received. There are two similar algorithms widely used that enable this verification: IBM-based and CCITT- based CRC.

The KW01 microcontroller’s CRC is based on the CCITT polynomial as shown in Figure 9. This implementation also detects errors due to leading and trailing zeros.

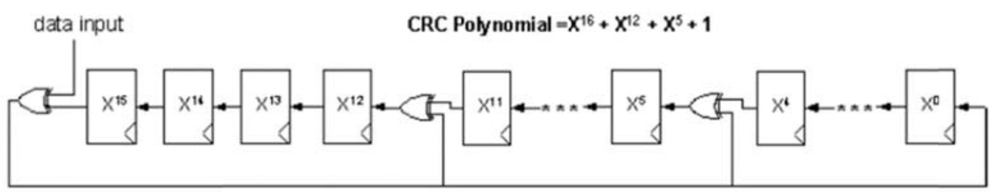

Figure 9. CRC validation in hardware (CCITT polynomial)

This CRC algorithm is enabled in hardware by setting the *CrcOn* bit in *RegPacketConfig1* as shown in Figure 1.

5.4 CRC validation in software

When CRC implementation in hardware is not compatible between devices (for example, one device supports CCITT CRC (KW01 MCUs) and a third party device supports IBM CRC) it is also required to implement the CRC calculation algorithm in software.

To use the CRC algorithm within software the user must disable CRC calculation in hardware, this is done by setting the *CrcOn* bit to zero in *RegPacketConfig* register.

The algorithm to calculate CRC by software is shown in Figure 10.

```
uint16_t RadioComputeCRC( uint8_t *buffer, uint8_t length, uint8_t crcType )
{
 uint8_t i = 0;
 uint16_t crc = 0;
 uint16_t polynomial = 0;
 polynomial = ( crcType == CRC_TYPE_IBM ) ? POLYNOMIAL_IBM : POLYNOMIAL_CCITT;
 crc = ( crcType == CRC_TYPE_IBM ) ? CRC_IBM_SEED : CRC_CCITT_SEED;
   for( i = 0; i < length; i++ )
   {
    crc = ComputeCrc( crc, buffer[i], polynomial );
   }
   if( crcType == CRC_TYPE_IBM )
   {
    return crc;
   }
   else
   {
    return( ( uint16_t ) ( ~crc ));
   }
}

uint16_t ComputeCrc( uint16_t crc, uint8_t dataByte, uint16_t polynomial )
{
 uint8_t i;
   for( i = 0; i < 8; i++ )
   {
     if( ( ( ( crc & 0x8000 ) >> 8 ) ^ ( dataByte & 0x80 ) ) != 0 )
     {
       crc <<= 1;                                          // shift left once
       crc ^= polynomial;                                  // XOR with polynomial
     }
     else
     {
       crc <<= 1;                                          // shift left once
     }
    dataByte <<= 1;                                        // Next data bit
  }
 return crc;
}
```

Figure 10. CRC algorithm implementation in software

6 Software implementation application pre-configuration

The code for this application note is based on the simple range demo application. The simple range demonstration runs as a standalone application that enables performing dynamic range tests.

The simple demonstration consists of two nodes:

- TX node
- RX node

In addition to the simple range demonstration functionality a Packet Error Rate (PER) test is implemented. The PER test enables the user by way of a serial terminal interface to evaluate the performance of the communication between two devices.

The demo can be run without a serial terminal as a range test demo. However, if the user wants to evaluate PER test then the serial terminal interface is required.

6.1 Packet frames

For this application the radio is configured in Packet Mode (operation mode), with a variable length packet format.

Variable length packet format is selected when the *PacketFormat* bit is set to 1 in the *RegPacketConfig1* register.

