

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);
}
```

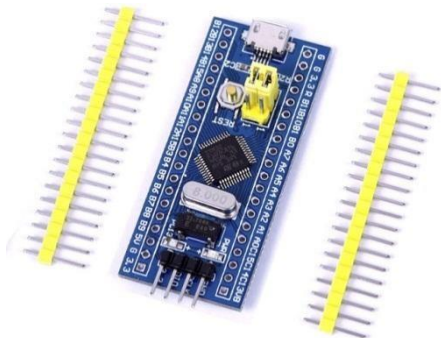
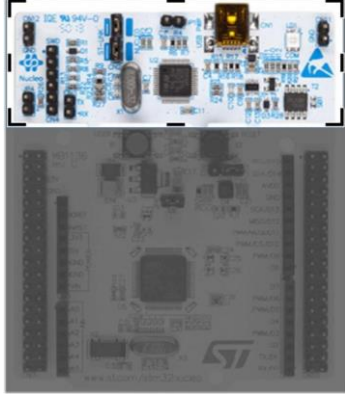
2022/2023

Objectifs de la mission :

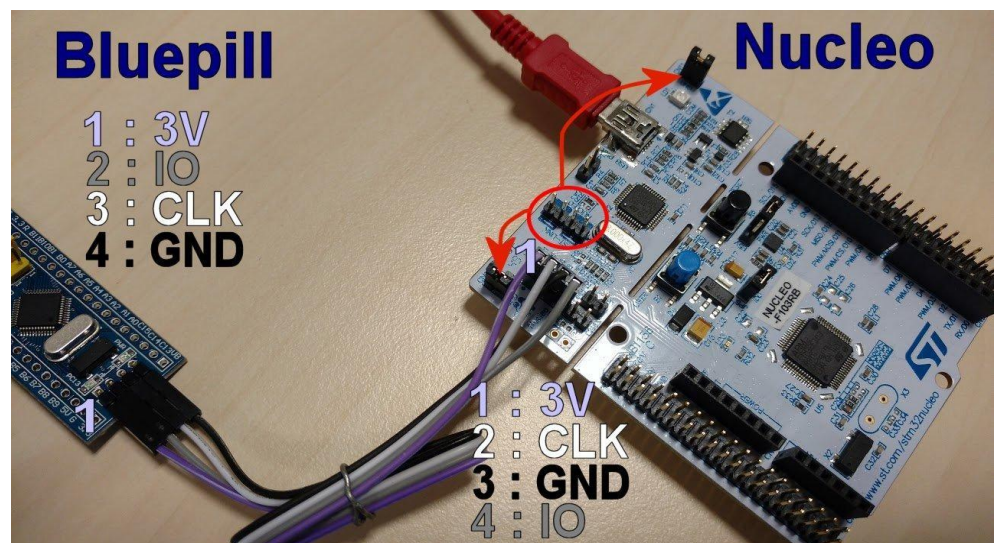
- Développement d'un logiciel embarqué pour répondre au sujet choisi
- Développement de briques logicielles
- Intégration de briques logicielles existantes (bibliothèques et modules fournis)
- Tests unitaires
- Tests d'intégration

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 :

Bluepill (carte support du STM32F103)	Sonde de débogage (partie haute de la carte Nucleo F103)
	

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



N'oubliez pas de retirer les deux cavaliers (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 noté « 3V » côté Nucleo vers le net 3,3V sans se tromper car un 5V serait fatal !!!!!!!)

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 "Electronique numérique".

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

Un « **projet-f103** », disponible sur le Moodle, contient plusieurs briques logicielles que vous pouvez utiliser.

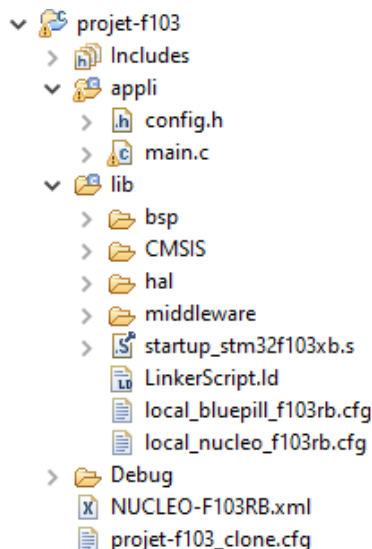
Le fichier du projet s'intitule "f103-master.zip"

Accéder au projet-f103.

1 Dans "STM32CubeIDE", importez le fichier " f103-master.zip "

- a. File
- b. Import
- c. Existing project into workspace
- d. Select archive file
- e. Finish

2 Vous obtenez l'arborescence ci-dessous :



3 **Compilez ce projet**



