

---

# **Documentation LoRa WAN Sécurisé**

**François Sevaux, Arthur Le Rest | M1 CSSE UBS Lorient**

**déc. 20, 2019**



---

## Contents:

---

<b>1</b>	<b>Utilisation de LoRaMac-node</b>	<b>1</b>
1.1	Installation . . . . .	1
1.2	Charger un programme dans la carte . . . . .	2
1.3	Faciliter la compilation et éviter les erreurs . . . . .	2
1.4	Configurer LoRaMAC-node . . . . .	2
<b>2</b>	<b>Creation d'un programme de test pour y mettre des fonctions de sécurités</b>	<b>5</b>
2.1	Configuration du projet . . . . .	5
2.2	Programmation . . . . .	7
2.3	Lancer le projet . . . . .	10
2.4	Fonctions de sécurités . . . . .	11
<b>3</b>	<b>Tutoriel pour ajouter une layer SELinux à notre OS chirpstack-gateway-os via Yocto :</b>	<b>13</b>
3.1	Connexion au serveur de compilation . . . . .	13
3.2	Problèmes rentrés . . . . .	13
3.3	Générer l'OS . . . . .	14
<b>4</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Bibliographie</b>	<b>19</b>



---

## Utilisation de LoRaMac-node

---

Dans un premier temps, nous allons utiliser le projet LoRaMac-node créé par Semtech pour utiliser notre carte B-L072Z-LRWAN1 comme nœud LoRaWAN.

### 1.1 Installation

Pour gagner du temps dans le développement du nœud, nous utilisons comme base un projet github [[LoRaMAC-node](#)] mettant en œuvre le LoRaWAN sur notre carte. Nous développons le nœud à partir d'une distribution *Linux* basée sur *Arch Linux*, les dépendances requises sont :

```
cmake
arm-none-eabi-gcc
arm-none-eabi-newlibS221 : https://www.st.com/en/mems-and-ss221 : https://www.st.com/
➔en/mems-and-sensors/hts221.html#overviewensors/hts221.html#overview
openocd
```

Pour utiliser le projet LoRaMAC-node dans notre projet, nous avons téléchargé le .zip du projet de la branche master en **version 4.4.2**. Installation de Stlink —————

Le St-link est un programme permettant d'avoir accès au débiter des cartes conçues par STMicroelectronics, il permet, entre autres, de charger des programmes dans les cartes. .. code-block : : bash

```
git clone https://github.com/texane/stlink.git make release cd build/Release sudo make install sudo ldcon-
fig
```

### 1.1.1 Installation de LoRaMAC-node

Après avoir extrait le zip téléchargé précédemment, placez-vous dans le dossier LoRaMac-node-master. .. code-block : : bash

```
mkdir build cd build BOARD=B-L072Z-LRWAN1 cmake -DCMAKE_TOOLCHAIN_FILE= »cmake/toolchain-arm-none-eabi.cmake » -DBOARD= »$BOARD » -DAPPLICATION= »LoRaMac » -DCLASS= »classA » -DACTIVEREGION= »LORAMAC_REGION_EU868 » .. make
```

## 1.2 Charger un programme dans la carte

Pour charger le programme dans la carte nous utilisons St-Link, dans un premier temps brancher la carte à l'aide d'un câble USB à l'ordinateur. Sur la carte, vous devez utiliser le port USB le plus éloigné de l'antenne.

```
st-info --probe #Permet de verifier que la carte est reconnue par l'ordinateur
#Placez-vous dans le dossier build créé précédement.
st-flash write build/src/apps/LoRaMac/LoRaMac-classA.bin 0x8000000
```

Pour lancer le programme, vous n'avez plus qu'à appuyer sur le bouton noir (RESET) de la carte.

## 1.3 Faciliter la compilation et éviter les erreurs

Pour faciliter la compilation et le chargement du projet nous vous recommandons de faire un script dans le dossier build.

```
nano my-LoRa.sh
#A l'intérieur écrivez les lignes suivantes
BOARD=B-L072Z-LRWAN1
cmake -DCMAKE_TOOLCHAIN_FILE="cmake/toolchain-arm-none-eabi.cmake" -DBOARD="$BOARD" -
-DAPPLICATION="LoRaMac" -DCLASS="classA" -DACTIVEREGION="LORAMAC_REGION_EU868" ..
make
```

Maintenant pour mettre le programme sur la carte vous n'aurez plus qu'à la brancher et lancer la commande suivante.

```
./my-LoRa.sh
```

## 1.4 Configurer LoRaMAC-node

Nous vous recommandons d'utiliser dans un premier temps le programme *exemple* de LoRaMAC-node. Pour cela, vous devrez utiliser les programmes se trouvant dans le dossier « LoRaWAN\_securise/noeud/LoRaMac-node-master/src/apps/LoRaMac/classA/B-L072Z-LRWAN1/ »

Le programme *main.c* est le programme principal, c'est dans celui-ci que vous aller créer votre programme personnel. Le fichier *Commissioning.h* contient les différentes variables nécessaire à une liaison LoRaWAN (DevEUI, AppKey ...).