RegPacketConfig1 (0x37)	7	PacketFormat	rw	0	Defines the packet format used: 0 → Fixed length 1 → Variable length
	6-5	DcFree	rw	00	Defines DC-free encoding/decoding performed: 00 → None (Off) 01 → Manchester 10 → Whitening 11 → reserved

Figure 11. *PacketFormat* bit in *RegPacketConfig1* register

An illustration of a variable length packet is shown in Figure 12 and contains the following fields:

- Preamble (1010…)
- Sync word (Network ID)
- Length byte
- Optional address byte (Node ID)
- Message data
- Optional 2 bytes CRC checksum

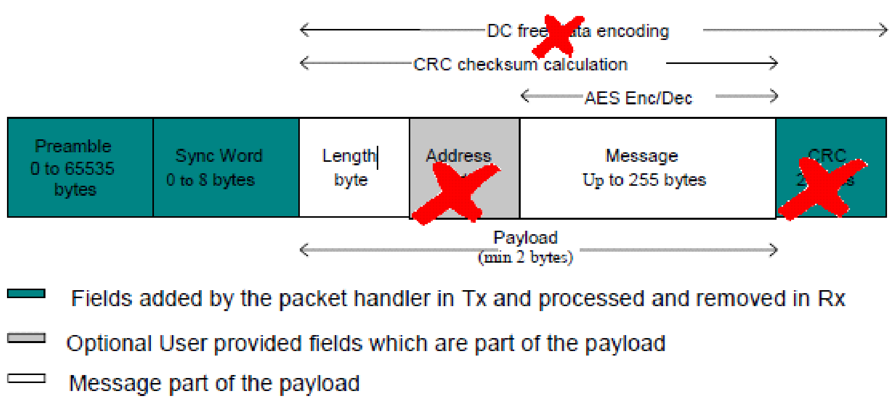

Figure 12. Variable length packet frame used in this application

In this application the default variable length packet frame has been modified as shown in Figure 12. The following list provides a summary of these modifications:

1. Address filtering: To disable this option set *AddressFiltering* bits to 00 in *RegPacketConfig1*.
2. CRC checksum: Provided as part of the payload (not included automatically by hardware). CRC is disabled in hardware by turning off bit *CrcOn* from *PacketConfig1*.
3. DC free (data whitening): Not included because it will be implemented by software. DC free is disabled in hardware by turning off *DcFree* bits from *PacketConfig1*.

RegPacketConfig1 (0x37)	7	PacketFormat	rw	0	Defines the packet format used: 0 → Fixed length 1 → Variable length
	6-5	DcFree	rw	00	Defines DC-free encoding/decoding performed: 00 → None (Off) 01 → Manchester 10 → Whitening 11 → reserved
	4	CrcOn	rw	1	Enables CRC calculation/check (TX/RX): 0 → Off 1 → On
	3	CrcAutoClearOff	rw	0	Defines the behavior of the packet handler when CRC check fails: 0 → Clear FIFO and restart new packet reception. No *PayloadReady* interrupt issued. 1 → Do not clear FIFO. *PayloadReady* interrupt issued.
	2-1	AddressFiltering	rw	00	Defines address based filtering in RX: 00 → None (Off) 01 → Address field must match *NodeAddress* 10 → Address field must match *NodeAddress* or *BroadcastAddress* 11 → reserved
	0	-	rw	0	unused

Figure 13. *DcFree, CrcOn,* and *AddressFiltering* fields in *RegPacketConfig1* register

After applying the modifications as shown in Figure 13, the new packet frame identifies the CRC as now part of the payload.

Preamble	Sync Word	Length	Message	CRC
4 Bytes	4 Bytes	1 Byte	17 Bytes	2 Bytes

Payload

Figure 14. Application packet frame

Preamble:

The preample can be 0x55 or 0xAA, this is required for synchronization, the longer the synchronization the better the packet success rate. At least 12 bits are required for synchronization, in this application we use 4 bytes.

Sync Word (Network ID):

Sync word size can be set from 1 to 8 bytes; in this application we use 4 bytes (0xF4EEF4EE).

Length:

1 byte for data length (0x14). 1 byte for Payload length (part of the payload) + 17 payload bytes + 2 bytes for CRC (required to add CRC functionality by software)

Message:

17 bytes for message. 0x5A + 0xA5 + 4 bytes for Sequence Number (little-endian) + 0xCC+ 0xCC+ 0xCC+ 0xCC+ 0xCC+ 0xCC+ 0xCC+ 0xCC+ 0xCC+ 0xCC+ 0xCC

CRC:

2 bytes for CRC calculation by software.

6.2 Radio settings

The RF default configuration for the application is show in Table 7.

