Communication between TPMS FXTH87 emitter and MKW01 receiver

This document 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 the tests on the range of MKW01 + FXTH87.
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.
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), CRC and checksum frame verifications, and a choice of the MKW01 node and broadcast addresses. The new project is *KW01_IAR7v4_Project_TPMS_Rev0.6*

For latest information, refer to FXTH87 webpage, specifically Software&Tools.

Contents

Contact : Pascal BERNARD (R20246), Camille SAINT-JEAN (B50961)

1. FXTH87 emitter

a. Frame format

The format of the RF frames is the following. The length is 32 bytes and that corresponds to the maximum frame length of the emitter (it is the size of the FXTH87 RF buffer).

```
(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);
```

b. Concerning the tire IDs

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 to emulate different emitters (tires). 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…

Tire ID selection in the main.c source file:

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

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

#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, choice between 0xAA01AA01, 0xBB02BB02, 0xCC03CC03 or 0xDD04DD04

c. Frame verification

The user can choose at the beginning of the main which type of verification will be used for the frame. The two verification bytes are added at the end of the frame.
The possibilities are the following:

- 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. So 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_MKW01                1
#define VERIFICATION_CRC_FXTH                 0
#define VERIFICATION_CHECKSUM                 0
#define VERIFICATION_NO_VERIFICATION          0
```

On the MKW01 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 the following:

- 0x00: verification ok
- 0x11: checksum verification failed
- 0x22: FXTH CRC verification failed
- 0x33: MKW01 CRC verification failed
- 0x44: no verification selected

d. Frame display

The user can choose at the beginning of the main the way frames will be displayed when the MKW01 will receive them. Three options are available:

- Frames will be displayed on the hyperterminal (Frame_Display = 0x11)
- Frames will be displayed using the Freescale Sensor GUI (Frame_Display = 0x22)
- The choice of the display (hyperterminal or Sensor GUI) will be done in the MKW01 receiver project (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 MKW01 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 */
```

2. MKW01 receiver

In the file main.h the following settings can be selected:

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

a. Choice of the Rx carrier frequency

The user can choose between a RF carrier frequency of 315, 434 or 915MHz.

b. Choice the node and broadcast addresses

If the address byte sent on the emitter side does not match one of these two bytes the reception is aborted.

c. Choice between a display on the hyperterminal or on the FSL Sensor GUI

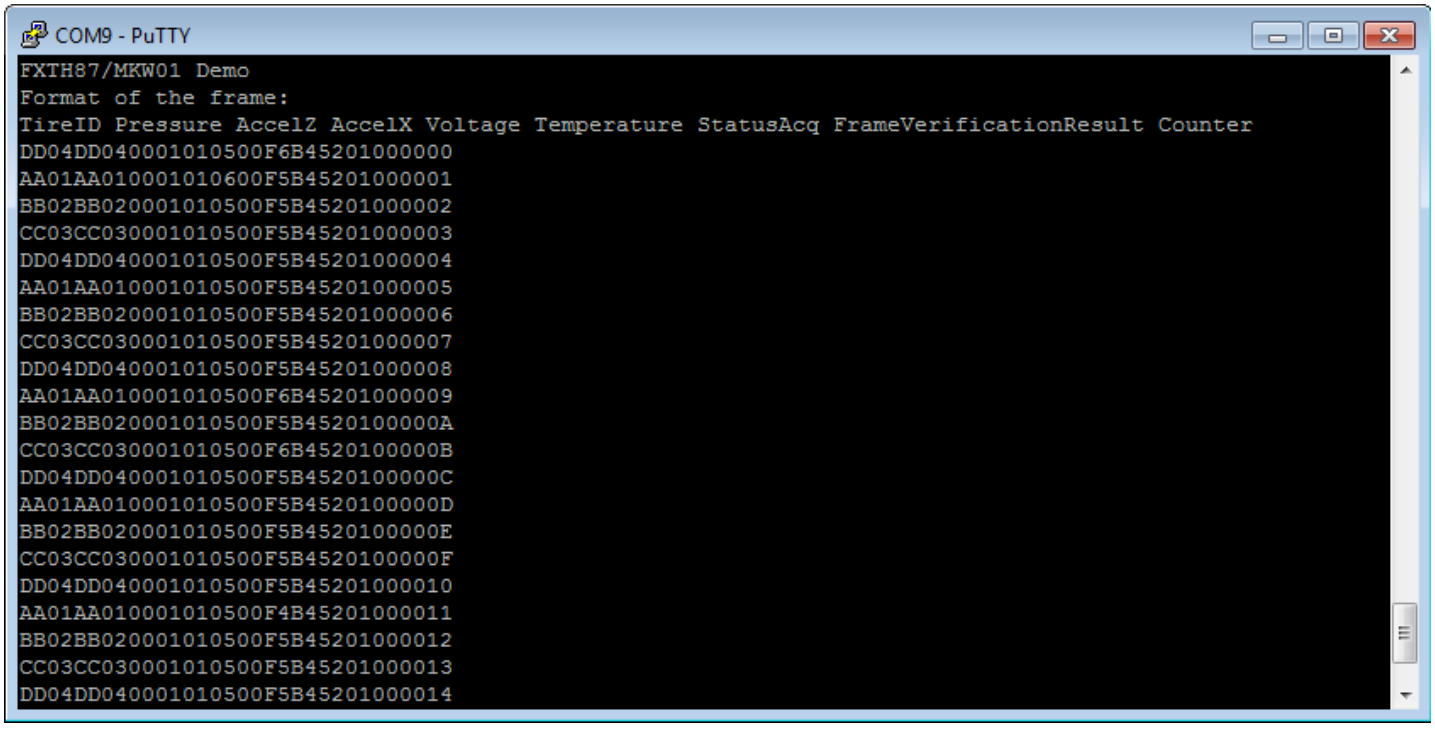
Once an RF frame has been received, a new frame containing the payload of the RF frame is sent through the UART.
The user can choose between watching the frames on the hyperterminal or on the Freescale LabVIEW Sensor GUI. This selection can be done in the FXTH87 emitter project (see above) or in the MKW01 project.