Lorsque l'on commence un projet, vous devez configurer dans un premier temps le fichier *Commissioning.h*.

Dans nôtres cas nous utilisons une méthode de connexion APB donc nous devons configurer dans ce fichier, le mode de connexion, devAddr, AppSKey et NwkSKey.

Voici les valeurs que nous utilisons pour ces différentes clés :

— devAddr = 0x00000000

- AppSkey = 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
- NwkSkey = 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C

Pour configurer notre programme, cherchez les lignes suivantes et mettez les valeurs indiquées à la place des valeurs par défaut.

```
#define OVER_THE_AIR_ACTIVATION 0
// Définit le mode de connexion APB
#define LORAWAN_DEVICE_ADDRESS ( uint32_t )0x00000000
// DevAddr
#define LORAWAN_NWK_S_ENC_KEY { 0x2B, 0x7E, 0x15, 0x16, ↵
↵ 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C }
//NwkSKey
#define LORAWAN_APP_S_KEY { 0x2B, 0x7E, 0x15, 0x16, ↵
↵ 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C }
//AppSKey
```

Enregistrez les modifications et lancer le programme à l'aide du script créé précédemment.





---

### Creation d'un programme de test pour y mettre des fonctions de sécurités

---

Pour faciliter l'implémentation des fonctions de sécurités, nous avons créé un programme simple utilisant uniquement des fonctions faites par STM. Le programme que nous avons créé inverse l'état de 3 LEDs à des intervalle différents, et tant que l'utilisateur appui sur le bouton bleu (USER) de la carte les LEDs restent éteintes. Le programme écrit aussi une clé secrète sur le port série. Si nous envoyons la clé secrète sur la liaison série, c'est pour éviter que le compilateur optimise le code en la supprimant, ce qui serait possible si cette variable n'est pas utilisée. Installation du logiciel #####

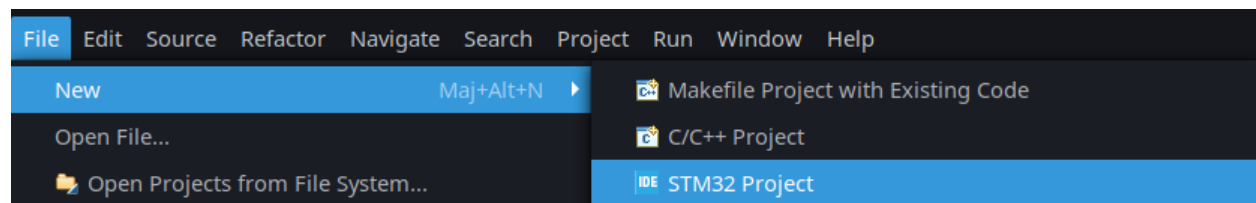
Pour ce programme, nous utilisons *CubeIDEv1.0.2* qui est l'IDE de *STMicroelectronic* permettant une configuration facile des horloges, des entrées sortie du micro contrôleur. Vous pouvez le télécharger pour vôtres plateforme à cette adresse : <https://www.st.com/en/development-tools/stm32cubeide.html>.

Une fois le fichier téléchargé, vous devez l'extraire. Et lancer le script d'installation avec les droits d'administrateur et suivez les consignes.

```
unzip en.en.st-stm32cubeide_1.1.0_4551_20191014_1140_amd64.sh.zip
sudo sh ./en.en.st-stm32cubeide_1.1.0_4551_20191014_1140_amd64.sh/st-stm32cubeide_1.1.
↪0_4551_20191014_1140_amd64.sh
```

## 2.1 Configuration du projet

Dans le logiciel, vous devez commencer par créer un nouveau projet. File -> New -> STM32 PProject



Dans la fenêtre qui c'est ouverte, aller sur l'onglet « board selector » et dans le champ de recherche « B-L072Z-LRWAN1 ».

