CLIP V1 & V2

Réception série asynchrone sur MSP430

Les liaisons au format série asynchrone sont très courantes. Par exemple :
- en courtes distances (qq mètres) :
  - liaisons PC ↔ PC
  - liaisons PC ↔ périphériques lents (imprimantes, modem, …)
- en longues distances par fil : on utilise le réseau téléphonique ce qui nécessite une modulation (V21, V23, ect.)
- en courtes distances par infrarouge (télécommandes)
- en longues distances par radio (modulation nécessaire).

Avantages :
– Facilité de mise en œuvre
– Bonne immunité au bruit

Inconvénients :
– Vitesse inférieure au format série synchrone

1. Format série asynchrone

Les messages à transmettre sont constitués d'un ensemble de bits structurés de façon très précise (fichier de données informatiques par exemple). Cette structure est accompagnée de règles complémentaires nécessaires au transfert entre l'émetteur et le récepteur. Bien entendu, les mêmes règles doivent utilisées des 2 côtés du canal de transmission.
Pour éviter une certaine anarchie, des organisations internationales (ETSI, CCITT, …) ont établi des règles rigoureuses décrites dans des documents appelés "standards" ou "recommandations".

Pour faciliter le traitement du message, les bits constitutifs sont regroupés en mots de 5 bits (code BAUDOT), 7 bits (code Ascii) ou par octets. Ces éléments de base sont nommés **symbole**.

Dans toute liaison série (synchrone ou asynchrone), le récepteur doit être capable de reconstituer le message dans son entier alors que les bits lui arrivent les uns après les autres.
Dans le format synchrone, cette opération nécessite d'ajouter au message utile des informations complémentaires dont le traitement dans le récepteur est assez complexe.
Dans le cas du **format asynchrone**, la reconstitution du message est facilitée car chaque symbole est accompagné de bits complémentaires de synchronisation et de contrôle :
- bit **Start** : indicateur du début de la transmission d'un symbole,
- bit **Stop** : indicateur de fin de transmission d'un symbole (dans certains cas la durée de cet indicateur est rallongée : on place 1,5 ou 2 bits Stop),
- bit de **Parité** (optionnel) : précède le bit Stop. Permet la détection d'une erreur de transmission.

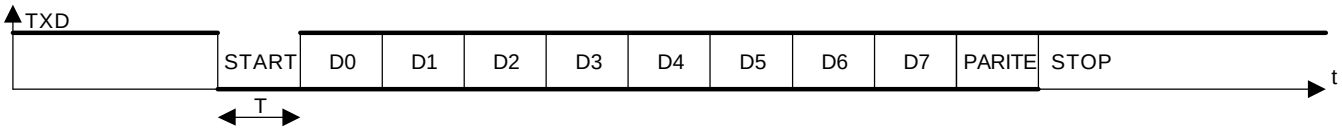
L'ensemble des bits : Start + Données + Parité + Stop est appelé : **trame**.

1.1 Chronogrammes typiques

On suppose que les symboles ont une longueur de 8 bits (cas le plus fréquent).
Identification des signaux caractéristiques :
- TXD : signal binaire transmis par l'émetteur
- TXC : horloge de transmission de l'émetteur (signal non transmis)
- RXD : signal binaire reçu par le récepteur. En principe identique à TXD s'il n'y a pas d'erreur
- RXC : horloge de réception reconstituée dans le récepteur

1.1.1 Transmission d'un symbole isolé

Un symbole est dit "isolé" quand il n'est ni précédé, ni suivi de la transmission d'un autre symbole.



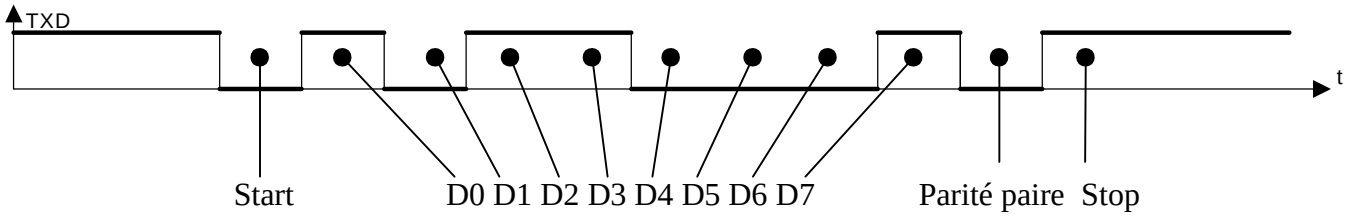
Le signal TXD est à "1" au repos, c'est à dire en l'absence de toute transmission (signal MARK).
Le bit "Start" est toujours à "0" et le bit "Stop" toujours à "1".
Les bits du symbole sont D0 à D7. D0 est toujours transmis en premier
Bit de parité :

