

Votre Projet – Réalisation software

```
void DEMO_init(void)
{
    HAL_Init();
    SYS_init(); //initialisation du systeme (horloge...)
    GPIO_Configure(); //Configuration des broches d'entree-sortie.
    TIMER2_run_1ms(); //Lancement du timer 2 a une periode d'1ms.

    UART_init(2,UART_RECEIVE_ON_MAIN); //Initialisation de l'USART2 (PA2=Tx, PA3=Rx, 115200 bauds/sec)
    UART_init(6,UART_RECEIVE_ON_MAIN); //Initialisation de l'USART6 (PC6=Tx, PC7=Rx, 115200 bauds/sec)
    SYS_set_std_usart(USART6,USART6,USART6);
}

/**
 * @brief cette fonction doit etre appelee periodiquement en tâche de fond. Elle assure la lecture du bo
 */
void DEMO_process_main(void)
{
    static bool_e bt_previous;
    bool_e bt_current;
    bool_e button_pressed;
    char touch_pressed;

    //Detection d'appui bouton.
    bt_current = (bool_e)HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0); //Lecture bouton.
    button_pressed = (!bt_previous && bt_current); //Detection d'un appui bouton
    bt_previous = bt_current; //Memorisation pour le prochain passage.

    touch_pressed = DEMO_IHM_uart_read(); //Bufferise chaque caractère reçu et le renvoi.

    //Detection de l'appui sur
    DEMO_state_machine(button_pressed || touch_pressed == 'm' || touch_pressed == 'M', touch_pressed);
}
```

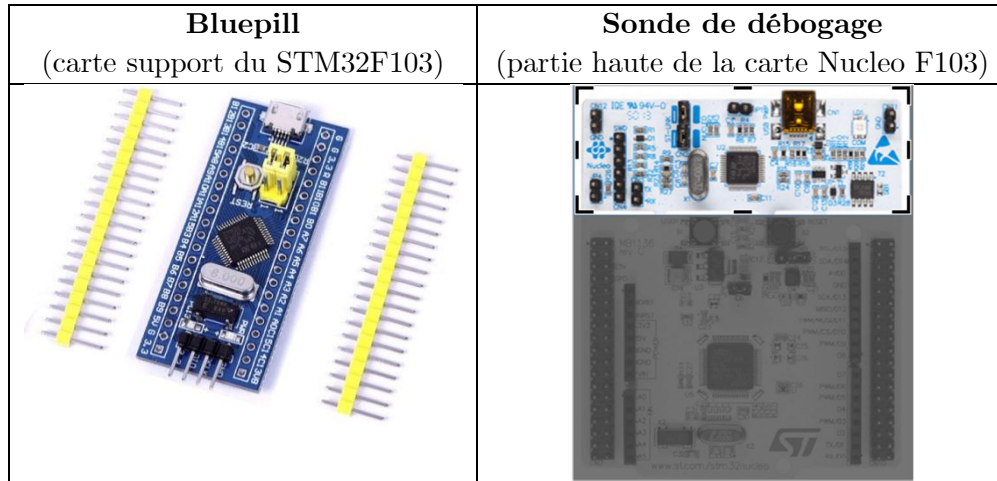
2022

Objectifs de la mission :

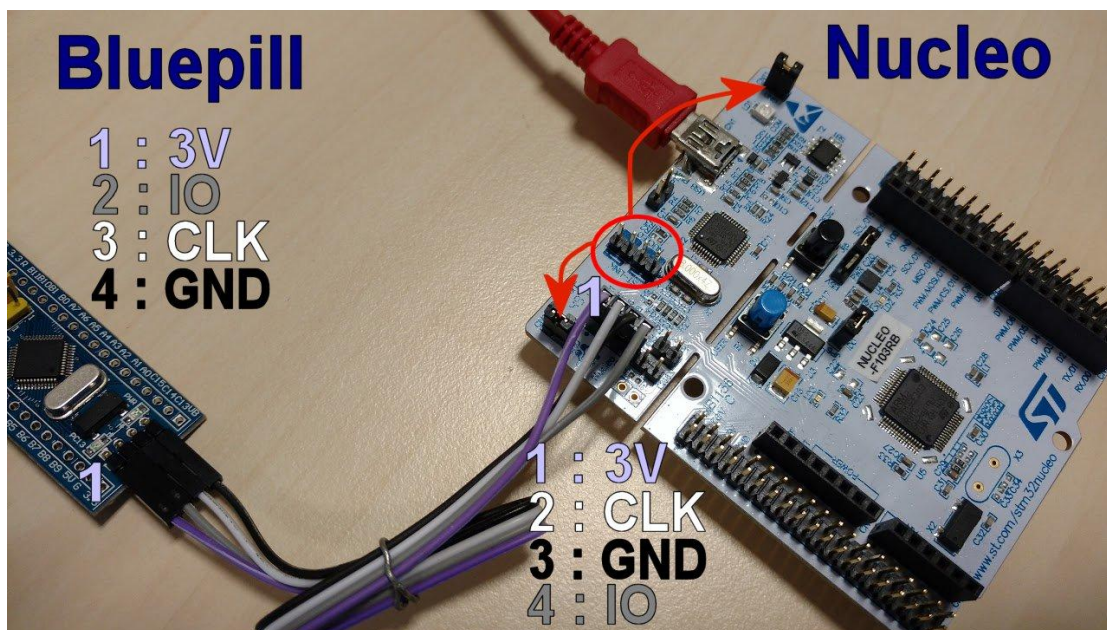
- Développement d'un logiciel embarqué pour répondre au sujet choisi
- Développement de briques logicielles
- Intégration de briques logicielles existantes (bibliothèques et modules fournis)
- Tests unitaires
- Tests d'intégration

Avant-propos.