MCU/MPU Selector

Board Selector

Cross Selector

Board Filters

★

📄

🔍

🔄

Part Number Search

Vendor

Type

MCU/MPU Series

Other

Price = 46.5

Oscillator Freq. = 0 (MHz)

Peripheral

☒ Accelerometer
 

00

☒ Analog I/O
 

00

☒ Arduino Form Factor
 

01

☒ Audio Line In
 

00

☒ Audio Line Out
 

00

☒ Battery
 

00

☒ Button
 

02

☒ CAN
 

00

☒ Camera
 

00

☒ Compass
 

00

☒ Custom Form Factor
 

00

☒ Digital I/O
 

00

☒ Ethernet
 

00

Features

Large Picture


Docs & Resources

Datasheet

Buy

★

B-L072Z-LRWAN1




ACTIVE

Active

Product is in mass production

Unit Price (US\$) : 46.5


Mounted device: STM32L072CZYx



arm MBED Enabled

The B-L072Z-LRWAN1 LoRaDiscovery kit is a development tool to learn and develop solutions based on LoRa and FSK/OOK technologies. This Discovery kit features an all-in-one open module CMWX1ZZABZ-091 (by Murata). The module is powered by an STM32L072CZ and an SX1276 transceiver. The transceiver features the LoRa long range modem, providing

Boards List: 1 item

	Overview	Part No.	Type	Marketing Status	Unit Price (US\$)	Mounted Device
★		B-L072Z-LRWAN1	Discovery kit	Active	46.5	STM32L072CZYx

?

