MPXY8300 Design Reference Manual

Document Number: MPXY8300RM
Rev. 2
12/2008

freescale™
semiconductor

MPXY8300

Design Reference Manual

by: Rudi Lenzen
Freescale Toulouse Systems Laboratories
Toulouse, France

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.freescale.com

The following revision history table summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.

Revision History

Date	Revision Level	Description	Page Number(s)
September, 2008	0	First draft	N/A
October, 2008	1	Corrected speed to ms-1, previously ms to 1. Changed “Z-axis offset shifted by 41” to “Z-axis offset shifted by 51”.	34 37
December, 2008	2	Changed NX3225GA to NX3225DA Updated URL	13, 14 9, 11

Contents

Tables

Figures

Chapter 1
Introduction and Setup

1.1 Introduction

The purpose of this documentation is to provide the end user with all the technical information required to become quickly familiar with the MPXY8300 device.

The tool set of the TPMS kit consists of two MPXY8300 modules: a Freescale ESTAR receiver and an LF 125 KHz transmitter. It also contains their associated software.

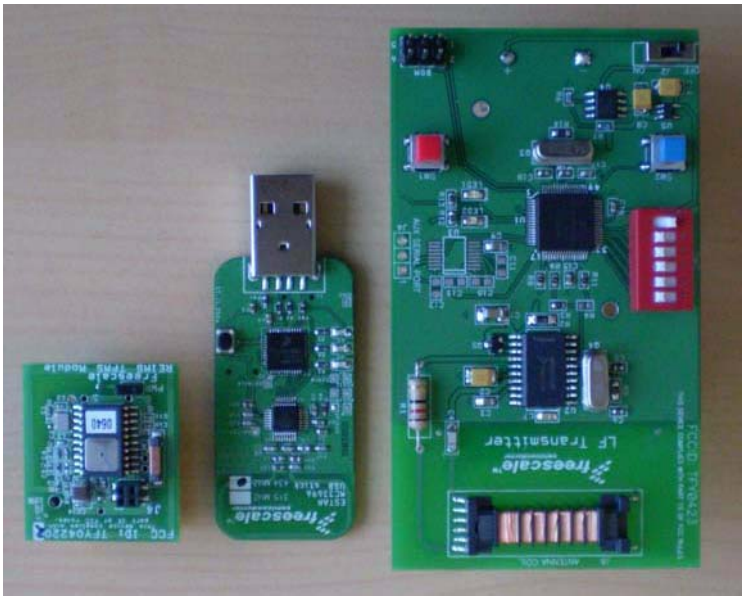

Figure 1-1. The TPMS Kit

This document provides information relative to the whole system.

1.2 MPXY8300 Module (Key Features)

- 8-bit MCU
- S08 Core with SIM, interrupt and debug/monitor
- 512 RAM
- 8K FLASH (in addition to 8K providing factory firmware and trim data)
- 32-byte, low power, parameter registers
- 4 GPIO pins with optional pull-ups/pull-downs and wake-up interrupt
- Real Time Interrupt driven by LFO with interrupt intervals of 8, 16, 32, 64, 128, 256, 512 or 1024 msec
- Low power wake-up timer and periodic reset driven by LFO
- Watchdog time-out with selectable times and clock sources
- 2-channel general purpose timer/PWM module (TPM1)
- Internal oscillators

- MCU bus clock of 0.5, 1, 2 and 4 MHz (1, 2, 4 and 8 MHz HFO)
- Charge pump clock of 10 MHz
- Low frequency, low power time clock (LFO) with 1 msec period
- Medium frequency, LFR decoder and sensor clock (MFO) of 8 sec period
- Low voltage detection
- Normal temperature restart in hardware (over temperature detected by software)
- Differential input LF detector/decoder
- Temperature sensor with signal interface to ADC10
- Pressure sensor with signal interface to ADC10
- Z- and X-axis accelerometers with signal interface to ADC10
- Voltage reference measured by ADC10
- Internal 315/434 MHz RF transmitter
  - External crystal oscillator
  - ASK and FSK modulation capability
  - Programmable data rate generator
  - Manchester or bi-phase data encoding
  - 128-bit RF data buffer with RTS/CTS handshake
  - Direct access to RF transmitter from MCU for unique formats
- Supply voltage charge pump to compensate for cold batteries with selectable charge times (30, 60, 120 and 240 msec)
  - 8-bit MCU programming capability through the 4 pins connector and the BDM multilink type
  - Flexible matching network circuit allowing to use either the footprint antenna or external antenna
  - Typical wireless range is 15 m in air with footprint antenna
  - 3 V household battery

Figure 1-2. MPXY8300 Module

1.3 TPMS Receiver (MC33696 Based) (Key Features)

- Wireless communication through 315 or 434 MHz
- Typical wireless range is 20 m
- Data rate is 9.6 kb/s
- USB communication
- Virtual serial port class - interface for GUI and terminal
- HID class - mouse for windows
- 1 push buttons providing the MCU programming through an embedded Bootloader

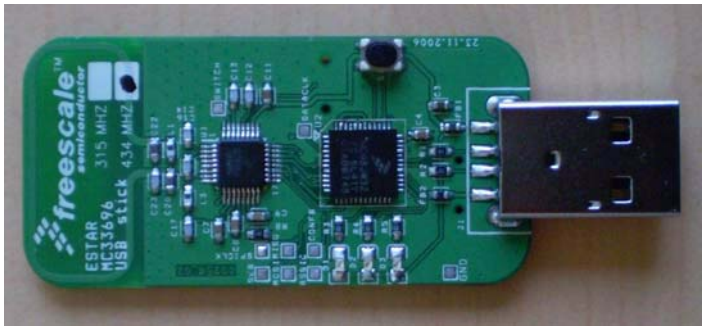

Figure 1-3. TPMS Receiver

1.4 LF 125 KHz Emitter (Key Features)

- 8-bit MCU programming capability through the 4 pins connector (BDM multilink type)
- Data format flexibility obtained by MCU program change
- Operating range up to 0.5 m
- Multi switch position (6) for signal type selection
- IRQ and KBI button to trigger a transmission
- LF-signal modulation through Freescale's MC33690 using either the on-board MCU's SPI MOSI pin (standard) or TPM2CH0 pin

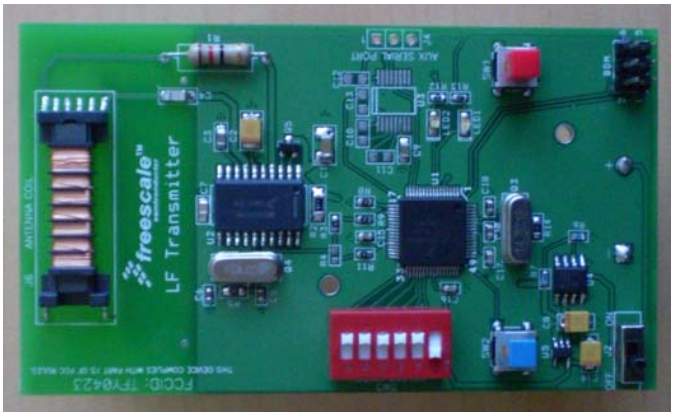

Figure 1-4. LF 125 kHz Emitter

1.5 Demonstration Software

1.5.1 MPXY8300 Module - Main Software Description

The operation of the device is controlled by the 8-bit SO8 microcontroller dedicated to the application and its embedded software. At specific timing intervals, sensor data are accessed, analyzed and placed into the RF data buffer for transmission.

The controller is intended to provide a basic data collection and transmission method while conserving battery power. This is achieved by having the MCU in very low power mode most of the time and by waking it up through the PWU timer or an LF telegram.

All timing intervals are based on the internal low frequency oscillator, LFO, and the Periodic wake-up module.

With the Low Frequency receiver block always ON, exit from STOP1 mode is achievable with the detection of an LF Manchester coded telegram.

In parallel with the main software, some specific routines are showed as examples for the LF and RF blocks.

1.5.2 MPXY8300 Receiver Software Description

See ESTAR documentation (http://www.freescale.com/files/sensors/doc/ref_manual/ESTARRM.pdf) at the USB Stick Board receiver chapter.

1.5.3 LF 125 kHz Emitter Software Description

The MC9S08GB60A application software drives the Freescale MC33690 used for 125 kHz LF-encoded datagram transmission.

When a high logical level is applied on the MC33690 MODE1 pin, the circuit is in configuration where a direct connection to a microcontroller configuration is used.

A KBI interrupt routine triggers the MOSI signal (Manchester Coded) applied on the AM pin.

Forcing high and low levels on AM achieves the 125 kHz amplitude modulation by switching on/off both antenna drivers.

The circuit can be put into standby mode by applying a low level on the MODE2 pin.

In standby mode, the oscillator and most of the internal biasing currents are switched off.

The driver outputs TD1 and TD2 are frozen in their state (high or low level) before entering into standby mode. DOUT forces a low level.

The types of signal modulation are selectable via the 6 pins switch. With the standard program, only Manchester telegrams are available. The difference between each modulated signal relies on the telegram bytes contents. This content can easily be changed to create other telegram types.

Finally, external communication between the MCU and a PC is achievable through the SCI MCU module (TxD1,RxD1 pins) upon software modification.

1.6 Board Programming Guide

1.6.1 Downloading Demo Software to MPXY8300 Modules or LF 125 kHz Emitter

The software project can be downloaded on the MPXY8300 or LF 125 kHz Emitter Modules using CodeWarrior 5.0 or latest. The connection to the BDM Multilink is direct for the LF125KHz board or is done through an interface cable for the MPXY8300 Module (Figure 1-5).

Figure 1-5. MPXY8300 and LF125kHz Connected to the USB Multilink

1.6.2 Downloading Demo Software to ESTAR for TPMS

See ESTAR documentation (http://www.freescale.com/files/sensors/doc/ref_manual/ESTARRM.pdf) at the USB Stick Board receiver chapter.