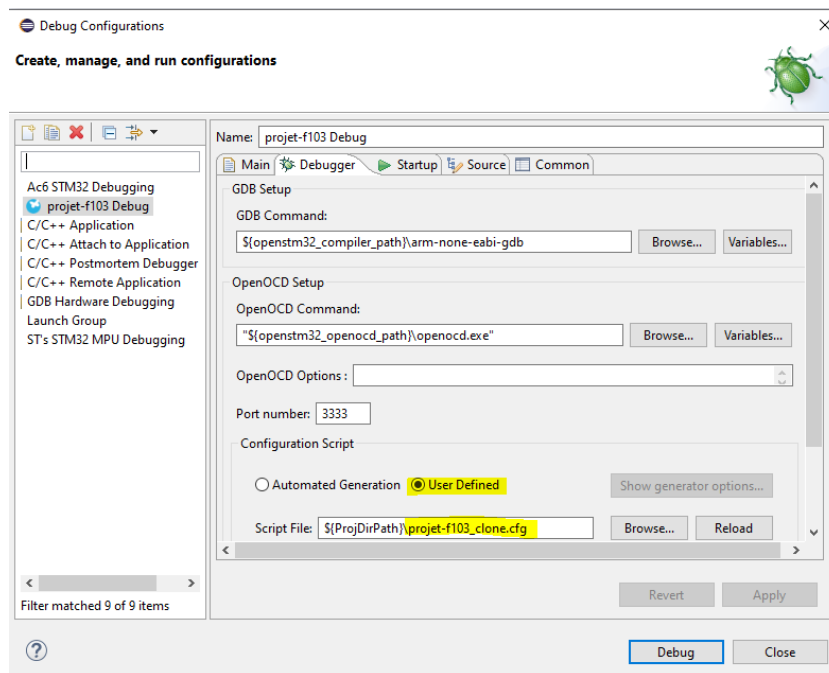
4 Sur le nom du projet : **Bouton-droit -> Refresh** (cela permet de rafraîchir la visibilité du binaire créé lors de la compilation)



5 Run -> Debug configuration -> double clic sur Ac6 STM32 Debugging (pour générer la cible de débogage)

6 Vérifiez que la case C/C++ Application contient bien « **Debug\projet-f103.elf** ». Dans l'onglet « **Debugger** », dans la section « **script** », choisissez « **user defined** ».

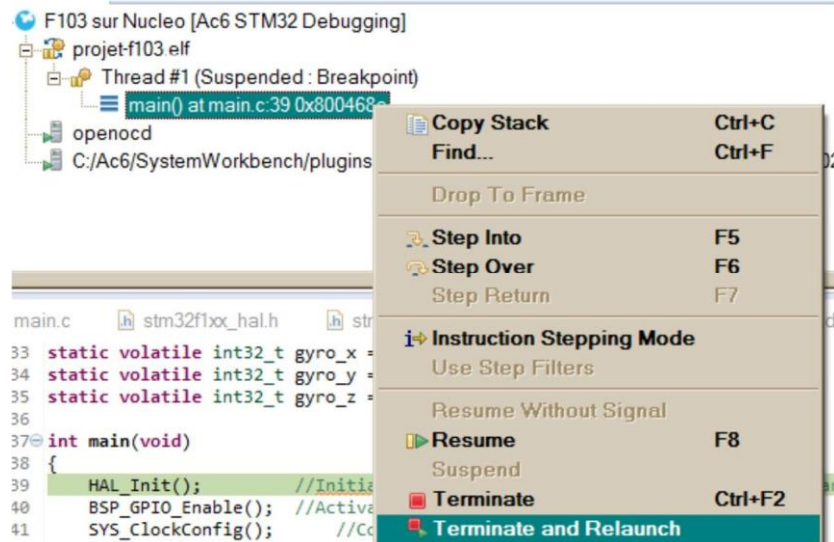
7 Cliquez sur « Browse », et localisez le fichier projet-f103_clone.cfg qui se trouve dans le projet. (Ce script est utile pour configurer le reset software de la cible. Il permet aussi de programmer les clones contrefaits.)



8 Lancez le programme sur la cible

9 **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.



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.

Comment utiliser une liaison série pour communiquer 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)

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

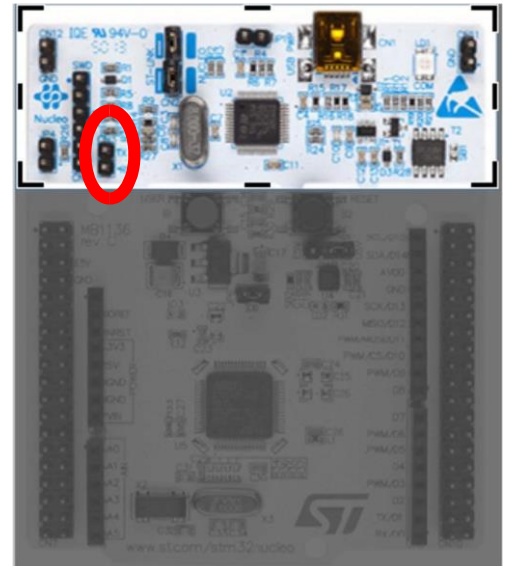
- Une sonde de débogage
- **Un port série virtuel**
- Un périphérique de stockage de masse

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 Reçoit sur Rx
- Le PC Transmet sur Tx

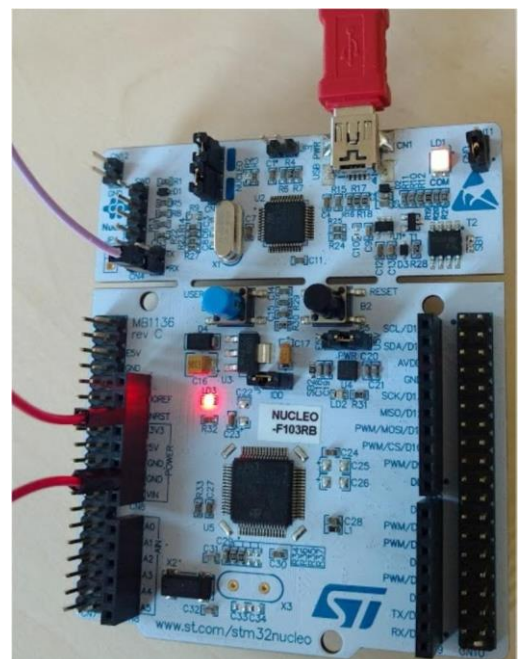


Attention, pour utiliser cette liaison avec une bluepill :

1. Relier le Tx (PA2 si UART2 ; PB6 si UART1 ...) de la Bluepill à la broche Rx du connecteur
