Chapter 8 Interfacing a JHD12864J Graphic Module to AT89S52

Chapter 8 Interfacing a JHD12864J Graphic Module to AT89S52

8.1 Introduction

JHD12864J is a light weight, low power consumption liquid crystal graphic display. The module measures 54.0x50.0mm only. Supply voltage is 5V matching the voltage for most microcontrollers. The LCD controller is Samsung KS0108B. This chapter describes a firmware developed to drive the LCD.

An extract of the schematic is shown in Figure 8.1.1

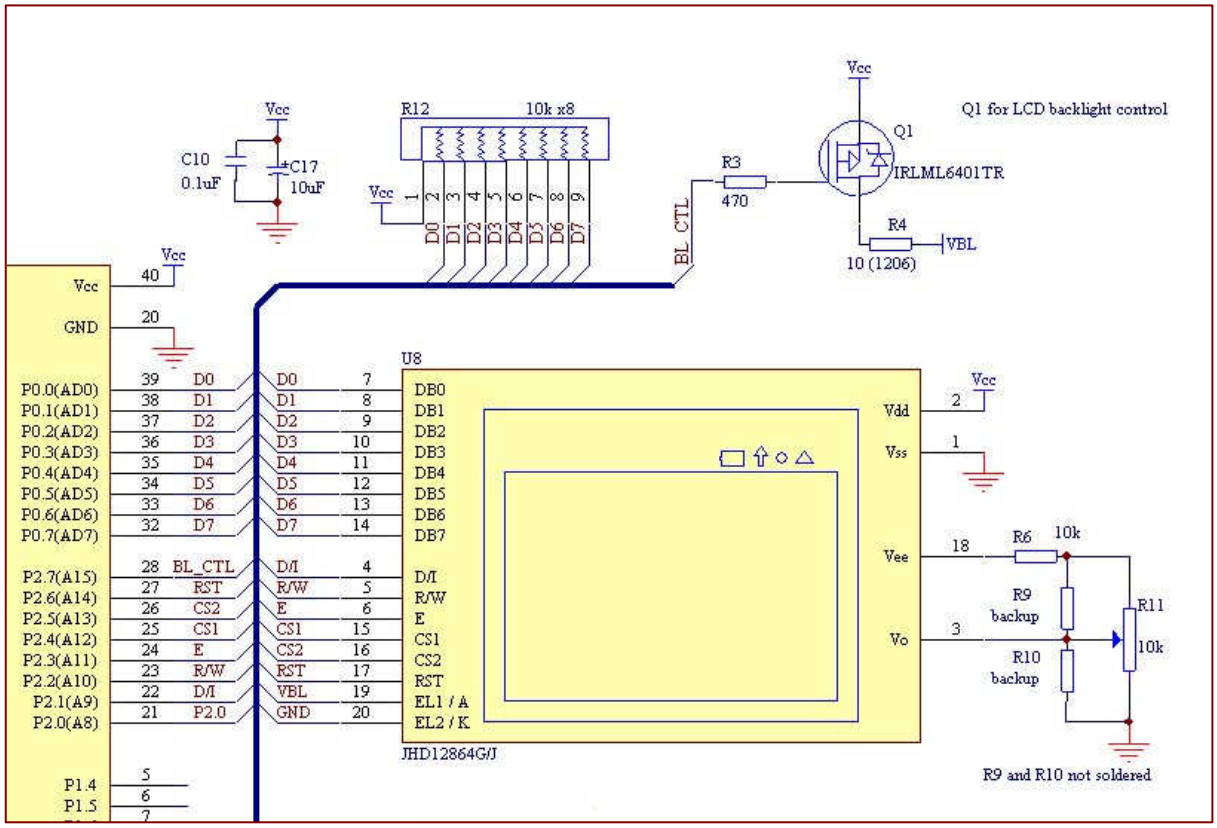


Figure 8.1.1 JHD12864J interface AT89S52 microcontroller

Unlike most character-based LCDs which use 4-bit data bus, JHD12864J module uses 8-bit data bus (DB0 – DB7). Nevertheless, it is a straight forward module comparing to other LCD series like T6963C[1]. JHD12864J is split logically in half with controller #1 (CS1) driving the left half of the display, and controller #2 (CS2) driving the right half. These two portions map directly to the physical display area as shown in Figure 8.1.2. With a correct controlling sequence on pin CSx (x=1,2), D/I, and R/W, we can write any pattern say, 0xAB directly to the LCD screen at a designated column position. The idea is illustrated in Figure 8.1.3.

[1] consult www.TechToys.com.hk -> Downloads section for an interface example with PIC16F877 microcontroller

