Carte Clavier
Afficheur graphique
Horloge temps réels

EID005

Auteur : KOMA N'Gally
Professeur BTS IRIS
IFA Delorozoy
CCI Versailles

didalab
ENSEIGNEMENT SUPERIEUR

DIDALAB
5 Rue du Groupe Manoukian
78990 Elancourt
Tel: 01.30.66.08.88 / Fax: 01.30.66.72.20
ge@didalab.fr

SOMMAIRE

EID005_TP 1 ECRITURE D'UNE CHAINE DE CARACTERES EN MODE TEXTE SUR L'AFFICHEUR 5
1.1 Enoncé du sujet 5
1.2 Eléments de solution 6
1.2.1 Description sommaire de l'afficheur 6
1.2.2 Programme en C 15
1.2.3 Programme en Assembleur 17
EID005_TP 2 ECRITURE D'UNE CHAINE DE CARACTERES EN MODE TEXTE SUR L'AFFICHEUR 22
2.1 Enoncé du sujet 22
2.2 Eléments de solution 23
2.2.1 Description sommaire de l'afficheur 23
2.2.2 Programme en C 33
2.2.3 Programme en Assembleur 34
EID005_TP 3 LECTURE D'UN CLAVIER MATRICIEL EN MODE POLLING 39
3.1 Enoncé du sujet 39
3.2 Eléments de solution 40
3.2.1 Description sommaire du clavier 40
3.2.2 Programme en C 44
3.2.3 Programme en Assembleur 46
EID005_TP 4 DETECTION D'ACTIVATION D'UNE TOUCHE ET LECTURE EN MODE POLLING 53
4.1 Enoncé du sujet 53
4.2 Eléments de solution 54
4.2.1 Description sommaire du clavier 54
4.2.2 Programme en C 58
4.2.3 Programme en Assembleur 60
EID005_TP 5 DESSIN DE LIGNES, CERCLES ET COURBES SUR LE LCD 69
5.1 Enoncé du sujet 69
5.2 Eléments de solution 70
5.2.1 Description de l'afficheur en mode graphique 70
5.2.2 Programme principal 72
5.2.3 Organigramme du tracé d' une sinusoïde sur un oscilloscope 73
5.2.4 Programme en " C " : 75
EID005_TP 6 DESSIN D'UNE HORLOGE SUR L'ECRAN GRAPHIQUE 78
6.1 Eléments de solution 79
6.1.1 Définition géométrique de l'horloge 79
6.1.2 Programme principal 82
6.1.3 Organigramme 82
6.1.4 Programme en C 86

EID005_TP 1 ECRITURE D'UNE CHAINE DE CARACTERES EN MODE TEXTE SUR L'AFFICHEUR

1.1 Enoncé du sujet

Objectifs :	Etre capable d'utiliser les utilitaires rangés en bibliothèque, permettant la gestion de l'afficheur LCD graphique 128*64 pixels.
Cahier des charges :	**Sujet** Ecrire un programme en assembleur et en C qui réalisera l'affichage d'une chaîne de caractères. La longueur maximale de la chaîne est de 128 caractères. On donnera les coordonnées du 1er caractère.

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,
Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID210000
Carte Clavier Afficheur Horloge Temps réel : EID005000
Câble de liaison USB, ou à défaut câble RS232, Réf : EGD000003
Alimentation AC/AC 8V, 1 A Réf : EGD000001,

Documentations nécessaires :

Document : DMS Carte Clavier Afficheur Horloge Temps réel : EID00500
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Durée : 1 séance de 3 heures

1.2 Eléments de solution

1.2.1Description sommaire de l'afficheur

1.2.1.1 Représentation de l'afficheur LCD 128x64

Attention :

Pour des raisons de conformité avec la documentation du constructeur, les variables x et y représentent respectivement l'ordonnée (verticale) et l'abscisse (horizontale). Le point **x = 0, y = 0** est en haut à gauche et le point **x = 63, y = 127** en bas à droite de l'écran du LCD.

Le contrôleur T6963C dispose d'une mémoire de 8 ko.

Mode texte *fig.1*

Dans l'étude qui suit, la zone texte est placée de l'adresse 0000 à 007F de la mémoire écran (VRAM), soit 128 caractères.
Octet de poids faible : **00 toujours fixe**
Octet de poids fort : **00 à 7F** en hexadécimal.
Le quartet de poids fort de cet octet désigne le numéro de ligne x.
Le quartet de poids faible désigne le numéro de la colonne y.

Exemple

Le caractère numéro **59** est placé en **: x = 3, y = B** ; ce qui donne en hexadécimal les valeurs : **x = 3, y = B** d'où en mémoire, les octets suivants pour le paramètre TH (Text Home Adress :

TH Adress lower = 0x3B (59 en décimal)
TH Adress upper = 0x00

N MODE TEXTE

Y

Colonne → / Ligne ↓	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	**59**	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

X

fig.1

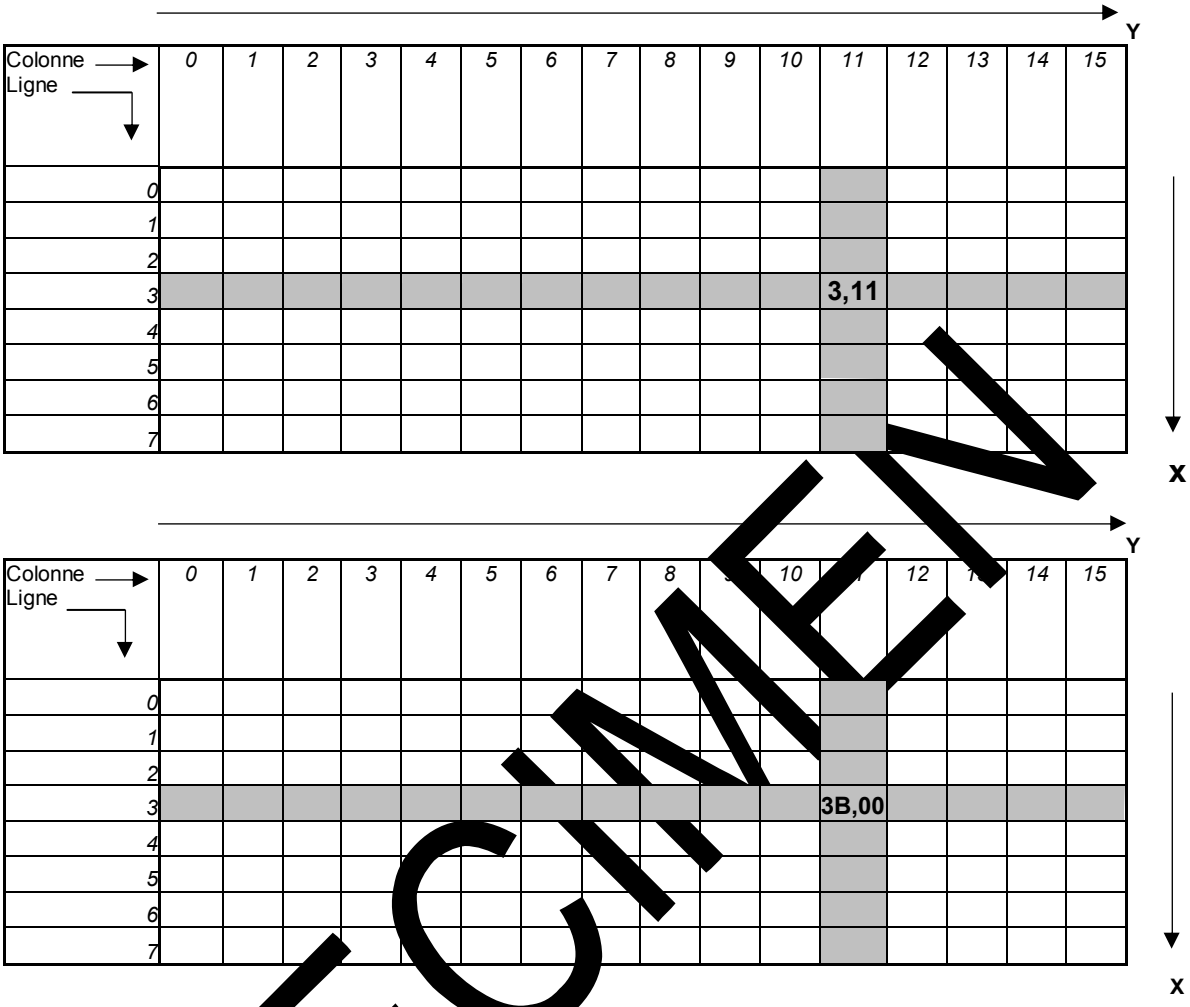


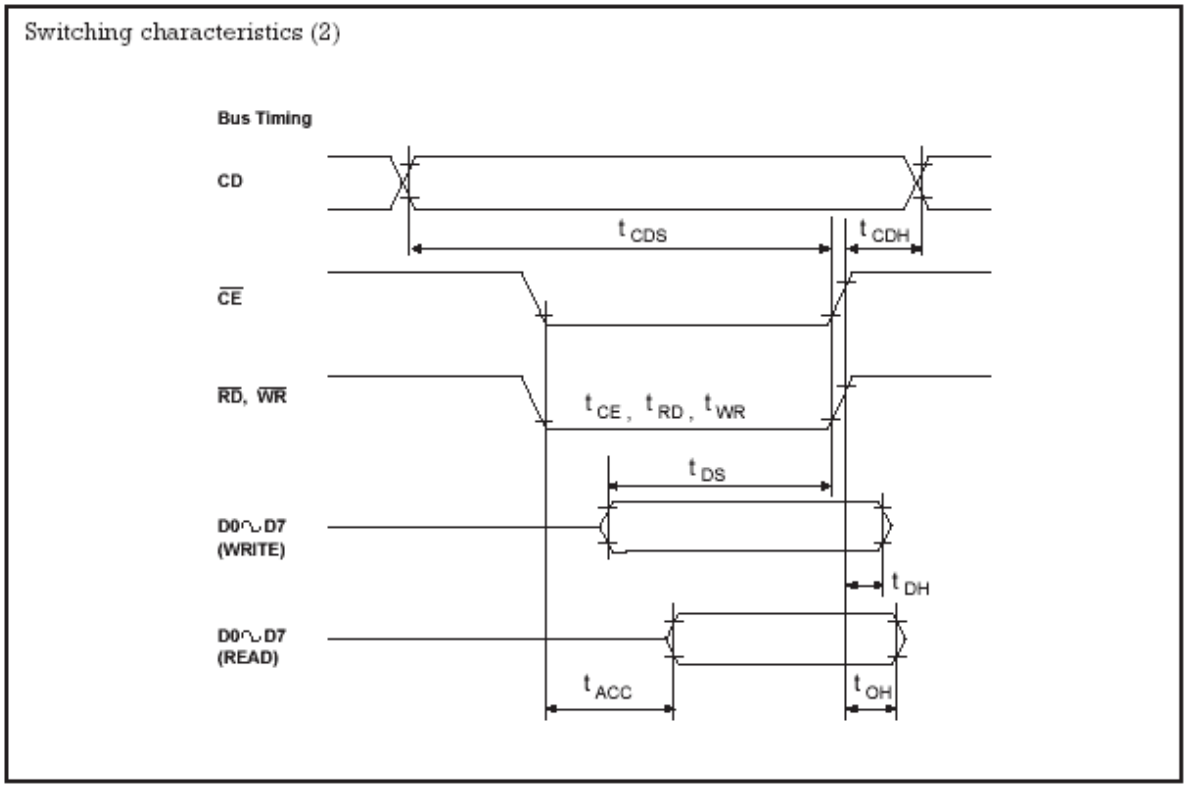
fig.1 suite

Générateur de caractères interne (CG ROM)

Le générateur de caractères interne utilise un code ASCII décalé de 0x20 ; exemple : la lettre A codée 0x41 en ASCII est représentée par la valeur 0x21 dans le T6963C (voir tableau ci-dessus).
Cela signifie que pour envoyer un caractère ASCII, il faut soustraire de son code, la valeur 0x20.
3

Chronogrammes Ecriture / Lecture de données ou commandes

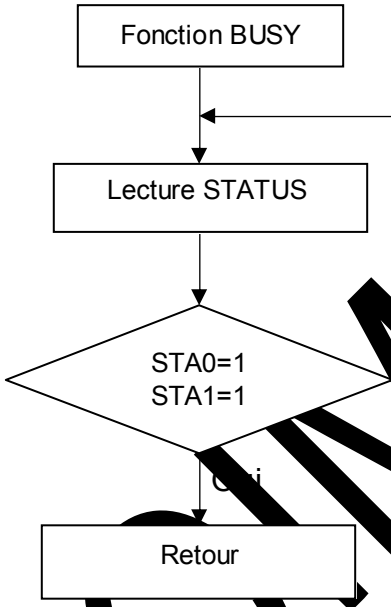
Ces chronogrammes doivent être générés pour chaque accès au LCD.
Ils sont réalisés et décrits en détail notamment dans les sous-programmes en Assembleur.



La génération de chacun de ces chronogrammes donnera lieu à un sous-programme correspondant.

1.2.1.2 Gestion de l'afficheur LCD

Avant chaque mouvement (écriture ou lecture) donnée (commande ou donnée) entre l'afficheur et le processeur de commande, il faut s'assurer que le LCD est prêt à exécuter l'opération.
D'où la nécessité de commencer par tester les bits d'état (ou de statut) de son registre d'état : STA0 et STA1 ; ce qui donne l'organigramme suivant :



1.2.1.3 Instructions de l'afficheur

Les instructions de l'afficheur sont données dans le tableau ci-dessous.

Les organigrammes ci-dessous définissent les modes d'écriture d'instruction de commande nécessitant 2, 1 ou 0 octets.

Les organigrammes suivants définissent en détail les modes d'écriture et de lecture des commandes et des données.

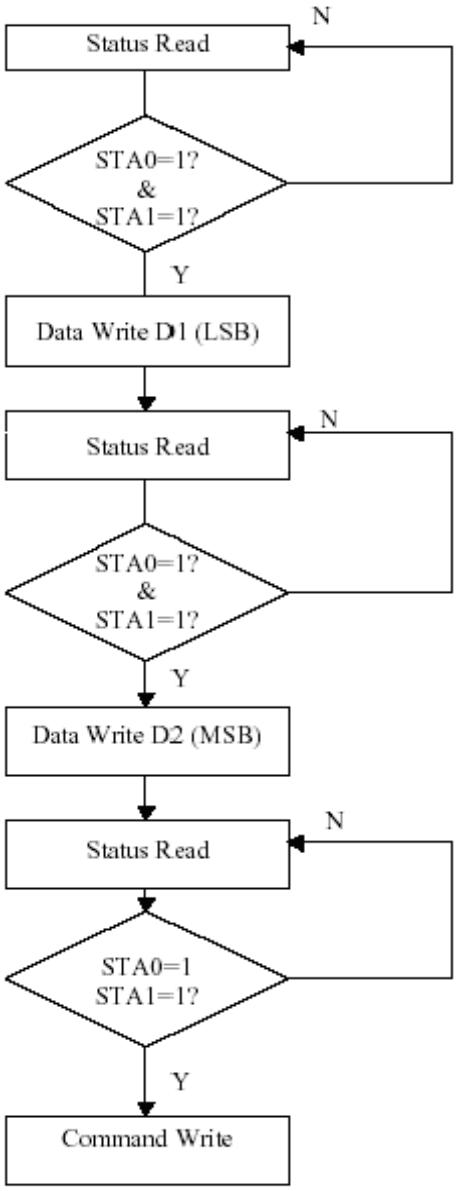
T6963C Instruction Set

Commands	D7	D6	D5	D4	D3	D2	D1	D0	Description	Execute Time
Pointer Set	0	0	1	0	0	N2	N1	N0		Status check
						0	0	1	Cursor Pointer Set	
						0	1	0	Offset Register Set	
						1	0	0	Address Pointer Set	
Control Word Set Commands	0	1	0	0	0	0	N1	N0		32 x 1/fosc
							0	0	Text Home Address Set	
							0	1	Text Area Set	
							1	0	Graphic Home Address Set	
							1	1	Graphic Area Set	
Mode Set	1	0	0	0	CG	N2	N1	N0		32 x 1/fosc
					0				CG ROM Mode	
					1				CG RAM Mode	
						0	0	0	"OR" Mode	
						0	0	1	"EXOR" Mode	
						0	1	1	"AND" Mode	
						1	0	0	Text only (attribute capability)	
Display Modes	1	0	0	1	N3	N2	N1	N0		32 x 1/fosc
					0				Graphics Off	
					1				Graphics On	
						0			Text Off	
						1			Text On	
							0		Cursor Off	
							1		Cursor On	
								0	Cursor blink Off	
								1	Cursor blink On	
Cursor Pattern Select	1	0	1	0	0	N2	N1	N0	N2–N0: No. of lines for cursor +1	32 x 1/fosc
						0	0	0	Bottom Line cursor	
						0	0	1	2 line cursor	
						|	|	|		
						1	1	1	8 line cursor (block cursor)	
Data Auto Read/Write	1	1	0	0	0	0	N1	N0		32 x 1/fosc
							0	0	Data Auto Write Set	
							0	1	Data Auto Read Set	
							1	0	Auto reset (Address pointer auto-incremented) for continuous rd/wr	
Data Read/Write	1	1	0	0	0	N2	N1	N0		
						0			Address Pointer up/down	
						1			Address Pointer unchanged	
							0		Address Pointer up	
							1		Address Pointer down	
								0	Data Write	
								1	Data Read	
Screen Peeking	1	1	1	0	0	0	0	0	Read Displayed Data	Status
Screen Copy (Note 3)	1	1	1	0	1	0	0	0	Copies 1 line of displayed data whose address is indicated by the Address Pointer to Graphic RAM area	Status check
Bit Set/Reset	1	1	1	1	N3	N2	N1	N0	N2–N0 indicates the bit in the pointed address	Status check
					0				Bit Reset	
					1				Bit Set	
						0	0	0	Bit 0 (LSB)	
						0	0	1	Bit 1	
						|	|	|		
						1	1	1	Bit 7 (MSB)	

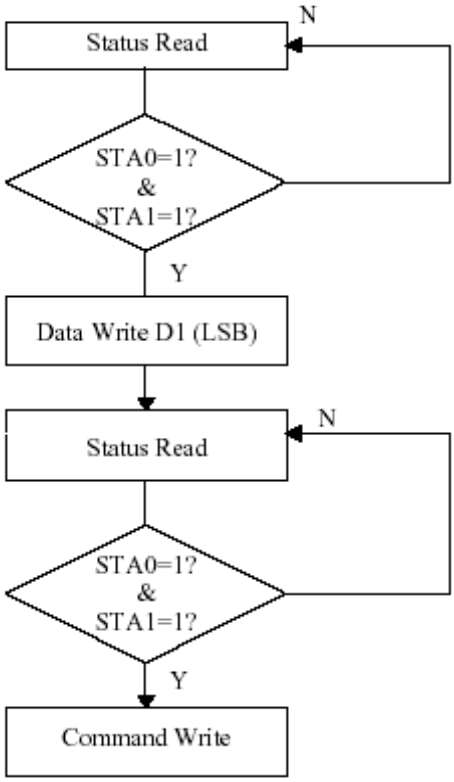
Note:

1. * = DONT CARE

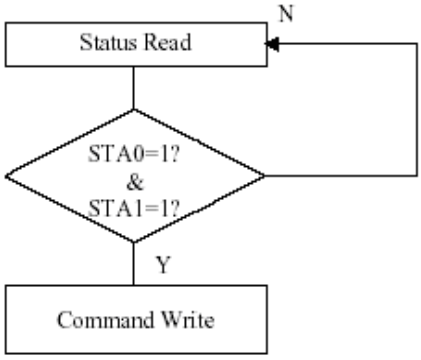
Commande nécessitant 2 octets



Commande nécessitant 1 octet

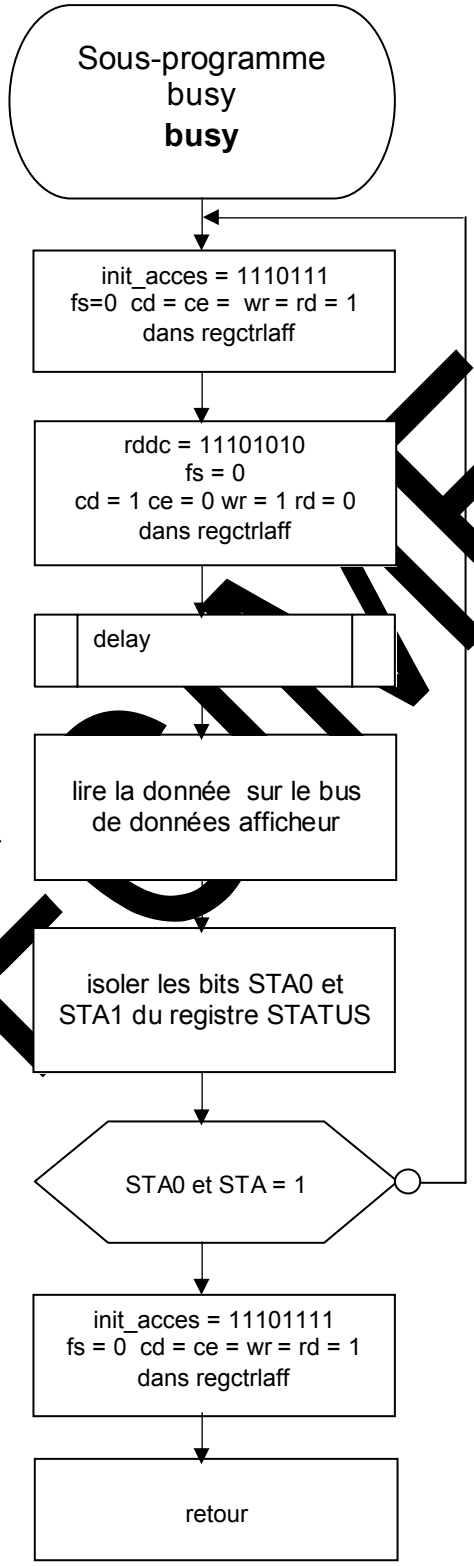


Commande nécessitant 0 octets

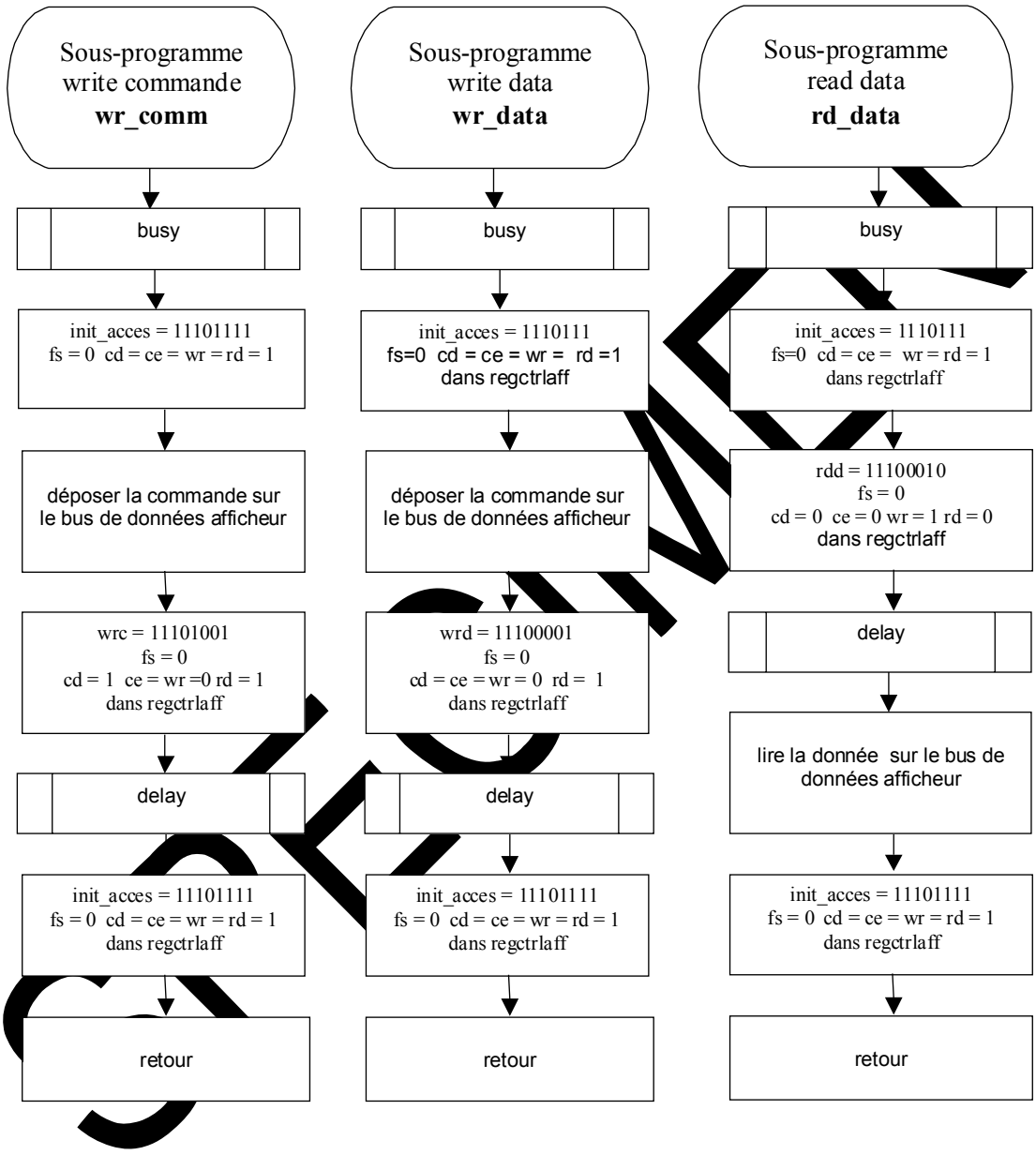


Organigrammes détaillés des sous-programmes principaux

Sous-programme busy



Sous-programmes écriture lecture



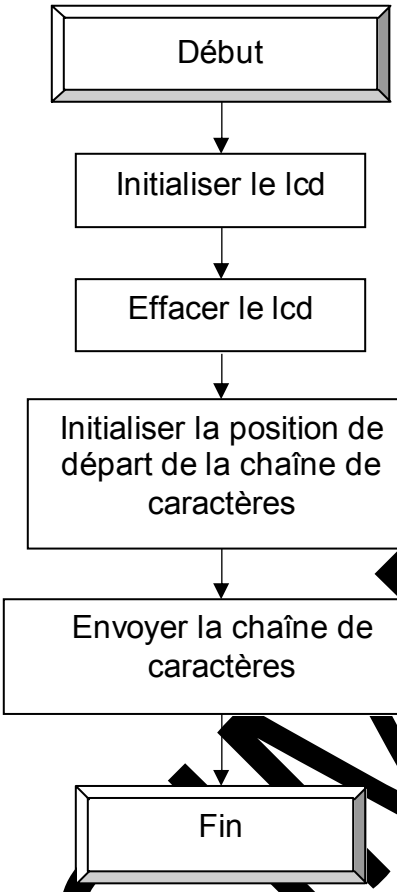
1.2.1.4 Organigramme principal

Vous disposez des utilitaires suivants avec leurs commentaires.

Utilitaires	Commentaires
void init_aff()	Initialisation des paramètres TH, TA, GH GA Mode …
void lcd_cls()	Effacement de l'écran
void lcd_write_command(unsigned char **commande**)	Ecriture d'une commande
void lcd_write_data(unsigned char **data**)	Ecriture d'une donnée
void lcd_gotoxy(unsigned char **px**, unsigned char **py**)	Définition de la position d'un caractère en pixel px et py sont les données lcd_x et lcd_y.
void lcd_out_str(char *texte)	Envoi d'une chaîne de caractères pontée par la variable * texte

Important :

Pour utiliser ses fonctions ainsi que le clavier afficheur avec le compilateur C/C++, il faut configurer le linker du logiciel eid210.
Pour cela : aller dans le menu configuration puis cliquer gnu C/C++.Ensuite aller dans l'onglet *linker*, cliquer sur *ajouter*, sélectionner le fichier « EID005_LCD.o » et finir en cliquant sur ouvrir.



1.2.2 Programme en C

```
/*********************************************************
*     TP SUR LA CARTE EID005                             *
*********************************************************
*     Ecriture d'un programme Assembleur                 *
*     et en C qui réalise l'affichage                    *
*     d'une chaîne de caractères.                        *
*     Longueur maximale de la chaîne : 128 caractères    *
*********************************************************
*     Nom du FICHIER : EID005_TP1.c                      *
*                                                        *
*********************************************************/

//    Inclusion des fichiers de définition

#include "eid005.h"
#include <stdio.h>
#include <string.h>
#include <math.h>
```

```
//============================
//    FONCTION PRINCIPALE
//============================

main()
{
init_aff();             //    Initialisation de l'afficheur
lcd_cls();              //    Effacement de l'afficheur
lcd_gotoxy(0x10,00);    //    Définition de la position de départ de la
                        //    Chaîne
//    Envoi de la chaîne de caractères

lcd_out_str ( "EID005000 :     CLAVIER,        AFFICHEUR,      HORLOGE
TEMPS   REEL");

}

// Les espaces dans le texte permettent d'éviter de couper un mot


//    Fin du programme
```

1.2.3 Programme en Assembleur

```
*******************************************************
*     TP SUR LA CARTE EID005                          *
*******************************************************
*     Ecrire un programme en Assembleur               *
*     et en C et qui réalise la lecture d'un touche   *
*     sur un clavier matricé 4 x 4, en mode pooling   *
*******************************************************
*     Nom du FICHIER : EID005_TP1.src                 *
*     **************                                  *
*******************************************************

**    La commande ou la donnée à écrire sont d'abord rangées dans D0
**    La donnée lue est rangée dans D0
**    La position x, y d'un caractère est placée respectivement dans D0, D1
**    L'adresse de début d'une chaîne de caractère est placée dans A0

*******************************************************
*     Bits du registre de contrôle de l'afficheur     *
*     ctrlaff :   b0=rd                               *
*                 b1=wr                               *
*                 b2=ce                               *
*                 b3=cd                               *
*                 b4=fs                               *
*******************************************************

* Inclusion du fichier définissant les différents labels
* de l'EID210

      include EID215.def

      section var

*     Définition des adresses physiques de l'afficheur et du clavier

eid005      equ   $B30000     * Adresse de base Carte clavier_afficheur_rtc
ctrlaff     equ   eid005+3    * Registre contrôle afficheur : bus de
                              * contrôle
dbaff       equ   eid005+4    * Bus de données afficheur
statut      equ   eid005+6    * Registre de statut de la carte
reg_ctrl    equ   eid005+7    * Registre de contrôle de la carte
reg_clavier equ   EID005+5

**    Table de définition des paramètres de l'afficheur

*************************************************************
*           $00,$04,$42       * Graphic Hom Adress et Command
*           $10,$00,$43       * Graphic Area et Command
*           $00,$00,$40       * Text Hom Adress et Command
*           $10,$00,$41       * Text Area et Command

ModSet      equ   $80         * OR  Graphique ou Texte
Pointeur    equ   $24         * Commande pointeur
DispMod     equ   $94         * Affichage Texte et ou Graphique
AutoInc     equ   $C0         * Auto incrémentation pixel ou caractère
```

```
init_acces  equ   $EF   * Pour accéder au bus de données du lcd avec fs=0
wrc         equ   $E9   * cd=1, ce=0, wr=0, rd=1, fs=0 1110 1001
                        * write commande
rdc         equ   $EA   * cd=1, ce=0, wr=1, rd=0, fs=0 1110 1010
                        * read commande (Status)
wrd         equ   $E1   * cd=0, ce=0, wr=0, rd=1, fs=0 1110 0001 write data
rdd         equ   $E2   * cd=0, ce=0, wr=1, rd=0, fs=0 1110 0010 read data

**    Une chaîne de caractère se termine par le caractère '$'

**    Attention les espaces dans le texte servent à afficher sans couper
**    les mots

Texte dc.b  'EID005 CLAVIER  AFFICHEUR       HORLOGE TEMPS   REL  $'

      section           code

*******************************
*     PROGRAMME PRINCIPALE    *
*******************************

      bsr         init_aff    * Initialisation afficheur
      bsr         lcd_cls     * Effacement écran

* Envoi Texte : x=2, y=1 : dans D0

      move.b      #$20,D0     * LSByte TH ligne colonne 1
      clr.b       D1          * MSByte TH
      bsr         lcd_gotoxy  * Position pour écrire caractère
      move.l      #Texte,A0
      bsr         lcd_out_st  * Sous-programme envoi de chaîne
                              * de caractère


*=============================================================*
      JMP         MONITEUR    *       Retour au Moniteur
*=============================================================*

*************************************
*     FIN PROGRAMME PRINCIPAL       *
*************************************


*************************************
*     LES SOUS-PROGRAMMES           *
*************************************

***    Busy ***
**********************

busy  move.b      #init_acces,ctrlaff
      move.b      #rdc,ctrlaff
      bsr         delay
      move.b      dbaff,D4          * Lecture data bus aff
      and.b       #03,D4            * Isoler ST0 STA1 (Status)
      cmp.b       #03,D4            * Si lcd pas prêt
      bne         busy              * attendre
      move.b      #init_acces,ctrlaff
      rts
```

```
***    wrcomm  : Ecriture commande placée dans D0 ***
*****************************************************

wr_comm
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Déposer la commande sur le bus
                                          * de donnée
      move.b      #wrc,ctrlaff            * Générer les impulsions
                                          * d'écriture d'une commande
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    wrdata  :  Ecriture data placée dans D0 ***
*************************************************

wr_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Déposer la data sur le bus de
                                          * donnée
      move.b      #wrd,ctrlaff            * Générer les impulsions
                                          * d'écriture d'une donnée
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    rddata  :  Lecture data à placer dans D0 ***
*************************************************

rd_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      #rdd,ctrlaff            * Générer les impulsions
                                          * d'écriture de
                                          * d'une donnée
      bsr         delay
      move.b      dbaff,D0                * Lire et mettre la donnée dans D0
      move.b      #init_acces,ctrlaff
      rts

***    init_aff  : Initialisation afficheur    ***
*************************************************

* Initialisation de la zone Adresse de début du Texte

init_aff
      clr.b       D0
      bsr         wr_data     * LSByte TH = 00
      bsr         wr_data     * MSByte TH = 00
      move.b      #$40,D0     * Commande d'écriture TH
      bsr         wr_comm     * Ecriture TH
      move.b      #$10,D0     * LSByte TA = $10 nombre de caractères/ligne
      bsr         wr_data
      clr.b       D0          * MSByte TA = 0
      bsr         wr_data
      move.b      #$41,D0     * Commande d'écriture TA
      bsr         wr_comm     * Ecriture  TA
```

```
* Initialisation  zone Adresse de début du Graphique

      clr.b       D0
      bsr         wr_data     * LSByte GH = 00
      move.b      #04,D0      * MSByte GH = 00
      bsr         wr_data     * Ecriture GH
      move.b      #$42,D0     * Commande d'écriture TH
      bsr         wr_comm
      move.b      #$10,D0     * LSByte GA = $10 nombre de caractères/ligne
      bsr         wr_data
      clr.b       D0          * MSByte GA = 0
      bsr         wr_data
      move.b      #$43,D0     * Commande d'écriture TA
      bsr         wr_comm     * Ecriture commande TH

* Définition Modes

      move.b      #DispMod,D0 * Mise en mode Graphique et/ou Texte
      bsr         wr_comm
      move.b      #ModSet,D0
      bsr         wr_comm     * Mise en mode Graphique OU Texte
      rts


***    lcd_gotoxy  : Init position caractère    ***
*****************************************************

lcd_gotoxy
      bsr         wr_data     * Ecrire la donnée LSByte contenue dans D0
      move.b      D1,D0       * MSByte dans dans D0
      bsr         wr_data     * Ecrire la donnée MSByte contenue dans D0
      move.b      #Pointeur,D0 * Code de commande du Pointeur dans D0
      bsr         wr_comm     * Ecriture de la commande du Pointeur dans D0
      rts

***    lcd_cls  : Effacement écran  ***
*************************************
lcd_cls
      clr.b       D0          * Adresse 00,00 dans D0 et D1
      clr.b       D1
      bsr         lcd_gotoxy  * Positionner sur le premier caractère
      move.w      #2048,D3    * Nombre d'octets de la mémoire écran
      clr.b       D0
suite bsr         wr_data
      move.b      #AutoInc,D0 * Code auto incrémentation dans D0
      bsr         wr_comm     * Incrémenter le pointeur de caractère
      move.b      #0,D0
      sub.w       #1,D3
      bne         suite       * Continuer d'effacer tout l'écran
      rts
```

```
***    lcd_out_str : Envoi chaîne de caractères  ***
*****************************************************

** Attention  : il faut soustraire la valeur $20 au code ASCII du caractère

lcd_out_str
      move.b      (A0)+,D0    * Pointeur un caractère
      cmp.b       #'$',D0     * Est-ce le caractère de fin ?
      beq         fin         * Fin si le caractère est le $
      sub.b       #$20,D0     * A cause du code ASCII décalé voir notice
      bsr         wr_data
      move.b      #AutoInc,D0
      bsr         wr_comm
      bra         lcd_out_str
fin   rts


***    delay      ***
**********************
delay move.b      #$10,D4
bc1   sub.b       #1,D4
      bne         bc1
      rts

tempo move.l      #$4000,D5
temp1 sub.l       #1,D5
      bne         temp1
      rts

      end
```

EID005_TP 2 ECRITURE D'UNE CHAINE DE CARACTERES EN MODE TEXTE SUR L'AFFICHEUR

2.1 Enoncé du sujet

Objectifs :	Etre capable d'utiliser les utilitaires rangés en bibliothèque, permettant la gestion de l'afficheur LCD graphique 128x64 pixels.
Cahier des charges :	**Sujet** Ecrire un programme en assembleur et en C qui réalisera l'affichage d'une chaîne de caractères en mode texte. La longueur maximale de la chaîne est de 128 caractères. On donne les coordonnées du 1er caractère.

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,
Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID210000
Carte Clavier afficheur Horloge Temps réel : EID00500
Câble de liaison USB, ou à défaut câble RS232, Réf : EGD000003
Alimentation AC/DC 8V, 1 A Réf : EGD000001,

Documentations nécessaires :

Document : DMS Carte Clavier Afficheur Horloge Temps réel : EID005000
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Durée : 1 séance de 3 heures

2.2 Eléments de solution

2.2.1 Description sommaire de l'afficheur

2.2.1.1 Représentation de l'afficheur LCD 128x64

Attention :

Pour des raisons de conformité avec la documentation du constructeur, les variables x et y représentent respectivement l'ordonnée (verticale) et l'abscisse (horizontale). Le point **x = 0, y = 0** est en haut à gauche et le point **x = 63, y = 127** en bas à droite de l'écran du LCD

Le contrôleur T6963C dispose d'une mémoire de 8 ko.

Mode texte *fig.1*

Dans l'étude qui suit, la zone texte est placée de l'adresse 0000 à 007F de la mémoire écran (VRAM), soit 128 caractères.
Octet de poids faible : **00 toujours fixe**
Octet de poids fort : **00 à 7F** en hexadécimal.
Le quartet de poids fort de cet octet désigne le numéro de ligne x.
Le quartet de poids faible désigne le numéro de la colonne y.

Exemple

Le caractère numéro **59** est placé en **: x = 3, y = B** ; ce qui donne en hexadécimal les valeurs : **x = 3, y = B** d'où en mémoire, les octets suivants pour le paramètre TH (Text Home Adress :

TH Adress lower = 0x3B (59 en decimal)
TH Adress upper = 0x00

N MODE TEXTE

Y

Colonne → / ligne ↓	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	**59**	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

X

fig.1

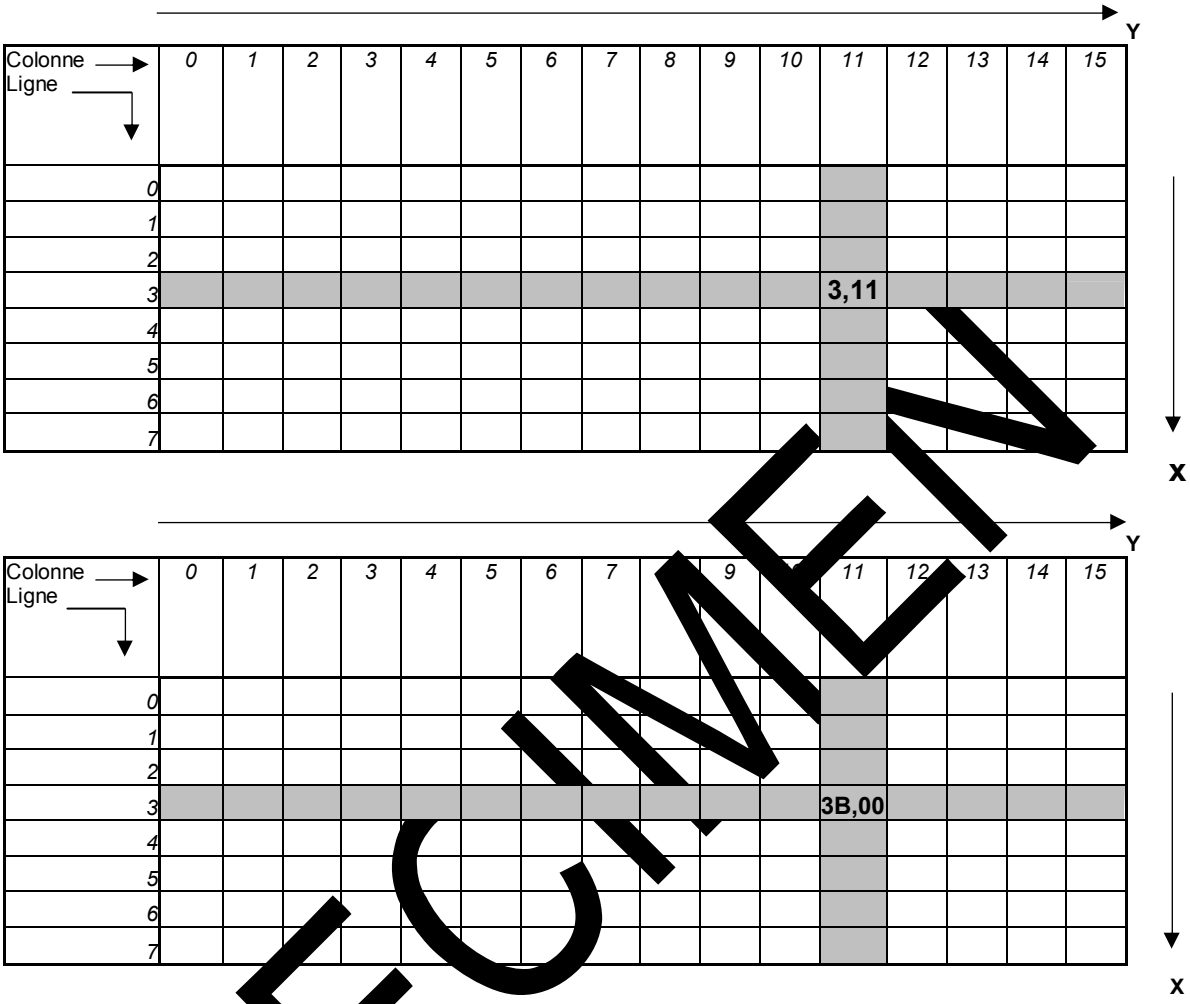


fig.1 suite

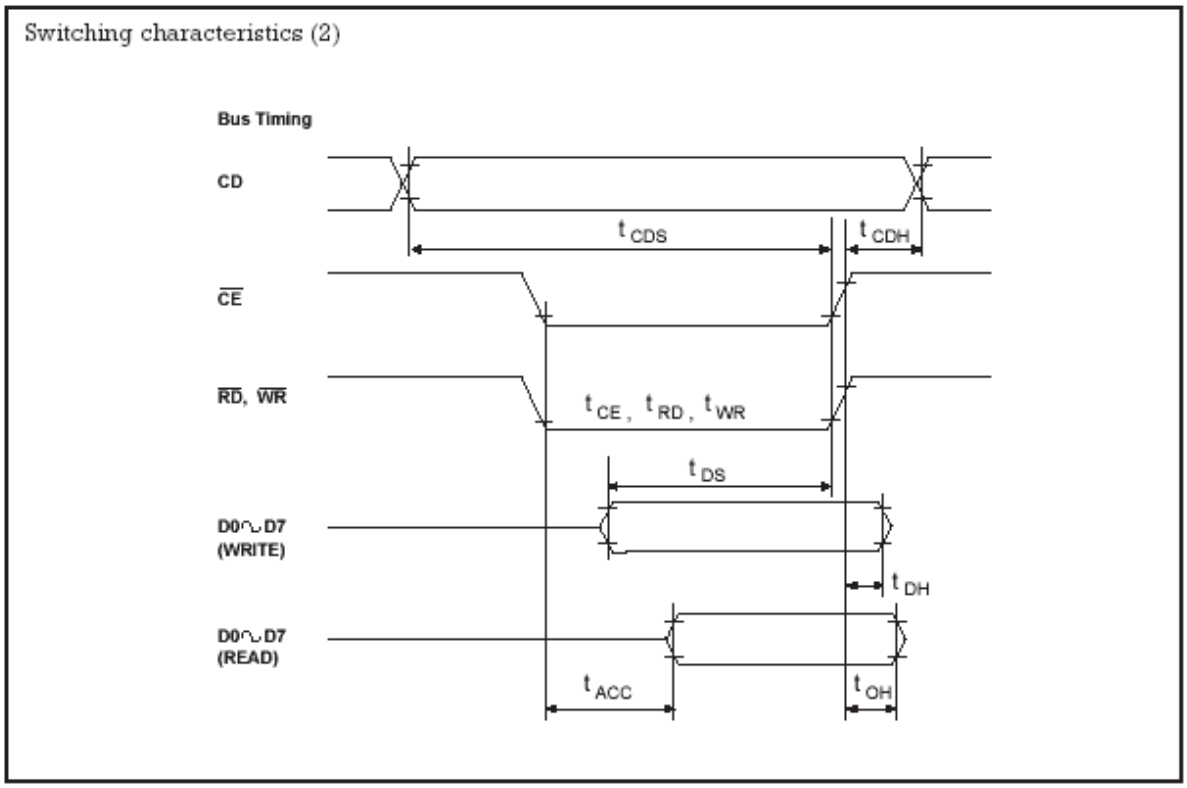
Générateur de caractères interne (CG ROM)

Le générateur de caractères interne utilise un code ASCII décalé de 0x20 ; exemple : la lettre A codée 0x41 en ASCII est représentée par la valeur 0x21 dans le T6963C (voir table ci-dessous).
Cela signifie que pour envoyer un caractère ASCII, il faut soustraire de son code, la valeur 0x20.

↓MSB LSB→	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	blank	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
1	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
2	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
3	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
4	\	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
5	p	q	r	s	t	u	v	w	x	y	z	{	:	}	~	blank
6	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
7	É			ô	ö	ò	û	ù	ÿ	ö	ü	¢	£	¥		ƒ

Chronogrammes Ecriture / Lecture de données ou commandes

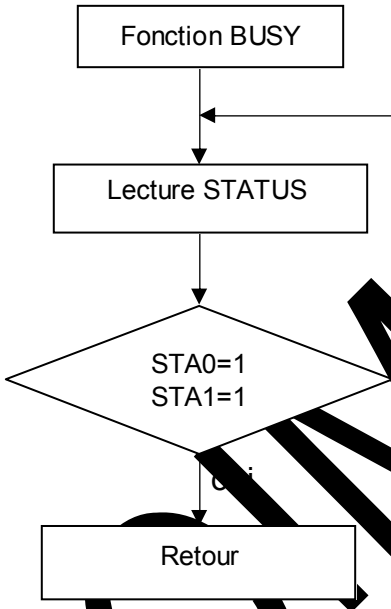
Ces chronogrammes doivent être générés pour chaque accès au LCD.
Ils sont réalisés et décrits en détail notamment dans les sous-programmes en Assembleur.



La génération de chacun de ces chronogrammes donnera lieu à un sous-programme correspondant.

2.2.1.2 Gestion de l'afficheur LCD

Avant chaque mouvement (écriture ou lecture) donnée (commande ou donnée) entre l'afficheur et le processeur de commande, il faut s'assurer que le LCD est prêt à exécuter l'opération.
D'où la nécessité de commencer par tester les bits d'état (ou de statut) de son registre d'état : STA0 et STA1 ; ce qui donne l'organigramme suivant :



2.2.1.3 Instructions de l'afficheur

Les instructions de l'afficheur sont données dans le tableau ci-dessous.

Les organigrammes ci-dessous définissent les modes d'écriture d'instruction de commande nécessitant 2, 1 ou 0 octets.

Les organigrammes suivants définissent en détail les modes d'écriture et de lecture des commandes et des données.

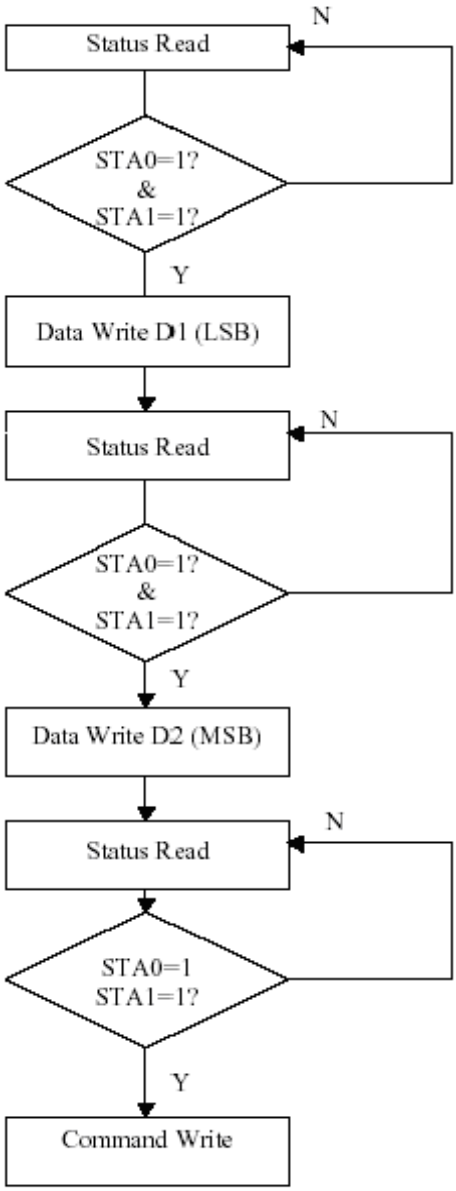
T6963C Instruction Set

Commands	D7	D6	D5	D4	D3	D2	D1	D0	Description	Execute Time
Pointer Set	0	0	1	0	0	N2	N1	N0		Status check
						0	0	1	Cursor Pointer Set	
						0	1	0	Offset Register Set	
						1	0	0	Address Pointer Set	
Control Word Set Commands	0	1	0	0	0	0	N1	N0		32 x 1/fosc
							0	0	Text Home Address Set	
							0	1	Text Area Set	
							1	0	Graphic Home Address Set	
							1	1	Graphic Area Set	
Mode Set	1	0	0	0	CG	N2	N1	N0		32 x 1/fosc
					0				CG ROM Mode	
					1				CG RAM Mode	
						0	0	0	"OR" Mode	
						0	0	1	"EXOR" Mode	
						0	1	1	"AND" Mode	
						1	0	0	Text only (attribute capability)	
Display Modes	1	0	0	1	N3	N2	N1	N0		32 x 1/fosc
					0				Graphics Off	
					1				Graphics On	
						0			Text Off	
						1			Text On	
							0		Cursor Off	
							1		Cursor On	
								0	Cursor blink Off	
								1	Cursor blink On	
Cursor Pattern Select	1	0	1	0	0	N2	N1	N0	N2–N0: No. of lines for cursor +1	32 x 1/fosc
						0	0	0	Bottom Line cursor	
						0	0	1	2 line cursor	
						|	|	|		
						1	1	1	8 line cursor (block cursor)	
Data Auto Read/Write	1	1	0	0	0	0	N1	N0		32 x 1/fosc
							0	0	Data Auto Write Set	
							0	1	Data Auto Read Set	
							1	0	Auto reset (Address pointer auto-incremented) for continuous rd/wr	
Data Read/Write	1	1	0	0	0	N2	N1	N0		
						0			Address Pointer up/down	
						1			Address Pointer unchanged	
							0		Address Pointer up	
							1		Address Pointer down	
								0	Data Write	
								1	Data Read	
Screen Peeking	1	1	1	0	0	0	0	0	Read Displayed Data	Status
Screen Copy (Note 3)	1	1	1	0	1	0	0	0	Copies 1 line of displayed data whose address is indicated by the Address Pointer to Graphic RAM area	Status check
Bit Set/Reset	1	1	1	1	N3	N2	N1	N0	N2–N0 indicates the bit in the pointed address	Status check
					0				Bit Reset	
					1				Bit Set	
						0	0	0	Bit 0 (LSB)	
						0	0	1	Bit 1	
						|	|	|		
						1	1	1	Bit 7 (MSB)	

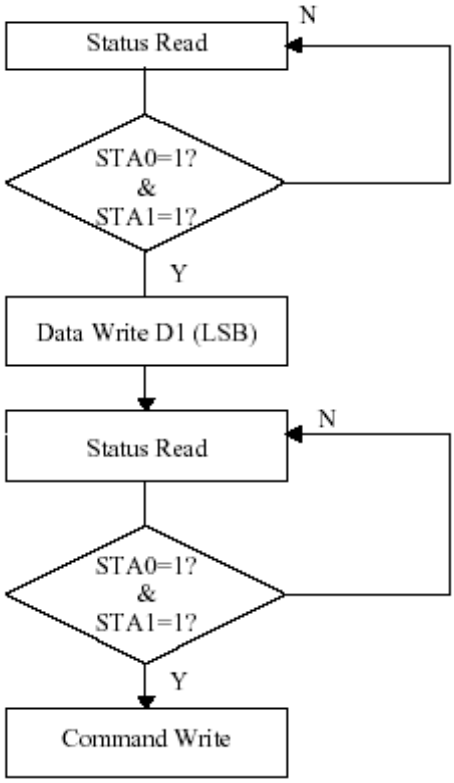
Note:

1. * = DONT CARE

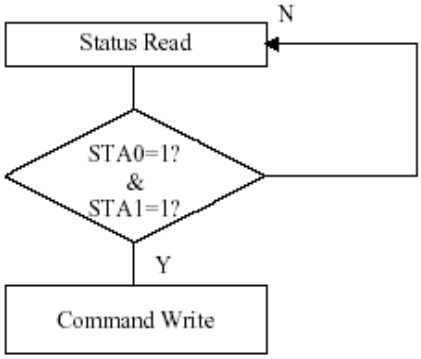
Commande nécessitant 2 octets



Commande nécessitant 1 octet

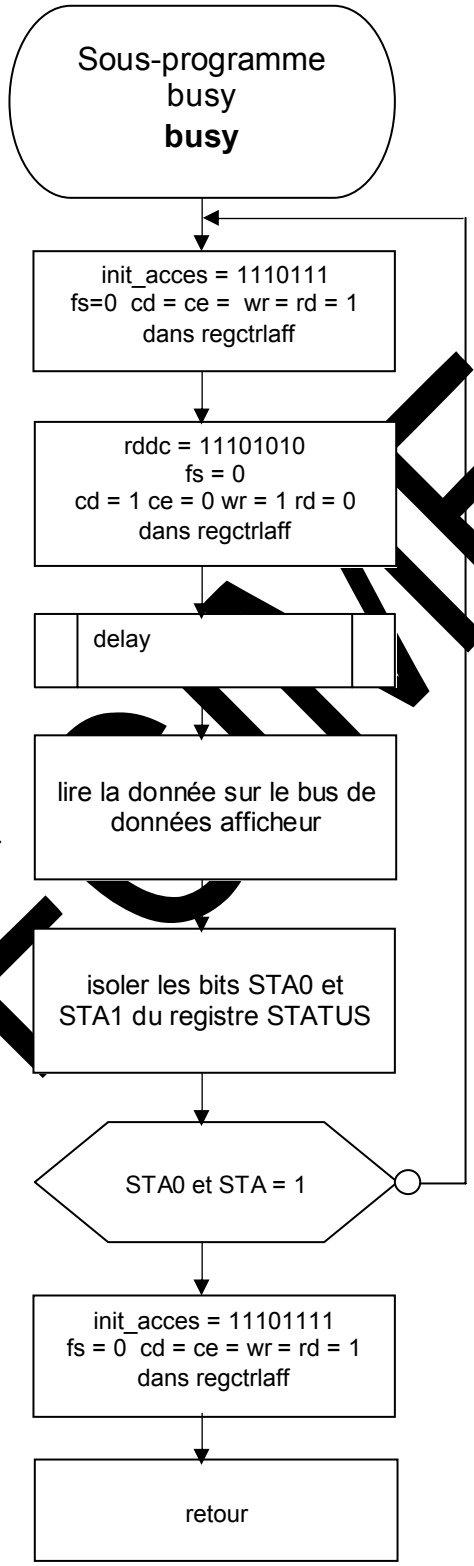


Commande nécessitant 0 octet

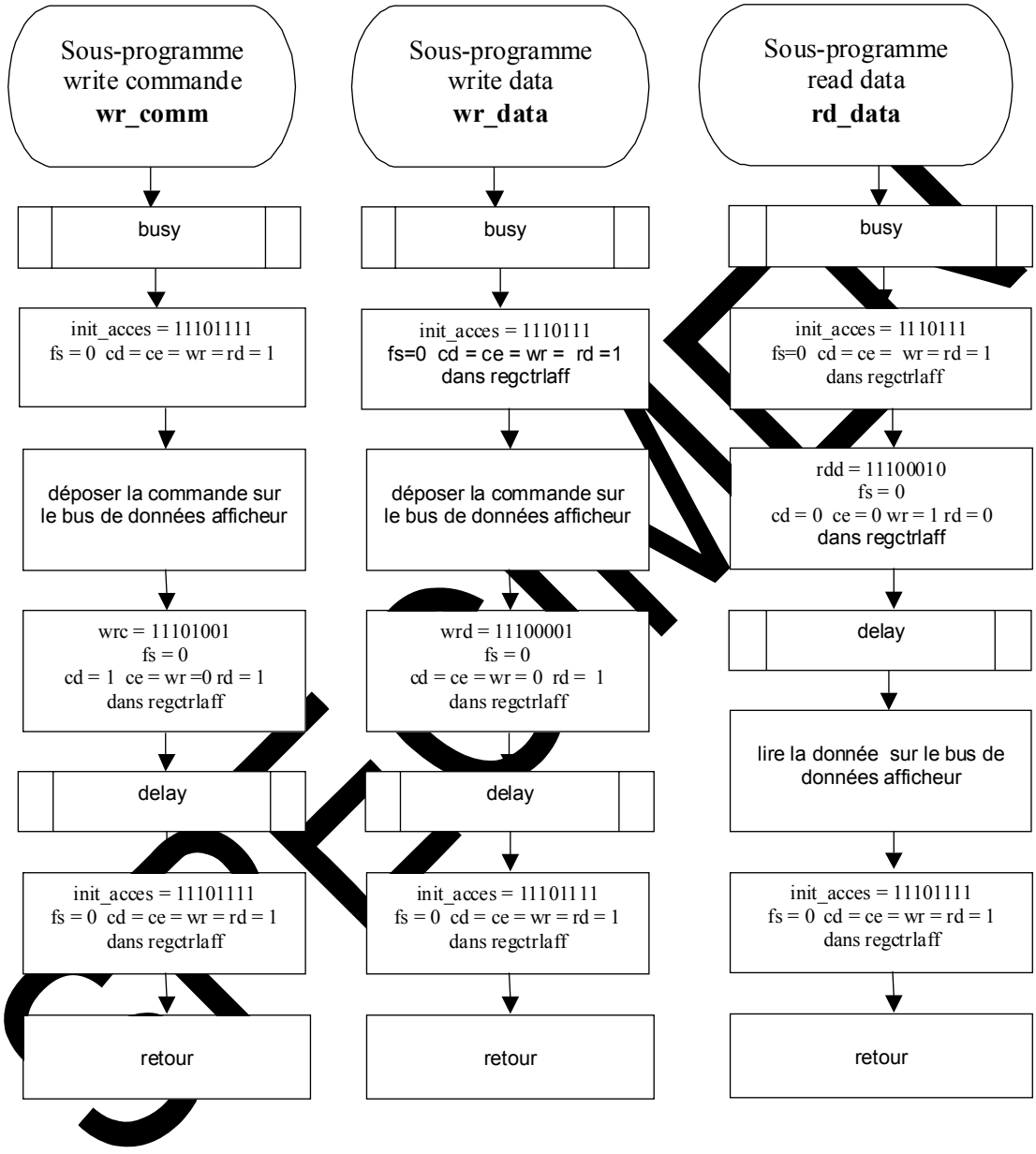


Organigrammes détaillés des sous-programmes principaux

Sous-programme busy



Sous-programmes écriture lecture



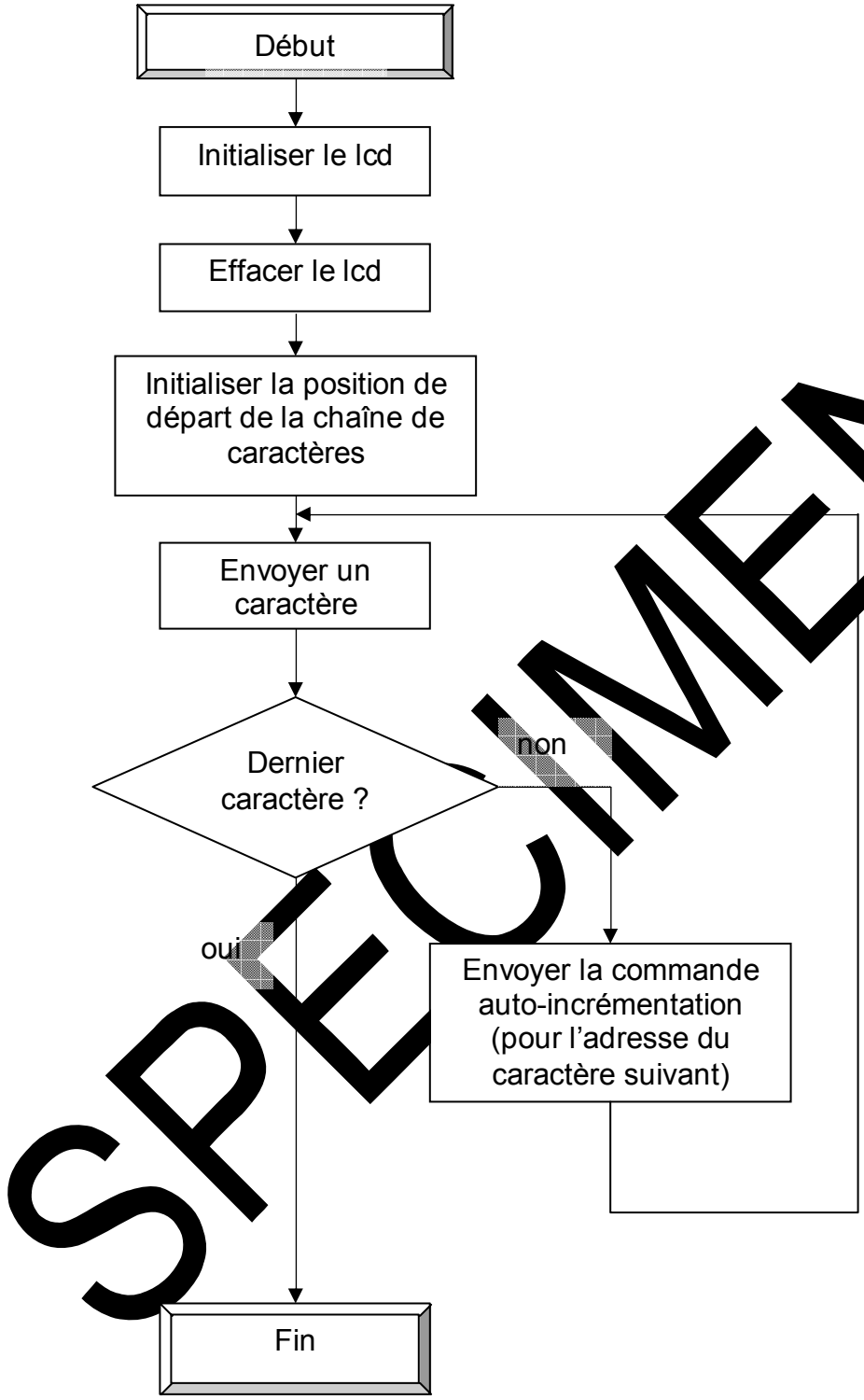
2.2.1.4 Organigramme principal

Vous disposez des utilitaires suivants avec leurs commentaires ; tous ces utilitaires font déjà appel au test de STA0 et STA1

Utilitaires	Commentaires
`void init_aff()`	Initialisation des paramètres TH, TA, GH GA Mode …
`void lcd_cls()`	Effacement de l'écran
`void lcd_write_command(unsigned char **commande**)`	Ecriture d'une commande
`void lcd_write_data(unsigned char **data**)`	Ecriture d'une donnée
`void lcd_gotoxy(unsigned char **px**, unsigned char **py**)`	Définition de la position d'un caractère ou d'un pixel ; px et py sont les données lower et upper.

Important :

Pour utiliser ses fonctions ainsi que le fichier afficheur avec le compilateur C/C++, il faut, dans le menu du logiciel eid210, aller dans le menu configuration puis gnu C/C++.Ensuite aller dans l'onglet *linker*, cliquer sur *ajouter*, sélectionner le fichier « EID005_Lib.o » et finir en cliquant sur ouvrir.


Début
Initialiser le lcd
Effacer le lcd
Initialiser la position de départ de la chaîne de caractères
Envoyer un caractère
Dernier caractère ?
non
oui
Envoyer la commande auto-incrémentation (pour l'adresse du caractère suivant)
Fin

2.2.2 Programme en C

```
/*******************************************************
*     TP SUR LA CARTE EID005                           *
*******************************************************
*     Ecrire un programme en Assembleur                *
*     et en C et qui réalise l'affichage               *
*     d'une chaîne de caractères.                      *
*     longueur maximale de la chaîne : 128 caractères *
*******************************************************
*     Nom du FICHIER : EID005_TP2.c                    *
*     **************                                   *
*******************************************************/

//    Inclusion des fichiers de définition

#include "eid005.h"
#include <stdio.h>
#include <string.h>
#include <math.h>


//*********************
//  DECLARATIONS
//*********************

unsigned char Texte []= { "EID005000 :     CLAVIER,        AFFICHEUR,
HORLOGE TEMPS   REEL $"};

// Les espaces dan le texte permettent d'éviter de couper un mot

//=====================================
//    FONCTION PRINCIPALE
//=====================================

main()
{
int i ;                         //    Variable de comptage des caractères
init_afficheur();               //    Initialisation de l'afficheur
lcd_effacer();                  //    Effacement de l'afficheur
lcd_gotoxy(0,07,00              //    Définition de la position de départ de
la chaîne

//    Boucle de comptage du nombre de caractères

for (i=0; Texte [i]!= 0x24; i++)    // Fin de comptage par la détection du
caractère "$"
{
lcd_write_data((unsigned char) (Texte[i]-0x20));     // Envoyer un
caractère
lcd_write_command(0xC0);      // Envoyer la commande d'incrémentation
d'adresse
}
}

//    Fin du programme
```

2.2.3 Programme en Assembleur

```
*******************************************************
*     TP SUR LA CARTE EID005                          *
*******************************************************
*     Ecrire un programme en Assembleur               *
*     et en C et qui réalise la lecture d'un touche   *
*     sur un clavier matricé 4 x 4, en mode pooling   *
*******************************************************
*     Nom du FICHIER : EID005_TP2.src                 *
*     **************                                  *
*******************************************************

**    La commande ou la donnée à écrire sont d'abord rangées dans D
**    La donnée lue est rangée dans D0
**    La position x,y d'un caractère est placée respectivement dans D0 D1
**    L'adresse de début d'une chaîne de caractère est placée dans A0

*******************************************************
*     Bits du registre de contrôle de l'afficheur     *
*     ctrlaff :   b0=rd                               *
*                 b1=wr                               *
*                 b2=ce                               *
*                 b3=cd                               *
*                 b4=fs                               *
*******************************************************

* Inclusion du fichier définissant les différents labels
* de l'EID210

      include EID210.def

      section var

*     Définition des adresses physiques de l'afficheur et du clavier

eid005      equ   $8000          * Adresse de base Carte clavier_afficheur_rtc
ctrlaff     equ   eid005+3       * Registre contrôle afficheur : bus de
                                 * contrôle
dbaff       equ   eid005+4       * Bus de données afficheur
status      equ   eid005+6       * Registre de status de la carte
reg_ctrl    equ   eid005+7       * Registre de contrôle de la carte
                                 * reg_clavier     equ   EID005+5

**    Table de définition des paramètres de l'afficheur

*************************************************************
*           $00,$04,$42        * Graphic Hom Adress et Command
*           $10,$00,$43        * Graphic Area et Command
*           $00,$00,$40        * Text Hom Adress et Command
*           $10,$00,$41        * Text Area et Command

ModSet      equ   $80            * OR  Graphique ou Texte
Pointeur    equ   $24            * Commande pointeur
DispMod     equ   $94            * Affichage Texte et ou Graphique
AutoInc     equ   $C0            * Auto incrémentation pixel ou caractère
```

```
init_acces  equ   $EF   * Pour accéder au bus de données du lcd avec fs=0
wrc         equ   $E9   * cd=1, ce=0, wr=0, rd=1, fs=0 1110 1001
                        *write commande
rdc         equ   $EA   * cd=1, ce=0, wr=1, rd=0, fs=0 1110 1010
                        *read commande (Status)
wrd         equ   $E1   * cd=0, ce=0, wr=0, rd=1, fs=0 1110 0001 write data
rdd         equ   $E2   * cd=0, ce=0, wr=1, rd=0, fs=0 1110 0010 reda data

**    Une chaîne de caractère se termine par le caractère '$'

**    Attention les espaces dans le texte servent à un afficher sans couper
**    les mots

Texte dc.b  'EID005 CLAVIER  AFFICHEUR       HORLOGE TEMPS   REEL  '


      section           code

*******************************
*     PROGRAMME PRINCIPALE    *
*******************************

      bsr         init_aff    * Initialisation afficheur
      bsr         lcd_cls     * Effacer écran

* Envoi Texte : x=2, y=1 : dans D0

      move.b      #$20,D0     * Byte 1 : ligne 0 colonne 1
      clr.b       D1          * MSByte TH
      bsr         lcd_gotoxy  * Positionnement 1er caractère
      move.l      #Texte,A0

***    lcd_out_str : envoi chaîne de caractères lcd_out_str  ***
*******************************************************

** Attention il faut soustraire la valeur $20 au code ASCII du caractère

lcd_out_str
      move.b      (A0)+,D0    * Pointeur un caractère
      cmp.b       #'$',D0     * Est-ce le caractère de fin ?
      beq         fin         * Fin si le caractère est le $
      sub.b       #$20,D0     * A cause du code ASCII décalé voir notice
      bsr         wr_data
      move.b      #AutoInc,D0
      bsr         wr_comm
      bra         lcd_out_str

*=========================================================*
fin   JMP         MONITEUR    *     Retour au Moniteur
*=========================================================*

*******************************
*     FIN PROGRAMME PRINCIPAL *
*******************************
```

```
*******************************
*     LES SOUS-PROGRAMMES     *
*******************************

***    Busy ***
***************

busy  move.b      #init_acces,ctrlaff
      move.b      #rdc,ctrlaff
      bsr         delay
      move.b      dbaff,D4          * Lecture data bus aff
      and.b       #03,D4            * Isoler ST0 STA1 (Status)
      cmp.b       #03,D4            * Si lcd pas prêt
      bne         busy              * attendre
      move.b      #init_acces,ctrlaff
      rts

***    wrcomm  : Ecriture commande placée dans D0 ***
*****************************************************

wr_comm
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Déposer la commande sur le bus
                                          *    donnée
      move.b      #wrc,ctrlaff            * Générer les impulsions
                                          * d'écriture de
                                          * d'une commande
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    wrdata  :  Ecriture data placée dans D0 ***
**************** ****** ***** ************

wr_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Déposer la data sur le bus de
                                          * donnée
      move.b      #wrd,ctrlaff            * Générer les impulsions
                                  * d'écriture de
                                          * d'une donnée
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts
```

```
***    rddata  :  Lecture data à placer dans D0 ***
*************************************************

rd_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      #rdd,ctrlaff            * Générer les impulsions
                                          * d'écriture de
                                          * d'une donnée
      bsr         delay
      move.b      dbaff,D0          * Lire et la mettre la donnée dans D0
      move.b      #init_acces,ctrlaff
      rts

***    init_aff  : Initialisation afficheur    ***
*************************************************

init_aff

* Initialisation de la zone Adresse de début du Texte

      clr.b       D0
      bsr         wr_data     * LSByte TH = 00
      bsr         wr_data     * MSByte TH = 00
      move.b      #$40,D0     * Commande d'écriture TH
      bsr         wr_comm     * Ecriture TH
      move.b      #$10,D0     * LSByte TA = $10 nombre de caractères/ligne
      bsr         wr_data
      clr.b       D0          * MSByte TA = 0
      bsr         wr_data
      move.b      #$41,D0     * Commande d'écriture TA
      bsr         wr_comm     * Ecriture TA

* Initialisation de la zone Adresse de début du Graphique

      clr.b       D0
      bsr         wr_data     * LSByte GH = 00
      move.b      #$00,D0     * MSByte GH = 00
      bsr         wr_data     * Ecriture GH
      move.b      #$42,D0     * Commande d'écriture TH
      bsr         wr_comm
      move.b      #$10,D0     * LSByte GA = $10 nombre de caractères/ligne
      bsr         wr_data
      clr.b       D0          * MSByte GA = 0
      bsr         wr_data
      move.b      #$43,D0     * Commande d'écriture TA
      bsr         wr_comm     * Ecriture commande TH

* Définition Modes

      move.b      #DispMod,D0 * Mise en mode Graphique et/ou Texte
      bsr         wr_comm
      move.b      #ModSet,D0
      bsr         wr_comm     * Mise en mode Graphique OU Texte
      rts
```

```
***    lcd_gotoxy  : Init position caractère   ***
*************************************************

lcd_gotoxy
      bsr         wr_data     * Ecrire la donnée LSByte contenue dans D0
      move.b      D1,D0       * MSByte dans dans D0
      bsr         wr_data     * Ecrire la donnée MSByte contenue dans D0
      move.b      #Pointeur,D0 * Code de commande du Pointeur dans D0
      bsr         wr_comm     * Ecriture de la commande du Pointeur dans D0
      rts

***    lcd_cls  : Effacement écran  ***
***************************************
lcd_cls
      clr.b       D0          * Adresse 00,00 dans D0 et
      clr.b       D1
      bsr         lcd_gotoxy  * Positionner sur le premier caractère
      move.w      #2048,D3    * Nombre d'octets de la mémoire écran
      clr.b       D0
suite bsr         wr_data
      move.b      #AutoInc,D0 * Code auto incrémentation dans D0
      bsr         wr_comm     * Incrémenter le pointeur de caractère
      move.b      #0,D0
      sub.w       #1,D3
      bne         suite       * Continuer d'effacer toute l'écran
      rts

***    delay      ***
**********************
delay move.b      #$10,D4
bc1   sub.b       #1,D4
      bne         bc1
      rts



tempo move.l      #400,D5
temp1 sub.l       #1,D5
      bne         temp1
      rts

      end
```

EID005_TP 3 LECTURE D'UN CLAVIER MATRICE EN MODE POLLING

3.1 Enoncé du sujet

Objectifs :	Etre capable d'utiliser les utilitaires rangés en bibliothèque, permettant la gestion d'un clavier matricé 16 points (4x4).
Cahier des charges :	**Sujet** Ecrire un programme en assembleur et en C qui réalisera la lecture d'une touche du clavier matricé en mode pooling. Cette lecture se fera par détection de l'activation d'une touche appuyée.

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,
Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID210000
Carte Clavier Afficheur Horloge Temps réel : EID00500
Câble de liaison USB ou à défaut câble RS232, Réf : EGD000003
Alimentation VAC/AC 8V, 1 A Réf : EGD000001,

Documentations nécessaires :

Document : DMS Carte Clavier Afficheur Horloge Temps réel : EID005000
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Durée : 1 séance de 3 heures

3.2 Eléments de solution

3.2.1 Description sommaire du clavier

3.2.1.1 Représentation du clavier 4x4 fig.1

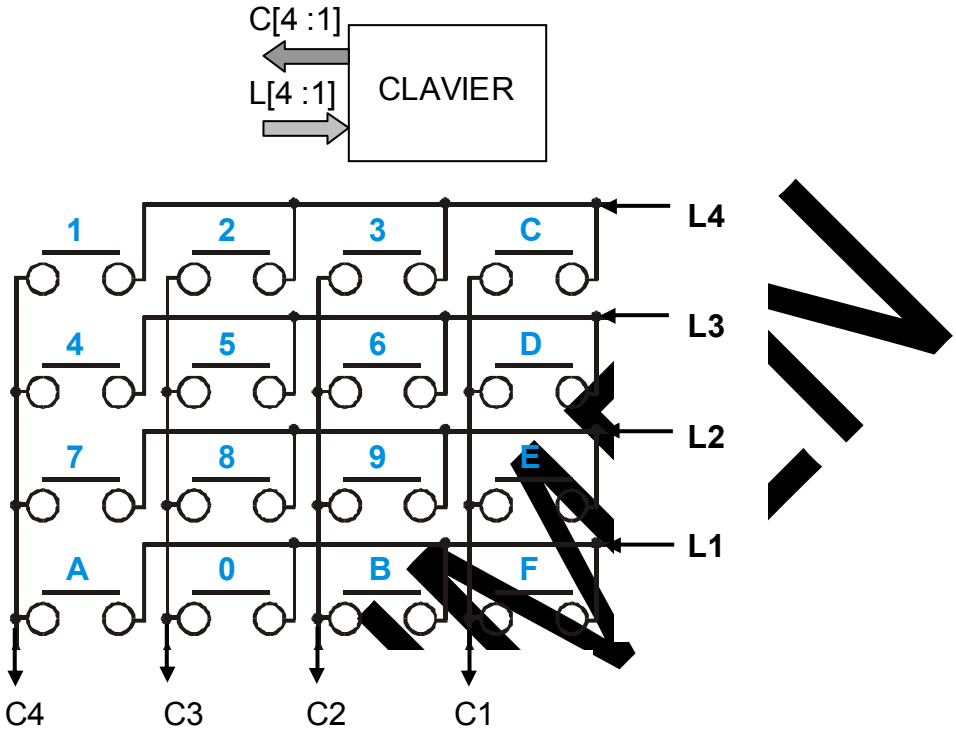


fig.1

Les 4 lignes du clavier sont câblées sur la sortie de l'EPLD de gestion de l'EID005.

Chaque sortie de l'EPLD est munie d'une résistance de pull-up (tirage à Vcc).

Les 4 colonnes sont câblées sur des entrées.

3.2.1.2 Principe général de lecture d'un clavier matricé

Lorsque les sorties commandant les 4 lignes sont au niveau logique **1**, les 4 bits correspondant aux 4 colonnes sont à **1** quelque soit le nombre de touches appuyées.

Le principe de la lecture par la méthode pooling consiste à fixer **1 ligne L_j 0**, **les 3 autres à 1** (balayage lignes) puis à détecter le numéro de la **colonne C_i** à **0** (balayage colonnes).

L'intersection $L_j x C_i$ donne le caractère correspondant.

Il ne reste plus qu'à coder ce caractère pour lui donner la valeur binaire ou hexadécimale voulue.

3.2.1.3 Lecture d'une touche appuyée en mode pooling

Pour faire cette lecture il faut

a) balayer les lignes en positionnant une ligne n° i à 0
b) lire les colonnes
c) tester si une colonne est à 0 (au moins 1 touche appuyée)
d) déterminer le poids de la colonne
e) extraire la valeur de la touche
f) afficher la valeur de la touche à l'écran du PC.
g) tester toutes les touches.

3.2.1.4 Organigramme principal

Le clavier est codé par une matrice 4x4 en hexadécimal comme indiqué ci-dessous.

Attention à la position des lignes dans le tableau de codage par rapport à celle de la matrice elle-même.

Poids de la colonne	8	4	2	1
C_I ——— L_J ———	C_4	C_3	C_2	C_1
L4	01	02	03	0C
L3	04	05	06	0D
L2	07	08	09	0E
L1	0A	00	0B	0F

Les lignes et les colonnes sont décrites dans le fichier eid005.h de la carte EID005.

La portion correspondante est ci-dessous.

```
/* clavier */
union reg_clavier
{
struct
        {
        unsigned char ligne:4;
        unsigned char colonne:4;
        } matrice;
unsigned char valeur;
};
#define touche       status.r_bit.b0
#define clavier (*(union reg_clavier *) (eid005+5))
```

Une ligne i est définie par la variable :

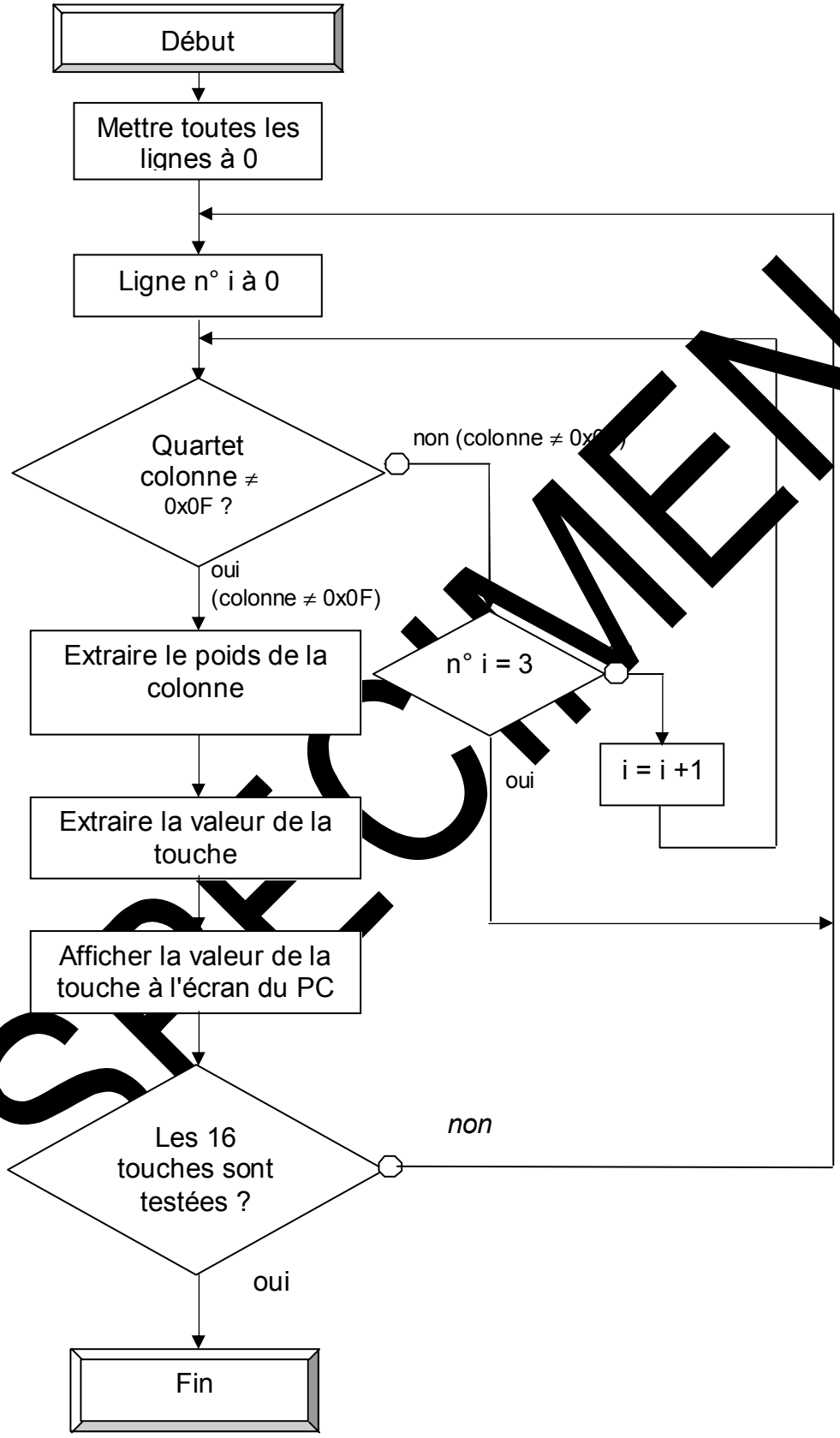
- clavier.matrice.ligne = i ; (0 =< i =< 3)

Les colonnes sont lues à travers la variable :

- clavier.matrice.colonne.

Important :

Pour utiliser le clavier afficheur avec le compilateur C/C++, il faut, dans le menu du logiciel eid210, aller dans le menu configuration puis gnu C/C++.Ensuite aller dans l'onglet *linker*, cliquer sur *ajouter*, sélectionner le fichier EID005_Lib.o et finir en cliquant sur ouvrir.


Début
Mettre toutes les lignes à 0
Ligne n° i à 0
Quartet colonne ≠ 0x0F ?
non (colonne ≠ 0x0F)
oui (colonne ≠ 0x0F)
Extraire le poids de la colonne
n° i = 3
oui
i = i +1
Extraire la valeur de la touche
Afficher la valeur de la touche à l'écran du PC
Les 16 touches sont testées ?
non
oui
Fin

3.2.2 Programme en C

```
/*****************************************************
*     TP SUR LA CARTE EID005                         *
*****************************************************
*     Ecrire un programme en Assembleur              *
*     et en C et qui réalise la lecture d'un touche  *
*     sur un clavier matricé 4 x 4, en mode pooling  *
*****************************************************
*     Nom du FICHIER : EID005_TP3.c                  *
*     **************                                 *
*****************************************************/

//    Inclusion des fichiers de définition

#include "eid005.h"
#include <stdio.h>
#include "eid210.h"

//============================
//    FONCTION PRINCIPALE
//============================


main ()
{
short    Touche[4][4] = { 0x0A, 0x0B, 0x0C, 0x0F,
                          0x07, 0x08, 0x09, 0x0E,
                          0x04, 0x05, 0x06, 0x0D,
                          0x01, 0x02, 0x03, 0x00 } ;
short V, ValeurTouche;
int tmp ;
int i, j, k, m;

//    Positionner toutes les colonnes de la matrice à 0
clavier.matrice.ligne = 0x0F; // Elément ligne (4 bits) de la structure
appartenant
                              //à la variable clavier de l'union
reg_clavier
j=0;
do
{j++;
printf ("Appuyer sur une touche \n");
//    Touche appuyée ?
//    Balayage ligne
while ((clavier.matrice.colonne & 0x0F) != 0x0F)                  // Attente
appui touche
{
      for (i = 0; i<4; )
      {
            clavier.matrice.ligne = ~(1 << i); // Toutes les colonnes à 1
sauf la n° i
            tmp = i;
            if (touche == 1) break;             // touche = 1 ==> au moins
une touche appuyée
            i++ ;
            for (k = 0 ; k<10000 ; k++);        // temporisation
      }
}
//    Extraire le poids de la colonne
```

```
//    clavier.matrice.colonne = Elément colonne (4 bits) de la structure
//    appartenant à la variable clavier de l'union reg_clavier
//    Puis estraire la valeur de la touche appuyée
//    Extraire la valeur de la touche appuyée
//    clavier.matrice.colonne & 0xF;

switch((~(clavier.matrice.colonne)) & 0x0F )
{
      case 1 : ValeurTouche= Touche [tmp][3]; break; //Colonne poids 1
      case 2 : ValeurTouche= Touche [tmp][2]; break; //Colonne poids 2
      case 4 : ValeurTouche= Touche [tmp][1]; break; //Colonne poids 4
      case 8 : ValeurTouche= Touche [tmp][0]; break; //Colonne poids 8

}

//    Afficher loa valeur de la touche à l'écran du PC
printf ("La touche appuyee est : %x\n\n", ValeurTouche );
while (touche)                // Attente relâche touche
{
            for (k = 0 ; k<1000000 ; k++);      //temporisation
}
}
//    Toutes les touches sont testées ?
while (j<17);

}

//    Fin du programme principal
```

3.2.3 Programme en Assembleur

```
*******************************************************
*     TP SUR LA CARTE EID005                          *
*******************************************************
*     Ecrire un programme en Assembleur               *
*     et en C et qui réalise la lecture d'un touche   *
*     sur un clavier matricé 4 x 4, en mode pooling   *
*******************************************************
*     Nom du FICHIER : EID005_TP3.src                 *
*     **************                                  *
*******************************************************


**    La commande ou la donnée à écrire sont d'abord rangées dans D0
**    La donnée lue est rangée dans D0
**    La position x,y d'un caractère est placée respectivement dans D0 et
D1
**    L'adresse de début d'une chaîne de caractère est placée dans A0

*************************************************
*     Bits du registre de contrôle de l'afficheur *
*     ctrlaff :   b0=rd                           *
*                 b1=wr                           *
*                 b2=ce                           *
*                 b3=cd                           *
*                 b4=fs                           *
*************************************************

* Inclusion du fichier définissant les différents labels
* de l'EID210

      include EID210.def
      section var

*     Définition des adresses physiques de l'afficheur et du clavier

eid005      equ   $30000       * Adresse de base Carte clavier_afficheur_rtc
ctrlaff     equ   eid005+3     * Registre contrôle afficheur : bus de
                               * contrôle
dbaff       equ   eid005+4          * Bus de données afficheur
statut      equ   eid005+6     * Registre de status de la carte
reg_ctrl    equ   eid005+7     * Registre de contrôle de la carte
reg_clavier equ   EID005+5

*     Table de définition des paramètres de l'afficheur

TabDef      dc.b  $00,$04,$42        * GH
            dc.b  $10,$00,$43        * GA
            dc.b  $00,$00,$40        * TH
            dc.b  $10,$00,$41        * TA
ModSet      equ   $80                * OR  Graphique ou Texte
Pointeur    equ   $24                * Commande pointeur
DispMod     equ   $94                * Affichage Texte et ou Graphique
AutoInc     equ   $C0                * Auto incrémentation pixel ou caractère
```

```
init_acces  equ   $EF   * Pour accéder au bus de données du lcd avec fs=0
wrc         equ   $E9   * cd=1, ce=0, wr=0, rd=1, fs=0 1110 1001
                        * write  commande
rdc         equ   $EA   * cd=1, ce=0, wr=1, rd=0, fs=0 1110 1010
                        * read commande (Status)
wrd         equ   $E1   * cd=0, ce=0, wr=0, rd=1, fs=0 1110 0001
                        * write data
rdd         equ   $E2   * cd=0, ce=0, wr=1, rd=0, fs=0 1110 0010
                        * reda data

Texte1      dc.b  'EDD----------005 $ '
Texte2      dc.b  '?!$'
Texte3      dc.b  'FIN $4


**    Tableau des valeurs des touches du clavier

Colonne0          dc.b  $0C,$0D,$0E,$0F
Colonne1          dc.b  $03,$06,$09,$0B
Colonne2          dc.b  $02,$05,$08,$00
Colonne3          dc.b  $01,$04,$07,$0A

**    Tableau des coordonnées d'affichage des touches du clavier

                  dc.b  $1A,$3A,$5A,$7A
                  dc.b  $18,$38,$58,$78
                  dc.b  $16,$36,$56,$7
                  dc.b  $14,$3     ,$74

      section           code

**********************************
*     PROGRAMME PRINCIPALE       *
**********************************

      bsr         init_lcd      * Initialisation Afficheur
      bsr         lcd_cls       * Effacement écran

* Envoi Texte1 : x=0, y=0
      move.b      #$00,D0       * LSByte TH : ligne 0 colonne 1
      move.b      D1            * MSByte TH
      bsr         lcd_gotoxy
      move.l      #Texte1,A0
      bsr         lcd_out_str

* Définition début affichage touches clavier

      move.l      #15,D7
      clr.b       D1
```

```
* Lecture du clavier

test_clav
      bsr         Touch
      bsr         Val_Touch   * La valeur de la touche est dans D0
      cmp.b       #$0A,D0     * Conversion Hexa --> ASCII décalé de $20,
      bcc         sup10       * voir Générateur de caractères T6963C :
      add.b       #$10,D0     * add $10 si code < $0A
      bra         envoi       *
sup10 add.b       #$17,D0     * si non add $17
envoi bsr         wr_data
      move.b      #$C0,D0
      bsr         wr_comm
      dbeq        D7,test_clav

* Envoi Texte2 : x=7, y=0

      move.b      #$70,D0     * LSByte TH : ligne 7 colonne 0
      clr.b       D1          * MSByte TH
      bsr         lcd_gotoxy
      move.l      #Texte2,A0
      bsr         lcd_out_str

* Envoi Texte3 : x=7, y=d

      move.b      #$7d,D0     * LSByte TH : ligne 7 colonne 13
      clr.b       D1          * MSByte TH
      bsr         lcd_gotoxy
      move.l      #Texte3,A0
      bsr         lcd_out_str

*=============================================================*
      JMP         MONITEUR          Retour au Moniteur
*=============================================================*

*********************************
*     FIN PROGRAMME PRINCIPAL   *
*********************************

*********************************
*     LES SOUS PROGRAMMES       *
*********************************

**    SOUS PROGRAMMES GESTION DE L'AFFICHEUR    **
*****************************************************

***    Busy **
**********

busy  move.b      #init_acces,ctrlaff
      move.b      #rdc,ctrlaff
      bsr         delay
      move.b      dbaff,D4          * Lecture data bus aff
      and.b       #03,D4            * Isoler ST0 STA1 (Status)
      cmp.b       #03,D4            * Si lcd pas prêt
      bne         busy              * attendre
      move.b      #init_acces,ctrlaff
      rts
```

```
***    wrcomm  : Ecriture commande placée dans D0 ***
*****************************************************

wr_comm
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Déposer la commande sur le bus
                                          * de donnée
      move.b      #wrc,ctrlaff            * Générer les impulsions
                                          *d'écriture de
                                          * d'une commande
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    wrdata  :  Ecriture data placée dans D0 ***
*************************************************

wr_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Déposer la data sur le bus de
                                          * donnée
      move.b      #wrd,ctrlaff            * Générer les impulsions
                                          * d'écriture de
                                          * d'une donnée
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    rddata  :  Lecture data à placer dans D0 ***
*************************************************

rd_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      #rdc,ctrlaff            * Générer les impulsions
                                          * d'écriture de
                                          * d'une donnée
      bsr         delay
      move.b      dbaff,D0                * Lire et mettre la donnée dans D0
      move.b      #init_acces,ctrlaff
      rts

***    init_aff  : Initialisation afficheur    ***
*************************************************

init_aff

*  Initialisation de la zone Adresse de début du Texte

      clr.b       D0
      bsr         wr_data     * LSByte TH = 00
      bsr         wr_data     * MSByte TH = 00
      move.b      #$40,D0     * Commande d'écriture TH
      bsr         wr_comm     * Ecriture TH
      move.b      #$10,D0     * LSByte TA = $10 nombre de caractères/ligne
      bsr         wr_data
      clr.b       D0          * MSByte TA = 0
      bsr         wr_data
```

```
      move.b      #$41,D0     * Commande d'écriture TA
      bsr         wr_comm     * Ecriture  TA

* Initialisation de la zone Adresse de début du Graphique

      clr.b       D0
      bsr         wr_data     * LSByte GH = 00
      move.b      #04,D0      * MSByte GH = 00
      bsr         wr_data     * Ecriture GH
      move.b      #$42,D0     * Commande d'écriture TH
      bsr         wr_comm
      move.b      #$10,D0     * LSByte GA = $10 nombre de caractères/ligne
      bsr         wr_data
      clr.b       D0          * MSByte GA = 0
      bsr         wr_data
      move.b      #$43,D0     * Commande d'écriture TA
      bsr         wr_comm     * Ecriture commande TH

* Définition Modes

      move.b      #DispMod,D0 * Mise en mode Graphique et/ou Texte
      bsr         wr_comm
      move.b      #ModSet,D0
      bsr         wr_comm     * Mise en mode Graphique ou Texte
      rts


***    lcd_gotoxy  : Init position caractère  ***
*********************************** **** **** ****

lcd_gotoxy
      bsr         wr_data     * Ecrire la donnée LSByte contenue dans D0
      move.b      D1,D0       * MSByte dans dans D0
      bsr         wr_data     * Ecrire la donnée MSByte contenue dans D0
      move.b      #$4C,D0 * Code de commande du Pointeur dans D0
      bsr         wr_comm     * Ecriture de la commande du Pointeur dans D0
      rts


***    lcd_cls  : Effacement écran  ***
*********** ***  ********************
lcd_cls
      clr.b       D0          * Adresse 00,00 dans D0 et D1
      clr.b       D1
      bsr         lcd_gotoxy  * Positionner sur le premier caractère
      move.w      #2048,D3    * Nombre d'octets de la mémoire écran
      clr.b       D0
suite bsr         wr_data
      move.b      #AutoInc,D0 * Code auto-incrémentation dans D0
      bsr         wr_comm     * Incrémenter le pointeur de caractère
      move.b      #0,D0
      sub.w       #1,D3
      bne         suite       * Continuer d'effacer tout l'écran
      rts
```

```
***    lcd_out_str : Envoi chaîne de caractères  ***
*****************************************************

** Attention il faut soustraire la valeur $20 au code ASCII du caractère

lcd_out_str
      move.b      (A0)+,D0    * Pointeur un caractère
      cmp.b       #'$',D0     * Est-ce le caractère de fin ?
      beq         fin         * Fin si le caractère est le $
      sub.b       #$20,D0     * A cause du code ASCII décalé voir notice
      bsr         wr_data
      move.b      #AutoInc,D0
      bsr         wr_comm
      bra         lcd_out_str
fin   rts


***    delay      ***
*********************
delay move.b      #$10,D4
bc1   sub.b       #1,D4
      bne         bc1
      rts

tempo move.l      #$4000,D5
temp1 sub.l       #1,D5
      bne         temp1
      rts

*****************************************
*     Sous programmes Gestion clavier   *
*****************************************

**    Détection d'une touche appuyée en mode pooling **
**    test du regsitre clavier : reg_clavier         **
*******************************************************
Touch
      move.b      #$EF,D1           * Ligne 3
      move.w      #$D2              * N° ligne
col_i move.l      D2,reg_clavier    * Pointer une colonne
      move.b      reg_clavier,D0    * Lire les lignes
      move.l      #$4000,D4
      bsr         tempo

      and.b       #$0F,D0
      cmp.b       #$0F,D0           * Touche appuyée pour cette colonne ?
      beq         balayer           * Pas de touche appuyée pour cette
colonne
      bsr         Lacher            * Aller tester si touche lâchée
      rts

balayer
      rol.b       #1,D1             * Ligne suivante
      sub.w       #1,D2             * n° ligne suivante
      bge         col_i
      bra         Touch
```

```
Lacher                                  * Attendre de relâcher la touche
      move.b      reg_clavier,D3
      bsr         tempo
      and.b       #$0F,D3               * Isoler le quartet Colonne
      cmp.b       #$0F,D3               * Touche non relâchée ?
      bne         Lacher                * Relire les colonnes si touche toujours
appuyée
      rts                               * Sinon retour de sous prog.


**    Calcul de la valeur de la touche appuyée **
************************************************

Val_Touch
      and.l       #$F,D2                * Isoler le n° de la ligne
      move.l      #Colonne0,A0          * Pointer sur la col
      cmp.b       #$0E,D0               * Est-ce la colonne 0 ?
      beq         Val_Colonne0          * Si oui aller lire la valeur
      move.l      #Colonne1,A0          * Pointer sur la colonne 1
      cmp.b       #$0D,D0               * Est-ce la colonne 1 ?
      beq         Val_Colonne1          * Si oui aller lire la valeur
      move.l      #Colonne2,A0          * Pointer sur la colonne
      cmp.b       #$0B,D0               * Est-ce la colonne 2 ?
      beq         Val_Colonne2          * Si oui aller lire la valeur
      move.l      #Colonne3,A0          * Pointer sur la colonne 3
      bra         Val_Colonne3          * Aller lire la valeur

**    Gestion de la position d'affichage **
*****************************************

Val_Colonne0
      move.b      $10(A0,D2),D0         * Position de la touche
      clr.b       D1
      bsr         lcd_gotoxy
      move.b      0(A0,D2),D0
      rts

Val_Colonne1
      move.b      $10(A0,D2),D0         * Position de la touche
      clr.b       D1
      bsr         lcd_gotoxy
      move.b      0(A0,D2),D0
      rts

Val_Colonne2
      move.b      $10(A0,D2),D0         * Position de la touche
      clr.b       D1
      bsr         lcd_gotoxy
      move.b      0(A0,D2),D0
      rts

Val_Colonne3
      move.b      $10(A0,D2),D0         * Position de la touche
      clr.b       D1
      bsr         lcd_gotoxy
      move.b      0(A0,D2),D0
      rts

      end
```

EID005_TP 4 DETECTION D'ACTIVATION D'UNE TOUCHE ET LECTURE EN MODE POLLING

4.1 Enoncé du sujet

Objectifs :	Etre capable d'utiliser les utilitaires rangés en bibliothèque, permettant la gestion d'un clavier matricé 16 points (4x4).
Cahier des charges :	**Sujet** Ecrire un programme en assembleur et en C qui réalisera la lecture d'une touche du clavier matriciel en mode pooling. Cette lecture se fera par le mode pooling.

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,
Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID210000
Carte Clavier Afficheur Horloge Temps réel : EID00500
Câble de liaison USB, ou à défaut câble RS232, Réf : EGD000003
Alimentation AC/DC 8V, 1 A Réf : EGD000001,

Documentations nécessaires :

Document : DM Carte Clavier Afficheur Horloge Temps réel : EID005000
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Durée : 1 séance de 3 heures

4.2 Eléments de solution

4.2.1 Description sommaire du clavier

4.2.1.1 Représentation du clavier 4x4 fig.1

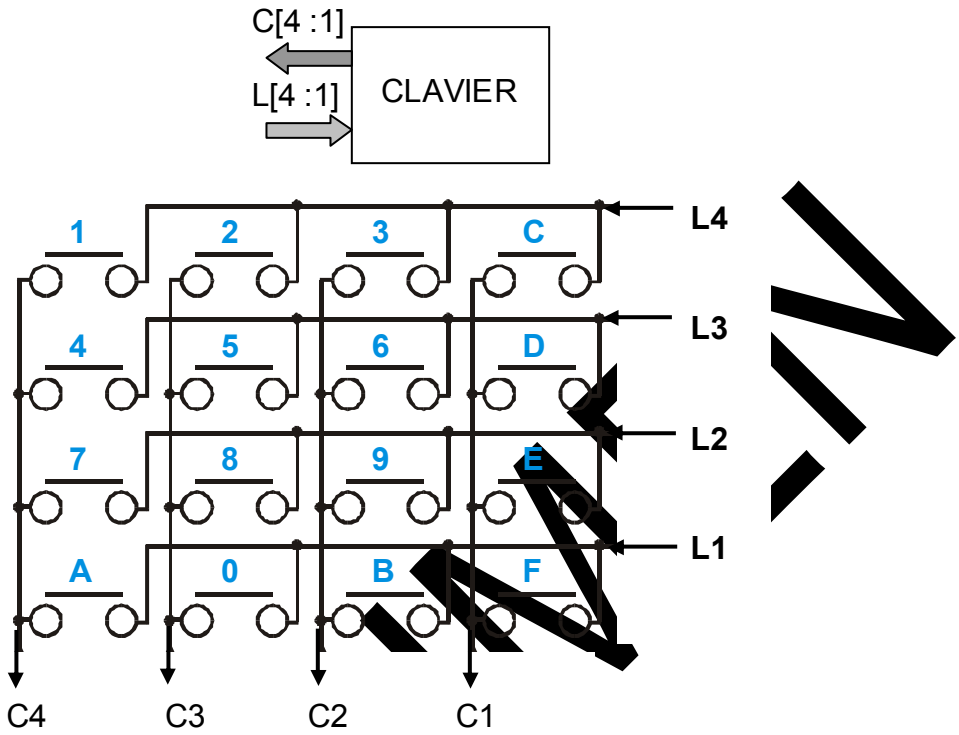


fig.1

Les 4 lignes du clavier sont câblées sur la sortie de l'EPLD de gestion de l'EID005.

Chaque sortie de l'EPLD est munie d'une résistance de pull-up (tirage à Vcc).

Les 4 colonnes sont câblées sur des entrées.

4.2.1.2 Principe général de lecture d'un clavier matricé

Lorsque les sorties commandant les 4 lignes sont au niveau logique **1**, les 4 bits correspondant aux 4 colonnes sont à **1** quelque soit le nombre de touches appuyées.

Le principe de la lecture par la méthode pooling consiste à fixer **1 ligne L_j 0**, **les 3 autres à 1** (balayage lignes) puis à détecter le numéro de la **colonne C_i** à **0** (balayage colonnes).

L'intersection $L_j x C_i$ donne le caractère correspondant.

Il ne reste plus qu'à coder ce caractère pour lui donner la valeur binaire ou hexadécimale voulue.

4.2.1.3 Détection de l'activation d'une touche appuyée

Cette détection se fait à travers le bit ETAT_CLAVIER (bit 0) du registre STATUS de la carte EID005.

Ce bit STATUS_CLAVIER est positionné à 1 lorsqu'au moins une touche est appuyée.

C'est la variable touche définie dans le fichier de définition eid005.h, à la ligne suivante :

```
#define touche    status.r_bit.b0
```

4.2.1.4 Organigramme principal

Le clavier est codé par une matrice 4x4 en hexadécimal comme indiqué ci-dessous.

Attention à la position des lignes dans le tableau de codage par rapport à celle de la matrice elle-même.

Poids de la colonne	8	4	2	1
Ligne / Colonne	C_4	C_3	C_2	C_1
L4	01	02	03	0C
L3	04	05	06	0D
L2	07	08	09	0E
L1	0A	00	0B	0F

Les lignes et les colonnes sont décrites dans le fichier eid005.h de la carte EID005.

La portion correspondante est ci-dessous.

```
/* clavier */
union reg_clavier
{
struct
        {
        unsigned char ligne:4;
        unsigned char colonne:4;
        } matrice;
unsigned char valeur;
};
#define touche       status.r_bit.b0
#define clavier (*(union reg_clavier *) (eid005+5))
```

Une ligne i est définie par la variable :

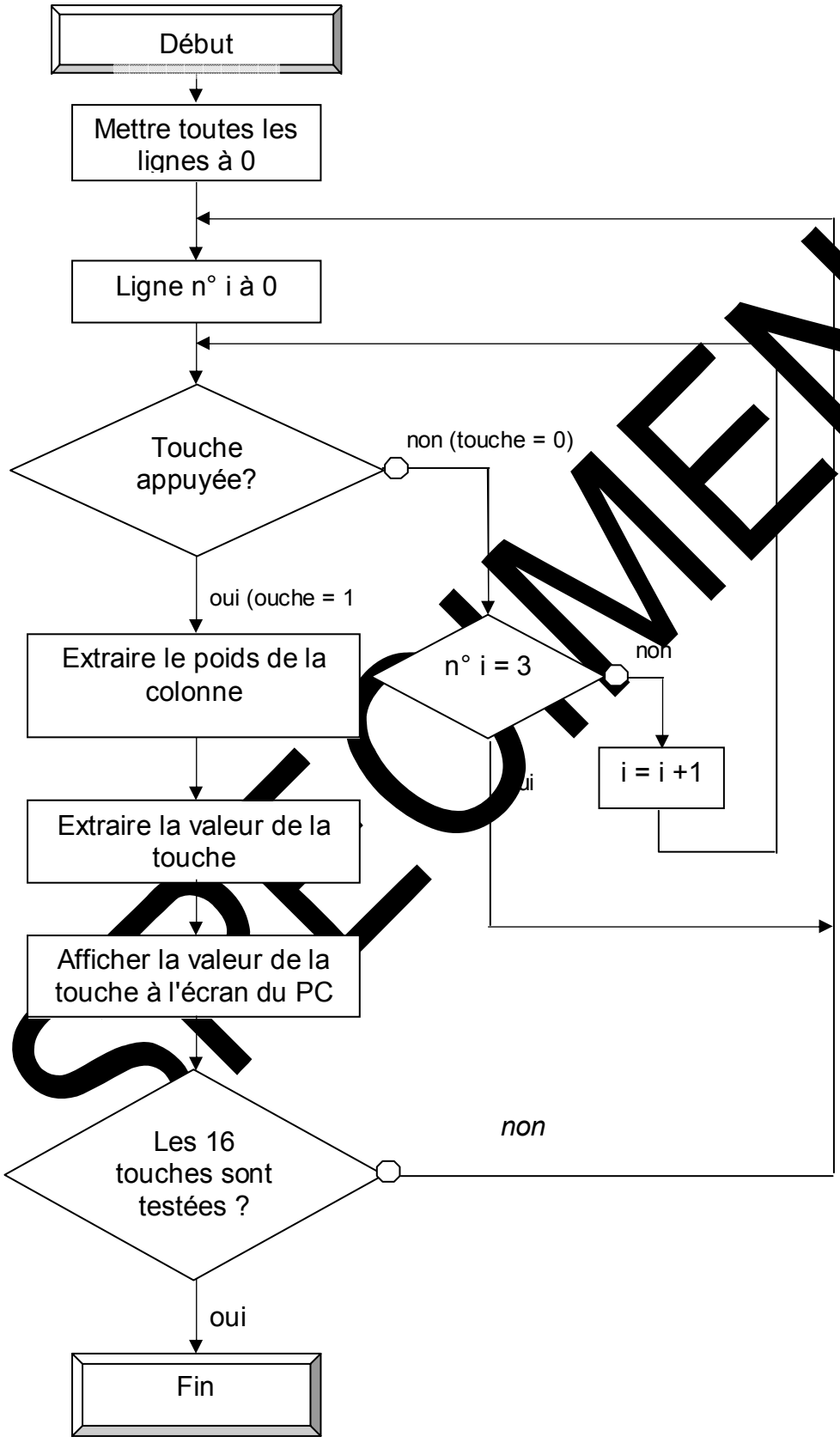
- clavier.matrice.ligne = i ; (0 =< i =< 3)

Les colonnes sont lues à travers la variable :

- clavier.matrice.colonne.

Important :

Pour utiliser le clavier afficheur avec le compilateur C/C++, il faut, dans le menu du logiciel eid210, aller dans le menu configuration puis gnu C/C++.Ensuite aller dans l'onglet *linker*, cliquer sur *ajouter*, sélectionner le fichier EID005_Lib.o et finir en cliquant sur ouvrir.


Début
Mettre toutes les lignes à 0
Ligne n° i à 0
Touche appuyée?
non (touche = 0)
oui (ouche = 1
Extraire le poids de la colonne
n° i = 3
non
i = i +1
ui
Extraire la valeur de la touche
Afficher la valeur de la touche à l'écran du PC
Les 16 touches sont testées ?
non
oui
Fin

4.2.2 Programme en C

```
/*****************************************************
*     TP SUR LA CARTE EID005                         *
*****************************************************
*     Ecrire un programme en C et qui réalise la     *
*     lecture d'un touche sur un clavier matricé 4 x 4*
*     en mode pooling par détection de l'activation d'une touche *
*****************************************************
*     Nom du FICHIER : EID005_TP4.c                  *
*****************************************************/
//    Inclusion des fichiers de définition

#include "eid210.h"
#include "eid005.h"

//============================
//    FONCTION PRINCIPALE
//============================

main ()
{
short    Touche[4][4] = { 0x0A, 0x00, 0x0B, 0x0E,
                          0x07, 0x08, 0x09, 0x0F,
                          0x04, 0x05, 0x06, 0x0D,
                          0x01, 0x02, 0x03, 0x0C   ;
short N, V, ValeurTouche;
int tmp ;
int i, j, k;

// Préparation pour affichage sur LCD

init_aff();             // Initialisation de l'afficheur
lcd_cls();              // Effacement de l'afficheur
lcd_gotoxy(0x30,00)     // Définition de la position de départ de la chaîne
de caractères

//    Balayage des lignes du clavier en donnant successivement à la
//variable
//    clavier.matrice.ligne les quartets : 1110, 1101, 1011, 0111 .

printf ("----------------------\n");
printf ("TESTER TOUTES LES TOUCHES \n");
printf ("---------------------- \n\n");

j=0;
do
{j++;
 printf ("Appuyer sur une touche \n");
//    Touche appuyée ?
//    Balayage ligne
while (!touche)               // Attente appui touche
{
      for (i = 0; i<4; i++ )
      {
            clavier.matrice.ligne = ~(1 << i); // Toutes les colonnes à 1
                                               //sauf la n° i
tmp = i ;
```

```
if (touche == 1) break;       // touche = 1 ==> au moins une touche appuyée

for (k = 0 ; k<10000 ; k++);        // temporisation
      }
}
/*************************************************************************
*     Extraire le poids de la colonne :
*     clavier.matrice.colonne = Elément colonne (4 bits) de la structure
*     appartenant à la variable clavier de l'union reg_clavier.
*     Puis extraire la valeur de la touche appuyée donnée par le poids de
*     la colonne :
*     Ce poids est donnée par la variable : clavier.matrice.colonne & 0xF
*************************************************************************/
switch((~(clavier.matrice.colonne)) & 0x0F )
{
      case 1 : ValeurTouche= Touche [tmp][3]; break; //Colonne poids 1
      case 2 : ValeurTouche= Touche [tmp][2]; break; //Colonne poids 2
      case 4 : ValeurTouche= Touche [tmp][1]; break; //Colonne poids 4
      case 8 : ValeurTouche= Touche [tmp][0]; break; //Colonne poids 8
}

//    Afficher loa valeur de la touche à l'écran du PC

printf ("La touche appuyee est : %x\n\n", ValeurTouche);

// Afficher loa valeur de la touche sur l'écran LCD

if(ValeurTouche < 0x0A) // Conversion hexa --> ASCII : chiffres
N= ValeurTouche+0x10;
else
N= ValeurTouche+0x17;   // Conversion hexa --> ASCII : lettres
lcd_write_data(N);      // Envoi du caractère au LCD
lcd_write_command(0xC0);      // Incrémentation pour la position du
caractère suivant

while (touche)                  // Attente relâche touche
for (k = 0 ; k<1000 ; k++);  // Temporisation
}
//    Toutes les touches sont testées ?
while (j<16)
}
```

4.2.3 Programme en Assembleur

```
*******************************************************
*     TP SUR LA CARTE EID005                          *
*******************************************************
*     Ecrire un programme en Assembleur               *
*     et en C et qui réalise la lecture d'un touche   *
*     sur un clavier matricé 4 x 4, en mode pooling   *
*******************************************************
*     Nom du FICHIER : EID005_TP4.src                 *
*     **************                                  *
*******************************************************


**    La commande ou la donnée à écrire sont d'abord rangées dans D0
**    La donnée lue est rangée dans D0
**    La posision x,y d'un caractère est placée respectivement dans D0 et
D1
**    L'adresse de début d'une chaîne de caractère est placée dans A0

*************************************************************
*     Bits du registre de contrôle de l'afficheur           *
*     ctrlaff :   b0=rd                                     *
*                 b1=wr                                     *
*                 b2=ce                                     *
*                 b3=cd                                     *
*                 b4=fs                                     *
*************************************************************

* Inclusion du fichier définissant les différents labels
* de l'EID210

      include EID210_def

      section code


*     Définition des adresses physiques de l'afficheur et du clavier

eid005      equ   $B30000     * Adresse de base Carte clavier_afficheur_rtc
ctrlaff     equ   eid005+3    * Registre contrôle afficheur : bus de
                              * contrôle
dbaff       equ   eid005+4    * Bus de données afficheur
status      equ   eid005+6    * Registre de status de la carte
reg_ctrl    equ   eid005+7    * Registre de contrôle de la carte
reg_clavier equ   EID005+5

*     Table de définition des paramètres de l'afficheur l'afficheur


TabDef      dc.b  $00,$04,$42       * GH
            dc.b  $10,$00,$43       * GA
            dc.b  $00,$00,$40       * TH
            dc.b  $10,$00,$41       * TA
ModSet      equ   $80               * OR  Graphique ou Texte
Pointeur    equ   $24               * Commande pointeur
DispMod     equ   $94               * Affichage Texte et ou Graphique
```

```
AutoInc     equ   $C0              * Auto incrémentation pixel ou caractère

Pile        equ   $802000          * Adresse de sauvegarder des contextes

init_acces  equ   $EF   * Pour accéder au bus de données du lcd avec fs=0
wrc         equ   $E9   * cd=1, ce=0, wr=0, rd=1, fs=0 1110 1001  write
                        * commande
rdc         equ   $EA   * cd=1, ce=0, wr=1, rd=0, fs=0 1110 1010  read
                        * commande (Status)
wrd         equ   $E1   * cd=0, ce=0, wr=0, rd=1, fs=0 1110 0001  write data
rdd         equ   $E2   * cd=0, ce=0, wr=1, rd=0, fs=0 1110 0010  reda data

Texte1      dc.b  'EID----------005 $ '
Texte2      dc.b  '?!$'
Texte3      dc.b  'FIN $4


**    Tableau des valeurs des touches du clavier

Colonne0          dc.b  $0C,$0D,$0E,$0F
Colonne1          dc.b  $03,$06,$09,$0B
Colonne2          dc.b  $02,$05,$08,$00
Colonne3          dc.b  $01,$04,$07,$0A

**    Tableau des coordonnées d'affichage des touches du clavier

                  dc.b  $1A,$3A,$5A,$7A
                  dc.b  $18,$38,$58,$78
                  dc.b  $16,$36,$56,$7
                  dc.b  $14,$3     ,$74

      section           code

*************************
*     PROGRAMME PRINCIPALE    *
*************************

      bsr         init_
      bsr         lcd_c      * Effacement écran

* Envoi Texte1 : x=0, y=0
      move.b      #$00,D0     * LSByte TH : ligne 0 colonne 1
      move.b      D1          * MSByte TH
      bsr         lcd_gotoxy
      move.l      #Texte1,A0
      bsr         lcd_out_str

*************************************

* Définition début affichage touches clavier

      move.l      #15,D7
      clr.b       D1
```

```
* Lecture du clavier

test_clav
      bsr         Touch
      bsr         Val_Touch   * La valeur de la touche est dans D0
      cmp.b       #$0A,D0     * Conversion Hexa --> ASCII décalé de $20,
      bcc         sup10       * voir Générateur de caractères T6963C :
      add.b       #$10,D0     * add $10 si code < $0A
      bra         envoi       *
sup10 add.b       #$17,D0     * si non add $17
envoi bsr         wr_data
      move.b      #$C0,D0
      bsr         wr_comm
      dbeq        D7,test_clav

* Envoi Texte2 : x=7, y=0

      move.b      #$70,D0     * LSByte TH : ligne 7 colonne 0
      clr.b       D1          * MSByte TH
      bsr         lcd_gotoxy
      move.l      #Texte2,A0
      bsr         lcd_out_str

* Envoi Texte3 : x=7, y=d

      move.b      #$7d,D0     * LSByte TH : ligne 7 colonne 13
      clr.b       D1          * MSByte TH
      bsr         lcd_gotoxy
      move.l      #Texte3,A0
      bsr         lcd_out_str

*=============================================================*
      JMP         MONITEUR          Retour au Moniteur
*=============================================================*

*********************************
*     FIN PROGRAMME PRINCIPAL   *
*********************************

*********************************
*     LES SOUS PROGRAMMES       *
*********************************

**    SOUS PROGRAMMES GESTION DE L'AFFICHEUR    **
*****************************************************

***    Busy **
**********

busy  move.b      #init_acces,ctrlaff
      move.b      #rdc,ctrlaff
      bsr         delay
      move.b      dbaff,D4          * Lecture data bus aff
      and.b       #03,D4            * Isoler ST0 STA1 (Status)
      cmp.b       #03,D4            * Si lcd pas prêt
      bne         busy              * attendre
      move.b      #init_acces,ctrlaff
      rts
```

```
***    wrcomm  : Ecriture commande placée dans D0 ***
*****************************************************

wr_comm
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Déposer la commande sur le bus
                                          * de donnée
      move.b      #wrc,ctrlaff            * Générer les impulsions
                                          * d'écriture de
                                          * d'une commande
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    wrdata  :  Ecriture data placée dans D0 ***
*************************************************

wr_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Déposer la data sur le bus de
                                          * donnée
      move.b      #wrd,ctrlaff            * Générer les impulsions
                                          * d'écriture de
                                          * d'une donnée
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    rddata  :  Lecture data à placer dans D0 ***
*************************************************

rd_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      #rdc,ctrlaff            * Générer les impulsions
                                          * d'écriture de
                                          * d'une donnée
      bsr         delay
      move.b      dbaff,D0                * Lire et mettre la donnée dans D0
      move.b      #init_acces,ctrlaff
      rts
```

```
***    init_aff  : Initialisation afficheur    ***
*************************************************

init_aff

* Initialisation de la zone Adresse de début du Texte

      clr.b       D0
      bsr         wr_data     * LSByte TH = 00
      bsr         wr_data     * MSByte TH = 00
      move.b      #$40,D0     * Commande d'écriture TH
      bsr         wr_comm     * Ecriture TH
      move.b      #$10,D0     * LSByte TA = $10 nombre de caractères/ligne
      bsr         wr_data
      clr.b       D0          * MSByte TA = 0
      bsr         wr_data
      move.b      #$41,D0     * Commande d'écriture TA
      bsr         wr_comm     * Ecriture  TA

* Initialisation de la zone Adresse de début du Graphique

      clr.b       D0
      bsr         wr_data     * LSByte GH = 00
      move.b      #04,D0      * MSByte GH = 00
      bsr         wr_data     * Ecriture GH
      move.b      #$42,D0     * Commande d'écriture TH
      bsr         wr_comm
      move.b      #$10,D0     * LSByte GA = 10 nombre de caractères/ligne
      bsr         wr_data
      clr.b       D0          * MSByte GA = 0
      bsr         wr_data
      move.b      #$43,D0     * Commande d'écriture TA
      bsr         wr_comm     * Ecriture commande TH

* Définition Modes

      move.b      #Dis_Mod,D0 * Mise en mode Graphique et/ou Texte
      bsr         wr_data
      move.b      #ModSet,D0
      bsr         wr_comm     * Mise en mode Graphique OU Texte
      rts


***    lcd_gotoxy  : Init position caractère   ***
*************************************************

lcd_gotoxy
      bsr         wr_data     * Ecrire la donnée LSByte contenue dans D0
      move.b      D1,D0       * MSByte dans dans D0
      bsr         wr_data     * Ecrire la donnée MSByte contenue dans D0
      move.b      #Pointeur,D0 * Code de commande du Pointeur dans D0
      bsr         wr_comm     * Ecriture de la commande du Pointeur dans D0
      rts
```

```
***    lcd_cls  : Effacement écran  ***
**************************************

lcd_cls
      clr.b       D0          * Adresse 00,00 dans D0 et D1
      clr.b       D1
      bsr         lcd_gotoxy  * Positionner sur le premier caractère
      move.w      #2048,D3    * Nombre d'octets de la mémoire écran
      clr.b       D0
suite bsr         wr_data
      move.b      #AutoInc,D0 * Code auto incrémentation dans D0
      bsr         wr_comm     * Incrémenter le pointeur de caractère
      move.b      #0,D0
      sub.w       #1,D3
      bne         suite       * Continuer d'effacer tout l'écran
      rts


***    lcd_out_str : Envoi chaîne de caractères     *
*****************************************************

** Attention il faut soustraire la valeur $20 au code ASCII du caractère
** lcd_out_str

      move.b      (A0)+,D0    * Pointe le caractère
      cmp.b       #'$',D0     * Est-ce le caractère de fin ?
      beq         fin         * Fin si le caractère est le $
      sub.b       #$20,D0     * A cause du code ASCII décalé voir notice
      bsr         wr_data
      move.b      #AutoInc,D0
      bsr         wr_comm
      bra         lcd_out_str
fin   rts


***    delay      *
*********************
delay move.b      #10,D4
bc1   sub.b       #1,D4
      bne         bc1
      rts

temp  move.l      #$4000,D5
temp1 sub.l       #1,D5
      bne         temp1
      rts
```

```
*****************************************
*     Sous programmes Gestion clavier    *
*****************************************

**    Détection d'activation une touche en mode pooling    **
**    par test du registre status de l'EID005              **
************************************************************

Touch
      move.b      #$EF,D1           * Ligne 3
      move.w      #3,D2             * N° ligne
col_i move.b      D1,reg_clavier    * Pointer une colonne
      move.b      reg_clavier,D0    * Lire les lignes
      and.b       #$0F,D0           * Isoler le quartet colonne
      bsr         tempo

      btst.b      #0,status         * Test bit 0 du registre d'état de la
carte : status
                                    * Touche appuyée pour cette colonne ?
      beq         balayer           * Si aucune touche appuyée pour cette
colonne
                                    * aller à la colonne suivante
      bsr         Lacher            * Si oui attendre tester si la touche est
relachée
      rts

balayer
      rol.b       #1,D1             * Ligne suivante
      sub.w       #1,D2             * n° ligne suivante
      bge         col_i
      bra         Touch

Lacher                              * Attendre de relâcher la touche
      move.b      reg_clavier,D0
      bsr         tempo
      and.b       #$0F,D3           * Isoler le quartet Colonne
      cmp.b       #$0F,D3           * Touche non relâchée ?
      bne         Lacher            * Relire les colonnes si touche toujours
appuyée
      rts                           * Sinon retour de sous prog.

**    Calcul de la valeur de la touche appuyée  **
************************************************

Val_
      and.l       #$F,D2            * Isoler le n° de la ligne
      move.l      #Colonne0,A0      * Pointer sur la colonne 0
      cmp.b       #$0E,D0           * Est-ce la colonne 0 ?
      beq         Val_Colonne0      * Si oui aller lire la valeur
      move.l      #Colonne1,A0      * Pointer sur la colonne 1
      cmp.b       #$0D,D0           * Est-ce la colonne 1 ?
      beq         Val_Colonne1      * Si oui aller lire la valeur
      move.l      #Colonne2,A0      * Pointer sur la colonne 2
      cmp.b       #$0B,D0           * Est-ce la colonne 2 ?
      beq         Val_Colonne2      * Si oui aller lire la valeur
      move.l      #Colonne3,A0      * Pointer sur la colonne 2
      bra         Val_Colonne3      * Aller lire la valeur
```

```
**    Gestion de la position d'affichage  **
*********************************************

Val_Colonne0
      move.b      $10(A0,D2),D0           * Position de la touche
      clr.b       D1
      bsr         lcd_gotoxy
      move.b      0(A0,D2),D0
      rts

Val_Colonne1
      move.b      $10(A0,D2),D0           * Position de la touche
      clr.b       D1
      bsr         lcd_gotoxy
      move.b      0(A0,D2),D0
      rts

Val_Colonne2
      move.b      $10(A0,D2),D0           * Position de la touche
      clr.b       D1
      bsr         lcd_gotoxy
      move.b      0(A0,D2),D0
      rts

Val_Colonne3
      move.b      $10(A0,D2),D0           * Position de la touche
      clr.b       D1
      bsr         lcd_gotoxy
      move.b      0(A0,D2),D0
      rts

      end
```

EID005_TP 5 DESSIN DE LIGNES, CERCLES ET COURBES SUR LE LCD

5.1 Enoncé du sujet

Objectifs :	Etre capable d'utiliser les utilitaires rangés en bibliothèque, permettant le tracé de droites et de cercles sur l'afficheur LCD.
Cahier des charges :	**Sujet** Ecrire un programme en C qui réalise le tracé de droites verticale, horizontale, oblique et de cercles, …

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,
Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID210000
Carte Clavier Afficheur Horloge Temps réel : EID00500
Câble de liaison USB, ou à défaut câble RS232, Réf : EGD000003
Alimentation AC/AC 12V, 1 A Réf : EGD000001,

Documentations nécessaires :

Document : DMS Carte Clavier Afficheur Horloge Temps réel : EID005000
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Durée : 1 séance de 4 heures

5.2 Eléments de solution

5.2.1 Description de l'afficheur en mode graphique

Attention :

Pour des raisons de conformité avec la documentation du constructeur, les variables x et y représentent respectivement l'ordonnée (axe vertical) et y représente l'abscisse (axe horizontal).
Le point **x = 0, y = 0** est en haut à gauche et le point **x = 63, y = 127** en bas à droite de l'écran du LCD

Le contrôleur T6963C dispose d'une mémoire de 8 ko.

Mode graphique

La zone graphique est située à partir de l'adresse 0004, offset à ajouter à l'octet de poids fort du paramètre GH.

Le LCD en mode graphique contient 64 lignes et 128 points par ligne,.
Ce qui donne : 128x64 = 8192 points soit des numéros allant de 0 à 8191.

Un point est défini en premier lieu par le numéro de l'octet dans lequel il est situé.

Les 8192 points sont donc définis dans 1024 octets (8192/8).

Un pixel de cordonnées x, y a pour Numéro Nu = 128*x + y.

Son adresse Adr dans la mémoire écran (VRAM) est déterminée par le nombre entier d'octets contenus dans son numéro Nu ; sa position sera entièrement définie par la connaissance de numéro du point dans l'octet.

- **Adr = Partie entière (Nu / 8) ; Adr est sur 2 octets.**

Soit r le reste de la division entière de Nu par 8.

Dans l'octet, un pixel est repéré par son numéro codé sur 3 bits.
- Le bit de poids faible est allumé à droite de l'octet sur la ligne ; c'est donc le 8ème pixel dans l'octet et son numéro est 7.
- Le bit de poids fort est allumé à gauche de l'octet sur la ligne ; c'est donc le 1er pixel dans l'octet et son numéro est 0.

Le numéro du point à allumer est donc np = 7- r.

pixel n°7							pixel n°0
np = 0							np = 7
B7	B6	B5	B4	B3	B2	B1	B0

Exemple fig.1

Soit le pixel situé en : **x = 18, y = 91**
Le numéro du point est Nu = 18x128 + 91 = 2395 ; Nu / 8 = 299,375
Adr = 299 = 0x012B
r = Nu – 299*8 = 4
np = 7 – 4 = 3
Ce qui donne les octets du paramètre GH (Graphic Home Adress) :
GH Adress lower = 0x2B **GH Adress upper = 0x01 + 0x04.**

Les paramètres **GH Adress lower**, **GH Adress upper** **et** **np** sont calculés chaque fois qu'un pixel doit être allumé ou éteint.