Chapter 2
Schematics and Bill of Materials

2.1 MPXY8300 Module Evaluation Board

2.1.1 MPXY8300 Module EVB Schematic

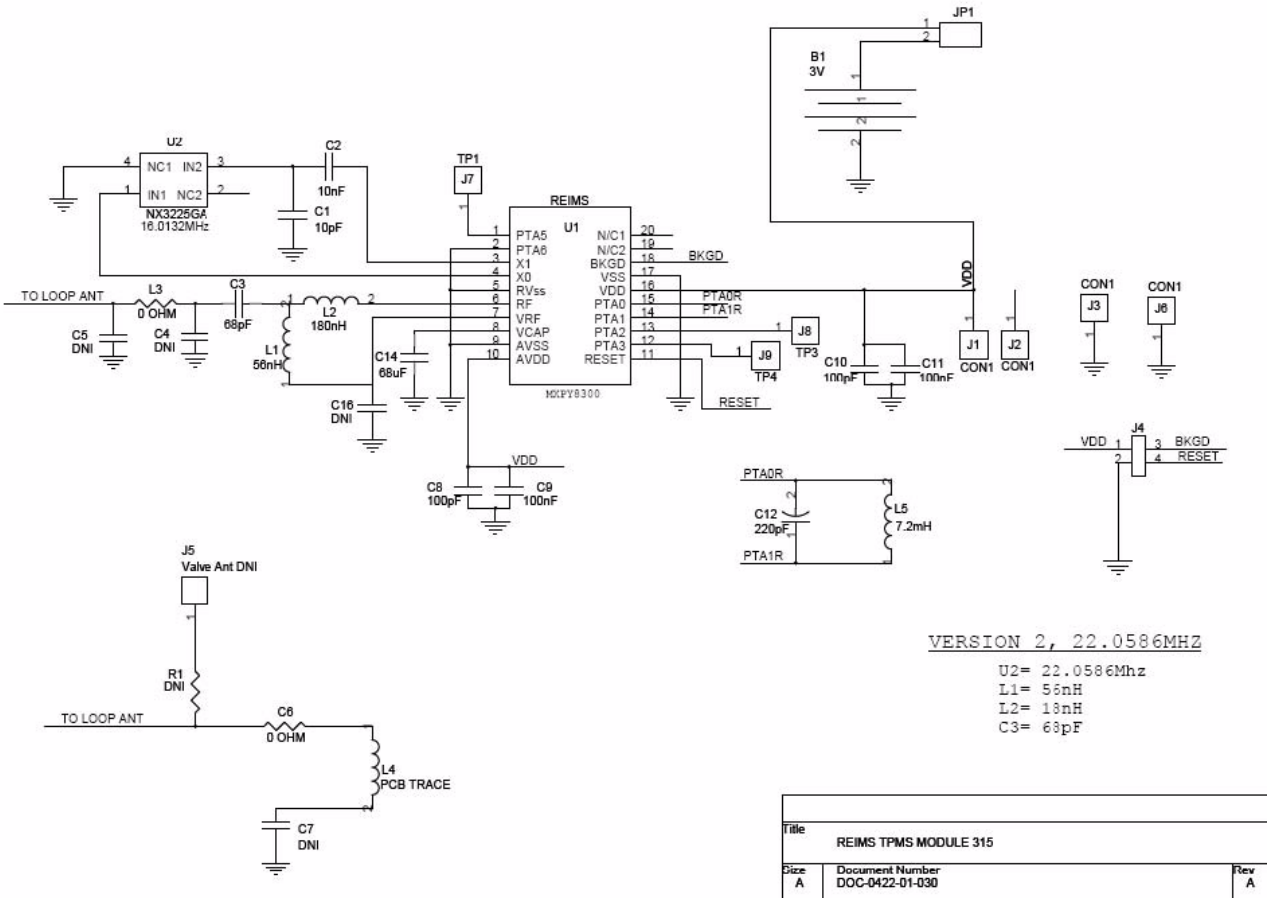

Figure 2-6. MPXY8300 Module Schematic

L1, L2 and L3 matching network values are applicable when using the loop antenna available on the module. If an external antenna is used (connected to J5) the matching network must be changed accordingly and based on the matching network described in the MPXY8300 data sheet assuming the new antenna is a 50 Ω load. If not, component C4, C5, L3 footprints can be used to finely tune the new antenna and obtain the desired output power.

2.1.2 MPXY8300 Module EVB Bill of Material

Table 2-1. 315 MHz Module BOM

REF Nbr.	VALUES	PACKAGE	FOOTPRINT	PART NUMBER	SUPPLIER
C1	10 pF	C	SM/C_0402	C1005C0G1H100DT	TDK
C2	10 nF	C	SM/C_0402	C1005X7R1E103KT	TDK
C3	68 pF	C	SM/C_0402	C1005C0G1H680JT	TDK
C4	OPEN	C	SM/C_0402	-	
C5	OPEN	C	SM/C_0402	-	
C6	0 Ω RESISTOR	C	SM/C_0402		
C7	OPEN	C	SM/C_0402	-	
C8	100 pF	C	SM/C_0402	C1005C0G1H101JT	TDK
C9	100 nF	C	SM/C_0402	C1005X7R1E103KT	TDK
C10	100 pF	C	SM/C_0402	C1005C0G1H101JT	TDK
C11	100 nF	C	SM/C_0402	C1005X7R1E103KT	TDK
C12	220 pF	CAP	SM/C_0402	C1005C0G1H221JT	TDK
C14	22 µF	C	SM/C_1210	C3225X5R1C226MT	TDK
C16	OPEN	C	SM/C_0402	-	
J4	4 pins socket	CON4	SMD 2x2	S6010-02-ND	DIGIKEY
L1	56 nH	INDUCTOR	SM/C_0402	MLG1005S56NJT	TDK
L2	180 nH	INDUCTOR	SM/C_0402	MLG1005S180NJT	TDK
L3	0 Ω RESISTOR	INDUCTOR	SM/C_0402		
L4	LOOP ANTENNA	INDUCTOR	LOOP-ANTENNA-REV1		
L5	7.2 mH	INDUCTOR	SELF 125K	TP0702-0720J or TPLS8027-7,2mH	PREMO or TDK
R1	OPEN	R	SM/C_0402		
U1	MPXY8300	SOIC20	SOJ.050/20/WB.400/L.500	MPXY8300A	FREESCALE
U2	16.0132 MHz	NX3225DA	Quartz 3225	NX3225DA	NDK

Table 2-2. 434MHz Module BOM

REF Nbr.	VALUES	PACKAGE	FOOTPRINT	PART NUMBER	SUPPLIER
C1	10 pF	C	SM/C_0402	C1005C0G1H100DT	TDK
C2	10 nF	C	SM/C_0402	C1005X7R1E103KT	TDK
C3	47 pF	C	SM/C_0402	C1005C0G1H470JT	TDK
C4	OPEN	C	SM/C_0402	-	
C5	OPEN	C	SM/C_0402	-	
C6	0 Ω RESISTOR	C	SM/C_0402		
C7	OPEN	C	SM/C_0402	-	
C8	100 pF	C	SM/C_0402	C1005C0G1H101JT	TDK
C9	100 nF	C	SM/C_0402	C1005X7R1E104KT	TDK
C10	100 pF	C	SM/C_0402	C1005C0G1H101JT	TDK
C11	100 nF	C	SM/C_0402	C1005X7R1E104KT	TDK
C12	220 pF	C	SM/C_0402	C1005C0G1H221JT	TDK
C14	22 µF	C	SM/C_1210	C3225X5R1C226MT	TDK
C16	OPEN	C	SM/C_0402	-	
J4	4 pins socket	CON4	SMD 2x2	S6010-02-ND	DIGIKEY
L1	18 nH	INDUCTOR	SM/C_0402	MLG1005S18NJT	TDK
L2	56 nH	INDUCTOR	SM/C_0402	MLG1005S56NJT	TDK
L3	0 Ω RESISTOR	R	SM/C_0402		
L4	LOOP ANTENNA	INDUCTOR	LOOP-ANTENNA-REV1		
L5	7.2 mH	INDUCTOR	SELF 125K	TP0702-0720J or TPLS8027-7,2mH	PREMO or TDK
R1	OPEN	R	SM/C_0402		
U1	MPXY8300	SOIC20	SOJ.050/20/WB.400/L.500	MPXY8300A	FREESCALE
U2	22.0586 MHz	NX3225DA	Quartz 3225	NX3225DA	NDK

2.2 LF 125 kHz Emitter Evaluation Board

2.2.1 LF 125 kHz Emitter EVB Schematic

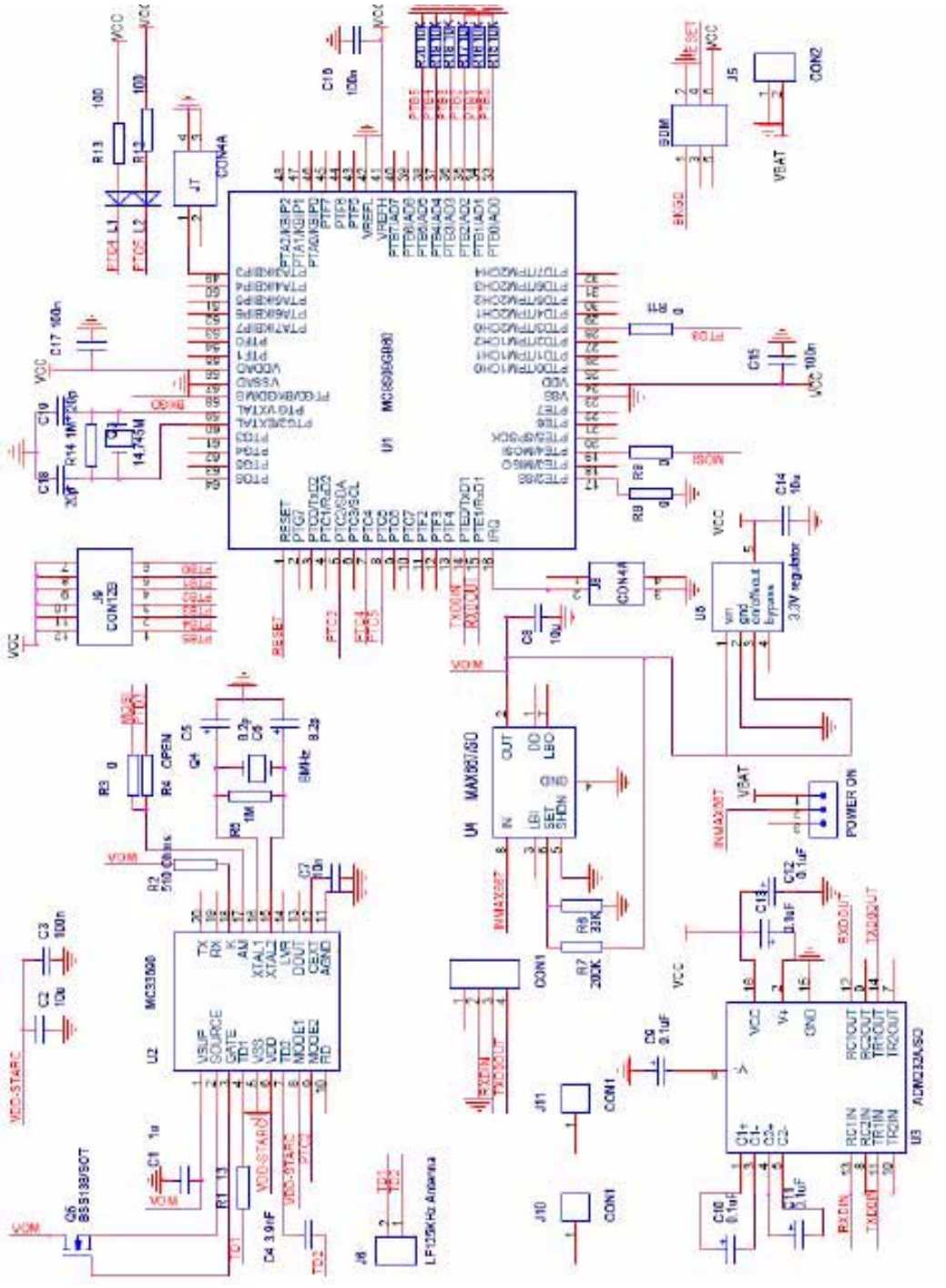

Figure 2-7. LF 125 kHz Transmitter Schematic

Table 2-3. LF 125 kHz Emitter EVB Bill of Material

REF Nbr.	VALUES	PACKAGE	FOOTPRINT
BDM	CON6A	CON6A	BLKCON.100/VH/TM2OE/W.200/6
C1	1 µ	C	SM/C_1210
C2	10 µ	C	SM/C_1210
C3	100 n	C	SM/C_0805
C4	3.9 nF	C	SM/C_1206
C5	8.2 p	C	SM/C_0603
C6	8.2 p	C	SM/C_0603
C7	10 n	C	SM/C_0805
C8	10 µ	C	SM/C_1206
C9	0.1 µF	C	SM/C_0805
C10	0.1 µF	C	SM/C_0805
C11	0.1 µF	C	SM/C_0805
C12	0.1 µF	C	SM/C_0805
C13	0.1 µF	C	SM/C_0805
C14	10 µ	C	SM/C_1206
C15	100 n	C	SM/C_0603
C16	100 n	C	SM/C_0603
C17	100 n	C	SM/C_0603
C18	20 p	C	SM/C_0603
C19	20 p	C	SM/C_0603
J2	POWER ON	SIPSOC-3	TO221AB
J4	CON1	CON4	JST-SR-4PIN
J5	CON2	CON2	SUPPORT BAT 9V
J6	PREMO P/N: P-830-003	CON8	ANTENNALF125
J7	CON4A	CON4A_0	SWICTH BUTTON LF
J8	CON4A	CON4A_0	SWICTH BUTTON LF
J9	CON12B	CON12B_0	DIP.100/12/W.300/L.700
J10	CON1	CON1	HOLE250MM-2006
J11	CON1	CON1	HOLE250MM-2006
L1	DIODE	DIODE	SM/C_0805
L2	DIODE	DIODE	SM/C_0805
Q3	14.745 M	CRYSTAL	RAD/.400X.150/LS.200/.034
Q4	8 MHz	CRYSTAL	RAD/.400X.150/LS.200/.034
Q5	BSS138/SOT	BSS138/SOT_1	SM/SOT23_123
R1	13	R	AX/.575X.150/.034
R2	510 Ωs	R	SM/C_1206
R3	0	R	SM/C_0603
R4	OPEN	R	SM/C_0603
R5	1 M	R	SM/C_0805
R6	33 K	R	SM/C_0603
R7	200 K	R	SM/C_0603
R8	0	R	SM/C_0603

Table 2-3. LF 125 kHz Emitter EVB Bill of Material

REF Nbr.	VALUES	PACKAGE	FOOTPRINT
R9	0	R	SM/C_0603
R11	0	R	SM/C_0603
R12	100	R	SM/C_0603
R13	100	R	SM/C_0603
R14	1 M	R	SM/C_0805
U1	MC9S08GB60	64 Pins LQFP	QUAD.50M/64/WG12.00
U2	MC33690	SOIC20	SOG.050/20/WG.420/L.500
U3	ADM232A/SO	SO-16	SOG.65M/16/WG8.20/L6.35
U4	MAX667/SO	SO-8	SOG.050/8/WG.244/L.175
U5	3 V regulator	LP2985_3	REG2

2.3 TPMS Receiver (MC33696 based) Evaluation Board

For details on the receiver, see ESTARRM Rev. 0 02/2007.