Chapter 8 Interfacing a JHD12864J Graphic Module to AT89S52

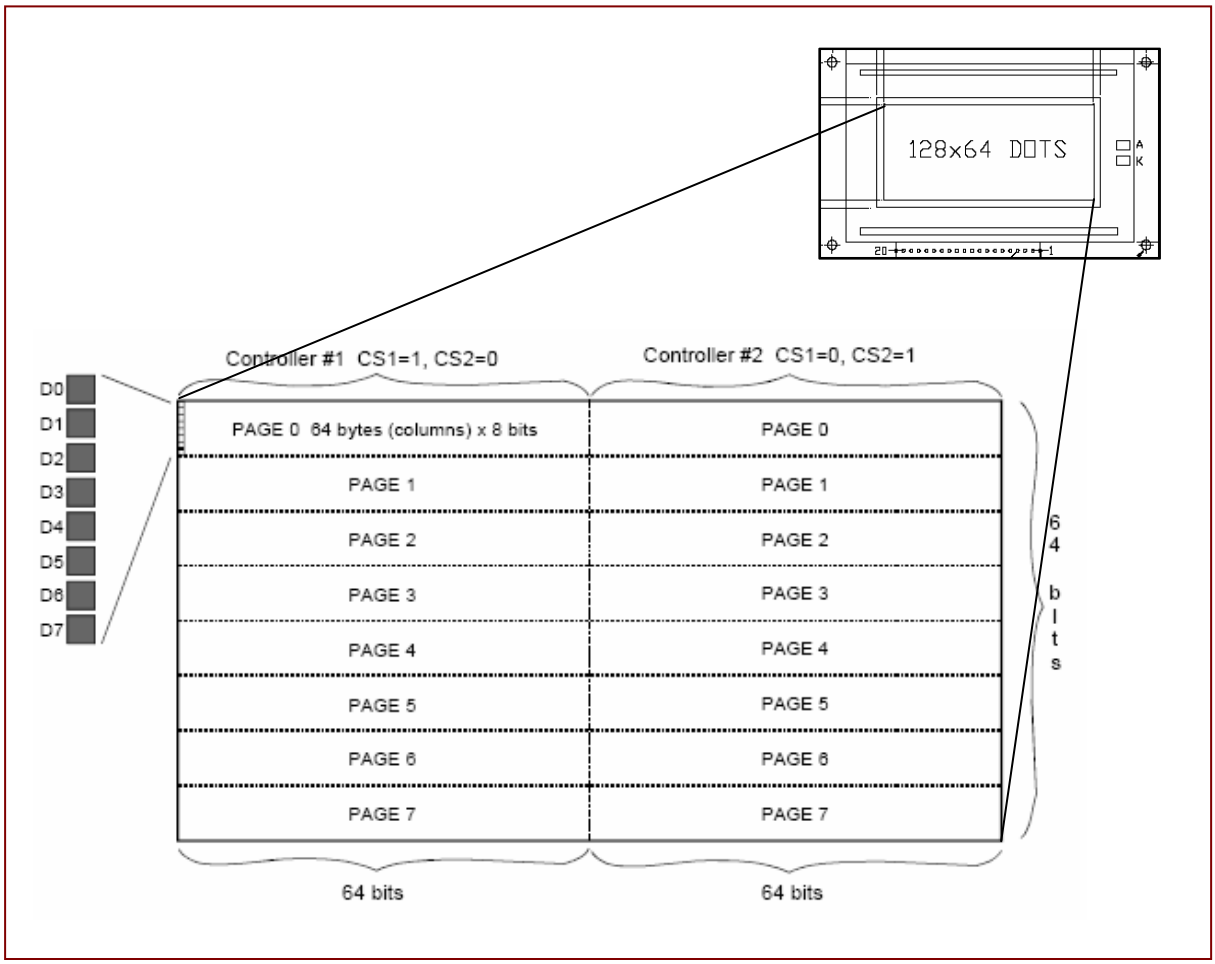


Figure 8.1.2 A map of LCD pixels

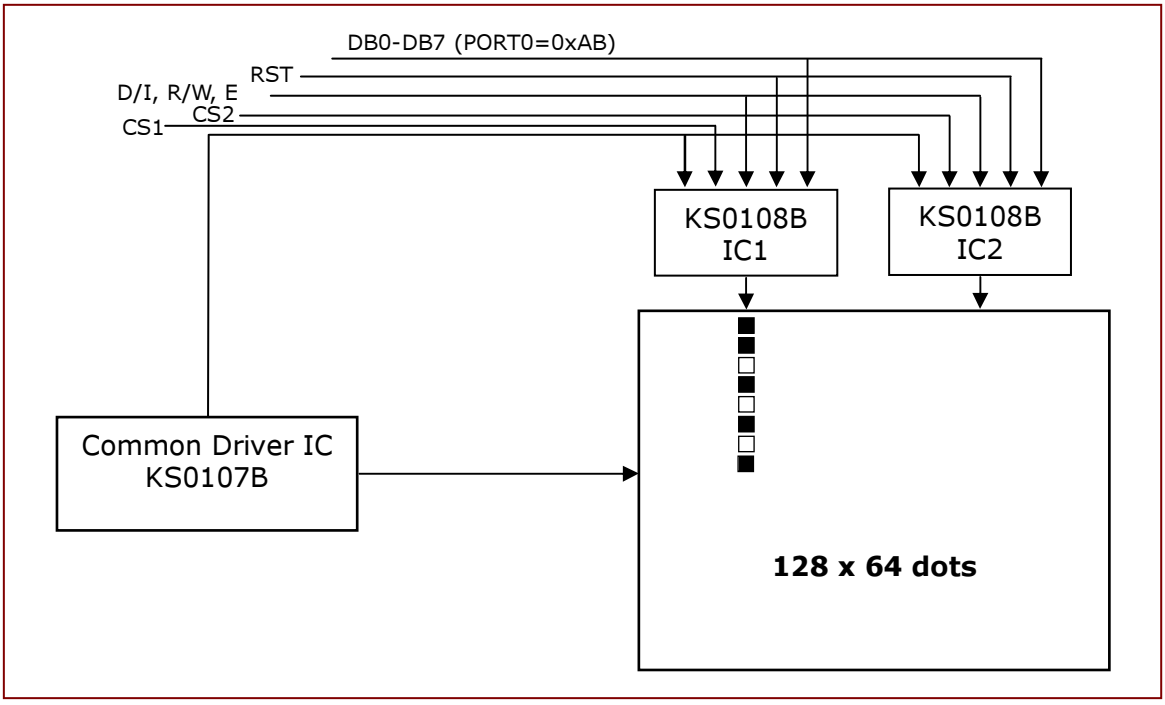


Figure 8.1.3 Byte write 0xAB to LCD (drawing not to scale)

Chapter 8 Interfacing a JHD12864J Graphic Module to AT89S52

8.2 Display Control Instruction

There are two ways to interface the LCD with AT89S52:

1. Address/data bus memory map, i.e. use the LCD as a piece of external RAM
2. Direct I/O connection by writing high/low signal to LCD.

Because there are a lot of other microcontrollers that don't have address/data bus system, we will use method 2 for now so this driver could be easily ported to other microcontrollers in future. The first task is to understand the control sequence required to read or write data to the LCD module. Figure 8.2.1 shows the DISPLAY CONTROL INSTRUCTION extracted from its controller's data sheet (KS0108).

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display ON/OFF	L	L	L	L	H	H	H	H	H	L/H	Controls the display on or off. Internal status and display RAM data is not affected. L:OFF, H:ON
Set Address	L	L	L	H	Y address (0~63)						Sets the Y address in the Y address counter.
Set Page (X address)	L	L	H	L	H	H	H	Page (0~7)			Sets the X address at the X address register.
Display Start Line	L	L	H	H	Display start line (0~63)						Indicates the display data RAM displayed at the top of the screen.
Status Read	L	H	BUSY	L	ON/OFF	RESET	L	L	L	L	Read status. BUSY L: Ready H: In operation ON/OFF L: Display ON H: Display OFF RESET L: Normal H: Reset
Write Display Data	H	L	Write Data								Writes data (DB0:7) into display data RAM. After writing intruction, Y address is increased by 1 automatically.
Read Display Data	H	H	Read Data								Reads data (DB0:7) from display data RAM to the data bus.

