Guide de démarrage et de développement pour la NxP Cup

Pour châssis Model-C et carte microcontrôleur FRDM KL25Z

T. Grandpierre (*t.grandpierre@esiee.fr*)

Remerciements

Nous tenons à remercier les membres de l'équipe Freescale NxP qui ont permis à nos élèves de participer à cette aventure formidable que représente ce challenge. En particulier Stiffan Flavio qui, avec sa bonne humeur permanente, s'est toujours démené pour trouver des solutions que ce soit pour nous aider à résoudre des problèmes techniques, logistiques ou organisationnels, et il y'en a eu !

Figure 1 - Finale EMEA 2013 à l'ESIEE

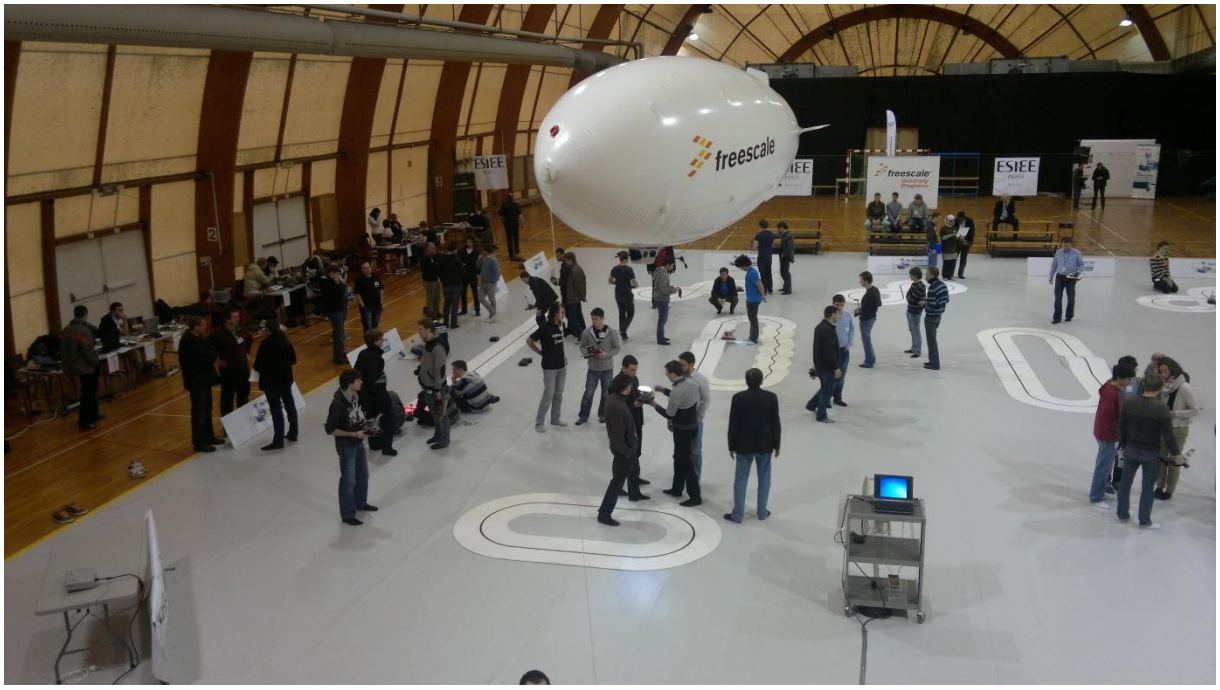


Figure 2 - Préparation de la finale EMEA 2013 ESIEE

Introduction

Présentation

La NxP Cup est une compétition mondiale qui commence par des sélections dans chaque continent. C'est une course de voiture miniatures autonomes sur un circuit qui n'est pas connu à l'avance. Le circuit est matérialisé par deux lignes noires tracées sur une bande blanche. Il peut y avoir des croisements, des ponts, des ralentisseurs, des tunnels.

Le châssis (groupe de propulsion et direction) est imposé, mais le choix de la carte de contrôle (microcontrôleur NxP) et des capteurs est libre. L'objectif est donc de construire une voiture qui soit autonome et roule le plus vite possible en détectant la piste.

Figure 2 - Vidéo complète : https://perso.esiee.fr/~antonraj/mes_projets/Nxp_cup/wp-content/video/nxp_cup_2018_esiee.mp4

La page principale de présentation de la NxP Cup où toutes les infos sont données : https://community.nxp.com/groups/tfc-emea

EMEA : region Europe, Middle-East (Moyen-Orient), Africa.

Intéret pédagogique

- Travailler en équipe
- Réalisation complète

- Prog. des microcontrôleurs (langage C) + programmation PC
- Electronique (contrôle moteur, caméra, radio...)
- Apprendre à résoudre des problèmes concrets
- Sujet transversal, robotique
- Autonomie
- Découverte, contacts avec le monde industriel, les contraintes, les délais

Principe

Il faut identifier en temps réel les bords de la piste qui sont tracés en noir sur fond blanc. Il existe plusieurs solutions pour cela : utilisation de caméras noir et blanc ou couleur, utilisation de caméras linéaires, utilisation d'une barre de capteur infra-rouge…

L'utilisation d'une barre est peu appropriée car encombrante. En effet elle doit être disposée juste au-dessus de la bande noire, de plus la sensibilité et la précision seront assez faible.

L'utilisation d'une caméra 2D standard noir et blanc ou couleur est idéale mais demande une relativement forte puissance de calcul embarquée ainsi que de la mémoire RAM pour stocker la ou les images : jusqu'en en 2017 seul certain microcontrôleurs étaient autorisés ce qui limitait cette possibilité relativement complexe.

L'utilisation d'une caméra linéaire est très adaptée, elle est fournie dans le kit de démarrage NxP.

Kit de démarrage NxP

Jusqu'en 2017, le seul châssis autorisé était le « modèle-C », celui dont on dispose à l'ESIEE :

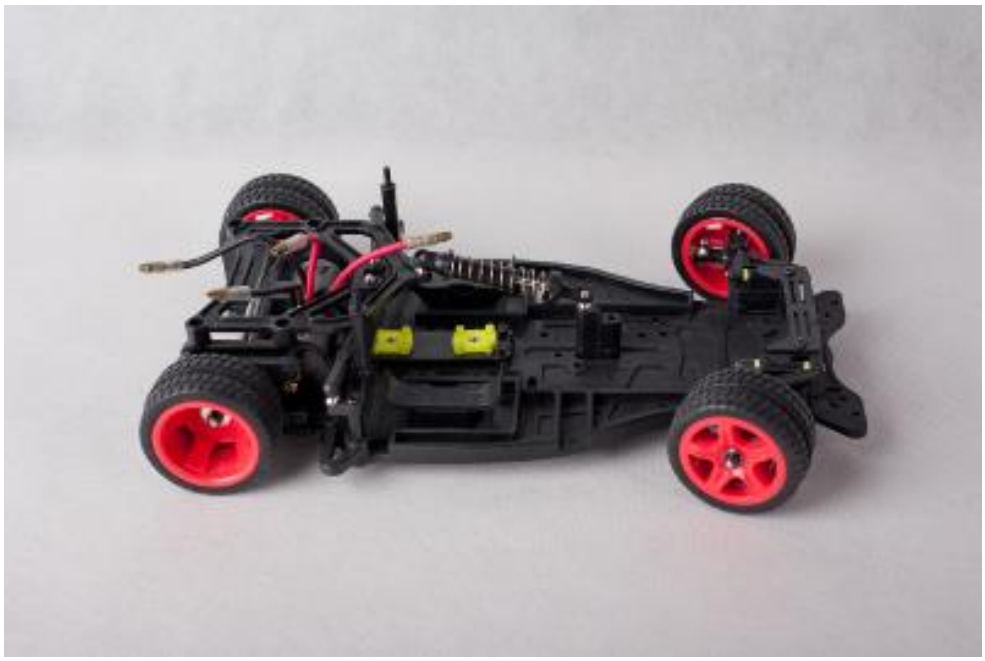


Figure 3- Châssis "Model C"

Depuis 2017 un nouveau châssis a vu le jour et fait l'objet d'une course spécifique, le modèle « Alamak » :

Figure 4- Châssis « Alamak »

Ces châssis comportent 2 moteurs à balais 7.2v maximum pour la propulsion et un servomoteur standard de modélisme pour la direction.

Pour piloter le servo et les moteurs, NxP a développé des cartes d'interface de puissance (2 versions successives pour le modèle C et une nouvelle version pour le modèle Alamak). Ces cartes offrent les mêmes fonctionnalités : elles comportent des ponts en H pour la commande des moteurs de propulsion et la connectique pour une ou deux caméras linéaires et pour un ou deux servo de direction.

Enfin, pour « l'intelligence » de la voiture, NxP propose plusieurs kits de développement à base de leurs microcontrôleurs : Kinetis FRDM-KL 25Z, TWR-K40X256, Qoriva MPC-5604B, K40 KwikStik…

La page suivante regroupe des liens vers les différentes options possibles pour le matériel : https://community.nxp.com/docs/DOC-1284

Architecture d'une voiture

Un véhicule est donc composé au minimum des éléments suivants :

- Un châssis « Alamak » ou « model C »
  - 2 moteurs de propulsion
  - 1 servo de direction
- Une carte de puissance pour les moteurs de propulsion
- Une caméra linéaire
- Une carte microcontrôleur
- Une batterie Nimh 7.2v 2500mAh max (imposé)

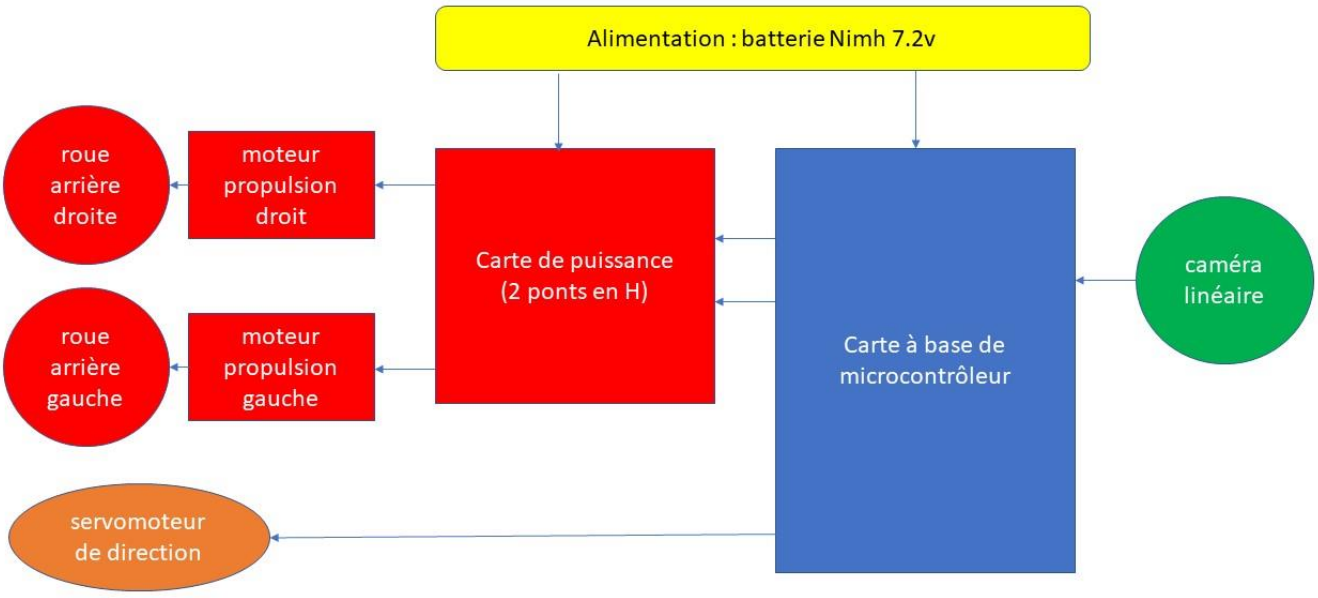


Figure 5 -Architecture fonctionnelle de la voiture

Choix du matériel ESIEE :

- Châssis « modèle C »,
- Cartes microcontrôleurs FRDM-KL25Z (et KL46Z plus puissantes dont les connecteurs sont compatibles),
- Cartes filles de puissance compatible FRDM-KL25Z (rev. 1 : https://community.nxp.com/docs/DOC-1059)
- Caméra linéaire « TAOS 1401 » (https://community.nxp.com/docs/DOC-1030)

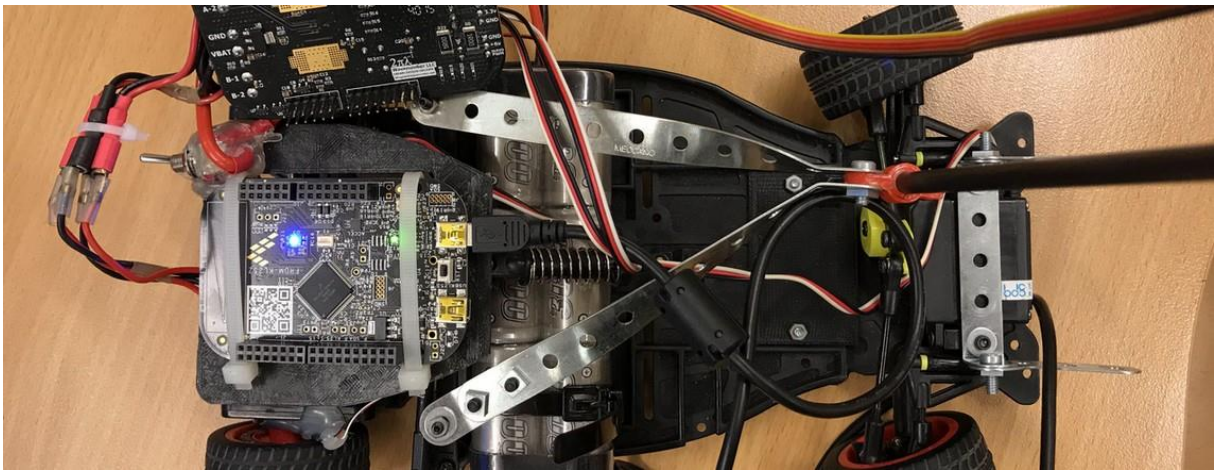


Figure 6 - Carte KL25Z sur châssis « modèle C »

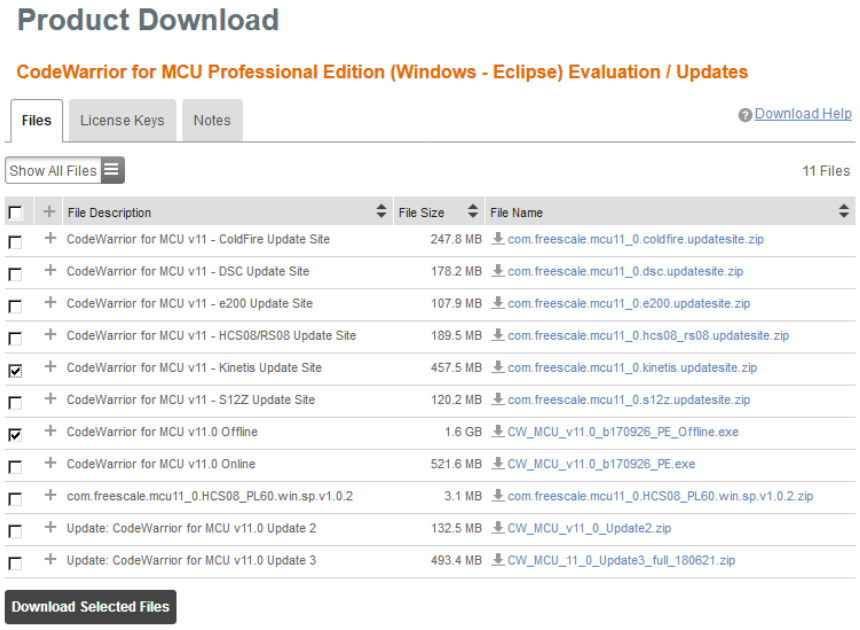
Prise en main Kinetis KL25Z

Avant de pouvoir utiliser la carte de développement FRDM-KL25Z, nous commençons par présenter l'installation de l'environnement, puis nous présentons la carte, comment la configurer avant de terminer par la présentation d'un exemple complet qui fait clignoter ses leds.

Environnement de développement logiciel

Sous windows : 3 grandes possibilités Code Warrior (l'IDE Freescale), Kinetis Studio, ou Keil-MDK µVision. Ils permettent tous de saisir du code C/asm, de le compiler, de le télécharger sur le micro, de faire du débug pas à pas.

1. **Code Warrior 10.6, 10.7 ou 11 (« CW ») :** c'est l'outil de dev. Freescale/NxP , l'avantage quand on l'installe c'est que l'on va disposer des exemples freescale pour chacune des cartes. **Les versions 10.6 et 10.7 sont les anciennes versions de CW (voir 2.) utilisées pour créer les exemples pour la voiture. La version 11 semble aussi le permettre, à vérifier.**
    a. « CodeWarrior 11 » est téléchargeable en version d'évaluation ici : https://nxp.flexnetoperations.com/control/frse/download?element=9346447
    Il faut sélectionner l'extension pour Kinetis :



    b. « CodeWarrior MCU v10.6 Evaluation Edition » la version complète limitée à 1 mois qui devient limitée en taille de binaire généré quand le mois est écoulé.
    c. « CodeWarrior MCU v10.6 Special Edition » la version illimitée en durée mais limitée en taille**. Je vous conseille d'utiliser celle-ci**, puis plus tard si vous êtes limité par les 128Ko (!!), d'installer la version Evaluation Edition.
    Remarque importante : les versions CW 10.6 et 10.7 ont été conçues avant Win7, le répertoire d'installation des programmes s'appelait alors « c:\Program Files », il n'y

avait pas de répertoire « c:\programme file x86 » dans lequel s'installent par défaut tous les softs 32bits => **ATTENTION si possible installer CW dans c:\Program Files** car si il est installé ailleurs des exemples risquent de ne pas fonctionner car les *includes* sont parfois codés en dur. Cependant pour mes tests tout s'est bien passé en l'installant dans c:/freescale/CW etc.

