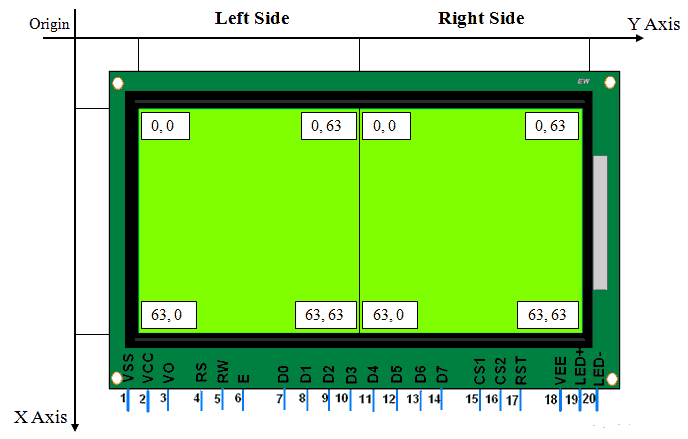
**Graphic LCD với PIC**

**I, Tổng quan về Graphic LCD**

Chúng ta đã biết về Text LCD (LCD 16x2, 20x4), chúng có các ô hiển thị các mã ASCII, do có bộ nhớ CGRAM (Character Generation RAM). Graphic LCD (gọi tắt là GLCD) thì không có bộ nhớ CGRAM. GLCD 128x64 có 128 cột và 64 hàng tương ứng với 128x64 = 8192 chấm (dot), mỗi chấm tương ứng với 1bit dữ liệu. Chính vì thế để hiển thị đầy mỗi GLCD, chúng ta cần 8192 bits (1024 bytes) RAM.

Tùy theo loại chip điều khiển mà nguyên lý hoạt động của GLCD là khác nhau. Trong bài viết này chúng ta sẽ tìm hiểu GLCD được điểu khiển bởi chip KS0108 của Samsung, đây là dòng GLCD phổ biến nhất trong các loại GLCD không màu.

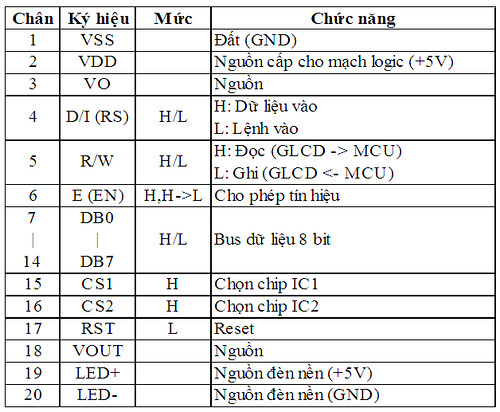


**1, Sơ đồ chân của GLCD 128x64**

GLCD 128x64 gồm 20 chân, trong đó chỉ có 18 chân dùng để điều khiển trực tiếp lên GLCD, 2 chân (thường là chân 19, 20) là Anode và Cathode của Led nền.

Trong 18 chân còn lại, có 4 chân dùng để cấp nguồn và 14 chân để điều khiển dữ liệu. GLCD chỉ hỗ trợ giao tiếp 8 bit.

Chức năng của các chân được miêu tả tổng quan trong bảng sau (đối với các GLCD loại chip điều khiển khác thì thứ tự chân có thể thay đổi):



* Chân **VSS** được nối trực tiếp với GND.
* Chân **VDD** được nối với nguồn +5V.
* Một biến trở sẽ được dùng để chia điện áp giữa chân VDD và chân Vout (VEE) cho chân VO. Khi thay đổi giá trị biến trở ta sẽ điều chỉnh được độ tương phản của GLCD.
* Các chân điều khiển RS, R/W, EN và các đường dữ liệu được nối trực tiếp với vi điều khiển, riêng chân RST(Reset) có thể nối trực tiếp lên nguồn +5V.
* **EN (Enable)**: cho phép 1 quá trình nào đó được thực hiện với GLCD. Khi chân **EN** đang được giữ ở mức Low, nếu muốn thực hiện 1 quá trình (đọc, ghi lệnh, dữ liệu) các chân điều khiển khác sẽ được cài đặt sẵn sàng, sau đó sẽ kéo chân **EN** lên mức High. Lúc này GLCD sẽ bắt đầu thực hiện yêu cầu của chúng ta, chúng ta cần Delay 1 khoảng thời gian ngắn để chờ cho GLCD đọc hoặc gửi dữ lệnh, dữ liệu. Cuối cùng, ta cần kéo **EN** xuống lại mức Low để kết húc quá trình cũng như chuẩn bị cho quá trình kế tiếp của GLCD.
* **RS (Register Select)**: chân lựa chọn giữa dữ liệu (**Data**) hoặc lệnh (**Instruction**), trong 1 số tài liệu chân này được gi là **DI(Data/Instruction).** RS = 1(High), tín hiệu trên các đường D0-7 là dữ liệu (Data),

RS = 0(Low), tín hiệu trên các đường D0-7 là 1 mã lệnh (Instruction).