POSITION GRAPHIQUE D'UN POINT DANS LE PLAN EN MODE GRAPHIQUE

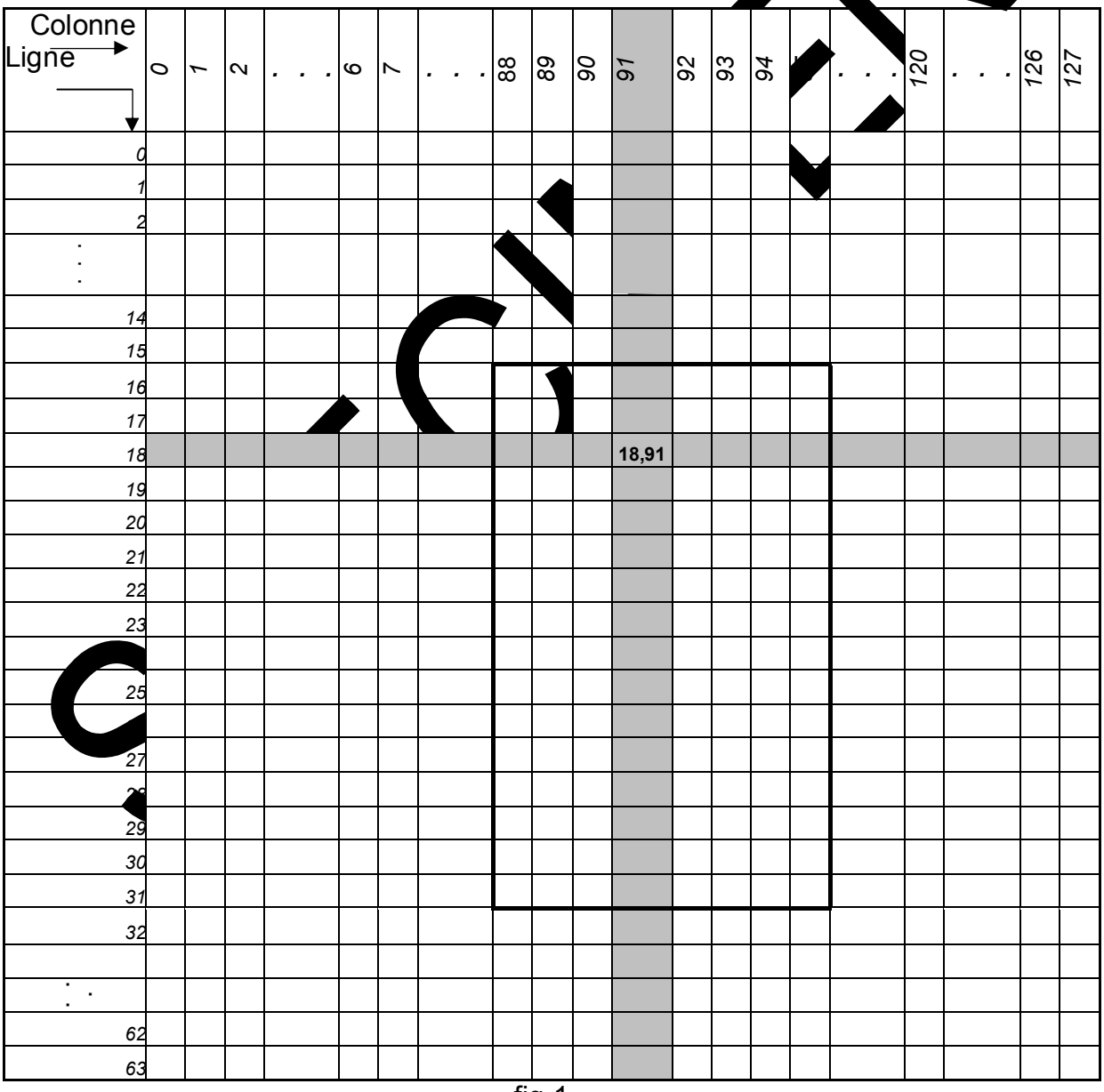


fig.1

5.2.2 Programme principal

Vous disposez de l'utilitaire **void Tracer_Pixel (int x, int y, unsigned char Pen)** pour tracer des courbes en élaborant votre propre algorithme ou les utilitaires suivants avec leurs commentaires pour aboutir au même résultat.

Les commandes 0xF8 et 0xF0 permettent d'allumer ou d'éteindre le pixel de numéro np.
Cela donne la définition suivante de la variable Pen :

Pen = 0xF8 + np ➔ allumage du pixel de numéro np.
Pen = 0xF0 + np ➔ extinction du pixel de numéro np..

Utilitaires	Commentaires
void init_aff()	Initialisation des paramètres TH, TA, GH, GA, Mode
void lcd_cls()	Effacement de l'écran
void lcd_write_command(unsigned char **commande**)	Ecriture d'une commande
void lcd_write_data(unsigned char **data**)	Ecriture d'une donnée
void lcd_clear_TXT()	Effacer l'écran texte
void lcd_clear_Graph()	Effacer l'écran graphique
void lcd_gotoxy(unsigned char **px**, unsigned char **py**)	Définition de la position d'un caractère ou d'un pixel ; px et py sont les données lower et upper. Remarque : pour un pixel il faut ajouter 0x04 à py.
void lcd_out_str(char * texte)	Envoi d'une chaîne de caractères pontée par la variable * texte
void Tracer_Pixel(int x, int y, unsigned char Pen)	Pen = 0xF8 ➔ Allumer le pixel Pen = 0xF0 ➔ Eteindre le pixel
void Tracer_LH(unsigned char M, unsigned char N, unsigned char P, unsigned char Q, unsigned char Pen)	M, N : coordonnées du point de départ P,Q : coordonnées du point de départ
void Tracer_LV (unsigned char M, unsigned char N, unsigned char P, unsigned char Q, unsigned char Pen)	idem
void Droite (int x1, int y1, int x2, int y2, int Pen)	idem

Utilitaires	Commentaires

`void Cercle_NPoints ( int x0, int y0, int r, unsigned int Pen, unsigned int N)`	`x0, y0 : coordonnées du centre`
	`r : rayon (en nombre de points)`
	`N : nombre de points sur le cercle`
	`Pen = 0xF8 ➔ Allumer le pixel`
	`Pen = 0xF0 ➔ Eteindre le pixel`

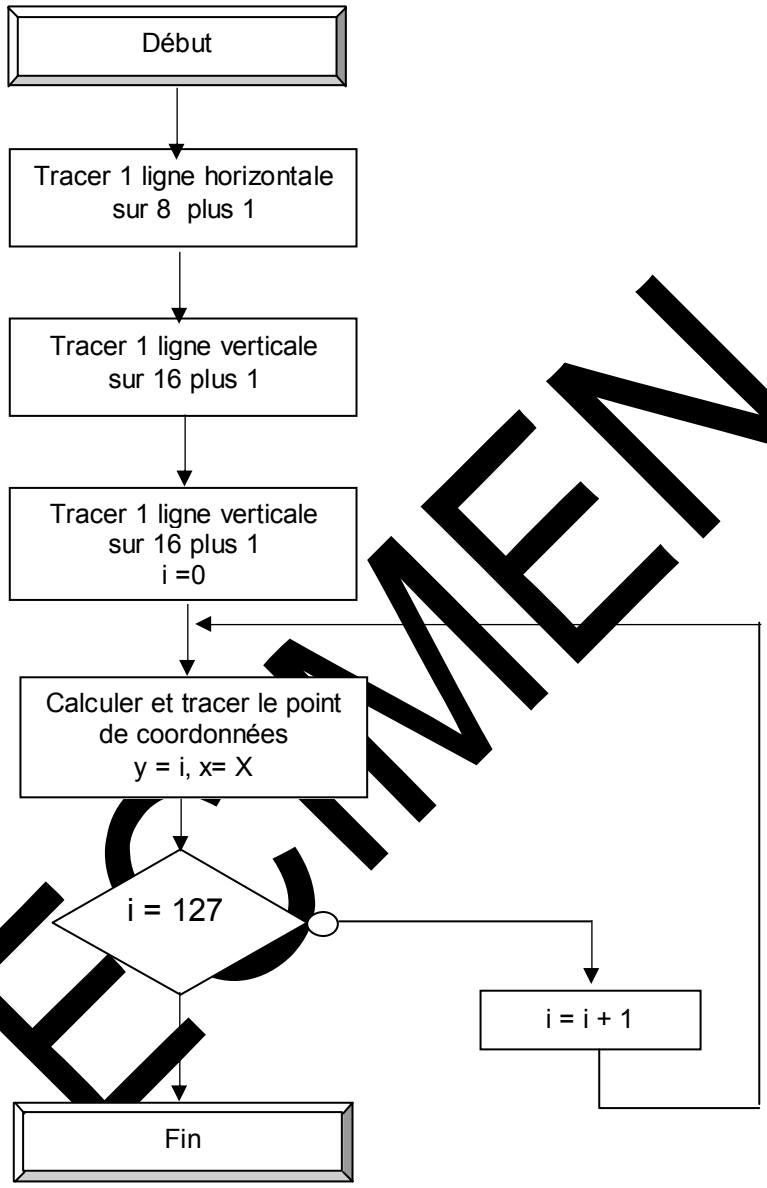
5.2.3 Organigramme du tracé d'une sinusoïde sur un oscilloscope

Tracé par point : Organigramme 1

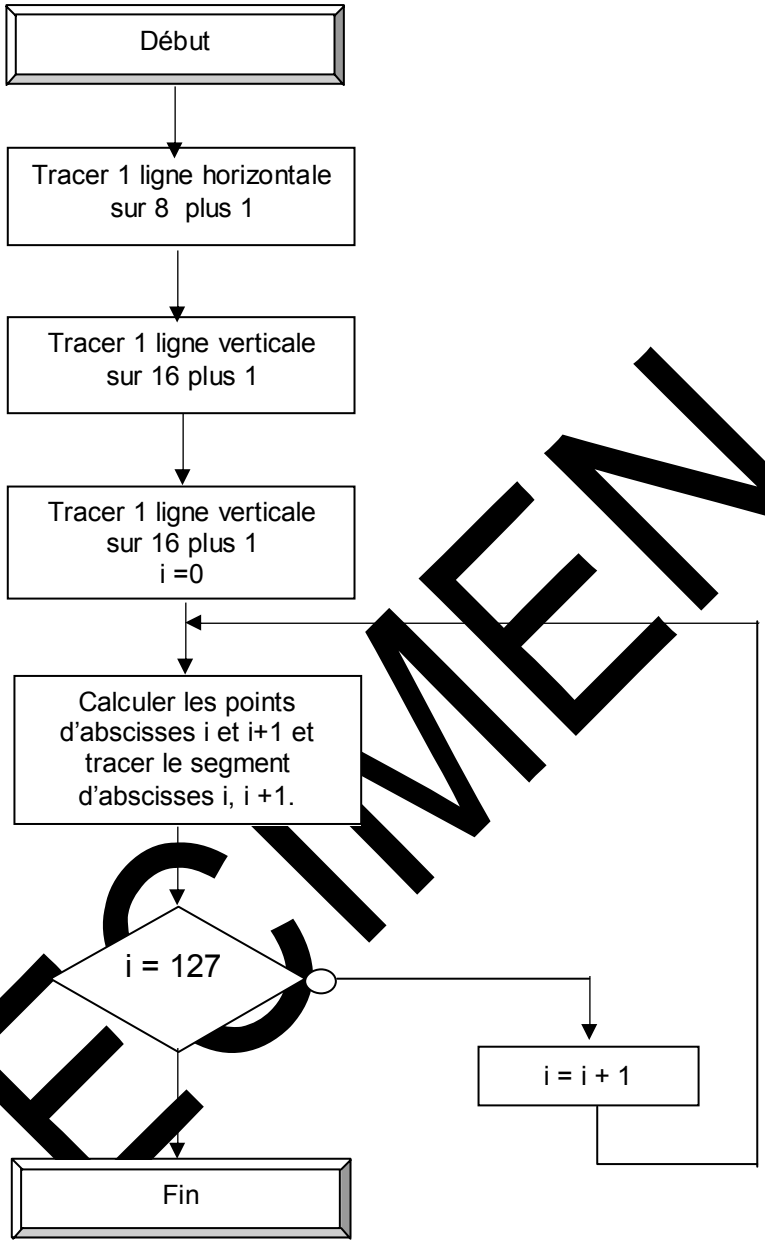
L'angle élémentaire est : pi/N avec (exemple N = 30)
Le principe est de calculer X = k*sin (i*pi/N) où k représente en nombre de points, l'amplitude maximale de la sinusoïde.
Le point à tracer a pour cordonnées : y = i et x = partie entière de X.
abscisse : y ordonnée : x.

Tracé par segments de droite : Organigramme 2

La méthode est la même à ceci près qu'il faut calculer les coordonnées de deux points consécutifs et tracer le segment de droite reliant les deux points.



Organigramme 1



Organigramme 2

5.2.4 Programme en " C " :

```
/*****************************************************
*     TP SUR LA CARTE EID005                         *
*****************************************************
*     Tracé de lignes de :                           *
*     horizontale, verticale, oblique, de cercles ... *
*                                                    *
*****************************************************
*     Nom du FICHIER : EID005_TP5.c                  *
*     **************                                 *
*****************************************************/
```

```
//    Inclusion des fichiers de définition

#include "eid005.h"
#include <stdio.h>
#include <string.h>
#include <math.h>


//============================
//    FONCTION PRINCIPALE
//============================

main()
{
double  pi = 3.14159265 ;
int i, N, m1, m2, k=28; //    k : amplitude maximale de 28*sin(i*pi/N)

int PB = 0xF8 ;   //    Plume baissée : allumer un pixel
int PL = 0xF0 ;   //    Plume levée : éteindre le pixel


init_aff();       //    Initialisation de l'afficheur
lcd_cls();        //    Effacement de l'afficheur
Tab_Sin_Cos ();   //    Calcul sinus et cosinus sur 360° par pas de 1°
                  //    Pour le tracer des cercles

/*****************************************************************
*     Exemple : Tracé d'un rectangle au contour de l'afficheur, de ces  *
*     diagonales, et des deux axes de symétrie horizontale et verticale
      *
/*****************************************************************/

Tracer_LV(0,0,63,0,PB);
Tracer_LV(0,127,63,127,PB);
Tracer_LH(0,0,0,127,PB);
Tracer_LH(63,0,63,127,PB);

Droite(0,0,63,127,PB);
Droite(63,0,0,127,PB);
Cercle_NPoints(32,64,28,PB);

delay (500000);
lcd_clear_Graph();                  // Effacer graphique

Rectangle(0,20,40,90,PB);
Triangle(15,25,35,50,25,100,PB);
delay (500000);
lcd_clear_Graph();
```

```
// Tracer d'une sinusoïde par POINTS sur un oscilloscope en utilisant la
fonction
// void Tracer_Pixel(int x, int y, unsigned char Pen)

for (i=0; i<64; i +=8 )             // Grille 8 lignes verticales
      Tracer_LH(i,0,i,127,PB);
      Tracer_LH(63,0,63,127,PB);
for (i=0; i<127; i +=16 )           // Grille 8 lignes horizontales
      Tracer_LV(0,i,63,i,PB);
      Tracer_LV(0,127,63,127,PB);

for(i=0; i<126; i++)
```

```
{
N = 30;
m1 = k*sin(i*(pi/N));
Tracer_Pixel(32-m1,i,PB);     // Tracer par POINTS
}

delay (500000);
lcd_clear_Graph();            // Effacer graphique


// Tracer d'une sinusoïde par SEGMENTS sur un oscilloscope en utilisant la
//fonction :
// void Droite ( int x1, int y1,  int x2, int y2, int  Pen )

for (i=0; i<64; i +=8 )             // Grille 8 lignes verticales
      Tracer_LH(i,0,i,127,PB);
      Tracer_LH(63,0,63,127,PB);
for (i=0; i<127; i +=16 )           // Grille 8 lignes horizontales
      Tracer_LV(0,i,63,i,PB);
      Tracer_LV(0,127,63,127,PB);

for(i=0; i<126; i++)
      {
      m1 = k*sin(i*(pi/30));
      m2 =k*sin((i+1)*(pi/30));
      Droite(32-m1 , i,32-m2, i+1, PB);
}

}

//    Fin du programme
```

EID005_TP 6 DESSIN D'UNE HORLOGE SUR L'ECRAN GRAPHIQUE

Objectifs :	Etre capable d'utiliser les utilitaires rangés en bibliothèque, permettant la gestion de l'afficheur LCD graphique 128x64 pixels.
Cahier des charges :	**Sujet** Ecrire un programme en assembleur et en C qui réalisera le dessin d'une horloge et l'affichage de l'heure, la date et l'année sur le LCD.

Matériel nécessaire :

Micro ordinateur type PC sous Windows 95 ou ultérieur,
Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID210000
Carte Clavier Afficheur Horloge Temps réel : EID00500
Câble de liaison USB, ou à défaut câble RS232, Réf : EGD000003
Alimentation AC/DC 8V, 1 A Réf : EGD000001,

Documentations nécessaires :

Document PDF Carte Clavier Afficheur Horloge Temps réel : EID005000
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)
Real Time Clock DS14285 (DALLAS Semiconductor)

Durée : 1 séance de 4 heures

6.1 Eléments de solution

6.1.1 Définition géométrique de l'horloge

Les 12 repères représentant les 12 heures sont dessinées sur un cercle de rayon

R.

Chaque repère est en réalité formé d'un carré dont le centre est sur le cercle de rayon R et de coté égal à 4 pixels.

Le cercle des secondes est identique à celui des heures. Son rayon est de R' et les points sont des carrés de coté 2 pixels, centrés sur le cercle.

Pour la gestion du LCD, il faut r= se reporter au TP .

6.1.1.1 Représentation du circuit

Le modèle du programmeur du circuit DS14285 est décrit par la figure 1.

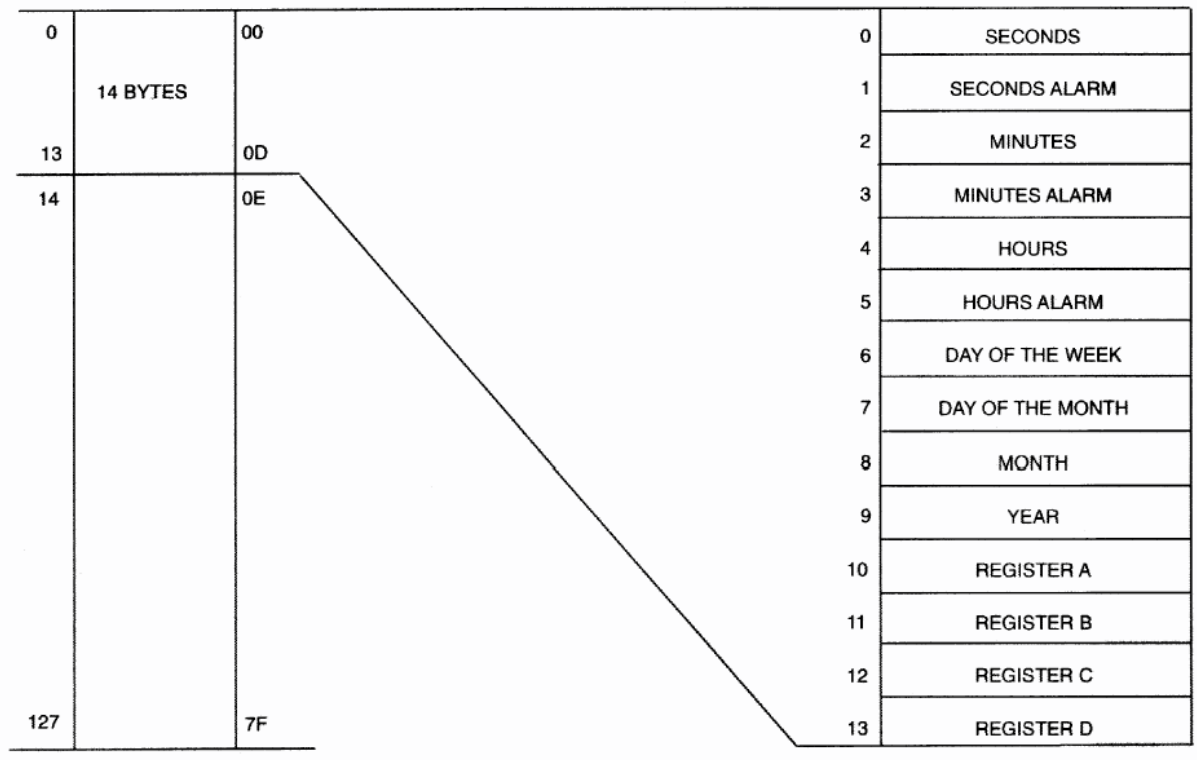


fig.1

Les adresses de 0 à 13 définissent l'emplacement des différents registres du circuit en programmation ou en lecture.

6.1.1.2 Gestion du circuit RTC DS14285

Dans ce TP, seul les deux registres de contrôle A et B du circuit seront programmés.

CONTROL REGISTERS

The DS14285/DS14287 has four control registers which are accessible at all times, even during the update cycle.

REGISTER A

MSB **LSB**

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0

UIP - The Update In Progress (UIP) bit is a status flag that can be monitored. When the UIP bit is a 1, the update transfer will soon occur. When UIP is a 0, the update transfer will not occur for at least 244 µs. The time, calendar, and alarm information in RAM is fully available for access when the UIP bit is 0. The UIP bit is read-only and is not affected by $\overline{\text{RESET}}$. Writing the SET bit in Register B to a 1 inhibits any update transfer and clears the UIP status bit.

DV0, DV1, DV2 - These 3 bits are used to turn the oscillator on or off and to reset the countdown chain. A pattern of 010 is the only combination of bits that will turn the oscillator on and allow the RTC to keep time. A pattern of 11X will enable the oscillator but holds the countdown chain in reset. The next update will occur at 500 ms after a pattern of 010 is written to DV0, DV1, and DV2.