2. Kinetis Studio : il a été un temps le successeur de CodeWarrior pour la gamme des microcontrôleurs Kinetis, mais je pense finalement en voie d'abandon pour revenir à CodeWarrior 11.0 for MCU

3. **Keil MDK µVision** est l'outil supporté par ARM pour tous les micros de la famille des cortex M : Freescale (Kinetis), NXP (LPC), STM (32F 32L), Texas Instrument (Stelaris, Tiva) etc.). Son IDE est très simple et très rapide à prendre en main (c'est celui que nous utilisons pour nos TPs d'introduction aux Cortex M à l'ESIEE en première année). Comme CW, il existe en version limitée (MDK-Lite) en taille de code généré mais seulement 32Ko (ce qui est quand même largement suffisant pour les premiers projets). Il est téléchargeable depuis cette page, il faudra remplir un formulaire : http://www.keil.com/arm/mdk.asp Il est intéressant de le connaitre pour passer d'un constructeur à l'autre. De plus, il est fourni avec au moins un exemple (clignotement d'une led) pour quasiment toutes les cartes de dev. de tous les constructeurs. Il est aussi très intéressant en pédagogie car il contient aussi un simulateur assez complet permettant entre autre de voir les GPIO et autres périphériques (attention cependant, ce simulateur n'existe pas pour toutes les variantes de micro/constructeur, pour un constructeur donné il faut chercher la référence de la variante pour laquelle il existe un simulateur).

Le projet de démonstration fournit par NxP pour tester la voiture étant sous Code Warrior, nous choisissons donc

Présentation de la carte de développement « Freedom » FRDM-KL25Z

Le microcontrôleur à programmer est au centre « KL25Z ». « 80 LQFP » correspond au type du boitier (LGFP) et au nombre de broches (80). Il est en effet plusieurs variantes de ce processeurs avec plus ou moins de broche et donc des types de boitiers plus ou moins gros.

Pour mieux comprendre cette carte il faut analyser son manuel utilisateur et ses schémas disponibles dans l'archive suivante (chercher « frdm kl25z sur le site NxP), puis cliquez sur « User Guide »:

https://www.nxp.com/support/developer-resources/evaluation-and-development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-kl14-kl15-kl24-kl25-mcus:FRDM-KL25Z?&tab=Documentation_Tab&linkline=Users-Guide

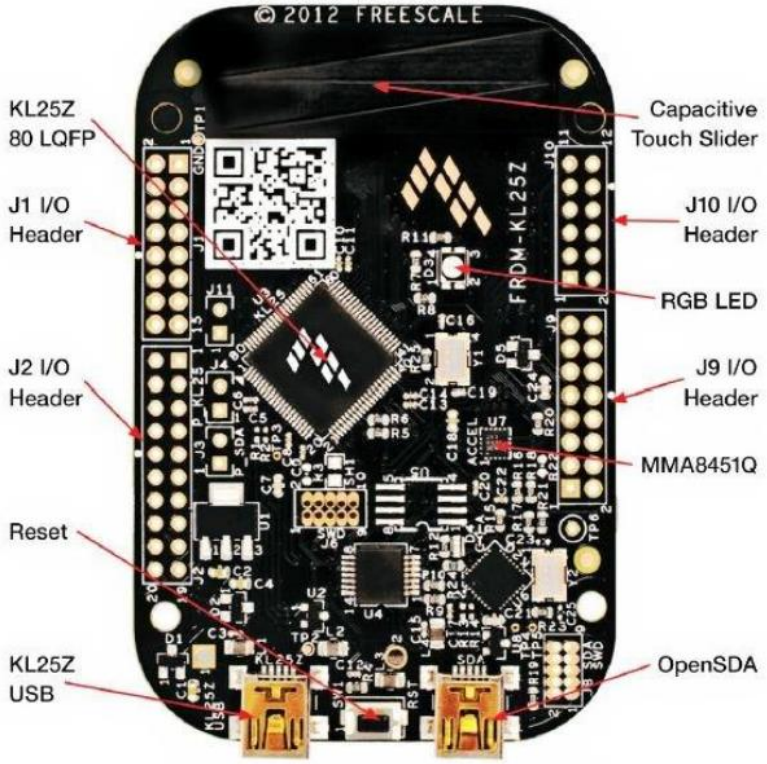
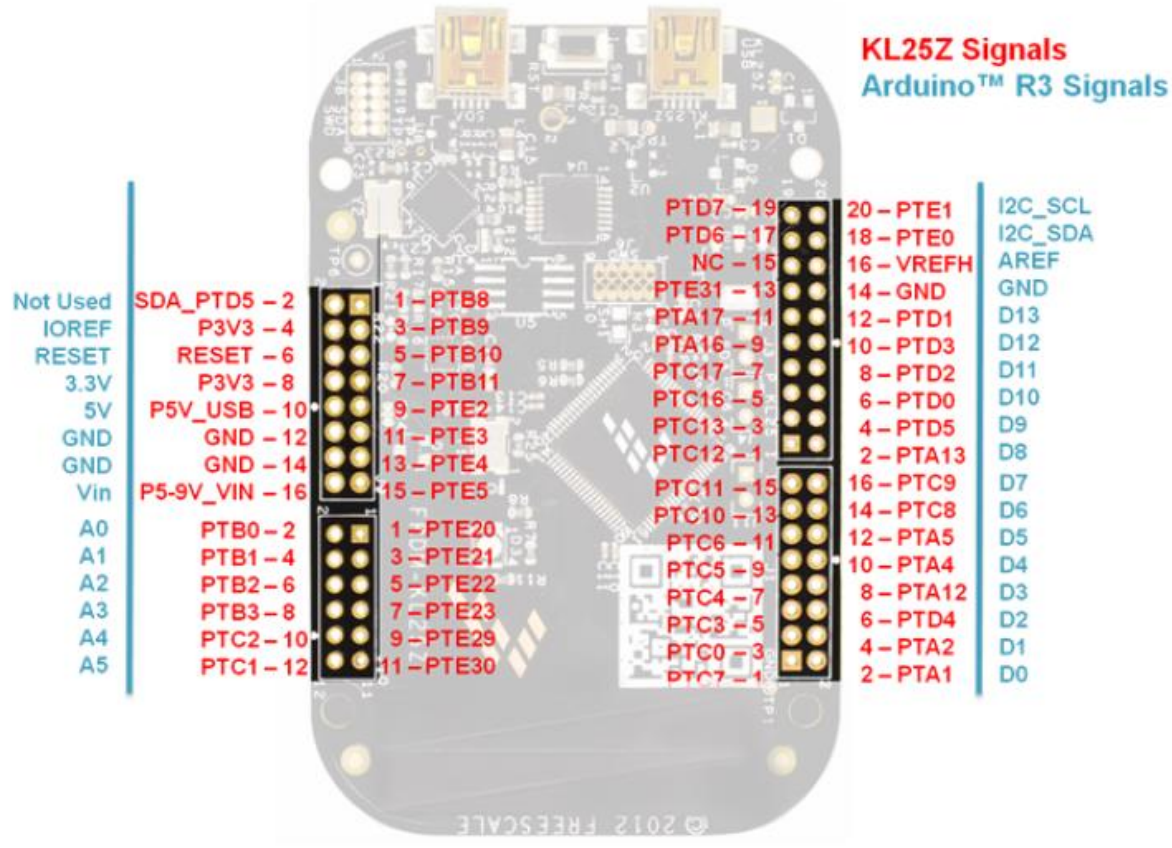


Figure 7- Carte FRDM-KL25Z -Vu côté composants

On constate que la carte comporte :

- une led RGB (constituée de trois leds dans un même boitier),
- un capteur inertiel MMA8451Q (accéléromètre)
- un « slider » tactile capacitif (en haut)
- 2 connecteurs USB,
- 4 connecteurs d'entrées-sortie de chaque côté (J1, J2, J9 et j10).

Le rôle de ces connecteurs est récapitulé dans la figure page 14 :



On constate ainsi que la disposition des broches est faites pour être compatible (en bleu) au maximum avec le brochage des cartes filles (« shield ») Arduino.

Le « block diagram » (figure suivante extraite de la page 4 du « user manual ») nous apprend que le KL25Z est un KL25Z128VLK4 80 LQFP », et qu'il est alimenté par les deux connecteurs USB par le biais d'un convertisseur 3,3v à faible perte (en rouge) mais alimentable aussi par l'un des connecteurs (vin).

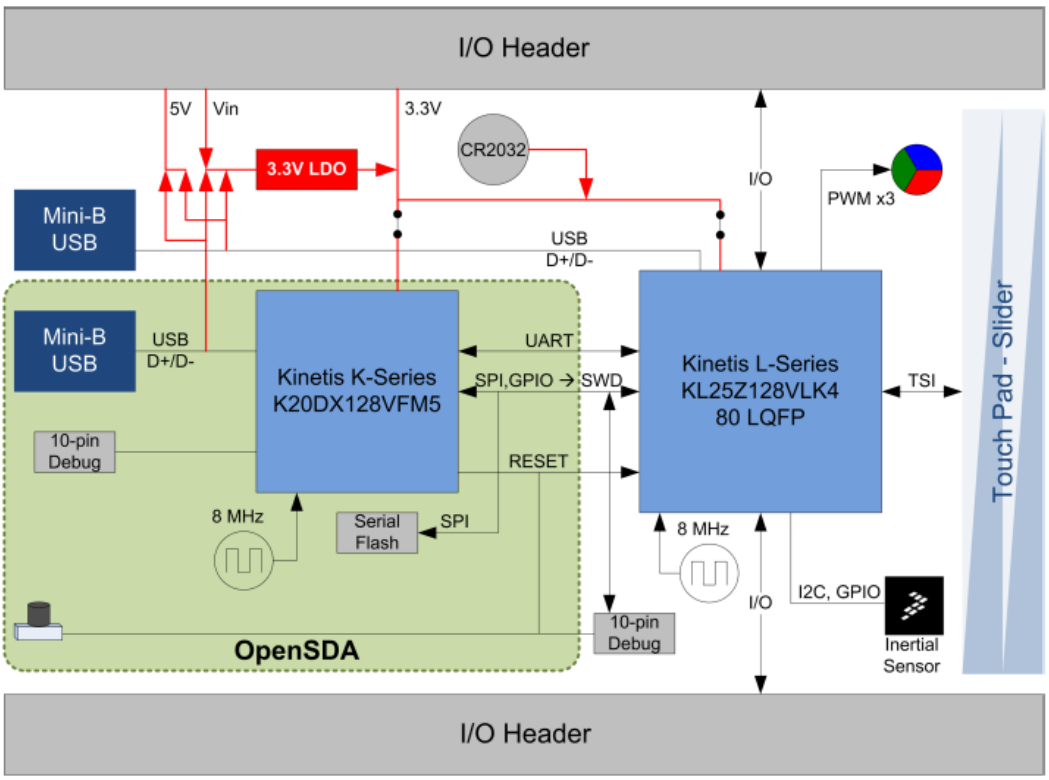


Figure 8 - Architecture de la carte FRDM-KL25Z

On y découvre aussi que les cartes Kinetis possèdent une interface de débogage incorporée : via une connexion USB entre le PC (qui exécute Code Warrior) et la carte il est possible de télécharger du code dans la RAM ou la FLASH du microcontrôleur KL25Z et ce sans matériel supplémentaire. C'est la partie sur fond vert de la figure ci-dessus suivante.

Cette interface de programmation fonctionne selon un protocole série appelée « OpenSDA ». Il utilise un autre microcontrôleur (K20DX128VFM5) dédié à cet usage uniquement. Pour travailler sur cette carte depuis le PC il faut donc connecter un câble USB entre le PC et le connecteur USB mini marqué « OpenSDA » (c'est-à-dire celui de droite sur la photo page précédente).

Configuration de la carte pour mode debug

Par défaut, quand la carte est neuve la carte elle est préprogrammée avec une démo qui fait varier les couleurs des leds en fonction de la position spatiale de la carte, mesurée à l'aide du capteur inertiel.

De plus, en connectant cette carte au PC (après avoir éventuellement installé les drivers) on constate que le microcontrôleur secondaire (K20DX128) n'est pas en mode OpenSDA mais émule une carte mémoire (firmware « mass storage ») : la carte est alors vue par le PC comme une clef USB. C'est très intéressant car pour placer une application dans la carte depuis windows il suffit de copier/coller le binaire de cette application (fichier d'extension .sc) dans cette « clef usb ». C'est donc très simple, NxP fournit ainsi plusieurs programme compilé dans le sous-répertoire « Precompiled Examples » de

l'archive « FRDM-KL25Z Quick Start Package » téléchargeable sur la page NxP dédié à cette carte.
Remarque : ne pas copier directement les fichiers entre l'application d'archivage et la clef, il est préférable de d'extraire les fichiers avant de la placer sur la carte.

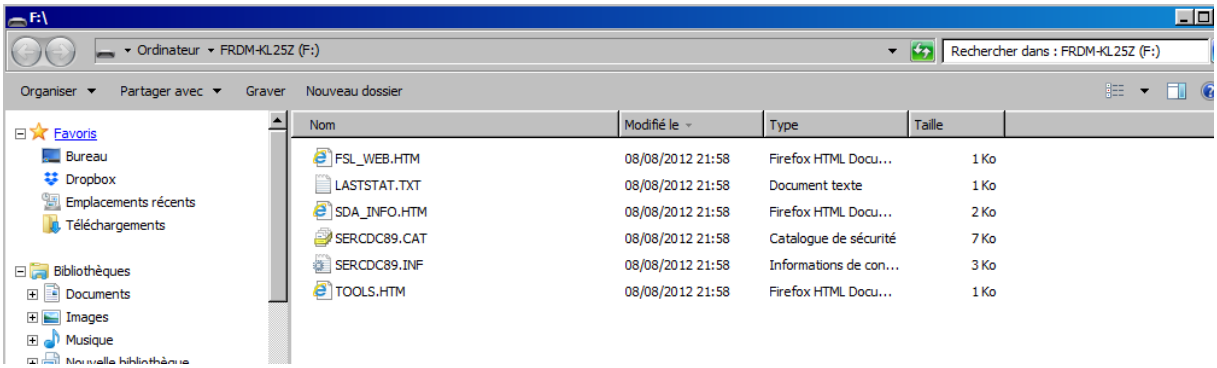


Figure 9 - FRDM-KL25Z vue par le PC comme une clef USB de même nom

En revanche dans ce mode, il n'est pas possible de faire du débogage. Pour la passer en mode débogage « OpenSDA » il faut reflasher le firmware du microcontrôleur K20 (pas le KL25Z) de la carte, ceci compte plusieurs étapes :

1. Tout d'abord il faut récupérer le firmware qui est développé par PEMICRO (partenaire de NxP). Le lien peut être obtenu en cliquant directement sur le fichier « SDA_INFO.HTM » présent sur la carte (voir figure précédente) , ou bien en allant sur leur site : http://www.pemicro.com/opensda/index.cfm Il faut alors télécharger le fichier archive de la section « OpenSDA Firmaware » (« Firmware Apps ») : Pemicro_OpenSDA_Debug_MSD_Update_Apps..date...zip. Cette archive contient les firmwares pour plusieurs cartes de développement, il faut extraire les 2 fichiers qui correspondent à notre FRDM-KL25Z, c'est-à-dire : DEBUG-APP_Pemicro_v108.SDA et si besoin MSD-DEBUG-FRDM-KL25Z_Pemicro_v118.SDA qui est le firmware d'origine si l'on voulait remettre la carte dans le mode « Mass Storage ».
2. Ensuite il faut placer la carte en mode « Firmware update », pour cela :
    a. Débrancher le câble USB de la carte
    b. Appuyer sur le bouton poussoir de la carte
    c. Reconnecter le câble USB sur le même port USB (OpenSDA) en maintenant toujours le bouton poussoir appuyé
    d. Relâcher le bouton poussoir après quelques secondes
3. La carte devrait être vu maintenant comme une clef USB dont le nom n'est plus KL25Z comme précédemment, mais « BOOTLOADER »

Figure 10- La FRDM-KL25Z apparait comme une clef USB « BOOTLOADER »

4. Il suffit de placer le fichier de firmware (DEBUG-APP_Pemicro_v108.SDA) sur ce nouveau lecteur, attendre 1 minute pour être certain que la mise à jour s'effectue,
5. puis brancher/rebrancher l'USB (sans appuyer sur le poussoir puisque le firmware est maintenant à jour).
6. Cette fois, plus aucune fenêtre ne doit s'ouvrir, la carte n'est plus vue comme une clef USB mais comme une interface de programmation OpenSDA (voir ci-dessous), mais pour cela il peut être nécessaire d'installer le driver windows de cette interface de programmation OpenSDA : sur la même page PEMICRO où les drivers ont été trouvés, il y a une rubrique « Windows USB Drivers » avec un exécutable (.exe) qu'il faut exécuter. Débrancher la carte avant de l'exécuter.
7. Il est toujours préférable de s'assurer que la carte est bien configurée en mode OpenSDA et que le driver est bien installé. Pour cela il faut ouvrir le « gestionnaire de périphériques » (taper son nom dans la barre de recherche windows), et dans la rubrique « Ports (COM et LPT) » vous devez voir apparaitre « OpenSDA » ainsi que le numéro de port série associé (COM35 dans la capture ci-dessous) :

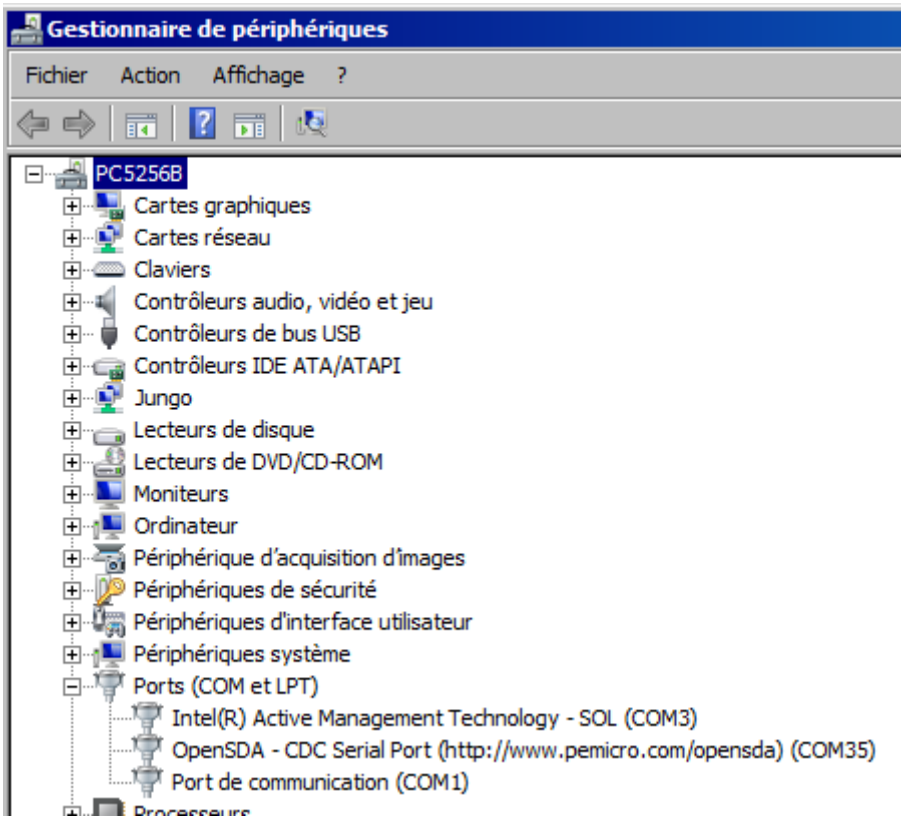


Figure 11- Gestionnaire de périphériques

REMARQUE WINDOWS 10 : sous windows 10, en fonction des versions, il se peut que les transferts vers la carte ne fonctionne pas, que ce soit en mode « Mass Storage » ou en mode de mise à jour du bootloader. Cela est dû au fait que windows 10 créé des fichiers cachés d'indéxation et qu'ils sont aussi copiés sur la carte. Pour que cela fonctionne sous windows 10, il faut utiliser une des trois solutions suivantes après avoir lancé le gestionnaire de services (« services.msc »), puis désactivé le service de stockage et windows search :

- ⇨ 1ère possibilité : éditeur de statégie

  Lancer l'éditeur de stratégie (« gpedit.msc »), puis aller dans « Config. De l'ordinateur »➔ « modèle d'administration »➔composants windows ➔ Rechercher et cliquer sur la politique « Ne pas autoriser d'emplacement sur les disques amovibles aux bibliothèques », activer cette politique et OK. Redémarrer le PC.