Figure 8.2.1 Display Control Instruction (Samsung KS0108)

There are several points to watch out:

1. RS is equivalent to PIN D/I as stated on JHD12864J data sheet. It controls data or command action (D/I=LOW ⇒ command; D/I=HIGH ⇒ data).
2. Horizontal pixel addressed by ***Y address counter*** (0-63). The nomenclature is not the same as our convention of Cartesian coordinate system (x-y) learned in secondary school. The ***Y address*** indicates the column position in the horizontal direction. Why only 64 pixels but not 128 pixels? Because the LCD is splitted logically in half with controller #1 (CS1) driving the left half of the display, and controller #2 (CS2) driving the right half. We need to handle each half individually.
3. The term ***Page*** refers to 8-pixel vertically. There are 8 pages ranging from 0 to 7, thus matching a vertical matrix size of 64 pixels. Refer to Figure 8.1.2 for illustration.

4. R/W controls data READ/WRITE (R/W=LOW ⇒ write; R/W=HIGH ⇒ read). The reason of writing bytes to the LCD is obvious: we need to display something on the LCD. However, being capable of reading from the module is also important because it is only possible to write to a whole ***Page*** in 8-bit format. As an example, we want to display a single pixel at the 10th column on 3rd pixel down the top ***Page*** where there is an existing byte 0xAB. If we simply output 0x40 (0b0000 0100), the byte pattern 0xAB would be erased. One possible way is to perform a data read first, store the byte in background to a temporary variable, and do a bitwise OR operation with 0x40. This makes a new byte value of 0xAF. The whole procedure is illustrated below.

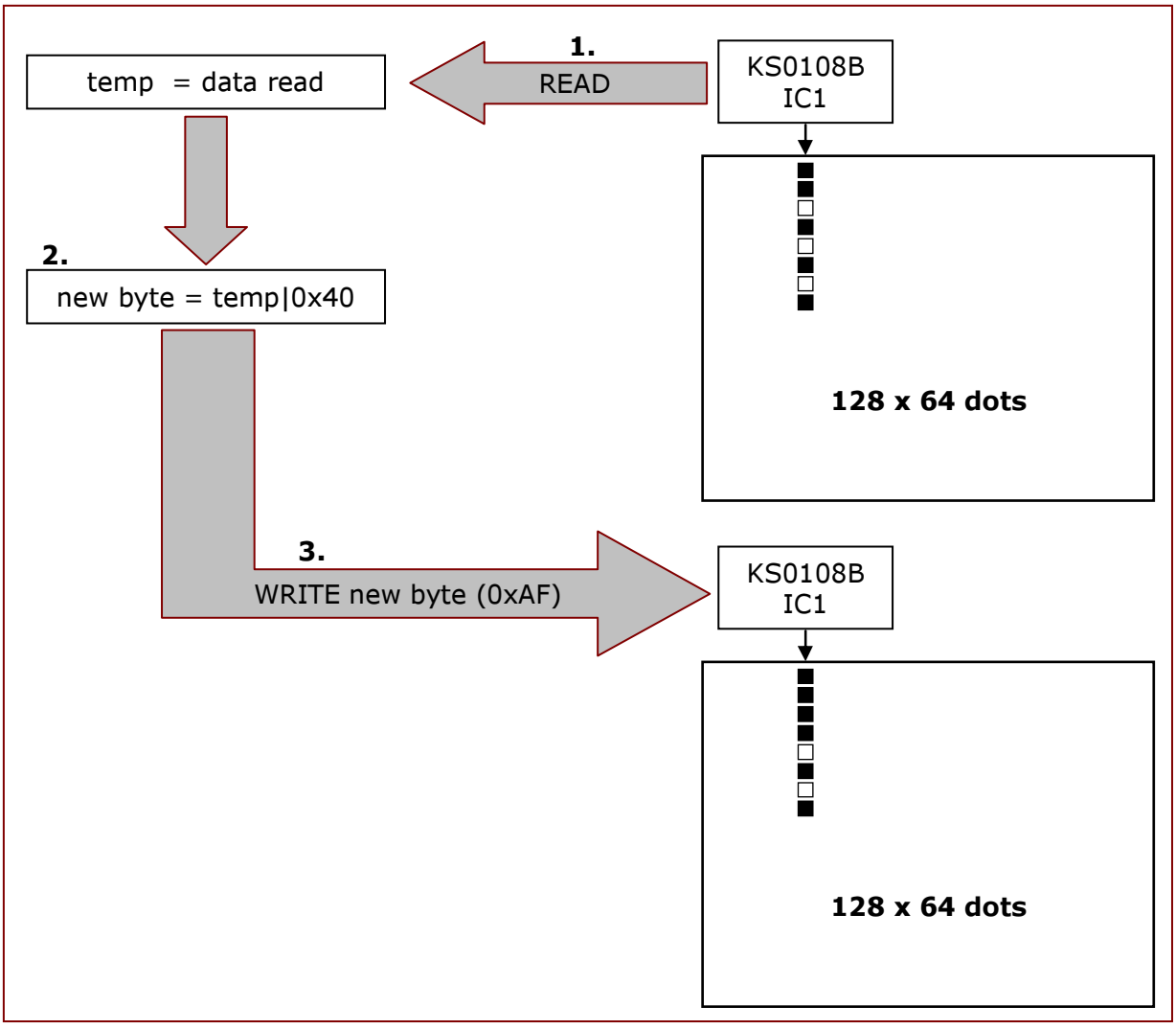


Figure 8.2.2 Procedure to write a single pixel

5. Besides D/I, and R/W pins, there are other control pins to take care including CS1, CS2, E, and RST pins. Direct low-level access and signal timing requirement will be taken care by hardware dependent functions. The application interface function (API) is hardware-independent. This idea is to allow easy porting to other microcontrollers since we only have to re-write the hardware interface for other microcontrollers or compliers.

