# Mission 1 – Le détecteur de gravité

**2015**

Objectifs de la mission :

- Prise en main des outils, de la cible, de l'environnement de développement
- Lecture et modification d'un programme existant
- Utilisation du périphérique GPIO
- Utilisation du débogueur

# 1 Quelques rappels

Dans une majorité des systèmes embarqués, on trouve aujourd'hui un ou plusieurs microcontrôleurs.
Nous vous proposons de prendre en main un microcontrôleur STM32.

**Qu'est-ce qu'un microcontrôleur ?**

C'est un circuit intégré qui contient un microprocesseur (capable d'effectuer des opérations arithmétiques et logiques, et d'exécuter des instructions élémentaires), de la mémoire et des périphériques. C'est un système autonome à lui tout seul. On peut se représenter un microcontrôleur comme un « petit ordinateur ».

**Que peut-on faire avec un microcontrôleur ?**

Un microcontrôleur contient un **programme** qu'il exécute. Il interagit avec son environnement électrique. On peut s'en servir pour interfacer des capteurs et des actionneurs.

**Qu'est-ce qu'un programme ?**

Le **programme informatique**, c'est comme une recette de cuisine. Chaque instruction de la recette de cuisine informe le cuisinier des ingrédients à mélanger dans un ordre précis. Les mots et la langue employés sont assimilables au **langage informatique** (grammaire, syntaxe). Les ingrédients sont les ressources disponibles, dans notre cas ce sera des **variables**. Le processeur exécute bêtement le programme informatique, soit l'ensemble des instructions dans l'ordre prédéfini. La manière d'écrire un ensemble d'instructions ayant un objectif bien précis s'appelle un **algorithme.** Toute intelligence artificielle, aussi développée soit elle, est un simple programme.

**Comment écrire un programme et le faire exécuter ?**

Les différentes étapes entre l'écriture du programme et son exécution sur la cible sont :

- Rédaction du programme. (On utilise souvent un logiciel qui nous aide en proposant une coloration, un pré-remplissage, une navigation intelligente)
- Compilation (conversion du programme en instructions élémentaires que le processeur peut comprendre).
- Edition des liens (on rassemble les différents modules logiciels)
- Chargement du programme dans la mémoire du microcontrôleur
- (Débogage) : on contrôle l'exécution du programme (en regardant au fur et à mesure l'état des variables par exemple)

**Qu'est-ce qu'un périphérique ?**

Le microcontrôleur contient un ensemble de périphériques spécifiques. En voici quelques exemples avec leurs usages typiques :

**Traiter des signaux d'entrée sortie.**

- GPIO (General Purpose Input Output) : lire l'état logique d'une broche d'entrée ou produire un niveau logique sur une broche de sortie
- ADC : (Analog to Digital Converter), convertir une tension analogique en un nombre
- DAC : (Digital to Analog converter), convertir un nombre en une tension analogique
- PWM : produire un signal à rapport cyclique variable (valeur moyenne variable)

**Mémoriser des données.**

- EEPROM : mémoire non volatile

**Communiquer avec l'extérieur selon des protocoles définis.**

- UART : (Universal Asynchronous Receiver Transmitter), transmettre et recevoir des octets sur une liaison série
- USB : (Universal Serial Bus). Acceuillir un périphérique USB ou bien se faire reconnaître comme un périphérique USB
- SPI / I2C / I2S / CAN : interfaçage sur un bus de communication standard, pour communiquer avec un capteur, un actionneur, un autre microcontrôleur…

**Compter le temps.**

- RTC : horloge temps réel, assurer la connaissance de l'heure, du jour dans l'année…. Fournir un calendrier.
- Timer : compter des évènements, appeler une fonction périodiquement

**Qu'allons-nous faire dans la suite du projet ?**

1- Choisir un sujet

La contrainte : utiliser ce microcontrôleur STM32F407VGT6 et sa carte support.

2- Prendre en main quelques périphériques pour comprendre comment sont constituées et comment utiliser des librairies fournies

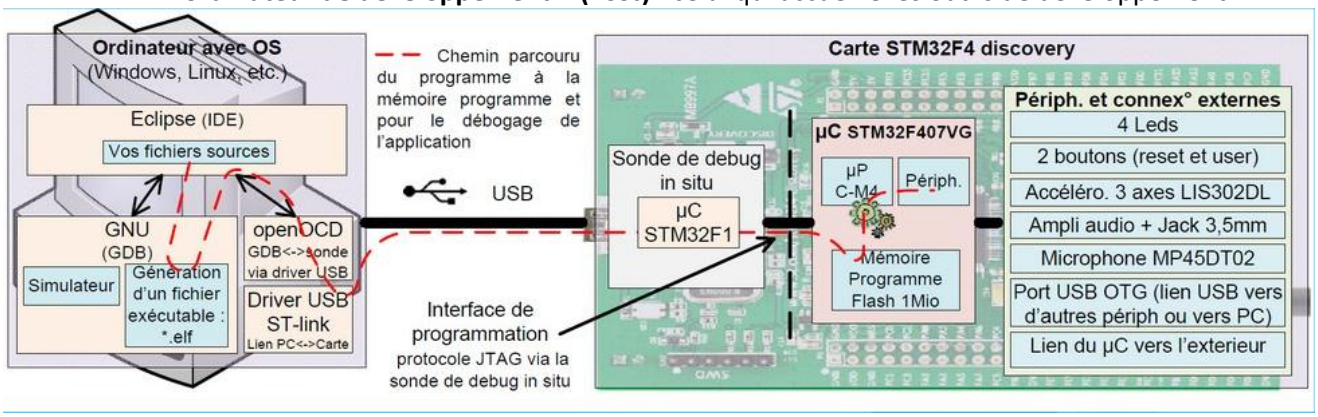
3- Relier (selon votre sujet) quelques périphériques externes et répondre à l'objectif choisi en programmer le microcontrôleur.

Il existe, pour chaque architecture d'ordinateur, une suite spécifique d'outils de développement. Fort heureusement, de nombreux outils sont les mêmes pour différentes architectures (au pire, ils sont grandement similaires). Ainsi, la maîtrise d'une chaîne d'outils spécifique à une plateforme vous facilitera la découverte d'une autre chaîne d'outils.
Dans cette activité pratique, nous vous proposons de découvrir progressivement une suite d'outils de développement pour le microcontrôleur STM32F407VGT6.

## 1.1 Avant propos sur le « développement croisé »

Dans la mesure où **l'ordinateur que nous allons programmer** ne dispose ni d'écran ni de ressources suffisantes pour accueillir une suite d'outils de développement, nous devons procéder à ce que nous appelons du « développement croisé» (*cross development*).
Cette technique de développement s'oppose au « développement natif» (*native development*) où les outils de développement et le logiciel développé s'exécutent sur le même ordinateur (en tout cas sur le même type d'ordinateur).
Par la suite, nous désignons ainsi :

- **« cible » (*target*)** : l'ordinateur que nous programmons (ici le microcontrôleur STM32).
- **« ordinateur de développement » (*host*)** : celui qui accueille les outils de développement.

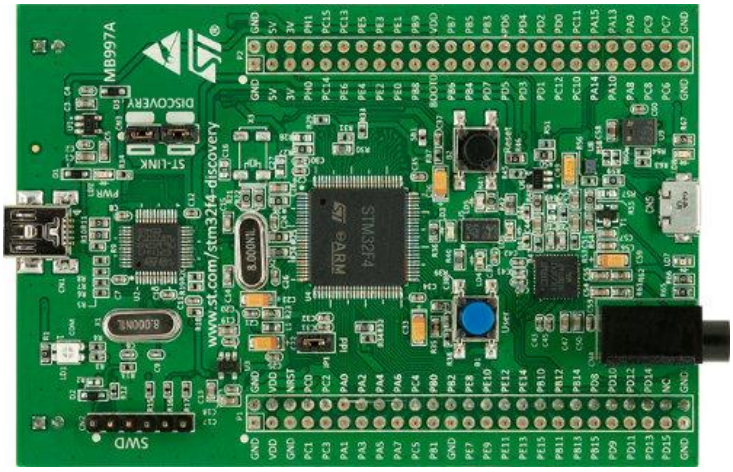


## 1.2 Description et branchement du matériel

Vous devez disposer :

- D'un câble usb mini B
- D'une carte STM32F4-DISCOVERY

Cette carte est produite et commercialisée par ST à destination des développeurs qui peuvent utiliser simplement et à moindre frais le microcontrôleur de ST.
Cette stratégie commerciale est très répandue et pas uniquement pour les microcontrôleurs : « *regardez cette carte de démo, jouez avez… vous avez vu comme il est bien mon micro… faites votre proto facilement avec…vous pouvez maintenant en*

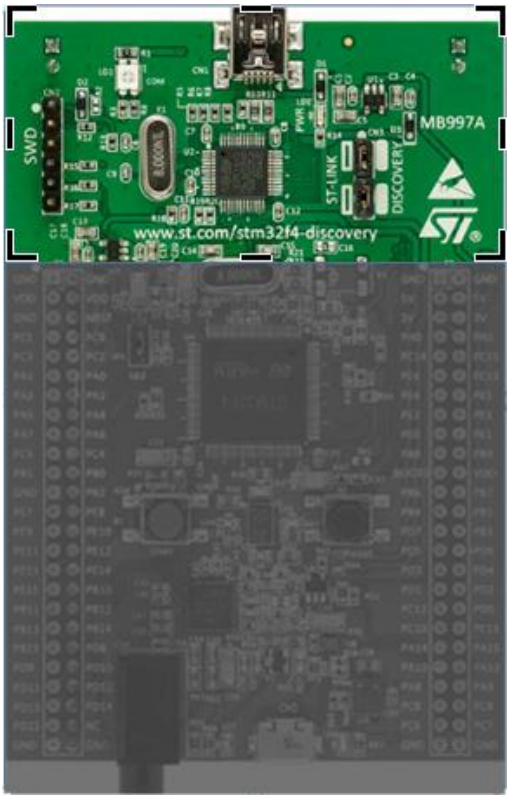



La carte dispose d'un ensemble de périphériques prêts à l'emploi : boutons, leds, accéléromètre, ampli audio, microphone. On peut également acheter une *mezzanine* qui enrichit encore les possibilités d'extensions (LCD, Caméra, Ethernet). La carte est pensée pour faciliter la fabrication d'une *mezzanine custom*.

La sonde de débogage in situ (qui assure le lien et permet la programmation du microcontrôleur par un ordinateur) est intégrée à la carte de démonstration :

Un second microcontrôleur, déjà programmé à cet effet, joue ce rôle de sonde de débogage et de programmation.

En retirant les deux cavaliers (en haut de l'image), on peut d'ailleurs utiliser cette sonde pour programmer un autre microcontrôleur que celui qui est sur la carte.




Nous constatons que de nombreuses broches peuvent nous permettre de connecter à la carte à un support quelconque (propre à chaque application). Ces broches sont… métalliques… et donc… conductrices. Il est donc judicieux de ne pas poser la carte alimentée sur un matériau conducteur ! Sinon, pschiit…

**Caractéristiques techniques du microcontrôleur STM32F407VGT6 :**

life.augmented

# STM32F405xx
# STM32F407xx

## ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera

**Datasheet - production data**

LQFP64 (10 × 10 mm)
LQFP100 (14 × 14 mm)
LQFP144 (20 × 20 mm)
LQFP176 (24 × 24 mm)

WLCSP90


UFBGA176
(10 × 10 mm)

## Features

- Core: ARM 32-bit Cortex™-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 168 MHz, memory protection unit, 210 DMIPS/ 1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
  - Up to 1 Mbyte of Flash memory
  - Up to 192+4 Kbytes of SRAM including 64-Kbyte of CCM (core coupled memory) data RAM
  - Flexible static memory controller supporting Compact Flash, SRAM, PSRAM, NOR and NAND memories
- LCD parallel interface, 8080/6800 modes
- Clock, reset and supply management
  - 1.8 V to 3.6 V application supply and I/Os
  - POR, PDR, PVD and BOR
  - 4-to-26 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC (1% accuracy)
  - 32 kHz oscillator for RTC with calibration
  - Internal 32 kHz RC with calibration
- Low power
  - Sleep, Stop and Standby modes
  - $V_{BAT}$ supply for RTC, 20×32 bit backup registers + optional 4 KB backup SRAM
- 3×12-bit, 2.4 MSPS A/D converters: up to 24 channels and 7.2 MSPS in triple interleaved mode
- 2×12-bit D/A converters
- General-purpose DMA: 16-stream DMA controller with FIFOs and burst support
- Up to 17 timers: up to twelve 16-bit and two 32-bit timers up to 168 MHz, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
  - Cortex-M4 Embedded Trace Macrocell™
- Up to 140 I/O ports with interrupt capability
  - Up to 136 fast I/Os up to 84 MHz
  - Up to 138 5 V-tolerant I/Os
- Up to 15 communication interfaces
  - Up to 3 × $I^2C$ interfaces (SMBus/PMBus)
  - Up to 4 USARTs/2 UARTs (10.5 Mbit/s, ISO 7816 interface, LIN, IrDA, modem control)
  - Up to 3 SPIs (42 Mbits/s), 2 with muxed full-duplex $I^2S$ to achieve audio class accuracy via internal audio PLL or external clock
  - 2 × CAN interfaces (2.0B Active)
  - SDIO interface
- Advanced connectivity
  - USB 2.0 full-speed device/host/OTG controller with on-chip PHY
  - USB 2.0 high-speed/full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY and ULPI
  - 10/100 Ethernet MAC with dedicated DMA: supports IEEE 1588v2 hardware, MII/RMII
- 8- to 14-bit parallel camera interface up to 54 Mbytes/s
- True random number generator
- CRC calculation unit
- 96-bit unique ID
- RTC: subsecond accuracy, hardware calendar

**Table 1. Device summary**

| Reference | Part number |
|---|---|
| STM32F405xx | STM32F405RG, STM32F405VG, STM32F405ZG, STM32F405OG, STM32F405OE |
| STM32F407xx | STM32F407VG, STM32F407IG, STM32F407ZG, STM32F407VE, STM32F407ZE, STM32F407IE |

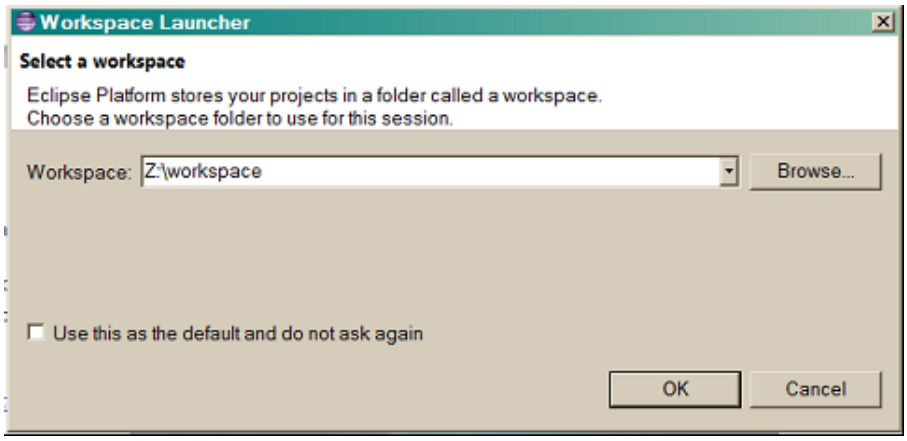
## 2- Lancement de l'environnement de développement intégré (en anglais : IDE)

Nous allons utiliser l'IDE « **System Workbench for STM32** ». Cet environnement est basé sur Eclipse, qui est un IDE libre de droit, très utilisé pour de nombreux langages.

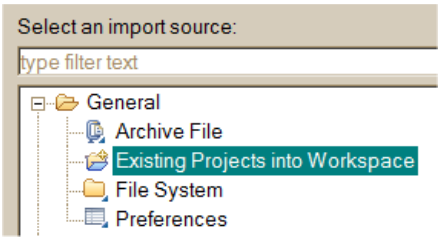
Si vous souhaitez installer les outils logiciels sur votre ordinateur, rendez-vous ici :
http://www.openstm32.org/
Ou directement sur le réseau de l'école :
\\eseonas\depots\depot_profs\depot_tmp\spoiraud_setup\stm32_win_64bits-v1.2.exe

1- Lancez **System Workbench for STM32**
2- Choisissez l'emplacement de votre espace de travail (workspace)

> Nous exigeons :
> Z:\un_dossier_pour_bien_ranger_les_choses\workspace
>
> *Comme pour tout projet, il est conseillé **d'éviter** les **caractères spéciaux ou accentués** dans le chemin… !*




3- Téléchargez l'archive zip du TD sur le campus.
4- Importez là : File -> Import -> General -> Existing project into workspace



> Lorsque l'on importe un projet, on peut choisir de le copier dans le workspace. (Cela devient d'ailleurs obligatoire pour l'importation d'un projet au format archive zip.)
>
> A l'inverse, un projet peut également être inséré au workspace, tout en étant rangé ailleurs.

Le projet importé est déjà configuré pour permettre la compilation, et la programmation de la cible. Nous reviendrons plus tard sur ces différentes configurations.

Si besoin, fermer la fenêtre ‘home’

5- Sélectionnez dans la fenêtre ‘project explorer’, le dossier du projet à compiler.
6- Cliquez sur le marteau pour « build Debug »


7- Branchez la carte STM32F4DISCOVERY au PC, avec un câble USB mini B.
8- Pour le premier lancement, il faut aller dans :

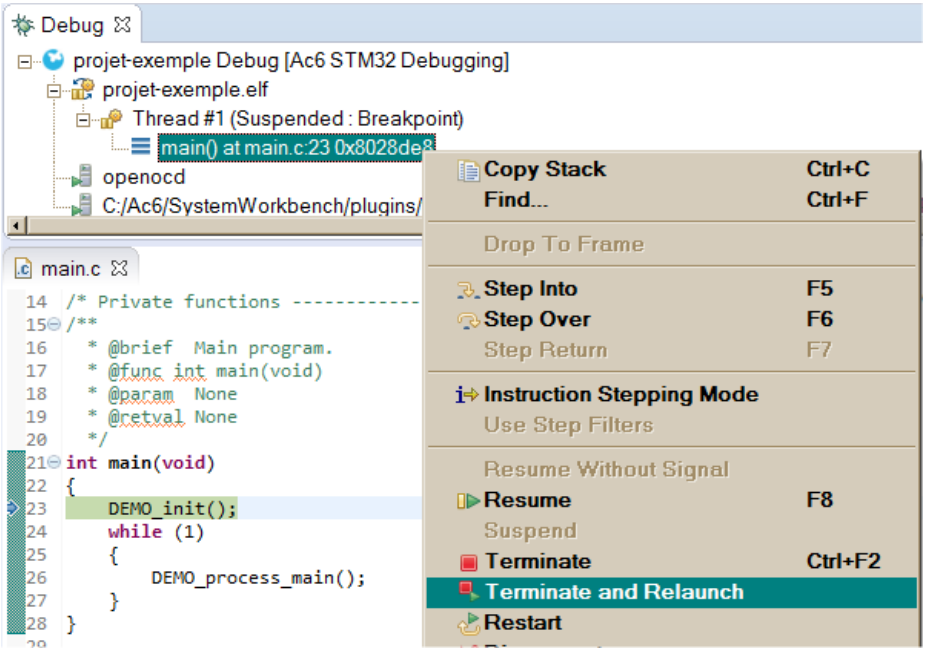
- Run -> Debug Configuration ->
- Dans l’arborescence, développez ‘Ac6 STM32 Debugging’, choisissez ‘ projet-base Debug’
- Debug
- Les prochains lancements peuvent se faire avec l’icône Debug : insecte vert


9- Lancez le programme : bouton Resume 

10- Vous pouvez ajouter la fenêtre ‘project explorer’ dans la vue Debug :
windows –> show view -> general -> project explorer

Pour exécuter **à nouveau** votre programme, n’oubliez pas de passer par chacune de ces étapes :

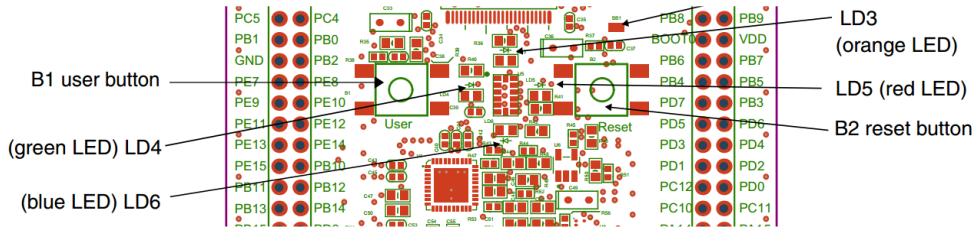
1- Enregistrer les fichiers sources — pas de ‘*’ dans les onglets, avant le nom des fichiers modifiés
2- Build — vérifiez bien qu’il n’y a pas eu d’erreurs !
3- Terminate and relaunch — dans la vue Debug, clic droit sur la session de Debug



## 3- Let's go

Votre mission :

> **Réaliser un détecteur de gravité.**
>
> ➔ Chacune des 4 leds indiquera **le sol** en s'allumant selon l'orientation de la carte.
> ➔ Lorsque la carte est placée à plat, les 4 leds s'éteignent.
> ➔ Lorsque la carte est en chute libre, les 4 leds s'allument




Méthode de travail : [1]

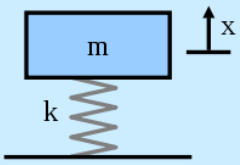
> En bon ingénieur, nous appliquerons la méthode dite du « **TTDM** »[1]
>
> **Cependant, nous travaillons en équipe :**
>
> ➔ Toute question pertinente pourra être répondue au tableau, pour que tout le monde en profite.

[1] TTDM - cet acronyme peut également être décliné dans une version allemande francisée originale, comique mais néammoins parlante : la méthode « DMSS » ; *DeMerden Sie Sich*.

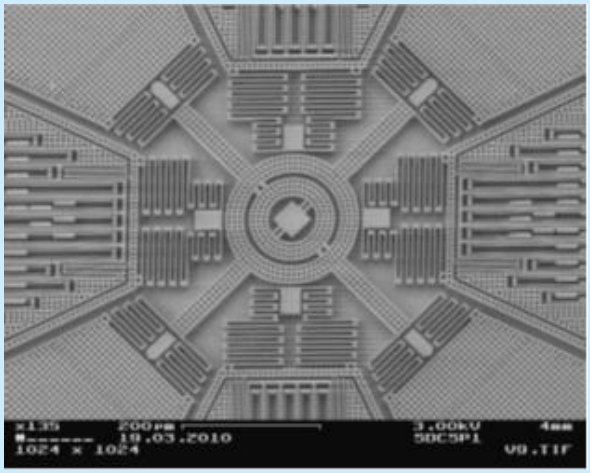
Nous n'entrerons pas dans le détail du fonctionnement de l'accéléromètre, il est néanmoins utile de disposer de quelques connaissances :

Un accéléromètre est un capteur qui mesure l'accélération linéaire, selon un ou plusieurs axes.

Pour comprendre le fonctionnement de l'accéléromètre, on peut imaginer une masse en suspension avec des élastiques. Au 'repos', le poids de la masse l'attire vers le sol (ou plus exactement le centre de gravité de la Terre). En mesurant la dimension des élastiques, on peut en déduire la direction de la gravité. Lorsque la masse est soumise à une autre force (une voiture qui freine, un choc…), ses mouvements entre les élastiques traduiront l'accélération qu'elle subit.




Représentation selon un axe.



Physiquement, dans le composant…

Le projet de départ qui vous est fourni contient un programme dont le comportement consiste à allumer une led verte en permanence, et une led bleu lorsqu'on appui sur le bouton bleu.

Selon le modèle de cartes (version B ou C), l'accéléromètre LIS302DL ou LIS3DSH est relié au microcontrôleur par l'intermédiaire d'un bus SPI. Une librairie logicielle déjà prête nous apporte des fonctionnalités opérationelles concernant la récupération des données mesurées par l'accéléromètre. Pour en savoir plus, nous vous conseillons la lecture des commentaires associés aux fonctions :

**uint8_t BSP_ACCELERO_Init(void)**

**void BSP_ACCELERO_GetXYZ(int16_t *pDataXYZ)**

Ce que vous devez faire :

- Faites fonctionner le programme de départ sur la carte.
- Localisez la fonction main, et lisez son contenu.
- Localisez l'endroit où les leds verte et bleu sont pilotées.
- Avec l'aide de la documentation appropriée, localisez sur quels ports du microcontrôleur sont reliées les LEDs sur la carte STM32F4DISCOVERY.
- Ajoutez dans l'initialisation les appels de fonction permettant de configuer les deux autres leds comme sorties numériques.
- Localisez les fonctions BSP_ACCELERO_Init() et BSP_ACCELERO_GetXYZ(), et expliquez comment on peut les utiliser
- Répondez au cahier des charges…
- Critiquez le cahier des charges