Below are screenshots of what is displayed for the user:

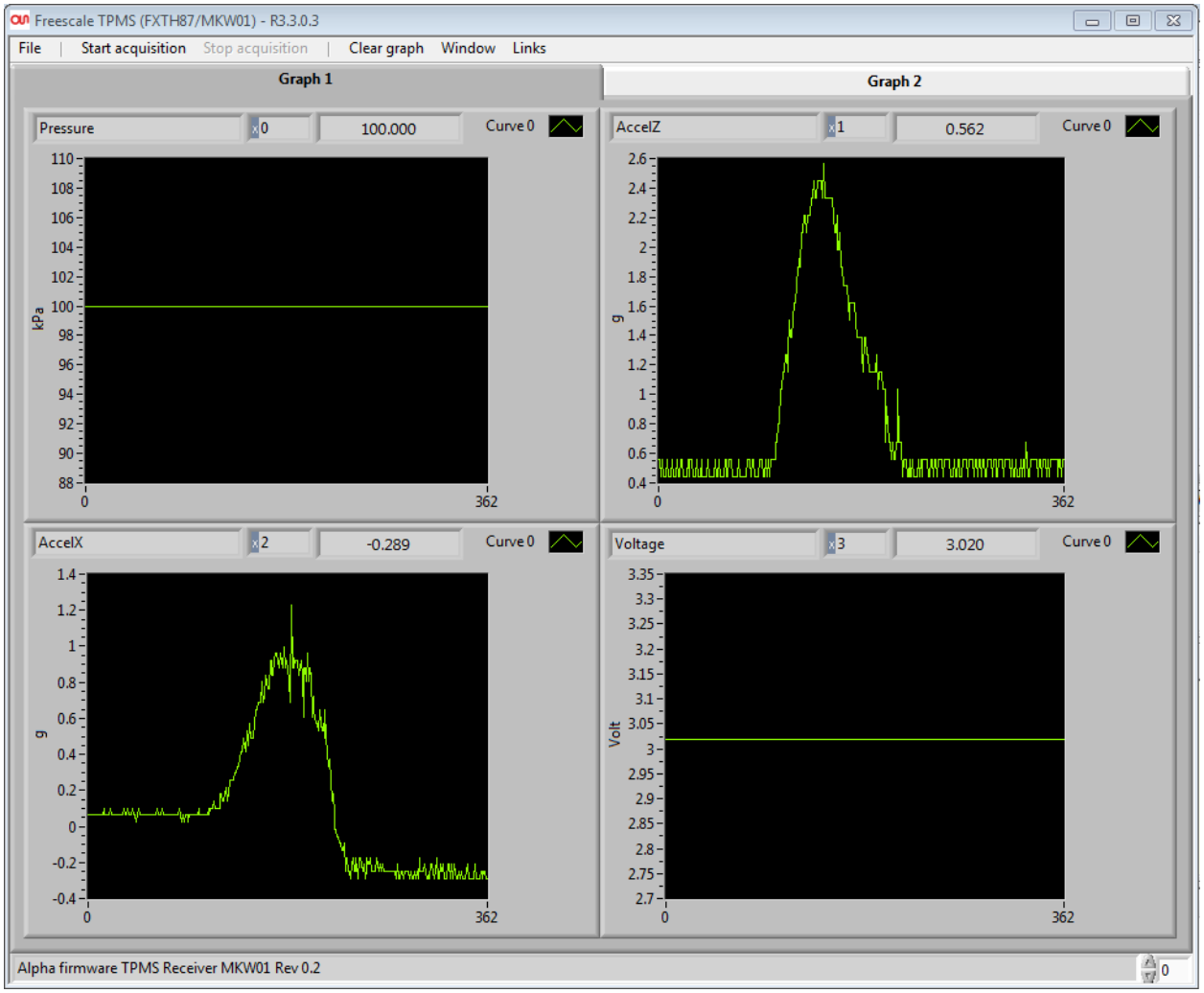
In case of the hyperterminal (CYCLING_ID has been selected):