Chapter 8 Interfacing a JHD12864J Graphic Module to AT89S52

8.3 The First Demonstration Program

Figure 8.3.1 shows a block diagram of the LCD driver module. There are not a lot of functions involved.

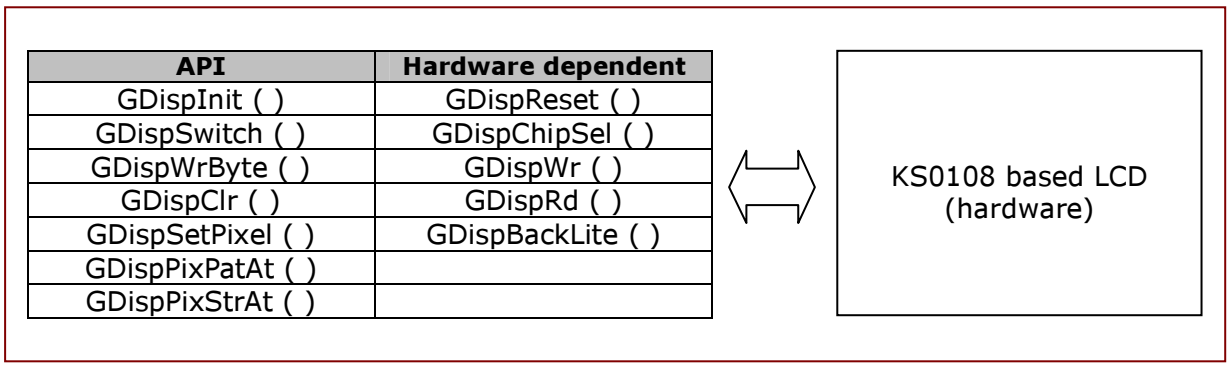


Figure 8.3.1 Graphic LCD driver block diagram

Without much knowledge on the driver module, let's look at the demonstration program first. Listing 8.3.1 shows the gLCD_STK1_Demo.c file. This project can be found in the directory cd:\src\chp8\src8_1\JHD12864_demo1.Uv2.

```
#include <regx52.h>
#include "delay.h"
#include "ks0108.h"

void main(void)
{
        unsigned char x;

        GDispInit();                                                    (1)

        for(;;){
        GDispPixStrAt(50,32, "Temp: 25.5C", 1, BLACK);                  (2)
        GDispPixStrAt(50,42, "Humidity: 50.3%", 1, BLACK);              (3)
        for(x=0;x<128;x++) GDispSetPixel(x,0,BLACK);                    (4)
        for(x=0;x<128;x++) GDispSetPixel(x,63,BLACK);                   (5)
        DelayMs(3000);
        GDispClr(0x00);                                                 (6)
        GDispPixStrAt(50,32, "12:35 pm", 1, BLACK);                     (7)
        GDispPixStrAt(50,42, "4th July '06", 1, BLACK);                 (8)
        for(x=0;x<128;x++) GDispSetPixel(x,0,BLACK);                    (9)
        for(x=0;x<128;x++) GDispSetPixel(x,63,BLACK);                   (10)
        DelayMs(3000);
        GDispClr(0xFF);                                                 (11)
        DelayMs(3000);
        GDispPixStrAt(0,10, " !#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI  (12)
        JKLMNOPQRST UVWXYZ[\\]^_'abcdefghijklmnopqrstuvwxyz{|}~", 2, WHITE);
        DelayMs(3000);
        GDispClr(0x00);                                                 (13)
        }
}
```

Listing 8.3.1 gLCD_STK1_Demo.c

Line (1) initializes the LCD module. The actions as shown in Figure 8.3.2 are performed by this initialization function. Simply saying, this function resets and clears the module. It should be called before using all other functions.

```
GDispInit()
{
GDispBackLite(ENABLE);                  //enable the backlight
GDispReset();                           //reset the module
GDispSwitch(DISABLE);                   //disable the module to avoid screen flicker
GDispWr(CMD,DISPLAY_START_LINE);        //set display start line at the top of the LCD
GDispClr(0x00);                         //clear the display by repeat writing 0x00
GDispSwitch(ENABLE);                    //enable the module
}
```

Figure 8.3.2 GDispInit ( ) function

Line (2) displays the string "Temp: 25.5C" at the coordinates 50, 32; font spacing 1.

Line (3) displays the string "Humidity: 50.3%" at the coordinates 50, 42; font spacing 1.

Line (4) draws 128 pixels on the top line with x running from 0 to 127 at y=0. Coordinate x runs from 0 to 127 in the horizontal direction.
Line (5) draws 128 pixels at the bottom line with x running from 0 to 127 at y=63. Little icons at the top right hand corner represent coordinates (124,63), (125,63), (126,63), and (127,63).

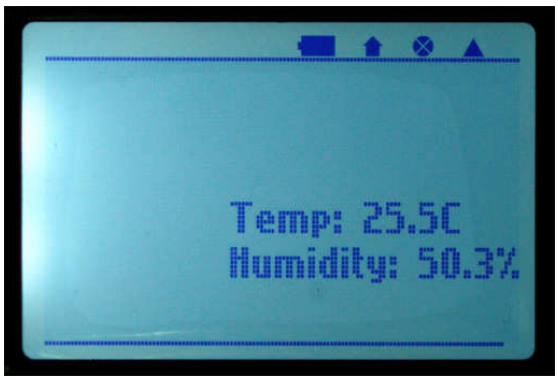


Figure 8.3.3

Line (6) clears the display after 3 seconds; otherwise, the existing graphic content will overlap with the date/time coming over.

Line (7) – **Line (10)** displays the strings "12:35 pm" and "4th July '06" with the same function call as Line (2) – (5). Result shown in Figure 8.3.4.

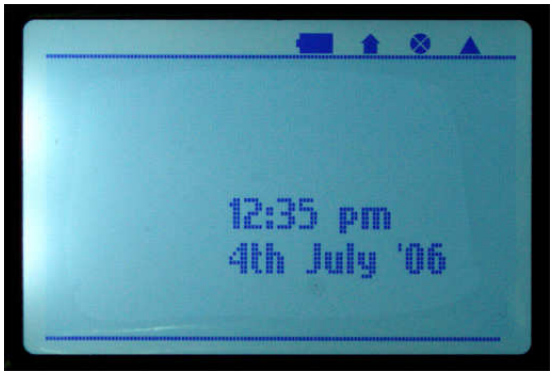


Figure 8.3.4

Line (11) fills the whole LCD module with a pattern of 0xFF, i.e. with all pixel turns black (Figure 8.3.5).