* **RW (Read/Write Select):** chân lựa chọn giữa việc đọc hoặc ghi.

Khi **RW** = 1, đọc từ GLCD ra PIC **(GLCD => PIC**).

Khi **RW** = 0, ghi từ PIC vào GLCD (**PIC => GLCD**). Giao tiếp được thực hiện chủ yếu là ghi từ PIC vào GLCD, chỉ duy nhất trường hợp đọc dữ liệu từ GLCD ra, đó chính là đọc bit BUSY và đọc dữ liệu từ RAM.

Việc đọc bit BUSY sẽ giúp ta kiểm tra xem GLCD có bận hay không, việc đọc này sẽ được dùng để viết hàm đợi GLCD. Còn việc đọc dữ liệu từ RAM là 1 tính năng mà Text LCD không có, tính năng này giúp chúng ta thực hiện được nhiều phép logic hình để hiển thị lên GLCD.

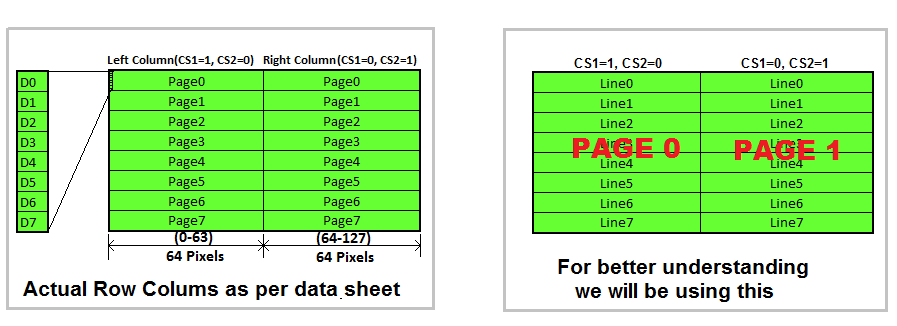
* **CS2 và CS1 (Chip Select):** chip KS0108 chỉ có khả năng điều khiển **một** GLCD 64x64. Trên các GLCD 128x64, có 2 chip KS0108 làm việc cùng nhau, mỗi chip điều khiển 1 GLCD 64x64 (tức là nửa trái hoặc nửa phải của GLCD 128x64). Chính vì vậy 2 chân CS2 và CS1 được tạo ra để cho phép chọn 1 chip KS0108 để làm việc.

CS2 = 0, CS1 = 1 thì nửa trái được chọn. Ngược lại nếu CS2 = 1, CS1 = 0 thì nửa phải được chọn.

**2, Tổ chức bộ nhớ của GLCD**

Chip KS0108 chỉ có 1 loại bộ nhớ duy nhất là RAM, vì vậy dữ liệu ghi và RAM sẽ được hiển thị trực tiếp lên GLCD. Mỗi chip KS0108 có 4096 chấm (512 bytes) RAM trên một nửa GLCD (64x64). RAM của KS0108 chỉ cho phép truy cập theo byte chứ không cho truy xuất từng bit, đồng nghĩa với việc khi ta viết 1 giá trị vào 1 byte trên RAM thì sẽ có 8 chấm trên cùng 1 cột của màn hình bị tác động. Do đó, 64 dòng của GLCD thường được chia thành 8 pages, mỗi pages có độ cao 8 bit và độ rộng 128 cột.

Hình dưới đây sẽ mô tả khá chi tiết về các sắp xếp RAM của chip KS0108:



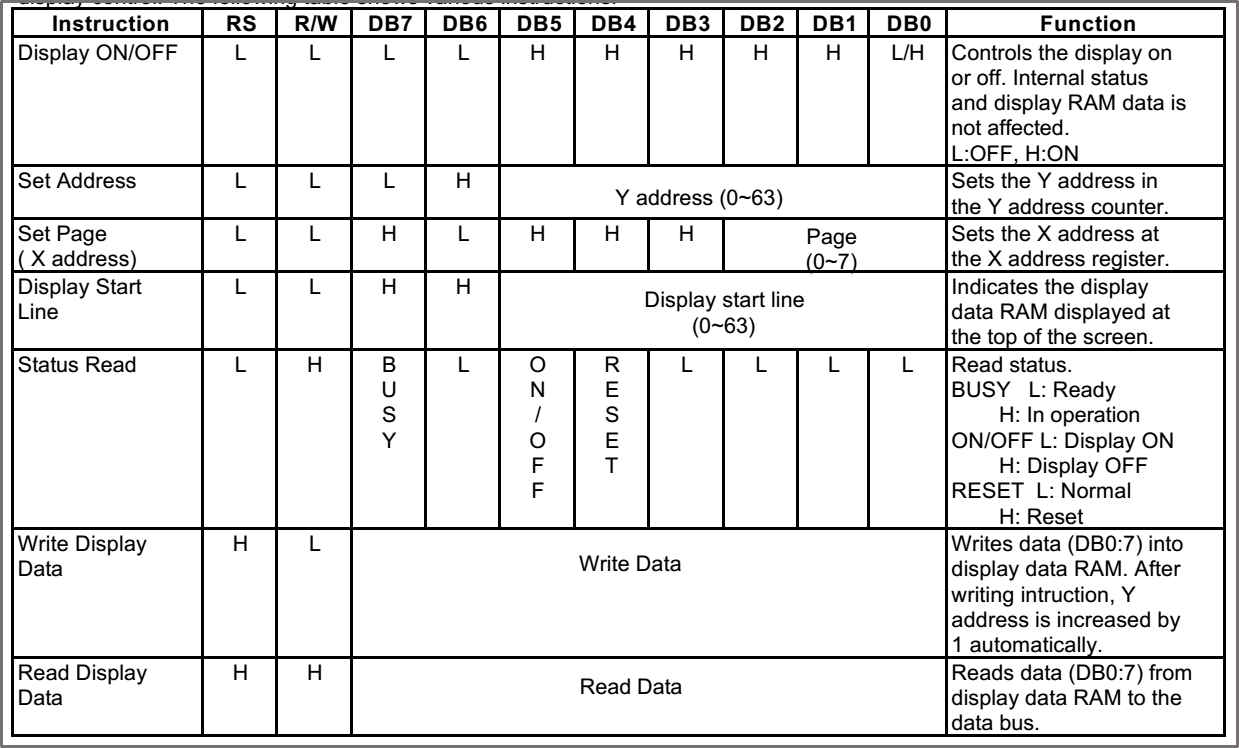
Tổ chức RAM của 2 chip KS0108 trái và phải hoàn toàn tương đương, vì thế việc đọc hay ghi vào RAM của 2 chip được thực hiện như nhau.

Nhìn vào hình trên, ta có thể thấy Page trên cùng là Page 0, page dưới cùng là Page 7. Trong GLCD, Page còn được gọi là **X (X address)**. Mỗi Page chứ 128 cột (từ 0-127), mỗi cột tương ứng với 1byte RAM (8bit D0-D7). Thứ tự các cột trong 1 Page được gọi là **Y (Y address)**, ví dụ: cột đầu tiên có địa chỉ Y = 0, cột cuối cùng có địa chỉ Y = 127.

* Bằng cách phối hợp địa chỉ X, Y ta có thể xác định vị trí của byte cần đọc, ghi. Và dĩ nhiên chip KS0108 có hỗ trợ các lệnh di chuyển đến địa chỉ (X, Y).

**3, Tập lệnh cho chip KS0108**

Các lệnh mà chip KS0108 hỗ trợ sẵn để di chuyển đến các địa chỉ (X, Y) sẽ được mô tả trong bảng dưới đây:



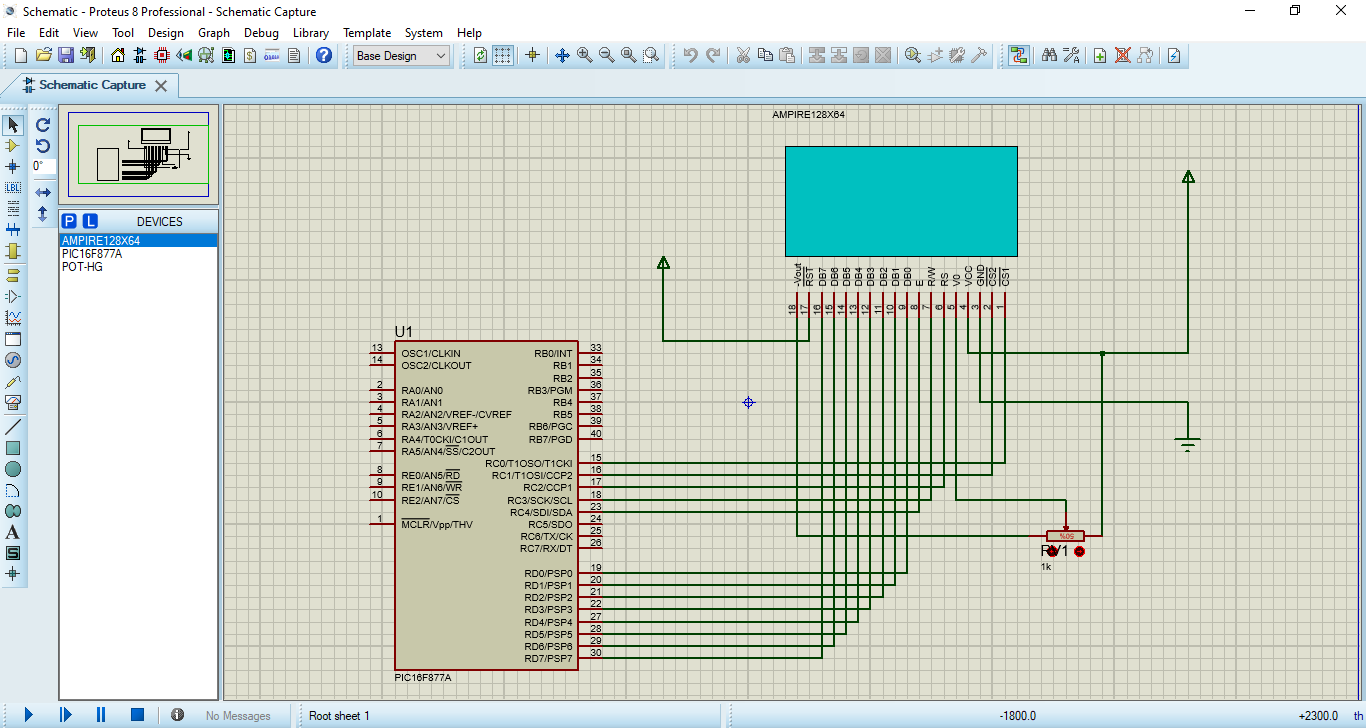
* **Display ON/OFF:** hiển thị GLCD, lệnh này cho phép hiển thị nội dung trên RAM lên GLCD. Để viết lệnh này, 2 chân RS và RW được kéo xuống mức thấp. Mã lệnh được chứa trong 7bits (D7-1), bit D0 chứa thông số. Dựa vào bảng trên, ta dễ thấy mã lệnh cho Display ON/OFF là 0011 111x (0x3E+x). Nếu x = 1, cho phép GLCD hiển thị, x = 0 thì tắt hiển thị
* **Set Address:** chọn cột, hoặc nói chính xác là chọn Y address. Hai bit D7 và D6 lần lượt được set ở mức Low, High, 6bits còn lại chứa chỉ số của cột muốn chọn.
* Ta có thể dễ dàng viết được mã lệnh này: **0x40 (0100 000) + Y**, trong đó Y là chỉ số của cột ta muốn chọn. Ví dụ: muốn chuyển đến cột 25 thì ta ghi vào GLCD mã lệnh 0x40 + 25.
* **Set Page:** chọn X address. Tương tự như lệnh Set Address. Ta có thể viết được mã lệnh này như sau: **0xB8 (1011 1000) + X**, trong đó X là chỉ sổ của Page mà ta muốn chọn. Ví dụ: muốn chọn Page 7 thì ta ghi lệnh 0XB8 + 7.
* **Display Start Line:** chọn line đầu tiên, hay còn gọi là lệnh “cuộn”. Lệnh này cho phép cuộn toàn bộ những gì đang hiển thị trên GLCD (hay RAM) lên phía trên 1 số dòng nào đó. Mã lệnh cho lệnh này là:

**0XC0 (1100 000) + L**, trong đó L là số dòng mà ta muốn cuộn lên

* **Status Read:** đọc trạng thái của GLCD. Lệnh này chủ yếu được dùng để xét bit BUSY (bit thứ 7) xem GLCD có đang bận hay không, để viết hàm đợi GLCD. Chân RW phải được set lên mức High (chế độ Read).
* **Write Display Data:** ghi dữ liệu cần hiển thị vào GLCD hay RAM. Lệnh này cho phép ghi 1byte dữ liệu vào RAM của KS1080 và cũng là dữ liệu hiển thị lên GLCD tại vị trí (X, Y) đang set. 8bits dữ liệu này sẽ tương ứng với 8 chấm trên cột Y ở page X. Sau khi thực hiện lệnh này, Y sẽ tự động tăng thêm 1, sau khi ghi ở cột Y = 63 (cột cuối trong 1 page của 1 chip KS0108) thì Y sẽ về 0.
* **Read Display Data:** đọc dữ liệu hiển thị từ GLCD. Lệnh này cho phép ta đọc 1byte dữ liệu từ RAM của KS0108 tại vị trí hiện hang về PIC. Sau khi đọc xong thì giá trị của X, Y không đổi, do đó nến đọc tiếp mà không thay đổi giá trị X, Y thì vẫn thu được kết quả cũ.

**II, Giao tiếp GLCD với PIC**

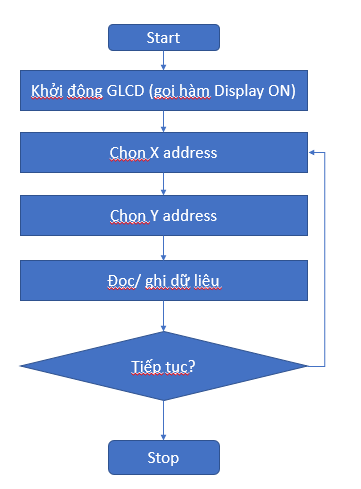
**Sơ đồ nguyên lý:**



**1, Trình tự giao tiếp**

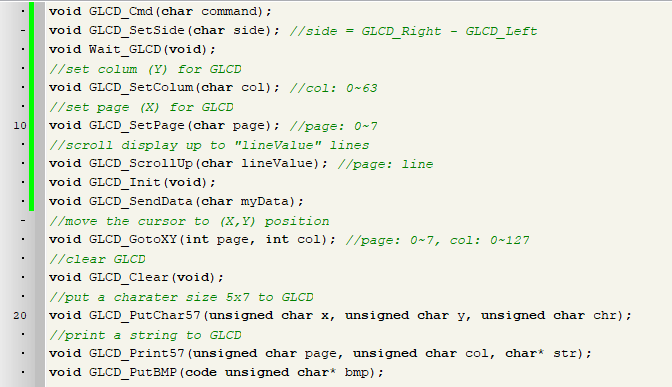
GLCD đơn giản hơn Text LCD vì có ít Instruction hơn, GLCD chỉ có 1 loại bộ nhớ là RAM, tương ứng trực tiếp hiển thị lên màn hình. GLCD chỉ hỗ trợ giao tiếp 8bit. Quá trình khởi động cho GLCD vì vậy đơn giản hơn rất nhiều, bằng cách gọi lệnh Display ON/OFF.

Sau khi khởi động GLCD bằng hàm Display ON, ta có thể set địa chỉ X, Y để ghi dữ liệu. Tuy nhiên tất cả quá trình cần phải thực hiển cho cả 2 chip KS0108.

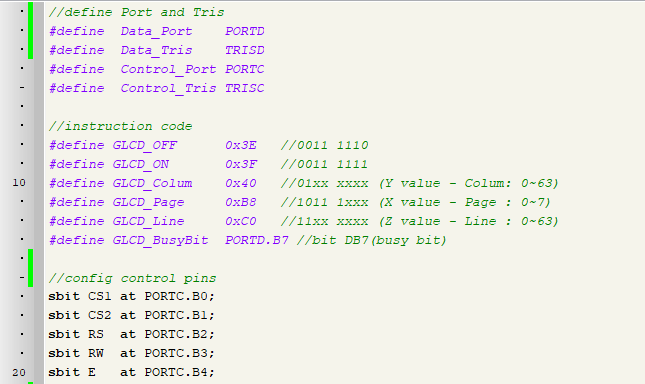


**2, Các hàm cơ bản trong giao tiếp PIC với GLCD**

Chúng ta sẽ cùng tìm hiểu các hàm cơ bản sau khi giao tiếp PIC với GLCD.



Trước khi đi sâu vào các hàm giao tiếp, chúng ta cần khai báo 1 số các macro và biến như sau:



Ở đây, ta sẽ dùng Port D để kết nối đến các chân Data (D7-D0), dùng Port C để kết nối đến các chân điều khiển của GLCD.

Các instruction code sẽ được định nghĩa như những gì chúng ta đã phân tích ở phần các lệnh của chip KS0108.

Các pin của Port C sẽ được chỉ định với gán với các chân điều khiển tương ứng của GLCD như khai báo ở trên và đúng với mạch nguyên lý.

Do GLCD không hỗ trợ bộ font, nên nếu muốn hiển thị các ký tự chúng ta cần định nghĩa chúng bằng 1 bảng font, thư viện “font5x7.h” sẽ được add vào Project. Mình sẽ để file Project ở phía dưới bài viết.

**2.1 Các hàm hỗ trợ**

//GLCD send command function

void GLCD\_Cmd**(**char command**)**

**{**

Data\_Tris **=** 0x00**;** //set data port as output

Data\_Port **=** command**;**

RS **=** 0**;** //RS pin LOW to select instruction mode

RW **=** 0**;** //RW pin LOW to select Write mode

E **=** 1**;**

Delay\_ms**(**2**);**

E **=** 0**;**

Delay\_ms**(**2**);**

**}**

//set side for GLCD function

void GLCD\_SetSide**(**char side**)** //side = 0: right / 1: left

**{**

// Control\_Tris = 0x00; //set port control as output

**if(**side **==** 0**)** //right

**{**

CS2 **=** 1**;**

CS1 **=** 0**;**

**}**

**else** **if(**side **==** 1**)**//left

**{**

CS2 **=** 0**;**

CS1 **=** 1**;**

**}**

Delay\_ms**(**2**);**

**}**

//wait GLCD

void Wait\_GLCD**(**void**)**

**{**

// Control\_Tris = 0x00; //set port control as output

Data\_Tris **=** 0xFF**;** //set port data as input

RS **=** 0**;** RW **=** 1**;** //read RLCD -> PIC

E **=** 1**;**

Delay\_ms**(**2**);**

E **=** 0**;**

**while(**GLCD\_BusyBit **==** 1**)** //GLCD in operatin

**{**

E **=** 1**;**

Delay\_ms**(**2**);**

E **=** 0**;**

**}**

**}**

Hàm **GLCD\_Cmd(char cmd)**, trong hàm này ta sẽ truyền vào 1 tham số là lệnh muốn gửi lên GLCD, chân **RS** và **RW** đều được kéo xuống mức 0, Port Data sẽ được ghi giá trị bởi lệnh ‘cmd’ truyền vào.

Hàm **GLCD\_SetSide(char side)** dùng để trọn nửa trái hoặc phải của GLCD. Nếu ‘side’ = 0 thì sẽ chọn nửa màn hình bên phải để thao tác, nếu bằng 1 thì ngược lại, sẽ chọn nửa màn hình bên trái.

Hàm **Wait\_GLCD()** có tác dụng kiểm tra xem GLCD có đang bận thực hiện 1 quá trình nào đó hay không bằng cách kiểm tra bit BUSY. Nếu bit BUSY = 1 thì GLCD đang bận, nếu bit BUSY = 0 tức là GLCD đang rảnh.

Ở trong hàm này ta sẽ sử dụng 1 vòng lặp while. Nếu bit BUSY = 1 (đang bận) thì vòng lặp tiếc tục với việc tạo ra một xung trên chân EN rồi quay lại tiếp tục kiểm tra bit BUSY. Nếu bit BUSY = 0 thì sẽ thoát vòng lặp while, quá trình chờ kết thúc.

**2.2 Các hàm set địa chỉ**

//set colum (Y) for GLCD

void GLCD\_SetColum**(**char col**)** //col: 0~63

**{**

Wait\_GLCD**();**

Data\_Tris **=** 0x00**;** //set data port as output

RS **=** RW **=** 0**;** //write instruction PIC -> GLCD

GLCD\_Cmd**(**GLCD\_Colum **+** col**);**

E **=** 1**;**

Delay\_ms**(**2**);**

E **=** 0**;**

**}**

//set page (X) for GLCD

void GLCD\_SetPage**(**char page**)** //page: 0~7

**{**

Wait\_GLCD**();**

Data\_Tris **=** 0x00**;** //set data port as output

RS **=** RW **=** 0**;** //write instruction PIC -> GLCD

GLCD\_Cmd**(**GLCD\_Page **+** page**);**

E **=** 1**;**

Delay\_ms**(**2**);**

E **=** 0**;**

**}**

//scroll display up to "lineValue" lines

void GLCD\_ScrollUp**(**char lineValue**)** //lineValue: 0~63

**{**

Wait\_GLCD**();**

Data\_Tris **=** 0x00**;** //set data port as output

RS **=** RW **=** 0**;** //write instruction PIC -> GLCD

GLCD\_Cmd**(**GLCD\_Line **+** lineValue**);**

E **=** 1**;**

Delay\_ms**(**2**);**

E **=** 0**;**

**}**

Hàm **GLCD\_SetColum(char col)** là hàm để chọn cột ta đang muốn để thao tác. Tham số ‘col’ có giá trị từ 0 đến 63.

Hàm **GLCD\_SetPage(char page)** là hàm để chọn page cần thao tác. Tham số ‘page’ có giá trị từ 0 đến 7.

Hàm **GLCD\_ScrollUp(char lineValue)** là hàm cuộn toàn bộ màn hình GLCD lên ‘lineValue’ dòng.Tham số lineValue có giá trị từ 0 đến 63.

* Ta sẽ viết 3 hàm này tương ứng với những gì đã nêu trên ở phần lệnh của chip KS0108.

**2.3 Hàm khởi tạo GLCD và gửi dữ liệu**

//initialize GLCD function

void GLCD\_Init**(**void**)**

**{**

Control\_Tris **=** 0x00**;** //config port control as output

CS1 **=** CS2 **=** 1**;** //select left & right display

Delay\_ms**(**20**);**

GLCD\_Cmd**(**GLCD\_OFF**);** //display OFF

GLCD\_Cmd**(**GLCD\_Page**);** //set X (page = 0)

GLCD\_Cmd**(**GLCD\_Colum**);** //set Y (colum = 0)

GLCD\_Cmd**(**GLCD\_Line**);** //set Z (line = 0)

GLCD\_Cmd**(**GLCD\_ON**);** //display ON

**}**

//GLDC send data function

void GLCD\_SendData**(**char myData**)**

**{**

Data\_Tris **=** 0x00**;** //set data port as output

Data\_Port **=** myData**;**

RS **=** 1**;** //RS pin HIGH to select data mode

RW **=** 0**;** //RW pin LOW to select Write mode

E **=** 1**;**

Delay\_ms**(**2**);**

E **=** 0**;**

Delay\_ms**(**2**);**

**}**

Hàm **GLCD\_Init(void)** để khởi động GLCD. Trước hết ta cần chọn cả 2 nửa màn hình GLCD để thực hiện lệnh. Chúng ta sẽ cấu hình cho các địa chỉ X, Y, Z ở trạng thái ban đầu (đều bằng 0). Hàm này phải được gọi trước khi bắt đầu các thao tác với GLCD trong chương trình chính.

Hàm **GLCD\_SenData(char myData)** cho phép ghi 1byte dữ liệu vào RAM của KS0108. Byte này dùng để hiển thị lên màn hình GLCD. Tham số ‘myData’ là dữ liệu người dùng cần truyền vào.

**2.4 Hàm set địa chỉ X, Y và xóa màn hình**

//move the cursor to (X,Y) position

void GLCD\_GotoXY**(**int page**,** int col**)** //page: 0~7, col: 0~127

**{**

int side **=** 0**;**

side **=** col**/**64**;** //0: right; 1:left

GLCD\_SetSide**(**side**);**

col **-=** 64 **\*** side**;** //update real colum: 0~63

GLCD\_SetPage**(**page**);**

GLCD\_SetColum**(**col**);**

**}**

//clear GLCD

void GLCD\_Clear**(**void**)**

**{**

int page**,** colum**;**

**for(**page **=** 0**;** page **<=** 7**;** page**++)** //clear left display

**{**

GLCD\_GotoXY**(**page**,** 0**);** //start at colum of page

**for(**colum **=** 0**;** colum **<=** 63**;** colum**++)** GLCD\_SendData**(**0**);**

**}**

**for(**page **=** 0**;** page **<=** 7**;** page**++)** //clear right display

**{**

GLCD\_GotoXY**(**page**,** 64**);** //start at colum of page

**for(**colum **=** 0**;** colum **<=** 63**;** colum**++)** GLCD\_SendData**(**0**);**

**}**

**}**

Hàm **GLCD\_GotoXY(int page, int col)** di chuyển con trỏ hiển thị đến địa chỉ (X, Y) ta mong muốn. Tham số ‘page’ có giá trị 0-7, tham số ‘col’ (tức là colum) có giá trị 0-127.

Trước hết, ta cần xác định xem giá trị ‘col’ ta mong muốn thuộc về nửa trái hay nửa phải của GLCD. Biến ‘**side’** được dùng làm tham số để truyền vào hàm **GLCD\_SetSide()**. Nếu col < 64 thì vị trí của nó thuộc về nửa phải (side = 0), ngược lại, nếu col > 64 thì vị trí của nó thuộc về nửa trái (side = 1).

Câu lệnh **col -= 64 \* side;** sẽ cập nhật lại địa chỉ Y chính xác để truyền vào hàm **GLCD\_SetColum().**

Hàm **GLCD\_Clear(void)** thực hiện thao tác xóa toàn bộ màn hình GLCD.

**2.5 Các chương trình con mở rộng**

//put a charater size 5x7 to GLCD

void GLCD\_PutChar57**(**unsigned char x**,** unsigned char y**,** unsigned char chr**)**

**{**

unsigned char i**;**

**if((**y**>**59**)** **&&** **(**y**<**64**))**

**{**

GLCD\_GotoXY**(**x**,** y**);**

**for(**i **=** 0**;** i **<** 64**-**y**;** i**++)**

GLCD\_SendData**(**my\_font5x7**[((**chr **-** 32**)\***5**)** **+** i**]);**

GLCD\_GotoXY**(**x**,** 64**);**

**for(**i **=** 64 **-** y**;** i **<** 5**;** i**++)**

GLCD\_SendData**(**my\_font5x7**[((**chr **-** 32**)\***5**)** **+** i**]);**

**}**

**else**

**{**

GLCD\_GotoXY**(**x**,** y**);**

**for(**i **=** 0**;** i **<** 5**;** i**++)**

GLCD\_SendData**(**my\_font5x7**[((**chr **-** 32**)\***5**)** **+** i**]);**

**}**

**}**

//print a string to GLCD

void GLCD\_Print57**(**unsigned char page**,** unsigned char col**,** char**\*** str**)**

**{**

unsigned char i**,** j**;**

j **=** col**;**

**for(** i **=** 0**;** str**[**i**]** **!=** 0**;** i**++)**

**{**

**if(**j**>=**128**)**

**{**

col **=** 0**;**

j **=** col**;**

page**++;**

**}**

GLCD\_PutChar57**(**page**,** j**,** str**[**i**]);**

j **+=** 7**;**

**}**

**}**

void GLCD\_PutBMP**(**code unsigned char**\*** bmp**)**

**{**

unsigned char line**,** col**;**

unsigned int i**;**

**for(**line **=** 0**;** line **<** 8**;** line**++)**

**{**

GLCD\_GotoXY**(**line**,** 0**);**

**for(**col **=** 0**;** col **<** 64**;** col**++)** //left

**{**

i **=** 128 **\*** line **+** col**;**

GLCD\_SendData**(**bmp**[**i**]);**

**}**

GLCD\_GotoXY**(**line**,** 64**);**

**for(**col **=** 64**;** col **<** 128**;** col**++)** //right

**{**

i **=** 128 **\*** line **+** col**;**

GLCD\_SendData**(**bmp**[**i**]);**

**}**

**}**

**}**

Đây là 3 chương trình con cuối cùng mà chúng ta sẽ tìm hiểu trong bài viết này. Trong đó có 2 hàm in các ký tự có size 5x7 (5 cột, 7 dòng) được định nghĩa trong thư viện ‘font5x7.h” và 1 hàm in toàn bộ màn hình GLCD với một hình ảnh kích thước 128x64.

