Freescale Semiconductor, Inc.
Application Note

Document Number: AN5136
Rev. 0, 07/2015

Using the Kinetis KW01 as the Receiver for the FXTH870xD TPMS Sensor

A basic TPMS receiver demonstration

Contents

1. Introduction

The KW01 device is a highly-integrated, cost-effective, smart radio, sub-1 GHz wireless node solution composed of a transceiver supporting FSK, GFSK, MSK, or OOK modulation with a low-power ARM® Cortex® M0+CPU. The highly integrated RF transceiver operates over a wide frequency range including 315 MHz, 433 MHz, 470 MHz, 868 MHz, 915 MHz, 928 MHz, and 955 MHz in the license-free Industrial, Scientific and Medical (ISM) frequency bands.

The FXTH870xD is a sensor for use in applications that monitor tire pressure and temperature. It contains the pressure and temperature sensors, an X-axis and a Z-axis accelerometer, a microcontroller, an LF receiver and an RF transmitter all within a single package.

This application note will first detail KW01 and FXTH870xD register settings, followed by guidance to modify the demonstration software for both parts and to set up a TPMS system using the KW01 connected to a PC as a receiver with a stand-alone FXTH870xD transmitting.

CodeWarrior 10.x and IAR 7.4 will be used in programming.

2. TPMS overview

Tire Pressure Monitoring System (TPMS) is the standard for improved vehicle safety. The TPMS system is widely used for all types of vehicles from small passenger cars to trucks, Recreational Vehicles, and buses. It provides accurate tire pressure data to a console or warning system visible to the driver. This allows the driver to keep the tires at optimum pressure for best fuel mileage and tire life and to reduce the risk of blow-out, fire caused by tire-overheating or an accident caused by low pressure.

Typically the full system will consist of a sensor in each tire and a receiver that feeds the console or instrument panel.

The FXTH870xD serves as the sensor and consists of a pressure sensor and RF transmitter as well as other features and a processor tying it all together. In a typical passenger car system, there would be four such sensors, one in each tire, each transmitting a unique identifier in addition to data. This is often implemented as a tiny module on the valve stem that mounts inside the tire, a factory equipped TPMS system would be so implemented. Alternately an add-on or retrofit system might have a sensor that mounts on the outside of the valve stem.

The KW01 can serve as the RF receiver, with a Kinetis ARM core MCU it has the power to interpret received data and feed message information to the processors powering the vehicle's instrument cluster in a factory equipped TPMS system or to drive the display and alarms of an after-market or retro-fit TPMS receiver.

3. Kinetis KW01 receiver

3.1. Kinetis MRB-KW01 modular reference board

The MRB-KW01 is a modular reference board for the MKW01Z128CHN platform, part of the Kinetis W series of MCUs.

Powered by the ultra-low-power 48 MHz ARM Cortex-M0+ 32-bit core, the Kinetis KW0x family of MCUs embeds a rich set of peripherals such as a high-performance, bi-directional sub-1 GHz radio capable of operating over a wide frequency range of 315, 433, 470, 868, 915, 928 and 960 MHz in the license-free industrial, scientific and medical (ISM) frequency bands.

There are three development boards available for the Kinetis KW0x platform, based on regional needs:

- MRB-KW019032NA - 915 MHz with 32 MHz XTAL for North American Applications
- MRB-KW019032EU - 868 MHz with 32 MHz XTAL for European Applications
- MRB-KW019030JA - 900 MHz with 30 MHz XTAL for Japan Applications

This application note adapts the Kinetis MRB-KW019032EU modular reference board (see Figure 1, below) as a TPMS Receiver. The NA, EU, or JA denote the regional software loaded on the MRB, so either the NA or EU version of the MRB-KW019032 can be used.

The Kinetis MRB-KW01 configured as TPMS receiver needs to operate at one of two different frequencies: 315 MHz or 434 MHz. Parameters for setting the packet configuration for use at 315 MHz and 434 MHz are described in the next section, 3.2.

Figure 1. MRB-KW01 Modular Reference Board

3.2. Kinetis KW01 configuration for packet reception at 315 MHz or 434 MHz

It is important that some radio parameters be set to the proper configuration to correspond to those of the TPMS sensor. Those parameters are modulation type, frequency, deviation, bit rate and encoded data and packet format.

An illustration of a variable length packet is shown in figure 2 below. It contains the following fields:

- Preamble (1010...)
- Sync word (programmed into Data registers of the TPMS sensor)
- Length Byte (0x0A, 10-bytes, in this example)
- Optional Address byte (0xFF is transmitted in this demo application.)
- Message data (this is where the TPMS sensor data is transmitted)
- Optional 2-bytes CRC checksum

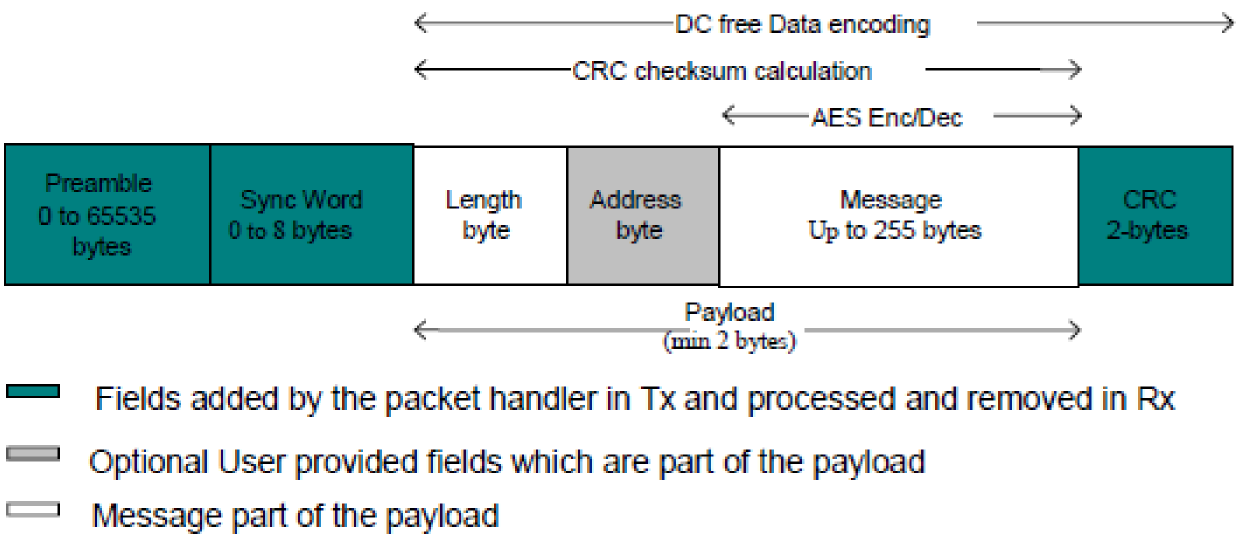

Figure 2. KW01 Variable Length Packet Format.

Elsewhere in the code (not shown in the example below) register *PacketConfig1* is configured to set the KW01 to Variable Length Packet Format by programming a 1 to bit 7.

While the basic KW01 can operate in the 300 MHz, 450 MHz, and 900 MHz bands, the external RF matching and filtering components should be optimized to the desired band. The MRB-KW019032 has those components optimized for the 900 MHz band. With 900 MHz components there will be some mismatch loss and hence some transmit power degradation (as well as some receiver sensitivity degradation) when operated at 315 MHz or 434 MHz. The most unacceptable result of using 900 MHz components for a 315 MHz or 434 MHz application is that there is no harmonic rejection in transmit, resulting in failing certification testing and transmitting signal outside of authorized bands. This TPMS receiver application does not use the transmit function of KW01, so that is not an issue. The harmonic rejection filter does protect the 315 MHz or 434 MHz receiver front end overload in the presence of a large 900 MHz signal.

