Institut National des Sciences Appliquées
135, avenue de Rangueil – 31077 Toulouse cedex 4 - France

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE

DEPARTEMENT GENIE ELECTRIQUE & INFORMATIQUE

Année Universitaire
2011 - 2012

SUPPORT DE COURS

Modulation, Démodulation numérique

ORIENTATION : IR -RT **4ème ANNEE**

AUTEUR : Thierry ROCACHER, Pascal ACCO

Introduction

Ce cours est construit pour servir de support à l'enseignement de ***modulation et démodulation numérique***. L'outil nécessaire et incontournable pour matérialiser ces concepts est le ***microcontrôleur***.

Ainsi, il est nécessaire d'acquérir la double compétence :

- Compétences dans le domaine de la ***télécommunication*** et du ***traitement de signal*** (connaissance des principes de modulation de base, manipulation du signal dans les domaines temporel et spectral)
- Compétences dans le domaine des ***microcontrôleurs*** (connaissance des périphériques de base, programmation C, mise au point sur cible, exécution temps réel sous interruption)

Afin d'atteindre ces objectifs, deux documents supports sont à votre disposition. Le premier, *« Modulateurs et démodulateurs »*, donne une vue d'ensemble des principales modulations et de leurs mise en oeuvre par des **techniques analogiques**. Le second, *« Modulation et démodulation numérique »*, le présent document, porte plus spécifiquement sur le **micro-contrôleur et ses applications en télécommunication.** On s'intéressera notamment à la démodulation IQ.

Le présent document est découpé en deux parties :

Chapitre 1 : La programmation sur microcontrôleur
Ce chapitre abordera l'étude des principaux périphériques d'un microcontrôleur utilisés dans un contexte de télécommunication. On abordera aussi les principes de base concernant la structure logicielle d'un programme qui s'éxécute sous interruption(s) , en temps réel.

Chapitre 2 : ***La démodulation IQ sur micro-contrôleur***
Cette seconde partie exploite les éléments théoriques vus dans *« Modulateurs et démodulateurs »* et les techniques de mise en oeuvre numérique du précédent chapitre, pour donner des pistes de réflexion visant à opérer une démodulation directe par micro-contrôleur.

Auteur : Thierry ROCACHER, Pascal ACCCO

Sommaire :

Sommaire

Chapitre 1 : La programmation sur microcontrôleur

1. Introduction

Un microcontrôleur est un élément doté d'une CPU (Central Processing Unit) qui lui permet d'exécuter un programme. Son véritable intérêt par rapport à un micro ordinateur, PC par exemple, est qu'il prend place dans des ***applications embarquées***. Par ailleurs, comme son nom l'indique, il est destiné à ***contrôler*** des processus. Ceci implique la présence dans le microcontrôleur d'élements d'échange avec l'extérieur (entrée / sortie) appelé ***périphérique***.

Définition d'un périphérique de microcontrôleur :

Un périphérique est une structure électronique autorisant des échanges entre l'extérieurs du microcontrôleur et la CPU. Parfois, les périphériques sont même capables de communiquer entre eux, sans faire intervenir la CPU (DMA, déclenchement automatique d'un ADC par un Timer...).

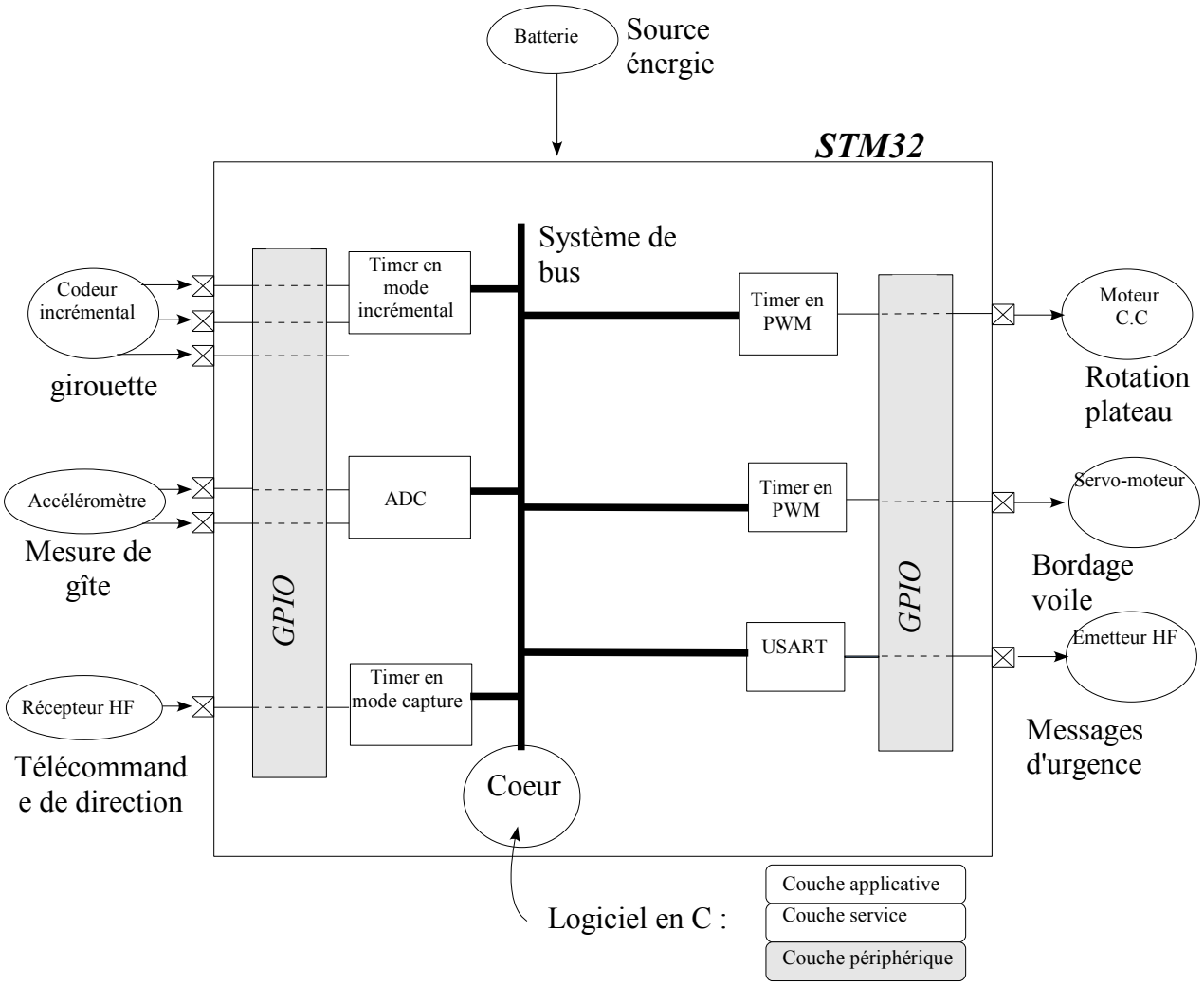
Un périphérique se **configure** avant son utilisation proprement dite. Ainsi, sa structure électronique contient des **registres** adressables par la CPU. On peut faire la distinction suivante sur les registres d'un périphérique :

- Les ***registres de configuration*** : Ce sont ceux qui vont être initialisés, configurés, au démarrage et généralement une seule fois, de manière à faire travailler le périphérique d'une certaine manière. Par exemple, une broche d'un microcontrôleur sera configurée en sortie et en mode « push-pull ».
- Les ***registres d'utilisation*** : Ce sont les registres qui sont utilisés au cours de l'application. Par exemple, la broche configurée au départ en sortie, doit produire une alternance de '1' et de '0' au cours de l'application : un registre d'utilisation sera utilisé pour cela.

2. Les périphériques principaux

- ***Les ports d'entrée/sortie (GPIO) :*** C'est par ces broches que *transitent toutes les communications avec l'extérieur,* sous la forme de tension ('1' = 3.3V environ, '0' = 0V environ).
- ***Les Timers*** : Ce sont des compteurs électroniques qui permettent de construire des *bases de temps* (application temps réel) ou de compter des impulsions extérieures.
- ***Les ADC / DAC*** : Ce sont les interfaces qui rendent possible la manipulation de *tensions analogiques* (au lieu de traiter simplement 2 niveaux, '0' et '1', les ADC et DAC travaillent typiquement sur 256, 1024, ...16 millions de niveaux de tension)
- ***Gestionnaire d'interruption*** : Il permet de définir par exemple, quels sont les périphériques susceptibles de générer une interruption, de départager plusieurs demandes d'interruptions simultannées...

Voici, à titre d'exemple, un microcontrôleur de type *STM32*, dans le contexte d'application du « contrôle de bordage d'une voile sur un bateau » :



On distingue en périphérie les GPIO. Plus à l'intérieur du contrôleur, on trouve les périphériques cités précédemment (sauf le DAC). On peut constater que le périphérique Timer est décliné dans de multiples contextes d'utilisation (*PWM*, *Capture d'impulsions*, *Codeur incrémental*). On voit aussi un circuit appelé *USART* (**U**niversal **S**ynchronuous, **A**synchronuous **R**eceiver **T**ransmetter). Tous sont reliés au CPU (coeur).
Enfin, nous y reviendrons plus tard, on laisse entrevoir une logique dans la structure de programmation : les logiciels seront systématiquement structurés en 3 couches : la *couche applicative*, la couche *service*, la couche *périphérique*.

2.1. Les ports d'entrée / sortie (GPIO General Purpose Input Output)

Un *port d'entrée/sortie* communique avec l'extérieur du micro-contrôleur par le biais de plusieurs fils (broches), en général regroupés par paquets de 8 ou 16. Il communique avec le processeur par sa seule et unique possibilité : les *bus d'adresses* et de *données*. Ceci est commun à TOUS les périphériques du micro-contrôleur.

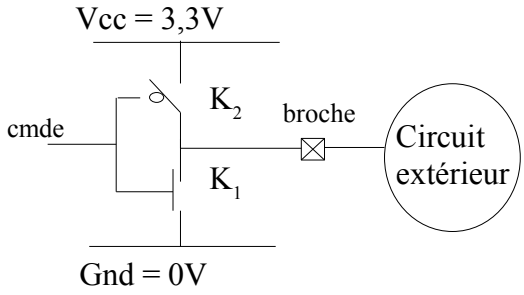
Un port d'entrée / sortie a donc pour rôle *d'imposer (Output)* ou de *lire (Input)* un niveau de tension (associé aux niveaux logique '0' ou '1') sur l'ensemble de ses broches. Selon le microcontrôleur, le niveau logique '1' peut être 5V, 3V3 ou encore 1V8. Pour ce qui nous concerne, les IO sont sensibles à 0/3,3V.

Le port d'E/S possède donc au minimum deux registres de configuration (l'un qui spécifie pour chaque broche sa ***direction***, l'autre spécifiant la ***technologie*** utilisée) et un registre d'utilisation qui est à l'image logique des broches.

Parmi les technologies possibles on trouve :

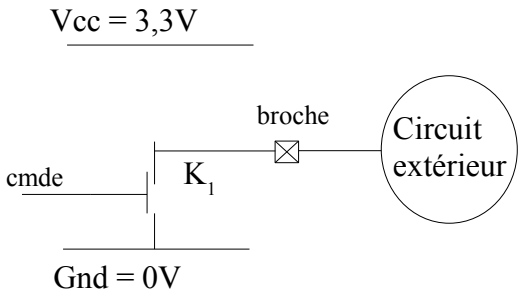
2.1.1. IO en sortie

Mode Push Pull :



La structure (technologie) possède deux interrupteurs (K_1, K_2, des transistors MOS complémentaires). K_1 et K_2 sont systématiquement inversés : La broche peut donc être portée au potentiel 0V ou 3,3V.

Mode Open Drain :



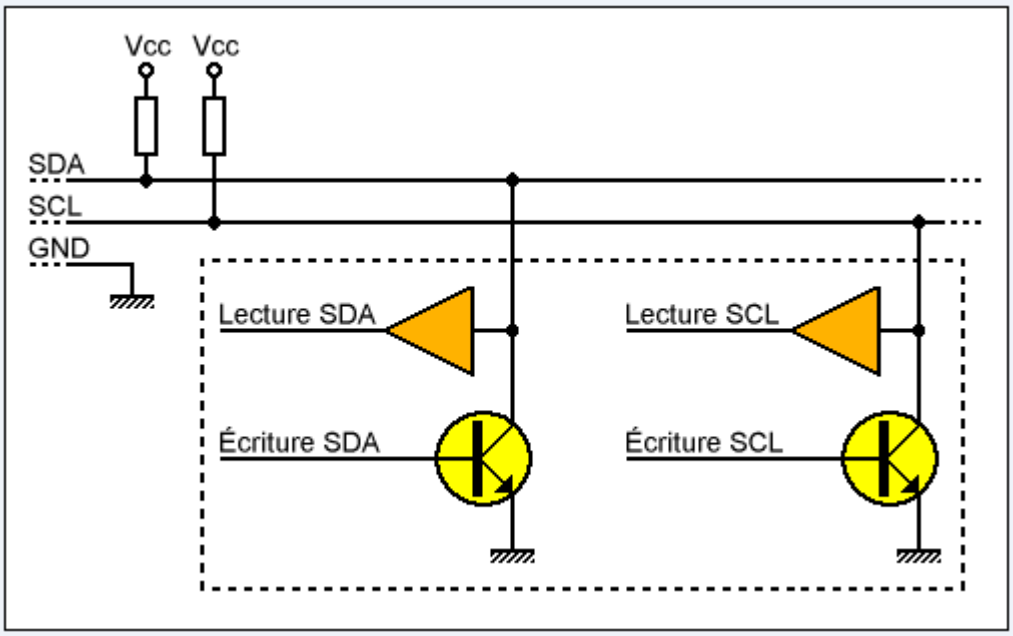
Ici, un seul intérrupteur est commandé, l'autre est maintenu bloqué (on ne le fait pas apparaître): la broche ne peut être portée <u>par le port</u> qu'à une tension de 0V.

Remarquons que si K_1 est ouvert (tentative de mise à '1' par le GPIO), la broche est “en l'air”. Ce sera donc au <u>circuit extérieur</u> de fixer le potentiel de la broche dans cet état précis.

Exemple d'utilité du mode « Open Drain »

Le Bus I2C est un bus qui permet de mettre en communication plusieurs circuits numériques (micro-contrôleur, EEPROM, …). Deux fils sont utilisés *SCL* (horloge), *SDA* (donnée) sans oublier la masse. Il s'agit donc d'un bus série synchrone.

