TD IGI 741

Exercice 1 : E/S via un UART

Le registre de commande de l'UART est composé des bits suivants :

7	6	5	4	3	2	1	0

- bit 7 = 1 : Armer IT Réception
- bit 6 = 0 : activer RTS
- bit 5 = 1 : Armer IT Emission
- bit 4 = 0/1 : caractère 7/8 bits
- bit 3 = 0/1 : 2 STOP / 1 STOP
- bit 2 = 0/1 : PAIR/IMPAIR
- bit 1 et 0 = 00 : Horloge : 1
- = 01 : Horloge : 16
- = 10 : Horloge : 64
- = 11 : RESET

Le registre d'état de l'UART est composé des bits suivants :

IRQ	ERREURS : P. C. T.	CTS	DCD	E.Vide	R.Plein

- bit 7 = 1 : IT émise
- bit 6 = 1 : erreur de parité
- bit 5 = 1 : erreur de cadence (Overrun)
- bit 4 = 1 : erreur de Trame (Frame)
- bit 3 = 1 : copie de la borne CTS
- bit 2 = 1 : copie de la borne DCD
- bit 1 = 1 : Reg. Emission vide
- bit 0 = 1 : Reg. Réception plein

Le circuit UART est représenté en informatique par la variable globale ***uart*** de type enregistrement

cntrl de type octet non signé
status de type octet non signé
data_in de type octet non signé
data_out de type octet non signé
fin enregistrement

1. Fonctions d'entrée par interrogation

a) Ecrire en langage C une fonction **bloquante** (boucle d'attente) baptisée **lire_b()** qui retourne la valeur du registre d'entrée.
b) Ecrire en langage C une fonction **non bloquante** baptisée **lire_nb(*result*)** qui copie dans *result* (*paramètre de sortie*) une nouvelle donnée si elle est disponible et retourne 0 (-1 dans l'autre cas).

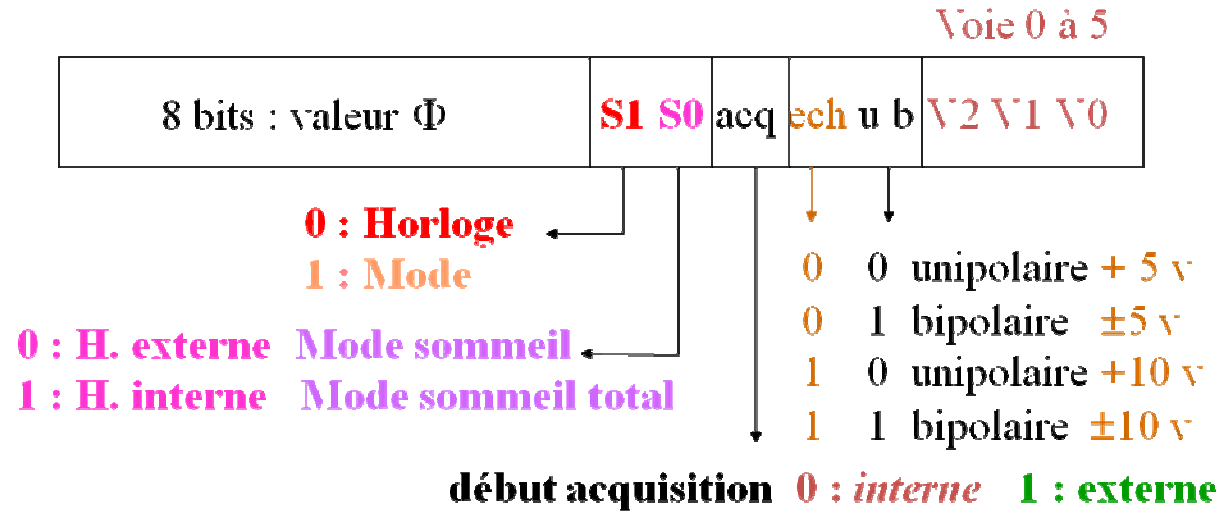
2. Fonctions de sortie par interrogation
a) Ecrire en langage C une procédure **bloquante** (boucle d'attente) baptisée **ecrire_b(*valeur*)** qui écrit la valeur donnée en paramètre dans le registre de sortie, dès que possible .
b) Ecrire en langage C une fonction **non bloquante** baptisée **ecrire_nb(*valeur*)** qui écrit la valeur donnée en paramètre dans le reg. de sortie si c'est possible et retourne 0 (sinon -1).