- si on choisit une parité paire : état du bit de parité tel que le nombre d'états "1" (sur D0 à D7, y compris le bit de parité) est pair
- si on choisit une parité impaire : état du bit de parité tel que le nombre d'états "1" (sur D0 à D7, y compris le bit de parité) est impair

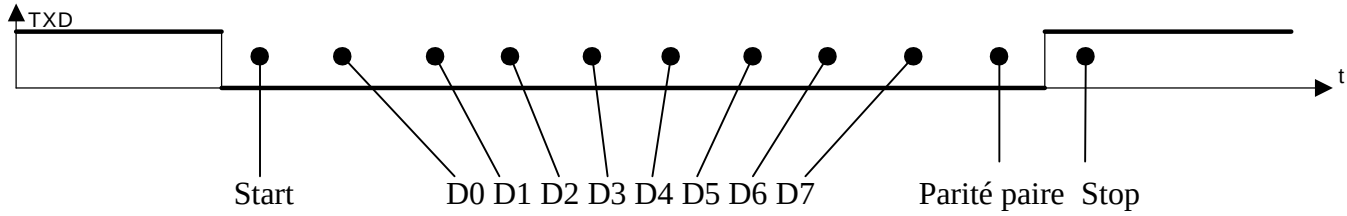
La durée T définit le débit de moments (nombre maximum de changements d'états par seconde) :

$$R = \frac{1}{T} \quad \text{(en Bauds)}$$

Exemples de trames



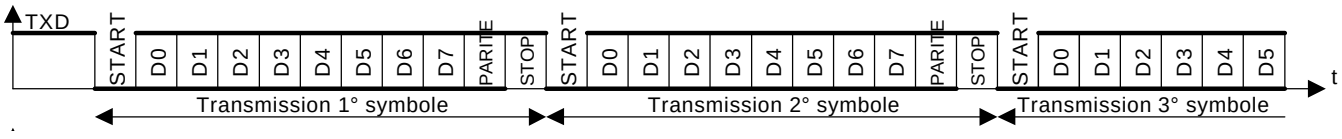
Le symbole transmis vaut ici 10001101b = 8Dh (poids fort à gauche)
Le nombre de "1" (y compris le bit de parité) est pair : CQFD



Le symbole transmis vaut ici 00000000b = 00h
Le nombre de "1" (y compris le bit de parité) est pair (il est nul) : CQFD

Note : ce dernier chronogramme est remarquable : à l'exception des bits Start et Stop, le signal TXD ne présente aucun changement d'état ! Le récepteur doit être capable d'identifier les 8 bits du symbole malgré cela.

1.1.2 Transmission de plusieurs symboles sans temps mort



On comprend, avec ce chronogramme, la présence nécessaire du bit Stop. Grâce à lui, il y a toujours un flanc descendant sur TXD au début de chaque bit Start. Si le bit Stop était absent, le bit de parité, ou D7 quand on ne transmet pas la parité, serait le dernier bit d'une trame. Celui-ci peut être à "0" et le bit Start ne crée alors pas de changement d'état et ne peut alors être identifié.
Le flanc descendant de TXD entre les bits Stop et Start est exploité dans le récepteur pour reconstituer l'horloge RXC (voir plus loin).

Exemple

Exercice : Identifier les bits de chaque symbole
Déterminer les valeurs des symboles transmis

Note : si le récepteur n'est pas adapté au format de transmission (par exemple : 8 bits, parité paire et 1 bit Stop), il va très rapidement faire des confusions. Par exemple, si le récepteur s'attend à recevoir des données au format : 7 bits, pas de parité et 1 bit Stop, les 7 premiers bits du premier symbole seront correctement identifiés et traités, mais il va interpréter le bit D7 comme un bit Stop et le bit de parité comme un éventuel bit Start. La suite de la transmission est perturbée !
Note : dans ce cas (pas de temps morts entre 2 transmissions), le débit binaire "utile" est maximum. Pour l'exemple, il vaut : $\frac{1}{T} \times \frac{8}{11}$ bits/S

1.2 Formats standards

Bien que standards, les formats sont nombreux. En effet, on peut choisir :