Table 7. Radio settings

Feature	Default Value
Center Frequency	868 MHz
BitRate	50 Kbps
Frequency Deviation	25 KHz
RxBw	135 KHz
Modulation	FSK
Filter	BT_1
Output power	13 dBm
PA_Boost	Disabled (RFIO output)
SMAC Transmission	Broadcast

7 Application procedure

A pre-requisite to implement the software application in the Kinetis KW01 device is the Freescale Tower System and the TWR-RF development board. Also required is the IAR Embedded Workbench by ARM Limited.

A PER test application with CRC and data whitening features by software can be found in AN5070SW. This PER test application is mounted on the simple range demo application.

7.1 Open the application in IAR Embedded Workbench for ARM v7.10 (or newer)

1. Open IAR Embedded Workbench for ARM v7.10.
2. Select *Open* and then *Workspace* from the File menu and locate the folder where the project was extracted as shown in Figure 15.

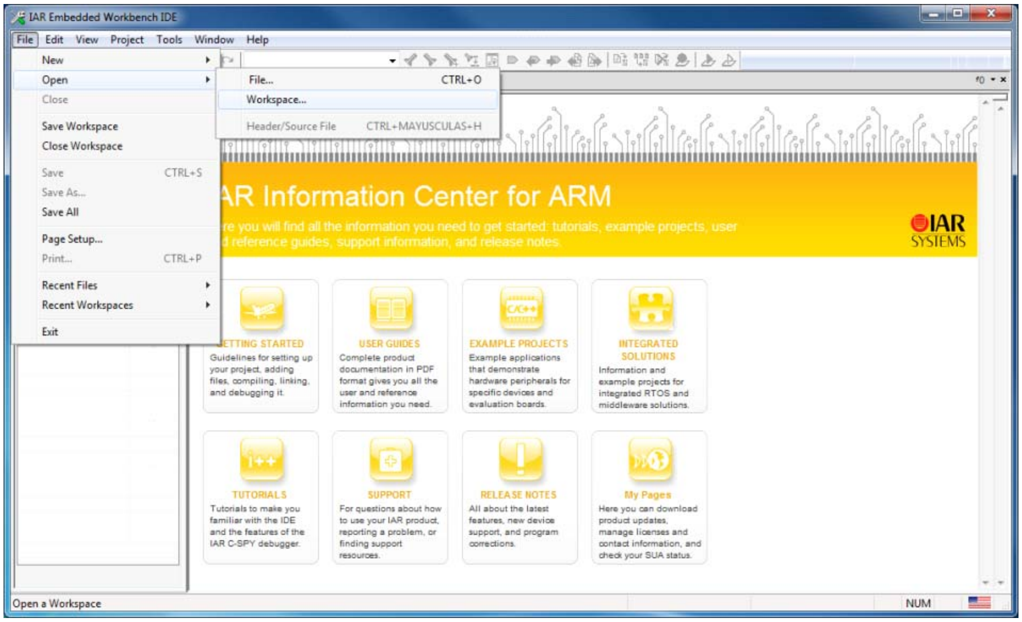

Figure 15. Opening a Workspace in IAR Embedded Workbench

3. Obtain the application project AN5070SW. Extract the project, locate the folder where the application project was extracted.
4. Select the project (SimpleRangeDemo.eww) and drag and drop it into IAR Embedded Workbench workspace as shown in Figure 16.