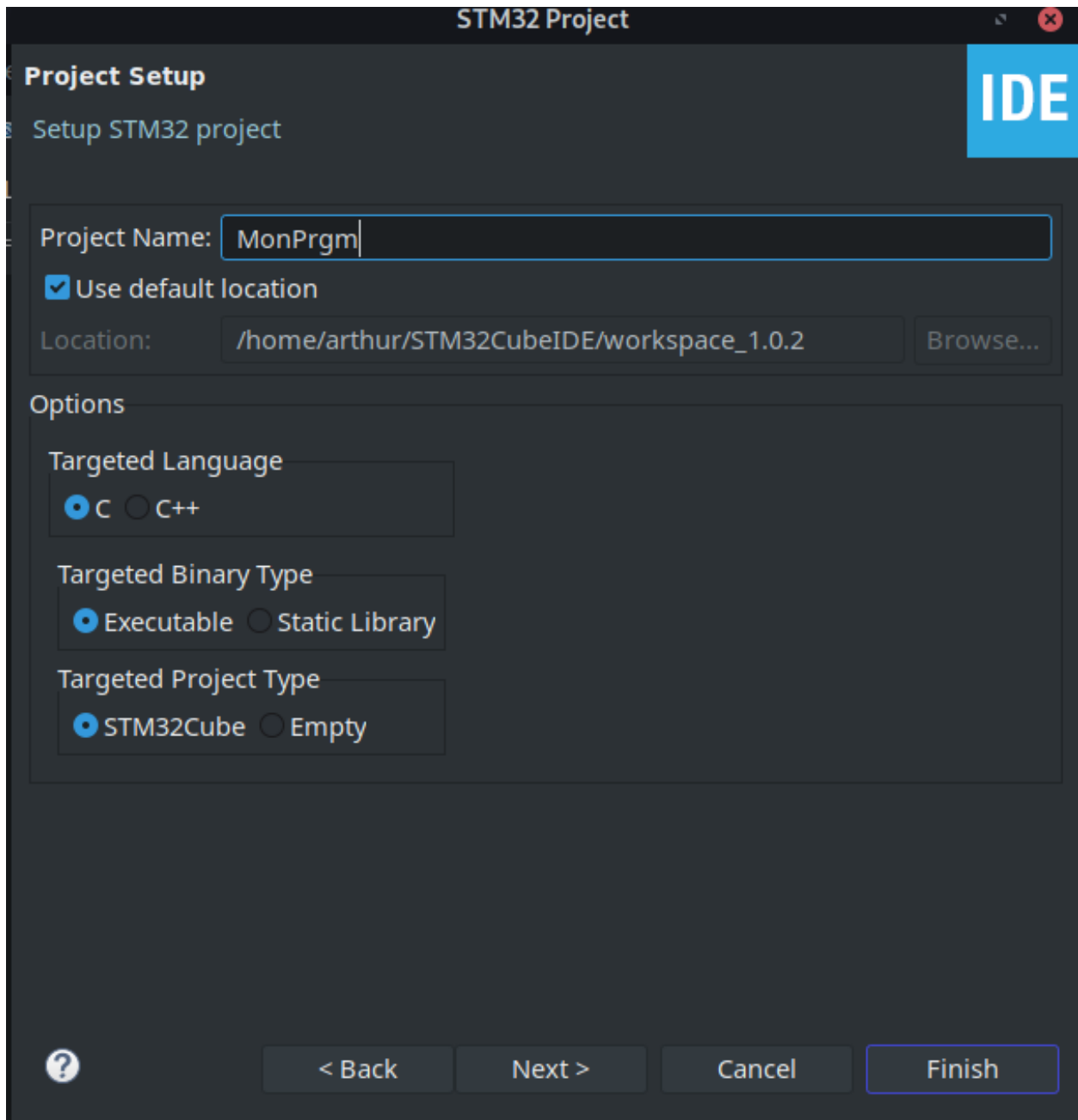
< Back

Next >

Cancel

Finish

Sélectionner la carte puis cliquez sur « Next », entrer le nom de votre projet puis cliquez sur « Finish ».



## 2.2 Programmation

Pour commencer, nous allons paramétrer les I/O, à savoir 3 LEDS et un Bouton Poussoir.

Dans l'arborescence du projet (fenêtre à gauche) sélectionner le fichier avec l'extension *.ioc* c'est un fichier qui va vous permettre de facilement configurer les I/O.

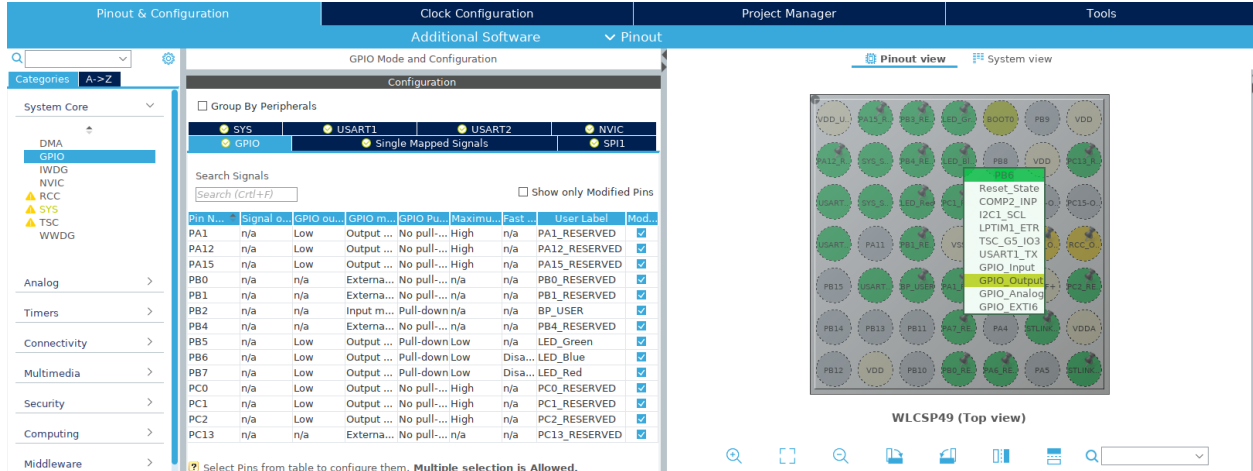
Pour configurer les I/O cliquez sur leurs identifiants, sur la figure de droite. Sélectionner *GPIO\_Output* pour les LEDS et *GPIO\_Input* pour le bouton. En se référant à la documentation les I/O à configurer sont les suivantes :