The MRB-KW019032 can be used for 315 MHz and 434 MHz testing. It will be entirely satisfactory for software development and receive sensitivity degradation is minimal.

When developing a product for 315 MHz or 434 MHz it is best to change the 900 MHz components to those optimized for either 315 MHz or 434 MHz as needed. As only the RFIO path is used, and that only in receive, the PA_Boost path components need not be changed on an MRB and can be eliminated in a new layout. Ten components are affected, they are listed in table 1 below. The reference designators refer to those on the MRB, see the *MRB-KW01-9032 Schematic*. Note that with any new layout or change in board material it is best to optimize the components for best performance.

Table 1. Band-Specific MRB Components

MRB Ref Des	L2	L4	L7	C4	C9	C11	C13	C15
9032 board (868 or 915 MHz)	6.8nH	6.8nH	4.7nH	5.6pF	7.5pF	dnp	6.8pF	2pF
4532 board (434 MHz)	10nH	12nH	10nH	8.2pF	15pF	2.7p	22p	22p
3032 board (315 MHz)	18nH	22nH	4.7nH	12pF	15pF	dnp	33pF	15pF

3.2.1 Carrier frequency, frequency deviation, bit rate, preamble and sync value

Carrier frequency, frequency deviation, bit rate, preamble, and sync word of KW01 can be properly set up in the software file ApplicationConf.h in the SimpleRangeDemo project file. That file can be modified with various settings. See the MKW01 Reference Manual (MKW01XXRM) for register details. There are some limitations in the minimums and maximums to which these parameters can be set but those do not affect this project.

3.2.1.1 Carrier frequency

The transceiver PLL embeds a 19-bit sigma-delta modulator and its frequency resolution, constant over the whole frequency range, and is given by:

$$F_{STEP} = \frac{F_{XOSC}}{2^{19}}$$

Eqn. 1

The carrier frequency is programmed through *RegFrf*, split across registers *RegFrfMsb*, *RegFrfMid* and *RegFrfLsb* (addresses 0x07 to 0x09):

Eqn. 2

$$F_{RF} = F_{STEP} \times Frf(23,0)$$

3.2.1.2 Frequency deviation

FSK modulation is performed inside the PLL bandwidth, by changing the fractional divider ratio in the feedback loop of PLL. The large resolution of the sigma-delta modulator, allows for very narrow frequency deviation. The frequency deviation Fdev is given by:

Eqn. 3

$$F_{DEV} = F_{STEP} \times Fdev(13,0)$$

3.2.1.3 Bit rate

The bit rate is controlled by bits *BitRate* in *RegBitrate*. The bit rate is given by:

$$BR = \frac{F_{XOSC}}{BitRate}$$

Eqn. 4

3.2.1.4 Preamble and sync value

The Preamble and sync word need to be configured the same in both the KW01 receiver and the TPMS transmitter. The preamble length is programmed in the *RegPreambleMsb* and *RegPreambleLsb* registers (0x2C and 0x2D) and can have a value of 0x55 or 0xAA (1010… sequence). It is recommended that a minimum of 12 bits of preamble precede the rest of the packet. The up to 8 bytes of sync word are programmed in the RegSyncValue1-8 registers (0x2F to 0x36). The RegSyncConfig register (0x2E) configures the sync word recognition circuit by allowing it to be enabled or disabled and setting the length of the sync word, and its tolerance to bit errors.

3.3. Communication between TPMS FXTH87 emitter and KW01 receiver

This section shows the modifications that have been done in the emitter and receiver projects as well as information about the demo setup and the results of over the air tests of KW01 + FXTH87.

3.3.1. Code examples from the KW01 SimpleRangeDemo firmware

On the receiver side the original project is *SimpleRangeDemo;* a choice for the RX carrier frequency has been added as well as files allowing the user to watch the frames either on the hyperterminal or on the Freescale LabVIEW Sensor GUI (available on the [Freescale FXTH87 web page]). The new project is *KW01_IAR7v4_Project_TPMS_Rev0.6* .

In the SimpleRangeDemo file the main.h settings can be selected for RF band, data output system and KW01 node and broadcast addresses:

Example 1. Simple Range Demonstration Configuration

```
/***************************************************************************
****************************************************************************
*                          DEMO CONFIGURATION
*  RF carrier frequency: select below the RF carrier frequency.
*  Data output: Select below to watch the frames with the Frescale LabVIEW Sensor
*               GUI or the hyperterminal. This choice can be done here or in the
*               FXTH87 emitter project.
*  Addresses: Choose the values of the node and broadcast addresses.
*             Frames with an address byte different from one of these two values
*             will be discarded on the MKW01 side
*
****************************************************************************
***************************************************************************/
```

```
/* Uncomment one of the defines */
#define CARRIER_FREQ 315                                  /* for a 315Mhz carrier
frequency */
//#define CARRIER_FREQ 434                                /* for a 434MHz carrier
frequency */
//#define CARRIER_FREQ 915                                /* for a 915MHz carrier
frequency */

/* Uncomment one of the defines */
#define FRAME_DISPLAY HYPERTERMINAL                                    /* To use
the hyperterminal */
//#define FRAME_DISPLAY FREESCALE_LABVIEW_SENSOR_GUI    /* To use the Freescale
LabVIEW Sensor GUI */
//#define FRAME_DISPLAY NO_SELECTION_HERE                     /* The selection is
done in the FXTH87 emitter project */

/* Choose the value of the MKW01 node and broadcast addresses */
#define NODE_ADDRESS 0xF0
#define BROADCAST_ADDRESS 0xFF
```

Within the file ApplicationConf.h, the carrier frequency band selected above is converted to the *RegFrf* words for the appropriate frequency for the given geographical region (NA or EU):

```
#ifdef MKW01_NA
#if (CARRIER_FREQ == 315)
#define gDefaultRfFreq_c            ( 0x4EC000 )
#elif (CARRIER_FREQ == 434)
#define gDefaultRfFreq_c            ( 0x6C7FFB )
#elif (CARRIER_FREQ == 915)
#define gDefaultRfFreq_c            ( 0xE4C000 )                            //rf
freq 915 MHz (US).
#endif // CARRIER_FREQ
#endif
#ifdef MKW01_EU
#if (CARRIER_FREQ == 315)
#define gDefaultRfFreq_c            ( 0x4EC000 )
#elif (CARRIER_FREQ == 434)
#define gDefaultRfFreq_c            ( 0x6C7FFB )
#else
```

```
#define gDefaultRfFreq_c            ( 0xD8A199 )                       //rf
freq 866.525 MHz (EU).
#endif // CARRIER_FREQ
#endif
```

Further down within the file ApplicationConf.h, the bitrate can be set by commenting out or removing those options that are not chosen. The 19230 bps option is the code line shown:

```
//#define gDefaultBitRate_c    ( 0x0D05 )                              //9600
#define gDefaultBitRate_c      ( 0x0680 )                              //19230
```

Further down within the file ApplicationConf.h additional settings can be set (unrelated lines and commented-out alternatives have been removed for brevity):

```
#define gDefaultFreqDv_c            ( 0x0333 )
#define gDefaultSequencer_c         ( OpMode_Sequencer_On )
#define gDefaultDataMode_c          ( DataModul_DataMode_Packet )
#define gDefaultModulationType_c    ( DataModul_Modulation_Fsk )
…
#define gDefaultDccFreq_c           ( DccFreq_7 )
#define gDefaultRxFilterBw_c  ( RxBw_83300 )
#define gDefaultDccFreqAfc_c  ( DccFreq_5 )
#define gDefaultRxFilterBwAfc_c     ( RxBw_100000 )
…
#define gDefaultSyncValue_c             ( 0x01 )
…
#define gDefaultPreambleLength_c    ( 3 )
…
#define gDefaultSyncWordSize_c      ( SyncConfig_SyncSize_4 )
#define gPrbs9BufferLength_c  ( 65 )
#define gReset_c              ( 'R' )
/* Default Radio Registers Values_End */
```

The frequency word in the example above are for 315.000 MHz, 434.000 MHz and so on, and will have to be calculated and programmed for other frequencies, such as 433.920 MHz in the 434 MHz band. If a factory frequency trimming technique is utilized, the LSBs of the final value may be different than shown.

