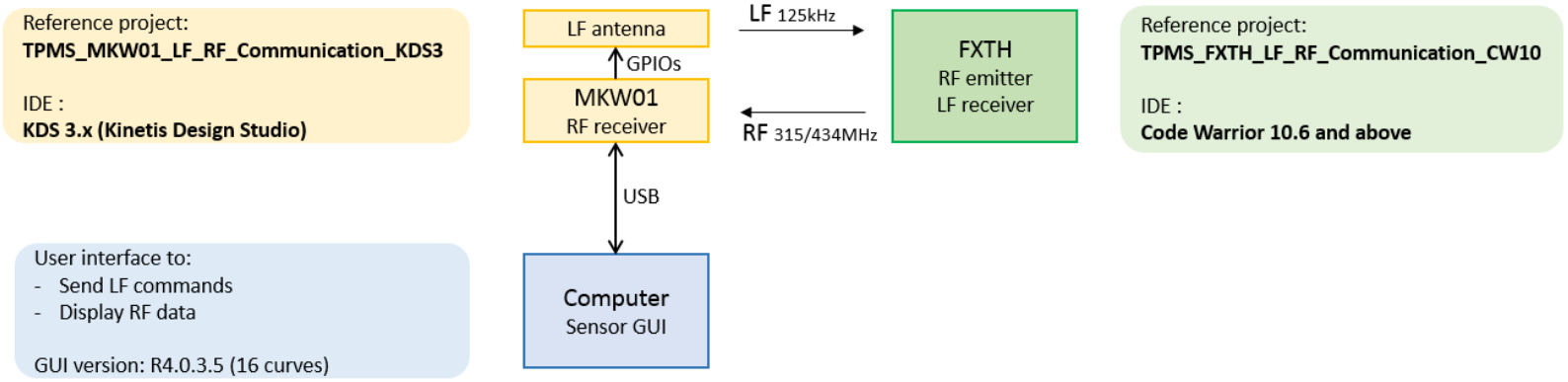
User Guide TPMS FXTH/MKW01 reference projects

Contents

1. Demo overview



Both reference projects (for FXTH and MKW01) as well as the Sensor GUI can be downloaded from NXP website, FXTH87 Software&Tools webpage (www.nxp.com > Sensors > Pressure Sensors > Tire Pressure Monitoring Sensors > FXTH87 > Software & Tools), under *Lab and Test Software*.
Information on demo setup is given later on in the user guide.

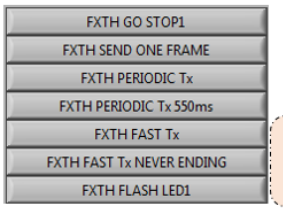
This demo can operate in three different modes. The operating mode is selected in the FXTH reference project, in the file *user_configuration.h.* The MKW01 project is the same for the three modes (no modification to do on the MKW01 side).

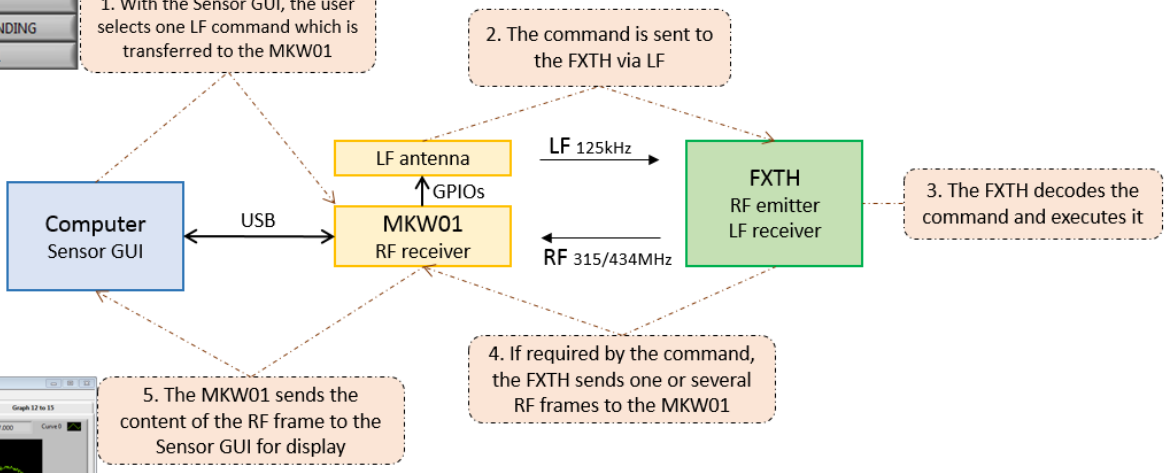
Selection of the operating mode in the FXTH reference project:

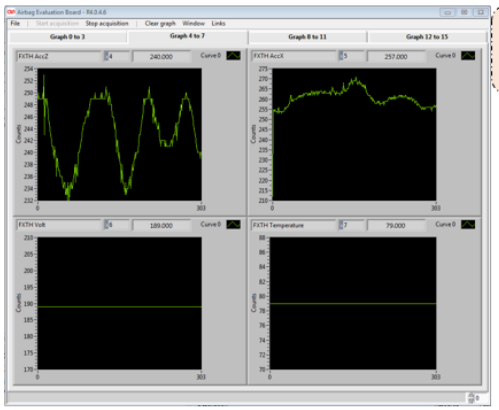
```
/*****************************************************************************
 *  - The defines below are to choose the operating mode
 *
 *  - LF_COMMUNICATION : FXTH is waiting for LF commands in STOP1 with LF block enabled.
 *                       When a valid LF command is received it is decoded and executed
 *                       Refer to function vfnUpdateStateMachine() in MKW01_Communication.c
 *
 *  - NO_LF_FASTEST_TX_ONLY : this mode does not use the LF communication.
 *                          The module executes the following sequence continuously:
 *                           initialization -> sensors acquisition -> RF frame transmission in STOP1 -> initialization ...
 *                           In summary, a new sequence of sensor acquisition and RF transmission starts as soon as
 *                           the previous RF frame has been transmitted.
 *
 *  - NO_LF_PWU_WAKEUP : this mode does not use the LF communication.
 *                          The modules executes the following sequence continuously:
 *                          initialization -> sensor acquisition -> RF frame transmission in STOP1
 *                          -> wait for PWU interrupt in STOP1 -> initialization ...
 *                          In summary, a new sequence of sensor acquisition and RF transmission starts at each PWU interrupt.
 *                          PWU period is configurable.
 *
 */

#define MODE    LF_COMMUNICATION
//#define MODE  NO_LF_FASTEST_TX_ONLY
//#define MODE  NO_LF_PWU_TX    // Choose the PWU period just below
```

Below is a brief description of each mode. Each operating mode is described with more details in the following sections.

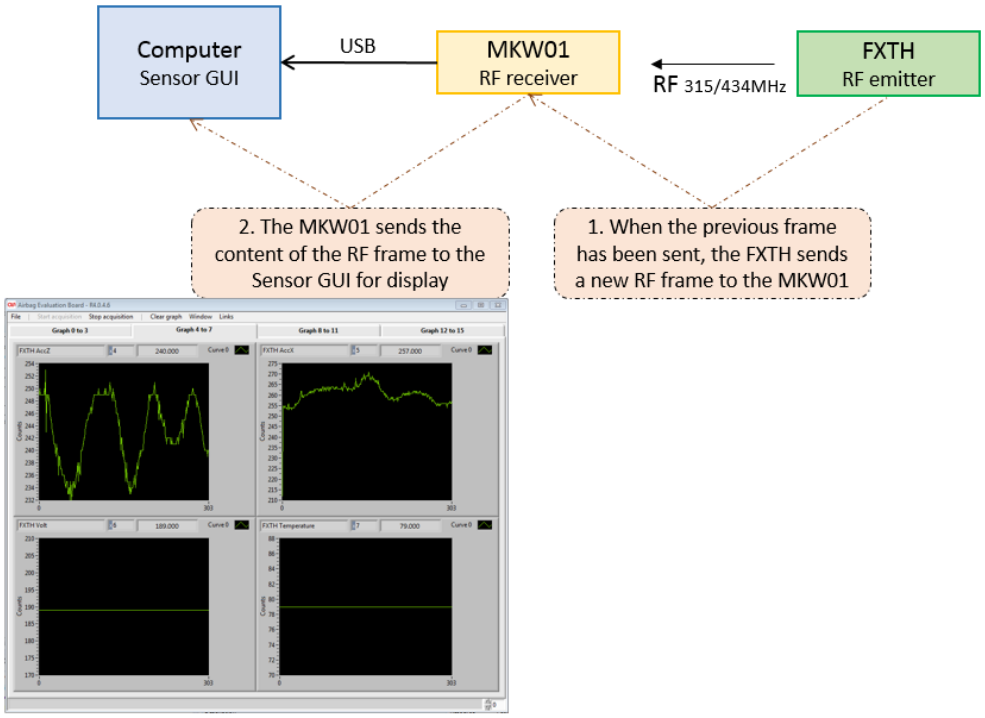
- **LF_COMMUNICATION**: this mode uses LF.
  The user can choose the command to be sent to the FXTH thanks to the Sensor GUI: the GUI transfers the command to the MKW01 and the MKW01 then sends the commands by LF to the FXTH.
  The FXTH87 is waiting for LF commands in STOP1 with the LF block enabled. When a valid LF command is received, it is decoded and executed.
  When the FXTH sends RF frames, the MKW01 receives them and transfers the data to the Sensor GUI for display.



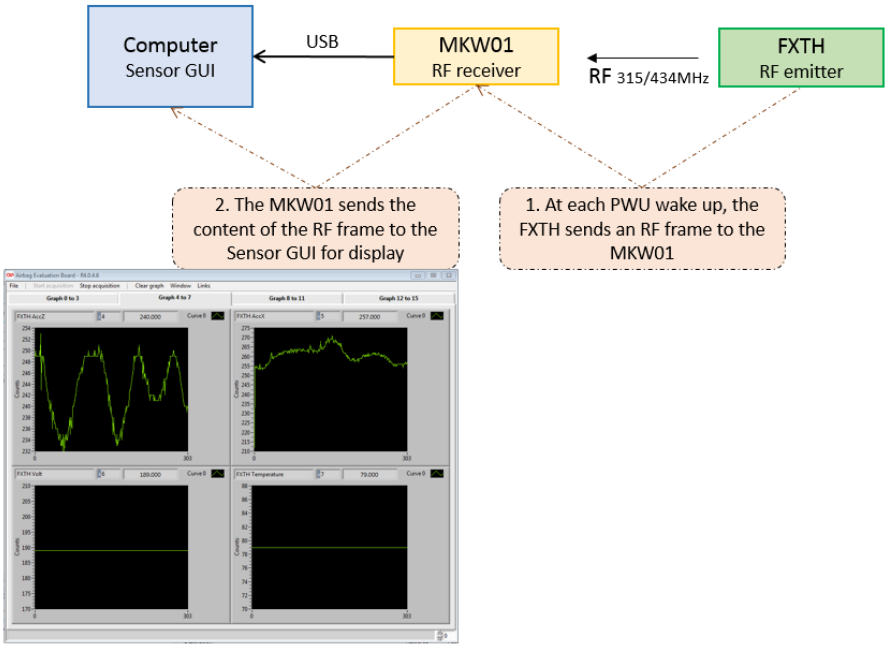




- **NO_LF_FASTEST_TX_ONLY**: this mode does not use LF. RF frames are sent by the FXTH to the MKW01 as fast as possible. The MKW01 transfers the RF data received to the GUI for display.



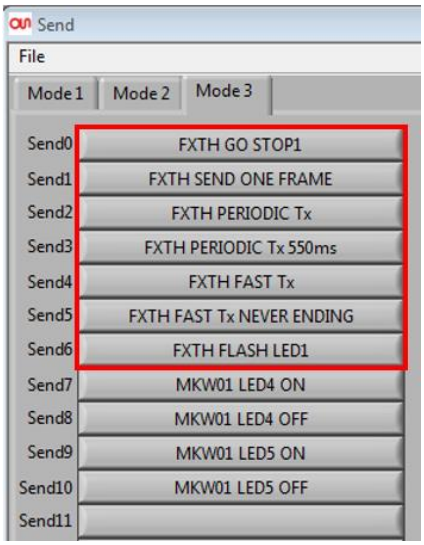
- **NO_LF_PWU_WAKEUP**: this mode does not use LF. An RF frame is sent by the FXTH to the MKW01 at each PWU wake up. The MKW01 transfers the RF data received to the GUI for display.



2. FXTH: LF COMMUNICATION mode

The user can choose the command to be sent to the FXTH thanks to the Sensor GUI: the GUI transfers the command to the MKW01 and the MKW01 then sends the commands by LF to the FXTH.
The FXTH87 is waiting for LF commands in STOP1 with the LF block enabled. When a valid LF command is received, it is decoded and executed.
When the FXTH sends RF frames, the MKW01 receives them and transfers the data to the Sensor GUI for display.

List of available commands on the Sensor GUI:

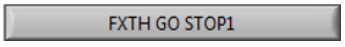


On the *Send* panel of the Sensor GUI, the user has the choice between several commands (refer to the *Demo Setup* section). The first seven commands (in the red square) are for the FXTH: the MKW01 will send these commands by LF to the FXTH. The LF frame format is described later on in the document.
The four remaining commands are for the MKW01 only (to control LED4 and LED5 of the MKW01). They will not be forwarded to the FXTH. Refer to section 6.f to see the mapping of LED4 and LED5 on the MKW01 board.

Each command from the GUI to the MKW01 is four-byte long. For some commands only three bytes are taken into account, but four bytes need to be sent nonetheless because the MKW01 is expecting four bytes.

Below is a description of each FXTH command.

- **Go to STOP1**: the FXTH87 enters STOP1 mode with LF block enabled to be able to receive further LF commands.

  Command: 0x5E 0x31 0x01 0xAA
  0x5E 0x31: FXTH87 LF ID (as programmed in the FXTH87 module)
  0x01: command to go to STOP1 with LF
  0xAA: not used at the moment

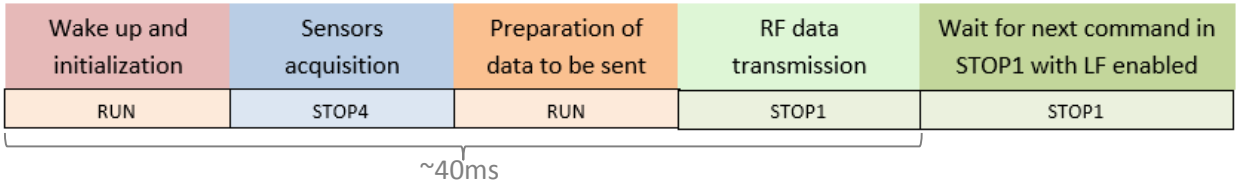
- **Send one frame**: when the FXTH87 receives the *Send one frame* command, it sends back a complete RF frame containing all the compensated measurements (P, X, Z, V, T).

Command: 0x5E 0x31 0x02 0xAA

0x5E 0x31: FXTH87 LF ID (as programmed in the FXTH87 module)
0x02: command to start all the measurements, do compensation and send all the values by RF.
0xAA: not used at the moment

Note: by default in the FXTH project, LED5 is ON during 30ms at the end of the RF transmission. This is not shown here. This can be disabled, refer to section 6.a.

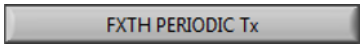
FXTH execution:



Each RF frame is received by the MKW01 which forwards the data to the Sensor GUI for display.

- **Periodic Tx**: when the FXTH87 receives the *Periodic Tx* command, the periodic wake up (PWU) is set to the period chosen by the user (fourth byte) and a complete RF frame is sent at each PWU wake up. The user chooses the wake up period between 1 second and 95 seconds (maximum period possible) and at each wake up, the FXTH87 takes all the measurements (P, X, Z, V, T), the compensations and sends all the values in an RF frame.

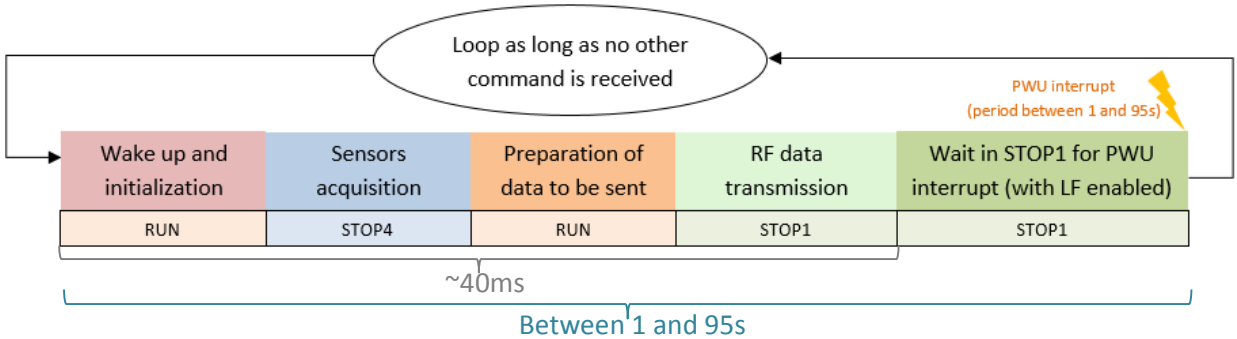
Command: 0x5E 0x31 0x03 0xXX



0x5E 0x31: FXTH87 LF ID (as programmed in the FXTH87 module)
0x03: command to set the PWU to period indicated in fourth byte and send a complete RF frame at each wake up.
0xXX: Wake up period in seconds chosen by the user. **Possible values: from 0x01 to 0x5F $(95)_{10}$** which allows a PWU wake up period between 1 second and 95 seconds.

Note: by default in the FXTH project, LED5 is ON during 30ms at the end of the RF transmission. This is not shown here. This can be disabled, refer to section 6.a.

FXTH execution:



Each RF frame is received by the MKW01 which forwards the data to the Sensor GUI for display.

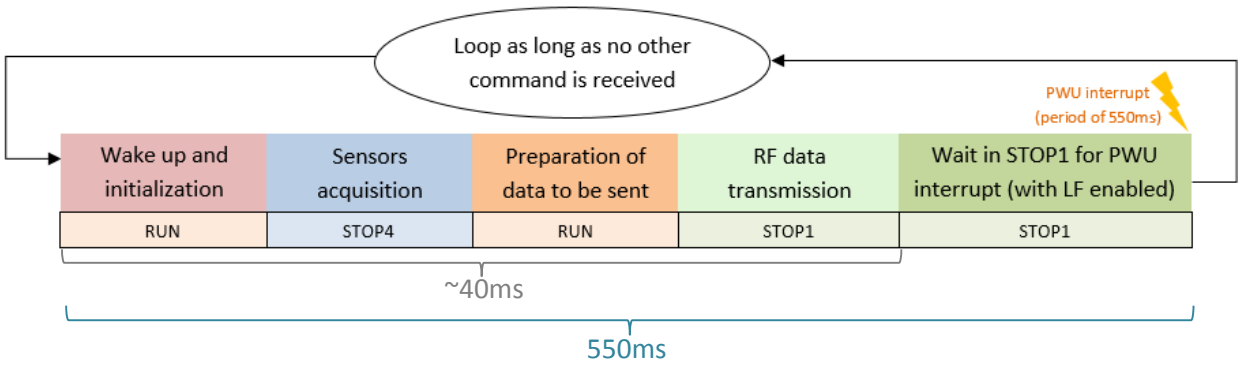
- **Periodic Tx 550ms**: when the FXTH87 receives the *Periodic Tx 550ms* command, the periodic wake up (PWU) is set to 550ms and a complete RF frame is sent at each wake up. So every 550ms, the FXTH87 takes all the measurements (P, X, Z, V, T), the compensations and sends all the values in an RF frame.

Command: 0x5E 0x31 0x04 0xAA

0x5E 0x31: FXTH87 LF ID (as programmed in the FXTH87 module)
0x04: command to set the PWU to 550ms and send a complete RF frame at each wake up.
0xAA: not used at the moment

FXTH execution:

Note: by default in the FXTH project, LED5 is ON during 30ms at the end of the RF transmission. This is not shown here. This can be disabled, refer to section 6.a.



