FXTH87 Family Evaluation Design Reference Manual

Devices Supported:
FXTH870511
FXTH870911
FXTH871511

Document Number: FXTH87EDRM
Rev. 1.0, 09/2015

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/salestermsandconditions.

Document Number: FXTH87EDRM
Rev. 1.0
09/2015

Contents

Related Documentation

The FXTH87 device features and operations are described in a variety of reference manuals, user guides, and application notes. To find the most-current versions of these documents:

1. Go to the Freescale homepage at:

   http://www.freescale.com/

2. In the Keyword search box at the top of the page, enter the device number FXTH87.
3. In the Refine Your Result pane on the left, click on the Documentation link.

Chapter 1 Introduction and Setup

1.1 Introduction

The purpose of this document is to provide the end user with all the technical information required to become quickly familiar with the FXTH87 Family Evaluation Tire Pressure Monitoring Sensor (TPMS) devices.

The tool set of the TPMS kit consists of one FXTH870511 module for 450-kPa evaluations, one FXTH870911 module for 900-kPa evaluations, one FXTH871511 module for 1500-kPa evaluations and the associated software.

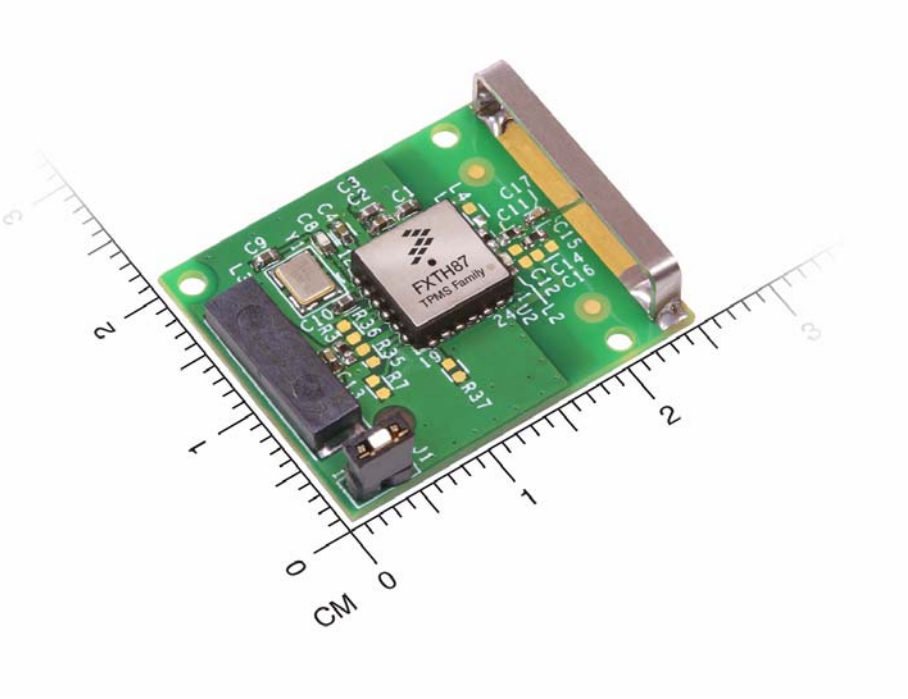

Figure 1-1. Tool set of TPMS

1.2 FXTH87 Family Tire Pressure Monitoring Sensor module 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)
- 64-bytes, low power, parameter registers
- Four GPIO pins with optional pullups/pulldowns and wake-up interrupt
- Real-time interrupt driven by low-frequency oscillator (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 timeout with selectable times and clock sources
- Two-channel general purpose timer/PWM module (TPM1)
- Internal oscillators
- MCU bus clock nominal of 0.5, 1, 2 and 4 MHz (1, 2, 4 and 8 MHz HFO)
- Low-frequency, low-power time clock (LFO) with 1 msec nominal period
- Low-frequency receiver (LFR) decoder and sensor clock (MFO) of 8 µsec nominal 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
- X- and Z-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
- 8-bit MCU programming capability through the four pins connector and the BDM multilink type
- Dedicated matching network circuit for optimal performances with the antenna
- Typical wireless range is 50 meters in air with a 5 dBm output power
- 3 V battery solder pads

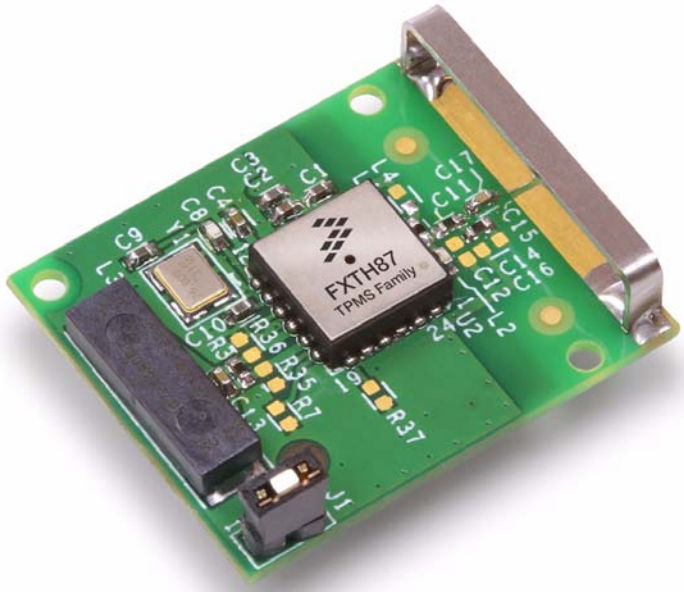

Figure 1-2. FXTH87 Module

1.3 Board programming guide

1.3.1 Downloading demo software to FXTH87 modules, LF 125-kHz emitter and TPMS receiver

The software project can be downloaded on the FXTH87 modules. The connection to the BDM Multilink is done through an interface socket for the FXTH87 modules and a P&E BDM Multilink interface (see Figure 1-3).

