# Supply chain support

## Project description

The aim of the project is to develop asset tracking system that will enable monitoring of the location and status of the goods being transported.

The system comprises of following components:

* Ocelot embedded gateway implementing 3G radio and GPS receiver
* temperature, humidity and accelerometer sensors
* Communication protocol
* Server-side application
* Web application

The embedded gateway reads the sensor values and the current position from GPS and sends them to the server using a dedicated communication protocol. The gateway is based on STM32 family microcontroller running FreeRTOS. The data is received by the server application and stored into database.

A web-based dashboard will display position of the monitored asset on the map including track (past positions). Also, current and historical sensor values will be displayed.

## Sprint 01 – (weeks 2 and 3)

During the first 14 day sprint, the main goal is to get acquainted with the technologies that will be used for the solution implementation and to generate a technical description of the solution.

Following topics should be investigated and covered in the Technical specification document:

### FreeRTOS basics

FreeRTOS will be used on Ocelot embedded gateway, so before starting with development, you have to get acquainted with some basic concepts (tasks, priorities, queues, semaphores..). You will find plenty of resources on the Internet including the free book written by the author of the OS Richard Barry.

|  |
| --- |
| https://www.freertos.org/ |

### STM32 basics

Ocelot embedded gateway is built around ST Microelectronics Cortex-M ARM microcontroller. This is well known 32-bit architecture used in embedded systems and before starting to work with it, you should understand the basics of the architecture and the peripheries it implements (UART, GPIO, i2c, SPI, ADC...)

|  |
| --- |
| https://www.st.com/en/microcontrollers/stm32-32-bit-arm-cortex-mcus.html |

### Server side architecture

Based on the project requirements, your skills and previous experiences in development, select the appropriate server application architecture. You have to define following:

* web server (Apache, ...?)
* programming framework and language (Java, Python, .NET...?)
* Database (mySql, ..?)
* web framework (jQuery, NodeJS, Angular, ...?)

### Communication protocol

It is important to select and specify the appropriate communication protocol over 3G/4G radio based on the project requirements. There are many available application protocols that can be used such as (HTTP, HTTPS, MQTT) or you can specify own messaging over TCP or UDP. Bear in mind some specifics related to mobile networks:

* IP address you get will be dynamic
* IP address will be in the private range
* this means that the device (embedded computer) will have to initiate and maintain the connection. It will not be possible to address the device from the server, the device must initiate the connection.

### Eclipse IDE installation

STM32 System Workbench will be used for developing the code for Ocelot embedded gateway, so it has to be installed and configured. It is a free tool and can be downloaded and installed using following instructions:

Download System Workbench for STM32

|  |
| --- |
| https://www.st.com/en/development-tools/sw4stm32.html |

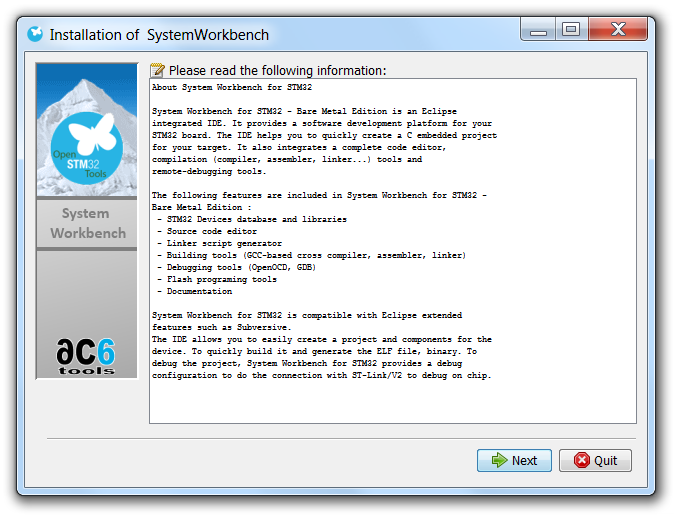
First register on openstm32.com

|  |
| --- |
| <http://www.openstm32.org> |

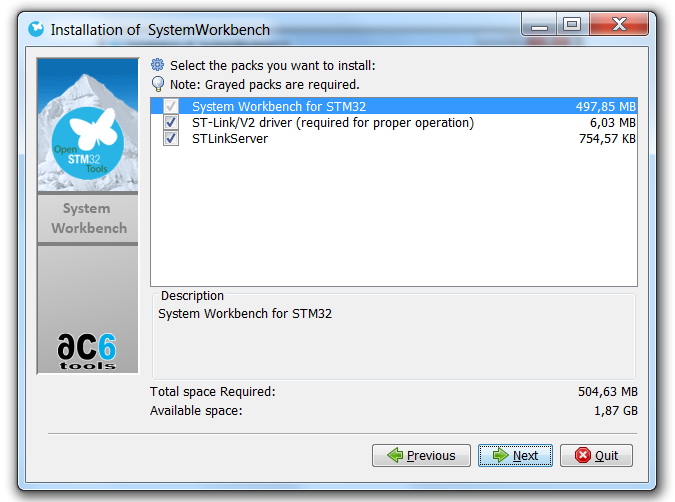
After registration, read available information on openstm32 site, especially one related to the System Workbench for STM32. About installation procedure, download versions etc.

Choose full installer that will install both Eclipse and required plugins.

Start the installation and follow instructions:



Follow default configuration:



## Sprint 02

### STM32 Nucleo-64

We will be developing STM32 software on STM32 Nucleo-64 development board first and move to Ocelot board later. The aim is to cover basic STM32 microcontroller programming on the more friendly Nucleo board.

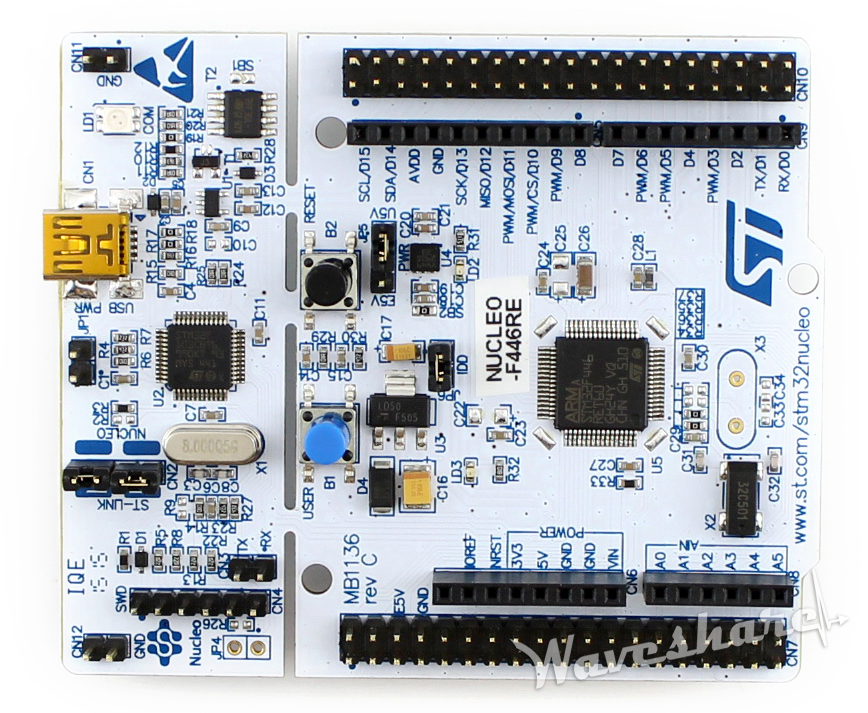


Figure - STM32 Nucleo-64 board

In this sprint you must write the C application that runs on Nucleo board that has following functionality:

1. Blinking onboard LED on Nucleo-64 development board
2. Writing „Hello world“ to serial port

### STM32 Cube MX

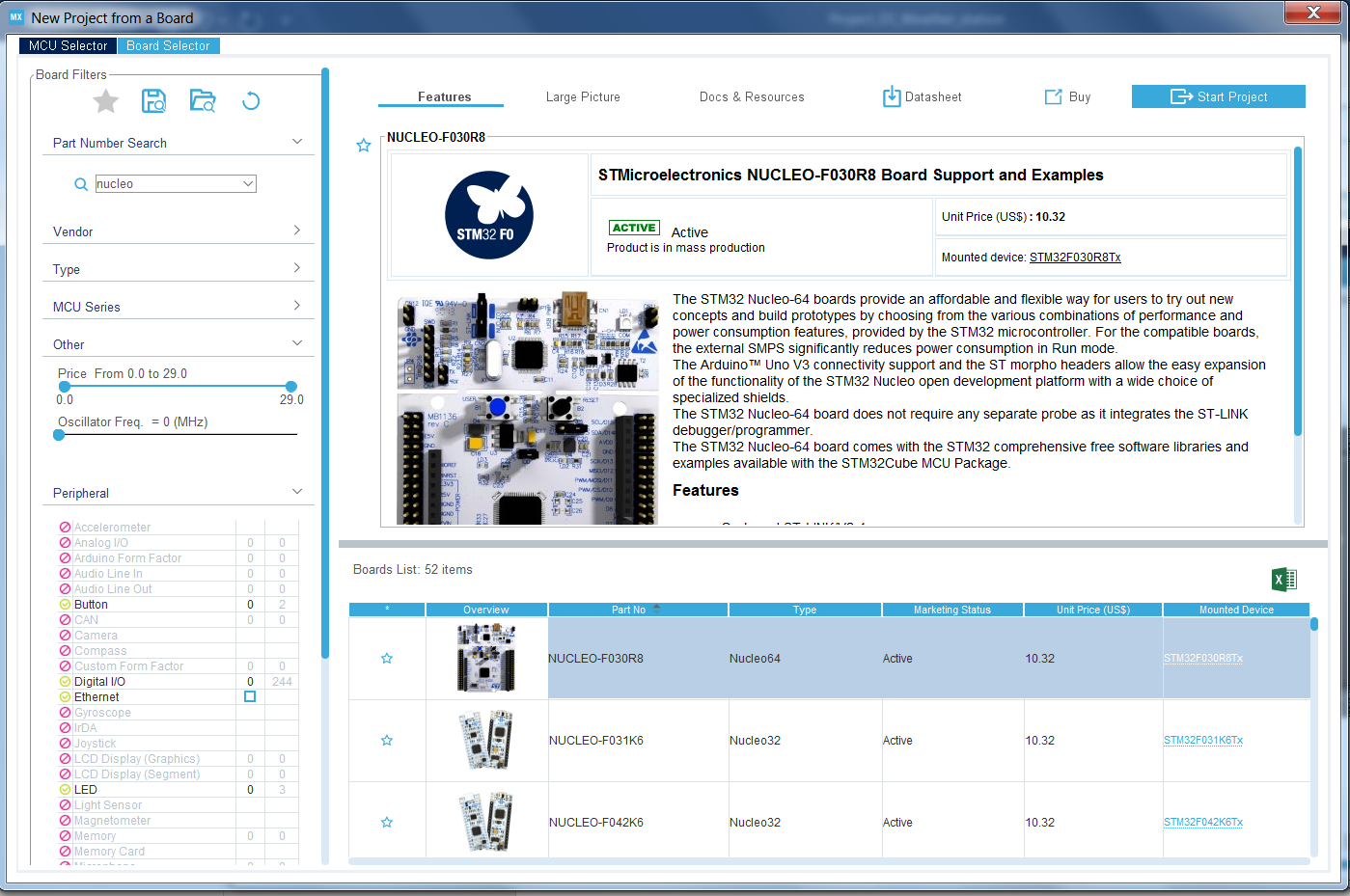
To make it easier creating new projects and configuring microcontroller registers, STM32 has provided free tool: STM32 CubeMX.

|  |
| --- |
| <https://www.st.com/en/development-tools/stm32cubemx.html> |

STM32CubeMX is a graphical tool that allows a very easy configuration of STM32 microcontrollers and microprocessors, as well as the generation of the corresponding initialization C code.