Afin de programmer votre microcontrôleur qui se trouve sur le support « Bluepill », vous devez relier la sonde de programmation et de débogage à ce microcontrôleur :



Voici comment relier les deux cartes. Soyez vigilants et respectez les couleurs !



N'oubliez pas de retirer les deux cavalier (cf marquage ci-dessus en rouge) et de les placer sur les supports de stockage réservés à cet effet. Dans cette position, les cavaliers permettent de programmer le microcontrôleur distant. On peut les remettre en place pour programmer le microcontrôleur situé sur la carte Nucleo.

Attention, le fil noté « 3V » ne permet pas d'alimenter la bluepill à partir de la Nucleo. Il s'agit simplement d'une lecture de la tension d'alimentation.

Il faut amener cette alimentation autrement (via votre PCB, ou en déplaçant le fil '3V' côté Nucleo vers le net 3,3V sans se tromper car un 5V serait fatal !!!!!!!)

Avant-propos.

Nous nous sommes rendus compte que la dernière version de STM32 Cube IDE n'est pas (plus !) compatible nativement avec les cibles utilisées en DEEP et reçues dernièrement (Bluepill F103, pour lesquelles seules des contrefaçons du microcontrôleur sont approvisionnables en ce moment).

Si vous avez installé une version ultérieure à 1.7.0 (1.10.1 par exemple), il est donc nécessaire de choisir l'une des deux méthodes proposées ci-dessous :

Méthode 1 : désinstaller Cube IDE et de revenir à la version 1.7.0.

Vous trouverez l'installateur ici (Windows/MacOS/Linux) :

Lien cours : <https://t.ly/tzqF> → STM32CubeIDE

Lien complet :

https://reseaueseo-my.sharepoint.com/personal/samuel_poiraud_eseo_fr/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fsamuel%5Fpoiraud%5Feseo%5Ffr%2FDocuments%2FSetups%2FSTM32CubeIDE

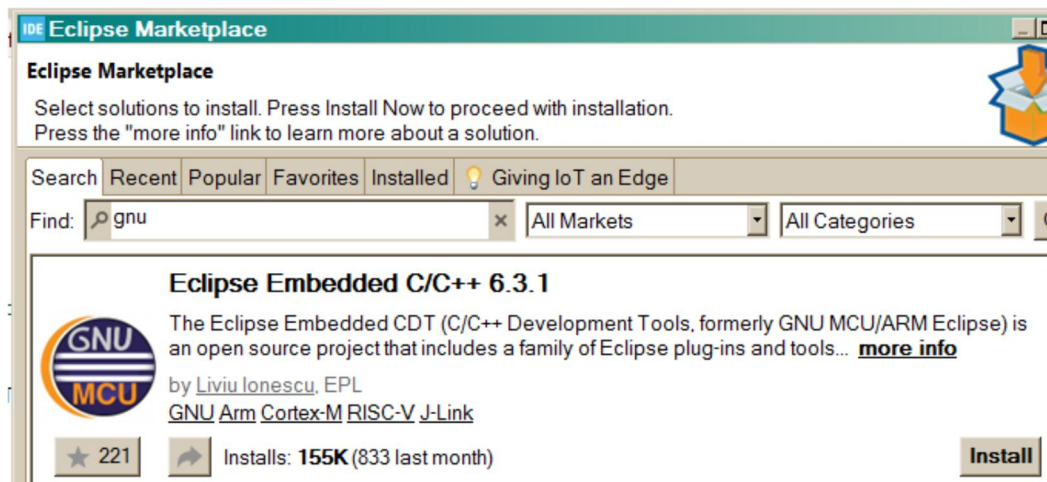
Recréez ensuite un nouveau workspace vierge, réimportez-y vos projets si besoin.

Méthode 2 : Installation des outils GNU Eclipse Embedded + OpenOCD

(testé sous windows uniquement, à adapter au besoin aux autres OS)

Installer un plugin permettant de contourner le dysfonctionnement :

- Help -> Eclipse Marketplace -> rechercher 'gnu' -> installer Eclipse Embedded C/C++ 6.3.1



- Télécharger l'archive zip « **openocd.zip** » disponible sur ce dépôt, et extrayez la sur votre ordinateur (à l'emplacement suggéré suivant : C:\ST\openocd)

Lien cours : <https://t.ly/tzqF> → STM32CubeIDE

Lien complet :

https://reseaueseo-my.sharepoint.com/personal/samuel_poiraud_eseo_fr/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fsamuel%5Fpoiraud%5Feseo%5Ffr%2FDocuments%2FSetups%2FSTM32CubeIDE

La suite de cette notice vous indiquera, pour chacune des deux méthodes, comment configurer la cible de débogage.

Introduction.

Vous avez choisi un sujet, vous avez réalisé (ou allez réaliser) une carte électronique permettant d’y répondre... Il faut maintenant construire le logiciel de votre application.

Il est primordial d’avancer **étape par étape**, sans viser dès le début l’application complète, mais en privilégiant la **validation unitaire** de chaque module logiciel.