2. **Si besoin**, relier le Rx (PA3 si UART2; PB7 si UART1 ...) de la Bluepill à la broche Tx
3. **ATTENTION !** Il ne faut pas être perturbé par le microcontrôleur de la Nucleo. 4 solutions sont envisageables :
 - a. Effacer la mémoire flash du microcontrôleur ('erase' avec le logiciel ST-LINK Utility)
 - b. Insérer un programme sans initialisation de l'UART2 (en mettant un while(1); dès le début de la fonction main)
 - c. Placer un fil entre NRST et GND pour imposer son reset permanent (comme ci-contre)
 - d. 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.

- Identifiez les briques logicielles qui seront utiles à votre projet et qui sont déjà fournies ou partiellement fournies.
- Architecturez votre application avec cette même méthode, 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 le logiciel développé.

A chaque bogue rencontré, il faut chercher à l'isoler en "resserrant 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

Une trame de rapport finale à remplir est fournie sur le Moodle. 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 des capteurs et actionneurs disponibles.

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 inclut 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 'bloquantes'. 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.

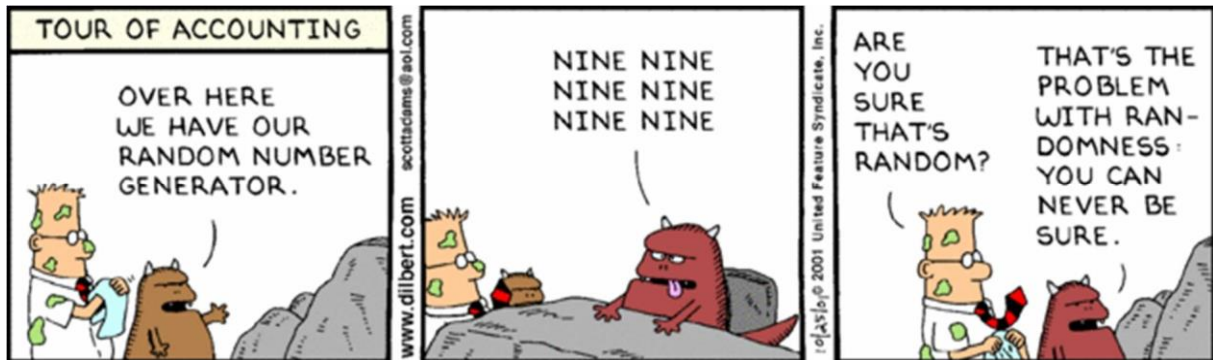
Vous pouvez construire le module logiciel rain.c / rain.h.

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;
}
```