In the examples above, a default 0x01 is set for the sync word. This value is used in other Freescale demonstration software.

Elsewhere in the code (in main.c) both sync word bytes are set to that default value using:

- PhyPib_SetSyncWordValue(MKW01_Reg_SyncValue1, gDefaultSyncValue_c);
- PhyPib_SetSyncWordValue(MKW01_Reg_SyncValue2, gDefaultSyncValue_c);

In an actual application it will likely be desired to set the Sync word values to a word used in the system. This can be accomplished by changing or removing the unwanted commands and using commands such as:

```
#define    MKW01_Reg_SyncValue1       0xYY
#define    MKW01_Reg_SyncValue2       0xYY
```

Or

```
MKW01Drv_WriteRegister(MKW01_Reg_SyncValue1, 0xYY );
MKW01Drv_WriteRegister(MKW01_Reg_SyncValue2, 0xYY );
```

(where Y is a placeholder for the hex actual value, in the relevant regions of the code).

4. FXTH870xD transmitter

This application note details adapting the FXTH87 TPMS sensor as the TPMS Transmitter. The FXTH87 sensor is available in a in a 7 x 7 mm QFN package.

The FXTH870xD is the recommended TPMS transmitter solution for new designs. Please note that the older MPXx86xxD is not recommended for new designs, but is included in this applications note to assist in the support of legacy products. The firmware and application examples below refer to the FXTH870xD series, the MPXx86xx series is similar.

On the emitter side, the original project that has been modified is *TPMS_Periodic_RF_Transmission*. The format of the RF frame has been changed, RF settings have been modified and a choice for the tire ID has been added with optional frame verification (MKW01 CRC, FXTH87 CRC or checksum) and a choice for the frame display. The name of the new project is TPMS_FXTH87_MKW01_rev4.

Figure 3 shows the FXTH87 on its evaluation board. The TPMS sensor can be configured to operate at one of two different frequencies: 315 MHz or 434 MHz. Parameters for setting up the TPMS sensors are described in sections 4.1 and 4.2.

Figure 3. FXTH870xD TPMS Sensor board

4.1. TPMS sensor configuration for 315 MHz or 434 MHz

It is important that some radio parameters be set to the proper configuration to correspond to those of the Receiver. Those parameters are modulation type, frequency, deviation, bit rate and encoded data format.

4.1.1. Frequency, deviation and encoded data format

Frequency of a 0 or 1 (which defines deviation) and Encoded data format is programmed using the PLLCR0 ~ PLLCR3 registers in the main.c file for the FXTH870xD series. A snippet of that code is shown below in example 2. See FXTH870xD Tire Pressure Monitor Sensor Data Sheet (FXTH870XD) and Xtrinsic MPXx85/86xxD Tire Pressure Monitor Sensor Data Sheet (MPXX8XXXD).

```
315 MHz:

/***** Final PLL values for 315 MHz  - FSK=+/-50 kHz – Non_Manchester Coded POL 0
- No Dx Signal****/

   PLLCR0=0x1C;

   PLLCR1=0xD2;

   PLLCR2=0x1D;

   PLLCR3=0xB2;

434 MHz:

/***** Final PLL values for 434 MHz  - FSK=+/-5 kHz – Non_Manchester Coded POL 0  -
No Dx Signal****/

   PLLCR0=0xB0;

   PLLCR1=0x62;

   PLLCR2=0xB1;

   PLLCR3=0x56;
```

4.1.2. Bit rate

The bit rate is set using the RFCR0 register in the above files.

4.1.3. TPMS data set up

Set up for the RF data buffer and the tire ID are described in the next two sections.

4.1.3.1. RF data buffer

The 32-byte format of the RF Frames in the above files are set up as shown below. The first seven bytes are programmed with the preamble and sync data to match those set in the KW01. This is followed by a length byte and an address byte. Implementing a system with a different receiver may require different settings for these registers. The subsequent data bytes represent the payload that the KW01 will extract which contains several bytes of tire ID followed by the Pressure, Accelerometer, Temperature, Voltage and status data readings of the TPMS sensor along with Frame ID, a selection of the verification type, a selection of the data output (for the receiver), counter, and CRC or checksum bytes.

```
(UINT8)(0x55);                  // Preamble
(UINT8)(0x55);                  // Preamble
(UINT8)(0x55);                  // Preamble
(UINT8)(0x01);                  // Sync word
(UINT8)(0x01);                  // Sync word
(UINT8)(0x01);                  // Sync word
(UINT8)(0x01);                  // Sync word
(UINT8)(Payload_Length);        // Payload length, 0x18=24 bytes
(UINT8)(0xF0);                  // MKW01 receiver address (NODE_ADDRESS 0xF0 or BROADCAST_ADDRESS 0xFF)
(UINT8)(Tire_ID >> 24u);        // Tire ID
(UINT8)(Tire_ID >> 16u);        // Tire ID
(UINT8)(Tire_ID >> 8u);         // Tire ID
(UINT8)(Tire_ID);               // Tire ID
(UINT8)(u16CompPressure >> 8u); // Pressure
(UINT8)(u16CompPressure);
(UINT8)(u16CompAccelZ >> 8u);   // Z-axis acceleration
(UINT8)(u16CompAccelZ);
(UINT8)(u16CompAccelX >> 8u);   // X-axis acceleration
(UINT8)(u16CompAccelX);
(UINT8)(gu8CompVolt);           // Voltage
(UINT8)(gu8CompTemp);           // Temperature
(UINT8)(u8StatusAcq);           // Status Acquisition
(UINT8)(FrameID >> 8);          // Frame ID: keep alive counter
(UINT8)(FrameID);
(UINT8)(Verification_Type);     // Verification Type: MKW01 CRC, FXTH CRC, checksum or no verification
(UINT8)(Frame_Display);         // Frame display: hyperterminal, Sensor GUI or selection to be done on the MKW01 side
(UINT8)(0xC1);                  // Fixed data => can be modified by the user
(UINT8)(0xC2);                  // Fixed data => can be modified by the user
(UINT8)(0xC3);                  // Fixed data => can be modified by the user
(UINT8)(0xC4);                  // Fixed data => can be modified by the user
(UINT8)(Verification_Value>>8); // CRC, checksum or fixed data
(UINT8)(Verification_Value);
```

Figure 4. RF data frame configuration

4.1.3.2. Tire ID

The user can choose the tire ID (4 bytes) in the main.c source file.

Two options are available:

- A fixed ID: 0xAA01AA01, 0xBB02BB02, 0xCC03CC03 or 0xDD04DD04. The ID remains the same all the time.
- A cycling ID: it changes each time an RF frame is sent; this is to simulate the four wheels of the car. The ID starts with 0xAA01AA01 in the first frame, 0xBB02BB02 in the second, 0xCC03CC03 in the third, 0xDD04DD04 in the fourth, 0xAA01AA01 in the fifth and then cycles back.