Voici l'exemple typique d'une structure à bus I2C:



Nous voyons ici un seul circuit connecté au bus (intérieur des pointillés), il faut imaginer que n circuits sont susceptibles d'y être reliés de la même manière. Chaque circuit est composé de deux structures de lecture / écriture. En clair, chaque circuit peut prendre la main sur les lignes *SCL* et *SDA* (lecture ou écriture). Plusieurs circuits sont donc susceptibles d'émettre en même temps, c'est à dire de chercher à écrire sur la ligne *SDA* par exemple en même temps. Un conflit a donc lieu aux conséquences potentiellement néfastes sur le plan matériel (destruction) et logiciel (brouillage des messages) . Dans ce dernier cas, un arbitrage de bus permet de régler le problème.

Sur le plan matériel, il n'y a en réalité aucun problème non plus. En effet, les transistors (jaune) sont assimilables à des interrupteurs. On reconnaît donc une structure ***Open Drain***. Ainsi, chaque circuit peut imposer un '0' logique, mais jamais un '1'. C'est le circuit extérieur (la résistance de tirage) qui remplit ce rôle.

Si deux circuits imposent en même temps un '0', ou un '1', il n'y a pas de conflit, et ce, quelque soit la technologie utilisée.

Si par contre, un circuit impose un '0' et l'autre un '1', c'est celui qui impose le '0' qui « l'emporte » car le transistor, en position fermée, est équivalent à une résistance quasi nulle. Le circuit qui proposait '1' (équivalent, lui à Vcc avec une résistance série grande) n'est donc pas compris, mais il ne subit strictement aucun dégât. Si la technologie avait été *push-pull*, alors danc ce cas, un court-circuit se serait produit avec pour conséquence un courant trop fort débité par le circuit qui proposait '1' (ainsi qu'à celui qui proposait '0', c'est le plus fragile qui fait fusible...définitif).

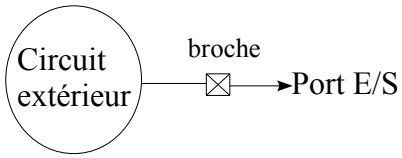
Le montage *Open Drain*, d'un microcontrôleur permet donc à celui-ci de se relier à un bus I2C par exemple.

Résumons :

En mode **Push-Pull**, c'est le **port d'E/S qui impose** le niveau logique d'une broche, **que ce soit un niveau '1' ou '0'**. Il est le maître, le circuit extérieur est l'esclave, récepteur.
En mode **Open Drain** (= *Drain laissé ouvert, Drain* est le nom de la broche du transistor MOS reliée à la broche), le port d'E/S ne peut **imposer que le niveau logique '0'**. Le niveau '1' est fixé par le circuit extérieur. Le port d'E/S n'est donc pas le seul maître du potentiel sur la broche. Le circuit extérieur l'est aussi.

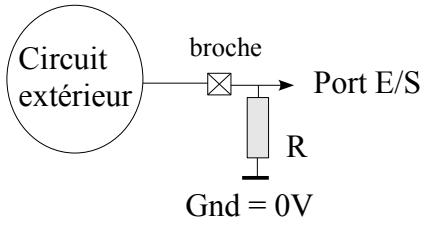
2.1.2. IO en entrée

***Mode entrée flottante (floating input)**:*



La broche, côté du port E/S, est laissée libre, flottante. Ainsi, c'est le circuit extérieur qui est totalement maître du potentiel de la broche.
Cela veut aussi dire, que si le circuit extérieur est déconnecté, le broche possède un potentiel inconnu (à proscrire car favorise le captage de parasites).

***Mode entrée tirée au niveau bas (Pull Down input)**:*



La broche, côté du port E/S, est reliée au 0V par l'intermédiaire d'une résistance (dite de rappel). L'avantage, c'est que si le circuit extérieur est déconnecté, le potentiel de la broche se retrouve à 0V grâce à la résistance de rappel.
Cela veut aussi dire, que le circuit extérieur, pour imposer un potentiel, doit avoir une résistance de sortie faible devant R, sinon, la tension chute. Elle est faussée.

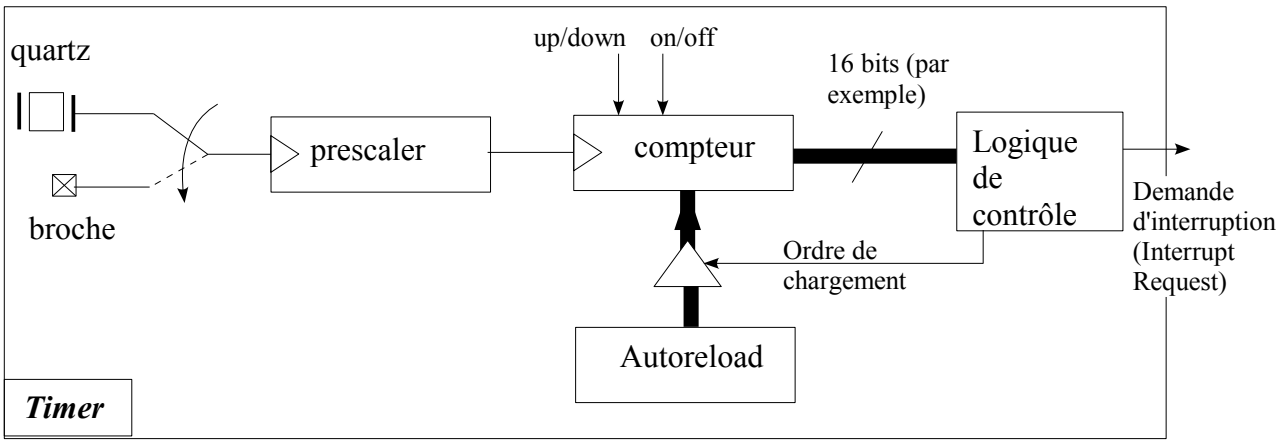
***Mode entrée tirée au niveau haut (Pull up input)**:*
Même principe que précédemment, mais la résistance est reliée au Vcc (5V, 3V3, ou 1V8)

2.2. Le Timer

Le coeur d'un Timer est un ***compteur électronique***. Celui-ci peut avoir une résolution de 8 bits (0 à 255), 16 bits (0 à 65 535) ou encore 32 bits (0 à 4 294 967 295 !).

Ce Timer possède donc bien évidemment une ***horloge***. Selon la nature de l'horloge, on parlera d'un *fonctionnement en Timer* (l'horloge est dérivée d'un quartz de référence, fixe) ou d'un *fonctionnement en compteur* (l'horloge est dérivée d'une broche du microcontrôleur). L'entrée d'horloge est très souvent précédée d'un ***Prescaler***. Son rôle est d'opérer une première division de la fréquence de l'horloge avant d'attaquer concrètement l'horloge du compteur (voir schéma).
Enfin, précisions que le compteur est très souvent associé à un registre dit ***Autoreload***. Celui-ci contient la valeur de redémarrage du compteur après un débordement (haut ou bas).

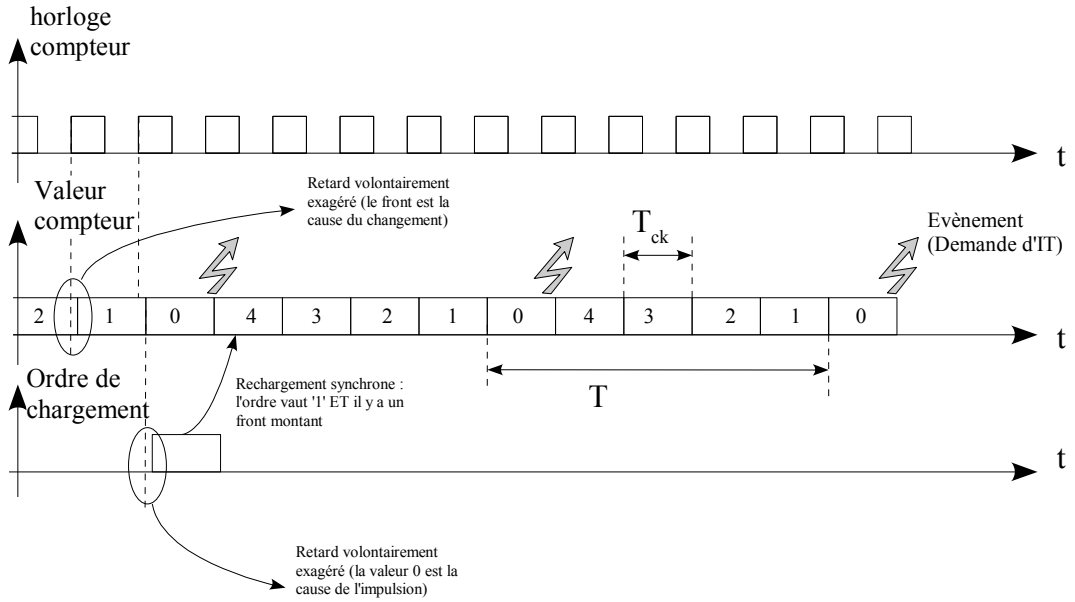
Voici le schéma typique d'un timer dans sa plus simple expression :



Remarque : *La figure précédente est très simpliste, et ne permet d'appréhender le Timer uniquement dans le but de générer un évènement périodique.*

La plupart du temps, le compteur possède une entrée de contrôle *up / down* et le compteur fonctionne de manière purement **synchrone**. Ci-après un chronogramme qui montre le timer en mode décomptage ainsi qu'une analyse de *l'autorelaod*.

On suppose un mode de décomptage (down) et une valeur d'autoreload de 4 et qu'au départ, le compteur a la valeur 2.



Sur cet exemple, la durée T est la période d'évolution de la séquence du Timer. Elle vaut 5 T_{ck}. T_{ck} étant la période d'horloge.
Ainsi, en généralisant, si l'on souhaite une période pour la séquence du Timer de n.T_{ck}, on placera dans le registre T_{ck} la valeur n-1.

La génération de l'évènement (demande d'interruption par exemple, mais pas forcément) se traduit physiquement par une impulsion qui a lieu à l'underflow (décomptage) ou overflow (comptage). L'underflow est le passage de 0 à la valeur suivante (0xFFFF si l'autorelaod n'est pas configuré, valeur de l'autorelaod dans le cas contraire).

Exemple :
On dispose d'une horloge de 40MHz en entrée d'un Timer 16 bits (H_ref). Le Prescaler est sur 16 bits. On veut génerer une impulsion périodique de 0,5s.
Quelle valeur donner au Prescaler, ainsi qu'à l'auto-reload pour parvenir à cela, sachant que l'on veut une fréquence la plus grande possible sur le compteur (cela revient à dire une valeur d'autoreload maximale).

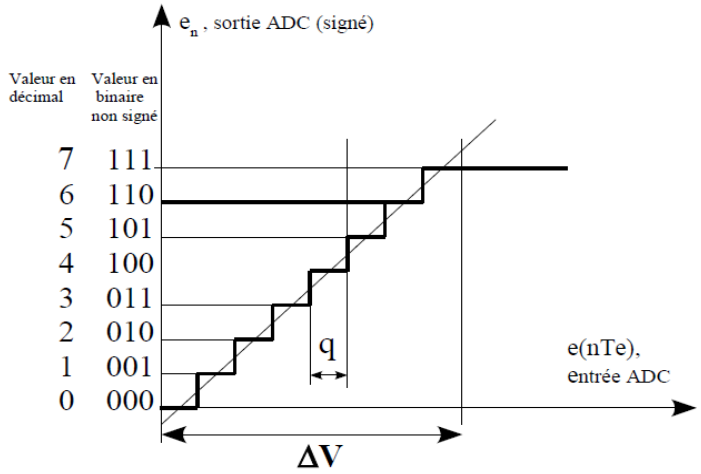
Solution :
La valeur maximale d'autorelaod est de 2^{16} = 65536. La période correspondante en entrée de comptage est donc 0,5s / 65536 = 7,629µs.
A partie d'une fréquence de 40MHz, le Presceler doit donc donner une fréquence de 1/7,629 µs = 131 072 Hz.
Le Prescaler vaudra donc P = 40MHz / 131 072 Hz =305,17.
Il faut arrondir à l'entier supérieur. En effet, dans le sens inverse (arrondi inférieur) la fréquence en entrée de comptage serait trop forte => Autorelaod saturé.
Prenons donc P=306 => Freq_Cpt = H_ref / P.
La valeur de l'autoreload est telle que : Auto_Rel*T_Cpt = 0,5s => Auto_Rel/ Freq_Cpt = 0,5s soit Auto_Rel = 0,5* Freq_Cpt = 0,5 * H_ref / P
AN: Auto_Rel = 0,5*40M/306 = 65359,477, arrondi à 65359, ce qui donne une période exacte de 499,99635 ms, à la précision du quartz près !
Finalement, pour le réglage on placera 306 -1 = 305 dans le prescaler et 65359-1 = 65358 dans l'Autorelaod, eu égard aux comptage synchrone expliqué ci-dessus

2.3. L'ADC (Analog To Digital Converter)

2.3.1. Description, vocabulaire

Un ADC a pour rôle de fournir un **nombre entier** à **l'image d'une tension**. Une tension possède par définition une infinité de valeurs possibles (grandeur analogique), alors que le nombre entier fourni, lui, est forcément limité en quantité de valeurs.

La caractéristique typique d'un ADC est la suivante (ADC 3 bits) :



Le **quantum d'un ADC** est l'incrément de tension **q,** qui fait évoluer le nombre entier d'une valeur à la suivante.
L'**étendue de mesure** est la plage d'entrée de l'ADC, noté **ΔV** sur la caractéristique.

La **résolution d'un ADC** renseigne plus ou moins directement sur la **quantité de valeurs entières** que peut prendre le nombre de sortie de l'ADC. Les données suivantes, bien que différentes, renseignent sur la résolution :

- ADC 1024 points
- ADC 10 bits
- ADC dont l'étendue de mesure est 5V et le quantum vaut 4,883mV

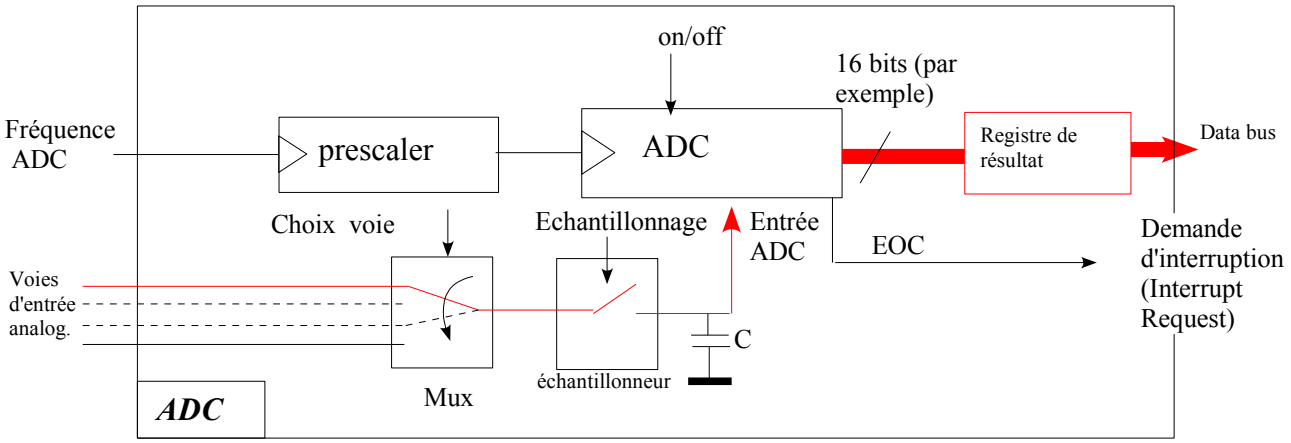
} 3 manières de donner la résolution d'un même ADC

La **précision d'un ADC** dénote l'aptitude de l'ADC à être fidèle. Autrement dit, elle renseigne sur l'alignement de la caractéristique réelle avec la caractéristique théorique en forme d'escalier (dont les quantums sont parfaitement réguliers).