Figure 1-3. FXTH87 Family TPMS module

Chapter 2 Schematics and Bill of Materials

2.1 FXTH87 module evaluation boards (EVB)

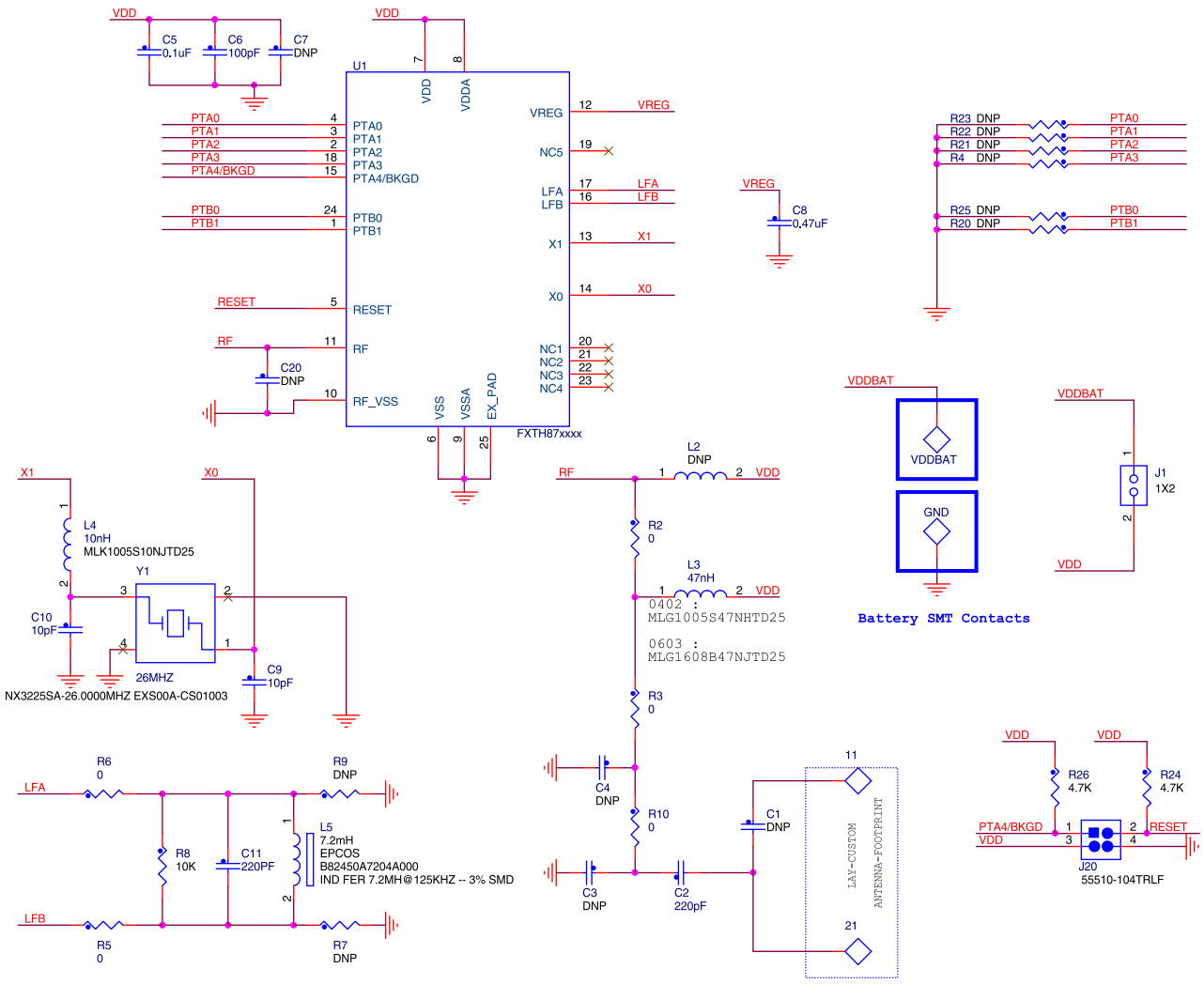

Figure 2-1. Typical reference demonstrator for FXTH87 315 MHz

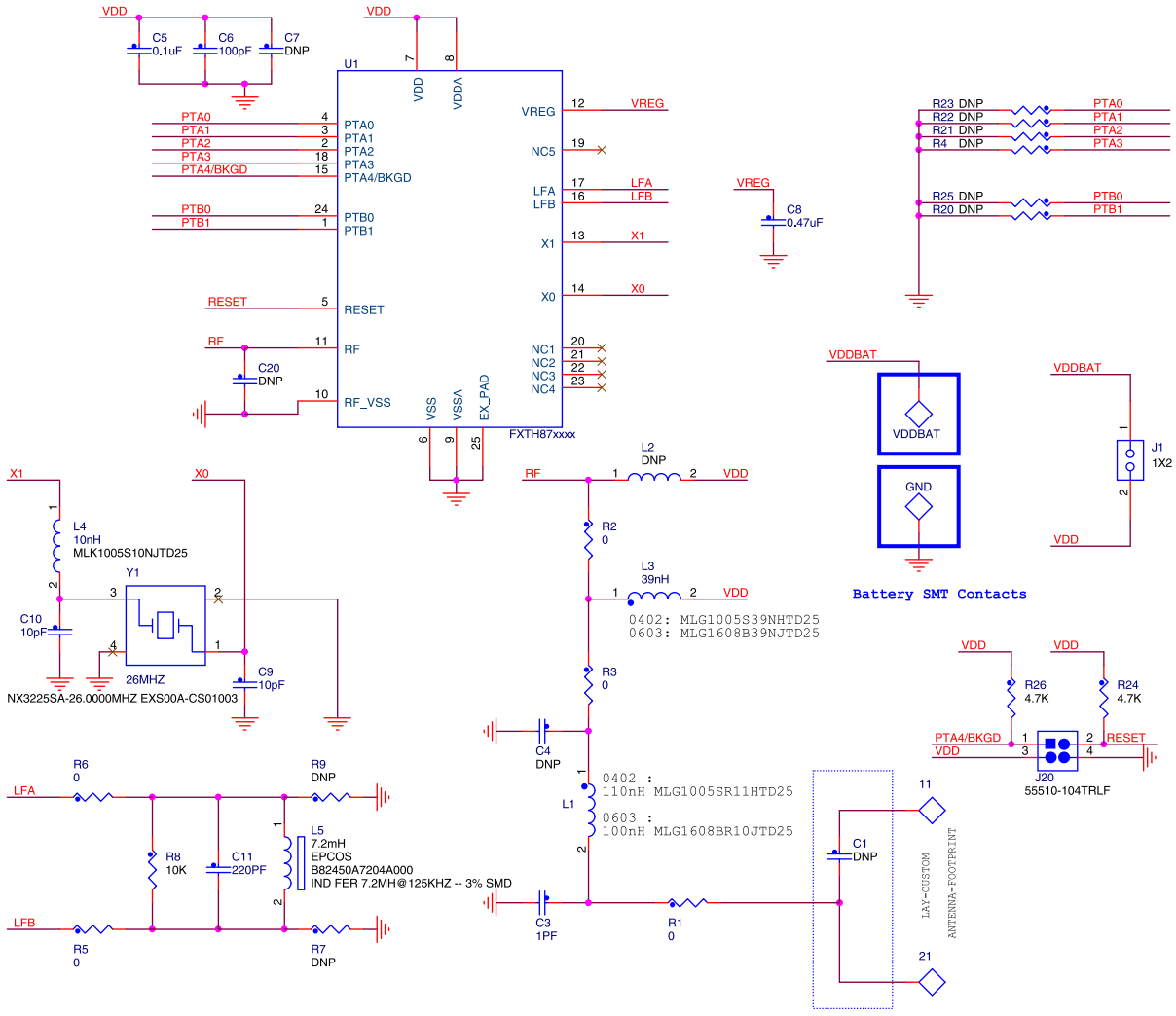

Figure 2-2. Typical reference demonstrator for FXTH87 434 MHz

L2 and R2 matching network values are applicable when using the hook antenna available on the module. Any change in the antenna characteristics will need a re-adjustment of the matching network.