Tire ID selection in the main.c source file:

```
/*****************************************************************
 * Tire ID Selection
 *****************************************************************/

#define ID1 0xAA01AA01
#define ID2 0xBB02BB02
#define ID3 0xCC03CC03
#define ID4 0XDD04DD04

// Uncomment one of the two following l
#define FIXED_ID         0
#define CYCLING_ID    1
/*
 *****************************************************************************
 *                        M A I N
 *****************************************************************************
 */
#pragma CODE_SEG DEFAULT
void main(void)
{

#ifdef FIXED_ID
   /* Uncomment one of the four IDs */
      //Tire_ID = ID1;
         Tire_ID = ID2;
      //Tire_ID = ID3;
      //Tire_ID = ID4;
#endif
```

Choice between fixed and cycling tire ID

In case of a fixed ID, you have a choice of 0xAA01AA01, 0xBB02BB02, 0xCC03CC03 or 0xDD04DD04

4.1.3.3. Frame verification

The user can choose at the beginning of the main.c source file which type of verification will be used for the frame. The two verification bytes are added at the end of the frame.

There are four possibilities:

- MKW01 CRC: CRC (2 bytes) calculated with the algorithm used by the MKW01. A software routine has been added on the FXTH emitter side to compute this CRC.
- FXTH CRC: CRC (2 bytes) calculated with the FXTH firmware function. A software routine has been added on the MKW01 side to compute this CRC.
- Checksum: checksum (1 byte). As the checksum is 1 byte only, the other byte is fixed data.
- No verification: the two bytes are fixed data.

A *Verification_Type* byte has also been added to the frame to indicate which type of verification has been chosen. On the MKW01 side, this byte is read and the correct verification algorithm is used accordingly. The possible values of this byte are the following (transparent to the user):

- 0x03: MKW01 CRC
- 0x02: FXTH CRC
- 0x01: checksum
- 0x00: no verification

Frame verification selection in the main.c source file:

```
/*****************************************************************
*
* Data verification selection
*
*****************************************************************/
#define VERIFICATION_CRC_KW01              1
#define VERIFICATION_CRC_FXTH              0
#define VERIFICATION_CHECKSUM              0
#define VERIFICATION_NO_VERIFICATION       0
```

On the KW01 side, whatever the verification result is, the reception is not aborted. The result is given by one byte: *Frame_Verification_Result.* This byte is sent to the hyperterminal or to the Sensor GUI.

The possible values are:

- 0x00: verification ok
- 0x11: checksum verification failed
- 0x22: FXTH CRC verification failed

- 0x33: MKW01 CRC verification failed
- 0x44: no verification selected

4.1.3.4. Data output

The user can choose at the beginning of the main.c source file the way frames will be displayed when the KW01 will receive them. Three options are available:

- Frames will be displayed on the hyperterminal (byte Frame_Display = 0x11)
- Frames will be displayed using the Freescale Sensor GUI (byte Frame_Display = 0x22)
- The choice of the display (hyperterminal or Sensor GUI) will be done in the KW01 receiver project (byte Frame_Display = 0x00)

Frame display selection in the main.c source file:

```
/****************************************************************
 *
 * Frame display selection
 * Choose here the way frames will be displayed on the MKW01 receiver side:
 * using the hyperterminal or the Freescale Sensor GUI.
 * Or the selection can be done in the KW01 project.
 *
 ****************************************************************/
/* Uncomment one of the defines */
//#define Frame_Display       0x11  /* Frame will be displayed using the
HYPERTERMINAL */
//#define Frame_Display       0x22  /* Frame will be displayed using the SENSOR GUI
*/
#define Frame_Display         0x00   /* The choice of the display will be done on
the MKW01 side */
```

5. KW01 receiver demonstration setup

This section describes how to program the KW01. The *SimpleRangeDemo* demo file modified as described in Section 3, sets the carrier frequency to 434 MHz, frequency deviation to 50 KHz, Bitrate to nominally 19.2 Kbps (actual bitrate is close to 19230 bps), NRZ mode, 3 byte preamble and 4 bytes of sync value and verification as an example to demonstrate the implementation of the KW01 as receiver. A similar technique is used to program the FXTH87xx further below.

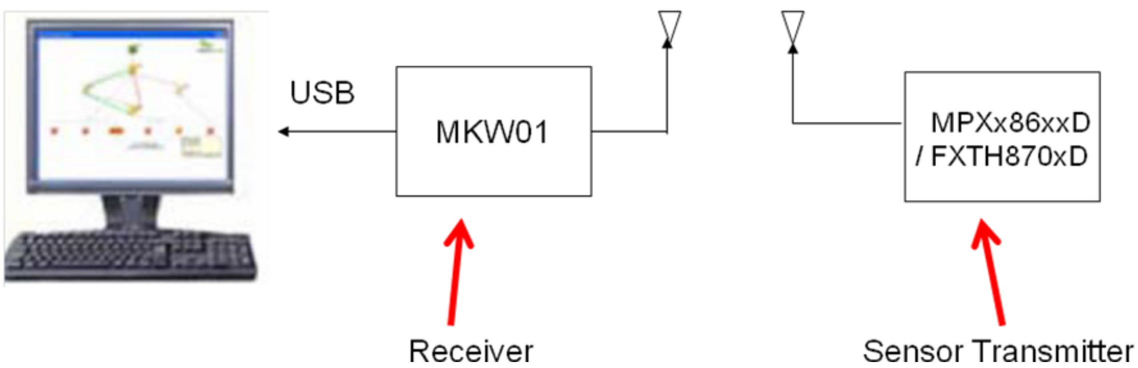

Figure 5. KW01 Receiver Demonstration

5.1. Downloading the firmware

1. Go to freescale.com/TPMS
2. Click on the FXTH87 link.
3. Click on the [Software and Tools] tab.
4. Expand the Software Development Tools section.
5. Download the relevant TPMS projects. (TPMS FXTH87 MKW01 CW10 Rev4 and TPMS MKW01 IAR7v4 Project Rev0.6).

TPMS FXTH87 MKW01 CW10 Rev4: TPMS FXTH87 emitter project on CodeWarrior 10. This project is to be used with the TPMS-MKW01-IAR7v4-Project_Rev0.6 **Size (K):** 2395 **Format:** zip **Rev #:** 4 **Modified:** 5/6/2015	FREESCALE	Download	☆
TPMS FXTH87 MKW01 LF Triggered CW10 Rev1: TPMS FXTH87 emitter project on CodeWarrior 10. RF Data Frame transmission is triggered by LF (125kHz). This project is to be used with the TPMS-MKW01-IAR7v4-Project_Rev0.6. **Size (K):** 4767 **Format:** zip **Rev #:** 1 **Modified:** 5/20/2015	FREESCALE	Download	☆
TPMS MKW01 IAR7v4 Project Rev0.6: TPMS MKW01 receiver project on IAR7v4 using MRB-KW019032EU. This project is to be used with the TPMS-FXTH87-MKW01-CW10-Rev4 emitter project to display RF data received from TPMS FXTH87 emitter **Size (K):** 12674 **Format:** zip **Rev #:** 0.6 **Modified:** 5/6/2015	FREESCALE	Download	☆

5.2. Loading an application into the KW01 Modular Reference Board (MRB)

1. Locate the KW01 demo application on your computer.
2. Open IAR Embedded Workbench (version 7.4 is recommended).
3. If using an evaluation license, select the Time Limited (30 days) license as the Code Size Limited license will not allow reprogramming.
4. Select ‘Open’ and then select ‘Workspace’ from the File menu and locate the folder where the project was extracted.
5. Select the project (SimpleRangeDemo.eww) and click ‘Open’. SimpleRangeDemo.eww is located in the folder to which the project file is extracted, for instance: <filepath>/KW01_IAR7v4_Project_TPMS_Rev0.5/SimpleRangeDemo.eww. (or other folder name chosen). See Figure 6 below.
6. Use a JTAG connection interface to download the application into MRB-KW01.
7. Click the “Download and Debug” button in IAR.
8. After the FLASH memory is programmed, JTAG should be disconnected and power to the MRB cycled.
9. Proceed to section 5.2 for setup of the terminal emulator.