Chapter 3
MPXY8300 Main Features Specific Information

3.1 LF Receiver

The LF block allows wake-up of the MPXY8300 device from STOP (low power) mode when a carrier or a Manchester datagram is detected. If the LF datagram is coded Manchester, it will be decoded by the embedded LF state machine assuming the LF module has been configured appropriately.

If the datagram is not Manchester coded, the MCU will need first to be waked-up by a Carrier (Carrier Mode detection) and then go in MCU Direct mode to decode the telegram.

3.1.1 LF Sampling Frequency

To minimize the power consumption, the LF block has a programmable periodic wake-up mechanism for carrier and Manchester data modes, running with the 1 kHz Low frequency oscillator.

The Low frequency oscillator allows the module to be normally in a very low power state and to be periodically wake up every 32,768 sec down to every 1 msec. When the LF block is awake, the ON time last during 200 μsec or 2.12 msec depending on the value set to the LFON bit.

For adequate LF telegram sampling, the first point to take into consideration is the accuracy of the sampling time directly linked to the accuracy of the 1 kHz LFO clock.

The LFO clock varies from 769 Hz to 1428 Hz according to the product specification.

It is therefore recommended to use the appropriate sampling timing in regards with the incoming LF signal duration. Sampling time selection is achieved with the LFCR register content.

As an example, if there is a request for sampling an incoming 8 msec carrier length every 8 msec (LFON = 200 μsec), a hit rate of 100 percent will not be achieved.

With Flfo = 1428 Hz, the signal will be sampled every 8/1428 = 5.6 msec giving a high hit rate.
With Flfo = 769 Hz, the signal will be sampled every 8/769 = 10.4 msec giving a low hit rate.

For a 8 msec carrier length LF signal, the recommendation is to sample the signal every 4 msec.This will give a sampling interval between 2.8 and 5.2 msec when taking into account the frequency of the LFO, improving the quality of the hit rate.

3.1.2 LF Datagram Decoding Using the Carrier Detect Followed by MCU Direct

In this example, the LF datagram to decode is composed by:

-8 msec of Synchronization

-5 manchester data bit time (250 μsec each)

-3 msec preamble

-5 bytes of data's (xD5,x55,xAA,xCA,xEE) (d213,d85,d170,d202,d238)

0 is coded with a 1 msec width, 1 is coded with a 2 msec width

The LF receiver has been designed to decode Manchester datagram within 5 tDATA Manchester bit timings (5 Preamble Manchester zeros @ nominal data rate = 1.25 ms) (Figure 5). This is the time for the Automatic gain Control AGC to finish its "high-level" search for the proper gain range to keep the signal tuned and enter "fine" tracking mode. Once this is completed the LFDO will start to track the incoming data stream, allowing the decoding of the telegram. The tracking of the LFDO is possible only in MCU direct mode. This is what we observe with the CH2 going to a high state after a first tDATA Manchester bit timing showing that the LFDO bit is ready for tracking.

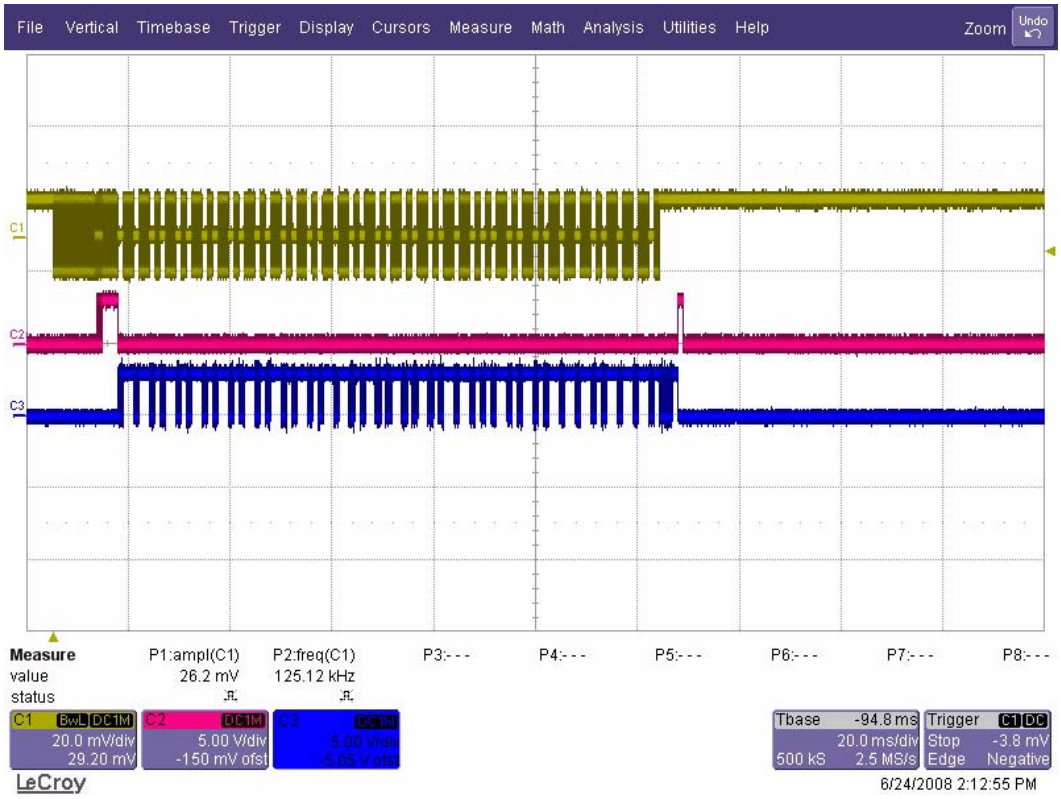

Figure 3-1. LF Telegram and Decoding Check

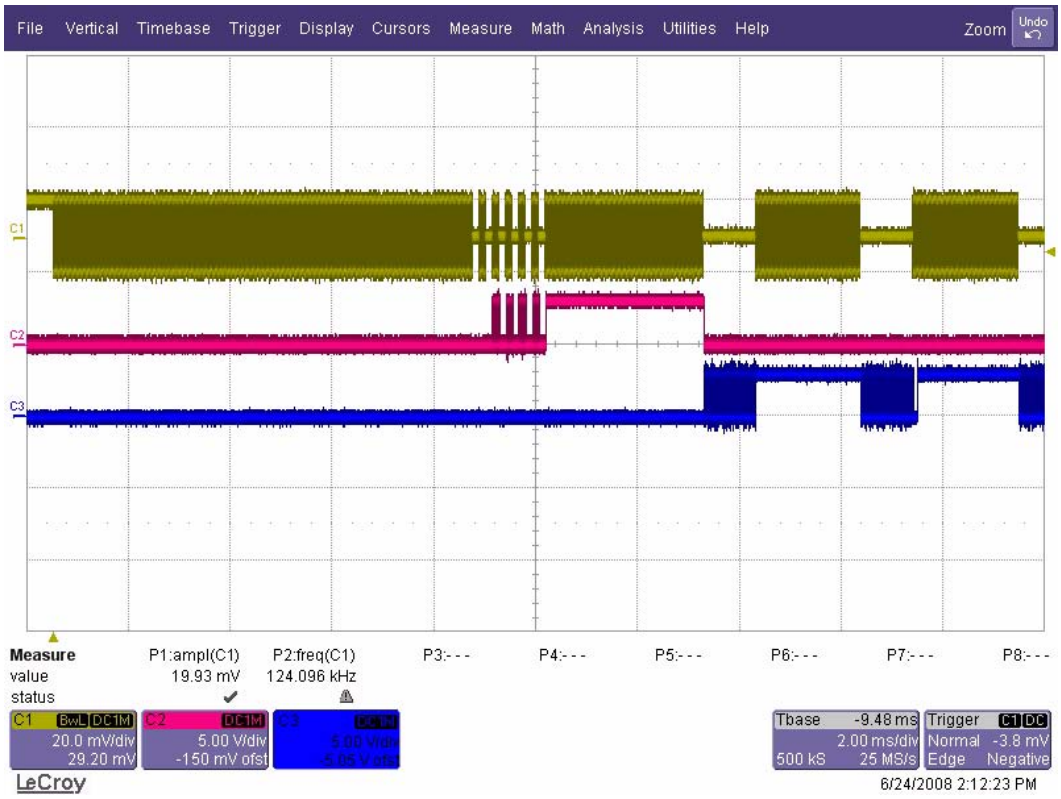

Figure 3-2. LF Telegram (Synchro/5 Tbit/Preamble Detail)

Example of decoding function with MCU direct telegram:

```
/*****************************************************************************
function    :   initLF1()
NOTES    :                      This function initializes the Carrier detect Mode
*****************************************************************************/

void InitLF1(void)

{

LFCS1_LFCFAK=1;
LFCS0_LFEN=0;     //Mandatory before any other writings
LFCR=0x00;
LFCS0 = 0x00; // LF0&1 disable  for autozero
LFCS1=0x00;
LFCR=0x41; // Carrier detect - 4msec  : 0x42   - 2msec : 0x41
LFCS0=0x48;    //  0x48 : Detection Level = 3.5mV , LF0 Enable , Tlfrst duration = 3msec
         //  Interrupt Enable

}
/*****************************************************************************
function    :   initLF2()
NOTES    :                          This function initializes the MCU direct Mode
*****************************************************************************/

void InitLF2(void)

{
LFCS0_LFEN=0;     //Mandatory before any other writings
LFCS0 = 0x00;  // LF0&1 disable  for autozero
LFCR=0x10;     //10 for MCU Direct LF always ON !!! LFON must be set here
LFCS0=0x08;    //  0x08 : LF Enable and Detection Level = 3.5mV
         //  0x0A : LF Enable and Detection Level = 10 mV
         //  Interrupt Disable

}

/*****************************************************************************
Function: Decode_LF
NOTES    :                                                  Decode LF telegram
*****************************************************************************/

void Decode_LF()

{
byte i,j,l;
DisableInterrupts;
j=0;
LFCS0_LFEN=0;     //Mandatory before any other writings
InitLF2();
PTAD=0x00;
PTADD = 0x60; /*PTA5&6 configured as an Output */
InitTPM1CH0();
TPM1CNT=0;  //clear timer contents to avoid overflow
                     /* Wait for the LFDO bit set to 1 */
```

```
while (LFCS0_LFDO==0);


/* Manchester DETECTION - 5 bit times of 250usec each */
     do
            { TPM1CNT=0;

              while (LFCS0_LFDO!=0)  //wait LFDO stays to One
              {
              PTAD=0x40;
              PTAD=0x00;
              }
              while (LFCS0_LFDO==0);  //wait LFDO stays to Zero
              j++;
            }
   while (j<4 && TPM1CNT<300);


/********3 msec Preamble detect  ******/

      TPM1CNT=0;  //clear timer contents to avoid overflow

       e=TPM1CNT;
       PTAD=0x40;

        while (LFCS0_LFDO!=0);  //wait LFDO stays to One

        PTAD=0x00;

       while (LFCS0_LFDO==0)
       {
        PTAD=0x20;
        PTAD=0x00;
       }

/********DECODING LOOP****/

for (l=0;l<5;l++)


   {
    for (j=0;j<8;j++)
       {

       TPM1CNT=0;  //clear timer contents to avoid overflow

       e=TPM1CNT;
       PTAD=0x20;

        while (LFCS0_LFDO!=0 )  //wait LFDO stays to One
        {
        }

        f=TPM1CNT;
        PTAD=0x00;
```

```
                        while (LFCS0_LFDO==0 && TPM1CNT<3500)   //wait LFDO goes to zero

                             {
                              PTAD=0x20;
                              PTAD=0x00;
                             }

                               if (f>e && (f-e)>1850 && (f-e)<2300)

                                    {
                              LFDatagram[l] |= (1 << (7-j));   //    CREATE A ONE  2msec
                                    }


                              if (f>e && (f-e)>925 && (f-e)<1150)


                                    {
                              LFDatagram[l] &= ~(1 << (7-j));    //   CREATE A ZERO  1msec
                                    }

                            } // end of for (j=0;j<8;j++)

                          } //end of  for (l=0;l<5;l++)


            /****Check of decoded values **********/

if (LFDatagram[0]==213 &&  LFDatagram[1]==85 && LFDatagram[2]==170 && LFDatagram[3]==202
&& LFDatagram[4]==238)

            {
            TogglePTA5();
            TogglePTA6();
            }


for (i=0;i<5;i++)

{
  LFDatagram[i]=0;
}

asm{cli;
     }
}
```

3.1.3 LF Decoding Using a Manchester Coded Datagram

The Manchester data mode is used to detect and decode Manchester LF datagram. The information being transmitted will be modulated using on/off keying and will have a carrier frequency of approximately 125 kHz. The module will use automatic gain control to acquire and track the incoming signal, which may vary greatly due to changes in antenna distance and orientation from the transmitter. The module will monitor: amplitude, carrier frequency, decoded bit timing, and message ID. If all criteria's are met, the MCU will be interrupted when each byte of information is available.

