Graphic Display -
Keyboard -
Real Time Clock
EID005 Board

Author: KOMA N'Gally
Professor BTS IRIS
IFA Delorozoy
CCI Versailles


Sample

SUMMARY

Ex.1 : WTITE A STRING IN TEXT FORM ON THE SCREEN 5
1.1 Topic 5
1.2 Analysis and solution 6
1.2.1 Display brief description 6
1.2.2 C Program 16
Ex.2 : WTITE A STRING IN TEXT FORM ON THE SCREEN 18
2.1 Topic 18
2.2 Analysis and solution 19
2.2.1 Display brief description 19
2.2.2 C Program 29
Ex.3 : READING A MATRIX KEYBOARD ON POLLING MODE 30
3.1 Topic 30
3.2 Analysis and solution 31
3.2.1 Keyboard brief description 31
3.2.2 C Program 35
3.2.3 Assembler Program 37
Ex.4 : KEY ACTIVATION DETECTION AND READING ON POLLING MODE 44
4.1 Topic 44
4.2 Analysis and solution 45
4.2.1 Keyboard brief description 45
4.2.2 C Program 49
4.2.3 Assembler Program 51
Ex.5 : LINES, CIRCLES AND CURVES DRAWING ON THE LCD 58
5.1 Topic 58
5.2 Analysis and solution 59
5.2.1 Display description on graphic mode 59
5.2.2 Main program 61
5.2.3 Flowchart of a sinusoid drawing on an oscilloscope 62
5.2.4 C Program 65
Ex.6 : A CLOCK DRAWING ON THEGRAPHIC SCREEN 67
6.1 Topic 67
6.2 Analysis and solution 68
6.2.1 Clock geometric definition 68
6.2.2 Main program 71
6.2.3 Flowchart 71
6.2.4 C Program 75


Sample

EX.1 : WTITE A STRING IN TEXT FORM ON THE SCREEN

1.1 Topic

Purposes :	Be able to use the utilities stored in the library, allowing the 128x64 pixels graphic LCD display management
Specifications:	**Subject** Write a C program which allows the string display. The maximum string length is 32 characters. We give the coordinate of the first character.

Necessary Equipment :

PC Micro-Computer using Windows 95 or latter,
68332 Micro-Controller board with its motherboard, Ref: EID 210 001
Keyboard-Display-Real Time Clock board: EID005001
Network connection cable and RS232 cable, Ref. : EGD 000 003
AC/AC 8V Power Supply, 1A, Ref. : EGD000001,

Necessary Document :

DMS Keyboard-Display-Real Time Clock board document: EID00500
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Time : 3 hours

1.2 Analysis and solution

1.2.1Display brief description

1.2.1.1 LCD 128x64 display presentation

Attention :

Because of the regulation in the manufacturer's documentation, the x and y variables represent respectively the ordinate (vertical) and the abscissa (horizontal). The point "x = 0, y = 0" is at the left bottom of the LCD, while the point "x = 63, y = 127" is at the right top of the LCD.

The T6963C controller has an 8 kB memory.

Text Mode *fig.1*

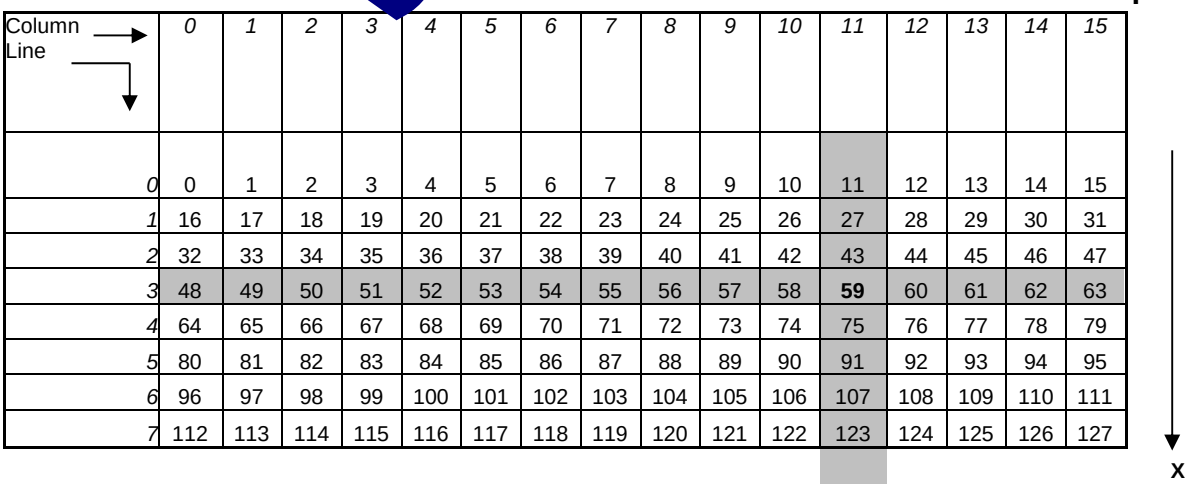
In the following studies, the text zone is put in the screen memory (VRAM) from address 0000 to 007F, having128 characters.
Least significant byte : **00 always fixed**
Most significant byte : **00 to 7F** on hexadecim
The quartet of least significant byte indicate mber x.
The quartet of most significant byte indicat nn number y.

Example

The character number **59** is put in the **x = 3, y = 11**. The coordinate which transforms on hexadecimal is **x y B.**
In the memory, there are the following by r the TH (Text Home) parameter address:

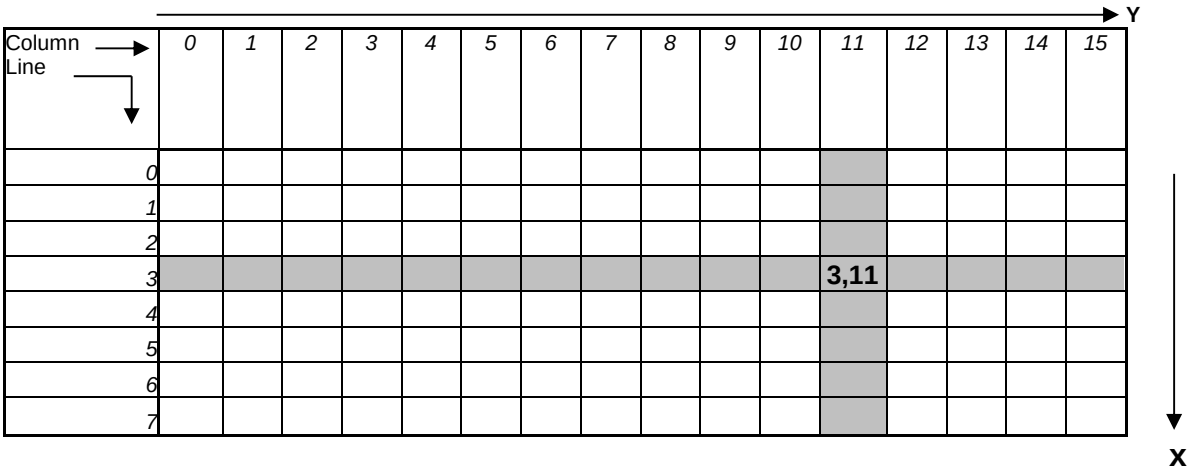
TH lower 3B (59 on decimal)
TH 0x00

On Text Mode



Line \ Column	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

fig.1



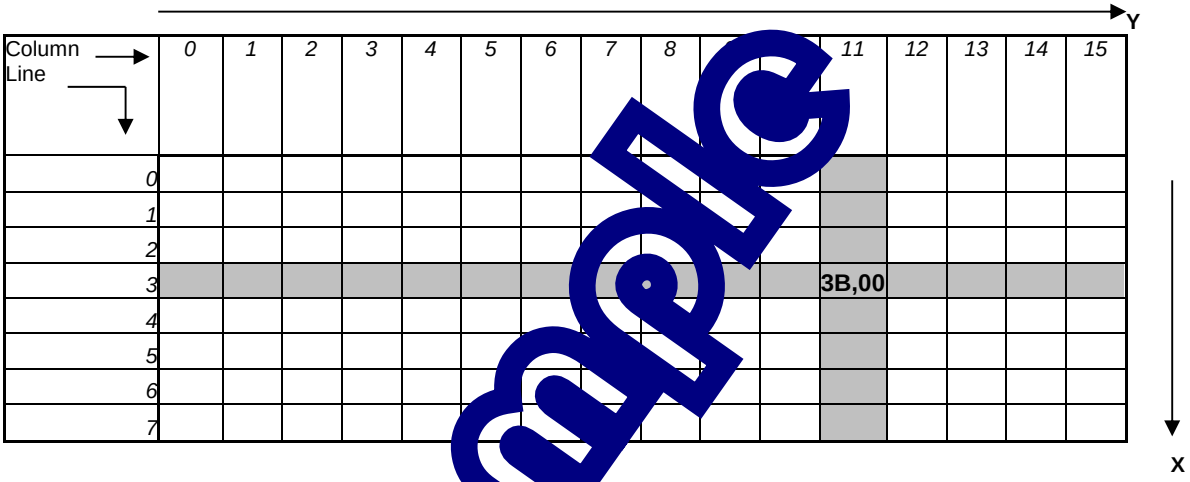


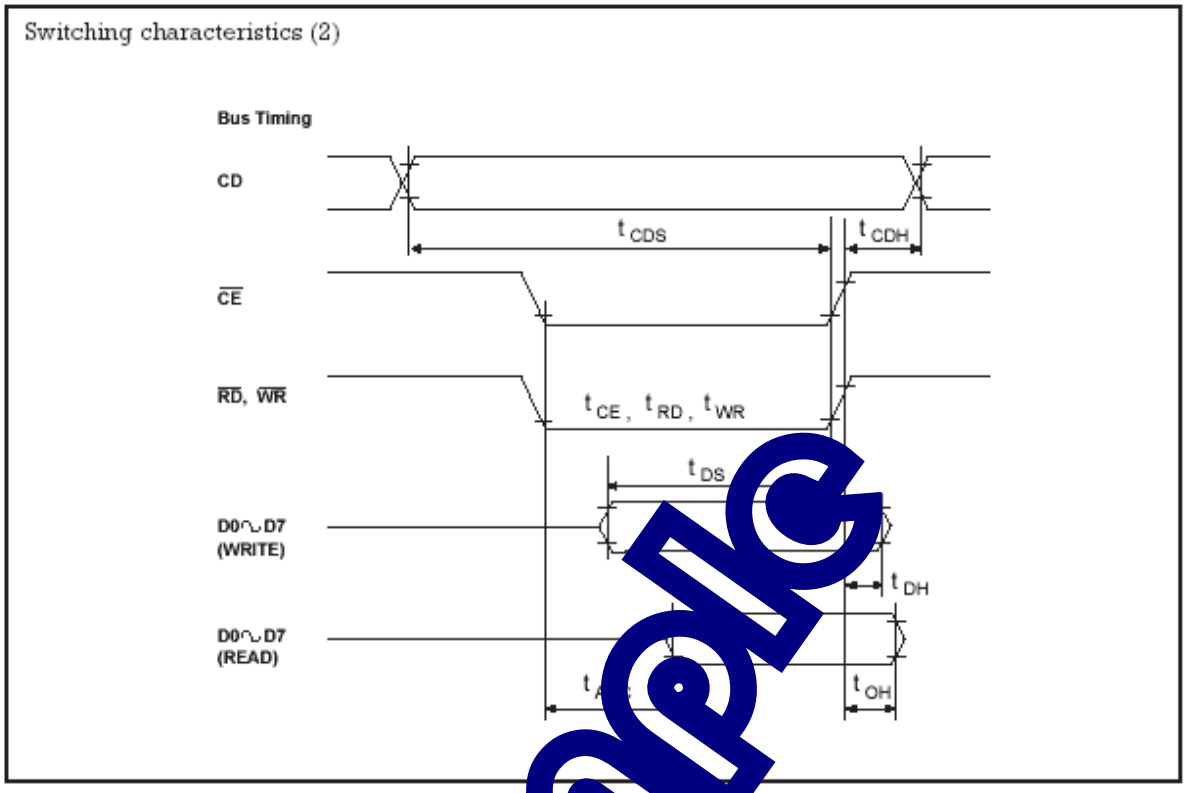
fig owing

Internal character gene ato G RO)

The internal character ge tor ses an ASCII code called 0x20;
For example: the letter A in C encoded 0x41 is represented by the value 0x21 in the T6963C (see table below).
This means that to send an ASCII character, we have to subtract its code, the value 0x20.

Timetable data or command Writing / Reading

These timetables must be generated for each access to the LCD.
They are realized and described in detail especially in the Assembler sub-program.

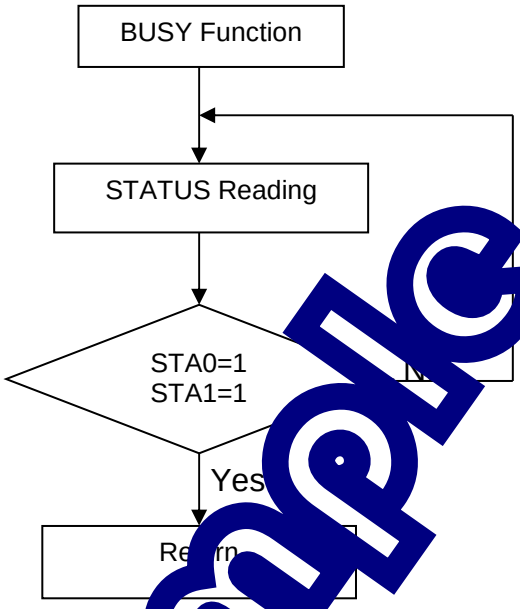


Each timetable generation will result in a corresponding sub-program.

x

1.2.1.2 LCD display management

Before each data (or command) movement (write or read) between the display and the control processor, we must ensure that the LCD is ready to execute the function. Thus it's necessary to start by testing the status bits of its status register: STA0 and STA1.
The flowchart is given as the following:



1.2.1.3 Display instructions

The display instructions are defined in the table below.

The flowcharts below define the modes of command instruction writing which needs 0, 1 or 2 bytes.

The following flowcharts define in detail the mode of command and data reading and writing.

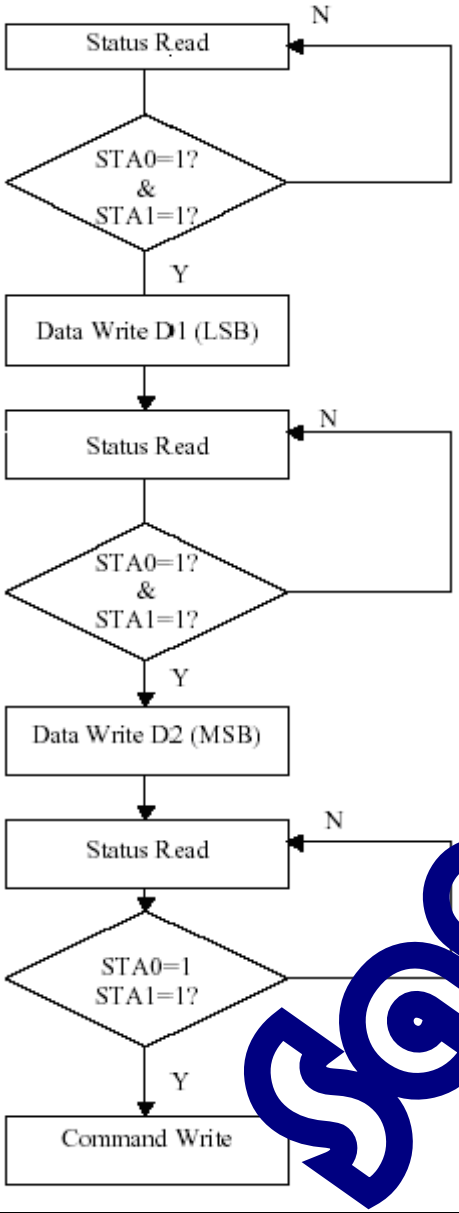
T6963C Instruction Set

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 (no attribute capability)	
Display Modes	1	0	0	1	N3	N2	N1	N0		32 x 1/fosc
					0				Gr[illegible]	
					1				[illegible]ics	
						0			[illegible]	
						1			Text [illegible]	
							0		[illegible] Off	
							1		[illegible]	
								0	Cursor blink Off	
								1	Cursor blink On	
Cursor Pattern Select	1	0	1	0	0	N2	N1	N0	0: No. of lines for cursor +1	32 x 1/fosc
						0	0		Bottom Line cursor	
						0	1		[illegible] cursor	
						1	0			
						1	1		8 line cursor (block cursor)	
Data Auto Read/Write	1	1	0	0	0	0	0	N0		32 x 1/fosc
								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	0	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)	