Résolution et *précision* sont parfois confondues. Un dernier exemple qui permet de bien faire la différence, est celui d'un voltmètre. En effet, un voltmètre ayant une *résolution* de 2000pts, est un appareil dont l'afficheur LCD possédant les 4 digits, mais ne pouvant indiquer que des valeurs de 0 à 1999 (la virgule se déplaçant au gré des calibres).
La *précision*, indique le degré d'exactitude de la valeur annoncée. Si par exemple on mesure 10.05V

avec le voltmètre, alors que la tension réelle vaut 10,09V, la précision sera donc de 0,04/10,09=0,4%. Dans cet exemple, l'ADC est d'une précision insuffisante vis à vis de la résolution. Habituellement, un système de mesure discret (l'ADC en est un) possède une précision cohérente vis à vis de la résolution. C'est à verifier en toute rigueur.

2.3.2. Structure générale, et fonctionnement d'un périphérique ADC



Sur le synoptique ci-dessus, le chemin en rouge représente le flow d'information, analogique au départ, numérisé à la fin.
Un périphérique ADC possède quasi systématiquement un nombre de voies analogiques supérieur à un (typiquement jusqu'à 16), ainsi qu'un multiplexeur associé qui permet de faire le choix (via un registre de configuration).
Le coeur du périphérique est bien entendu le module ADC. Quelque soit sa technologie, il nécessite une horloge pour fonctionner, c'est un système séquentiel synchrone. Actuellement, le type d'ADC le plus couramment rencontré dans les micro-contrôleurs est l'***ADC à pesées successives***.

Typiquement, le périphérique possède un bit de mise en service, un bit de démarrage, un drapeau (flag) qui indique la fin de conversion, EOC (End Of Conversion). Il peut servir de demande d'interruption, si celle-ci est activée.

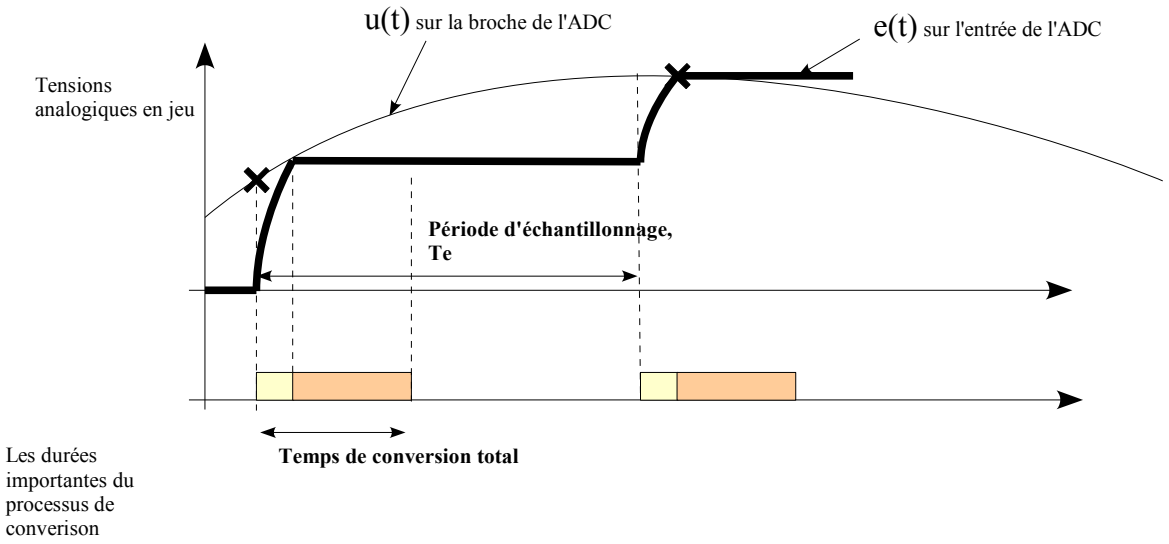
Plusieurs mode de fonctionnement existent, on peut citer (la liste n'est pas exhaustive) :

- le mode *single conversion* : l'utilisateur lance une conversion et récupère le résultat dans le registre associé, soit en scrutant le drapeau EOC, soit en attendant l'entrée en interruption.
- Le mode *auto conversion* : l'utilisateur lance une conversion, et dès que l'ADC finit sa conversion, il en lance une autre, eventuellement sur un autre canal, et ce de manière cyclique. Les résultats sont donc à lire régulièrement, sinon ils sont perdus. Selon le microcontrôleur les résultats peuvent être stockés dans une table de registres internes au périphérique, ou directement dans une table en RAM. On parle alors de DMA (Direct Memory Access).

2.3.3. Les timings dans un périphérique ADC

Commençons par définr les durées "clé" du processus de conversion :

Représentations des tensions et durées importantes dans la converision A/N



Temps d'échantillonnage, t_{ech} : c'est la durée pendant laquelle l'échantillonneur (l'interrupteur, voir synoptique) est fermé. La tension e(t) évolue de manière exponentielle à cause du condensateur et de la résistance équivalente du circuit. Il est donc nécessaire, suivant la valeur de cette résistance, de tenir l'échantillonneur fermé un certain temps, t_{ech}.

Temps de conversion, t_{conv} : Dès lors que l'échantillonneur est ouvert, la tension e(t) reste constante, et la conversion peut commencer. Elle dure un temps lié à l'horloge d'entrée de l'ADC et à la technologie de l'ADC.

Ces deux grandeurs temporelles, t_{ech} et t_{conv}, sont habituellement paramétrables via un registre de configuration de l'ADC.

Temps de conversion total : Ce terme, non générique, traduit simplement le temps nécessaire à une conversion totale, depuis la demande de conversion, jusqu'à l'écriture du résultat dans le registre associé.

Période d'échantillonnage, Te (à ne pas confondre avec le temps d'échantillonnage) : C'est l'intervalle de temps entre deux prises d'échantillons successifs. L'inverse étant la ***fréquence d'échantillonage***, que l'on choisit selon le ***critère de Shannon***.

On voit immédiatement sur les chronogrammes précédents, que la fréquence d'échantillonnage maximale pour un ADC donné est :

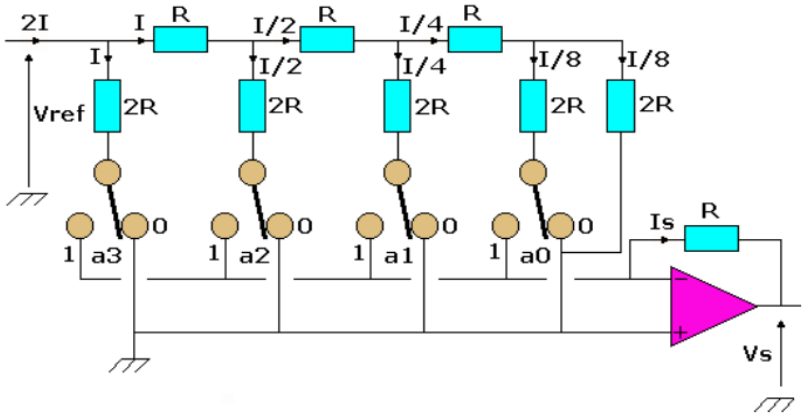
$$F_e Max = \frac{1}{(t_{ech} + t_{conv})}$$

2.4. Le DAC (Digital to Analog Converter)

Il est rare de rencontrer ce périphérique dans un microcontrôleur. Le STM32F107VC en comporte un. C'est d'ailleurs pour cela que nous l'avons retenu pour l'application télécom.

Il est le réciproque de l'ADC. Il possède un temps de réponse très rapide. En effet, le procéssus de conversion est très souvent basé sur une échelle de résistances de type réseau R-2R. Chacun des N bits sont appliqués, sous la forme de tension (3,3V ou 0V) en parallèle sur le circuit. Le temps de réponse est donc négligeable.

Schéma d'un CNA sur le principe du réseau R-2R, 4 bits (a3..a0) :



Ce périphérique est à priori très simple : après une configuration minimale (nombre de bits, activation), il suffit d'écire la valeur numérique représentative de la valeur analogique souhaitée (voir ADC), pour observer la tension effective sur la broche du GPIO correspondante.

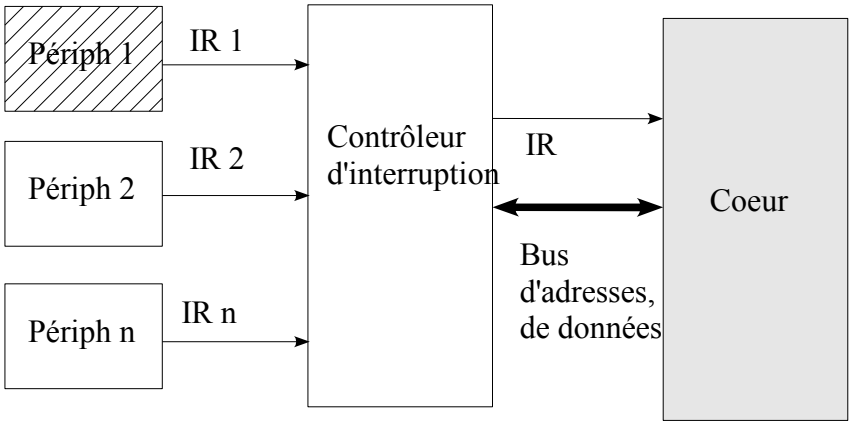
NB: Tout comme pour l'ADC, le DAC peut être directement associé à la RAM du microcontrôleur, via un circuit de DMA. La configuration se complique alors, et chaque constructeur possède sa propre spécificité. A voir au cas par cas.

2.5. Le système d'interruptions

Le *système d'interruption* est un des éléments essentiels d'un microcontrôleur. Il permet de décharger le processeur (minimiser son travail de traitement logiciel), pour laisser des actions se faire de manière matérielle. Typiquement, surveiller l'état d'un périphérique est bien inutile. Mieux vaut laisser au périphérique le soin de signaler lui-même son état. C'est donc bien le système d'interruptions qui va permettre de mettre en place ce type de dialogue coeur – périphérique, qui va donner en quelques sortes de “l'autonomie” au périphérique.

En principe, un périphérique donné possède un fil de sortie que l'on peut appeler *Interrupt Flag*, ou encore *Interrupt Request*. Le périphérique est relié au coeur, via un *gestionnaire d'interruptions*. Ce dernier est capable d'interrompre le coeur et de lui fournir une adresse où est situé le *programme d'interruption* à traiter (le *Handler*). Cette adresse est connue sous le nom de *vecteur d'interruption*.
Finalement, si chaque périphérique possède son propre vecteur d'interruption, alors il est très simple de dérouter le coeur de son activité pour l'aiguiller directement sur le programme correspondant.
Parfois, pour des raisons de simplicité de fabrication, plusieurs périphériques, possèdent le même vecteur d'interruption. Dans ce cas, le programmeur doit prévoir un test sur les *Interrupt Flag*, afin de savoir qui est à l'origine de l'interruption et donc de lancer le traitement approprié.

Schéma très simplifié de la structure matérielle :



Structure logicielle et évolution dans le temps :