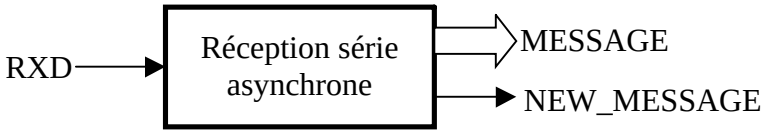
- Le nombre de bits du symbole : 5 , 7 ou 8
- La présence et le type de bit de parité : absent, parité paire, parité impaire
- La durée minimum du bit Stop (il n'y a pas de maximum !) : 1T, 1,5T ou 2T
- Le débit de moments: 110, 330, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bauds

2. Réalisation du récepteur série asynchrone

Pour l'application CLIP, le canal de transmission est le réseau téléphonique. Une modulation FSK est nécessaire pour satisfaire les contraintes électriques du réseau (bande passante, niveaux). Celle-ci respecte les recommandations V23.
Une partie du logiciel implanté dans le MSP430 réalise cette fonction de démodulation FSK. Celle-ci produit le signal RXD normalement identique à TXD en l'absence d'erreur de transmission. Suivant la recommandation V23, RXD est conforme au format suivant :

- 8 bits
- Pas de parité
- 1 bit Stop
- 1200 bauds

La fonction à réaliser reçoit le signal RXD et sauvegarde les symboles (ou octets) du message dans la variable MESSAGE.



MESSAGE : message reçu (tous les champs du message)
NEW_MESSAGE : indique la réception d'un nouveau message

Pour simplifier, on n'envisage pas de produire des signaux d'identification d'erreurs de format.

2.1 Algorithme

Pour recevoir chaque symbole du MESSAGE, la fonction "réception série asynchrone" doit :

- Détecter les champs SMMR et MARK (voir document STI4)
- Détecter les bits Start et Stop de chaque transfert pour identifier les bits utiles de chaque symbole du message

- Mettre en phase le signal d'horloge RXC pour échantillonner de façon fiable tous les bits transmis
- A chaque trame : accumuler les bits utiles reçus dans un registre à décalage

Pour recevoir l'ensemble du message, la fonction doit aussi :

- Identifier le champ "Longueur du message" (voir STI4) et mémoriser cette information
- Affecter la variable MESSAGE avec tous les octets du message jusqu'à réception du dernier
- Affecter alors l'indicateur NEW_MESSAGE

2.1.1 Choix techniques

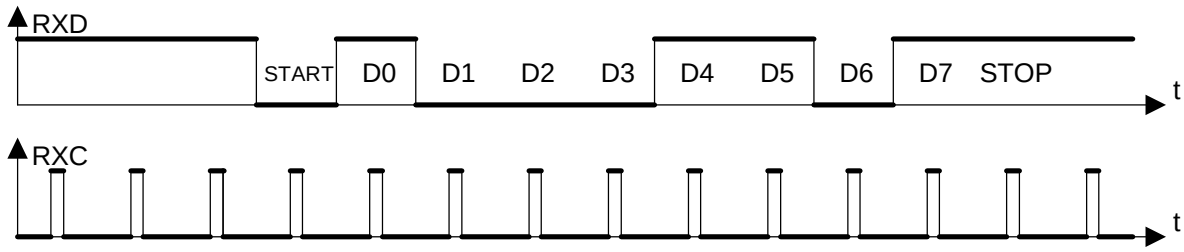
Les versions de MSP430 utilisées ne comportent pas de périphérique dédié à cette fonction (généralement nommée UART : Universal Asynchronous Receiver/Transmitter). Elle doit donc être réalisée par le logiciel.
Toutefois, on utilise les structures matérielles du "Timer A" pour réduire le logiciel mais aussi rendre la fonction plus performante.
Certaines parties du Timer A sont déjà utilisées pour d'autres fonctions (démodulation FSK notamment). Mais on dispose des structures "Capture/Compare Register CCR2" à condition de ne pas modifier le mode de fonctionnement du compteur TAR.

Pour réaliser la fonction "Démodulation FSK", le "Timer A" est configuré comme suit :

- Horloge compteur TAR = ACLK = 4Mhz
- Mode de fonctionnement compteur TAR : "continuous up"
- Structure "Capture/Compare Register CCR0" utilisée pour produire des interruptions au rythme de 1700x8 = 13600 Hz (en fait 4000000/294 = 13605Hz exactement). Ces interruptions lancent le programme "TIM_A_Int" qui réalise la démodulation FSK.

La fonction "réception série asynchrone" est incluse dans le programme d'interruption "TIM_A_Int", mais n'est activée qu'une fois sur 2. Son rythme d'activation est donc de 6800 par seconde.

La structure "Capture/Compare Register CCR2" est utilisée pour produire le signal d'horloge RXC nécessaire pour indiquer les instants d'échantillonnage idéaux de RXD, à savoir : au milieu de chaque bit comme le montre le chronogramme ci-dessous.