Liste des modules disponibles

Drivers de périphériques internes

ADC	<p>Convertisseur analogique-numérique. Cette fonctionnalité permet de mesurer la tension analogique sur certaines entrées dédiées du microcontrôleur puis de la convertir en un nombre que le programme pourra utiliser.</p> <p>Le STM32F103RB comporte 2 ADCs utilisables sur 16 ports physiques.</p> <p>Voir les fichiers : <i>stm32f1_adc.c</i>, <i>stm32f1_adc.h</i></p>
GPIO	<p>Les ports GPIO (General Purpose Input Output) sont des ports d'entrée/sortie à usage général. Configurés en sortie, ils permettent d'émettre un signal logique, tandis qu'ils les réceptionnent s'ils sont configurés en entrée. Les signaux doivent être numériques. Ces ports peuvent être reliés à des périphériques internes/externes et chacun d'entre eux possède donc des usages possibles différents. Ainsi, le port PA1 peut être utilisé pour le timer 2, pour l'UART (sortie RTS), ou encore pour l'ADC par exemple (voir le fichier PDF <i>Ports_STM32F1</i>).</p> <p>Il est également possible de configurer un port en <u>sortie haute impédance</u> (drain ouvert), ou bien <u>une entrée avec un tirage pull-up</u> (ou pull-down).</p> <p>Voir les fichiers : <i>stm32f1_gpio.c</i>, <i>stm32f1_gpio.h</i></p>
I²C	<p>L'I²C est un bus de données série synchrone bidirectionnel. Il permet d'effectuer des communications entre un maître et un ou plusieurs esclaves. De nombreux périphériques externes utilisent le protocole I²C, tels que des capteurs.</p> <p>Voir les fichiers : <i>stm32f1_i2c.c</i>, <i>stm32f1_i2c.h</i>, ainsi que la page wikipedia sur l'I²C.</p>
PWM	<p>Ce module sert à générer des signaux PWM (Pulse Width Modulation), c'est-à-dire des signaux logiques à fréquence choisie et dont on fait varier le rapport cyclique ; donc la valeur moyenne. Ce module utilise des timers en mode PWM pour contrôler l'envoi des signaux.</p> <p>Les signaux PWM sont utilisés dans des applications comme le contrôle de moteur ou les télécommunications.</p> <p>Le STM32F103RB comporte 4 timers, pouvant chacun générer 4 canaux PWM.</p> <p>Voir les fichiers : <i>stm32f1_pwm.c</i>, <i>stm32f1_pwm.h</i></p>
RTC	<p>Module horloge temps réel fournissant des fonctionnalités de réglage et de lecture de l'heure et de la date, l'alarme... La RTC connaît le nombre de jour par mois selon l'année ! Pour l'utiliser avec une bonne précision, il faut exploiter un quartz externe au lieu de l'oscillateur interne.</p>