Table 2-1. FXTH87 reference demonstrator bill of material (315 MHz and 434 MHz)

Ref. #	Value	Footprint	Freescale recommended preferred products	Supplier
Y1	26 MHz	QUARTZ_3225	NX3225SA-26.0000 MHz EXS00A-CS01003	NDK
C1	OPEN	SM/C_0402	—	—
C2	220 pF@315 MHz	SM/C_0402	—	—
C3	OPEN@315 MHz and 1 pF@434 MHz	SM/C_0402	—	—
C4	OPEN	SM/C_0402	—	—
C5	0.1 µF	SM/C_0402	—	—
C6	100 pF	SM/C_0402	—	—
C7	OPEN	SM/C_0402	—	—
C8	0.47 µF	SM/C_0402	—	—
C9	10 pF	SM/C_0402	—	—
C10	10 pF	SM/C_0402	—	—
C11	220 pF	SM/C_0402	—	—
ANTENNA	N/A	RF ANTENNA	N/A	Freescale design
J1	JUMPER	POWER JUMPER	—	—
J20	CON4	PROG. CONN.	—	—
L1	110 nH@434 MHz **0402** 100 nH@434 MHz **0603**	SM/C_0402 or SM/C_0603	MLG1005SR11HTD25 MLG1608BR10JTD25	TDK TDK
L2	OPEN	SM/C_0402	—	—
L3	**0402** and **0603** 47 nH @315 MHz 39 nH @434 MHz	SM/C_0402 or SM/C_0603	MLG1005S47NHTD25@315 MHz **0402** MLG1608B47NJTD25@315 MHz **0603** MLG1005S39NHTD25@434 MHz **0402** MLG1608B39NJTD25@434 MHz **0603**	TDK TDK TDK TDK
L4	10 nH	SM/C_0402	MLK1005S10NJTD25	TDK
L5	7.2 mH	LF COIL	B82450A7204A000	EPCOS
R1	0 Ω @434 MHz	SM/C_0402	—	—
R2	0 Ω	SM/C_0402	—	—
R3	0 Ω	SM/C_0402	—	—
R4	OPEN	SM/C_0402	—	—
R5	0 Ω	SM/C_0402	—	—
R6	0 Ω	SM/C_0402	—	—
R7	OPEN	SM/C_0402	—	—
R8	10 kΩ	SM/C_0402	—	—
R9	OPEN	SM/C_0402	—	—
R10	0 Ω @315 MHz	SM/C_0402	—	—
R20	OPEN	SM/C_0402	—	—

Table 2-1. FXTH87 reference demonstrator bill of material (315 MHz and 434 MHz) (continued)

Ref. #	Value	Footprint	Freescale recommended preferred products	Supplier
R21	OPEN	SM/C_0402	—	—
R22	OPEN	SM/C_0402	—	—
R23	OPEN	SM/C_0402	—	—
R24	4.7 kΩ	SM/C_0402	—	—
R25	OPEN	SM/C_0402	—	—
R26	4.7 kΩ	SM/C_0402	—	—
U1	FXTH870511 – 450 kPa	QFN 7 x 7	FXTH870511 – 450 kPa	Freescale
U1	FXTH870911 – 900 kPa	QFN 7 x 7	FXTH870911 – 900 kPa	Freescale
U1	FXTH871511 – 1500 kPa	QFN 7 x 7	FXTH871511 – 1500 kPa	Freescale

Chapter 3 FXTH87 Main Features and Specific Information

3.1 LF receiver

The LF block allows wake-up of the FXTH87 device from STOP1 (low-power) mode when a carrier or a Manchester datagram is detected. If the LF datagram is coded in Manchester, it will be decoded by the embedded LF state machine assuming the LF module has been configured appropriately. In addition, this LF receiver (LFR) system can autonomously listen for valid LF signals, check for protocol and ID information so the main MCU can remain in a very low-power standby mode until valid message data has been received. The LFR does not wake the MCU unless a valid message is being received and a data byte is ready to be read.

3.1.1 LF sampling frequency

Power consumption optimization is obtained with the LFR cycling between an OFF state, where everything is disabled, and an ON state, where it listens for a carrier signal.

When the LFR is listening for a carrier signal, only the internal 1-kHz clock source (LFO), a portion of the input amplifier and a periodic auto-zero circuitry are running. If a carrier signal with the correct frequency and higher-than-threshold amplitude is detected, and the internal oscillator to the LFR (LFRO) is enabled, the module begins to decode the incoming message. For adequate LF telegram sampling, a point to take into consideration is the accuracy of the sampling time directly linked to the accuracy of the 1-kHz LFO clock, which varies from 769 to 1428 Hz, according to the product specification.

It is therefore recommended to use the appropriate sampling timing with respect to the incoming LF signal duration. Sampling time is selectable through the LFCR register content.

As an example, if there is a request for sampling an incoming 16-msec carrier length every 16 msec (LFON = 1 msec), a hit rate of 100 percent will not be achieved.

- With LFO = 1428 Hz, the signal will be sampled every 16/1428 = 11.2 msec giving a high hit rate.
- With LFO = 769 Hz, the signal will be sampled every 16/769 = 20.8 msec giving a low hit rate.

For a 16-msec carrier length LF signal, the recommendation is to sample the signal every 8 msec with the LF sampling interval set continuously ON. This will give a sampling interval between 5.6 and 10.4 msec when taking into account the frequency of the LFO, improving the quality of the hit rate.

3.1.2 LF datagram recommended shape using the data mode

In data mode when a carrier is detected, the averaging filter is powered on and the LFR continues to the next state to look for the rest of a message telegram. The LFR module will search for a valid SYNC word (with length programmed through the SYNC bits in the LFCTL3 register depending on preamble type). If the external LF field is not a valid TPMS frame, a timeout will turn off the LFR module decoder, and will cycle back to the on-off detection scheme as described in Section 3.1.1, “LF sampling frequency”.

Data-clock recovery and synchronization takes place before or during the SYNC portion of an incoming message. The preamble can be nothing, a carrier only or modulated Manchester data. The type of required SYNC pattern determines the allowed preamble type depending on the SYNC[1:0] control bits.

The design data rate is 3.906 kbps which gives a bit time equivalent to about 32 cycles of the LF-carrier frequency. In a Manchester encoded bit time, the carrier should be present for either the first half or the second half of the bit time depending on whether the bit is a logic zero or a logic one.

The example below provides a recommended Freescale LF datagram format.

NOTE

Different automotive manufacturers have their own defined protocols, the example above is only provided as reference.

It is composed of:

- A 4 msec of CW preamble
- Two data transitions to allow a clean demodulation establishment
- Nine t_{DATA} synchronizations
- 16 bits of wake-up bytes (x5E and x31)
- Four bytes of data (x13, xC6, x6C, x3A)
- One end of message frame (illegal Manchester bit at a databyte boundary)