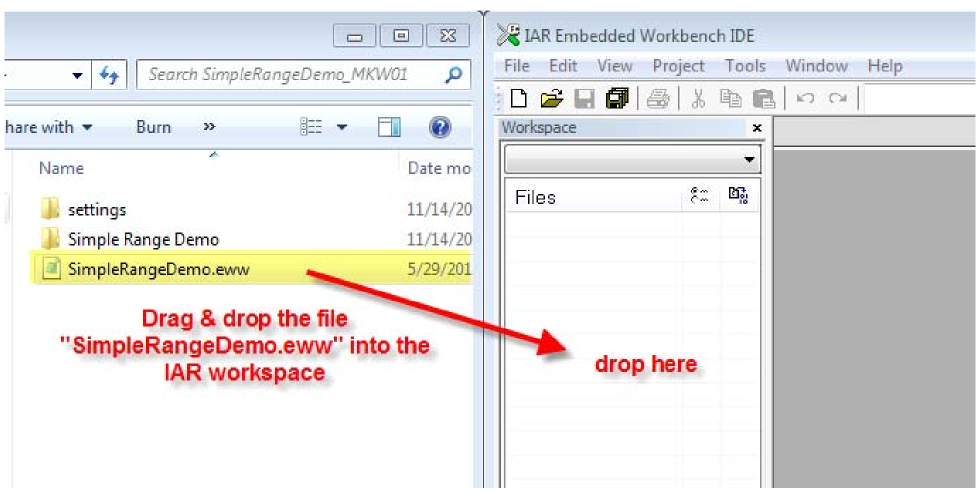

Figure 16. Import the application into IAR workspace

7.1.1 Configure the application (TX and RX nodes)

The device type and settings are configured in the application configuration file (*ApplicationConf.h*).

- Select **gTxNode_c** for the transmitter device. (Figure 17)

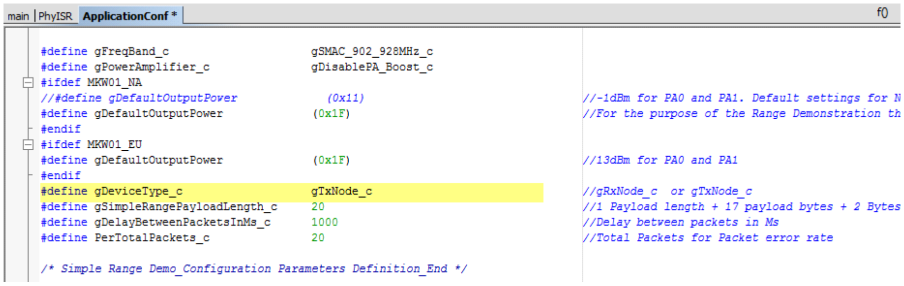

Figure 17. Selecting TX device in software

- Select **gRxNode_c** for the receiver device. (Figure 18)

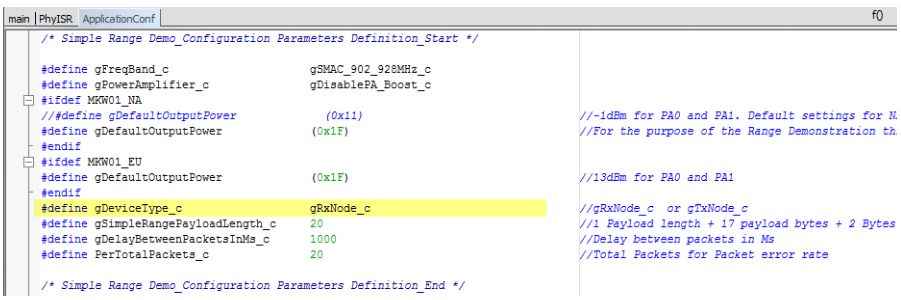

Figure 18. Selecting RX device in software

- Select the total number of packets to send for the PER test by modifying **PerTotalPackets_c**. (Figure 19)

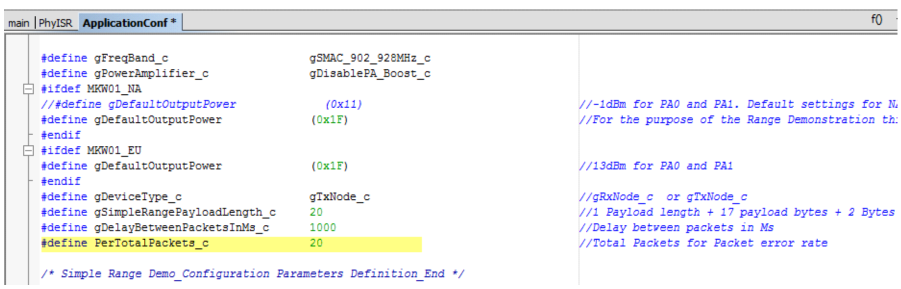

Figure 19. Selecting number of packets for PER test

- Select time delay in ms between packets in **gDelayBetweenPacketsInMs_c**. (Figure 20)