La fréquence du signal RXC est de 1200Hz et sa phase doit être contrôlée par rapport à RXD pour indiquer les "milieux" de chaque bit reçu. On dispose ainsi d'une certaine marge d'erreur quand le signal RXD n'est pas aussi parfait que représenté ci-dessus.

Principe de production de RXC :

- Le signal RXC est en fait l'indicateur CCIFG du registre CCTL2 du Timer A. Celui-ci est mis à "1" quand le compteur TAR atteint l'état du registre CCR2.
- Cet indicateur est testé dans le programme d'interruption "TIM_A_Int", donc 6800 fois par seconde. Ce rythme est suffisant vis à vis de la fréquence de RXC.
- Quand il est détecté à "1" :
  - le logiciel prend en compte l'état correspondant de RXD, si une acquisition est en cours.
  - le logiciel affecte le registre CCR2 pour que la prochaine mise à "1 se produise dans (1/1200) s
  - l'indicateur CCIFG est mis à "0"

- La mise en phase est réalisée à chaque changement d'état de RXD : le registre (CCR2) est affecté pour que la prochaine mise à "1" de RXC=CCIFG se produise dans (1/2400) s, soit une ½ période de 1200Hz (au "milieu" du bit).

2.1.2 Algorithme

La fonction utilise un certain nombre de variables et d'indicateurs :

Variables :

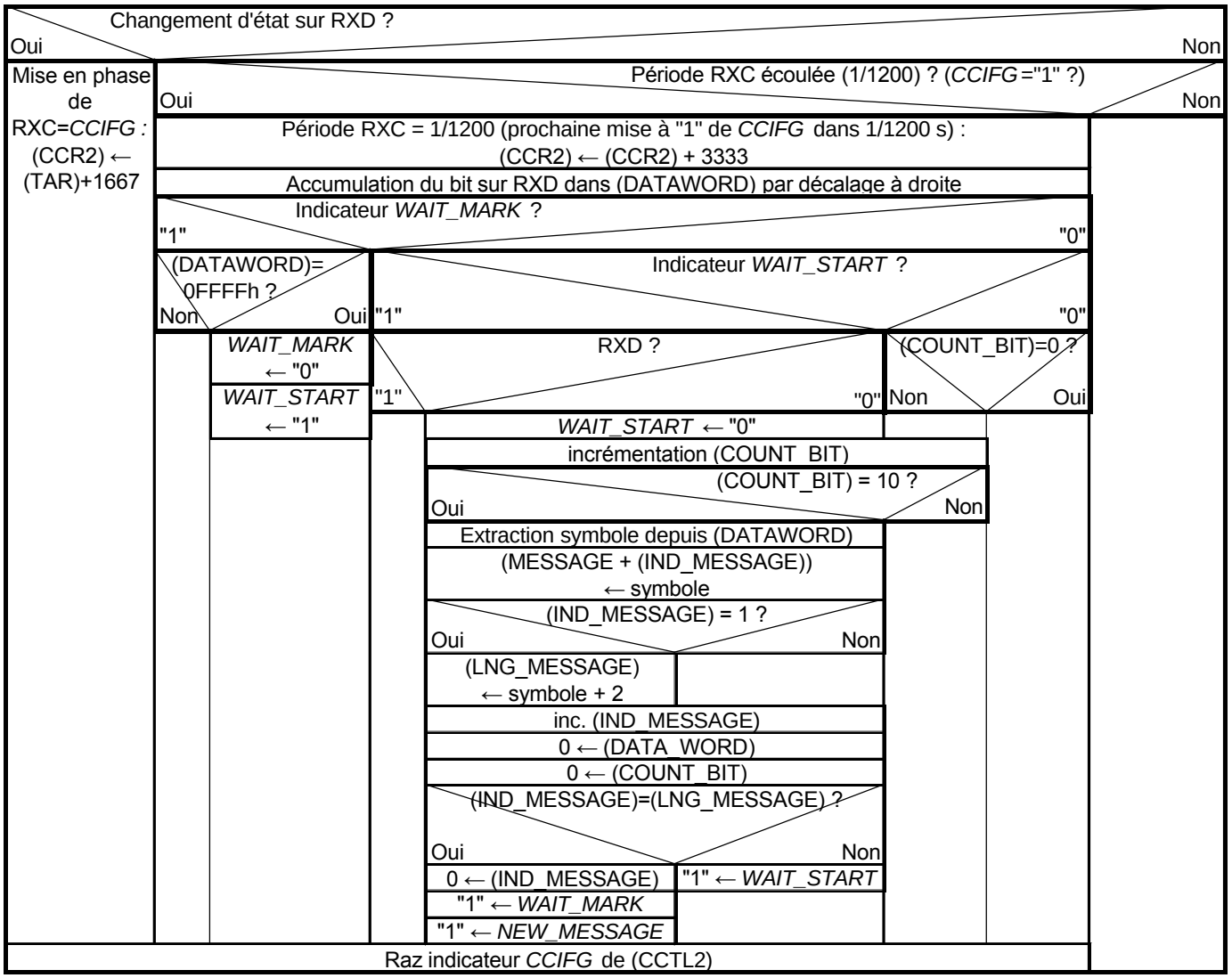
- (MESSAGE) : variable de xx octets destinée à recevoir l'ensemble des symboles du message
- (IND_MESSAGE) : variable de 2 octets mémorisant la position (l'index) d'un symbole dans (MESSAGE). Sa valeur évolue entre 0 et (xx – 1)
- (DATAWORD) : variable sur 2 octets (16 bits) utilisée pour accumuler tous les bits d'une trame.
- (COUNT_BIT) : variable sur 2 octets (16 bits) utilisée pour compter les bits reçus dans une trame. Permet d'identifier le bit STOP.