Comme dans une majorité des projets que vous rencontrerez, le temps imparti ne permet pas de rédiger l’ensemble des lignes de code qui seront exécutées sur le produit fini. Une partie de la mission de l’ingénieur logiciel consiste à réutiliser des briques logicielles existantes. Ces briques peuvent être fournies par le fabricant du microcontrôleur, par une équipe de développement de l’entreprise, par un sous-traitant, ou par les enseignants du module DEEP...

Les différentes missions réalisées au début de la partie numérique du DEEP vous ont permis d’aborder et de développer des briques logicielles indispensables pour la majorité des projets.

Un « **projet-f103** », disponible sur un serveur de version GIT, contient plusieurs briques logicielles que vous pouvez utiliser.

Vous avez dit GIT ?

Les développeurs qui travaillent en équipe dans les entreprises ont besoin au quotidien d’outils qui facilitent leur collaboration.

- Comment travailler ensemble sur les mêmes fichiers, sans se perdre dans les versions, les copies manuelles ?
- Comment savoir quelques mois après quelle version a été livrée au client ?
- Comment détecter et corriger des bogues apparus lors des évolutions du programme ?

Les outils de versionning répondent notamment à cette problématique en permettant à plusieurs développeurs de travailler ensemble sur un projet commun. Ces outils conservent l’historique des révisions et fournissent un moyen de sauvegarde.

Exemples d’outils :

- Subversion (SVN) (Le cœur du rédacteur de ce document penche sérieusement pour cette solution technique... bien qu’un peu dépasser aujourd’hui)
- Git (très utilisé par les développeurs, notamment via des plateformes comme Github)
- Dropbox ou autres solutions cloud : utile pour le partage de fichiers divers, nettement moins adaptées au développement logiciel

Accéder au projet-f103.

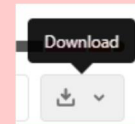
Ce document présente la démarche à suivre pour télécharger les fichiers depuis le serveur gitlab de l'école.

- 0- Vous devez préalablement être connecté au réseau de l'école (accessible depuis l'extérieur via le VPN).

Le projet 'f103' est disponible sur un **dépôt GIT**, à l'emplacement suivant :

`git@172.24.0.69:deep/f103.git`

ou via l'URL et l'interface web ; <https://172.24.0.69/deep/f103/>

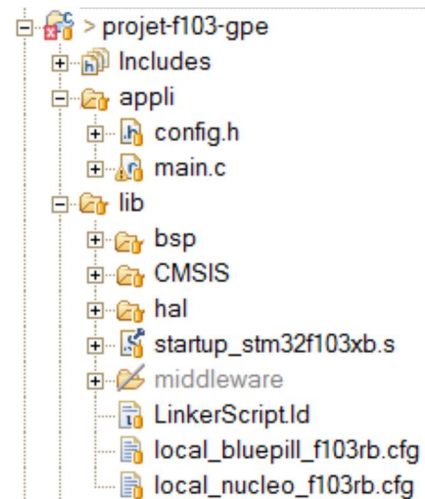


- 1- Récupérez ce dépôt git, et placez le sur votre ordinateur, dans un chemin ne contenant ni accent ni caractère spécial

- 2- Vous obtenez l'arborescence ci-contre :

- 3- Dans « STM32CubeIDE », importez ce projet f103

- File
- Import
- Existing project into workspace
- Select **root directory**
- Assurez vous que la case 'copy projects into workspace' est bien décochée
- Finish



- 4- Vérifiez que votre workspace est bien configuré en ISO-8859-1 (et pas en UTF-8) :

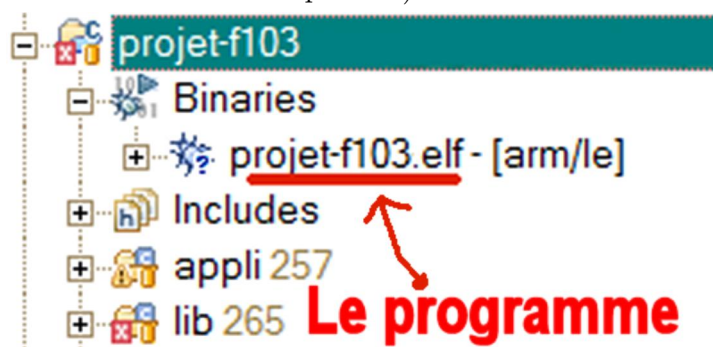
Windows -> Preferences -> General -> Workspace ->

Text File Encoding -> choisir : "Other: ISO-8859-1"

- 5- Compilez ce projet



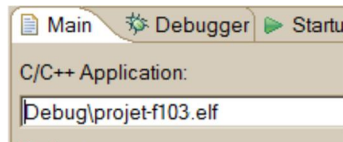
- 6- Sur le nom du projet : **Bouton-droit -> Refresh** (cela permet de rafraichir la visibilité du binaire créé lors de la compilation)



Il nous faut maintenant configurer la « cible de débogage ».