Download the STM32 CubeMX tool, install it and learn how to use it from the available documentation.

In STM32 CubeMX select the Nucleo-F030R8 development boad and start a new project.



Make sure to select SW4STM32 as a development tool in project manager:



Follow the instructions, configure the tool and generate the skeleton code. Import this code in SW4STM32 IDE (Eclipse), verify it cmpiles and then start implementing your code for blinking the LED and writing to console. You will be able to see console printouts on a virtual COM port over USB.

Learn how to use ST-Link debugger by setting breakpoints in SW4STM32 IDE.

By covering all this you will learn the basics of STM32 programming which is needed for writing code for modem controll and sensor reading which will be your task in the next sprints.

## Sprint 03

In this sprint, you will start using Ocelot target platform.

A circuit board

Description automatically generated

Figure - Ocelot Industrial Gateway

Ocelot is an industrial embedded gateway running on powerful STM32F437 – 32 bit ARM Cortex M4 microcontroller. It packs various communication interfaces – 3G or NB-IoT modem, WiFi, CAN, RS485. It also has onboard GPS receiver and can be further expanded with LoRa or other interfaces.

### Getting started with Ocelot

You will use Ocelot in a similar way to the Nucleo board from the last sprint. After all, it is the same microprocessor architecture, so you will be using same set of tools like with Nucleo board, most important of which will be ***STM32 System Workbench*** which you have installed in the first sprint and used in second sprint. There are however some differences, while you had to generate the project for Nucleo board from scratch in ***STM32 Cube MX***, here you will not have to repeat that step, you will start with already created project that contains everything needed to compile basic Ocelot application.

You will connect to Ocelot board using ST-Link V2 debugger according to the following diagram:

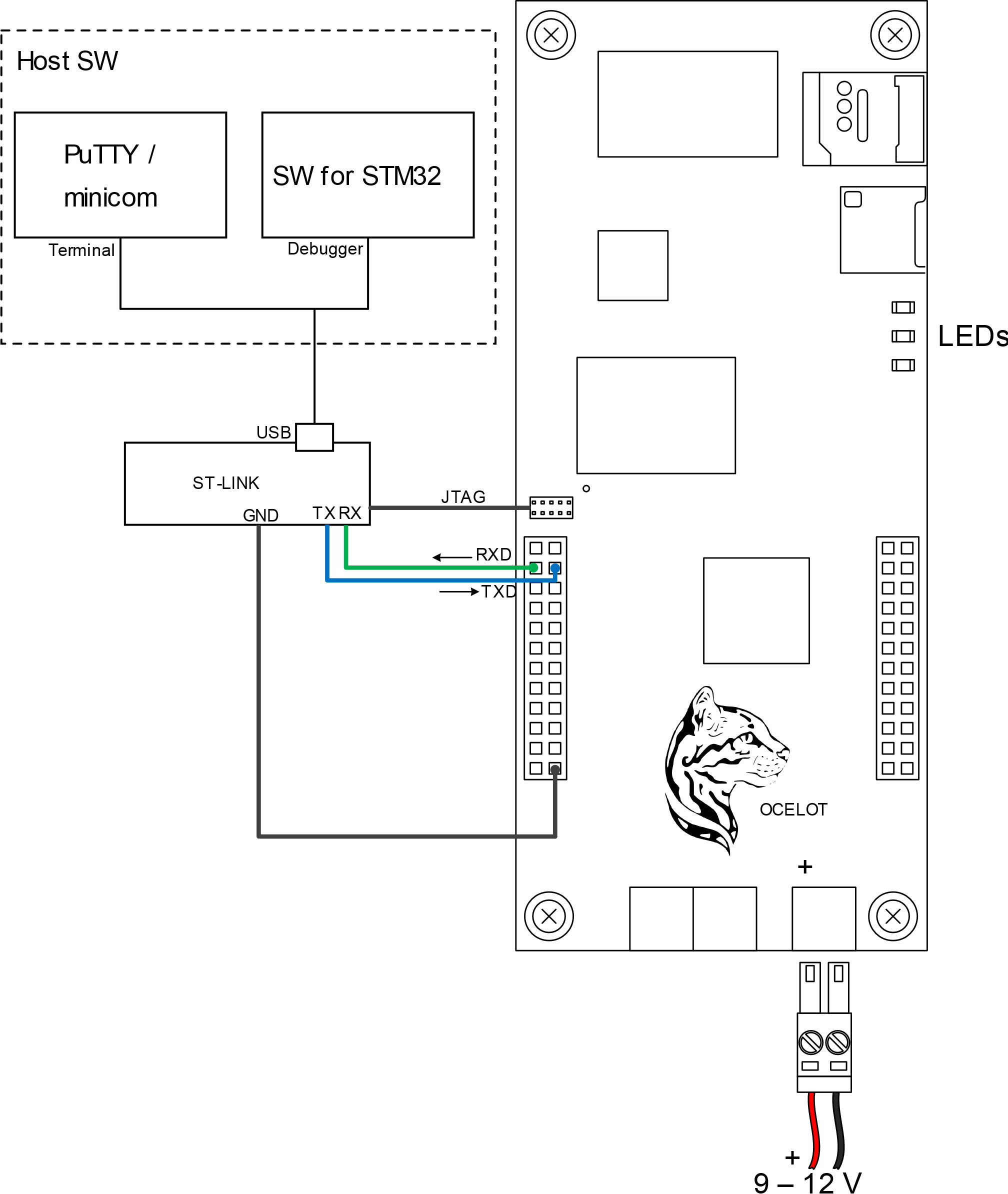


Figure - Interfacing Ocelot

ST-LINK serves as a JTAG debugger used by ***STM32 System Workbench*** IDE and also as a USB virtual COM port device connected to one of the UARTs available on Ocelot.

Open the following project in the ***weather-io*** repository:

|  |
| --- |
| weather-io\device\IOT\Ocelot\Projects\Weather\_3G\SW4STM32\proj\ .project |

Open it in the ***STM32 System Workbench*** IDE workspace, compile it and run on Ocelot. You should see LEDs on Ocelot blinking. Analyze the code and try to understand how it works. Identify the tasks, how they are defined and how they are started. Get acquainted with the basic directory structure (Projects, Drivers, Middlewares)..

### Bringing up the modem

|  |
| --- |
| **WARNING:** Connect 3G antenna to Ocelot before configuring modem. Be careful when handling antenna pigtail uFL connector as it is fragile. |

|  |
| --- |
| **WARNING:** Insert SIM card to Ocelot before configuring modem. Be careful not to break SIM card holder contacts when inserting or removing the card, because they are very sensitive. |

Now that we have development environment and Ocelot running, it is time to bring up the modem. Ocelot is equipped with uBlox SARA U270 3G modem which is connected to microcontroller using serial port (USART3). Other serial port (USART6) is connected to ST-LINK serial to USB interface which can be used as terminal . Project **modem\_passtrough** implements two tasks that forward terminal to modem and modem to terminal. This way user has the same feeling as if terminal was directly connected to the modem without microcontroller in the middle.

This is a good way to start experimenting with various modem configurations by typing AT commands in terminal and analyzing modem responses. Use ***Ublox Sara U270 AT Commands manual*** as a reference. Your first goal is to manually configure modem to connect to the Internet and then to send the data in UDP packets. You should be able to see your packets on the server.

Later you will automatize this scenario by sending the same set of AT commands from dedicated task on the microcontroller.

### Implementing main sending logic on Ocelot

Based on AT commands you have used in the previous exercise, write a main task that will configure the modem and then send data (position, temperature, humidity) to the server once per minute. Use dummy values for data because we are still not using real sensors in this phase.