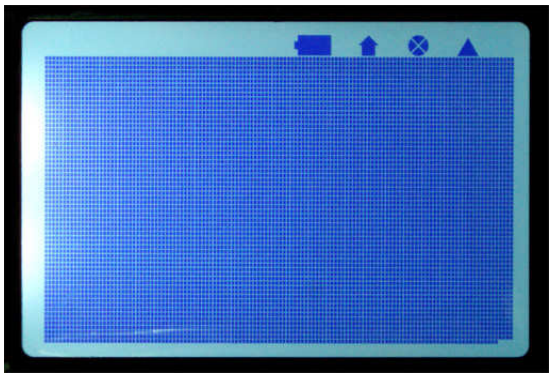


Figure 8.3.5 All pixels black

Line (12) displays the ASCII table (Figure 8.3.6) in white color from "SPACE" to "~" with font spacing equal 2, starting from the coordinate 0, 10.

Line (13) clears the screen again after 3 seconds and the whole sequence starts all over in an infinite for-loop.

Figure 8.3.6 ASCII Table

8.4 How the first demonstration was done?

First-of-all, we need to understand the application interface. Listing 8.4.1 shows the header file (ks0108.h). We need to include this header file in our main program and include the ks0108.c source code in project workspace to use any of the functions inside.

```
#ifndef _KS0108_H_
#define _KS0108_H_

#define PIXEL_FONT_EN           1                                   (1)
#if     PIXEL_FONT_EN
#define MAX_FONT_WIDTH          8                                   (2)

typedef struct _font                                                (3)
        {
        unsigned char fontWidth;
        unsigned char fontBody[MAX_FONT_WIDTH];
        }font;
#endif

#define PIXEL_PAT_EN            0                                   (4)
#if PIXEL_PAT_EN
#define MAX_PAT_SIZE            48*8                                (5)
typedef struct _pattern                                             (6)
        {
        unsigned char patWidth;
        unsigned char patHeight;
        unsigned char patBody[MAX_PAT_SIZE];
        }pattern;
#endif
```

Code continues on next page...

Listing 8.4.1 ks0108.h header file

Line (1) specifies if we are using the system font or not. Enable font when 1, else 0. We need to accept, that there is no Operating System (OS) with our little 8-bit microcontroller system yet. Let's take Microsoft® Windows as an example. System font, Times New Roman font, and many other font types are all built-in the OS. However, there is no such nice feature with our system. We need to define a font table in code space for ASCII characters from "SPACE" to "~"; otherwise, there is no font as "A", "B", "C" at all.

Header file SYSFONT.h defines the font table. Figure 8.4.1 shows an extract.

```
code font SYS_FNT[95] = {
{4, {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}},   //SPACE
{4, {0x60,0xF0,0xF0,0x60,0x60,0x00,0x60,0x00}},   //!
{3, {0xA0,0xA0,0x00,0x00,0x00,0x00,0x00,0x00}},   //"
…..
{6, {0x00,0x00,0x00,0x60,0xB4,0x18,0x00,0x00}}    //~
};
```

Figure 8.4.1 An extract of SYSFONT.h font table

The keyword "code" is used to allocate the font table in code space. A structure is defined in Line (3) with two members as font width (***fontWidth***) and font body (***fontBody***). Figure 8.4.2 shows the bitmap layout for all 95 ASCII characters defined in SYSFONT.h.

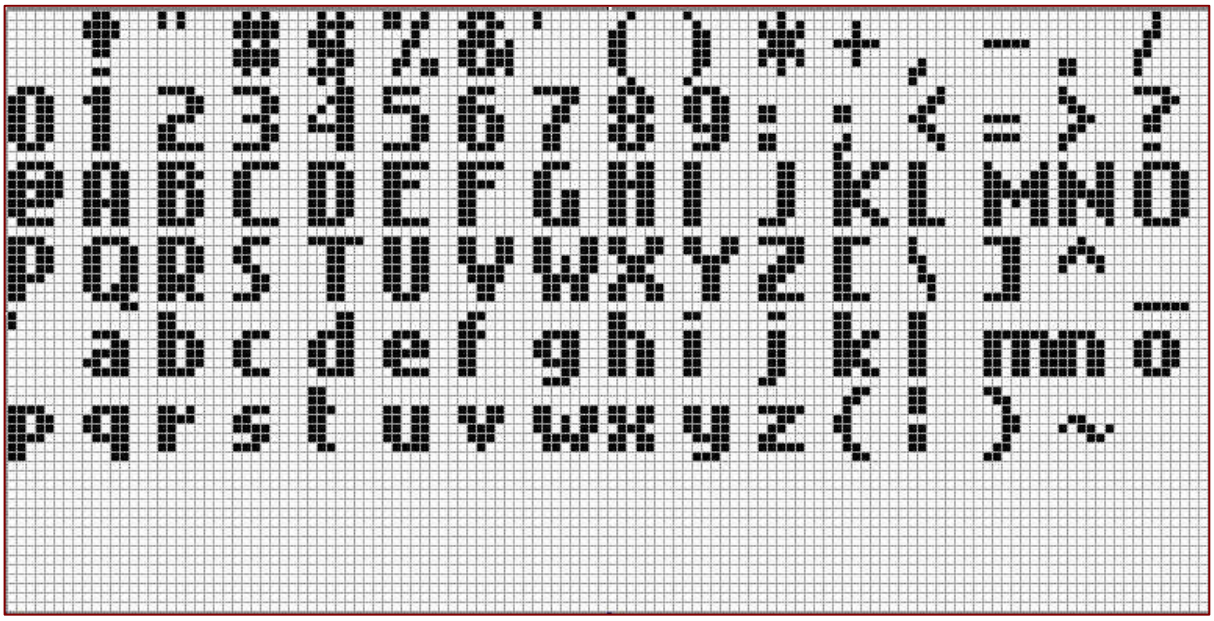
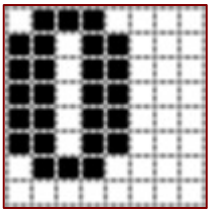