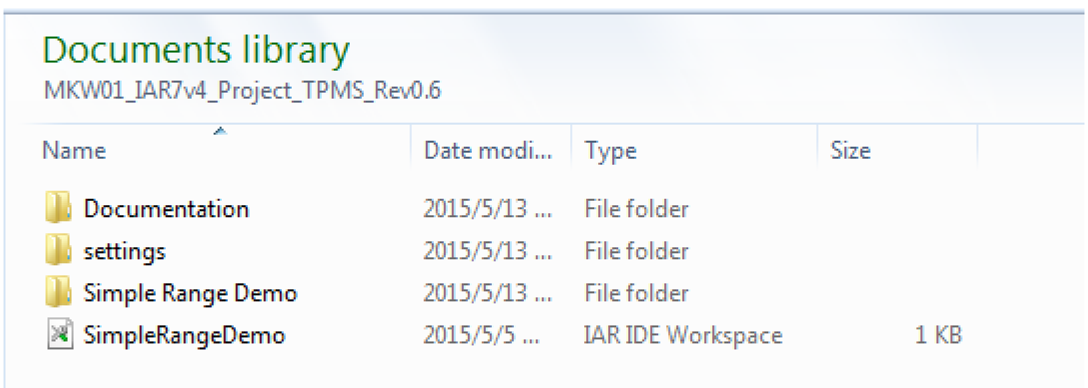

Figure 6. SimpleRangeDemo.eww location

5.3. Virtual COM port setup

A terminal emulator is used to communicate with the MRB-KW01 configured as TPMS receiver.

You must ensure that the COM port is set to 115200, 8, N, 1 and No Flow Control.

5.4. Loading an application into the FXTH87 series sensor board

1. Download one of the TPMS_FXTH87_MKW01_CW projects available on the FXTH87 webpage at freescale.com/TPMS, specifically: Software&Tools.
2. Open CodeWarrior 10.x.
3. Click on Import project.
4. Click on Browse and select the project folder, as shown in the following figure.

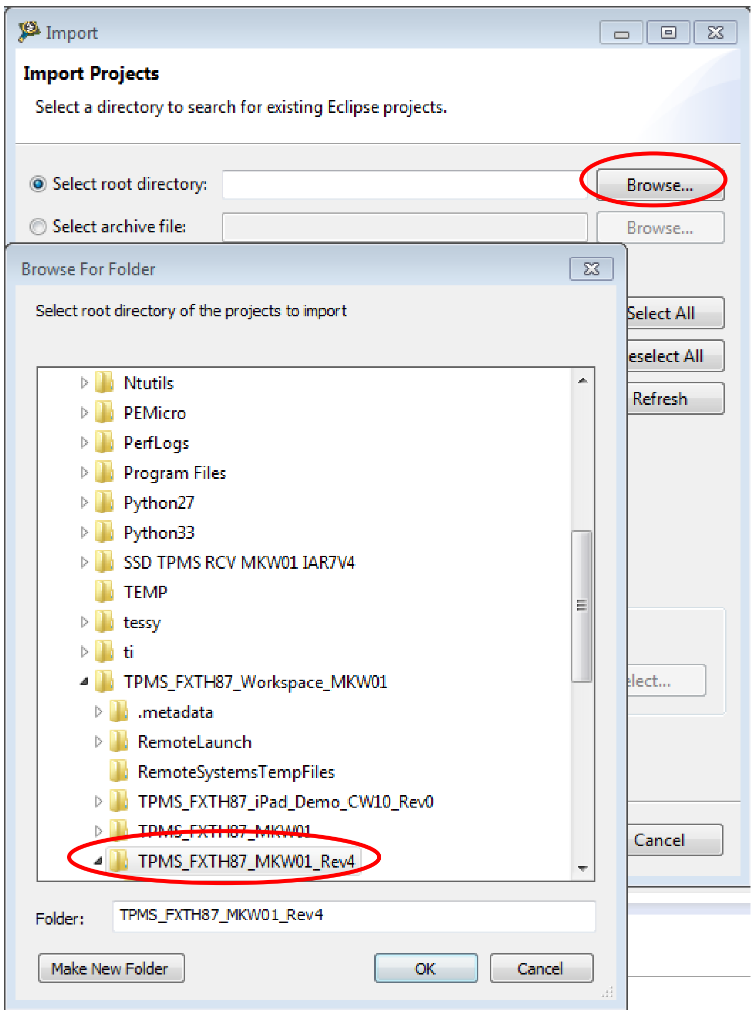

Figure 7. Selection of the project folder

1. Use multi-link debugger connection interface to download the application into FXTH870xD board.
2. Click the "Debug" button in Codewarrior (see figure 8 below).
3. After the FLASH memory is programmed, disconnect the debugger and cycle the power on the FXTH870xD board.
4. The FXTH870xD board will transmit data after cycling power as shown in section 5.4 below.

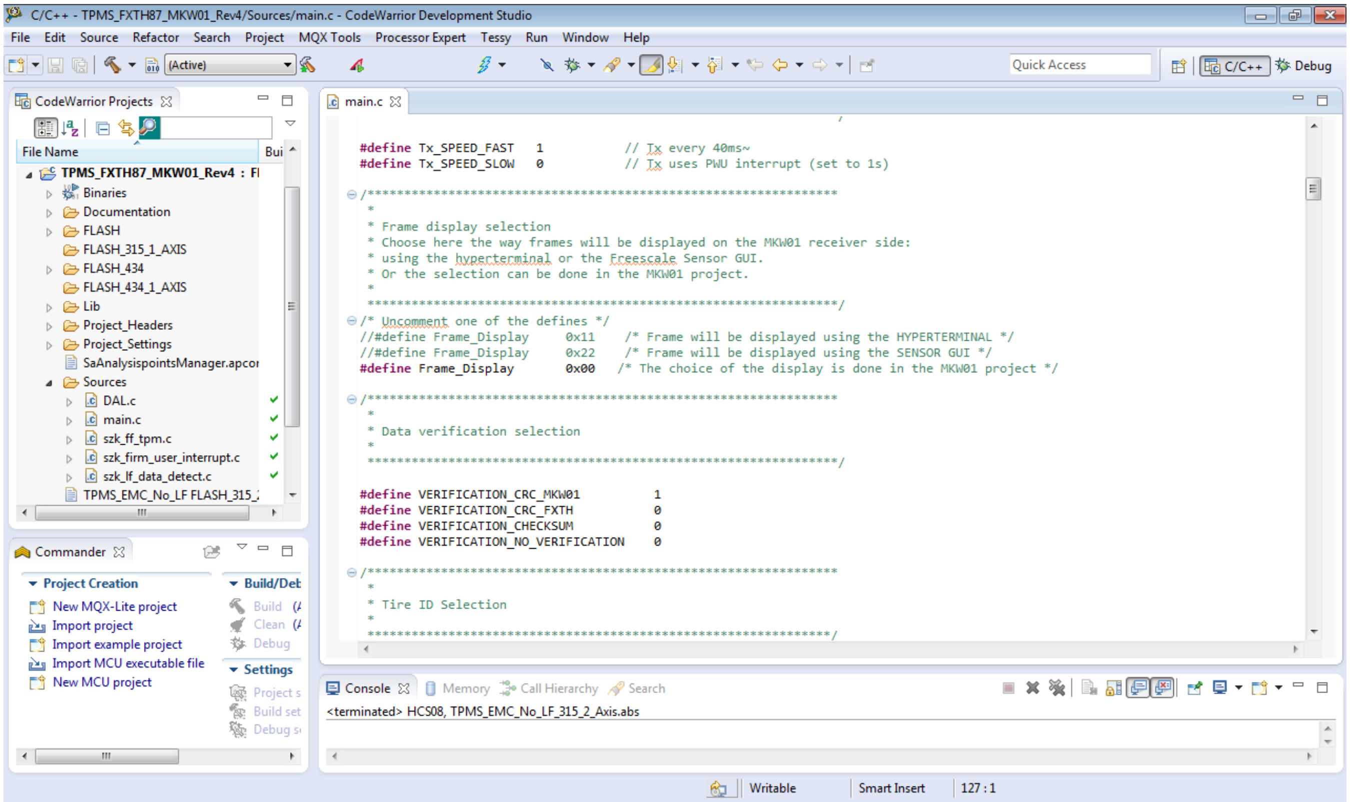

Figure 8. Codewarrior Screen

5.5. KW01 data output system

The choices available for data output system are hyperterminal or the FSL Sensor GUI. Screenshots of the displays are shown below.

