

ON Semiconductor

Is Now



To learn more about onsemi™, please visit our website at
www.onsemi.com

onsemi and onsemi. and other names, marks, and brands are registered and/or common law trademarks of Semiconductor Components Industries, LLC dba "onsemi" or its affiliates and/or subsidiaries in the United States and/or other countries. onsemi owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of onsemi product/patent coverage may be accessed at www.onsemi.com/site/pdf/Patent-Marking.pdf. onsemi reserves the right to make changes at any time to any products or information herein, without notice. The information herein is provided "as-is" and onsemi makes no warranty, representation or guarantee regarding the accuracy of the information, product features, availability, functionality, or suitability of its products for any particular purpose, nor does onsemi 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 special, consequential or incidental damages. Buyer is responsible for its products and applications using onsemi products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by onsemi. "Typical" parameters which may be provided in onsemi 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. onsemi does not convey any license under any of its intellectual property rights nor the rights of others. onsemi products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use onsemi products for any such unintended or unauthorized application, Buyer shall indemnify and hold onsemi and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that onsemi was negligent regarding the design or manufacture of the part. onsemi is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner. Other names and brands may be claimed as the property of others.

Workflow for an IoT Product

Detailed Example of a Development Process for a Wireless Alarm System based on Sigfox® Protocol

Introduction

In this application note a complete list of process steps is described in order to develop a specific system in the IoT environment. In order to make the description more practical the PIR sensor alarm example is used.

The PIR motion sensor alarm is a product demo developed by ON Semiconductor for detecting movement events and sending messages on a cloud by means of Sigfox network. The following picture shows the finished product. The alarm system is built around two main ON Semiconductor devices: the AX-SFxx-API-y RF microcontroller (Sigfox API stack SW version, xx representing the Sigfox Radio Configuration Zone) and the NCS36000 passive infrared (PIR) detector controller.

This alarm system is only intended for showing capabilities of ON Semiconductor devices and cannot be considered as real monitoring system for smart home applications. It has some limitations that will be discussed in more details in the concept analysis paragraph.

Process Flow

The main steps of the process followed for the development of the PIR sensor alarm system are here summarized:

- From Concept Analysis to “System Requirements”, Specification for HW and SW, included Mechanical Constraints
- HW/SW Development and First Prototype Development
- Testing, HW and SW Bug Tracking and Corrective Action for Fixing Potential Issue
- SW Development for Certification Approval
- Product Prototype Testing
- CE-ETSI regulatory certification
- Sigfox Ready™ and Sigfox Verified™ Certification
- Final Production

Most of the steps are practical then they will not be discussed in the document. The main part that will be treated are HW and SW development and certification procedure.



ON Semiconductor®

www.onsemi.com

APPLICATION NOTE



Figure 1. ON Semiconductor Alarm System

CONCEPT ANALYSIS

System

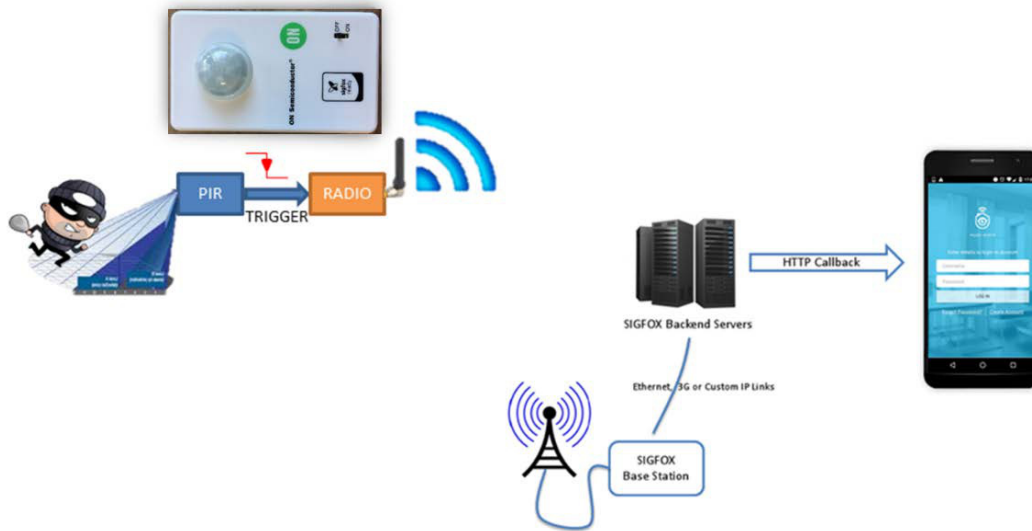


Figure 2. Block Diagram, from Movement Detection to Remote Control through Sigfox Network

Scope and Behavior

The Alarm system is used for monitoring limited environment, like rooms, detecting movement events. In order to be fully working, it should be placed in an area covered by Sigfox network.

When an event is recognized, a Sigfox message is sent on the cloud and it can be read back by the user accessing the Sigfox backend or, could be read developing a web or mobile application getting access to the SIGFOX backend server.

Visual feedbacks are provided to the user in order to understand the system activity by means of LEDs: yellow, red and green.

- Yellow is turning ON whenever the PIR sensor detects a movement.
- Red is turning ON when a radio communication is performed.
- Green is turning ON as result of the Sigfox communication: blinking when error, hold on for a while when no error.

Sigfox Transmission Specification

Since Sigfox contracts provides up to 140 messages per day in uplink and since the PIR sensor could detect a much higher number of events per day, a rule for avoiding transmission saturation has been considered. Thus the SW will send up to 6 messages per hour where the message content is slightly different according to the timing. In fact, the number of events detected by the PIR sensor during the silent period will be recorded and when the one hour time event is triggered and a further movement is detected, a Sigfox communication with the PIR events' counter is sent out on the cloud (always 1st message of the next hour). Only after the first message of the new hour the event counter is reset.

For sake of simplicity the following picture will show the concept.

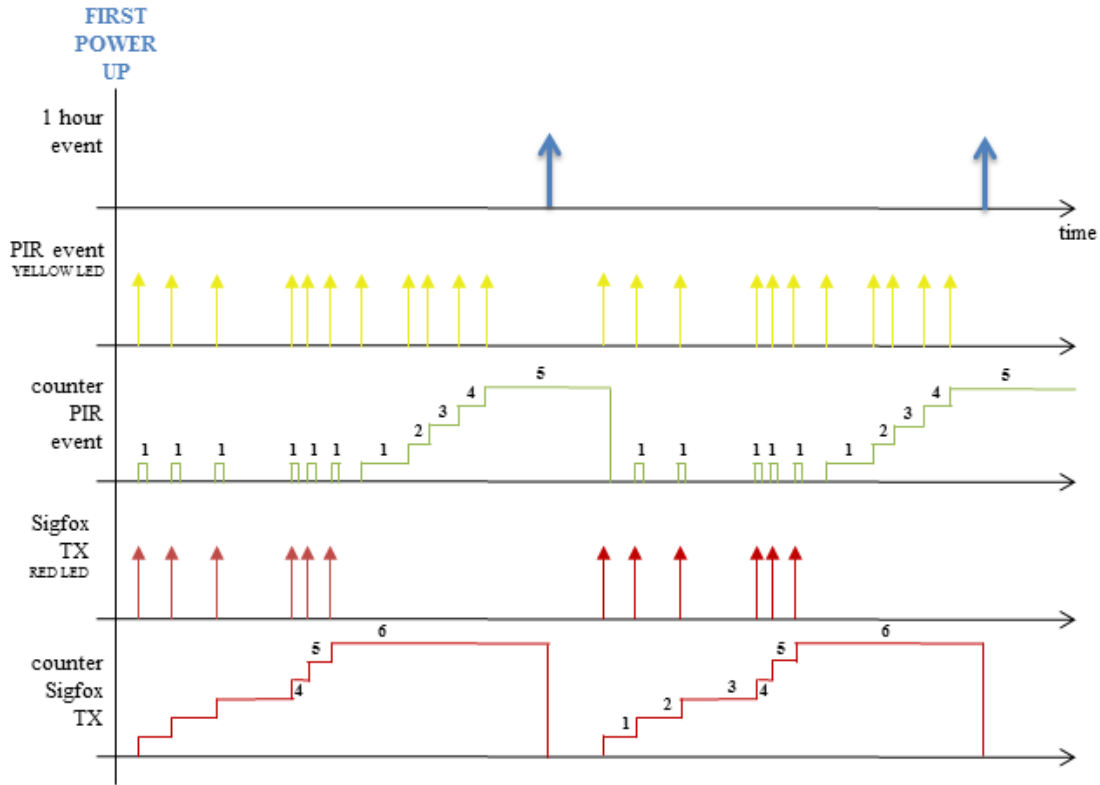


Figure 3. Alarm System Behavior

Power Management, Supply Voltage and Battery Life

In order to maximize the lifetime of the Alarm system, the power consumption has to be minimized as much as possible. Therefore the RF microcontroller is kept in sleep mode when no PIR events are detected. To entering the low power consumption stage, the management of the event has to be successfully completed. The selection of sleep mode, instead of deep sleep, is preferred because important functionalities are running during low power stage, and they can issue a wake up event to the CPU.