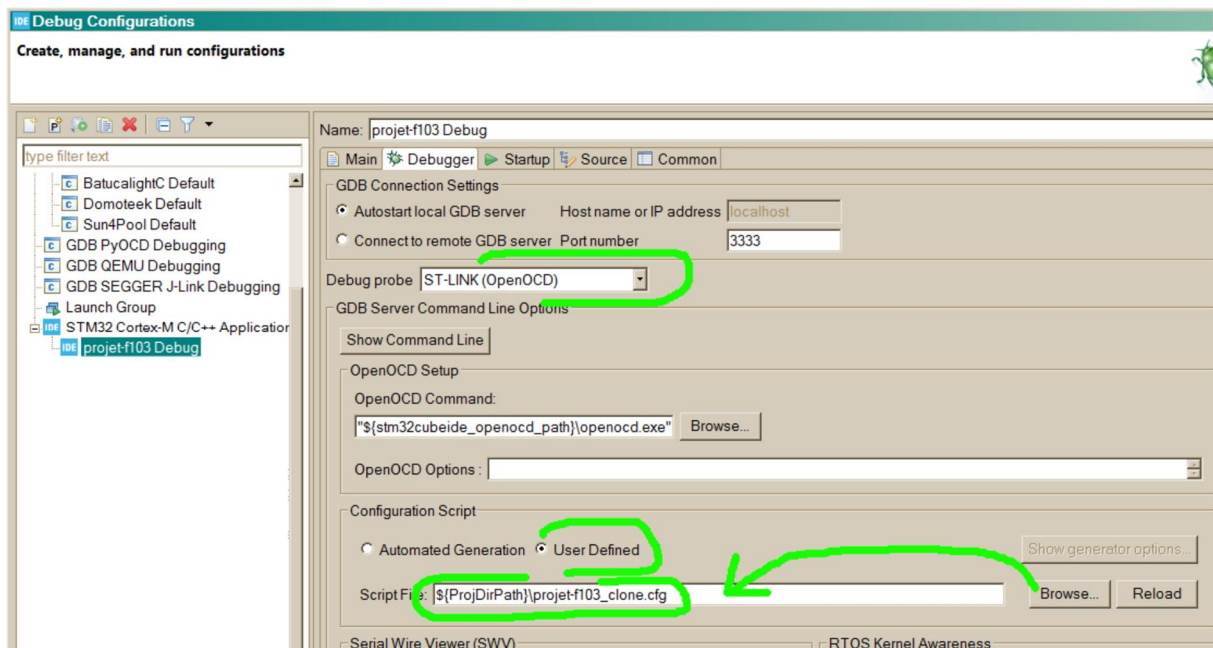
Si vous avez opté pour la méthode n°1 (utilisation de Cube IDE <= 1.7.0), suivez ceci (sinon, RDV page suivante) :

- Run -> Debug configuration -> double clic sur STM32 Cortex-M C/C++ Application (pour générer la cible de débogage)
- Vérifier que la case C/C++ Application contient bien « **Debug\projet-f103.elf** »



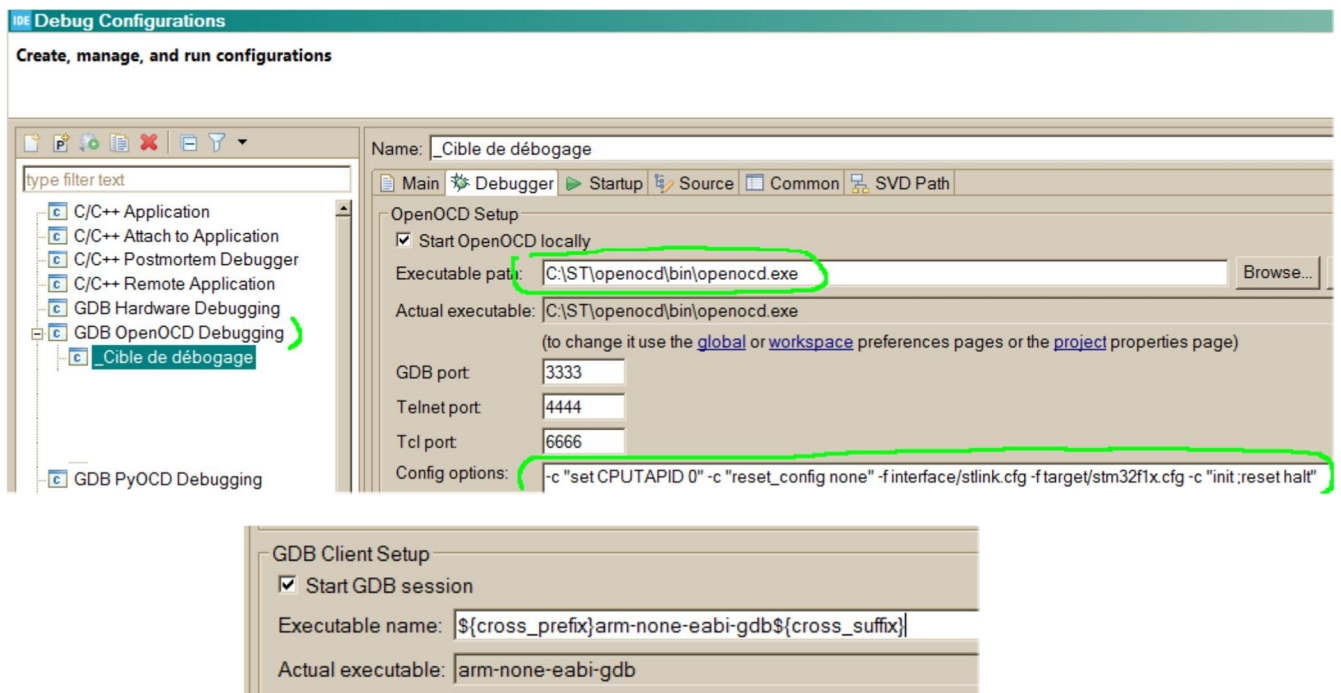
- Dans l'onglet « **Debugger** », dans la section « **Debug probe** », choisissez « ST-Link (OpenOCD) ».