The LFR decoder has a time-out mechanism that turns off the LFR detector and the Medium Frequency Oscillator MFO for a period of time tRST when the LFR decoder receives only Manchester zeros (in preamble) for an extended time, tTIMEOUT, which is defined by counting 65 rising edges of the LFO clock. The time-out counter will always be enabled when the module is first enabled and will begin after the signal has been acquired and valid Manchester data coded zeros are received.

Therefore the preamble length must not exceed 65/1500 = 43 msec to avoid the LFR being turned off before the synchronization is met.

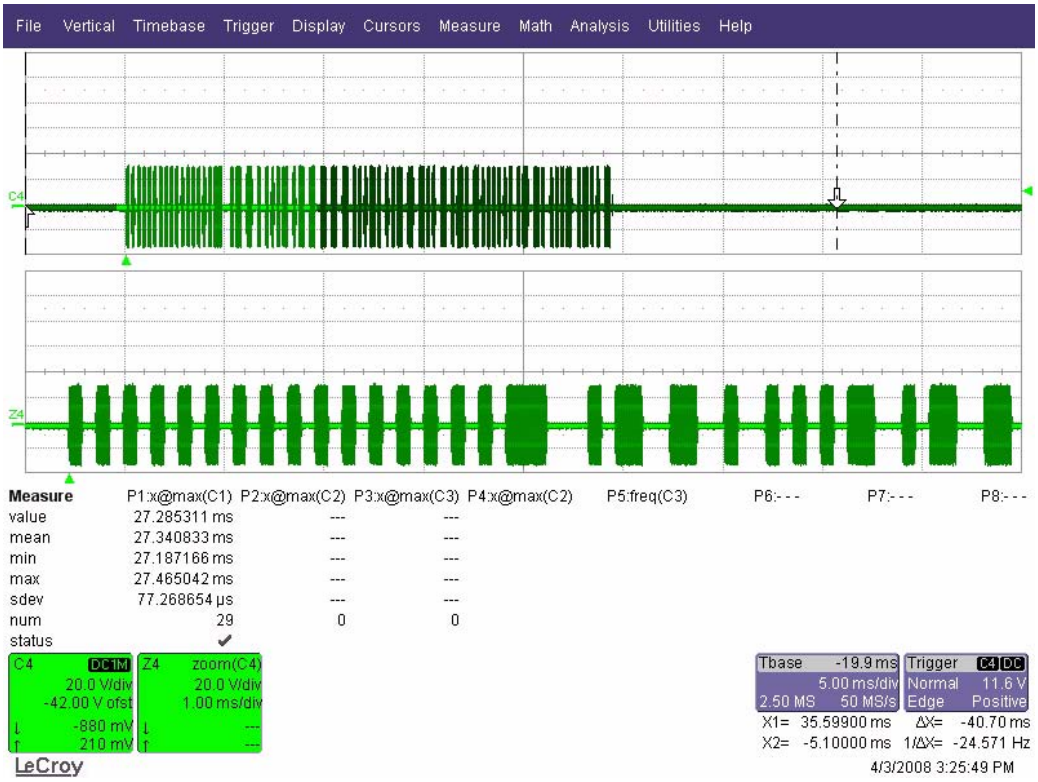

Figure 3-3. Example of Manchester LF Telegram

- Blue rectangle shows the 16 bits preamble signal
- Red rectangle shows the 9 tDATA synchronization
- Green rectangle shows a part of the 16 bits of Wake-up bytes (xF6 and xAA)

Example of decoding function with Manchester telegram:

```
/*****************************************************************************
function    :   InitLF()
NOTES    :                  This function initializes the Manchester detect Mode
*****************************************************************************/

void InitLF(void)

{

LFCS1_LFCFAK=1;
LFCS0_LFEN=0;            //Mandatory before any other writings
LFCR=0x00;
LFCS0=0x00;                // LF0&1 disable  for autozero
LFCS1=0x00;
LFCR=0x81;//x81 sampling every 2 msec
LFCA0=0xAA;//Wake-up codes defined by user                                    LSB
LFCA1=0xF6;//Wake-up codes defined by user                                    MSB

LFCS0=0x48;    //   //  0x48 : Detection Level = 3.5mV , LF0 Enable , Tlfrst duration =
1*TLFS

}

/*****************************************************************************
Function: Decode_LF_Datagram  -LF data decoding through Interrupt
*****************************************************************************/

void Decode_LF_Datagram()

{
unsigned char ii;
PWUSC0=0x33;        // every 60 sec

for (ii=0;ii<=5;ii++)

   {
            {
            while((LFCS1 & 0x80)==0);
                        {
                         LFDatagram[ii]=LFDATA;
                         LFCS1=0x40;                // clear the flag
                        }
            }

          if (ii==5)

          {
            LF_OFF();
            break;
          }

   }

/**************LF0 detection*******************/
```

3.2 RF Transmitter

3.2.1 RF Block general information

The RFX embedded module in the MPXY8300 consists of an RF output driver for an antenna and a hardware data buffer for automated output or direct control from the MCU. It also has a charge pump to generate a higher supply voltage for the RF transmission, if desired, when the supply battery voltage is too low.

It has its own memory map containing 32 register locations. These registers contain control and status bits for the RFX modules, data locations for the RF data buffer, trim variables and test registers. Access to these bits is through the internal SPI interface. The firmware subroutines control this interface so that the use only needs to call the REIMS_RFRD or REIMS_RFWRT subroutines to pass the address and data that needs to be accessed or changed in the RFX.

3.2.2 RF Output Impedance

Table 3-1 gives the RF Output impedances of the MPXY8300 at the two frequencies of use:

Table 3-1. RF Output Impedances

Frequency	Real (Ω)	Imaginary (Ω)
315 MHz	360	J 315
434 MHz	360	J 300

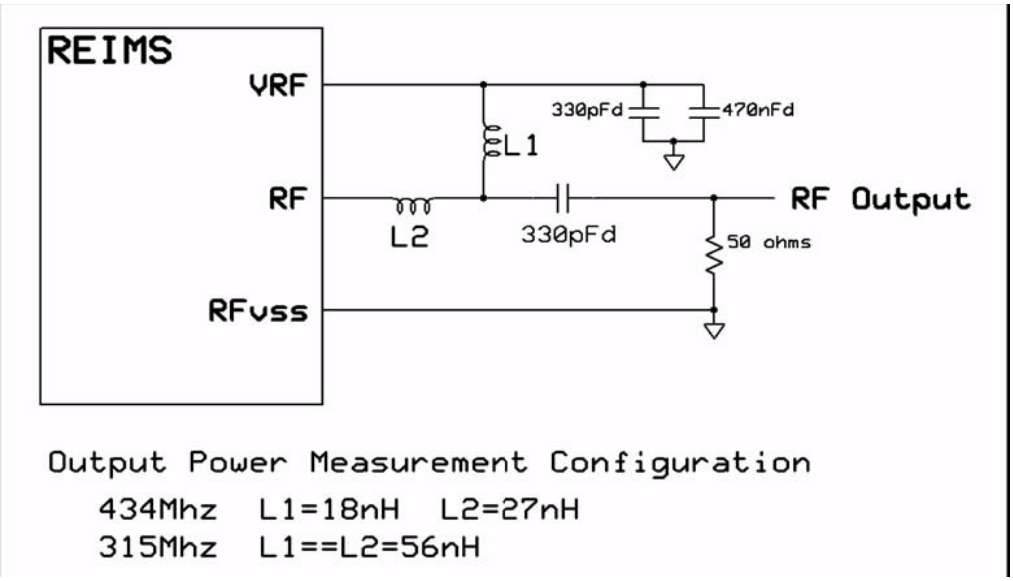

Figure 3-4. Recommended Matching Network

3.2.3 Smith Chart Matching Network at 315 MHz and 434 MHz

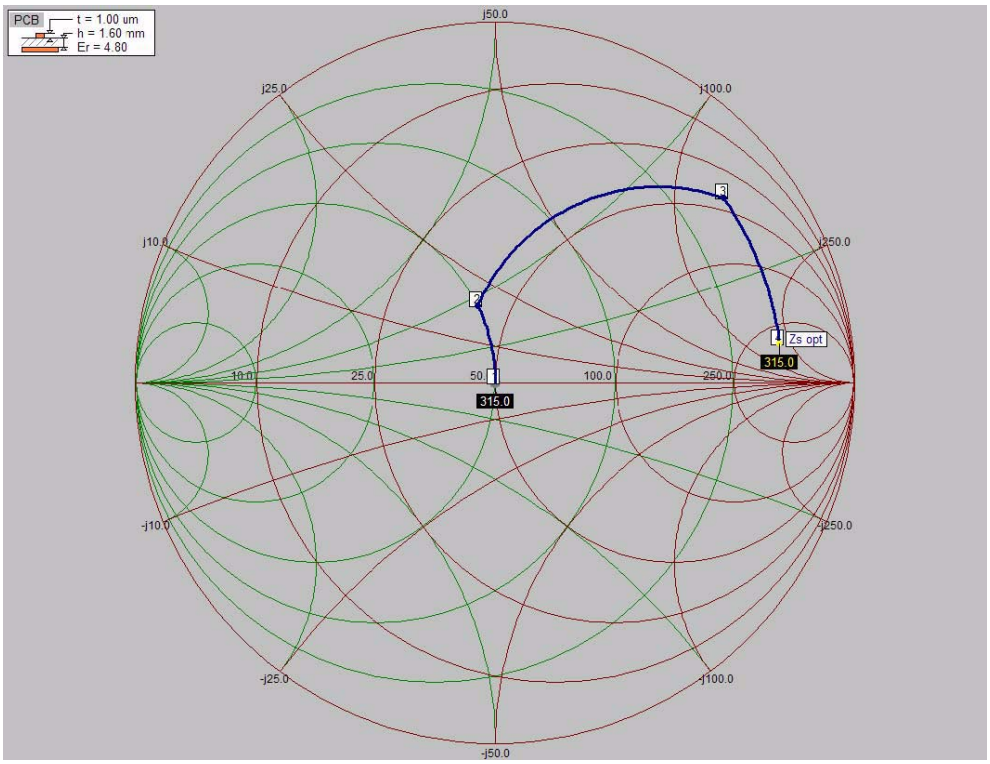

Figure 3-5. Smith Chart at 315 MHz

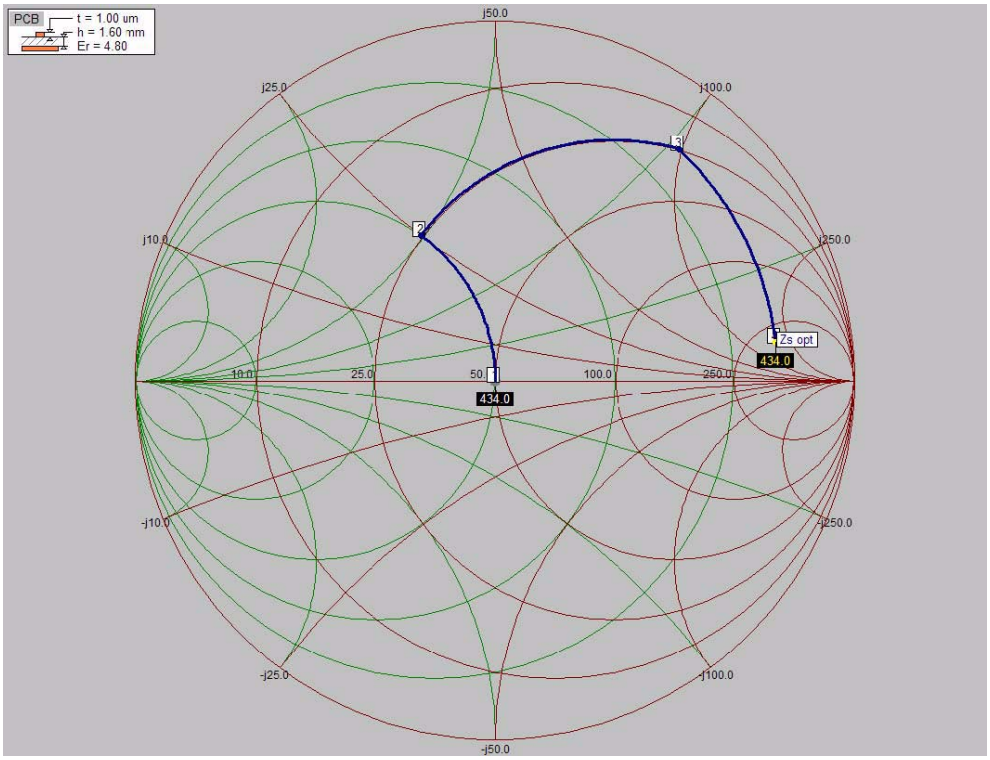

Figure 3-6. Smith Chart at 434 MHz

For both frequencies, the values for L1 and L2 correspond to the recommended values showed on Figure 3-3 .

A parasitic capacitance of 3 pF is added between the RF out and ground to reach the device output impedance.

Typically this capacitance is composed by PCB parasitics, plus internal bonding and PADs.

3.2.4 Charge Pump Complementary Information

When re-connecting VRF node to AVDD, if VCAP voltage is higher than AVDD voltage then a negative current spike will occur. The spike value is depending from AVDD / VCAP voltages difference. To avoid this current peak the VCAP capacitor must be discharged with the REIMS_RF_DISCHARGE() firmware function directly after an RF transmission using the charge pump.

In case of no use of the charge pump, we recommend to tie both VCAP and VRF pins to AVDD pin and setting the 4 LSB RFCR2 to 0 (i.e. AVDD=VCAP=CPUMP=0) (See MPXY8300 data sheet for details on the RF registers). In addition, ESD networks of the three pins will be associated, improving thus immunity to ESD. As long as CPUMP is 0, no leakage current consumption will appear on VCAP pin.

3.2.5 RF MCU Direct Example

In the MCU direct mode the data to the RF output stage is driven directly from the MCU. In this mode the user software must control the RF output stage to power up (using the SEND control bit), wait for the RF output stage to stabilize (monitor the RCTS status bit) and clock the DATA to the RF output stage. The DATA needs to be Manchester or Bi-phase coded by the user. In this mode the data rate and its stability will depend on the internal 8 MHz oscillator.

Any transfers of data from the MCU will use the DX signal as modulated data on the RF pin once the RF output stage is set up to transmit. The maximum data rate in this mode will depend on the complexity of the user software.

The following software sends a table of 122 bytes which takes 98 msec at 9600 bauds.

Figure 3-7. RF Telegram Received and Decoded

Software example:

```
/*
 ****************************************************************************
 *
 *                          M A I N
 *
 ****************************************************************************
 */
#pragma CODE_SEG DEFAULT

void main(void)
{

  DisableInterrupts;
  SPMSC1_BGBE=1;
  SOPT1=0x33;       //   COP DISABLE //STOP ENABLE
  REIMS_RF_RESET();
  RF_Setup(); // SetupRFX();
  PWUSC0_WUFACK=1;
  PWUDIV = 0x1F;
  PWUSC0 = 0x3F;
  PWUSC1 = 0x00;
  SOPT2=0x70;
  SPMSC1=0x1C;
array1[0] = 0xFF;
array1[1] = 0xFE;
array1[2] = 0xFF;
array1[3] = 0x00;
array1[4] = 0xFE;
array1[5] = 0x01;

delay(10);
while(1)

         {
          RF_Setup();
          Code_Manchester_Frame(&array1[0]);
          RFRD_data[0]=High_Manch;
          RFRD_data[1]=Low_Manch;
          Code_Manchester_Frame(&array1[1]);
          RFRD_data[2]=High_Manch;
          RFRD_data[3]=Low_Manch;
          REIMS_READ_COMP_TEMP_8(&Temp8BitData);
i=4;

do
{
                         REIMS_READ_COMP_ACCEL_XZ(&Acc_XZ_Array,Xout,4,0);

                         Code_Manchester_Frame(&array1[2]);
                         RFRD_data[i]=High_Manch;
                         RFRD_data[i+1]=Low_Manch;

                         Code_Manchester_Frame(&array1[3]);
                         RFRD_data[i+2]=High_Manch;
                         RFRD_data[i+3]=Low_Manch;
```

```
                        array1[6]=Acc_XZ_Array[1]>>2;             //X results

                        Code_Manchester_Frame(&array1[6]);
                        RFRD_data[i+4]=High_Manch;
                        RFRD_data[i+5]=Low_Manch;

                        Code_Manchester_Frame(&array1[4]);
                        RFRD_data[i+6]=High_Manch;
                        RFRD_data[i+7]=Low_Manch;

                        Code_Manchester_Frame(&array1[5]);
                        RFRD_data[i+8]=High_Manch;
                        RFRD_data[i+9]=Low_Manch;


                        array1[7]=Acc_XZ_Array[3]>>2; //Z results

                        Code_Manchester_Frame(&array1[7]);
                        RFRD_data[i+10]=High_Manch;
                        RFRD_data[i+11]=Low_Manch;

                      i+=12;
}


while (i<=243);

 REIMS_RF_DIRECT_TX(256,0,&RFRD_data[0],1);
 REIMS_RF_DIRECT_TX(256,0,&RFRD_data[32],0);
 REIMS_RF_DIRECT_TX(256,0,&RFRD_data[64],0);
 REIMS_RF_DIRECT_TX(256,0,&RFRD_data[96],0);
 REIMS_RF_DIRECT_TX(256,0,&RFRD_data[128],0);
 REIMS_RF_DIRECT_TX(256,0,&RFRD_data[160],0);
 REIMS_RF_DIRECT_TX(256,0,&RFRD_data[192],0);
 REIMS_RF_DIRECT_TX(160,0,&RFRD_data[224],0);
 DisableInterrupts;
         }

}
/*****************************************************************************
function   :RF_Setup(void)
parameters  :void
returns     :void
type        :low level c
description:    RF Setup at 434 MHz
*****************************************************************************/

void RF_Setup(void)

{
    REIMS_RFWRT(2,4);    //  4 for MCU direct
    REIMS_RFWRT(3,142);  //  128 + 14
    REIMS_RFWRT(4,171);   // 171   0xAB    for FSK @ 315 MHz   at +/-35KHz
    REIMS_RFWRT(5,80);    // 76    0x48    +4 for pll fail bit         "          "
    REIMS_RFWRT(6,172);   // 172   0xAC             "
    REIMS_RFWRT(7,106);   // 106   0x68
```

```
}

/*****************************************************************************
function   :Code_Manchester_Frame()
parameters  :void
returns     :void
type        :low level c
description:    This function code the RF datagram in Manchester

*****************************************************************************/
void Code_Manchester_Frame(unsigned char *ptr)
{
signed char ii;

 for (ii=7;ii>-1;ii--)

       {    if ((*ptr)&(1<<ii))   // If bit is 1 = 10 ---> 2

              Temp_Word = (Temp_Word<<2) + 2;

              else                // If bit is 0 = 01 ---> 1

              Temp_Word = (Temp_Word<<2) + 1;

       }

       High_Manch=Temp_Word>>8;

       Low_Manch=Temp_Word;
}
```

3.3 X and Z Accelerometer

Acceleration X and Z Acquisition

The purpose of the X and Z-axis g cells is to allow the tire recognition with appropriate algorithms analyzing the rotating signal caused by the Earth's gravitational field.

Motion will use either the Z-axis g-cell to detect acceleration level or use the X-axis g-cell to detect a ±1 g signal caused by the Earth's gravitational field.

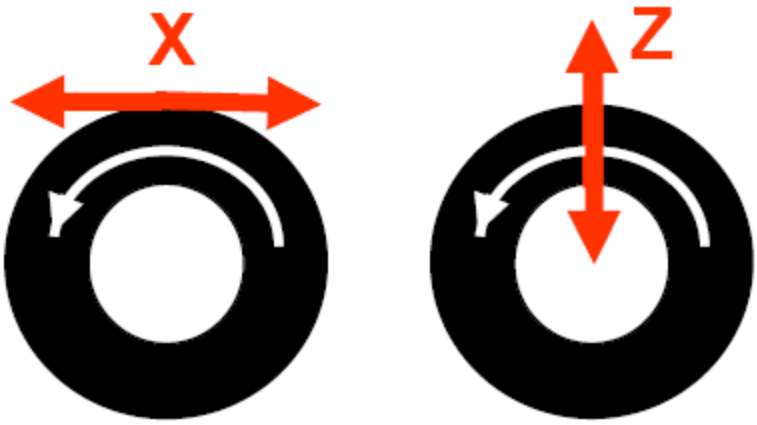

Figure 3-8. X and Z-Axis Sensing Direction

Figure 3-9 and Figure 3-10 show the response of the X and Z-axis sensor when the tire is in motion. They also show the sensors response to a continuous acceleration. The signal frequency for both axes increases with the speed. Thanks to the sensitivity of the sensor, some very simple algorithms allow to differentiate the Parking from the Motion state.

