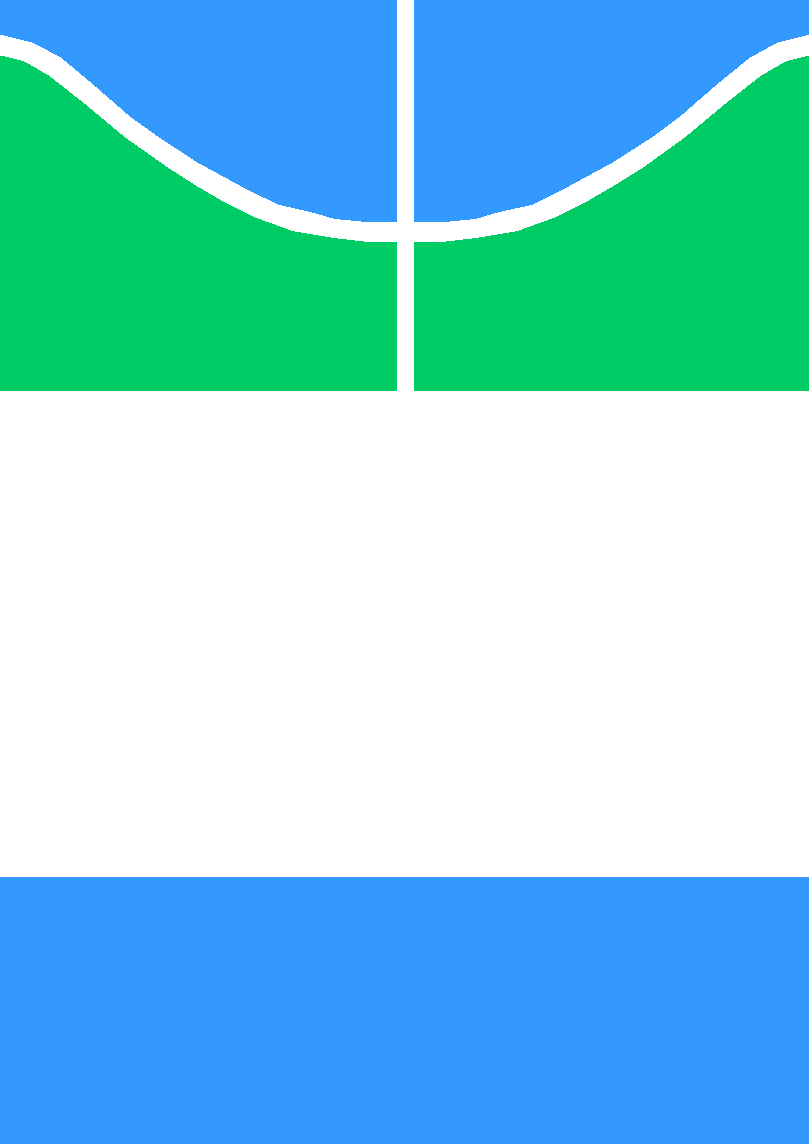
****

Développent d’une plateforme de programmation et utilisation de FPGAs à distance.

Edson Trigueros Cardenas

**Rapport de stage de 2ème année**

Stage Assistant ingénieur 2ème année

Pays : Brésil

Lieu : Laboratoire de Projet de Circuits Intégrées (LPCI) au Département d’Ingénierie Electrique (ENE) de Faculté de Technologie (FT) de Université de Brasilia (UnB)

Durée : 18/04/2018 - 18/07/2018

Remerciements

Je remercie Daniel Café pour m’avoir guidé dans cette expérience à l’international.

Table des matières

[L’université et laboratoire d’accueil 6](#_Toc523753814)

[Histoire de l’université 6](#_Toc523753815)

[Organigramme de l’université 7](#_Toc523753816)

[La faculté de Technologie 8](#_Toc523753817)

[Le laboratoire LPCI 9](#_Toc523753818)

[Définition du projet 9](#_Toc523753819)

[Mise en situation 9](#_Toc523753820)

[Matériel nécessaire pour le projet : 11](#_Toc523753821)

[Développement du projet 13](#_Toc523753822)

[Choix du matériel 13](#_Toc523753823)

[Étude de la carte MSP430F5529 13](#_Toc523753824)

[Comment s’en servir du USB Developper Package? 14](#_Toc523753825)

[Quelle est la signification de CDC, HID et MSC ? 15](#_Toc523753826)

[Configuration du MSP430 Descriptor Tool 16](#_Toc523753827)

[Choisir une configuration d’horloge : 16](#_Toc523753828)

[Les exemples disponibles sur le USB Developper Package : 18](#_Toc523753829)

[Qu’est ce que cet exemple m’a appris? 18](#_Toc523753830)

[Programmation de la carte sur code composer studio 19](#_Toc523753831)

[Comment j’ai réussi pour programmer la carte ? 19](#_Toc523753832)

[Développement de l’interface graphique 22](#_Toc523753833)

[Sélection de la technologie 22](#_Toc523753834)

[L’application 22](#_Toc523753835)

[Problèmes rencontrés et conclusion 24](#_Toc523753836)

[Problèmes 24](#_Toc523753837)

[Conclusion 25](#_Toc523753838)

[Annexes 26](#_Toc523753839)

[Pin Diagram 26](#_Toc523753840)

[USB Power System 27](#_Toc523753841)

[Modes de fonctionnement 27](#_Toc523753842)

[Horloges 28](#_Toc523753843)

[Code Implémenté 30](#_Toc523753844)

[Code de l’interface graphique 33](#_Toc523753845)

[Références techniques 41](#_Toc523753846)

# L’université et laboratoire d’accueil

## Histoire de l’université

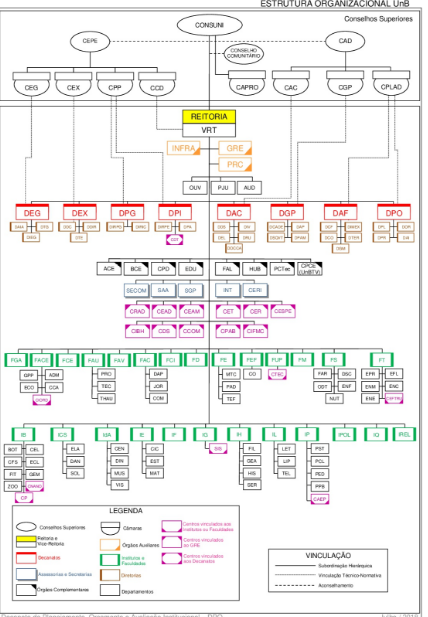
L’université de Brasília est le résultat du rêve et du travail des éducateurs comme Darcy Ribeiro et Anisio Teixeira, depuis 1962, l'année de sa création, une des références universitaires nationales principales. La diversité culturelle présente dans leurs quatre campus est une de leurs caractéristiques remarquables. La pluralité, alliée à la recherche permanente de solutions innovatrices, il déplace la production scientifique et le quotidien de l'institution.

L’université de Brasília a été inaugurée le 21 avril 1962 avec la promesse de réinventer l'enseignement supérieur, entrelacer plusieurs formes du savoir et former des professionnels engagés dans la transformation du pays.



## **Organigramme de l’université**

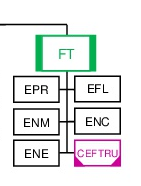
Voici un organigramme de l’université de Brasília :



On observe les différents départements qu’existent. On observe la direction générale en jaune “Reitoria” et les différentes facultés en vert. L’université étant assez grande, on se centrera seulement sur la Faculté de technologie que c’est là où se trouve le département d’ingénierie électrique (ENE).

## La faculté de Technologie

Maintenant la Faculté de Technologie (FT) est intégrée par quatre Départements : Ingénierie Électrique, Ingénierie Mécanique, Génie civile-environnementale et l'Ingénierie Forestière. L’université offre neuf formations diplômantes régulièrement (Ingénierie Environnemental, Génie Civil, l'Ingénierie Informatique, l'Ingénierie Électrique, Ingénierie Forestière, Ingénierie Mécanique, Ingénierie Mécatronique, Ingénierie de Production, le Génie chimique et l'Ingénierie des Réseaux de Communication). Aujourd'hui, avec 48 ans d'existence, la Faculté de Technologie possède 198 enseignants fortement qualifié (175 hommes et 23 femmes), dont 96 % sont contractés dans le régime dit Exclusif et 92 % possèdent le titre de doctorat ou supérieur. Elle compte avec 72 administratif-techniciens (59 hommes, 23 femmes, 43 techniciens et 29 administratif) et elle maintient 3216 étudiants régulièrement inscrits dans leurs formations diplômantes et 1102 étudiants dans les programmes de master “stricto sensu”.

*Organisation de la Faculté de Technologie et l’entrée principale*

Actuellement elle possède quatre amphithéâtres pour l'accomplissement de congrès, de cours et d'autres événements et environ 80 laboratoires de support à l'enseignement d'obtention de diplôme, de masters et à la recherche, y compris des installations équipées de matériel spécifique et des atmosphères d'utilisation partagées parmi des cours différents.

## Le laboratoire LPCI

Le laboratoire LPCI fait partie du Laboratoire de Dispositifs et Circuits Intégrés (LDCI). Ce laboratoire concentre les activités de recherche en matériaux semi-conducteurs inorganiques et organiques, circuits et dispositifs analogiques et numériques en y incluant des systèmes embarqués. Mon travail a été développé avec l'équipe des systèmes embarqués sous l'encadrement du Prof. Daniel Café.

Un poste de travail a été mis à disposition pour toute la période du stage. J’ai reçu un compte utilisateur pour pouvoir travailler au laboratoire. On m’a aussi donné la liberté de travailler depuis chez moi si nécessaire.



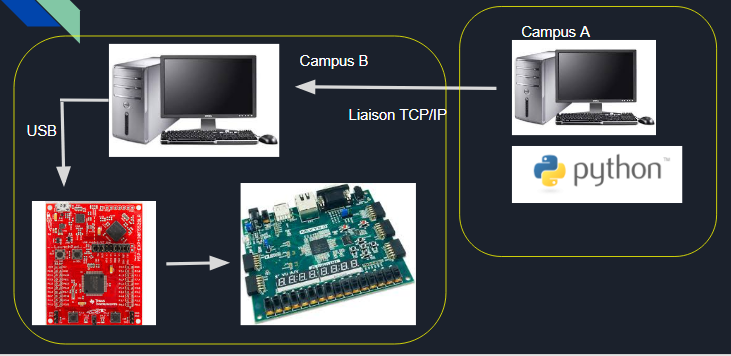
*Laboratoire LPCI*

# Définition du projet

## Mise en situation

Le laboratoire LPCI en partenariat avec le laboratoire GRACO (Groupe d'automation et Contrôle) possède une suite de FPGAs qui sont utilisés pour l'enseignement et la recherche. Ces FPGAs ne peuvent pas être déplacés car certaines sont déjà assemblés à des montages expérimentaux des laboratoires d'enseignement. Les chercheurs et étudiants des différents campus travaillent ensemble dans des différents projets avec ces FPGAs. Il est devenu encombrant de se déplacer des kilomètres de distance pour tester ou implémenter des circuits.

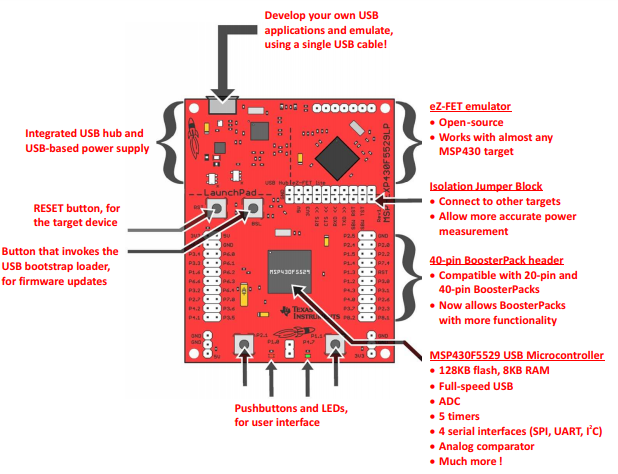
Ainsi, dans l'objective de réduire le temps de déplacement des étudiants et chercheurs, ce projet propose un framework de manipulation de FPGAs à distance. Le système développé doit permettre que les étudiants puissent interagir avec l'FPGA. Les PC des différents campus devront être lié par une liaison TCP/IP. Puis l’ordinateur recevant les données devra être connecté à un microcontrôleur qui lui va être connectée au FPGA en utilisation. Le projet était un projet dédié pour deux personnes, une personne ayant des connaissances avec la création de sites web et la liaison TCP/IP et une autre personne ayant de l’expérience avec des microcontrôleurs et FPGA. Cependant l’autre élève a été indisponible et donc vu que je n’ai pas de connaissances en création de sites web, j’ai proposé à mon encadrant de réaliser une interface sur Java pour simuler le projet vu que j’en ai appris un peu à l’ENSEA.



*Diagramme conceptuel du système*

## Matériel nécessaire pour le projet :

* Carte MSP430F5529



Cette carte possède alors

* Fonctionnement à une fréquence de jusqu’à 25MHz pour l’horloge du système.
* 4 Interfaces sérielles (SPI, UART, I2C)
* 5 timers
* Un module USB intégré avec des outils complémentaire, librairies exemples et des références pour se guider lors du développent d’une application USB.
* Un émulateur intégré eZ-FET avec une application UART (simpleBackChannel)
* Capacité pour simuler et développer des applications USB avec un simple câble USB
* Alimenté depuis l’hôte USB. Le bus de puissance de 5V est réduit à 3.3 V en utilisant un convertisseur DC/DC.
* Compatible avec le LaunchPad BoosterPack de 40 pins.

Logiciels

* Code composer studio pour coder le programme de la carte MSP430F5529. Ce logiciel possède un debugger, et on peut utiliser des fonctions faites par le fournisseur pour coder plus facilement les cartes.
* MSP Descriptors Tool. Cet outil nous a permis de créer quelques fichiers dont on a besoin pour la création du module USB.
* Python et NetBeans de Java. Il est important de notifier qu’au début j’ai commencé par créer une interface pour établir une liaison sérielle entre l’ordinateur et le microcontrôleur à l’aide du NetBeans de Java mais j’ai décidé d’utiliser plutôt le logiciel Python tout simplement puisque j’ai trouvé plus d’information sur internet.

# Développement du projet

Programmation sur Code Composer Studio

## Choix du matériel

La première étape a consisté en choisir le matériel que j‘allais utiliser. Premièrement j’ai parlé avec le professeur encadrant pour lui dire qu’à l’ENSEA on travaille avec la STM32. La programmation pouvait être plus facile et que le projet était faisable avec les caractéristiques quelle possède et en plus j’en avais une avec moi donc il n’y avait pas besoin d’en acheter. Cependant, le professeur m’a proposé de travailler avec une carte MSP430F5529 de la compagnie Texas Instruments car il s’y connaissait mieux donc il pouvait être capable de m’aider en cas de blocage, en plus cette carte est idéale pour ce type de projets et aussi l’école pouvait m’en prêter une pour faire le projet. Donc j’ai commencé par faire mes recherches sur la carte et essayer de trouver une méthode pour réaliser la liaison entre les différents composants.

## Étude de la carte MSP430F5529

J’ai commencé par étudier la datasheet de la carte ainsi que le guide d’utilisateur pour savoir ce qu’elle possédait et étudier les différents registres de la carte.

Le producteur de la carte MSP430F5529 fournit un document nommé **“Starting a USB Design Using MSP430”** qui est dédié pour les clients qui désirent créer une liaison sérielle USB avec la MSP430. On utilise le module USB inclus dans cette carte car c’est plus simple et contient des caractéristiques uniques que l’utilisateur ne peut faire avec une liaison UART, SPI ou I2C.

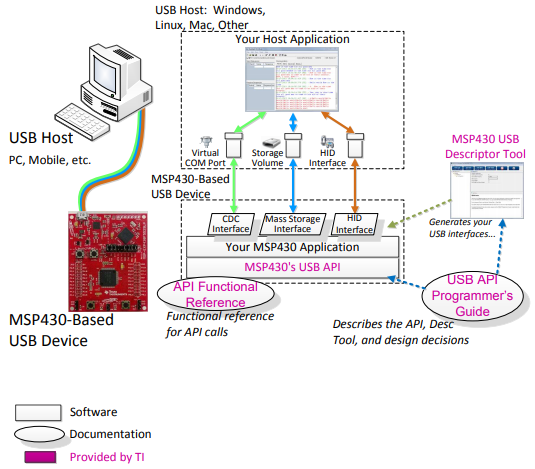
La compagnie TI a développé un **“USB Developers Package”** qui fournit une solution plus simple et accessible pour tout type d’utilisateurs. Le **“USB Developpers Package”** contient un **“USB Descriptors tool”** qui est un outil qui permet de générer un code selon l’usage qu’on va lui donner à notre application, on expliquera le but de cet outil un peu plus tard. Ce **“USB Developpers Package”** contient aussi de l’aide pour guider l’utilisateur selon les besoins qu’il a, comme des exemples déjà faits, de la documentation sur la programmation, etc. Dans le site de Texas Instrument on peut trouver beaucoup d’information et des forums aussi pour guider les utilisateurs.

Voici quelques caractéristiques importantes du module USB qui possède la carte MSP430F5529

* Vitesse du composant USB : 12 Mbps qui facilitent la communication avec l’hôte USB.
* Commande, interruptions et transferts de masse.
* 8 input et 8 output “endopoints”: Plus de endpoints possède le système, plus de interfaces USB peuvent être implémentées dans le composant. Ce nombre de endpoints est suffisant pour notre application.
* Un 3.3 V LDO pour fonctionner directement avec les 5V VBUS de l’hôte.
* Un D+ pullup intégré: Ce pullup permet au système de l’USB de communiquer avec l’hôte pour informer qu’il est prêt à être utilisé.
* Une PLL programmable: Une PLL qui génère les 48-MHz nécessaires pour le fonctionnement de l’USB.

## Comment s’en servir du USB Developper Package?

L’**USB Developpers Package** est un paquet qui, comme son nom l’indique, contient les logiciels fournis par TI pour tester nos programmes comme par exemple le Java HID Demo App. Il contient aussi de la documentation comme le : **MSP430 USB API Programmers** guide qui contient tous les registres et fonctions disponibles qui vont nous servir pour développer le projet. En plus, il contient aussi des exemples lorsque l’interface est de type CDC (Communications Device Class), des exemples lorsque l’interface est de type HID (Human Interface Device Class) et aussi des exemples lorsque l’interface est MSC (Mass Storage Device) ou aussi composite (c’est à dire un mélange des classes qu’on a cité ci-dessus). Finalement il contient le **USB Descriptors tool** qui va nous permettre de développer nos interfaces CDC, HID, MSC ou composites on expliquera un peu plus tard le principe de chacune de ces différents dispositifs.



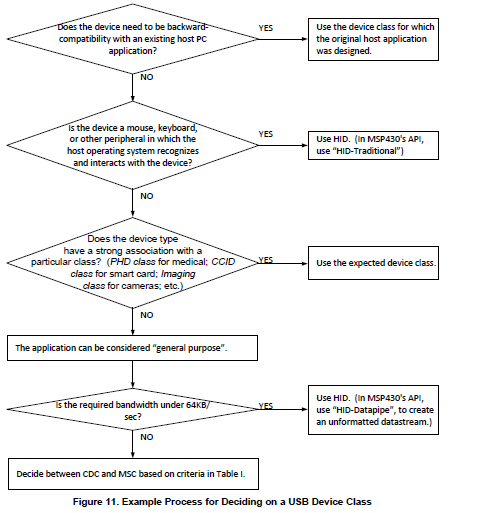
Le Descriptors Tool génère l’interface USB dont on a besoin. On écrit après le code de la MSP430 pour qu’elle communique avec l’hôte à travers ces interfaces. Texas Instruments a développé des exemples pour que les utilisateurs puissent comprendre le fonctionnement plus rapidement. Les interfaces CDC et MSC sont très communes donc il y a beaucoup d’aide sur internet. Cependant pour le HID est un peu plus difficile c’est pourquoi TI a développé une application en Java pour mieux guider l’utilisateur.

## Quelle est la signification de CDC, HID et MSC ?

Dans cette partie on va expliquer les différentes classes CDC, HID et MSC

* **Communications Device Class (CDC)** : Aboutit à un port COM virtuel sur l'hôte.
* **Human Interface Device Class (HID)** : Grâce à cette configuration on peut choisir entre 4 sous configurations différentes tels que : Datapipe (Qui a un comportement similaire à celui d’un CDC), Souris, Clavier ou Custom.
* **Mass Storage Class :** Lorsque l’hôte détecte une configuration MSC, il commence à monter un stockage d’information.

Voici l’organigramme qui m’a aidé pour choisir le type de l’application



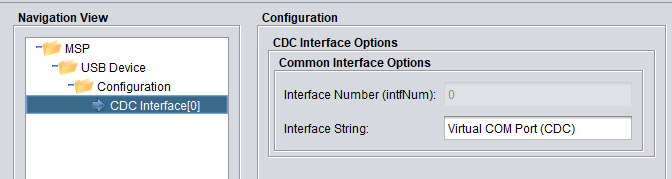
Donc il suffit de créer une application du type CDC à l’aide du **Descriptor Tool** et puis envoyer des données à l’aide des fonctions déjà existantes sur le programme code composer. On va se focaliser sur expliquer comment j’ai crée l’interface CDC, on aura pas le temps d’expliquer les autres interfaces mais je vais expliquer dans la section **“Annexes”** les difficultés que j’ai rencontré pour choisir le bon type d’application dont j’avais besoin.

## Configuration du MSP430 Descriptor Tool

Il faut remarquer que les fichiers de description sont nécessaires pour l’utilisation du module USB puisqu’il nous aide à **identifier** le composant qu’on est en train d’utiliser ainsi que paramétrer l’horloge de fonctionnement.

La première partie que j’ai commencé à faire est la création des fichiers “descriptors” à l’aide du **MSP430 Descriptor Tool.**

Voici la configuration que j’ai fait :



Champs descriptifs Associés : bInterfaceNumber cette valeur il faut la passer comme paramètre "intfNum" dans des appels d'API CDC pour pouvoir choisir cette interface. L’outil produit aussi des constantes dans descriptors.h reflétant cette valeur ; par exemple, CDC0\_INTFNUM. Donc on crée une seule interface de type CDC et on lui assigne un identifiant par défaut puisqu’il lui en faut une identification pour être détecté par le dispositif. Cet outil comme son nom l’indique il génère les fichiers “descriptors" de manière rapide et automatique. C’est un outil très efficace et utile puisque coder les “descriptors” peut être compliqué et on peut faire beaucoup de fautes donc ça peut nous prendre beaucoup de temps.

Donc une fois les fichiers descriptors.c et discriptors.h crées on peut les mettre dans notre projet pour construire notre application CDC.

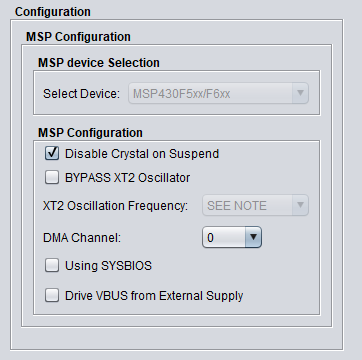
## Choisir une configuration d’horloge :

Le module USB de la MSP430 utilise une horloge de 48-MHz qui est généré par une PLL. Cette PLL a besoin d’une horloge de référence, la source de cette horloge est l’oscillateur XT2 qui est un résonateur céramique de 4-MHz avec une précision de +-2500-ppm. Le premier pas à faire est de choisir les paramètres corrects pour avoir la meilleure précision possible. L’utilisateur a trois choix pour fixer la source de la PLL :

* Utiliser une source externe (met l’oscillateur XT2 en mode “bypass”) avec une gamme d’entre 1.5 et 35 MHz. On l’utilise lorsqu’il y a une horloge toujours présente pendant la communication USB car on économise le coût d’un cristal.
* Utiliser un cristal dans une gamme de fréquence comprise entre 4 et 32 MHz, le cristal a une excellente précision.
* Utiliser un résonateur céramique dans une gamme de fréquence comprise entre 4 et 32 MHz. Cette méthode est moins chère que celle avec un cristal.

Or, on doit bien choisir cette source afin de ne pas bloquer la PLL et aussi à fin que les fréquences soient compatibles avec celles de la PLL. Choisir un cristal serait un peu plus compliqué à mettre en place car il faudra choisir des condensateurs de charge selon les spécifications du cristal à cause d’une possible mise hors tension de l’oscillateur pendant l’utilisation du module USB et donc un dédommagement des composants. Finalement, il est important d’informer que on peut réaliser cette configuration avec le MSP430 Descriptors Tool.

Voici la configuration que j’ai fait avec le USB Descriptors Tool :

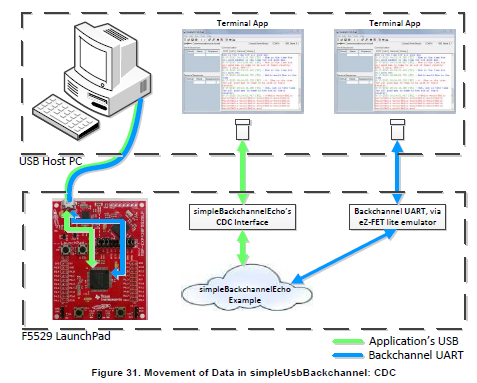


Dans cette configuration l’API fait marcher l’oscillateur XT2 (4MHz) lorsque l’USB est actif et de manière optionnelle arrête son fonctionnement lorsque l’USB n’est pas utilisé.

## Les exemples disponibles sur le USB Developper Package :

Le premier objectif que je me suis fixé consistait en créer un programme simple pour faire communiquer la carte MSP430F5529 avec une application sur l’ordinateur simple. Par exemple avec Hyperterminal ou Putty, il suffit juste d’ouvrir un port et de communiquer avec le dispositif connecté au même port. Pour faire cela il faut identifier le COM port qui est connecté sur la carte à l’aide du “Device Manager” de l’ ordinateur, dans notre cas la carte a été connecté au port COM 10.

Je me suis guidé avec l’exemple simpleUsbBackchannel qui est détaillé dans le document nommé <<MSP430F5529 LaunchPad Development Kit>> qui est un programme qui utilise une communication de type CDC . Cet exemple reçoit de l’information sur le UART backchannel et la renvoie vers l’ordinateur à travers le USB. Donc ce qu’il faut remarquer c’est qu’il y a un trafic de données à travers deux voies différentes (UART et USB). Voici une image qui explique les flux des données :



## **Qu’est ce que cet exemple m’a appris?**

Cet exemple m’a beaucoup aidé à savoir comment faire la connexion entre le microprocesseur et l’ordinateur, et à savoir comment utiliser une terminale à fin d’envoyer des données, dans ce document on nous informe où se situent les fonctions déjà faites qu’on peut utiliser pour faciliter la programmation. Par exemple il existent des fonctions de type <<void bcUartSend(uint8\_t \*buf,uint8\_t len)>> qui envoie len bytes qui sont stockés dans buf à travers le UART.

Ce document m’a aidé à configurer les horloges :

Il existe l’horloge rapide (MCLK). Les horloges lentes sont celles qui consomment le moins de puissance donc plus l’horloge rapide est inhabilité moins de puissance on consomme. C’est un critère qu’il faut prendre en compte.

Il existent 3 horloges dites lentes disponibles dans la carte F5529:

* REFO: Qui travaille à 32kHz et n’a pas besoin de cristal.
* LFXT1: Est un oscillateur de cristal qui est très précis et a besoin de moins de puissance que le REFO. Il travaille aussi à 32kHz.
* VLO: Il s’agit d’un oscillateur qui travaille entre 12kHz et 20KHz.

L’exemple utilise l’horloge rapide, le CPU se maintiens actif tout le temps. Dans annexe je adjoints un document qui nous parle clairement des horloges présentes dans le système.

Je me suis rendu compte aussi qu’il existe une fonction nommée <<initPorts()>> qui nous facilite la configuration des ports comme entrées ou sorties. J’ai appris aussi qu’il existe une fonction nommée USB\_setup() qui doit être appelée pour initialiser l’API de l’USB et permet le fonctionnement des événements de l’USB. Cette fonction nous permet de vérifier si l’hôte USB est bien connecté, cette information est connue grâce à la présence des 5V sur le pin du VBUS. Donc si cette vérification est faite, le signal D+ est activé pour informer que l’hôte est présent et les interruptions sont activées aussi. Finalement, j’ai appris aussi à coder un peu avec Code Composer car il est assez différent par rapport à Keil UVision lorsqu’on programmait la carte STM32.

## Programmation de la carte sur code composer studio

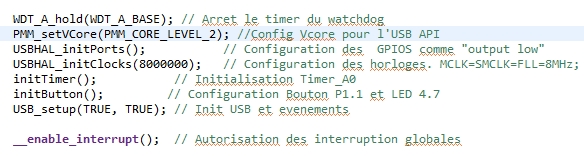
J’ai commencé par créer un nouveau projet sur Code Composer Studio, et puis à recopier quelques librairies dont j’allais avoir besoin. Je me suis guidé bien sûr de l’exemple précédent et du document <<MSP430F5529 LaunchPad Development Kit>>. J’ai copié aussi quelques fichiers propres à l’utilisation du module USB et remplacer les fichiers “descriptors.c” et “descriptors.h” pour ceux que j’avais généré à l’aide du **Descriptor Tool.**

J’ai vu un exemple sur la documentation de Texas Instruments où on pouvait contrôler les composants de la carte MSP430 en envoyant des chaînes de caractères sous forme de commandes. Donc je me suis guidé de cet exemple car on pouvait allumer ou éteindre une LED juste en envoyant une chaîne de caractères et je me suis dit que ça pourrait m’aider pour dialoguer avec l’ordinateur.

## Comment j’ai réussi pour programmer la carte ?

Ce programme fait intervenir un Timer et un bouton poussoir de la carte. Le timer va être utile pour faire clignoter une LED et interagir avec l’application faite à l’aide de Python.Tout d’abord dans notre boucle “main” de notre fichier principal, on initialise les Ports et les horloges à l’aide des fonctions <<USBHAL\_initPorts()>> et <<USBHAL\_initClocks(8000000)>>, on initialise le Timer\_A0 avec une interruption, on initialise aussi le pin P4.7 comme sortie qui correspond à la LED rouge de la carte et le pin P1.1 comme entrée qui correspond au bouton poussoir avec une interruption (On a programme ce bouton poussoir pour montrer qu’on pouvait envoyer des commandes aussi de la carte MSP430 vers l’ordinateur). On utilise aussi la fonction <<USB\_setup(TRUE,TRUE)>> qui initialise les événements et le module USB lorsqu’un dispositif USB est connecté. On autorise les interruption avec la fonction “\_\_enable\_interrupt();

Voici le code de cette première partie qu’on vient de décrire :



Après avoir code cette partie, j’ai commencé à coder une boucle infinie définie par un “while(1)”.Dans la boucle infinie on utilise des fonctions propres au module de l’USB qui vérifient la connexion de la carte, on utilise des boucles “if” pour que la carte réagisse selon les différentes commandes qu’on lui envoie. Les commandes se font à travers d’une chaîne de caractères, une fois la carte MSP430 connecté à un port de l’ordinateur, on peut envoyer des commandes à travers d’une terminale en écrivant les chaînes de caractères **existantes** dans le programme. Par exemple lorsqu’on écrit dans la terminale “Allumer LED” on allume la LED connecté au pin P1.0. Il existent pour le moment 4 commandes existantes “Allumer LED”, “Eteindre LED”, “Clignote lentement”,“Clignote rapidement”. Le programme nous permet de renvoyer vers l’ordinateur aussi une chaîne de caractères on a donc une communication dans les deux sens. Par exemple dans la terminale lorsqu’on écrit “Eteindre LED”, on reçoit automatiquement comme réponse “ la LED est eteinte”. La complexité du code pour envoyer des données vers la carte est simplifié par l’utilisation des fonctions <<USBCDC\_sendDataInBackground()>> et <<USBCDC\_receiveDataInBuffer()>>. On utilise une fonction nommée <<retInString()>> qui est une fonction qui envoie “TRUE” s’il y a un symbole “0x0D” dans la chaîne de caractères envoyée. Or dans la table ASCII “0X0D” correspond à la touche “carriage return” pour informer qu’on à un retour à la ligne. Ainsi lorsqu’on a fini d’écrire un mot et on rentre la touche “Enter” on envoie une commande directement.

Il faut remarquer dans un denier temps que la manière pour configurer un pin en sortie ou entrée ou établir une interruption est différente que celle qu’on a appris avec Keil uVision donc ça m’a pris un peu de temps.

**Je mettrais dans l’annexe des copies d’écran du code commenté mais ci-dessus j’explique les fonctions qui sont un peu plus compliquées à comprendre.**

# Développement de l’interface graphique

Programmation sur Python

## Sélection de la technologie

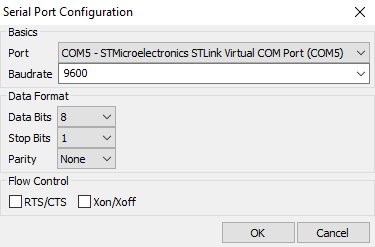
Comme je l’avais déjà dit avant, je voulais commencer par créer cette application sous Java mais j’ai bloqué et je ne savais pas comment résoudre ce problème donc j’ai décidé de coder avec Python. J’ai trouvé un peu plus d’aide sur internet et le langage était un peu plus facile pour moi. J’expliquerais dans la section “Problèmes rencontrés” le problème que j’ai eu avec le code Java.

Pour réussir à créer l’application je me suis guidé des exemples sur internet, de l’aide en ligne et je suis passé beaucoup de temps dessus. Je n’expliquerais pas toutes fonctions citées, j’expliquerais seulement comment cette application a pu me permettre d’interagir avec la carte MSP430F5529.

Tout d’abord j’ai commencé par me guider d’un code que j’ai trouvé sur internet où on nous montrait comment faire la liaison sérielle avec un port COM. Cet exemple m’a aidé pour apprendre à utiliser la librairie “serial” qui est la bibliothèque plus importante ici pour connecter la MSP430 au PC. Par exemple cette librairie nous permet d’écrire des chaînes de caractères à travers le port COM une fois celui-ci connecté. Donc ça a été parfaitement adapté car c’était ça dont on avait besoin pour communiquer avec la carte. Je me suis renseigné juste après sur la bibliothèque “wxPython”, c’est elle qui m’a permis de dessiner des boutons, d’afficher des images, d’ajouter des lignes de texte.

## L’application

Lorsqu’on exécute le programme la première fenêtre qui s’affiche est celle-ci :



Remarquons ici que je me suis connecté avec une carte STM32 car je n’avais plus avec moi la carte MSP430F5529

On choisit tout d’abord le port auquel on veut se connecter (Pour savoir à quel port on doit se connecter, on doit chercher dans le “Device Manager” la carte MSP430). Les autres paramètres normalement doivent rester intacts. Puis lorsqu’on clique sur OK. On nous envoie directement vers la page suivante :



J’ai mis le symbole de chacune des écoles partenaires juste pour savoir comment positionner les différents éléments mais aussi pour montrer qu’il s’agit d’un projet des deux écoles.

Donc l’idée c’était que lorsqu’on appuyait sur chacun des switchs (ci-dessus) de la BASYS3 on pouvait actionner leurs boutons réels et donc il fallait juste appliquer cette même méthode au FPGA du laboratoire LDCI pour que les élèves puissent les utiliser à distance. On obtiendrait de même les données envoyées par les cartes directement sur le cadre de texte (en blanc) montre ci-dessus.

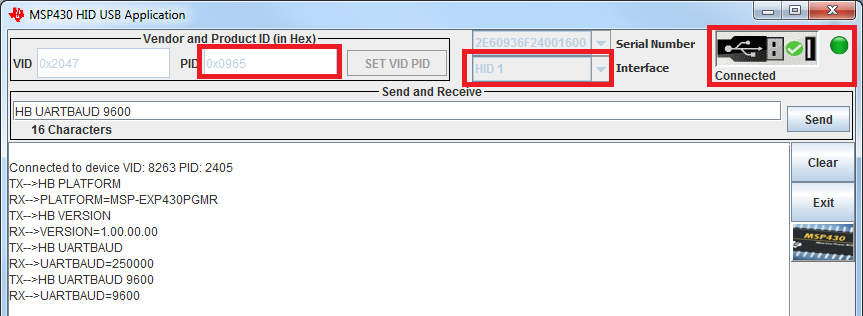
En général dans le code que je vous présente dans l’Annexe, je me focalise sur l’emplacement des éléments qui constituent l’application, sur la création des événements lorsqu’on appuie sur un bouton, sur la barre des menus où on peut modifier les paramètres de configuration, et sur la liaison sérielle (envoyer des commandes en sortie, recevoir des données dans le cadre de texte, etc).

# Problèmes rencontrés et conclusion

## Problèmes

Je vais faire une liste des principaux problèmes et difficultés rencontrés lors de la réalisation de ce projet.

a) La première difficulté que j’ai rencontrée a été comprendre les différents dispositifs qui existent pour une liaison USB. J’ai passé du temps à comprendre les différents dispositifs CDC, HID et MSC. Au début j’ai commencé par créer un dispositif HID (Human Interface Device) car j’ai vu beaucoup de tutoriels sur internet qui s’approchait à ce que je souhaitais réaliser. J’ai utilisé le Java HID demo qui est une application développée par TI pour tester les exemples dédiés à l’HID . Donc j’ai testé un exemple proposé qui marchait bien puisque je pouvais envoyer des commandes sous forme de texte. Voici une image de ce qu’on peut faire avec le Java HID demo:



Donc une fois la carte connecté on pouvait envoyer des commandes avec le bouton “Send”. Le problème a été que je devais faire ma propre application soit sur Java soit sur Python pour l’adapter à mon projet. Puis en voulant faire l’adaptation du programme sur Java avec un dispositif HID j’ai trouvé très peu d’information et apparemment le code était beaucoup plus compliqué donc en consultant mon professeur encadrant il m’a dit que faire un dispositif CDC était idéal pour ce type d’applications. J’ai compris que le HID on l’utilise plutôt lorsqu’on veut créer un dispositif comme un clavier ou une souris connecté au port USB.

b) Un deuxième problème que j’ai rencontré a été la programmation sur Java puisque une fois que je me suis lancé sur la création de l’application, j’ai eu des difficultés pour utiliser la librairie Java\_RxTx. En fait, je ne trouvais pas l’information dont j’avais besoin sur internet. Cette librairie est celle dont on a besoin pour établir une liaison sérielle donc j’ai essayé de me débrouiller avec mes faibles connaissances en Java que j’avais appris à l’école. Cependant après des longues recherches et beaucoup de temps perdu, j’ai décidé d’utiliser Python puisqu’un étudiant de l’université me l’a recommandé. Heureusement que j’ai trouvé beaucoup d’information sur le réseau et j’ai pu l’adapter à mon projet. Évidemment, j’ai dû passer du temps aussi car j’ai dû apprendre à utiliser quelques librairies et faire des tutoriels pour apprendre ce langage de programmation.

c) Le dernier inconvénient que j’ai eu est le manque de temps puisque normalement j’étais censé de lier la connexion avec un FPGA, cependant je n’ai pas eu le temps de faire des tests avec les FPGA du laboratoire vu qu’ils sont de forte puissance et ont besoin d’un analyse assez long . J’ai décidé de tester mon programme avec une basys3, mon but était de lancer un simple programme à l’aide de VHDL de la compagnie Xilinx et donc récupérer les données envoyées dans une zone de texte de l’application faite sur Python. Cependant je n’ai eu aucun résultat, en fait la programmation sur Python m’a pris plus de temps de ce que je pensais.

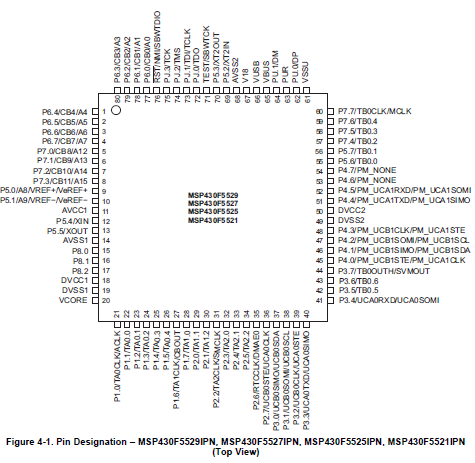
## Conclusion

En conclusion, j’ai beaucoup aimé de réaliser cette expérience à l’international puisque j’ai connu une nouvelle culture, des nouvelles personnes, une nouvelle ambiance académique et une nouvelle forme de travail. Le laboratoire était très calme et j’avais tout ce dont j’avais besoin pour réaliser le projet. Une autre chose que j’ai aimé est le fait que j’étais en totale autonomie donc je pouvais décider quels logiciels utiliser, de créer ma propre application de la façon que je voulais mais aussi j’ai eu des difficultés lorsque j’étais bloqué quelque part dans le projet et je ne savais pas comment le résoudre.

En plus le fait de réaliser quelque chose que je n’avais jamais fait (établir une liaison sérielle et dialoguer avec une application propre à moi) m’a beaucoup plu et j’aimerais dans un futur proche développer une application HID, MSC ou composite. J’ai aimé aussi ce stage puisque j’ai appris à programmer en Python, en code composer studio (CCS), j’ai utilisé un nouveau microprocesseur que même si je pensais que ça allait être similaire à la programmation de la carte STM32, j’ai découvert qu’il y a beaucoup de différences et finalement puisque j’ai appris une nouvelle langue, la langue portugaise.

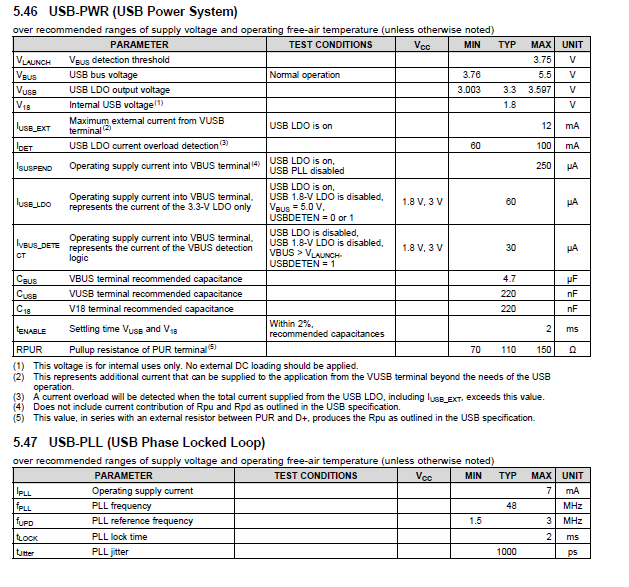
# Annexes

## Pin Diagram



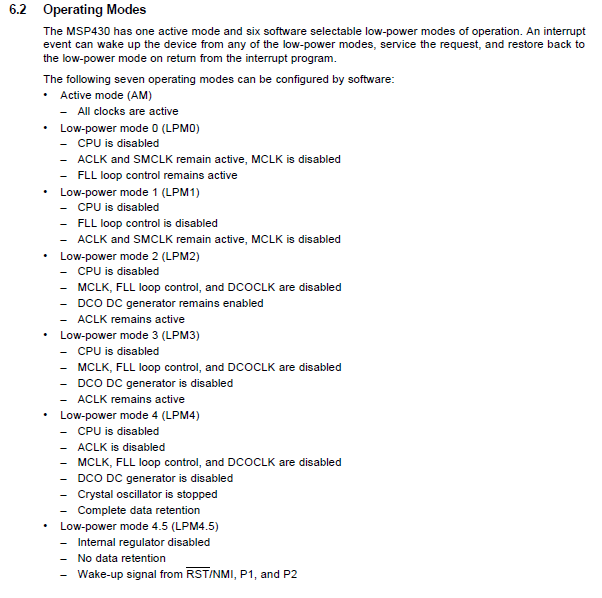
## USB Power System

On trouve ici la puissance nécessaire pour faire fonctionner le module USB et aussi les fréquences auxquelles la PLL fonctionne.



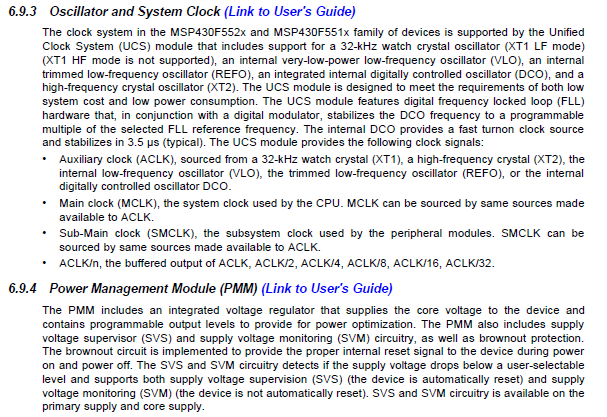
## Modes de fonctionnement

Dans quelques parties du programme je fais intervenir les modes de fonctionnement LPMx et je ne les explique très clairement. Voici le document qui les explique correctement :

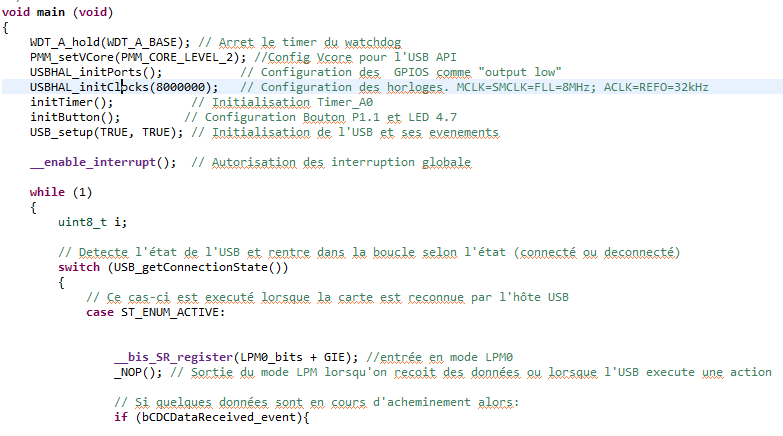


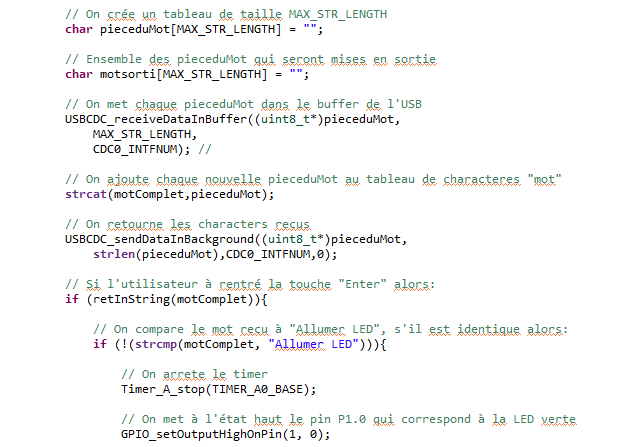
## Horloges

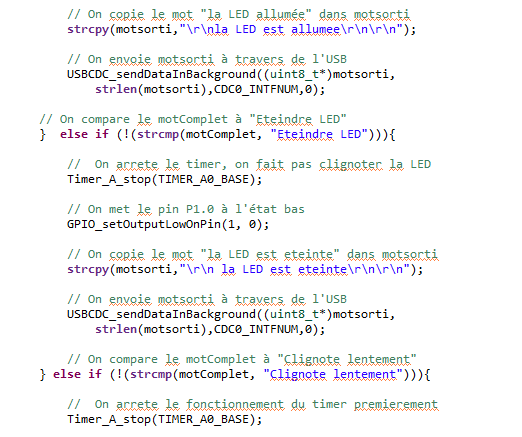
Voici un document qui explique les différentes horloges qu’existent :

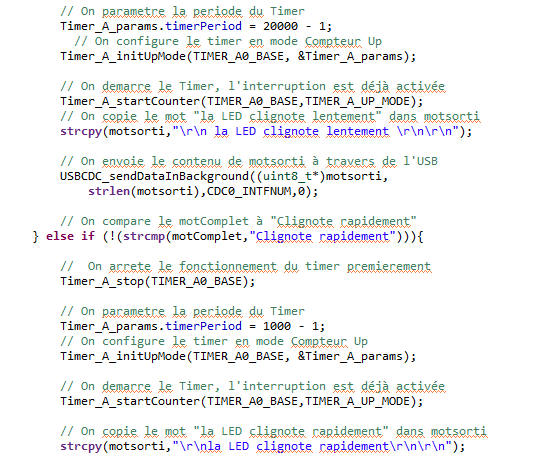


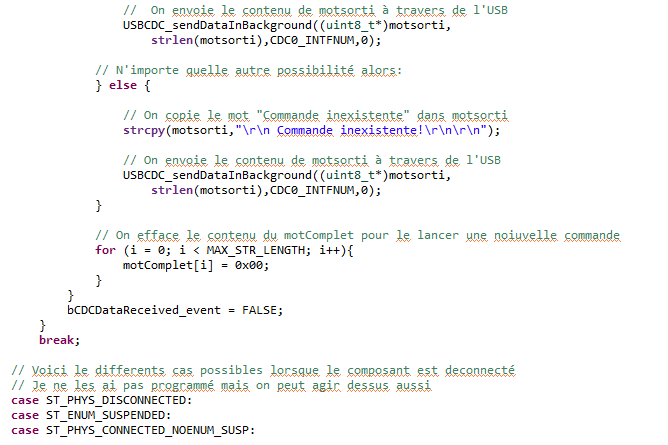
## Code Implémenté

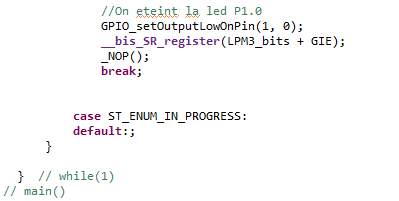




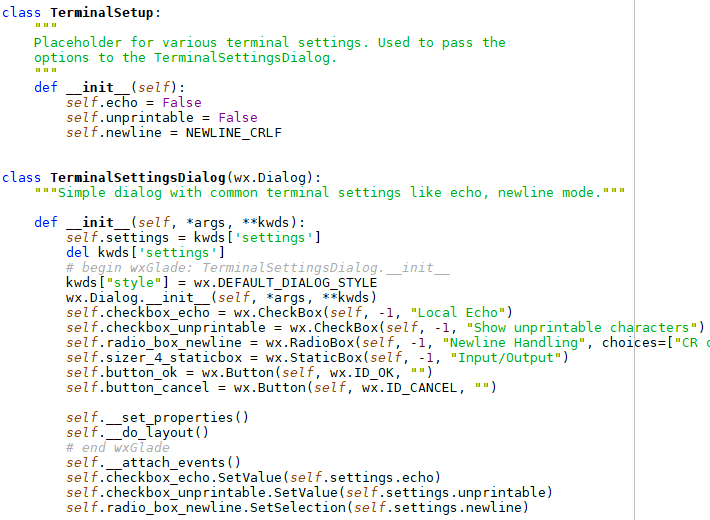


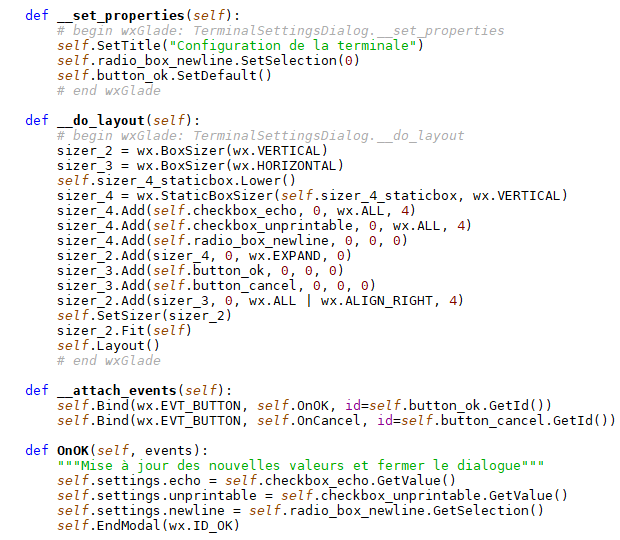


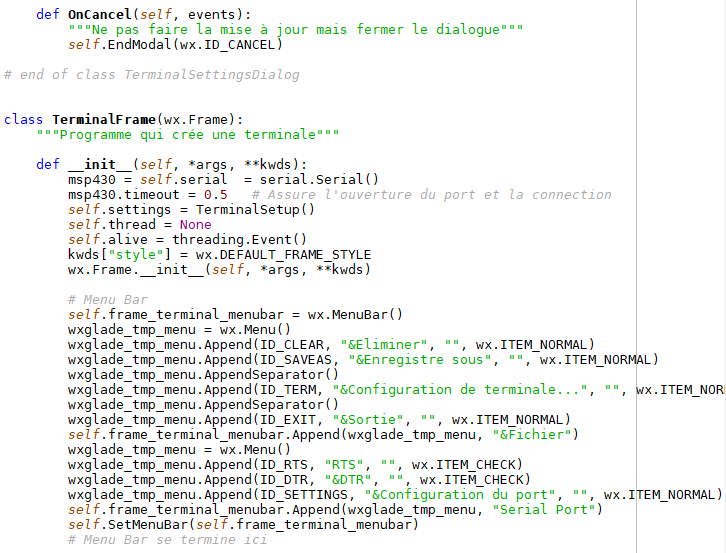


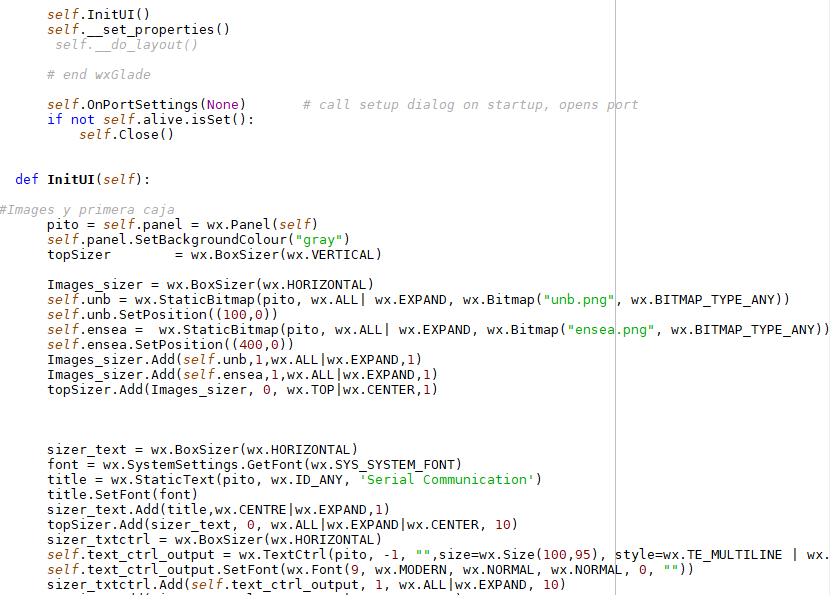


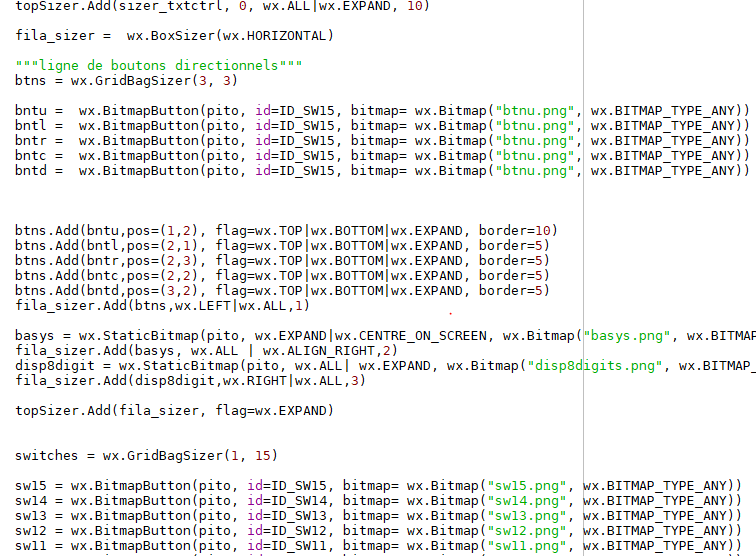
## Code de l’interface graphique

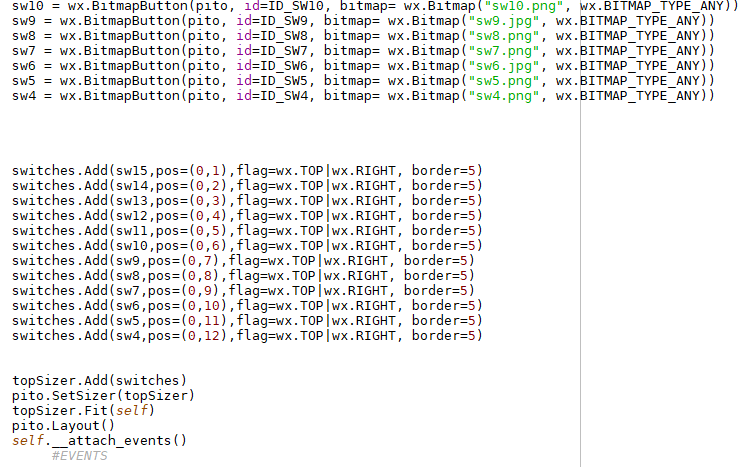




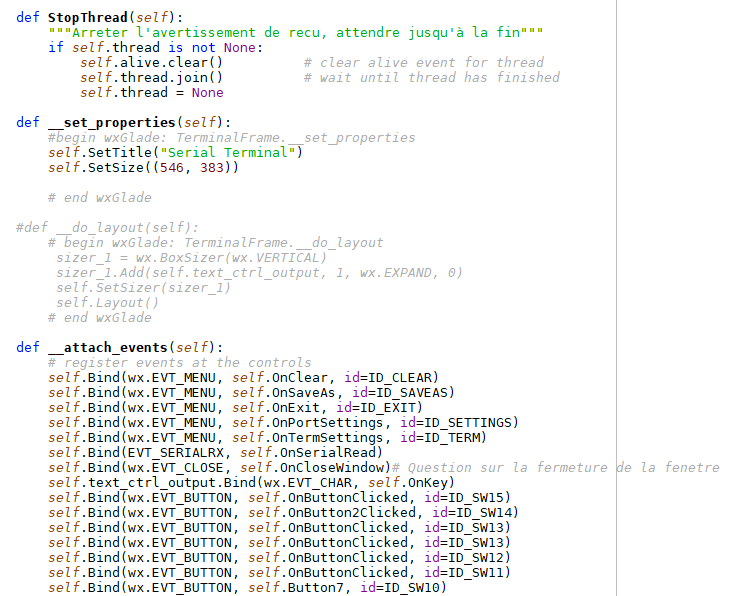


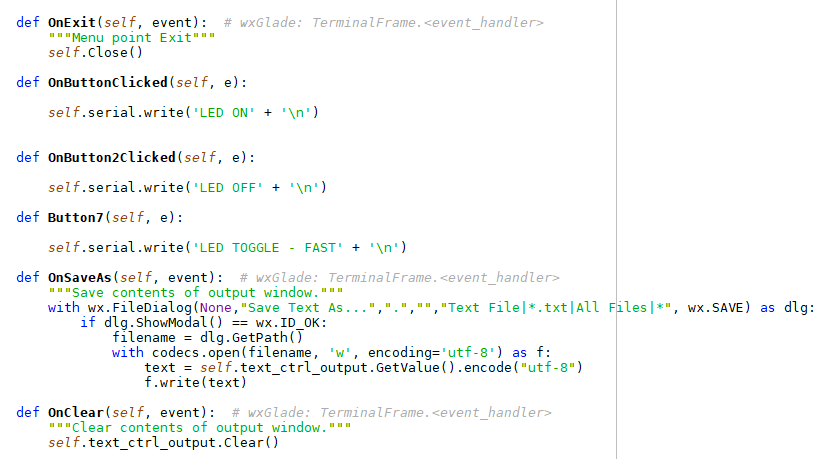


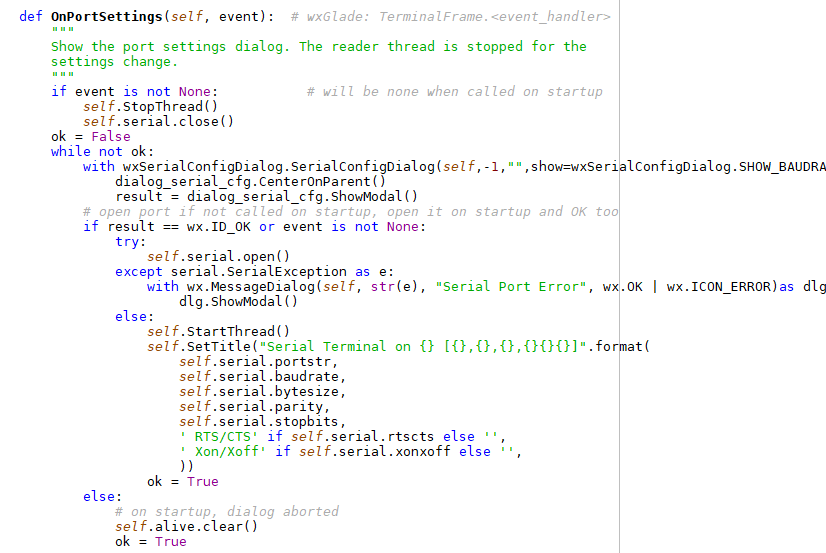


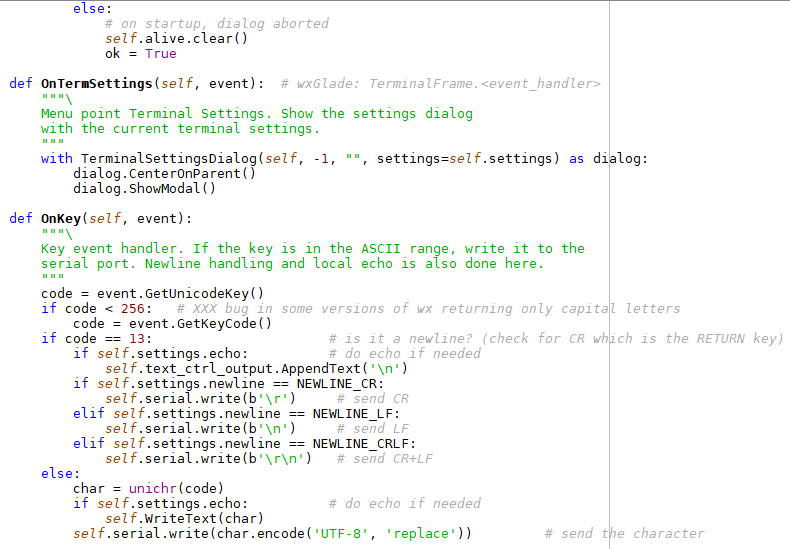


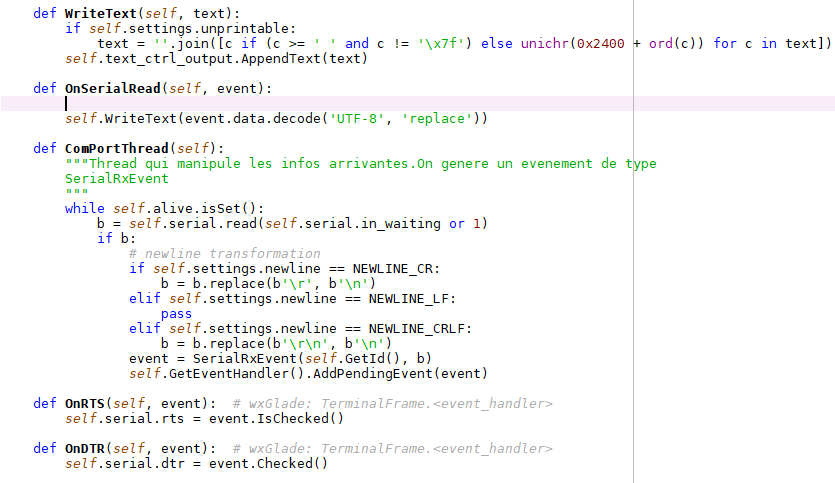


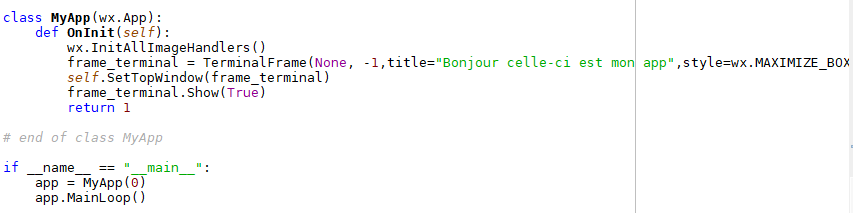












# Références techniques

1. Texas Instruments Incorporated, MSP430F552X and MSP430F551X Mixed-Signal Microcontrollers DATASHEET
2. Texas Instruments Incorporated, MSP430x5xx and MSP430x6xx Family User’s Guide, Revised March 2018
3. Texas Instruments Incorporated, Starting a USB Design Using MSP430 MCUs, Revised May 2014
4. Texas Instruments Incorporated, MSP430F5529 LaunchPad Development Kit (MSP-EXP430F5529LP), Revised April 2017
5. <http://www.ti.com/tool/MSP430USBDEVPACK> (Lien pour télécharger le USB Developpers Package)
6. <https://wiki.wxpython.org/>(Tutoriel pour apprendre à programmer en Python)
7. <https://gist.github.com/openp2pdesign/a027a9fc5aac66e6a382> (Exemple pour créer une interface sérielle sur Python)
8. <http://zetcode.com/wxpython/> (Tutoriel pour dessiner des boutons,etc)
9. <https://pythonhosted.org/pyserial/> (Informations sur la librairie “serial”)