3. Procédure de service d'IT de l'UART. On supposera que le périphérique désarme les ITs en rentrant dans la routine mais ne les réarme automatiquement en sortant de la routine.
a) On veut échanger des données de type char *car_in* et *car_out*. Comment déclarer ces variables ?
b) Ecrire en langage C une procédure de service d'IT baptisée **isr_out()** qui émet le caractère *car_out* via l'UART.
c) Ecrire en langage C une procédure de service d'IT baptisée **isr_in()** qui récupère le caractère reçu par l'UART et le range à *car_in.*
d) Soit *mesure[128]* de type tableau d'entier court et *nb_mes* de type entier court déclarés en variables globales.
Ecrire en langage C la procédure de service d'IT **isr_input()** qui récupère le caractère reçu par l'UART et le range à sa place dans le tableau *mesure[].*

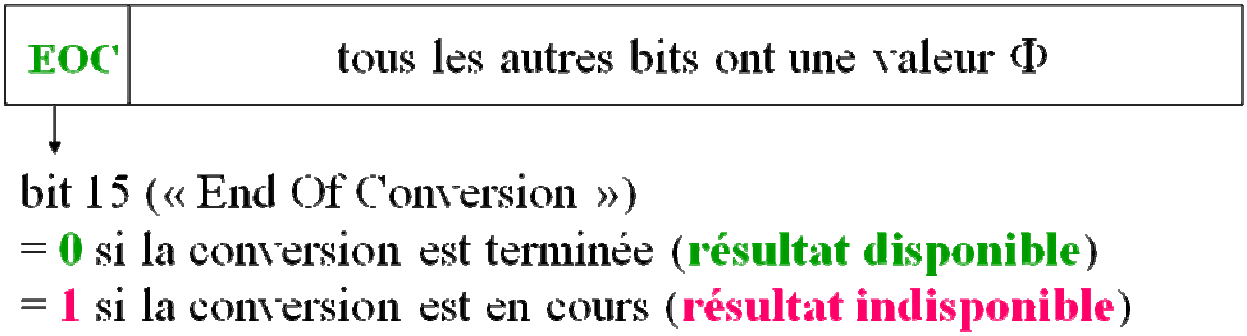
main: *…*
nb_mes ← *0*
armer les IT de réception
Tantque *nb_mes* < 128 Faire
// instructions d'une tache (interruptible) à faire
FinTantque
désarmer les IT de réception
…

Exercice 2 : CA/N par scrutation

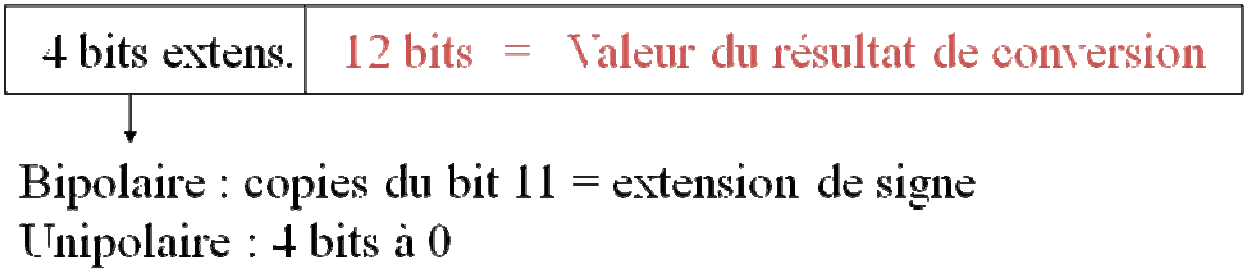
Soit le registre de commande d'un CA/N de 12 bits



Soit le registre d'état du CA/N :



Et finalement le registre de données :



Du point de vue informatique, le CA/N est une variable globale :

can de type enregistrement
champ *control* de type entier
champ *status* de type entier
champ *data* de type entier signé.
fin enregistrement

On utilise le CAN en mode : acquisition interne, unipolaire avec V_{ref} = 5v. Donnez au mieux la valeur du registre *can.control*.

a) Ecrire la fonction bloquante **convertir** (*voie*) qui retourne le nombre entier N_S résultat de la conversion de la tension V_E appliquée sur la voie dont le numéro est passé en paramètre.
b) Modifier la fonction pour qu'elle retourne la valeur de la tension d'entrée exprimée en millivolts.