Each RF frame is received by the MKW01 which forwards the data to the Sensor GUI for display.

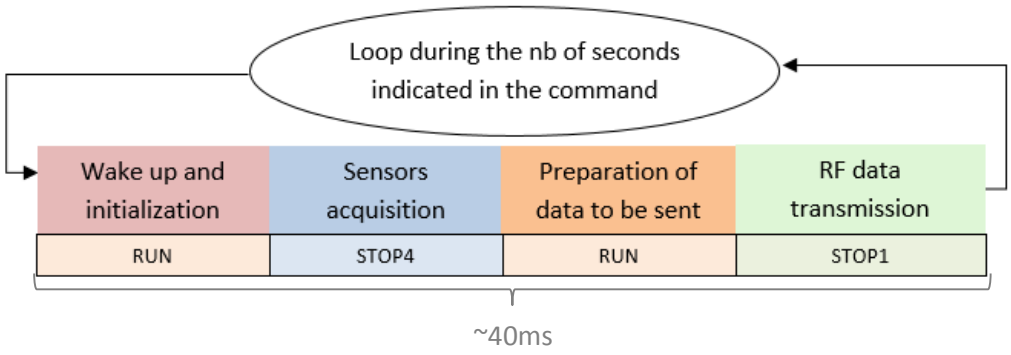
- **Fastest Tx**: when the FXTH87 receives the *Fastest Tx* command, full measurements + compensations (P, X, Z, V, T) + RF Tx are done continuously **during** the number of seconds indicated by the fourth byte. The time needed for full measurements + compensations + RF Tx is around 40ms, so in this mode a frame is sent every 40ms approx. during the number of seconds indicated in the fourth byte. During this time the LF block is disabled so it is not possible to communicate with the FXTH87 by LF. After this time the FXTH87 goes back to STOP1 with LF enabled.
  For example, if the fourth byte is 0x0A then frames will be sent every 40ms during 10 seconds (LF block disabled during this time), and after 10 seconds the FXTH87 will go back to STOP1 with LF enabled.
  Note: the value of 40ms mentioned above is in case the user disabled the use of LEDs in the FXTH project (refer to section 6.a). If LEDs are enabled, an additional delay of 30ms at the end of the RF transmission is to take into account (and in this case an RF frame is sent every 70ms).

  Command: 0x5E 0x31 0x05 0xXX 

  0x5E 0x31: FXTH87 LF ID (as programmed in the FXTH87 module)
  0x05: command to set the PWU to the period indicated in fourth byte and send a complete RF frame at each wake up.
  0xXX: Number of seconds chosen by the user during which measurements, compensation and RF Tx are done continuously. **Possible values: from 0x01 to 0x5F $(95)_{10}$** which allows to receive frames continuously during 1 second to 95 seconds.

  FXTH execution:

  Note: by default in the FXTH project, LED5 is ON during 30ms at the end of the RF transmission. This is not shown here. This can be disabled, refer to section 6.a.



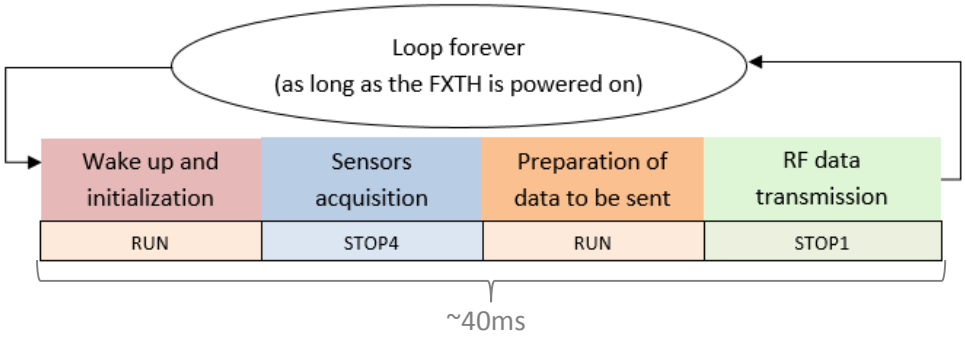
  Each RF frame is received by the MKW01 which forwards the data to the Sensor GUI for display.

- **Fastest Tx Never ending**: when the FXTH87 receives the *Fastest Tx Never ending* command, full measurements + compensations (P, X, Z, V, T) + RF Tx are done continuously **forever** (LF communication is stopped). The time needed for full measurements + compensations + RF Tx is around 40ms, so in this mode a frame is sent every 40ms approx. The LF block is disabled so it is not possible to communicate with the FXTH87 anymore. In order to re-start the LF communication, the FXTH needs to be powered off then powered on.
  Note: the value of 40ms mentioned above is in case the user disabled the use of LEDs in the FXTH project (refer to section 6.a). If LEDs are enabled, an additional delay of 30ms at the end of the RF transmission is to take into account (and in this case an RF frame is sent every 70ms).

  Command format: 0x5E 0x31 0x06 0xAA FXTH FAST Tx NEVER ENDING
  0x5E 0x31: FXTH87 LF ID (as programmed in the FXTH87 module)
  0x05: command to set the PWU to period indicated in fourth byte and send a complete RF frame at each wake up.
  0xAA: not used at the moment

  FXTH execution:

  Note: by default in the FXTH project, LED5 is ON during 30ms at the end of the RF transmission. This is not shown here. This can be disabled, refer to section 6.a.



  Each RF frame is received by the MKW01 which forwards the data to the Sensor GUI for display.

- **Flash LED1**: when the FXTH87 receives the *Flash LED1* command, LED1 (PTA0) will be ON during 100ms, then will be OFF again. Then the FXTH will go back to STOP1 with LF enabled to wait for further commands.
  If the use of LEDs has been disabled in *user_configuration.h* (refer to section 6.a), the FXTH will directly go back to STOP1 with LF enabled (the GPIO will not be turned on).

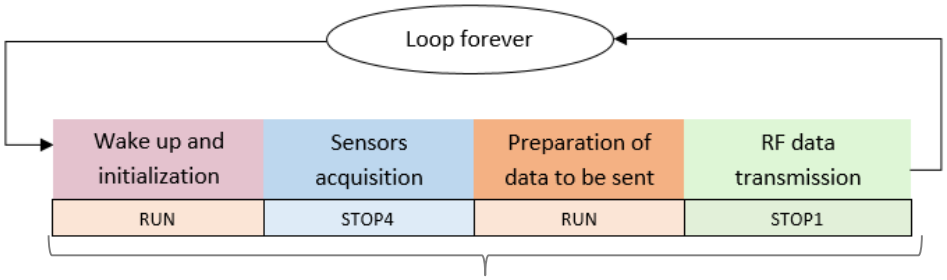
  Command format: 0x5E 0x31 0x0F 0xAA FXTH FLASH LED1
  0x5E 0x31: FXTH87 LF ID (as programmed in the FXTH87 module)
  0x0F: command to turn ON LED1 (PTA0) during 100ms, if the use of LEDs has been enabled.
  0xAA: not used at the moment

3. FXTH: NO LF FASTEST TX ONLY mode

This mode does not use LF. The LF block is always disabled on the FXTH side so the FXTH cannot receive LF commands.

The execution of the FXTH is the following:

Note: by default in the FXTH project, LED5 is ON during 30ms at the end of the RF transmission. This is not shown here. This can be disabled, refer to section 6.a.



The FXTH wakes up from STOP1, takes all sensors measurements and compensation, prepares the RF frame and initializes the RF. Then it triggers the RF transmission and enters STOP1 to wait for the end of transmission. When the transmission is finished, the RF block generates an *end-of-transmission* interrupt which wakes up the FXTH and the entire sequence starts again.

The time taken by the entire sequence is around 40ms so it means an RF frame is sent every 40ms.

> *Note:* the value of 40ms mentioned above is in case the user disabled the use of LEDs in the FXTH project (refer to section 6.a). If LEDs are enabled, an additional delay of 30ms at the end of the RF transmission is to take into account (and in this case an RF frame is sent every 70ms).

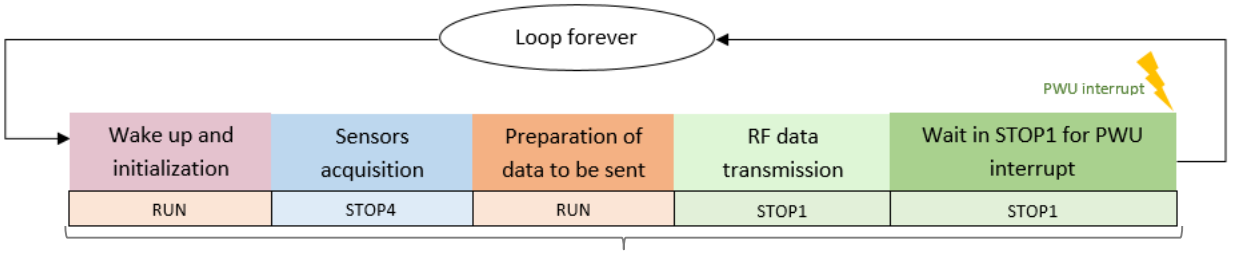
The MKW01 receives the RF frames and forwards the data to the Sensor GUI for display.

4. FXTH: NO LF PWU WAKEUP mode

This mode does not use LF. The LF block is always disabled on the FXTH side so the FXTH cannot receive LF commands.

The execution of the FXTH is the following:

Note: by default in the FXTH project, LED5 is ON during 30ms at the end of the RF transmission. This is not shown here. This can be disabled, refer to section 6.a.



Between 504ms and 95s

The FXTH wakes up from STOP1, takes all sensors measurements and compensation, prepares the RF frame and initializes the RF. Then it triggers the RF transmission and enters STOP1 the time of the RF transmission and then stay in STOP1 to wait for the next PWU interrupt (in this mode, there is no interrupt generated by the RF block at the end of the transmission). When the PWU wake up interrupt occurs, this wakes up the FXTH and the entire sequence starts again.

The PWU can be configured to generate a wake up interrupt with a period between 504ms and 95s.
In the FXTH project, in *user_configuration.h,* the PWU period can be configured between 1 and 95s with the #define PWU_PERIOD. In the example below, the PWU period is set to two seconds i.e. the FXTH will wake up and send an RF frame every two seconds.
If the user wants a PWU period between 504ms and 1s, this cannot be done in *user_configuration.h,* it has to be configured in the function *vfnNoLfConfigureStartPeriodicTx* in *MKW01_Communication.c*

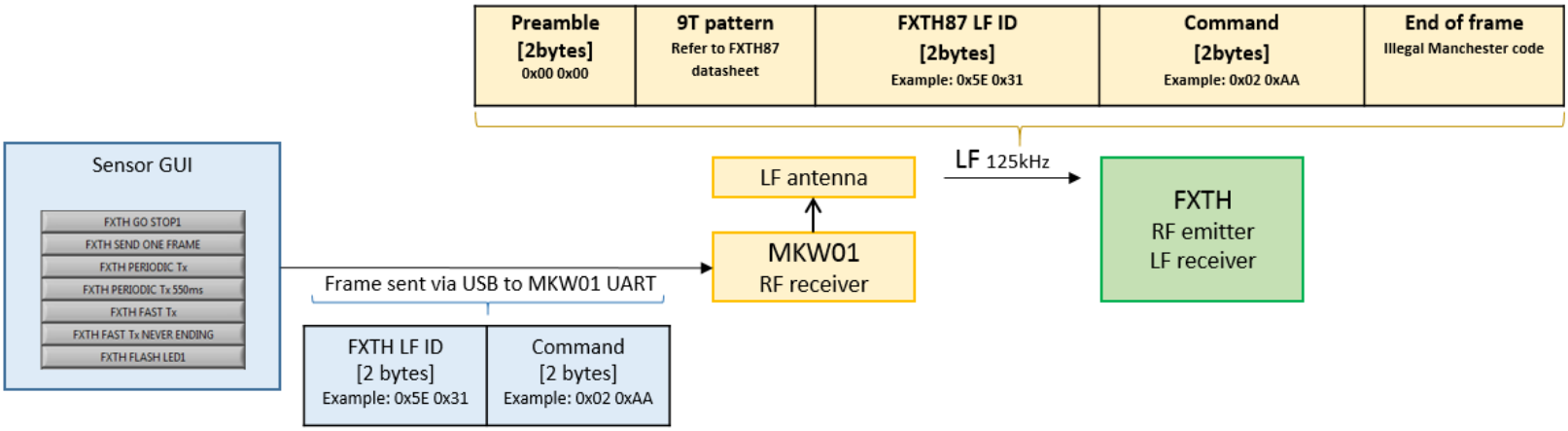
```
//#define MODE  LF_COMMUNICATION
//#define MODE  NO_LF_FASTEST_TX_ONLY
#define MODE    NO_LF_PWU_TX    // Choose the PWU period just below


#if (MODE == NO_LF_PWU_TX)
// PWU period defined must be between 1 and 95 seconds
// In order to set a period between 504ms and 1 second,
// configure it manually in the function vfnNoLfConfigureStartPeriodicTx in MKW01_communication.c

#define PWU_PERIOD      2   // in seconds
```

The MKW01 receives the RF frames and forwards the data to the Sensor GUI for display.

5. Frame formats

a. LF frame format (MKW01 to FXTH)

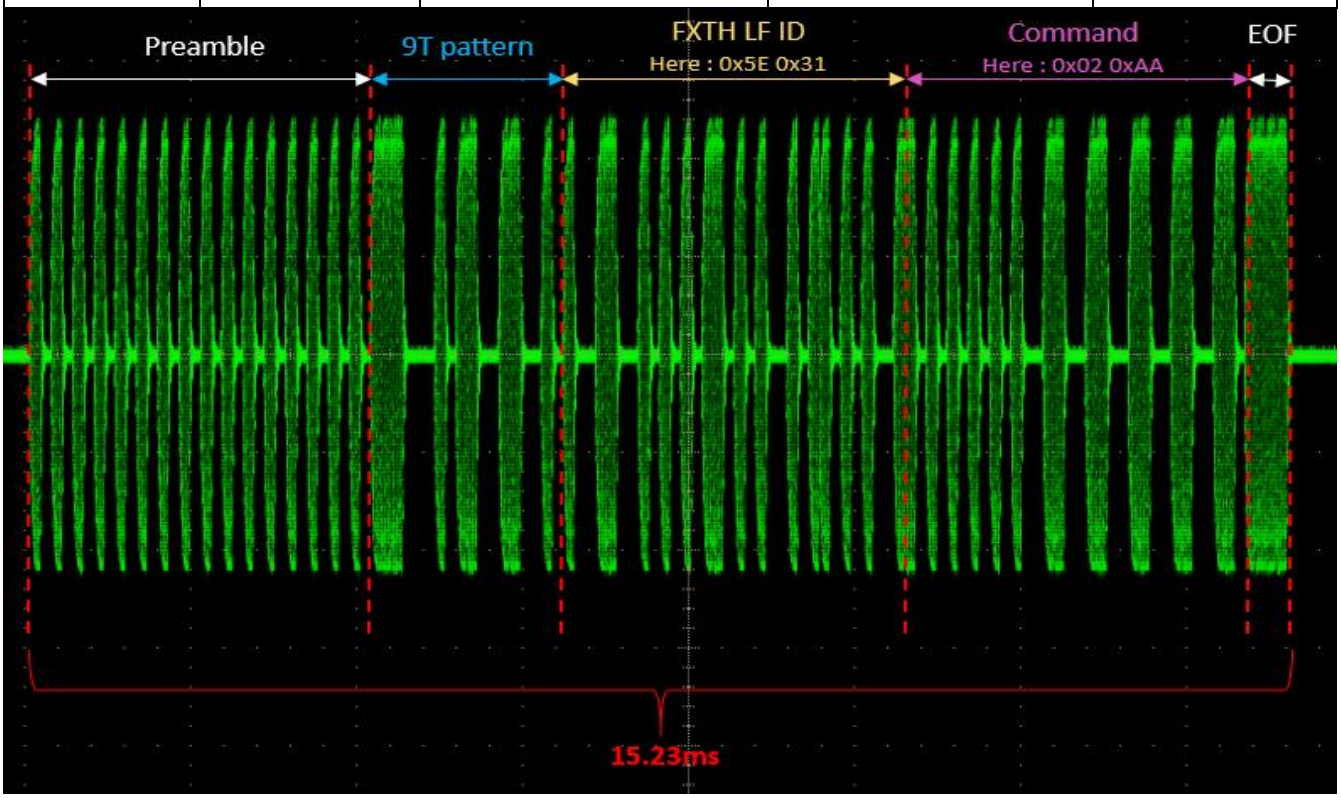
Overview of the command transfer from Sensor GUI to FXTH:



LF frame format (from MKW01 to FXTH):

Data is **Manchester-encoded** and sent using **OOK modulation** at a baud rate of **3906 b/s**.

Preamble [2bytes] 0x00 0x00	9T pattern Refer to FXTH87 datasheet	FXTH87 LF ID [2bytes] Example: 0x5E 0x31	Command [2bytes] Example: 0x02 0xAA	End of frame Illegal Manchester code



b. RF frame format (FXTH to MKW01)

The FXTH sends the following frame to the MKW01:

```
au8RFDataForCS[0u] = (UINT8) (0x55);              // Preamble
au8RFDataForCS[1u] = (UINT8) (0x55);              // Preamble
au8RFDataForCS[2u] = (UINT8) (0x55);              // Preamble
au8RFDataForCS[3u] = (UINT8) (0x01);              // Sync word
au8RFDataForCS[4u] = (UINT8) (0x01);              // Sync word
au8RFDataForCS[5u] = (UINT8) (0x01);              // Sync word
au8RFDataForCS[6u] = (UINT8) (0x01);              // Sync word
au8RFDataForCS[7u] = (UINT8) (Payload_Length);    // Payload length, does not include CRC
au8RFDataForCS[8u] = (UINT8) (0xF0);              // MKW01 receiver address. Address check disabled
au8RFDataForCS[9u] = (UINT8) (Tire_ID >> 24u);    // Tire ID
au8RFDataForCS[10u] = (UINT8) (Tire_ID >> 16u);
au8RFDataForCS[11u] = (UINT8) (Tire_ID >> 8u);
au8RFDataForCS[12u] = (UINT8) (Tire_ID);
au8RFDataForCS[13u] = (UINT8) (Firmware_Version);         // Firmware Version
au8RFDataForCS[14u] = (UINT8) (Derivative_Descriptor);    // Derivative Descriptor
au8RFDataForCS[15u] = (UINT8) (u16CompPressure >> 8u);    // Pressure
au8RFDataForCS[16u] = (UINT8) (u16CompPressure);
au8RFDataForCS[17u] = (UINT8) (u16CompAccelZ >> 8u);      // Z-axis acceleration
au8RFDataForCS[18u] = (UINT8) (u16CompAccelZ);
au8RFDataForCS[19u] = (UINT8) (u16CompAccelX >> 8u);      // X-axis acceleration
au8RFDataForCS[20u] = (UINT8) (u16CompAccelX);
au8RFDataForCS[21u] = (UINT8) (gu8CompVolt);      // Voltage
au8RFDataForCS[22u] = (UINT8) (gu8CompTemp);      // Temperature
au8RFDataForCS[23u] = (UINT8) (u8StatusAcqRead); // Status Acquisition for READ functions
au8RFDataForCS[24u] = (UINT8) (u8StatusAcqComp); // Status Acquisition for COMP functions
au8RFDataForCS[25u] = (UINT8) (FrameID >> 8);     // Frame ID: keep alive counter
au8RFDataForCS[26u] = (UINT8) (FrameID);
au8RFDataForCS[27u] = (UINT8) (0xC0);             // Fixed data => can be modified by the user
au8RFDataForCS[28u] = (UINT8) (0xC1);             // Fixed data => can be modified by the user
au8RFDataForCS[29u] = (UINT8) (0xC2);             // Fixed data => can be modified by the user
au8RFDataForCS[30u] = (UINT8) (Verification_Value >> 8); // CRC
au8RFDataForCS[31u] = (UINT8) (Verification_Value);
```

- *MKW01 receiver address* is not currently used in the demo.
- *TireID* is the 4-byte unique ID.
- *Derivative Descriptor* is *CODE1* mentioned in the FXTH datasheet.
- The 2-byte CRC included at the end is calculated from *Payload Length* to *0xC2* (included).

The frame is 32-byte long, this is the maximum size of the FXTH frame.

Data is **NRZ-encoded** and sent using **FSK modulation** at a baud rate of **19200 b/s**.

By default in the demo project the RF frequency used is 434MHz. This, as well as all the other RF settings, can be modified. Refer to the section on RF settings.

c. UART frame format (GUI to MKW01 and MKW01 to GUI)

Baud rate used for communication with the Sensor GUI is **115200 b/s**. This is configurable in the demo, refer to UART settings section.

From Sensor GUI to MKW01:

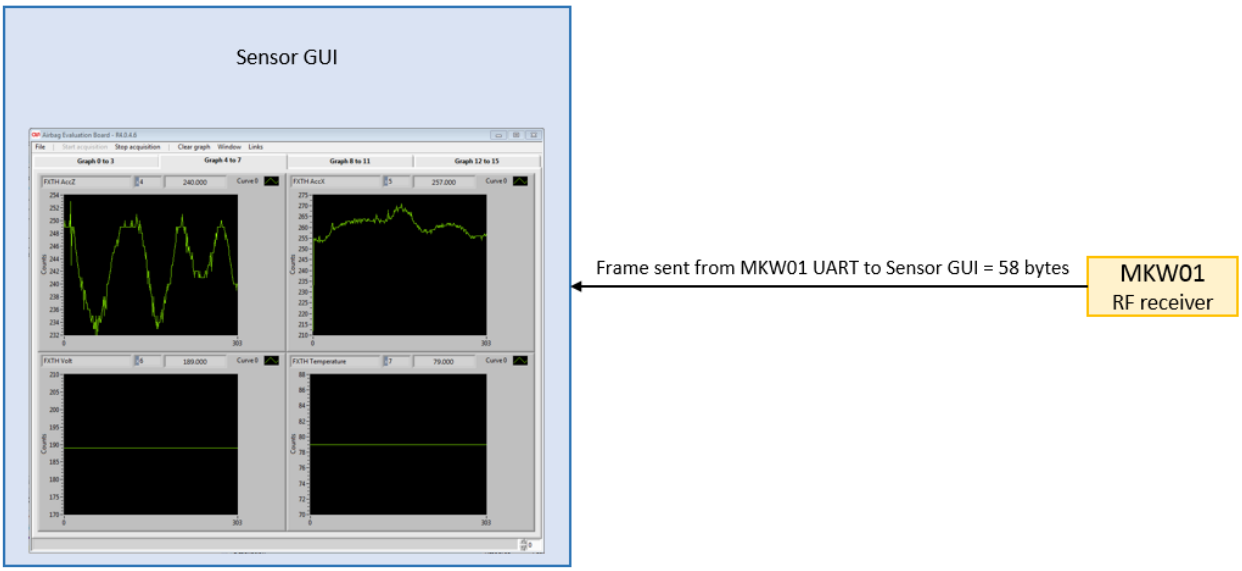
All commands are 4-byte long.

The first seven commands (for FXTH) have been described in the sections above. Refer to section 2.

The last four commands (for MKW01 only) are the following:

- **MKW01 LED4 ON** : 0xE0 0x0E 0x04 0x01
  Turns ON LED4 (PTA4).
- **MKW01 LED4 OFF** : 0xE0 0x0E 0x04 0x10
  Turns OFF LED4 (PTA4).
- **MKW01 LED5 ON** : 0xE0 0x0E 0x05 0x01
  Turns ON LED5 (PTE0).
- **MKW01 LED5 OFF** : 0xE0 0x0E 0x05 0x10
  Turns OFF LED5 (PTE0).

From MKW01 to Sensor GUI:



The frame is the following:

```
tmpbuf[0]  = '[';
tmpbuf[1]  = 'D';
tmpbuf[2]  = 'A';
tmpbuf[3]  = '0';
tmpbuf[4]  = '8';
tmpbuf[5]  = '0';
tmpbuf[6]  = '0';
tmpbuf[7]  = '0';
tmpbuf[8]  = '1';
tmpbuf[9]  = '1';
tmpbuf[10] = '2';
tmpbuf[11] = '3';
tmpbuf[12] = '4';
```

Required by the Sensor GUI

```
index = 13;
for (i=2;i<=payloadSize-3;i++) // Do not display constant bytes at the end
 {
     tmpbuf[index++] = hex_word[(RF_Rx_Buffer[i] >> 4) &0x0f];
     tmpbuf[index++] = hex_word[(RF_Rx_Buffer[i] & 0x0f)];
 }
```

Data received in the RF frame, from *TireID* to *FrameID*

```
tmpbuf[index++] = hex_word[((uint8_t)(counter>>8) >> 4)&0x0f]; // MKW01 RF counter
tmpbuf[index++] = hex_word[((uint8_t)(counter>>8) & 0x0f)];
tmpbuf[index++] = hex_word[((uint8_t)(counter&0x00FF) >> 4)&0x0f];
tmpbuf[index++] = hex_word[((uint8_t)(counter&0x00FF) & 0x0f)];
tmpbuf[index++] = hex_word[((uint8_t)(RF_CrcNotOk) >> 4)&0x0f]; // 0 if CRC OK, 1 if not OK
tmpbuf[index++] = hex_word[((uint8_t)(RF_CrcNotOk) & 0x0f)];
tmpbuf[index++] = hex_word[((uint8_t)(RF_RSSI_value) >> 4)&0x0f]; // RSSI value
tmpbuf[index++] = hex_word[((uint8_t)(RF_RSSI_value) & 0x0f)];
tmpbuf[index++] = ']';
```

2-byte MKW01 RF counter: increments at each RF frame received

1-byte CRC check: 0 if CRC OK (RF frame not corrupted), 1 if CRC check failed (RF frame corrupted)

1-byte MKW01 RSSI: RSSI value when the RF frame was received

Required by the Sensor GUI

The data received in the RF frame needs to be formatted before being sent to the GUI: each data byte is sent as two hexa-bytes. For example, 0x0A will be sent as ‘0’ and ‘A’.

6. Demo parameters and settings

a. FXTH side: GPIO/LEDs configuration

By default in the FXTH project, LEDs are used to follow the execution flow or for the LF command. LEDs can be enabled or disabled. On the user's FXTH board, if the GPIOs are physically connected to GND or VDD and cannot toggle, they need to be disabled. Indicate it with the #define LEDS in *user_configuration.h*
When LEDs are not used, all instructions related to LEDs (GPIO ON or OFF) are replaced by __asm nop;
When LEDs are used, the mapping is described below:

```
/*******************************************************************************
 * LEDs (GPIOs) can be used in the project.
 * Choose below if you want to use them or not. For example, do not use them
 * if the GPIOs are physically connected to GND or VDD on your board.
 * If DO_NOT_USE_LEDS is selected, all instructions related to LEDs will be replaced
 * by __asm nop;
 *
 * In this project:
 * LED1 : PTA0
 * LED2 : PTA1
 * LED3 : PTA2
 * LED4 : PTA3
 * LED5 : PTB0
 * LED6 : PTB1
 */

#define LEDS    USE_LEDS
//#define LEDS  DO_NOT_USE_LEDS
```

In the project, the LEDs are used in the following way:

- **LED1 (PTA0)**: it is turned ON during 100ms with the LF command 0x5E 0x31 0x0F 0xAA.
- **LED2 (PTA1)**: it is turned ON during 100ms when an unknown LF command is received.
- **LED3 (PTA2)**: it is turned ON when a LF command is received. It is turned off just before going to STOP1.

b. FXTH side: TireID

By default in the FXTH project, the 4-byte TireID is the unique ID of the device. However, it is possible for the user to choose the value of the 4-byte TireID. This is done in *user_configuration.h:*

```
/*******************************************************************************
 *  Below is to configure the tire ID sent in the RF frame.
 *  CHOSEN_BY_USER means that tire ID will be one of the ID pre-determined by the user,
 *  and UNIQUE_ID means that it is the unique 32-bit ID of the product that will be used
 *  as tire ID.
 *  In case of configurable IDs (not unique IDs), FIXED means that the same ID will be used
 *  for every transmission and CYCLING means that the ID will change at each transmission (this
 *  is to simulate four wheels).
 */


//#define TIRE_ID_VALUE CHOSEN_BY_USER  // One of the four IDs below
#define TIRE_ID_VALUE   UNIQUE_ID       // The unique ID of the product


#if (TIRE_ID_VALUE == CHOSEN_BY_USER)

#define ID1 0xAA01AA01
#define ID2 0xBB02BB02
#define ID3 0xCC03CC03
#define ID4 0xDD04DD04

#define METHOD  FIXED_ID        // Choose the ID nb in the function vfnUpdateTireID in main.c
//#define METHOD    CYCLING_ID      // ID changes each time a new frame is sent
```

If the user selects #define TIRE_ID_VALUE CHOSEN_BY_USER then the TireID will be one of the four IDs pre-defined (ID1, ID2, ID3 or ID4).
In this case, the user has the choice between FIXED_ID and CYCLING_ID:

- FIXED_ID: the TireID will be one of the four defined IDs and will always be the same (it will not change). To choose which one of the four pre-defined IDs will be used, go to the function *vfnUpdateTireID* in main.c :

```
#if (METHOD == FIXED_ID)
    /* Uncomment one of the four IDs */
    Tire_ID = ID1;
    //Tire_ID = ID2;
    //Tire_ID = ID3;
    //Tire_ID = ID4;
```

- CYCLING_ID: the ID will change at each RF transmission. It will first be ID1, then ID2, then ID3 then ID4 then ID1… This is to simulate four wheels.

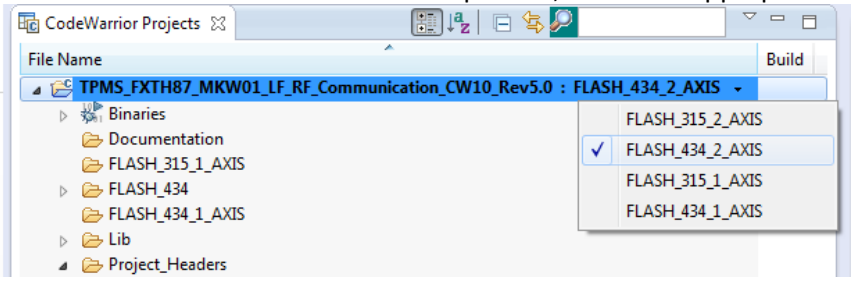
c. FXTH side: LF settings

The LF configuration (LF ID…) is done in the function *vfnDataModeInit* in the file *szk_lf_data_detect.c*

d. FXTH and MKW01: RF settings

FXTH side:
The RF configuration (baud rate, modulation…) is done in the function *Init_RF* in *main.c*
In the demo, data is **NRZ-encoded** and sent using **FSK modulation** at a baud rate of **19200 b/s**.

The choice between 315MHz or 434MHz is done at compilation, select the appropriate frequency:



MKW01 side:
The RF configuration is done in the function *RF_InitRx* in *RF_Rx_Appli.c*

```
/* Choose below between one of the defined frequencies:
 * FREQ_3135 (313.5MHz), FREQ_314 (314MHz), FREQ_3145, FREQ_315, FREQ_3155, FREQ_316, FREQ_3165,
 * FREQ_4325, FREQ_433, FREQ_4335, FREQ_434, FREQ_4345, FREQ_435, FREQ_4355 */
MKW01Drv_SetFrequency(FREQ_434);

/* Choose between one of the defined baud rates:
 * BR_4800, BR_9600, BR_10000, BR_19200, BR_20000, BR_40000, BR_50000, BR_100000, BR_150000, BR_200000 */
MKW01Drv_SetBaudRate(BR_19200);

/* Choose between one of the defined frequency deviations:
 * FREQ_DEV_5000, FREQ_DEV_19000, FREQ_DEV_25000, FREQ_DEV_50000, FREQ_DEV_50049, FREQ_DEV_100000 */
MKW01Drv_SetFreqDev(FREQ_DEV_50000);
```

Several settings have been implemented for the frequency, baud rate and frequency deviation. The user only needs to change the parameter of the functions.
For the other settings related to RF (like modulation) or related to the RF frame format (length of preamble, number of synchronization words…), the configuration has to be done directly in the MKW01 RF registers, in the file *TransceiverDrv.c*

e. MKW01 side: UART settings

In the MKW01 project, the UART configuration is done in the function *Init_UART0_General* in *UART.c*
Several baud rates have been predefined and can be selected with the function *UART0_SetBaudRate:* the user can choose one of the available baud rates:

```
/* Available baud rates: UART_BAUD_RATE_9600, UART_BAUD_RATE_19200, UART_BAUD_RATE_115200 */
UART0_SetBaudRate(UART_BAUD_RATE_115200);
```

By default, the demo project uses a baud rate of 115200 b/s.

f. MKW01 side: GPIO configuration

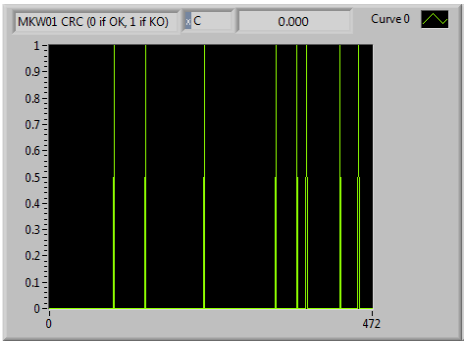
The following GPIOs are used in the MKW01 demo project:

- **PTA4**: used as LED4 (in *ExecuteAction.c*)
- **PTB0**: used to generate LF signal (*LF.c*)
- **PTB1**: used to generate LF signal (*LF.c*)
- **PTC2**: used as LED16, LED16 is ON while an RF frame is received and displayed (*main.c*)
- **PTC3**: not used but physically connected to DIO0 on MRB-KW019032 board
- **PTC4**: not used but physically connected to DIO1 on MRB-KW019032 board
- **PTE0**: used as LED5 (*ExecuteAction.c*)
- **PTE1**: used as LED10, LED10 in ON while an LF frame is being sent (*ExecuteAction.c*)
- **PTE2**: connected to DIO0 for RF (*RF_Rx_Appli.c*)
- **PTE3**: connected to DIO1 for RF (*RF_Rx_Appli.c*)
- **PTE16**: used as LED15, LED15 is ON during 200ms when an unknown UART command is received (*ExecuteAction.c*)

g. MKW01 side: display (or not) of corrupted RF data

When the RF frame received has been corrupted the CRC check fails. By default all frames are received and displayed. In the GUI, when the frame is corrupted, the panel MKW01 CRC will display 1. In order to discard the corrupted frame (no display of the corrupted data), select DISCARD_DATA in *RF_Rx_Appli.h*

```
/*****************************************************************************
 *                          User configuration
 *
 * Choose here what to do when the RF data received is corrupted (CRC not ok):
 * RECEIVE_DATA: the corrupted frame is received and displayed and the variable
 * RF_CrcNotOk is updated to indicate a corrupted frame (this variable is displayed on GUI).
 * DISCARD_DATA: the corrupted frame is discarded (only correct data will be received and displayed)
 *
 *****************************************************************************/
#define RF_DATA_CORRUPTED       RECEIVE_DATA
//#define RF_DATA_CORRUPTED     DISCARD_DATA
```



7. Demo setup

a. Setup procedure

Concerning software:

- The FXTH demo project has been developed under Code Warrior 10.6 (compatible with CW versions above).
  Refer to the pdf document *FXTH87_CW10.x_SetUp* for installation and programming procedure, located in the FXTH webpage, *Documentation* panel (NXP > Sensors > Pressure Sensors > Tire Pressure Monitoring Sensors > Documentation).

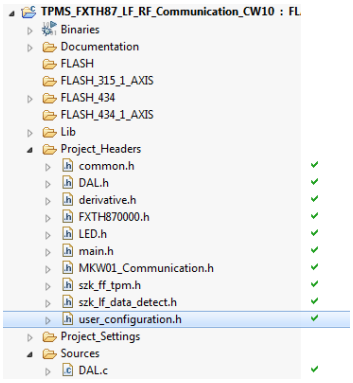
- The MKW01 demo project has been developed under Kinetis Design Studio (KDS) v3.x. Installation procedure and programming are described in section 8 of this document.

Concerning hardware:

- FXTH: if using the TPMS870000-SKT board, refer to TPMS870000-SKT_HW_SetUp.ppt located in the documentation folder of the FXTH project.
- MKW01: if using the FRDM-SENSORS-LF, the MRB-KW019032 or the FRDM-KW019032, refer to MKW01_FRDM-SENSORS-LF_HW_SetUp.ppt located in the documentation folder of the MKW01 project.

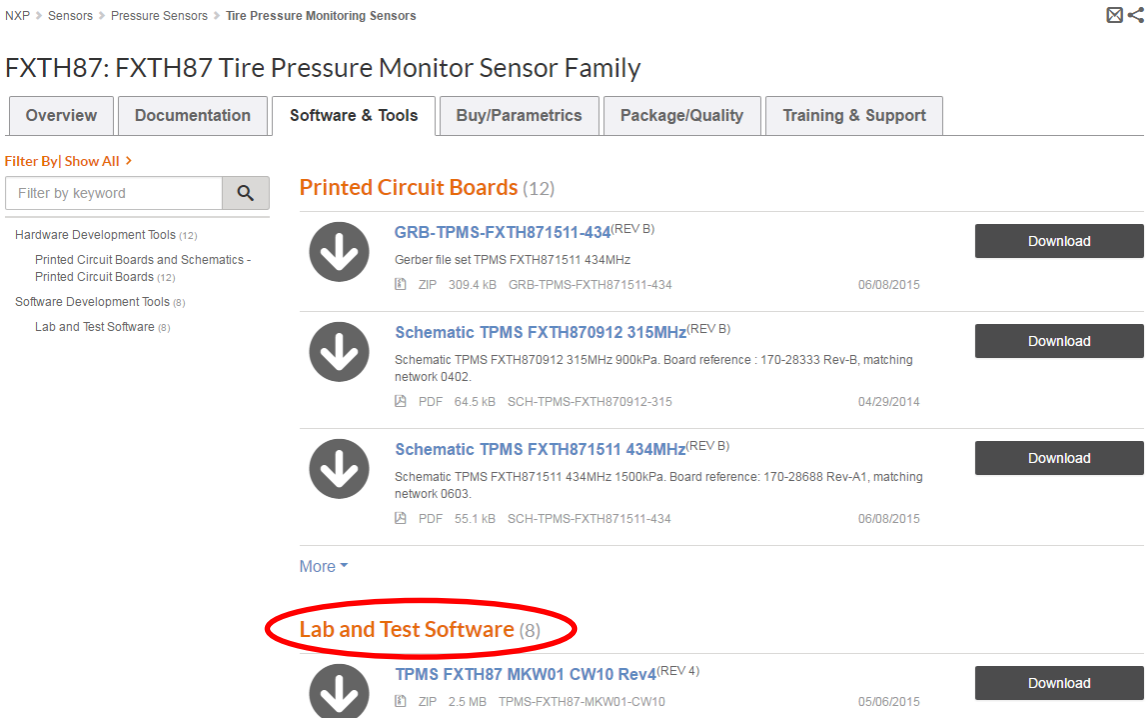
Demo setup procedure:

1. In the FXTH demo project, go to *user_configuration.h* and modify the demo parameters if necessary (refer to section 6). Select the RF frequency at compilation (refer to section 6.d). By default the demo uses 434MHz.
   The module needs to be supplied with 3.3V.
   Program the FXTH, remove the BDM and then do a hardware reset of the module (power off then power on). If the hardware reset is not done after programming, the module may not start correctly. (Refer to the documents cited above for information on programming and how to connect the BDM to the board).



2. Program the MKW01. The first time the MKW01 is connected to the computer the driver will be installed, wait until the installation is complete. Then disconnect the MKW01 from the computer.

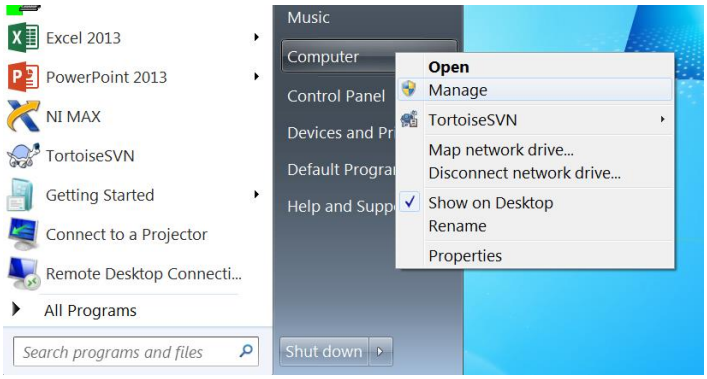
3. Install the updated version of the LabVIEW Sensor GUI. For that, go to the FXTH87 Software&Tools webpage (NXP > Sensors > Pressure Sensors > Tire Pressure Monitoring Sensors > FXTH87 > Software & Tools) and download the ***NXP Sensor GUI R4.0.3.5*** under *Lab and Test Software:*



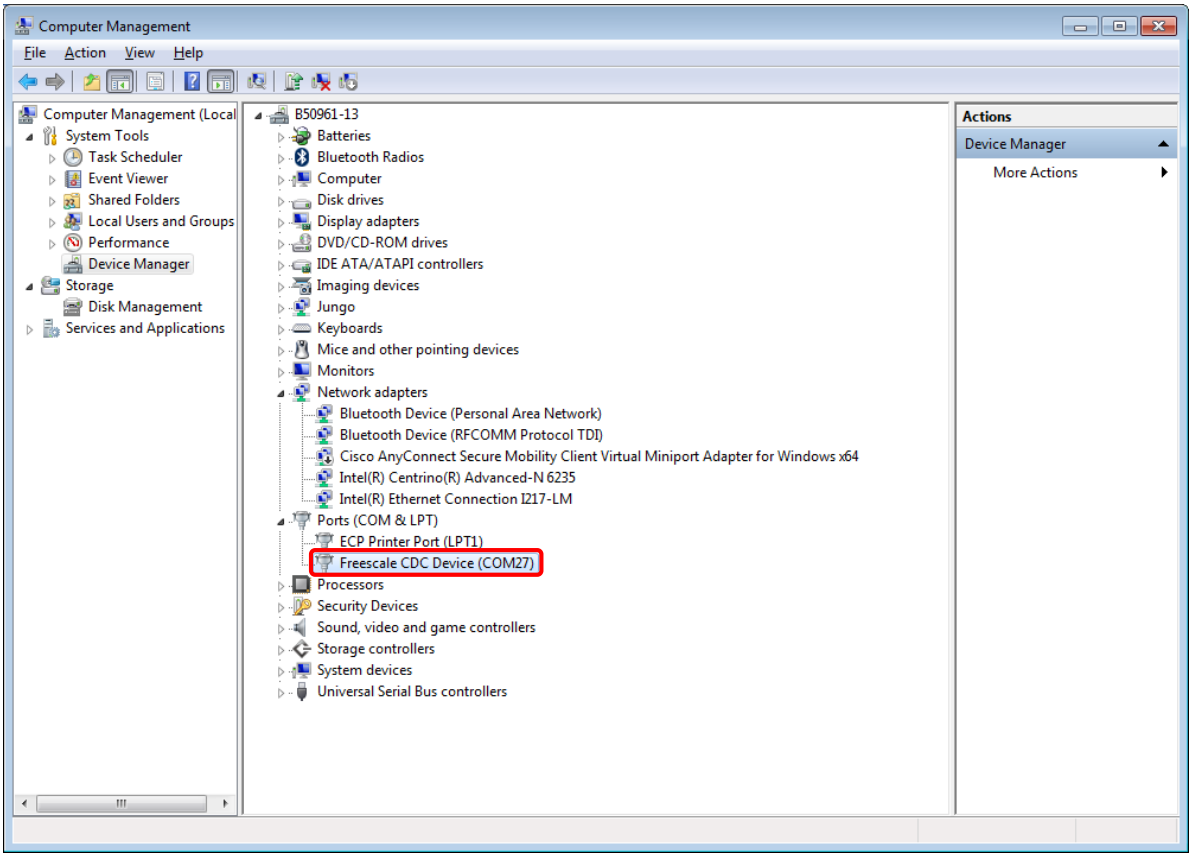
Follow the installation procedure explained in *README_Installation.txt* (*Installation procedure* only, not *Demo setup procedure* as it is explained below).

4. Connect the MKW01 to the computer via the USB connection.

5. Check the port COM number associated to this connection. For that, right click on *Computer* then *Manage:*

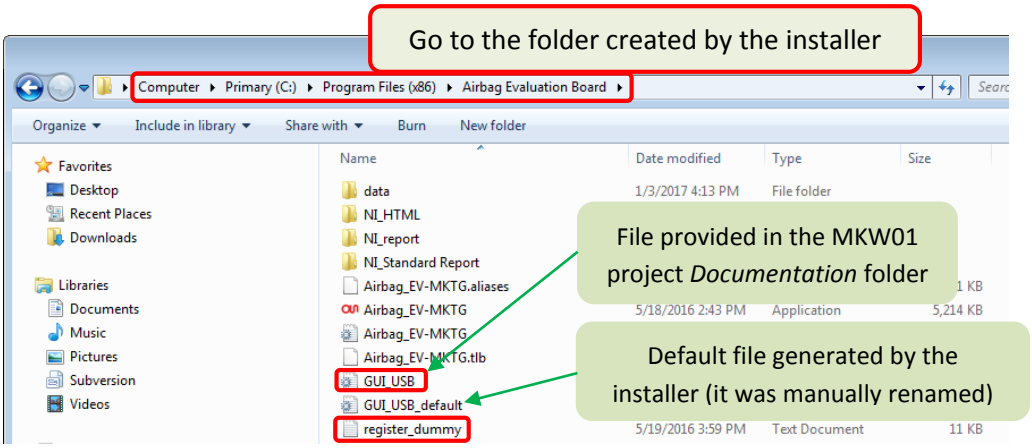


Go to *Device Manager* then check the COM number under *Ports:*

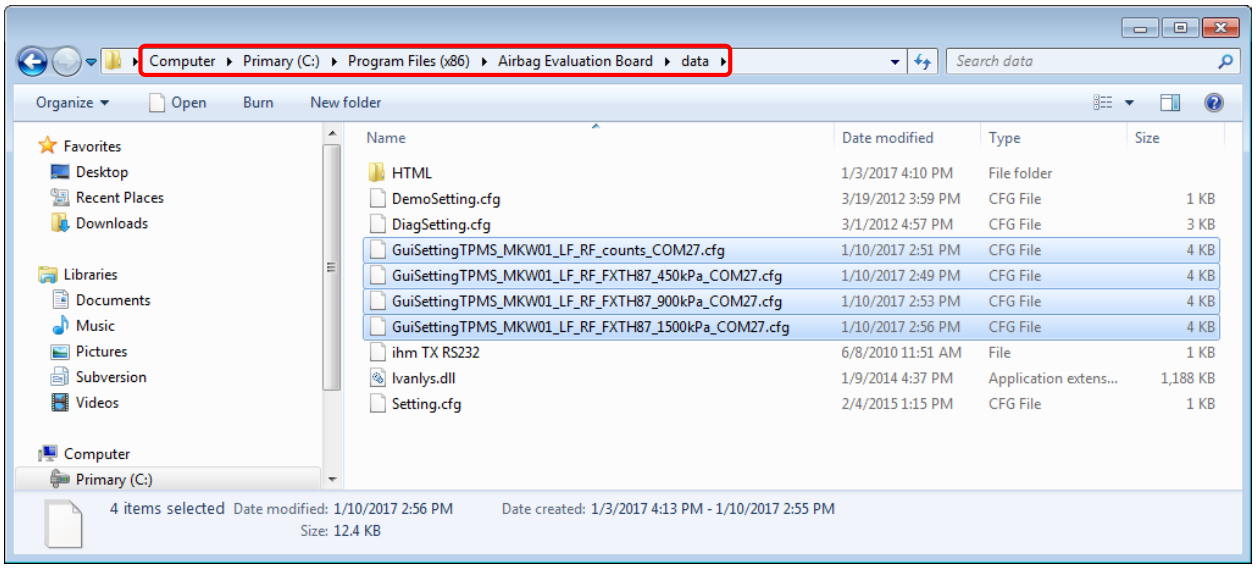


If the COM port does not show up, check the driver install and try to disconnect/reconnect the USB cable.

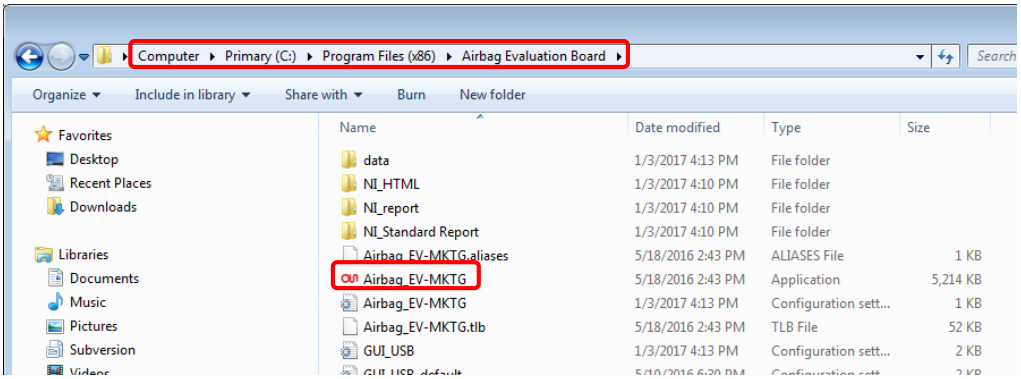
6. Into the folder created by the installer (*GUI_INSTALL_PATH)/Airbag Evaluation Board,* do two things:
   - Replace the existing *GUI_USB.cfg* file generated by the installer (delete it or rename it) by the *GUI_USB.cfg* provided in the *Documentation* folder of the MKW01 project.
   - Copy the file *register_dummy.txt* provided in the *Documentation* folder of the MKW01 project (this file is not used in this demo but will remove an error message):



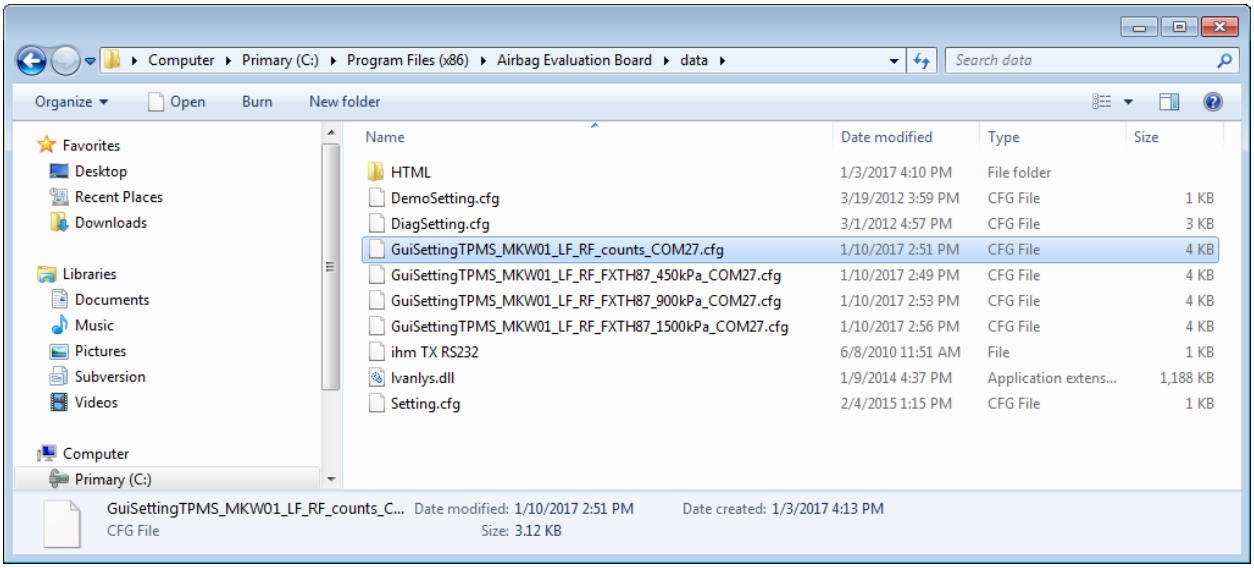
7. Then go the *data* folder (*GUI_INSTALL_PATH)/Airbag Evaluation Board/data*, and copy the four LabVIEW config file *GuiSettingTPMS_MKW01_LF_RF_FXTH87_xxx_COM27.cfg* provided in the MKW01 project (*Documentation* folder). Explanation on these files is given at the end of the section.



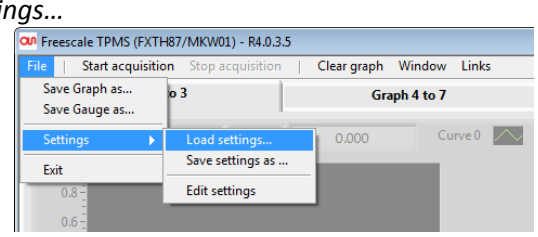
8. Open the LabVIEW GUI with the *Airbag_EV-MKTG.exe* in (*GUI_INSTALL_PATH)/Airbag Evaluation Board.*



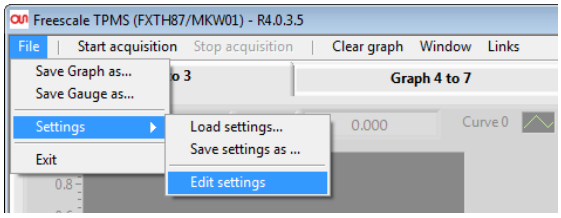
9. After opening the GUI, a window pops up: select one of the configuration files copied in the *data* folder:

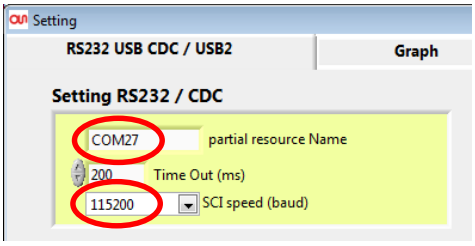


If the window does not pop up automatically or if you want to select another file, load the configuration file via: *File > Settings > Load settings…*

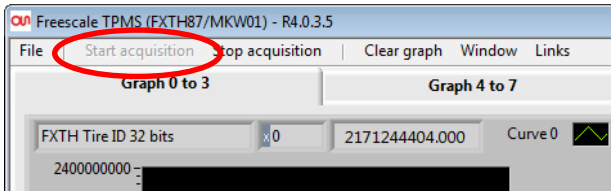


10. Update the port COM number (if it is different from COM27). For that, go to *File > Settings > Edit settings* and indicate the right COM number. Then click *SAVE AS* and save the updated config file under a new name (with updated COM port).

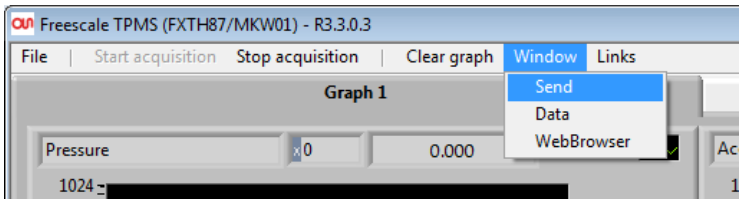




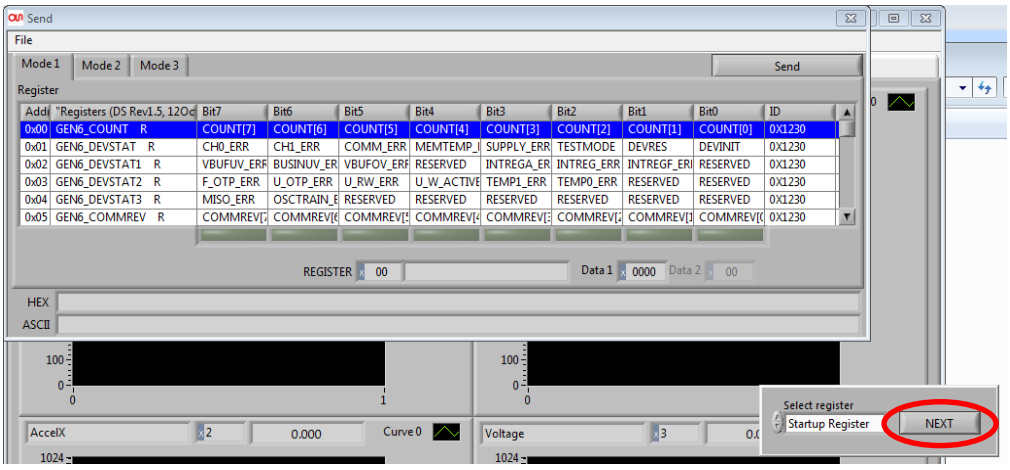
11. Click on *Start Acquisition* in order to send and receive data (*Start Acquisition* must be grayed out):



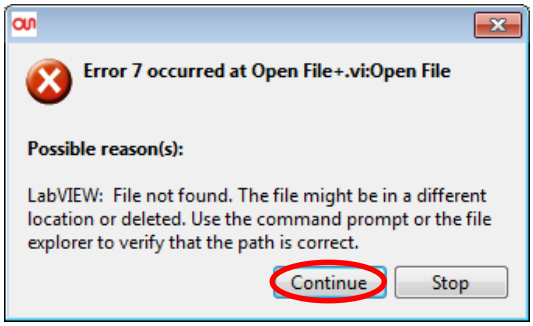
12. Click on *Window > Send* (this will open the window allowing the user to send data from the LabVIEW to the MKW01).



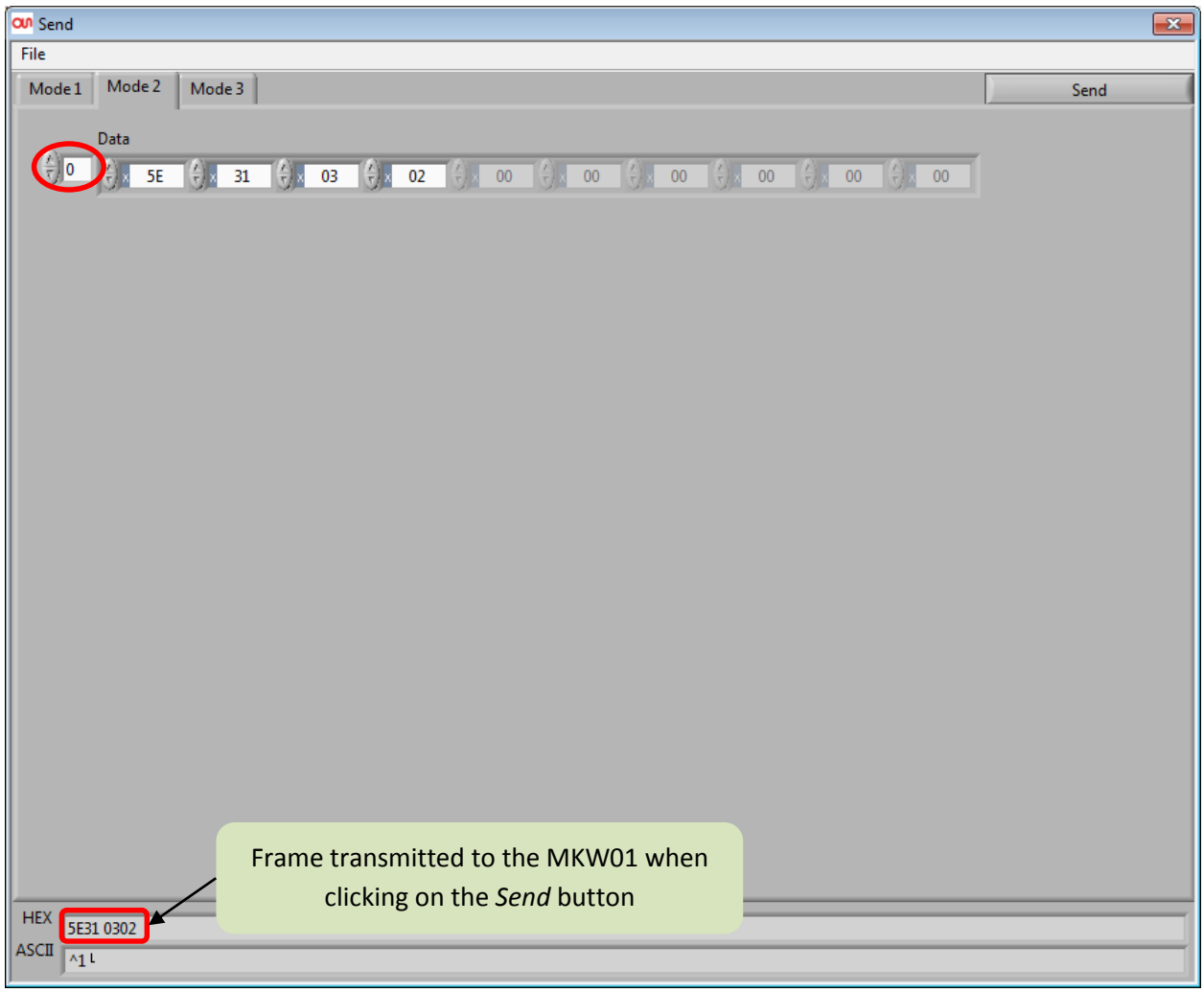
13. A window pops up, click on *NEXT:*



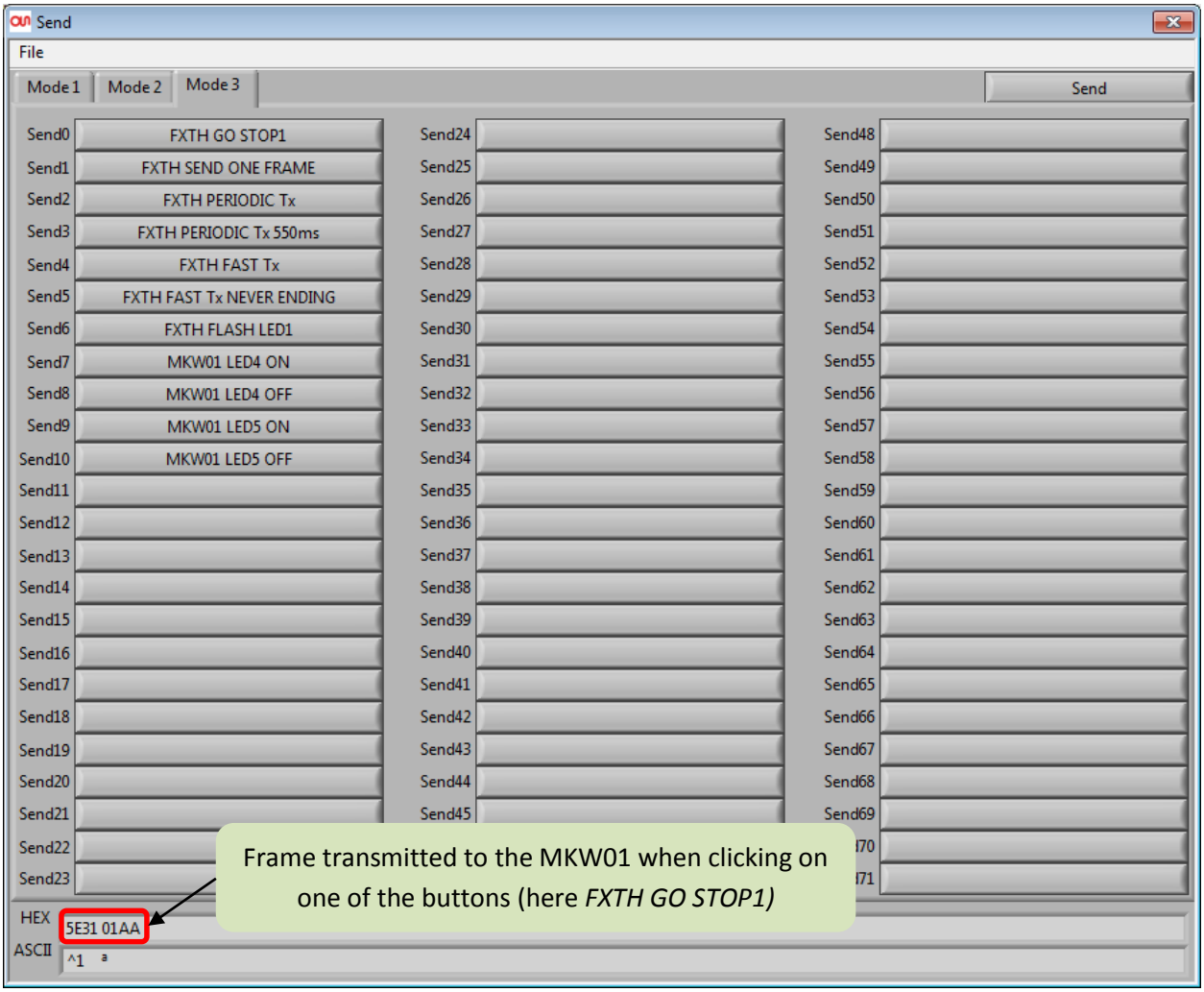
If the error below occurs, ignore it and click *Continue.* In order to remove this message, refer to the end of the section (*Troubleshooting).*



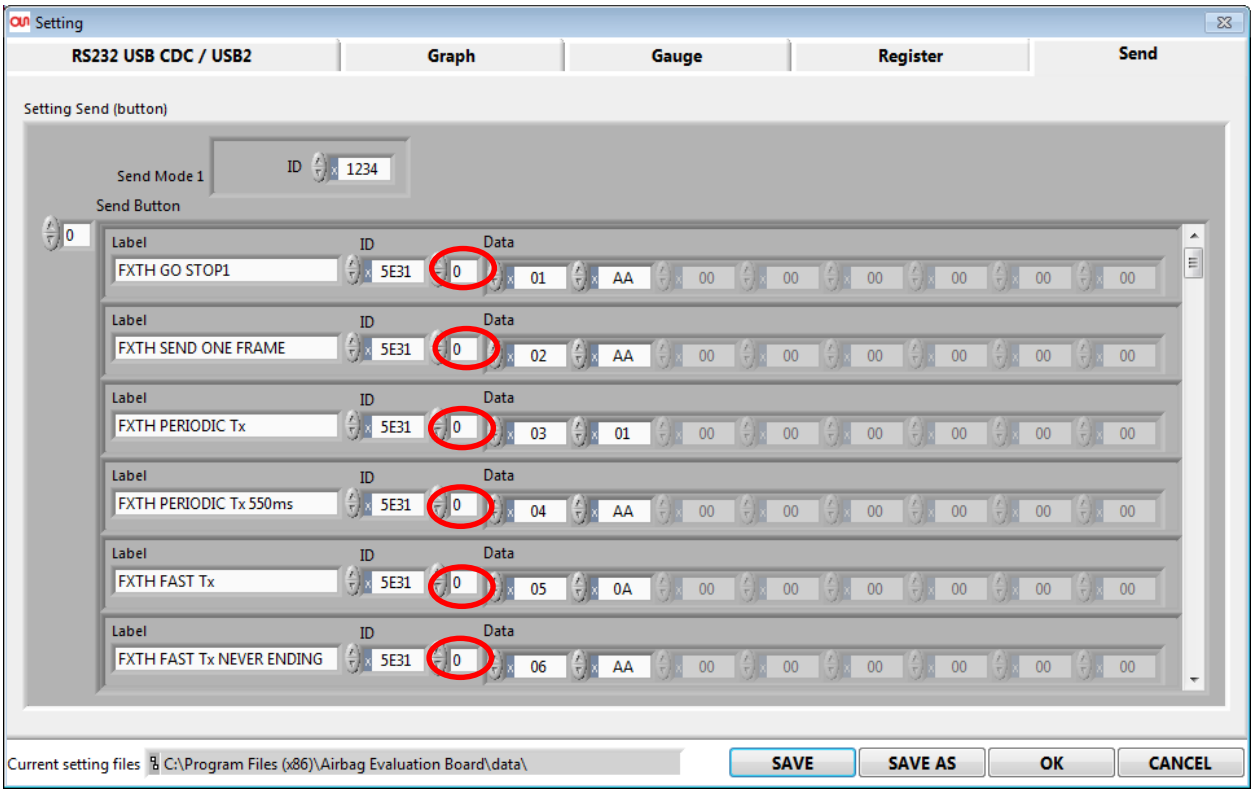
14. A window with 3 panels appears (if not, do again *Window > Send).*
    The first panel (Mode 1) is not used in this demo.
    The second panel (Mode 2) allows the user to send any frame through the COM port. For that, fill the array and then click *Send.* For example, to send the *Periodic Tx* command with a 2 seconds period, fill as follows:

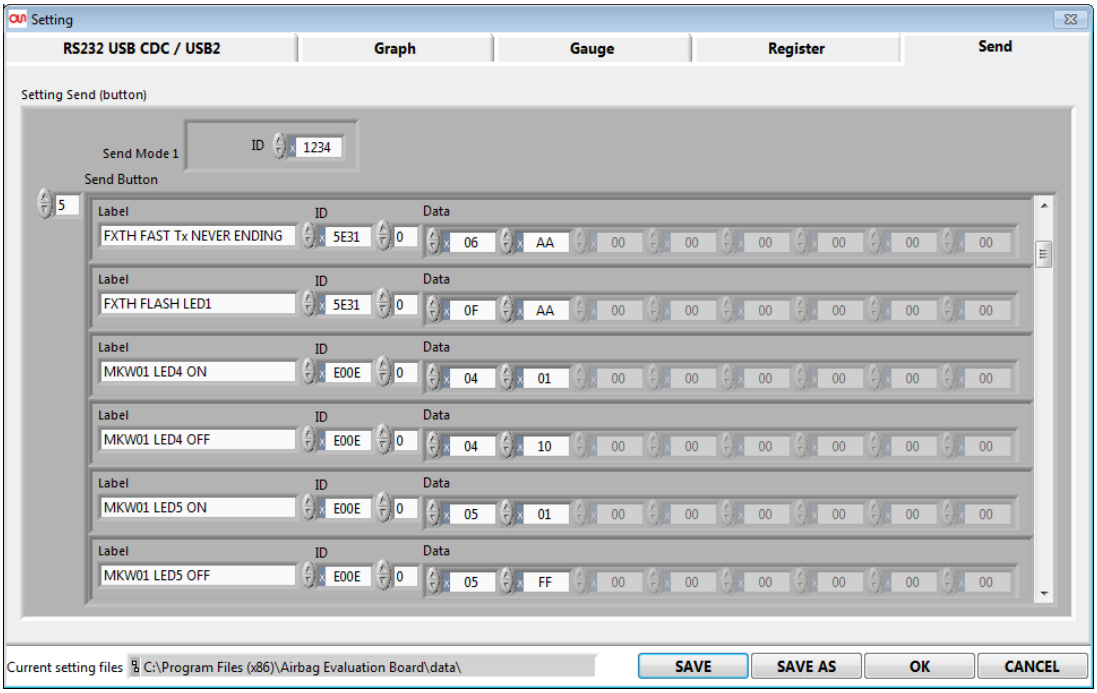


The third panel (Mode 3) displays buttons with pre-defined commands. The user only needs to click on one of these buttons to send a command (click on the button directly, not on *Send)*. Refer to section 2 for information on these commands:

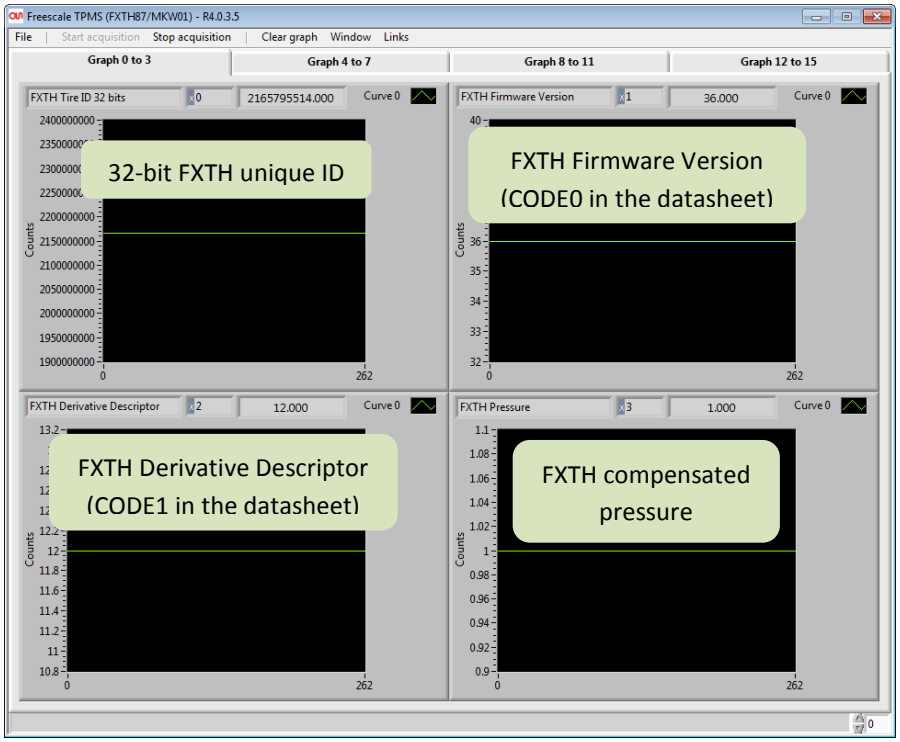


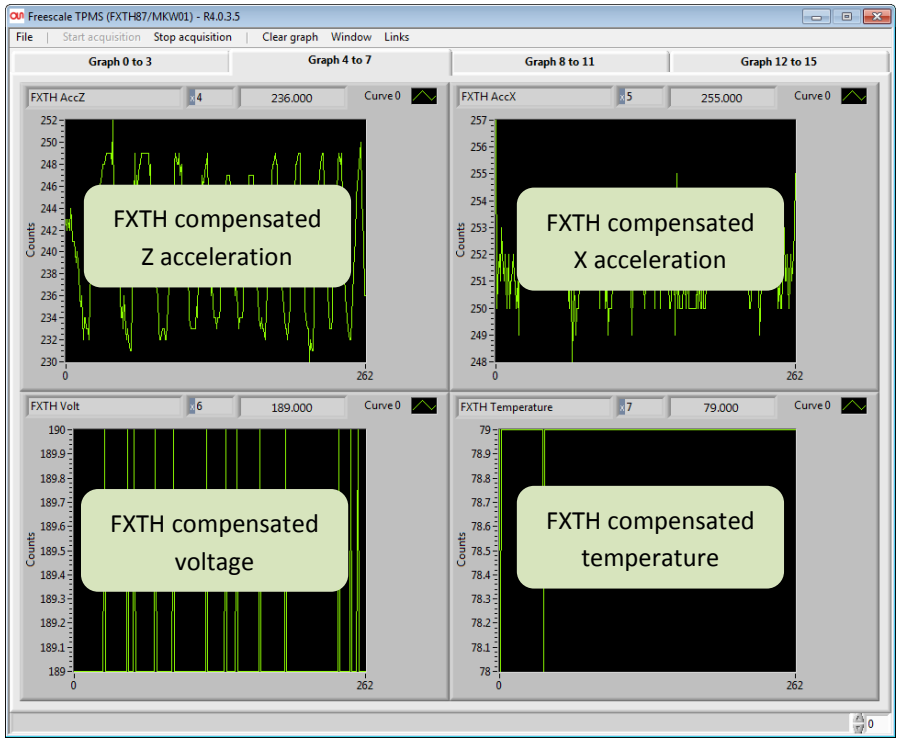
Important note: in order to modify the pre-defined frames of this panel, go to *File* > *Settings* > *Edit settings* and go to the *Send* panel. Here the name of the buttons and corresponding frame can be modified:

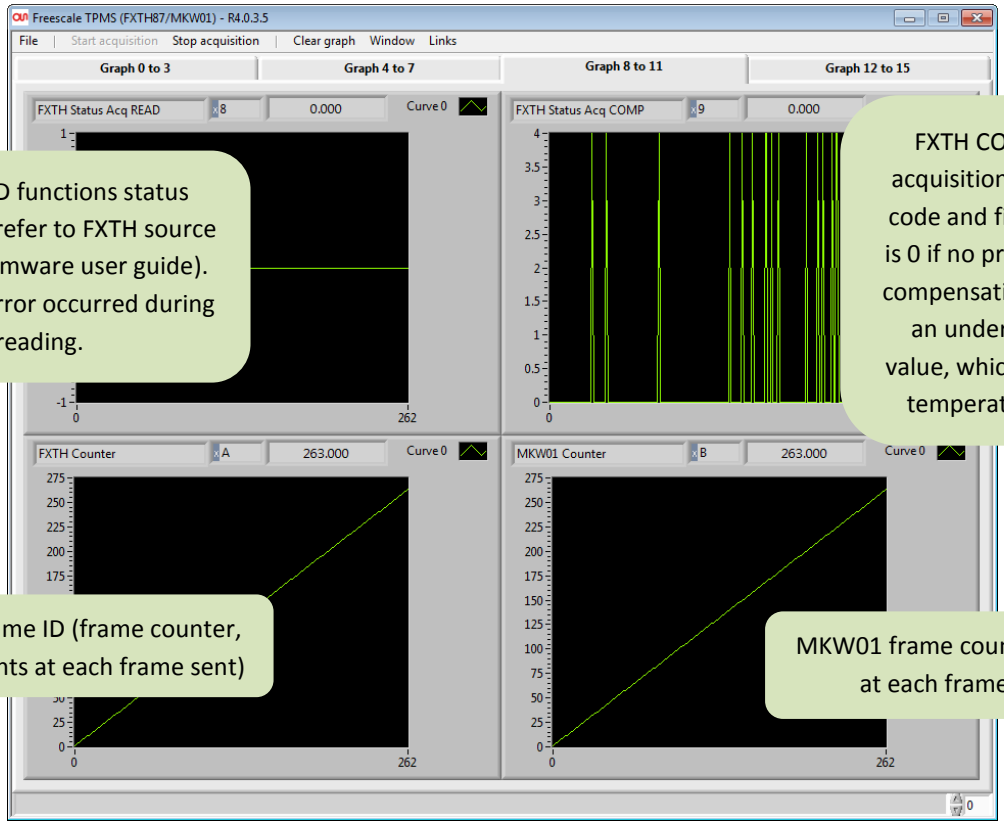




15. When RF data is transmitted to the GUI, it is displayed on the graphs:





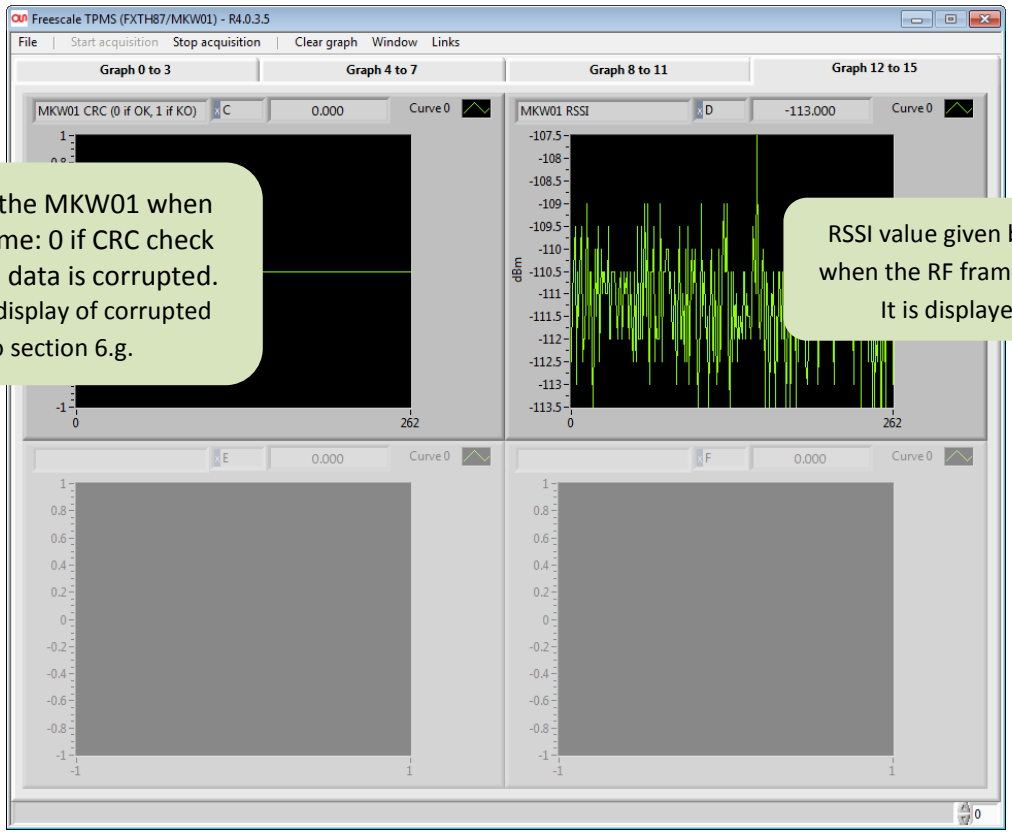


FXTH READ functions status acquisition (refer to FXTH source code and firmware user guide). It is 0 if no error occurred during reading.

FXTH COMP functions status acquisition (refer to FXTH source code and firmware user guide). It is 0 if no problem occurred during compensation. A value of 4 means an underflow of the pressure value, which is normal at ambient temperature (as shown here).

FXTH frame ID (frame counter, increments at each frame sent)

MKW01 frame counter (increments at each frame received)

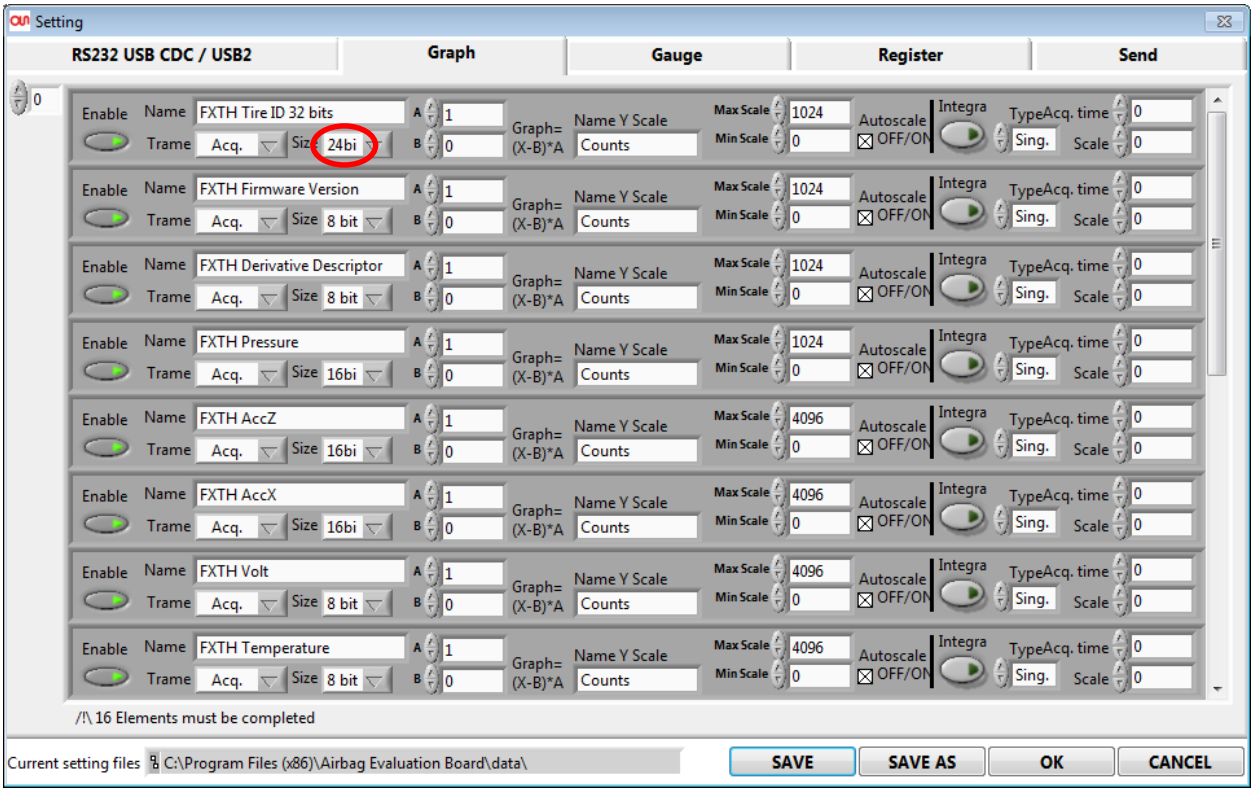


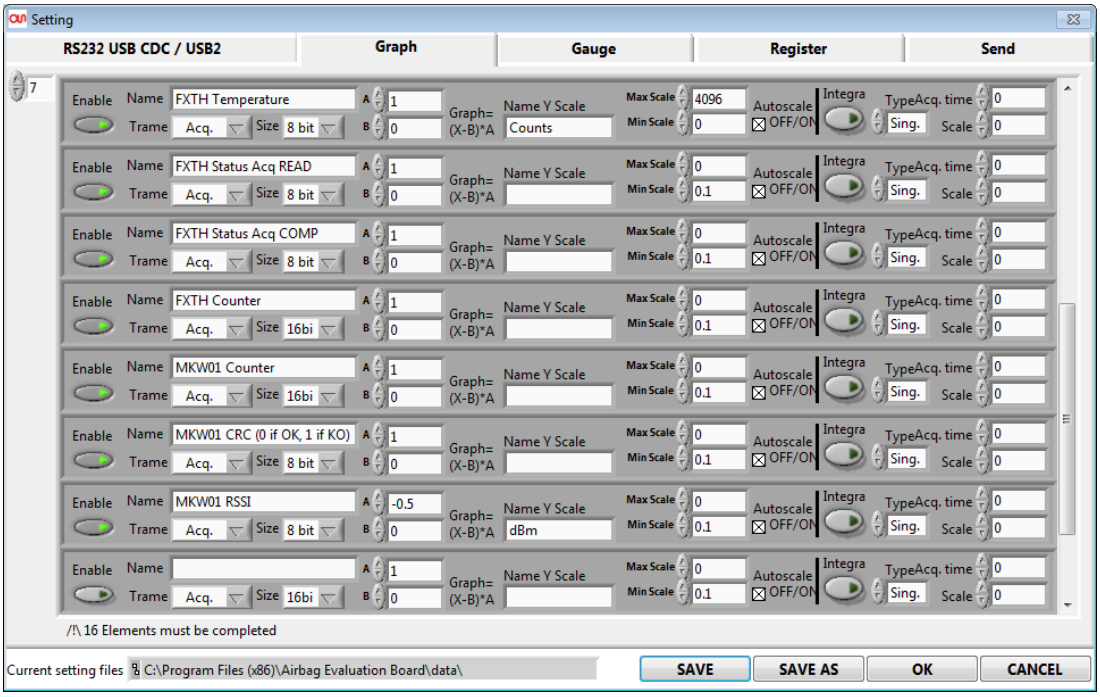
CRC check done by the MKW01 when receiving the RF frame: 0 if CRC check OK (data ok), 1 if RF data is corrupted. For information on display of corrupted data, refer to section 6.g.

RSSI value given by the MKW01 when the RF frame was received. It is displayed in dBm.

The configuration of the name of the graphs and length of each data is done in *File > Settings > Edit Settings > Graph*

Be careful!
In order to indicate a data of 32 bits, select 24 bits (and not 32), as done for the 32-bit unique ID:





b. Note on the Sensor GUI configuration files provided for the demo

There are four .cfg files provided for the Sensor GUI:

- *GuiSettingTPMS_MKW01_LF_RF_counts_COM27.cfg*: when using this file, pressure, acceleration, voltage and temperature are displayed in counts. The GUI does not process these compensated data (they are displayed as they are received).

- *GuiSettingTPMS_MKW01_LF_RF_FXTH87_450kPa_COM27.cfg*: to be used if using a TPMS emitter containing a 450kPa pressure sensor. Data received is processed by the GUI (not by the MKW01) so that pressure is displayed in kPa, acceleration in g, temperature in °C and voltage in volts. The GUI receives the data in counts and then converts them in common units before displaying them.
  For more information on the conversion in common units, refer to the pdf document *Compensated_Data_Conversion* included in the documentation folder of the MKW01 project.

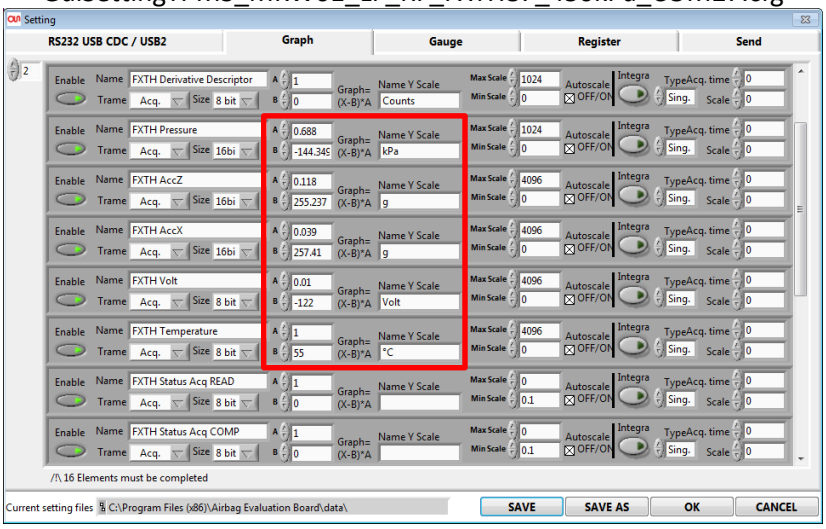
- *GuiSettingTPMS_MKW01_LF_RF_FXTH87_900kPa_COM27.cfg*: to be used if using a TPMS emitter containing a 900kPa pressure sensor. Data received is processed by the GUI (not by the MKW01) so that pressure is displayed in kPa, acceleration in g, temperature in °C and voltage in volts. The GUI receives the data in counts and then converts them in common units before displaying them.
  For more information on the conversion in common units, refer to the pdf document *Compensated_Data_Conversion* included in the documentation folder of the MKW01 project.

- *GuiSettingTPMS_MKW01_LF_RF_FXTH87_1500kPa_COM27.cfg*: to be used if using a TPMS emitter containing a 1500kPa pressure sensor. Data received is processed by the GUI (not by the MKW01) so that pressure is displayed in kPa, acceleration in g, temperature in °C and voltage in volts. The GUI receives the data in counts and then converts them in common units before displaying them.
  For more information on the conversion in common units, refer to the pdf document *Compensated_Data_Conversion* included in the documentation folder of the MKW01 project.

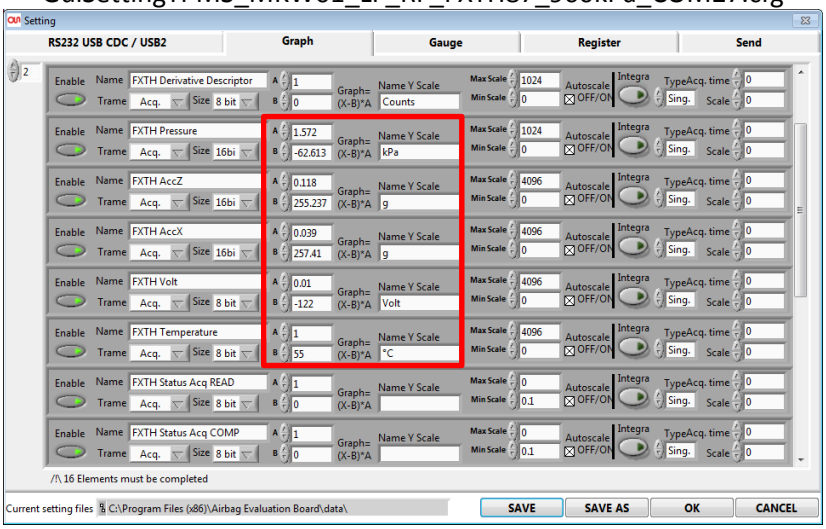
For each configuration file, the conversion applied to the compensated data received can be seen in *File > Settings > Edit Settings > Graph.*
Below are shown the settings of pressure, acceleration, temperature and voltage for each configuration file.

Note: the coefficients used in the conversions to common units have been taken from the FXTH87xx11 datasheet. For the other devices (1-axis, 87E), refer to the appropriate datasheet.

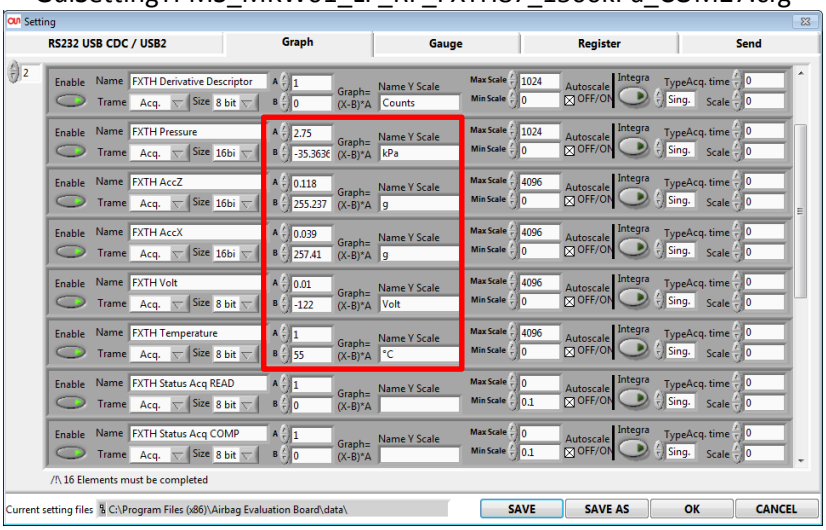
GuiSettingTPMS_MKW01_LF_RF_FXTH87_450kPa_COM27.cfg



GuiSettingTPMS_MKW01_LF_RF_FXTH87_900kPa_COM27.cfg

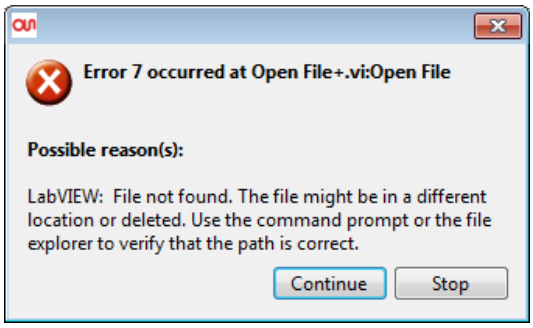


GuiSettingTPMS_MKW01_LF_RF_FXTH87_1500kPa_COM27.cfg

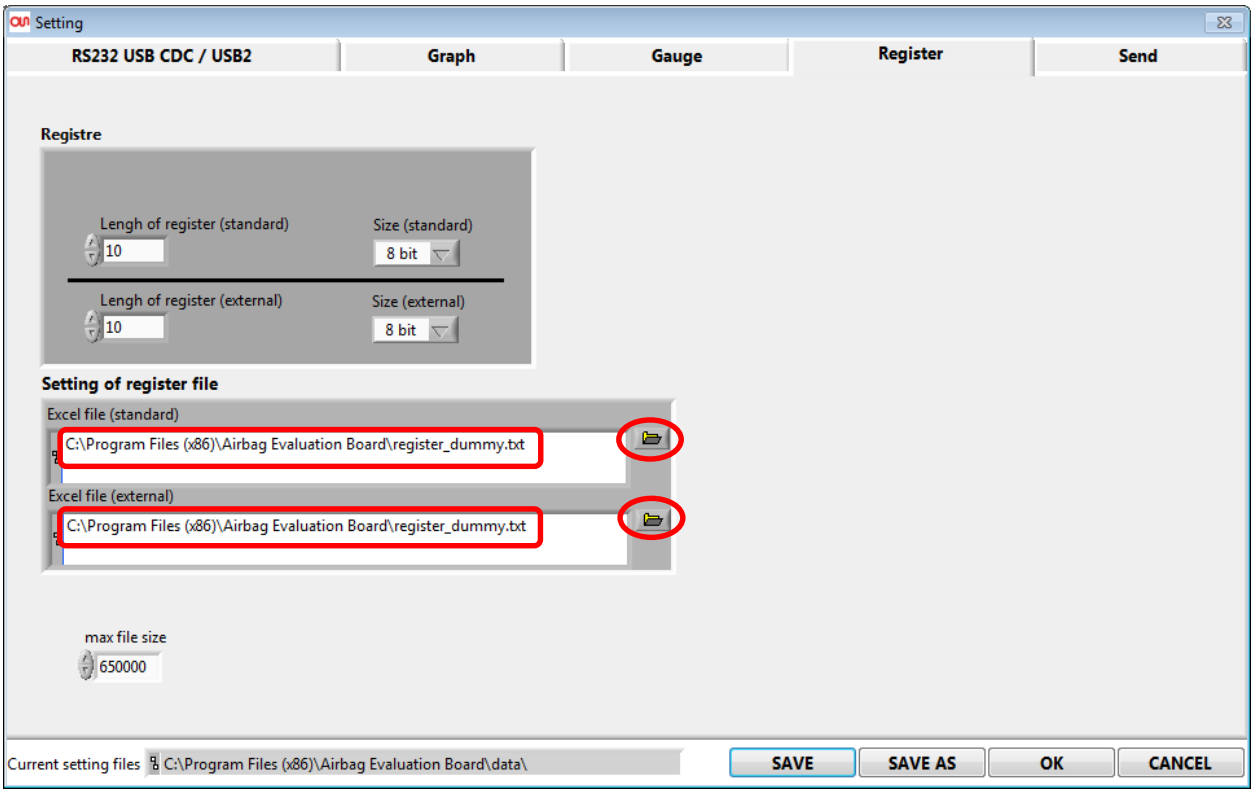


c. Troubleshooting

i. Sensor GUI: 'Error 7 occurred at Open File'



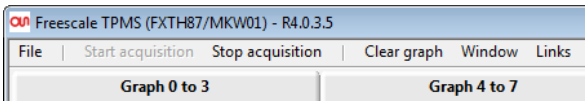
In order to remove the error message, copy the file *register_dummy.txt* provided in the MKW01 documentation folder and copy it in the installation folder of the GUI.
Then in the GUI go to *File > Settings > Edit Settings > Register* and select twice this file. This will remove the error.



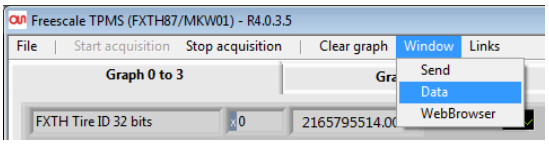
ii. MKW01: no RF frame received

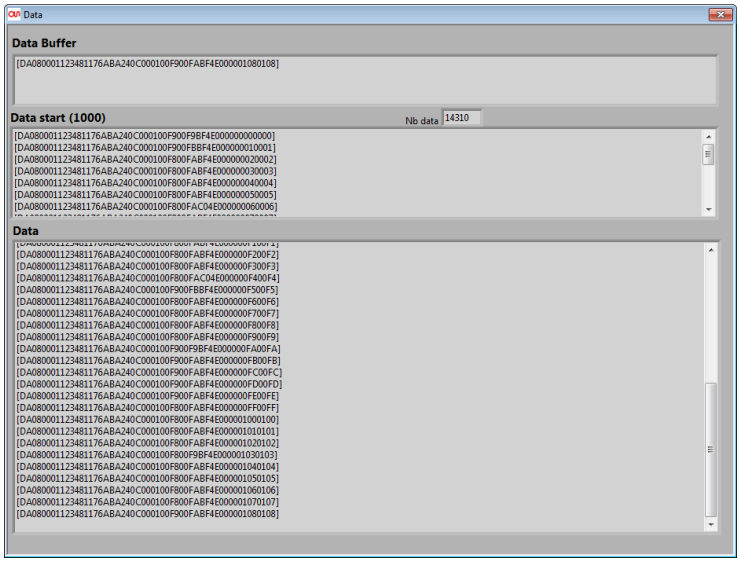
On the MKW01 side, PTC2 (LED16) is ON when an RF frame is received.