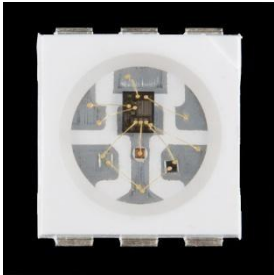
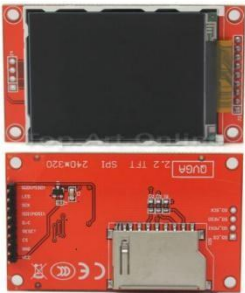
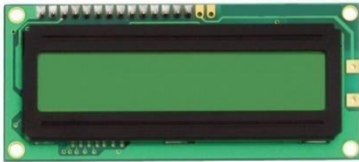
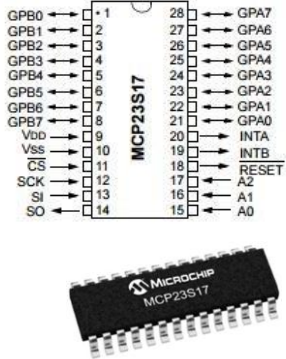
	Voir les fichiers : <i>stm32f1_rtc.c, stm32f1_rtc.h</i>
SPI	<p>Le SPI est un bus de données série synchrone bidirectionnel. Il sert à effectuer des communication entre un maître et un ou plusieurs esclaves.</p> <p>Il est souvent utilisé pour communiquer avec, par exemple, des mémoires, des cartes SD, ou encore des systèmes d'affichage.</p> <p>Généralement plus rapide que l'I2C, il impose un signal CS (Chip Select) par esclave.</p> <p>Voir les fichiers : <i>stm32f1_spi.c, stm32f1_spi.h</i></p>
Timer	<p>Ce module permet de gérer les 4 timers présents sur le STM32F103. On peut ainsi les configurer, choisir si l'on veut activer ou non les interruptions timer et, bien sûr, les lancer.</p> <p>Voir les fichiers : <i>stm32f1_timer.c, stm32f1_timer.h</i></p>
UART	<p>Ce module sert à utiliser les 3 périphériques USART (Universal Synchronous/Asynchronous Receiver Transmitter) du STM32F103. Un périphérique USART permet d'envoyer/recevoir des données via une liaison série.</p> <p>Avec le module fourni, il est possible d'initialiser les UARTs et d'envoyer/de recevoir des caractères ou chaînes de caractères sur la liaison série.</p> <p>L'UART est bien utile pour débiter un programme, par exemple, en affichant les données que l'on souhaite visualiser.</p> <p>Voir les fichiers : <i>stm32f1_uart.c, stm32f1_uart.h</i></p>
SysTick	<p>Ce module permet d'utiliser le timer système pour générer des interruptions. Il offre la possibilité d'ajouter (et de retirer) jusqu'à 16 fonctions appelées lors de chaque IT déclenchée par le timer système. Les fonctions sont stockées dans un tableau de pointeur sur fonctions.</p> <p>Voir les fichiers : <i>stm32f1_systick.c, stm32f1_systick.h</i></p>

Drivers de périphériques externes

Pour chacun de ces modules logiciel, le fichier appli/config.h fournit un #define qui peut être placé à 1 ou à 0 pour activer ou désactiver le module logiciel. Cette désactivation permettant notamment de gagner de la place sur la mémoire du microcontrôleur.