Dans « Configuration Script », choisissez « **User Defined** », cliquez sur « **Browse** », et localisez le fichier projet-f103_clone.cfg qui se trouve dans le projet. (ce script est utile pour configurer le reset software de la cible. Il permet aussi de programmer les clones contrefaits.)




Si vous avez opté pour la méthode n°2 (installation des outils GNU Eclipse Embedded + openocd), suivez ceci (sinon, RDV page précédente) :

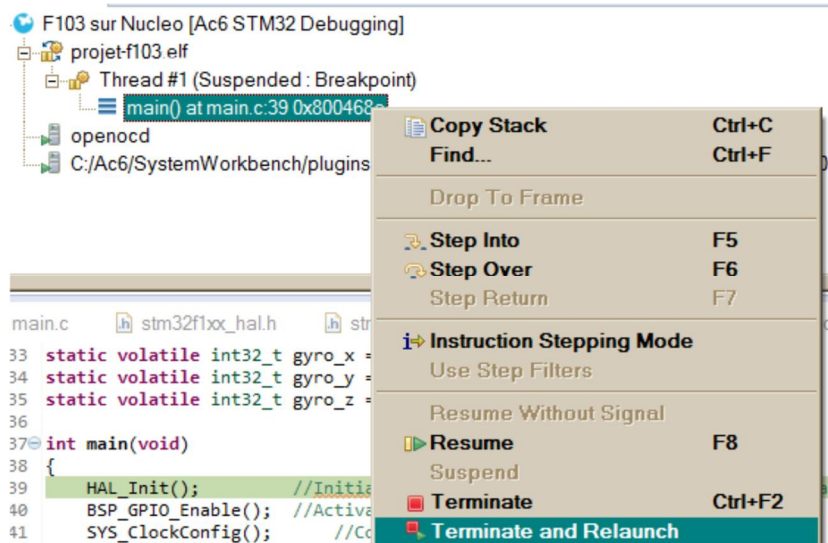
- Dans la fenêtre « Debug Configuration », choisir « **GDB OpenOCD Debugging** » (au lieu de « STM32 C/C++ Application »)
- Dans l'onglet « Debugger », la ligne « executable path » doit contenir le chemin complet vers votre exécutable openocd :
C:\ST\openocd\bin\openocd.exe
 - Si vous êtes sur un MAC / Linux, adaptez ce chemin...
- Dans ce même onglet Debugger, la zone config options doit contenir :
-c "set CPUPID 0" -c "reset_config none" -f interface/stlink.cfg -f target/stm32f1x.cfg -c "init ;reset halt"
- Dans ce même onglet Debugger, la zone 'Executable name' doit contenir :
\${cross_prefix}arm-none-eabi-gdb\${cross_suffix}




Après avoir compilé (avec succès !) votre programme, et configuré la cible de débogage, lancez le programme sur la cible !



- **Rappel** : si une session de débogage est déjà lancée, on ne peut en lancer une autre. On utilise alors la fonctionnalité bouton-droit -> « **terminate & relaunch** ».  Le programme est alors recompilé (si nécessaire), puis renvoyé sur la cible et relancé. Ce bouton est également accessible depuis la barre supérieure (en perspective debug).



- Soyez curieux et essayez de comprendre le cheminement du programme à partir de la fonction main().
- Utilisez sans modération le CTRL+Clic pour naviguer dans le code source.
- Pour reseter le programme sans reprogrammer le microcontrôleur, on peut utiliser avantageusement le bouton reset ! 

Comment utiliser un adaptateur USB série pour communiquer via une liaison série avec le microcontrôleur ?

Branchez une liaison série sur les ports de l'UART 1 : PC6 (Tx) et PC7 (Rx) ou de l'UART2 : PA2 (Tx) et PA3 (Rx)

Nous donnons en **annexe de la mission 1** deux exemples : Yat et Docklight.

Adaptateur filaire PL2303



Attention, le driver fonctionnel pour ces câbles PL2303 n'est pas installé sur les PCs de laboratoires. Vous pouvez utiliser votre propre machine avec le driver disponible ici :

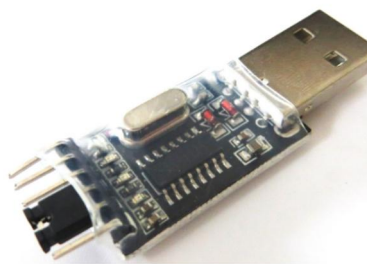
\\eseo-etudiants\\depots\\depot_prof\\depot_tmp\\spoiraud_setup\\Drivers_USB_Serie\\PL2303_Windows10

Les adaptateurs USB série PL2303 sont câblés ainsi :

- **Noir** : GND
- **Vert** : Tx du PC, données du PC vers la carte (relié au PC7, Rx du microcontrôleur)
- **Blanc** : Rx du PC, données de la carte vers le PC (relié au PC6, Tx du microcontrôleur)
 - o Mnémotechnie : le PC regarde avec le **Blanc** (comme le **Blanc** des yeux)
- **Rouge** : NON utilisé

Ouvrez un logiciel permettant d'accéder aux données du port série.

Adaptateur sur port USB CH340G



Fonctionnent dans les PCs de labos, mais « accrochés » au port USB du PC...

Pensez à « croiser » :

- Rx de l'adaptateur relié au Tx du microcontrôleur
- Tx du microcontrôleur relié au Rx de l'adaptateur

Comment utiliser l'interface série de la sonde de débogage en lieu et place d'un adaptateur USB série ?

La sonde de débogage de la Nucleo (ci-contre) se fait reconnaître (par l'ordinateur auquel on la branche) en tant que :

- Une sonde de débogage
- **Un port série virtuel**
- Un périphérique de stockage de masse

Ce port série virtuel est l'équivalent d'un câble USB série (cf page précédente).

La liaison série reliée à ce port virtuel déclaré au PC est :

- Accessible via le connecteur RxTx (en rouge ci-contre)
- Reliée au microcontrôleur de la carte Nucleo (sur son UART2, sur PA2 et PA3).

Rx et Tx sont à considérer par rapport à la sonde de débogage... donc par rapport au PC :

- Le PC **R**eçoit sur **R**x
- Le PC **T**ransmet sur **T**x

Attention, pour utiliser cette liaison avec une bluepill :

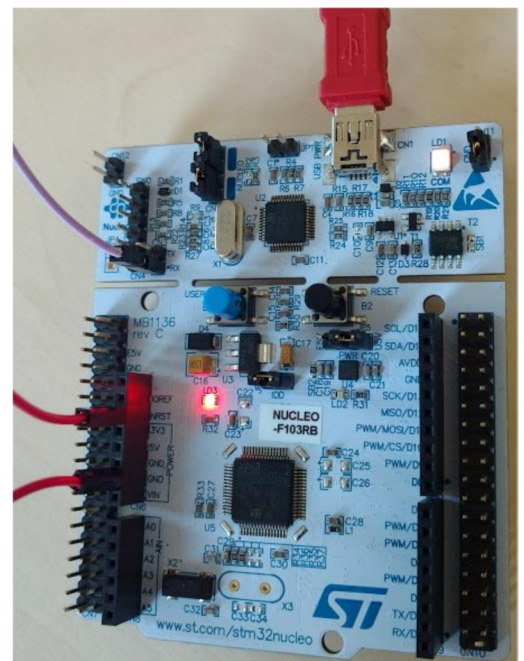
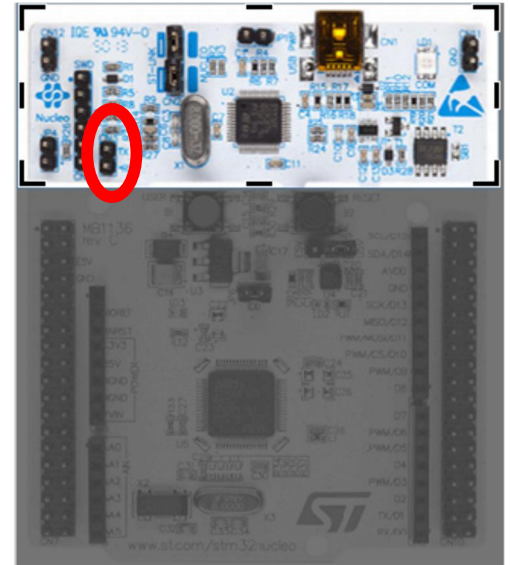
- Relier le Tx (PA2 si UART2 ; PB6 si UART1 ...) de la bluepill à la broche Rx du connecteur
- **Si besoin**, relier le Rx (PA3 si UART2; PB7 si UART1 ...) de la bluepill à la broche Tx
- **ATTENTION !** Il ne faut pas être perturbé par le microcontrôleur de la Nucleo.

4 solutions sont envisageables :

- o Effacer la mémoire flash du microcontrôleur ('erase' avec le logiciel ST-LINK Utility)
- o insérer un programme sans initialisation de l'UART2 (en mettant un while(1); dès le début de la fonction main)
- o placer un fil entre NRST et GND pour imposer son reset permanent (comme ci-dessous)
- o Embaucher un binôme pour maintenir le bouton reset enfoncé durant des heures... (solution non recommandée, sauf si vous avez une revanche à prendre).

Photo ci-contre :

- fil violet = vers le Tx de la Bluepill
- fil rouge = reset !



Démarche à suivre pour le développement de votre application.

- Consultez le document « [périphériques.pdf](#) » disponible sur le campus pour voir si les composants que vous utilisez s’y trouvent présentés.
- Identifiez les autres briques logicielles qui seront utiles à votre projet et qui sont déjà fournies ou partiellement fournies.
- Architecturez votre application, en séparant chaque capteur ou actionneur pour chaque module.
- **Décrivez précisément le comportement** que vous souhaitez pour votre application du point de vue de l'utilisateur
 - L'utilisateur appuie sur tel bouton... il se passe ceci...
 - L'utilisateur voit ceci sur l'écran et doit faire cela...
 - Le buzzer sonne lorsque ceci...
 - La led rouge s'allume lorsque cela...
- Rassemblez les informations nécessaires sur les composants, capteurs ou actionneurs que vous utilisez. Cherchez à comprendre comment ils se pilotent et quels signaux transitent.
- Votre démarche de développement doit chercher à **découper les fonctionnalités pour valider chaque module indépendamment des autres avant de rassembler le tout** : il NE FAUT PAS chercher à construire l'application directement... mais adopter une démarche progressive, pas à pas, pour valider au fur et à mesure les briques élémentaires du logiciel développé.
- Lorsque chaque module fonctionne individuellement, construisez la machine à états qui rassemble ces modules au service de votre application complète. N'hésitez pas à (re)-consulter la vidéo d'explication sur comment développer une machine à états :
 - <https://www.youtube.com/watch?v=OfzIS3BExxM>

A chaque bogue rencontré, il faut chercher à l'isoler en « resserant l'étau ». Les outils de débogage d'Eclipse, l'oscilloscope ou le multimètre, sont autant de pistes pour chercher à isoler ces erreurs.