- ⇨ 2ème possibilité : éditeur de registre

  Lancer l'éditeur de registre (« regedit »), puis chercher l'une clefs « search » ou « windows search » qui contient la propriété « DisableRemovableDriveIndexing » qu'il faut passer à 1.

Si elle n'existe pas vous pouvez la créer dans "HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows Search". Puis ajouter une clef nommée "DisableRemovableDriveIndexing" avec le type "REG_DWORD" et la placer à 1. Redémarrer le PC.

⇨ 3ème possibilité : trouver une machine windows 7 pour changer le firmware en OpenSDA. Une fois le changement effectué, il n'y a plus de problème liés à windows 10.

(voir https://os.mbed.com/users/maripogoda/notebook/update-frdm-kl25z-bl-with-win8/ pour des informations complémentaires).

Premier programme « From Scratch » avec Code Warrior

Lancer Code Warrior. Comme tous les outils basés sur l'environnement Eclipse il vous demande de choisir l'emplacement d'un espace de travail « worspace », vous pouvez laisser celui par défaut ou le renommer par exemple « KL25Workspace ». Ainsi vous pourrez switcher d'une espace à un autre facilement si vous avez plusieurs projets.

Ensuite il suffit de créer un nouveau projet : menu new→BareBoard Project et choisir le processeur qui correspond à la carte cible en regardant sa référence, tout simplement :

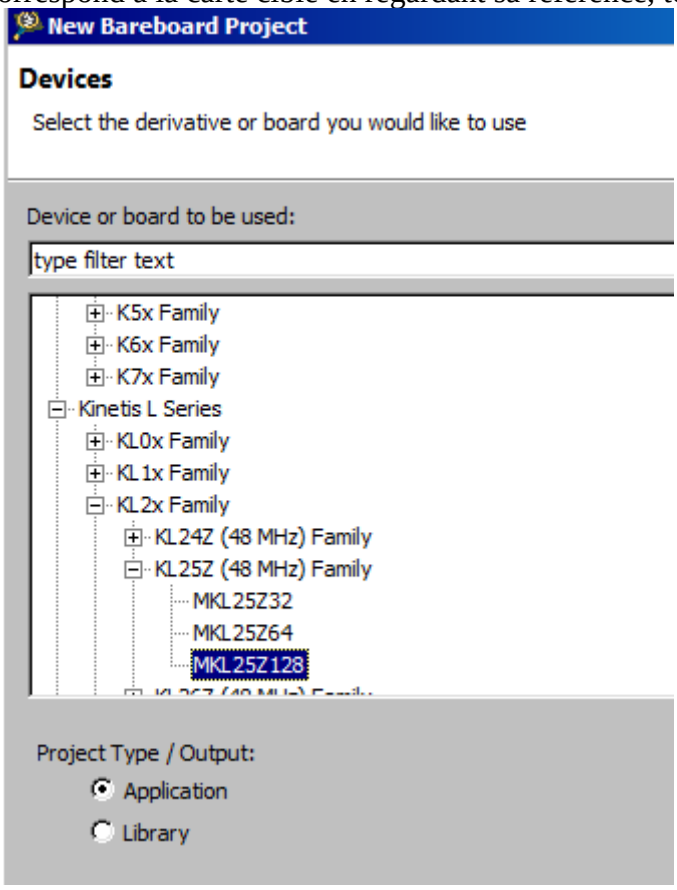
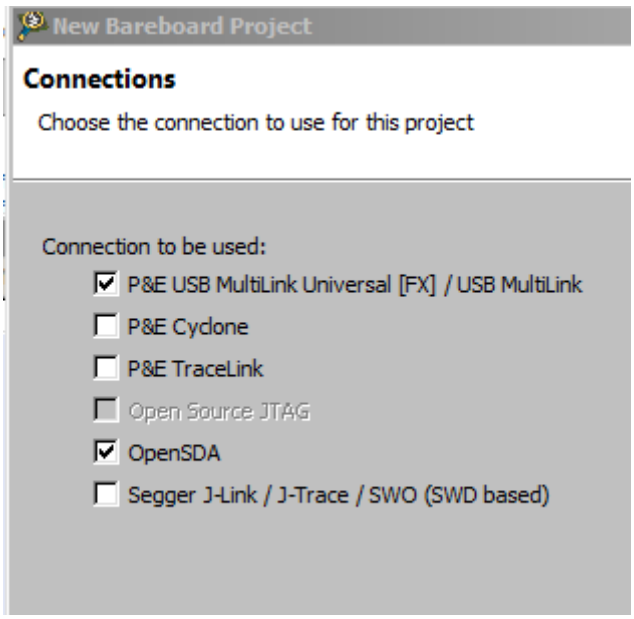


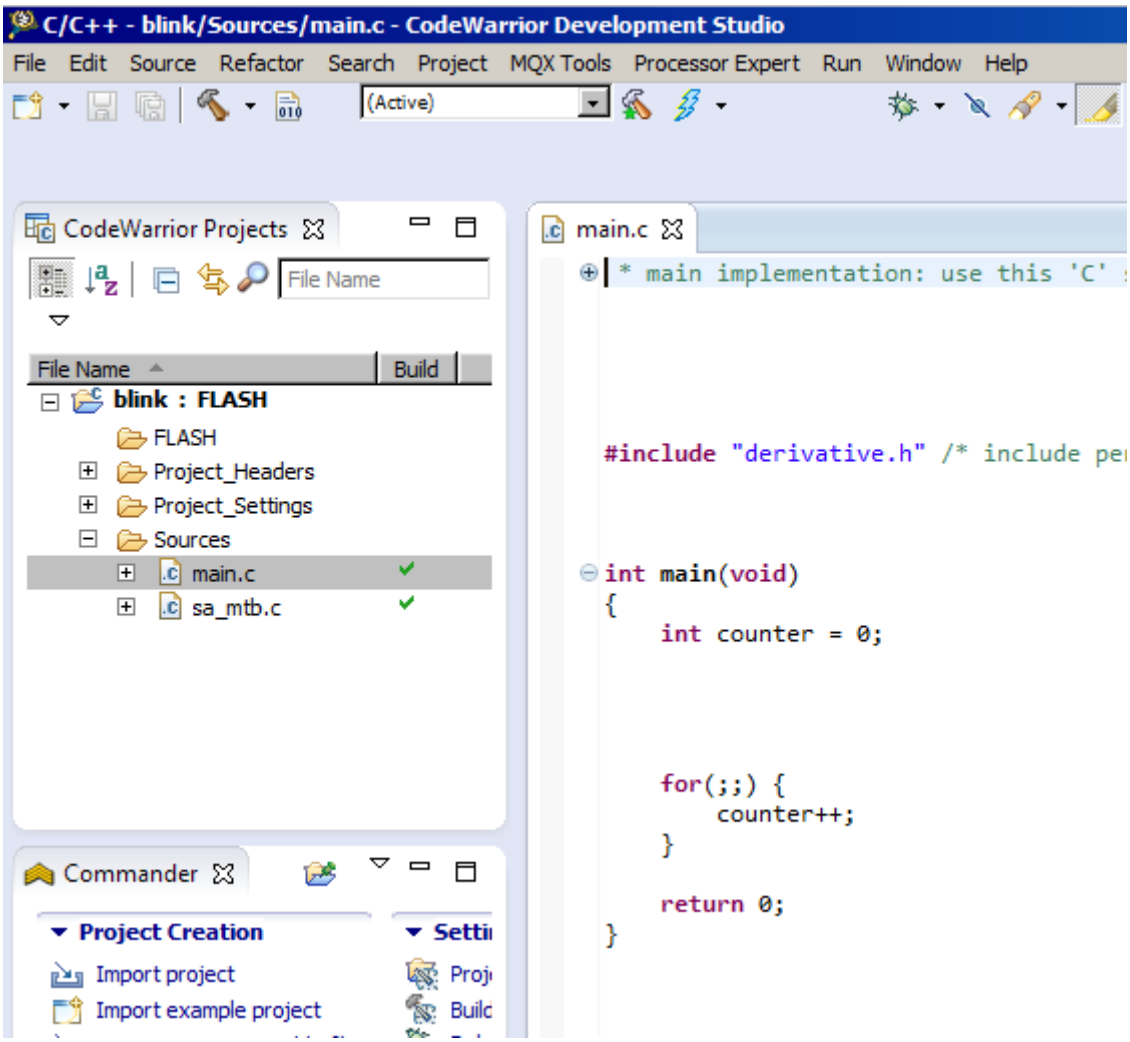
Figure 12- Reference du processeur KL25Z128

Choisir ensuite la connexion OpenSDA

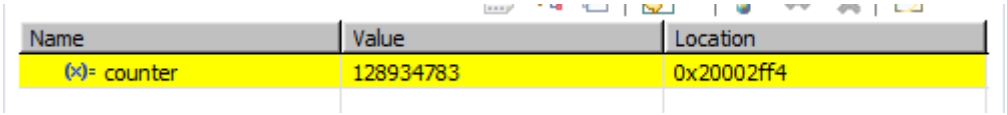


Puis laisser les configurations proposées par défaut sur les onlgets qui suivent jusqu'à cliquer sur « Finish ».

Code Warrior génère alors un projet de base avec un fichier main.c contenant une boucle infinie :



Vous devez pouvoir le compiler sans erreur en cliquant sur le marteau de la barre de menu. Il faut ensuite le transférer dans la mémoire de votre carte en cliquant sur le petit insecte vert. CW communique alors avec votre carte (qui doit être connectée par USB !), prend le contrôle du processeur et y télécharge votre programme dans sa mémoire RAM. Cliquer enfin sur le rectangle vert pour lancer l'exécution. Bravo, le programme tourne….mais comment le savoir ? Faites pause (bouton à gauche du rectangle rouge) . Ceci stoppe le processeur et lit la valeur des variables de votre programme (la valeur de counter s'affiche en haut à droite) :



et une flèche vous indique là où s'est arrêté le processeur :

```
main.c
{
    int counter = 0;

    for(;;) {
        counter++;
    }

    return 0;
}
```

Vous pouvez relancer/arrêter à nouveau pour vérifier les incréments du compteur :

Name	Value	Location
(x)= counter	132460722	0x20002ff4

Figure 13- nouvelle valeur du compteur

Oui, il compte très vite : à 48 MHZ !

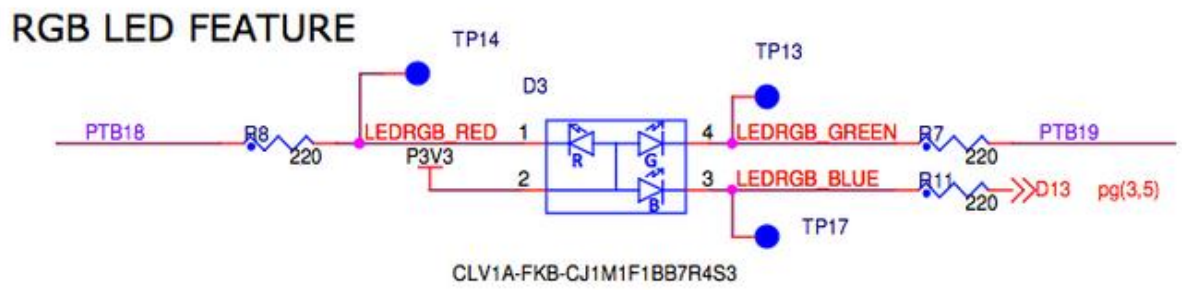
Vous pouvez mettre fin à la session de debbug en cliquant sur le rectangle rouge, puis revenir dans le projet en cliquant sur « C/C++ » dans la barre du haut à droite.

Notez que ce programme inclut un fichier « derivative.h » qui lui-même inclut un fichier <MKL25Z4.h> (faites simplement un clic droit sur le nom pour ouvrir le fichier). Parcourez ce dernier pour découvrit qu'il contient les définitions des adresses de tous les registres internes du processeur, cela va s'avérer très pratique pour la section suivante !

Faire clignoter une led

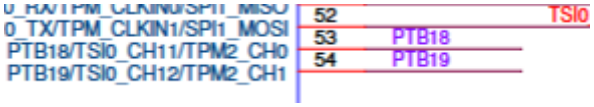
C'est le « hello world » de l'embarqué, on commence souvent par allumer une led pour tester la chaîne de développement. Sur cette carte il y a 3 leds dans un même boitier, choisissons de faire clignoter la rouge (vous pourrez changer facilement !).

Pour l'allumer, il faut l'identifier sur le schéma du « User manual » de cette carte (p.13) ou directement dans le pdf contenant ces schémas : FRDM-KL25Z_SCH_REV_E.pdf page 3 en bas à gauche :



Les leds rouge, verte et bleu sont respectivement connectées sur le bit 18 du port B, le bit 19 du port B et le bit1 du port D.

En suivant le nom du fil on trouve même le numéro des broches du KL25Z : 53 pour PTB18 et 54 pour PTB19.



En cherchant cette broche sur la data sheet page du processeur (KL25P80M48SF0.pdf), on découvre (p48 de KL25P80M48SF0.pdf et dans la colonne « 80 LQFP » puisque c'est le suffixe de la référence de notre processeur soudé sur cette carte, cela correspond au nombre de broches) que cette broche peut être configurée pour l'une des deux fonctions : PTB18 (colonne ALT1) ou TMP2_CH0 (colonne ALT 3).

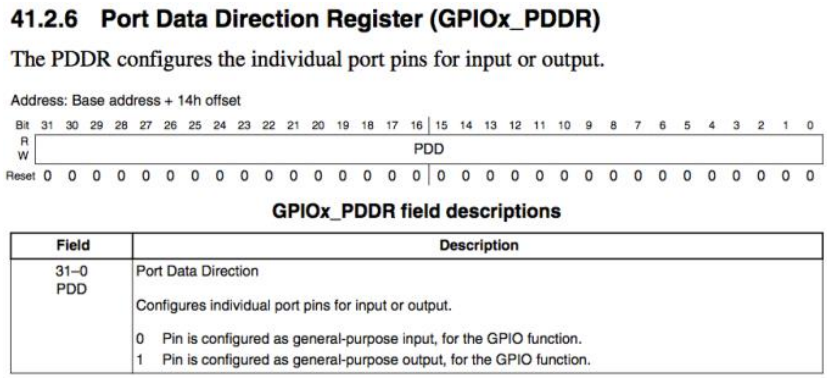
80 LQFP	64 LQFP	48 QFN	32 QFN	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	
51	39	31	—	PTB16	TSI0_CH9	TSI0_CH9	PTB16	SPI1_MOSI	UART0_RX	T
52	40	32	—	PTB17	TSI0_CH10	TSI0_CH10	PTB17	SPI1_MISO	UART0_TX	T
53	41	—	—	PTB18	TSI0_CH11	TSI0_CH11	PTB18		TPM2_CH0	

En effet, en lisant le référence manuel (KL25P80M48SF0RM.pdf) et en particulier le chapitre 11 « Port control and interrupts) , on découvre qu'il est possible de choisir le périphérique auquel on veut connecter la broche en fonction de la valeur « ALT » qu'il faut placer dans les bits 10 à 8 du registre « Pin Control Register n », communément appelé « PORTx_PCRn » (p184).

Reserved	This read-only field is reserved and always has the value 0.
10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific).

On découvre que PTC13 correspond au rôle de broche d'entrée-sortie à usage générale : GPIO. Le chapitre 41 du reference manual est dédié aux entrées-sorties. On y découvre que les broches sont regroupées par paquets de 32, pour former des ports : port A, port B, port C etc. (le nombre de port dépend du microcontrôleur, de son boitier etc.). La broche 53 nommé PTB18 fait donc partie du port GPIO B, bit 18.

Il nous faut ensuite indiquer dans le registre GPIOx_PDDR si cette broche est une entrée ou une sortie. Les informations ci-dessous, toujours tirées du reference manual nous indique que par défaut toutes les broches sont configurées en entrées (0 au reset). Il nous faut donc placer le bit 18 à 1 (en sortie).



41.2.6 Port Data Direction Register (GPIOx_PDDR)

The PDDR configures the individual port pins for input or output.

Address: Base address + 14h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PDD																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIOx_PDDR field descriptions

Field	Description
31–0 PDD	Port Data Direction Configures individual port pins for input or output. 0 Pin is configured as general-purpose input, for the GPIO function. 1 Pin is configured as general-purpose output, for the GPIO function.

Figure 14- Direction p.778

Grâce aux macros qui existent dans le fichier d'include du projet crée on peut directement écrire :

GPIOB_PDDR = (1<<18) ;

Ensuite pour allumer la led rouge il suffira d'activer (mettre à 1) le bit correspondant grâce au registre GPIOx_PDOR

41.2.1 Port Data Output Register (GPIOx_PDOR)

This register configures the logic levels that are driven on each general-purpose output pins.

NOTE

Do not modify pin configuration registers associated with pins not available in your selected package. All un-bonded pins not available in your package will default to DISABLE state for lowest power consumption.

Address: Base address + 0h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PDO																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIOx_PDOR field descriptions

Field	Description
31–0 PDO	Port Data Output Register bits for un-bonded pins return a undefined value when read. 0 Logic level 0 is driven on pin, provided pin is configured for general-purpose output. 1 Logic level 1 is driven on pin, provided pin is configured for general-purpose output.

Figure 15- Output p775

Mais il y a encore mieux : il est possible de faire un « toggle » (une inversion) de l'état d'une broche en utilisant le registre GPIx_PTOR : (ce type de registre n'existe pas sur tous les microcontrôleurs !)

41.2.4 Port Toggle Output Register (GPIOx_PTOR)

Address: Base address + Ch offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0																															
W	PTTO																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIOx_PTOR field descriptions

Field	Description
31–0 PTTO	Port Toggle Output Writing to this register will update the contents of the corresponding bit in the PDOR as follows: 0 Corresponding bit in PDORn does not change. 1 Corresponding bit in PDORn is set to the inverse of its existing logic state.

Figure 16- Toggle, p777

Nous avons presque tous les élément necessaire pour faire clignoter une led, mais une chose importante reste à faire pour pouvoir utiliser le port B : l'activer !

En effet, par défaut au demarrage du processeur, la plupart de ses périphériques sont désactivés pour minimiser sa consommation électrique. La page 206 du reference manual nous explique que pour activer les périphériques il suffit d'activer un bit dans le registre SIM_SCG5. Ceci a pour effet de piloter un multiplexeur qui fait arriver l'horloge sur le périphérique voulu :

12.2.9 System Clock Gating Control Register 5 (SIM_SCGC5)

Address: 4004_7000h base + 1038h offset = 4004_8038h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0												0	0		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0		PORTE	PORTD	PORTC	PORTB	PORTA		1	0	TSI	0			0	LPTMR
W																
Reset	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

SIM_SCGC5 field descriptions

Field	Description
10 PORTB	Port B Clock Gate Control This bit controls the clock gate to the Port B module. 0 Clock disabled 1 Clock enabled

Dans le cas du port B c'est donc le bit numéro 10 qu'il faut mettre à 1. On peut le faire très simplement avec la ligne suivante, si l'on a préalablement remarqué qu'il existait un symbole tout prêt dans le fichier MKL25Z4.h : #define SIM_SCGC5_PORTB_MASK 0x400u

On peut donc écrire :

SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;

Finalement, le programme complet devient :

```
#include "derivative.h" /* include peripheral declarations */

int main(void)
{
	SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK; //on active le PORT B
	PORTB_PCR18 = (PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK); //broche 53 configuré sur bit 18 du PORT B avec DSE_MASK pour que la broche puisse sortir plus de courant
	GPIOB_PDDR = (1<<18); //bit 18 du PORT B en sortie
	for(;;) // boucle infinie
	{
		GPIOB_PTOR = (1<<18);
	}

}
```

On le télécharge et on lance l'exécution….

Arg, la led rouge reste annulée, mais pourquoi ?

En fait elle clignote tellement vite que notre œil ne s'en aperçoit pas. Il faut ajouter une boucle d'attente dans la boucle infinie afin de la ralentir :

```
#include "derivative.h" /* include peripheral declarations */

int main(void)
{   int time;
      SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK; //on active le PORT B
      PORTB_PCR18 = (PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK); //broche 53 configuré
sur bit 18 du PORT B avec DSE_MASK pour que la broche puisse sortir plus de
courant
      GPIOB_PDDR = (1<<18); //bit 18 du PORT B en sortie
      for(;;) // boucle infinie
      {
            GPIOB_PTOR = (1<<18);
            time = 100000;
            while (time>=0)
                  time--; //pour perdre du temps

      }
}
```

Prise en main de la voiture

Environnement matériel

Il faut enficher la carte de puissance au-dessus de la carte de développement FRDM-KL25Z.

Ensuite il faut connecter les moteurs de propulsions, la batterie, le servo et la caméra comme indiqué sur la figure ci-dessous :

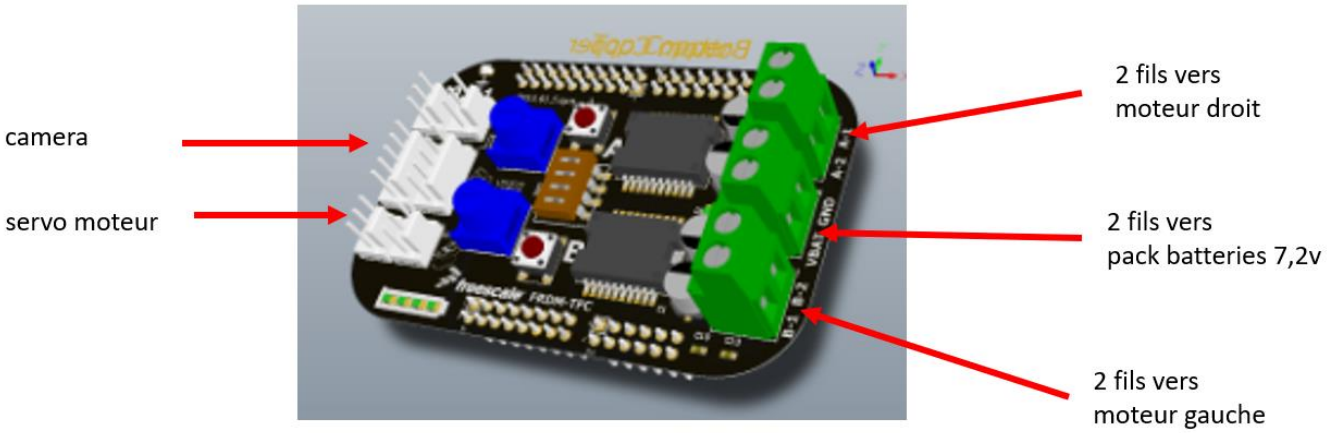


Figure 17 – Connexions de la carte de puissance

Programme d'exemple NxP

NxP fournit un programme complet d'exemple (« FRDM-TFC-R1.0 »)qui permet de prendre en main la carte microcontroleur ainsi que la caméra, les moteurs et le servo. Ce programme est uniquement compatible avec la carte FRDM-KL25Z connectée à la carte fille de puissance rev. 1.

Lien : https://community.nxp.com/docs/DOC-93914 , section « Codewarrior « Bare Metal Example Code »

Pour essayer ce programme :

a) **Brancher votre carte** en USB (il faudra peut-être rechercher les drivers manuellement dans le répertoire de freescale/CW si windows ne les trouve pas, chez moi c'est c:/Freescale/CW MCUV10.5/Drivers/Segger)
Il est important de le faire avant de lancer CW, ainsi la carte sera présente dans les menus dès le début (les profils de compilation seront existant)

b) Lancer CW qui propose alors un emplacement par défaut pour le workspace : vous pouvez le changer si vous voulez, et quand vous voudrez passer d'un projet à l'autre ou l'archiver le transmettre le plus simple sera de le zipper.
c) Ensuite ouvrir un projet d'exemple existant, utilisez « Help->Welcome -> Example project » : il faut choisir le type de famille de processeur voulu (des kinetis dans notre cas) :). Attention ne pas prendre les "Processors Expert example project" pour commencer. Processor expert est très bien pour générer automatiquement des configurations de ports d'IO, de modules etc, mais n'est pas adapté quand on ne sait pas ce que l'on fait.
d) Une liste de projets pour Kinetis va alors s'afficher. Je vous conseille d'en choisir un seul à la fois. Pour savoir lequel est adapté à votre carte, élargissez la fenêtre de façon à voir le nom complet du projet exemple ainsi que son emplacement : en fonction de son emplacement (/k60 /k60 etc.) vous saurez pour quel processeur est destiné l'exemple. En cochant une case CW propose alors de copier les fichiers dans le workspace ou non, il me semble qu'une seule des 2 options fonctionne bien (il est toujours préférable de copier et laisser ainsi les fichiers originaux intacts.
e) Une fois le projet ouvert, vous avez 2 choix :
    a. Soit on téléchargera le code en RAM (le programme sera perdu dès un reset ou une coupure d'alimentation)
    b. Soit on téléchargera le code en mémoire Flash : c'est plus long mais le programme reste en mémoire ensuite.

    Pour changer entre Flash ou RAM il suffit de cliquer sur le nom du projet (partie de gauche de CW : explorateur de fichier) et de sélectionner le mode voulu dans la liste déroulante. Parfois il n'y aura pas de choix, une seule des deux configurations aura été prévue.

f) Ensuite il suffit de compiler (build, le petit icone d'un marteau) le programme, puis de le télécharger sur la cible (download/debug, l'icône du petit insecte "bug"). Il est très important d'avoir branché la carte avant, sinon CW demande de créer soit même un profil de download/debug ce qui est fastidieux. Si il faut choisir, sur les FRDM par exemple, on utilise OpenSDA. Si aucune carte n'est trouvé, ou bien que le driver n'a pas été installé on tombe sur une fenêtre "P&E Assistant" : il faut alors essayer Refresh/Retry jusqu'à voir la carte, éventuellement changer la carte de port USB…
g) Une fois le programme télécharger, faire « run » dans le debuggeur et admirez vos leds clignoter (si c'est l'exemple que vous avez choisi) !

Autres exemples : il se peut que vous trouviez des exemples de code directement sous forme de zip sur le site de freescale ou ailleurs. Si ce sont des projets CW il faut les ouvrir en faisant "File" puis "Import" puis "Existing Projects into Workspace"

Utilisation du programme d'exemple

Sur la carte fille de puissance fournit par NxP, il y a 4 microswitchs, 4 leds et 2 potentiomètres :

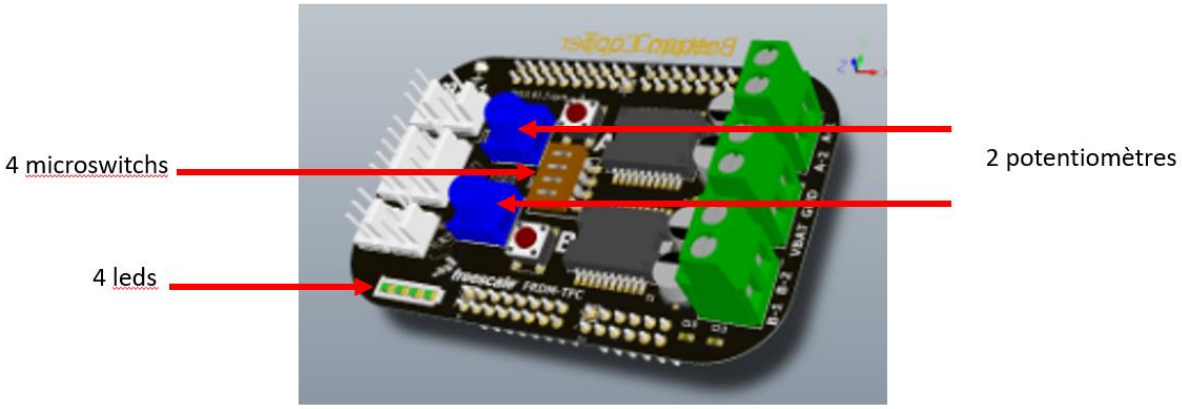


Figure 18 - Dip switch, leds de la carte de puissance

Le programme d'exemple permet de tester un à un les différents éléments de la voiture en fonction de la configuration des DIP switch : (remarque connecter la batterie pour faire ces tests)

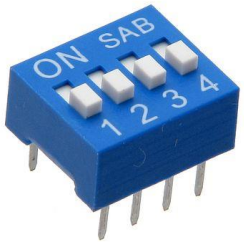


Figure 19 - DIP switch configuré à 0000

- **Position 0000** : (tout sur OFF) permet de tester les 2 poussoirs (points rouges sur la photo ci-dessus) et d'allumer les leds en fonction du poussoir appuyé.
- **Position 0010 :** (OFF OFF ON OFF) permet de tester le servo moteur de direction. La direction tourne en fonction de la position du potentiomètre. Remarque : il est possible de connecter un 2nd servomoteur de direction et de le piloter avec le 2nd potentiomètre
- **Position 0100 :** permet de tester les moteurs de propulsion individuellement. La vitesse et le sens de rotation dépendent de la position des potentiomètres.
- **Position 0110 :** appelé aussi « Garage Mode », les valeurs correspondants à la luminosités de chacun des 128 pixels de la caméra seront envoyé sur le port série du PC au moyen du câble USB connecté entre la carte FRDM-KL25Z et le PC. Pour visualiser ces valeurs il faut :
    a. Identifier sur quel port série (COM sous windows) apparait la carte KL25Z. Pour cela lancer le « gestionnaire de périphériques » windows et regarder dans « Ports(COM et LPT) » le numéro du port COM qui a été attribué à cette liaison (attention, cela peut

changer entre deux connexions…). C'est par exemple COM3 sur la capture ci-dessous :

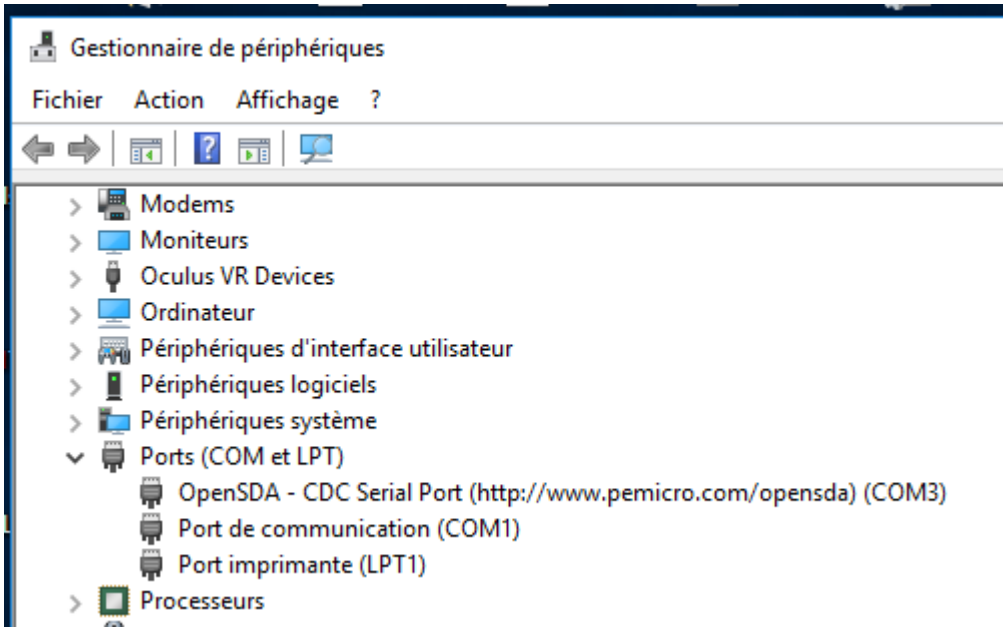


Figure 20 - Gestionnaire de périphérique où la carte apparait sur le COM3

b. Lancer un émulateur de terminal série (hyperterminal, putty, teraterm par exemple) et configurer cet émulateur pour qu'il communique avec le port COM identifié précédemment. Il faut aussi fixer la vitesse de transmission à 115200 (comme indiqué dans le fichier TFC_Config.h).

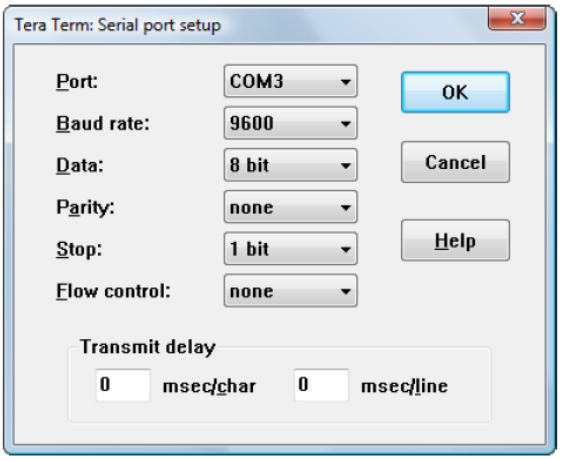


Figure 21-Configuration TeraTerm : modifier le Baudrate et le port COM

c. Une fois lancé, l'émulateur affiche à l'écran tout les octets reçus sur le port série et le code ASCII des caractères frappés au clavier sont envoyés sur ce même port série (donc vers la carte KL25Z). On doit donc voir à l'écran s'afficher des lignes de 128 nombres compris entre 0 et 4096 correspondant à l'intensité lumineuse de chacun des 128 pixels de la caméra linéaire

Remarque : en fait il est possible de connecter 2 caméras linéaires, il y a donc à chaque fois affichage de 2 lignes.

En mode « garage », plutôt que d'afficher les pixels que voit la caméra sous forme numérique, il est aussi possible de les afficher graphiquement à l'aide d'un programme pour Labview (fourni dans le même zip que l'exemple TFC) ou pour Matlab (téléchargeable sur le lien suivant : https://www.mathworks.com/matlabcentral/fileexchange/46997-nxp-cup-companion).

Une troisième possibilité pour afficher l'image de la caméra est d'utiliser le programme que j'ai créé spécifiquement à base du framework « processing », voir annexe 1.

Description du programme d'exemple en C

Pré-requis

Le projet contient plusieurs sous répertoires et fichiers.

- Le sous-répertoire « Sources » contient le fichier main.c c'est-à-dire le programme principal. Il contient aussi un sous-répertoire TFC : chaque fichier du programme TFC contient des fonctions spécifiques à une fonctionnalité données. C'est donc dans ce répertoire que l'on trouve le code des fonctions qui permettent de lire l'image de la caméra, de piloter les moteurs, de piloter les servos etc.
- Le sous-répertoire « Project_Headers » contient
    - le fichier « MKL25Z.h » regroupant les déclaration de toutes les adresses des périphériques
    - un sous-répertoire TFC qui contient les signatures des différentes fonctions définis dans « Sources\TFC »

Le fichier main.c est le plus intéressant pour commencer puisque c'est notre point d'entrée

Il commence par initialiser chacun des périphériques un à un grâce à la fonction TFC_init() ;

Puis vient la boucle infinie principale (**for(;;))** qui successivement :

1. Test si des caractères ont été envoyé par l'interface série (je n'ai pas encore évoqué ce mode)
2. Récupère l'état des DIPs switchs (position 0000, 0010, etc.) et en fonction des DIPs switch exécute le code correspondant

Voici une version simplifiée de ce main :

```
TFC_Init();
for(;;){
        switch((TFC_GetDIP_Switch()>>1)&0x03)
        {
        default:
                case 0 :  //Demo mode 0 just tests the switches and LED's
                case 1 : //Demo mode 1 will just move the servos with the on-board
        potentiometers
                case 2 : //Demo Mode 2 will use the Pots to make the motors move
                case 3 : //Demo Mode 3 will be in Freescale Garage Mode.  It will beam data
        from the Camera to the Labview Application
        }
}
```

Gestion du temps : Systick et PIT

Systick

Avant d'étudier les différents modes, commençons par analyser comment la mesure du temps est effectuée dans ce code.

Le microcontroleur KL25Z repose sur un cœur ARM Cortex M0 générique. Ces processeurs possèdent tous un timer interne « SysTick » qui peut générer une interruption (IT) à chaque « tic » du timer. La fréquence des tic est configurable et liée à la fréquence du processeur.
Remarque : chaque constructeur rajoute ensuite ses propres périphériques timer en plus du systick, ainsi nous verrons beaucoup plus tard que NxP a aussi ajouté un périphérique « PIT ».

Dans le code TFC de l'exemple NxP, on trouve la configuration de ce timer dans la fonction « TFC_InitSystick () » appelé par TFC_Init( ) au début du main :

```
static volatile unsigned int DelayTimerTick = 0;
volatile uint32_t TFC_Ticker[NUM_TFC_TICKERS];

void TFC_InitSysTick()
{
      uint8_t i;

      for(i=0;i<NUM_TFC_TICKERS;i++)
            TFC_Ticker[i] = 0;

      SYST_RVR = CORE_CLOCK/SYSTICK_FREQUENCY;
      SYST_CSR = SysTick_CSR_ENABLE_MASK | SysTick_CSR_TICKINT_MASK |
SysTick_CSR_CLKSOURCE_MASK;

//Important!  Since the Systick is part of the Cortex core and NOT a
kinetis peripheral its Interrupt line is not passed through NVIC.   You
need to make sure that the SysTickIRQ function is poopulated in the vector
table.  See the kinetis_sysinit.c file
}
```

Elle :

1. initialise un tableau de 4 entiers sur 32 bits déclarés dans le même fichier (le tableau étant déclaré en dehors du main il est donc global et accessible à toutes les fonctions)
2. indique la fréquence du quartz utilisé pour cadencer le processeur (48 Mhz ici)
3. autorise les interruptions timer : quand une interruption Systick surviendra, le processeur cherchera dans sa table de vecteur d'interruption l'adresse de la fonction d'interruption associée. Cette table est configurée dans le fichier kinetis_sysinit.c du répertoire /Project_Settings/Startup_Code. On y découvre que la fonction qui sera appelée s'intitule « SysTick_Handler »

```
4. void SysTick_Handler()
```

```
5. {
6.       uint8_t i;
7.
8.       if(DelayTimerTick<0xFFFFFFFF)
9.       {
10.                  DelayTimerTick++;
11.      }
12.
13.      for(i=0;i<NUM_TFC_TICKERS;i++)
14.                  if(TFC_Ticker[i]<0xFFFFFFFF)
15.                        TFC_Ticker[i]++;
16. }
```

A chaque interruption (donc chaque milleseconde), la variable globale « DelayTimerTick » qui sert de compteur est incrémentée sauf si la valeur maximum du compteur est atteinte.
De cette façon, grâce à la fonction suivante, il est possible de compter le temps :

```
void TFC_Delay_mS(unsigned int TicksIn_mS)
{
      DelayTimerTick = 0;

      while(DelayTimerTick<TicksIn_mS)
      {
      }
}
```

En plus de cela, les 4 éléments du tableau TFC_Ticker sont aussi incrémentés à chaque tick ce qui permet de disposer de 4 compteurs individuels supplémentaires de temps que l'on pourra utiliser plus tard dans les fonctions.

PIT

Le PIT (Periodoc Interrupt Timer) est un périphérique timer spécifique de la famille Kinetis de NxP. Il n'est pas capable de piloter des broches du processeurs, mais en revanche il peut piloter des périphériques interne et générer des interruptions. Nous verrons par exemple qu'il sera utilisé pour déclencher périodiquement les mesures de tension electrique (batterie, potentiomètres et caméra) mais aussi pour régler le temps d'exposition de la caméra.

Pour des raisons que nous verrons plus tard (mode 1), le PIT est initialisé (en rouge) dans la fonction qui initialise l'ADC :

```
void TFC_InitADCs()
{

       InitADC0();
```

```
      //All Adc processing of the Pots and linescan will be done in the
ADC0 IRQ!
      //A state machine will scan through the channels.
      //This is done to automate the linescan capture on Channel 0 to
ensure that timing is very even
      CurrentADC_State =      ADC_STATE_INIT;

    //The pump will be primed with the PIT interrupt.  upon
timeout/interrupt it will set the SI signal high
      //for the camera and then start the conversions for the pots.

      //Enable clock to the PIT
      SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;

      //We will use PIT0
      TFC_SetLineScanExposureTime(TFC_DEFAULT_LINESCAN_EXPOSURE_TIME_uS);
      //enable PIT0 and its interrupt
      PIT_TCTRL0 = PIT_TCTRL_TEN_MASK | PIT_TCTRL_TIE_MASK;

      PIT_MCR |= PIT_MCR_FRZ_MASK; // stop the pit when in debug mode
      //Enable the PIT module
      PIT_MCR &= ~PIT_MCR_MDIS_MASK;

      enable_irq(INT_PIT-16);
      enable_irq(INT_ADC0-16);



}
```

Avec la fonction qui configure simplement la période des interruptions générées par le PIT :

```
void TFC_SetLineScanExposureTime(uint32_t  TimeIn_uS)
{
    float t;
      //Figure out how many Pit ticks we need for for the exposure time
            t = (TimeIn_uS /1000000.0) * (float)(PERIPHERAL_BUS_CLOCK);
            PIT_LDVAL0 = (uint32_t)t;

}
```

Ensuite la fonction **`PIT_IRQHandler()`** est exécuté à chaque interruption de ce timer :

Etude du Mode 0 : test des switchs et leds

Rappelons que dans ce mode, les leds reflètent l'état des switchs et DIP switch.

```
                  case 0 :
                        //Demo mode 0 just tests the switches and LED's
                        if(TFC_PUSH_BUTTON_0_PRESSED)
                              TFC_BAT_LED0_ON;
                        else
                              TFC_BAT_LED0_OFF;
```

```
if(TFC_PUSH_BUTTON_1_PRESSED)
      TFC_BAT_LED3_ON;
else
      TFC_BAT_LED3_OFF;


if(TFC_GetDIP_Switch()&0x01)
      TFC_BAT_LED1_ON;
else
      TFC_BAT_LED1_OFF;

if(TFC_GetDIP_Switch()&0x08)
      TFC_BAT_LED2_ON;
else
      TFC_BAT_LED2_OFF;

break;
```

TFC_PUSH_BUTTON_0_PRESSED : ce n'est pas une fonction mais une macro ! Il faut donc chercher sa défintion. On verra qu'elle interroge le matériel pour connaitre l'état du bouton poussoir. Cette macro est codée dans le fichier TFC-BoardSupport .h

#define TFC_PUSH_BUTTON_0_PRESSED ((GPIOC_PDIR&TFC_PUSH_BUTT0N0_LOC)>0)

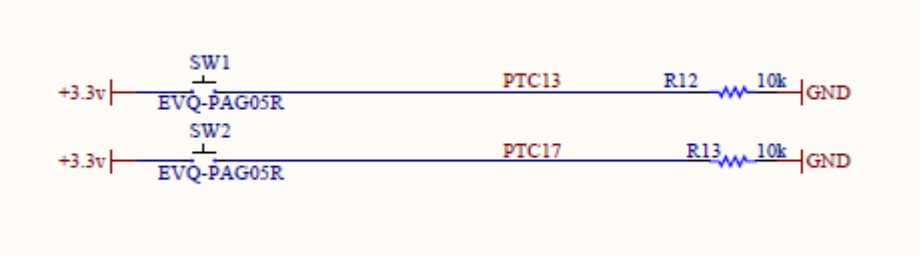
Lors de la compilation cette macro sera donc remplacée par :
(GPIOC_PDIR&TFC_PUSH_BUTT0N0_LOC)>0

Pour comprendre cette ligne, il faut :

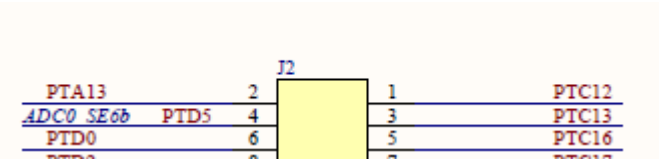
- Le schéma de la carte fille où apparait le bouton poussoir : **FRDM-TFC_Schematics.pdf**
- Le schéma de la carte FRDK-KL25Z pour savoir sur quelle broche est connecté le bouton poussoir : **FRDM-KL25Z_SCH_REV_E.pdf**
- La datasheet du processeur pour connaitre le rôle de cette broche et le périphérique associé : **KL25P80M48SF0.pdf**
- Le référence manuel du processeur KL25Z pour savoir comment accéder à l'état de cette broche (liste des registres à configurer etc.) : **KL25P80M48SF0RM.pdf**

Jeu de piste :

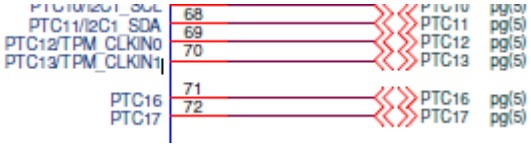
1. On part du schéma de la carte fille de puissance (page 1 de : FRDM-TFC_Schematics.pdf)
2. On cherche le bouton poussoir. Il est nommé SW1.



3. On découvre qu'il est connecté au fil nommé PTC13.



4. On retrouve ce fil sur la broche 3 du connecteur J2 de la carte de puissance.
5. Quand cette carte est enfichée sur la carte KL25Z on découvre (p5 de FRDM-KL25Z_SCH_REV_E.pdf) que ce fil conserve le même nom (merci aux ingénieurs NxP qui sont très rigoureux !) et conduit à la broche numéro 70 de nom « PTC13/TPM_CLKIN1 ».



6. En cherchant cette broche sur la data sheet page du processeur, on découvre (p48 de KL25P80M48SF0.pdf et dans la colonne « 80 LQFP » puisque c'est le suffixe de la référence de notre processeur soudé sur cette carte, cela correspond au nombres de broches) que cette broche peut être configurée pour l'une des deux fonctions : PTC13 (colonne ALT1) ou TPM_CLKIN1 (colonne ALT 4), bien sûr on aurait pu s'en douter d'après le nom donné sur cette broche, mais croyez-moi il est toujours préférable de vérifier !!).

80 LQFP	64 LQFP	48 QFN	32 QFN	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	
64	52	40	28	PTC7	CMP0_IN1	CMP0_IN1	PTC7	SPI0_MISO			SI
65	53	—	—	PTC8	CMP0_IN2	CMP0_IN2	PTC8	I2C0_SCL	TPM0_CH4		
66	54	—	—	PTC9	CMP0_IN3	CMP0_IN3	PTC9	I2C0_SDA	TPM0_CH5		
67	55	—	—	PTC10	DISABLED		PTC10	I2C1_SCL			
68	56	—	—	PTC11	DISABLED		PTC11	I2C1_SDA			
69	—	—	—	PTC12	DISABLED		PTC12			TPM_CLKIN0	
70	—	—	—	PTC13	DISABLED		PTC13			TPM_CLKIN1	

7. En effet, en lisant le référence manuel (KL25P80M48SF0RM.pdf) et en particulier le chapitre 11 « Port control and interrupts) , on découvre qu'il est possible de choisir le périphérique auquel on veut connecter la broche en fonction de la valeur « ALT » qu'il faut placer dans les bits 10 à 8 du registre « Pin Control Register n », communément appelé « PORTx_PCRn » (p184).

Reserved	This read-only field is reserved and always has the value 0.
10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific).

8. Toujours d'après le référence manuel, on découvre que PTC13 correspond au rôle de broche d'entrée-sortie à usage générale : GPIO. Les broches sont regroupées par paquets de 16, pour former des ports : port A, port B, port C etc. (le nombre de port dépend du microcontrôleur, de son boitier etc.).
9. Le switch SW1 est donc connecté au port C, bit n°13 !

Pour connaitre l'état du switch il faut donc lire l'état du registre GPIO associé à cette broche. C'est très simple sur cette famille de processeur car en lisant le registre « Port Data Input Register » (p 777), appelé communément « GPIOx_PDIR) on obtient l'état de chaque bit du port correspondant.

Dans notre cas, puisque SW1 est sur la broche 13 du port C il suffit de lire le bit n°13 (à l'aide d'un masque) le registre GPIOC_PDIR :

41.2.5 Port Data Input Register (GPIOx_PDIR)

NOTE

Do not modify pin configuration registers associated with pins not available in your selected package. All un-bonded pins not available in your package will default to DISABLE state for lowest power consumption.

Address: Base address + 10h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PDI																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIOx_PDIR field descriptions

Field	Description
31–0 PDI	Port Data Input Reads 0 at the unimplemented pins for a particular device. Pins that are not configured for a digital function read 0. If the Port Control and Interrupt module is disabled, then the corresponding bit in PDIR does not update. 0 Pin logic level is logic 0, or is not configured for use by digital function. 1 Pin logic level is logic 1.

En écrivant donc a=GPIOC_PDIR& (1<<13) ; la valeur de a sera l'état du switch.

Comme on évite de coder des valeurs en « dur », les ingénieurs qui ont conçu ce code, ont déclaré dans un fichier spécifique à la carte (TFC_BoardSupport.c), la position de ce bit grâce à la ligne

***#define** TFC_PUSH_BUTT0N0_LOC ((uint32_t)(1<<13))*

Maintenant on peut donc comprendre toute la ligne :

#define TFC_PUSH_BUTTON_0_PRESSED ((GPIOC_PDIR&TFC_PUSH_BUTT0N0_LOC)>0)

Oui mais….ceci fonctionne car la broche PTC13 à correctement été initialisée en entrée et pour le mode ALT1. Ceci a en effet été effectué dans la fonction « `void TFC_Init()` » qui appelle la fonction `TFC_InitGPIO();`

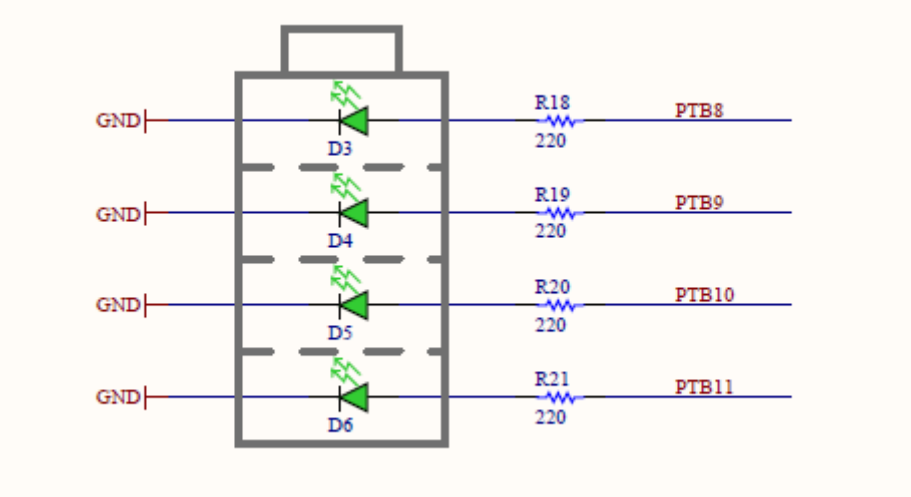
Dans cette fonction on retrouve en effet les lignes suivante : «PORTC_PCR13 = PORT_PCR_MUX(1); »
Avec :
#define PORTC_PCR13 PORT_PCR_REG(PORTC_BASE_PTR,13)
#define PORT_PCR_MUX(x)
(((uint32_t)(((uint32_t)(x))<<PORT_PCR_MUX_SHIFT))&PORT_PCR_MUX_MASK

Notons que par défaut les broches d'un port sont configurées en entrées après un reset (page 778, section 41.2.6) , donc rien à effectuer ici.

La suite du code du mode 0 repose sur le même principe. Il suffit de regarder sur les schémas où sont connectés les DIPs switchs, les leds.

Notons cependant que pour les leds il faut configurer les broches en sortie à l'aide du registre GPIOn_PDDR, avec n=B puisque les leds sont sur les bits 8 à 11 du port B., d'où la ligne suivante
GPIOB_PDDR = TFC_BAT_LED0_LOC | TFC_BAT_LED1_LOC | TFC_BAT_LED2_LOC | TFC_BAT_LED3_LOC;

Leds sur le schema de la carte fille :



Connection des leds aux processeur

Configuration des multiplexeur pour commander les leds en mode GPIO :

80 LQFP	64 LQFP	48 QFN	32 QFN	Pin Name	Default	ALT0	ALT1	ALT2	ALT3
32	28	—	—	PTA12	DISABLED		PTA12		TPM1_CH0
47	—	—	—	PTB8	DISABLED		PTB8		EXTRG_IN
48	—	—	—	PTB9	DISABLED		PTB9		
49	—	—	—	PTB10	DISABLED		PTB10	SPI1_PCS0	
50	—	—	—	PTB11	DISABLED		PTB11	SPI1_SCK	

D'où les lignes de configuration

```
//Ports for LEDs
PORTB_PCR8 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK;
PORTB_PCR9 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK;
PORTB_PCR10 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK;
PORTB_PCR11 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK;
```

Mode 1 : test du servomoteur de direction

Rappelons que ce mode permet de tester les servomoteurs grâce aux potentiomètres.

Voici le code :

```
case 1:
//Demo mode 1 will just move the servos with the on-board potentiometers
      if(TFC_Ticker[0]>=20)
      {
            TFC_Ticker[0] = 0; //reset the Ticker
                              //Every 20 mSeconds, update the Servos
            TFC_SetServo(0,TFC_ReadPot(0));
            TFC_SetServo(1,TFC_ReadPot(1));
      }

      //Let's put a pattern on the LEDs
      if(TFC_Ticker[1] >= 125)
      {
            TFC_Ticker[1] = 0;
            t++;
            if(t>4)
```

```
        {
              t=0;
        }
        TFC_SetBatteryLED_Level(t);
}

TFC_SetMotorPWM(0,0); //Make sure motors are off
TFC_HBRIDGE_DISABLE;

break;
```

Il contient deux parties :

Une partie qui lit l'état des potentiomètres grâce à la fonction ReadPot(x) où x = 0 ou 1 en fonction du potentiomètre à lire, puis applique la valeur de consigne lue au servo-moteur grâce à la fonction TFC_SetServo dont le premier argument est le numéro de servo (on en utilise qu'un, le 0) et le second argument la valeur de position à appliquer au servo.

Une partie qui affiche un chenillard sur les 4 leds. Cela s'effectue à l'aide d'une boucle et d'un compteur t. Cette partie utilise la fonction TFC_SetBatteryLED_Level(x) où x est le numéro de la led à allumer.

On remarque que dans ce mode les moteurs de propulsion sont désactivés (voir explication mode suivant).

Notons que ce code utilise intensivement le tableau de compteur de temps « TFC_Ticker[4] ». Grâce à la ligne `TFC_Ticker[0] = 0;` on remet le compteur 0 à zéro, puis on attend que 20 ms se soient écoulées pour recommencer : `if(TFC_Ticker[0]>=20)…`

Pour le chenillard à led, c'est un deuxieme compteur de temps TFC_Ticker[1] qui est utilisé.

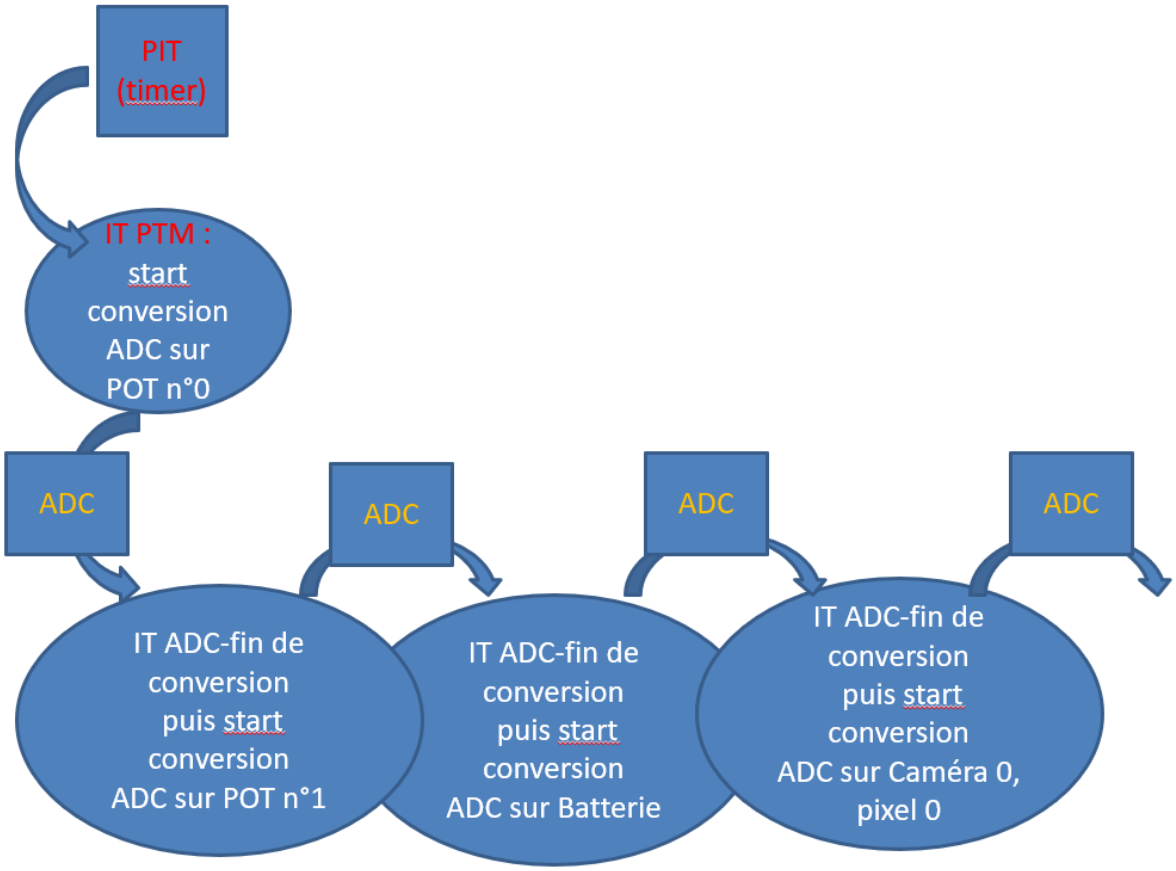
Configuration de l'ADC

Dans ce mode la valeur des potentiomètres est lue grâce à l'ADC (Analog to Digital Converter), mais il est aussi périodiquement utilisé pour :

- Lire l'état de la batterie
- Lire la position des 2 potentiomètres
- Lire les 128 pixels des 2 caméras linéaires.

Pout cela le PIT (périphérique Timer) génère une interruption (IT) périodiquement. Chaque interruption déclenche le démarrage d'une conversion ADC (lecture de la tension du potentiomètre 0).

Quand l'ADC a fini la conversion, il génère une IT de fin de conversion qui démarre une nouvelle conversion sur une autre entrée de l'ADC. Le changement d'entrée et le lancement de la conversion se font grâce à une machine à état codée dans l'interruption de fin de conversion ADC.



Voici le code de la fonction d'initialisation de l'ADC « TFC_InitADCs() »

```
void TFC_InitADCs()

{

        InitADC0();

       //All Adc processing of the Pots and linescan will be done in
       the ADC0 IRQ!

       //A state machine will scan through the channels.

       //This is done to automate the linescan capture on Channel 0 to
       ensure that timing is very even
```

```
        CurrentADC_State =ADC_STATE_INIT;

         //The pump will be primed with the PIT interrupt.  upon
        timeout/interrupt it will set the SI signal high

        //for the camera and then start the conversions for the pots.

        //Enable clock to the PIT

        SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;

        //We will use PIT0

        TFC_SetLineScanExposureTime(TFC_DEFAULT_LINESCAN_EXPOSURE_TIME_
        uS);

        //enable PIT0 and its interrupt

        PIT_TCTRL0 = PIT_TCTRL_TEN_MASK | PIT_TCTRL_TIE_MASK;

        PIT_MCR |= PIT_MCR_FRZ_MASK; // stop the pit when in debug mode

        //Enable the PIT module

        PIT_MCR &= ~PIT_MCR_MDIS_MASK;

        enable_irq(INT_PIT-16);

        enable_irq(INT_ADC0-16);

}
```

Et une partie de la machine à état codée dans l'interruption ADC :

```
#define ADC_STATE_INIT              0
#define ADC_STATE_CAPTURE_POT_0     1
#define ADC_STATE_CAPTURE_POT_1     2
#define ADC_STATE_CAPTURE_BATTERY_LEVEL3
#define ADC_STATE_CAPTURE_LINE_SCAN     4
void ADC0_IRQHandler() {
        uint8_t Junk;
        switch(CurrentADC_State) {
          default:  Junk =  ADC0_RA; break
          case ADC_STATE_CAPTURE_POT_0:
                  PotADC_Value[0] = ADC0_RA;
                  ADC0_CFG2  &= ~ADC_CFG2_MUXSEL_MASK; //Select the A side of the mux
                  ADC0_SC1A  =  TFC_POT_1_ADC_CHANNEL | ADC_SC1_AIEN_MASK;
                  CurrentADC_State = ADC_STATE_CAPTURE_POT_1;
```

```
            break;
        case ADC_STATE_CAPTURE_POT_1:
            PotADC_Value[1] = ADC0_RA;
            ADC0_CFG2  |= ADC_CFG2_MUXSEL_MASK; //Select the B side of the mux
            ADC0_SC1A  =  TFC_BAT_SENSE_CHANNEL| ADC_SC1_AIEN_MASK;
            CurrentADC_State = ADC_STATE_CAPTURE_BATTERY_LEVEL;
            break;
        case ADC_STATE_CAPTURE_BATTERY_LEVEL:
```

……/……

Mode 2 : test des moteurs de propulsion

Rappelons que ce mode permet de tester les 2 moteurs de propulsions à l'aide de la position des 2 potentiomètres.

Voici le code correspondant :

```
                case 2 :

                        //Demo Mode 2 will use the Pots to make the motors
move
                        TFC_HBRIDGE_ENABLE;
                        TFC_SetMotorPWM(TFC_ReadPot(0),TFC_ReadPot(1));

                        //Let's put a pattern on the LEDs
                        if(TFC_Ticker[1] >= 125)
                              {
                                    TFC_Ticker[1] = 0;
                                          t++;
                                          if(t>4)
                                          {
                                                t=0;
                                          }
                                    TFC_SetBatteryLED_Level(t);
                              }
                        break;
```

Comme pour le mode 1, la fonction TFC_ReadPot (x) permet de lire la tension des potentiomètres, et applique cette valeur aux moteurs grâce à la fonction TFC_SetMotorPWM qui reçoit 2 arguments signés correspondants aux consignes de vitesse des 2 moteurs.

En parallèle, comme pour le mode 1, ce code affiche un chenillard sur les 4 leds de batterie.

Mode 3 : test des caméras (mode garage)

```
case 3 :
```

```
//Demo Mode 3 will be in Freescale Garage Mode.  It will beam data from the
Camera to the Labview Application

if(TFC_Ticker[0]>100 && LineScanImageReady==1)
      {
       TFC_Ticker[0] = 0;
       LineScanImageReady=0;
       TERMINAL_PRINTF("L:");
       if(t==0)
            t=3;
       else
            t--;
       TFC_SetBatteryLED_Level(t);

       for(i=0;i<128;i++)
       {
             TERMINAL_PRINTF("%d ",LineScanImage0[i]);
       }

       ../..

      break;
     }
```

On attend que 100 ms soient écoulées (`TFC_Ticker[0]>100) et qu'une image ait été acquise et disponible dans le buffer d'image (LineScanImageReady==1)`

Si c'est le cas : on rafraîchit la position du chenillard de led (code en bleu), et on récupère la valeur de chacun des 128 pixels de l'image et on l'envoie sur le port série vers le PC grâce à la macro TERMINAL_PRINTF.

L'information à retenir : la valeur des pixels de l'image est stockée dans le tableau « LineScanImage0[ ] » . Notons que ce code affiche les pixels d'une seconde caméra qui pourrait être connectée à côté de la première et dont les pixels seraient stockés dans « LineScanImage1[ ] » .

Travail à effectuer

A partir des fonctions étudiées précédemment, il s'agit maintenant de coder l'application de suivi de piste. Cette application doit comporter plusieurs parties :

- Détection des bandes noires latérales
- Calcul de la position de la voiture par rapport aux bandes
- Calcul de consignes à appliquer au servo de direction (pour être parallèle aux lignes)
- Calcul de la consigne de tension à appliquer aux moteurs de propulsion (fonction de la trajectoire, ralentir dans les virages pour ne pas sortir)
- Détection de la fin du parcours.

La vitesse de propulsion est initialement pilotée à l'aide de la valeur de PWM appliquée au moteur. Hors la tension qui va être commutée par PWM n'est pas constante, elle dépend de l'état des batteries, il n'y a donc pas actuellement de consigne en vitesse précise. Le premier objectif est donc d'ajouter un asservissement en vitesse avant de travailler sur l'asservissement en position de la voiture.

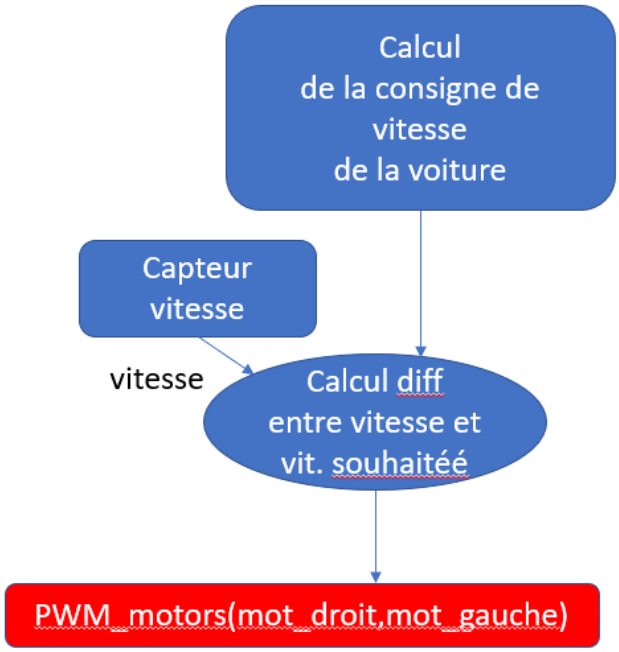
D'autres éléments seront à ajouter pour les phases de tests : capteur d'obstacle ultrasons ou bumper à l'avant pour arrêter la voiture face à un mur. Ajout de module radio.

Asservissement en vitesse : implémentation

Il faut :

- Mesurer la vitesse de la voiture
- Calculer la différence entre la vitesse souhaitée et la vitesse voulue
- Transformer cette valeur en consigne PWM (multiplication par un coefficient, implémentation éventuelle d'un PID)
- Appliquer ce résultat aux moteurs

Voici une représentation graphique de cette asservissement. Sur ce schéma la première boite « Calcul de la consigne de vitesse de la voiture » est fonction de la position de la voiture i.e. est-elle dans un virage, une ligne droite etc.



Pour mesurer la vitesse il existe plusieurs solutions. Nous avons retenu le principe de compter le nombre de tours de roues en un temps fixe donné. Pour cela il faut ajouter un capteur de rotation sur l'une ou les deux roues.

Puisque le règlement impose des capteurs NxP, nous avons utilisé le capteur KMI16/1 « Rotational Sensor » qui est capable de detecter les variations du champ magnétique qu'il génère grâce à un aimant solidaire de son boitier. Nous avons donc fixé une dizaine de petites vis métalliques sur une bague en PLA (fabriquée par imprimante 3D) et fixée autour de l'une des roues de la voiture.

Figure 22 - Bague en PLA noir avec les vis métalliques (à gauche) et capteur KMI16 (à droite)

Le capteur KMI17 génère une impulsion électrique chaque fois qu'une des vis passe devant le capteur.

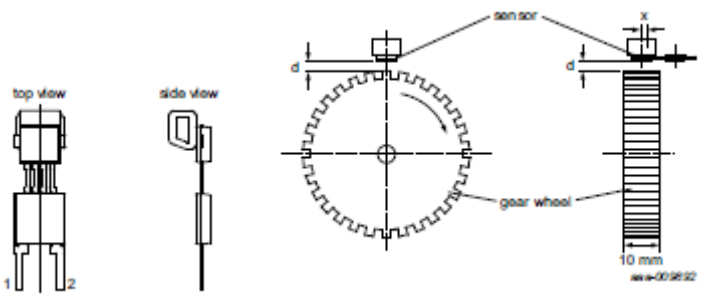
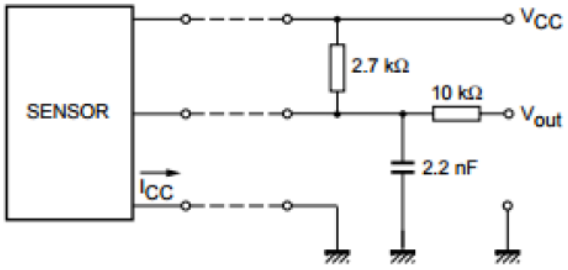


Figure 23-KMI16 Sensor –(Source NxP)

Ce capteur est très facile à interfacer comme le montre le schéma suivant tiré de la note d'application. Nous l'avons utilisé exactement comme préconisé :



Remarque : les moteurs génèrent beaucoup de parasites qui peuvent perturber le comptage, pour les absorber il est important de câbler des condensateurs au plus près des bornes de chaque moteur

Pour convertir ces impulsions en vitesse de rotation il faut en permanence compter le nombre d'impulsion pendant un temps connu. Pour cela, plusieurs techniques logicielles :

- Utiliser le processeur pour faire du polling, c'est-à-dire tester en permanence l'état de la broche d'entrée connectée au capteur : c'est très consommateur de temps CPU puisque le CPU ne peut rien faire d'autre, et on risque de manquer des impulsions si la boucle ne s'exécute pas assez vite.
- Utiliser une interruption matérielle : chaque impulsion sur la broche d'entrée connectée au capteur génère une impulsion. Le processeur exécute le programme d'interruption correspondant à chaque fois que le niveau passe à l'état haut (par exemple). Dans le programme d'interruption on incrémente un compteur puis on quitte l'interruption, le processeur peut ainsi poursuivre l'exécution du travail en cours. Ceci à l'avantage de décharger partiellement le processeur qui peut faire autre chose i.e. il n'est plus monopolisé. Cependant, les interruptions seront très nombreuses ce qui fait perdre beaucoup de temps.
- Utiliser un périphérique dédié du processeur, capable de compter des évènements extérieurs. En général par le biais d'un timer dont l'horloge est le signal du capteur. Chaque constructeur propose une ou plusieurs solutions pour ce problème de comptage assez

classique. Cette solution est la meilleure puisque le processeur n'a plus qu'à interroger périodiquement ou sporadiquement ce périphérique pour connaitre le nombre d'impulsion survenue. Aucun temps processeurs n'est donc gaspillé à compter.

Sur la famille Kinetis, nous avons choisi d'utiliser le périphérique DMA pour réaliser cette tâche. C'est assez inhabituel d'utiliser ce périphérique habituellement dédié à transférer des blocs de données entre 2 espaces mémoires.

En effet un DMA (Direct Memory Access) est conçu pour transférer un bloc de données d'une zone mémoire à une autre, sans intervention du processeur. Il peut même effectuer plusieurs transferts simultanément grâce à plusieurs canaux.

Pour simplifier, la configuration d'un transfert par un canal DMA consiste à programmer quelques registres :

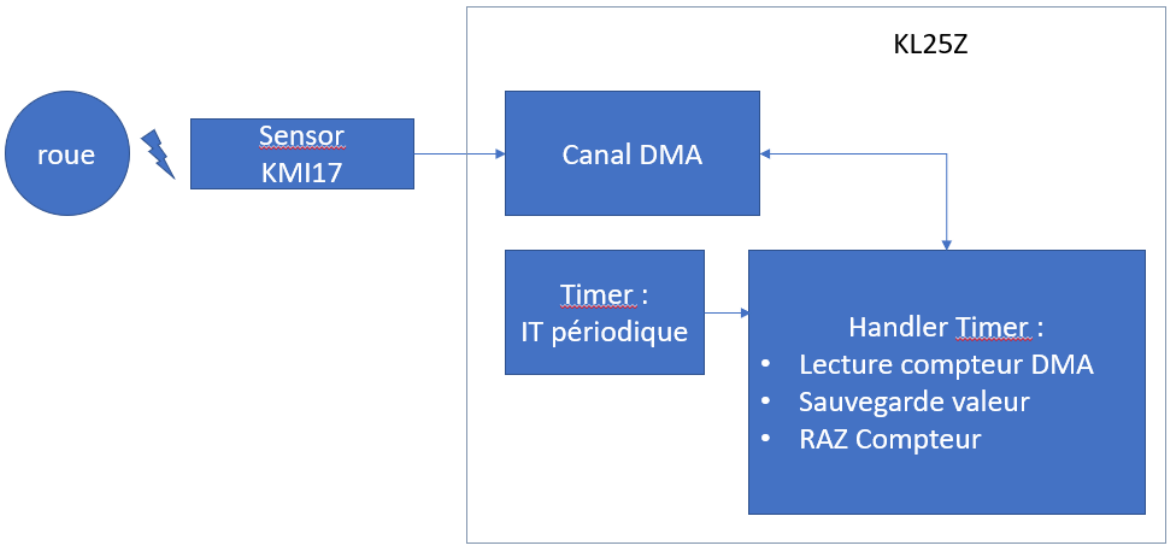
- Un registre où l'on indique l'adresse de base de la zone à transférer
- Un registre où l'on indique l'adresse de destination
- Un registre où l'on indique le nombre d'éléments à copier
- Un registre de configuration du canal DMA.

Parmi les configurations possibles on peut choisir de synchroniser chaque transfert élémentaire avec un évènement interne ou externe. C'est cette dernière possibilité que l'on va choisir.

Ainsi, on va configurer le DMA comme si l'on voulait effectuer le transfert d'un bloc de données, vers une autre zone de donnée en indiquant un très grand nombre d'élément. Puis dans les options on va configurer le DMA pour que chaque transfert soit synchronisé avec le changement d'état de la broche sur laquelle sera connecté le capteur.

A chaque impulsion, le compteur du DMA est donc incrémenté. Il suffit ensuite de lire périodiquement le compteur du DMA pour connaitre le nombre de tour de roues effectués par la voiture, et de remettre le compteur à zéro.

La lecture et la remise à zéro du compteur peuvent se faire dans une interruption exécutée périodiquement, afin d'avoir une mesure de la vitesse précise en permanence.



Asservissement position : implémentation

Temps d'exposition

Nous avons vu que la démo NxP fournit une fonction qui retourne le contenu de la caméra sous la forme d'un tableau de 128 pixels dont l'intensité lumineuse est comprise en 0 et 4096 (utilisation d'un ADC configuré en mode 12 bits). Le 0 correspond à la couleur noir, 4096 au blanc. Toutes les valeurs intermédiaires correspondent à des niveaux de gris entre le blanc et le noir.

Bien sûr, ceci est fonction de l'éclairement, et donc tu temps d'exposition de la caméra. Pour simplifier l'explication, on peut dire que chaque pixel du capteur est un constitué d'un capteur qui « compte » le nombre de photon qui arrivent pendant le temps qu'on appelle « d'exposition ».

Dans cette implémentation, le temps d'exposition correspond au temps qui s'écoule entre deux lectures du capteur : plus on fixe un temps d'exposition élevé, plus on laisse de temps aux photons d'arriver sur le capteur, plus l'image sera claire. C'est donc idéal dans le cas de zones peu éclairées pour augmenter la sensibilité. En revanche, si la scène est déjà très éclairée, les objets sombres risquent d'apparaitre très clairs, il sera donc difficile de les distinguer du fond.

Le temps d'exposition est configurable, il se fixe avec la fonction « `void TFC_SetLineScanExposureTime(uint32_t TimeIn_uS)` » qui reçoit un entier sur 32 bits.

```
void TFC_SetLineScanExposureTime(uint32_t  TimeIn_uS)
{
            float t;

            //Figure out how many Pit ticks we need for for the exposure
time
            t = (TimeIn_uS /1000000.0) * (float)(PERIPHERAL_BUS_CLOCK);
            PIT_LDVAL0 = (uint32_t)t;

}
```

Comme nous l'avons vu précédemment, cette fonction configure simplement la période du PIT qui sert à déclencher l'ADC.

Le réglage de l'exposition est un élément crucial pendant la compétition où les éclairages peuvent être très différents des salles de notre école. Il est donc important de réfléchir à un algorithme de calcul d'exposition idéale et automatique permettant d'avoir l'amplitude de valeur maximale à chaque instant.

Détection des bandes noires

Il existe plusieurs techniques pour détecter les bords des objets dans une image. Nous présentons ici la plus simple basée sur la variation du gradient.

Dans l'image, nous recherchons les bords des bandes noires sur fond blanc. Comme évoqué plus haut, le fond blanc se traduit par des valeurs élevées dans le tableau de pixels alors que les bandes noires se traduisent par des valeurs faibles dans le tableau de pixels. Nous pouvons donc détecter les bords des bandes noires en calculant la variation de luminosité entre 2 pixels consécutifs. Ainsi quand nous sommes dans une zone homogène, la variation de valeur sera faible. En revanche quand nous sommes sur un bord, la variation de la valeur sera beaucoup plus forte.

Par exemple sur cette photo on distingue des valeurs élevées au centre de la courbe (i.e. bande blanche) et des valeurs faibles de chaque côtés (i.e. bandes noires) qui n'atteignent jamais 0.

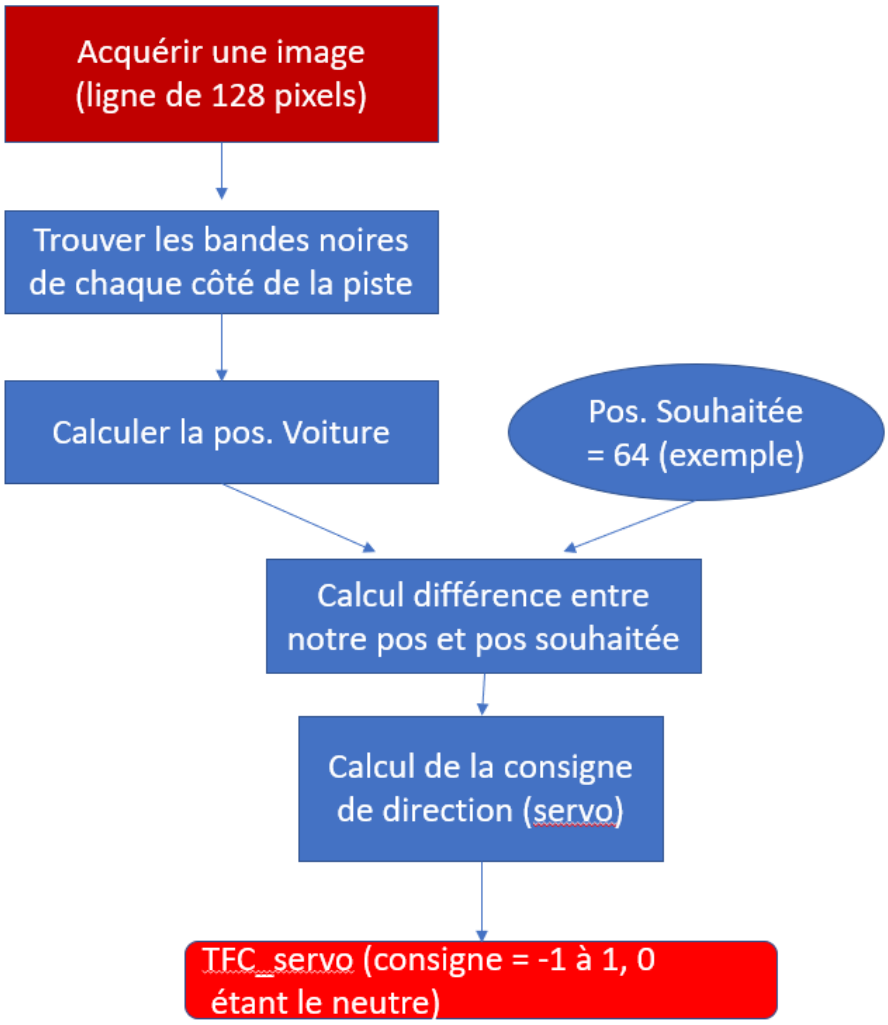
Il faut donc construire un deuxième tableau « gradient » dans lequel on calcul les variations de luminosité entre chaque pixel consécutif : une simple boucle qui parcourt les 128 éléments et fait la différence entre la case courante et la case suivante.

Ensuite, dans ce tableau « gradient » il suffit de rechercher les valeurs maximales puisqu'elles correspondent aux bords des bandes noires. Pour chaque valeur trouvée on conserve sont indice dans le tableau : cet indice correspond à la position physique du bord correspondant.

Remarque : ceci n'est qu'une première possibilité de détection, il en existe d'autres, ont peut par exemple utiliser l'algorithme des K plus proches voisins (K-means)

Pour calculer ensuite la consigne à donner au servo, il faut ensuite avoir décidé à quelle position on souhaite toujours maintenir la voiture du bord i.e. quelle valeur d'indice on souhaite trouver ci-dessus. Il suffit ensuite de faire la différence entre cette valeur et la valeur trouvée, de la convertir en une valeur de consigne pour le servo et l'appliquer au servo. L'utilisation d'un PID peut s'avérer indispensable pour éviter les oscillations de la voiture.

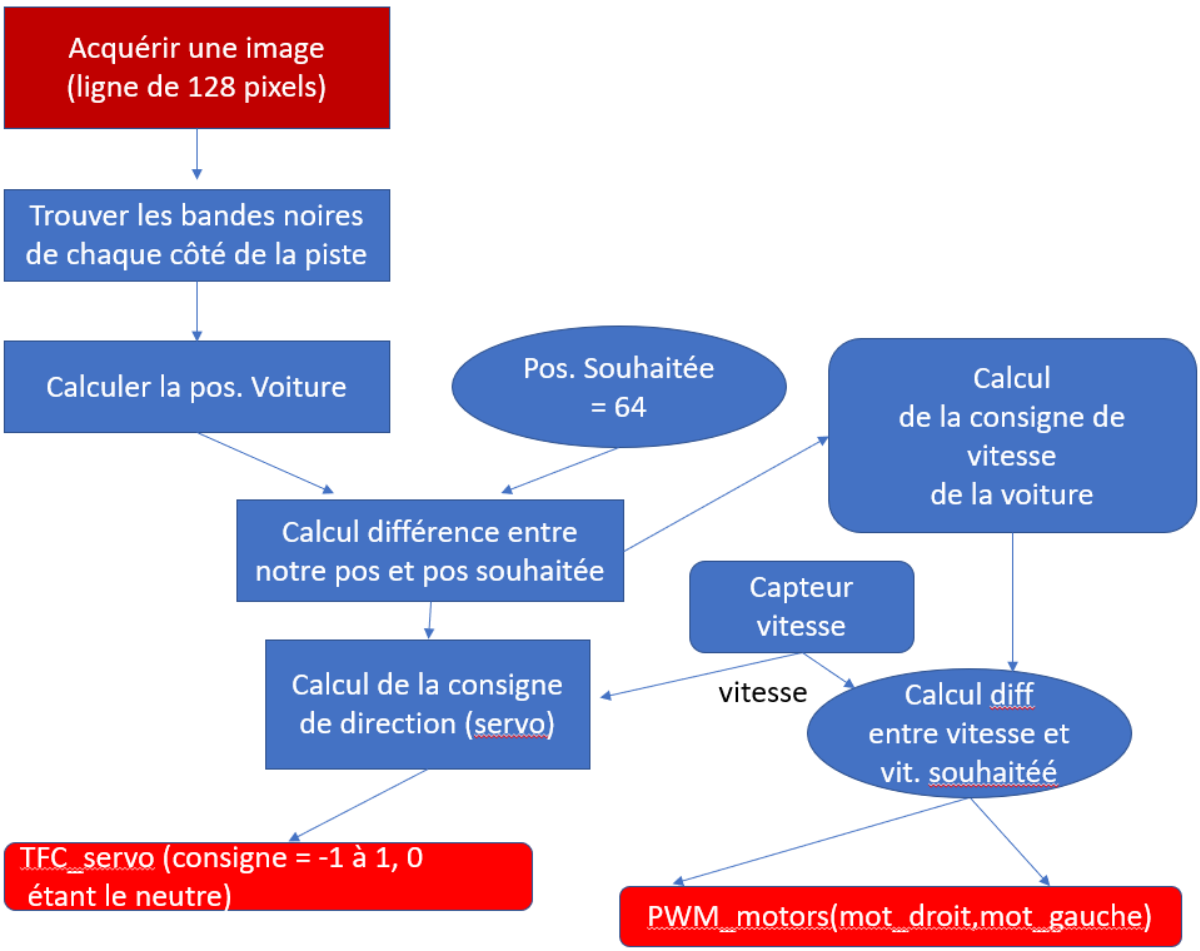
Voici le schéma récapitulatif de cette partie



Fusion des asservissements

Il s'agit ensuite de regrouper l'ensemble des asservissements développés dans une même application. Il faudra aussi ajouter une machine à état qui permet de décider de la vitesse à adopter en fonction de la trajectoire à effectuer pour ne pas sortir des virages par exemple, et accélérer au maximum dans les lignes droites.

Ci-dessous le schéma fonctionnel du travail à réaliser : la partie de droite représente l'asservissement en vitesse, la partie de gauche l'asservissement en position.



Détection de collision

Lors des essais, il est courant que la voiture quitte la piste et finissent bloquée contre un obstacle avec les moteurs alimentés. Ceci est très néfaste pour les engrenages en plastique du chassis Model-C. Il est donc prudent de placer un dispositif qui arretera les moteurs dans cette situation.

Pour cela il existe plusieurs solutions :

- Pare-chocs actif : on dispose des boutons poussoirs fixés sur un pare-choc à l'avant du véhicule.
- Détecteur à ultrasons : on place un émetteur / récepteur à ultrasons pour mesurer la distance entre l'avant du véhicule et les obstacles. Si la distance devient inférieur à une certaine limite on coupe les moteurs .

- Utilisation d'un LIDAR : comme les ultrasons, ce dispositif permet de mesurer une distance mais à l'aide d'un Laser.

La première solution est la plus simple à mettre en œuvre bien qu'elle ait l'inconvénient de ne pas empêcher la collision.

Pour cela il suffit de câbler un bouton poussoir sur une entrée GPIO du processeur. Puis de configurer cette broche pour qu'elle génère une interruption. Dans le handler d'interruption associé on coupera l'alimentation des moteurs.

Il faut donc d'abord configurer la broche qui sera connectée au pare-chocs :

- Identifier une broche disponible
- Configurer la broche en entrée (cas par défaut),
- Configurer le multiplexeur de cette broche en fonction de son rôle (voir tableau pour valeur du MUX)
- Autoriser cette broche à générer des interruptions (registre PORT_PCR_IRQx(n°), PORT_PCR_PE_MASK, PORT_PCR_PS_MASK
- Autoriser les interruptions du port correspondant : enable _irq (INT_PORTn-16)

Il faut ensuite écrire la fonction handler de l'interruption. Remarque : dans ce handler il faut rechercher la broche qui a généré l'interruption car il n'y a qu'une IT par PORT .

Par exemple si l'on a configuré la broche 12 du port C pour qu'elle génère une interruption :

```
int PortC_IRQHandler(int stop)
 {
     uint32_t res=0;
     res = (GPIOC_PDIR) & (1<<12);  // test  de la broche 12 du port C

     ../..

}
```

Annexe 1 – Programme processing pour afficher l'image de la caméra linéaire

1) téléchargez et installer processing https://processing.org/

2) Modifiez la démo fourni par Freescale/NxP (programme FRDM-TFC-R1.0) qui permet de tester les moteur, le servo etc en remplacant le code qui suit (à partir de « case 3 ») par :

```
/*********************************DEBUT*****************************/

            case 3 :

                if(TFC_Ticker[0]>100 && LineScanImageReady==1) {
                     TFC_Ticker[0] = 0;
                     LineScanImageReady=0;
                     //TERMINAL_PRINTF("\r\n");
                     TERMINAL_PRINTF("L:");
                         if(t==0)
                             t=3;
                         else
                             t--;
                         TFC_SetBatteryLED_Level(t);

                         TERMINAL_PRINTF("%d ",0);
                         //TERMINAL_PRINTF("%d ",255);
                         for(i=0;i<128;i++)     {
                                 TERMINAL_PRINTF("%d
",LineScanImage0[i]);
                         }

          }

/*********************************FIN*****************************/
```

3) Dans processing, créer un fichier : fichier -> nouveau, et copier le code ci-dessous (jusqu'en bas de la page)

Remarque : modifier le code en fonction du COM port qui est attribué par windows, dans cet exemple ma carte KL25Z était associé au port COM5 et apparaissait en 2nd dans la liste des ports COM.

```
/**********************DEBUT*************************************/

/*
T.Grandpierre fevrier 2016 - Processing 3
Pour récupérer l'image de la camera freescale KL25Z  et l'afficher
sous forme oscillo + forme image
*/
```

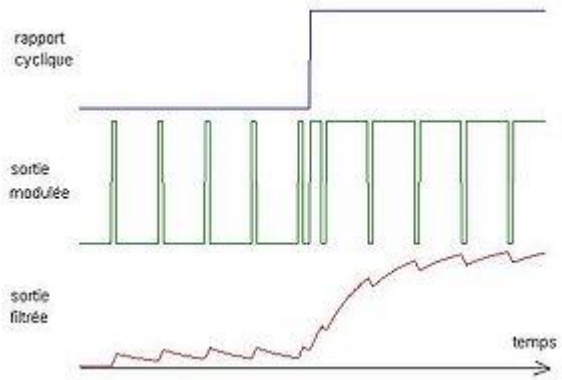
```
import processing.serial.*; // importation de la librairie de
communication serie
//variables
Serial maConnection; // Crée un objet de communication série
int x=0;
int y=0;
int i=0;
int[] ligne0;



void setup() {
    size(650, 650); // taille de la fenetre
    String NomDuPort = Serial.list()[1]; // récupère la 2nd
interface serie trouvée
    //car sur mon PC j'ai COM1 puis COM5 et c'est COM5 qui est
associé à la KL25Z
    println("connection a "+NomDuPort);
    maConnection = new Serial(this, NomDuPort, 115200); // création
de la connexion série
    smooth(); // lisser les dessins
    strokeWeight(1); // largeur de trait
    stroke(40); // couleur du trait gris
    ligne0 = new int[139]; //tableau de pixels qu'on va récuperer
par l'interface serie
     i=0; //compteur de pixels, pour le tableau de pixels
    background(0);
    stroke(255);
}

void draw() { //boucle de dessin principale
    background(0); //fond noir
    noFill(); //pas de remplissage des formes (rectangle)
    stroke(255); //ligne de couleurs blanches
    //dessine forme du signal :
    for (int x=0; x<128;x++) {
            //ligne0[i] contient la valeur du pixel i
            line(x,300-ligne0[x],x+1,300-ligne0[x+1]);
    }
    //dessine image de la caméra
    for (int x=0; x<128;x++) {
          stroke(ligne0[x]); //couleur du pixel
            line(x,310,x,350); //trace une ligne verticale
    }
    stroke(255,0,0); //rouge
    rectMode(CORNERS); //mode rectangle à 2 coordonnees (plutot que
taille)
```

```
    rect (0,45,130,301); //rect rouge
}

void serialEvent (Serial maConnection) { // si des données arrivent
par la connexion série
      String myString = null;
      int pixel; //valeur du pixel camera qu'on va lire sur le port
serie
      myString = maConnection.readStringUntil(32); //lecture sur
port serie
      if (myString !=null) {
        myString = myString.substring(0, myString.length()-1);
//pour retirer l'espace
       if ( myString.equals("L:0")){ //chaine envoyée à chaque
nouvelle image
         //println("debut ligne"+i); //debub : affiche debut ligne
et le compteur de pixels
        i=0;
        x=0;
      } else {
        i++;
        x+=2;
        try {
            pixel = Integer.parseInt(myString);
          }
          catch(NumberFormatException e){
            pixel = 0;
          }
          pixel=pixel/16; //car ADC sur 12bits (MODE_12) donc 0 a
4096, on veux 0-255
          ligne0[i]=pixel;
           //print("pixel["+i+"]="+pixel+" "); //debuf affiche
valeur

          }
      }
}

/************************FIN*************************************/
```

Il ne reste plus qu'à exécuter ce programme dans processing, puis de télécharger et exécuter le fo

Annexe 2 – TPM Module : génération des PWMs moteur de propulsion

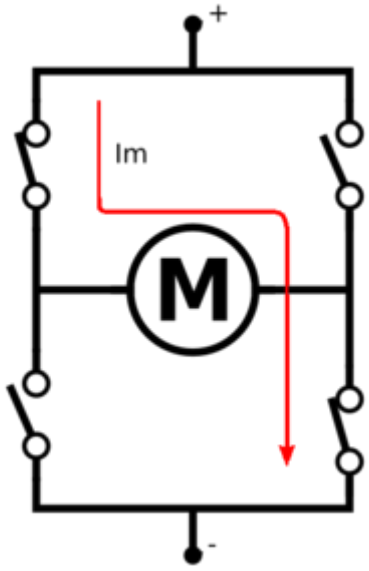
On fait varier la vitesse des moteurs à balais (par opposition aux sans balais/Brushless) en faisant varier leur tension d'alimentation. Pour faire varier une tension à l'aide d'un signal numérique, on utilise le plus souvent la modulation en largeur d'impulsion (MLI ou PWM Pulse Width Modulation).



(source Wikipedia)

Pour cela on génère un signal périodique (à quelques kilohertz généralement) dont le rapport de la durée de l'état haut par rapport à la durée de l'etat bas permet de faire varier la tension moyenne en sortie. Ce signal PWM pilote en général des transistors de puissance qui alimentent à leur tour les moteurs en « hachant » leur tension d'alimentation.

Afin de pouvoir inverser le sens de rotation des moteurs, il faut pouvoir inverser la polarité de l'alimentation aux bornes des moteurs. Pour cela on utilise des « ponts en H ». Ce nom vient de la forme en H qui apparait sur le schéma qui utilise alors au moins 4 transistors en mode commuté.



(source wikipedia)

Au niveau logiciel il faut donc être capable de générer un signal PWM par moteur . Comme le pont en H comporte 2 parties à piloter en opposition de phase il faut finalement générer 2 signaux par moteur.

Pour cela il existe plusieurs techniques qui utilisent plus ou moins le processeur. Dans le cas des Kinetis il existe un périphérique dédié capable de générer ces signaux sans intervention du processeur (autre que sa configuration) : le TPM (Timer/PWM Module) décrit chapitre 31 du référence manual.

The following figure shows the TPM structure. The central component of the TPM is the 16-bit counter with programmable final value and its counting can be up or up-down.

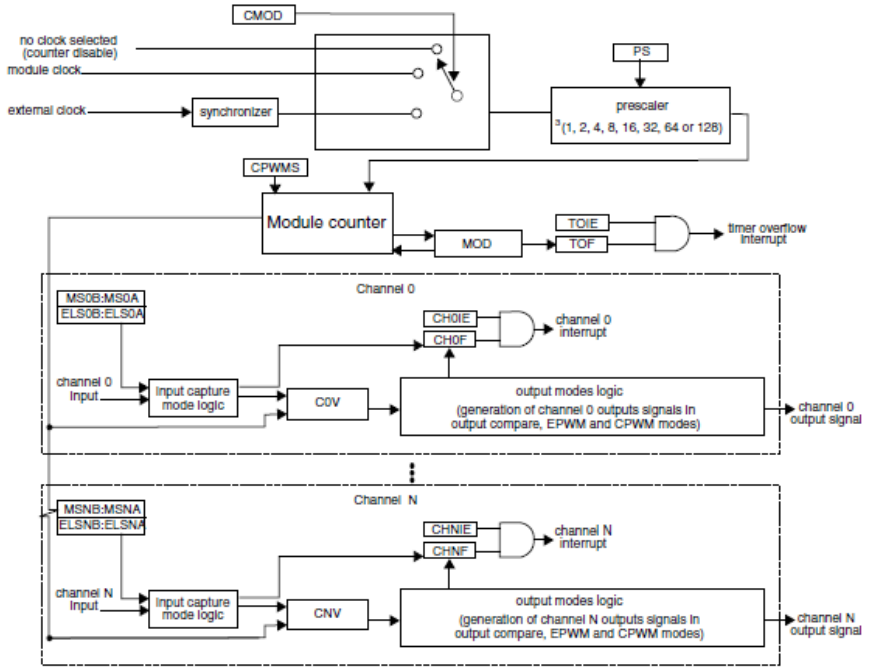


Figure 31-1. TPM block diagram

Dans le KL25Z il existe 3 instances de ce Timer, le TPM0, le TPM1 et le TPM2. Chaque PTM est capable de générer 4 signaux electriques externes indépendants mais utilisant la même base de temps. On parle de 4 canaux : C0, C1, C2 et C3.

Les 4 sorties PWM sont donc nommées : TPMx_C0SC, TPMx_C1SC, TPMx_C2SC, TPMx_C3SC.

Le code fournit par NxP utilise le TPM0 pour générer les signaux de PWM de propulsion. La fonction TFC_InitMotorPWM( ) initialise donc le TPM0 ainsi que les 4 canaux :

- **TPM0_C0SC** = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK;
- **TPM0_C1SC** = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSA_MASK; // invert the second PWM signal for a complimentary output;
- **TPM0_C2SC** = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK;
- **TPM0_C3SC** = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSA_MASK; // invert the second PWM signal for a complimentary output;

Remarque : on a en effet besoin de piloter un pont en H par deux signaux en opposition de phase, les signaux TPM0_C1SC et TPM0_C3SC correspondent donc aux signaux TPM0_C0SC et TPM0_C2SC inversé)

Configuration des broches en sorties, connectées au PTM0 :

- PORTC_PCR1 = PORT_PCR_MUX(4);
- PORTC_PCR2 = PORT_PCR_MUX(4);
- PORTC_PCR3 = PORT_PCR_MUX(4);
- PORTC_PCR4 = PORT_PCR_MUX(4);

On note donc que ce sont les sorties PortC 1, 2, 3 et 4 qui sont dédiées à la commande moteur.

Le reste du code initialise différents registres ainsi que la fréquence PWM (ici 5 khz) :

```
void TFC_InitMotorPWM()
{
      //Clock Setup for the TPM requires a couple steps.


    //1st,  set the clock mux
    //See Page 124 of f the KL25 Sub-Family Reference Manual, Rev. 3,
September 2012
    SIM_SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK;// We Want MCGPLLCLK/2 (See Page
196 of the KL25 Sub-Family Reference Manual, Rev. 3, September 2012)
    SIM_SOPT2 &= ~(SIM_SOPT2_TPMSRC_MASK);
    SIM_SOPT2 |= SIM_SOPT2_TPMSRC(1); //We want the MCGPLLCLK/2 (See Page
196 of the KL25 Sub-Family Reference Manual, Rev. 3, September 2012)


      //Enable the Clock to the FTM0 Module
      //See Page 207 of f the KL25 Sub-Family Reference Manual, Rev. 3,
September 2012
      SIM_SCGC6 |= SIM_SCGC6_TPM0_MASK;

    //The TPM Module has Clock.  Now set up the peripheral

    //Blow away the control registers to ensure that the counter is not
running
    TPM0_SC = 0;
    TPM0_CONF = 0;

    //While the counter is disabled we can setup the prescaler

    TPM0_SC = TPM_SC_PS(FTM0_CLK_PRESCALE);

    //Setup the mod register to get the correct PWM Period

    TPM0_MOD = FTM0_CLOCK/(1<<FTM0_CLK_PRESCALE)/FTM0_OVERFLOW_FREQUENCY;

    //Setup Channels 0,1,2,3
    TPM0_C0SC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK;
```

```
    TPM0_C1SC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSA_MASK; // invert the
second PWM signal for a complimentary output;
    TPM0_C2SC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK;
    TPM0_C3SC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSA_MASK; // invert the
second PWM signal for a complimentary output;


    //Enable the Counter

    //Set the Default duty cycle to 50% duty cycle
    TFC_SetMotorPWM(0.0,0.0);

    //Enable the TPM COunter
    TPM0_SC |= TPM_SC_CMOD(1);


    //Enable the FTM functions on the the port
    PORTC_PCR1 = PORT_PCR_MUX(4);
    PORTC_PCR2 = PORT_PCR_MUX(4);
    PORTC_PCR3 = PORT_PCR_MUX(4);
    PORTC_PCR4 = PORT_PCR_MUX(4);

}
```

Une fois la configuration terminée, le changement de vitesse de rotation des moteurs s'effectue en modifiant la valeur des registres du PTM0 correspondant (en rouge ci-dessous) :

```
void TFC_SetMotorPWM(float MotorA , float MotorB)
{
      if(MotorA>1.0)
            MotorA = 1.0;
      else if(MotorA<-1.0)
            MotorA = -1.0;

      if(MotorB>1.0)
                  MotorB = 1.0;
            else if(MotorB<-1.0)
                  MotorB = -1.0;

      TPM0_C2V = (uint16_t) ((float)TPM0_MOD * (float)((MotorA +
1.0)/2.0));
      TPM0_C3V = TPM0_C2V;
      TPM0_C0V = (uint16_t) ((float)TPM0_MOD * (float)((MotorB +
1.0)/2.0));
      TPM0_C1V = TPM0_C0V;

}
```

Annexe 3 – TPM Module : génération des PWMs pour servo-moteur

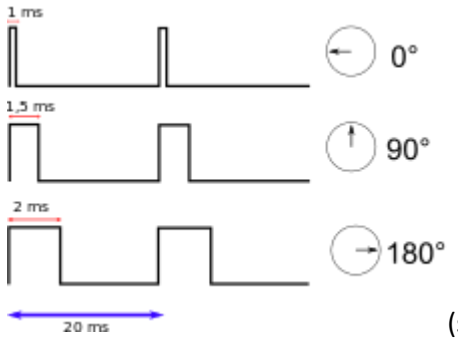
Le servomoteur utilisé dans les chassis Alamak et Model C sont des servomoteurs de modélisme dont la position angulaire est commandée à l'aide d'une PWM. Il est possible de controler la rotation de l'arbre de sortie de ces servos sur une plage de -90° à +90°, la position neutre étant 0°.

On pilote ces servo au moyen de 3 fils : 2 pour l'alimentation (qui peut aller de 5 à 7,2v), et un fil de commande.

Pour placer l'arbre de sortie de ces servos au neutre (0°) il faut envoyer un créneau de 1,5 millisecondes à la période de 20ms.

Pour le faire tourner à droite il faut allonger la durée de ce créneau, le maximum étant un créneau de 2ms pour une rotation maximum de +90°.

Inversement, pour le faire tourner dans l'autre sens il suffit de diminuer la durée du créneau, jusqu'à un minimum de 1ms.



(source Wikipédia)

Comme pour les moteurs de propulsion, nous allons utiliser le périphérique PTM pour piloter les servos. La différence sera principalement dans la fréquence du signal qui sera cette fois beaucoup plus faible puisque 20ms contre 5 khz.

Le principe de configuration reste donc le même, le code utilise cette fois le TPM1 configuré à 50 hz (20 ms). Le code ci-dessous indique que ce sont **les sorties 0 et 1 du port B** qui seront utilisées pour piloter les servos.

```
void TFC_InitServos()
{

 //Clock Setup for the TPM requires a couple steps.
```

```
//1st,  set the clock mux
//See Page 124 of f the KL25 Sub-Family Reference Manual, Rev. 3, September
2012
SIM_SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK;// We Want MCGPLLCLK/2 (See Page 196
of the KL25 Sub-Family Reference Manual, Rev. 3, September 2012)
SIM_SOPT2 &= ~(SIM_SOPT2_TPMSRC_MASK);
SIM_SOPT2 |= SIM_SOPT2_TPMSRC(1); //We want the MCGPLLCLK/2 (See Page 196
of the KL25 Sub-Family Reference Manual, Rev. 3, September 2012)


//Enable the Clock to the FTM0 Module
//See Page 207 of f the KL25 Sub-Family Reference Manual, Rev. 3, September
2012
SIM_SCGC6 |= SIM_SCGC6_TPM1_MASK;

//The TPM Module has Clock.  Now set up the peripheral

//Blow away the control registers to ensure that the counter is not running
  TPM1_SC = 0;
  TPM1_CONF = 0;

  //While the counter is disabled we can setup the prescaler

  TPM1_SC = TPM_SC_PS(FTM1_CLK_PRESCALE);
  TPM1_SC |= TPM_SC_TOIE_MASK; //Enable Interrupts for the Timer Overflow

  //Setup the mod register to get the correct PWM Period

  TPM1_MOD = FTM1_CLOCK/(1<<(FTM1_CLK_PRESCALE+1))/FTM1_OVERFLOW_FREQUENCY;

 //Setup Channels 0 and 1

 TPM1_C0SC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK;
 TPM1_C1SC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK;

 //Enable the Counter

 //Set the Default duty cycle to servo neutral
 TFC_SetServo(0, 0.0);
 TFC_SetServo(1, 0.0);

 //Enable the TPM COunter
 TPM1_SC |= TPM_SC_CMOD(1);

 //Enable TPM1 IRQ on the NVIC
 enable_irq (INT_TPM1-16);

 //Enable the FTM functions on the the port

 PORTB_PCR0 = PORT_PCR_MUX(3);
 PORTB_PCR1 = PORT_PCR_MUX(3);
}
```

La fonction **TFC_SetServo permet ensuite de piloter la position des servos.**

```
//Position is -1.0 to 1.0.   Use SERVO_X_MIN_DUTY_CYCLE and
SERVO_MAX_DUTY_CYCLE  to calibrate the extremes
void TFC_SetServo(uint8_t ServoNumber, float Position)
{
                                TFC_SetServoDutyCycle(ServoNumber ,
                                                                (((Position +
1.0)/2)*(SERVO_MAX_DUTY_CYCLE -
SERVO_MIN_DUTY_CYCLE))+SERVO_MIN_DUTY_CYCLE);

}
```

Annexe 4 – Récapitulatif de l'affectation des broches

Camera 0 et 1 (3.3v sur br 4, gnd vr. 5)

- IA br 1 camera 0 : chanel 6 ADC0 –PTD5 ADC0_SE6b
- IA br 1 camera 1: PTD 6 ADC0_SE7b
- SI br 2 : PTD7 (commun aux 2 cameras)
- Clk br 3 : PTE1(commun aux 2 cameras)

Speed sensor :

- PTA1/FTM2 CH0 br 2 (pullup 3.3)
- PTA2/FTM2 CH1 br 2 (pullup 3.3)

Servos :

- PTB0_FTM1_CH0
- PTB1_FTM0_CH1

Potentiomètres :

- POT0 : PTB3 ADC0_SE13 (100k 3,3v)
- POT1 : PTB2 ADC0_SE12 idem

Batterie :

- pont div. 47k/10k sur PTE29 ADC0_SE4b

Switchs :

- sw1 PTC13, pulldown
- sw2 PTC17 pulldown
- 4 dip-switchs : (pulldown ouvert)
- 1 sur PTE2, PTE3, PTE4, PTE5
- 4 leds : PTB8, PTB9, PTB10, PTB11

Bridge PWM :

- PTC3-FTM0 CH2 = IN1 moteur1
- PTC3-FTM0 CH3 = IN2 moteur 1
- PTE23 ADC0_SE7a = retour courant M1
- PE21 : H bridge enable (commun aux 2)
- PTE20 : Fault
- PTC1 FTM0_CH0 = IN1 moteur 2
- PTC2 FTM0_CH1 = IN2 moteur2
- PTE22 ADC0_SE3 = retour courant M2