Module	Description
Clavier 16 touches TTP229-BSF	<p>Clavier à 16 touches capacitives, numérotées de 1 à 16. 1 appui maximal en même temps.</p> <p>Voir le fichier : <i>CapacitiveKeyboard.c</i></p> <p>http://www.tontek.com.tw/download.asp?sn=726</p> 
Clavier matriciel 16 touches	<p>Clavier à 16 touches permettant de saisir des combinaisons de chiffres et de lettres ainsi que certains caractères spéciaux (de 0 à 9, de A à D, # et *).</p> <p>Bien que le driver logiciel limite à 1 seul appui simultané, plusieurs appuis simultanés peuvent être détectés par cette technologie.</p> <p>Voir le dossier : <i>MatrixKeyboard</i></p>
Capteur de température à distance MLX90614	<p>Capteur infrarouge permettant de mesurer :</p> <ul style="list-style-type: none"> □ La température ambiante □ La température d'un objet distant <p>Il communique avec le microcontrôleur via I2C.</p> <p>Voir le dossier : <i>MLX90614</i></p> <p>https://www.sparkfun.com/datasheets/Sensors/Temperature/MLX90614_rev001.pdf</p> 
Accéléromètre + Gyroscope + Capteur de température MPU-6050	<p>Capteur regroupant 3 fonctionnalités :</p> <ul style="list-style-type: none"> • Accéléromètre 3 axes • Gyroscope 3 axes • Capteur de température <p>Il communique avec le microcontrôleur via I2C.</p> <p>ATTENTION, il est conseillé de relier la broche VCC à un GPIO du microcontrôleur. On pourra ainsi alimenter le capteur directement via ce GPIO, et provoquer un reset au début du programme ; indispensable pour maîtriser l'état du capteur après reset du microcontrôleur.</p> <p>Voir le fichier : <i>stm32f1_mpu6050.c</i></p> <p>https://www.cdiweb.com/datasheets/invensense/MPU6050_DataSheet_V3%204.pdf</p> 

un

Matrice de LEDs WS2812	<p>Matrice de 64 LEDs RGB WS2812. Ce module permet de contrôler chacune des LEDs (chacun des pixels) de la matrice en lui envoyant des trames RGB via le protocole NRZ pour choisir les couleurs des LEDs. Cicontre, photo d'une LED WS2812.</p> <p>Ces boitiers embarquent chacun 3 leds (R,G,B) ainsi qu'une petite intelligence permettant ce pilotage.</p> <p>Voir les fichiers : <i>MatrixLed/WS2812S.c</i> et <i>MatrixLed/WS2812S.h</i>.</p>	
	<p>https://cdn-shop.adafruit.com/datasheets/WS2812.pdf</p>	
Ecran TFT ILI9341	<p>Ecran TFT rétroéclairé permettant d'afficher des formes et des chaînes de caractères dans les couleurs de son choix.</p> <p>65536 couleurs. 240*320 pixels.</p> <p>Voir le fichier : <i>lcd_ili9341</i></p> <p>https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf</p>	
Ecran LCD 2*16	<p>Ecran LCD 2*16 permettant d'afficher des chaînes de caractères et de lire les caractères à l'écran.</p> <p>2 lignes de 16 caractères alphanumériques.</p> <p>Voir le dossier : <i>lcd_2x16</i></p>	
Extension d'E/S MCP23S17	<p>Extension d'entrées/sorties 16 bits (2*8 bits, GPIOA et GPIOB) avec interface SPI. Comme son nom l'indique, le module permet d'ajouter des ports d'entrée/sortie au microcontrôleur en se servant de son bus SPI. Via le SPI, l'utilisateur peut sélectionner chacun des ports de l'extension pour les configurer en entrée ou en sortie, leur envoyer des données ou les lire. Ces ports peuvent aussi fonctionner en interruption.</p> <p>le driver logiciel correspondant à ce composant n'est pas fourni dans la lib, consultez M. Poiraud.</p> <p>Voir le dossier : <i>MCP23S17</i></p> <p>http://ww1.microchip.com/downloads/en/DeviceDoc/20001952C.pdf</p>	

Moteur DC	<p>Ce module permet de piloter un moteur à courant continu via un pont en H BD6221F-E2 (ou autre composant utilisant les mêmes commandes). En fonction de la combinaison de signaux logiques envoyée sur les broches FIN et RIN du pont en H, le moteur fonctionnera en roue libre, en marche avant, en marche arrière ou en court-circuit.</p> <p>Le module utilise le driver PWM pour générer les signaux.</p> <p>Voir les fichiers : <i>stm32f1_motorDC.c</i>, <i>stm32f1_motorDC.h</i></p> <p>http://rohmfs.rohm.com/en/products/databook/datasheet/ic/motor/dc/bd622x-e.pdf</p>
------------------	---