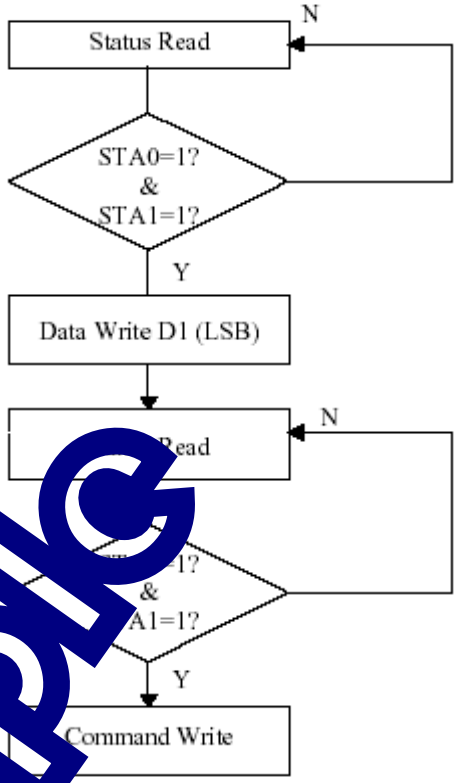
Note:

1. * = DONT CARE

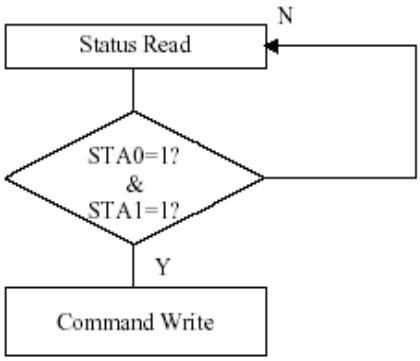
2 bytes Command



1 byte Command

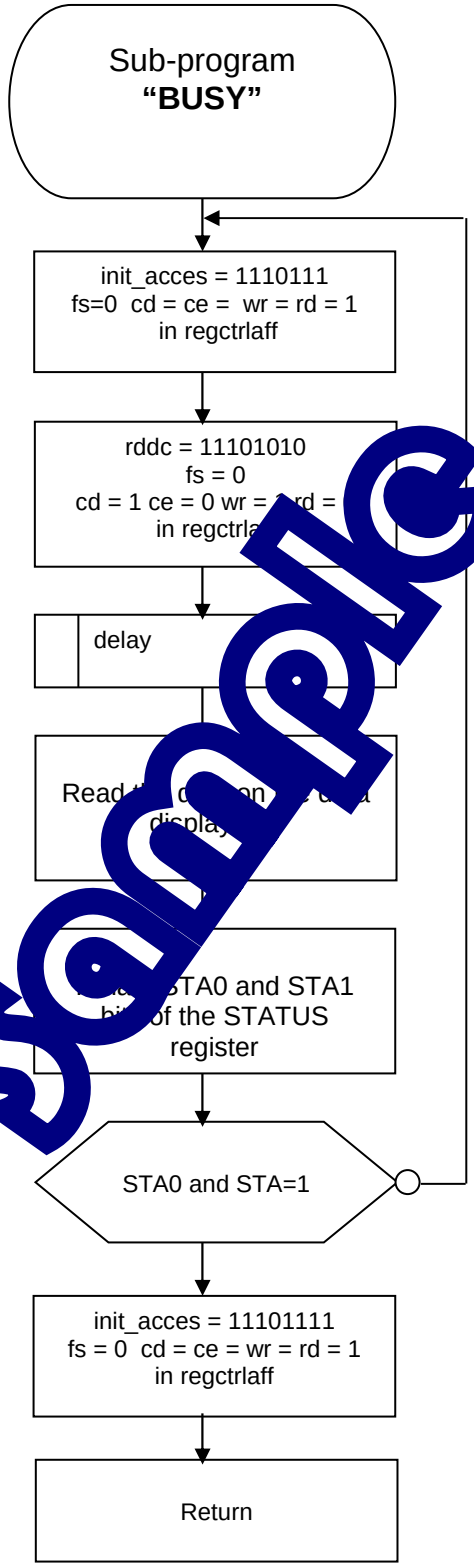


0 byte Command

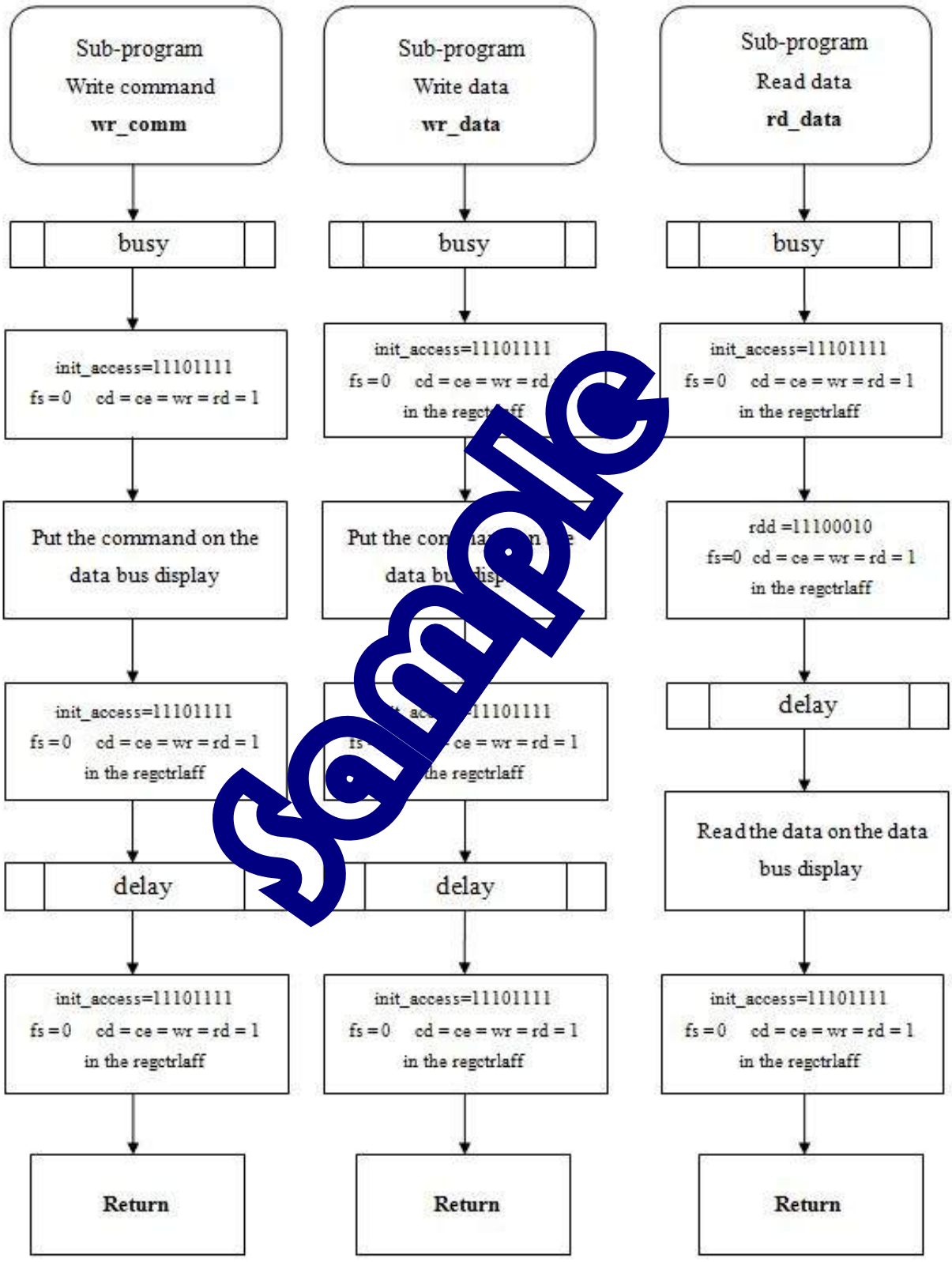


Detailed flowcharts of the main sub-programs

Sub-program « BUSY »



Sub-programs Reading/Writing



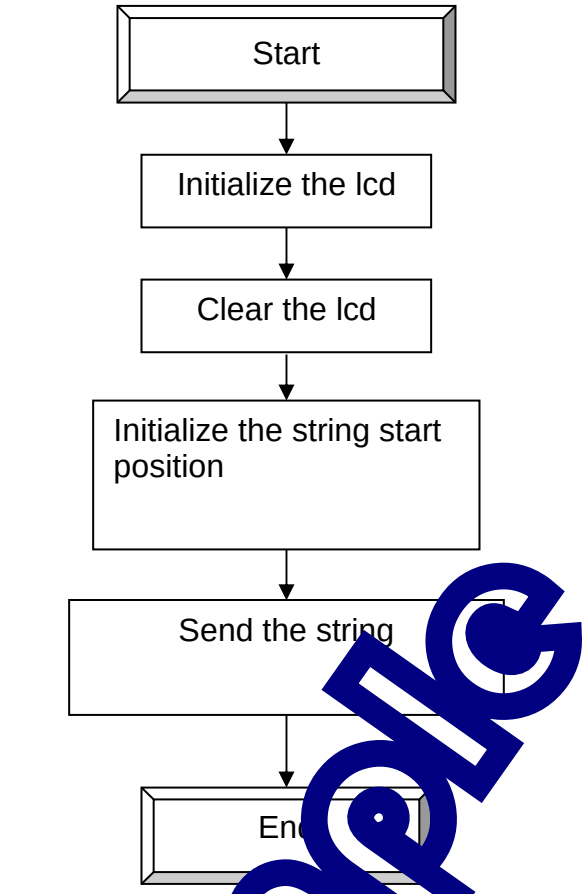
1.2.1.4 Main flowchart

You have the following functions with their comments.

Functions	Comments
`void init_aff()`	Parameters TH, TA, GH GA Mode initialization
`void lcd_cls()`	Clear the screen
`void lcd_write_command(unsigned char commande)`	Command writing
`void lcd_write_data(unsigned char data)`	Data writing
`void lcd_gotoxy(unsigned char px, unsigned char py)`	Definition of a character or a pixel position; px and py are the lower and upper data.
`void lcd_out_str(char *texte)`	Sending a string by the *texte variable

Important :

To use their functions as well as the keyboard display with C / C + + compiler, we must configure the eid210 software linker.
To do this: Go to the configuration menu and click GNU C / C + +, then go to the * linker * click * add * tab, and then select the "EID005_Lib.o" file and at last click open.


Start
Initialize the lcd
Clear the lcd
Initialize the string start position
Send the string
End

1.2.2 C Program

```
/*******************************************************
*     PRACTICAL WORKS ON EID005 BOARD                                *
*******************************************************
*     Write a program in assembler                          *
*     and in C which realize the display                *
*     of a string.                              *
*     the maximum size of string is: 128 characters  *
*******************************************************
*     File Name: EID005_TP1.c                       *
*     *************                                 *
*******************************************************/

//    Inclusion of definition files

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

//============================
//    Main Function
//============================

main()
{
init_aff();             //    Initializ t i lay
lcd_cls();              //    Clear the cre
lcd_gotoxy(0x10,00);    //    Defin    the tring start position

//    Sending the string

lcd_out_str ( "EID005000 :     K BO          DISPLAY,      REAL TIME
CLOCK");

}

// Spaces in the tex  ca    id    ting a word


//    End of the program
```


Samplic

EX.2 : WTITE A STRING IN TEXT FORM ON THE SCREEN

2.1 Topic

Purposes :	Be able to use the utilities stored in the library, allowing the 128x64 pixels graphic LCD display management
Specifications:	**Subject** Write a program in C a[illegible] which realizes the string display. The maximum string le[illegible] characters. We give the coordina[illegible] character.

Necessary Equipment :

PC Micro-Computer us[illegible] or latter,
68332 Micro-Controller [illegible] Mother Board, Ref: EID 210 001
Keyboard-Display-Real Ti[illegible] board: EID005001
Network connection cable a[illegible] 232 cable, Ref. : EGD 000 003
AC/AC 8V Power Supply, 1 A, Ref. : EGD000001,

Necessary Document :

DMS Keyboard-Display-Real Time Clock board document: EID00500
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Time : 3 hours

2.2 Analysis and solution

2.2.1 Display brief description

2.2.1.1 LCD 128x64 display presentation

Attention :

Because of the regulation in the manufacturer's documentation, the x and y variables represent respectively the ordinate (vertical) and the abscissa (horizontal). The point "x = 0, y = 0" is at the left top of the LCD, while the point "x = 63, y = 127" is at the right bottom of the LCD.

The T6963C controller has an 8 kB memory.

Text Mode *fig.1*

In the following studies, the text zone is put in the scre[illegible]mory (VRAM) from address 0000 to 007F, having128 characters.
Least significant byte : **00 always fixed**
Most significant byte : **00 to 7F** on hexadecim[illegible]
The quartet of least significant byte indicates [illegible]n li[illegible]number x.
The quartet of most significant byte indica[illegible]s t[illegible]mn number y.

Example

The character number **59** is put in th[illegible] coor[illegible]: **x = 3, y = 11**. The coordinate which transforms on hexadecimal is **x [illegible] y [illegible]B.**
In the memory, there are the follow[illegible]byte[illegible]or the TH (Text Home) parameter address:

TH lower [illegible]d[illegible]s = [illegible]x3B (59 on decimal)
TH [illegible]pp[illegible]r [illegible]ddres[illegible] = 0x00

