

## Getting started with the Contiki OS/6LoWPAN on STM32 Nucleo with SPIRIT1 and sensors expansion boards

---

### Introduction

Contiki is an open source operating system for the Internet of Things.

STMicroelectronics has developed a Contiki 3.x port for the STM32 Nucleo L1 series equipped with the X-NUCLEO-IDS01A\* expansion boards (sub-1GHz RF communication boards based on the SPIRIT1 transceiver) and optionally with the X-NUCLEO-IKS01A1 expansion board featuring temperature, humidity, pressure, magnetometer and motion MEMS sensors.

This document explains how to get started with this system.

---

# Contents

<b>1</b>	<b>Acronyms and abbreviations .....</b>	<b>4</b>
<b>2</b>	<b>System Overview .....</b>	<b>5</b>
2.1	Features .....	5
2.2	Getting the software package.....	5
2.3	Software package structure.....	6
2.4	Sample applications .....	6
2.4.1	Building Contiki native examples.....	6
2.4.2	Building examples provided by ST .....	8
2.5	Building an overall system with an IPv6 host, an edge router and a sensor node .....	9
2.5.1	Edge (border) router setup .....	10
2.5.2	Wireless sensor node setup .....	10
2.5.3	IPv6 host setup.....	11
2.5.4	Contiki server access (border router) and connectivity test .....	14
2.5.5	Wireless node resources access demo using CoAP .....	15
2.5.6	Wireless node sensor resource access demo with CoAP.....	16
<b>3</b>	<b>System setup guide.....</b>	<b>18</b>
3.1	Hardware requirements.....	18
3.1.1	NUCLEO-L152RE board .....	18
3.1.2	X-NUCLEO-IDS01A* expansion board .....	19
3.1.3	X-NUCLEO-IKS01A1 expansion board (optional).....	19
3.2	Software requirements .....	20
3.3	Board setup guide .....	20
<b>4</b>	<b>Revision history .....</b>	<b>23</b>

## List of figures

Figure 1: Contiki-3x package structure .....	6
Figure 2: Terminal utility showing the running Unicast-sender example .....	7
Figure 3: Terminal utility showing the running Unicast-receiver example .....	7
Figure 4: Terminal utility snapshot when sensor-demo runs .....	9
Figure 5: Overall system architecture .....	10
Figure 6: Adding the network adapter in Windows .....	11
Figure 7: STM32 Nucleo board Virtual COM Port value .....	12
Figure 8: MAC address of the Microsoft Loopback Adapter .....	12
Figure 9: wpcapslip6 terminal window .....	13
Figure 10: tunslip6 terminal window .....	14
Figure 11: ContikiRPL server access – neighbors and routes .....	15
Figure 12: Ping6 command window snapshot .....	15
Figure 13: Copper (Cu) CoAP user-agent GET access to wireless remote node .....	16
Figure 14: Example CoAP GET access to temperature sensor (running sensor-er-rest-example firmware) .....	17
Figure 15: STM32 Nucleo Board .....	18
Figure 16: X-NUCLEO-IDS1A* SPIRIT1 expansion board .....	19
Figure 17: X-NUCLEO-IKS01A1 sensor expansion board .....	20
Figure 18: X-NUCLEO-IDS01A* expansion board connected to STM32 Nucleo Board .....	21
Figure 19: X-NUCLEO-IKS01A1 and X-NUCLEO-IDS01A* expansion boards connected to STM32 Nucleo Board .....	22

# 1 Acronyms and abbreviations

Table 1: Acronyms

Acronym	Description
CoAP	Constrained Application Protocol

## 2 System Overview

### 2.1 Features

The port allows running the Contiki OS 6LoWPAN protocol stack and related applications on a STM32 Nucleo board equipped with sub-1GHz RF connectivity and an optional sensors expansion board.

In particular, the port supports the following boards:

- NUCLEO-L152RE board based on the STM32L152RET6 ultra-low power microcontroller
- X-NUCLEO-IDS01A4 based on sub-1GHz SPSGRF-868 SPIRIT1 module (operating at 868 MHz)
- X-NUCLEO-IDS01A5 based on sub-1GHz SPSGRF-915 SPIRIT1 module (operating at 915 MHz)
- X-NUCLEO-IKS01A1 featuring motion MEMS and environmental sensors (optional)

The following drivers are included:

- LEDs and buttons (user, reset)
- USB
- SPIRIT1 sub-1GHz transceiver
- HTS221, LIS3MDL, LPS25HB, LSM6DS0 sensors

### 2.2 Getting the software package

The port is available in the Contiki master repository in GitHub:

<https://github.com/contiki-os/contiki>

A library of low layer components, including the drivers and Hardware Abstraction Layer based on the STM32 Cube framework is available in a separate repository:

<https://github.com/STclab/stm32nucleo-spirit1-lib>

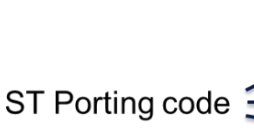
This library is imported as a sub-module of the Contiki repository using standard GIT commands. The online repository can be cloned to a local “contiki” folder with the following command:

```
git clone --recursive https://github.com/contiki-os/contiki.git
```

The content of the separate “stm32nucleo-spirit1-lib” library is automatically stored in the “contiki/platform/stm32nucleo-spirit1/stm32cube-lib” directory.

## 2.3 Software package structure

Figure 1: Contiki-3x package structure



Name	Date modified	Type	Size
apps	7/9/2015 5:19 PM	File folder	
core	7/9/2015 5:19 PM	File folder	
cpu	7/9/2015 5:19 PM	File folder	
dev	7/9/2015 5:21 PM	File folder	
doc	7/9/2015 5:19 PM	File folder	
examples	7/9/2015 5:21 PM	File folder	
platform	7/9/2015 5:18 PM	File folder	
regression-tests	7/9/2015 5:18 PM	File folder	
tools	7/9/2015 5:19 PM	File folder	
.gitignore	7/9/2015 5:19 PM	Text Docu...	2 KB
.gitmodules	7/9/2015 5:19 PM	Text Docu...	1 KB
.travis.yml	7/9/2015 5:18 PM	YML File	5 KB
CONTRIBUTING.md	7/9/2015 5:19 PM	MD File	13 KB
LICENSE	7/9/2015 5:18 PM	File	2 KB
Makefile.include	7/9/2015 5:19 PM	INCLUDE F...	8 KB
README.md	7/9/2015 5:19 PM	MD File	1 KB
README-BUILDING.md	7/9/2015 5:18 PM	MD File	5 KB
README-EXAMPLES.md	7/9/2015 5:19 PM	MD File	7 KB

The ST porting code is included in the following folders:

- **cpu**: this folder contains the processor-related support functions. The code to support the STM32L152 processor is included in the “./cpu/arm/stm32l152” folder.
- **platform**: this folder contains the hardware platforms compliant with the Contiki platform APIs. The code to support the new ST “platform” ported in Contiki is contained in the “./platform/stm32nucleo-spirit1” folder.
- **examples**: this folder contains the sample applications. The ST examples using this port are included in the folder named “./examples/stm32nucleo-spirit1”.

## 2.4 Sample applications

In Contiki, there are two types of application “examples”:

- Native examples outline the generic functionality of the Contiki OS and networking protocol stack, among other things. Many such examples are available, covering a broad range of use cases.
- Platform specific examples outline the specific functionality of a given hardware “platform”. ST provides one such example for the “stm32nucleo-spirit1” platform, called “sensor-demo”.

### 2.4.1 Building Contiki native examples

The “examples” directory in the Contiki repository contains many application examples.

The following examples have been successfully tested on this port:

- examples/hello-world
- examples/ipv6/simple-udp-rpl (multicast, rpl-border-router, simple-udp-rpl)

The following sections outline how to build these examples.

### 2.4.1.1 simple-udp-rpl example

This example demonstrates simple communication between two nodes using the UDP protocol. One node is the sender of UDP datagrams and another node is the receiver.

To build the example called "simple-udp-rpl", go to examples/ipv6/simple-udp-rpl directory.

If the X-NUCLEO-IDS01A5 sub-1GHz RF expansion board is used, run the following commands:

```
make TARGET=stm32nucleo-spirit1 BOARD=ids01a5 clean
make TARGET=stm32nucleo-spirit1 BOARD=ids01a5
arm-none-eabi-objcopy -O binary unicast-sender.stm32nucleo-spirit1 sender.bin
arm-none-eabi-objcopy -O binary unicast-receiver.stm32nucleo-spirit1 receiver.bin
```



If the X-NUCLEO-IDS01A4 sub-1GHz RF expansion board is used, use "ids01a4" in place of "ids01a5".

This will create two binary executables files, one to be programmed on the sender node called "sender.bin", and one to be programmed on the receiver node called "receiver.bin". Refer to [Section 3.3: "Board setup guide"](#) for information on how to setup the boards and flash the files generated by the previous command.

For all the examples that use a terminal, the minicom application can be used in the following way (assuming ttyACM0 is the device used by the board):

- `minicom -b 115200 -D /dev/ttyACM0`
- Press CTRL-A-Z and then U, to enable a proper carriage return to be printed.

The next figure shows an example of data printed by the "unicast-sender" node.

**Figure 2: Terminal utility showing the running Unicast-sender example**

```
Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Jan  1 2014, 17:13:22.
Port /dev/ttyACM0, 16:42:55

Press CTRL-A Z for help on special keys

Sending unicast to aaaa:a00:f7ff:b9bc:4643
Sending unicast to aaaa:a00:f7ff:b9bc:4643
Sending unicast to aaaa:a00:f7ff:b9bc:4643
Sending unicast to aaaa:a00:f7ff:b9bc:4643
Sending unicast to aaaa:a00:f7ff:b9bc:4643
Sending unicast to aaaa:a00:f7ff:b9bc:4643
Sending unicast to aaaa:a00:f7ff:b9bc:4643
Sending unicast to aaaa:a00:f7ff:b9bc:4643
```

The next figure shows an example of data printed by the "unicast-receiver" node associated with the unicast packets sent by the sender node.

**Figure 3: Terminal utility showing the running Unicast-receiver example**

```
Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Jan  1 2014, 17:13:22.
Port /dev/ttyACM1, 16:44:40

Press CTRL-A Z for help on special keys

Data received from aaaa:800:f5ff:eb3a:14c5 on port 1234 from port 1234 with length 10: 'Message 0'
Data received from aaaa:800:f5ff:eb3a:14c5 on port 1234 from port 1234 with length 10: 'Message 1'
Data received from aaaa:800:f5ff:eb3a:14c5 on port 1234 from port 1234 with length 10: 'Message 2'
Data received from aaaa:800:f5ff:eb3a:14c5 on port 1234 from port 1234 with length 10: 'Message 3'
Data received from aaaa:800:f5ff:eb3a:14c5 on port 1234 from port 1234 with length 10: 'Message 4'
Data received from aaaa:800:f5ff:eb3a:14c5 on port 1234 from port 1234 with length 10: 'Message 5'
Data received from aaaa:800:f5ff:eb3a:14c5 on port 1234 from port 1234 with length 10: 'Message 6'
Data received from aaaa:800:f5ff:eb3a:14c5 on port 1234 from port 1234 with length 10: 'Message 7'
```

### 2.4.1.2 rpl-border-router example

This example lets you configure a node as an “RPL Border Router”, which is the device that stands between 6LoWPAN network and a host system (e.g., a Linux PC) connected to the Wide Area Network.

From the top Contiki folder, run the following commands to generate a suitable binary file to be flashed on the device.

```
cd examples/ipv6/rpl-border-router
make TARGET=stm32nucleo-spirit1 BOARD=ids01a5
arm-none-eabi-objcopy -O binary border-router.stm32nucleo-spirit1 br.bin
```



If the X-NUCLEO-IDS01A4 sub-1GHz RF expansion board is used, use “ids01a4” in place of “ids01a5”.

This will create a binary executable file called “br.bin” that implements the border router application for the stm32nucleo-spirit1 platform. Refer to for information on how to setup the board and flash the “br.bin” file generated by the previous command.

### 2.4.1.3 Erbium REST example

This example lets you configure a node as a REST server or client.

In our case, we want to setup a wireless node as REST server, in order to enable the access to the hosted resources by the means of the CoAP protocol.

To build the example, assuming the sub-1GHz expansion board used is the X-NUCLEO-IDS01A5, run the following commands:

```
cd examples/er-rest-example
make TARGET=stm32nucleo-spirit1 BOARD=ids01a5
arm-none-eabi-objcopy -O binary er-example-server.stm32nucleo-spirit1 er-example-server.bin
```



If the X-NUCLEO-IDS01A4 sub-1GHz RF expansion board is used, use “ids01a4” in place of “ids01a5”.

## 2.4.2 Building examples provided by ST

### 2.4.2.1 “Sensor Demo” example

ST has developed a sample application to demonstrate how sensor data can be read. In order to use this example, the X-NUCLEO-IKS01A1 expansion board must be connected to the system (see [Section 3: “System setup guide”](#)). Sensor data is accessed by this example using the Contiki APIs for sensors.

The sensor-demo example can be found in: examples/stm32nucleo-spirit1/sensor-demo

To build the example, assuming the sub-1GHz expansion board is the X-NUCLEO-IDS01A5, run the following commands:

```
cd examples/stm32nucleo-spirit1/sensor-demo
make TARGET=stm32nucleo-spirit1 BOARD=ids01a5
arm-none-eabi-objcopy -O binary sensor-demo.stm32nucleo-spirit1 sensordemo.bin
```





If the X-NUCLEO-IDS01A4 sub-1GHz RF expansion board is used, use “ids01a4” in place of “ids01a5”.

Refer to [Section 3.3: "Board setup guide"](#) for information about how to program the board using the “sensordemo.bin” file.

When the demo is running, data from the X-NUCLEO-IKS01A1 board sensors (humidity, pressure, magnetometer, acceleration, gyroscope), along with button and LED status and the radio link parameters (RSSI and LQI) is printed on the terminal every five seconds. In this demo, pressing the user button toggles the LED status.

The serial interface parameters for this demo are 115200, n, 8, 1.

The next figure shows the terminal window when the sensor-demo is running.

**Figure 4: Terminal utility snapshot when sensor-demo runs**

```
File Edit View Search Terminal Help
Gyroscope: 23/12/3 (X/Y/Z) mdps

Button state: Released (pressed 3 times)
LEDs status: RED:n/a GREEN:on
Radio (RSSI): -130.0 dBm
Radio (LQI): 0
Temperature: 26.8 C
Humidity: 58.9 rH
Pressure: 995.0 mbar
Magneto: 1820/2161/1285 (X/Y/Z) mgauss
Acceleration: 143/135/16740 (X/Y/Z) mg
Gyroscope: 20/12/4 (X/Y/Z) mdps

Button state: Released (pressed 3 times)
LEDs status: RED:n/a GREEN:on
Radio (RSSI): -72.5 dBm
Radio (LQI): 4
Temperature: 26.8 C
Humidity: 59.0 rH
Pressure: 994.9 mbar
Magneto: 1845/2223/1278 (X/Y/Z) mgauss
Acceleration: 151/135/16736 (X/Y/Z) mg
Gyroscope: 20/12/5 (X/Y/Z) mdps
```

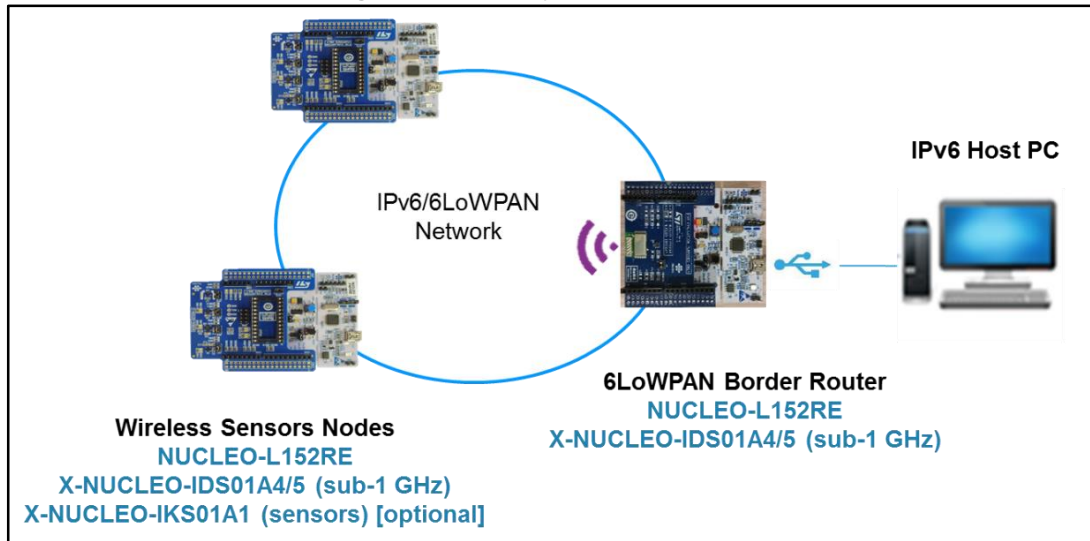
## 2.5 Building an overall system with an IPv6 host, an edge router and a sensor node

In this section we use some examples explained in the previous section, and we provide information to setup a system with:

- An IPv6 host:
  - this device runs the client application like a web browser over an IPv6 based protocol stack
  - it can also provide connectivity to a Wide Area Network
- An Edge (or border) router:
  - This device creates a 6LoWPAN network and is connected to the wireless nodes from one side, and is connected to the IPv6 host from the other side
  - In our case, it is implemented on an STM32 Nucleo board with a Sub-1GHz RF expansion board
- A wireless sensor node:
  - a low-power wireless device connected to the 6LoWPAN network

- In our case, it is implemented on an STM32 Nucleo board with a Sub-1GHz RF expansion board and a sensors expansion board

Figure 5: Overall system architecture



### 2.5.1 Edge (border) router setup

The border router is implemented with a NUCLEO-L152RE board with an X-NUCLEO-IDS01A4 (or A5) expansion board plugged on top.

The border router application used in this setup is the “rpl-border-router” example.

Refer to [Section 2.4.1.2: “rpl-border-router example”](#) for information about how to generate the firmware for this application.

To facilitate analysis, binary files ready to be flashed on the border router node can be downloaded from these GitHub repositories:

- With the X-NUCLEO-IDS01A4 expansion board:  
<https://github.com/STclab/stm32nucleo-spirit1-examples/blob/master/binaries/868-MHz/rpl-border-router/border-router-868.bin>
- With the X-NUCLEO-IDS01A5 expansion board:  
<https://github.com/STclab/stm32nucleo-spirit1-examples/blob/master/binaries/915-MHz/rpl-border-router/border-router-915.bin>

### 2.5.2 Wireless sensor node setup

The wireless sensor node is implemented with a NUCLEO-L152RE board with an X-NUCLEO-IDS01A4 (or A5) expansion board and X-NUCLEO-IKS01A1 sensor expansion board plugged on top.

There are various Contiki examples that could be used to generate a wireless node firmware.

We propose using the firmware from [Section 2.4.1.3: “Erbium REST example”](#) to demonstrate how the resources can be accessed using the CoAP protocol.

To facilitate analysis, binary files ready to be flashed on the border router node can be downloaded from these GitHub repositories:

- With the X-NUCLEO-IDS01A4 expansion board:  
<https://github.com/STclab/stm32nucleo-spirit1-examples/blob/master/binaries/868-MHz/rpl-border-router/border-router-868.bin>

- With the X-NUCLEO-IDS01A5 expansion board:  
<https://github.com/STclab/stm32nucleo-spirit1-examples/blob/master/binaries/915-MHz/rpl-border-router/border-router-915.bin>

These binary files implement the “sensor-er-rest-example”, which is a slightly modified version of Contiki’s native “er-rest-example”, with added support for the X-NUCLEO-IDS01A1 sensor expansion board.

## 2.5.3 IPv6 host setup

### 2.5.3.1 IPv6 packets bridging - setup for Windows environment

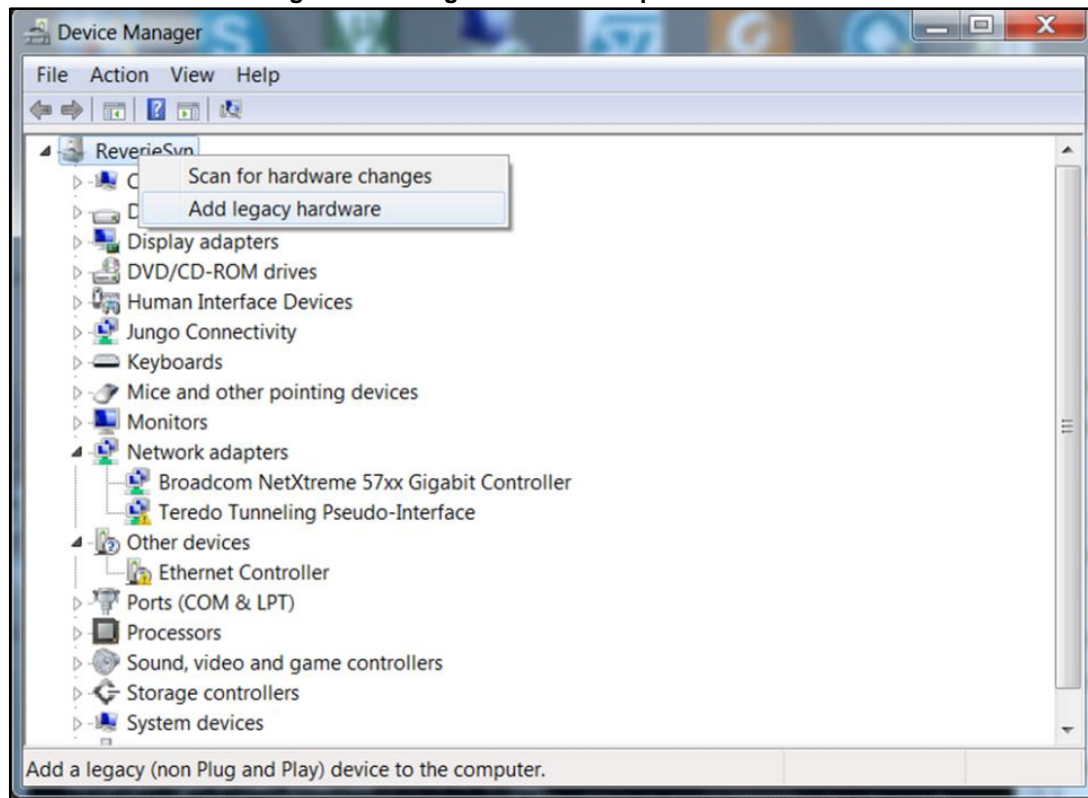
This section provides setup information when the host PC is running Windows (version 7 and later). The host side implements a standard IPv6 based networking stack.

A software utility named “wpcapslip6” is required to exchange IPv6 packets over the serial line between the host PC stack and the border router IPv6 stack.

To set up this utility:

- The “wpcapslip6” utility needs a working network adapter, so the first step in the setup is to install one. This utility is provided in the Contiki package in the “tools/stm32w/wpcapslip6” folder. The Microsoft loopback adapter can be installed using “Add legacy hardware” in the Windows Device Manager, as shown below.

**Figure 6: Adding the network adapter in Windows**

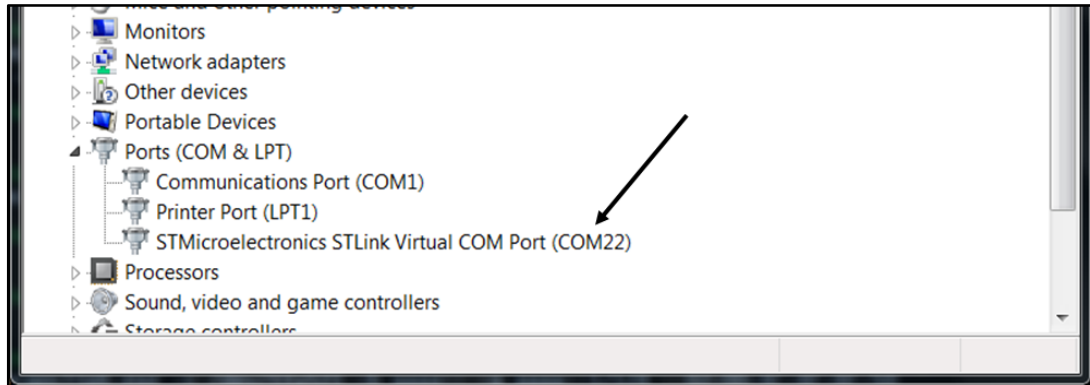


- Reboot the PC after the installation of the Microsoft loopback adapter.
- Copy the “cygwin1.dll” file from the “contiki/tools/cygwin” folder to the wpcapslip6 folder.
- Install the WinPcap Windows packet capture library (<https://www.winpcap.org/install/>).
- In a DOS terminal, run the wpcaslip6 utility with the rpl-border-router example

```
cd tools\stm32w\wpcapslip6
wpcapslip6.exe -s /dev/ttyS21 -b aaaa:: -a aaaa::1/128 [addr]
```

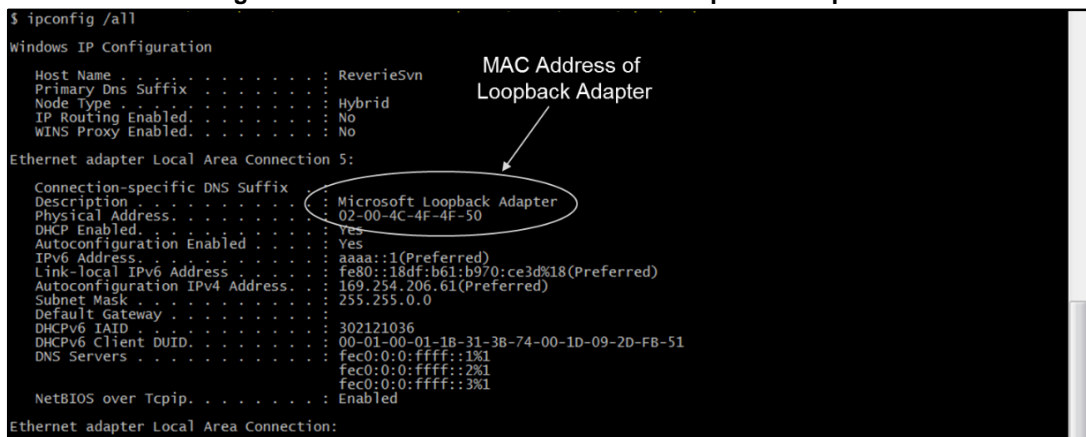
The serial device number (“ttyS21” in this example) can be obtained by looking at the Virtual COM Port number associated with the STM32 Nucleo board (see the next screenshot). The value to use when running the “wpcapslip6.exe” command is the COM port number minus 1. In this example, it is number 22, so “ttyS21” is the device parameter used.

**Figure 7: STM32 Nucleo board Virtual COM Port value**



The [addr] parameter is the MAC address of the local network adapter. This information can be found by using the “ipconfig /all” command, as shown in the next screenshot:

**Figure 8: MAC address of the Microsoft Loopback Adapter**



Finally, the next figure shows a snapshot of the terminal in which the wpcapslip6 utility is run.

Figure 9: wpcapslip6 terminal window

```
~/workspace/contiki-stm32nucleo-spirit1/tools/stm32w/wpcapslip6
$ ./wpcapslip6.exe -s /dev/ttyS21 -b aaaa:: -a aaaa::1/128 02-00-4C-4F-4F-50
Using local network interface: Local Area Connection 5

10:10:56 netsh interface ipv6 add address "Local Area Connection 5" aaaa::1/128

10:10:58 wpcapslip6 started on ``/dev/ttyS21``
10:10:58 Got request message of type M
10:10:58 *** Gateway's MAC address: 08-00-f7-ff-b7-bd-48-42
10:10:58 Fictitious MAC-48: 0A-00-F7-BD-48-42
10:10:58 netsh interface ipv6 add route aaaa::/64 "Local Area Connection 5" aaaa::a00:f7ff:b7bd:4842
Ok.

10:10:58 netsh interface ipv6 add neighbor "Local Area Connection 5" aaaa::a00:f7ff:b7bd:4842 "0A-00-F7-BD-48-42"

10:10:58 Got configuration message of type O
10:10:58 *** Address:aaaa:: => aaaa:0000:0000:0000
10:10:58 Got configuration message of type P
10:10:58 Setting prefix aaaa::
10:10:59 Server IPv6 addresses:
10:10:59 aaaa::a00:f7ff:b7bd:4842
10:10:59 fc00::a00:f7ff:b7bd:4842
10:10:59 fe80::a00:f7ff:b7bd:4842
```

### 2.5.3.2 IPv6 packets bridging - Setup for Linux environment

In this section we provide setup information in case the host PC is a Linux machine. The host side implements a standard IPv6 based networking stack.

A software module named “tunslip6” is needed in order to exchange IPv6 packets over the serial line between the host PC stack and the border router IPv6 stack. This utility is provided in the Contiki package.

To compile it, from the top level contiki directory run:

```
cd ./tools
make tunslip6
```

We assume that the border router device that was set up in the previous section is connected to a Linux host.

The next step is to launch the tunslip6 application by providing the virtual communication port to which the border router device is connected (located in /dev/ttyACMx where x is the corresponding port number):

```
sudo ./tunslip6 -s /dev/ttyACM0 aaaa::1/64
```

This command creates a new virtual interface in the Linux host called “tun0”.

After launching this command, reset the STM32 Nucleo board on which the border router is implemented by pressing the RESET (black) button to trigger system initialization and an exchange of configuration information with the tunslip6 application.

The next figure shows a snapshot of the terminal in which tunslip6 is run.

Figure 10: tunslip6 terminal window

```

*****SLIP started on ``/dev/ttyACM0''
opened tun device ``/dev/tun0'
ifconfig tun0 inet `hostname` up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 inet 172.16.0.1 pointopoint 172.16.0.2
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:172.16.0.1  P-t-P:172.16.0.2  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
  aaaa::800:f5ff:eb3a:14c5
  fc00::800:f5ff:eb3a:14c5
  fe80::800:f5ff:eb3a:14c5
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
  aaaa::800:f5ff:eb3a:14c5
  fc00::800:f5ff:eb3a:14c5
  fe80::800:f5ff:eb3a:14c5

```

### 2.5.3.3 IPv6 Host setup troubleshooting

- With Ubuntu kernel 3.13.0-65-generic (part of the Ubuntu 14.04 LTS) the tunslip6 application does not work properly
- Cygwin limits /dev/ttyS\* number to 100. If it has a device with a higher number, either:
  - Recompile Cygwin as suggested here: <https://www.cygwin.com/ml/cygwin/2008-08/msg00151.html>
  - Remove allocated COM ports, some procedures are described here: <http://superuser.com/questions/408976/how-do-i-clean-up-com-ports-in-use>
- You must disable the firewall
- Ensure IPv6 is enabled:
  - In the properties of the Loopback Adapter, set the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\services\TCPIP6\Parameters\DisabledComponents registry key to the value 0x8E
  - You may need to change the default name of the Network Connection associated with the Microsoft Loopback Adapter to a name that does not contain spaces

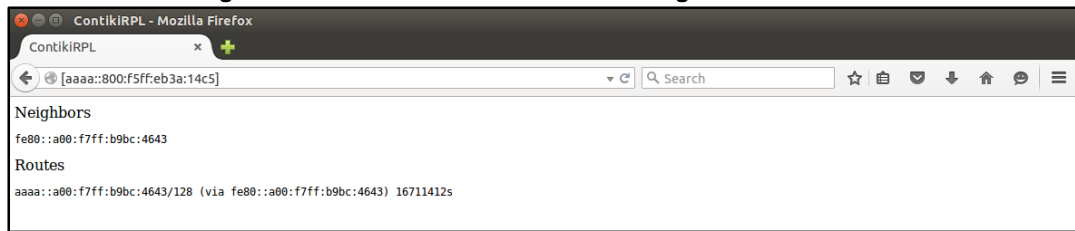
### 2.5.4 Contiki server access (border router) and connectivity test

This step involves opening a web browser to access the Contiki server providing the RPL neighbor and route information. You may use the first address listed in the wpcapslip6 (on Windows PC) or tunslip6 (on Linux PC) terminal window, after the “Server IPv6 addresses:” line.

The following figure shows the content of the web browser page after inserting the server IPv6 address (square brackets for IPv6 addresses) in the URL.



Figure 11: ContikiRPL server access – neighbors and routes



We assume that a wireless sensor node set up as described in the previous section is up and running.

At this point, the IPv6 address of the wireless sensor node should be present in the “Neighbors” list.

In order to demonstrate the end to end IPv6 connectivity between the Linux IPv6 host and the wireless sensor node, a simple ping6 command can be run. The IPv6 address to be used is the full 128-bit address in the “Routes” list.

For this specific example, the ping6 command (or “ping -6” under Microsoft Windows) to run would be:

```
ping6 aaaa::a00:f7ff:b9bc:4643
```

The next figure is a snapshot of the ping6 command execution, with the remote wireless sensor node replying to the ping from the Linux host.

Figure 12: Ping6 command window snapshot

```
PING aaaa::a00:f7ff:b9bc:4643(aaaa::a00:f7ff:b9bc:4643) 56 data bytes
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=1 ttl=63 time=70.0 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=2 ttl=63 time=70.7 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=3 ttl=63 time=76.8 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=4 ttl=63 time=65.8 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=5 ttl=63 time=72.8 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=6 ttl=63 time=67.8 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=7 ttl=63 time=74.8 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=8 ttl=63 time=68.9 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=9 ttl=63 time=75.9 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=10 ttl=63 time=64.9 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=11 ttl=63 time=65.9 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=12 ttl=63 time=72.9 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=13 ttl=63 time=67.8 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=14 ttl=63 time=74.8 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=15 ttl=63 time=69.8 ms
64 bytes from aaaa::a00:f7ff:b9bc:4643: icmp_seq=16 ttl=63 time=70.8 ms
^C
--- aaaa::a00:f7ff:b9bc:4643 ping statistics ---
16 packets transmitted, 16 received, 0% packet loss, time 15017ms
rtt min/avg/max/mdev = 64.936/70.685/76.827/3.620 ms
```

## 2.5.5

### Wireless node resources access demo using CoAP

We can now leverage the CoAP-based server previously set up on the wireless node (see [Section 2.4.1.3: "Erbium REST example"](#)).

To do so, you can open a web browser with CoAP client support. In this example, we are working with the Copper (Cu) CoAP user-agent for Firefox (<https://addons.mozilla.org/en-US/firefox/addon/copper-270430>)

After installing the Copper addon, open the web browser and insert in the URL tab the “coap” command with the IPv6 address of the node (as shown in the previous ping6 example, but enclosed in square brackets) and the port number used by the CoAP Erbium REST server (5683).

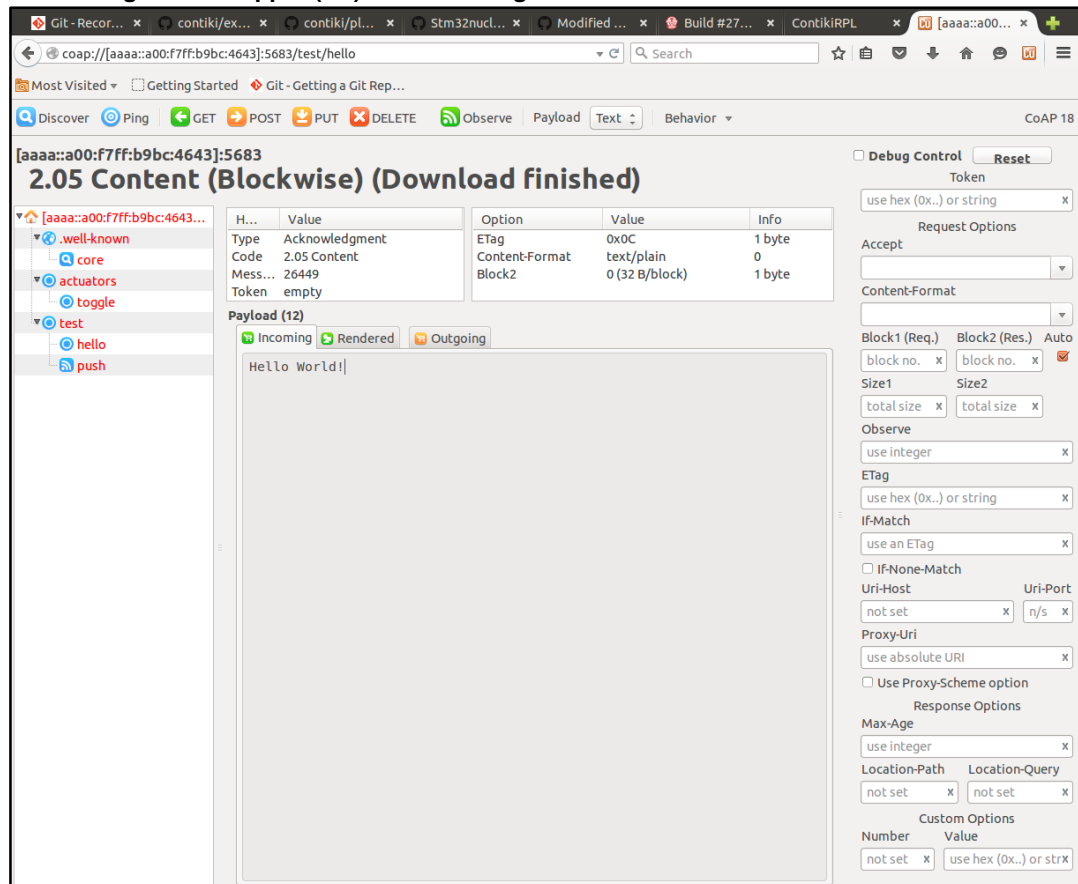
The URL inserted in the Firefox browser tab in our specific example is:

```
coap://[aaaa::a00:f7ff:b9bc:4643]:5683/
```

Initially, the page will be empty. Clicking on the “Discover” button triggers the resource discovery process so the client can acknowledge the available resources (sensors, actuators, etc.) from the CoAP server.

In the simple Erbium REST example, a few resources available for test purposes are listed on the left tab. For instance, selecting “hello” under “test” and clicking “GET” generates a CoAP GET message from the Copper CoAP client to the Erbium CoAP server running on the wireless node. The server replies with a CoAP acknowledgment “Hello World!” (see [Figure 13: “Copper \(Cu\) CoAP user-agent GET access to wireless remote node”](#)) text in the payload.

**Figure 13: Copper (Cu) CoAP user-agent GET access to wireless remote node**



## 2.5.6 Wireless node sensor resource access demo with CoAP

You can use a modified implementation of the Contiki “er-rest-example” application to access the resources (sensors and actuators) of the X-NUCLEO-IKS01A1 expansion board. It is called “sensor-er-rest example” and is available in the following GitHub repository: <https://github.com/STclab/stm32nucleo-spirit1-examples/tree/master/sensor-er-rest-example>

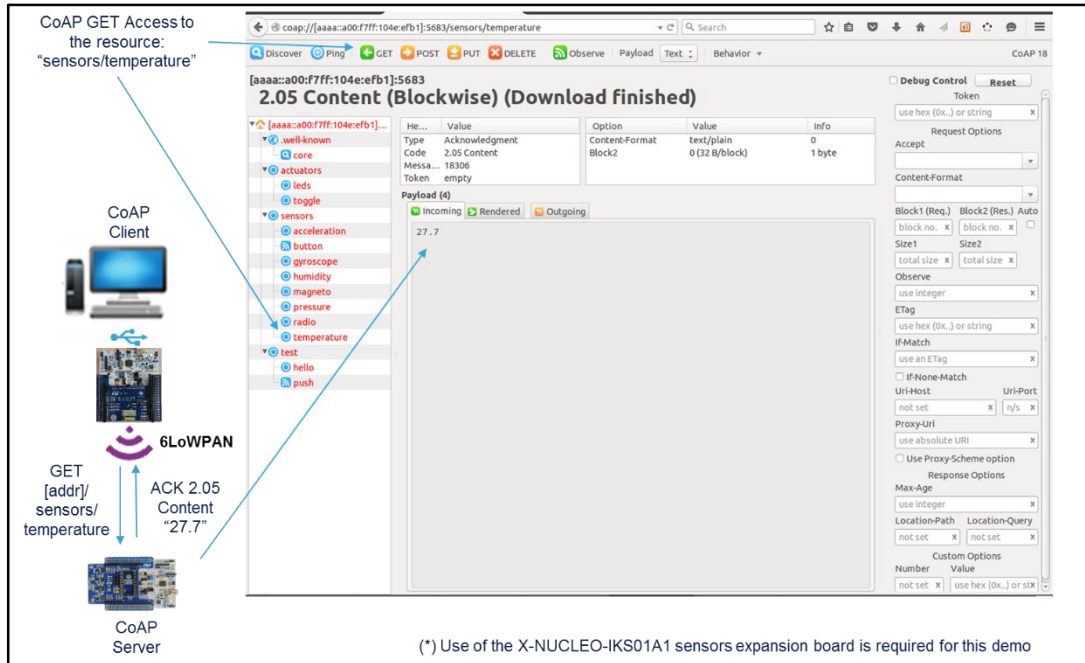
The “sensor-er-rest example” can be built exactly like the original “er-rest-example” described in [Section 2.4.1.3: “Erbium REST example”](#). Alternatively, you can use pre-built binary files for this example in the “binaries” folder of the same repository.

The demo is similar to the “hello world” example described in the previous section, except that clicking “Discover” returns a list of sensors (temperature, humidity, etc.) and actuators (LED) in a resource tree. Temperature sensor data can then, for example, be accessed by



using the GET command on the “temperature” resource. The response from the CoAP server running on the wireless node is, in this example, a CoAP ACK message with a body containing the text “27.7”, which corresponds to the current temperature reading from the sensor (on-board the X-NUCLEO-IKS01A1 expansion board).

**Figure 14: Example CoAP GET access to temperature sensor (running sensor-er-rest-example firmware)**



As it is based on the standard contiki er-rest-example, the green LED on the Nucleo board can be turned ON/OFF by:

- A CoAP PUT or POST on `/actuators/LEDs` with attached query option `?color=g`, so the complete path to be used in the address bar is `/actuators/leds?color=g`
- With payload (put in the “Outgoing” tab) `mode=on` or `mode=off`

The radio link parameters (RSSI or LQI) can be read by:

- A CoAP GET on `/sensors/radio` with attached query option `?p=lqi` or `?p=rssi`, so the complete path in the address bar is either:
  - `/sensors/radio?p=lqi`
  - `/sensors/radio?p=rssi`

## 3 System setup guide

### 3.1 Hardware requirements

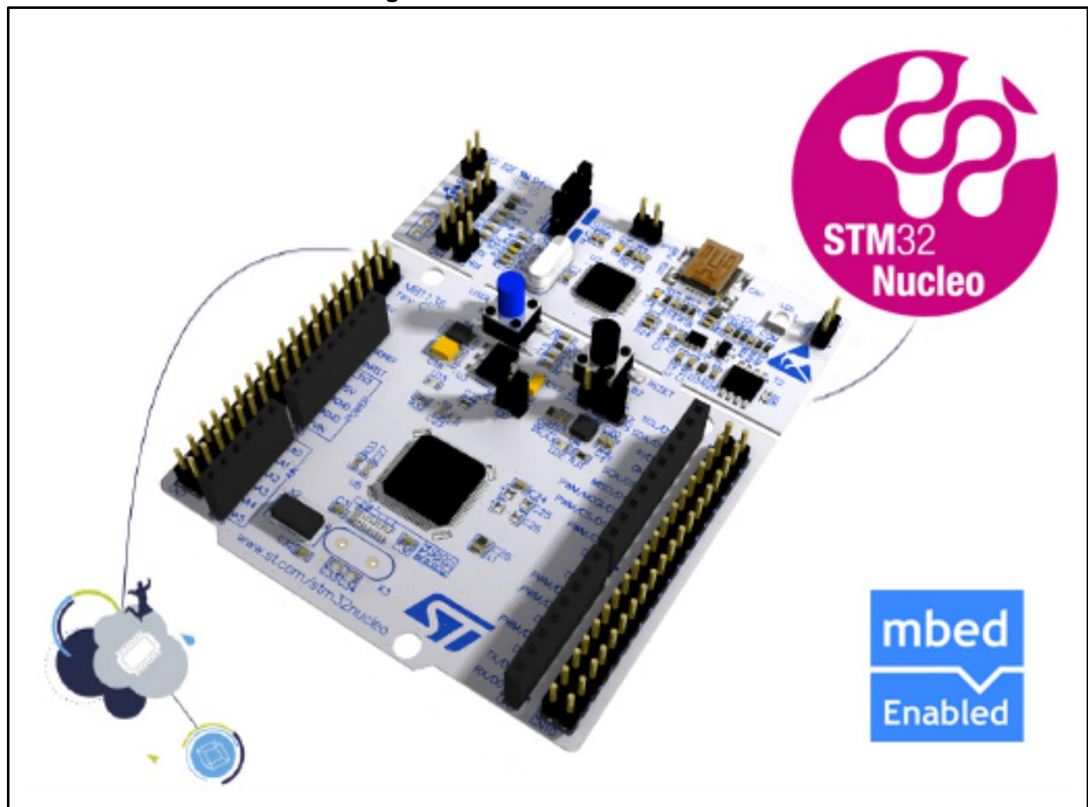
This section describes the hardware components required for using the Contiki 3.x ST port.

#### 3.1.1 NUCLEO-L152RE board

STM32 Nucleo boards provide an affordable and flexible way for users to try out new ideas and build prototypes with any of the STM32 microcontroller lines. Arduino™ connectivity support and ST morpho headers make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide choice of specialized expansion boards.

The STM32 Nucleo board does not require any separate probes as it integrates the ST-LINK/V2-1 debugger/programmer. It also comes with the STM32 comprehensive software HAL library together with various packaged software examples.

Figure 15: STM32 Nucleo Board



The NUCLEO-L152RE board belongs to the STM32 Nucleo family. It features an STM32L152RET6 ultra-low power microcontroller based on the ARM Cortex M3 MCU. Information regarding the NUCLEO-L152RE board is available on [www.st.com](http://www.st.com) at: <http://www.st.com/stm32nucleo>.

### 3.1.2 X-NUCLEO-IDS01A\* expansion board

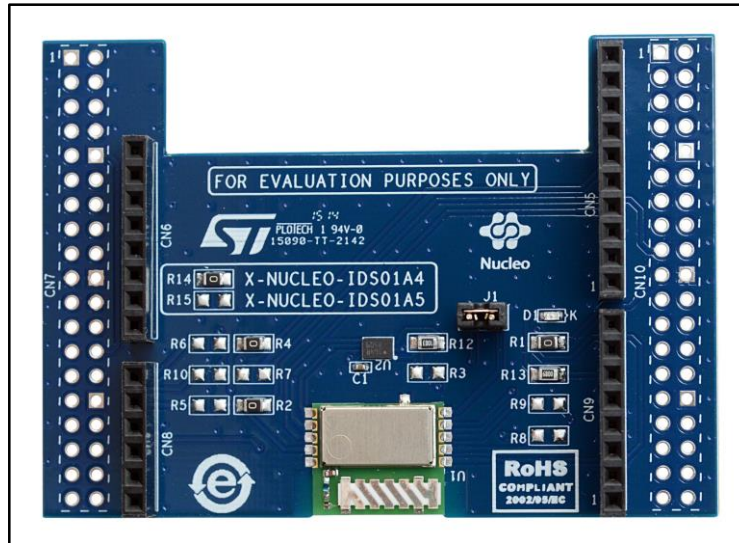
The X-NUCLEO-IDS01A\* is a demonstration kit to evaluate the features and capabilities of the SPSGRF module based on the SPIRIT1 low data rate, low power sub-1GHz transceiver device

The expansion board has onboard SPI EEPROM to store parameter settings and a user interface LED.

You must select either:

- X-NUCLEO-IDS01A4 to operate the SPIRIT1 transceiver at 868MHz, or
- X-NUCLEO-IDS01A5 to operate the SPIRIT1 transceiver at 915 MHz.

Figure 16: X-NUCLEO-IDS1A\* SPIRIT1 expansion board

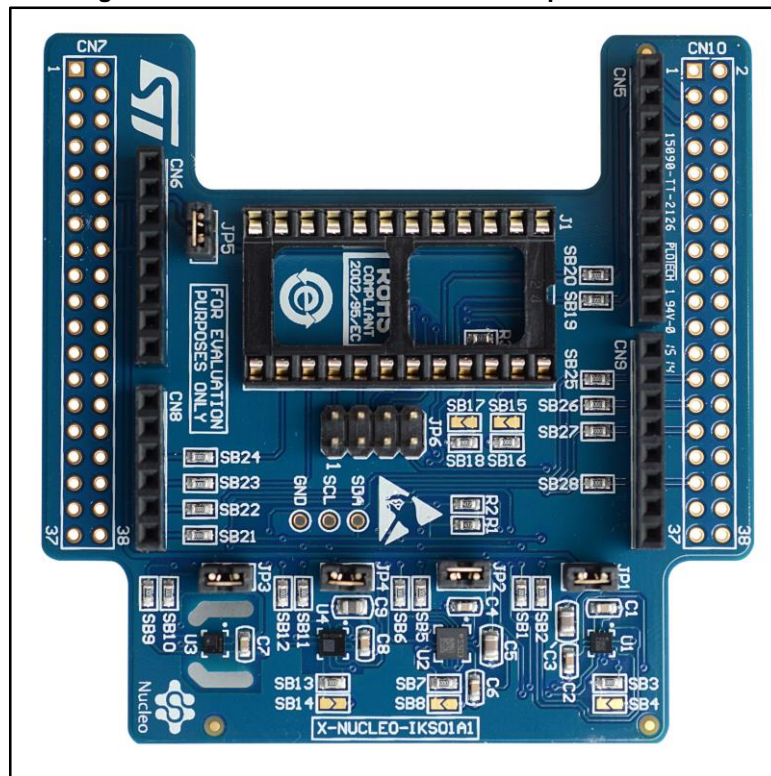


Information regarding the X-NUCLEO-IDS01A4 and X-NUCLEO-IDS01A5 expansion board is available on [www.st.com](http://www.st.com) at: <http://www.st.com/x-nucleo>.

### 3.1.3 X-NUCLEO-IKS01A1 expansion board (optional)

The X-NUCLEO-IKS01A1 is a sensor expansion board for the STM32 Nucleo. It is also compatible with Arduino UNO R3 connector layout, and is based on the STMicroelectronics humidity (HTS221), pressure (LPS25HB) and motion sensors (LIS3MDL and LSM6DS0).

Figure 17: X-NUCLEO-IKS01A1 sensor expansion board



The use of this board is optional in the stm32nucleo-spirit1 Contiki platform. When used, sensor data readings can be obtained via the Contiki sensor APIs.

Information regarding the X-NUCLEO-IKS01A1 expansion board is available on [www.st.com](http://www.st.com) at: <http://www.st.com/x-nucleo>.