Exercice 3 : Programmation TIMER

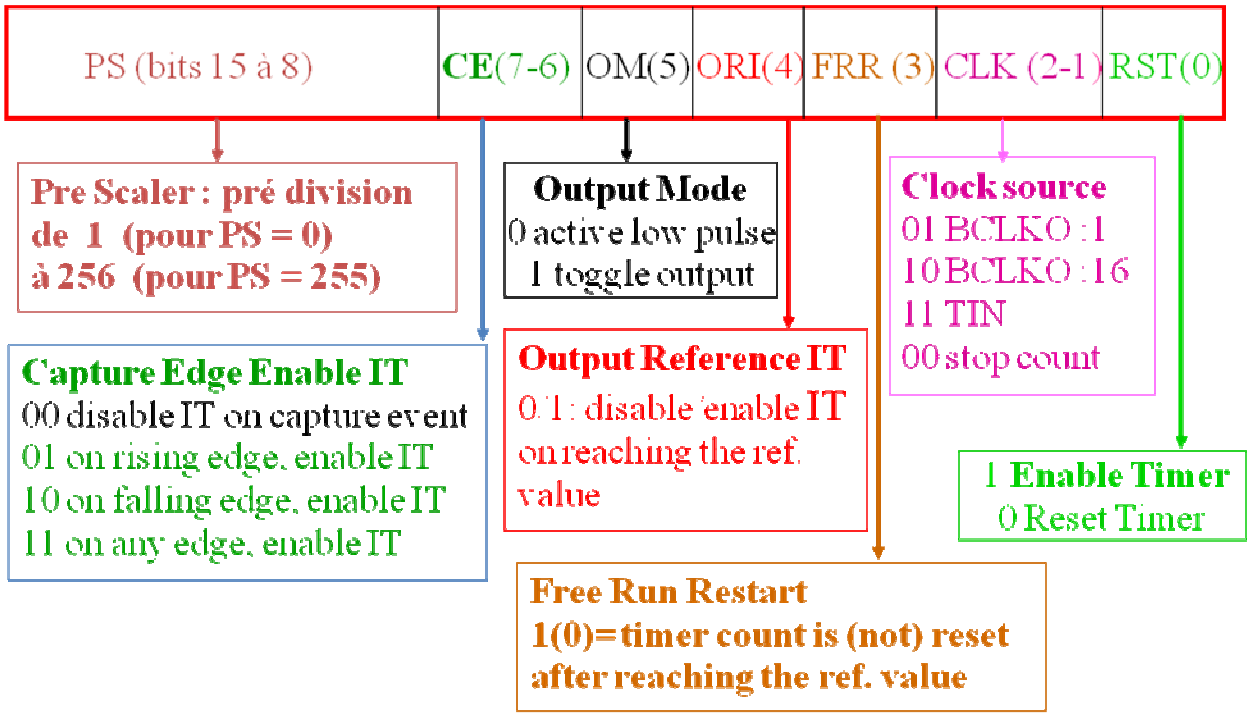
Le timer étudié fonctionne avec l'horloge interne de fréquence **BCLKO = 22,5 MHz**. Il comporte 5 registres :

- TMR : choix de mode
- TCN : compteur
- TRR : référence
- TER : état
- TCR : capture

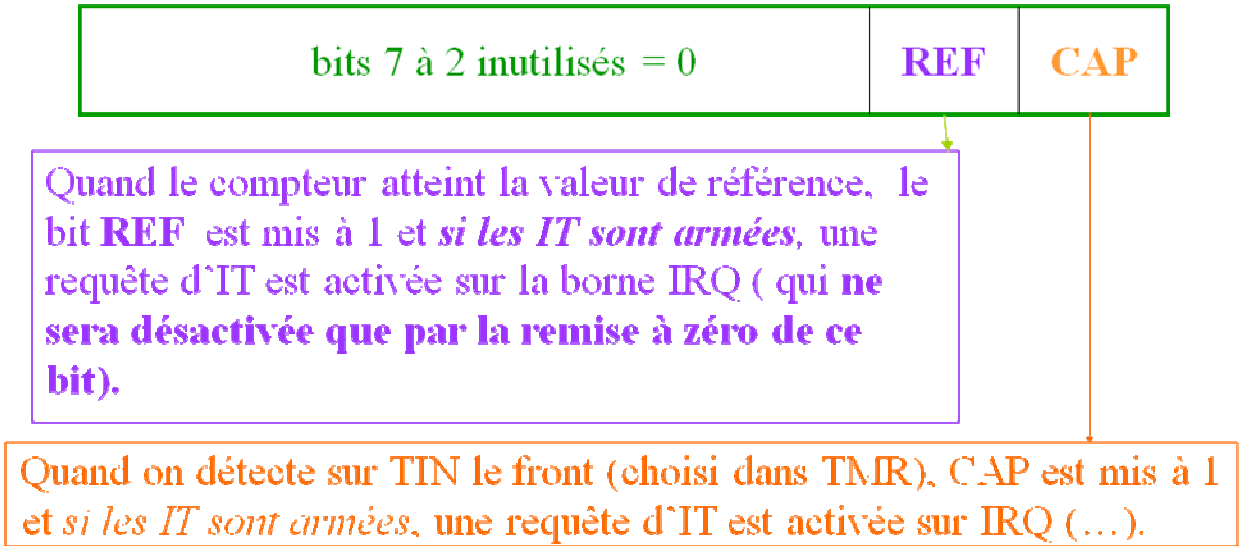
Lorsque la fonction est **démarrée**, **TCN est mis à jour** (+1) à la **cadence d'une horloge pré-divisée** par la valeur mise dans **TMR**. Quand le compteur atteint la valeur de référence de **TRR**,

- le bit **REF de TER** est mis à 1;
- une IT est émise *si les IT sont armées dans* **TMR ;**
- **TCN** est remis à 0 *si on est en mode périodique dans* **TMR**;

Le registre TMR est composé des bits suivants :



Le registre TER est composé des bits suivants :



ATTENTION : pour la rémise à zéro du bit REF/CAP, il faut écrire un 1 dans le bit 1 (REF)/0 (CAP) de TER.

Le registre TRR est composé des bits suivants :

Valeur de réf. (16 bits) qui est comparée à celle du compteur

Le registre TCN est composé des bits suivants :

Valeur courante (16 bits) mise à jour par BCLKO ou TIN

1. Programmer le timer pour qu'il envoie une IT périodique **toutes les secondes**. On utilisera une variable *timer0* de type enregistrement avec 5 champs *tmr, trr, tcn, tcr et ter.* On mettra ces lignes de code dans une fonction ***init_timer()***.

2. Ecrire la procédure de service ***isr_clk()*** qui met à jour une variable globale *temps* de type enregistrement avec 3 champs *heu*, *min*, *sec* de type octet.

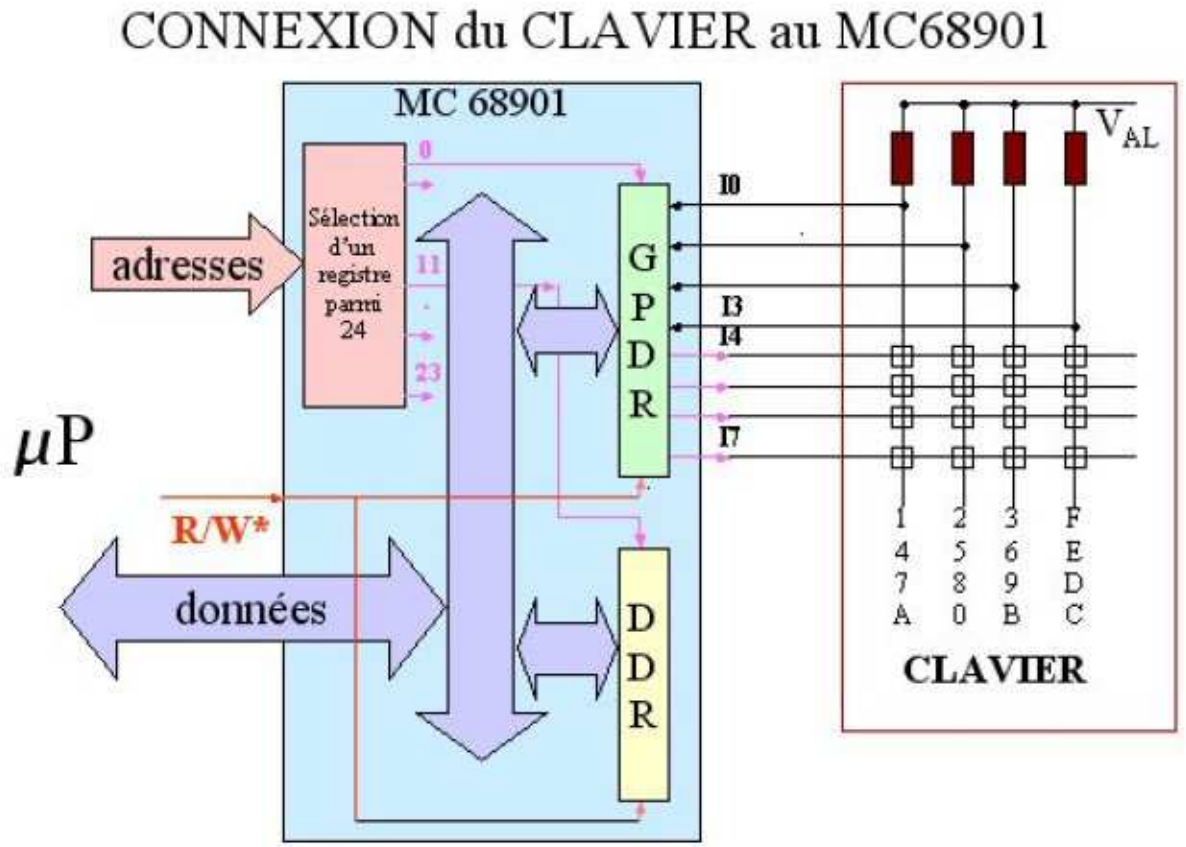
3. Le progr. *main,* interrompu par le Timer0, gère une application qui utilise une procédure *horloge()* pour afficher l'instant d'arrivée d'un événement particulier. Montrer par un exemple que le temps affiché peut être erroné. Proposer une solution au problème mis en évidence.

Exercice 3 : Programmation Clavier

Le clavier que l'on va utiliser comprend 16 touches hexadécimales (10 chiffres et 6 lettres). Ce clavier fonctionne sur le principe d'une matrice de contacts de 4 lignes et 4 colonnes. Lors de l'appui d'une touche, un contact est établi entre une ligne et une colonne.

Le principe de lecture d'un tel clavier consiste à identifier quelle ligne et quelle colonne sont en contact. Pour cela, on configure les colonnes en entrée à l'état haut et les lignes en sortie à l'état bas. Donc si un contact est établi, la colonne correspondante passera à 0. Pour trouver la ligne correspondante, on teste les lignes une à une en mettant la ligne sous test à 1 et les autres à 0. Si c'est la bonne ligne, alors la valeur de la colonne (déjà détectée) changera.

Pour interfacer un tel dispositif, il suffit de disposer de 8 lignes d'entrée/sortie tout ou rien, 4 d'entre elles seront positionnées en sortie, les 4 autres seront positionnées en entrée. Un seul port parallèle 8 bits suffira donc à contrôler le clavier.



La variable enregistrement associée au port parallèle est une variable globale nommée **cf_parallele**. Ainsi pour adresser le registre **gpdr**, on écrira **cf_parallele.gpdr** et pour

adresser le registre **ddr**, il faudra écrire **cf_parallele.ddr**. Le registre **ddr** permet de configurer les bits de gpdr en entrée (0) et en sortie (1).

1. Donner le programme de la fonction void init_kb_port() permettant de configurer le port parallèle.

2. Donner le programme de la fonction non bloquante int kb_hit() permettant de tester si on a appuyé sur une touche. Elle renvoie 1 si une touche a été appuyée.

3. Donner l'algorithme de la fonction char kb_getc() permettant de renvoyer le caractère appuyée sur le clavier.

- détecter la colonne correspondante

- détecter la ligne correspondante

4. Proposer un programme principal permettant de tester le fonctionnement du clavier.