- Check that the port COM and the baud rate indicated in *File > Settings > Edit settings > RS232 USB CDC* are correct.
- Make sure the distance FXTH/MKW01 is not too long: try with a distance around 30cm to be sure the frames are received.
- Start Acquisition must be grayed out in order to be able to receive data:



- If it still does not work: close the GUI, disconnect the MKW01 from the computer, then reconnect it and re-open the GUI.
- In the MKW01 project, if the option of not displaying the corrupted data has been selected (RF_DATA_CORRUPTED is DISCARD_DATA in Rf_Rx_Appli.h), maybe all the received frames are corrupted so nothing is displayed. To check that, select the other option (RECEIVE_DATA), recompile the project and reprogram the MKW01.
- Check that the format of the frame sent to the GUI is correct. Go to *Window > Data* and check the frame format looks like the following: [DA0800011234 *data*]
  If it does not look like this and the MKW01 source code has not been modified, adjust the baud rate of the MKW01: in UART.c in the function UART0_SetBaudRate, adjust the values of UART0_BDL (due to the MKW01 32kHz oscillator imprecision, the baudrate may need to be adjusted on some boards).

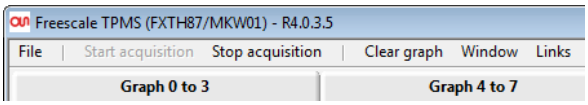




iii. MKW01: no LF frame is sent

On the MKW01 side, PTE1 (LED10) is ON while an LF frame is being sent (be careful: it does not mean the LF antenna is supplied so the LF frame may not be actually sent).

- Check that the port COM and the baud rate indicated in *File > Settings > Edit settings > RS232 USB CDC* are correct.
- Check that *Start Acquisition* is grayed out:

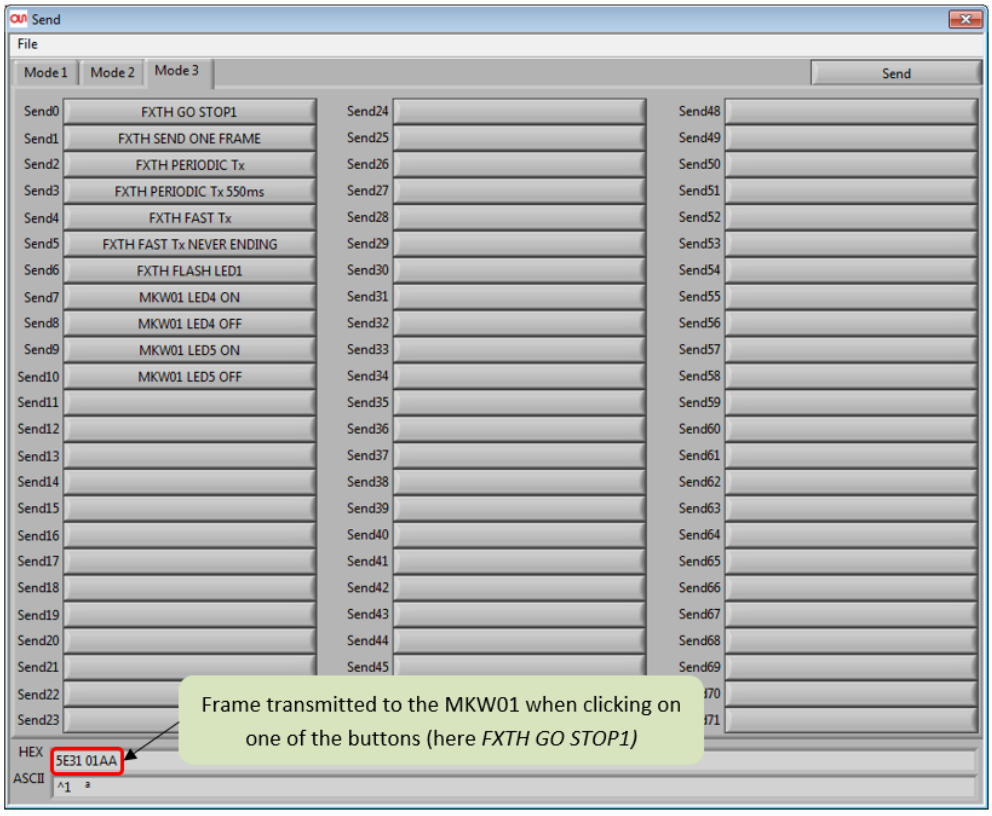


- Check that the LF antenna is supplied. If using the FRDM-SENSORS-LF board, the LF antenna can be supplied between 5V (for short distance) and 18V (long distance). Typical: 9V for a distance emitter/receiver of 35cm free-air.
- Click on the command MKW01 LED4 ON. If LED4 (PTA4) does not turn ON on the MKW01 board it means there is no communication between the GUI and the MKW01. So close the GUI, disconnect the MKW01 from the computer, then reconnect it and re-open the GUI.
  If the LED turns ON but the other FXTH commands does not seem to work then it may be a LF FXTH reception problem (see next point) or a RF MKW01 reception problem (see previous point).

iv. FXTH: no LF frame is received

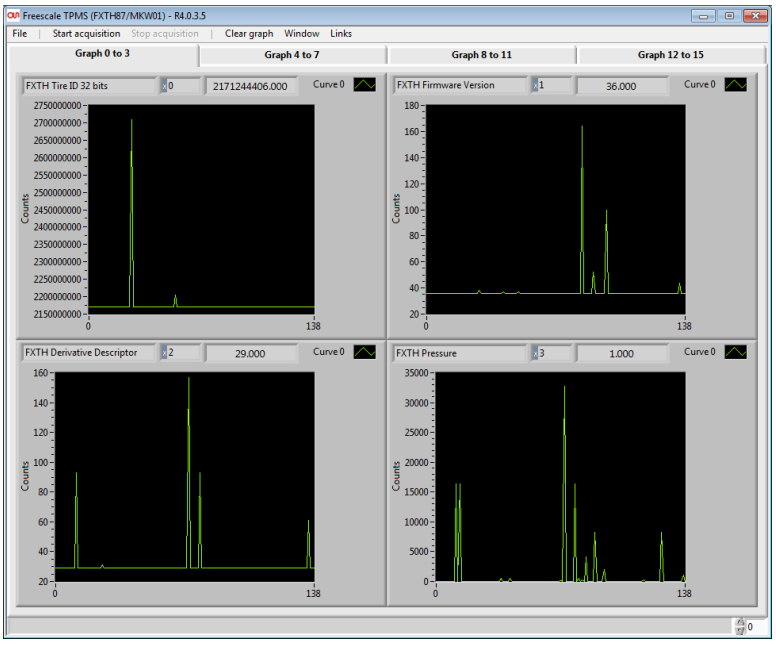
If LEDs are enabled in the FXTH project (refer to section 6.a), PTA2 (LED3) is turned ON when an LF command is received and PTA1 (LED2) is ON during 100ms when the LF command is unknown (or wrongly decoded).

- Check that an LF frame is actually sent by the MKW01 (previous section).
- Turn off the FXTH, disconnect the BDM if it was connected and turn on the FXTH.
- Adjust the LF antenna power supply (the antenna connected to the MKW01 board). Typical power supply is 9V for a distance emitter/receiver of 35cm free-air, but it can be adjusted between 5V and 18V maximum.
    - If the distance between the LF antenna and the FXTH coil is short with no obstacle (less than 30cm free-air), the FXTH LF coil may saturate if the LF signal is too strong (the LF will be received but not decoded). In this case use a 5V power supply instead.
    - If the distance is longer or if the FXTH is behind an obstacle (inside a tire), a higher power supply may be needed.
- Check that the frame sent to the MKW01 is correct and the command has been implemented (if default source code has been modified).
  The frame sent can be seen on the bottom left corner of the window:



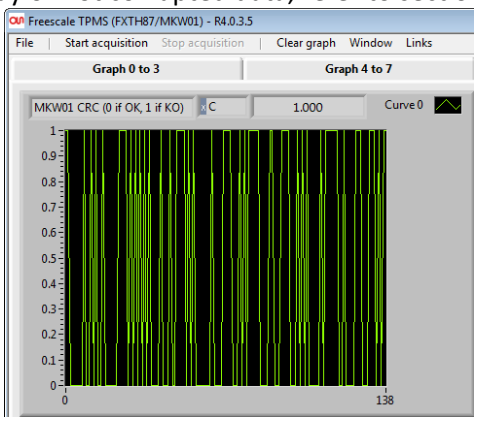
v. Sensor GUI: data displayed is corrupted

Data displayed is corrupted if bytes are taking wrong values. For example, if some bytes supposed to be fixed like ID or firmware version take different values as shown below:



This can be due to two things:

- RF data corruption: data received by the MKW01 is corrupted so we see corrupted data displayed. To verify that, look at the MKW01 CRC panel of the GUI: if CRC is 1 it means RF data received is corrupted. Also, in case of RF data corruption, MKW01 counter, MKW01 CRC and MKW01 RSSI should not be corrupted (because they do not come from the RF frame but are added by the MKW01). These data are displayed on the GUI.
  In order to choose to display or not corrupted data, refer to section 6.g.



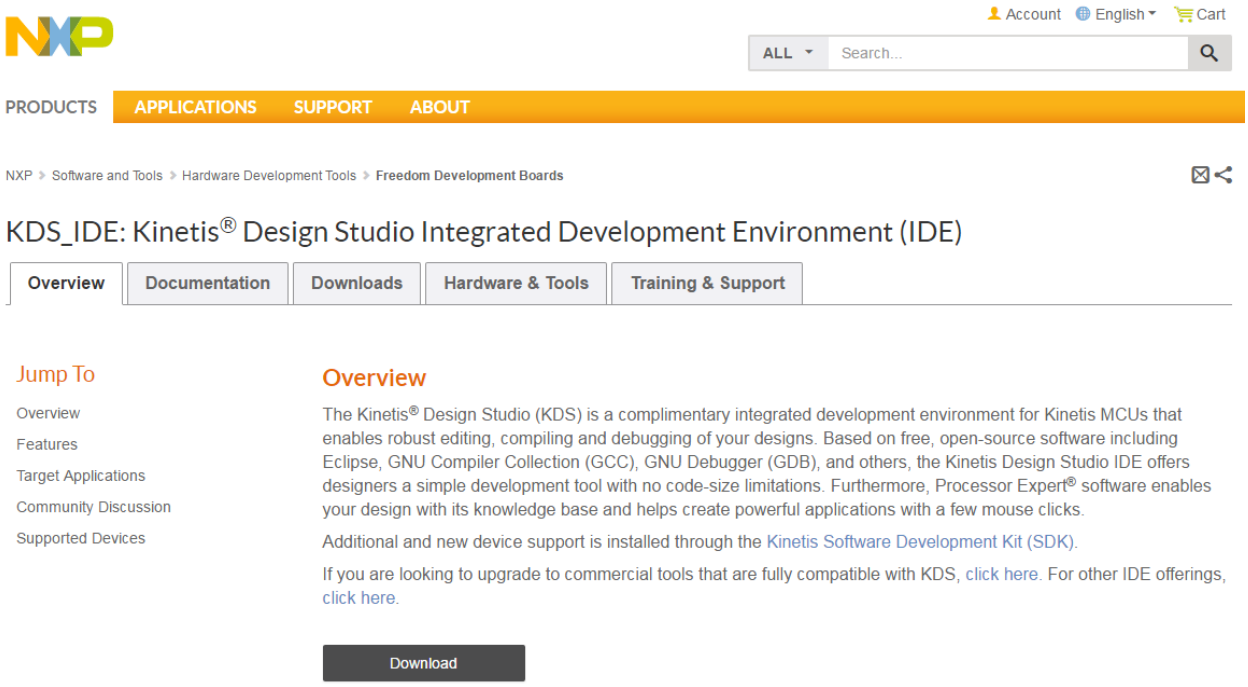
  In order to reduce the amount of data corrupted, adjust the FXTH/MKW01 distance (to be around 35cm) and try several orientations of the boards (parallel, perpendicular, rotated by 90°…).

- If all panels show wrong data (including MKW01 counter, MKW01 CRC and MKW01 RSSI), this can be due to the MKW01 UART baud rate: the UART baud rate may need to be slightly adjusted. For that refer to the function *UART0_SetBaudRate* of the file *UART.c* in the MKW01 project.

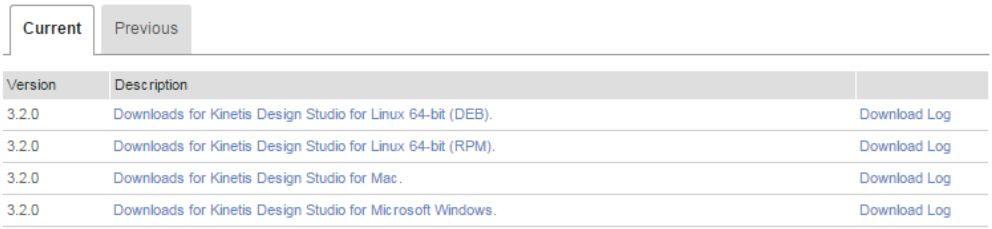
8. MKW01: Kinetis Design Studio v3.x installation

The demo project does not use Processor Export and Kinetis SDK. A simple installation of KDS v3.x is enough. The screenshots below refer to version 3.2, but the same procedure can be followed for all versions of KDS 3.x.

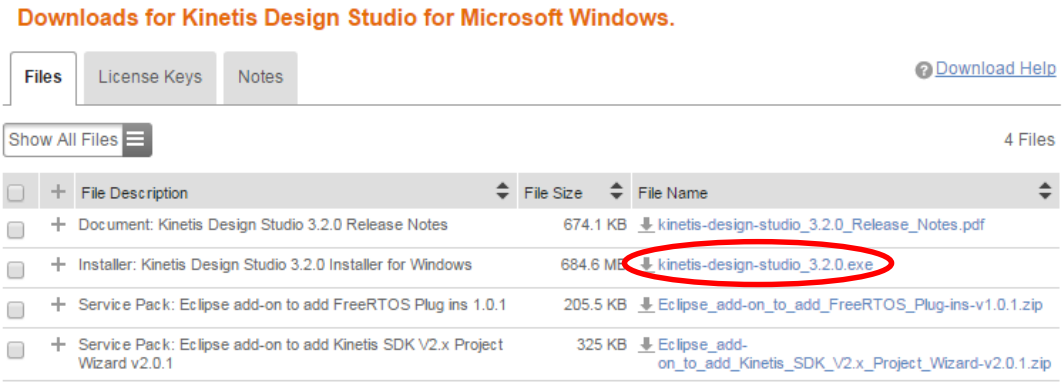
1. Go to the KDS webpage of the NXP website: KDS Webpage (if the link does not work, follow *NXP > Software and Tools > Hardware Development Tools > Freedom Development Boards > Kinetis Design Studio IDE)*



2. Click on *Download* (need to be signed in) and select the appropriate version:

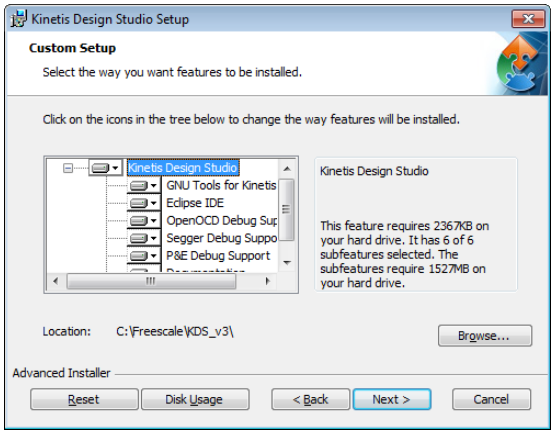


3. Download *kinetis-design-studio_3.x.exe* (it can also be downloaded via the installer):



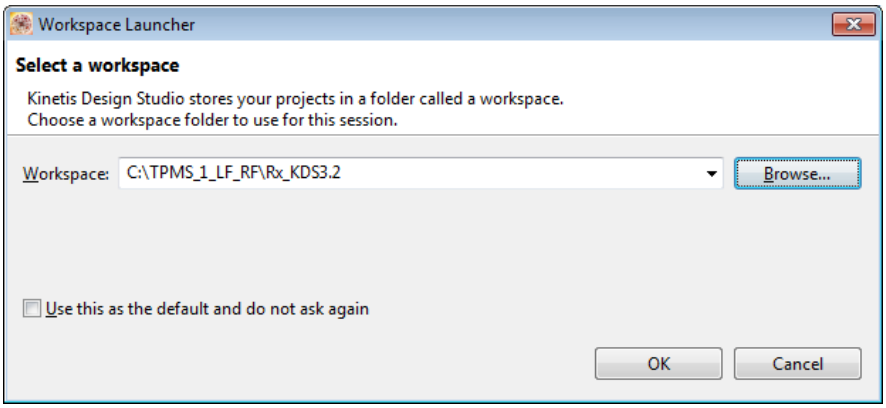
4. When *kinetis-design-studio_3.x.exe* is downloaded, execute it and follow the instructions.