5.5.1. KW01 output to the hyperterminal

In the screenshots below, CYCLING_ID has been selected. The firmware in the KW01 hides some of the bits and data transmitted by the FXTH870xD such as the preamble and sync word at the beginning and the checksum/CRC and other bytes at the end and displays the packets as single lines starting with the Tire ID and ending with a packet counter, as shown below.

The frame format consists of the following (more details can be found in file Hyperterminal.h of the MKW01 project to convert data in raw hex bytes into numbers with common units):

- TireID (4 bytes)
- Pressure (2 bytes)
- AccelZ (2 bytes)
- AccelX (2 bytes)
  - Volt (1 byte)
  - Temp (1 byte)

- StatusAcq (1 byte)
- FrameVerificationResult (1 byte)
- Counter (2 bytes)

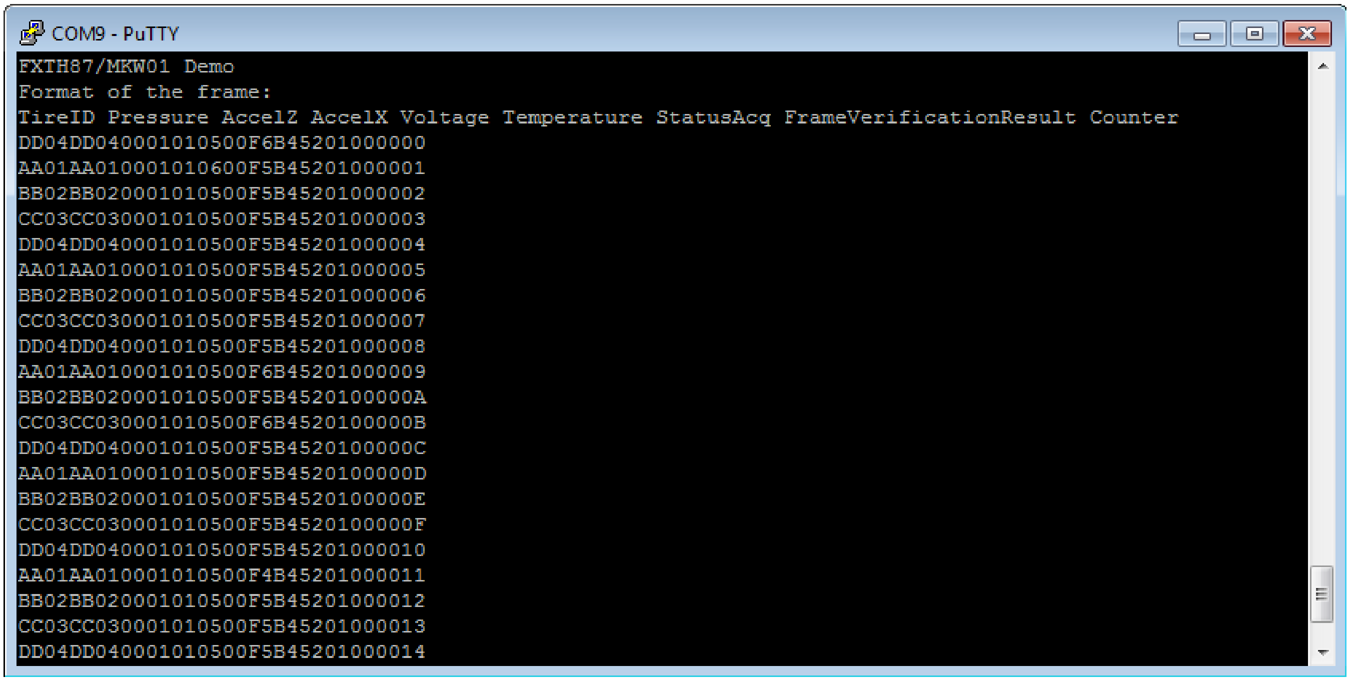

Figure 9. Received frame data

5.5.2. KW01 output to the LabVIEW Sensor GUI

5.5.2.1. Sensor GUI display

The Sensor GUI can be downloaded from the FXTH87 web page, specifically: Software&Tools

In the screenshots captured below in Figures 10 and 11, of a TPMS871xxx 1500 kPa emitter, eight graphs are available for data display.