- PB5 = Led\_Vert
- PB6 = Led\_Bleu

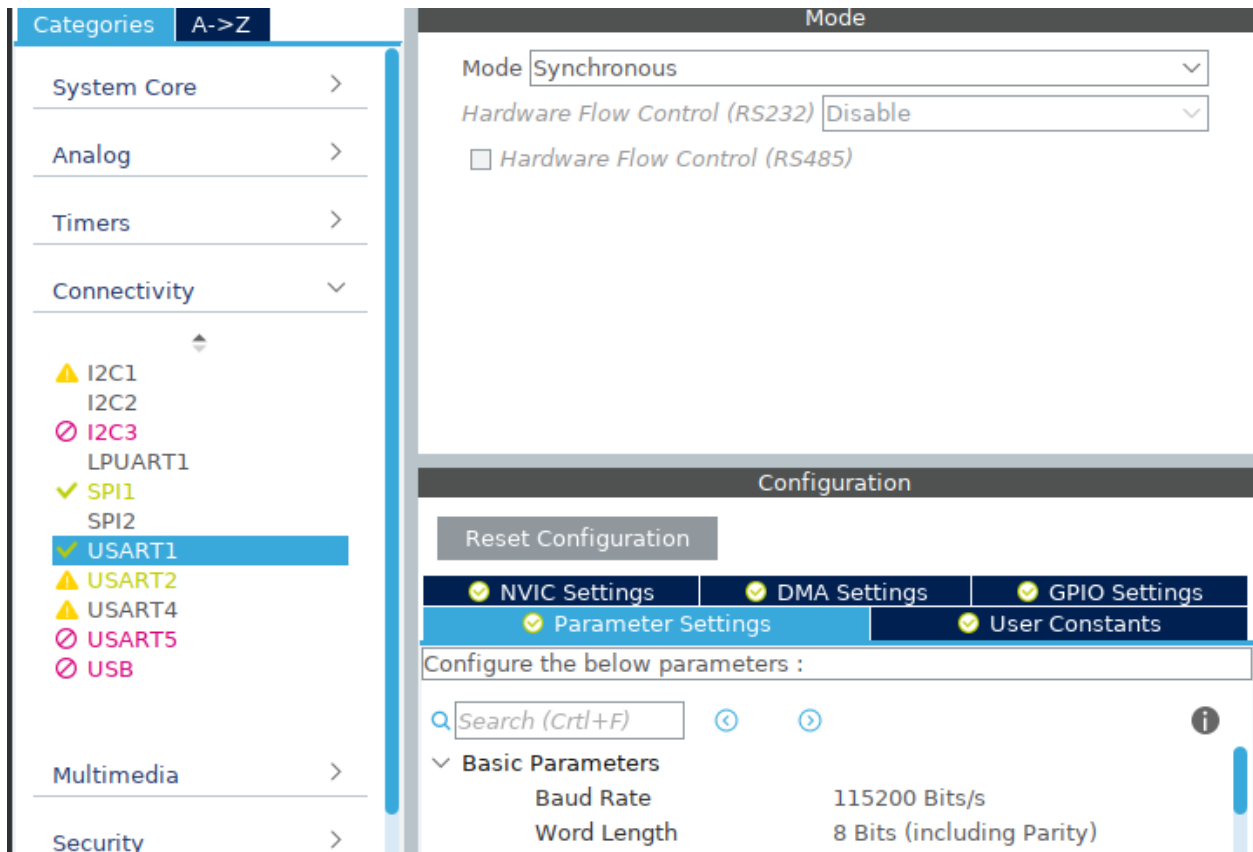
- PB7 = Led\_Rouge
- PB2 = Bouton Pression

Pour faciliter la programmation avec des noms personnalisés pour ces I/O, vous pouvez définir des *label* dans la colonne « User label » du tableau « GPIO ».

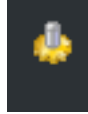
Vous devriez avoir une configuration comparable à celle de la figure suivante.



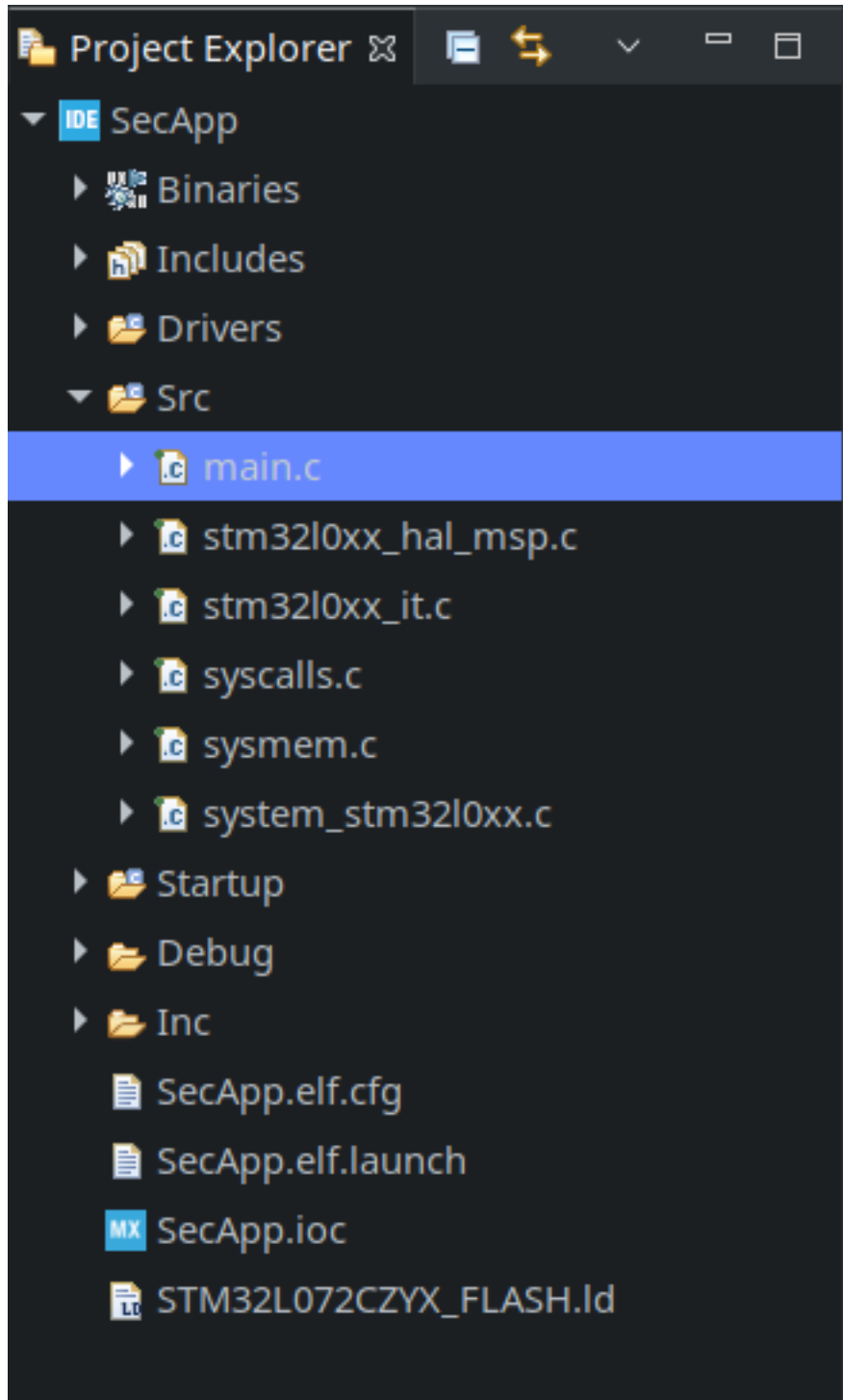
Maintenant nous allons configurer L'UART1, dans la catégorie « Connectivity » cliquer sur « USART1 » mettez le, en mode « Synchronous » et réglez le *Baud Rate* à 115200.



Pour générer le code vous devez cliquer sur l'icône « Device Configuration Tool Code Generation » dans la barre d'outil en haut de l'écran.



Dans l'arborescence du projet, déplacez-vous jusqu'au fichier *main.c*



Dans les variable privé ~ligne 52 vous allez ajouter un buffer et nôtres clés secrète.

```
uint8_t buffer[2];
char SECRET_KEY[16] = {0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7,
↪0x15, 0x88, 0x09, 0xCF, 0x3F, 0x3C};
```

Dans la partie « USER CODE BEGIN 2 » ~ ligne 156 nous allons initialiser nos LEDs dans l'état éteint, puis envoyer notre clé sur la liaison série.

```
HAL_GPIO_WritePin(LED_Blue_GPIO_Port, LED_Blue_Pin, 0);
HAL_GPIO_WritePin(LED_Green_GPIO_Port, LED_Green_Pin, 0);
HAL_GPIO_WritePin(LED_Red_GPIO_Port, LED_Red_Pin, 0);

for (int i = 0; i < 16; i++) {

    buffer[0] = SECRET_KEY[i];

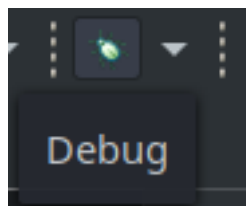
    buffer[1] = '\n';
    HAL_USART_Transmit(&husart1, buffer, sizeof(buffer), 1000);
}
```

Dans la boucle « while(1) » ~ligne 187, de la fonction « main » du programme vous allez écrire le programme faisant clignoter les LEDs gère le bouton, tout en envoyant la clé sur la liaison série.