Indicateurs :

Tous les indicateurs sont placés dans le registre R13 du CPU. Ce registre a une longueur de 16 bits, ce qui permet d'y placer jusqu'à 16 indicateurs binaires. L'indice choisi pour chaque indicateur est arbitraire.

- WAIT_MARK : indique la phase d'attente du signal MARK (ligne RXD au repos : état "1" permanent).
- WAIT_START : indique la phase d'attente du bit START
- NEW_MESSAGE : indique qu'un nouveau message a été reçu. Cet indicateur est utilisé pour rafraîchir l'affichage.

Réception série asynchrone



Notation :

- Variables notées sans parenthèses : l'identificateur représente son adresse
- Variables notées avec des parenthèses : l'identificateur représente sa valeur (contenu de l'adresse)
- Indicateurs : notés en italique. L'identificateur représente toujours son état

Commentaires :

- **États initiaux des variables et indicateurs** :
  - (MESSAGE) : quelconque
  - (IND_MESSAGE) : 0
  - (DATAWORD) : 0
  - (COUNT_BIT) : 0
  - WAIT_MARK : "1"
  - WAIT_START : "0"
  - NEW_MESSAGE : "0"
- **Signal RXD**
  Il s'agit en fait d'un signal virtuel. Il n'existe pas physiquement.
  La fonction "réception série asynchrone" est placée dans le programme d'interruption "TIM_A_Int" juste après la fonction "Démodulation FSK". Le résultat de cette fonction se trouve dans les registres R11 et R14 codés en complément à 2 sur 16 bits :
  - R11 : échantillon actuel du signal numérique démodulé
  - R14 : échantillon calculé à l'interruption précédente (retard d'une période d'échantillonnage)

  Le signal RXD correspond au signe du signal numérique de l'échantillon actuel : si R11 est positif alors RXD="1" et si R11 est négatif alors RXD="0".
  Pour déterminer un changement d'état de RXD, on compare le signe des 2 échantillons R11 et R14 : s'il a changé, alors le signal numérique démodulé est passé par 0 (en croissance ou décroissance).
- **Mise en phase de RXC = CCIFG**
  Quand un changement d'état de RXD est détecté (bit Start ou bit de donnée), le signal RXC doit passer à "1" une ½ période T après cet événement pour correspondre ainsi au "milieu" du bit. Or l'indicateur CCIFG de (CCTL2) passe à "1" quand le compteur (TAR) atteint (CCR2).
  Dans cette application, le compteur s'incrémente au rythme d'une horloge de 4MHz. Une ½ période T vaut 416,66µS (pour 1200 bauds), ce qui représente 1667 périodes d'horloge 4MHz.
  Pour obtenir le résultat attendu, il faut donc :
  - prendre l'état du compteur TAR au changement d'état de RXD : TAR1
  - affecter le registre CCR2 avec TAR1 + 1667
  - par prudence (c'est généralement inutile) : mettre l'indicateur CCIFG à "0"
- **Période RXC écoulée ?**
  Il s'agit de détecter l'instant idéal de prise en compte de RXD (milieu des bits). Pour cela, on teste simplement la mise à "1" de l'indicateur CCIFG (= RXC). Ces tests sont effectués 6800 fois par seconde, ce qui est suffisant compte-tenu du débit de moments (1200 bauds).
- **Traitements à la détection d'une période RXC**
  L'action principale est évidemment la prise en compte de l'état de RXD, mais il faut aussi poursuivre le traitement des bits précédents et gérer les phases de fonctionnement.
  - Fixer la période de RXC : la prochaine mise à "1" de CCIFG doit se produire (1/1200) seconde après la précédente. Pour cela, il suffit d'augmenter la valeur de (CCR2) de 3333, car 3333 périodes de 4MHz = 833,33µS.
  - L'état de RXD est obtenu en testant le signe du registre R11. Le résultat est accumulé dans la variable (DATA_WORD) par décalage à droite (la transmission commence par le poids faible). On accumule tous les bits de la trame reçue : y compris Start et Stop.