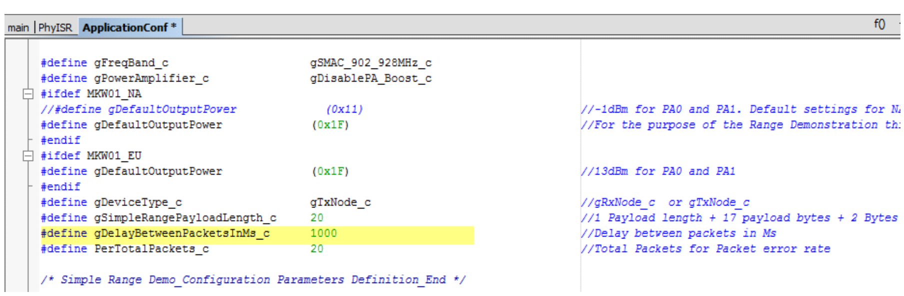

Figure 20. Selecting delay (in mili seconds) between packets

7.1.2 Run the application

To run the application you must connect and configure the transmit and receive nodes as follows.

- On the TX node: the LEDs will flash twice to indicate that this is the TX device.
- On the RX node: the LEDs will flash once to indicate that this is the RX device.

Connect both boards to a serial terminal and configure using the parameters shown in Figure 21.

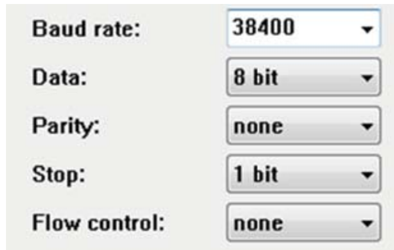

Figure 21. Terminal configuration

After the application begins it displays a message indicating that the application is waiting for the user to press switch 1 in both (TX/RX) devices.

Both devices turn on LED D1 on the Freescale Tower System, TWR-RF board to indicate that the application has begun.

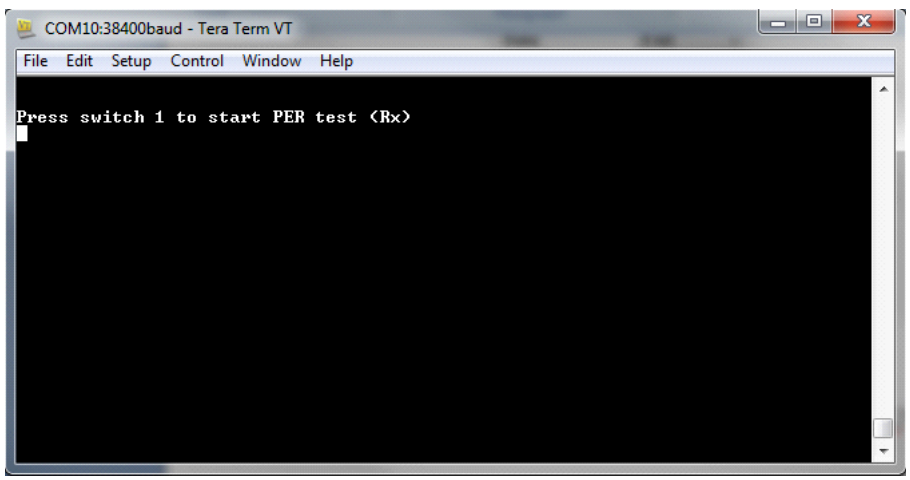

Figure 22. First message in terminal in PER test RX node

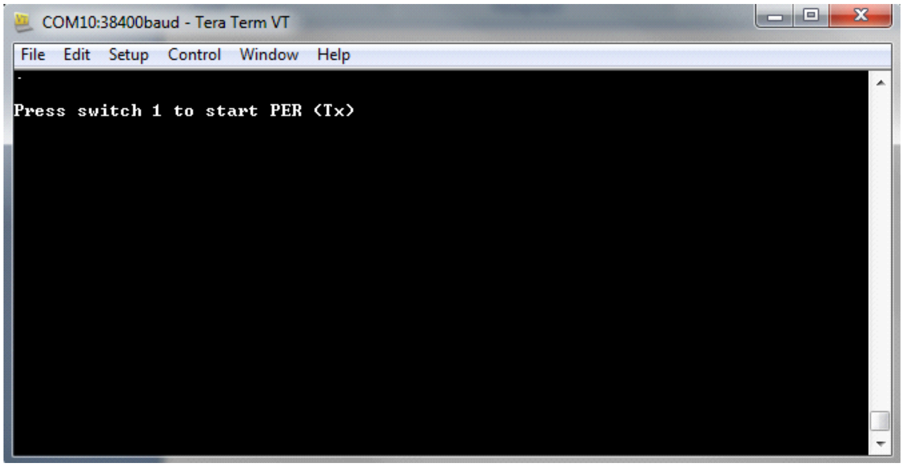

Figure 23. First message in terminal in PER test TX node

Press switch 1 in RX node to begin the receive process.