When required, install all features:

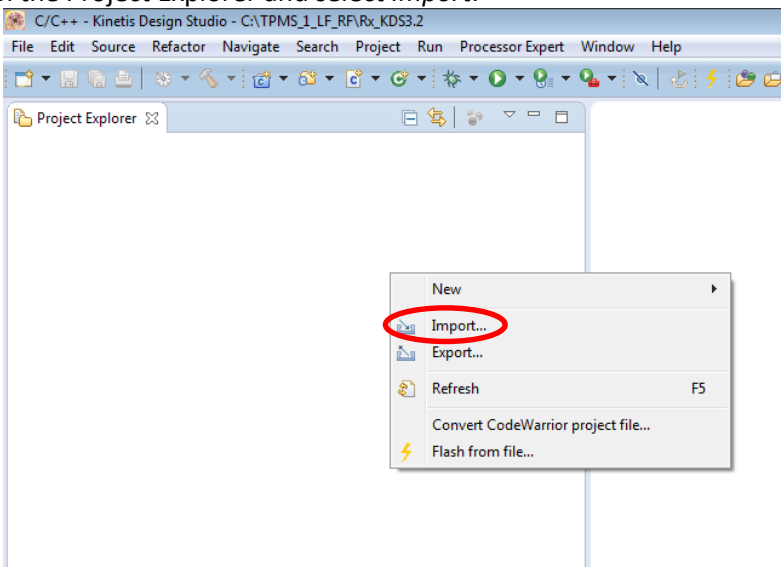


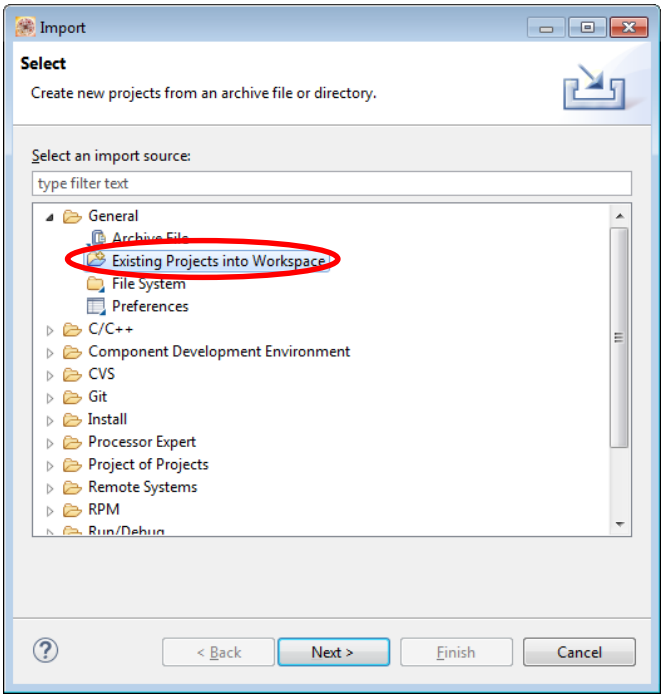
5. When the installation is complete, start KDS 3.x

Choose your workspace:

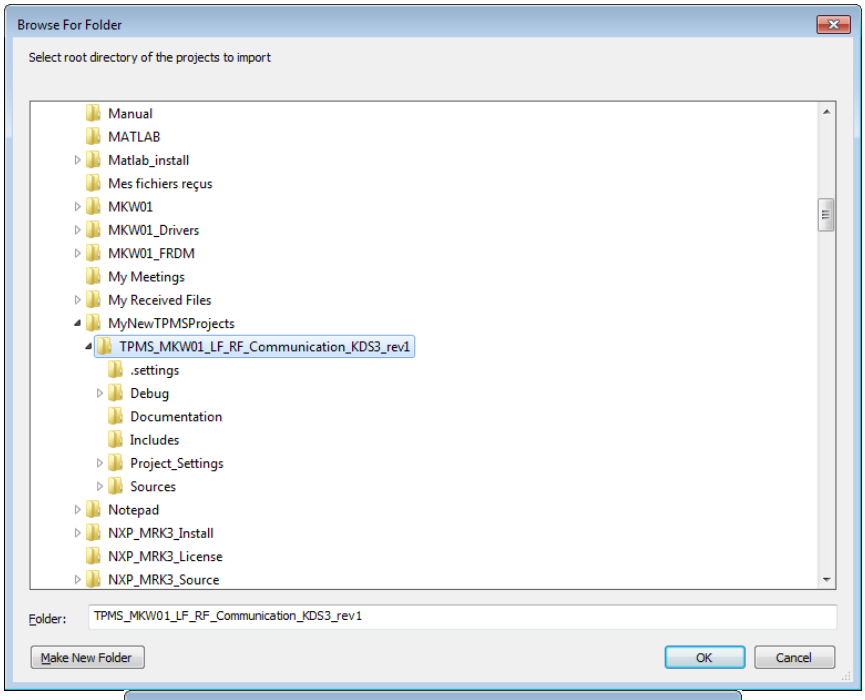


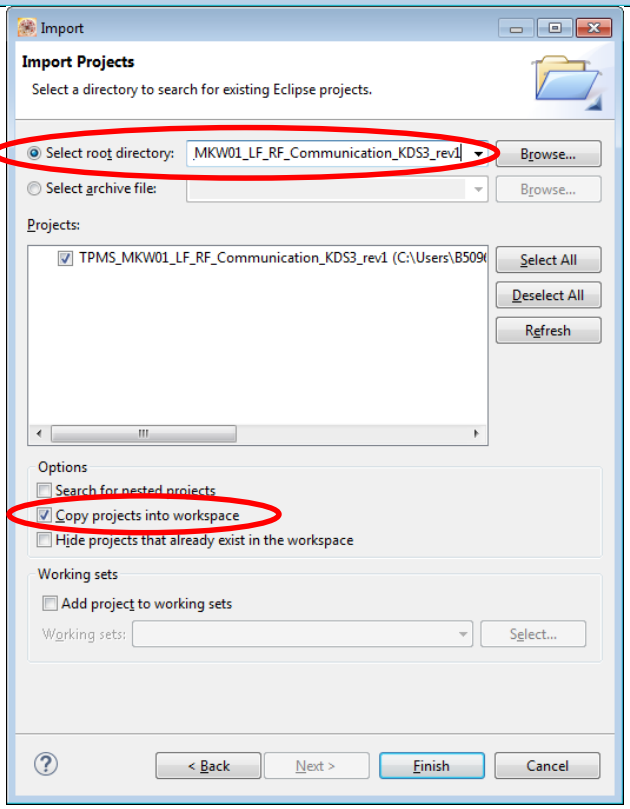
6. Right click in the Project Explorer and select *Import:*



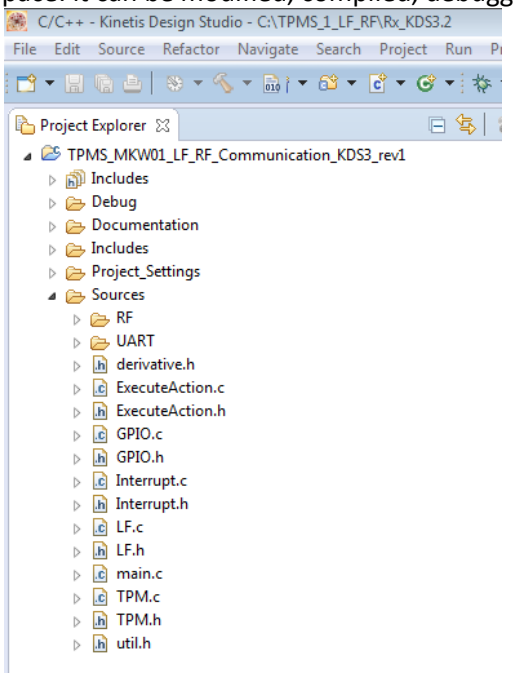


7. Select the project's folder:

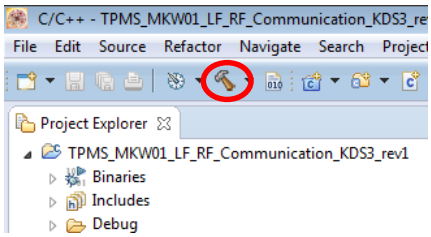




The project is now in the workspace: it can be modified, compiled, debugged…

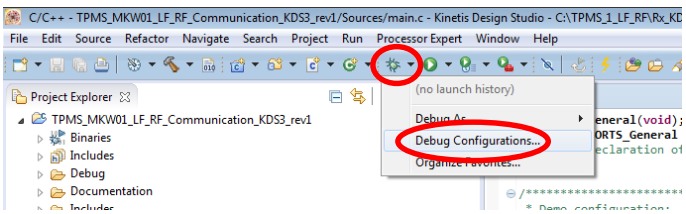


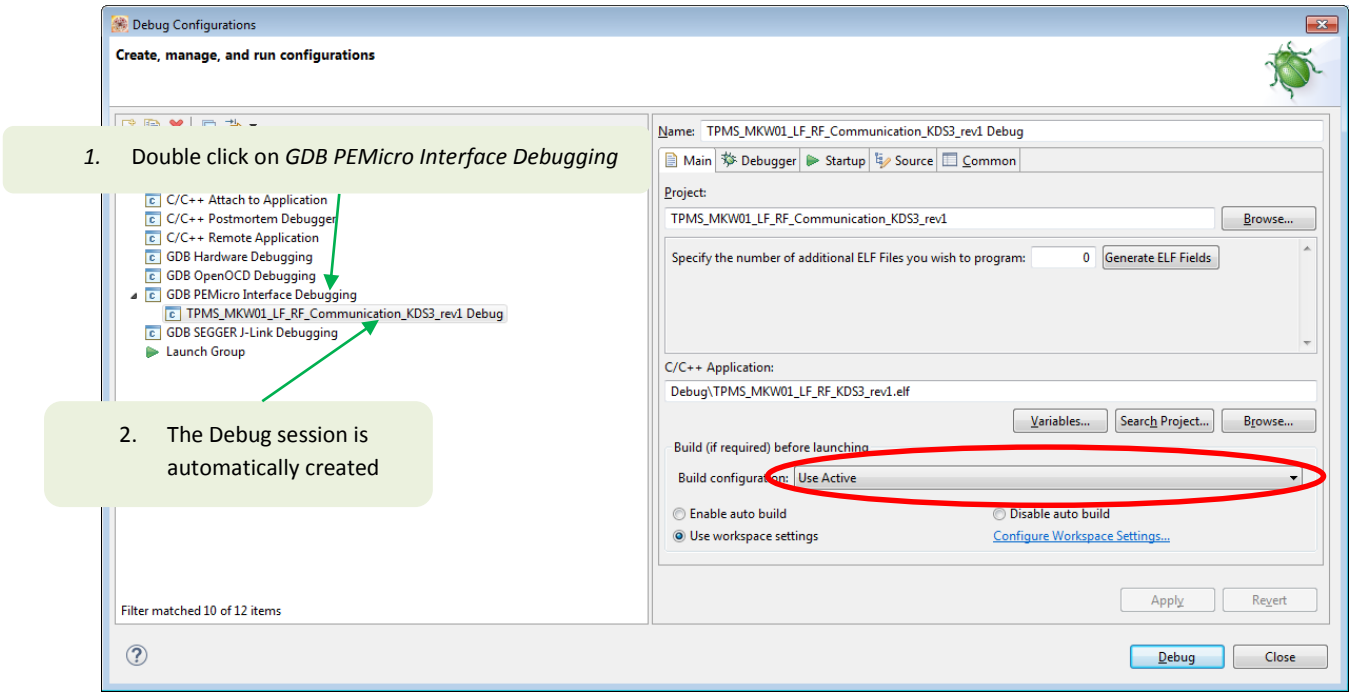
8. In order to compile (build) the project, click on the hammer. If the button is not accessible (grayed out), open a source file of the project (double click on it). The button will then be enabled:



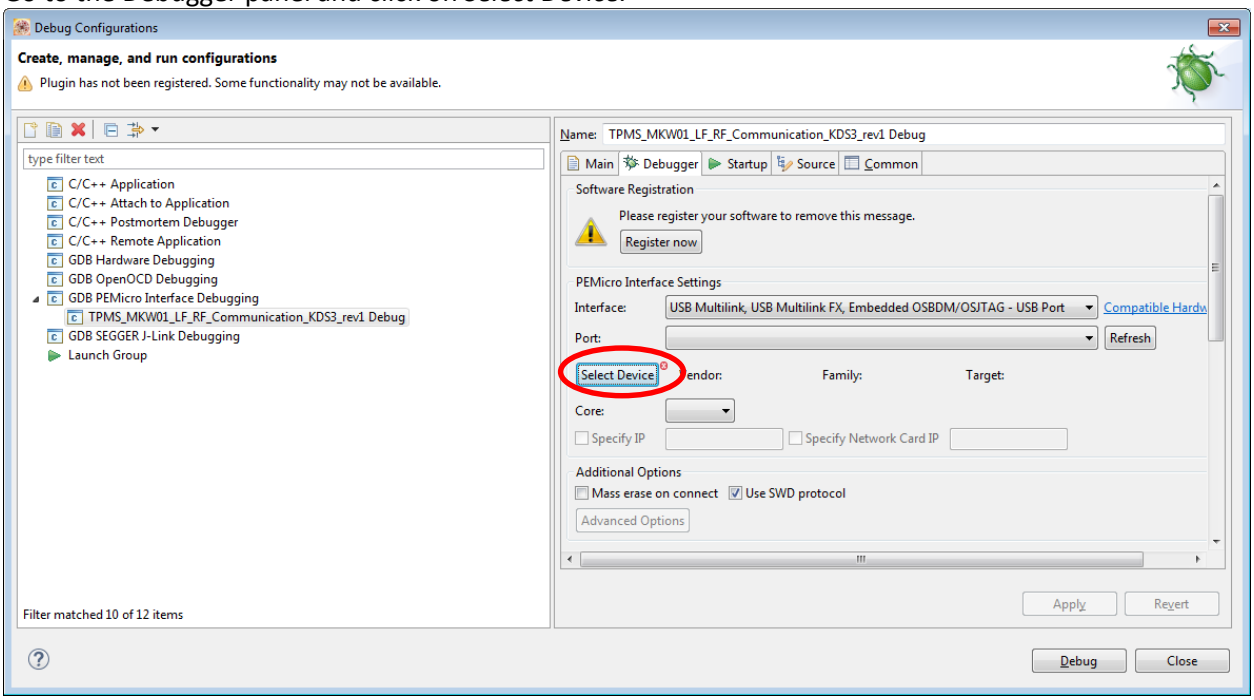
If the error *'No rule to make target'* occurs, right click on the project folder and select *Clean Project.* Then build again.

9. In order to program the MKW01, click on *Debug Configurations…* (the project first needs to be built)

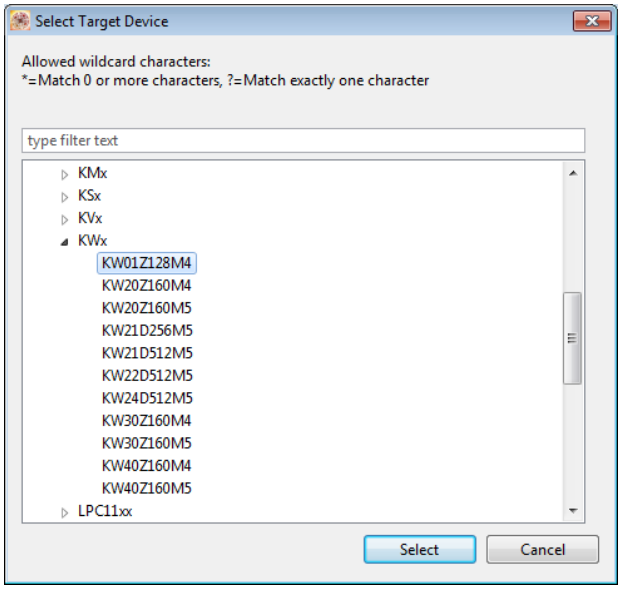


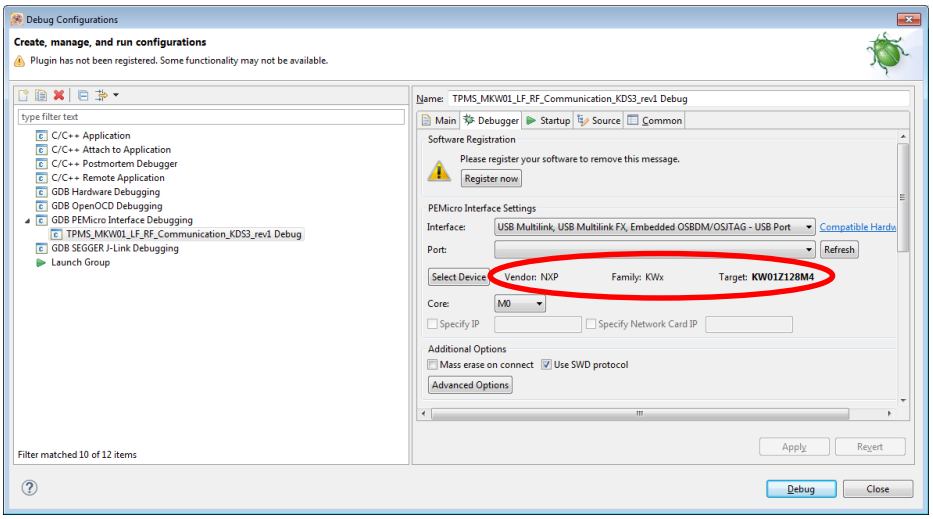


Go to the Debugger panel and click on *Select Device:*

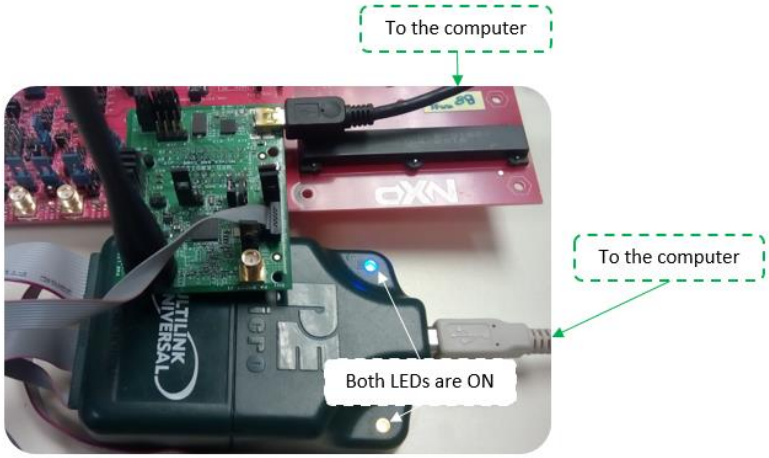


Select the KW01Z128M4:



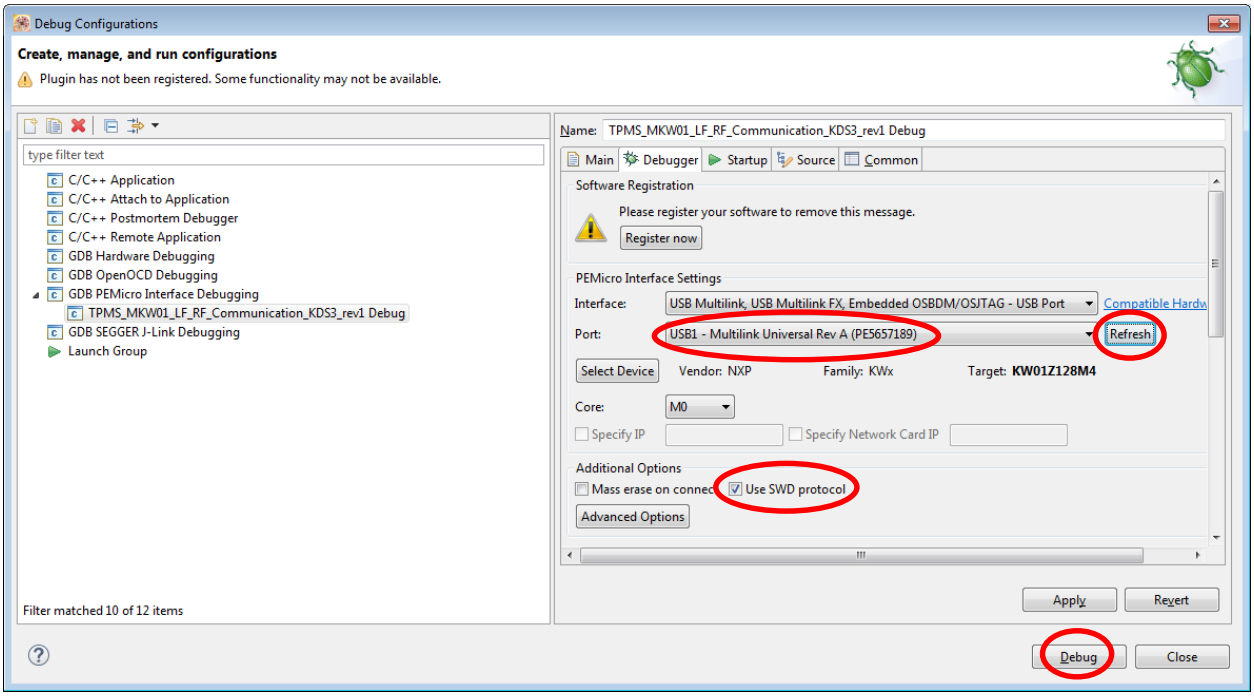


Now connect the Multilink Universal to the computer on one side and to the MKW01 on the other side:

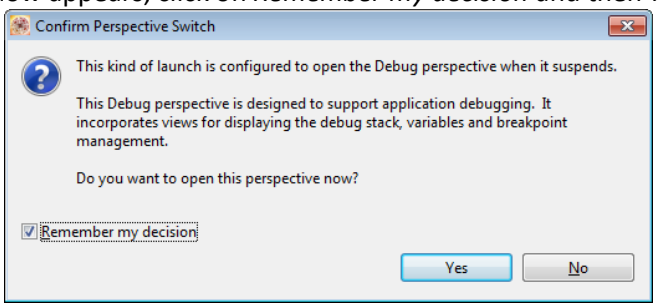


Click on *Refresh* and the USB port should appear.

Once this is done, click on *Debug* to program the MKW01:



When the following window appears, click on *Remember my decision* and then *Yes:*



Run the program: