Lab 01 – Sensor Device

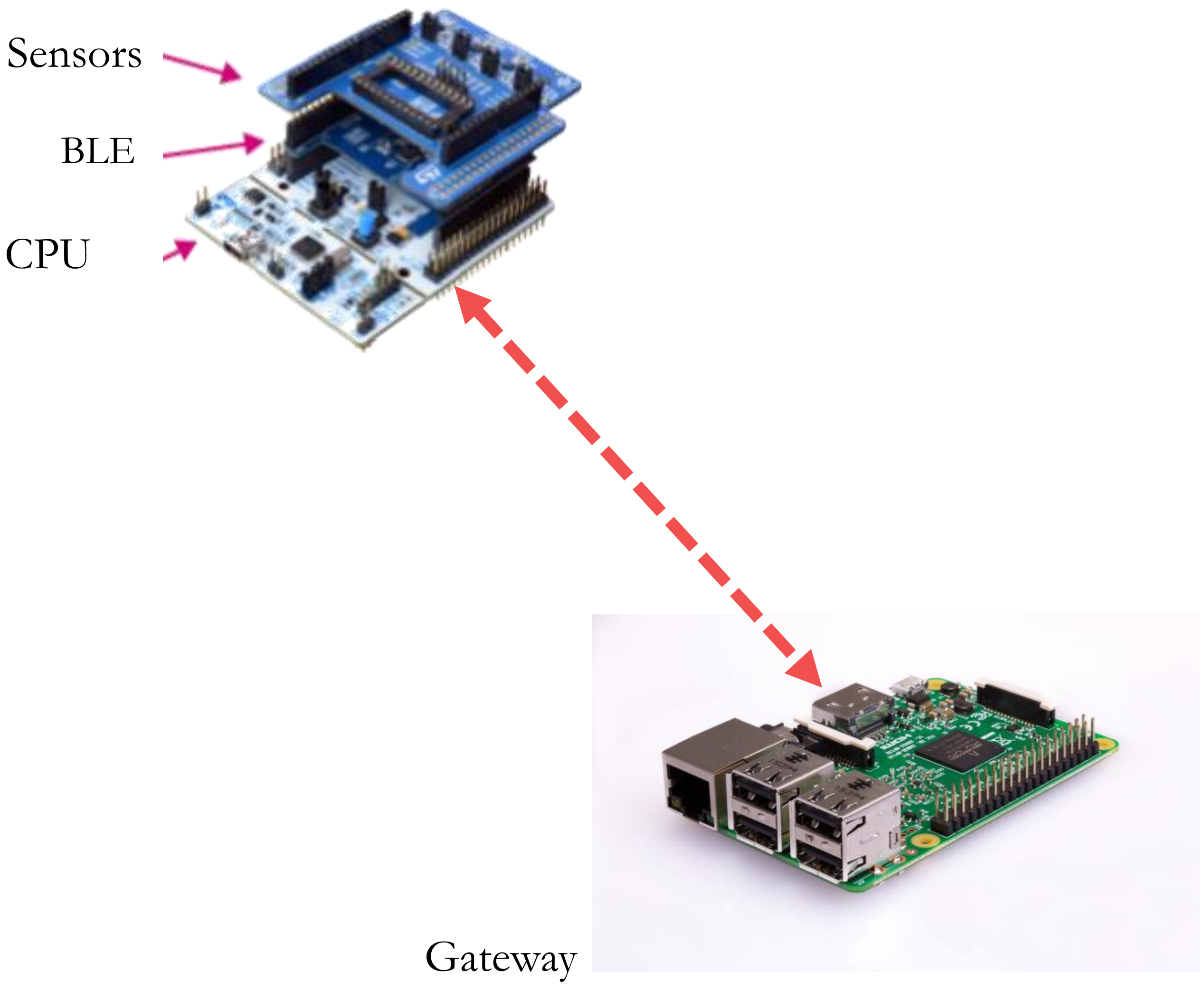
Develop and debug a Sensor device

Pre-requisits:

- *VMWare Player 12*
- *Lab Virtual Machine*
- *NUCLEO-L053R8 + X-NUCLEO-IDB05A1 + X-NUCLEO-IKS01A1 + USB Cable*
- *Gateway platform with BLE support*

In this Lab you will learn the process to develop a Sensor device based on an STM32 micro controller, using a Bluetooth Low Energy interface to expose onboard sensor data to a device Gateway.

Hardware environment



Development Tools

ST Microelectronics provides different support for firmware development, in this session you will use a free software solution called System Workbench for STM32 also named SW4STM32.

arm mbed: (online tool)

URL: https://developer.mbed.org/platforms/ST-Nucleo-F302R8/
Host OS: full online IDE.
License : Free

IAR Embedded Workbench® for ARM® (EWARM) by IAR systems®

URL: https://www.iar.com/iar-embedded-workbench/#!?architecture=ARM&device=STM32L053x8
Host OS: Windows
License: **Commercial**
Restrictions of Evaluation license:

The evaluation license is completely free of charge and allows you to try the integrated development environment and evaluate its efficiency and ease of use. When you start the product for the first time, you will be asked to register to get your evaluation license.

After installation, you have the following evaluation options to choose from:
a 30-day time-limited but fully functional license
a size-limited Kickstart license without any time limit

Restrictions to the 30-day time-limited evaluation
A 30-day time limitation.
Source code for runtime libraries is not included.
No support for MISRA C.
C-RUN is size-limited to 12 Kbytes of code, excluding constant data.
Limited technical support.
Must not be used for product development or any other kind of commercial use.

Restrictions to the Kickstart, size-limited evaluation
A 32 Kbyte code size limitation (16 Kbyte for Cortex-M0/M0+/M1).
Source code for runtime libraries is not included.
No support for MISRA C.
C-RUN is not available.
Limited technical support.

Microcontroller Development Kit for ARM® (MDK-ARM) by Keil®

Main URL: http://www2.keil.com/stmicroelectronics-stm32/mdk
Download URL: https://www.keil.com/demo/eval/arm.htm
Host OS: Windows
License: **free-to-use for STM32L0 and STM32F0**. Product Serial Number (PSN): U1E21-CM9GY-L3G4L
Restrictions: STM32L0 and STM32F0 for free

TrueSTUDIO® by Atollic®

URL: http://timor.atollic.com/
Host OS: Windows
License: Free + Pro
Restrictions of Free license:
    No advanced debugging capabilities
    No static code analysis tools

System Workbench for STM32 (SW4STM32) by AC6

Main URL: http://www.openstm32.org/HomePage
Download URL:
http://www.openstm32.org/Downloading+the+System+Workbench+for+STM32+installer?structure=Documentation
Host OS: Windows / Linux
License: Free
Restrictions: /

Other tools

STM32CubeMX - STM32Cube initialization code generator

URL: http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF259242?sc=stm32cube#
Host OS: Windows and Linux. For Linux, please follow the non-standard installation procedure:
http://www.openstm32.org/forumthread136
License: Free
Restrictions: **Usable only to generate initialization code and drivers**.

ST-Link

ST-Link is an USB debugger included to the NUCLEO-64 board which allows developers to deploy and debug applications without any additional tool.

Linux tools installation instructions

This chapter describes the development tool installation steps used to prepare the lab environment. You don't have to do it, as we already performed all the steps for you.

Prerequisites

The development tools require the following components to be installed first:

- openjdk-7-jre
- gksu
- libc6:i386 lib32ncurses5 (if missing on 64bits Linux, it gives ***"arm-none-eabi-XXX: not found"*** error)

Installation of System Workbench for STM32 (SW4STM32)

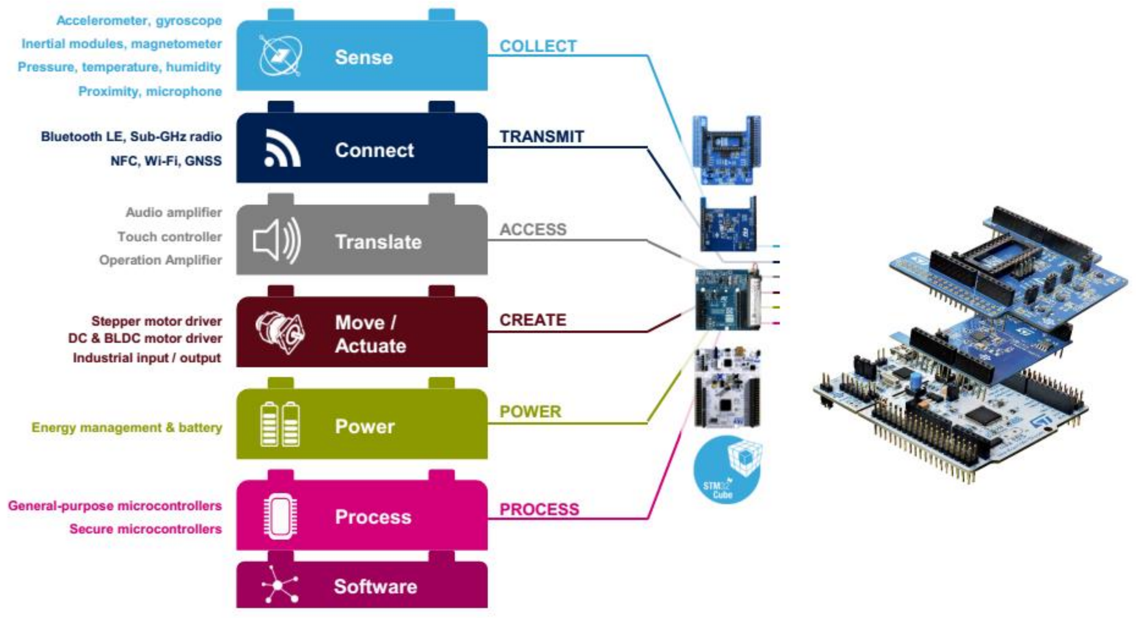
From a terminal, launch '*install_sw4stm32_linux_**64**bits-**v1.3**.run*' installer (**64** should be replaced by **32** in case of you are using a 32 bits Linux. **V1.3** is the program revision which could be different). Just follow the installation wizard.

For more details, visit:
http://www.openstm32.org/Installing+System+Workbench+for+STM32?structure=Documentation

Shields concept

Introduction

The 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 shields.

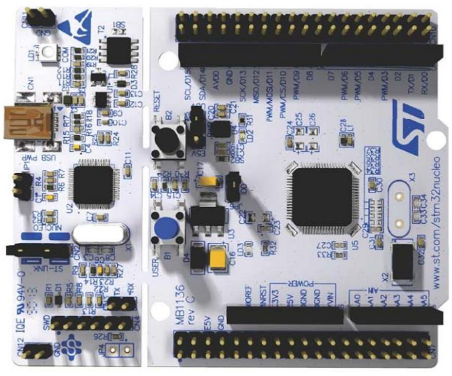


Hardware used in this lab

To reach the lab goals, you will use a NUCLEO-64 board equipped with an IDB05A1 BLE (Bluetooth Low Energy) shield and an IKS01A1 MEMS Inertial and Environmental shield stacked.

NUCLEO-64 (NUCLEO-L053R8)

Board details available at: http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/LN1847/PF260001



The STM32 Nucleo board provides an affordable and flexible way for users to try out new ideas and build prototypes with any STM32 microcontroller line, choosing from the various combinations of performance, power consumption and features. The 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 shields. The STM32 Nucleo board does not require any separate probe as it integrates the ST-LINK/V2-1 debugger and programmer. The STM32 Nucleo board comes with the STM32 comprehensive software HAL library together with various packaged software examples, as well as direct access to mbed online resources.

Key Features

- STM32 microcontroller with LQFP64 package
- Two types of extension resources
- Arduino Uno Revision 3 connectivity
- STMicroelectronics Morpho extension pin headers for full access to all STM32 I/Os
- mbed-enabled (http://mbed.org)
- On-board ST-LINK/V2-1 debugger/programmer with SWD connector
    - selection-mode switch to use the kit as a standalone ST-LINK/V2-1
- Flexible board power supply
    - USB VBUS or external source(3.3 V, 5 V, 7 - 12 V)
    - Power management access point
- Three LEDs
    - USB communication (LD1), user LED (LD2), power LED (LD3)
- Two push buttons: USER and RESET
- USB re-enumeration capability: three different interfaces supported on USB
    - Virtual Com port
    - Mass storage
    - Debug port
- Supported by wide choice of Integrated Development Environments (IDEs) including IAR™, Keil®, GCC-based IDEs

BLE shield (IDB05A1 or IDB04A1)

Board details available at: http://www.st.com/web/en/catalog/tools/FM116/SC1971/PF262191

The X-NUCLEO-IDB05A1 is a Bluetooth Low Energy evaluation board based on the SPBTLE-RF BlueNRG-MS RF module to allow expansion of the STM32 Nucleo boards. The SPBTLE-RF module is FCC (FCC ID: S9NSPBTLERF) and IC certified (IC: 8976C-SPBTLERF).

The X-NUCLEO-IDB05A1 is compatible with the ST Morpho and Arduino UNO R3 connector layout (the user can mount the ST Morpho connectors, if required). The X-NUCLEO-IDB05A1 interfaces with the

STM32 microcontroller via the SPI pin, and the user can change the default SPI clock, the SPI chip select and SPI IRQ by changing one resistor on the evaluation board.

Key Features

- STM32 expansion board based on the SPBTLE-RF module for STM32 Nucleo
- X-NUCLEO-IDB05A1 contains FCC and IC certified module SPBTLE-RF (FCC ID: S9NSPBTLERF and IC: 8976C-SPBTLERF)
- SPBTLE-RF:
    - Bluetooth Low Energy FCC and IC certified module based on Bluetooth® SMART 4.1 network processor BlueNRG-MS
    - Integrated Balun (BALF-NRG-01D3)
    - Chip antenna
- Compatible with STM32 Nucleo boards
- Equipped with Arduino UNO R3 connector
- Scalable solution, capable of cascading multiple boards for larger systems
- Free comprehensive development firmware library and example for BlueNRG-MS, compatible with STM32Cube firmware
- RoHS compliant

MEMS shield (IKS01A1)

Board details available at: http://www.st.com/web/en/catalog/tools/FM116/SC1248/PF261191

The X-NUCLEO-IKS01A1 is a motion MEMS and environmental sensor evaluation board system.

It is compatible with the Arduino UNO R3 connector layout, and is designed around STMicroelectronics' LSM6DS0 3-axis accelerometer + 3-axis gyroscope, the LIS3MDL 3-axis magnetometer, the HTS221 humidity and temperature sensor and the LPS25HB* pressure sensor.

The X-NUCLEO-IKS01A1 interfaces with the STM32 microcontroller via the I²C pin, and it is possible to change the default I²C port.

Key Features

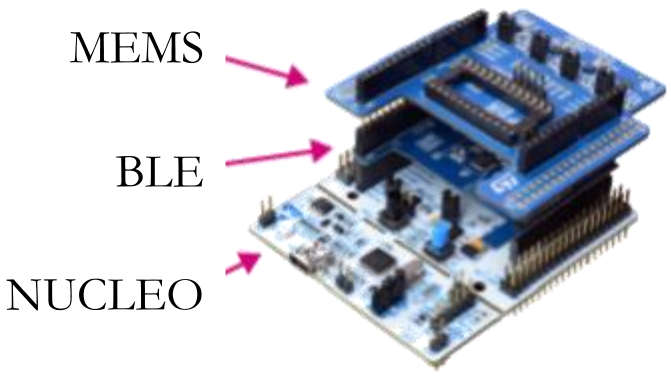
- LSM6DS0: MEMS 3D accelerometer (±2/±4/±8 g) + 3D gyroscope (±245/±500/±2000 dps)
- LIS3MDL: MEMS 3D magnetometer (±4/ ±8/ ±12/ 16 gauss)
- LPS25HB: MEMS pressure sensor, 260-1260 hPa absolute digital output barometer
- HTS221: capacitive digital relative humidity and temperature

- DIL 24-pin socket available for additional MEMS adapters and other sensors (UV index)
- Free comprehensive development firmware library and example for all sensors compatible with STM32Cube firmware
- Compatible with STM32 Nucleo boards
- Equipped with Arduino UNO R3 connector
- RoHS compliant

Hardware preparation

Before deploying your project binary on the target, you have to stack the different shields on the NUCLEO board. To have an easy access to the temperature sensor (U3) of the MEMS shield, we suggest stacking the board in the following order (from top to bottom):

1. MEMS
2. BLE
3. NUCLEO

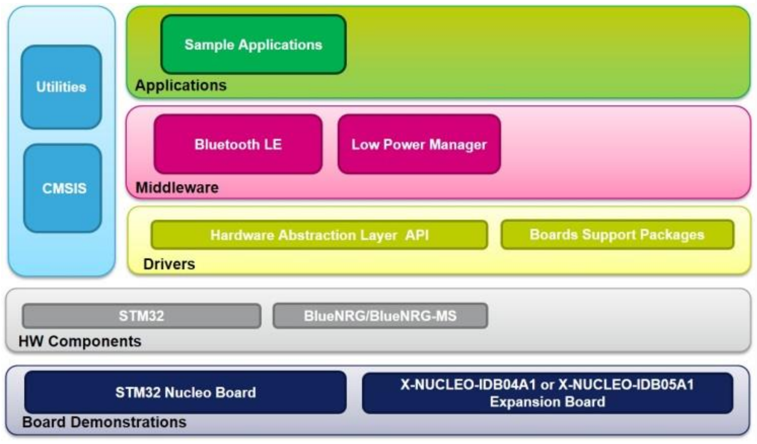


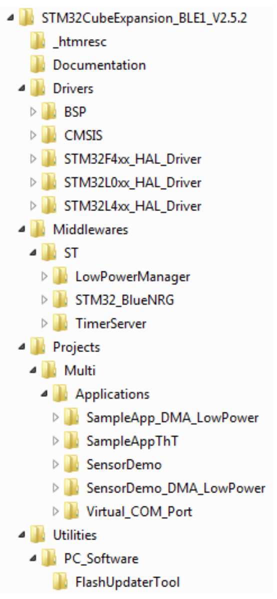
Shields libraries and samples

For every ST shields for STM32 boards, ST provides libraries and samples packages which are autonomous, i.e. without usage of STM32CubeMX or integration of main board firmware (STM32Cube_FW_L0_V1.4.0) in your projects. Actually, these packages are generated by ST with STM32CubeMX software.

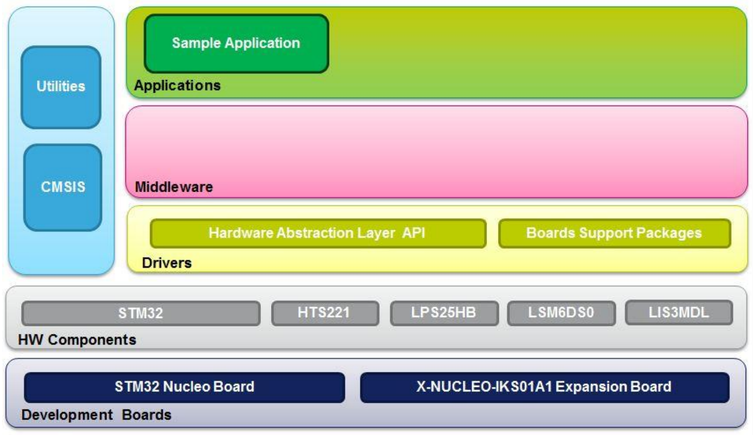
You can directly use the provided application samples, use these samples as a working base to customize your application (case of stacked shields) or only integrate the drivers into an existing project.

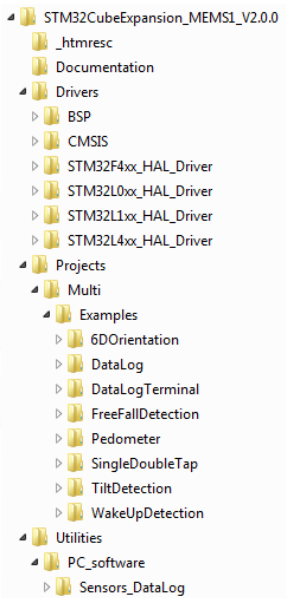
The shield package samples architecture is divided in layers. Here is the Bluetooth Low Energy sample stack and its file folder tree:




```
STM32CubeExpansion_BLE1_V2.5.2
  _htmresc
  Documentation
  Drivers
    BSP
    CMSIS
    STM32F4xx_HAL_Driver
    STM32L0xx_HAL_Driver
    STM32L4xx_HAL_Driver
  Middlewares
    ST
      LowPowerManager
      STM32_BlueNRG
      TimerServer
  Projects
    Multi
      Applications
        SampleApp_DMA_LowPower
        SampleAppThT
        SensorDemo
        SensorDemo_DMA_LowPower
        Virtual_COM_Port
  Utilities
    PC_Software
      FlashUpdaterTool
```

And the MEMS stack and package folder tree is:

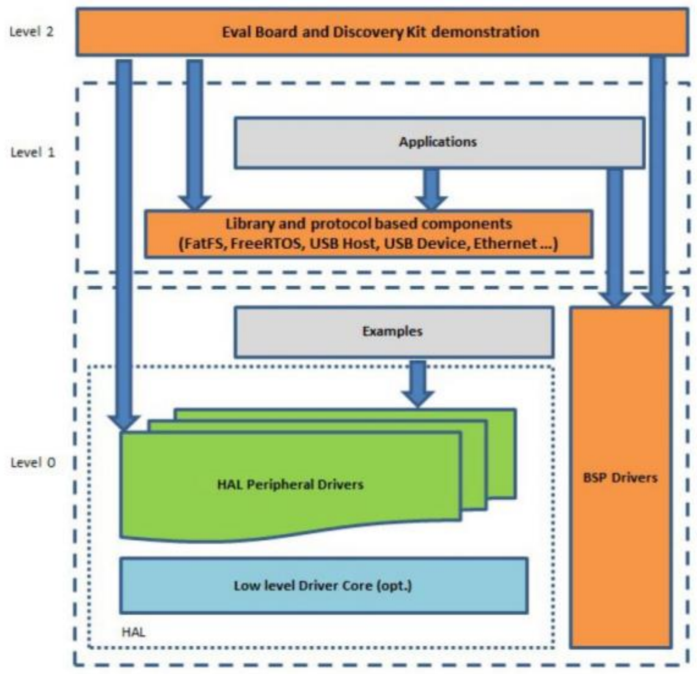



```
STM32CubeExpansion_MEMS1_V2.0.0
  _htmresc
  Documentation
  Drivers
    BSP
    CMSIS
    STM32F4xx_HAL_Driver
    STM32L0xx_HAL_Driver
    STM32L1xx_HAL_Driver
    STM32L4xx_HAL_Driver
  Projects
    Multi
      Examples
        6DOrientation
        DataLog
        DataLogTerminal
        FreeFallDetection
        Pedometer
        SingleDoubleTap
        TiltDetection
        WakeUpDetection
  Utilities
    PC_software
      Sensors_DataLog
```

Libraries common organization

ST provides its shield libraries with a common folder structure:

- The **Documentation** folder contains a compiled HTML file generated from the source code and detailed documentation regarding the software components and APIs.
- The **Drivers** folder contains:
  - The **H**ardware **A**bstraction **L**ayer (**HAL)** provides the low-level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks).
  - The **B**oards **S**upport **P**ackages (**BSP**) for each supported board or hardware platform, including those for the on-board components.
  - The **CMSIS layer**, which is a vendor-independent hardware abstraction layer for the Cortex-M processor series.
  - The **Middlewares** folder contains libraries and protocols related to host software and applications (e.g. to interface the BlueNRG controller).
- The **Projects** folder contains a sample application for the NUCLEO-L053R8, NUCLEO-L152RE, NUCLEO-L476RG and NUCLEO-F401RE platforms to access sensor data; it is provided with three development environments (IAR Embedded Workbench for ARM, RealView MDK-ARM Microcontroller Development Kit and System Workbench for STM32).
- The optional **Utilities** folder usually contains PC softwares.


Level 2
Eval Board and Discovery Kit demonstration
Level 1
Applications
Library and protocol based components
(FatFS, FreeRTOS, USB Host, USB Device, Ethernet ...)
Examples
Level 0
HAL Peripheral Drivers
BSP Drivers
Low level Driver Core (opt.)
HAL

Lab preparations

VMWare player

1. Before starting, you have to install VMWare player (v6 or higher) on your physical PC regarding its OS (Windows , Linux or MAC) and its CPU architecture (x86 or x64).
2. Then open the provided Lubuntu 18.04 x64 virtual machine.

Lubuntu 18.04 x64 VM

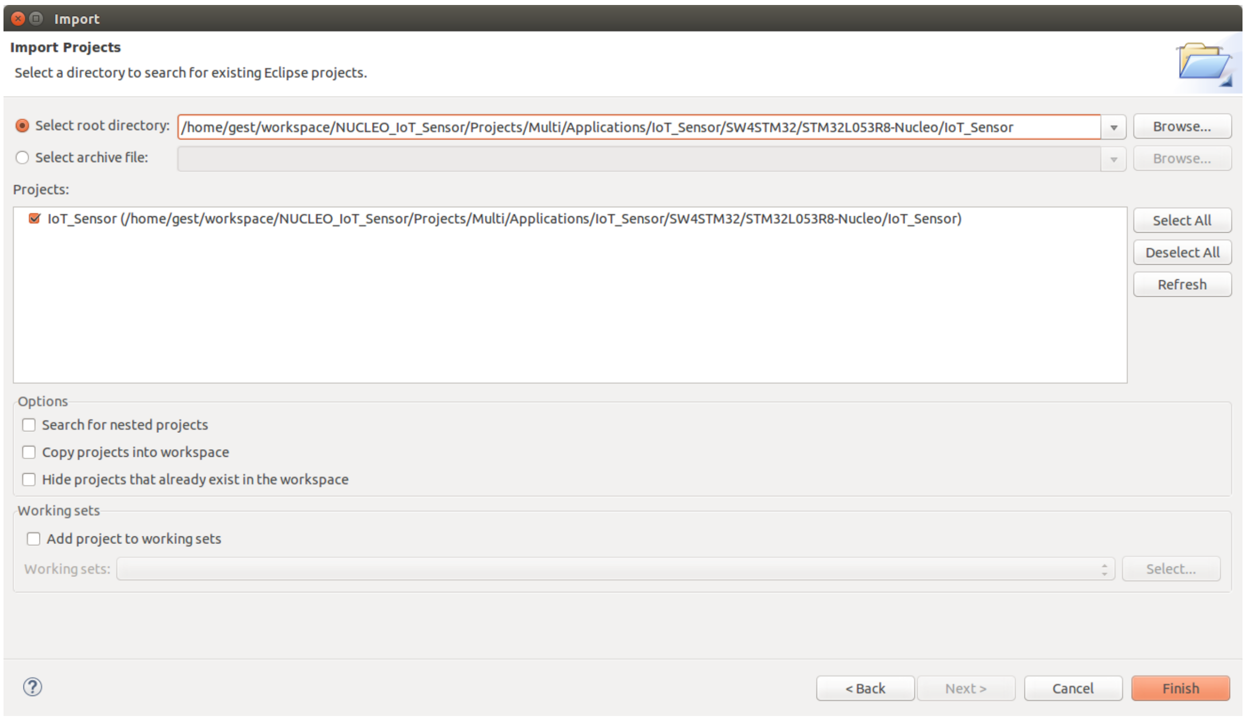
1. Open the session with login 'osboxes' and password 'Witekio' (case sensitive and without the quote)
2. The SW4STM32 tool is already installed and ready to generate application for the NUCLEO-64 board.
3. By default, the SW4STM32 projects will be located in the '/home/gest/workspace' folder

Import the Lab's project

Make sure that you copy the project files to the /home/workspace folder as mentioned below. Otherwise is might results in build and link failures.

Don't forget to store on your training git repository the lab files in a dedicated folder named Lab01. Commit and push when major steps are passed and at the end of the Lab Session. You will be evaluated on the resulting written code and git comments.

1. Copy and extract the Lab project in the 'workspace' folder.
2. In SW4STM32 menu, select *File*, *Import, General, Existing project into workspace*. In the *Import* dialog box which appeared, browse to the /home/gest/workspace/**NUCLEO_IoT_Sensor/Projects/Multi/Applications/IoT_Sensor/SW4STM32/STM32L053R8-Nucleo/IoT_Sensor** folder.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

(●) Select root directory: /home/gest/workspace/NUCLEO_IoT_Sensor/Projects/Multi/Applications/IoT_Sensor/SW4STM32/STM32L053R8-Nucleo/IoT_Sensor

( ) Select archive file:

Projects:

[✓] IoT_Sensor (/home/gest/workspace/NUCLEO_IoT_Sensor/Projects/Multi/Applications/IoT_Sensor/SW4STM32/STM32L053R8-Nucleo/IoT_Sensor)

Options

[ ] Search for nested projects

[ ] Copy projects into workspace

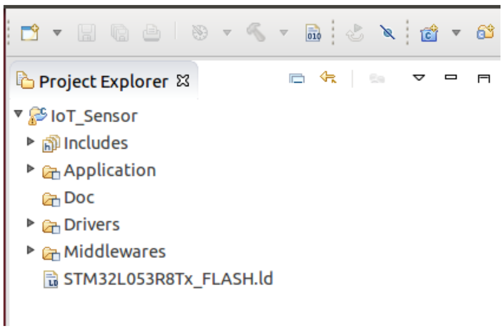
[ ] Hide projects that already exist in the workspace

Working sets

[ ] Add project to working sets

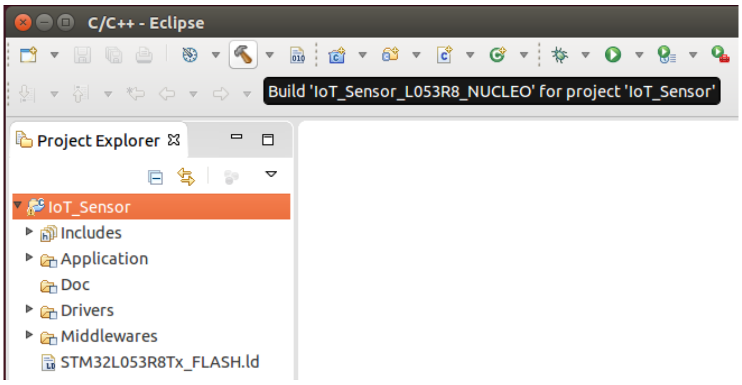
Working sets:

3. Then click on the *Finish* button.
4. You should have a project in the *Project Explorer*:

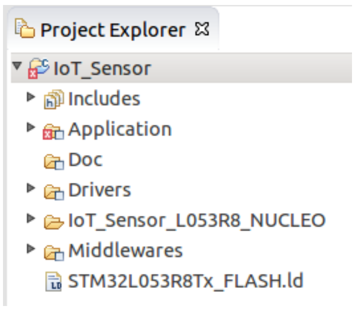


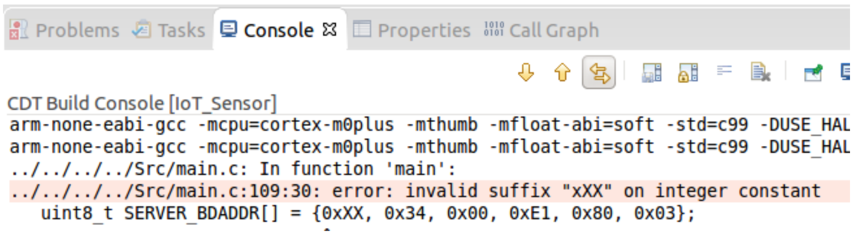
How to build the project?

- Select the imported project and click on the 'hammer' button (the highlighted one in the following snapshot):

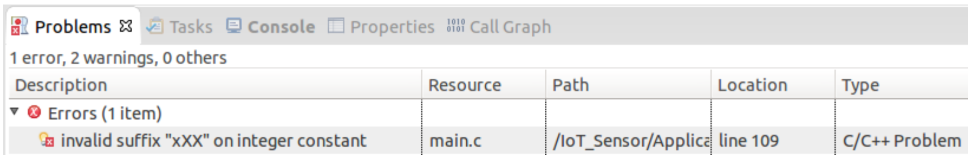


- You can also right-click on the project in the *Project Explorer* and select *Build Project*.
- If the compiler encounters errors, you can see red crosses in the *Project Explorer*, errors in the *Console* tab and errors in the *Problems* tab.





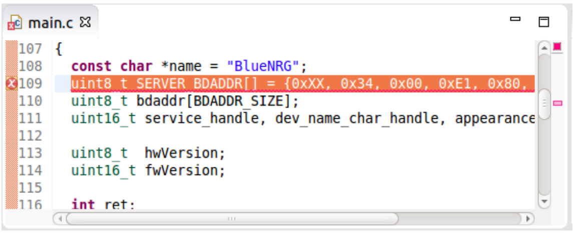
```
CDT Build Console [IoT_Sensor]
arm-none-eabi-gcc -mcpu=cortex-m0plus -mthumb -mfloat-abi=soft -std=c99 -DUSE_HAL
arm-none-eabi-gcc -mcpu=cortex-m0plus -mthumb -mfloat-abi=soft -std=c99 -DUSE_HAL
../../../../Src/main.c: In function 'main':
../../../../Src/main.c:109:30: error: invalid suffix "xXX" on integer constant
   uint8_t SERVER_BDADDR[] = {0xXX, 0x34, 0x00, 0xE1, 0x80, 0x03};
```



1 error, 2 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (1 item)				
invalid suffix "xXX" on integer constant	main.c	/IoT_Sensor/Applica	line 109	C/C++ Problem

- If you double-click on the error in the *Console* or in the *Problems* tabs, the IDE open the corresponding file and select the line which contains the error.



```
107 {
108   const char *name = "BlueNRG";
109   uint8_t SERVER_BDADDR[] = {0xXX, 0x34, 0x00, 0xE1, 0x80,
110   uint8_t bdaddr[BDADDR_SIZE];
111   uint16_t service_handle, dev_name_char_handle, appearance
112
113   uint8_t  hwVersion;
114   uint16_t fwVersion;
115
116   int ret;
```

IoT Sensor lab

Goal

The main goal of this Lab is to communicate sensors values (accelerometer) and LED state on gateway request and to drive the LED according to the gateway requests. The communication media is Bluetooth Low Energy.

On gateway request, the system has to be able to send the following values:

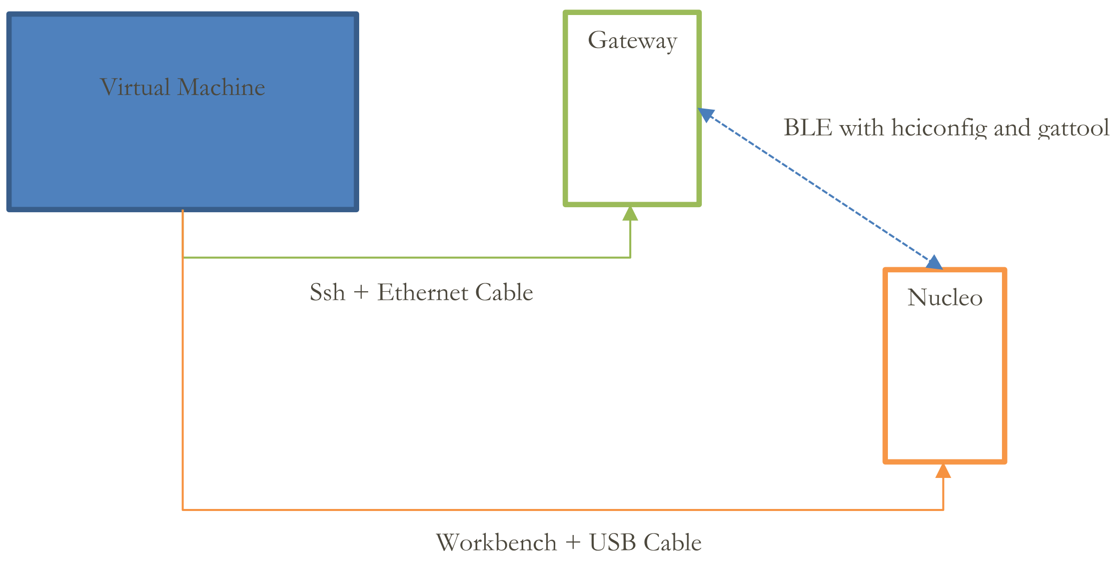
- Temperature
- Relative hygrometry
- Absolute 3 axes acceleration

- LED state

On gateway request, the system has to be able to:

- Drive the LED On or Off

System Architecture and tools



What is already done?

The provided Lab project is already able to communicate with the gateway via BLE but it currently exposes only the temperature and the relative humidity.

What you have to do?

You have to:

- change the BLE MAC address and give a unique address to your device.
- implement, on gateway request:
  - Accelerometer management:
    - 3 axis measurement and values transmission.
  - LED management:
    - Set or clear the LED regarding the gateway request
    - Returning the current LED state (On or Off)

What you have to implement?

This is a checklist of what you must add to your project to implement the missing functionalities.

Advice: don't try to implement all these features in only one step. Each section is a step by step procedure.

Set a Unique BLE MAC address

The BLE MAC address has to be unique in the classroom to differentiate each device in the classroom. The BLE MAC address is the Id which allows YOUR gateway to communicate with YOUR BLE device. So, you have to make a list of the used BLE MAC addresses in the classroom to be sure to have a unique one.

<u>Tasks list:</u>

- Set the BLE MAC Address:
  At the beginning of the `main()` function, **change the MSB(Most Significant Bit) of the BLE MAC address** to obtain an unique MAC address in the classroom. Be careful, the MSB is the 1st byte in the declaration. In the provided lab project, it's currently set to '`0xXX`' which generates an error if you compile without changing it.

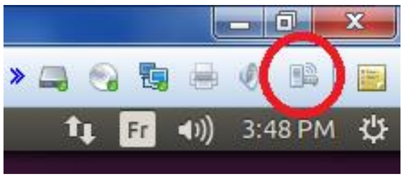
Deploying your project

Before going further within the Lab, it is important to test and validate that the Nucleo device is functional and operational.

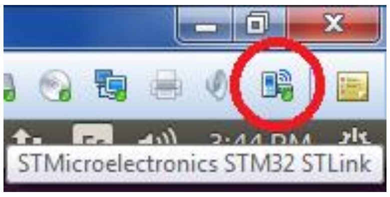
Connect the NUCLEO board to the VM

Connect the PC to the NUCLEO with a Mini USB cable. When the NUCLEO board is connected to the PC, in the Virtual Machine, you have to connect the STLink USB device (NUCLEO board) to the virtual machine instance.

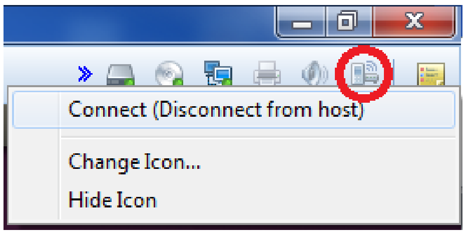
Disconnected state:



Connected state:



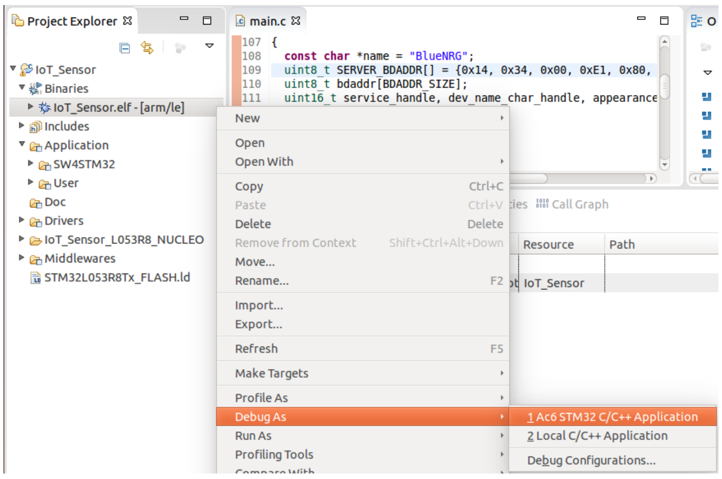
To connect the NUCLEO in the virtual machine, right-click on the icon and select *Connect*:



Flash the project binary

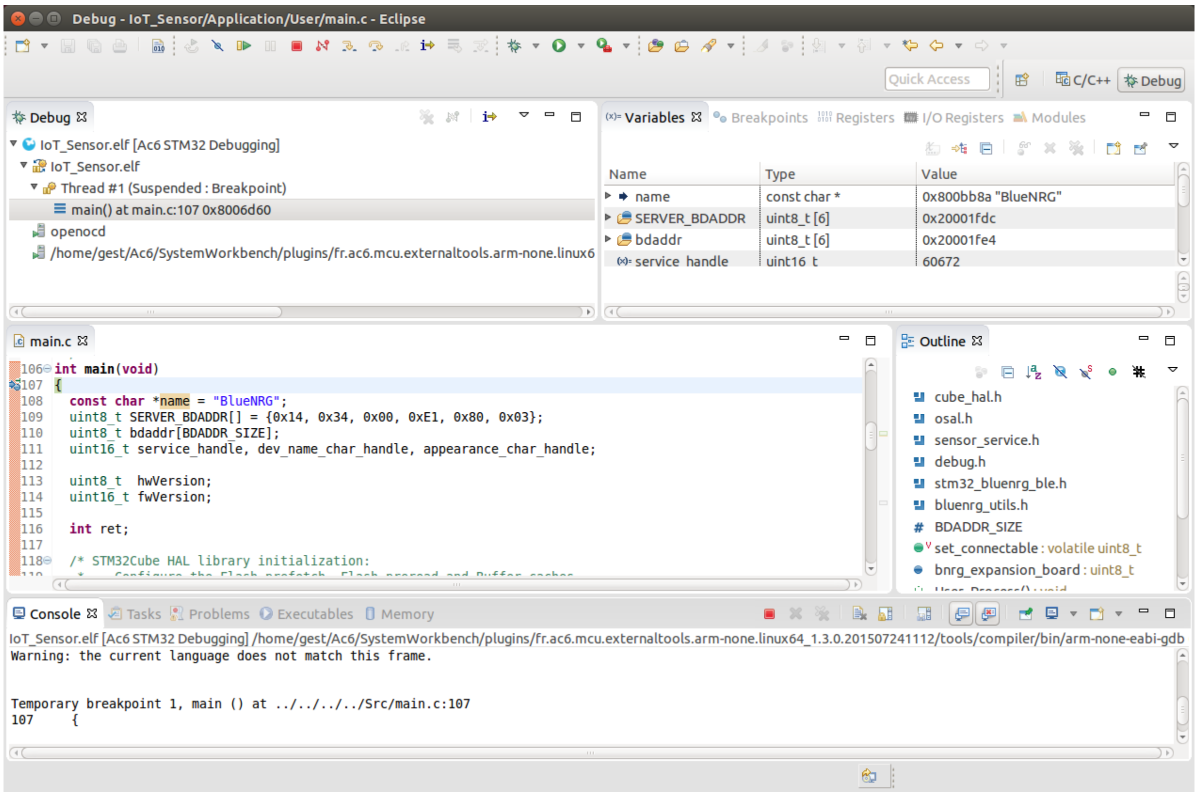
Before being able to flash the firmware, your project has to be generated without any error.

Right-click on the '*elf* file, then select *Debug As* / *1 Ac6 STM32 C/C++ Application* menu item.



```
{
  const char *name = "BlueNRG";
  uint8_t SERVER_BDADDR[] = {0x14, 0x34, 0x00, 0xE1, 0x80,
  uint8_t bdaddr[BDADDR_SIZE];
  uint16_t service_handle, dev_name_char_handle, appearance
```

If the NUCLEO board is correctly connected to the VM, a *Debug Perspective* should be displayed (with your permission at the first time) and the program should be stopped at the beginning of the main function:



```
IoT_Sensor.elf [Ac6 STM32 Debugging]
  IoT_Sensor.elf
    Thread #1 (Suspended : Breakpoint)
      main() at main.c:107 0x8006d60
  openocd
  /home/gest/Ac6/SystemWorkbench/plugins/fr.ac6.mcu.externaltools.arm-none.linux6
```

Name	Type	Value
name	const char *	0x800bb8a "BlueNRG"
SERVER_BDADDR	uint8_t [6]	0x20001fdc
bdaddr	uint8_t [6]	0x20001fe4
service_handle	uint16_t	60672

```
int main(void)
{
  const char *name = "BlueNRG";
  uint8_t SERVER_BDADDR[] = {0x14, 0x34, 0x00, 0xE1, 0x80, 0x03};
  uint8_t bdaddr[BDADDR_SIZE];
  uint16_t service_handle, dev_name_char_handle, appearance_char_handle;

  uint8_t  hwVersion;
  uint16_t fwVersion;

  int ret;

  /* STM32Cube HAL library initialization:
```

```
cube_hal.h
osal.h
sensor_service.h
debug.h
stm32_bluenrg_ble.h
bluenrg_utils.h
BDADDR_SIZE
set_connectable : volatile uint8_t
bnrg_expansion_board : uint8_t
```

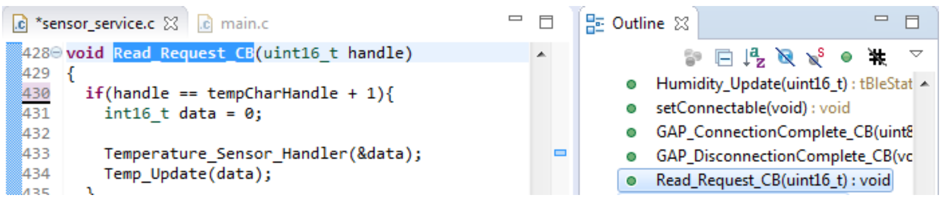
```
IoT_Sensor.elf [Ac6 STM32 Debugging] /home/gest/Ac6/SystemWorkbench/plugins/fr.ac6.mcu.externaltools.arm-none.linux64_1.3.0.201507241112/tools/compiler/bin/arm-none-eabi-gdb
Warning: the current language does not match this frame.


Temporary breakpoint 1, main () at ../../../../Src/main.c:107
107     {
```

S

Debugging your project

Breakpoint Management

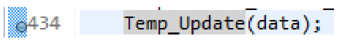
Open the Application/User/sensor_service.c file and in the *Outline* view, select the **`Read_Request_CB`** function. This action should select the corresponding function in the code editor view:



```
void Read_Request_CB(uint16_t handle)
{
  if(handle == tempCharHandle + 1){
    int16_t data = 0;

    Temperature_Sensor_Handler(&data);
    Temp_Update(data);
```

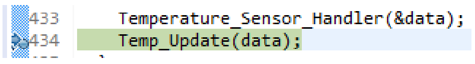
```
Humidity_Update(uint16_t) : tBleStat
setConnectable(void) : void
GAP_ConnectionComplete_CB(uint8
GAP_DisconnectionComplete_CB(vc
Read_Request_CB(uint16_t) : void
```

Select the `Temp_Update(data);` line and in the menu, select *Run / Toggle Breakpoint*. In the margin a circle should appear:



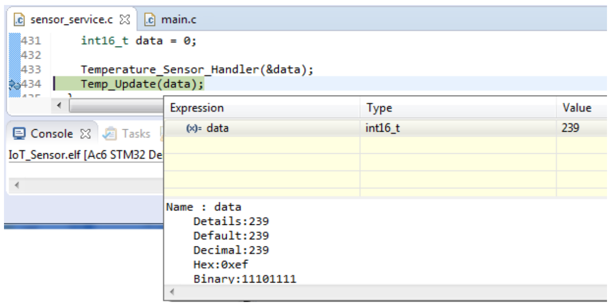
```
    Temp_Update(data);
```

Resume the execution and when the gateway requests the current temperature, the program should stop at the breakpoint and the little arrow shows the next line to be executed:



```
    Temperature_Sensor_Handler(&data);
    Temp_Update(data);
```

If you move over a variable with the mouse pointer, a context watch window appears displaying the current variable value (239 => 23.9°C in that case):



```
    int16_t data = 0;

    Temperature_Sensor_Handler(&data);
    Temp_Update(data);
```

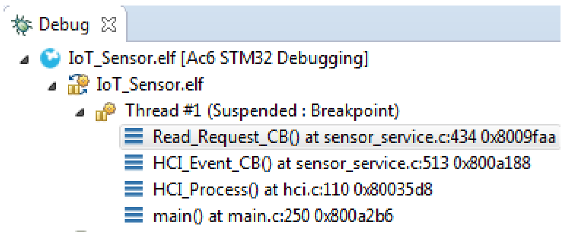
IoT_Sensor.elf [Ac6 STM32 De

Expression	Type	Value
(x)= data	int16_t	239

```
Name : data
    Details:239
    Default:239
    Decimal:239
    Hex:0xef
    Binary:11101111
```

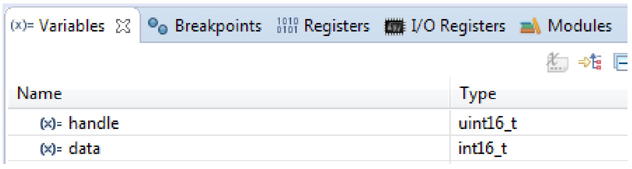
Views windows

The *Debug* view shows the call stack:



```
IoT_Sensor.elf [Ac6 STM32 Debugging]
  IoT_Sensor.elf
    Thread #1 (Suspended : Breakpoint)
      Read_Request_CB() at sensor_service.c:434 0x8009faa
      HCI_Event_CB() at sensor_service.c:513 0x800a188
      HCI_Process() at hci.c:110 0x80035d8
      main() at main.c:250 0x800a2b6
```

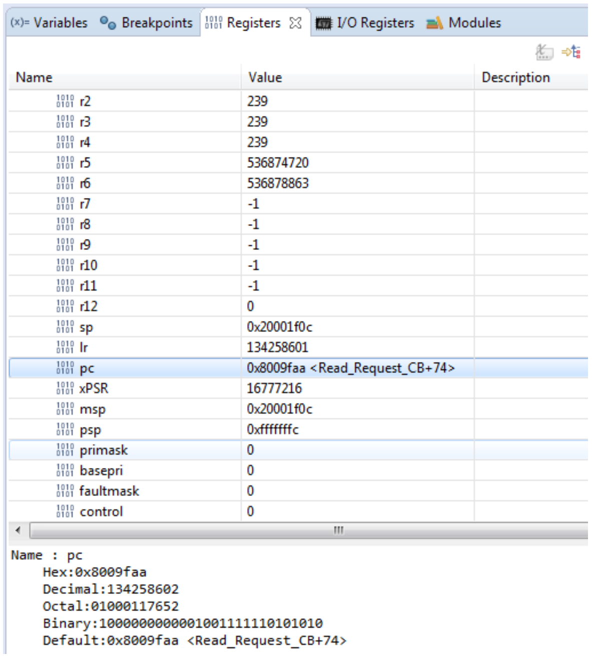
The *Variables* view shows the local variables value:



(x)= Variables | Breakpoints | Registers | I/O Registers | Modules

Name	Type
(x)= handle	uint16_t
(x)= data	int16_t

The *Registers* view shows the STM32 registers value:

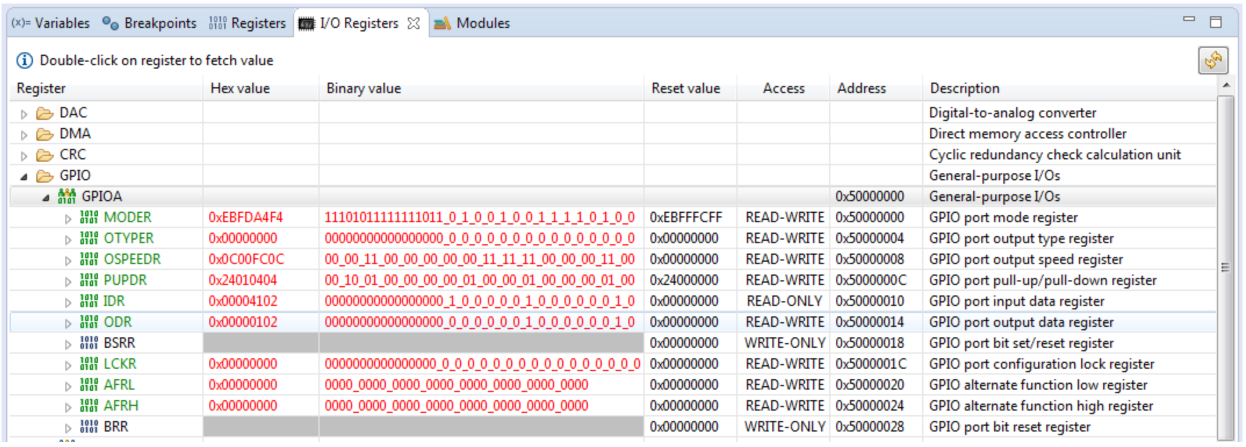


(x)= Variables | Breakpoints | Registers | I/O Registers | Modules

Name	Value	Description
r2	239	
r3	239	
r4	239	
r5	536874720	
r6	536878863	
r7	-1	
r8	-1	
r9	-1	
r10	-1	
r11	-1	
r12	0	
sp	0x20001f0c	
lr	134258601	
pc	0x8009faa <Read_Request_CB+74>	
xPSR	16777216	
msp	0x20001f0c	
psp	0xfffffffc	
primask	0	
basepri	0	
faultmask	0	
control	0	

```
Name : pc
    Hex:0x8009faa
    Decimal:134258602
    Octal:01000117652
    Binary:1000000000001001111110101010
    Default:0x8009faa <Read_Request_CB+74>
```

The *I/O Registers* shows the peripherals registers value:



(x)= Variables | Breakpoints | Registers | I/O Registers | Modules

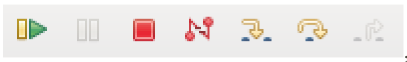
Double-click on register to fetch value

Register	Hex value	Binary value	Reset value	Access	Address	Description
DAC						Digital-to-analog converter
DMA						Direct memory access controller
CRC						Cyclic redundancy check calculation unit
GPIO						General-purpose I/Os
GPIOA					0x50000000	General-purpose I/Os
MODER	0xEBFDA4F4	1110101111111011_0_1_0_0_1_0_0_1_1_1_1_0_1_0_0	0xEBFFFCFF	READ-WRITE	0x50000000	GPIO port mode register
OTYPER	0x00000000	0000000000000000_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0	0x00000000	READ-WRITE	0x50000004	GPIO port output type register
OSPEEDR	0x0C00FC0C	00_00_11_00_00_00_00_00_11_11_11_00_00_00_11_00	0x00000000	READ-WRITE	0x50000008	GPIO port output speed register
PUPDR	0x24010404	00_10_01_00_00_00_00_01_00_00_01_00_00_00_01_00	0x24000000	READ-WRITE	0x5000000C	GPIO port pull-up/pull-down register
IDR	0x00004102	0000000000000000_1_0_0_0_0_0_1_0_0_0_0_0_0_1_0	0x00000000	READ-ONLY	0x50000010	GPIO port input data register
ODR	0x00000102	0000000000000000_0_0_0_0_0_0_1_0_0_0_0_0_0_1_0	0x00000000	READ-WRITE	0x50000014	GPIO port output data register
BSRR			0x00000000	WRITE-ONLY	0x50000018	GPIO port bit set/reset register
LCKR	0x00000000	000000000000000_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0	0x00000000	READ-WRITE	0x5000001C	GPIO port configuration lock register
AFRL	0x00000000	0000_0000_0000_0000_0000_0000_0000_0000	0x00000000	READ-WRITE	0x50000020	GPIO alternate function low register
AFRH	0x00000000	0000_0000_0000_0000_0000_0000_0000_0000	0x00000000	READ-WRITE	0x50000024	GPIO alternate function high register
BRR			0x00000000	WRITE-ONLY	0x50000028	GPIO port bit reset register

Debugger command summary.

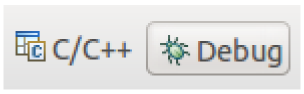
From the *Run* menu you can:

- *Instruction stepping mode*: To go through the code step by step. In Disassembly view, the step by step mode is attached to the assembler code.
- *Skip all breakpoints:* To disable all the breakpoint previously set.
- *Step over:* To step over (without entering) the next function at the currently executing line code (while you are in suspended mode).
- *Step into:* To step into the next function at the currently executing line code (while you are in suspended mode).
- *Resume:* To resume the execution of the currently suspended debug target.
- *Suspend:* To halt the execution of the currently selected thread in the debug target. In suspended mode, you can examine the stack frames.
- *Terminate:* To terminate the selected debug target.
- *Disconnect:* To terminate the connection between the debugger and the remote debug target.
- *Remove:* To remove the selected debug target (if the target is terminated).

With these toolbar debug buttons , you can (from left to right): *Resume*, *Suspend*, *Terminate*, *Disconnect*, *Step into*, *Step Over* and *Step return*.

Switching between Editing and Debugging

You can switch between Edit and Debug perspectives by clicking on the corresponding button:

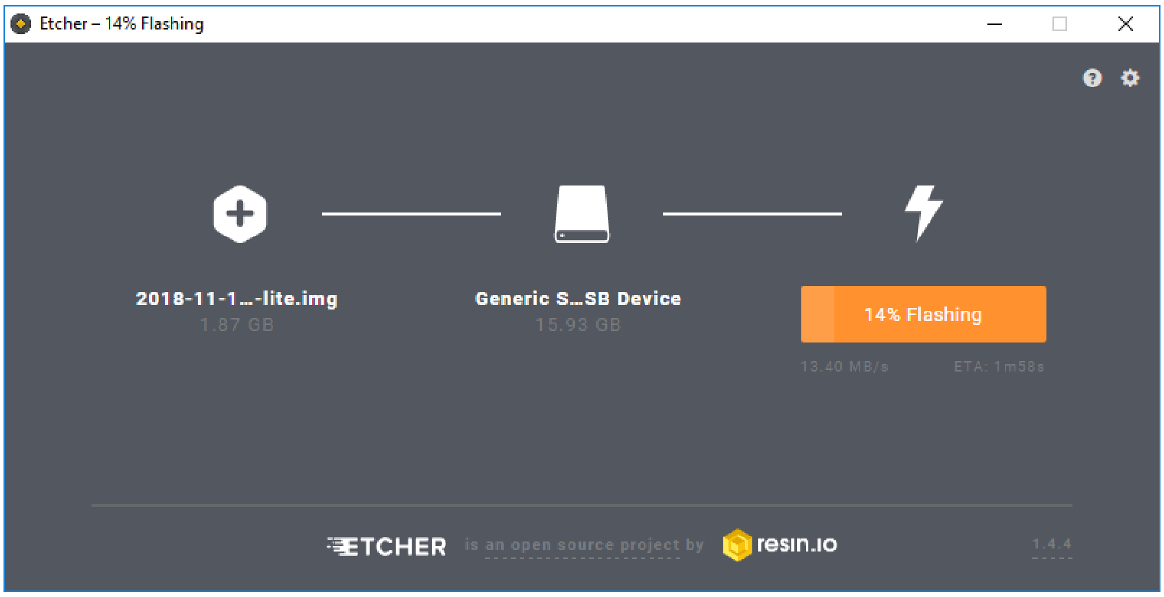


Bring up the gateway

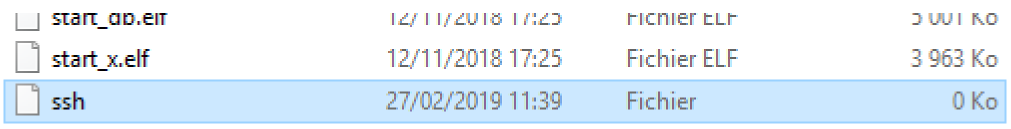
In order to perform the Lab instructions, it is required to program the Raspberry Pi platform that will be used as a Gateway in the next project steps. For this, it is required to download the Raspbian Stretch Lite image from this page https://www.raspberrypi.org/downloads/raspbian/ or simply use the direct download link https://downloads.raspberrypi.org/raspbian_lite_latest

Flash the Operating System

To program the micro SDCard provided with the Raspberry Pi 3 platform, follow the instructions available at this address : https://www.raspberrypi.org/documentation/installation/installing-images/README.md



Once done it is mandatory to change the system parameters to first enable the ssh access. From you windows explorer create a file called ssh on the SDCard drive identified as boot.



start_db.elf	12/11/2018 17:25	Fichier ELF	5 001 Ko
start_x.elf	12/11/2018 17:25	Fichier ELF	3 963 Ko
ssh	27/02/2019 11:39	Fichier	0 Ko

Now you are ready to power up the board and identify its IP address.

Connect platform to secured Wifi

The Raspberry Pi can be configured to automatically connect to a secure wireless network. For this you have to create a file on the SDCard called wpa_supplicant.conf. The file should contain the following details:

For Raspbian Jessie:

```
network={
    ssid="YOUR_NETWORK_NAME"
    psk="YOUR_PASSWORD"
    key_mgmt=WPA-PSK
}
```

For Raspbian Stretch:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
network={
    ssid="YOUR_NETWORK_NAME"
    psk="YOUR_PASSWORD"
    key_mgmt=WPA-PSK
}
```

With this file in place, Raspbian will move it in /etc/wpa_supplicant/ when the system is booted.

Connect platform to unsecured Wifi

The Raspberry Pi can be configured to automatically connect to a unsecure wireless network (wireless access with no password). For this you have to create a file on the SDCard called wpa_supplicant.conf. The file should contain the following details:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev # Include this
line for Stretch
network={
    ssid="YOUR_NETWORK_NAME"
    key_mgmt=NONE
}
```

With this file in place, Raspbian will move it in /etc/wpa_supplicant/ when the system is booted.

Connect over ssh to the platform

To connect to the target platform over ssh you have to use the following command :

```
[user@machine ~]$ ssh pi@<Gateway ip>
```

The requested password is the one by default (**raspberry**)for the Raspbian OS.

Update local platform list

Before starting any system package installation, it is mandatory to update the local packages repository.

```
[pi@raspberrypi ~]$ sudo apt-get update
```

```
Get:1 http://raspbian.raspberrypi.org/raspbian stretch InRelease [15.0
kB]
Get:2 http://archive.raspberrypi.org/debian stretch InRelease [25.4 kB]
Get:3 http://raspbian.raspberrypi.org/raspbian stretch/main armhf
Packages [11.7 MB]
Get:4 http://archive.raspberrypi.org/debian stretch/main armhf Packages
[212 kB]
Get:5 http://archive.raspberrypi.org/debian stretch/ui armhf Packages
[42.0 kB]
Get:6 http://raspbian.raspberrypi.org/raspbian stretch/non-free armhf
Packages [95.5 kB]
Fetched 12.1 MB in 21s (558 kB/s)
Reading package lists... Done
```

Install platform required packages

Docker

The hosted services on the gateway will be contained into docker images, so docker is a required package for the next steps.

```
[pi@raspberrypi ~]$ sudo curl -sSL get.docker.com | sh
```

Add current user to the docker user group.

```
[pi@raspberrypi ~]$ sudo usermod -aG docker pi
```

Testing with the gateway

Connect to the Gateway

1. Boot from SD Card
2. Insert the micro SD card in the board and power up the board.
3. Once you found your IP you can connect to it with

```
[user@machine ~]$ ssh pi@<Gateway ip>
```

Use the following credentials:

Login: pi

Password: **raspberry**

Activate or restart a device:

```
pi@raspberry:~$ sudo hciconfig
pi@raspberry:~$ sudo hciconfig hciX up
pi@raspberry:~$ sudo hciconfig hciX reset
```

Once you have the mac address you can communicate with your device as this below:

List the in range BLE devices

You can list the in range device using hcitool.

```
pi@raspberry:~$ hcitool lescan
LE Scan ...
2F:F2:EC:33:F9:B2 (unknown)
54:DB:30:A6:67:D4 (unknown)
54:DB:30:A6:67:D4 (unknown)
18:3E:81:7D:B4:C6 (unknown)
38:B3:57:B3:81:4E (unknown)
06:32:A1:E4:88:E6 (unknown)
02:80:E1:00:34:XX (unknown)
```

Connect according the BLE MAC address set in the NUCLEO firmware

```
pi@raspberry:~$ gatttool -I -b 02:80:E1:00:34:XX
[   ][02:80:E1:00:34:12][LE]> connect
```

In the command above, replace 03 by 02 in MAC address if the BLE Shield isn't the reference IDB04A1

Read services and characteristics

```
[CON][02:80:E1:00:34:12][LE]> primary
[CON][02:80:E1:00:34:12][LE]>
attr handle: 0x0001, end grp handle: 0x0004 uuid: 00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x0005, end grp handle: 0x000b uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x000c, end grp handle: 0x000e uuid: 01366e80-cf3a-11e1-9ab4-0002a5d5c51b
attr handle: 0x0013, end grp handle: 0x0019 uuid: 04366e80-cf3a-11e1-9ab4-0002a5d5c51b
attr handle: 0x001d, end grp handle: 0x001f uuid: 0b366e80-cf3a-11e1-9ab4-0002a5d5c51b
[CON][02:80:E1:00:34:12][LE]> characteristics
[CON][02:80:E1:00:34:12][LE]>
handle: 0x0002, char properties: 0x20, char value handle: 0x0003, uuid: 00002a05-0000-
1000-8000-00805f9b34fb
handle: 0x0006, char properties: 0x4e, char value handle: 0x0007, uuid: 00002a00-0000-
1000-8000-00805f9b34fb
handle: 0x0008, char properties: 0x4e, char value handle: 0x0009, uuid: 00002a01-0000-
1000-8000-00805f9b34fb
handle: 0x000a, char properties: 0x0a, char value handle: 0x000b, uuid: 00002a04-0000-
1000-8000-00805f9b34fb
handle: 0x000d, char properties: 0x02, char value handle: 0x000e, uuid: 03366e80-cf3a-
11e1-9ab4-0002a5d5c51b
handle: 0x0014, char properties: 0x02, char value handle: 0x0015, uuid: 05366e80-cf3a-
11e1-9ab4-0002a5d5c51b
handle: 0x0017, char properties: 0x02, char value handle: 0x0018, uuid: 07366e80-cf3a-
11e1-9ab4-0002a5d5c51b
handle: 0x001e, char properties: 0x0e, char value handle: 0x001f, uuid: 0c366e80-cf3a-
11e1-9ab4-0002a5d5c51b
```

Taking a look at the [Application/User/sensor_service.c] file, you should able to recognize the UUID's displayed above:

- Green bytes: service UUID differentiating bytes
- Cyan bytes: characteristic UUID differentiating bytes which give the corresponding handle in red

Here is the table of correspondence between value items and its handle:

UUID	Handle	Access	Item
`03366e80-cf3a-11e1-9ab4-0002a5d5c51b`	`0x000e`	Read	Accelerator values
`05366e80-cf3a-11e1-9ab4-0002a5d5c51b`	`0x0015`	Read	Temperature value
`07366e80-cf3a-11e1-9ab4-0002a5d5c51b`	`0x0018`	Read	Relative Humidity value
`0c366e80-cf3a-11e1-9ab4-0002a5d5c51b`	`0x001f`	Read / Write	LED status

You may not have the same handle like above, use UUID and Item fields to determine the right value of Handle.

The Grayed lines should only be visible at the end of the Lab. It is provided as reference, as only Temperature and Humidity UUID are implemented in the provided project.

Read sensor values with the *char-read-hnd* command:

Temperature handle: 0x0015

```
 [CON][02:80:E1:00:34:12][LE]> char-read-hnd 0x0015
Characteristic value/descriptor: ee 00
```

Temperature value = 0x00EE = 238 = 23.8°C
Verify also the Humidity sensor…

Gattool is a solution to test the services provided by a Bluetooth Low Energy device. We are using it today to ensure that the service are correctly implemented and functional.

Accelerometer management

Tasks list:

- Initialize and enable the Accelerometer peripheral:
  The **main()** function already calls 2 sub-functions to initialize and enable the temperature and the hygrometry sensors : **initializeAllSensors()** and **enableAllSensors()**. So, you have to complete these 2 functions to initialize the Accelerometer driver.
  Tips: *ACCELERO_SENSORS_AUTO* could be useful…
  [Application/User/sensor_service.c]
- Expose the Accelerometer measure to the gateway:
  a. Create an Accelerometer Characteristic and an Accelerometer Service UUID constants.
     Looking at the other UUID declarations, create this following constants:
     - **COPY_ACC_SERVICE_UUID**:
     1B.C5.D5.A5.02.00 – B4.9A – E1.11 – 3A.CF – 80.6E.36.**01**
     - **COPY_ACC_UUID**:
     1B.C5.D5.A5.02.00 – B4.9A – E1.11 – 3A.CF – 80.6E.36.**03**
     [Application/User/sensor_service.c]
  b. Add an Accelerator Service:
     Take a look at the **Add_Environmental_Sensor_Service()** function and create an **Add_Acc_Service()** function to add the Accelerator Service.
     [Application/User/sensor_service.c
  c. Call the **Add_Acc_Service()** function:
     In the **main()** function, call your **Add_Acc_Service()** function close to the existing **Add_Environmental_Sensor_Service()** function.
     [Application/User/main.c]
- Answer to the gateway read requests
  a. Read the accelerometer sensor:
     Take a look at the **Temperature_Sensor_Handler()** function and create an **Accelero_Sensor_Handler()** function to read the sensor.
     [Application/User/ sensor_service.c]
  b. Update the Accelerometer BLE Characteristic value:
     Take a look at the **Temp_Update()** function and create an **Acc_Update(AxesRaw_t *data)**
     [Application/User/ sensor_service.c]
  c. Answer to the gateway request:
     Take a look at the **Read_Request_CB()** callback function and complete it by adding a call to your **Add_Environmental_Sensor_Service()** and your **Acc_Update()** functions.
     [Application/User/sensor_service.c]

Testing with the gateway

As described in previous step, read the Accelerometer sensor (handle 0x000E) with the NUCLEO board in normal position on the desktop:

- You should read values close to '`f1 ff` `16 00` `de 03`'
- Which gives `0xFFF1`, `0x0016`, `0x03DE` in hex
- Which gives -15, +22, +990 (in mg)
- Which is close to 0g, 0g, +1g (x, y, z axis)

And if you remember what you learnt at school, you know that 1g is the hearth gravity…

Now put the board on one of its side on the desktop. Read again the Accelerometer values. You should find the +/-1g value on another axe…

LED2 Management

Tasks list:

- Initialize LED2 GPIO:
  In the **`main()`** function, initialize the LED GPIO with the BSP API. To achieve this task, take a look at the Drivers/BSP/STM32L0xx_Nucleo/stm32l0xx_nucleo.c file. You should find everything to enable the LED2.
- Expose the LED2 services to the gateway:
  a. Create an LED2 Characteristic and an LED2 Service UUID constants:
  Looking at the other UUID declarations, create this following constants:
  `- COPY_LED_SERVICE_UUID`: ending with **0B**
  `- COPY_LED_UUID`: ending with **0C**
  b. Add an LED Service:
  Similar to the **`Add_Acc_Service()`** you did, create an **`Add_LED_Service()`** function to add the LED Service. Please note that this service is Read/Write opposed to the Accelerator one.
  c. Ensure that the **`Add_LED_Service()`** function is called.
- Answer to the gateway **write requests**:
  Update the LED state:
  In the **`Attribute_Modified_CB(uint16_t handle, uint8_t data_length, uint8_t *att_data)`** callback, drive the LED according to the parameters of the callback:
  `- handle` should be equal to **`(ledCharHandle + 1)`**
  `- data_length` should be equal to **`sizeof(uint8_t)`**
  `- att_data` is a pointer to the gateway data. If the pointed value is equal to 0, then clear the LED else set the LED.
- Answer to the gateway **read requests**:
  a. Update the LED BLE Characteristic value:
  Create an **`LedState_Update(uint8_t ledState)`** function.
  b. Answer to the gateway request:
  Take a look at the **`Read_Request_CB()`** callback function and complete it by adding a call to your **`LedState_Update()`** function with the saved current LED state.

Testing with the gateway

- Read the LED status (handle 0x001F): '`char-read-hnd 0x001F`'. The return value should be 0.
- Set the LED to on: '`char-write-cmd 0x001f 01`'. The LED should be on.
- Read the LED status again. The return value should be 1.
- Set the LED to on: '`char-write-cmd 0x001f 00`'. The LED should return to off state.
- Read the LED status again. The return value should be 0.