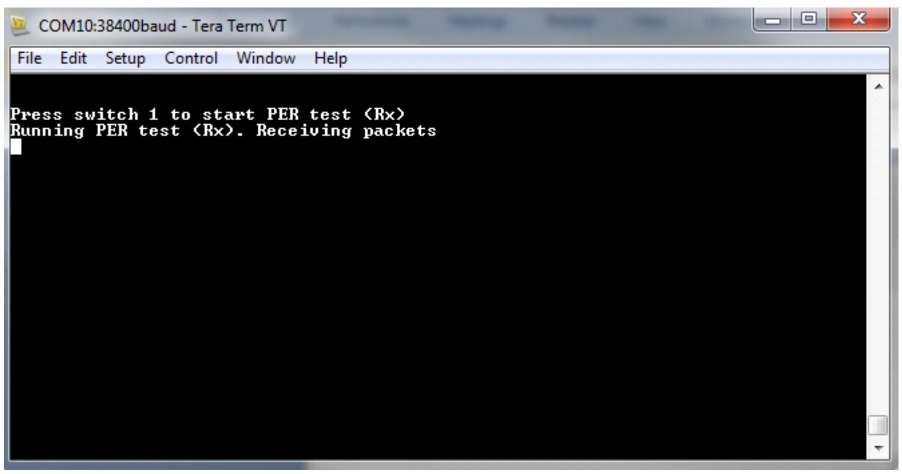

Figure 24. Running message in PER test RX node

Press switch 2 in TX node to begin transmitting packets.

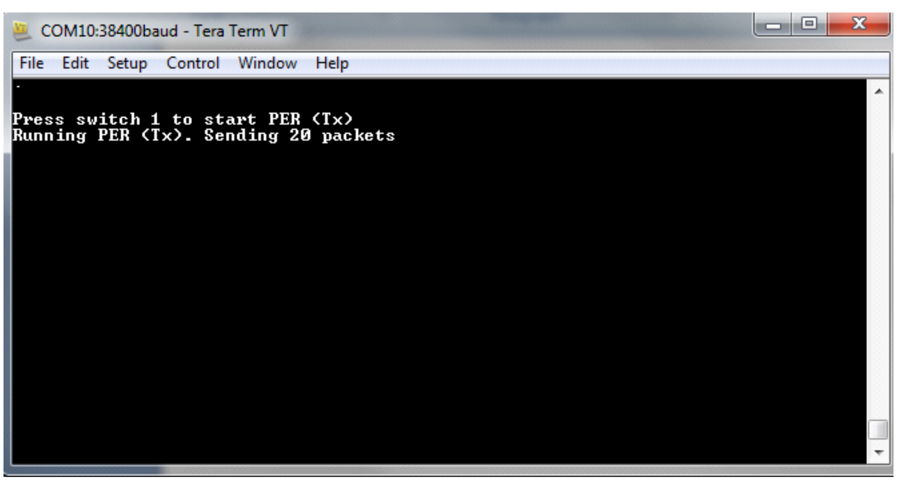

Figure 25. Running message in PER test TX node

7.1.3 Identify application behavior

The application behavior for transmit and receive nodes is identified as follows.

On the TX node: LED D4 from the TWR-RF board flashes when a packet has been transmitted.

On the RX node: LED D4 from the TWR-RF board flashes when a valid CRC is identified, and LED D2 flashes for an invalid CRC.

When the PER test is completed, the results of the test can be viewed in the terminal and the user has the option to restart the test by pressing software switch 1 as shown in Figure 26.