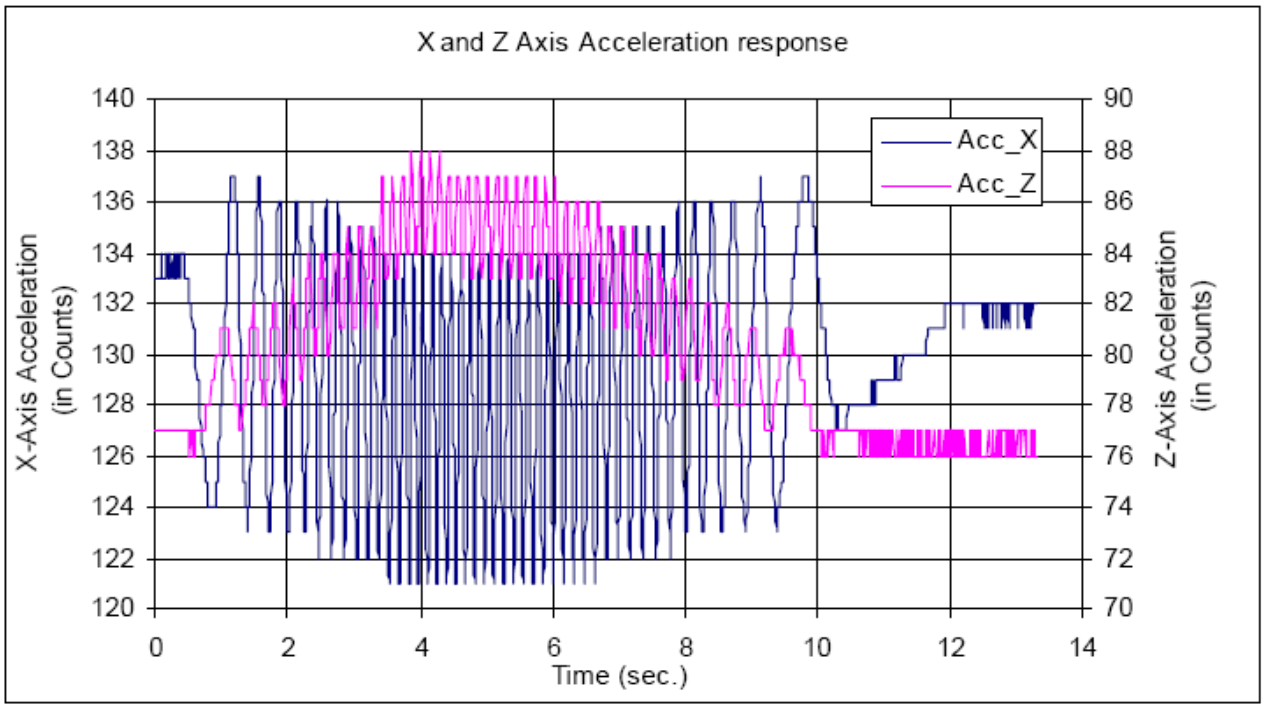

Figure 3-9. X and Z-Axis Response in a Change of Speed

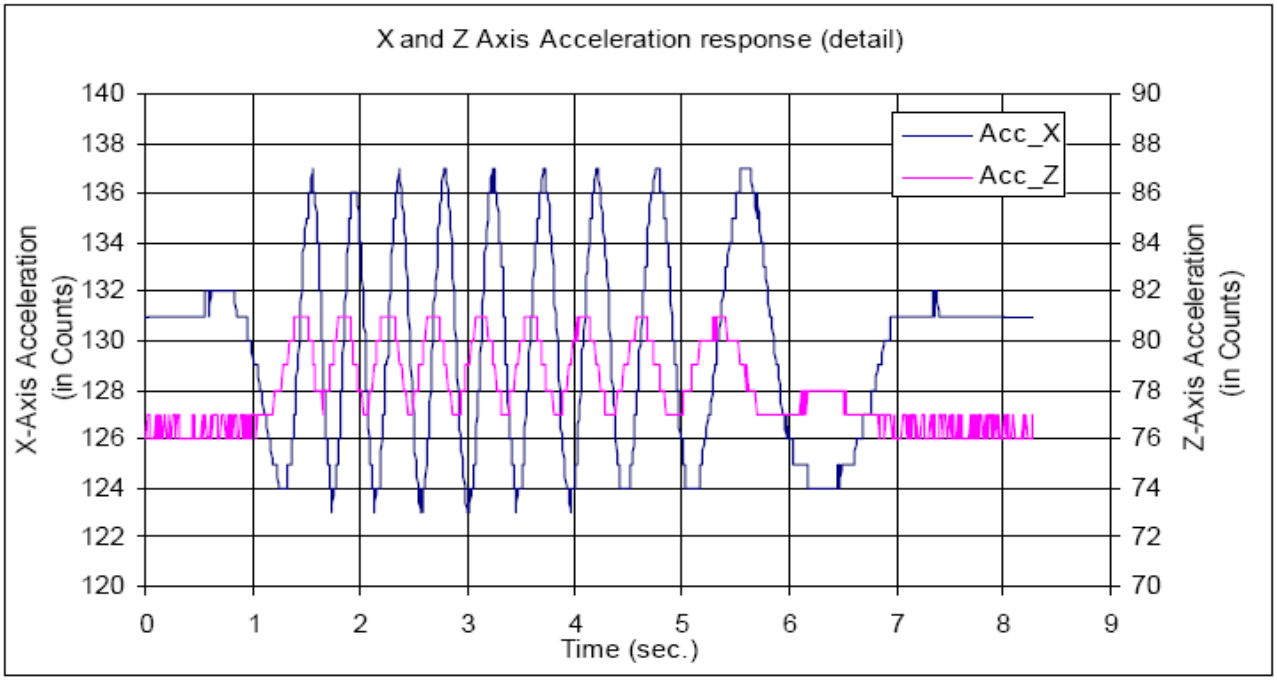

Figure 3-10. X and Z-Axis Response With Speed in the Range of 1-2 ms-1

Chapter 4
Firmware Function Example of Use

The purpose here is to become quickly familiar with the use of the Firmware function.

```
/*****************************************************
 Variable declaration used in the examples
 *****************************************************/
unsigned char Voltage,Temperature,Pressure,readReg,pnew1,Acc_X,Acc_Z,Power;
unsigned char static RFRD_data[16];
unsigned char RF_Red_Register[13];
unsigned char RFRD_SETUP_data[4];
unsigned char RFRD_SETUP_data_ReadBack[4];
unsigned int Pressure_Array[3];
unsigned int Acc_X_Array[4];
unsigned int Acc_Z_Array[4];
unsigned int Acc_XZ_Array[4];
unsigned int Data_Array[4];
unsigned int Value;
unsigned char ReadingNew;
unsigned int AveragOLD;
unsigned char LFDatagram[6];
unsigned char Temp8BitData;
unsigned int a,b,aa;
```

Firmware Function type	Example of use
Master Reset	REIMS_RESET() ; this subroutine is the one called during an initial power up of the device. it can be called again as a master reset. It resets SP, RFX, and loads ACI trim codes.
Acceleration Data Trim	REIMS_ACITD(); Loads default acceleration trim data values into the ACI
Voltage Acquisition	REIMS_READ_COMP_VOLTAGE(&Temp8BitData); Voltage (8 bit) =Temp8BitData;
Temperature Acquisition	REIMS_READ_COMP_TEMP_8(&Temp8BitData); Temperature (8 bit) =Temp8BitData;
Pressure acquisition	REIMS_READ_COMP_PRESSURE(&Pressure_Array[1],8); Pressure (8 bit) = Pressure_Array[1]>>1; Note: possible values are only 1, 2, 4, 8, 16, or 32. Recommended setting for accuracy is 8
X Acceleration Acquisition	REIMS_READ_COMP_ACCEL_X(&Acc_X_Array,Data_Array[1],1,0); Acc_X (8bit)=Acc_X_Array[1]>>2; Possible values for averaging are 1, 2, 4, 8, 16, or 32. Recommended setting is 1.
Z Acceleration Acquisition	REIMS_READ_COMP_ACCEL_Z(&Acc_Z_Array,Data_Array[1],1,0); Acc_Z (8 bit)=Acc_Z_Array[1]>>2; Possible values for averaging are 1, 2, 4, 8, 16, or 32. Recommended setting is 1.
Reset the RFX	REIMS_RF_RESET();

Read One RF register	RF_Red_Register[1]=REIMS_RFRD(2); // Read RFX add2 contents and
	store it in RF_Red_Register[1] array.

Read Consecutive RF Registers	REIMS_RF_READ_REGISTER(4,&RFRD_SETUP_data_ReadBack,0) Read 4 data's from the RFX buffer, starting at address 0 (address 0 corresponds to BUFF0), and write those 4 data's to the RFRD_SETUP_data_ReadBack
Write One RF Register	REIMS_RFWRT(0,24); Write 24 (in decimal) at address 0
Write Consecutive RF Registers	REIMS_RF_WRITE_REGISTER(4,&RFRD_SETUP_data,0); Write 4 data's from RFRD_SETUP_data to the RFX buffer(address 0 corresponds to BUFF0)
Read RFX Data Buffer	REIMS_RF_READ_DATA(5,&RF_Red_Register, 0); Read 5 data from RF data Register (address 0 corresponds to BUFF0) and write those 5 data's LSB first to RF_Red_Register table.
Read RFX Data Buffer with Bit Order Transposed	REIMS_RF_READ_DATA_REVERSE(5,&RF_Red_Register, 0); Read 5 data from RF data Register (address 0 corresponds to BUFF0) and write those 5 data's MSB first to RF_Red_Register table. The data bit order is transposed.
Write RFX Data Buffer	REIMS_RF_WRITE_DATA(5,&RFRD_data, 0); Read 5 data from RFRD_data table and write those 5 data's LSB first to the RFX buffer at BUFF0 (address 0 corresponds to BUFF0)
Write RFX data buffer with bit order transposed	REIMS_RF_WRITE_DATA_REVERSE(5,&RFRD_data[0],0); Read 5 data from RFRD_data table and write those 5 data's MSB first to the RFX buffer at BUFF0 (address 0 corresponds to BUFF0). The data bit order is transposed.
Switch ON the XCO	REIMS_RF_XCO_ON(1); 1 for ON, 0 for OFF
Send RF data buffer	REIMS_RF_SET_TX(0,96); Sent 96 bits of data,. Value 0 configures the charge pump (0 means charge pump off)

Weighted Average with R=2,4,8,16,32	REIMS_WAVG_R (word pold, byte pnew) This subroutine calculates a new weighed average value for any given new and old measurement readings by using the following method AverageNew = (AveragOLD * (R-1) + ReadingNew)/R Value=REIMS_WAVG_R(AveragOLD, ReadingNew);
LFO Calibration	REIMS_LFOCAL(); Temp8BitData= REIMS_LFOCAL() ; Resulting value for use in the WDIV0:5 bits is stored in Temp8BitData giving a WCLK period of 1 second
Flash Write	REIMS_FLASH_WRITE(0xc4b0,RFRD_data,4); RFRD_dat[0…3] must be defined locally so they are place before 0x200.Otherwise they will be erased by the program when running.
Check Bounding Wires	REIMS_WIRE_CHECK();
Read identification codes	REIMS_READ_ID(CODE);
Enable LF for Carrier or Data Detection	REIMS_LF_ENABLE(1); 1 for LF ON, 0 for LF 0FF
Reading LF Data	REIMS_LF_READ_DATA(&LFDatagram,LF_AGC,6);
Signed Multiplication	REIMS_SWORD_MULT(&aa,a,b); Result: aa=150000 (long) With a=300;b=500; (integers)
Square Root	b=REIMS_SQUARE_ROOT(a); With a=169 the result is 130 (result multiplied by 10 per definition)
CRC with 8 Bits Polynomial	uu=REIMS_CRC8(0x07,&RFRD_data,32); 0x07 is the number of bit at 1 in the polynom. In this case P=X8+X2+X+1 (X8 is always 1 and is not counted)
Initialization of the Serial Communication	REIMS_MSG_INIT (void)
Writing Data on Emulated Serial Interface	byte _REIMS_MSG_WRITE (byte Data) This function is in charge to write a message at a network level via an emulated serial interface on CLK and DTA (PTA5 and 6). As the master, the module must manage the clock on PTA5. On rising edge of the clock, the module puts down a new data bit on PTA6 (programmed as output), MSB first. Use for Device testing only.Allow to access to some REIMS parameters
Reading Data from Emulated Serial Interface	byte _REIMS_MSG_READ (void) This function is in charge to read any incoming message at a network level via an emulated serial interface on CLK and DTA (PTA5 and 6). As the master, the module must manage the clock on PTA5. On falling edge of the clock, the module puts reads a new data bit on PTA6 (programmed as input), MSB first. REIMS_READ_COMP_ACCEL_XZ(&Acc_Z_Array,Data_Array[1],1,0);
Takes X Acceleration Followed by Z Acceleration	REIMS_READ_COMP_ACCEL_XZ(&Acc_XZ_Array, Data_Array[1],4,0); Acc_X=Acc_XZ_Array[1]>>2; Acc_Z=Acc_XZ_Array[3]>>2;
Calculates a Checksum	ee=REIMS_CHECKSUM_XOR(&Acc_Z_Array,3,0);
Sets the RF Power Dynamically	REIMS_RF_DYNAMIC_POWER(Temperature,Voltage,Power);
Set Offset Levels for X or Z	REIMS_ACCEL_DYNAMIC_OFFSET(0x33,1); Z-axis offset shifted by 51 (decimal)
Send a Manchester-encoded Datagram	REIMS_RF_DIRECT_TX(1952,0,&RFRD_data[0],1); 1952 bits from RFRD_data sent, Baud rate =9600, Preconditioning set
Charge Pump Filling	REIMS_RF_CP_SET(3); Argument is always 3