Cahier de suivi

Rappel :

le documents livrables.pdf, disponible sur le campus, liste les livrables attendus en fin de projet. Une trame de rapport à remplir est fournie. Nous vous invitons à consulter cette trame, à la remplir au fur et à mesure. Nous attirons également votre attention sur la rubrique « **cahier de suivi** », qui nécessite un **remplissage régulier**, séance par séance.

Conception modulaire – comment utiliser un module logiciel fourni.

Nous mettons à votre disposition plusieurs modules logiciels, permettant de piloter bon nombre de capteurs et actionneurs disponibles pour le DEEP.

Afin d’alléger la compilation, et de faire en sorte que le programme construit tienne dans l’espace mémoire restreint de la cible STM32F103, une majorité de ces modules logiciels sont désactivés par défaut (non inclus à la compilation et/ou vides à la compilation par un mécanisme de macros).

Voici comment activer un module logiciel fourni, sur la base d’un exemple représentatif.

- 0- Je veux activer le module logiciel MPU6050 (composant accéléromètre + gyroscope)
- 1- Dans eclipse, dans la vue projet, dans lib/bsp,
 - ➔ Clic droit sur le dossier « barré » : ‘MPU6050’
 - ➔ Ressources config
 - ➔ Exclude from build
 - ➔ Décocher les deux cases ‘Debug’ et ‘Release’

Le dossier fait maintenant partie des sources compilées.
- 2- On remarque que le contenu du fichier est grisé à partir du `#if USE_MPU6050`
 - ➔ Cette macro doit être placée à 1 dans config.h !
- 3- Il faut imposer à Eclipse de rafraîchir sa lecture des fichiers pour qu’il voit ce qui a changé
 - ➔ Bouton-droit sur le nom du projet, index, freshen all files
 - ➔ Bouton-droit sur le nom du projet, index, Rebuild

Le module logiciel est maintenant utilisable. Pour certains modules, une fonction de **test ou de demo** est intégrée. (parfois dans le fichier appli/test.c, parfois directement dans le module).

Attention, certaines de ces demos sont dites ‘blocantes’. Si vous les appelez, ces fonctions ne se terminent jamais (et ne rendront donc pas la main à l’appelant). Inspirez vous des codes fournis dans ces démos pour les adapter à votre contexte.

Conception modulaire – comment ajouter un module logiciel.

Dans le même esprit que l'application déjà proposée, vous devez construire votre programme sous la forme de modules logiciels (un module = 1 fichier C + 1 fichier H).

Référez vous à la feuille A3 « Synthèse du langage C » ou au document « Le C embarqué, par l'exemple » pour savoir ce que contient chaque fichier C ou H.

Prenons un exemple purement stupide.

Admettons que vous réalisez un véhicule mobile et que votre application utilise un capteur de pluie.

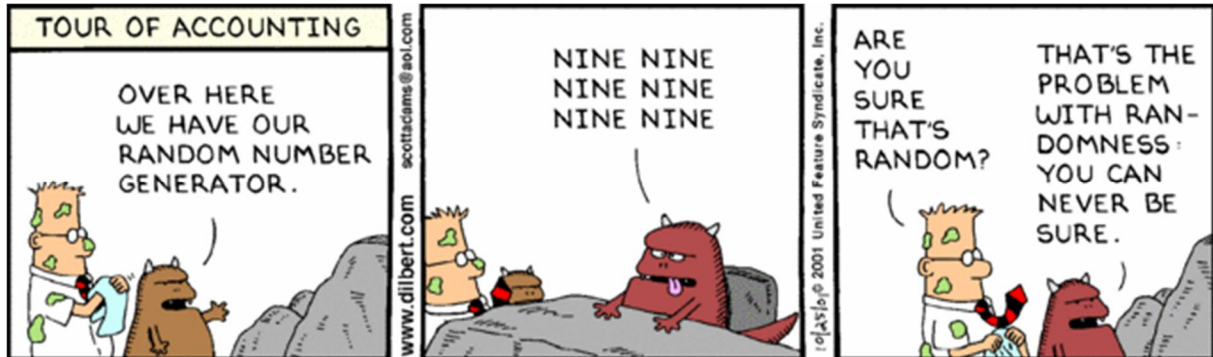
Vous pouvez construire le module logiciel rain.c / rain.h.

Ce module contient **par exemple** les fonctions suivantes :

void RAIN_init(void);	Initialise le capteur. (configuration des broches et des périphériques utilisés...)
uint8_t RAIN_get_intensity();	Retourne l'intensité de la pluie mesurée.
void RAIN_process_main(void);	Cette fonction doit être appelée par la boucle de tâche de fond. Elle contient par exemple une machine à état et assure le bon fonctionnement du module logiciel.
void RAIN_process_1ms(void);	Cette fonction doit être appelée chaque milliseconde par la routine d'interruption d'un timer.