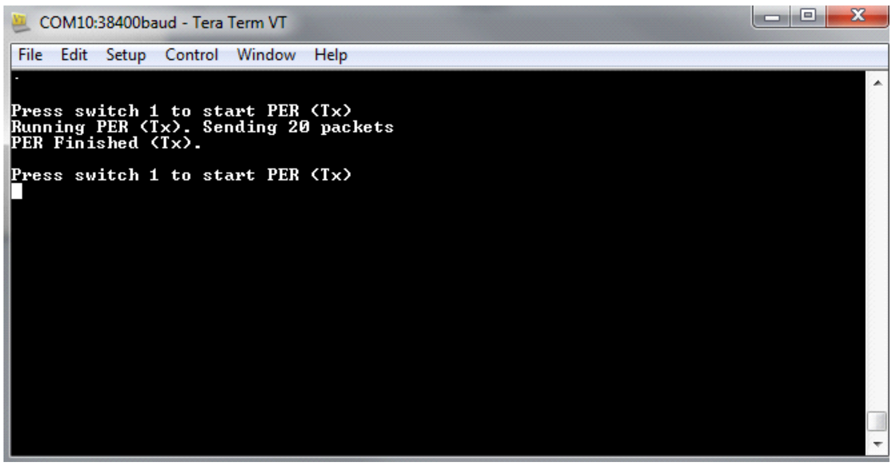

Figure 26. PER test finished message in TX node

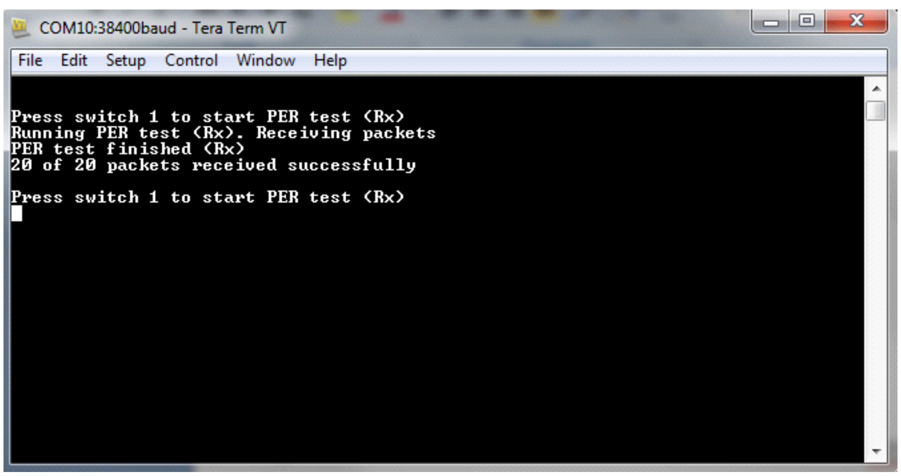

Figure 27. PER test finished message in RX node

8 Summary of software implementation

In brief, to apply data whitening and CRC verification to your application with software use the following the sequences:

For TX:

1. Calculate CRC of the payload starting from the data length byte, but do not include the last two bytes (those are reserved to store the CRC calculation result).
2. Include the CRC calculation to the last two bytes of the payload.
3. Compute data whitening to the entire payload buffer (including the last two CRC bytes).
4. Transmit the message.

For RX:

1. Compute data de-whitening to the reception buffer.
2. Calculate CRC verification to this de-whitened buffer, but do not include the last two CRC bytes.
3. Compare the result of this CRC calculation to the CRC of the reception buffer (last two bytes of this buffer).
4. If the CRC is equal, then the reception process continues to *Successful RX packet*; if the CRC is not equal then the process goes to *CRC Error Indication*.

9 Conclusions

Full compatibility in hardware among products in the market is a rare. To overcome these constraints software implementations such as the data whitening and CRC verification discussed in this document are necessary to enable devices to communicate with each other regardless of the limitation in hardware.

Prior to implementing data whitening and CRC verification by software, the programmer must disable this functionality in the device hardware by writing to the corresponding registers.

10 References

The chapters within this application note summarize the important details of each topic. For more details on the specific topics within this document, see the device reference manual, specifically for information and details about the KW01 transceiver's registers and device hardware capabilities.

- Kinetis *Sub-1 GHz Low Power Transceiver plus Microcontroller Reference Manual* (Document Number: MKW01xxRM)

A PER test application with CRC and data whitening is located at *www.freescale.com* with the filename:

- AN5070SW

11 Revision history

Revision 0 is the initial release of this application note.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN5070
Rev. 0
07/2015