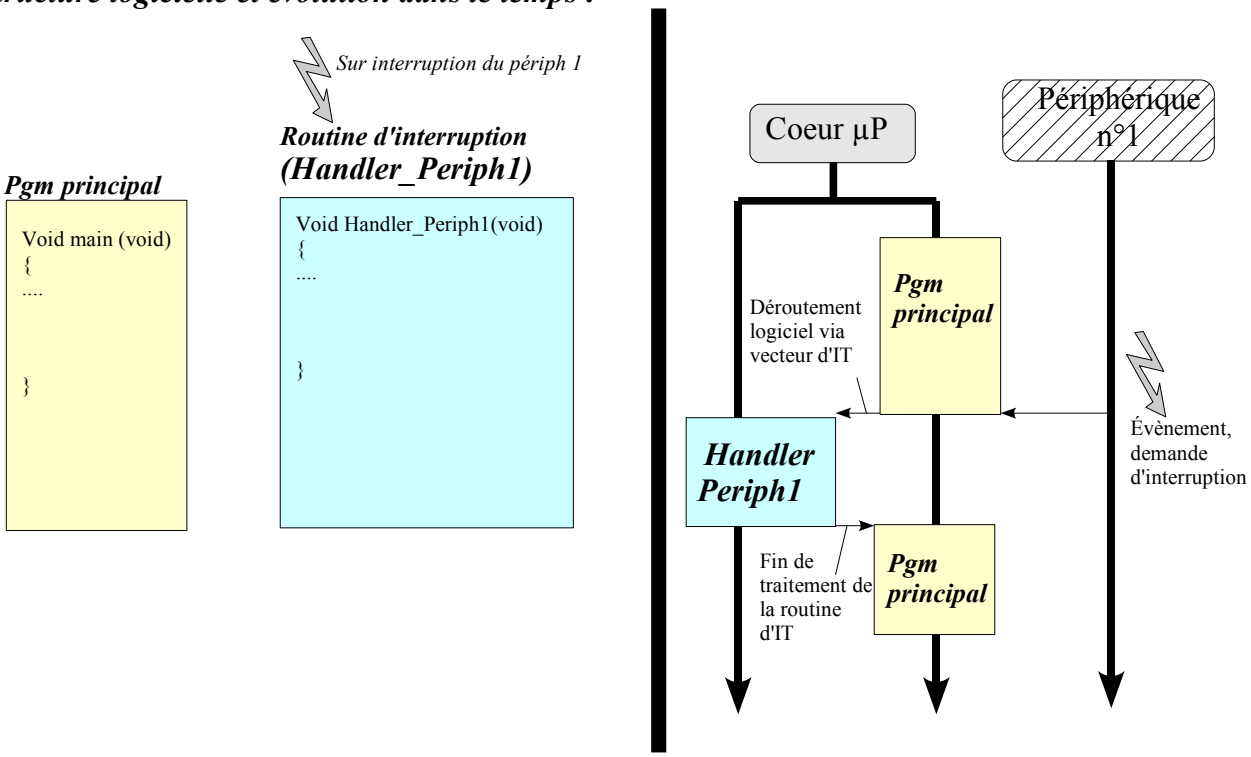
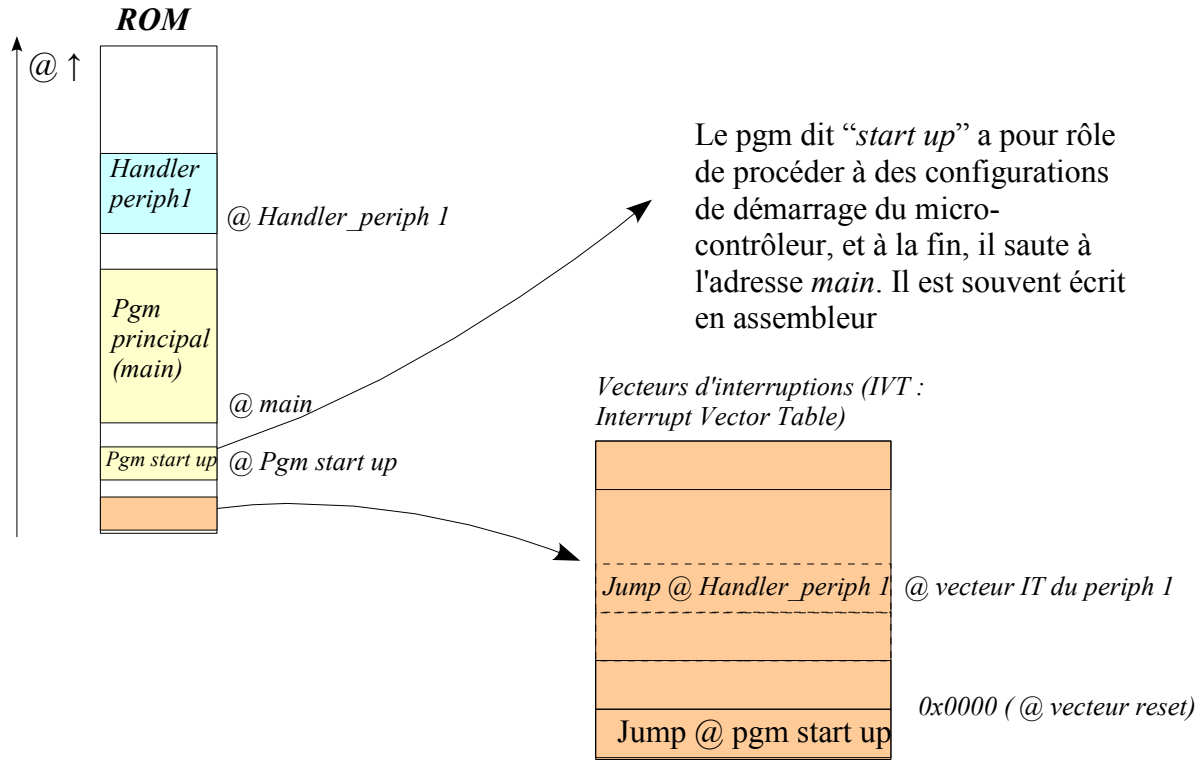


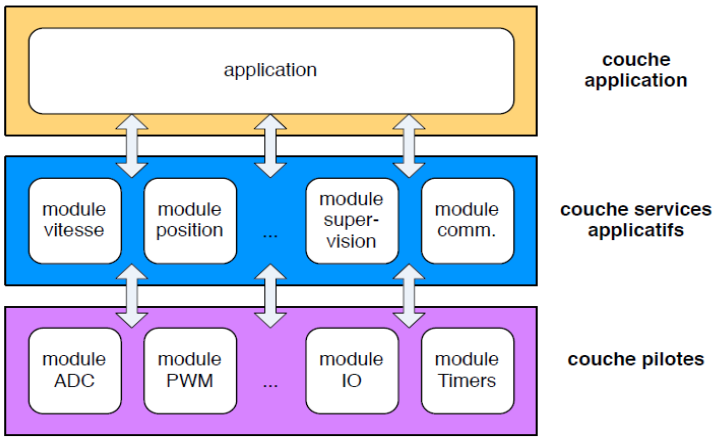
Schéma représentant les espaces ROM :



Précisons que le contrôleur d'interruptions permet aussi de gérer les priorités d'interruptions (dans le cas où plusieurs sont demandées en même temps). Enfin, pour qu'un périphérique puisse émettre une interruption il faut d'une part que la condition de réalisation soit rencontrée (débordement d'un timer par exemple), mais il faut aussi que la demande soit autorisée. Très souvent, au sein d'un périphérique, on trouve un bit qu'on peut appeler *IT_Periph_IE*, où *IE* signifie *Interrupt Enable*. Signalons enfin, que dans de nombreux cas, un bit de commande global, *GIE* (*Global Interrupt Enable*), appartenant au coeur permet de bloquer ou pas l'ensemble des interruptions validées localement dans chaque périphérique.

3. Structure logicielle en couches

Pour des raisons de qualité et de portabilité, on demande de construire le logiciel en trois couches :



La couche application : Cette couche ne comporte que du code dédié à une application. Elle comporte l'ensemble des fonctions de traitement et d'orchestration de l'application.

La couche de services applicatifs : Cette couche est un ensemble de fonctions qui masquent les périphériques matériels au niveau de l'application pour en offrir une représentation abstraite.
Par exemple, le service d'acquisition d'une vitesse masque l'appel au périphérique ADC pour offrir une vision purement abstraite de la variable de vitesse.

La couche de pilotes : Cette couche ne comporte que du code lié à la configuration et l'utilisation des périphériques sans présupposer une utilisation particulière par une application.
Une telle architecture offre de nombreux avantages :

- Seules les couches « services applicatifs »et « pilotes » doivent changer si le matériel évolue, la couche application n'ayant a priori besoin que de quelques modifications pour prendre en considération les nouvelles capacités du support d'exécution (surtout en ce qui concerne le temps).
- Les modules qui composent la couche pilotes peuvent être utilisés pour différentes applications et ainsi offrir des composants pris sur étagères.
- La couche services applicatifs fournit des services génériques pour un même ensemble de familles d'applications sans avoir besoin d'être réécrit à chaque développement ou évolution d'une application sur le même support.
- Le test des modules est fait de manière unitaire, ce qui facilite le débogage sans avoir besoin de chercher la petite bête.

Les éléments qui composent une couche ne peuvent communiquer qu'avec un élément d'une couche inférieure et cela se fait uniquement à travers les API des différents modules.

Pour plus d'informations, lire le document *« poly_periph.pdf »* disponible sur le site WIKI.

4. Structure logicielle pour un traitement en temps réel

Afin d'écrire proprement un programme temps réel, on vous propose une démarche rigoureuse. Elle repose sur la décomposition de l'exécution en tâches successives, dont une et une seule peut être active à la fois. Par ailleurs, chacune des tâches entamée devra être achevée avant de passer à la suivante. Il ne s'agit donc pas de « multi-tâche », mais de tâches qui s'enchaînent les unes après les autres.

4.1. Conception en séquences

Il s'agit donc, dans un premier temps, d'identifier le « *quatum temporel* », c'est à dire la durée la plus petite pour une tâche. On l'appellera *T_Ref.*
Une fois cette durée définie, on met en place un système d'interruption basé sur un timer dont on aura pris soin de configurer sa périodicité égale à cette durée minimale *T_Ref*.

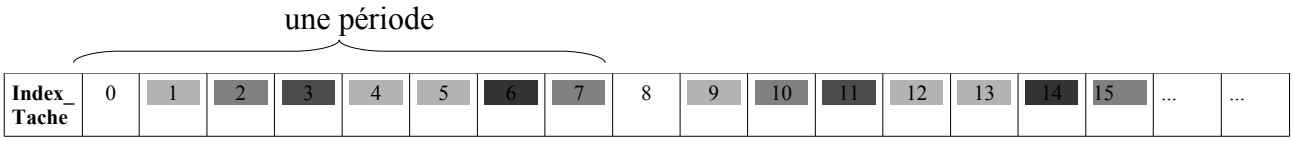
Ensuite, on établit une planification des tâches. On commence donc par repérer la périodicité des occurrences de tâches. On notera cette périodicité *Periode = N*T_ref.* La variable *Index_Tache* servira de repérage des tâches.

Il ne reste plus qu'à définir l'ensemble des tâches à exécuter, et à définir l'instant de démarrage.

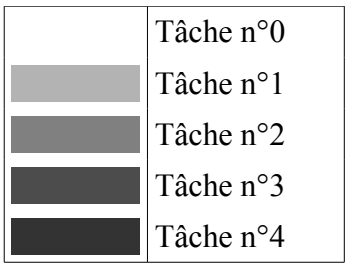
NB: Chaque tâche doit avoir une durée inférieure à *T_Ref* de manière à être sûr qu'elle soit achevée avant le lancement de la prochaine.

Exemple :

Soient 5 tâches qui se succèdent comme suit :



Une période est décrite sur 8 durées élémentaires *T_Ref*. *N*= 8.



Voici ce que peut donner, en langage C, une telle gestion de type « *tourniquet* »:

```
void IT_Timer(void)
{
// Tourniquer sur Index_Tache :
 Index_Tache++;
if ( Index_Tache==N)  Index_Tache=0;   // dans l'exemple, la constante N vaut 8
if ( Index_Tache%8)== 0)          // Si Index_Tache = 0 modulo 8
                {
                Tache_0();                // lancement de la tâche 0
                }
else if ( (Index_Tache%8 == 1)||(Index_Tache%8 == 4)||(Index_Tache%8 == 5))
                {
                Tache_1();
                }
else if ( (Index_Tache%8 == 2)||(Index_Tache%8 == 7))
                {
                Tache_2();
                }
else if (Index_Tache%4 == 3)
                {
                Tache_3();
                }
else if (Index_Tache%4 == 6)
                {
                Tache_4();
                }
}
```

La variable *Index_Tache* doit être globale. En effet, elle doit pouvoir être initialisée, donc vue, par le programme d'initialisation qui lance le tourniquet. Ce programme doit initialiser le timer en débordement sur *T_Ref*. Le programme *IT_Timer* qui gère le tourniquet, est le programme d'interruption lancé sur le débordement du timer.

4.2. Les échanges d'information entre tâches

4.2.1. Vocabulaire associé

Au cours du déroulement du programme, les tâches seront amenées à échanger des informations. On distingues ces dernières en deux types :

- les informations « *Mémoire partagée*» : elles contiennent des valeurs comme par exemple le résultat d'une conversion, le résultat d'un calcul...
- les informations de type « *drapeau* » : Elles permettent d'influencer le déroulement d'une tâche. Elle servent d'indicateur binaire pour opérer un choix conditionnel.

Ces deux types d'informations sont des variables visibles par les tâches concernées, donc globale et permanentes en mémoire (pas dynamique).

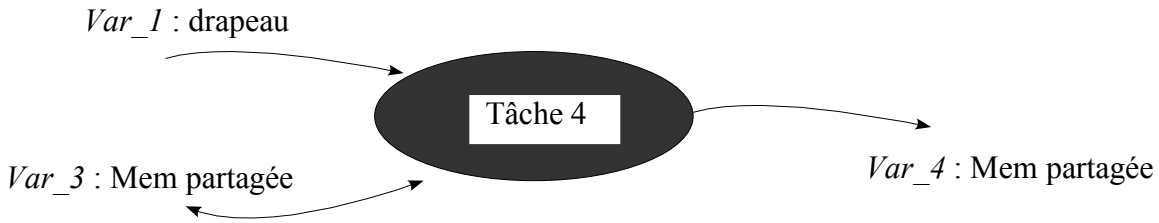
Les tâches, selon qu'elles sont à l'origine de telle ou telle information, ou qu'elles exploitent ces informations sont classées en deux catégories (du point de vue d'une information d'échange):

- *Emetteur :* la tâche affecte, écrit la variable
- *Recepteur* : la tâche exploite, donc lit la variable.

4.2.2. Représentation des échanges

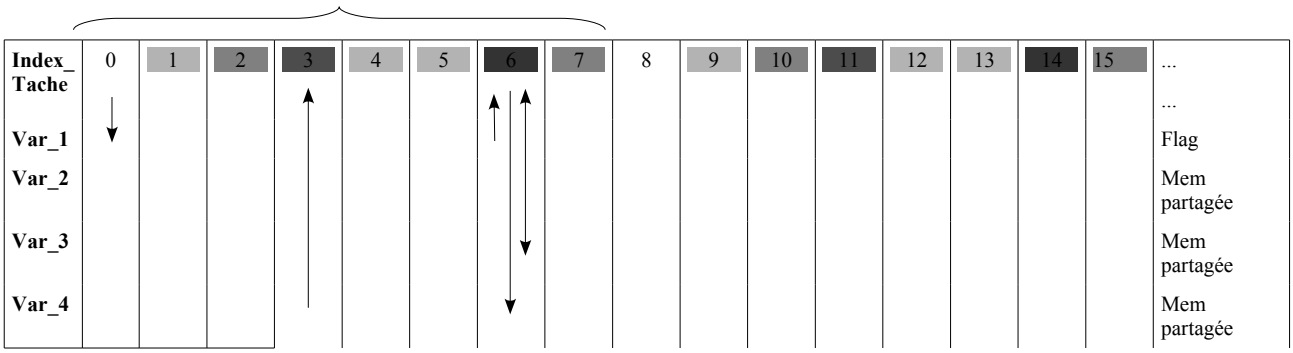
Afin de se faire une idée des échanges qui peuvent avoir lieu entre les diverses tâches, on propose de procéder de la manière suivante :

Définition des informations échangées pour une tâche :



Cette illustration, montre l'ensemble des drapeaux et variables de type mémoire partagée, qui sont utilisés pour la tâche x. Elle indique aussi, pour chaque variable ou drapeau, si la tâche est de type émettrice ou réceptrice. C'est le sens des flèches qui renseigne.

Illustration des échanges dans le temps :



Lecture du tableau :

Le drapeau (flag) *Var_1* est généré par la tâche n°0 puis exploitée pour une décision dans la tâche n°4.
La tâche n°4 mets à jour une variable de type mémoire partagée, *var_3*, pour sa prochaine exécution.
….

4.2.3. La fonction main : une tâche particulière...

La fonction *main* est utilisée pour le lancement de toutes les initialisations, pour la préparation du tourniquet...
Lorsque ces diverses configurations sont exécutées, alors peut commencer le balai des tâches. Entre deux tâches successives, il subsiste toujours un temps mort. En effet, rappelons que dans le cadre de

ce cours, chacune des tâches doit se loger dans le créneau de durée T_ref. Le temps mort correspond au retour d'interruption vers la fonction *main*.
On peut donc considérer que la fonction main est une ***tâche de fond*** non prioritaire. Sa durée d'exécution sur une période complète est difficile à calculer et surtout peut fluctuer, selon les exécutions conditionnées (par des sémaphores) de certaines tâches. Néanmoins, son avantage est que c'est la seule tâche qui peut s'étaler au delà d'une période T_Ref. Elle sera par exemple utilisée pour un calcul de fond, plus ou moins complexe et non critique en terme de vitesse d'éxecution.

Chapitre 2 : La démodulation IQ sur microcontrôleur

1. Principe de modulation & démodulation MAQ analogique

Nous allons voir dans cette partie, comment on peut élaborer un signal MAQ (Modulation d'Amplitude en Quadrature) et nous allons surtout amener la notion de *signal analytique* et surtout d'***enveloppe complexe***. Nous allons voir que le *diagramme de constellation* est la représentation graphique de l'enveloppe complexe.

1.1. Génération analogique d'un signal MAQ

Voici l'expression mathématique d'un signal réel MAQ qui fait apparaître l'amplitude et la phase instantanée de la porteuse : $x(t)=a(t).\cos(2.\pi.f_0.t+\varphi(t))$.
On notera que l'amplitude *a(t)* est positive strictement. C'est un module.
Cette écriture a l'avantage de séparer clairement amplitude et phase, le problème, c'est qu'il n'est pas évident d'en faire une représentation spectrale (transformée de Fourier) simple. C'est la phase, non constante, qui apparaît dans le cosinus qui est gênante.

L'idée est donc d'utiliser le formalisme des *signaux analytiques*, et de *l'enveloppe complexe*. (voir annexe pour plus de détail) :
Au signal réel *x(t)*, on associe son signal analytique, le complexe $z_x(t)$. De plus, *x(t)* étant un signal à bande étroite, on écrit directement :

$$z_x(t)=a(t).e^{j.\varphi(t)}.e^{j.2.\pi.f_0.t}=\alpha_x(t).e^{j.2.\pi.f_0.t}$$ sous condition que a(t)>0

La représentation par *signal analytique*, permet de sortir la phase du cosinus et rend alors possible la réprésentation spectrale.
Le complexe $\alpha_x(t)$ est ***l'enveloppe complexe*** de *x(t)*. Ecrivons la, en faisant apparaître les parties réelles et imaginaires :
$\alpha_x(t)=p(t)+j.q(t)$

Cela donne, pour le signal analytique :

$$z_x(t)=\alpha_x(t).e^{j.2.\pi.f_0.t}=(p(t)+j\,q(t)).(\cos(2.\pi.f_0.t)+j\sin(2.\pi.f_0.t))$$

Puis pour le signal réel *x(t)* :

$$x(t)=R[z_x(t)]=p(t).\cos(2.\pi.f_0.t)-q(t).\sin(2.\pi.f_0.t)$$

puisque, par définition, le signal réel est la partie réelle de son signal analytique.

Ainsi, le signal réel $x(t)=a(t).\cos(2.\pi.f_0.t+\varphi(t))$ est strictement équivalent à

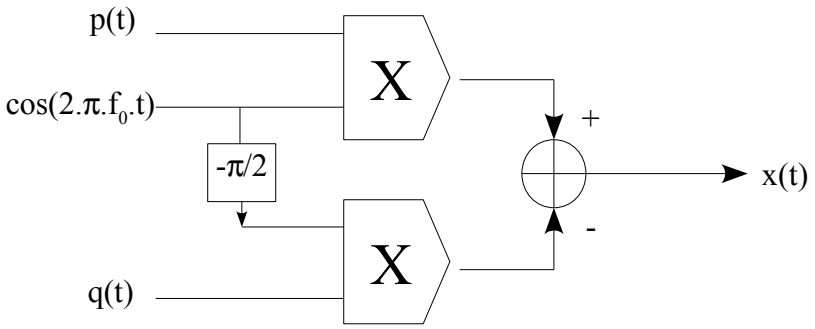
$x(t)=p(t).\cos(2.\pi.f_0.t)-q(t).\sin(2.\pi.f_0.t)$, avec

$p(t)=a(t).\cos(\varphi(t))$ et $q(t)=a(t).\sin(\varphi(t))$

Il s'agit d'une double modulation AM en quadrature, simple à réaliser (d'où le nom de MAQ).

NB : sans passer par toute la théorie de l'enveloppe complexe, on peut directement montrer l'équivalence des deux expressions, le plus simple étant de partir de l'expression MAQ.

La génération d'un signal MAQ se fait donc de la manière suivante :



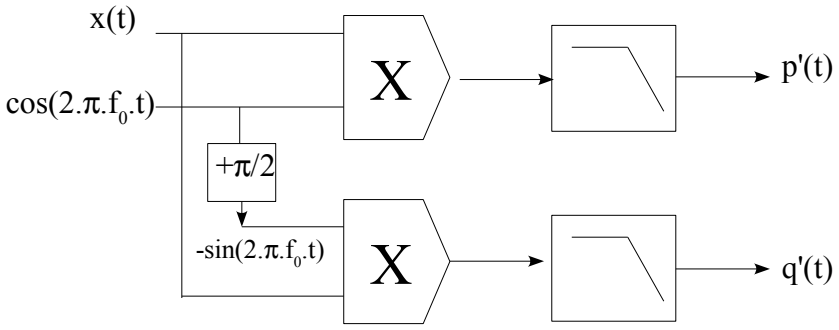
Les deux porteuses transportent la partie réelle et imaginaire de l'enveloppe complexe.

1.2. Démodulation IQ analogique

La démodulation MAQ consiste à faire une double démodulation synchrone. Comme les ondes en cosinus et sinus sont orthogonales (dans le sens $\int_T \cos(x).\sin(x)dx = 0$) , on peut récupérer les deux signaux p(t) et q(t). Dans la littérature, on rencontre souvent le terme de *démodulation IQ*, ce qui fait référence, en anglais, aux parties réelles et imaginaires :

p ⇔ I (In phase)
q ⇔ Q (in Quadrature)

Le principe de la démodulation IQ



1.2.1. Démodulation rigoureusement synchrone

Rappelons rapidement le principe de démodulation, dans un cas idéal de synchronisation parfaite des oscillateurs et déphaseurs de réception :

Le premier produit donne :

$$\Pi_1(t) = x(t).\cos(2.\pi.f_0.t)$$

$$\Pi_1(t) = [p(t).\cos(2.\pi.f_0.t) - q(t).\sin(2.\pi.f_0.t)].\cos(2.\pi.f_0.t)$$

$$\Pi_1(t) = [p(t).\cos(2.\pi.f_0.t) - q(t).\cos(2.\pi.f_0.t - \frac{\pi}{2})].\cos(2.\pi.f_0.t)$$

$$\Pi_1(t) = \frac{p(t)}{2}[1 + \cos(2.\pi.2.f_0.t)] - \frac{q(t)}{2}[0 + \cos(2.\pi.2.f_0.t - \frac{\pi}{2})]$$

Ce qui donne après filtrage $p'(t) = \frac{p(t)}{2}$

De la même manière, $q'(t) = \frac{q(t)}{2}$

1.2.2. Démodulation asynchrone

Analysons ici le cas d'un récepteur dont l'oscillateur local est décalé en fréquence et en phase

Afin d'alléger les notations, nous allons raisonner en pulsation. On considère qu'en réception on dispose d'un cosinus légèrement décalé en fréquence par rapport à la porteuse : $\cos(\omega_0'.t + \psi)$ $\omega_0' = \omega_0 - \Delta\omega$. La référence au récepteur s'écrit $\cos((\omega_0 - \Delta\omega).t + \psi)$.

Nous nous intéresserons par la suite uniquement au terme “différence” issu de chacun des produits, sachant que la composante “somme” est filtrée. Donc volontairement, dans le calcul des produits, le terme somme n'apparaît pas.

$$\Pi_1(t) = [p(t).\cos(\omega_0.t) - q(t).\sin(\omega_0.t)].\cos((\omega_0 - \Delta\omega).t + \psi)$$

$$\Pi_1(t) = [p(t).\cos(\omega_0.t) - q(t).\cos(\omega_0.t - \frac{\pi}{2})].\cos((\omega_0 - \Delta\omega).t + \psi)$$

$$p'(t) = \frac{1}{2}.p(t).\cos(\Delta\omega.t - \psi) - \frac{1}{2}.q(t).\cos(\Delta\omega.t - \psi - \frac{\pi}{2})$$

$$p'(t) = \frac{1}{2}.p(t).\cos(\Delta\omega.t - \psi) - \frac{1}{2}.q(t).\sin(\Delta\omega.t - \psi)$$

A ce stade, faisons intervenir la représentation “module et argument” de l'enveloppe complexe :

$p(t) = a(t).\cos(\varphi)$ et $q(t) = a(t).\sin(\varphi)$:

$$p'(t)=\frac{1}{2}.a(t).\cos(\varphi).\cos(\Delta\omega.t-\psi)-\frac{1}{2}.a(t).\sin(\varphi).\sin(\Delta\omega.t-\psi)$$

finalement :

$$\boxed{p'(t)=\frac{1}{2}.a(t).\cos(\varphi+\Delta\omega.t-\psi)}$$

Le second produit donne :

$$\Pi_2(t)=[p(t).\cos(\omega_0.t)-q(t).\cos(\omega_0.t-\frac{\pi}{2})].\cos((\omega_0-\Delta\omega).t+\psi+\frac{\pi}{2})$$

$$q'(t)=\frac{1}{2}.p(t).\cos(\Delta\omega.t-\psi-\frac{\pi}{2})+\frac{1}{2}.q(t).\cos(\Delta\omega.t-\psi)$$

$$q'(t)=\frac{1}{2}.p(t).\sin(\Delta\omega.t-\psi)+\frac{1}{2}.q(t).\cos(\Delta\omega.t-\psi)$$

soit

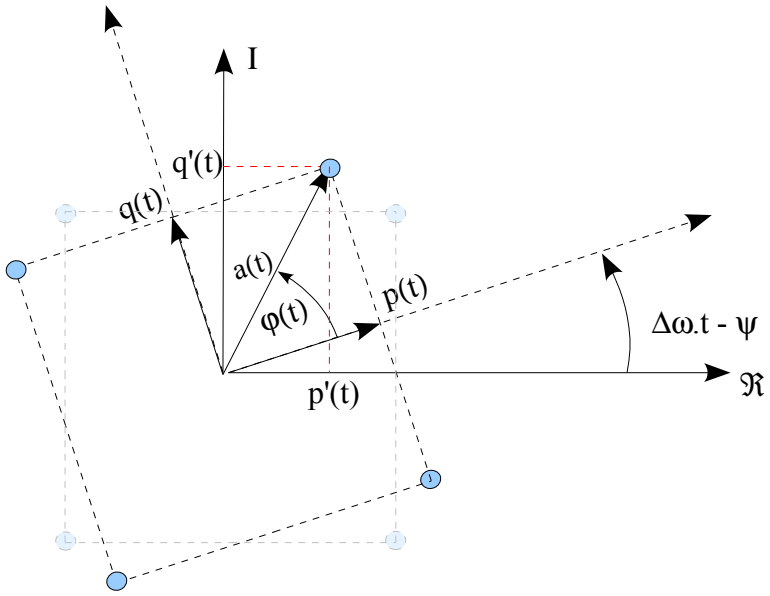
$$q'(t)=\frac{1}{2}.a(t).\cos(\varphi).\sin(\Delta\omega.t-\psi)+\frac{1}{2}.a(t).\sin(\varphi).\cos(\Delta\omega.t-\psi)$$

$$\boxed{q'(t)=\frac{1}{2}.a(t).\sin(\varphi+\Delta\omega.t-\psi)}$$

Après ces développements, on se rend compte que l'enveloppe complexe obtenue possède un argument faussé, qui “glisse” à la vitesse qui est l'écart de fréquence de démodulation.

Le diagramme de constellation s'obtient dans le plan complexe (représentation de I et Q).

La constellation obtenue dans ces conditions (après multiplication par 2) est :

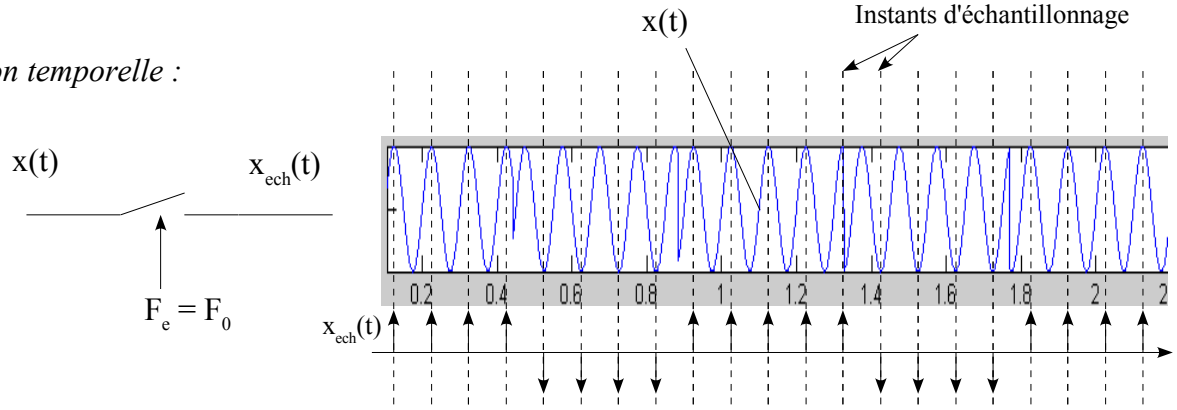


2. Démodulation directe par échantillonnage à la fréquence porteuse

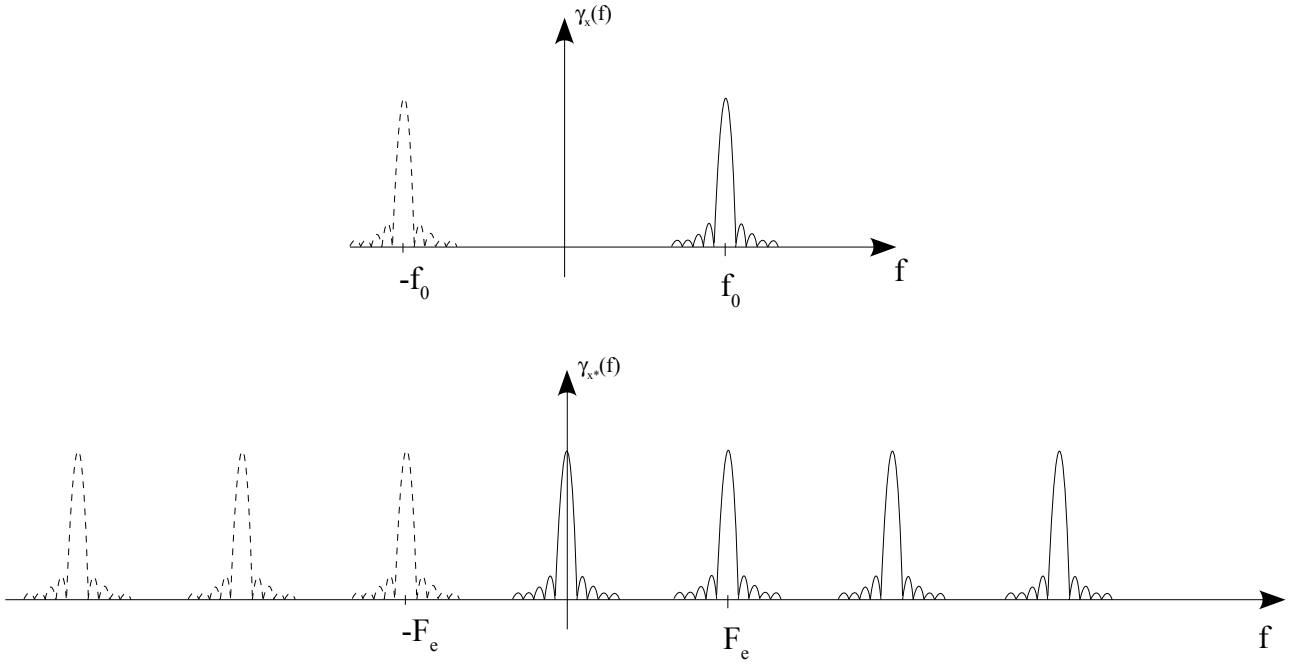
2.1. Introduction

En lieu et place d'un mélangeur, on peut utiliser directement un micro-contrôleur et plus particulièrement un ADC qui échantillonne justement à la fréquence porteuse. Il s'agit en fait d'un sous-échantillonnage, puisque le signal d'entrée s'étend au delà de $F_e/2$ (limite de Shannon).

Opération temporelle :



Opération fréquentielle :



L'observation du spectre ci-dessus montre que l'opération d'échantillonnage génère une infinité de spectres, ceux d'origine qui sont translatés tous les $k.F_e$. Parmi cette infinité, un seul nous intéresse celui situé en bande de base, autour de 0Hz.
Afin de rapprocher le phénomène d'échantillonnage de celui de la modulation, nous allons dans un premier temps montrer qu'un échantillonnage est équivalent à une infinité de modulation AM.

2.2. Echantillonnage et modulation AM

Le but de ce chapitre est de montrer qu'une chaîne d'acquisition numérique (ADC + calculateur + DAC) peut être équivalente à un système de démodulation synchrone. Pour y parvenir, nous allons revoir le concept purement mathématique d'échantillonage (à priori abstrait puisque faisant intervenir des signaux échantillonnés sans réalité physique). Nous ferons un rapprochement avec la modulation AM. Ensuite, après une analyse des diverses parties constitutives d'une chaîne d'acquisition, nous allons pouvoir matérialiser cette théorie avec le soucis de concilier signaux réels (palpables) et signaux fictifs (mais nécessaires !).

2.2.1. Théorie

La modulation AM est ici prise au sens de la multiplication par un cosinus (modulation AM sans porteuse).
Rappelons que l'échantillonnage peut être vu comme la multiplication d'un signal x(t) par un peigne de Dirac que l'on notera $Peigne(t)=\sum_{k=-\infty}^{k=+\infty}\delta(t-kT_e)$

On montre que la transformée de Fourier du peigne temporel est un peigne fréquentiel que l'on notera :

$$Peigne(f)=\frac{1}{T_e}\sum_{k=-\infty}^{k=+\infty}\delta(f-kF_e)$$

On peut alors regrouper les termes deux à deux, par fréquences opposées. Cela donne alors :

$$Peigne(f)=\frac{1}{T_e}\sum_{k=1}^{k=+\infty}[\delta(f-kF_e)+\delta(f+kF_e)]+\frac{1}{T_e}\delta(0)$$

En introduisant un facteur 2, on fait apparaître une somme de transformée de cosinus :

$$Peigne(f)=\frac{2}{T_e}\sum_{k=1}^{k=+\infty}[\frac{1}{2}.\delta(f-kF_e)+\frac{1}{2}.\delta(f+kF_e)]+\frac{1}{T_e}\delta(0)$$

On rappelle la transformée de Fourier d'un cosinus :

$A_p.\cos(2.\pi.f_p.t+\varphi_p)$ donne $S(f)=\frac{1}{2}.\delta(f-f_p).e^{+j\varphi}+\frac{1}{2}.\delta(f+f_p).e^{-j\varphi}$

Par transformée de Fourier inverse, on obtient donc :

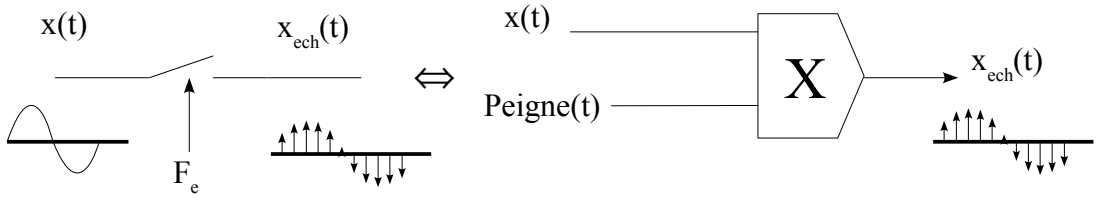
$$Peigne(t)=\frac{1}{T_e}+\frac{2}{T_e}\sum_{k=1}^{k=+\infty}\cos(2.\pi.k.F_e.t)$$

L'opération d'échantillonnage s'écrit donc :

$$x_{ech}(t)=Peigne(t).x(t)=\frac{1}{T_e}.x(t)+\frac{2}{T_e}\sum_{k=1}^{k=+\infty}[x(t).\cos(2.\pi.k.F_e.t)]$$

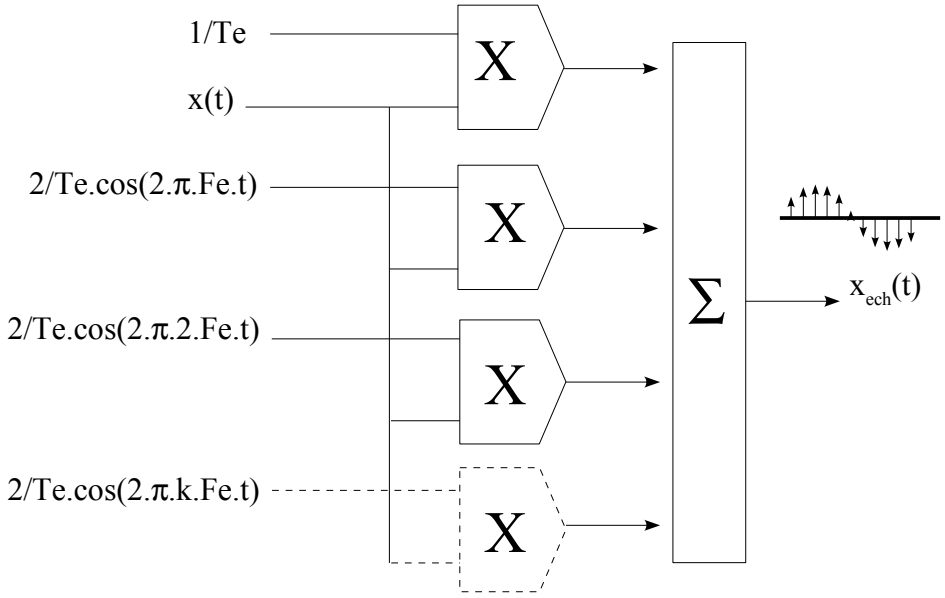
Résumé

Le fait d'introduire la multiplication analogique de x(t) par un peigne de Dirac (que nous considérons ici comme une fonction continue) conduit à faire un rapprochement avec ce que nous connaissons des modulations classiques :



Représentation discrète *Représentation continue*

ou encore :



Les représentations discrètes et continues sont tout à fait équivalentes. Elles produisent le même type de signal. Soulignons que ce dernier n'a pas de réalité physique, puisqu'il s'agit de suite de Dirac (d'amplitude infinie par définition). Néanmoins, si nous acceptons que l'on puisse construire une infinité de multiplieurs, si nous acceptons aussi le fait que le sommateur ne sature pas (pas de limite d'amplitude), et enfin si nous considérons les multiplieurs et sommateurs comme des éléments à bande passante infinie, alors nous avons recréé sur la figure ci-dessus, **l'échantillonnage de x(t) avec des <u>fonctions et opérateurs continus</u>**.

Cet effort d'imagination, d'abstraction étant fait, on peut envisager de traiter $x_{ech}(t)$ comme un signal réel Lambda, bien qu'il ne possède pas de réalité physique.

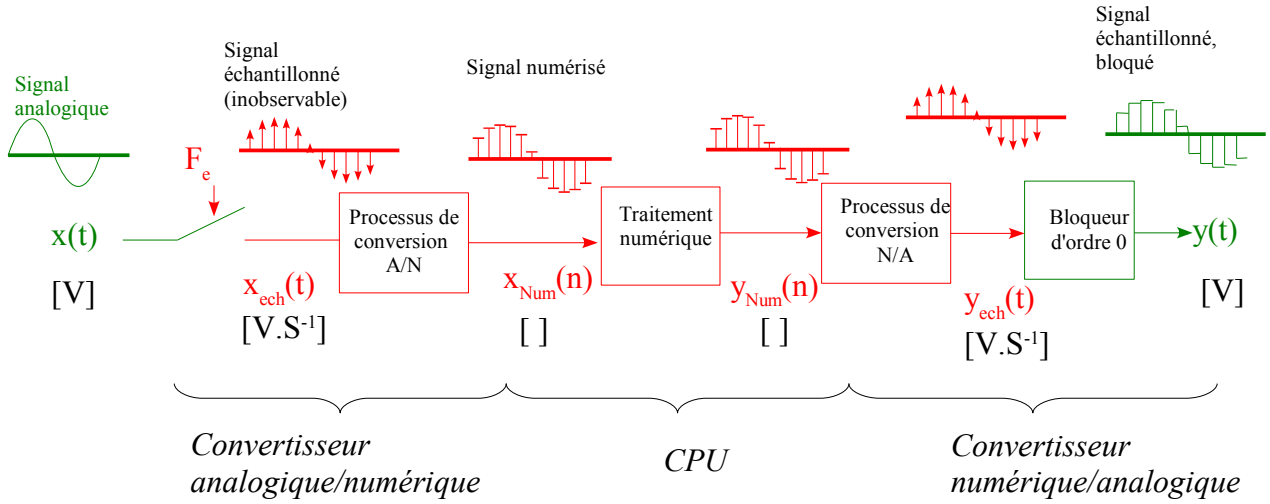
2.2.2. Echantillonnage et convertisseur A/N

Tout ce qui a été vu jusqu'à maintenant sur la théorie de l'échantillonnage est nécessaire et indispensable pour comprendre ce qui se joue en terme de signal, temporel, fréquentiel.

Nous avons notamment montré que du point de vue mathématique, échantillonner un signal réel, de

type tension par exemple, revient à obtenir un signal sans existance physique, qui aurait la forme d'une suite d'impulsions de durée nulle, d'amplitude infinie, et dont le poids (la surface) contiendrait l'information de la grandeur traitée.
On représente (à tord mais on peut difficilement faire autrement), un signal échantillonné par une suite de traits ou de flèches, dont l'amplitude est à l'image du signal échantillonné (faux, puisque c'est la surface, rappelons-le qui contient l'information).
Il est nécessaire maintenant de faire le lien entre cette abstraction mathématique et la réalité des systèmes échantillonnés par micro-contrôleur.

Synoptique classique d'un ADC + traitement + DAC. Entre crochet, la dimension :



Légende :

impulsion de dirac : purement théorique, amplitude infinie, surface contenant l'information. Dimension [$V.s^{-1}$] puisque sa surface est en volt.

nombre, valeur numérique à un instant donné. Son amplitude contient l'information. C'est une grandeur palpable. C'est un nombre sans dimension.

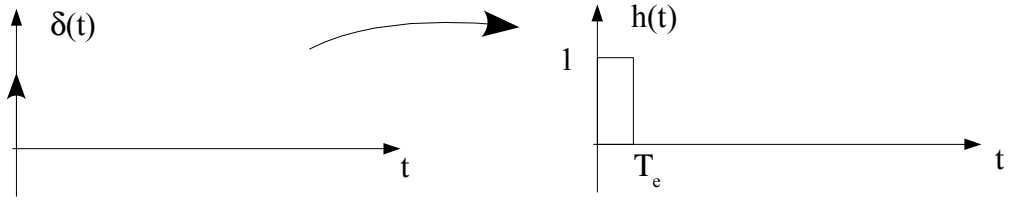
Signal échantillonné bloqué : c'est une grandeur analogique (tension) qui est constante entre deux échantillons.

Explication :
<u>L'ADC (Analog to Digital Converter) :</u>
Il transforme le signal analogique en une suite d'impulsion $X_{num}(n)$. Sur le schéma, le travail de l'ADC est décomposé en deux : l'échantillonnage (opération purement mathématique), et la conversion proprement dite (quantification). On note au passage que le signal échantillonné $X_{ech}(t)$ est devenu $X_{num}(n)$ qui est une suite de nombre, et non plus une suite de Diracs.

<u>La CPU :</u>
C'est le lieu du calcul numérique. A ce niveau là, les signaux on tous des spectres infinis et qui se répètent tous les Fe (fréquence d'échantillonnage), les spectres sont périodiques.

Le DAC (Digital to Analog Converter) :

Le DAC prend les valeurs numériques $Y_{Num}(n)$ et les transforme en une tension analogique y(t). Un élément très important du DAC est le bloqueur d'ordre 0. On considère cet élément comme une fonction de transfert analogique. Comme tel, on l'analyse par sa réponse impulsionnelle (réponse à un Dirac) :



Afin de le caractériser, on va chercher la fonction de transfert (linéaire) :

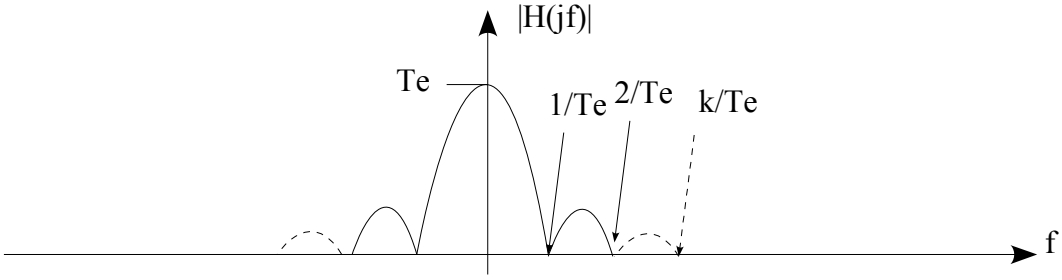
$$\mathcal{F}h(t) = \int_0^{+T_e} 1.e^{-j.2.\pi.f.t}\,dt = \frac{1}{-j.2.\pi.f}.(e^{-j.2.\pi.f.T_e}-1) = H(jf)$$

$$= \frac{e^{j.2.\pi.f.T_e/2}}{j.2.\pi.f}.(e^{j.2.\pi.f.T_e/2}-e^{-j.2.\pi.f.T_e/2})$$

$$= \boxed{\frac{T_e.e^{j.2.\pi.f.T_e/2}}{j.2.(\pi.f.T_e)}.(e^{+j.\pi.f.T_e}-e^{-j.\pi.f.T_e}) = T_e e^{j.2.\pi.f.T_e/2}\,sinc(\pi.f.T_e)}$$

Cette expression montre deux choses :

- Le changement de nature de la sortie, qui correspondra à la dimension du signal d'entrée que multiplie Te : d'une dimension [V.s-1] on passe à de volts, [V].
- La fonction sinus cardinal va opérer un filtrage passe-bas sur le spectre d'entrée

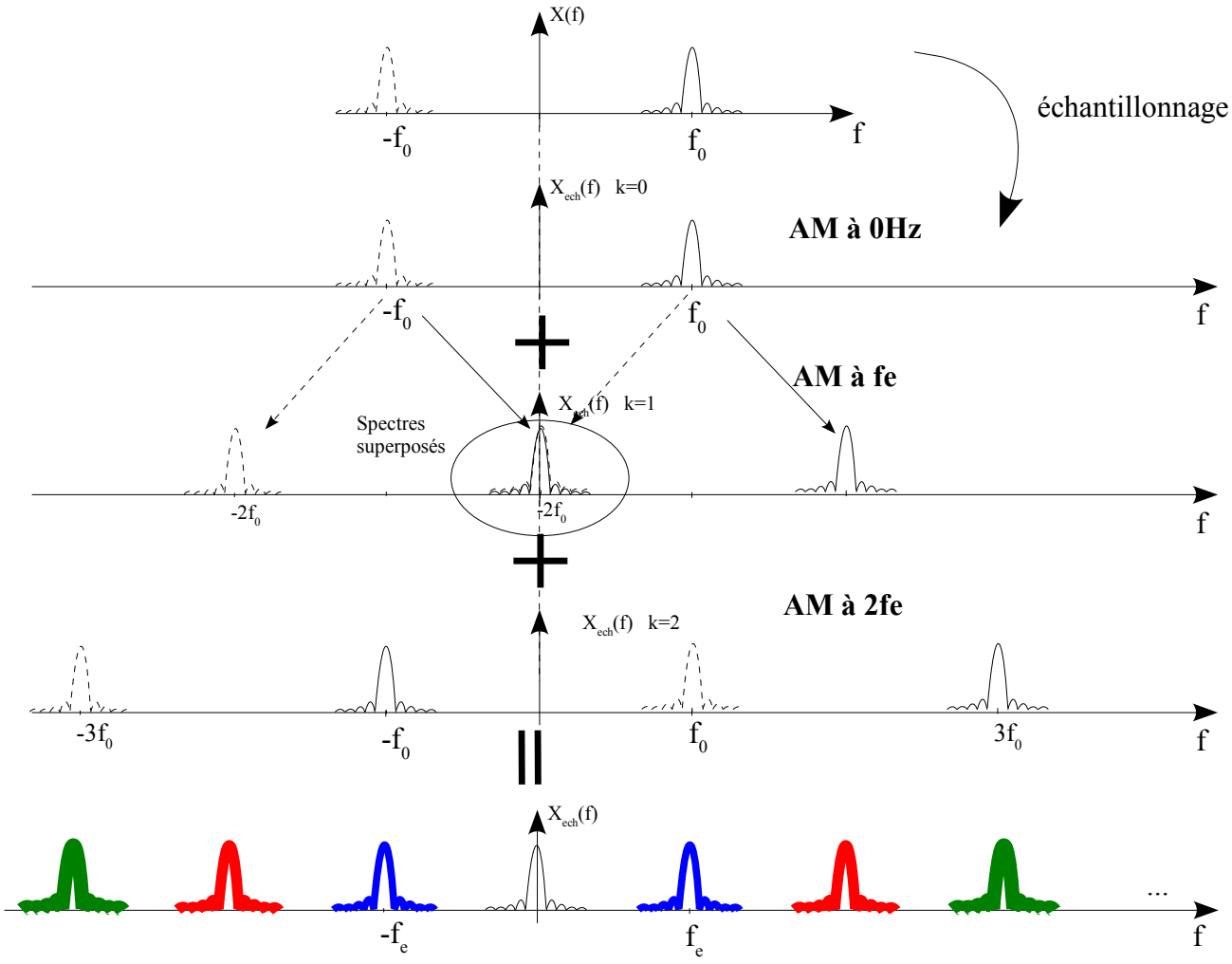


Le bloqueur d'ordre 0 aura donc la propriété essentielle de filtrage **passe-bas**, ce qui préserve les fréquences en dessous de Fe (avec une distorsion). Le spectre n'est plus infini, n'est plus périodique : le bloqueur d'ordre 0 a donc opéré le changement de nature échantillonné / continu.

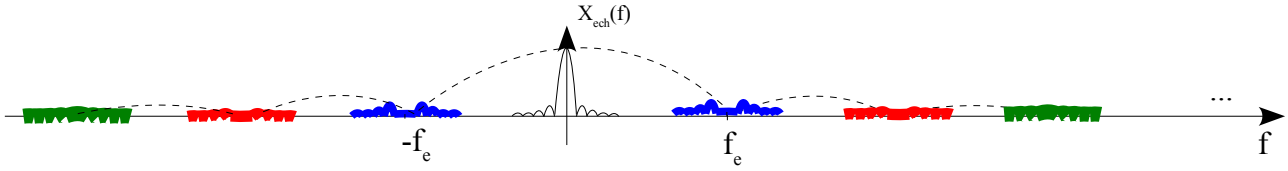
Enfin, le bloc « *processus de conversion N/A* » du schéma permet de transformer les nombres $Y_{Num}(n)$ en une suite d'impulsions de Dirac, $Y_{ech}(t)$, compatible en terme de dimension avec le bloqueur d'ordre 0. On note également que c'est à ce niveau que se passe la quantification du DAC.

2.2.3. Echantillonnage d'un signal MAQ

Représentons le spectre $X_{ech}(f)$ progressivement , pour x(t) de type MAQ à la fréquence f_0:



L'intérêt de voir l'échantillonnage comme une combinaison infinie de modulation AM est de pouvoir interpréter directement la partie rabattue en bande de base comme étant le résultat de la multiplication de x(t) par $2/Te.\cos(2.\pi.f_e t)$: premier élément du détecteur synchrone analogique bien connu.
Le bloqueur d'ordre 0 va servir naturellement de filtre passe-bas du détecteur synchrone (second élément). Voici le spectre du signal bloqué :



2.3. Echantillonnage en quadrature

Nous venons de voir que l'échantillonnage direct est équivalent à une multiplication par un cosinus suivi d'un filtre passe-bas (le bloqueur d'ordre 0 d'entrée). Comment obtenir l'équivalent d'une multiplication par un sinus ?
L'idée est de créer un déphasage de π/2 sur l'échantillonnage, c'est à dire , provoquer un retard pur d'un quart de période.

A ce moment là, le peigne de Dirac s'écrit :

$$PeigneQuad(t)=Peigne(t-\frac{T_e}{4})=\sum_{k=-\infty}^{k=+\infty}\delta(t-kT_e-\frac{T_e}{4})$$

Par définition, la transformée de Fourier s'écrit :

$$PeigneQuad(f)=[\frac{1}{T_e}\sum_{k=-\infty}^{k=+\infty}\delta(f-kF_e)].e^{-j.2.\pi.f.\frac{T_e}{4}}=\frac{1}{T_e}\sum_{k=-\infty}^{k=+\infty}\delta(f-kF_e).e^{-j.\frac{\pi}{2}.f.T_e}$$

soit

$$PeigneQuad(f)=\frac{1}{T_e}\sum_{k=-\infty}^{k=+\infty}\delta(f-kF_e).e^{-j.k.\frac{\pi}{2}}$$

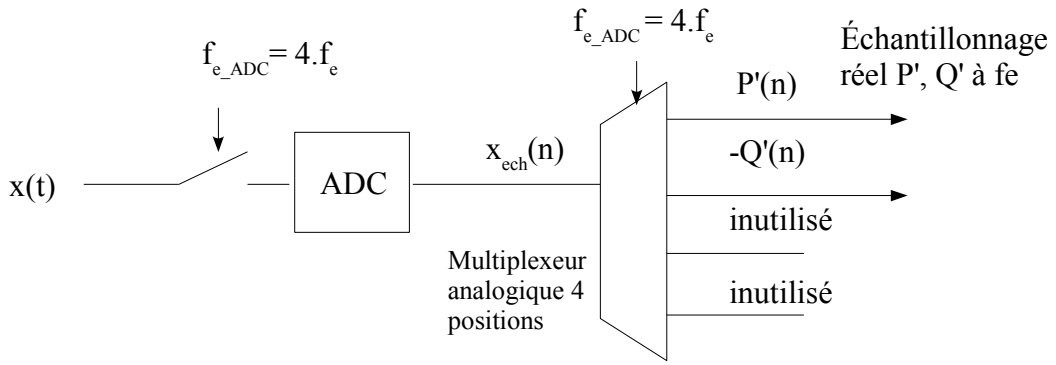
En regroupant les termes de coefficients k opposés, on obtient

$$PeigneQuad(t)=\frac{1}{T_e}+\frac{2}{T_e}\sum_{k=1}^{k=+\infty}\cos(2.\pi.k.F_e.t-k.\frac{\pi}{2})$$

Ainsi, pour k=1, on se retrouve avec l'équivalent d'une multiplication par $\frac{2}{T_e}\sin(2.\pi.k.F_e.t)$

Afin de reproduire à la fois la multiplication en cosinus et celle en sinus, on va procéder à un échantillonnage unique, dont la fréquence réelle d'échantillonnage de l'ADC, f_{e_ADC}, est **quatre fois plus grande** que la fréquence d'échantillonnage f_e du signal. A l'issue d'une période complète d'échantillonnage Te = 1/fe, on disposera de 4 échantillons (prélevés à T_{e_ADC} = 1 / f_{e_ADC}), dont les deux premiers seront respectivement représentatif due la multiplication en cosinus et en sinus.

Le démodulateur IQ par échantillonnage à la fréquence porteuse devient alors le suivant :



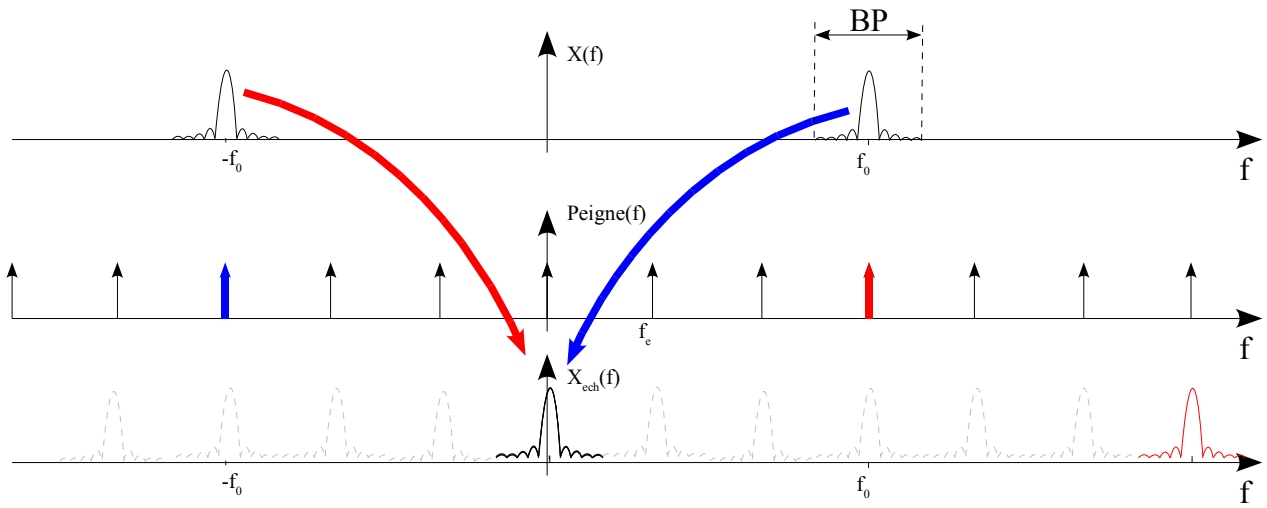
Le signal IQ présent à l'entrée est $x(t)=p(t).\cos(2.\pi.f_0.t)-q(t).\sin(2.\pi.f_0.t)$

Ce dispositif d'échantillonnage est aussi appelé *Filtre polyphase numérique.* Il est donc vu comme un filtre numérique ultra-simple, fonctionnant à la fréquence Fe, mais possédant plusieurs phases en sortie, 4 pour être précis.

3. Démodulation directe par sous-échantillonnage

3.1. Présentation

Dans cette partie, nous allons exploiter les résultats vus précédemment, et procéder à un sous échantillonnage. La fréquence f_e vaut donc f_0/M, M étant entier. Voyons cela au niveau spectral :



Dans cet exemple, la “modulation d'amplitude” qui nous intéresse est celle dont la fréquence est $3.f_e$. Bien entendu, le spectre $X_{ech}(f)$ est encombré par l'infinité des autres modulations. Le bloqueur d'ordre 0 fera son travail d'élimination de ces éléments gênants.

Si l'on observe le signal avant échantillonnage, X(f), dont le spectre est bien plus élevé que f_e, le théorème de Shannon n'est pas respecté. Par contre, il faut noter que si on ne veut pas de repliement de spectre du modulant, il faut nécessairement que fe > Bp, Bp est la bande passante du signal à bande étroite. Dans l'exemple, on se trouve juste à la limite.

3.2. Sous échantillonnage de la partie en quadrature

Le peigne en phase produit uniquement des cosinus. Ce n'est pas le cas du peigne décalé d'un quart de période. Toutes les harmoniques ne donnent pas un sinus.

Reprenons l'analyse faite en 2.3 :

$$PeigneQuad(t) = \frac{1}{T_e} + \frac{2}{T_e}\sum_{k=1}^{k=+\infty} \cos(2.\pi.k.F_e.t - k.\frac{\pi}{2})$$

Nous pouvons dresser un tableau qui indique la nature de l'onde sinusoïdale en fonction de son rang k:

rang k	**nature de la composante de rang k**	**Utilisable en I/Q**
K = 0, 4, 8...= 4.i	+cosinus	non
K = 1, 5, 9... = 1+4.i	+sinus	oui
K = 2, 6, ... = 2+4.i	-cosinus	non
K = 3, 7, ... = 3+4.i	-sinus	oui

On déduit donc de ce tableau, que si l'on sous-échantillonne à la fréquence $f_e = f_0/M$, l'harmonique utilisée pour opérer la démodulation IQ valant M ($f_0 = M.f_e$), il faut **M impair**. De plus, selon que M s'écrit 1+4.i ou 3+4.i, il faudra corriger d'un signe '– ' la partie imaginaire de l'enveloppe complexe Q'.

La structure de démodulation est exactement la même que pour l'échantillonnage à f_0.

Annexe

1. Représentation analytique des signaux réels

Soit un signal réel, que l'on note x(t).

On lui associe une autre représentation, dite représentation analytique définie par :

$$z_x(t) = x(t) + j.\hat{x}(t)$$ ***représentation analytique du signal x(t)***.

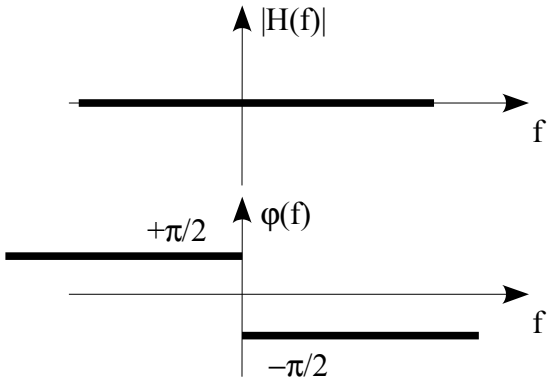
Il s'agit d'un nombre complexe composé pour partie réelle du signal x(t) et pour partie imaginaire de la **transformée de Hilbert** de x(t).

Remarque : $z_x(t)$ n'a pas de réalité physique, mais nous verrons que cette notation est pratique pour l'analyse des signaux à bande étroite.

2. Transformée de Hilbert

La transformée de Hilbert d'un signal réel x(t) a pour effet de déphaser de $\pi/2$ (retard) chacune des composantes spectrales de X(f). Le module vaut 1.

Sa fonction de transfert, H(f), peut se représenter par :



La fonction de transfert peut s'exprimer par $H(f) = -j.sign(f)$,

où *sign*(x) vaut +1 si x>0 et -1 si x<0.

Remarque : La transformée de Hilbert peut être vue comme un filtre linéaire. Ceci dit, celui-ci n'est pas réalisable car non causal. Il peut être approximé dans une bande de fréquence donnée.

Exemple : $x(t)=A.cos(2.\pi.f_x.t+\varphi)$, sa transformée de Hilbert s'écrit, par définition :

$$\hat{x}(t)=A.cos(2.\pi.f_x.t+\varphi-\frac{\pi}{2})=A.sin(2.\pi.f_x.t+\varphi)$$

On en déduit la représentation analytique (bien connue) :

$$z_x(t)=x(t)+j.\hat{x}(t)=A.cos(2.\pi.f_x.t+\varphi)+j.A.sin(2.\pi.f_x.t+\varphi)=A.e^{j.(2.\pi.f_x.t+\varphi)}$$

3. Transformée de Fourier d'un signal analytique

Déterminons cette représentation spectrale pour un signal réel $x(t)=A.cos(2.\pi.f_x.t+\varphi)$. Nous savons que sa représentation analytique est $z_x(t)=A.e^{j.(2.\pi.f_x.t+\varphi)}$

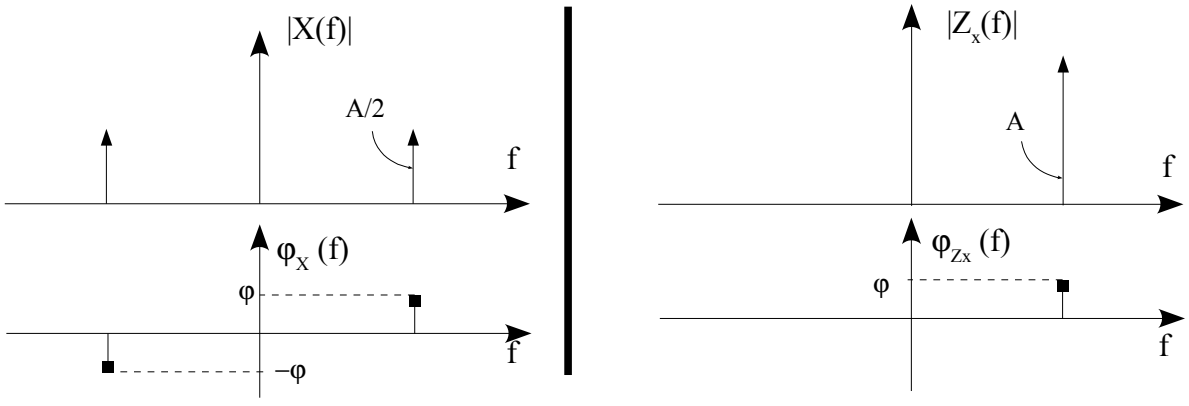
La transformée de Fourier de $e^{j.(2.\pi.f_x.t+\varphi)}$ est $\delta(f-f_x).e^{j.\varphi}$

Ainsi $z_x(t)=A.e^{j.(2.\pi.f_x.t+\varphi)}$ donne $Z_x(f)=A.\delta(f-f_x).e^{j.\varphi}$

Rappelons que pour le signal réel :

$x(t)=A.cos(2.\pi.f_x.t+\varphi)$ donne $X(f)=A.\delta(f+f_x).e^{-j.\varphi}+A.\delta(f-f_x).e^{j.\varphi}$

Comparons les deux spectres :



Signal réel *Signal analytique*

Les observations immédiates que l'on peut faire sont :

- La représentation spectrale d'un signal analytique est **monolatérale,** seul le côté positif des fréquences est représenté. Inversement, la représentation spectrale d'un signal réel est bilatérale (module pair, phase impaire : symétrie hermitienne)
- Le Dirac représentant la raie spectrale de fréquence f_x pèse **deux fois plus** que ceux qui apparaissent dans le spectre bilatéral du signal réel.

Cas général :

Appelons $\hat{X}(f)$ la transformée de Fourier de $\hat{x}(t)$.

$Z_x(f)=\mathcal{F}(x(t)+j.\hat{x}(t))=X(f)+j.\hat{X}(f)$ et $\hat{X}(f)=-j.sign(f).X(f)$

Pour f > 0, $\hat{X}(f)=-j.X(f)$ soit $Z_x(f)=X(f)+X(f)=2.\mathrm{X}(f)$

et pour f < 0, $\hat{X}(f)=j.X(f)$ soit $Z_x(f)=X(f)-X(f)=0$

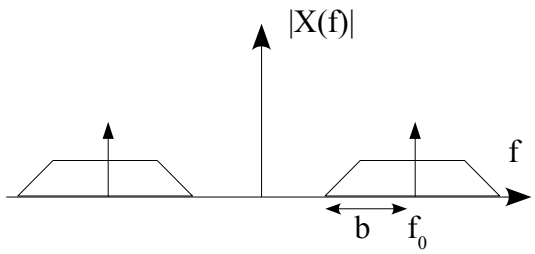
Finalement, on peut donc généraliser en écrivant directement pour un signal analytique :

$Z_x(f)=2.\mathrm{U}(f).X(f)$ où U(f) est l'échelon unitaire.

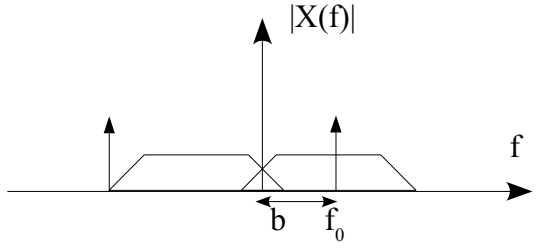
Cette relation traduit clairement le fait que $Z_x(f)$ est asymétrique unilatéral, avec la mise à l'échelle d'un facteur 2.

4. Enveloppe complexe : signal à bande étroite

L'enveloppe complexe est une représentation qui est directement associée aux signaux analytiques. Elle est précieuse lorsque l'on étudie les modulations d'amplitude et de phase (MAQ, QPSK...). Elle est adaptée aux signaux dits “à bande étroite”. Cela signifie que la bande passante du message est très inférieure à f_0.



Ici, $b<f_0$, on peut associer directement un signal analytique



Dans ce cas, on ne peut pas faire correspondre simplement un signal analytique : il faut $f_0>b$

Ainsi, on ne parlera d'enveloppe complexe que pour des signaux à bande étroite.
Prenons l'expression générale qui définit un signal modulé à la fois en amplitude et en phase :

$x(t)=a(t).\cos(2.\pi.f_0.t+\varphi(t))$, ou a(t) et φ(t) sont réels. De plus a(t) >0.

Il s'écrit aussi :

$$x(t)=a(t).\frac{e^{j(2.\pi.f_0.t+\varphi(t))}}{2}+a(t).\frac{e^{-j(2.\pi.f_0.t+\varphi(t))}}{2}=a(t).e^{j.\varphi(t)}.\frac{e^{j.2.\pi.f_0.t}}{2}+a(t).e^{-j.\varphi(t)}.\frac{e^{-j.2.\pi.f_0.t}}{2}$$

En prenant la transformée de Fourier :

$$X(f)=\mathcal{F}[a(t).e^{j.\varphi(t)}]\otimes\frac{1}{2}.\delta(f-f_0)+\mathcal{F}[a(t).e^{-j.\varphi(t)}]\otimes\frac{1}{2}.\delta(f+f_0)$$

Comme le signal est à bande étroite, le premier terme, qui correspond aux fréquences positives, ne déborde pas du côté négatif des fréquences. Ainsi, par définition du signal analytique, $Z_x(f)=2.\mathrm{U}(f).X(f)$, il vient

$Z_x(f)=\mathcal{F}[a(t).e^{j.\varphi(t)}]\otimes\frac{1}{2}.\delta(f-f_0)$, ce qui donne par transformée de Fourier inverse le signal analytique associé :

$$z_x(t)=a(t).e^{j.\varphi(t)}.e^{j.2.\pi.f_0.t}$$

En exprimant $z_x(t)$ sous la forme partie réelle et partie imaginaire (transformée de Hilbert), on obtient :

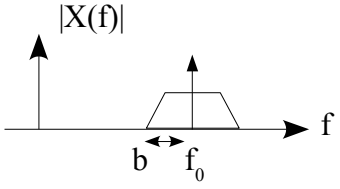
$$z_x(t)=x(t)+j\hat{x}(t)=a(t).\cos(2.\pi.f_0.t+\varphi(t))+j.a(t).\sin(2.\pi.f_0.t+\varphi(t))$$

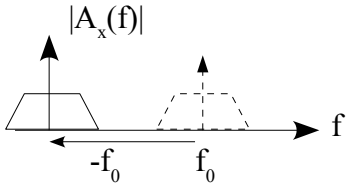
La boucle est bouclée, on retrouve bien $\Re(z_x(t)) = a(t).\cos(2.\pi.f_0.t+\varphi(t))$

Interprétation de l'enveloppe complexe :

L'enveloppe complexe que l'on note $\alpha_x(t)= a(t)e^{j\varphi(t)}$, est un nombre complexe : $z_x(t)=\alpha_x.e^{j.2.\pi.f_0.t}$

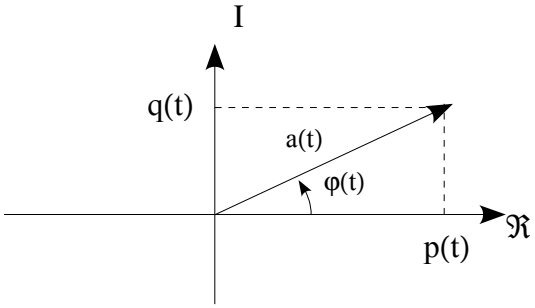
- qui représente à l'instant t, l'amplitude et le déphasage de la porteuse $\cos(2.\pi.f_0.t)$
- dont la transformée de Fourier, $A_x(f)$ est la translation de $-f_0$, du signal analytique associé au signal à bande étroite centré sur f_0.





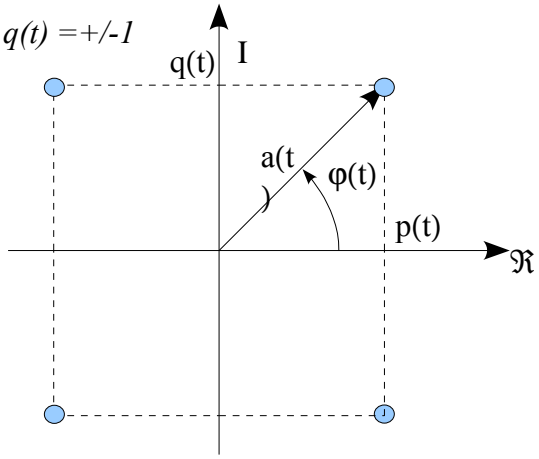
NB: L'enveloppe complexe n'est pas un signal analytique, puisque son spectre est réparti également sur les fréquences négatives

L'enveloppe complexe peut se représenter dans le plan complexe (et c'est l'intérêt !) :



Pour une modulation de phase et d'amplitude (MAQ), l'enveloppe complexe, observée sur un temps suffisamment long (l'ensemble des symboles possible ayant pu défiler), devient le diagramme de constellation.

MAQ-4 : p(t) =+/-1, q(t) =+/-1



Précisons : $p(t)=a(t).\cos(\varphi(t))$ et $q(t)=a(t).\sin(\varphi(t))$

5. Résumé signaux réels / signaux analytiques

Signaux réel – x(t) à bande étroite **Signal analytique $z_x(t)$**

$$x(t)=a(t).\cos(2.\pi.f_0.t+\varphi(t)) \quad \xrightarrow{z_x(t)=x(t)+j.\hat{x}(t)} \quad z_x(t)=a(t).e^{j.\varphi(t)}.e^{j.2.\pi.f_0.t}$$

$$\xleftarrow{x(t)=R[z_x(t)]}$$

$$X(f)=\mathcal{F}[s(t)] \quad \xrightarrow{Z_x(f)=2.U(f).X(f)} \quad Z_x(f)=\mathcal{F}[z_x(t)]$$