Comment générer un nombre aléatoire.

Dans certaines application (notamment les jeux), on peut avoir besoin de générer des nombres aléatoires.



Tout en étant conscient qu'un algorithme ne peut pas produire un nombre aléatoire, il est toutefois possible de générer des suites pseudo-aléatoires...

Un site Internet, www.random.org, se targue de proposer un « vrai aléatoire », en utilisant du bruit blanc issu d'une mesure atmosphérique.

Perhaps you have wondered how predictable machines like computers can generate randomness. In reality, most random numbers used in computer programs are pseudo-random, which means they are generated in a predictable fashion using a mathematical formula. This is fine for many purposes, but it may not be random in the way you expect if you're used to dice rolls and lottery drawings.

*RANDOM.ORG offers **true** random numbers to anyone on the Internet. The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs.*

Sur la plateforme proposée, vous disposez de plusieurs solutions pour générer une suite de nombre aléatoire :

- ➔ Dans certains cas, on peut utiliser le fait que l'utilisateur est lent dans ses mouvements. S'il doit appuyer sur un bouton, on peut compter le temps qu'il met avant d'appuyer, ou le temps pendant lequel il appuie, et interpreter ce temps comme une variable aléatoire. Bien malin sera l'utilisateur, incapable de piloter son appui à la milliseconde près !
- ➔ Il est également possible d'utiliser des algorithmes qui génèrent des suites de nombres « pseudo-aléatoires »... mais comme tout algorithme, si les conditions initiales sont les mêmes, les suites de nombres sont également répétables.

Comment recevoir une chaîne de caractère sur une liaison série (UART) ?

La fonction « `uint8_t UART_getc(uart_id_e uart_id)` » renvoie :

- soit 0 (0x00) si aucun caractère n'a été reçu depuis le précédent appel de la fonction
- soit le caractère reçu

(Vous noterez qu'il n'est donc pas possible avec cette fonction de savoir que l'on reçoit des caractères nuls.)

Pour recevoir une chaîne complète, il faut remplir un tableau d'`uint8_t`, en faisant avancer un index.

La fonction en question ne doit pas être bloquante... donc pas de **while**.

Les variables **tab** et **index** doivent garder leur valeur d'un appel à l'autre de la fonction : donc elles doivent être déclarées avec le mot clé **static**.

```
int main(void)
{
    //...
    while(1)
    {
        process_uart_rx();
        //...
    }
}

//Exemple de fonction permettant de recevoir une chaîne de caractère et de la traiter
lorsqu'elle est entièrement reçue
//Adaptez là à votre situation !
#define TAB_SIZE      20
void process_uart_rx(void)
{
    static uint8_t tab[TAB_SIZE];
    static uint16_t index = 0;
    uint8_t c;
    c = UART_getc(UART1_ID);           //lecture du prochain caractère
    if(c)                               //Si on a reçu un caractère (!=0)
    {
        tab[index] = c;                //On mémorise le caractère dans le tableau
        if(c=='\n')                    //Si c'est la fin de la chaîne
        {
            tab[index] = 0;             //fin de chaîne, en écrasant le \n par un 0
            analyse_string(tab);         //traitement de la chaîne...(au choix)
            index = 0;                  //Remise à zéro de l'index
        }
        else if(index < TAB_SIZE - 1)
        {
            index++;                    //Pour tout caractère différent de \r ou \n
                                        //on incrémente l'index (si < TAB_SIZE !)
        }
    }
}
```

Comment faire une détection d'appui pour un bouton poussoir ?

Vous êtes invités à comprendre ce code et à l'expliquer à votre enseignant pour vous assurer que vous l'avez bien compris.

```
//Détecteur d'appui sur un bouton
bool_e button_press_event(void)
{
    static bool_e previous_state = FALSE;
    bool_e current_state;
    bool_e ret;
    current_state = !HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_15);
    ret = current_state && !previous_state;
    previous_state = current_state;
    return ret;
}
```

Et pour plusieurs boutons ?

Vous pouvez vous inspirer de cet exemple (qui n'est qu'une possibilité parmi d'autres) !

```
typedef enum
{
    BUTTON_ID_NONE = 0,
    BUTTON_ID_LEFT,
    BUTTON_ID_RIGHT,
    BUTTON_ID_UP,
    BUTTON_ID_DOWN,
    BUTTON_ID_NB // Le nombre de boutons dans la liste...
}button_id_e;

button_id_e button_press_event(void)
{
    static bool_e previous_state[BUTTON_ID_NB] = {FALSE};
    button_id_e ret = BUTTON_ID_NONE;
    button_id_e button_id;
    bool_e current_state;
    for(button_id = BUTTON_ID_LEFT; button_id<BUTTON_ID_NB; button_id++)
    {
        switch(button_id)
        {
            case BUTTON_ID_LEFT:
                current_state = !HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_13);
                break;
            case BUTTON_ID_RIGHT:
                current_state = !HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_11);
                break;
            case BUTTON_ID_UP:
                current_state = !HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_15);
                break;
            case BUTTON_ID_DOWN:
                current_state = !HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_12);
                break;
            default:
                current_state = 0; break;
        }
        if(current_state && !previous_state[button_id])
            ret = button_id;
        previous_state[button_id] = current_state;
    }
    return ret;
}
```