  Le traitement suivant dépend de la phase de fonctionnement : attente du signal MARK, attente du signal START et accumulation des bits :
  - Si "**attente du signal MARK**" (indicateur WAIT_MARK à "1"). Cette phase est active pendant l'attente d'un nouveau message :

  - Le traitement consiste en fait à détecter cet événement pour passer à la phase suivante (attente du signal START). On admet que le signal MARK est présent si les 16 derniers bits reçus sont tous à "1" (en fait la norme en prévoit 180 environ). Les 16 derniers bits reçus sont accumulés dans (DATA_WORD), il suffit donc de comparer cette variable avec 0FFFFh.
  - Quand le signal MARK est détecté, l'indicateur WAIT_MARK est mis à "0" et l'indicateur WAIT_START à "1" pour indiquer le changement de phase.
  - On termine en mettant l'indicateur CCIFG à "0"
- Si "**attente du signal START**" (indicateur WAIT_START à "1" et variable (COUNT_BIT) à 0). Cette phase existe en attendant la 1° trame et pendant les intervalles de temps séparant les trames suivantes.
  - Le traitement consiste en fait à détecter cet événement pour passer à la phase suivante (accumulation des bits). Cela est réalisé simplement en testant l'état de RXD qui doit être à "0"
  - Dans ce cas, l'indicateur WAIT_START est mis à "0" et la variable (COUNT_BIT) est incrémentée pour indiquer la nouvelle phase : accumulation des bits
- Si phase "**accumulation des bits**" (WAIT_START = "0" et (COUNT_BIT) <> 0).
  - On incrémente (COUNT_BIT) pour compter les bits reçus (sauf si bit Start, car c'est déjà fait)
  - Il n'y a pas d'autre traitement à effectuer jusqu'au bit Stop. L'arrivée du bit Stop correspond à (COUNT_BIT) = 10.
  - Si (COUNT_BIT) = 10, la trame complète est reçue et les 10 bits se trouvent dans les 10 positions de poids fort de (DATA_WORD) :
    - On extrait l'octet de données depuis (DATA_WORD) et on le range dans la position actuellement pointée dans (MESSAGE). L'adresse de la position pointée est obtenue en calculant MESSAGE + (IND_MESSAGE)
    - Le 2° octet d'un message donne sa longueur à l'exclusion du 1° et du dernier. Pour identifier cet octet de longueur, on teste (IND_MESSAGE) qui doit être à "1". Dans ce cas on ajoute la valeur 2 à la longueur avant de la mémoriser dans (LNG_MESSAGE).
    - On incrémente (IND_MESSAGE) et on initialise les variables (DATA_WORD) et (COUNT_BIT) pour préparer la réception de la prochaine trame.
    - La dernière trame est détectée si (IND_MESSAGE) = (LNG_MESSAGE).
      - Si ce n'est pas le cas : on place l'indicateur WAIT_START à "1" pour préparer le logiciel à l'arrivée de la prochaine trame.
      - Si c'est le cas : (IND_MESSAGE) est mis à 0 et l'indicateur WAIT_MARK est mis à "1" pour revenir à la phase initiale d'attente du signal MARK. L'indicateur NEW_MESSAGE est aussi positionné pour indiquer la disponibilité d'un nouveau message.
  - On termine en mettant l'indicateur CCIFG à "0"