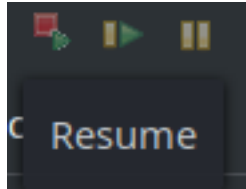
```
for (int i = 0; i < 16; i++) {
    buffer[0] = SECRET_KEY[i];
    HAL_USART_Transmit(&husart1, buffer, sizeof(buffer), 1000);
}
buffer[0] = '\n';
HAL_USART_Transmit(&husart1, buffer, sizeof(buffer), 1000);
HAL_Delay(1000);
if (HAL_GPIO_ReadPin(BP_USER_GPIO_Port, BP_USER_Pin) == 0) {
    HAL_GPIO_WritePin(LED_Blue_GPIO_Port, LED_Blue_Pin, 0);
    HAL_GPIO_WritePin(LED_Green_GPIO_Port, LED_Green_Pin, 0);
    HAL_GPIO_WritePin(LED_Red_GPIO_Port, LED_Red_Pin, 0);
}
else {
    //HAL_Delay(1000);
    HAL_GPIO_TogglePin(LED_Red_GPIO_Port, LED_Red_Pin);
    HAL_Delay(500);
    HAL_GPIO_TogglePin(LED_Blue_GPIO_Port, LED_Blue_Pin);
    HAL_Delay(250);
    HAL_GPIO_TogglePin(LED_Green_GPIO_Port, LED_Green_Pin);
}
```

## 2.3 Lancer le projet

Pour écrire le code sur la carte, vous devez d'abord la brancher à votre ordinateur. Puis lancer le débogage, pour ça cliquez sur l'icône en forme de virus vert dans la barre d'outils au-dessus du code.



CubeIDE va passer en mode « Débuggage » une fois que le programme est chargé, vous pouvez soit, débrancher et rebrancher la carte pour le lancer ou vous pouvez aussi le démarrer depuis l'interface de l'IDE en cliquant sur l'icône « Resume »



## 2.4 Fonctions de sécurités

### 2.4.1 RDP

Pour implémenter la contre mesure RDP, vous devez tout d'abord créer une variable en dessous de celles créées précédemment. Cette variable contiendra nôtres configuration personnalisée des octets d'Options.

```
FLASH_OBProgramInitTypeDef myconf;           //Option bytes config perso for RDP
```

En suite, il faut ajouter le code suivant à l'initialisation du système ~ligne 113.

```
// Implémentation de RDP
if (HAL_FLASH_Unlock() == HAL_OK){           //deverouille la memoire flash

    if(HAL_FLASH_OB_Unlock() == HAL_OK){ // deverouille les options byte dans
↳la memoire flash

        HAL_FLASHEx_OBGetConfig(&myconf);    // Recupere la configuration
↳actuelle des options bytes

        if(myconf.RDPLevel == OB_RDP_LEVEL0){ // Test si RDP est désactiver

            myconf.RDPLevel = OB_RDP_LEVEL1;    //Mettre à 1 pour
↳avoir le niveau 1

            HAL_FLASHEx_OBProgram(&myconf);    // Ecrit la
↳configuration

            HAL_FLASH_OB_Launch();            // applique la configuration
↳des options bytes

            HAL_FLASH_OB_Lock(); //verouille les options byte dans la
↳memoire flash
        }

    }

    HAL_FLASH_Lock(); //verouille la memoire flash
}
```

## 2.4.2 PCROP et RDP

Pour implémenter PCROP en plus de RDP, il faut rajouter une variable pour la configuration des octets d'option avancé à côté de la variable créée précédemment.

```
FLASH_AdvOBProgramInitTypeDef myconfAdv;
```

Il faut ensuite modifier le morceau de code écrit précédemment pour y ajouter la configuration de PCROP.

```
if (HAL_FLASH_Unlock() == HAL_OK) {           //deverouille la memoire flash

if(HAL_FLASH_OB_Unlock() == HAL_OK){         // deverouille les options byte dans la_
↪memoire flash

    HAL_FLASHEx_OBGetConfig(&myconf);          // Recupere la configuration actuelle_
↪des options bytes
    HAL_FLASHEx_AdvOBGetConfig(&myconfAdv); //Récupère la configuration du_
↪registre où sont stocké les configurations de pcrop

    if(myconf.RDPLevel == OB_RDP_LEVEL0){

        myconfAdv.PCROPSector = OB_PCROP_AllPages - OB_PCROP_Pages32to63;
↪ //Choix des secteur à sécuriser
        myconfAdv.PCROPSector2 = OB_PCROP_AllPages;
        myconfAdv.PCROPState = OB_PCROP_STATE_ENABLE;
        myconf.RDPLevel = OB_RDP_LEVEL1;      //Mettre à 1 pour avoir le_
↪niveau 1

        HAL_FLASHEx_OB_SelectPCROP(); //Active PcROP
        HAL_FLASHEx_AdvOBProgram(&myconfAdv);
        HAL_FLASHEx_OBProgram(&myconf);      // Ecrit la configuration
        HAL_FLASH_OB_Launch(); // applique la configuration des options bytes
        HAL_FLASH_OB_Lock(); //verouille les options byte dans la memoire_
↪flash
    }
}
HAL_FLASH_Lock(); //verouille la memoire flash
}
```

La liste des constantes définissant les secteurs de la mémoire se trouve dans le fichier : *Drivers/STM32L0xx\_HAL\_Driver/Inc/stm32l0xx\_hal\_flash\_ex.h*, les définitions commence à partir de la ligne 489.