System supply voltage is driven by the most demanding device, in this case the PIR sensor, where the minimum supply voltage required is 3 V, therefore a good trade-off for meeting both voltage level with enough energy capability and minimizing mechanical occupancy is 2 AAA batteries. By using a boost regulator, the battery life time can be extended by regulation of the voltage at 3 V while the battery

is being discharged. Even if it is out of the scope of this document, it would be possible to reduce the system power consumption removing all the LEDs since a feedback on Sigfox is always available.

Here a quick calculation of battery life is proposed considering the Sigfox transmission need.

Table 1.

| | Value | Unit |
|---------------------------|-------|-------------|
| Sigfox transmission | 51 | mA |
| System in sleep mode | 91 | μA |
| Battery capability | 1500 | mAh |
| Time for a transmission | 6 | sec |
| Number of Sigfox messages | 140 | Msg per day |

Considering the worst case scenario (see Table 2):

Table 2.

| | | | | | | |
|--------------------------|------|-----|-------|-----|---------|-------|
| 2 AAA Alkaline Capacity | 1500 | mAh | 3600 | s/h | 5400.00 | C |
| Sigfox TX (140 messages) | 51 | mA | 6 | s | 42.84 | C/day |
| Sleep Charge per day | 91 | μA | 85560 | s | 7.79 | C/day |
| OOB frame transmission | | | | | 0.28 | C/day |
| Total Charge Consumption | | | | | 50.91 | C/day |
| Battery Life | | | | | 106.08 | day |

AND9675/D

Mechanical Aspects

The selection of the mechanical enclosure is another critical point that will drive next phases. A very basic housing has been selected for this project: rectangular section with enough height to host AAA batteries, lens and PCB thickness. Moreover PCB thickness is a constraint

coming from the Sigfox reference design. The housing length is selected in order to host the battery and the antenna length.

As soon as a box has been selected, its drawing will drive the PCB shaping on which the HW design has to properly fit.

HW DESIGN

The schematic has been a relative simple task since it was generated by merging two existing schematics. Therefore it has been used.

- The reference design for the radio part [3];
 - The schematic developed for PIR sensor controller application note, refer to [4].
- More info regarding the controller can be fetched in [5][6].

For finalizing the schematic following the concept described in the previous chapter, some HW modifications were needed to the existing schematics:

- A connection between the PIR output controller to an IO of the RF micro (the selected IO has to be able of interrupt generation for waking up the micro from sleep mode).
- A green LED to a micro IO.
- A power switch for turning on the alarm system only when is really needed and preserve battery life.

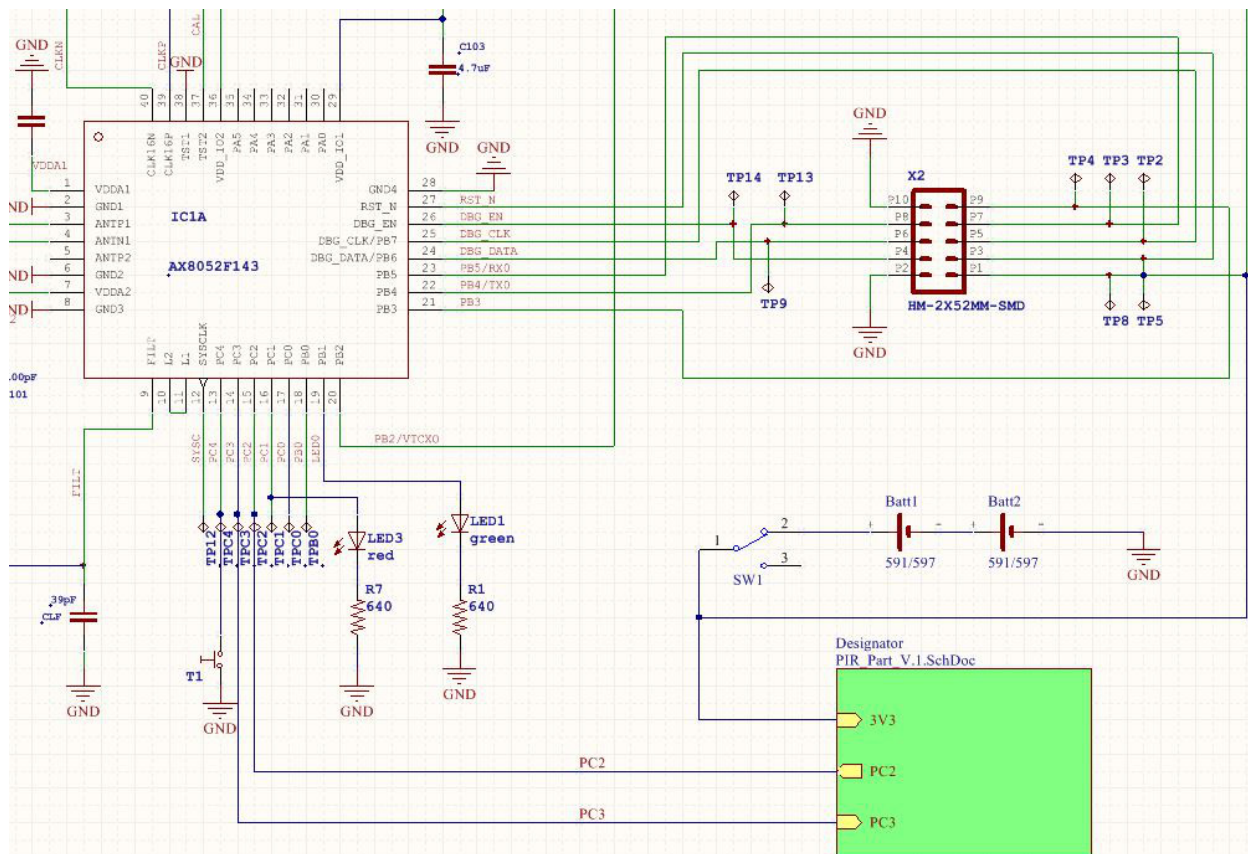


Figure 4. Schematic Modifications for Alarm System

In order to avoid potential issues on the RF side, here is a list of hints for the PCB layout:

1. RF performances are affected by the ground plane design, it is necessary to maximize his size, it has

to be present both on top and bottom side and keep the two sections connected.

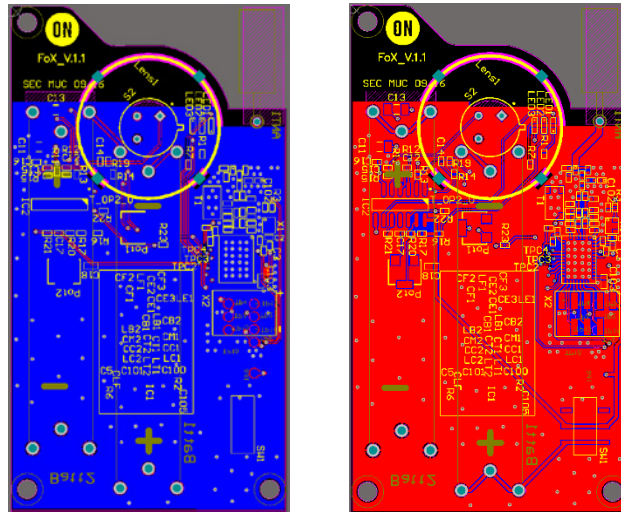


Figure 5. Bottom and Top Ground Plane of Alarm System

Figure 5 is showing the shape of the ground plane of the alarm system. As it is clearly visible, it is spread over the entire PCB and it is surrounding all the components except the antenna portion. Top and bottom are connected through the ground pad of the battery connector.