Figure 8.4.2 SYSFONT.h in bitmap layout, picture in 128x64

Individual character occupies a different width. For example, the letter "I" occupies 2 pixels, whereas the letter "M" occupies 7 pixels in width. This is the reason for declaring ***fontWdith*** in the ***font*** structure. Function GDispPixStrAt() uses this information to display strings with proportional appearance. The actual display result has been shown in Figure 8.3.6.

fontBody[MAX_FONT_WIDTH] declares an array of 8 elements in horizontal direction. Let's take the letter "0" as an example (Figure 8.4.3):



```
font.fontWidth = 5;
font.fontBody={0x70,0xD8,0xD8,0xD8,0xD8,0xD8,0x70,0x00};
```

The HEX code 0x70 indicates the top row of pixels.

Figure 8.4.3

If we want more efficient display, we need to define arrays corresponding to pixels in vertical direction to make use of the intrinsic ***Page-write(vertical)*** properties. We will leave this for future versions.

Line (4) specifies if we need to display any icon or pattern of size ***patWidth*** and ***patHeight*** defined in the structure ***pattern***. We are not using this for this example; therefore, PIXEL_PAT_EN=0. Refer to section 8.5 for this.

```
//I/O port for data definition
#define LCD_DATA          P0                                          (7)

/*********************************************************************/
//Control pin setting (Keil C specific for 8051)
sbit    LCD_BL      =     P2^7;                                       (8)
sbit    LCD_CS1     =     P2^4;
sbit    LCD_CS2     =     P2^5;
sbit    LCD_RST     =     P2^6;
sbit    LCD_EN      =     P2^3;
sbit    LCD_RW      =     P2^2;
sbit    LCD_DI      =     P2^1;

/*********************************************************************/
#define MAX_ROW_PIXEL          64                                     (9)
#define MAX_COL_PIXEL          128
#define ENABLE                 1
#define DISABLE                0
#define BLACK                  1
#define WHITE                  0
/*********************************************************************/

void GDispInit(void);                                                 (10)
void GDispSwitch(bit sw);                                             (11)
void GDispWrByte(unsigned char col, unsigned char page, unsigned char c);  (12)
void GDispClr(unsigned char pat);                                     (13)
void GDispSetPixel(unsigned char x, unsigned char y, bit color);      (14)
#if PIXEL_PAT_EN                                                      (15)
void GDispPixPatAt(unsigned char x, unsigned char y, pattern* pPat, bit color);
#endif
#if PIXEL_FONT_EN                                                     (16)
void GDispPixStrAt(unsigned char x, unsigned char y, unsigned char *pStr, \  (17)
unsigned char fontSpace, bit color);
#endif

/*********************************************************************/
void                GDispReset(void);                                 (18)
void                GDispChipSel(unsigned char wing);                 (19)
void                GDispWr(bit type, unsigned char dat);             (20)
unsigned char       GDispRd(bit type);                                (21)
void                GDispBackLite(bit ctrl);                          (22)
#endif
```

Listing 8.4.1 ks0108.h header file

Chapter 8 Interfacing a JHD12864J Graphic Module to AT89S52

Line (7) & **Line (8)** define the hardware interface.

Line (9) defines the row and column width and constants for portability.

Here come the API functions:

Line (10)

GDispInit() void GDispInit(void);	
Description:	This function initializes the LCD module by the following steps: 1. Enable the backlight 2. Reset the LCD module 3. Disable the LCD to avoid screen flicking during initialization 4. Set display start line at the top of the screen 5. Clear the display by writing 0 to the whole screen 6. Re-enable the display
Arguments:	none
Returns:	none
Note:	This function should be called before any of the other functions
Example : Please refer to listing 8.3.1	

Line (11)

GDispSwitch() void GDispSwitch(bit sw);	
Description:	This function turns display ON/OFF by writing the command DISPLAY_ON / DISPLAY_OFF as illustrated in Figure 8.2.1
Arguments:	(bit) sw ENABLE for turning ON; DISABLE for turning OFF, constants defined under ks0108.h
Returns:	none
Notes:	Internal status and display RAM data not changed by switching between ON and OFF. Data type 'bit' is Keil C specific

Example :

```
GDispSwitch(DISABLE);        //disable the LCD, screen turns blank
for(x=0;x<128;x++)           //demonstrate a slow drawing process here
{ DelayMs(10);
  GDispSetPixel(x,0,BLACK);
}
GDispSwitch(ENABLE);         //re-enable the LCD, graphics pops up
```

It takes time for drawing large and complicated icons of big size because we are drawing individual pixel. If we don't want to see the whole process of drawing in pixel-by-pixel, we may disable the screen before the drawing process and re-enable it after finish. Screen content is not changed by switching ON/OFF.

Line (12)

GDispWrByte() void GDispWrByte(unsigned char col, unsigned char page, unsigned char c);	
Description:	This function writes a single byte 'c' to a column position defined by 'col' and to PAGE# indicated by the variable 'page'.
Arguments:	'col' the column position (0-127) on LCD 'page' the *PAGE* number (0-7) on LCD. It is NOT the y coordinate as our usual Cartesian coordinate system. It is the PAGE number defined in JHD12864J data sheet. 'c' byte to write
Returns:	none

Notes:	This function should has been declared as a local function because it is rarely called in main. However, if we want to make a different version of string display with higher efficiency, we may make use of this function to write byte in vertical manner. Besides, this function is the best way to illustrate how data write action is performed. Figure 8.4.4 shows the screen after the code:
	
	for(i=0;i<64;i++) GDispWrByte(i, 0, 0xAB);
	//i runs from 0 to 63, *Page* = 0, c = 0xAB.
	
	Now, we have successfully verified Figure 8.1.3!