## 3.2 Software requirements

The following software components are needed in order to set Contiki up on the STM32 Nucleo with ST expansion boards:

- ST Contiki port for STM32 Nucleo and expansion boards; installed automatically when the Contiki repository is cloned. See [Section 2.2: "Getting the software package"](#) for information on how to obtain a local copy of the repository. The Contiki Platform name for this port is: `stm32nucleo-spirit1`
- A toolchain to build the firmware. The port has been developed and tested with GNU Tools for ARM Embedded Processors (<https://launchpad.net/gcc-arm-embedded>). The port was developed and tested using version: `gcc-arm-none-eabi v4.83`

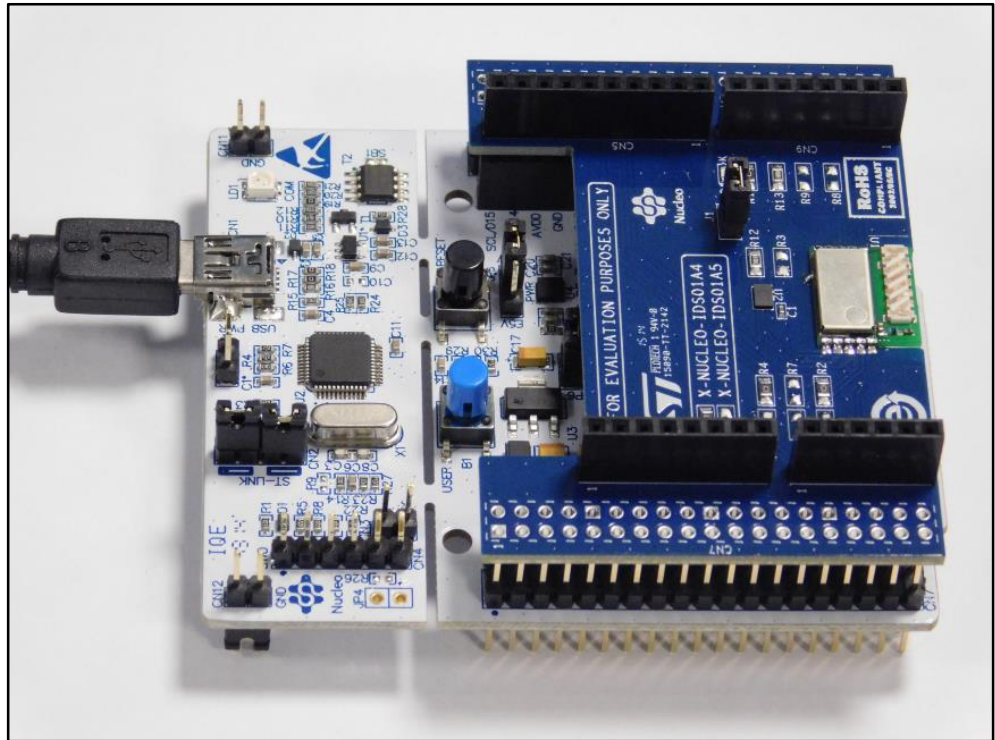
## 3.3 Board setup guide

Follow the steps below to connect the boards.

- 1 Check that the jumper on the J1 connector on the X-NUCLEO-IDS01A\* expansion board is connected. This jumper provides the required voltage to the devices on the board.

- 2 Connect the X-NUCLEO-IDS01A\* board onto the STM32 Nucleo board (NUCLEO-L152RE).

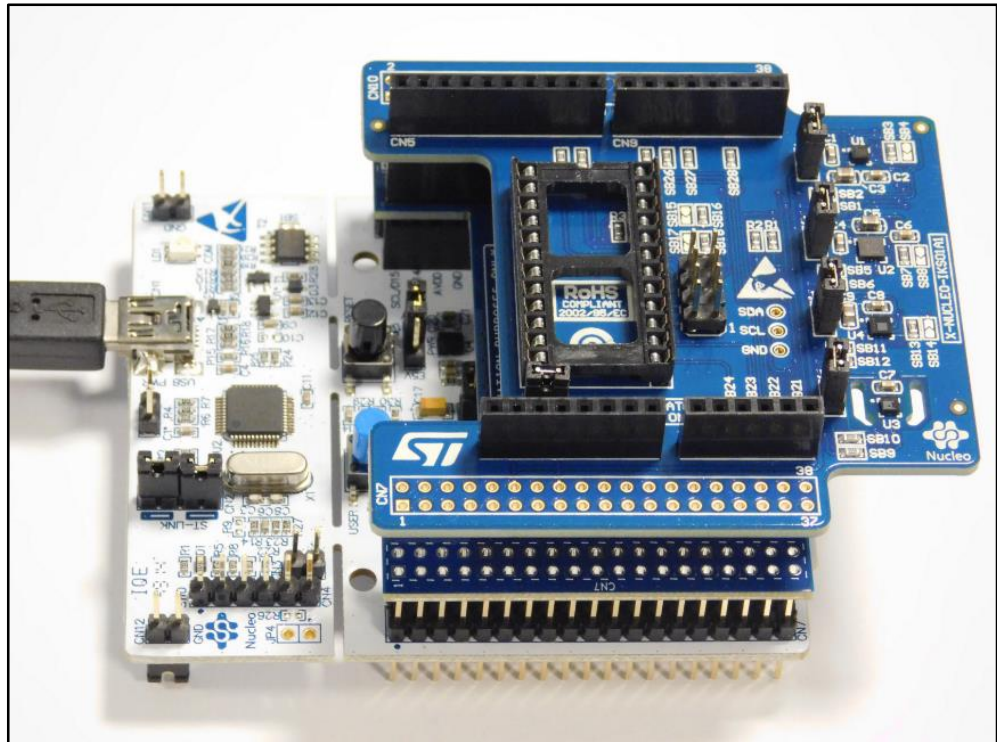
**Figure 18: X-NUCLEO-IDS01A\* expansion board connected to STM32 Nucleo Board**





- 3 If present, connect the optional X-NUCLEO-IKS01A1 board onto the X-NUCLEO-IDS01A\* board.

**Figure 19: X-NUCLEO-IKS01A1 and X-NUCLEO-IDS01A\* expansion boards connected to STM32 Nucleo Board**



- 4 Power the STM32 Nucleo board through a Mini-B USB cable connected to the PC.
- 5 Program the firmware on the STM32 Nucleo board by copying the binary file on the USB mass storage that is automatically created when plugging the STM32 Nucleo board to the PC. The STM32 Nucleo L1 device can be found in a Linux host in the /media folder with the name "NUCLEO\_L152RE" (if more than one STM32 Nucleo board is connected to the PC USB port, the name will be "NUCLEO\_L152REx", x=1, 2, etc.). For example, if the path to the mass storage created when connecting the STM32 Nucleo L1 platform is "/media/NUCLEO\_L152RE", then you can program the board with a binary file named "my\_firmware.bin" by simply running the following command:

```
cp my_firmware.bin /media/NUCLEO_L152RE
```

- 6 Press the MCU RESET (black) button on the STM32 Nucleo board
- 7 If necessary, open a terminal utility, select the serial port name to which the board is connected and configure it with the following parameters: 115200, n, 8, 1. The terminal utility will show the data printed by the selected example (the content depends on the example).

## 4 Revision history

Table 2: Document revision history

Date	Version	Changes
14-Dec-2015	1	Initial release.
29-Jan-2016	2	Updated <a href="#">Section 2.5.3.2: "IPv6 packets bridging - Setup for Linux environment"</a>

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved