GESTION DE LCD SUR 4 BITS

Nous avons vu le mois dernier comment connecter et gérer un module afficheur LCD sur un bus 8 bits. Ce montage est adéquat sur des systèmes disposant de mémoire externe et d'un décodage d'adresses sur lequel on peut prélever une ligne pour commander la validation du LCD. Cette disposition montre ses limitations lorsque l'on désire déporter l'afficheur à une distance importante (plus de quelques centimètres). En effet, le bus de données du microprocesseur est relié directement à l'afficheur, aussi, lorsque l'on augmente la distance entre l'afficheur et le contrôleur, les capacités parasites augmentent, et, très rapidement la charge sur le bus devient trop importante pour que le microcontrôleur puisse continuer à fonctionner.

On peut alors utiliser des buffers de bus mais on court le danger d'augmenter considérablement le rayonnement généré par le système, ce qui, en ces temps de normalisation sur les perturbations dues aux rayonnements électromagnétiques, n'est pas recommandé.
Pour diminuer le rayonnement associé à la liaison microcontroleur-afficheur, on gérera donc celui-ci par l'intermédiaire d'un port d'entrées/sorties. Les signaux évoluant moins rapidement, le rayonnement sera plus faible, et, par ailleurs, on pourra déporter l'afficheur LCD à une distance beaucoup plus importante.
Les afficheurs LCD peuvent fonctionner selon deux modes : le mode huit bits et le mode quatre bits. Le mode huit bits est le mode que nous avons

Instructions	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Description	Durée
Clear display	0	0	0	0	0	0	0	0	0	1	Efface l'afficheur et curseur en position «home»	.5ms
Return home	0	0	0	0	0	0	0	0	1	X	Curseur «home», si l'affichage était déplacé, le remet dans l'état initial, DD Ram inchangée	5ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Définit le sens de déplacement et si l'affichage se déplace ou non	120µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Valide ou invalide l'afficheur (D), le curseur (C), et le clignotement tement du curseur (B)	120µs
Cursor and display shift	0	0	0	0	0	1	S/C	R/L	X	X	Déplace le curseur et l'affichage sans changer la DD RAM	120µs
Function set	0	0	0	0	1	DL	N	F	X	X	Définit la taille de l'interface, le nombre de ligne, la taille des fontes	120µs
Set CG RAM address	0	0	0	1	A5	A4	A3	A2	A1	A0	Valide l'adresse de la CG RAM. Les données de la CG RAM sont envoyées aprés cette commande.	120µs
Set DD RAM address	0	0	1	A6	A5	A4	A3	A2	A1	A0	Valide l'adresse de la DD RAM. Les données de la DD RAM sont envoyées après cette commande	120µs
Read busy flag & address	0	1	BF	A6	A5	A4	A3	A2	A1	A0	Lit le flag busy et l'adresse courante (de la DD RAM ou CG RAM)	1µs
Write data to CG or DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Ecrit des données dans la DD RAM ou la CG RAM	120µs
Read data	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Lit les données de la DD RAM ou de la CG RAM	120µs

Tableau 1

I/D = 1 : Incrément
S = 1 : L'afficheur accompagne le curseur dans le déplacement
S/C = 1 : Afficheur se déplace, S/C = 0 : le curseur se déplace
R/L = 1 : Déplacement vers la droite, R/L = 0 : vers la gauche
DL = 1 : Interface 8 bits, DL = 0 : interface 4 bits
N = 1 : 2 (ou 4) lignes, N = 0 : 1 ligne
F = 1 : caractères 5 X 10, F = 0 : 5 X 7
BF = 1 : Occupé, BF = 0 : accepte une instruction
DD RAM : mémoire d'affichage, CG RAM : mémoire générateur de caractères

utilisé le mois dernier. Comme nous choisissons maintenant de connecter l'afficheur à un port d'entrées/sorties (P1 par exemple), il est souhaitable de connecter l'afficheur de façon à minimiser le nombre de liaisons. Nous allons donc connecter l'afficheur en mode quatre bits. Dans ce mode, seuls les quatre bits de poids fort de l'afficheur sont utilisés pour transmettre les données et pour les lire. Les lignes E, RW et RS sont connectées comme dans le mode huit bits, ce qui porte à sept le nombre de liaisons (hors alimentation) entre l'afficheur LCD et le microcontrôleur. La sélection du mode de fonctionnement de l'afficheur se fait à l'initialisation. Pendant celle-ci, on force l'afficheur dans le mode huit bits, puis quand on est sûr que celui-ci est valide, le mode quatre bits est validé. Comme on ne sait pas au début de l'initialisation si l'afficheur est positionné en huit ou quatre bits, il est nécessaire d'envoyer la commande de passage en mode huit bits plusieurs fois de façon à ce que celle-ci soit comprise, que le mode de départ soit quatre ou huit bits. Une fois le mode initialisé, les données sont écrites ou lues en envoyant séquentiellement les quatre bits de poids fort suivi des quatre bits de poids faible. Une impulsion positive doit être envoyée sur la ligne E pour valider chaque demi-octet (appelés nibbles). La ligne RW indique si l'on fait une écriture ou une lecture, la ligne RS indique si on envoie une commande ou un caractère. Le **tableau 1** résume les opérations que l'on peut effectuer avec les afficheurs LCD ainsi que la durée maximum de ces opérations. On notera cependant le point suivant :
les modules afficheurs LCD utilisent pour leur gestion interne un microcontrôleur qui est initialisé lors de la mise sous tension. Lorsque l'on travaille dans des milieux parasités ou sur des systèmes alimentés par batterie, il peut arriver que le contrôleur du LCD soit ré-initialisé alors que le microcontrôleur qui contient l'application ne l'est pas (ou inversement). Si on travaille en mode quatre bits, il peut donc y avoir une désynchronisation de l'afficheur et du microcontrôleur. Ce que le microcontrôleur transmet comme étant les quatre bits de poids fort est compris par le LCD comme étant les quatre bits de poids faible. En général le système se bloquera assez rapidement, lorsque le signal busy ne sera plus reconnu. Il est donc souhaitable de gérer un timer pendant la scrutation du signal busy pour ré-initialiser l'afficheur si celui-ci ne répond plus. On peut aussi dans certains cas ré-initialiser systématiquement le LCD périodiquement.
Vous trouverez dans le **listing 1** toutes les routines de base permettant de gérer l'afficheur LCD en mode quatre bits par l'intermédiaire d'un port, ici le port P1. Toutes ces routines sont compatibles avec celles du mois précédent, et donc les routines de haut niveau qui utilisent celles-ci fonctionnent indifféremment avec les unes ou les autres.

J.L VERN

```
PORT_LCD    EQU     P1
E           BIT     PORT_LCD.4
RW          BIT     PORT_LCD.5
RS          BIT     PORT_LCD.6
BITRS       EQU                 40H             ; position du bit RS precedant

; ROUTINES DE BAS NIVEAU DE GESTION DE LCD ALPHANUMERIQUE EN MODE 4 BITS
; Ces routines sont compatibles avec les routines devellopées pour etre
; utilisées avec le lcd directement sur le bus.
; Le lcd est utilise en mode 4 bits. Trois lignes de contrôles sont utilisées
; Toute les lignes (données et controles) aboutissent au meme port (PORT_LCD)
; Le huitieme bit du port est configuré en entrée

; Brochage du LCD (attention, peut varier d'un lcd à l'autre)
;LCD -> GND +5V Vo  RS  R/W  E   D0  D1  D2  D3  D4   D5   D6   D7
;8051->         P1.6 P1.5 P1.4 — — — — P1.0 P1.1 P1.2 P1.3
;

; Lit l'adresse courante (position du curseur)
lcd_read_cur:
    ACALL   chk_busy        ; attend la libération du lcd
; lit les deux nibbles (demi-octet) dans R7 et A.
; Si RS = 0, les bits de poids faibles sont l'adresse curseur et le bit 7 busy
; Si RS = 1, l'octet lu est le caractère courant (à la position du curseur).
read_lcd_byte:
    ORL     P1,#0FH                         ; port en lecture (a 1)
    SETB    RW                              ; lcd en lecture
    SETB    E
    MOV     A,PORT_LCD                      ; prend la donnée LCD
    CLR     E                               ; pulse sur E pour lire le MSB
    ANL     A,#0FH
    SWAP    A                               ; dans le MSB
    MOV     R7,A                            ; sauve A dans R7
    SETB    E
    MOV     A,PORT_LCD                      ; lit le LSB
    CLR     E                               ; pulse sur E pour lire le LSB
    ANL     A,#0FH                          ; efface le MSB inutile
    ORL     A,R7                            ; ajoute le MSB sauvegardé
    MOV     R7,A                            ; et sort avec l'octet dans A et R7
    RET

; Attend la libération du lcd
chk_busy:
            MOV     A,R7
            PUSH    ACC                     ; sauve R7 (mieux que PUSH AR7)
            CLR     RS                      ; lcd contrôle
chk_0:      ACALL   read_lcd_byte
            JB      ACC.7,chk_0             ; bit busy a 1, attend
    POP     ACC                             ; restaure l'ancien R7
    XCH     A,R7                            ; à sa place
    RET

; Lit le caractère à l'adresse courante (position du curseur)
lcd_read:
            ACALL   chk_busy                ; attend la libération du lcd
            SETB    RS                      ; lit un caractère et non l'adresse
            AJMP    read_lcd_byte           ; et relit la valeur du curseur

; Affiche le caractère R7 à l'adresse courante. L'adresse courante est
; incrementée
lcd_outch:
            ACALL   chk_busy                ; attend la libération du lcd
            MOV     A,R7                    ; restaure le caractère à envoyer
            SWAP    A                       ; prend le msb
            ACALL   lcd_o0                  ; envoie le MSB
            MOV     A,R7                    ; prend le LSB
lcd_o0:     ANL     A,#0FH                  ; met a zero E, RS, RW
            ORL     A,#BITRS                ; (lcd en ecriture de data)
            AJMP    lcd_c1                  ; valide le lsb

; Envoie sur l'afficheur le caractère de controle R7
lcd_ctrl:
            ACALL   chk_busy                ; attend la liberation du lcd
            MOV     A,R7                    ; restaure le caractère à envoyer
            SWAP    A                       ; prend le msb
            ACALL   lcd_c0                  ; envoie le MSB
            MOV     A,R7                    ; prend le lsb
lcd_c0:     ANL     A,#0FH                  ; met à zéro E, RS, RW
lcd_c1:     SETB    ACC.7                   ; bit 7 du port en entrée
            MOV     PORT_LCD,A              ; et valide le lsb
            SETB    E                       ; sur le lcd
            CLR     E
    RET

; Initialise l'afficheur en deux lignes (ou quatre lignes) 4 bits
; Pendant la phase d'initialisation, l'afficheur est dans un état instable
; on ne peut pas tester la ligne busy. Une temporisation est donc faite
; entre deux envois de demi caractères
; La premiere ligne (0,1,0,0,1) n'est la que pour faire un effacement propre
; en cas de réinitialisation, que l'on redemarre en huit bits ou en quatre
; bits (même en cas de décalage des deux nibble du au port en tri-state)
LCD_TAB_INIT:
            DB      0,1,0,0,1               ; uniquement pour effacer au début
            DB      3,3,3                   ; force le lcd en 8 bits
            DB      2                       ; 20H passe de 8 bits en 4 bits
; à partir d'ici, on pourrait utiliser busy (lcd_ctrl)
            DB      2,8,0,0CH               ; 28H, 0CH : 4bits, 2lignes, disp on
            DB      0,6,0,1                 ; 06H, 01H : curseur incré, clrscr
LCD_TAB_END:
LCD_TAB EQU   LCD_TAB_END-LCD_TAB_INIT

temp:       MOV     R7,#0                   ; lance une tempo
tempo:      MOV     ACC,#60
tempo1:     DJNZ    R7,tempo1
            DJNZ    ACC,tempo1
            RET

lcd_init:
            MOV     R0,#LCD_TAB             ; nbr d'octets dans LCD_TAB_INIT
lcdini:     CLR     C
            MOV     A,#LCD_TAB
            SUBB    A,R0                    ; de 0 a LCD_TAB-1
            MOV     DPTR,#LCD_TAB_INIT      ; octets d'initialisation
            MOVC    A,@A+DPTR               ; prend l'octet suivant de LCD_TAB_INIT
            ACALL   lcd_c1                  ; et l'envoie sur le lcd
            ACALL   temp
            DJNZ    R0,lcdini               ; si reste des octets continue
            RET

; ROUTINES DE HAUT NIVEAU

; envoie un octet en hexa sur l'afficheur
lcdhex:     MOV     A,R7
            PUSH    ACC
            SWAP    A
            ACALL   lcdnib                  ; MSB
            POP     ACC                     ; restaure le LSB
; converti et envoie le nibble
lcdnib:     ANL     A,#0FH
            ADD     A,#90H
            DA      A                       ; mystères du décimal adjust ...
            ADDC    A,#40H
    DA      A
            MOV     R7,A                    ; dans R7 un caractère 0..9 ou A..F
            AJMP    lcd_outch

END
```

Listing 1