2.1.3 Programme

```
************************************************************
* Réception série asynchrone au format V23 sur MSP430F413 *
************************************************************
* Extrait du programme de démodulation FSK
* (cette fonction a été effacée dans cet extrait)

#include        "msp430x11x1.h"  ;
                col 130
**************
* Constantes *
**************
WDT_WRKEY       equ 05A00h
MYSTACK         set 300h     ;280h start of system stack
```

```
********************
* Variables en RAM *
********************
               ORG  0200h
DATA_WORD      ds  2       ;usect "FILTMEM",2
COUNT_BIT      ds  2       ;Compteur de bits de l'UART
IND_MESSAGE    ds  2       ;Index de l'octet reçu ds MESSAGE
LNG_MESSAGE    ds  2       ;Pour mémoriser la longueur des messages
MESSAGE        ds  32      ;Taille max des messages = 32 octets
FIN_MESSAGE    equ $

*******************************
* Registres à usage exclusif : *
*******************************
*   R11 : bit_data (sortie du filtre numérique du démodulateur FSK)
*   R13 : indicateurs divers
*   R14 : valeur précédente de R11

****************************
* Indicateurs du registre R13
****************************
INTERRUPT_TOGGLE    set 1
WAIT_MARK           set 2  ;Indique la phase d'attente du signal MARK
WAIT_START          set 4  ;Indique la phase d'attente d'un bit Start
NEW_MESSAGE         set 8  ;Indique un nouveau message LCD

***********************************************
* System Init
***********************************************
            org   0f000h
Start       mov   #MYSTACK,SP           ;initialize system stack pointer
            mov   #(WDTHOLD+WDT_wrkey),&WDTCTL ;Stop Watchdog Timer

            bis.b #XTS,&BCSCTL1         ;Validation XT1 hautes fréquences
            bic   #OSCOFF,SR            ;Validation XT1 (par prudence)
Attend1     bic.b #OFIFG,&IFG1          ;Raz flag "Oscfault"
            mov   #0FFFh,R15            ;
Attend2     dec   R15                   ; > Délai de stabilisation de l'oscillateur
            jnz   Attend2               ;/
            bit.b #OFIFG,&IFG1          ;Test indicateur "Oscfault"
            jnz   Attend1

            mov.b #11001000b,&BCSCTL2   ;MCLK = SMCLK = XT1CLK

            mov   #0220h,&tactl         ;Timer A:SMCLK, continuous up. Pas d'int TAIE

            mov   #2010h,&cctl0         ;Output Capture avec interruption
            mov   #294-1,&ccr0          ;Périodes int=4000000/294=13605,4Hz
*                                        (environ 8x1700Hz)

            mov   #2080h,&cctl2         ;Horloge RXC=1200 Hz avec TA2 sans interrupt.
            mov   #3333,&ccr2           ;Périodes int=4000000/3333=1200Hz

            mov  #0,COUNT_BIT           ;Compteur de bits de l'UART
            mov  #0h,&DATA_WORD         ;

            mov   #MESSAGE,R4           ; |
NEXT_RAZ    mov   #0,0(R4)              ; |
            add   #2,R4                 ;  > Raz (MESSAGE)
            cmp   #FIN_MESSAGE,R4       ; |
            jne   NEXT_RAZ              ;/

            bis   #WAIT_MARK,R13   ;On commence par la phase d'attente du signal MARK
            mov   #0,IND_MESSAGE   ;Initialisation du pointeur des messages reçus
```

```
            eint                        ;Validation des interruptions

WAIT_LOOP   jmp  WAIT_LOOP ;Boucle sans fin. La démodulation est réalisée dans le
                           ;programme d'interruption

*****************************************
* Programme d'interruption activé par TA0  *
*****************************************
* Rythme : 13600 Hz
* Fonctions: 2 séries fct. sont réalisées alternativement (donc au rythme de 6800 Hz)
*    - Si l'indicateur INTERRUPT_TOGGLE est à "1" : do_ADC
*         - INTERRUPT_TOGGLE = "0"
*         - Acquisition échantillon (Input Capture sur CCR1)
*         - Raz rampe
*         - Multiplication avec l'échantillon précédent : résultat dans R9
*    - Si l'indicateur INTERRUPT_TOGGLE est à "0" : filters
*         - INTERRUPT_TOGGLE = "1"
*         - Une étape du filtrage numérique de R9 : résultat dans R11
*         - Réception série asynchrone de l'éventuel message : résultat ds MESSAGE
* Registres utilisés : R6-R7-R8-R9-R10-R11-R13-R14-R15
TIM_A_Int       add #294,&CCR0 ;Périodes int=4000000/294=13605,4Hz (environ 8x1700Hz)
                bit #INTERRUPT_TOGGLE,R13 ;
                jz  filters

* Acquisition échantillon, raz rampe et déphaseur
***********************************************
* Résultat dans registre R9
do_ADC          bic #INTERRUPT_TOGGLE,R13 ;A la prochaine interruption : filtrage
*                                          numérique
*********************************************
* Le traitement a été volontairement effacé *
*********************************************
                reti

* Filtre passe-bas du démodulateur FSK