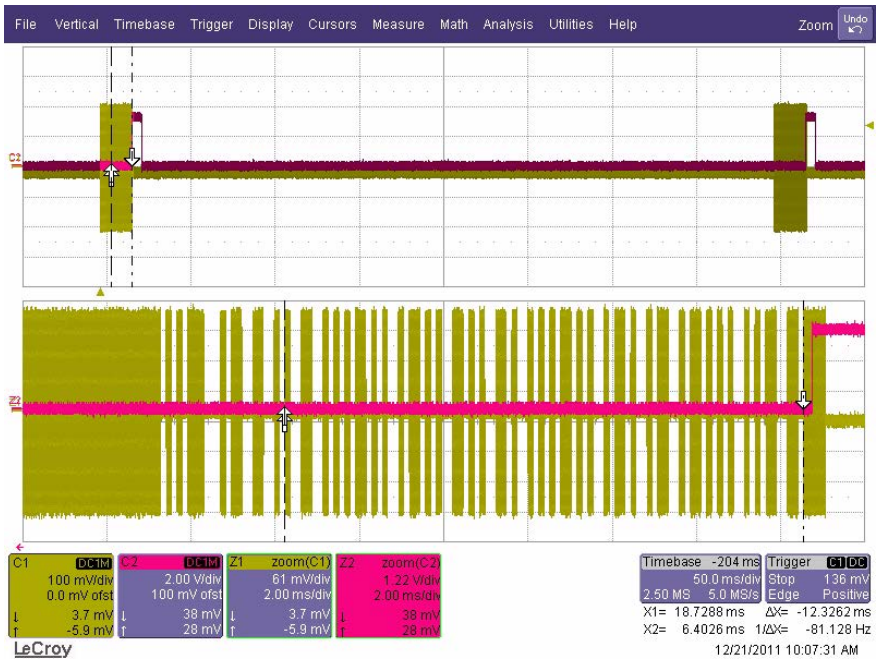

Figure 3-1. Example of Manchester LF telegram

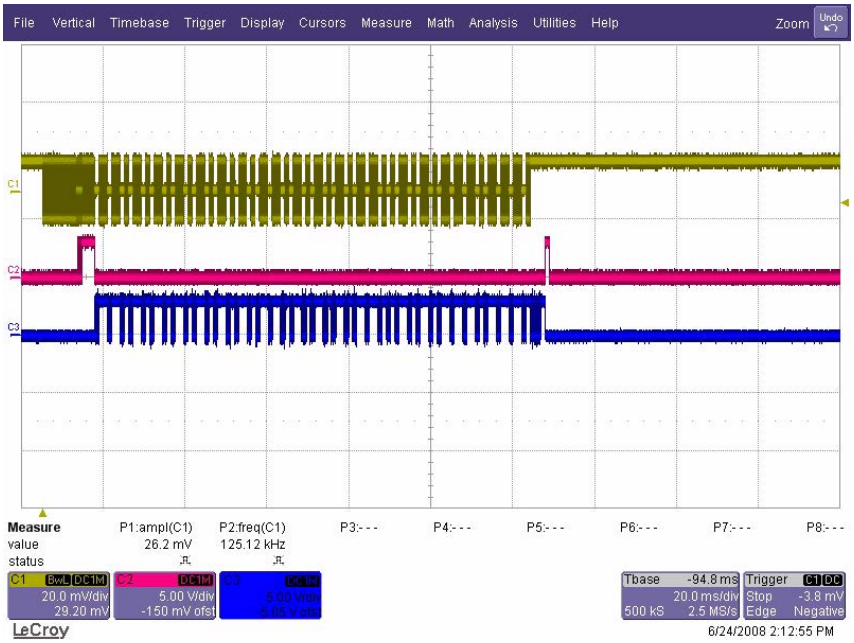

Figure 3-2. LF telegram and decoding check

3.1.3 Example of init and decoding function with data mode

Example 3-1. Init

```
/*****************************************************************************
function    :Init_LF(void)
parameters  :void
returns     :void
type        :low level c
description: LF Setup
*****************************************************************************/
void Init_LF(void)
{
  LFS=0x03; //Low consumption during sniff mode
  //Clears the LFDRF, LFERF, LFCDF, LFIDF, LFOVF and LFEOMF flag bits.
  LFCTL1=0x19; //  LPAGE=1
  LFCTRLD=0x03; //CARVAL validates on 1 times 4 edges inside a 32 usec window
  LFCTRLC=0xC9; //RESET Value with High Level Attenuator system enabled
  LFCTRLB=0xC4; //RESET Value with POL=0
  LFCTRLA=0x00; //RESET Value
  LFCTL1=0x8A; //  LPAGE=0 and LFON  - 16 bit ID - Sensitivity = Low
  LFCTRLA=0x00; //RESET Value
  LFIDL=0x31;  //Wake-up codes defined by user MSB
  LFIDH=0x5E; //Wake-up codes defined by user LSB
}
```

Example 3-2. Decoding

```
/*****************************************************************************
Function: Decode_LF_Datagram  - Manchester
NOTES: LF data decoding through Interrupt
 *****************************************************************************/
void Decode_LF_Datagram()
{
  unsigned char ii;
  LFDatagram[0]= LFDATA;
  for (ii=1;ii<4;ii++)
  {
    while(LFS_LFDRF==0); // Loop until LFDRF Flag is set
    {
      LFDatagram[ii]=LFDATA;
    }
  }

  LFDATA=0x00;
  /**************LF0 detection*******************/
  if (LFDatagram[0]==0x13 && LFDatagram[1]==0xC6 && LFDatagram[2]==0x6C &&
    LFDatagram[3]==0x3A)// && LFDatagram[4]==0x4F && LFDatagram[5]==0x62) //LF0
  {
    TogglePTA0();
  }
```

```
  else
  {
    /* Nothing */
  }

  LF_OFF();
  LFS_LFIACK=1;
  EnableInterrupts;
}
```

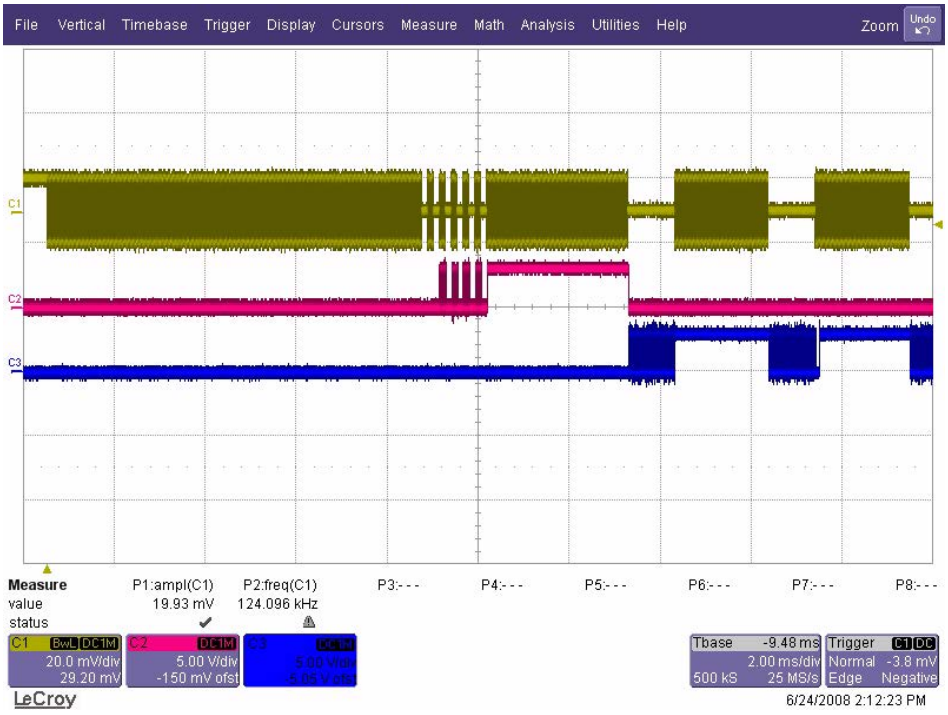

Figure 3-3. LF telegram (Synchro/5 Tbit/Preamble detail)

3.1.4 LF datagram recommended shape using the MCU-direct mode

When the LF is operating in MCU-direct mode, the MCU shall poll the LFDO bit to extract from the analog detector the bit stream. To assure an LFDO bit ready state, a transition in the XX register must be added between the preamble and the first data byte. This transition is used as the synchro. The CH2 is used to keep track of the preamble and the synchro detections and the test of a decoding. For more details on the MCU-direct mode, please refer to the FXTH87 family product specification.