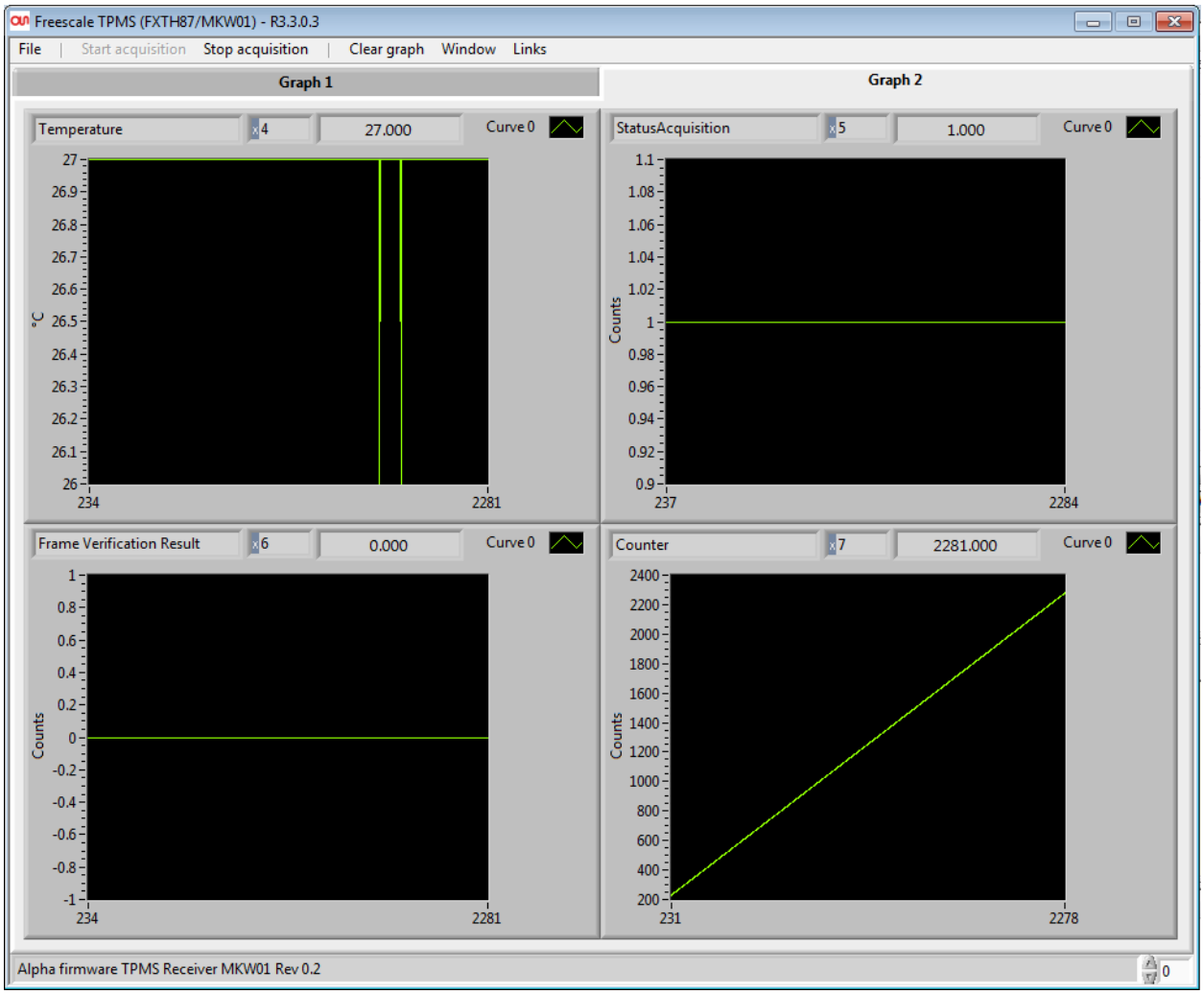
Frame format (more details in Hyperterminal.h of the MKW01 project):

