1. Background

The MX909A Wireless Packet Data Pump performs baseband signal processing and Medium Access Control (MAC) protocol functions for a GMSK wireless packet data modem. While the MX909A uses data block sizes and FEC/CRC algorithms compatible with the Mobitex™ network over-air-standard, it also provides flexible operating modes that make it suitable for private and general-purpose applications. The MX909A assembles application data received from the µC, adds forward error correction (FEC), adds an error detection (CRC) code, interleaves this data to provide burst-error protection, and randomizes (scrambles) the bit pattern. After automatically adding bit and frame sync codewords, the data packet is converted to analog levels and Gaussian filtered such that it is suitable for modulating a transmitter VCO (modulation index = 0.5) creating GMSK signals at the transmitter output.

Figure 1, shows the Mobitex™ frame structure.

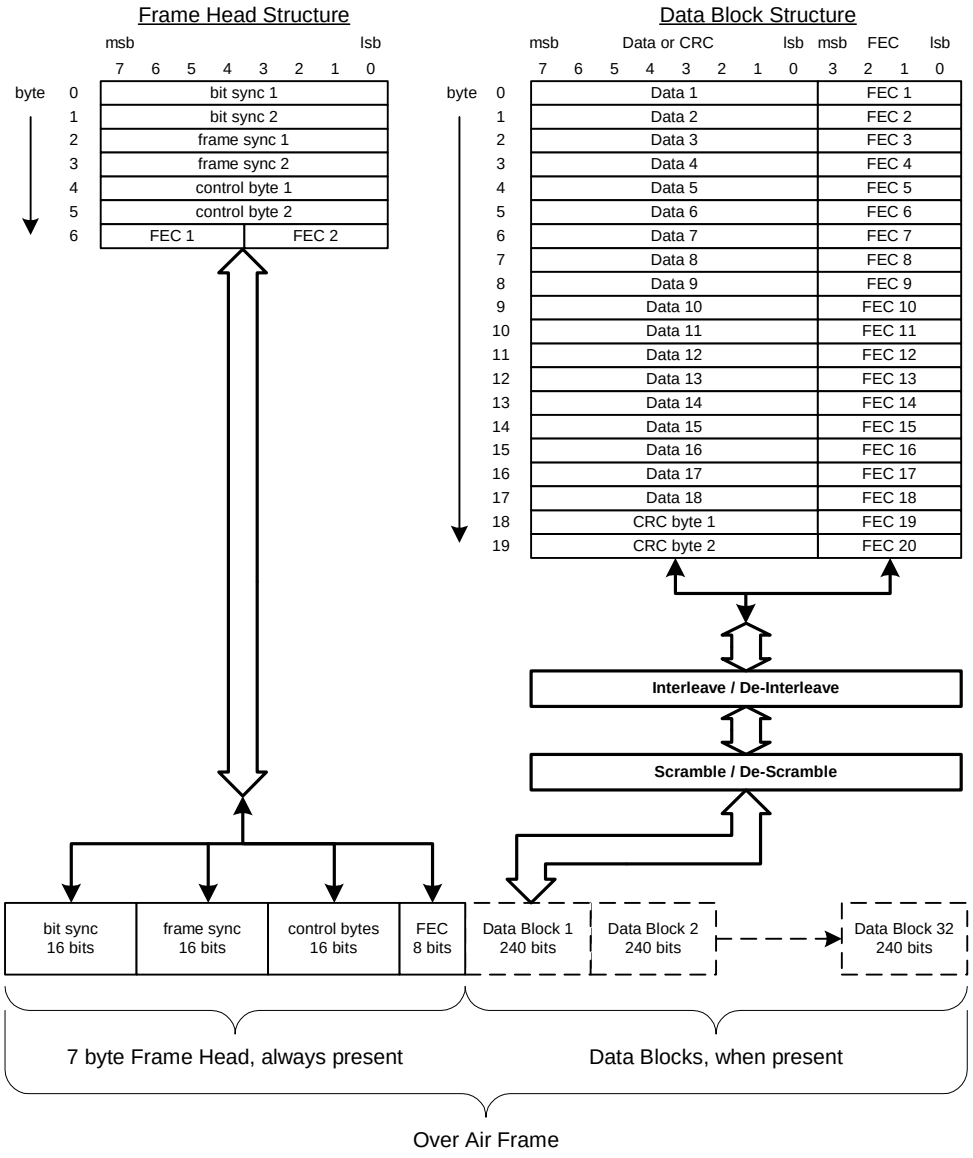

Figure 1: Mobitex™ Over Air Signal Format

Data is transmitted in the form of a Frame Head immediately optionally followed by up to 32 Data Blocks. The frame consists of:

Frame Head (7 bytes)

- 2 bytes of bit Sync (0xCCCC for the down-link and 0x3333 for the up-link)
- 2 bytes of frame Sync (System Specific)
- 2 bytes of control data
- 1 byte of FEC code (4 bits for each of the control bytes)

Data Block (30 bytes)

- 18 bytes of data
- 2 bytes of CRC (calculated from the 18 data bytes)
- 10 bytes of FEC (4 bits of FEC code for each of the data and CRC bytes)
- The resulting 240 bits are interleaved and scrambled before transmission

2. The R14N Short Block Frame Acknowledgement

A latest version of the Mobitex™ Interface Specification incorporates an extended battery saving protocol known as R14N that enhances Mobitex™ systems by supporting new services and levels of performance. The enhancements are in three key areas: better coverage, longer battery life and faster radio link reestablishment.

R14N introduces a new ROSI (**R**adi**O** **Si**gnaling) protocol link layer frame type called the SBF_ACK (Short Block Frame ACKnowledgement). This new frame type is shorter than others and improves base station reception success rate under adverse radio conditions. Another advantage is reduced power consumption used to transmit the shorter frame. R14N allows mobiles to respond using a shorter transmit period, thus increasing battery life by reducing the RF transmitter on period. Use of the shorter frame is only permitted in networks that support the extended battery saving protocol, so portable radio modems must first determine that a network provides support before using the new frame type.