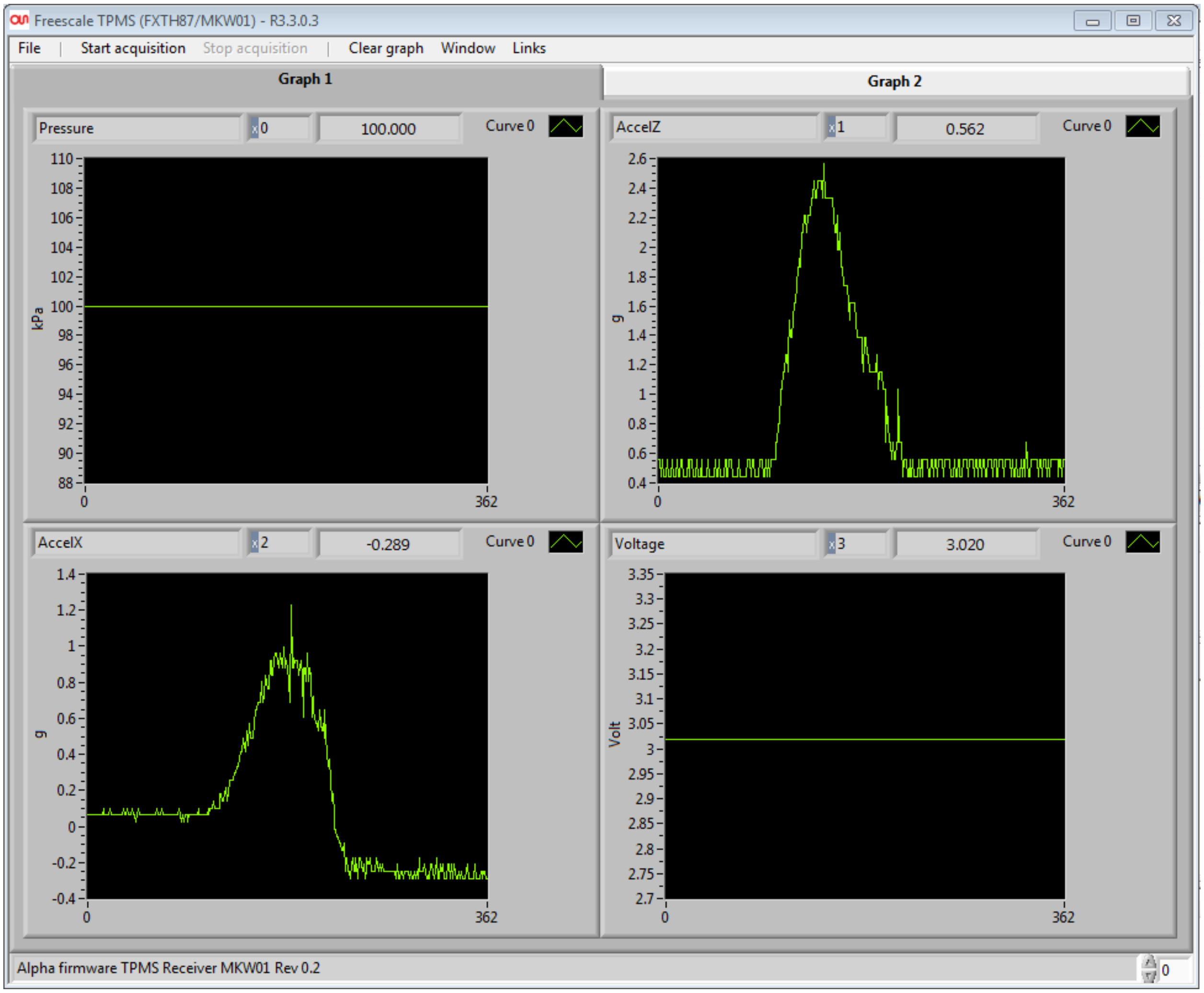

Figure 10. Screenshot of Labview Sensor GUI

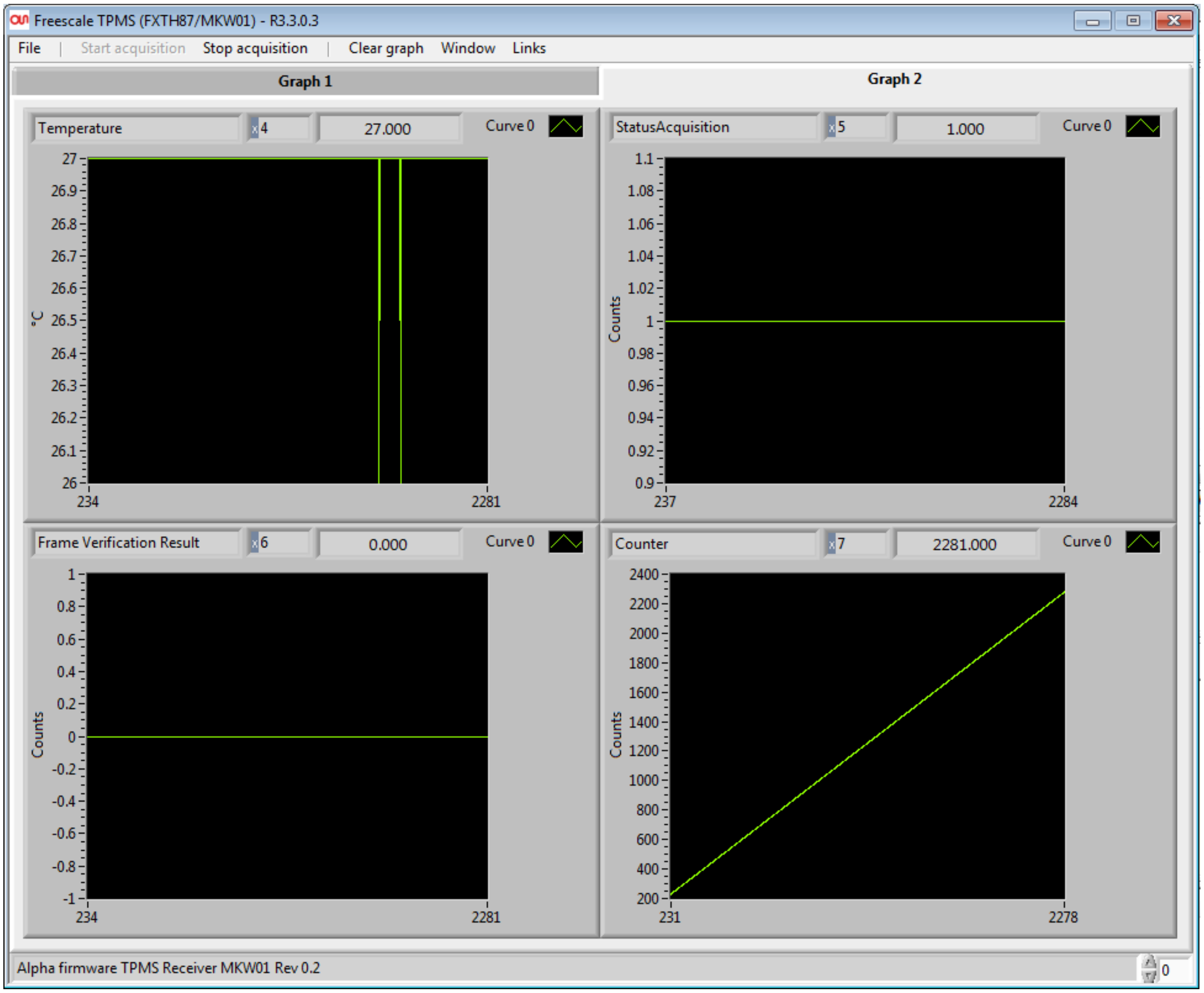

Figure 11. Screenshot of Labview Sensor GUI

Eight curves are displayed:

- Pressure: pressure given by the TPMS module, in kPa, shown above as a constant 100 kPa.
- AccelZ and AccelX: X and Z-axis accelerations given by the TPMS module, in g. The display shows varying data above in 2 windows.
- Voltage: voltage source (battery) reported by the TPMS module, in volts, shown above as a constant 3.02V.
- Temperature: temperature reported by the TPMS module, in °C. Shown as a constant 27 °C above.
- StatusAcquisition: status acquisition reported by the TPMS module (Typically 0 designates proper tire performance. The value shown above is 1, a flag indicating that pressure is below a set limit of approximately 350 kPa. This indicates an underinflated tire.)
- Frame verification result: indicating the result of the frame verification. Displayed above is 0, indicating that the verification passed.
- Counter: a counter incremented by the KW01 each time a frame is sent through the USB CDC

port. 2 Windows.

5.5.2.2. Sensor GUI frame format

The frames of data sent from the KW01 to the Sensor GUI are configured in the following format:

[DA080001123400010112 00E7B15301000017]

The various bytes are defined as:

- [the bracket indicates the start of the frame.
- DA indicates that data is transferred. The other possibilities are MS (for message screen) and MP (for message pop up). See the file LabVIEW_GUI.c of the KW01 project for more information.
- 08 indicates that eight curves will be displayed.
- 0001 indicates that one set of sensor acquisition is transferred.
- 1234 can be used for specific purposes (time stamp for example).
- 2 bytes for the pressure (here 0001).
- 2 bytes for the Z-axis acceleration (here 0112).
- 2 bytes for the X-axis acceleration (here 00E7).
- 1 byte for the voltage (here B1).
- 1 bytes for the temperature (here 53).
- 1 byte for the Status Acquisition (here 01).
- 1 byte for the frame verification result (here 00).
- 2 bytes for the counter (here 0017).
-] the bracket indicates the end of the frame.

5.5.2.3. Sensor GUI configuration panels

In the Sensor GUI the UART baud rate and the port COM number need to be updated.