The LF telegram to decode is composed by:

- Eight msec Preamble
- 600-µsec Synchro
- Five bytes of data's (xD5, x05, xAA, xCA, xEE)
- 0 is coded with a 1-msec width, 1 is coded with a 2-msec width

Figure 3-4. LF telegram used for MCU-direct mode example

3.1.5 Decoding function with MCU-direct mode

Example 3-3. Init

```
/*****************************************************************************
function    :   Init_TPM1CH0()
parameters  :   void
returns     :   void
type        : low level c
description:
This function initializes TIMER1 CHANNEL 0
*****************************************************************************/
void Init_TPM1CH0(void)
{
  TPM1SC=0x08 | 0x02;// 4F CLOCK TIMER = FBUS=8MHz/128 - set Timer ON
  TPM1C0SC=0x00; //Configure in TBM  , Interrupt Enable address
  TPM1MODH=0xFF; // FF Fixed the OC every 1.13sec
  TPM1MODL=0xFF; // FF Fixed the OC every 1.13sec
}
/*****************************************************************************
function    :Init_LF(void)
parameters  :void
returns     :void
type        :low level c
description: LF Setup
 *****************************************************************************/
void Init_LF(void)
{
  LFS_LFIACK=1;
  LFCTL1=0x10; // LPAGE=1
  delay(20);
  LFCTL1=0x00; // LPAGE=0
  delay(20);
  LFCTL3_TOGMOD=0;
  //Digital decoder is disabled.
  delay(20);
  LFCTL1_LFEN=1;
}
```

Example 3-4. Decoding

```
/*****************************************************************************
Function: Decode_LF_Datagram  -Manchester 01/04/2008  Optimized
NOTES: LF data decoding through Interrupt
 *****************************************************************************/
void Decode_LF_Datagram()
{
  unsigned int e,f;
  byte i,j,l;
  DisableInterrupts; /* Disable interrupts */
  /*** Init IO ***/
  PTAD=0x00;
  PTADD=0x03; /*PTA0,1 configured as an Output */
  e=0;
  Init_TPM1CH0();
  TPM1CNT=0; //clear timer contents to avoid overflow
  /*** Preambule detect and NAND condition  ***/

  PTAD=0x01;
  PTAD=0x00;
  //Time count to be in the area of NO CW before the 1 msec Synchro pulse
  while (TPM1CNT<8000);
  // Stays in the NO CW area before the Synchro and wait for the LFDO set to 1
  while (LFCTL3_LFDO==0);
  PTAD=0x01;
  PTAD=0x00;
  /**  Synchro  detection  **/

  TPM1CNT=0;  //clear timer contents to avoid timing cumulation
  while (!((LFCTL3_LFDO!=1) && (TPM1CNT>500)));  //wait LFDO stays to One
  e= TPM1CNT + e;    //Store Timer Value
  PTAD=0x01;
  PTAD=0x00;
  TPM1CNT=0;  //clear timer contents to avoid timing cumulation
  while (!((LFCTL3_LFDO!=0) && (TPM1CNT>500)));  //wait LFDO stays to
  Zero);
  e= TPM1CNT + e;    //Store Timer Value
  PTAD=0x01;
  PTAD=0x00;
  TPM1CNT=0;  //clear timer contents to avoid overflow
  /******** DECODING LOOP *******/
  j=0;
  for (l=0;l<=4;l++)
  {
    do
    {
      TPM1CNT=0;  //clear timer contents to avoid overflow
      e=TPM1CNT;
      PTAD=0x02;   //PTA1 ON
      while (!((LFCTL3_LFDO!=1) && (TPM1CNT>850)));  //wait LFDO stays to One
      f=TPM1CNT;
      e= TPM1CNT + e;    //Store Timer Value
```

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

      while (!((LFCTL3_LFDO!=0) && (TPM1CNT>850)));  //wait
      LFDO stays to Zero);
      e= TPM1CNT + e;    //Store Timer Value

      if ( f>1800 && f<2400)

      {
        LFDatagram[l] |= (1 << (7-j));   //     CREATE A ONE
      }
      if (f>650 && f<1250)
      {
        LFDatagram[l] &= ~(1 << (7-j));    //   CREATE A ZERO
      }

      j++;

    } while(j<8);
    j=0;
  }

  if (LFDatagram[0]==213 &&  LFDatagram[1]==85 && LFDatagram[2]==170 &&
      LFDatagram[3]==202 && LFDatagram[4]==238)
  {
    TogglePTA0();
  }

  LF_OFF();
  LFCTL4_LFCDIE=1;
  LFS_LFIACK=1;    // Clear   LF Flags
  asm{cli;}
}
```

3.2 RF transmitter

3.2.1 RF block general information

The RF embedded module (RFM) in the FXTH87 consists of an RF output driver for an antenna and a hardware data buffer for automated output or direct control from the MCU.

It has its own memory map containing 32 register locations. These registers contain control and status bits for the RFX modules, data location for the RF data buffer, trim variables and test registers. Please refer to the FXTH871 family product specification. The firmware subroutines control this interface so that the user only needs to fill the RFBUFFER and use the TPMS_RF_SET_TX with the appropriate parameter value.

3.2.2 RF output impedance

For an optimized matching network, the impedance measured as close as possible to the RF pin should be between 400 and 500 Ω, and the imaginary part should be slightly inductive to be compensated by a capacitive part due to pad, package and bonding (PA architecture gives the best performance at the resonance, meaning with imaginary part cancelled).

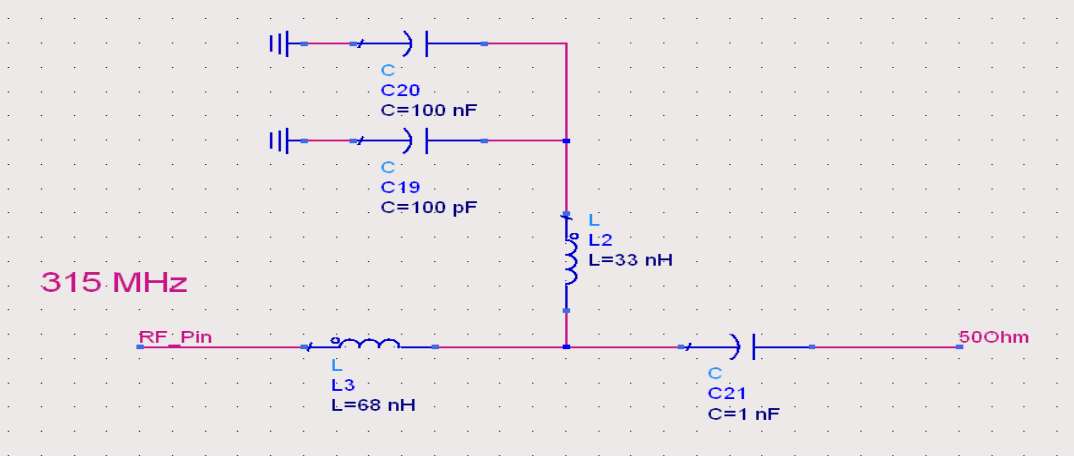

Figure 3-5. Recommended matching network to 50 Ω at 315 MHz

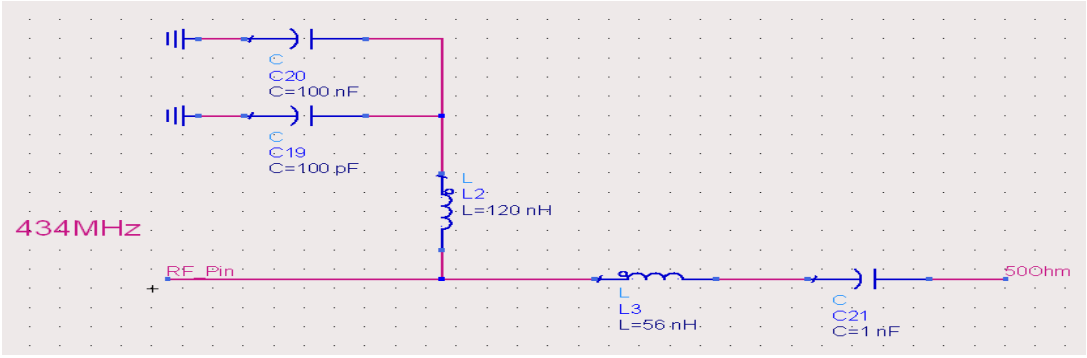

Figure 3-6. Recommended matching network to 50 Ω at 434 MHz

For both available transmission frequencies, the values for L1 and L2 correspond to the recommended values shown in Figure 3-6. 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.

3.2.3 RF MCU-direct code 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). Toggle the contents of the DATA bit in the RFCR3 register to clock data out the RF amplifier in real-time. In this mode, the data rate and its stability, will depend on the internal 8-MHz oscillator.

The maximum data rate in this mode will depend on the complexity of the user software.

The software screen capture in Figure 3-7 and in Example 3-5, sends a table of 15 bytes at 9600 bauds.

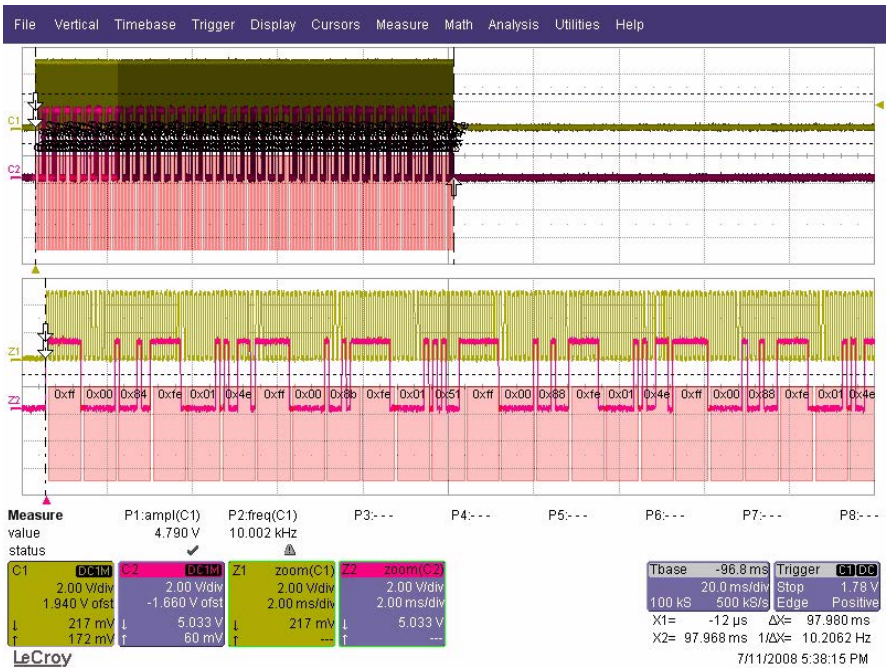

Figure 3-7. RF telegram received and decoded

Example 3-5. Software example

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

  RFCR0=0x34;    // 9600 bauds - Reset value
  RFCR1=0x78;    // set to 80 for 128 bits - largest frame
  RFCR2=0x0E;    // RF Transmission OFF - No EOM - Pout=5dBm    - RPAGE=0
  // 00001110

  RFCR2=0x8E;    // RF Transmission ON - No EOM - Pout=5dBm
  // 10001110
  RFCR3=0x00;    // RF Output Low - RF Power UP - One Frame Transmitted
  // 00000000
  RFCR4=0x01;     //Interframe timing set to 1
  RFCR5=0x00;     //No Pseudo-random number used
  RFCR6=0x01;     //VCO highest Power - interframe timing
  //
  RFCR7=0x00;     // RF Interrupt Disable - LVD Disable - RFM Not reset
  /******* Value for MC33696 receiver  434 MHz  +/-50KHz  *********/

  PLLCR0=0xB0;
  PLLCR1=0x63;
  PLLCR2=0xB1;
  PLLCR3=0x56;
  EnableInterrupts;
}
/*************************************************************************
function    :   initTPM1CH0()
parameters  :   void
returns     :   void
type        : low level c
description:
This function initializes TIMER1 CHANNEL 0  for TBM purpose
 *************************************************************************/
void InitTPM1CH0(void)
{
  TPM1SC=0x08 | 0x02;// 4F CLOCK TIMER = FBUS=8MHz/128 - set Timer ON
  TPM1C0SC=0x00; //Configure in TBM  , Interrupt Enable address
  TPM1MODH=0xFF;  // FF Fixed the OC every 1.13sec
  TPM1MODL=0xFF;// FF Fixed the OC every 1.13sec
}
/*************************************************************************
Function: Send_RF_Datagram
```

```
NOTES:   This function fills the buffers and codes the RF frame in Manchester
 ***************************************************************************/
void Send_RF_Datagram()
{
  unsigned char i;
  DisableInterrupts;
  /* Specific data's for MC33696 */
  RFD0=0xFF;
  RFD1=0xAA;   // AA 10101010 needed to understain that the rest are DATA's
  RFD2=0x9B;   // 9B   10 011011
  /* End of Specific data's for MC33696 */
  RFD3=0xA1;
  RFD4=0xB2;
  RFD5=0xC3;
  RFD6=0xD4;
  RFD7=0x06;
  RFD8=0xA1;
  RFD9=0xA1;
  RFD10=0xA1;
  RFD11=0xC3;
  RFD12=0xAA;
  RFD13= 0x90;
  RFD14=0xBB;
  InitTPM1CH0();
  RFCR7_RFIACK=1;  // Clear all Flags
  delay(200);

  Code_Manchester_Frame(&RFD0);
  Code_Manchester_Frame(&RFD1);
  Code_Manchester_Frame(&RFD2);
  Code_Manchester_Frame(&RFD3);
  Code_Manchester_Frame(&RFD4);
  Code_Manchester_Frame(&RFD5);
  Code_Manchester_Frame(&RFD6);
  Code_Manchester_Frame(&RFD7);
  Code_Manchester_Frame(&RFD8);
  Code_Manchester_Frame(&RFD9);
  Code_Manchester_Frame(&RFD10);
  Code_Manchester_Frame(&RFD11);
  Code_Manchester_Frame(&RFD12);
  Code_Manchester_Frame(&RFD13);
  Code_Manchester_Frame(&RFD14);

  EnableInterrupts;
}
/*****************************************************************************
function    :Code_Manchester_Frame()
parameters  :void
returns     :void
type        :low level c
description :This function code the RF datagram in Manchester for 9600 Bauds
 *****************************************************************************/
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
    {
      RFCR3_DATA=1;
      TPM1CNT=0;
      while (TPM1CNT<37);
      RFCR3_DATA=0;
      TPM1CNT=0;
      while (TPM1CNT<41);
    }
    else                  // If bit is 0 = 01 ---> 1
    {
      RFCR3_DATA=0;
      TPM1CNT=0;
      while (TPM1CNT<41);
      RFCR3_DATA=1;
      TPM1CNT=0;
      while (TPM1CNT<37);
    }
  }
}
```

3.3 X- and Z-axis accelerometers

3.3.1 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 ±1g signal caused by the Earth's gravitational field.