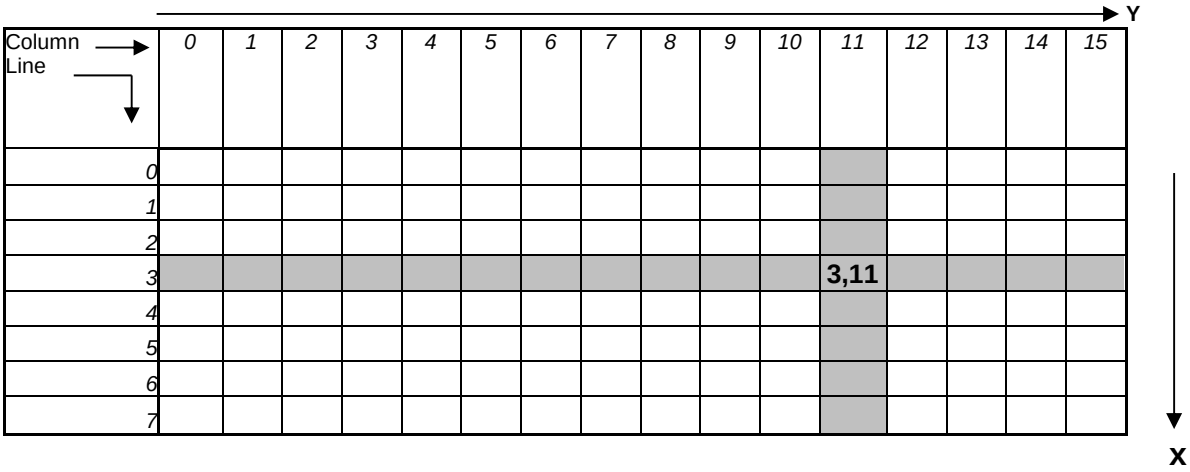
On Text Mode

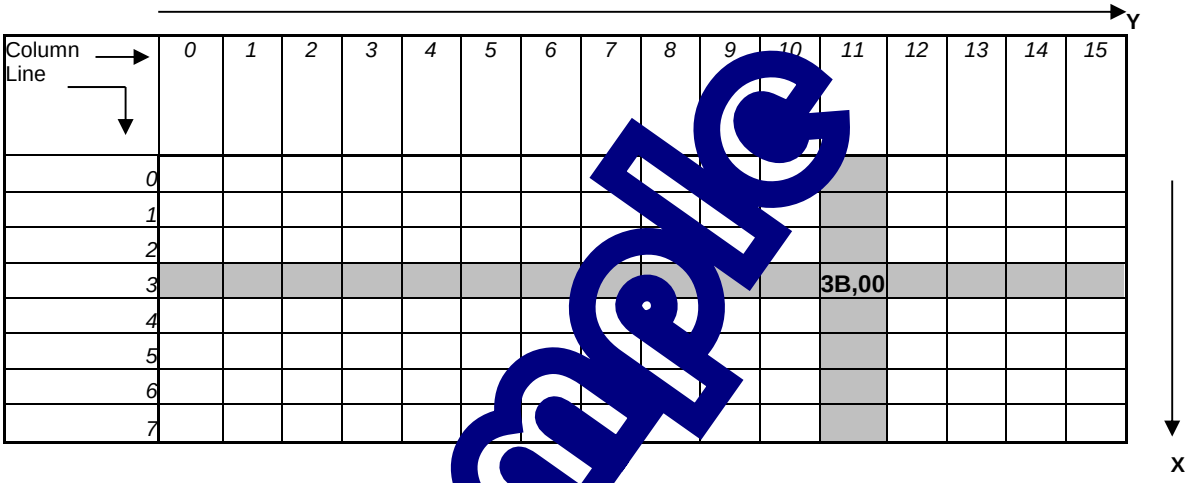
Y

Column → / Line ↓	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





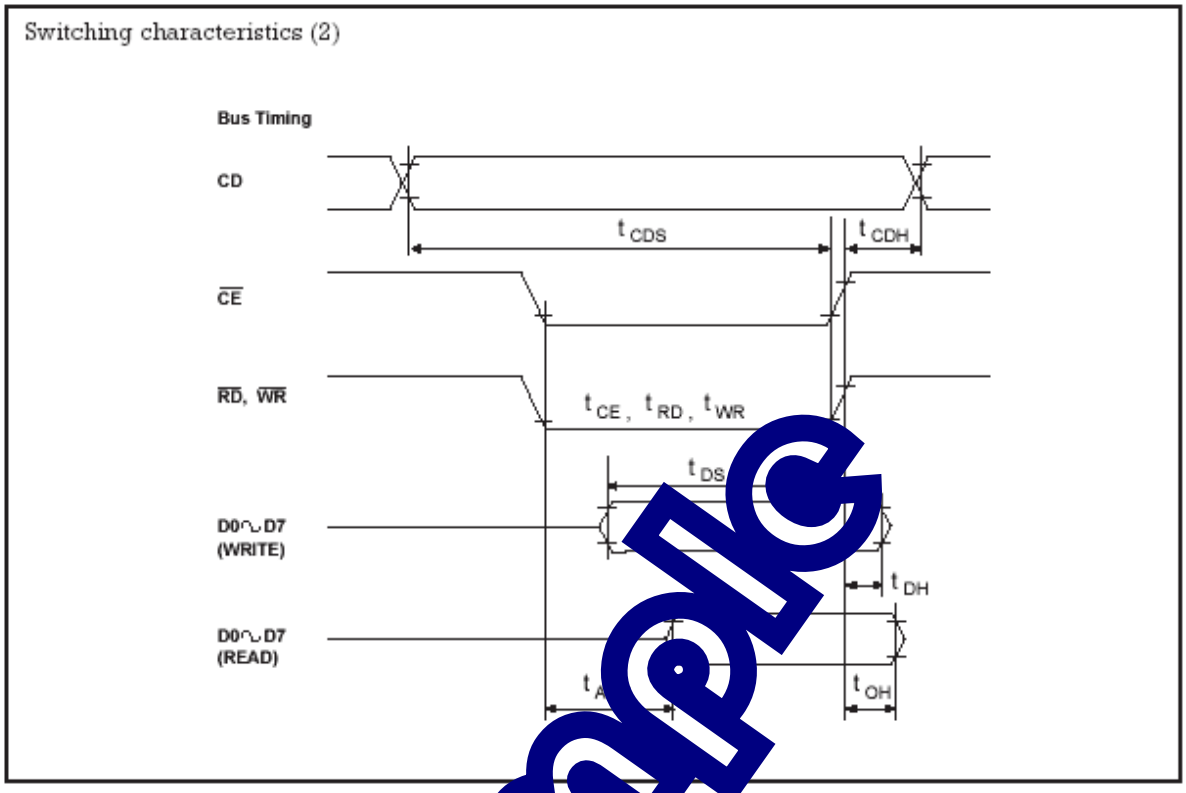
...owing

Internal character gene... (CG...

The internal character g...erato...ses an ASCII code called 0x20;
For example: the letter A in ...C ...encoded 0x41 is represented by the value 0x21 in the T6963C (see table below)
This means that to send an ASCII character, we have to subtract its code, the value 0x20.

Timetable data or command Writing / Reading

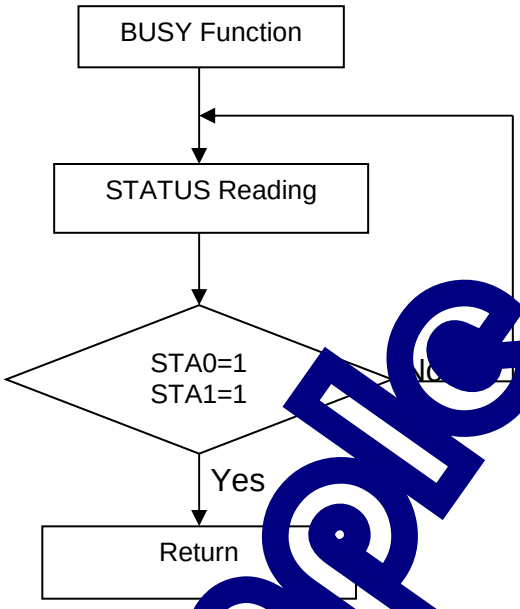
These timetables must be generated for each access to the LCD.
They are realized and described in detail especially in the Assembler sub-program.



Each timetable generation will re[illegible] corresponding sub-program.

2.2.1.2 LCD display management

Before each data (or command) movement (write or read) between the display and the control processor, we must ensure that the LCD is ready to execute the function. Thus it's necessary to start by testing the status bits of its status register: STA0 and STA1.
The flowchart is given as the following:



2.2.1.3 Display instructions

The display instructions are given in the table below.

The flowcharts below define the mode of command instruction writing which needs 0, 1 or 2 bytes.

The following flowcharts define in detail the mode of command and data reading and writing.

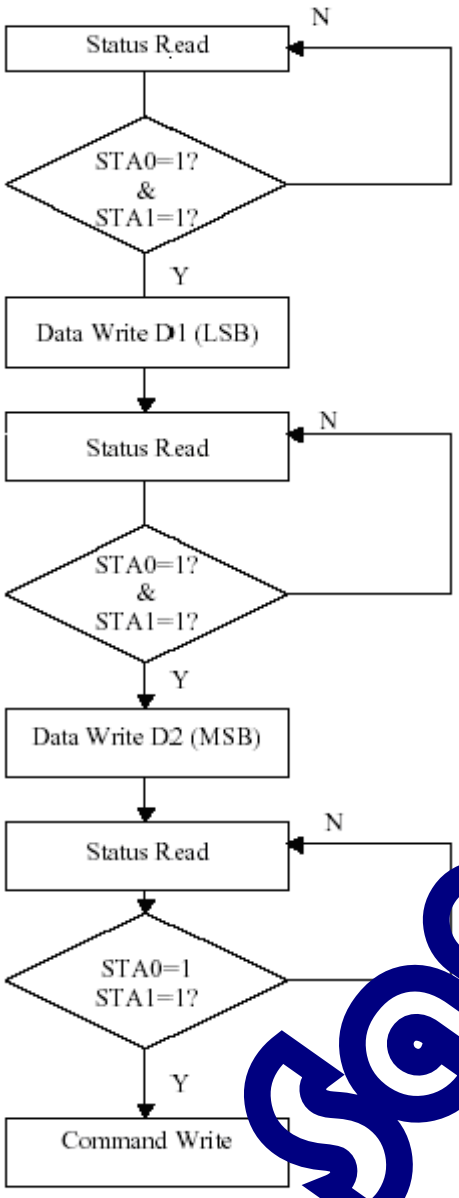
T6963C Instruction Set

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 [illegible] capability)	
Display Modes	1	0	0	1	N3	N2	N1	N0		32 x 1/fosc
					0				Gra[illegible]	
					1				[illegible] On	
						0			Tex[illegible]	
						1			[illegible] On	
							0		[illegible]	
							1		Cu[illegible]	
								0	[illegible] blink Off	
								1	[illegible] blink On	
Cursor Pattern Select	1	0	1	0	0	N2	N1	N0	[illegible]0: No. of lines for cursor +1	32 x 1/fosc
								0	[illegible] Line cursor	
								1	2 line cursor	
							1	0		
						1	1	1	8 line cursor (block cursor)	
Data Auto Read/Write	1	1	0	0	0	0	0	N0		32 x 1/fosc
								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)	

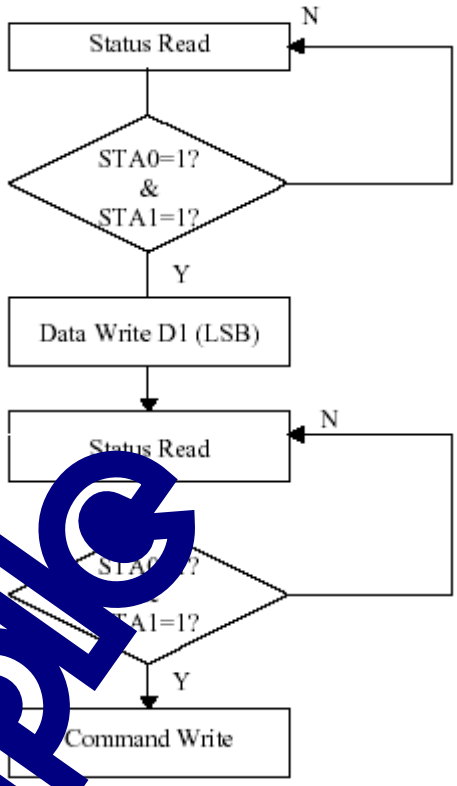
Note:

1. * = DONT CARE

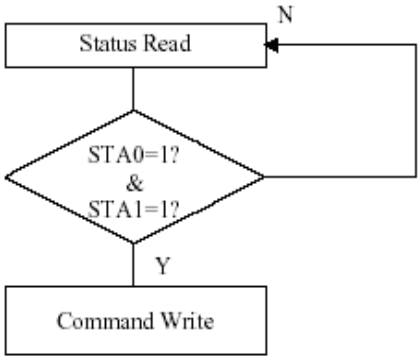
2 bytes Command



1 byte Command

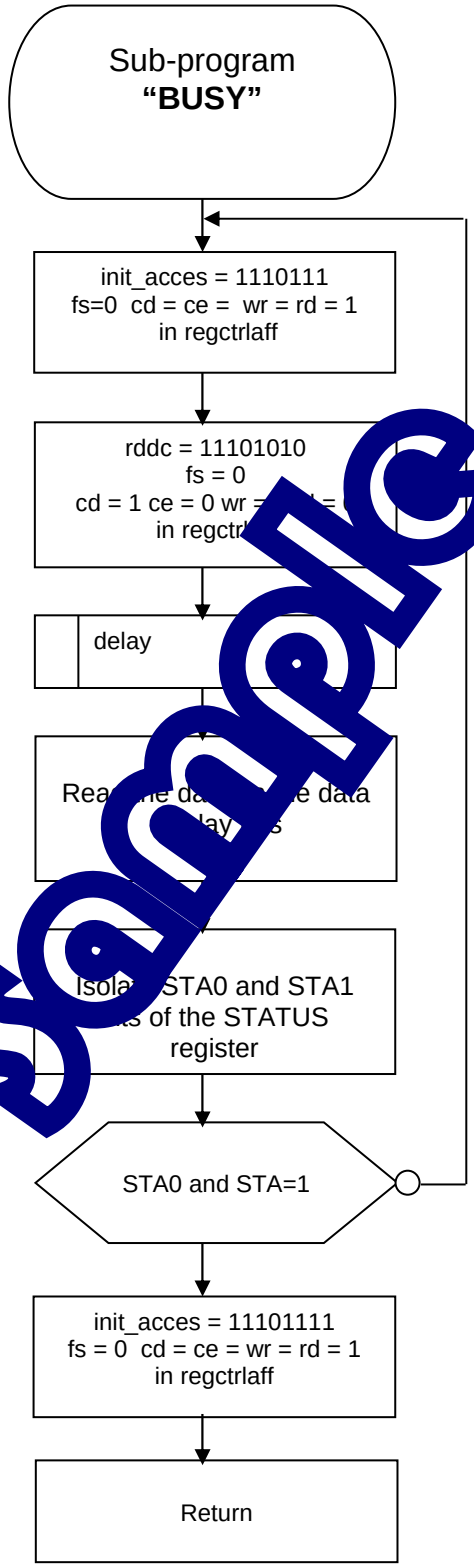


0 byte Command

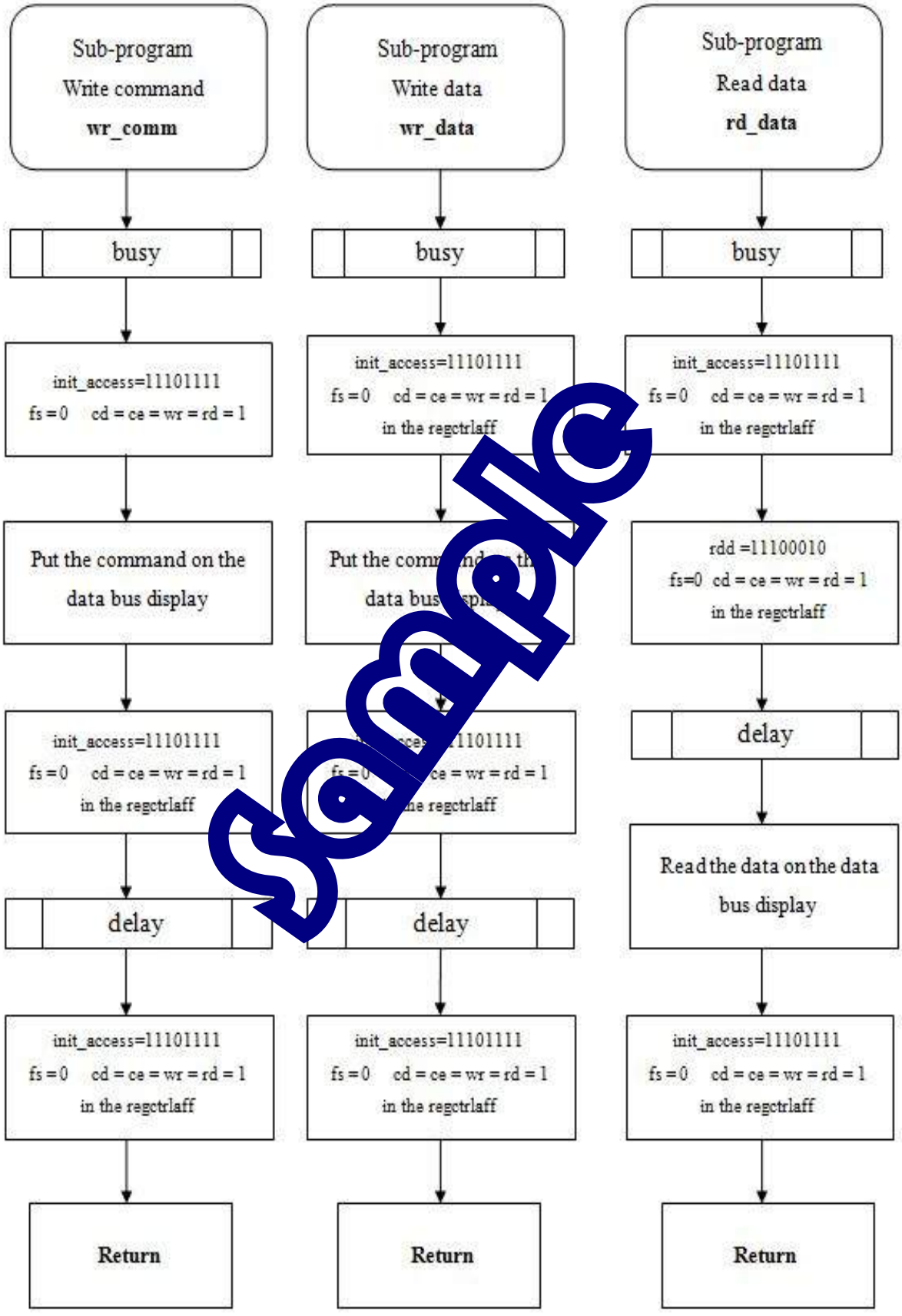


Detailed flowcharts of the main sub-programs

Sub-program « BUSY »



Sub-programs Reading/Writing



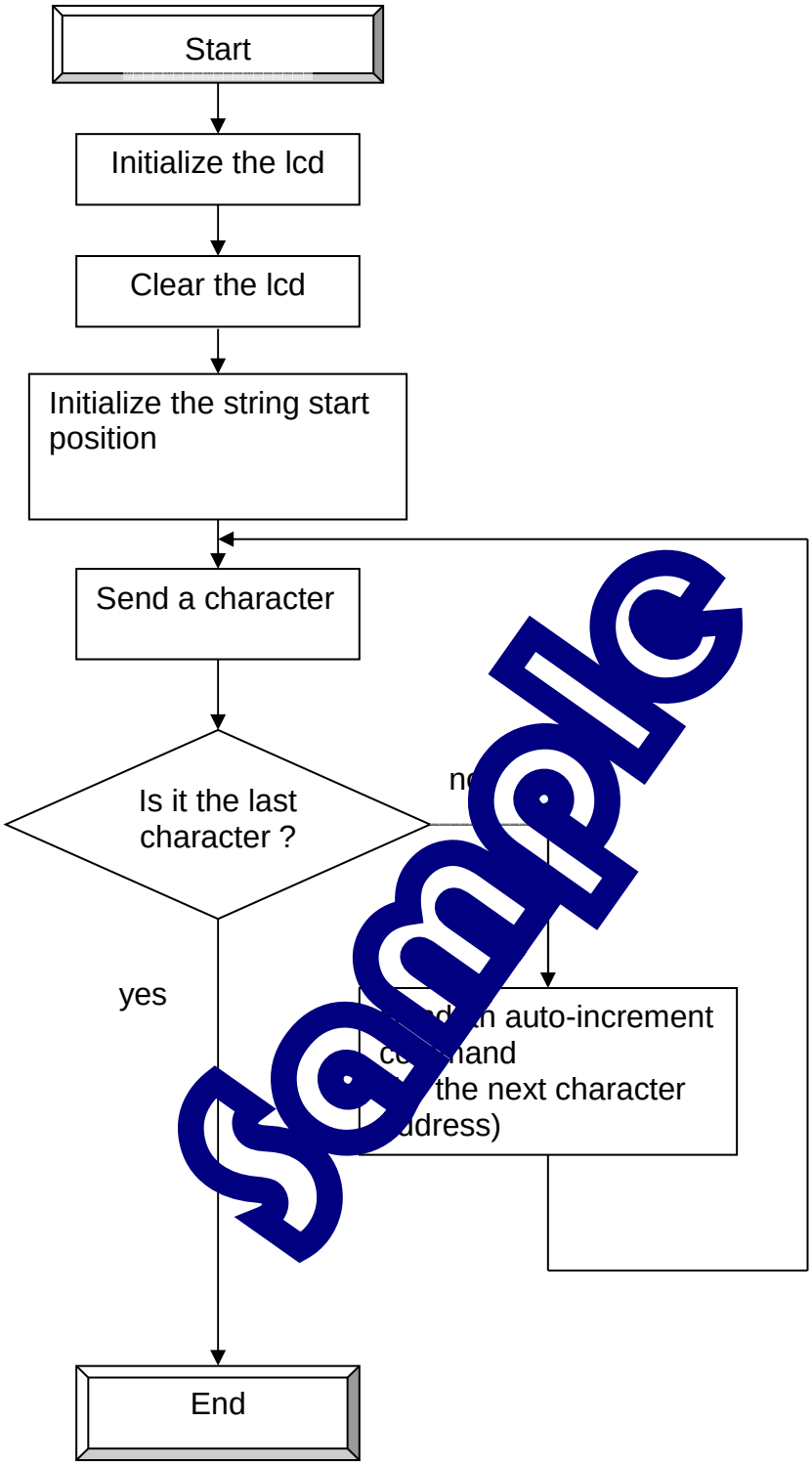
2.2.1.4 Main flowchart

You have the following functions with their comments.

Functions	Comments
void init_aff()	Parameters TH, TA, GH GA Mode initialization
void lcd_cls()	Clear the screen
void lcd_write_command(unsigned char **commande**)	Command writing
void lcd_write_data(unsigned char **data**)	Data writing
void lcd_gotoxy(unsigned char **px**, unsigned char **py**)	Definition of a character or a pixel position; px and py are the lower and upper data.
void lcd_out_str(char *texte)	Sending a string by the *texte variable

Important :

To use their functions as well as the board display with C / C + + compiler, we must configure the eid210 software linker.
To do this: Go to the configuration menu * click GNU C / C + +, then go to the * linker * click * add * tab, and then select the "EID005_Lib.o" file and at last click open.


Start
Initialize the lcd
Clear the lcd
Initialize the string start position
Send a character
Is it the last character ?
no
yes
auto-increment
the next character
End

2.2.2C Program

```
/*******************************************************
*     PRACTICAL WORKS ON EID005 BOARD                  *
*******************************************************
*     Write a C program                               *
*     which realize the display                        *
*     of a string.                                    *
*     the maximum size of string is: 128 characters   *
*******************************************************
*     File Name: EID005_TP2.c                         *
*     **************                                   *
*******************************************************/

//    Inclusion of definition files

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


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

unsigned char Texte []= { "EID005000 :     KEY  ,        DISPLAY,
      REAL TIME CLOCK$"};

// Spaces in the text can avoid cutt  wor

//============================
//    MAIN FUNCTION
//============================

main()
{
int i ;                          haracter counter variable
init_aff();                 //   Display initialization
lcd_cls();                       Clear the screen
lcd_gotoxy(0x10,00);        /    Definition of the string start position
//    Character number co te  loop

for (i=0; Texte [i]!= 0x24; i++)    // End the counter by detecting the
character "$"
{
lcd_write_data((unsigned char) (Texte[i]-0x20));     // Send a character
lcd_write_command(0xC0);     // Send the command of address incrementing
}
}

//    End of the program
```

EX.3 : READING A MATRIX KEYBOARD ON POLLING MODE

3.1 Topic

Purposes :	Be able to use the utilities stored in the library, allowing the 16-key-matrix (4x4) keyboard management.
Specifications:	**Subject** Write a program in C and in Assembler which realizes the reading of each pressed key from the 16-key-matrix keyboard on polling mode. This reading will start by detecting the activation of a pressed key.

Necessary Equipment :

PC Micro-Computer using Windows 95 or latter,
68332 Micro-Controller 16/32 bits Mother Board, Ref: EID 210 001
Keyboard-Display-Real Time Clock board: EID005001
Network connection cable and RS232 cable, Ref. : EGD 000 003
AC/AC 8V Power Supply, 1 A, Ref. : EGD000001,

Necessary Document :

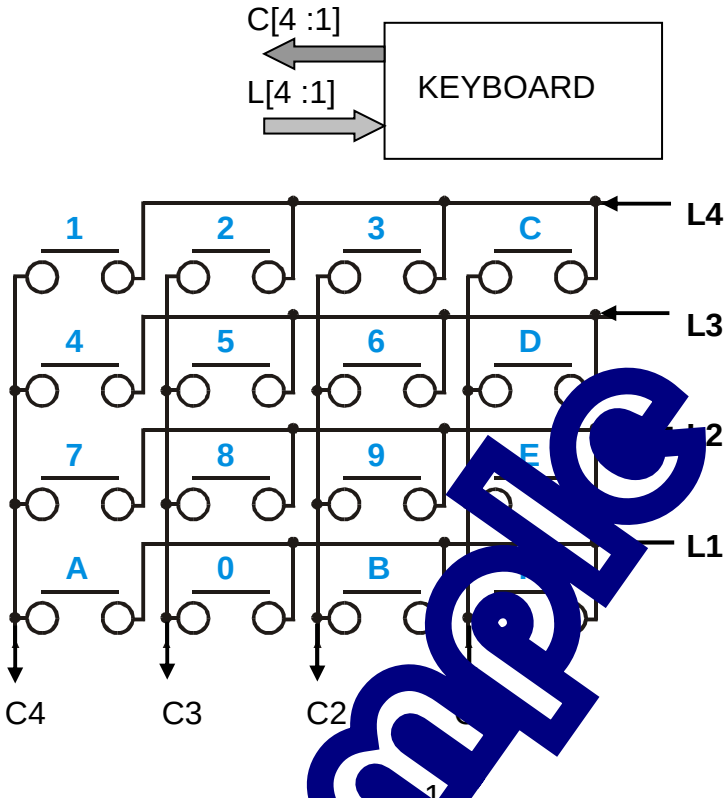
DMS Keyboard-Display-Real Time Clock board document: EID00500
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Time : 3 hours

3.2 Analysis and solution

3.2.1 Keyboard brief description

3.2.1.1 4x4 Keyboard representation fig.1



The 4 lines of keyboard are wired on the EPLD output for EID005 management.
Each EPLD output is provided with a pull-up resistor (Vcc drawing).

The 4 columns are wired on the inputs.

3.2.1.2 General principle of 16-key-matrix keyboard reading

When the outputs controlling the 4 lines are at 1 logic level, the 4 bits corresponding to 4 columns are set at 1 no matter what number of pressed keys is.
The reading principle by the polling method is to fix line Lj at 0, and to set the other three lines (scanning lines) at 1; then to set the column Ci number at 0 (scanning columns).

The $L_j x C_i$ intersection give the corresponding character.

It only remains to encode the character to give the wanted binary or hexadecimal value.

3.2.1.3 Pressed key reading on polling mode

To do this reading, we must :

a) Scan the line form “i” line to “0” line
b) Read the columns
c) Check whether a column is at 0 (at least 1 key pressed)
d) Determine the column value
e) Extract the key value
f) Display the key value on the computer screen.
g) Check all the keys.

3.2.1.4 Main flowchart

The keyboard is encoded by a 4x4 matrix in hexade[illegible]s shown below.

Pay attention to the lines position in the cod[illegible]

Column value	8	[illegible]	2	1
C_I → L_J ↓	C_4	C_3	C_2	C_1
L4	01	0[illegible]	03	0C
L3	04	[illegible]5	06	0D
L2	07	08	09	0E
L1	[illegible]	00	0B	0F

The lines and colum[illegible]s are d[illegible]ibed in the eid005.h file of the EID005 board.

The corresponding portion is below.

```
/* keyboard */
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))
```

« i » line is defined by the variable :

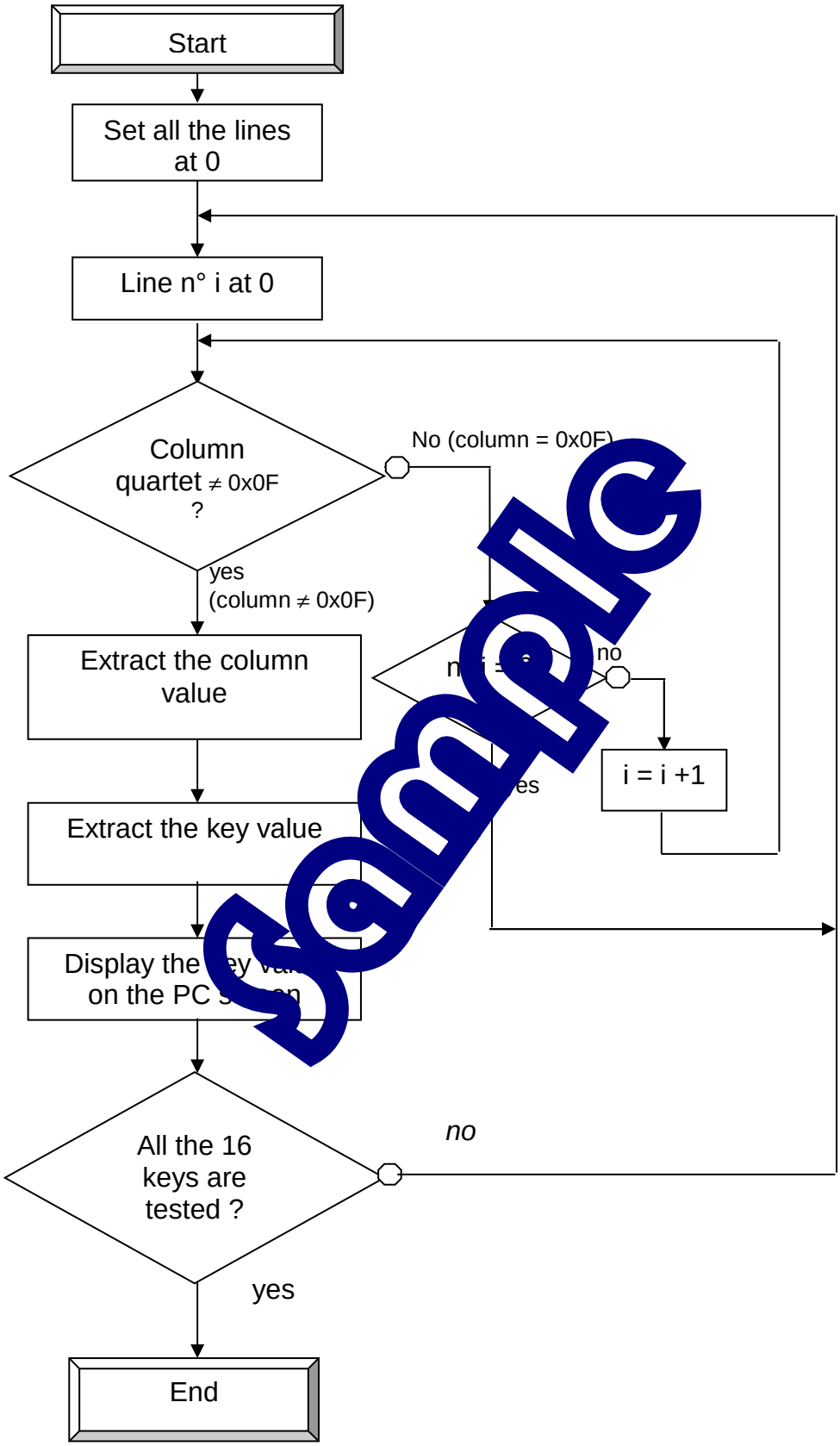
- clavier.matrice.ligne = i ; (0 =< i =< 3)

The columns are read through the variab

- clavier.matrice.colonne.

Important :

To use the Keyboard-Display with the C/C++ compiler, we must, in the eid210 software menu, go into the configuration menu then into the GNU C/C++ menu. Then go into the *linker* tab, click on *add*, select the « EID005_Lib.o » file and finish by clicking *open*.


Start
Set all the lines at 0
Line n° i at 0
Column quartet ≠ 0x0F ?
No (column = 0x0F)
yes (column ≠ 0x0F)
Extract the column value
no
i = i +1
Extract the key value
Display the
on the PC
All the 16 keys are tested ?
no
yes
End

3.2.2 C Program

```
/*******************************************************
*     PRACTICAL WORKS ON THE EID005 BOARD                *
*******************************************************
*     Write a program in C and in Assembler          *
*     which realizes the reading of every pressed key from *
*     the 16-key-matrix (4x4)keyboard on polling mode.          *
*******************************************************
*     FILE NAME: EID005_TP3.c            *
*     **************                                 *
*******************************************************/

//    Inclusion of definition files

#include "eid005.h"

//============================
//    MAIN FUNCTION
//============================


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

// Preparation for the display of the L

init_aff();             // Display initialization
lcd_cls();              // Clear the screen
lcd_gotoxy(0x30,00);    // Definition of the string start position
                        //


// Keyboard line scanning with giving successively the value to the
variable
// clavier.matrice.ligne, the quartets : 1110, 1101, 1011, 0111 .

printf ("\n-------------------------\n");
printf ("TEST ALL THE KEYS\n");
printf ("------------------------ \n\n");

j=0;
do
{j++;
printf ("Press a key\n");

//    Key is pressed ?
//    Line scanning

while ((clavier.matrice.colonne & 0x0F) == 0x0F)     // Wait for the
pressed key
{
      for (i = 0; i<4; i++ )
      {
```

```
            clavier.matrice.ligne = ~(1 << i);  // All the columns
                                                // are set at 1 except the
first column
            tmp = i;
            if (touche == 1) break;             // key = 1 ==>
                                                // at least one key is
pressed
            for (k = 0 ; k<10000 ; k++);        // Delay
      }
}

/*    Extract the column value:
      clavier.matrice.colonne = column element (4 bits)
       of the belonging structure
       the keyboard variable in  the union reg_clavier.
      Then extract the pressed key value given by
      the column value.
      This value is given by the variable:
      clavier.matrice.colonne & 0xF
*/

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

}

// Display the pressed key value on the screen

printf ("The pressed key is: %x\n", ValeurTouche );

// Display the pressed key value on the LCD screen

if(ValeurTouche < 0x0A)     // hexa --> ASCII Conversion : statistics
V = ValeurTouche+0x30;
else
V = ValeurTouche+0x1        // hexa --> ASCII Conversion : letters
lcd_write_data(V);          // Character sending to LCD
lcd_write_command(0xC0);    // Increment for the next character position


while (touche)              // Wait for the key released
for (k = 0 ; k<10000 ; k++); // Delay

}
//    All keys are tested?
while (j<16);
}
// End of the program
```