---

### Tutoriel pour ajouter une layer SELinux à notre OS chirpstack-gateway-os via Yocto :

---

#### 3.1 Connexion au serveur de compilation

Pour se connecter au serveur de compilation :

```
numeroetudiant@m1-isc-os  
motdepasse
```

choisir un shell bash

```
bash
```

#### 3.2 Problèmes rentrés

Problème de téléchargement qui ne marche pas :

Ajoutez le proxy : `export http_proxy=http://ocytohe.univ-ubs.fr:3128`

Problème Git et proxy

`git config --global http.proxy http://ocytohe.univ-ubs.fr:3128`

## 3.3 Générer l'OS

Cloner le répertoire :

```
git clone https://github.com/brocaar/chirpstack-gateway-os
```

On se met dans notre répertoire > ~/projet/os/chirpstack-gateway-os/

On active l'environnement de build en faisant

```
source oe-init-build-env
```

renvoie :

```
« You had no conf/local.conf file. This configuration file has therefore been created for you with some
default values. You may wish to edit it, for example, select a different machine (target hardware). See
conf/local.conf for more information as common configuration options are commented.
```

```
You had no conf/bblayers.conf file. This configuration file has therefore been created for you with
some default values. To add additional metadata layers into your configuration please add entries to
conf/bblayers.conf. »
```

On se retrouve dans le dossier build

Ajouter la layer meta-selinux, dans le dossier source

```
git clone https://git.yoctoproject.org/git/meta-selinux
```

Aller éditer manuellement le fichier bblayers.conf dans le dossier build, pour l'ajouter

```
vim bblayers.conf
i (entrer en mode insertion)
```

Ajouter à la fin du fichier, dans les guillemets :

```
/home/numeroetudiant/projet/os/chirpstack-gateway-os/meta-selinux \
echap
:wq
```

Maintenant, on cherche à compiler notre image, avec la commande

```
bitbake core-image-minimal
```

Cependant, nous ne pouvons pas effectuer cette commande, car les modules dont dépend le projet chirpstack-gateway-os ne peuvent pas être téléchargés, et bitbake en fait partie.

Le protocole ssh est bloqué par le serveur de compilation. Certains utilisent le protocole git qui lui aussi utilise le protocole ssh. on a donc remplacé les url git par des url https qui n'utilisent pas le protocole ssh pour les télécharger.

On suit le cheminement du makefile :

```
make submodules
```

Cependant, nous avons oublié de modifier également le fichier

```
git submodule init
git submodule => nous montre toutes les layers .git/modules/layers
```

On finit par tout télécharger à la main On peut relancer la compilation

```
bitbake core-image-minimal
```

On obtient une erreur :

« Layer selinux is not compatible with the core layer which only supports these series : thud (layer is compatible with zeus) »

On va donc copier notre répertoire ailleurs pour faire des essais et essayer de changer de version de yocto : nous étions sur rocko et nous passons à zeus :

```
git checkout -b zeus origin/zeus
```

Cependant erreur, nous travaillons sur le git du projet chirpstack-gateway-os donc incompatible pour les commandes avec la version zeus.

on passe tout le monde sur warrior (avant dernière version disponible et fonctionnelle)

On doit changer thud dans le fichier .gitmodules

exemple :

```
[submodule « layers/bsp/meta-raspberrypi »] path = layers/bsp/meta-raspberrypi url =
git ://git.yoctoproject.org/meta-raspberrypi branch = thud
```

On passe sur la branche warrior

```
branch = warrior
```

Rappatrier l'image sur la machine hôte

```
scp -r numeroetudiant@ml-isc-os:/<source>/ ./<cible>/
```

Flasher l'image sur carte SD

```
dd if=<source> of=<cible> bs=<taille des blocs> skip= seek= conv=<conversion>
```

Prêt à être utilisé sur la box LoRa. Les commandes d'administration peuvent s'effectuer via un terminal directement sur la box LoRa.



## CHAPITRE 4

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



---

## Bibliographie

---

[LoRaMAC-node] Projet permettant la mise en œuvre d'un Nœud LoRaWAN sur une carte *B-L072Z-LRWAN1* (<https://github.com/Lora-net/LoRaMac-node/tree/master/Doc>)