Having such a big plane, it is important to add thermal via in order to make the PCB assembling easy especially for

components that foreseen big contacts, like battery holders. The RF portion doesn't have to be modified; its portion is already provided with the necessary thermal via if the reference design is followed.

2. The antenna has to be ground plane free.

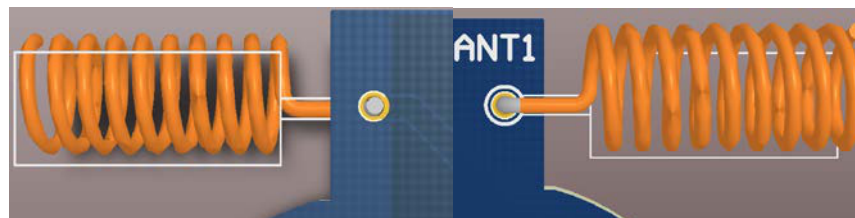


Figure 6. Top and Bottom Side of the Spring Antenna

Figure 6 is showing the connection of the Antenna to the top side. The antenna selected is a single ended antenna. The track is coming from the matching impedance section. In order to avoid any potential short circuit issue between antenna pad and ground plane, the distance between them can be increased. Mostly it is depending on the PCB manufacturer capabilities to achieve such challenging clearance.

Antenna has to be selected according to the communication specification: carrier frequency at 869 MHz

and matching impedance of 50 Ω . It has been here selected a spring antenna, this is a good compromise between costs and performances (2.1 VSWR).

3. PCB has to be 1 mm thick made by FR4.

Once layout and BOM are finalized, a prototyping phase can be performed for validation.

SW DESIGN

The high level behavior of the SW is here proposed:

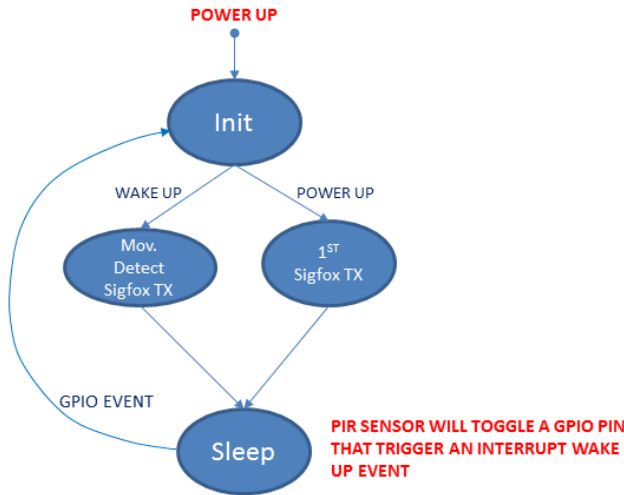


Figure 7. SW Flow Diagram, High Level

Preliminary Info

The SW design phase can start together with the HW design phase.

In order to write a SW for the AX8052F143 SoC it is recommended to use a specific IDE environment. It can be downloaded from the ON Semiconductor website; a link for selecting the suitable IDE according to the available OS is provided in [7].

The installed AS8035-IDE software package in [7] is so called “CodeBlocks”, an open source, cross platform IDE. The reference version for the development is 1.17. This version is pre-configured to properly operate the RF micro, thus compiler, linker and debugger doesn’t require any additional setup, as long as it is enough the free compiler provided, SDCC. However the Alarm system project is developed using a different compiler, IAR, therefore it is required to procure a license for 8051 core from other vendors. In this application note it is present a dedicated section for setting up the IDE environment for using IAR compiler.

Since this development it is based on Sigfox communication and since it is required to use only a microcontroller in the application, a specific SoC it is needed: AX-SFxx-API. This chip is not provided with Sigfox firmware stack, but any registered customer can download from On Semiconductor homepage.

The API acronym means Application Programming Interface, once downloaded the firmware, the SIGFOX library will be available and by means of functions calls, it will be possible to start SIGFOX communication with ease.

Starting with an Example Project: Toggle an LED by Wakeup Time Event

In order to ease the understanding of the Alarm system application, it is better to start from an example project: toggling an LED when a wake up time event is issued and as soon as the task is completed put the micro into sleep mode. This application is completely developed by means of ready to use API functions. To get the original source code of the PIR Alarm System please contact your local sales support at www.osnemi.com.

Sleep Mode

When this low power mode is entered, not all the microcontroller peripherals will be turned off. Only the GPIO and the System Controller are kept awake. Basically the system controller will allow to have one of the two wakeup timer resources always running, thus as soon as an interrupt is generated by this HW resource, the micro is forced to wake up and the program counter will start fetching instruction from location 0 of the program memory. Sleep mode can be considered as an hardware reset and the info regarding what caused the wakeup can be retrieved reading the PCON register. For more info, see Table 21 in [2].

As said, restoring the micro is depending on the capability of generating timer interrupt, thus the wakeup timer header file `libmftimer.h` must be included in the project.

Wake Up Timer: Time Event

The system controller manages two wake-up timers. They work at different frequencies but the behavior is the same. Their frequency is selected at initialization stage via SW.

`wtimer0` is recognized as the low power timer since it is connected to the low power oscillator, therefore is working at lower frequency. Both this example and the Alarm system application have been initialized with a frequency of 640 Hz. This is the minimum frequency that can be achieved and the supply current required by `wtimer0` is around 210 nA, but this value represents only the low power oscillator need.

When the `wtimer` skeleton code is used in an application, even if no events are scheduled in the SW, there will always be a wake up event generated by the `wtimer0` because of implementation of the `wtimer` strategy. This time event is fixed at 96 seconds. Of course it will be more visible when the micro is holding a low power mode, and the supply current is monitored.

The skeleton code consists of:

1. Wake up timer hardware initialization: selection of clock source;
2. Wake up timer software initialization;
3. Wake up timer call to the consumer queue;

- Wake up timer send to idle mode, sleep mode and waiting for a wake up time event.

Here the code for toggling the RED led and skeleton code is proposed.

This SW is written considering the Sigfox reference design and can be used without any issue with the mini-dvk evaluation board or with the Sigfox goodie PCB presented in the previous chapter “HW Design”.

Include

```
#include <ax8052.h>          /// Register declaration for the RF micro
#include <libmftypes.h>      /// libmf datatype definition and convenient function
#include <libmfflash.h>      /// flash memory management, like calibration
#include <libmfwtimer.h>     /// wake up timer structure definition
#include <libminikitleds.h>  /// led management according to the mini-dvk
```

Initialization

```
uint8_t _hw_init_AX8052(void)
{
    DPS = 0;          /// Data pointer select for XRAM reading
    wtimer0_setclksrc(CLKSRC_LPOSC, 1);  /// Hardware initialization of wtimer0
    wtimer1_setclksrc(CLKSRC_FRCOSC, 7);  /// Hardware initialization of wtimer1
    PORTA = 0xFF;      /// When 1, pull up resistor enabled.
    PORTB = 0xFD | (PINB & LED_MASK);
    PORTC = 0xFF;
    PORTR = 0xCB;
    DIRA = 0x00;        /// When 1, pin direction output
    DIRB = 0x06;        /// PB1 = LED, PB2 = TCXO ON/OFF
    DIRC = 0x00;
    DIRR = 0x15;
    ANALOGA = 0x18;      /// When 1, PORTA pin set to Analog
    MISCCTRL |= 0x02;    /// Its selection depends on the HW used.
                        /// if bit 1 is set (value = 0x02) =>
                        /// NO crystal connected to PA0,PA1.
    EIE = 0x00;          /// Peripheral interrupt valid through EA
    E2IE = 0x00;         /// Additional peripheral interrupt
    IE = 0x00;           /// Interrupts that don't require EA enabled
    GPIOENABLE = 1;      /// Enabling the IOs.
    if (PCON & 0x40)     /// Checking bit 6 for reset condition:
                        /// if 1, wake up from sleep or deep sleep
        return 1;
    return 0;            /// if 0, wake up from first power up.
}
```

In the main it is described how to call the remaining wtimer functions:

Main loop:

```

void main(void)
{
    _hw_init_AX8052();           /// micro HW initialization
    flash_apply_calibration();    /// fetching data from the flash CALIB section for
                                /// initializing correctly the HW
    CLKCON = 0x00;               /// CPU Clock configuration.
    wtimer_init();               /// Initialization of the wake up timer HW resource
                                /// and enabling wakeup interrupt
    if (!(PCON & 0x40))          /// At the first power on of the board, the "if" body
    {                             /// is executed.
        /// A wtimer element of a wtimer0/1 queue is defined by a wait time and a function call.
        /// wtdesc is a descriptor of the queue defined as: struct wtimer_desc __xdata wtdesc;
        wtdesc.time = 320;       /// wait time in timer0 tick before Fnc2call execution.
        wtdesc.handler = Fnc2call; /// Fnc2call is the address where the function is written.
        wtimer0_addrelative(&wtdesc); /// adding the wtdesc element to the wtimer0 queue
    }
    EA = 1;                      /// Enabling global interrupt
    PCON = 0x0C;                 /// Reteining both XRAM location
    for (;;) {                   /// main loop
        wtimer_runcallbacks();    /// when wait time is over, wtdesc handler is performed
                                /// fetching a queue element means remove it. If it is ///de-
                                /// sired to execute it multiple time
                                /// it has to be instatiate again.
        EA = 0;                  /// Disabling global interrupt
        {
            uint8_t flg;
            flg = WTFLLAG_CANSTANDBY; /// setting a flag for requesting to enter in standby
            flg |= WTFLLAG_CANSLEEP;  /// and requesting to sleep
            wtimer_idle(flq);          /// wtimer_idle will choose what low power status to enter.
        }
        EA = 1;                  /// if in stanby, micro will recover from here
                                /// otherwise it will start over from _hw_init_AX8052().
    }
}

```

All written so far doesn't explain how to achieve the goal:
toggle the red led every 200 ms, this is done in the Fnc2call
handler.

```

/// Body of the handler related to the queue element wtdesc.

void Fnc2call(struct wtimer_desc __xdata *desc)
{
    desc;                /// avoiding a warning when compiling
    wtdesc.time += 320;    /// re-definition of the queue element.
                          /// it is not needed to initialize the handler
                          /// again since no-one canceled it.

    wtimer0_addabsolute(&wtdesc);    /// adding the element on the wtimer0 queue
    led0_toggle();    /// toggling the RED led
}

```

A timed queue element can be added using two different approaches, by absolute value or by relative value. Also the method used for adding the element to the wtimer0 queue is different: wtimer0_addabsolute, wtimer0_addrelative.

The methods are slightly different; the relative one will guarantee more precise result since it is corrected with the actual time elapsed. On the other hand, the absolute method is relying on the variable time, which is a field of the descriptor, and it is not taking care about the real elapsing time. The execution time difference is caused by the delay between the event trigger generation and the function callback call. Both the method cannot be considered strictly real time event because of this time delay.

Remark

Although the wtimer0 is working at 640 Hz, the CPU clock is set at a different frequency, CLKCON is indeed initialized with 0 that represents the internal Fast RC oscillator with a frequency of 20 MHz. Therefore the latency for executing the callback task is not dramatically high but a delay will be anyway present.

Ascertain that a delay is present by implementation; the just introduced software structure will guarantee a very limited stack memory allocation for interrupt management. In fact the function calls are executed in main loop. What is actually left to the wtimer interrupt is:

- Comparison between queue element timings versus the wtimer0 counter.
- Any queue element that doesn't have to wait any longer will be added to the pending queue.
- The pending queue is then managed in the wtimer_runcallbacks function.

Wake Up Timer: Asynchronous Event

Besides the timed event, already discussed, the wtimer structure manages also asynchronous event. An asynchronous event is an unpredictable change in any of the monitored peripherals; for example pushing a button is an asynchronous action performed by an external user on a pre-defined IO where the button is linked to. Asynchronous means also that the micro should respond to this event immediately performing some specified set of instructions. Since the intention is still to keep the interrupt routine as small as possible, the event will cause the generation of a queue element that will be added at the first position of the pending queue, therefore they will be called by the wtimer_runcallback function immediately.

What is actually needed is specify the function to be called and pass its handler to the callback queue element. The following SW shows how reacting to a push button event turning on the red led and, inside this function call, trigger a timed event for blinking the red led for a specified number of times.

The code for adding an asynchronous call is pretty much similar to what described in the previous section, but the type qualifier of the queue object is different. On the other hand, the HW initialization function is the same of the previous example.

Initialization

```
uint8_t _hw_init_AX8052(void)
{
    DPS = 0; // data pointer selection, XRAM access
    wtimer0_setclksrc(CLKSRC_LPOSC, 1); // clock selection for wtimer0
    wtimer1_setclksrc(CLKSRC_FRCOSC, 7); // clock selection for wtimer1
    PORTA = 0xFF; // enable pull up on PORTA
    PORTB = 0xFD | (PINB & LED_MASK); // enable pull up but taking care about
    // B1 status when recovering from sleep
    PORTC = 0xFF; // enable pull up
    PORTR = 0xCB; // not really needed for this example
    DIRA = 0x00; // all input
    DIRB = 0x06; // B1 and B2 output
    DIRC = 0x00; // all input
    DIRR = 0x15; // radio init
    ANALOGA = 0x18; // A[3,4] analog
    MISCCTRL |= 0x02; // no crystal on A[0,1]
    EIE = 0x00; // Peripheral interrupt valid through EA
    E2IE = 0x00; // Additional peripheral interrupt
    IE = 0x00; // Interrupts that don't require EA enabled
    GPIOENABLE = 1; // Enabling the IOs.
    if (PCON & 0x40) // Checking bit 6 for reset condition:
    // if 1, wake up from sleep or deep sleep

    return warmstart;
    return coldstart; // if 0, wake up from first power up.
}
```

Let's just remark here that when the micro is waking up from sleep mode, the SW will start again from the beginning of the initialization function and, in order to avoid unwanted change on output pins (like the red led of this case) it is needed to read their actual status and considering in the IO

initialization. During sleep mode IOs are frozen, this is why it is present | (PINB & LED_MASK).

The SW will require some definition and global variable declaration in order to work properly.

```
/// --- List of constructor for wtimer queue
struct wtimer_callback __xdata aCallBk_pButt; // async push button queue element descriptor
struct wtimer_desc __xdata tCallBk_led; // timed queue element descriptor
/// --- Definition for Interrupt management
#define INPUT_INTCHG INTCHGC // Interrupt on Port C change
#define INPUT_PIN PINC // which pin to monitor
#define PB_MASK 0x10 // mask for push button reading, C[4]
#define LED_MASK 0x02 // red Led position on B[1]
/// --- Time constant based on wtimer0 tick: frequency @ 640hz
#define TIME_FIRST 640 // wait time before timed queue blinking
#define TIME_BLINKING 128 // half period for blinking
#define BLINK_REPET 40 // number of times the led will blink
/// --- Function prototype
void PusButtonEvent(struct wtimer_callback __xdata *callbk); // asynch event
void RedLedToggling(struct wtimer_desc __xdata *desc); // timed event
```

Main loop:

```

void main(void)
{
    static uint8_t __data pinState = 0xFF;  /// default value, all pin pull-up
                                           /// __data: pinState allocated in IRAM
                                           /// Internal RAM is the fastest available.

    _hw_init_AX8052();  /// micro HW initialization
    flash_apply_calibration();  /// fetching data from the flash CALIB section for
    CLKCON = 0x00;  /// Clock configuration.
    wtimer_init();  /// Initialization of the wake up timer HW resource
                  /// and enabling wakeup interrupt
    EA = 1;  /// Enabling global interrupt
    INPUT_INTCHG |= PB_MASK;  /// Enabling interrupt on change
    PCON = 0x0C;  /// keep both XRAM block
    for (;;)  /// main loop
    {
        wtimer_runcallbacks();  /// execution of the queue element
        /// --- Start of push button event management
        EA = 0;  /// disable global interrupt to make the current code atomic
        {
            uint8_t inPin;
            {
                /// the declaration with this brackets means...
                uint8_t pB;  /// that pB usage is limited to this section.
                pB = INPUT_PIN;  /// reading the PIN C status
                inPin = pinState & ~pB;  /// if no change on C4, PINC is 1, inPin = 0
                pinState = pB;  /// pinState is 0
            }
            if (inPin & PB_MASK)  /// as long as C4 is not changing, "if" is 0
            {
                /// when C4 is 0, the body is executed
                EA = 1;  /// re-enabling global interrupt
                aCallBk_pButt.handler = PusButtonEvent;  /// passing the handler
                wtimer_add_callback(&aCallBk_pButt);  /// adding to the pending queue
                continue;  /// jumping back to the "for" instruction and
                          /// performing new check until the C4 state is
                          /// set back to 1 again.
            }
            IE_3 = 1;  /// Enabling GPIO interrupt. Wakeup the micro
                     /// when a new button event is issued.
        }
        /// --- End of push button event management
    }
}