3.2.3 Assembler Program

```
*******************************************************
*     PRACTICAL WORKS ON THE EID005 BOARD                *
*******************************************************
*     Write a program in C and in Assembler          *
*     which realizes the reading of every pressed key from *
*     the 16-key-matrix (4x4)keyboard on polling mode.          *
*******************************************************
*     FILE NAME: EID005_TP3.src                  *
*     **************                             *
*******************************************************


**    The command or data to be written are firstly stored in D0
**    The read data is stored in D0
**    The character x, y position is placed respectively in D0 and D1
**    The string start address is placed in A0

*****************************************************
*     Display control register bits
*     ctrlaff :   b0=rd                             *
*                 b1=wr                             *
*                 b2=ce                             *
*                 b3=cd                             *
*                 b4=fs                             *
*****************************************************

* Inclusion of the file defining th      ent   abels
* of the EID200

      include EID210.def
      section var



*     Definition of t     la     the keyboard physical addresses

eid005      equ   $B3    000      clavier_afficheur_rtc Board basic address
ctrlaff     equ   eid0     2      Display control register : control bus
dbaff       equ   eid005+4        Data display bus
status      equ   eid005+6    * Board status register
reg_ctrl    equ   eid005+7    * Board control register
reg_clavier equ   EID005+5

*     Display parameter definition table


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  Graph or Text
Pointeur    equ   $24               * Command pointer
DispMod     equ   $94               * Text and/or Graphic Display
AutoInc     equ   $C0               * Auto increment pixel or character

Pile        equ   $802000           * Context protection address

init_acces  equ   $EF   * To access the data bus lcd with fs = 0
```

```
wrc         equ   $E9   * cd=1, ce=0, wr=0, rd=1, fs=0 1110 1001  write
command
rdc         equ   $EA   * cd=1, ce=0, wr=1, rd=0, fs=0 1110 1010  read
command (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

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


 **   Keyboard key value table

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

**    Keyboard key display coordinates table
                  dc.b  $1A,$3A,$5A,$7A
                  dc.b  $18,$38,$58,$78
                  dc.b  $16,$36,$56,$76
                  dc.b  $14,$34,$54,$74



      section           code

*******************************
*     MAIN PROGRAM      *
*******************************

      bsr         init_aff    * is[illegible]itialization
      bsr         lcd_cls     * c[illegible] screen

* Envoi Texte1 : x=0, y=0
      move.b      #$00,D0     * [illegible]e TH : line 0 column 1
      clr.b       D1          * MS[illegible]te TH
      bsr         lcd_go
      move.l      #Te[illegible]1,A0
      bsr         lcd_out[illegible]tr

*****************************[illegible]*********************

*  Definition of keyboard keys display start

      move.l      #15,D7
      clr.b       D1

* Keyboard reading

test_clav
      bsr         Touch
      bsr         Val_Touch   * The key value is in D0
      cmp.b       #$0A,D0     * Hexa --> ASCII Conversion,
      bcc         sup10       * see character generator T6963C :
      add.b       #$10,D0     * add $10 if code< $0A
      bra         envoi       *
sup10 add.b       #$17,D0     * otherwise add $17
envoi bsr         wr_data
```

```
      move.b      #$C0,D0
      bsr         wr_comm
      dbeq        D7,test_clav

* Texte2 : x=7, y=0

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

* Texte3 : x=7, y=d

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

*=================================================================*
      JMP         MONITEUR    *     Return to t
*=================================================================*

*******************************
*     END OF MAIN PROGRAM     *
*******************************

*******************************
*     THE SUB-PROGRAM   *
*******************************

**    SUB-PROGRAMS DISPLAY MANAGEMENT    *
*****************************************

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

busy  move.b      #in
      move.b      #rd
      bsr         delay
      move.b      dbaff,D
      and.b       #03,D4            * Data bus reading
      cmp.b       #03,D4            * Isolate ST0 STA1 (Status)
      bne         busy              * If lcd not ready
      move.b      #init_acces,ctrlaff   * wait
      rts

***    wrcomm  :  Writing Command located in D0 ***
*****************************************************

wr_comm
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Put the command on the data bus
      move.b      #wrc,ctrlaff            * Generate a writing command pulse
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    wrdata  :  Writing data located in D0 ***
```

```
************************************************

wr_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Put the data on the data bus
      move.b      #wrd,ctrlaff            * Generate a writing data  pulse
                                          *
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    rddata  :  Reading data located in D0 ***
************************************************

rd_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      #rdd,ctrlaff            * Generate a writing data  pulse
                                          *
      bsr         delay
      move.b      dbaff,D0                * Read the data and put it in D0
      move.b      #init_acces,ctrlaff
      rts

***    init_aff  : Display initialization **
****************************************

init_aff

* Init Text start address zone

      clr.b       D0
      bsr         wr_data     * LSByte TH = 00
      bsr         wr_data     * MSByte TH = 00
      move.b      #$40,D0                 command TH
      bsr         wr_comm     * Writing TH
      move.b      #$10,D0                 TA = $10 characters / line number
      bsr         wr_data
      clr.b       D0                      Byte TA = 0
      bsr         wr_data
      move.b      #$41,D0                 Writing command TA
      bsr         wr_comm                 Writing  TA

* Init Graphic start address zone

      clr.b       D0
      bsr         wr_data     * LSByte GH = 00
      move.b      #04,D0      * MSByte GH = 00
      bsr         wr_data     * Writing GH
      move.b      #$42,D0     * Writing command TH
      bsr         wr_comm
      move.b      #$10,D0     * LSByte GA = $10 characters / line number
      bsr         wr_data
      clr.b       D0          * MSByte GA = 0
      bsr         wr_data
      move.b      #$43,D0     * Writing command TA
      bsr         wr_comm     * Writing command TH

*  Modes Definition
```

```
      move.b      #DispMod,D0 * Getting Graphic AND/OR Text mode
      bsr         wr_comm
      move.b      #ModSet,D0
      bsr         wr_comm     * Getting Graphic OR Text mode
      rts


***    lcd_gotoxy  : Init character position    ***
************************************************

lcd_gotoxy
      bsr         wr_data     * Write the data LSByte content in D0
      move.b      D1,D0       * MSByte in D0
      bsr         wr_data     * Write the data MSByte content in D0
      move.b      #Pointeur,D0 * Pointer control code in D0
      bsr         wr_comm     * Write the pointer command in D0
      rts

***    lcd_cls  : Clear the screen  ***
****************************************
lcd_cls
      clr.b       D0          *  00,00 Address [illegible] D1
      clr.b       D1
      bsr         lcd_gotoxy  * Position [illegible] character
      move.w      #2048,D3    * Screen memor[illegible] number
      clr.b       D0
suite bsr         wr_data
      move.b      #AutoInc,D0 *  Auto i[illegible]ode in D0
      bsr         wr_comm     * Incremen[illegible]aracter pointer
      move.b      #0,D0
      sub.w       #1,D3
      bne         suite       * C[illegible] clear the screen
      rts

***    lcd_out_str : String s[illegible]
****************************[illegible]***************

** Attention: we must [illegible] 20 value to character ASCII code
lcd_out_str
      move.b      (A0[illegible],D0       Pointer a character
      cmp.b       #'$',D0     Is this the end of string?(detection of '$')
      beq         fin         End if the character is '$'
      sub.b       #$20,D0     * Because of the ASCII code, see the
instruction
      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


*****************************************
*     Sub program keyboard Management     *
*****************************************

**    Detection of a key pressed on polling mode      **
**    keyboard register test : reg_clavier            **
*******************************************************
Touch
      move.b      #$EF,D1           * Line 3
      move.w      #3,D2             * N° line
col_i move.b      D1,reg_clavier    * Pointer the column
      move.b      reg_clavier,D0    * Read the line
      move.l      #$4000,D4
      bsr         tempo

      and.b       #$0F,D0
      cmp.b       #$0F,D0           * Key pressed for this column ?
      beq         balayer           * No pressed key for this column
      bsr         Lacher            * Go to test if the key is released
      rts

balayer
      rol.b       #1,D1             * Next li
      sub.w       #1,D2             * n° ext
      bge         col_i
      bra         Touch

Lacher                              * Wa    r releasing the key
      move.b      reg_clavier,D3
      bsr         tempo
      and.b       #$0F,D3           * Isolate the column quartet
      cmp.b       #$0F,D3           *  isn't released?
      bne         Lacher            * Read again these columns if the key is
always pressed
      rts                           * If not, return.

**    Calculate the value of the pressed key    **
*****************************************************

Val_Touch
      and.l       #$F,D2            * Isolate the line n°
      move.l      #Colonne0,A0      * Pointer on the 0 column
      cmp.b       #$0E,D0           * Is it the 0 column ?
      beq         Val_Colonne0      * If so, go reading the value
      move.l      #Colonne1,A0      * Pointer on the 1 column
      cmp.b       #$0D,D0           * Is it the 1 column ?
      beq         Val_Colonne1      * If so, go reading the value
      move.l      #Colonne2,A0      * Pointer on the 2 column
      cmp.b       #$0B,D0           * Is it the 2 column ?
      beq         Val_Colonne2      * If so, go reading the value
      move.l      #Colonne3,A0      * Pointer on the 3 column
      bra         Val_Colonne3      * If so, go reading the value

**    Display position management   **
****************************************

Val_Colonne0
      move.b      $10(A0,D2),D0           * Key Position
```

```
      clr.b       D1
      bsr         lcd_gotoxy
      move.b      0(A0,D2),D0
      rts

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

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

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

      End
```

EX.4 : KEY ACTIVATION DETECTION AND READING ON POLLING MODE

4.1 Topic

Purposes :	Be able to use the utilities stored in the library, allowing the 16-key-matrix (4x4) keyboard management.
Specifications:	**Subject** Write a program in C and in Assembler which realizes the reading of each pressed key from the 16-key-matrix keyboard on polling mode. This reading will be activated by the polling mode.

Necessary Equipment :

PC Micro-Computer using Windows® 95 or latter,
68332 Micro-Controller 16 bits Mother Board, Ref: EID 210 001
Keyboard-Display-Real Time Clock board: EID005001
Network connection cable and RS232 cable, Ref. : EGD 000 003
AC/AC 8V Power Supply, 1 A, Ref. : EGD000001,

Necessary Document :

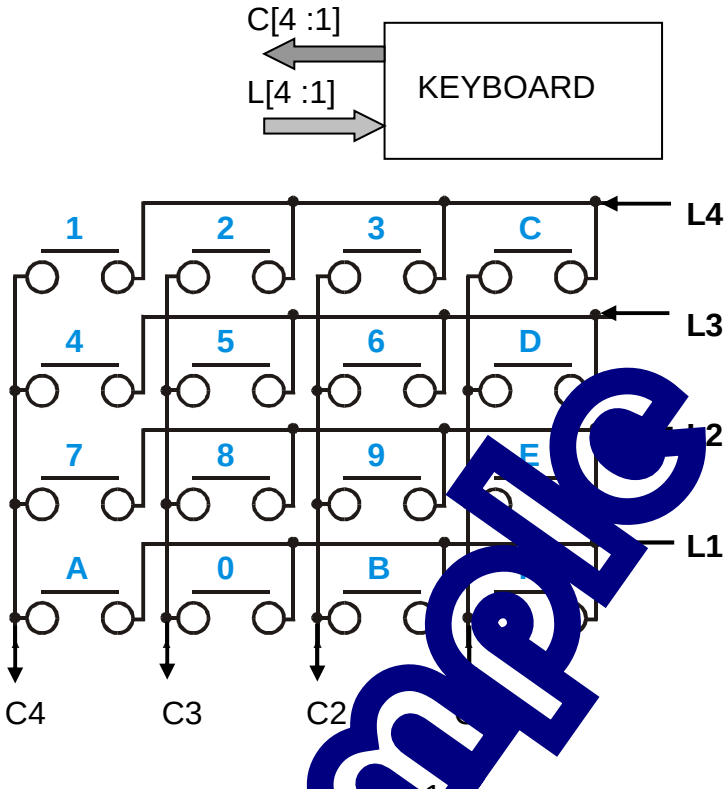
DMS Keyboard-Display-Real Time Clock board document: EID00500
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Time : 3 hours

4.2 Analysis and solution

4.2.1 Keyboard brief description

4.2.1.1 4x4 Keyboard representation fig.1



The 4 lines of keyboard are wired on the EPLD output for EID005 management.
Each EPLD output is provided with a pull-up resistor (Vcc drawing).

The 4 columns are wired on the inputs.

4.2.1.2 General principle of 16-key-matrix keyboard reading

When the outputs controlling the 4 lines are at 1 logic level, the 4 bits corresponding to 4 columns are set at 1 no matter what number of pressed keys is.
The reading principle by the polling method is to fix line Lj at 0, and to set the other three lines (scanning lines) at 1; then to set the column Ci number at 0 (scanning columns).

The $L_j \times C_i$ intersection give the corresponding character.

It only remains to encode the character to give the wanted binary or hexadecimal value.

4.2.1.3 Pressed key activation detection

This detection is done through the ETAT_CLAVIER (0 bit) bit of the EID005 board STATUS register.

This STATUS_CLAVIER bit is set at 1 once one key is pressed.

The « `touche` » variable is defined in the eid005.h file like this:

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

4.2.1.4 Main flowchart

The keyboard is encoded by a 4x4 matrix in hexadecimal as shown below.

Pay attention to the lines position in the code table.

Column value	8			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

The lines and columns are defined in the eid005.h file of the EID005 board.

The corresponding portion is below.

```
/* keyboard */
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))
```

« i » line is defined by the variable :

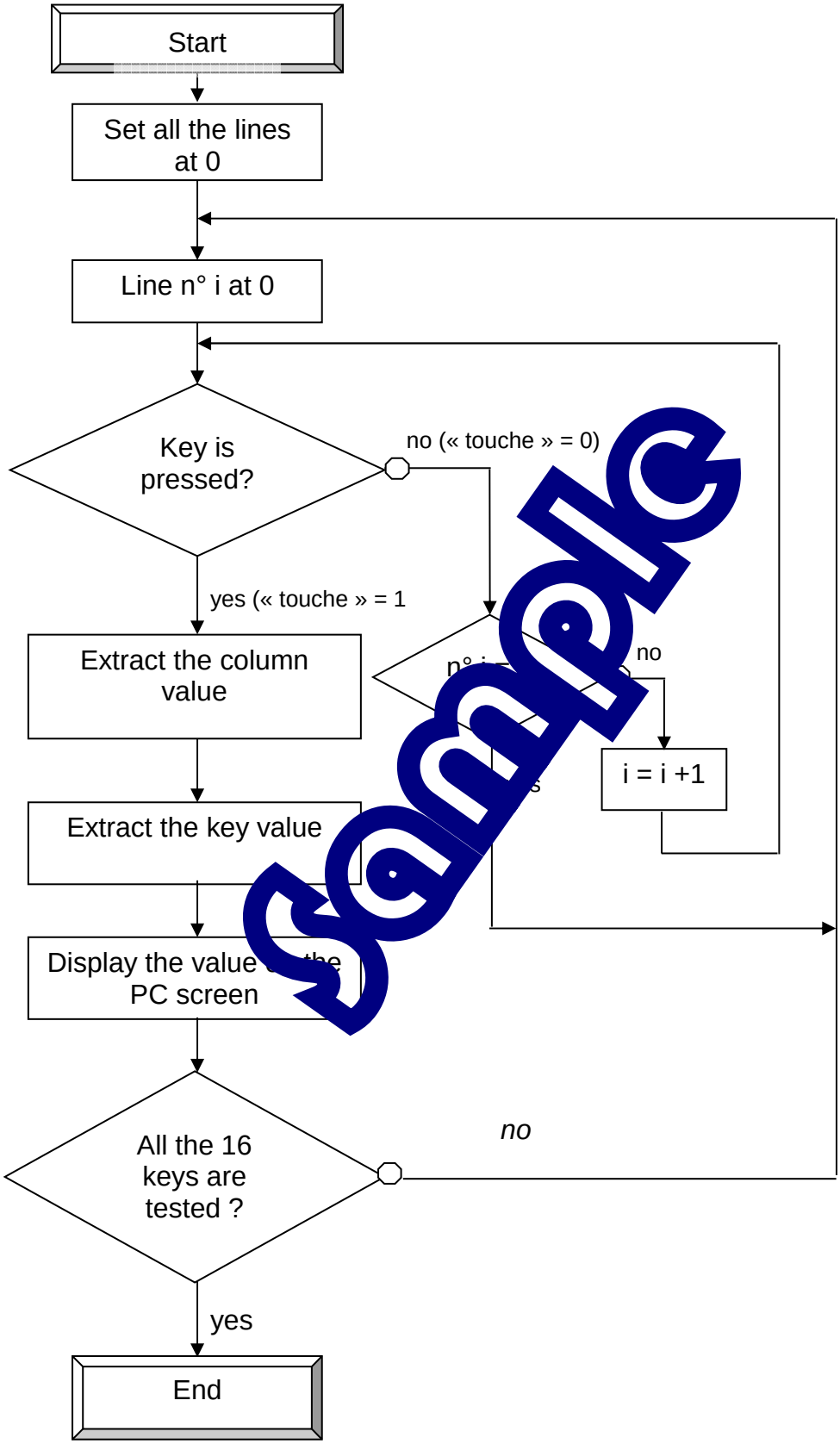
- clavier.matrice.ligne = i ; (0 =< i =< 3)

The columns are read through the variab

- clavier.matrice.colonne.