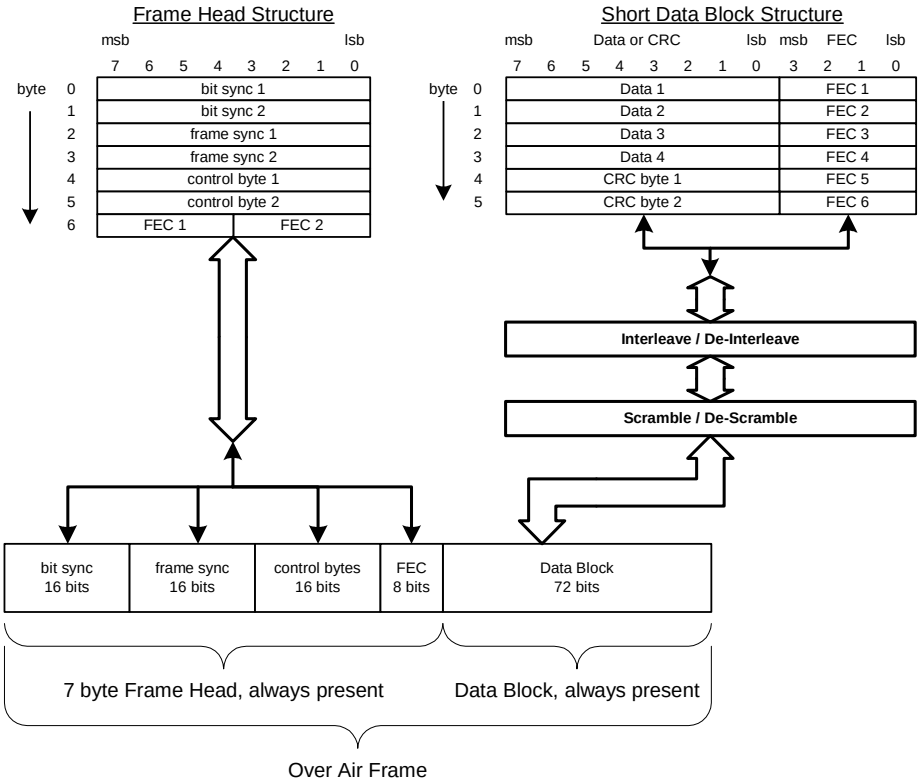

Figure 2: Mobitex™ Short Block Frame Over Air Signal Format

Figure 2, shows the Mobitex™ Short Block Frame. The format of the Short Block Frame differs from all other frames only in the number of bytes contained in the data block and the mandatory presence of one data block. The data block for the short frame would be formed as follows:

Short Data Block (9 bytes)

- 4 bytes of data
- 2 bytes of CRC (calculated from 4 data bytes)
- 3 bytes of FEC code (4 bits for each of the data and CRC bytes)
- The resulting 72 bits are interleaved and scrambled before transmission.

The Short Frame Block Acknowledgement always contains one data block and is only transmitted, never received, by the modem. (Base stations do not transmit SBF_ACKs to the modem.)

3. Implementing the R14N Short Frame on the MX909A

To support the Mobitex™ R14N battery saving protocol, an end product must support the link layer to physical layer conversion for the new Short Block Frame.

Data is transmitted over the air in the form of 'frames' consisting of the 'Frame Head' followed by one or more formatted data blocks. The MX909A constructs the data blocks from the 'raw' data using a combination of CRC (Cyclic Redundancy Checksum) generation, Forward Error Correction coding (FEC), Interleaving and scrambling. The MX909A treats operations such as transmitting or receiving frames (or parts of a frame) as 'tasks', which are initiated by the µC by writing instruction bytes to the command register. A frame, according to Figure 1, would be sent by writing aT7H (Transmit 7 byte Frame Head) task followed by a TDB (Transmit Data Block) task. The TDB 'task' takes 18 bytes of 'raw' data, calculates and applies a 16-bit CRC and forms the FEC for the 18 bytes and the CRC. This data is then interleaved and passed through the scrambler (if enabled) before sending it as the Data Block part of the Mobitex™ frame.

The MX909A does not have a 'task' similar to TDB for the short data block frame. The general-purpose 'tasks' TQB (Transmit 4 Bytes) or TSB (Transmit Single Byte) could be used to transmit the shorter frame. The TQB and the TSB 'tasks' read the data as presented to the modem and transmit them without adding CRC and FEC, and without passing the data through the interleaver and the scrambler. For this reason, the host µC must format the data before presenting it to the MX909A. Section 6 shows a C program that will take 4 bytes of data and generate 9 bytes of formatted data ready for transmission.

4. Transmit Short Frame Example

If the device is required to send a Mobitex™ Short Block Frame, the following control signals and data should be issued to the modem, provided the device is not starting from powersave state. TX//RX is set to '1' and the SCREN, DARA, CKDIV, and DQEN have been set as required after power was applied to the device.

1. 6 bytes forming the Frame Head are loaded into the Data Buffer, followed by a 2-bit pause to let the filter stabilize, followed by a T7H task.
2. Device interrupts host µC with /IRQ when the 6th byte is read from the Data Buffer
3. Status Register is read and the first 4 bytes of formatted data are loaded into the Data Buffer, followed by a TQB task.
4. Device interrupts host µC with /IRQ when 4th byte is read from Data buffer
5. Status Register is read, and the next 4 bytes of formatted data are loaded into the Data Buffer, followed by TQB task.
6. Status Register is read, and the last byte of formatted data is loaded into the Data Buffer, followed by a TSB task.
7. Status Register is read, host may load data and set next task as required:

   GOTO '1' If another Frame is to be immediately transmitted

   GOTO '8' If no more data is to be immediately transmitted.
8. 1 byte representing the 'hang byte' is loaded into the data buffer followed by a TSB task.