TireID (4bytes) Pressure (2bytes) AccelZ (2bytes) AccelX(2bytes) Volt(1byte) Temp(1byte) StatusAcq(1byte) FrameVerificationResult(1 byte) Counter(2bytes)



In case of the LabVIEW Sensor GUI (8 data can be displayed, see below):





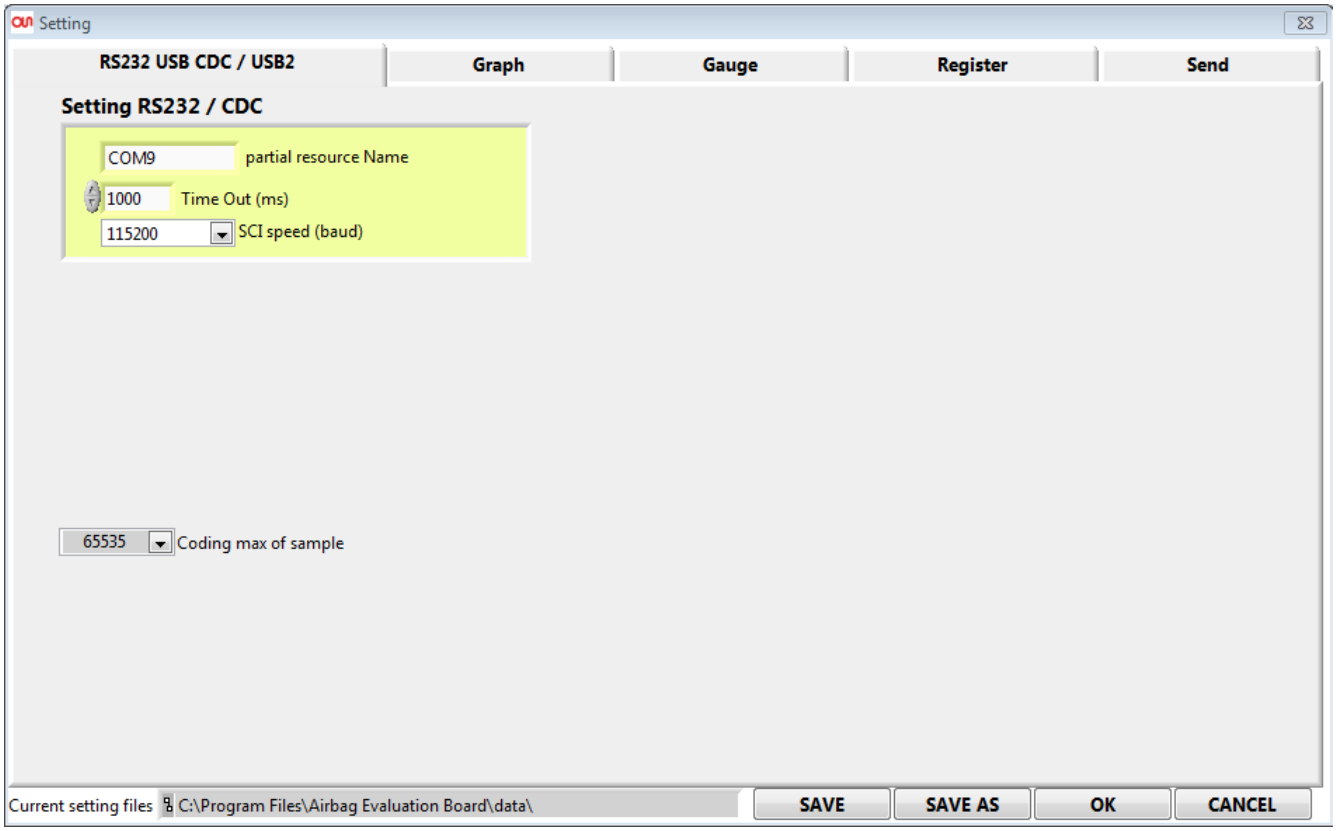
A TPMS870911 1500kPa emitter was used to do the screenshots.