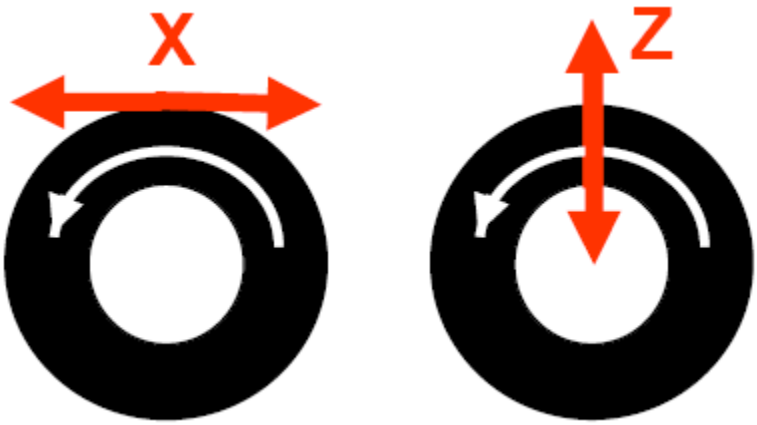

Figure 3-8. X- and Z-axis sensing direction

The two graphs in Figures 3-9 and 3-10 show the response of the X- and Z-axis sensor when the tire is in motion and the sensor's response to continuous acceleration. The signal frequency for both axes increases with the speed. Due to the sensitivity of the sensor, some very simple algorithms allow to differentiate the parking from the motion state.

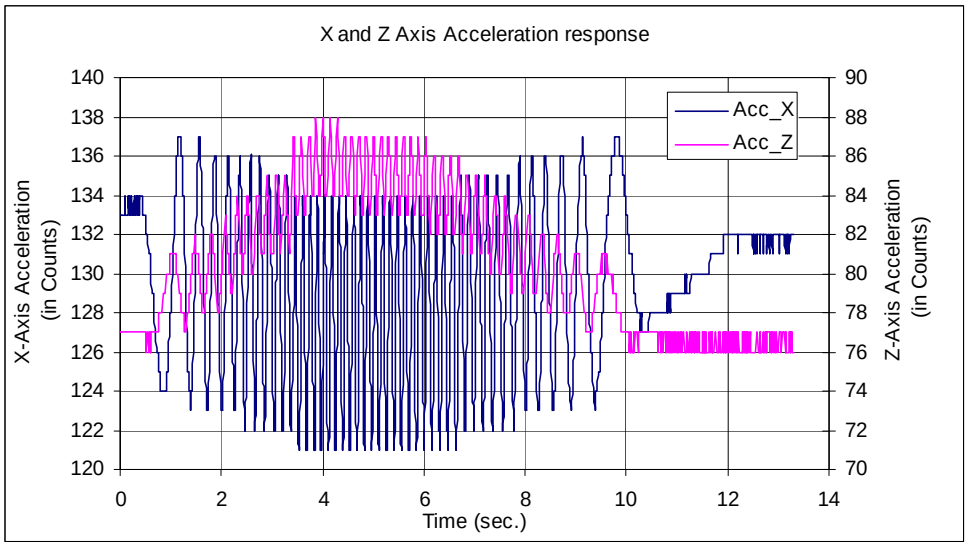

Figure 3-9. X- and Z-axis response in a change of speed

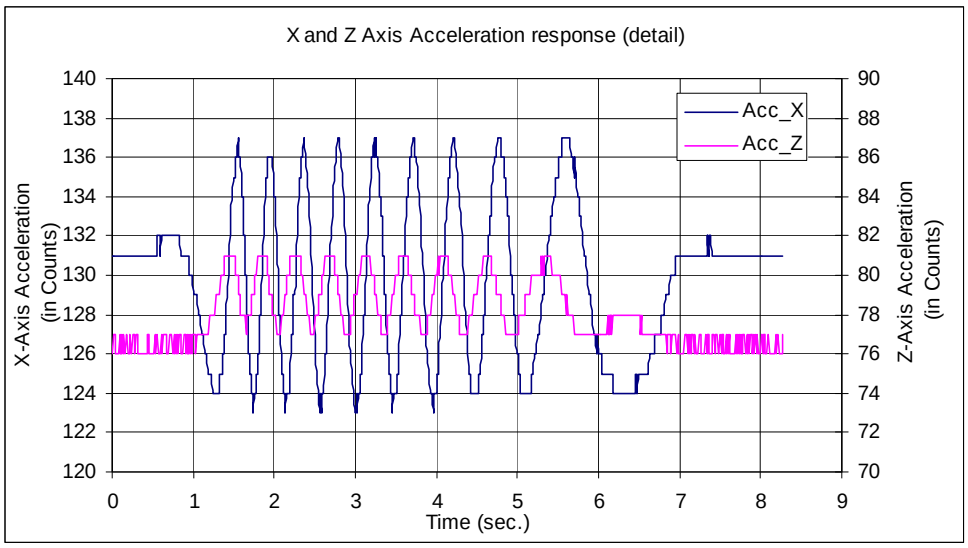

Figure 3-10. X- and Z-axis response with speed in the range of 1 to 2 ms^{-1}

Chapter 4 Firmware Function Example

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

Example 4-1.

```
/*****************************************************
 Variable declaration used in the examples
 *****************************************************/
UINT16 gu16UUMA[5u];
T_RDE  tRDEData;
UINT8  TPMS_INTERRUPT_FLAG @ 0x008Fu;
unsigned char PPRESSURE,PTEMPERATURE,PVOLTAGE,PACCX,PACCZ;
unsigned int  u16CompPressure,u16CompAccelX,u16CompAccelZ;
unsigned char  u8Status,u8CompTemperature,u8CompVoltage,u8Offset_X,u8Offset_Z;
unsigned char Acc_X_Results[20];
unsigned char Acc_Z_Results[20];
unsigned char
u8WDIV_TEST,u8Error,u8Avg,u8PNew,u8NewChecksum,u8NewCRC,u8Status,u8TestMask,u8DYN_X,
u8DYN_Z;
unsigned int
u16ResultV0,u16ResultV1,u16POld,u8NewAverage,u16NewCRC,u16MByteSize,u16Status,u16Key
Temp,u16Key,pu16Measurement;
unsigned int u16CompAccel[2];
unsigned int PU_ACCX,PU_ACCZ,PTA0_10Bit_Value,PTA1_10Bit_Value;
unsigned char u8PowerManagement[5]={0x00,0x01,0x02,0x03,0x04};
unsigned int u16RFParam[4]={0xAC68,0xAB48,0x0081,0x0018};
unsigned char RFRD_data_Manchester[6];
unsigned char RF_Red_Register[13];
unsigned char RFRD_SETUP_data[4];
unsigned char RFRD_SETUP_data_ReadBack[4];
unsigned char RFRD_data[16];
```