```

```

{
    uint8_t flg = WTFLAG_CANSTANDBY;    /// flag definition for low power mode stage
    flg |= WTFLAG_CANSLEEP;             /// asking also to go to sleep
    wtimer_idle(flg);                   /// wtimer will take care about going to sleep
}

IE_3 = 0;                             /// in case the micro is in standby, the micro will keep
                                       /// executing from here. The GPIO interrupt is removed and
                                       /// reading the button state will be performed correctly.

EA = 1;                               /// enabling of global interrupt
}
}

```

In order to read a push button, this SW makes use of two main instructions, disable of the global interrupt flag and `continue` instruction. A `continue` statement prevent the execution of the next portion of the code. It is actually tell the program counter to jump back to the nearest loop condition, in this case the `for` loop.

In order to enable the `if` body where `continue` is introduced, a pin change has to happen. The code will keep

looping between `for` and `continue` until a second pin change is happening.

When working with push buttons, they introduce spurious on/off repetitions, normally identify as bouncing effect. In order to avoid those glitches, a flag could be defined that will prevent the SW to execute multiple time the `wtimer_add_callback` instruction.

Regarding the function callback body:

```

/// -----
/// Body of the handler related to the asynchronous queue element
/// called when the push button event is recognized
/// -----
void PusButtonEvent(struct wtimer_callback __xdata *callbk)
{
    led0_set();                /// set the red led
    if (tCallBk_led.handler == 0)    /// if there is no handler specified in queue
    {
        tCallBk_led.handler = RedLedToggling; /// passing the handler
        wtimer0_remove(&tCallBk_led);    /// ensuring that no timed queue element is present
        tCallBk_led.time += TIME_FIRST;    /// definition of the timed-queue element.
        wtimer0_addabsolute(&tCallBk_led);    /// adding to the wtimer0 queue again
    }
}

/// -----
/// Body of the handler related to the asynchronous queue element
/// called when the push button event is recognized
/// -----
void RedLedToggling(struct wtimer_desc __xdata *desc)
{
    static uint8_t cntBlink = 0;    /// local counter var, keep the value
    /// since is defined static

    led0_toggle();                /// request of led toggling
    cntBlink = cntBlink + 1;        /// updating the counter
    if (cntBlink <= BLINK_REPET)    /// check if the counter is below threshold
    {
        tCallBk_led.time += TIME_BLINKING;    /// passing the new time
        wtimer0_addabsolute(&tCallBk_led);    /// add a new entry in the queue
    }
    if (cntBlink > BLINK_REPET)    /// when counter reach the threshold
    {
        led0_off();                /// ensuring the led is off
        tCallBk_led.handler = 0;    /// removing the handler
        cntBlink = 0;                /// resetting the counter
    }
}

```

This approach can be easily applied to the Alarm system developed for the Sigfox Goodie replacing the push button event with the pin that is coming from the NCS36000 controller.

Alarm System SW Behavior

The Goodie SW makes extensively use of the same SW structure analyzed in the previous paragraphs. Multiple timed queue elements are defined in order to achieve the concept description provided in the previous chapter “Concept Analysis”, therefore there will be:

- A queue timed element for managing the 1 hour event for triggering new Sigfox transmission,
- A queue timed element for blinking the green led if any error on the Sigfox communication,
- A queue timed element for green led management when no error on the communication,
- A queue timed element for enabling again the reading of the PIR sensor after sending a Sigfox message. 20 seconds of blind time is foreseen for avoiding the

management of movement detection issued by the PIR sensor due to EMI noise during radio transmission.

- A callback on asynchronous event for managing the detection of a movement.

The only added complexity is the management of the Sigfox communication and the interaction with the API stack. Moreover, by the time when this SW has been written some additional precautions have to be taken for letting the radio go to sleep properly. This stage is needed to achieve the low power condition required, supply current fixed at 90 μ A.

ON Semiconductor provides convenient API functions to easily set up a Sigfox transmission. Those are the functions used in the SW:

1. `ONSEMI_initialize(ONSEMI_INIT_CALIBRATE_LPOSC |`
`ONSEMI_INIT_CALIBRATE_FRCOSC);`
2. `ONSEMI_send_frame(transmit_data, 7,`
`0, false);`
3. `ONSEMI_cansleep();`

The first one is needed for initializing the HW properly. It will perform something similar to the `_hw_init_AX8052` plus the radio part.

The second one is actually responsible for the Sigfox communication. This function is blocking, therefore the SW execution is stuck here until the transmission is over.

`transmit_data` is the array where the message is stored, it will be sent to the Sigfox network, 7 is the number of byte in TX, 0 the number of bytes in RX, if downlink is foreseen, but in this case its flag is `false`. More detail about how it works can be found in the `onsemi_api.h` header file.

When the communication is over, the radio part is no longer needed and the send frame should complete his task setting the radio into deep sleep mode. Actually two more instructions have been added to ensure that this low power condition is reached:

- `ax5043_enter_deepsleep();`
- `SCRATCH3 = 0x01; ///does`
`POWER_STATE_RADIO =`
`POWER_STATE_RADIO_DEEPSLEEP;`

The third instruction is needed for checking if the microcontroller can go to sleep, therefore it is used in the main loop before the `wtimer_idle` call.

Setup Build Option for the Project

In case of unexpected building message errors will be shown on the Code::Blocks logs window, it is maybe possible that the build option are not properly set. In this case the easiest way to find out what is going wrong is generating a blank new project selecting the `Axsem AX8052` project template among all the available options. After entering the desired project name, Code::Blocks will ask to select the compiler.

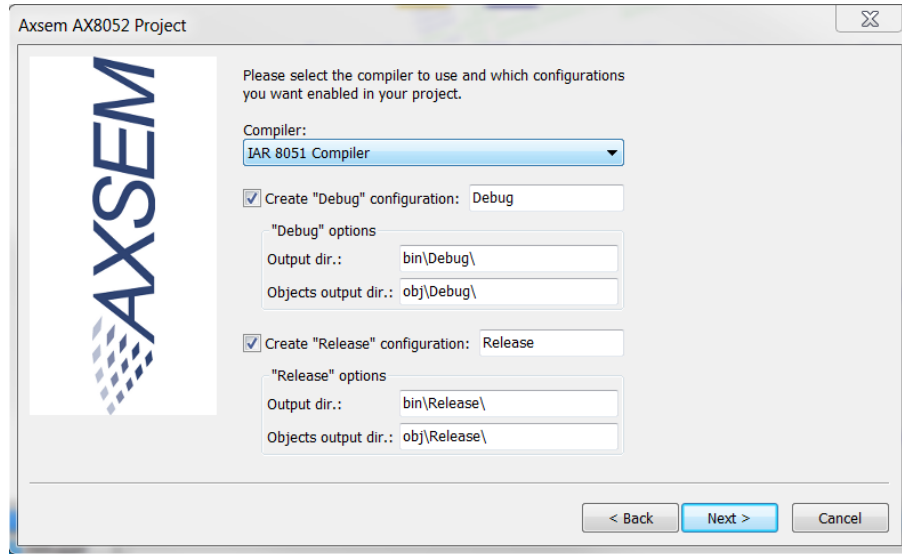


Figure 8. Compiler Selection

Pick the one that was causing trouble on the existing project and press “Next”.