Use ***atparser library*** for sending AT commands to modem and reading and interpreting responses. atparser library is a set of wrappers for interfacing modem. Here is one example of the code:

|  |
| --- |
| /\* init modem parser \*/  modem\_parser\_handle **=** atparser\_create**(**modem\_uart\_handle**);**  configASSERT**(**modem\_parser\_handle**);**  /\* power up modem \*/  **for(**int i**=**1**;** **!**success **&&** i**<=**10**;** i**++)**  **{**  ssLoggingPrint**(**ESsLoggingLevel\_Info**,** 0**,** "Power up modem."**);**  modem\_power\_up**();**  osDelay**(**1000**\***i**);**  success **=** atparser\_send**(**modem\_parser\_handle**,** "AT"**)** **&&** atparser\_recv**(**modem\_parser\_handle**,** "OK"**);**  **if(**success**)**  **{**  ssLoggingPrint**(**ESsLoggingLevel\_Info**,** 0**,** "Modem powered up successfully."**);**  **}**  **else**  **{**  ssLoggingPrint**(**ESsLoggingLevel\_Warning**,** 0**,** "Modem power up failed."**);**  **}**  **}** |

It is easy to understand the paradigm where you send commands using ***atparser\_send()*** function and receive result using ***atparser\_recv()*** function. They almost always come in pairs.

In the same way, if you are expecting some results within the response string, you can add parameters to the atparser\_recv():

|  |
| --- |
| int status**;**  **if(**atparser\_send**(**modem\_parser\_handle**,** "AT+CGATT?"**)** **&&** atparser\_recv**(**modem\_parser\_handle**,** "+CGATT: %d\n"**,** **&**status**))**  **{**  **if(**status **==** 1**)**  **{**  ret **=** true**;**  **}**  **else**  **{** |

In this example we expect the modem to answer our “AT+CGATT” command with the attach status;

|  |
| --- |
| +CGATT: 0  or  +CGATT: 1 |

In our function call, the result of the operation (0 or 1) will be stored to status variable which we can then use in the code, like in the given example, to evaluate the success of the operation. The formatting syntax is the same as for the scanf() C function (i.e %d is integer).

Atparser functions implement timeout that can be configured, so they will not block indefinitely if the modem does not respond.

When implementing this task, disable ***ModemToUartTask*** and ***UartToModemTask*** tasks that were used to forward data between modem and terminal. They are not needed because our new task will handle all modem configuration and communication. We can now use terminal for debug prints by using ssLoggingPrint () function:

|  |
| --- |
| ssLoggingPrint**(**ESsLoggingLevel\_Info**,** 0**,** "Initializing tasks..."**);** |

The given string will appear on the terminal.

For this excersise, for simplicity sake, use plain UDP or TCP sockets for sending data to the server. You will find AT commands for creating UDP or TCP socket in AT Command manual of the modem. When you get this working, consider porting some higher level protocol suchh as MQTT to the Ocelot, but first make it communicate using plain sockets. You can write small UDP server on server side that will pack received UDP payloads to MQTT topics and publish it on the server. That way, Ocelot can send raw, but your Azzure thigy can still subscribe to MQTT to get the data.

## Sprint 04

### Reading GPS position

Ocelot is equipped with the GPS/GNSS receiver – which is a little square module with the uBlox MAX-M8Q label on it. We will be using this module to determine our position. Before using it, you have to connect GPS antenna to the uFL RF connector next to the module.

GPS module uses UART serial port for communication with the microcontroller and is wired to the USART1 (PA9, PA10) port of the STM32 microcontroller on Ocelot.

You can configure and read/write this port in the same way as you used the USART for the modem.

Read the GPS module user manual to learn about the protocol and implement reading the position (longitude, latitude) from the module. The protocol is simple, the module periodically sends NMEA sentences containing various information. You just have to parse the appropriate string and extract position from it.

Pack the acquired position and send it to the server in UDP packets.

### Displaying position

Use Google Maps API or simmilar to display the current position of the device on the map. Implement storing of previous positions in the database and displaying them on the map as set of points connected with lines so that a path is visible.

## General remarks

You should document everything you do, all the information you gather during the sprint, describe the solution, draw diagrams, elaborate your decisions, explain why you have chosen one method or technology compared to the other. All sources and documents should be pushed to the github project repository