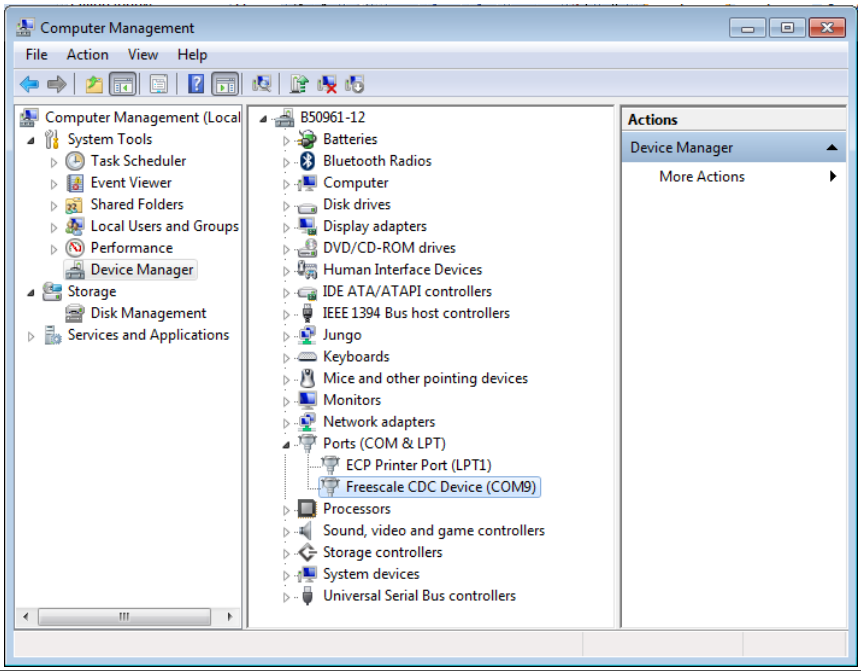
The 8 curves are showing:

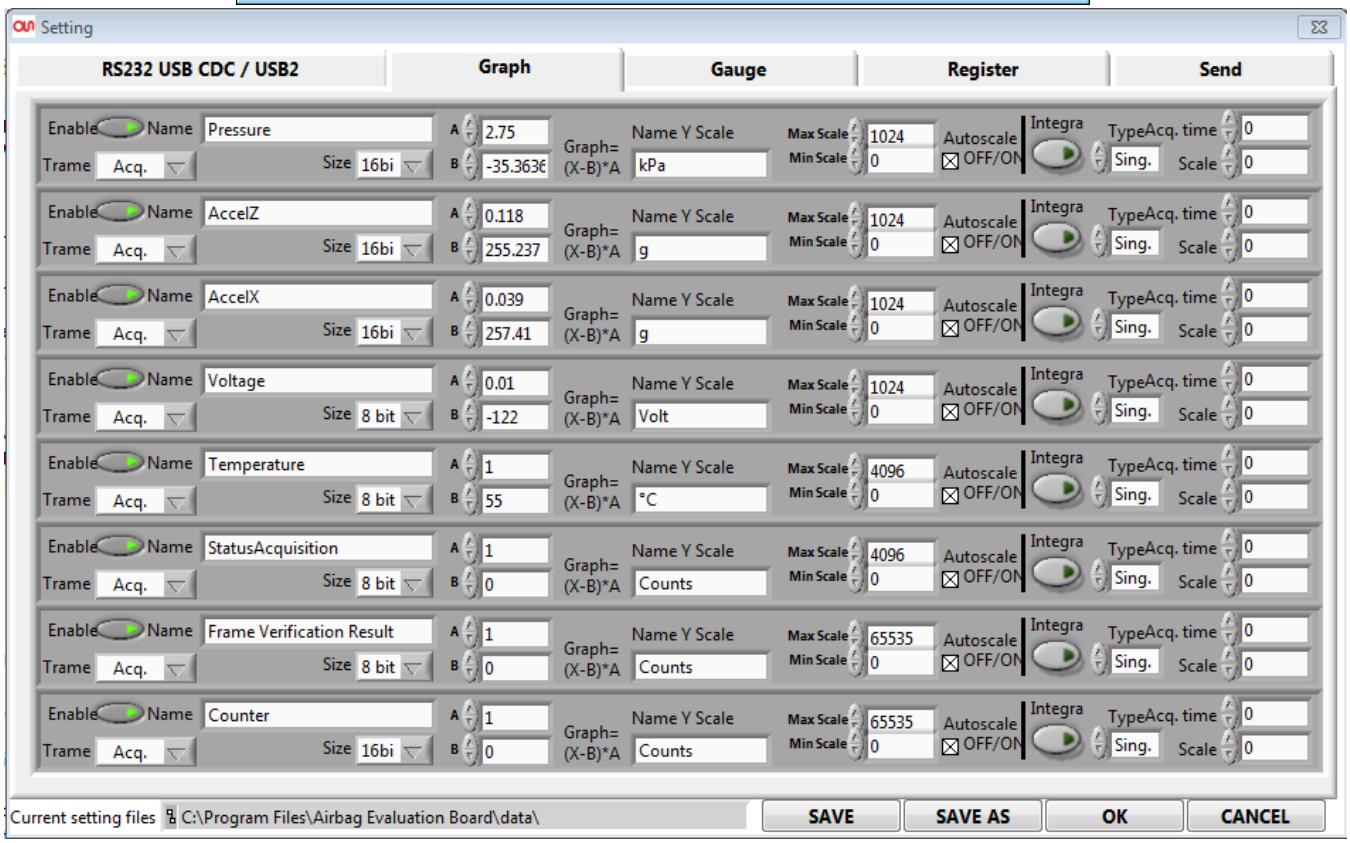
- Pressure: pressure given by the TPMS module, in kPa
- AccelZ and AccelX: X and Z-axis accelerations given by the TPMS module, in g
- Voltage: voltage (power supply) given by the TPMS module, in volts
- Temperature: temperature given by the TPMS module, in °C
- StatusAcquisition: status acquisition given by the TPMS module (0 if ok, other value if not ok – here it is 1 because pressure is under a certain level around 350kPa, so the tire is considered as flat)
- Frame Verification Result: see above, *Frame Verification Selection*
- Counter: a counter incremented by the MKW01 each time a frame is sent through the USB CDC port.

Concerning the Sensor GUI settings, below are screenshots of the configuration panels. The configuration file that has been used is SettingTPMS_MKW01_COM9_1500kPa.cfg. This file is to be used with a 1500kPa emitter device and directly converts raw data coming from the RF frame (in counts) to data in common units (kPa, g , °C, volts) .

The file can be found in the folder *Documentation* in the zip file of the MKW01 project.





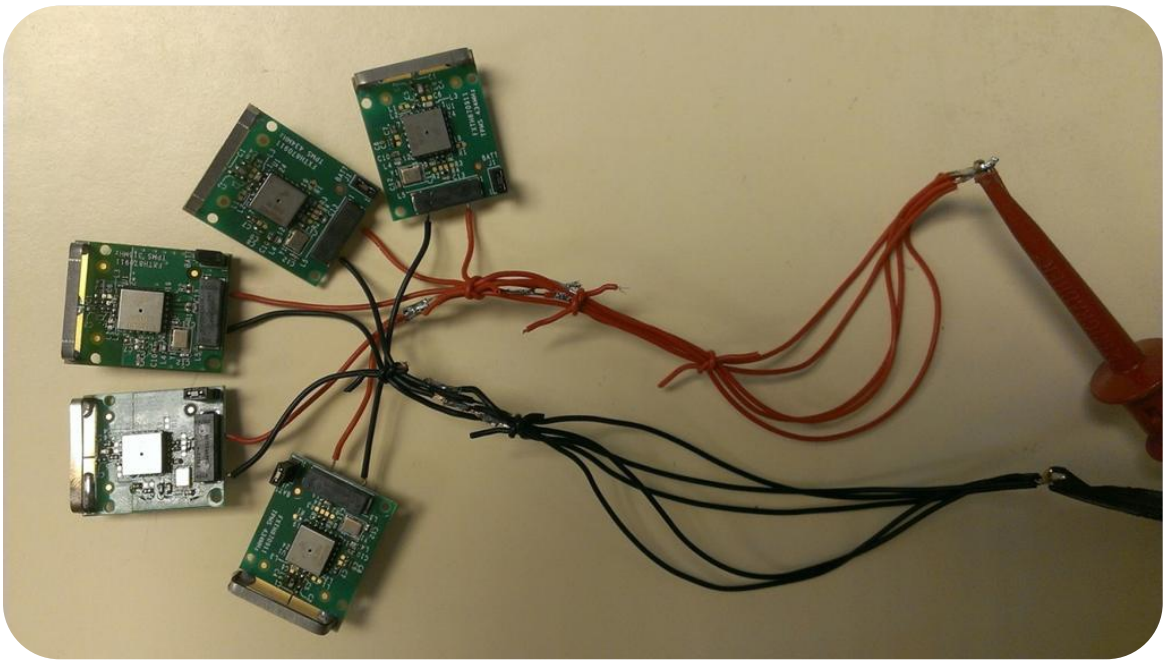


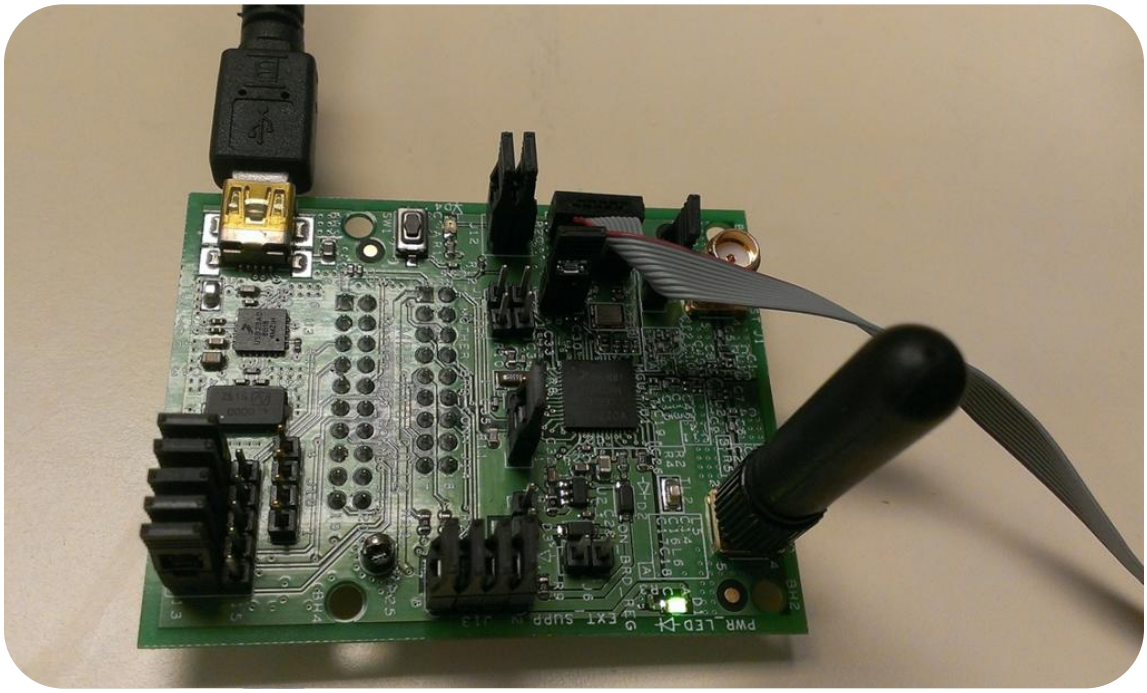
The Freescale LabVIEW Sensor GUI can be downloaded from freescale webpage:

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FXTH87&fpsp=1&tab=Design_Tools_Tab

3. Demo setup

In order to test the emitter and receiver projects, one TPMS emitter with a cycling ID or 4 emitters with fixed ID and an emitter with a wrong ID transmitted frames at the same time, and the received frames were watched with the hyperterminal or the LabVIEW Sensor GUI.





4. Influence of settings on the frame reception

These two tables are to explain the choices made concerning the baud rate (for the RF frame and the UART on the MKW01 side), as well as the number of preamble bytes and synchronization words.

The emitter and receiver were at a distance of 1m approximately with no obstacle in-between.

First settings:

- Preamble: 3 bytes
- Sync words: 4 bytes
- Payload: 18 bytes
- RF frame baud rate: 19230 baud/s
- MKW01 UART baud rate: 50000 baud/s

That gives a loss of 1.5%. And this value keeps constant when we increase the distance between emitter and receiver.
The distance between the emitter and receiver can go to more than 20 meters. The free air distance was calculated using a TPMSFXTH870911 emitter and a MRB-KW019032 receiver with RF frequencies at 315MHz and 434MHz.

Preamble size (bytes)	Number of sync words (bytes)	RF frame baud rate (b/s)	UART baud rate – MKW01 side (b/s)	Size of the RF frame (bytes)	Loss
3	4	9600	19200	25	4.7%
3	4	19230	19200	25	4.6%
3	4	19230	50000	25	1.5%
3	4	9600	50000	25	4.4%

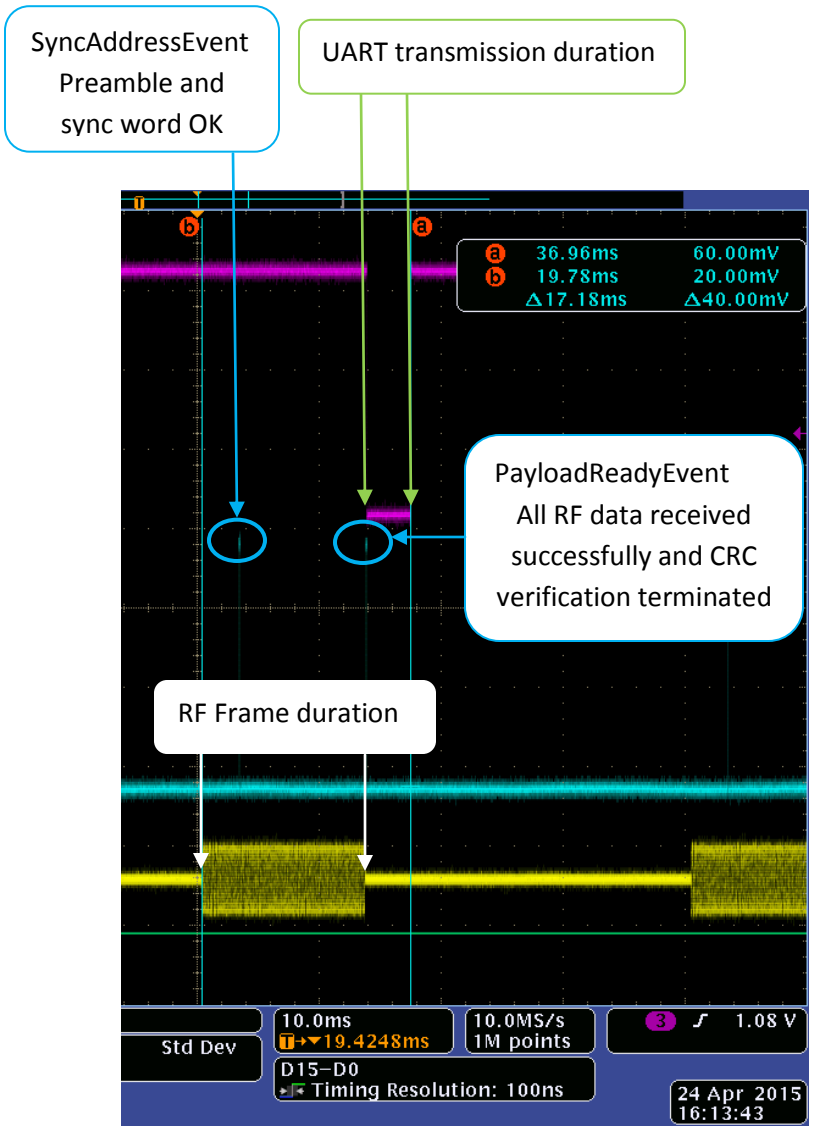
Preamble size (bytes)	Number of sync words (bytes)	RF frame baud rate (b/s)	UART baud rate – MKW01 side (b/s)	Size of the RF frame (bytes)	Loss
3	3	19230	50000	24	1.8%
3	2	19230	50000	23	5.3%
2	4	19230	50000	24	2.12%
1	4	19230	50000	23	2.3%

Final settings:

- Preamble: 3 bytes
- Sync words: 4 bytes
- Payload: 25 bytes
  Length, MKW01 address, TireID (4bytes), Sensing data (9bytes), counter (2bytes), CRC or checksum selection (1byte), fixed data (5bytes), CRC or checksum (2bytes)
- RF frame baud rate: 19230 baud/s
- MKW01 UART baud rate: 115200 baud/s

Successful RF reception: 98% (same setup as above, 1m distance between emitter and receiver).

Reception timing:



Reception duration

From the moment the RF frame is sent to the end of the UART transmission : 17.18ms

RF frames are sent every 40ms.

Between SyncAddressEvent and PayloadReadyEvent RF data frame is loaded in FIFO and accessed by SPI for processing, and then sent to hyperterminal or dedicated visual interface.