Go to *Files* > *Settings* > *Edit settings*

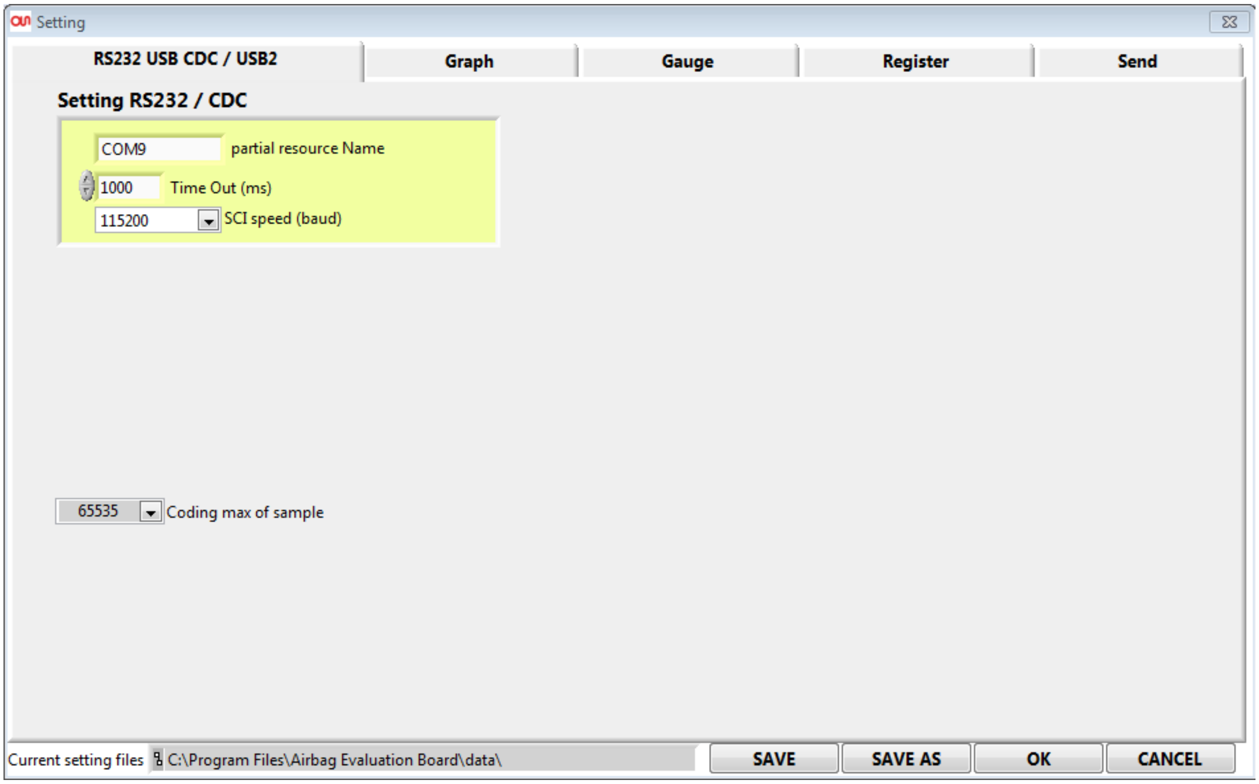

Figure 12. GUI COM Port setting window.

This figure shows 115200 bps chosen as the UART baud rate (this can be modified in the KW01 project), and allows update of the COM port number if necessary.

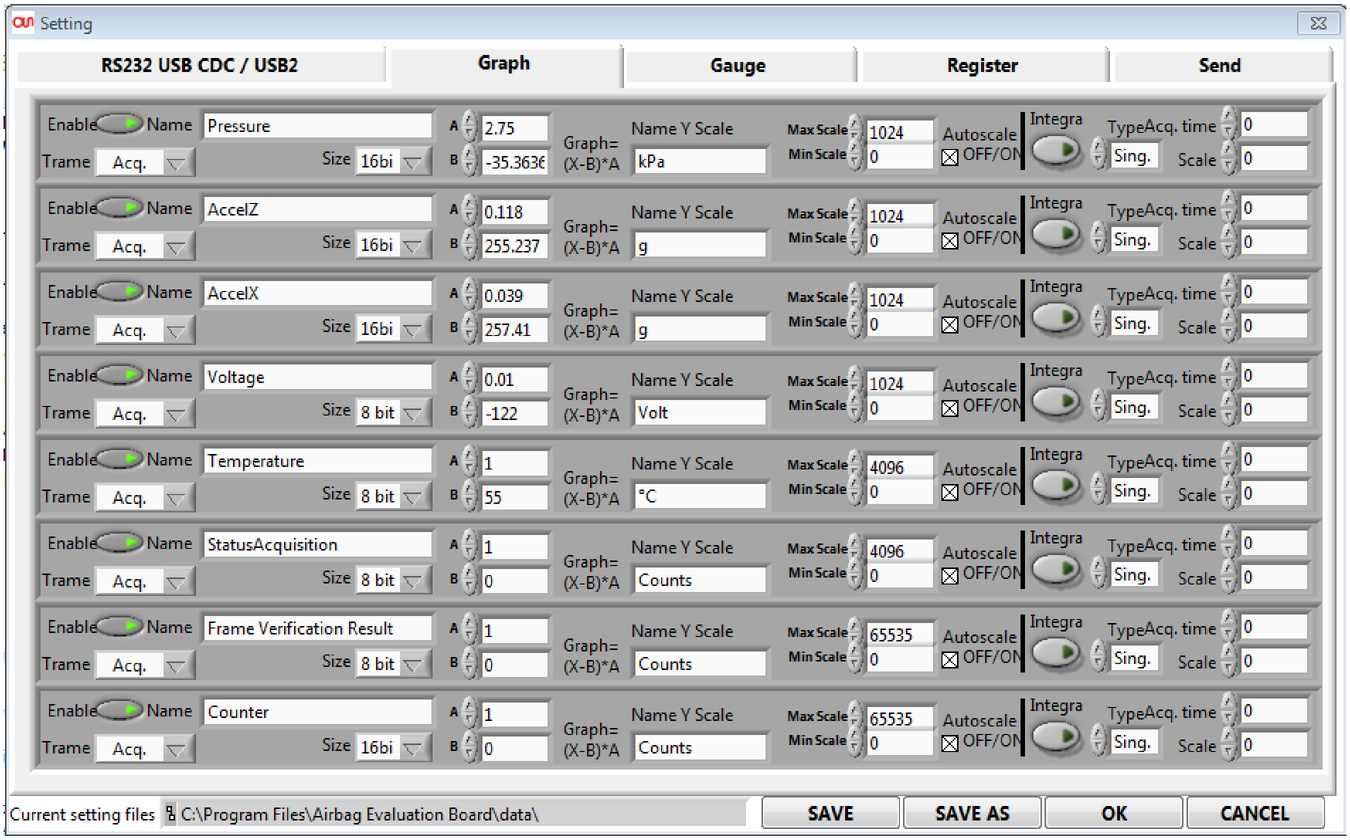

Figure 13. GUI data conversion set up window.

The settings shown in this figure enable the GUI to convert hexadecimal data coming from the KW01 (data in counts) into data displayed using familiar units (such as kPa, °C, and so on). These settings are adjusted for the FXTH871xxx family of products.

See the file LabVIEW_GUI.h of the KW01 project for more information on data conversion.

6. References

The chapters in this application note contain a summary of the most important details of each topic.

For more details on any of the topics of this document see the following documents.

1. *MKW01Z128* Sub 1 GHz Low Power Transceiver plus Microcontroller Reference Manual, (MKW01xxRM).
2. *KW01 Development Hardware*, (KW01DHRM).
3. *MRB-KW01-9032 Schematic*, (can be found at freescale.com/mrb-kw0x, click on the “downloads” tab, then select "MRB-KW0x Design Files”).
4. *MKW01 Simple Media Access Controller*, (MKW01SMACRM).
5. *MKW01 Demonstration Application User Guide*, (MKW01DAUG).
6. *Xtrinsic FXTH87 Family Evaluation Design Reference Manual*, (FXTH87EDRM).

7. *FXTH870xD Tire Pressure Monitor Sensor Data Sheet* (FXTH870xD)
8. *Xtrinsic MPXx85/86xxD Tire Pressure Monitor Sensor Data Sheet* (MPXX8XXXD)

7. Revision history

Table 2. Revision history

Revision number	Date	Substantive changes
0	07/2015	Initial release

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: AN5136
Rev. 0
07/2015