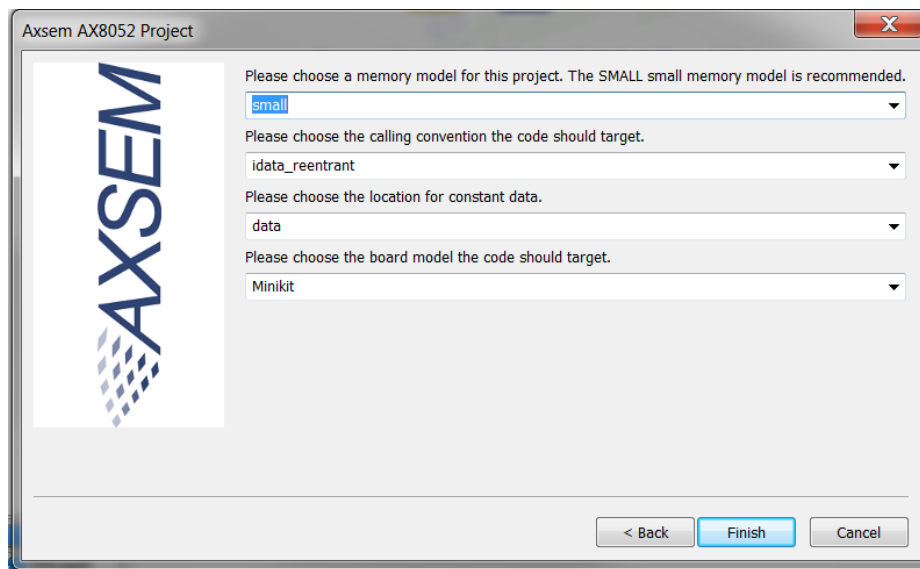


Figure 9. Board Selection

Besides the board selection, last entry of the current window, leave the default values unchanged, and then press “Finish”.

An example project is created.

Try to build it and verify if errors are shown in the Build Messages window.

If you get an error like this, most likely it is just a matter of wrong location for addressing library files.

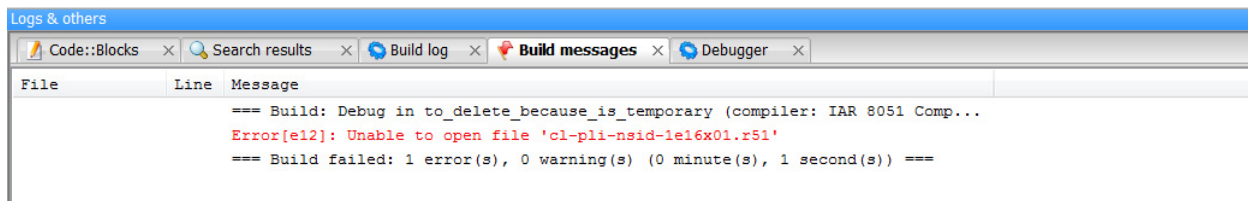


Figure 10. Build Failure Due to Wrong Link to Library File

All is needed is navigate into the “Build options” and update the link libraries location.

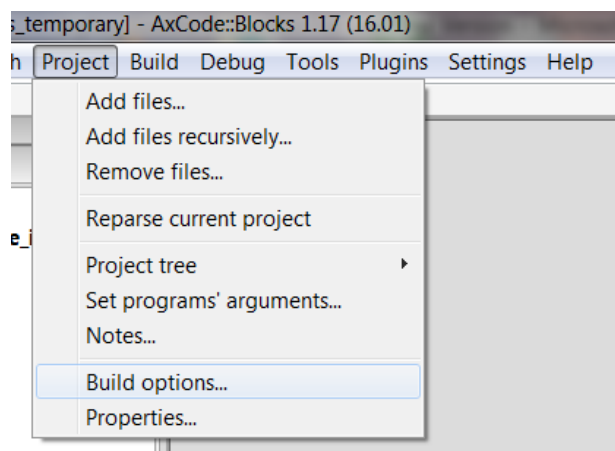


Figure 11. Build Options Location

When opening the window, be sure to select the project name on the left column in order to have a unique option set

both for Debug and Release version. If needed, further

options can then be added separately under Debug and Release version.

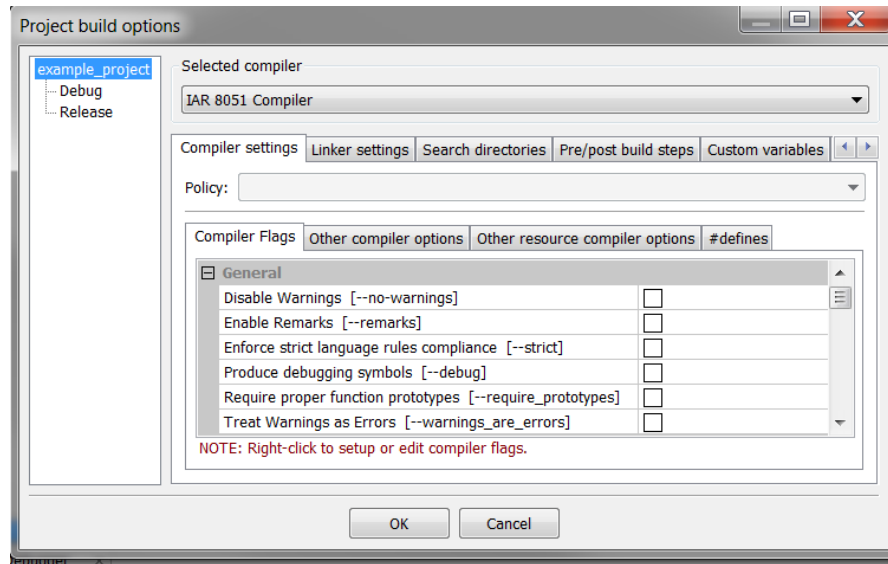


Figure 12. Build Options

Go to linker setting and fix the libraries location editing the path according to the user folder.

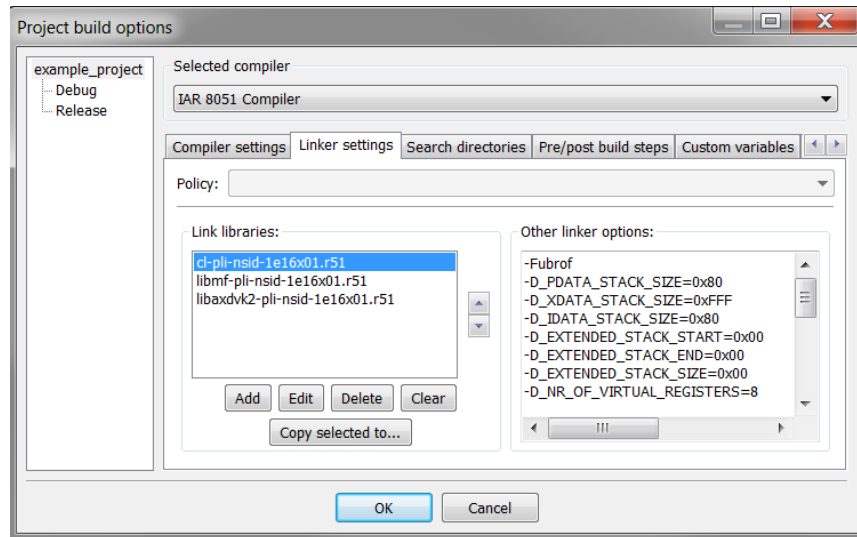


Figure 13. Link Libraries Location

When done, no further build error messages should be displayed on the Log window.

```

IAR Universal Linker V6.4.1.81
Copyright 1987-2015 IAR Systems AB.
2 656 bytes of CODE memory
14 bytes of DATA memory (+ 30 absolute )
26 bytes of XDATA memory (+ 113 absolute )
140 bytes of IDATA memory
8 bits of BIT memory (+ 2 absolute )
Errors: none
Warnings: none
Output file bin/Debug/example_project.ubr size is 41.06 KB, link time 00:00:00.213
Process terminated with status 0 (0 minute(s), 0 second(s))
0 error(s), 0 warning(s) (0 minute(s), 0 second(s))
  
```

Figure 14. Compiled Successfully

DEVICE CERTIFICATION

The alarm system was developed to demonstrate a Sigfox device as a wireless application for the mass production, thus two types of certifications are necessary to launch the Sigfox device as “off the shelf product” on the market.

1. Sigfox SA requires **Sigfox Certification** to access the Sigfox network with following certification types:
 - Sigfox Verified™ certification
 - Sigfox Ready™ certification
2. **Regulatory Compliance Certification** governed at each country or region before putting a device into its market
 - CE Mark in Europe, FCC in North America; ARIB in Japan, etc.

In general, the Sigfox certification is carried out by Sigfox or its accredited test houses and will not include regulatory compliance certification like CE marking in Europe or FCC in the USA. It is recommended to consult Sigfox to plan the Sigfox certification. Detailed information about the Sigfox certification requirement and process can be found in [10]. The Sigfox certification plan is illustrated in the following certification flow:

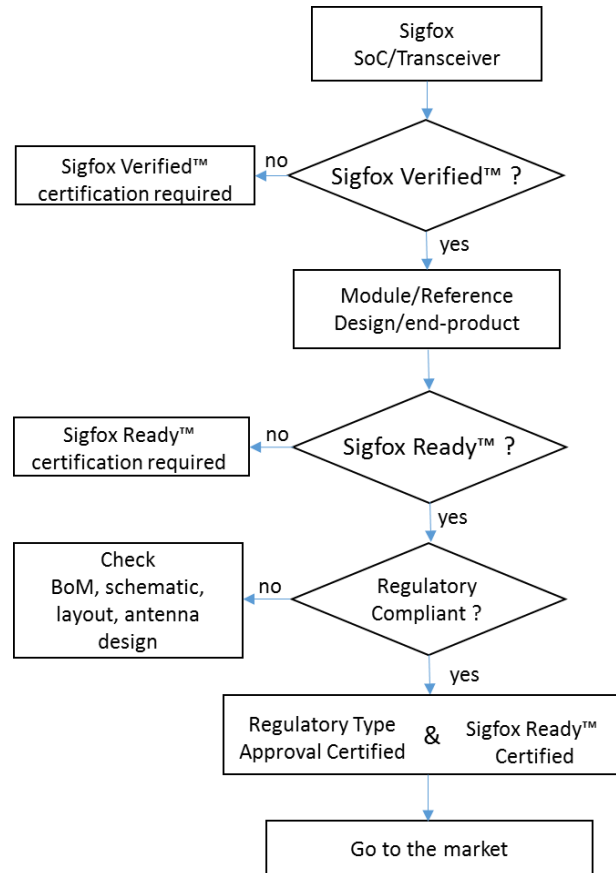


Figure 15. Sigfox Certification Plan

Sigfox Certification

The Sigfox Verified™ certification qualifies the Sigfox protocol and the RF modem performance and Sigfox Ready™ certification qualifies the end product.

The alarm system with the AX8052F143 SoC is already Sigfox Verified and Sigfox Ready certified device.

The AX-SFxx-API-y SoCs with respective development kits (DVK) are already Sigfox Verified and Sigfox Ready certified (see Table 3) so that the customer, OEM's who develop the end product for the mass market based on the reference design of the DVKs can reuse the Sigfox Ready certification under the similarity rules described in [3].

Table 3. OVERVIEW OF AX-SFxx SIGFOX CERTIFICATION

| Radio Configuration Zone (RCZ) | Frequency | AX8052F143 + API Sigfox Verified | Development Kit Sigfox Ready |
|--------------------------------|-----------|----------------------------------|------------------------------|
| RCZ1: EMEA | 868 MHz | AX-SFEU-API | DVK-SFEU-API-1-GEVK |
| RCZ2: US/Latin America | 902 MHz | AX-SFUS-API | DVK-SFUS-API-1-GEVK |
| RCZ3/7: Japan/Korea | 923 MHz | AX-SFJK-API | DVK-SFJK-API-1-GEVK |
| RCZ4: Australia/NZ | 915 MHz | AX-SFAZ-API | DVK-SFAZ-API-1-GEVK |

The Regulatory Compliance Standards

Since the Alarm System is intended to be used in Europe, the ETSI is the controlling organization that regulates the usage of RF equipment and the EN 300 220-1 is the reference standard.

In Europe the access to the 863-870 MHz is subject to ETSI 300-220 regulation and is under scope of RED. In the

United States, the Federal Communications Commission (FCC) the access to the 902-928MHz band, the FCC part 15-247. Generally, AX-SFxx-API-y SOC is compliant to the each regulatory compliance standards within the RCZ.

The following table shows an overview of the SIGFOX AX-SFxx-API-y SOC and relevant regulatory compliance standards worldwide.

Table 4. SIGFOX AX-Sfxx REGULATORY COMPLIANCE STANDARDS

| Radio Configuration Zone (RCZ) | Frequency | Regional Regulatory Compliance | Development Kit |
|--------------------------------|-----------|--------------------------------|---------------------|
| RCZ1: EMEA | 868 MHz | CE (ETSI) | DVK-SFEU-API-1-GEVK |
| RCZ2: US/Latin America | 902 MHz | FCC | DVK-SFUS-API-1-GEVK |
| RCZ3: Japan | 923 MHz | ARIB STD-T108 | DVK-SFJK-API-1-GEVK |
| RCZ4: Australia/NZ | 915 MHz | AS/NZS 4268 | DVK-SFAZ-API-1-GEVK |

AX-SFxx-API-y is designed and specified for SIGFOX wireless application using the unlicensed ISM/SRD band worldwide. In order to produce a compliant application or products, the vendor or manufacturer should consider the following guidelines for the final regulatory compliance certification of respective region.

- Consider the schematic, layout design and external component selection provided by the reference design board of ON Semiconductor
- Place any decoupling capacitors close to the IC
- The printed circuitry board should have a solid ground plane in the RF section
- Check the RF output matching network recommend in the datasheet [2] with the optional filter stage to suppress TX harmonic
- Check the proper RF configuration mode with AT command described in [2]

- Ensure ability of RF measurement equipment like Spectrum Analyzer/Signal Generator
- Ensure minimum of reflection and interference-free test environment

A sanity check or a pre-testing prior to the accredited regulatory compliant test procedure is recommended to ensure that the device under test operate properly and meets the compliant test requirements under the specified test conditions.

The figure below shows a typical conducted transmitter test setup as an example for the sanity check according to the EN 300 200-1 carried out at EMC Testing Laboratory at Slovak University of Technology with AX-SFEU Sigfox shield.

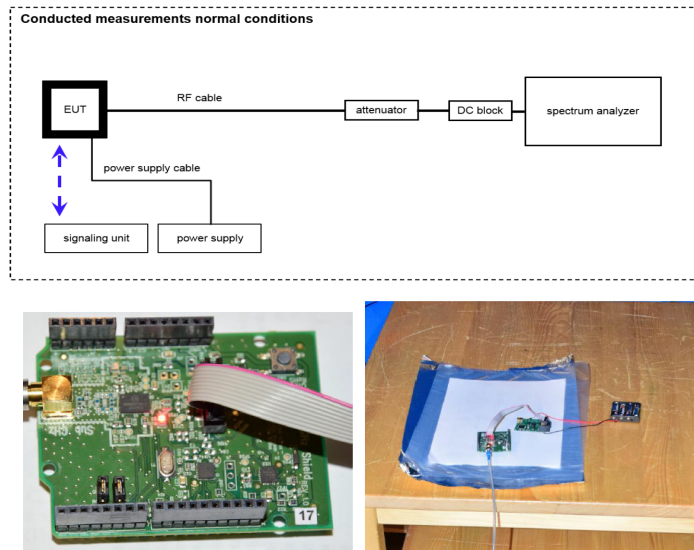


Figure 16. Typical Conducted Transmitter Setup

Conditions of the measurement:

The measurement place was arranged according to EN 300 220-1(Sub-Clause 7.1). The EUT was in continuous unmodulated mode.

Rated value: 868.13 MHz
Max frequency error: ± 12.5 kHz

Measurement results:

Measured value: 868.13036 MHz
Frequency error: 0.36 kHz

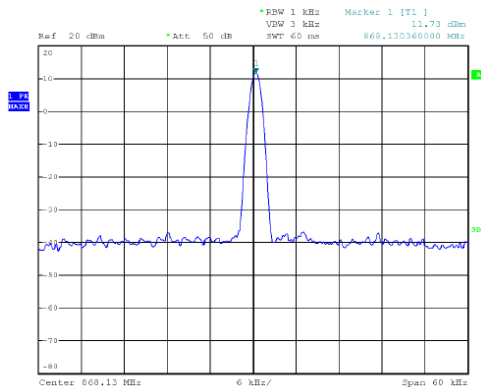


Fig. 2: The measured value of frequency error

Conclusion: The frequency error of the EUT is below the limit of the standard EN 300 220-1 V2.4.1:2012.