Configures the RFX	REIMS_RF_CONFIG_DATA(&RF_CONFIG); Format is as follows: MSBit - 16-bits PLLB value 16 bits - PLLA value 1 bit - EOM 1 bit - Data Format 4 bits - not used 1 bit - Frequency 1 bit - Modulation If RF_CONFIG= {0x12, 0x34, 0x56, 0x78, 0xC3, 0x12} PLLB = $1234, (PLL0 = $6D, PLL1 = $38) PLLA = $5678, (PLL2 = $86, PLL3 = $D0) EOM is set (BIT7 of RFCR1 is set). Data Format is set (BIT6 of RFCR2 is set. Also, by default, BIT7 and BIT5 are set), Frequency bit is set, (BIT2 of PLLCR3 is set) Modulation bit is set. (BIT1 of PLLCR3 is set) Prescaler value is $12 (RFCR0 = $12)
Polls the CP Bit Once	REIMS_RF_CP_POLL();
Sets or Clears the Discharge Bit	REIMS_RF_DISCHARGE(1); 1 or higher set the discharge bit, 0 clears the discharge bit

Chapter 5
Application Source Code

5.1 MPXY8300 Module Source Code

```
/*****************************************************************************
                                                    Copyright (c) Freescale 2008

File Name        :  MPXY8300 TPMS KIT.c

Authors          : Rudi Lenzen

Locations        :  Toulouse (France)

Date Created     :  June-2008 - Issue 0

*****************************************************************************/

/*
 ****************************************************************************
 *
 *  Main.c - Code will reside in here.
 *
 ****************************************************************************
 */
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "main.h"
#include "reims_firm_io_ES2.h"


#pragma CODE_SEG DEFAULT_ROM

/*************************************************
 Variable declaration
 Note : Some variables may not be used in this program
 *************************************************/
unsigned char static RFRD_data[14];

unsigned char PPRESSURE,PPRESSUREOLD,PTEMPERATURE,PVOLTAGE,PACCX,PACCZ;


unsigned int Pressure_Array[3];
static unsigned int Acc_X_Array[4];
static unsigned int Acc_Z_Array[4];

unsigned char Pressure_Results[4];
unsigned char Temperature_Results[4];
unsigned char Voltage_Results[4];


unsigned char LFlevel,ii=0;
unsigned char LFDatagram[6];
unsigned char Temp8BitData;
```

```
unsigned int  Temp9BitData;

unsigned char kk=0;
unsigned char temp_b01119;

UINT8 _REIMS_INTERRUPT_FLAG;
static unsigned char mm;


/*
 ***************************************************************************
 *
 *                          M A I N
 *
 ***************************************************************************
 */
#pragma CODE_SEG DEFAULT

void main(void)
{

if (SPMSC2_PDF==0)

    {

  /*****Initialize the PARAM 0-31 memory here after POWER-UP*****/

  }

  SRS=0x00;   //The COP timer is reset by writing any value to the address of
              //SRS. This write does not affect the data in the read-only SRS. Instead,
the act of writing to
              //this address is decoded and sends a reset signal to the COP timer.

   __RESET_WATCHDOG();

EnableInterrupts;
SOPT1=0x33;

/*****************Main Time Base for the whole application***********************/

    PWUSC0_WUFACK=1;
    PWUDIV = 0x2F;  //    EVERY SECOND
    PWUSC0 = 0x02;//    PERIODIC WAKE-UP EVERY 2 sec.
    PWUSC1 = 0x00;//    NO PERIODIC RESET

 /*****************************************************************************/

    InitLF();

    RF_Setup();

    SPMSC1_BGBE=1;  //Enable Bandgap

    Measure_P_T_V();

    // Acceleration Z byte
```

```
   REIMS_READ_COMP_ACCEL_Z(&Acc_Z_Array,Pressure_Array[1],1,0);
   delay(10);
   PACCZ=Acc_Z_Array[1]>>2;

    // Acceleration X byte
   REIMS_READ_COMP_ACCEL_X(&Acc_X_Array,Pressure_Array[1],1,0);
   delay(10);
   PACCX=Acc_X_Array[1]>>2;


    Send_RF_Datagram();



/*****************************************************************************/

/*****SetSTOPMode (1) or (3)*****  -  MUST BE HERE BEFORE THE STOP INSTRUCTION */

   SOPT2=0x70;       //   RESET VALUES
   SPMSC1=0x10;      //  0x10 :  STOP 1 MODE CONFIGURED LVDE & LVDSE = 0
                      //  0x1D :  STOP 4 MODE CONFIGURED LVDE & LVDSE = 1

   SPMSC2=0x02;      //   02 ---> PDC = 1 ; PPDC = 0    STOP1
                      //  03 ---> PDC = 1 ; PPDC = 1    STOP2
                      //  00 ---> PDC = 0 ; PPDC = 0    STOP3

/*****************************************************************************/

     __asm STOP;

}


/*****************************************************************************
Function:delay
NOTES    :                                                    realize un delay
*****************************************************************************/

void delay(unsigned char r_max)
{
    unsigned char r,q;
    for(r=0; r<r_max; r++)
    {
         for(q=0; q<2; q++);
    }
}

/*****************************************************************************
function    :   InitLF()
NOTES    :                  This function initializes the Manchester detect Mode
*****************************************************************************/

void InitLF(void)

{
LFCS1_LFCFAK=1;
LFCS0_LFEN=0;
```

```
LFCR=0x00;
LFCS0=0x00; // LF0&1 disable  for autozero
LFCS1=0x00;
LFCR=0x86;                                          //86 sampling every 128msec

LFCA0=0xAA;//Wake-up codes defined by user                                  LSB
LFCA1=0xF6;//Wake-up codes defined by user                                  MSB
LFCS0=0x48;    //  0x48 : Detection Level = 3.5mV , LF0 Enable , Tlfrst duration = 1*TLFS
         //  Interrupt Enable
                        //  LF data from Manchester Decoder

}
/*****************************************************************************
function   :RF_Setup(void)
parameters  :void
returns     :void
type        :low level c
description:    RF Setup at 315 MHz
*****************************************************************************/

void RF_Setup(void)

{

    REIMS_RFWRT(0,25);//RFCR0=0x18 (24) ---> 9600 Baud , Manchester Coding , POL =0

REIMS_RFWRT(1,240);//240 (128+112) ---> EOM set , Frame Bit Length= 14*8 = 112
REIMS_RFWRT(2,131);             //
     REIMS_RFWRT(3,14);//   --->    14= +5dBm
     REIMS_RFWRT(4,171);   // 171   0xAB    for FSK @ 315 MHz   at +/-35KHz
     REIMS_RFWRT(5,80);    // 76    0x48      "          "
     REIMS_RFWRT(6,172);   // 172   0xAC             "
     REIMS_RFWRT(7,106);   // 106   0x68
}
/*****************************************************************************
Function: Measure_P_T_V
NOTES    :   This function measures Pressure, Temperature, and Voltage and stores them in
RFRD_data
*****************************************************************************/
void Measure_P_T_V()
{

          DisableInterrupts;
           _REIMS_INTERRUPT_FLAG=0x00;

          REIMS_READ_COMP_PRESSURE(&Pressure_Array[1],4);
          PPRESSURE = (byte)Pressure_Array[1]>>1;
          REIMS_READ_COMP_TEMP_8(&Temp8BitData);
          PTEMPERATURE = Temp8BitData;
          REIMS_READ_COMP_VOLTAGE(&Temp8BitData);
          PVOLTAGE = Temp8BitData;
          EnableInterrupts;

}
/*****************************************************************************
Function: Send_RF_Datagram
NOTES    :   This function sends the RF datagram based on:
```

```
            SYNC1,SYNC2,SENSE TYPE,ID1-4,Pressure,Temperature,Voltage
*************************************************************************/
void Send_RF_Datagram()

{  unsigned char i,j;

for (i=0;i<15;i++)

    {
  RFRD_data[i]=0;

    }
// Two init byte - DO NOT CHANGE IT!
RFRD_data[0]= 0xFF;
RFRD_data[1]= 0xFE;

// FSL byte - DO NOT CHANGE IT P+HEADER

RFRD_data[2]= 0x0F; //
  //  ID 4 bytes - these bytes can be change - Module 1
RFRD_data[3]= 0xA1; //ID1 = A1
RFRD_data[4]= 0xB2; //ID2 = B2
RFRD_data[5]= 0xC3; //ID3 = C3
RFRD_data[6]= 0xD4; //ID4 = D4

// Message type byte - can be change
RFRD_data[7]= 0x06;

RFRD_data[8]= PPRESSURE;
RFRD_data[9]= PTEMPERATURE;
RFRD_data[10]=PVOLTAGE;
RFRD_data[11]=PACCZ;
RFRD_data[12]=PACCX;

/************************************************/
RFRD_data[13]=CheckSum_FSL_Full(&RFRD_data[2]);    //Checksum without two first bytes
(both is constants)
REIMS_RF_WRITE_DATA_REVERSE(14,&RFRD_data[0],0);
REIMS_RF_SET_TX(240);   //240 for 14 bytes sent
}

/*************************************************************************
Function: LF_OFF
NOTES    :   This function switches the LF OFF
*************************************************************************/
void LF_OFF()
{
LFCS0_LFEN=0;     //Mandatory before any other writings
LFCS0 = 0x00;  // LF0&1 disable  for autozero
}

/*************************************************************************
Function: Decode_LF_Datagram  -
NOTES    :                                    LF data decoding through Interrupt
*************************************************************************/

void Decode_LF_Datagram()
```

```
{unsigned char ii;
for (ii=0;ii<=5;ii++)

   {
            {
            while((LFCS1 & 0x80)==0);

                       {
                        LFDatagram[ii]=LFDATA;
                        LFCS1=0x40;                // clear the flag
                       }
            }

          if (ii==5)

          {
            LF_OFF();
            break;
          }
   }
 if (LFDatagram[0]==0xF3 && LFDatagram[1]==0xA8 && LFDatagram[2]==0xB2 &&
LFDatagram[3]==0x03 && LFDatagram[4]==0x4F && LFDatagram[5]==0x62) //LF0

          {

          /***Put your code here****/

          }

else if (LFDatagram[0]==0xA1 && LFDatagram[1]==0xB2 && LFDatagram[2]==0xC3 &&
LFDatagram[3]==0xD4 && LFDatagram[4]==0x4D && LFDatagram[5]==0x3C) //LF1


          {

           /***Put your code here****/
          }

   else if (LFDatagram[0]==0xE1 && LFDatagram[1]==0xE5 && LFDatagram[2]==0xF3 &&
LFDatagram[3]==0xF5 && LFDatagram[4]==0xA1 && LFDatagram[5]==0xA5) //LF2


          {

          /***Put your code here****/
          }

   else if (LFDatagram[0]==0xB1 && LFDatagram[1]==0xB5 && LFDatagram[2]==0xC1 &&
LFDatagram[3]==0xC5 && LFDatagram[4]==0x02 && LFDatagram[5]==0xAA) //LF3


          {
          /***Put your code here****/
          }
```

```
 else if (LFDatagram[0]==0xD3 && LFDatagram[1]==0xD6 && LFDatagram[2]==0xE4 &&
LFDatagram[3]==0xE8 && LFDatagram[4]==0x4F && LFDatagram[5]==0x02) //LF4


          {
          /***Put your code here****/
          }

}


/*****************************************************************************
Function: CheckSum_FSL_Full(byte *array)
NOTES    :         /* Compute checksum - has to be compliant to ESTAR receiver */
/*****************************************************************************/

byte CheckSum_FSL_Full(byte *array)
{
  byte j;
  byte ch=0;
  for(j=0;j<13;j++)  // 13 - length of dataframe
  {
    ch += *(array++);
  }
  ch = 0xff - ch; //  return(ch);
}

/*
 ******************************************************************************
 *
 *  End of file.
 *
 ******************************************************************************
 */

Functions prototypes included in main.h:

/*
 ******************************************************************************
 *
 * Function Prototypes
 *
 ******************************************************************************
 */
void main(void);
void delay(unsigned char);
void Decode_LF_Datagram(void);
void InitLF(void);
void RF_Setup(void);
void Measure_P_T_V(void);
void Send_RF_Datagram(void);
void LF_OFF(void);
byte CheckSum_FSL_Full(byte *array);
#pragma CODE_SEG DEFAULT
#endif /* MAIN_INCLUDED */
```