Figure 3 shows a top level flowchart of the short block frame transmit process:

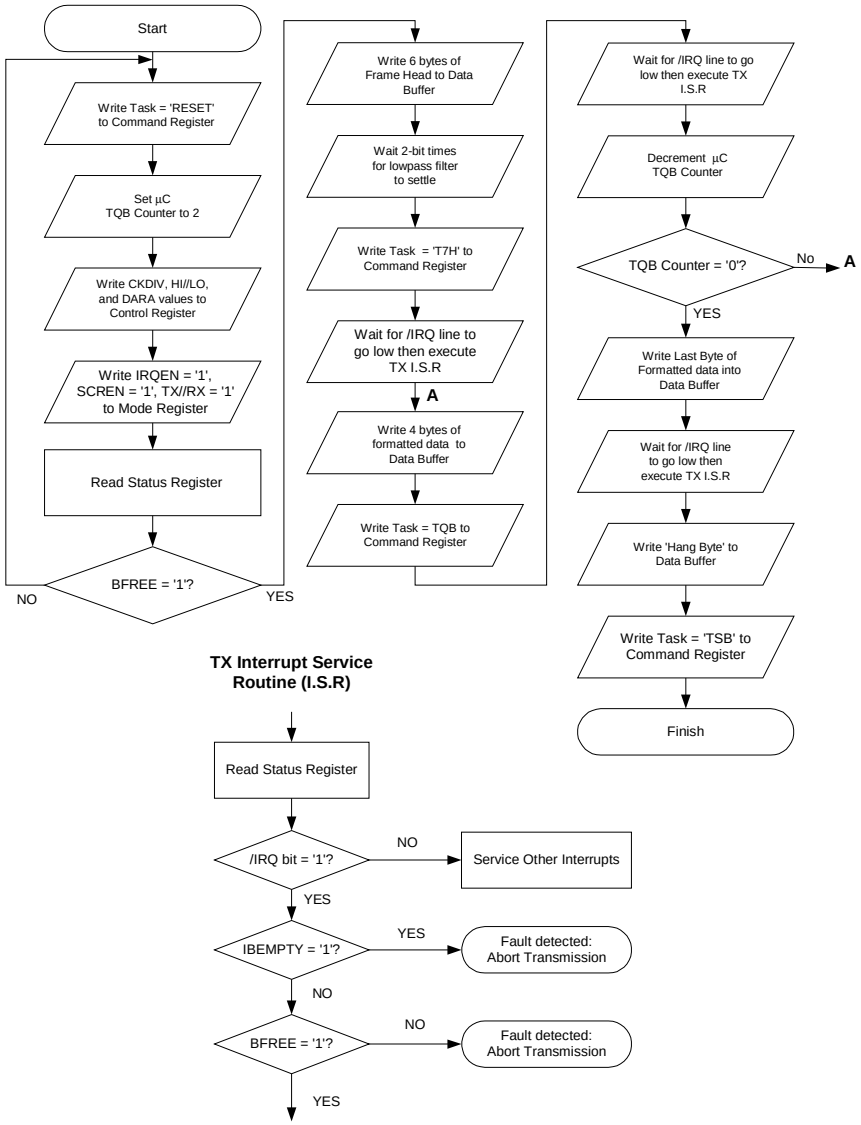

Figure 3: Transmit Short Block Frame Flow Chart

5. Conclusion

This document describes how to implement the functions required for the transmission of the Short Block Frame Acknowledgement as specified in the latest version of the Mobitex™ interface specifications. The flexibility of the MX909A allows this to be done by making use of some of the more general-purpose 'tasks' supported by the device. The implementation requires that the µC assemble the Short Data Block by taking the 4 bytes of 'raw' data, processing the data to add the CRC and FEC codes, interleaving the data for burst error protection and then randomizing the data if the scrambling function is enabled.

6. 'C' language program

```
/* Program takes 4 bytes of 'raw data' and generates 9 bytes */
/* of formatted data by adding CRC, FEC, then interleaving   */
/* and scrambling it (if scrambler enabled)                  */

#include <stdio.h>

int crc_16(int [], int);
int fec_add(int);
void interleave(int[]);                 /* function Declarations */
void scramble_block(int);

int outbits[72];
int bits[72];                    /* Global Variables */

void main(void)
{
int n_bytes = 6;
int data_bytes[6] = {0x09, 0xEB, 0xCA, 0x20, 0, 0}; /*enter 4 'raw data' bytes*/
int crc,i,j,z,w = 0xff;


crc = crc_16(data_bytes, n_bytes-2);           /*CRC generation function call*/
data_bytes[n_bytes - 2] = (crc >> 8) & 0x00ff; /*member 5 of array is CRC MSB*/
data_bytes[n_bytes - 1] = crc & 0x00ff;        /*member 6 of array is CRC LSB*/

for (i = 0; i < n_bytes; i++)
       {
       data_bytes[i] = fec_add(data_bytes[i]); /*FEC generation function call*/
       }

interleave(data_bytes);                        /* interleave function call*/
scramble_block(1);                      /* 1 enables scrambler*/

printf("\n The formatted bytes are:");  /* print the 9 formatted bytes*/
for(j = 0; j < 9; j++)
       {
       for(i = 0; i < 8; i++)
              {
              w = w << 1;
              z = outbits[j*8 + i] & 0x1;
              w = (w | z) & 0xff;
              }
       printf("%4x", w);
       }
}


int crc_16(int data_bytes[], int n_bytes)
        {
       int i, j, fb, mask;
       long crc = 0xffff;               /* CRC generation */
       for(i=0; i < n_bytes; i++)
              {
              mask = 1;                 /* as CCITT X25 */
              for(j = 0; j < 8; j++)          /* X16 + X12 + X5 + 1 */
                     {
                     fb = 0;
                     if((data_bytes[i] & mask) != 0)
                            fb = 1;
                     fb = (fb ^ crc) & 1;
                     crc = (crc >> 1) & 0x7fff;
                     if(fb != 0)
                            crc = crc ^ 0x8408;
                     mask = (mask << 1);
                     }
              }
       return(crc ^ 0xffff);            /* 1s complement result */

}
```



```
/* Add 4 bit FEC to 8 bit data byte. Return 12 bit result */
/* FEC coding matrix. MSB at left */

/* 11101100  1000 */
/* 11010011  0100 */
/* 10111010  0010 */
/* 01110101  0001 */

int fec_add(int x)
        {
       int y, i, b[8];

       y = x;
       for(i = 0; i < 8; i++)
              {
              b[i] = y & 1;
              y = y >> 1;
              }
       y = (x << 4) & 0xff0;
       if((b[7] ^ b[6] ^ b[5] ^ b[3] ^ b[2]) == 1) y |= 0x8;
       if((b[7] ^ b[6] ^ b[4] ^ b[1] ^ b[0]) == 1) y |= 0x4;
       if((b[7] ^ b[5] ^ b[4] ^ b[3] ^ b[1]) == 1) y |= 0x2;
       if((b[6] ^ b[5] ^ b[4] ^ b[2] ^ b[0]) == 1) y |= 0x1;
       return(y);
       }

void interleave(int inbytes[])                      /* Interleave */
       {
       int i, j, crc, data[6], bit, mask;

       mask = 0x800;
       for(i = 0; i < 12; i++)
              {
              for(j = 0; j < 6; j++)
                     {
                     bit = 0;
                     if((inbytes[j] & mask) != 0) bit = 1;
                            outbits[j + 6*i] = bit;
                     }
              mask = mask >> 1;
                     }
       }

void scramble_block(int reset)
        {
       static int sreg;
       int i;

       if(reset != 0)
              sreg = 0x1ff;
       else
              sreg = 0x000;
                     for(i = 0; i < 72; i++)
                            {
                            outbits[i] ^= (sreg & 1);
                            if( ((sreg & 0x11) == 0x10) || ((sreg & 0x11) == 0x01))
                            sreg |= 0x200;
                            sreg = (sreg >> 1) & 0x1ff;
                            }
       }
```