Important:

To use the Keyboard-Display with the C/C++ compiler, we must, in the eid210 software menu, go into the configuration menu then into the GNU C/C++ menu. Then go into the *linker* tab, click on *add*, select the « EID005_Lib.o » file and finish by clicking *open*.


Start
Set all the lines at 0
Line n° i at 0
Key is pressed?
no (« touche » = 0)
yes (« touche » = 1
Extract the column value
no
i = i +1
Extract the key value
Display the value
PC screen
All the 16 keys are tested ?
no
yes
End

4.2.2 C Program

```
/*******************************************************
*     PRACTICAL WORKS ON THE EID005 BOARD                *
*******************************************************
*     Write a C program which realizes          *
*     the reading of a key pressed on the 16-key-matrix *
*     (4x4) keyboard on polling mode by detecting a key activation.*
*******************************************************
*     FILE NAME: EID005_TP4.c             *
*     **************                              *
*******************************************************/


//    Inclusion of definition files

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

//============================
//   MAIN FUNCTION
//============================


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

// Preparation for the display on the LCD

init_aff();             // Display initialization
lcd_cls();              // Clear the screen
lcd_gotoxy(0x30,00);    // Definition of the string start position

//    Keyboard line scanning with giving successively the value to the
variable
//    clavier.matrice.ligne, the quartets : 1110, 1101, 1011, 0111.

printf ("\n------------------------\n");
printf ("TEST ALL THE KEYS\n");
printf ("------------------------ \n\n");

j=0;
do
{j++;
 printf ("Press on a key\n");
//    Key is pressed?
//    Line scanningwhile (!touche)

while ((clavier.matrice.colonne & 0x0F) == 0x0F)     // Wait for the
pressed key
{
      for (i = 0; i<4; i++ )
      {
```

```
            clavier.matrice.ligne = ~(1 << i); // All the columns are set
at 1 except the first column
            tmp = i;
            if (touche == 1) break;            // key = 1 ==> at least one
key is pressed
for (k = 0 ; k<10000 ; k++);       // delay
      }
}
/*    Extract the column value:
      clavier.matrice.colonne = column element (4 bits)
       of the belonging structure
       the keyboard variable in  the union reg_clavier.
      Then extract the pressed key value given by
      the column value.
      This value is given by the variable:
      clavier.matrice.colonne & 0xF
*/
switch((~(clavier.matrice.colonne)) & 0x0F )
{
      case 1 : ValeurTouche= Touche [tmp][3]; break;       //Column value 1
      case 2 : ValeurTouche= Touche [tmp][2]; break;       //Column value 2
      case 4 : ValeurTouche= Touche [tmp][1]; break;       //Column value 4
      case 8 : ValeurTouche= Touche [tmp][0]; break;       //Column value 8

}

//    Display the pressed key value on the PC screen

printf ("The pressed key is:%x\n\n", ValeurTouche );

// Display the pressed key value on the LCD screen

if(ValeurTouche < 0x0A) // hexa -> ASCII conversion : statistics
N= ValeurTouche+0x10;
else
N= ValeurTouche+0x17;   // hexa -> ASCII Conversion : letters
lcd_write_data(N);      // Characters sending to LCD
lcd_write_command(0xC0); // Displacement for the next character position

while (touche)                     // Wait for the key released
for (k = 0 ; k<10000 ; k++); // delay
}

//    All keys are tested?
while (j<16);
}
```

4.2.3Assembler Program

```
*******************************************************
*     PRACTICAL WORKS ON THE EID005 BOARD                *
*******************************************************
*     Write a program in C and in Assembler          *
*     which realizes the reading of every pressed key from *
*     the 16-key-matrix (4x4)keyboard on polling mode.          *
*******************************************************
*     FILE NAME: EID005_TP3.src                  *
*     **************                             *
*******************************************************


**    The command or data to be written are firstly stored in D0
**    The read data is stored in D0
**    The character x, y position is placed respectively in D0 and D1
**    The string start address is placed in A0

*************************************************
*           Display control register bits
*     ctrlaff :   b0=rd                          *
*                 b1=wr                          *
*                 b2=ce                          *
*                 b3=cd                          *
*                 b4=fs                          *
*************************************************

* Inclusion of the file defining th      abels
* of the EID200

      include EID210.def

      section var


*     Definition of t      the keyboard physical addresses

eid005      equ   $B3      clavier_afficheur_rtc Board basic address
ctrlaff     equ   eid0     Display control register : control bus
dbaff       equ   eid005+  Data display bus
status      equ   eid005+6    * Board status register
reg_ctrl    equ   eid005+7    * Board control register
reg_clavier equ   EID005+5

*     Display parameter definition table


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  Graph or Text
Pointeur    equ   $24               * Command pointer
DispMod     equ   $94               * Text and/or Graphic Display
AutoInc     equ   $C0               * Auto increment pixel or character

Pile        equ   $802000           * Context protection address

init_acces  equ   $EF   * To access the data bus lcd with fs = 0
```

```
wrc         equ   $E9   * cd=1, ce=0, wr=0, rd=1, fs=0 1110 1001  write
command
rdc         equ   $EA   * cd=1, ce=0, wr=1, rd=0, fs=0 1110 1010  read
command (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

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


**    Keyboard key value table

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

**    Keyboard key display coordinates table
                  dc.b  $1A,$3A,$5A,$7A
                  dc.b  $18,$38,$58,$78
                  dc.b  $16,$36,$56,$76
                  dc.b  $14,$34,$54,$74



      section           code


*******************************
*     MAIN PROGRAM      *
*******************************

      bsr         init_aff    * Display initialization
      bsr         lcd_cls       e screen

* Envoi Texte1 : x=0, y=0
      move.b      #$0,D       * LSByte TH : line 0, column 1
      clr.b       D1          * MSByte TH
      bsr         lcd_xytoxy
      move.l      #Texte1,D0
      bsr         lcd_out_fr

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

*  Definition of keyboard keys display start

      move.l      #15,D7
      clr.b       D1

* Keyboard reading

test_clav
      bsr         Touch
      bsr         Val_Touch   * The key value is in D0
      cmp.b       #$0A,D0     * Hexa --> ASCII Conversion,
      bcc         sup10       * see character generator T6963C :
      add.b       #$10,D0     * add $10 if code< $0A
      bra         envoi       *
sup10 add.b       #$17,D0     * otherwise add $17
```

```
envoi bsr         wr_data
      move.b      #$C0,D0
      bsr         wr_comm
      dbeq        D7,test_clav


* Texte2 : x=7, y=0

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

* Texte3 : x=7, y=d

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

*=======================================================*
      JMP         MONITEUR    *     Return to the monitor
*=======================================================*

*******************************
*     END OF MAIN PROGRAM     *
*******************************

*******************************
*     THE SUB-PROGRAM   *
*******************************

**    SUB-PROGRAMS DISPLAY MANAGEMENT     **
*****************************************

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

busy  move.b      #init_acces,ctrlaff
      move.b      #rdc,ctrlaff
      bsr         delay
      move.b      dbaff,D4          * Data bus reading
      and.b       #03,D4            * Isolate ST0 STA1 (Status)
      cmp.b       #03,D4            * If lcd not ready
      bne         busy              * wait
      move.b      #init_acces,ctrlaff
      rts

***    wrcomm  :  Writing Command located in D0 ***
*************************************************

wr_comm
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Put the command on the data bus
      move.b      #wrc,ctrlaff            * Generate a writing command pulse
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts
```

```
***    wrdata  :  Writing data located in D0 ***
************************************************

wr_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      D0,dbaff                * Put the data on the data bus
      move.b      #wrd,ctrlaff            * Generate a writing data  pulse
                                          *
      bsr         delay
      move.b      #init_acces,ctrlaff
      rts

***    rddata  :  Reading data located in D0 ***
************************************************

rd_data
      bsr         busy
      move.b      #init_acces,ctrlaff     * fs = 0; cd, ce, wr et rd = 1
      move.b      #rdd,ctrlaff            * Gene      riting data  pulse
                                          *
      bsr         delay
      move.b      dbaff,D0                R     he d    a and put it in D0
      move.b      #init_acces,ctrlaff
      rts

***    init_aff  : Display initializati
****************************************      *******

init_aff

* Init Text start address zone

      clr.b       D0
      bsr         wr_data           H = 00
      bsr         wr_data     * MSB     TH = 00
      move.b      #$40,D0           ng command TH
      bsr         wr_com            ing TH
      move.b      #$1,D             Byte TA = $10 characters / line number
      bsr         wr_data
      clr.b       D0          * MSByte TA = 0
      bsr         wr_data
      move.b      #$41,D0     * Writing command TA
      bsr         wr_comm     * Writing  TA


* Init Graphic start address zone

      clr.b       D0
      bsr         wr_data     * LSByte GH = 00
      move.b      #04,D0      * MSByte GH = 00
      bsr         wr_data     * Writing GH
      move.b      #$42,D0     * Writing command TH
      bsr         wr_comm
      move.b      #$10,D0     * LSByte GA = $10 characters / line number
      bsr         wr_data
      clr.b       D0          * MSByte GA = 0
      bsr         wr_data
      move.b      #$43,D0     * Writing command TA
      bsr         wr_comm     * Writing command TH
```

```
*  Modes Definition

      move.b      #DispMod,D0 * Getting Graphic AND/OR Text mode
      bsr         wr_comm
      move.b      #ModSet,D0
      bsr         wr_comm     * Getting Graphic OR Text mode
      rts


***    lcd_gotoxy  : Init character position    ***
************************************************

lcd_gotoxy
      bsr         wr_data     * Write the data LSByte content in D0
      move.b      D1,D0       * MSByte in D0
      bsr         wr_data     * Write the data MSByte content in D0
      move.b      #Pointeur,D0 * Pointer control code in D0
      bsr         wr_comm     * Write the pointer command in D0
      rts

***    lcd_cls  : Clear the screen  ***
***************************************

lcd_cls
      clr.b       D0          * 00,00 Add[illegible] D0 and D1
      clr.b       D1
      bsr         lcd_gotoxy  * Positio[illegible]e first character
      move.w      #2048,D3    * Screen m[illegible]tes number
      clr.b       D0
suite bsr         wr_data
      move.b      #AutoInc,D0 * [illegible]ent Code in D0
      bsr         wr_comm     * [illegible]he character pointer
      move.b      #0,D0
      sub.w       #1,D3
      bne         suite       * [illegible]e to clear the screen
      rts

***    lcd_out_str : [illegible]   ***
************************************

** Attention: we must sub[illegible] the $ 20 value to character ASCII code
lcd_out_str
      move.b      (A0)+,D0    * Pointer a character
      cmp.b       #'$',D0     * Is this the end of string?(detection of '$')
      beq         fin         * End if the character is '$'
      sub.b       #$20,D0     * Because of the ASCII code, see the
instruction
      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


*****************************************
*     Sub program keyboard Management     *
*****************************************

**    Detection of a key pressed on polling mode      **
**    Keyboard register test: reg_clavier       **
************************************************************
Touch
      move.b      #$EF,D1           * Line 3
      move.w      #3,D2             * N° line
col_i move.b      D1,reg_clavier    * Pointer the column
      move.b      reg_clavier,D0    * Read the line
      move.l      #$4000,D4
      bsr         tempo

      and.b       #$0F,D0
      cmp.b       #$0F,D0           * Key re   for  is column ?
      beq         balayer           * No pr   se    or this column
      bsr         Lacher            * Go to   i  he key is released
      rts

balayer
      rol.b       #1,D1             * Ne   li
      sub.w       #1,D2             N    l
      bge         col_i
      bra         Touch

Lacher                              *   t for releasing the key
      move.b      reg_clavie
      bsr         tempo
      and.b       #$0F  3           * Isolate the column quartet
      cmp.b       #$0 ,D            * Key isn't released?
      bne         La er             * Read again these columns if the key is
always pressed
      rts                           * If not, return.

**    Calculate the value o  the pressed key    **
************************************************

Val_Touch
      and.l       #$F,D2            * Isolate the line n°
      move.l      #Colonne0,A0      * Pointer on the 0 column
      cmp.b       #$0E,D0           * Is it the 0 column ?
      beq         Val_Colonne0      * If so, go reading the value
      move.l      #Colonne1,A0      * Pointer on the 1 column
      cmp.b       #$0D,D0           * Is it the 1 column ?
      beq         Val_Colonne1      * If so, go reading the value
      move.l      #Colonne2,A0      * Pointer on the 2 column
      cmp.b       #$0B,D0           * Is it the 2 column ?
      beq         Val_Colonne2      * If so, go reading the value
      move.l      #Colonne3,A0      * Pointer on the 3 column
      bra         Val_Colonne3      * If so, go reading the value

**    Display position management   **
```

```
*****************************************

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

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

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

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

      end
```

EX.5 : LINES, CIRCLES AND CURVES DRAWING ON THE LCD

5.1 Topic

Purposes :	Be able to use the utilities stored in the library, allowing the straight and the circle drawing on the LCD screen.
Specifications:	**Subject** Write a C program that will re[illegible]ertical, horizontal and oblique straights, and c[illegible]

Necessary Equipment :

PC Micro-Computer using Windows® [illegible] latter,
68332 Micro-Controller [illegible] Board, Ref: EID 210 001
Keyboard-Display-Real [illegible]rd: EID005001
Network connection cable and RS232 cable, Ref. : EGD 000 003
AC/AC 8V Power Supply, [illegible] : EGD000001,

Necessary Document :

DMS Keyboard-Display-Real Time Clock board document: EID00500
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Time : 4 hours

5.2 Analysis and solution

5.2.1 Display description on graphic mode

Attention:

Because of the regulation in the manufacturer's documentation, the x and y variables represent respectively the ordinate (vertical) and the abscissa (horizontal). The point “x = 0, y = 0” is at the left top of the LCD, while the point “x = 63, y = 127” is at the right bottom of the LCD.

The T6963C controller has an 8 kB memory.

Graphic Mode

The graphic zone is located in the address from 0004[illegible] + the most significant byte of the GH parameter).

The LCD on graphic mode contains 64 lines and 128 points per line.
There are: 128x64 = 8192 points which are from 0 to 8191.

A point is notably defined by the byte number in which it is located.

So the 8192 points are defined in 1024 bytes (8192/8).

A pixel of x, y coordinates is for the number Nu = 128*x + y.

Its address “Adr” in the display memory (VRAM) is determined by the bytes’ integer number in the Nu number; its position is completely defined with the knowledge of point number in the byte.

- **Adr = integer part (Nu / 8); Adr is on 2 bytes.**

r = reste(remain) is the remainder of Nu divided by 8.

In a byte, a pixel is identified by its number coded on 3 bits.

- The least significant bit is turned on at the right of the byte on the line; so this is the 7th pixel in byte and the number is 7.
- The most significant bit is turned on at the left of the byte on the line; so this is the 1st pixel in byte and the number is 0.

The point number to turn on is np = 7- r.

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

Example fig.1

The pixel is located in : **x = 18, y = 91**
The point number is Nu = 18x128 + 91 = 2395 ; Nu / 8 = 299,375
Adr = 299 = 0x012B
r = Nu – 299*8 = 4
np = 7 – 4 = 3
This is the GH (Graphic Home Address) parameter bytes:
GH Address lower = 0x2B **GH Address upper = 0x01 + 0x04.**

The **GH Address lower**, **GH Address upper** and **np** parameters are calculated every time once pixel should be turned on or off.

POINT GRAPHIC POSITION IN THE PLAN ON GRAPHIC MODE

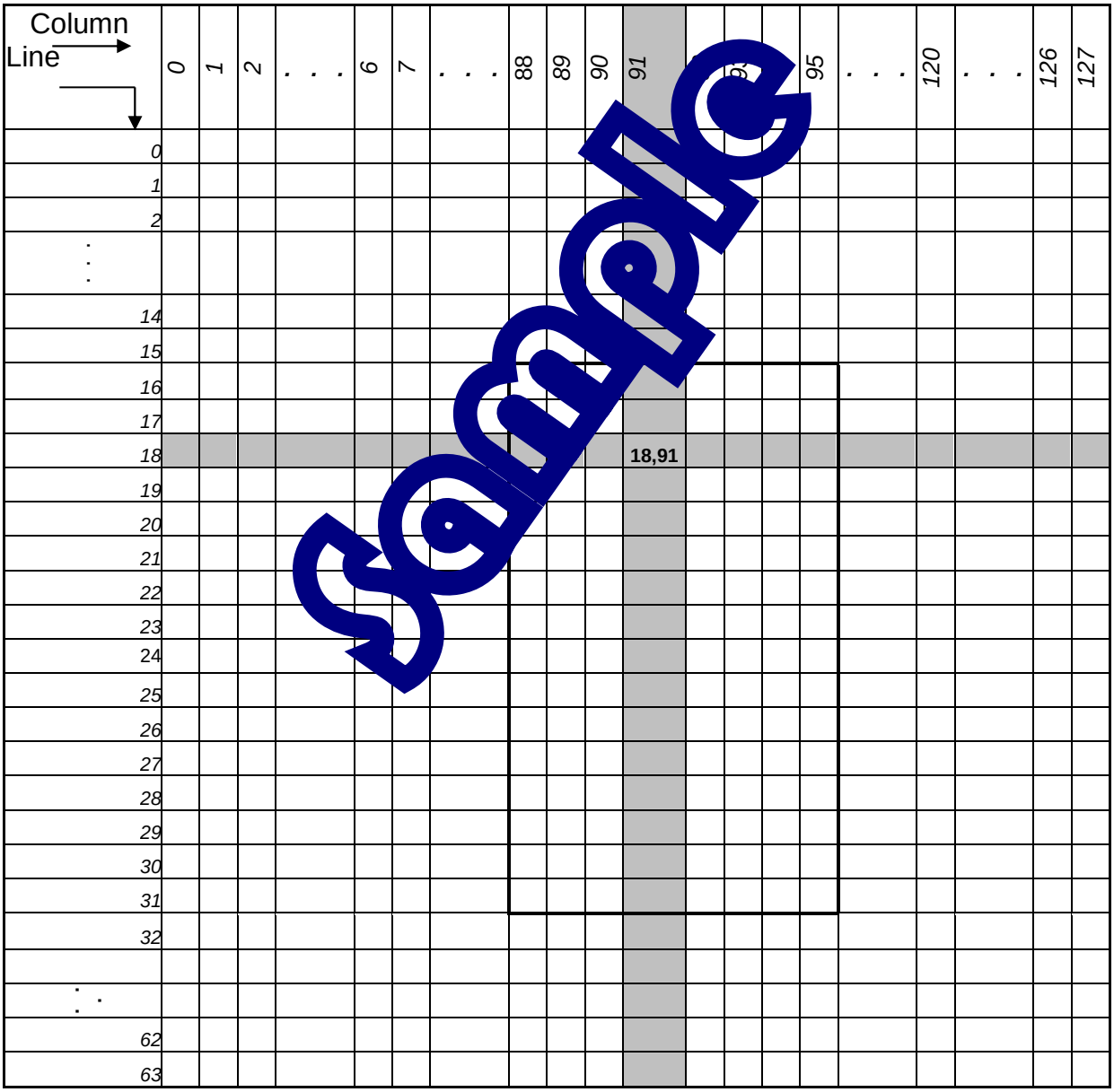


fig.1

5.2.2 Main program

We have the function **void Tracer_Pixel (int x, int y, unsigned char Pen)** to draw the curves by developing your own algorithm or the following function with their comments to achieve the same result.

The 0xF8 and 0xF0 commands can turn on or turn off the number np pixel.
This gives the definition of the variable “Pen”:

Pen = 0xF8 + np ➔ ignition of number np pixel.
Pen = 0xF0 + np ➔ extinction of number np pixel.

Function	Comment
`void init_aff()`	Initialization of parameters : TH, TA, GH, GA, Mode …
`void lcd_cls()`	Clear the screen
`void lcd_write_command(unsigned char commande)`	Write a command
`void lcd_write_data(unsigned char data)`	Write a data
`void lcd_clear_TXT()`	Clear the text screen
`void lcd_clear_Graph()`	Clear the graphic screen
`void lcd_gotoxy(unsigned char px, unsigned char py)`	Set the position of a character or pixel ; px and py are the lower and upper data. Remark: for a pixel, we must add 0x04 to py.
`void lcd_out_str(char *texte)`	Send a string pointed by the variable *text*
`void Tracer_Pixel(int x, int y, unsigned char Pen)`	Pen = 0xF8 ➔ turn on the pixel Pen = 0xF0 ➔ turn off the pixel
`void Tracer_LH ( unsigned char M, unsigned char N, unsigned char P, unsigned char Q, unsigned char Pen )`	M, N : coordinates of the starting point P,Q : coordinates of the ending point
`void Tracer_LV ( unsigned char M, unsigned char N, unsigned char P, unsigned char Q, unsigned char Pen )`	The same
`void Droite ( int x1, int y1, int x2, int y2, int Pen )`	The same

Function	Comment

`void Cercle_NPoints ( int x0, int y0, int r, unsigned int Pen, unsigned int N)`	`x0, y0 : center coordinates`
	`r : radius (points number)`
	`N : points number on the circle`
	`Pen = 0xF8 ➔ turn on the pixel`
	`Pen = 0xF0 ➔ turn off the pixel`

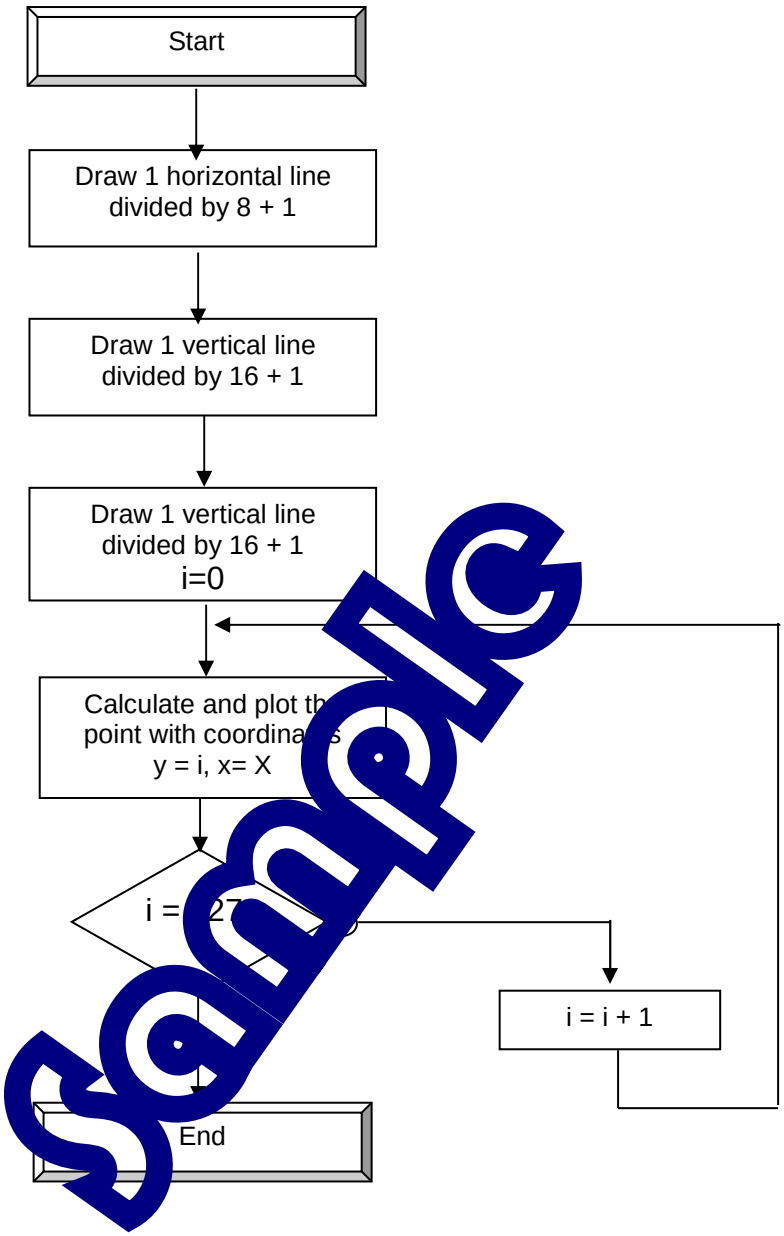
5.2.3 Flowchart of a sinusoid drawing on an oscilloscope

Traced by points: Flowchart 1

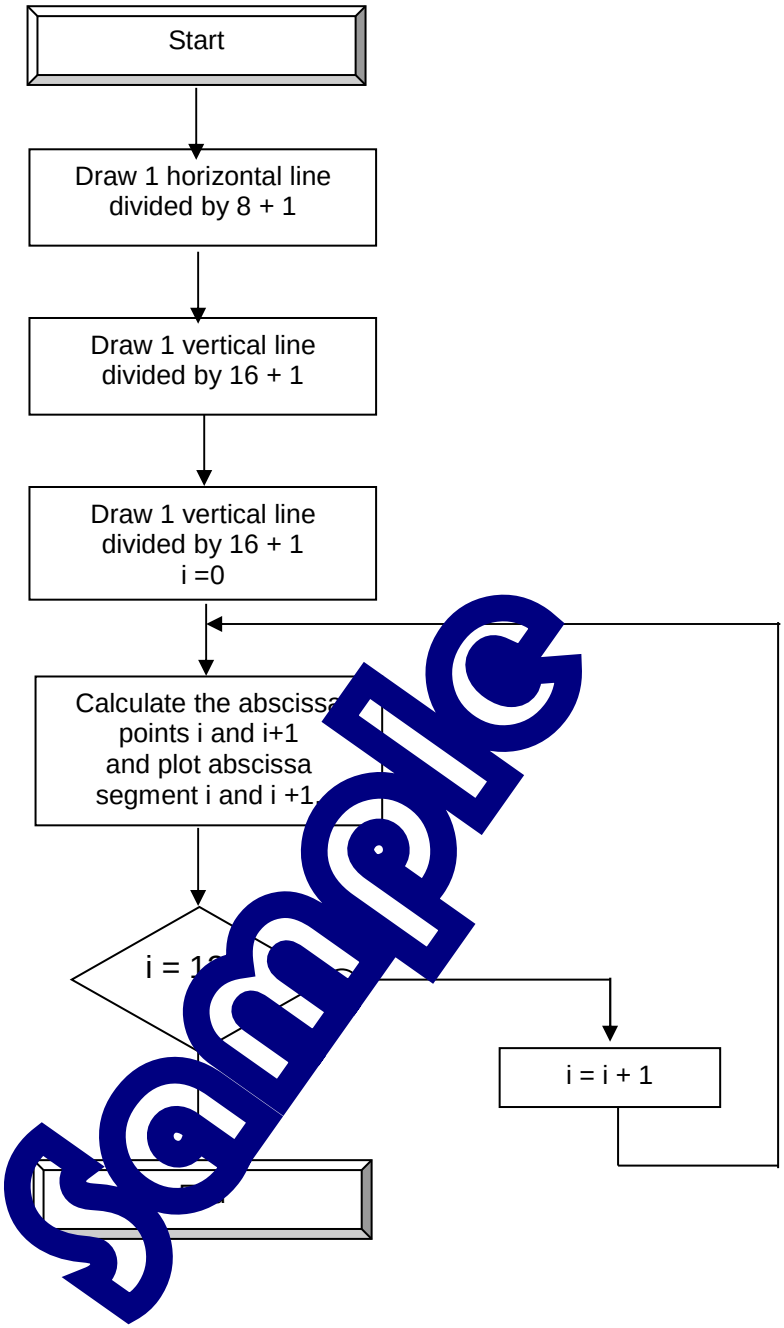
The basic angle: pi/N with (example N = 30).
The principle is to calculate X = k*sin (i*pi/N) where k represent the maximum amplitude of the sinusoid.
The point to plot is for the coordinates: y = i and x = integer part of X.
abscissa: x ordinate: y

Traced by line segments: Flowchart 2

The method is the same except that it is necessary to calculate the coordinates of two consecutive points to draw the line segments connecting the two points.



Flowchart 1



Flowchart 2

5.2.4 C Program

```
/*****************************************************
*     PRACTICAL WORKS ON THE EID005 BOARD                    *
*****************************************************
*     Write a C program that will realize the vertical,     *
*     horizontal and oblique straights, and circles         *
*****************************************************
*     FILE NAME: EID005_TP5.c                               *
*     **************                                        *
*****************************************************/

//    Inclusion of definition files

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


//============================
//   MAIN FUNCTION
//============================

main()
{
double  pi = 3.14159265 ;
int i, N, m1, m2, k=28; //    k : max[illegible]mplitude of sin(i*pi/N)

int PB = 0xF8 ;   //    begin : tu[illegible]a pixel
int PL = 0xF0 ;   //    end : tu[illegible]off [illegible]pixel


init_aff();       //    Disp[illegible]ization
lcd_cls();        //    Cle[illegible]the s[illegible]en
Tab_Sin_Cos ();   //    Cal[illegible]ne calculation on 360 ° by 1 °
                  //    [illegible]he c[illegible]cle drawing

/*********************[illegible]*****************************************
****
*     Example: Draw a rec[illegible]e at the edge of the display, of these
diagonals,*
*     and of the two axes of horizontal and vertical symmetry    *
*****************************************************************************
***/

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(32,63,28,PB, 60);

delay (500000);
lcd_clear_Graph();                  // Clear the graph

Rectangle(10,20,40,60,PB);
Triangle(15,25,55,50,25,100,PB);
delay (500000);
```

```
lcd_clear_Graph();

// Draw a sinusoid by the POINTS method on an oscilloscope with the
function
// void Tracer_Pixel(int x, int y, unsigned char Pen)

for (i=0; i<64; i +=8 )            // 8 grid horizontal lines
      Tracer_LH(i,0,i,127,PB);
      Tracer_LH(63,0,63,127,PB);
for (i=0; i<127; i +=16 )          // 8 grid vertical lines
      Tracer_LV(0,i,63,i,PB);
      Tracer_LV(0,127,63,127,PB);

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

{
N = 40;
m1 = k*sin(i*(pi/N));
Tracer_Pixel(32-m1,i,PB);     // Trace by POINTS
}

delay (500000);
lcd_clear_Graph();            // Clear the Graph


//  Draw a sinusoid by the POINTS method on an oscilloscope with the
function
// void Droite ( int x1, int y1,  int x2, int y2, int  Pen )

for (i=0; i<64; i +=8 )            // 8 grid horizontal lines
      Tracer_LH(i,0,i,127,PB);
      Tracer_LH(63,0,63,127,PB);
for (i=0; i<127; i +=16 )          // 8 grid vertical lines
      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/N));
      m2 =k*sin((i+1)*(pi/N));
      Droite(32-m1 , i, 32-m2, i+1, PB);
}


}

//    End of the program
```

EX.6 : A CLOCK DRAWING ON THEGRAPHIC SCREEN

6.1 Topic

Purposes :	Be able to use the utilities stored in the library, allowing the 128x64 pixels graphic LCD display management.
Specifications:	**Subject** Write a C program wh[illegible] re[illegible]s the clock drawing and time, date and year display on [illegible] LC

Ne[illegible] Equipment :

PC Micro-Computer usi[illegible] Wi[illegible]lo[illegible]s or latter,
68332 Micro-Controller [illegible]6/[illegible] [illegible]er Board, Ref: EID 210 001
Keyboard-Display-Real [illegible]ime Cl[illegible]k board: EID005001
Network connection cable a[illegible] R[illegible]232 cable, Ref. : EGD 000 003
AC/AC 8V Power Supply, 1 A, [illegible]f. : EGD000001,

Necessary Document :

DMS Keyboard-Display-Real Time Clock board document: EID00500
Application Notes for the T6963C LCD Graphics Controller Chip (TOSHIBA)
T6963c DOT MATRIX LCD CONTROL LSI (TOSHIBA)

Time : 4 hours

6.2 Analysis and solution

6.2.1 Clock geometric definition

12 marks representing 12 hours are drawn on a circle with R radius.

Each mark is actually made of a square whose center is the circle with R radius and whose side is equal to 4 pixels.

The second circle is the same as hour's. Its radius is R' and the points are squares with their 2 pixels side, centered on the circle.

For the LCD management, we must r = refer to EX 5.

6.2.1.1 Circuit representation

The DS14285 circuit programmer model is described in fig. 1.

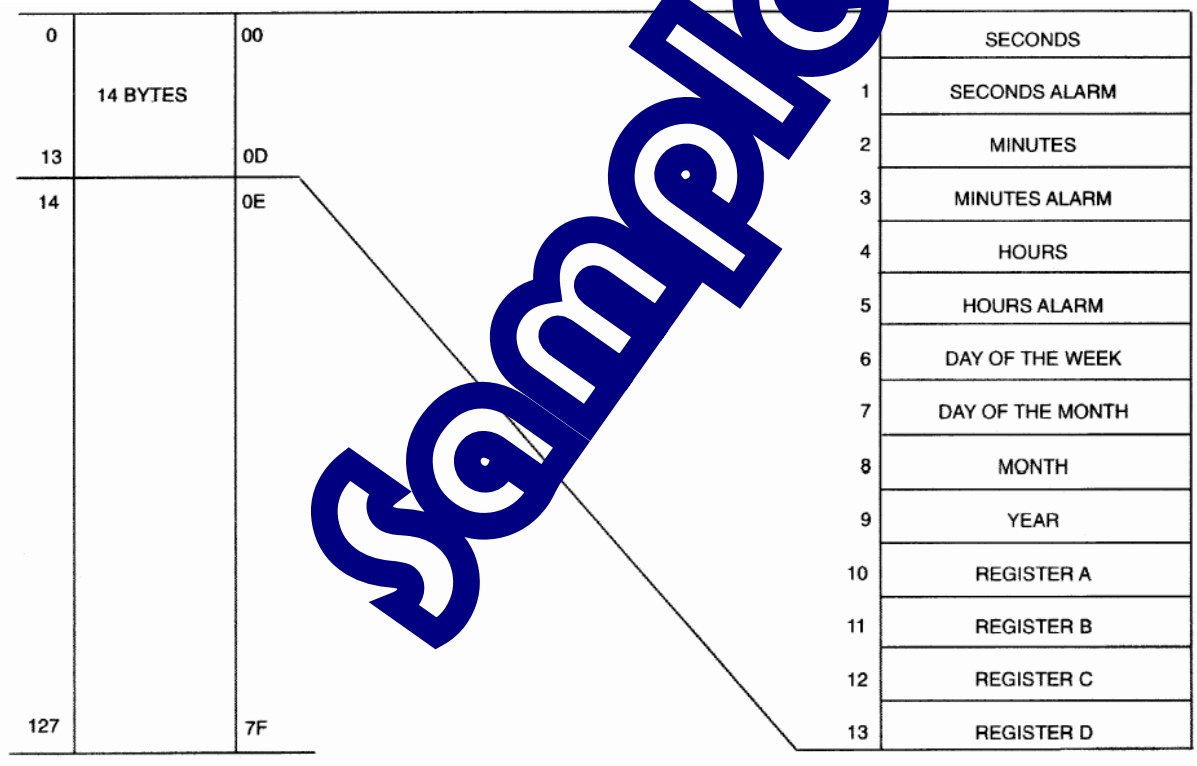


fig.1

The 0-13 Addresses define the location of the various programming or reading circuit registers.

6.2.1.2 RTC DS14285 circuit management

In this practical works, only two circuit registers (A and B controller) are programmed.

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 b[illegible]d. When the UIP bit is a 1, the update transfer will soon occur. When UIP is a 0, the u[illegible] t[illegible] st[illegible]ll[illegible]ot occur for at least 244 µs. The time, calendar, and alarm information in RAM is f[illegible] nva[illegible] for a[illegible]ss when the UIP bit is 0. The UIP bit is read-only and is not affected by $\overline{\text{RESET}}$. Writing[illegible] SE[illegible]t 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 o[illegible]il[illegible]or off and to reset the countdown chain. A pattern of 010 is the only combination of bits that w[illegible] tur[illegible] oscillator on and allow the RTC to keep time. A pattern of 11X will enable the oscillato[illegible]ut [illegible]he c[illegible]tdown chain in reset. The next update will occur at 500 ms after a pattern of 010 is w[illegible]te[illegible] D[illegible] and DV2.

RS3, RS2, RS1, RS0 - These four rate-se[illegible]ti[illegible]its [illegible]ect one of the 13 taps on the 15-stage divider or disable the divider output. The tap selecte[illegible]an [illegible]ed to generate an output square wave (SQW pin) and/or a periodic interrupt. The user [illegible] following:

1. Enable the interrupt with [illegible] PI[illegible]bi[illegible]
2. Enable the SQW outpu[illegible]pin[illegible]e SQ[illegible] bit;
3. Enable both at the sam[illegible]ime a[illegible]e rate; or
4. Enable neither.

Table 2 lists the periodic interrupt[illegible] 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 bit 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 SQW 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 $\overline{\text{RESET}}$.

6.2.2 Main program

The program is to enter the time, date, month and year in progress, then to check successively the data entry.

Then we must at first format the data in order to set the RTC circuit with the time, to allow it to count time and then display the information on the LCD screen.

Finally, we must read the RTC and display the data on the LCD every second. The second circle is cleared at the beginning of every minute.

6.2.3 Flowchart

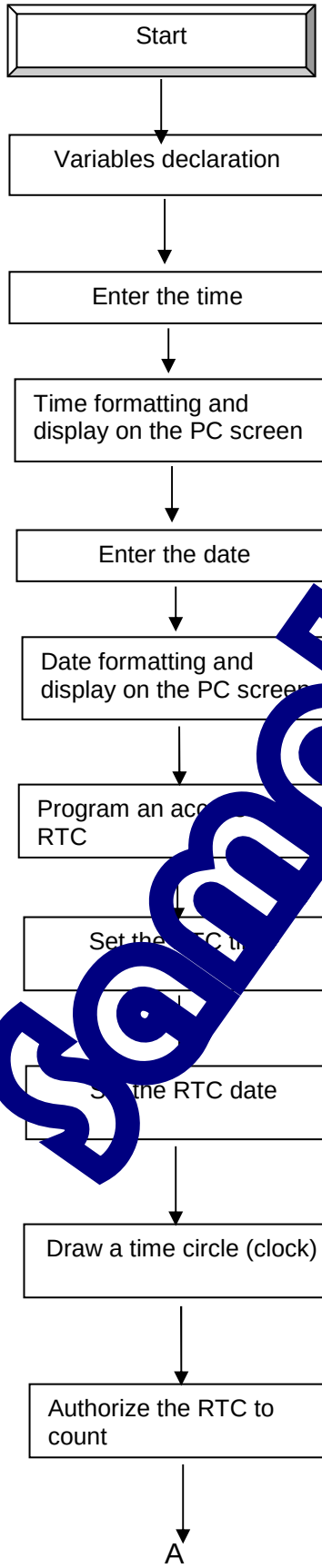
We have the function **void Tracer_Pixel (int x, int y, unsigned char Pen)** to draw the curves by developing your own algorithm or following function with their comments to achieve the same result.

The 0xF8 and 0xF0 commands can turn on or turn off a number np pixel.
This gives the definition of the variable "Pen":

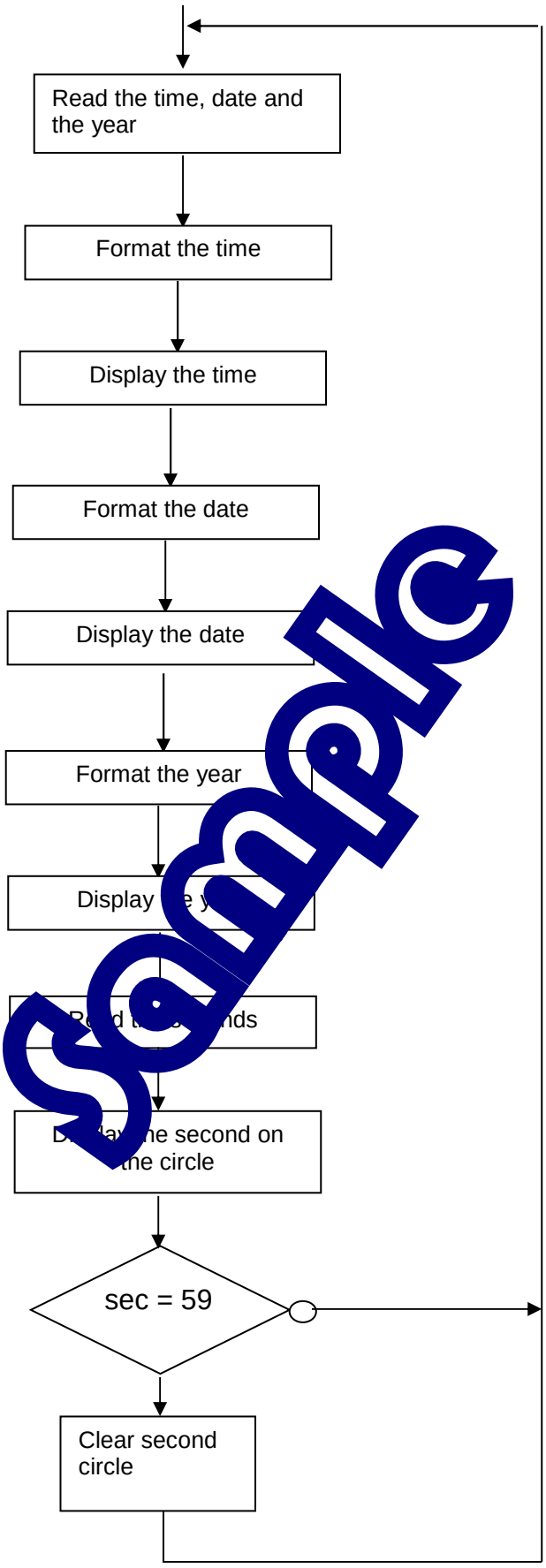
Pen = 0xF8 + np ➔ ignition of number np pixel.
Pen = 0xF0 + np ➔ extinction of number np pixel.

Function	Comment
void init_aff()	Initialization of parameters : TH, TA, GH, GA, Mode …
void lcd_cls()	Clear the screen
void lcd_write_command(unsigned char **commande**)	Write a command
void lcd_write_data(unsigned char data)	Write a data
void lcd_clear_TXT()	Clear the text screen
void lcd_clear_Graph()	Clear the graphic screen
void lcd_gotoxy(unsigned char **px**, unsigned char **py**)	Set the position of a character or pixel ; px and py are the lower and upper data. Remark: for every pixel, we must add 0x04 to py.
void lcd_out_str(char *texte)	Send a string pointed by the variable *text*
void Tracer_Pixel(int x, int y, unsigned char Pen)	Pen = 0xF8 ➔ turn on the pixel Pen = 0xF0 ➔ turn off the pixel
void Tracer_LH (unsigned char M, unsigned char N, unsigned char P, unsigned char Q, unsigned char Pen)	M, N : coordinates of the starting point P,Q : coordinates of the ending point
void Tracer_LV (unsigned char M, unsigned char N, unsigned char P, unsigned char Q, unsigned char Pen)	The same

Function	Comment
void Droite (int x1, int y1, int x2, int y2, int Pen)	The same
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 represents the register address in the RTC.
unsigned char readbyte(unsigned char adr)	The same


Start
Variables declaration
Enter the time
Time formatting and display on the PC screen
Enter the date
Date formatting and display on the PC screen
Program an acc
RTC
Set the
the RTC date
Draw a time circle (clock)
Authorize the RTC to count
A

A


Read the time, date and the year
Format the time
Display the time
Format the date
Display the date
Format the year
Display
nds
e second on
the circle
sec = 59
Clear second circle

6.2.4 C Program

```
/************************************************** ***
*     PRACTICAL WORKS ON THE EID005 BOARD             *
*****************************************************
*     A clock drawing with the time, the date         *
*     and the year displaying on the LCD              *
*                                                     *
*****************************************************
*     FILE NAME: EID005_TP6.c                         *
*     *************                                   *
*****************************************************/

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

/******************************/
/*    Global variables  */
/******************************/

/*
double Cos[60], Sin[60];  //  60 points Table representing the second
unsigned char X, Y, np ;  //  X , Y : a pixel position  in the plan [ x y ]
= [64 128]
                          //  np: n° of pixel on the X line
double  pi = 3.14159265 ;
double  deuxpi = 6.28318530 ;
//    K = Correction factor between the dimension separating 2 points in x
and in y;
      K = 1.4872 ;
*/

/******************************/
/*    Used Function           */
/******************************/

int PL = 0xF0 ;               //    End==> Clear
int PB = 0xF8 ;               //    Begin ==> Trace

main()
{
/******************************/
/* DECLARATION OF 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 ;

//    Input for updating the RTC
unsigned char Annee[4] =  { 0x2, 0x0, 0x0, 0x0 } ;   // Display the year
unsigned char DMA [7];              // Enter Date_Month_year
```

```
unsigned char Date_Mois[5] ;        // Display the date and the month
unsigned char Heure[6] ;            // Enter and display the time
unsigned char Heure_Alarme[3] ;     // Enter
unsigned char tmp;

char * Jours[7] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"};
char *_Mois[12]= {"Jan.", "Feb.", "Mar.", "Apr.", "May.", "Jun.", "Jul.",
"Aug.", "Sep.", "Oct.", "Nov.", "Dec."};

/*****************************/
/*****************************/
/*                           */
/*    MAIN PROGRAM           */
/*                           */
/*****************************/
/*****************************/

// calculate sine cosine 60 points: seconds

Tab_Sin_Cos ();   // To display hours and seconds points

// Display initialization

init_aff();
lcd_cls();

/*************************************/
/*    Enter the time or start reading*/
/*************************************/

/* Type :
            hh = hours then Enter
            mm = minutes then Enter
            ss = seconds then Enter.
*/


printf("    Type : h  to setting the time \n    or others to start \n");
scanf ("%c", &depart);
if (depart == 'h')

{     printf("    Enter the time: 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 ("                There still remain % d
statistic(s)\n", 5-u);
            }
      else
            {     u -= 2;           // To erase codes of Back Space and
Enter
                  printf ("\n  RECTIFICATION !\n");
            }
      }
```

```
/*********************************************/
/*    Format hhmmss : Table Heure[] */
/*********************************************/

      Heure[0] = 10*Heure[0] + Heure[1] ; // Hour
      Heure[1] = 10*Heure[2] + Heure[3] ; // Minute
      Heure[2] = 10*Heure[4] + Heure[5] ; // Second

//    Check the time entry
      printf ("    %d h %d mn %d s\t\n", Heure[0], Heure[1], Heure[2]);

/*****************************/
/*    Enter the date         */
/*****************************/

/* Type :
            j = day then Enter,
            dd = date then Enter,
            mm = month then Enter and
            aa = year then Enter.
*/

printf ("    Enter the date : jddmmaa \n");
for (u =0; u < 7; u++)
      {
      scanf("%c",&x);
      if (x != 0x08)
            {     if ( u <0)
                  u = 0;
                  DMA[u] = x & 0x0F;
                  printf("    \t\t%d \n",(x & 0x0F));
                  printf ("   The first %d statistic(s)\n", 6-u);
            }
      else
            {     u -= 2;          // To erase codes of Back Space and
Enter
                  printf ("\n CORRECTIFICATION !\n");
            }
      }

/*****************************************/
/*    Fortamat jddmm : Table DMA[]       */
/*****************************************/

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


//    Check the date entry

printf ("    %s %d %s 20%d  \n",Jours [DMA[0]-1],DMA[1], _Mois [DMA [2]-
1],DMA [3]);


/*********************************************/
/*    RTC circuit program: DS14285  */
/*********************************************/

//--- Set the RTC control register bit 0 at 1:  access to RTC
```

```
ctrl_rtc = 1 ;

//--------   A and B registers program

writebyte (REGB, 0x82); // B Register = 1000 0010
                        // Update authorization
                        // without incrementing counters (sec, mm, h).
writebyte (REGA, 0x20); // A Register  = 0010 0000 :

//--------  Load the time

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

/*--------  The alarm programming

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

//--------  The date programming

writebyte (6, DMA [0]);       // DAY : 1 = Sun, 2 = Mon, ...
writebyte (7, DMA [1]) ;      // Date in the month
writebyte (8, DMA [2]);       // Month of the year
writebyte (9, DMA [3]) ;      // Year
}
//--------  Authorization to start the RTC

printf ("    EID005 RTC in progress, press ESC to stop \n");

/*****************************/
/*    CLOCK DWARING     */
/*****************************

//-------   Draw hour circle ------------

for (u = 0; u <60; u+=
Cercle_H (32,63, 28,  u); // Clock darwing

Rectangle(0,0,63,127,PB); // Draw the rectangle around clock

//------- RTC Start: incrementing counters (sec, mm, h)

writebyte (REGB, 06);

/*********************************************/
/*    RTC reading and data displaying    */
/*********************************************/

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

//-----     Time reading

sc = readbyte(0);
```

```
mn = readbyte(2);
hr = readbyte(4);

//-----     Day, date, month and year reading

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


//-----     Formatting and Hour, minute, second display

      Heure[0]= hr / 10 ;
      Heure[1]= hr % 10 ;
      Heure[2]= 0x0a ;        // shifted ASCII code of the Display
      Heure[3]= mn / 10 ;
      Heure[4]= mn% 10 ;
      Heure[5]= 0x0a ;        // the same
      Heure[6]= sc / 10 ;
      Heure[7]= sc % 10 ;

lcd_gotoxy(0x55,0x0);
lcd_out_Tab(Heure, 5);

//-----     Formatting and day display

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

//-----     Formatting and date an  month

      Date_Mois[0]= date / 10 ;
      Date_Mois[1]= date % 10 ;

//  If the month digital di lay:
      Date_Mois[2]= 0xFF ;          shifted ASCII code of the Display-
0x10
      Date_Mois[3]= mois /  ;
      Date_Mois[4]= mois % 1 ;

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

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


//-----     Formatting and year display

Annee [2] = an / 10 ;
Annee [3] = an % 10 ;

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

//-----     Reading and second display

if (sc == 59)     // clear the seconds
{
      //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);

}

//----      End of the main program
```