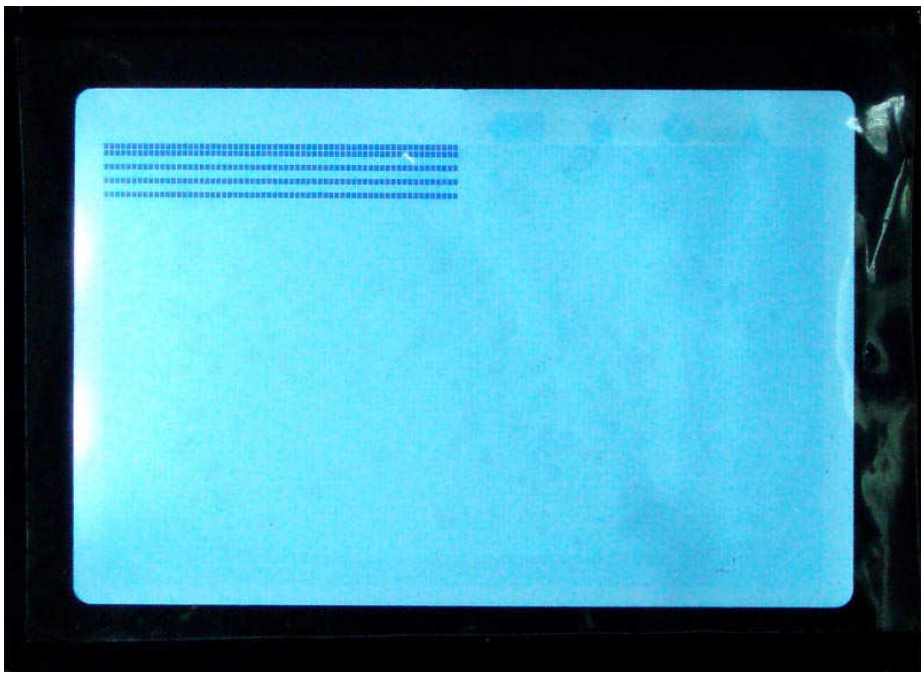


Figure 8.4.4 GDispWrByte(i,0,0xAB), with 0<=i<64

Line (13)

GDispClr() void GDispClr(unsigned char pat);	
Description:	This function writes a fixed byte to all pages (0-7) of both half. If pat is 0x00, the action is equivalent to clearing the display (turning all pixels WHITE)
Arguments:	'pat' is the pattern to write to the display in a repetitive manner. Writing 0xFF for example, will turn all pixels BLACK.
Returns:	none
Notes:	This function makes use of GDispWrByte()thus the speed of clearing is fast!
Example:	Please refer to Listing 8.3.1

Line (14)

GDispSetPixel() void GDispSetPixel(unsigned char x, unsigned char y, bit color)	
Description:	This function sets a pixel with color = BLACK / WHITE (defined in ks0108.h) at a position defined by (x,y) following Cartesian coordinate system. Coordinates (0,0) at the top left corner, coordinates (127,63) at the lower right corner of the LCD screen.
Arguments:	'x' 0....MAX_COL_PIXEL-1 is the matrix position in horizontal direction 'y' 0....MAX_ROW_PIXEL-1 is the matrix position in vertical direction 'color' sets BLACK / WHITE standing for pixel ON/OFF
Returns:	none
Notes:	Please refer to comment in source code for further details Data type 'bit' for color is Keil C specific!
Example:	Please refer to listing 8.3.1

Line (15)

GDispPixPatAt() void GDispPixPatAt(unsigned char x, unsigned char y, pattern* pPat, bit color)	
Description:	This function writes a particular pattern/icon of size patWidth and patHeight defined under the pattern structure in ks0108.h at a position defined by (x,y) following Cartesian coordinate system. *** Pixels displayed in horizontal manner ***
Arguments:	'x' 0....MAX_COL_PIXEL-1 is the matrix position in horizontal direction, top left corner of the pattern to write 'y' 0....MAX_ROW_PIXEL-1 is the matrix position in vertical direction, top left corner of the pattern to write '*pPat' pointer to pattern structure 'color' sets BLACK / WHITE standing for pixel ON/OFF
Returns:	none
Notes:	Turn this function ON/OFF at compile time with the PIXEL_PAT_EN switch, 1 to enable, 0 to disable. Data type 'bit' for color is Keil C specific!
Example:	Please refer to Listing 8.5.1

Line (16-17)

GDispPixStrAt() void GDispPixStrAt(unsigned char x, unsigned char y, unsigned char *pStr, unsigned char fontSpace, bit color)	
Description:	This function outputs a string in graphic mode (pixel-by-pixel) at a position defined by (x,y) following Cartesian coordinate system with variable font spacing defined by fontSpace. Character wrapping is allowed. Enable switch PIXEL_FONT_EN under ks0108.h to use this function **** REQUIRE AT LEAST ONE FONT SET (sysfont.h) ****
Arguments:	'x' 0....MAX_COL_PIXEL-1 is matrix position in horizontal direction, top left corner of each character to write 'y' 0....MAX_ROW_PIXEL-1 is matrix position in vertical direction, top left corner of each character to write '*pStr' pointer to an ASCII string 'fontSpace' font spacing from 1 onwards 'color' WHITE / BLACK
Returns:	none
Notes:	Turn this function ON/OFF at compile time with the PIXEL_FONT_EN switch, 1 to enable, 0 to disable. Data type 'bit' for color is Keil C specific!
Example:	Please refer to listing 8.3.1

It would be interesting to see what happens if we turn off GDispPixStrAt() by disabling PIXEL_FONT_EN switch. Change Line (1) of ks0108.h (Listing 8.4.1) to

```
#define PIXEL_FONT_EN              0
```

Comment all GDispPixStrAt() functions in Listing 8.3.1, re-compile the project:
Code = 596byte, in contrast with code = 1697 byte when PIXEL_FONT_EN = 1!

ASCII table SYSFONT.h occupies more than 1kb of our precious code space! It would be alright if we used a full version of Keil C, but not for now! There are several ways to tackle this problem:

1. Buy a legal Keil C compiler to release the 2K limit
2. Switch to other lower cost compilers including the following:
   a. RKITL51 from RAISONANCE, 4k demo version free! http://www.raisonance.com/products/8051.php
   b. µC/51 from Wickenhäuser Elektrotechnik. There is an 8k limit demo version free! http://www.wickenhaeuser.de
   c. SDCC – Small Device C Compiler under http://sdcc.sourceforge.net. No charge!
3. Use assembly language
4. Store the font table in external eeprom like AT24C256 onboard
5. Give up!