RS3, RS2, RS1, RS0 - These four rate-selection bits select one of the 13 taps on the 15-stage divider or disable the divider output. The tap selected can be used to generate an output square wave (SQW pin) and/or a periodic interrupt. The user can do one of the following:

1. Enable the interrupt with the PIE bit;
2. Enable the SQW output pin with the SQWE bit;
3. Enable both at the same time and the same rate; or
4. Enable neither.

Table 2 lists the periodic interrupt rates and the square wave frequencies that can be chosen with the RS bits. These 4 read/write bits are not affected by $\overline{\text{RESET}}$.

REGISTER B

MSB **LSB**

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE

SET - When the SET bit is a 0, the update transfer functions normally by advancing the counts once per second. When the SET bit is written to a 1, any update transfer is inhibited and the program can initialize the time and calendar bytes without an update occurring in the midst of initializing. Read cycles can be executed in a similar manner. SET is a read/write bit that is not modified by $\overline{\text{RESET}}$ or internal functions of the DS14285/DS14287.

PIE - The periodic interrupt enable PIE bit is a read/write bit which allows the Periodic Interrupt Flag (PF) bit in Register C to drive the $\overline{\text{IRQ}}$ pin low. When the PIE bit is set to 1, periodic interrupts are generated by driving the $\overline{\text{IRQ}}$ pin low at a rate specified by the RS3-RS0 bits of Register A. A 0 in the PIE bit blocks the $\overline{\text{IRQ}}$ output from being driven by a periodic interrupt, but the Periodic Flag (PF) bit is still set at the periodic rate. PIE is not modified by any internal DS14285/DS14287 functions, but is cleared to 0 on $\overline{\text{RESET}}$.

AIE - The Alarm Interrupt Enable (AIE) bit is a read/write bit which, when set to a 1, permits the Alarm Flag (AF) bit in register C to assert $\overline{\text{IRQ}}$. An alarm interrupt occurs for each second that the 3 time bytes equal the 3 alarm bytes including a "don't care" alarm code of binary 11XXXXXX. When the AIE bit is set to 0, the AF bit does not initiate the $\overline{\text{IRQ}}$ signal. The $\overline{\text{RESET}}$ pin clears AIE to 0. The internal functions of the DS14285/DS14287 do not affect the AIE bit.

UIE - The Update Ended Interrupt Enable (UIE) bit is a read/write that enables the Update End Flag (UF) bit in Register C to assert $\overline{\text{IRQ}}$. The $\overline{\text{RESET}}$ pin going low or the SET bit going high clears to UIE bit.

SQWE - When the Square Wave Enable (SQWE) bit is set to a 1, a square wave signal at the frequency set by the rate-selection bits RS3 through RS0 is driven out on a SQW pin. When the SQWE bit is set to 0, the SQW pin is held low; the state of SQWE is cleared by the $\overline{\text{RESET}}$ pin. SQWE is a read/write bit.

DM - The Data Mode (DM) bit indicates whether time and calendar information is in binary or BCD format. The DM bit is set by the program to the appropriate format and can be read as required. This bit is not modified by internal functions or $\overline{\text{RESET}}$. A one in DM signifies binary data while a 0 in DM specifies Binary Coded Decimal (BCD) data.

24/12 - The 24/12 control bit establishes the format of the hours byte. A 1 indicates the 24-hour mode and a 0 indicates the 12-hour mode. This bit is read/write and is not affected by internal functions of $\overline{\text{RESET}}$.

DSE - The Daylight Savings Enable (DSE) bit is a read/write bit which enables two special updates when DSE is set to 1. On the first Sunday in April the time increments from 1:59:59 AM to 3:00:00 AM. On the last Sunday in October when the time first reaches 1:59:59 AM it changes to 1:00:00 AM. These special updates do not occur when the DSE bit is a 0. This bit is not affected by internal functions or $\overline{\text{RESET}}$.

6.1.2 Programme principal

Le programme consiste à saisir l'heure, la date, le mois et l'année en cours, puis à vérifier la saisie au fur et à mesure.

Puis il faut formater ces données d'abord pour mettre le circuit RTC à l'heure, l'autoriser à compter le temps et ensuite à afficher les informations à l'écran du LCD,

Enfin il faut lire le RTC et afficher les données toutes les secondes sur le LCD. Le cercle des secondes est effacé au début de chaque minute.

6.1.3 Organigramme

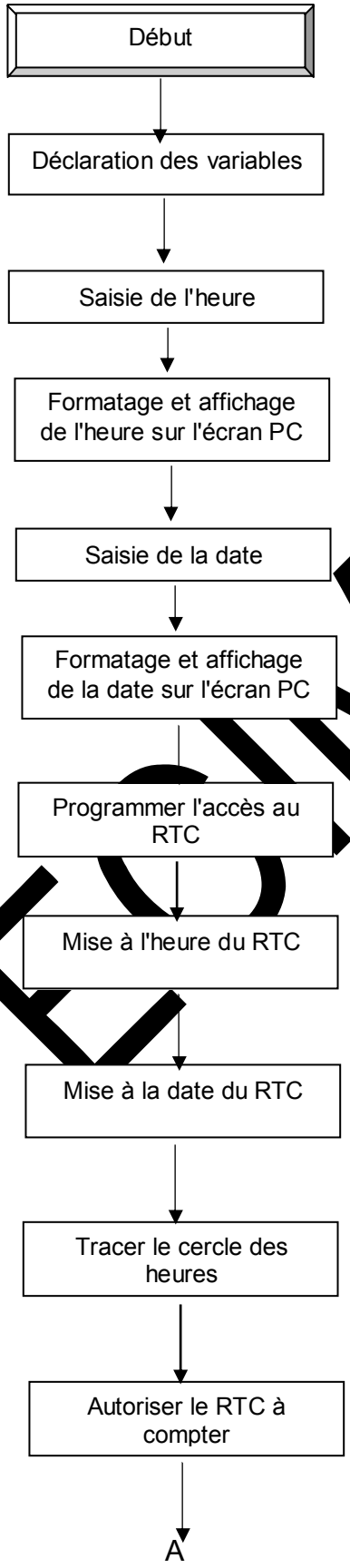
Vous disposez de l'utilitaire **void Tracer_Pixel (int x, int y, unsigned char Pen)** pour tracer des courbes en élaborant votre propre algorithme ou les utilitaires suivants avec leurs commentaires pour aboutir au même résultat.

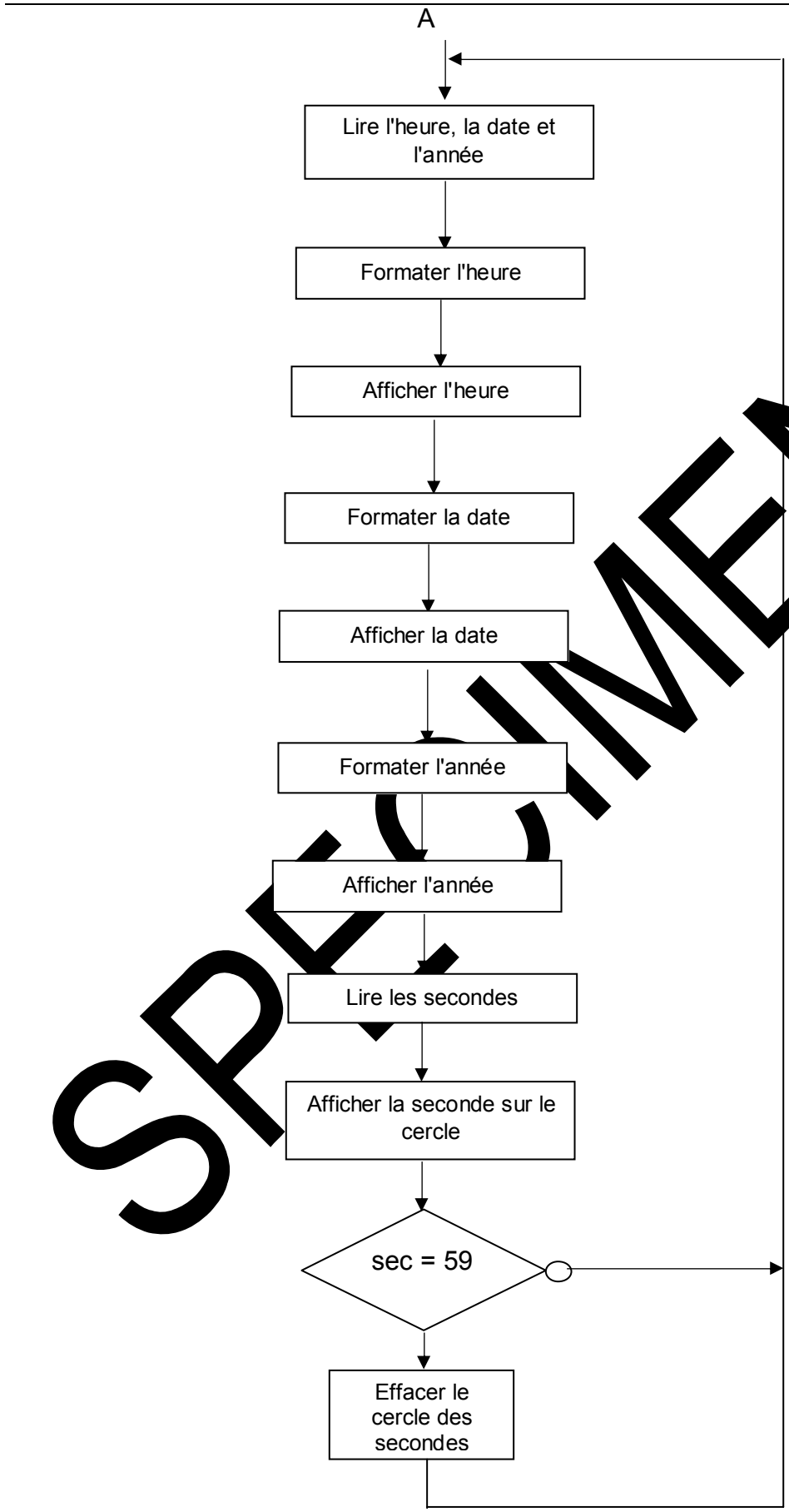
Les commandes 0xF8 et 0xF0 permettent d'allumer ou d'éteindre le pixel de numéro np.
Cela donne la définition suivante de la variable Pen :

Pen = 0xF8 + np ➔ allumage du pixel de numéro np.
Pen = 0xF0 + np ➔ extinction du pixel de numéro np..

Utilitaires	Commentaires
`void init_aff()`	Initialisation des paramètres TH, TA, GH, GA, Mode …
`void lcd_cls()`	Effacement de l'écran
`void lcd_write_command(unsigned char commande)`	Ecriture d'une commande
`void lcd_write_data(unsigned char data)`	Ecriture d'une donnée
`void lcd_clear_TXT()`	Effacer l'écran texte
`void lcd_clear_graph()`	Effacer l'écran graphique
`void lcd_gotoxy(unsigned char px, unsigned char py)`	Définition de la position d'un caractère ou d'un pixel ; px et py sont les données lower et upper. Remarque : pour un pixel il faut ajouter 0x04 à py.
`void lcd_out_str(char *texte)`	Envoi d'une chaîne de caractères pontée par la variable * texte
`void Tracer_Pixel(int x, int y, unsigned char Pen)`	Pen = 0xF8 ➔ Allumer le pixel Pen = 0xF0 ➔ Eteindre le pixel

void Tracer_LH (unsigned char M, unsigned char N, unsigned char P, unsigned char Q, unsigned char Pen)	M, N : coordonnées du point de départ P,Q : coordonnées du point de départ
void Tracer_LV (unsigned char M, unsigned char N, unsigned char P, unsigned char Q, unsigned char Pen)	idem
void Droite (int x1, int y1, int x2, int y2, int Pen)	idem
double Cercle_H (int x0, int y0, int r, int Pen, int h);	
void Cercle_S (int x0, int y0, int r, int Pen, int sec);	
void Effacer_Cercle_S (int x0, int y0, int r, int Pen, int sec);	
void writebyte(unsigned char adr, unsigned char dat);	adr[illegible]nte[illegible]'adr[illegible]se du [illegible]tre dans le PTC.
unsigned char readbyte(unsigned char adr)	[illegible]dem


Début
Déclaration des variables
Saisie de l'heure
Formatage et affichage de l'heure sur l'écran PC
Saisie de la date
Formatage et affichage de la date sur l'écran PC
Programmer l'accès au RTC
Mise à l'heure du RTC
Mise à la date du RTC
Tracer le cercle des heures
Autoriser le RTC à compter
A


A
Lire l'heure, la date et l'année
Formater l'heure
Afficher l'heure
Formater la date
Afficher la date
Formater l'année
Afficher l'année
Lire les secondes
Afficher la seconde sur le cercle
sec = 59
Effacer le cercle des secondes

6.1.4 Programme en C

```
/***************************************************** ***
*     TP SUR LA CARTE EID005                             *
*********************************************************
*     Dessin d'une horloge et l'affichage de l'heure,    *
*     la date et l'année sur le LCD                      *
*                                                        *
*********************************************************
*     Nom du FICHIER : EID005_TP6.c                      *
*     **************                                     *
*********************************************************/

//    Nom: Récap_OK.c

#include "eid005.h"
#include <stdio.h>
#include <string.h>
#include <math.h>

/*****************************/
/*    Variables globales     */
/*****************************/

/*
double Cos[60], Sin[60];  //  Tableaux des 60 points représentant les
secondes
unsigned char X, Y, np ;  //  X, Y : Position d'un pixel dans le plan
                          //  [ x y ] = [64 128]
                          //  np : n° du pixel sur la ligne X
double  pi = 3.14159265 ;
double  deuxpi = 6.283185530 ;
//    K = Facteur de correction entre les dimensions séparant 2 points en x
et en y ;
      K = 1.4872 ;
*/

/*****************************/
/*    Fonctions Utilisées    */
/*****************************/

int PL = 0xF0 ;               //    Plume Levée   ==> Effacer
int PB = 0 ;                  //    Plume Baissée ==> Tracer

main()
{
/*****************************/
/* DECLARATION DES VARIABLES  */
/*****************************/
char depart, x ;
int  R, hr, mn, sc, u, l, k ;
int jour, date, mois, an ;
int ahr, amn, asc ;
char * Jour ;
//char * Jour[7];
char * Mois ;
unsigned char stop ;

//    Saisie pour la mise à jour du RTC
```

```
unsigned char Annee[4] =  { 0x2, 0x0, 0x0, 0x0 } ; // Affichage année
unsigned char DMA [7];              // Saisie Date_Mois_Année
unsigned char Date_Mois[5] ;        // Affichage Date et Mois
unsigned char Heure[6] ;            // Saisie et affichage
unsigned char Heure_Alarme[3] ;     // Saisie
unsigned char tmp;
char * Jours[7] = {"Lundi", "Mardi", "Mercr", "Jeudi", "Vendr", "Samed",
"Diman"};
char *_Mois[12]= {"Janv", "Fevr", "Mars", "Avri", "Mai", "Juin", "Juill",
"Août",
                  "Sept", "Octo", "Nove", "Decem"};

/*****************************/
/*****************************/
/*                           */
/*    PROGRAMME PRINCIPAL    */
/*                           */
/*****************************/
/*****************************/

// Calcul sinus cosinus 60 points : secondes

Tab_Sin_Cos ();   // Pour affichage des points heures et secondes

// Initialisation afficheur

init_aff();
lcd_cls();

/*****************************************/
/*    Saisie de l'heure ou lancer lecture*/
/*****************************************/

/* Taper :
            hh = heures puis Entrée,
            mm = minutes puis Entrée
            ss = secondes puis Entrée.
*/
printf("    Taper hh pour mettre à l'heure \n    ou autre pour lancer
\n");
scanf ("%c", &depart);
if (depart == 'hh')

{     printf("    Entrer l'heure : hhmmss \n");
      for (u=0; u<6; u++)
      {
      scanf("%c", &x);
      if (x != 0x08)
            {     if ( u <0)
                  u = 0;
                  Heure[u] = x & 0x0F;
                  printf("    \t\t%d \n",(x & 0x0F));
                  printf ("                  Il reste encore %d
chiffre(s)\n", 5-u);
            }
      else
            {     u -= 2;           // Pour effacer les codes de Back Space
et Entrée
                  printf ("\n  RECTIFICATION !\n");
            }
      }
```

```
/*************************************************/
/*    Mise enforme hhmmss : Tableau Heure[]     */
/*************************************************/

      Heure[0] = 10*Heure[0] + Heure[1] ; // Heures
      Heure[1] = 10*Heure[2] + Heure[3] ; // Minutes
      Heure[2] = 10*Heure[4] + Heure[5] ; // Secondes

//    Verification de l'heure saisie
      printf ("    %d h %d mn %d s\t\n", Heure[0], Heure[1], Heure[2]);

/*****************************/
/*    Saisie de la date      */
/*****************************/

/* Taper :
            j = jour puis Entrée,
            dd = date puis Entrée,
            mm = mois puis Entrée et
            aa = année puis Entrée.
*/

printf ("    Entrer la date : jddmmaa \n");
for (u =0; u < 7; u++)
      {
      scanf("%c",&x);
      if (x != 0x08)
            {     if ( u <0)
                  u = 0;
                  DMA[u] = x &
                  printf("     \t\t%d \n",     & 0x0F));
                  printf ("                il reste encore %d
chiffre(s)\n", 6-u);
            }
      else
            {     u = 2;                Pour effacer les codes de Back Space
et Entrée
                  printf("\n  RECTIFICATION !\n");
            }
      }

/*************************************/
/*    Mise enforme jddmm : Tableau DMA[]        */
/*************************************/

DMA [1] = 10*DMA [1] + DMA [2];     // Date
DMA [2] = 10*DMA [3] + DMA [4];     // Mois
DMA [3] = 10*DMA [5] + DMA [6];     // Année


//    Vérification de la date saisie

printf ("    %s %d %s 200%d  \n",Jours [DMA[0]-1],DMA[1], _Mois [DMA [2]-
1],DMA [3]);
```

```
/*************************************************/
/*    Programmation du circuit RTC : DS14285    */
/*************************************************/

//---  Mettre à 1 le bit 0 du registre de contrôle du RTC : accès au RTC

ctrl_rtc = 1 ;

//--------   Programmation des registres A et B

writebyte (REGB, 0x82); // Registre B = 1000 0010
                        // Autorisation de la mise à jour
                        // Sans incrémentation des compteurs (sec, mm, h).
writebyte (REGA, 0x20); // Registre A = 0010 0000 :

//--------  Mise à l'heure

writebyte (0, Heure[2]);
writebyte (2, Heure[1]);
writebyte (4, Heure[0]);

/*--------  Programmation de l'alarme

writebyte (1, Heure_Alarme [0]);
writebyte (3, Heure_Alarme [1]);
writebyte (5, Heure_Alarme [2]);
*/

//--------  Programmation de la Date

writebyte (6, DMA [0]);      // Jour de la semaine : 1 = lun, 2 = mar, ...

writebyte (7, DMA [1]) ;     // Date dans le mois
writebyte (8, DMA [2]) ;     // Mois dans l'année
writebyte (9, DMA [3]) ;     // Année
}
//--------  Autorisation de lancer la RTC

printf ("    LED RTC en route. Appuyer sur RESET pour le stopper \n");

/*************************************************/
/*    TRACE DE L'HORLOGE      */
/*************************************************/

//---------- Cercle cercle heures    --------------------

for (u = 0; u < 60; u += 5)
Cercle_H (32,64, 28, PB, u);  // Dessin de l'horloge

Rectangle(0,0,63,127,PB);     // Rectangle au tour de l'horloge

//------- Lancement RTC : incrémentation des compteurs (sec, mm, h)

writebyte (REGB, 06);
```

```
/*************************************************/
/*    Lecture RTC et affichage de des données  */
/*************************************************/

do
{
      do
            tmp = readbyte(REGA);
      while ( tmp & 0x80);    // Attente UIP

//-----     Lecture de l'heure

sc = readbyte(0);
mn = readbyte(2);
hr = readbyte(4);

//-----     Lecture jour, date, mois

jour = readbyte(6);
date = readbyte(7);
mois = readbyte(8);
an   = readbyte(9);


//-----     Formatage et affichage Heur, minute, seconde

      Heure[0]= hr / 10 ;
      Heure[1]= hr % 10 ;
      Heure[2]= 0x0a ;        // Code ASCII décalé de l'Afficheur
      Heure[3]= mn / 10 ;
      Heure[4]= mn% 10 ;
      Heure[5]= 0x0a ;        // idem
      Heure[6]= sc / 10 ;
      Heure[7]= sc % 10 ;

lcd_gotoxy(0x55,0x0);
lcd_out_Tab(Heure, 8);

//-----     Formatage et affichage Jour

lcd_gotoxy(0x25,0x0);
lcd_out_str(Jours[jour-1]);

//-----     Formatage et affichage Date et Mois

      Date_Mois[0]= date / 10 ;     // Dizaine Date
      Date_Mois[1]= date % 10 ;     // Unité date

//  Si affichage du mois en chiffre :
      Date_Mois[2]= 0xFF ;          // Code ASCII décalé de l'Afficheur -
0x10
      Date_Mois[3]= mois / 10 ;     // Dizaine mois
      Date_Mois[4]= mois % 10 ;     // Unité mois

lcd_gotoxy(0x34,0x0);
lcd_out_Tab(Date_Mois, 2);

lcd_gotoxy(0x37,0x0);
lcd_out_str(_Mois[mois-1]);
```

```
//-----     Formatage et Affichage Année

Annee [2] = an / 10 ;   // Dizaine Année
Annee [3] = an % 10 ;   // Unité Année

lcd_gotoxy(0x46,0x0);
lcd_out_Tab(Annee, 4);

//-----     Lecture et Affichage des secondes

if (sc == 59)     // Effacer les secondes
{
      Cercle_S (32,63, 28, PB, sc);
      for( u = 0; u<60; u++)
      Effacer_Cercle_S (32,63, 28, PL, u);
}
else
      Cercle_S (32,63, 28, PB, sc);
}
while (1);

}

//----      FIN DU PROGRAMME PRINCIPAL
```