Figure 17. Typical RF Carrier Frequency Monitored with Spectrum Analyzer

| No. | Measurement | Standard | Conclusion | Page |
|-----|---|-------------------------------|------------|------|
| 1 | Frequency error | EN 300 220-1 (Sub-Clause 7.1) | PASS | 4 |
| 2 | Average power | EN 300 220-1 (Sub-Clause 7.2) | PASS | 5 |
| 3 | Adjacent channel power | EN 300 220-1 (Sub-Clause 7.6) | PASS | 6 |
| 4 | Unwanted emissions in the spurious domain | EN 300 220-1 (Sub-Clause 7.8) | PASS | 7 |

Conclusion: The EUT complies with requirements set by the standard EN 300 220-1 V2.4.1: 2012 for electromagnetic descriptions to the extent of the table 1.

Figure 18. Summary of Sanity Check for TX Test

A typical test description of spurious conducted emission according to the FCC part 15.247 compliance requirement performed at the FCC accredited test laboratory Cetecom in Germany with the US version of AX8052F143 is illustrated as following:

Description:

Measurement of the conducted spurious emissions in transmit mode. The EUT is set to single channel mode. The measurement is repeated for low, mid and high channel.

Settings:

| | Settings |
|-------------------|--|
| EUT | Set the output carrier channel <ul style="list-style-type: none"> • AT\$IF=902104000 Send modulated (pseudo-)random pattern with -1 and disable pattern testmode with 1 <ul style="list-style-type: none"> • AT\$CB=-1,1 Repeat for mid and high channel |
| Spectrum Analyzer | <ul style="list-style-type: none"> • Span = Low Band Edge 902 MHz; Upper Band Edge 928 MHz • RBW = 300 kHz (f < 1 GHz); = 3 MHz (f > 1 GHz) • VBW = 10/100 kHz (f < 1 GHz) = 1 MHz (f > 1 GHz) • Detector function peak • Trace = max hold • Sweep Time = Auto |

Limits:

20 dB below the carrier signal

Results:

No spurious emission detected below 1 GHz, - 36 dBc above 1 GHz

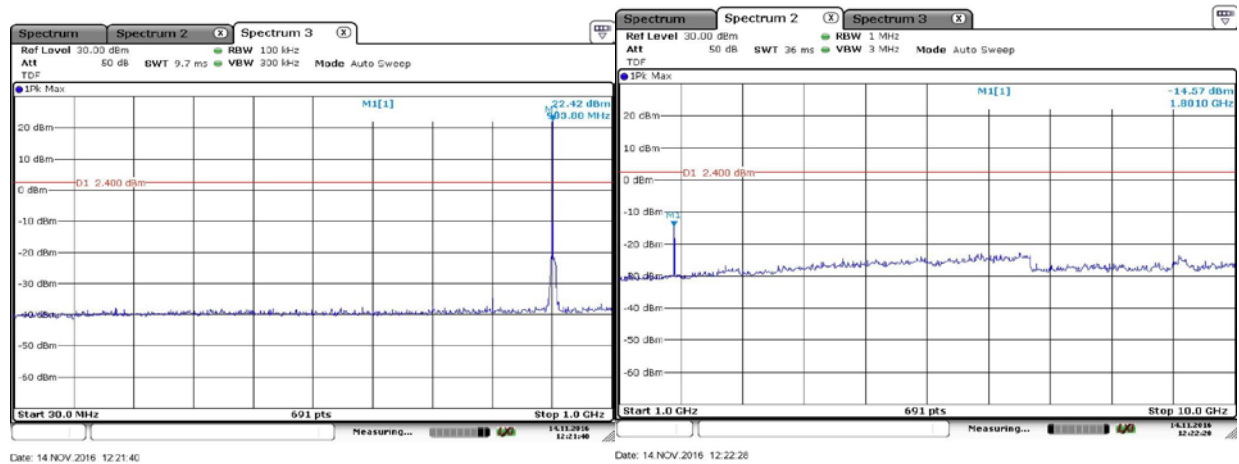


Figure 19. Spurious Conducted Emission Below and Above 1 GHz

The complete regulatory compliant test reports for the AX8052F143 all over by the Sigfox defined Radio Configuration Zone are available in [12].

As above mentioned, the required regulatory compliant standards are treated differently by region or countries. In order to sell the final product within the European market, the final product have to be compliant to all application specific EU directives and standards to get the CE

certification. The alarm system was developed to demonstrate a final Sigfox application and provide design example as reference for the customers who intend to develop a similar application for the market. In this case, the customer need to consider the regulatory compliance standard according to the Radio Equipment Directive (RED) and CE certification for the European market in addition to the Sigfox Ready Certification.

SIGFOX ONLINE PLATFORM

With the development of the PIR Alarm System, a process flow of a Sigfox application from hardware and software development up to the Sigfox Certification and regulatory compliant test scenario has been demonstrated to launch a Sigfox application on the mass market.


The Sigfox online platform Build.Sigfox.com provides further useful information for the Sigfox application developers with following features:

- A guide for each device development step which is continuously updated on user experience
- A single and centralized document resource center, populated and maintained by Sigfox
- Support for device development & tooling
- Access to re-usable designs
- Contextualization of some products and companies of Sigfox Partner Network

REFERENCES

- [1] ON Semiconductor web page regarding the RF micro
<http://www.onsemi.com/PowerSolutions/product.do?id=AX8052F143>
- [2] RF Micro Datasheet: **AX8052F143-D; AX-SFxx** in
www.onsemi.com
- [3] Reference Design for a Sigfox ready board:
DVK-SFEU-1-GEVK Combination AX-Sigfox RefDesign in www.onsemi.com,
<https://resources.sigfox.com/document/sigfox-ready-similarity-rules>
- [4] Schematic for PIR sensor controller:
NCS36000GEVB Schematic in www.onsemi.com
- [5] Regarding the PIR
<http://www.onsemi.com/PowerSolutions/product.do?id=NCS36000>
- [6] Application Note for PIR sensor controller:
EVBUM2304/D
- [7] <http://www.onsemi.com/PowerSolutions/supportDoc.do?type=software&rpn=AX8052F143>
- [8] Libmf (AX 8052 Support Library).pdf
- [9] Standard for RF communication in the European countries: EN 300 220-1
- [10] <http://makers.sigfox.com/getting-started/>;
<https://resources.sigfox.com/document/whycertification>
- [11] DA-00-705A1
- [12] CTC-SFAZ.pdf; CTC_SFJK.pdf; EMCKP2979A
ON Semiconductors_Sigfox_v1-1-0.pdf;
SFUS_FCC_1-2318_16-01-02.pdf

Sigfox is a registered trademark of Sigfox SARL.
Sigfox Ready and Sigfox Verified are trademarks of Sigfox SARL.

ON Semiconductor and  are trademarks of Semiconductor Components Industries, LLC dba ON Semiconductor or its subsidiaries in the United States and/or other countries. ON Semiconductor owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of ON Semiconductor's product/patent coverage may be accessed at www.onsemi.com/site/pdf/Patent-Marketing.pdf. ON Semiconductor reserves the right to make changes without further notice to any products herein. ON Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does ON Semiconductor 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 special, consequential or incidental damages. Buyer is responsible for its products and applications using ON Semiconductor products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by ON Semiconductor. "Typical" parameters which may be provided in ON Semiconductor 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. ON Semiconductor does not convey any license under its patent rights nor the rights of others. ON Semiconductor products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use ON Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold ON Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that ON Semiconductor was negligent regarding the design or manufacture of the part. ON Semiconductor is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:

Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303-675-2175 or 800-344-3860 Toll Free USA/Canada
Fax: 303-675-2176 or 800-344-3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800-282-9855 Toll Free
USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910
Japan Customer Focus Center
Phone: 81-3-5817-1050

ON Semiconductor Website: www.onsemi.com

Order Literature: <http://www.onsemi.com/orderlit>

For additional information, please contact your local Sales Representative