5.2 LF125KHz source code

```
/*****************************************************************************
                                                    Copyright (c) Freescale 2008

File Name        :  LF125KHz Emitter Module 2008-GB60A.c
Authors          : Rudi Lenzen
Locations        :  Toulouse (France)

Date Created     :  June-2008 - Issue 0
*****************************************************************************/
#include "LF125KHz Emitter Module 2008-GB60A.h"
#include <MC9S08GB60.h> /* include peripheral declarations */
#include <hidef.h> /* for EnableInterrupts macro */
#include <stdio.h>
#include <math.h>

/*****************************************************************************/

/*Global Variables (could be static)

    The following variables are defined as global. They are only used
    in specific functions and could also be declared as static. Global
    variables are more useful for debugging. The contents of global
    variables are listed in the debugger, static variables are not
    listed until the function is entered. These variables are also
    initialized manually by the init function.

/*****************************************************************************/

unsigned char readReg;
unsigned short Temp_Word;
unsigned static char Var1=0;

/*****************************************************************************
Function    :   main()
NOTES    :
/*****************************************************************************/

void main(void)

{  initMCU();
   initSPI();
   IRQSC=0x13;                                        /*IRQ interrupt activated */
   IRQSC_IRQACK=1; /*Clear IRQ flag*/
   PTDD_PTDD1=1; /* 1 on MODE2 - MC33690 = ON */
   EnableInterrupts;
  delay(100);
   while(1)

           {
         Led1();
          delay(100);
          }
```

```
}
/*****************************************************************************
Function:delay
NOTES    :                                                        Create a delay
*****************************************************************************/
void delay(unsigned char r_max)
{
    unsigned char r,q;
     for(r=0; r<r_max; r++)
    {
        for(q=0; q<200; q++);
    }
}

/*****************************************************************************
function    :   initMCU()
NOTES    :                    This function initializes the MCU config registers,
                                    clock (CGMXCLK or Fbus = 8MHz), ports,...
*****************************************************************************/

void initMCU(void)

{
     SOPT=0x13;
   SPMSC1=0x50;
    SPMSC2=0x00;
     ICGC1=0x74;
     ICGC2=0x08;
     while((ICGS1 & 0x80)==0x00);   /*wait for clock FLL bypassed (external reference)
to be locked */
     while((ICGS1 & 0x02)==0x00);  /*wait for External Reference Clock Status to be set */
     while((ICGS1 & 0x20)==0x00);   /*wait Reference Clock Status to be set */

    /*********************** config I/O *****************/


   PTFDD_PTFDD0=1; /* PTF0 configured as an Output */
   PTFDD_PTFDD1=1; /* PTF1 configured as an Output */
   PTADD_PTADD6=0; /*PTA6 configured as an Input */
   PTADD_PTADD7=0; /*PTA7 configured as an Input */
   PTADD_PTADD2=0; /*PTA2 configured as an Input */
   PTADD_PTADD3=0; /*PTA3 configured as an Input */

   /*J9 switch configuration*/
   ATD1C=0;
   ATD1SC=0;
   ATD1PE=0;
   PTBD=0x00;
   PTBPE=0xFF;
   PTBDD=0xFF;  //PTB configured as Outputs
   delay(10);
   PTBPE=0x00;
   delay(10);
   PTBDD=0x00;  //PTB configured as Inputs

   PTBD=0x00;
```

```
    PTCD_PTCD4=1;    /* Set PTC4 High */    //LED 1 OFF
   PTCD_PTCD5=1;    /* Set PTC5 High */    //LED 2 OFF


   PTCDD_PTCDD2=1; /* PTC2 configured as an Output */
   PTCDD_PTCDD3=0; /* PTC3 configured as an Input */
   PTCDD_PTCDD4=1; /* PTC2 configured as an Output */
   PTCDD_PTCDD5=1; /* PTC2 configured as an Output */
   PTDDD_PTDDD0=1; /* PTD0 configured as an Output */
   PTDDD_PTDDD1=1; /* PTD1 configured as an Output */
   PTDDD_PTDDD2=0; /* PTD2 configured as an Input */


         /* MC33690 Initialization */

   PTDD_PTDD1=1; /* 1 on MODE2 - MC33690 = ON */
   PTDD_PTDD0=1; /* 1 on LVR - RESET Pin */
   delay(50);
   PTDD_PTDD0=0; /* 0 on LVR - RESET Pin */
   PTDD_PTDD1=0; /* 0 on MODE2 - MC33690 = Standbye Mode */
}
/*************************************************************************
function   :initSPI()
parameters  :void
returns     :void
type        :low level c
description:    This function initializes the SPI
*************************************************************************/
void initSPI(void)
{
SPI1C1=0x54;     /*Interrupt Disabled , SPI Enable , Transmit Interrupt Disable ,
                  Master Mode , active high clock selected ,
              First clk edge is issued at the beginning of the 8 cycle transfer operation ,
                  SS pin define as standard I/O , MSB first */

SPI1C2=0x0A;
SPI1BR=0x66;       /*SPI Clk = 8.22 KHz >> MOSI data 4.114 KHz*/

}
/*************************************************************************
function   :initKBI()
parameters  :void
returns     :void
type        :low level c
description:    This function initializes the KBI
*************************************************************************/
void initKBI(void)

{
  PTAPE_PTAPE2 = 1;
  PTAPE_PTAPE3 = 1;
  PTAPE_PTAPE6 = 1;
  PTAPE_PTAPE7 = 1;
  KBI1PE_KBIPE2 = 1;      /* KBI2 and KBI3 activated as Interrupt Pins */
  KBI1PE_KBIPE3 = 1;
  KBI1PE_KBIPE6 = 1;
  KBI1PE_KBIPE7 = 1;
```

```
  KBI1SC=0x03;      /* KB IR Not masked,No pending Interrupt and KB IR occurs on failing
Edge and Low level*/
  KBI1SC_KBIMOD=0;  /* 0 allows KBF clear when KBACK=1 */
  KBI1SC_KBIE=1;
}

/*************************************************************************
function   :MOSI_byte_SPI()
parameters  :void
returns     :void
type        :low level c
description:    This function sends a byte to the SPI pin
*************************************************************************/
void MOSI_byte_SPI(unsigned char byte)
  {
while (SPI1S_SPTEF==0); /*Wait for SPI transmit buffer empty */
asm
   {
    lda _SPI1S.Byte;/*SPI1SR reading for SPIF and SPTEF clear*/
    lda _SPI1D.Byte;
   }
SPI1D=byte;
}
/*************************************************************************
function   :Sent_Manchester()
parameters  :void
returns     :void
type        :low level c
description:    This function sends the Manchester values to the MC33690
*************************************************************************/

void Sent_Manchester()

{
unsigned char i,ii;
unsigned char array1[8] = {0xF6,0xAA,0xF3,0xA8,0xB2,0x03,0x4F,0x62};    //LF0
unsigned char array2[8] = {0xF6,0xAA,0xA1,0xB2,0xC3,0xD4,0x4D,0x3C};    //LF1
unsigned char array3[8] = {0xF6,0xAA,0xE1,0xE5,0xF3,0xF5,0xA1,0xA5};    //LF2
unsigned char array4[8] = {0xF6,0xAA,0xB1,0xB5,0xC1,0xC5,0x02,0xAA};    //LF3
unsigned char array5[8] = {0xF6,0xAA,0xD3,0xD6,0xE4,0xE8,0x4F,0x02};    //LF4

DisableInterrupts;
PTDD_PTDD1=1; /* 1 on MODE2 - MC33690 = ON */
for (ii=0;ii<120;ii++)

      {MOSI_byte_SPI(0xAA);    //sent 480 bits = 0
      }

/***************** LF SYNCHRO  ***********************/

MOSI_byte_SPI(0xAB);
MOSI_byte_SPI(0x8B);
MOSI_byte_SPI(0x32);


/***************** LF DATA  ***********************/
```

```
 if (PTBD_PTBD5==1)       //ID for LF0
                       {
                       for (i=0;i<9;i++)
                       Code_Manchester_Send_Frame(&array1[i]);
                       }

  else if (PTBD_PTBD4==1) //ID for LF1

                      {
                       for (i=0;i<9;i++)
                       Code_Manchester_Send_Frame(&array2[i]);
                       }

  else if (PTBD_PTBD3==1) //ID for LF2

                       {
                      for (i=0;i<9;i++)
                      Code_Manchester_Send_Frame(&array3[i]);
                       }

else if (PTBD_PTBD2==1) //ID for LF3

                       {
                      for (i=0;i<9;i++)
                      Code_Manchester_Send_Frame(&array4[i]);
                        }

else if (PTBD_PTBD1==1) //ID for LF4

                       {
                      for (i=0;i<9;i++)
                      Code_Manchester_Send_Frame(&array5[i]);
                        }


  else                 {
                          }

}

/*****************************************************************************
function    :   Led1()
parameters  :   void
returns     :   void
type        :   normal
Description:

*****************************************************************************/
void Led1 ()
{
   PTCD_PTCD4=0;    /* Set PTC4 LOW */    //LED 1 ON
   delay(50);
   PTCD_PTCD4=1;    /* Set PTC4 HIGH */    //LED 1 OFF
}
/*****************************************************************************
```

```
function    :   Led2()
parameters  :   void
returns     :   void
type        :   normal

Description:

*****************************************************************************/
void Led2 ()

{   PTCD_PTCD5=0;    /* Set PTC5 LOW */    //LED 2 ON
    delay(200);
    PTCD_PTCD5=1;    /* Set PTC5 HIGH */    //LED 2 OFF
    delay(200);
 }
/*****************************************************************************
function    :   SentManchester_N_Times()

Description:
*****************************************************************************/
void SentManchester_N_Times()

{  PTCD_PTCD2=1; /* 1 on PTC2 - Tx=1 , Output driver = ON */
    IRQSC_IRQACK=1; /*Clear IRQ flag*/

   /* Repeat this function to send additional Manchester frames */

   Sent_Manchester();
     PTCD_PTCD2=0; /* 1 on PTC2 - Tx=1 , Output driver = OFF */
   if(PTBD_PTBD5==1)
      {
   IRQSC_IRQACK=1; /*Clear IRQ flag*/
   Led2();
  }
     else if(PTBD_PTBD4==1)
   {
   IRQSC_IRQACK=1; /*Clear IRQ flag*/
   Led2();
   Led2();

}
     else if(PTBD_PTBD3==1)
   {
   IRQSC_IRQACK=1; /*Clear IRQ flag*/
   Led2();
   Led2();
   Led2();
 }


   else if(PTBD_PTBD2==1)
   {
   IRQSC_IRQACK=1; /*Clear IRQ flag*/
   Led2();
   Led2();
   Led2();
   Led2();
```

```
}


   else if(PTBD_PTBD1==1)

   {IRQSC_IRQACK=1; /*Clear IRQ flag*/
   Led2();
   Led2();
   Led2();
   Led2();
   Led2();
   Var1=0;
   }

   else{
   }

delay(250);

  asm {rti;}
}


/*************************************************************************
function   :Code_Manchester_Send_Frame()
parameters  :void
description:
This function sends the Manchester values to the MC33690
*************************************************************************/
void Code_Manchester_Send_Frame(unsigned char *ptr)
{
signed char ii;
for (ii=7;ii>-1;ii--)

       {    if ((*ptr)&(1<<ii)) // If bit is 1 = 01
              Temp_Word = (Temp_Word<<2) + 0x1;
              else                // If bit is 0 = 10
              Temp_Word = (Temp_Word<<2) + 0x2;
       }

       MOSI_byte_SPI((unsigned char)(Temp_Word>>8));
       MOSI_byte_SPI((unsigned char) Temp_Word);
}



/*************************************************************************
                                                  Copyright (c) Freescale 2008
File Name        :  LF125KHz Emitter Module 2008-GB60A.h
Author          : Rudi Lenzen
Locations        :  Toulouse (France)
Date Created     :  June-2008 - Issue 0
*************************************************************************/

/*************************************************************************

function prototypes
```

```
/*****************************************************************************/
void main(void);
void initMCU(void);
void delay(unsigned char byte);
void Sent_Manchester(void);
void SentManchester_N_Times(void);
void initSPI(void);
void initKBI(void);
void MOSI_byte_SPI(unsigned char byte);
void Code_Manchester_Send_Frame(unsigned char *ptr);
void initTPM1CH0(void);
void Led1 (void);
void Led2 (void);

/*****************************************************************************/
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150

MPXY8300RM
Rev. 2
12/2008

freescale™
semiconductor