**************************************
* Résultats dans registres R11 (échantillon actuel) et R14 (échantillon précédent)
filters         bis #INTERRUPT_TOGGLE,R13
*********************************************
* Le traitement a été volontairement effacé *
*********************************************

* Réception série asynchrone au format V23
******************************************
RX_1200BAUDS    mov R14,R6          ;Avant-dernier échantillon du filtre dans R6
                mov R11,R14         ;Échantillon actuel de la sortie du filtre ds R14
                xor R11,R6          ; > Test changement d'état du bit de signe entre
                bit #8000h,R6       ;/  2 calculs (1/6800 de s)
                jz  Test_1200Hz
                mov &TAR,R6         ;|
                add #1667,R6        ; > Changement d'état détecté => mise en phase
                mov R6,&CCR2        ;/  de l'horloge 1200 Hz (1667 = 4000000/2400)
                jmp New_Sample_F
Test_1200Hz     bit #1,&CCTL2       ;Période 1200 Hz écoulée ?
                jz  exit_D_A
                add #3333,&CCR2     ;Oui => prochain échantillon ds 1 période 1200 Hz
                bit #8000h,R11      ;Bit de signe dans C, soit C!=DATA
New_DATA        rrc DATA_WORD       ;Accumulation du bit dans DATA_WORD
                bit #WAIT_MARK,R13  ;Phase d'attente du signal MARK ?
                jz  Test_START
Test_MARK       cmp #0FFFFh,DATA_WORD   ;Réception de "1111111111111111" ?
                jne New_Sample_F
                bic #WAIT_MARK,R13  ;Acquittement
                bis #WAIT_START,R13 ;Oui => on attend le bit Start du premier octet
                jmp New_Sample_F
```

```
Test_START      bit #WAIT_START,R13  ;Phase d'attente du bit START ?
                jz  Test_STOP
                bit #8000h,DATA_WORD ;Test état du dernier bit reçu
                jz  New_Sample_F     ;"1" -> ne rien faire
                bic #WAIT_START,R13  ;"0" -> acquittement
                jmp New_bit
Test_STOP       tst COUNT_BIT        ;Réception en cours ?
                jeq New_Sample_F
New_bit         inc COUNT_BIT
                cmp #10,COUNT_BIT    ;Dernier bit d'un transfert (Start + 8 bits +
*                                     Stop) ?
                jne New_Sample_F
                mov DATA_WORD,R7
                rlc R7               ;Gommer le bit Stop
                swpb R7
                xor.b #0ffh,R7       ;Inverser les bits
                mov IND_MESSAGE,R6   ;Index dans MESSAGE
                mov.b R7,MESSAGE(R6) ;Sauvegarde de l'octet ds MESSAGE
                cmp #1,R6            ;Octet "longueur du message" ?
                jne TO_INC_PTR
                add #2,R7
                mov.b R7,LNG_MESSAGE ;Sauvegarde de la longueur du message
TO_INC_PTR      add #1,R6            ;Pour pointer l'octet suivant
NEW_BYTE        clr DATA_WORD        ; > Pour le prochain octet
                clr COUNT_BIT        ;/
                cmp.b LNG_MESSAGE,R6 ;Dernier octet du message ?
                jeq INIT_MESSAGE
                bis #WAIT_START,R13  ;Prochaine phase : attente bit Start de la
*                                     prochaine trame
                jmp NEW_IND_MESS
INIT_MESSAGE    mov #0,R6            ;Dernier octet reçu -> attente d'un nouveau
*                                     message
                bis #WAIT_MARK,R13   ;Prochaine phase : attente signal MARK
                bis #NEW_MESSAGE,R13 ;Pour rafraîchir l'affichage
NEW_IND_MESS    mov R6,IND_MESSAGE   ;Sauvegarde du pointeur
NEW_SAMPLE_F    bic #1,&CCTL2        ;Raz indicateur périodes 1200 Hz
exit_D_A        reti

*************
* Vecteurs  *
*************

                org 0ffe0h    ;                                Priorité
                dw  START     ;                                   0 (la plus faible)
                dw  START     ;                                   1
                dw  START     ;I/O Port P1                        2
                dw  START     ;I/O Port P2                        3
                dw  START     ;                                   4
                dw  START     ;                                   5
                dw  START     ;                                   6
                dw  START     ;                                   7
                dw  START     ;Timer A3, CCIFG1, CCIFG2, TAIFG    8
                dw  TIM_A_Int ;Timer A3, CCIFG0                   9
                dw  START     ;Watchdog timer                     10
                dw  START     ;Comparateur A                      11
                dw  START     ;                                   12
                dw  START     ;                                   13
                dw  START     ;NMI                                14
                dw  START     ;POR, reset externe, watchdog       15 (la plus élévée)

                end
```