Table 4-1. Firmware function types and uses

Firmware Function type	Example of use
Master Reset	TPMS_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.
Initial measurements for compensation	TPMS_READ_VOLTAGE(gu16UUMA); TPMS_READ_TEMPERATURE(gu16UUMA);
Pressure acquisition	TPMS_READ_PRESSURE(gu16UUMA, 4u); TPMS_COMP_PRESSURE(&u16CompPressure, gu16UUMA); PPRESSURE = u16CompPressure>>1;
Temperature acquisition	TPMS_COMP_TEMPERATURE(&u8CompTemperature, gu16UUMA); PTEMPERATURE = u8CompTemperature;
Voltage acquisition	TPMS_COMP_VOLTAGE(&u8CompVoltage, gu16UUMA); PVOLTAGE=u8CompVoltage;
Acc_X data acquisition	TPMS_READ_ACCEL_X(gu16UUMA,2,0,7); TPMS_COMP_ACCEL_X (&u16CompAccelX,gu16UUMA); PACCX = u16CompAccelX>>1;
Acc_Z data acquisition	TPMS_READ_ACCEL_Z(gu16UUMA,2,0,7); TPMS_COMP_ACCEL_Z (&u16CompAccelZ,gu16UUMA); PACCZ = u16CompAccelZ>>1;
Dynamic Acc_X data acquisition	u8Offset_X=7; u8Status=TPMS_READ_DYNAMIC_ACCEL_X(1,&u8Offset_X,&gu16UUMA);
Dynamic Acc_Z data acquisition	u8Status=TPMS_READ_DYNAMIC_ACCEL_Z(1,&u8Offset_Z,&gu16UUMA);

Acc_X&Z data acquisition	TPMS_READ_ACCEL_XZ(gu16UUMA,2,0,7,7); //AVG = 2, 500 Hz Filter, 0g offset for X&Z
	TPMS_COMP_ACCEL_XZ (&u16CompAccel,gu16UUMA);
	PACCX = u16CompAccel[0]>>1;
	PACCZ = u16CompAccel[1]>>1;

Dynamic Acc_X&Z data acquisition	u8DYN_X=7;
	u8DYN_Z=7;
	TPMS_READ_DYNAMIC_ACCEL_XZ(0,&u8DYN_X,&u8DYN_Z,&gu16UUMA);
	PU_ACCX = gu16UUMA[3];
	PU_ACCZ = gu16UUMA[4];

ADC Acquisition	u8Status=TPMS_READ_V0(&u16ResultV0,4);
	PTA0_10Bit_Value=u16ResultV0;
	u8Status=TPMS_READ_V1(&u16ResultV1,8);
	PTA1_10Bit_Value=u16ResultV1;

LFO Calibration	u8WDIV_TEST=TPMS_LFOCAL(); PWUDIV = u8WDIV_TEST; // Every seconds with calibrated value

Average	u16POld=514;
	u8PNew=255;
	u8Avg=2;
	u8NewAverage=TPMS_WAVG(u8Avg,u16POld,u8PNew);

Table 4-1. Firmware function types and uses (continued)

RF Data Read	TPMS_RF_READ_DATA(5,&RF_Red_Register,0); //read 5 data from RF data Register (BUFF0 correspond to address 0) (RF die Buffer) and //write those 5 data's LSB first to RF_Red_Register table
RF Data Reverse Read	TPMS_RF_READ_DATA_REVERSE(5,&RF_Red_Register,0); //read 5 data from RF data Register (BUFF6 correspond to address 5) (RF die Buffer) and //write those 5 data's MSB first to the RFX buffer at BUFF0
RF Data write	TPMS_RF_WRITE_DATA_REVERSE(5,&RFRD_data,5); //read 5 data from RFRD_data table and //write those 5 data's MSB first to the RFX buffer at BUFF6 //(BUFF6 correspond to address 5)
RF Configuration	TPMS_RF_CONFIG_DATA(&u16RFParam);
RF Enable	TPMS_RF_Enable(1); // Switch RF ON
RF Reset	TPMS_RF_RESET();
Checksum	u8NewChecksum=TPMS_CHECKSUM_XOR(&RFRD_SETUP_data_ReadBack,4,0);
CRC8 Calculation	u8NewCRC=TPMS_CRC8(&RFRD_data,16,0);
CRC16 Calculation	u16NewCRC=TPMS_CRC16(RFRD_data,16,0)
Square root	a=69; b=TPMS_SQUARE_ROOT(a); // result multiplied by 10
Tire ID reading	TPMS_READ_ID(&CODE,0); //0 Index provides all related information
Switch LF ON	TPMS_LF_ENABLE(1);
Wire Checking	u8Status=TPMS_WIRE_AND_ADC_CHECK(0x9C);
Flash Write	TPMS_FLASH_WRITE(0xc4C0,CODE,6);
Flash Check	u16Status=TPMS_FLASH_CHECK(0);
Flash Erase	TPMS_FLASH_ERASE(0xD234);

Flash Protection	u16KeyTemp=CODE[4]<<8;
	u16Key=u16KeyTemp + CODE[5];
	u8Status=TPMS_FLASH_PROTECTION(u16Key);

Multiplication	TPMS_MULT_SIGN_INT16(0x1122,0xAABB,&pi32Result);
RDE Adjust	u8Status=TPMS_RDE_ADJUST_PRESSURE(&gu16UUMA,&Test_Array);
Initialization of the serial communication	TPMS_MSG_INIT (void)
Reading data from emulated serial interface	TPMS_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 DATA (PTA1 and PTA0). As the master, the module must manage the clock on PTA1. On falling edge of the clock, the module reads a new data bit on PTA0 (programmed as input), MSB first.
Writing data on emulated serial interface	TPMS_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 (PTA1 and PTA0). As the master, the module must manage the clock on PTA1. On rising edge of the clock, the module puts down a new data bit on PTA0 (programmed as output), MSB first.

NOTE

Software application example can be found on the CD provided with the TPMS kit.

Table 4-2. Universal uncompensated measurement array (UUMA)

Index	Content
0	Uncompensated voltage
1	Uncompensated temperature
2	Uncompensated Pressure
3	Uncompensated X-axis acceleration
4	Uncompensated Y-axis acceleration

This array is referred to as the universal uncompensated measurement array (UUMA) and can be located anywhere the customer decides.

Each element must be 16-bits long (two bytes) regardless of what the actual bit-width of the measurement is.

Each individual uncompensated measurement routing will only update its corresponding item. For example calling the TPMS_READ_VOLTAGE routine will only modify the voltage element of the array. The rest will remain unchanged.

Compensation routines do not modify any elements in the UUMA.

Revision History

Revision	Date	Description
0	07/2014	• Initial release of reference manual.
1.0	08/2015	• Updated format. • Added device number FXTH871511. • Chapter 2: Updated Figure 2-1 and Figure 2-2. Updated Table 2-1. • Removed Chapter 5.