Hàm **GLCD\_PutChar57(unsigned char x, unsigned char y, unsigned char chr)** cho phép in 1 ký tự có mã ASCII là biến ‘chr’, biến ‘x’ là địa chỉ X (0 đến 7), biến ‘y’ là địa chỉ Y (0 đến 127). Đặc biệt trong hàm này, chúng ta phải xử lý trường hợp 1 ký tự được in ra đúng vào “phần biên giới” của 2 nửa màn hình.

Vì 1 ký tự có độ rộng là 5 cột (5 bytes), giả dụ ta muốn in 1 ký tự ở cột 60, thì các byte 0, 1, 2, 3 nằm ở cột 60, 61, 62, 63, còn byte 4 phải nằm ở cột 0 của nửa màn hình còn lại (tức là cột 64 của toàn bộ màn hình GLCD).

Ta sẽ sử xét trường hợp này bằng 1 hàm **if((y>59) && (y<64))**, nếu cột lớn hơn 59 và nhỏ hơn 64 thì tức là sẽ có 1 sự chuyển tiếp 2 bên màn hình ở đây. Chia quá trình in thành 2 vòng for, vòng for thứ nhất in từ vị trí ‘y’ đến vị trí cột của nửa trái và vòng lặp for thứ hai in từ cột đầu tiên của nửa màn hình bên phải đến byte cuối cùng của ký tự cần in.

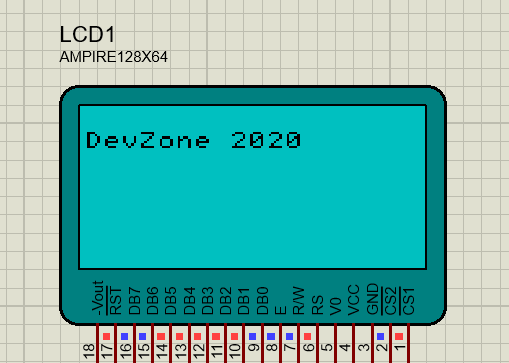
Nếu không có sự chuyển tiếp xảy ra, ta in ký tự bình thường.

Hàm **GLCD\_Print57(unsigned char page, unsigned char col, char\* str)** cho phép chúng ta in 1 chuỗi ký tự bất kỳ lên GLCD. Trong hàm này, nếu in quá 1 dòng thì các ký tự ở sau sẽ tiếp tục được in ở dòng dưới.

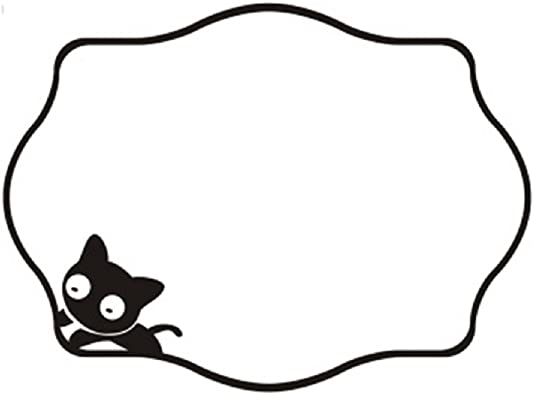
Hàm **GLCD\_PutBMP(code unsigned char\* bmp)** thực hiện in 1 hình ảnh có size 128x64 được định nghĩa trước lên toàn bộ màn hình GLCD, quá trình này sẽ là in đè.

**3. Hình ảnh kết quả dựa trên mô phỏng**

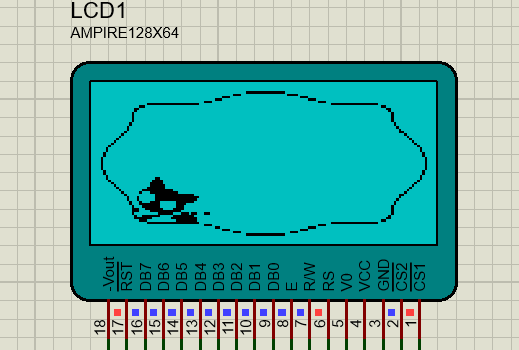
**Hiển thị dòng chữ “DevZone 2020”:**



**Hiển thị hình ảnh trên GLCD:**



Hình ảnh gốc



Hình ảnh mô phỏng

Link Project: https://bit.ly/2CEqGne