Think about it. There is more than one way to Rome.

Line (18) – Line (21) are hardware dependent functions. These functions are not called explicitly in main. Please refer to source code for comments.

Line (22)

GDispBackLite() void GDispBackLite(bit ctrl)	
Description: Arguments: Returns: Notes:	This function controls the LCD backlight ENABLE -OR- DISABLE 'bit' ENABLE for backlight ON DISABLE for backlight OFF none Hardware specific. Control done via MOSFET IRLML6401. Refer to schematic for details. Data type 'bit' for color is Keil C specific!

Example:	if(no keypad for longer than 15 sec) GDispBackLite(DISABLE);
	
	//turn off backlight if no user action for 15 sec or longer

Chapter 8 Interfacing a JHD12864J Graphic Module to AT89S52

8.5 Drawing Icons 

Here comes the final section of this manual. Honestly, I am a little bit tired so there won't be a lot here. The source code is found under cd:\src\chp8\src8_1\JHD12864_demo2.Uv2.

```
#include <regx52.h>
#include "delay.h"
#include "ks0108.h"                                          (1)
#include "icons.h"                                           (2)

void main(void)
{
        unsigned char x;

        GDispInit();

        for(;;){
        GDispPixPatAt(3,10,&ICONS[0],BLACK);                 (3)
        for(x=0;x<128;x++) GDispSetPixel(x,0,BLACK);         (4)
        for(x=0;x<128;x++) GDispSetPixel(x,63,BLACK);        (5)
        DelayMs(3000);
        GDispBackLite(DISABLE);                              (6)
        GDispSwitch(DISABLE);                                (7)
        GDispClr(0xFF);                                      (8)
        GDispPixPatAt(3,10,&ICONS[1],WHITE);                 (9)
        for(x=0;x<128;x++) GDispSetPixel(x,2,WHITE);         (10)
        for(x=0;x<128;x++) GDispSetPixel(x,61,WHITE);        (11)
        for(x=124; x<128; x++) GDispSetPixel(x,63,WHITE);    (12)
        GDispSwitch(ENABLE);                                 (13)
        GDispBackLite(ENABLE);                               (14)
        DelayMs(2000);
        for(x=124; x<128; x++)
                {
                        DelayMs(1500);
                        GDispSetPixel(x,63,BLACK);           (15)
                }
         DelayMs(2000);
         GDispClr(0x00);
        }
}
```

Listing 8.5.1 gLCD_STK1_Demo2.c

This demonstration program makes use of GDispPixPatAt( ) to display icons of size 48x48 pixels. It is the maximum size we can handle by declaring the MAX_PAT_SIZE to be 48*8 under ks0108.h. It is possible to change that size to any arbitrary number say 52*8 as long as the code size is within 2k limit of eval Keil C. Icon size of 48x48 seems appropriate anyway.

Line (1) includes ks0108.h header file. This time, PIXEL_FONT_EN switch disable, PIXEL_PAT_EN switch enable. So no GDispPixStrAt( ) allowed.

Line (2) includes the map of icons.h. It can be any file name as long as it has the file extension *.h. First of all, prepare an icon of some graphics you love. It can be bitmap file prepared by any software like Microsoft® Painting Program.

The problem is, how to prepare data that can be embedded in our main program? Modern electronic devices like mobile phones or digital cameras allow direct USB connection to a PC. Graphics downloaded via Windows software with nice user interface. Unfortunately, we don't have that feature yet. Graphic decode algorithm is required for the microcontroller if it has to understand jpeg or bitmap graphics. So, we need a software that translates a bitmap file to *.h file. A quick search on the Internet led me to FastLCD designed by Bojan I. MICRODESIGN. Figure 8.5.1 shows an extract of the icons.h header file.

```
code pattern ICONS[2] = {
{48, 48, {0x7F,0xFF,0xFF,0xFF,0xFF,0xFC,0xC0,0x00,0x00,0x00,0x00,0x06,
          0x80,0x00,0x00,0x00, 0x00,0x02,0x80,0x38,0x00,0x00,0x00,
          0x02,0x80,0x44,0x00,0x00,0x00,0x02,0x80,0x44,0x00,0x00,0x00,
….

0x00,0x00}}
};

#endif
```

Figure 8.5.1 Extract of icons.h

Please refer to ks0108.h for definition of structure ***pattern***. Please be reminded that array data should corresponds to pixels in horizontal manner.

Line (3) displays the first icon in black.
Line (4) – Line (5) draws two black lines on top and at the bottom of the screen. The result is shown in Figure 8.5.2.

Figure 8.5.2

Line (6) turns off the backlight.

Line (7) disables the LCD module. Graphics content preserved though.

Line (8) fills the whole screen with all pixels turned ON. However, you won't see individual pixel action anymore because the module has been disabled by Line (7).

Line (9) displays the second icon in white color. No graphics can be seen yet because the module has been disabled by Line (7).

Line (10) – Line (11) draw white lines on top 3rd and the 62nd at the bottom against a black background.

Line (12) clears individual pixel at coordinates (124,63), (125,63), (126,63), and (127, 63). These pixels correspond to intrinsic icons at the top right hand corner of the module.

Line (13) enables the whole screen finally. It is the moment you see something on screen. Please note that all graphics pop up at the same time. No individual pixel plotting could be seen in contrast with the way the first icon was drawn.

Line (15) turns on individual pixel from (124,63) to (127,63), result as shown in Figure 8.5.3.

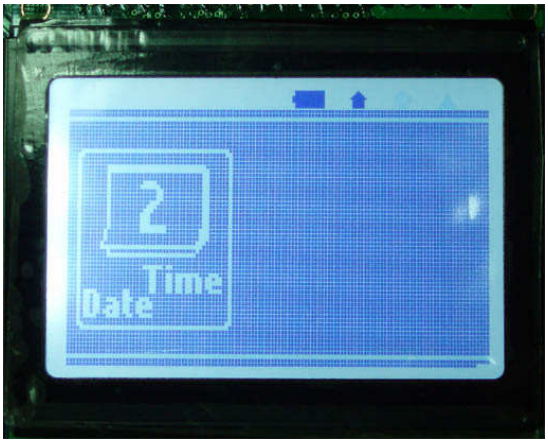


Figure 8.5.3

END