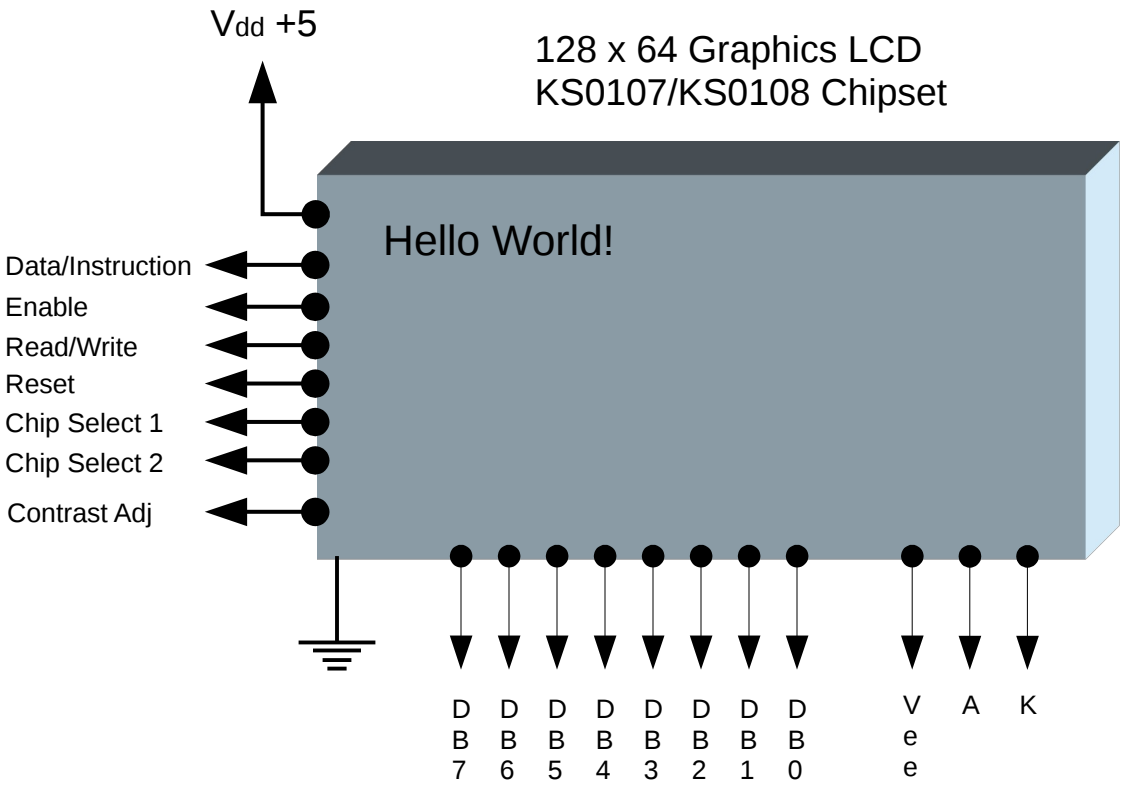
Graphics LCD Display C Library for the PIC18F4550 MCU (or similar advanced microcontrollers)

Version 1.00 – 5/2013 **www.muniac.com**



Introduction:

Many MCU applications may benefit from incorporating a graphics LCD output device. Aside from providing valuable debugging output of program variables and execution information, a GLCD can enhance a system with graphics, menus, text and process information. If you have searched the Internet you'll find much work has been done to provide software tools for interfacing a 128x64 GLCD. Everything from free casual hobby efforts to $500 professional packages are available. I was looking for a well documented package of basic functions in and around the $50-$100 price range. My searching didn't turn up exactly what I was looking for. $500 was too expensive and the free available options weren't documented well enough and/or didn't do what I was looking for. I decided to develop my own package and offer it to others looking for a reasonably priced, well documented library of basic GLCD routines. The goal is to save system developers and software engineers valuable time which would be better spent writing application specific code solving their own engineering problems. The value of purchasing an already developed package depends on how much you believe your time is worth. What's been done here isn't unique, cutting edge or profound. And anyone with the skills willing to invest the time and effort could reproduce this package. If the time and energy needed to undertake this kind of task are in short supply and/or a project deadline is tight, what's presented here has time saving value.

My MCU project(s) centers around Microchip's PIC18F4550 using a common 128x64 GLCD. I purchased my GLCD from Adafruit and it's a generic module based on Samsung's KS0107/KS0108 chipset. Sadly the device didn't come with very good documentation which sent me back searching the internet for basic things like pin out functions, timing, layout and control commands. I located a document covering Vishay's model LCD-128H064A which I used successfully in developing my application. Snippets of code were located to provide examples of timing, control and commands which I used as the basis for developing my GLCD library. All of this was helpful albeit time consuming. A secondary goal is to make this all available in one place for others wishing to use a GLCD. It's the documentation package I wished I had when I got started. Hopefully it saves the next system designer time, frustration and avoids needless aggravation. It's all about working efficiently, doing a good job, getting it done on time, learning, gaining experience and moving forward from there. As for software tools, I think there is always room for quality products that remain affordable. It is hoped that this package delivers in these areas.

This documentation package includes an ***Introduction***, ***GLCD Function Reference***, ***Installation Guide*** and ***Schematics***. Together they provide detailed information about all aspects of using the GLCD library from the GLCD itself through wiring and writing C code. A clear understanding of what's written herein assumes the reader possesses a basic knowledge of C and MCUs. Understanding simple CMOS circuits and digital logic blocks is also helpful.

Even though this project was developed on a PIC18F4550, the source code can be easily adapted to any PIC with adequate ROM/RAM memory and enough I/O pins. Obviously a companion development environment is required to compile and link the C source code. A PIC programming tool will need to be used to load the final program. Microchip's MPLAB IDE and IDC 3 were used for this project. The tools have worked well running under Windows XP as a guest O/S under Oracle's VirtualBox virtual machine. The host O/S was Linux Mint 14 running on a Dell Inspiron 1501 PC. The documentation was written in LibreOffice Writer Version 3.6.2.2 (Build ID: 360m1(Build:2)).

128x64 GLCD Description/Interface:

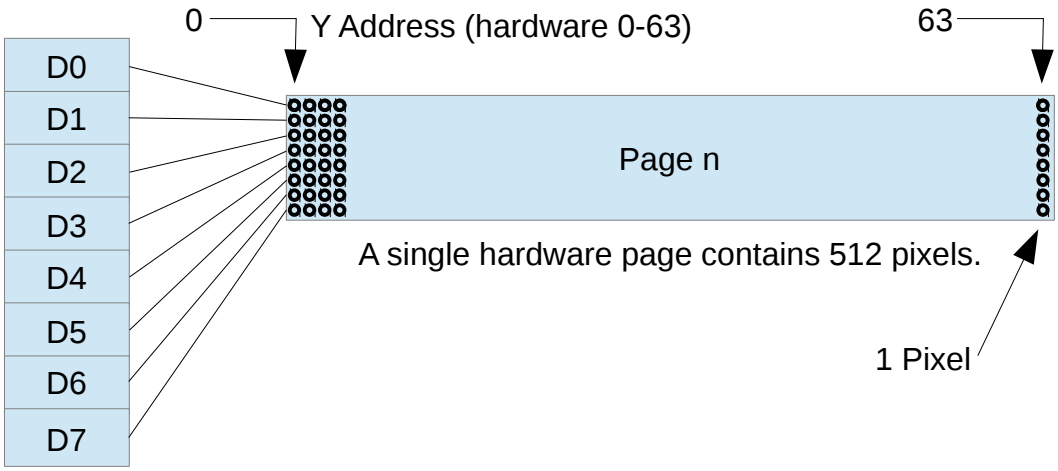
The Adafruit GLCD (Part #253) is a fairly generic module. Consult their website for a picture and general information about this display. It comes mounted to a PC board with 20 through holes spaced 0.1” apart. The LCD itself is back lit by a series of LEDs. Both the display and LEDs can be powered from a single +5 DC supply. The quality of the display is good considering its matrix is only 128x64 pixels. Even at this low resolution, very readable text and graphics output can be achieved. Keep in mind a graphics LCD does not have a character generator. All it can do is illuminate any one of 8,192 pixels. Creating meaningful displays requires illuminating groups of pixels in predefined ways. To display characters you'll need font description tables that map each character into a matrix appropriately sized for a given typeface. Graphic primitives don't require tables and are generated by algorithms in software. Being able to control each pixel to generate symbols (characters and graphics) comes at the expense of increased complexity of the software needed to interface the device. Applications requiring only a couple of lines of text can select much simpler LCD options that cost less. Consider these as better options if graphics aren't required.

A Typical 128x64 GLCD Layout

GLCD is organized into a left and right panel each containing 8 pages.

Assuming the GLCD is oriented in landscape view, it's organized as left and right panels. Each of these panels is 64x64 pixels and is controlled by its own chip. The virtual line that divides the two panels runs vertically right in the middle of the display. With proper software control, the panels seamlessly integrate into one 128x64 pixel grid. Setting up the interface so the individual panels function as a single 128x64 pixel matrix requires some special programming. The idea is to make the display easy to use and remove the application from needing to manage the hardware. In essence you'd want the GLCD to look like an output device written to by a collection of intuitive function calls. Such is the aim of the GLCD library and all its function calls. The diagram above shows how the GLCD hardware organizes the display space into panels and pages. Pixels are accessed through setting/clearing bits in one of the 1024 data bytes stored in GLCD RAM. A single byte is the smallest unit of accessible storage. The page mapping is shown below:

A Single GLCD Page Layout



Pixel column byte (1024 total).
1=On, 0=Off

A pixel is located by selecting a chip, page #, Y address and a bit position.

From the two diagrams on pages 3 and 4 you can see how the GLCD display is organized. The device has an internal memory which consists of 1024 bytes. Each byte represents a vertical column of 8 pixels. Setting the corresponding bits (1) turns on a pixel and clearing bits (0) turns off pixels. You must supply the GLCD with a chip select, page # and Y address to specify the location of a single byte. Writing this byte into the GLCD's memory automatically sets/clears 8 vertical pixels at the location specified with chip select, page and Y address. The results of this appear immediately on the screen. Bytes may be read back as well. Writes and reads will automatically advance the Y address by 1. Controlling the GLCD is done through six hardware control lines and sending commands. The GLCD discussed here has the following control lines:

1) Data/Instruction pin 4 (D/I sometimes referred to as R/S)
2) Read/Write pin 5 (R/W GLCD memory)
3) Enable pin 6 (H->L Strobe pulse)
4) Chip Select 1 pin 15 (Selects left panel)
5) Chip Select 2 pin 16 (Selects right panel)
6) Reset pin 17 (Hardware reset active low)

This is pretty standard with all 128x64 GLCDs that use the Samsung KS0107/KS0108 chipset. In addition to the 6 control lines there are 8 data lines (DB0-DB7), 2 lines for back lighting LEDs, power, ground, negative voltage out and contrast adjustment. This makes a total of 20 pins and all of them will be used. The instruction table below summarizes the control line and data byte settings for each of the display commands/instructions:

128x64 GLCD Display Control Instructions

(Samsung KS0107/KS0108 Chipset)

Instruction	D/I or R/S	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display On/Off	L	L	L	L	H	H	H	H	H	L or H	Controls the display on or off. Internal status and display RAM data are not affected. L=Off, H=On
Set Y Address	L	L	L	H	Y Address 0-63						Sets the address in the Y addr reg.
Set Page Number	L	L	H	L	H	H	H	Page (0-7)			Sets the page # in the page addr reg.
Set Display Top Line Used For Scrolling	L	L	H	H	Display Start Line # (0-63)						Controls which pixel line the display will use to begin the refresh from RAM. This is used for hardware scrolling.
Read GLCD Status Byte	L	H	Busy	L	On or Off	Reset	L	L	L	L	Read the status as follows: Busy-L=No, H=Yes On/Off-L=On, H=Off Reset-L=Normal, H=Reset
Write GLCD Data Byte	H	L	Write 8 Bit Pixel Data Byte Into RAM (Must specify chip, page and Y address)								Write DB0-DB7 into display RAM. Y Addr = Y Addr + 1 (after write)
Read GLCD Data Byte	H	H	Read 8 Bit Pixel Data Byte From RAM (Must specify chip, page and Y address)								Read DB0-DB7 from display RAM. Y Addr = Y Addr + 1 (after read)

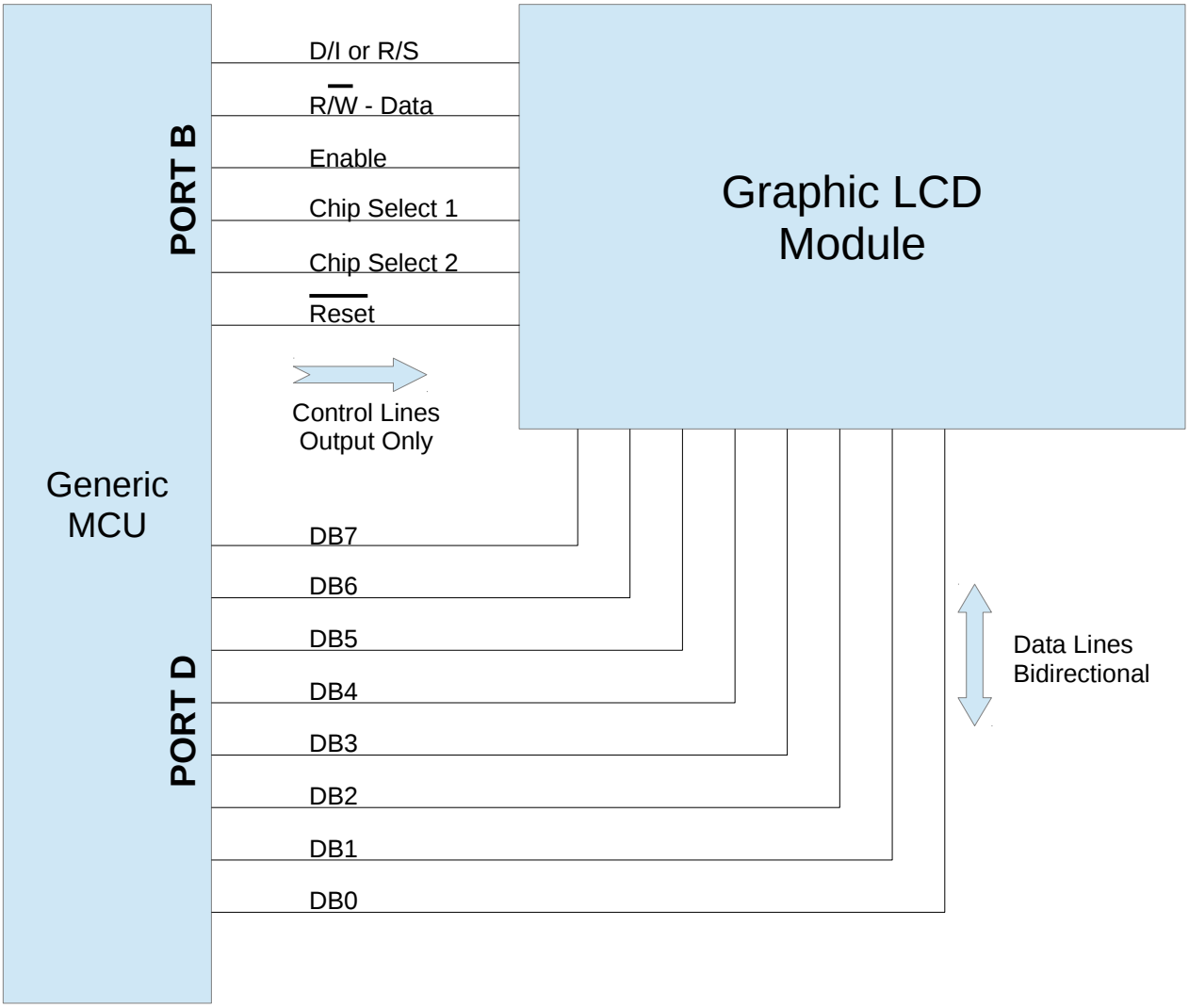
The above table is the heart of the GLCD library of functions. It is through this simple instruction set that all the display features/functions may be exercised. There are four major categories of control established by the R/S (or D/I) and R/W lines. In the case of both these lines being low (0), DB0-DB7 determine whether the instruction is **display on/off**, **set Y address**, **set page number** or **set display top line**. These bits also provide the display with data. The other three combinations are as documented in the instruction table.

Wiring Considerations:

Beyond the basic power, ground, back lighting LEDs and contrast connections interfacing the GLCD to a MCU requires managing 6 control and 8 data lines in software. The control lines will always be output while the data lines are bidirectional for reads and writes. This may influence which ports should be used. The easiest way of interfacing is dedicate one port for the control lines and another port for the data. That will require 14 dedicated I/O pins from the MCU. With special programming, control and data lines can be split across 2 ports. Additionally, if there simply aren't enough pins available, the control lines can be managed through a serial in parallel out shift register without any appreciable loss of performance. This will reduce the port pin count to 11 but requires extra circuitry. Refer to ***Schematics*** for more information and suggestions. A shift register arrangement could also be used to handle the data lines. Such a scheme would need to be bidirectional. The freeing up of another 4 port pins comes at the expense of slower display performance. Since MCU's are relatively inexpensive, one could be dedicated to managing the GLCD and an acceptable I/O interface be chosen from the MCU managing the GLCD to the outside world. If this were done, conserving the port pins would be less of a problem. Lots of options exist and the best one depends on the final application. The details of which are beyond the scope of this introduction. Whatever option is chosen, software is required to manage the control and data lines properly as well as issuing the commands required to instruct the GLCD. For the purposes of this introduction we'll assume the control and data lines have been connected to theoretical ports B and D as follows:

Example Generic Hookup For A GLCD

(Only Control And Data Lines Shown)



In a pinch, RESET can be tied to Vdd and the functions normally performed by a hardware reset can be done in software. The performance overhead for this is very small. A port pin can thus be saved at the expense of a few extra lines of code. Wiring specific control and data lines to the port pins establishes a relationship between those functions and bit positions within the control and data bytes passed between the MCU and GLCD. For example, hard wiring the D/I (or R/S) control line to logical pin one of port B would mean its low order bit would toggle that line. Writing a 0x01 to this port would drive the D/I (or R/S) line high. Writing a 0x00 would drive it low. Six individual bits (5 if reset is tied to Vdd) would thus toggle all the control lines (0x01, 0x02, 0x04, 0x08, 0x10, 0x20). ORing these together would set multiple bits. Clear bits with ANDing in their compliment. Port B bits 6 and 7 are not used (5 too if reset is tied to Vdd) for control and thus could be used for something else. The data lines would be wired to port D in order of magnitude so DB0 is on the port's least significant bit and DB7 is on the port's most significant bit. Port D needs to be bidirectional as it will send and receive data from the GLCD. Again the ports chosen are MCU and application specific. B and D have been chosen to illustrate this example.

The GLCD chosen for this project outputs a negative voltage intended to drive the contrast feature. A 10k ohm variable resistance is connected between this negative voltage and Vdd. The wiper goes to the contrast adjustment pin on the GLCD. This is important for proper operation of the GLCD. Power also needs to be supplied to the back lighting LEDs. Connect these to Vdd through a current limiting resistance of 300 ohms. Refer to **Schematics** for more details.

The ENABLE strobes instructions in and data in to or out of the data lines. Check the specific GLCD data sheet for the transition logic, pulse width and timing. It isn't uncommon to apply 5 and 10 microsecond delays to this signal to allow for data setup times on MCUs running at 20 MHz. Improper timing will result in a garbled display and/or no communications at all. Refer to the library source code for a working implementation and example. The ENABLE line needs to be toggled independent from the other control signals. A single C function has been included to manage the ENABLE line.

Once the data and control lines are hooked up and working you can begin sending and receiving data from the GLCD. For example, driving the D/I (or R/S) and R/W lines low followed by writing 0x3f to port D followed by a proper ENABLE pulse will turn the GLCD on. That same sequence with port D being 0x0e will turn the GLCD off. Consult the commands/instructions table on page 6 to see how this was derived.

The GLCD Library:

The GLCD library is a collection of over 30 C callable functions designed to handle the control and data lines of the GLCD. The software also manages all the GLCD commands/instructions required to output fonts and graphics. All the commanding, control, mapping, data flow and timing are handled within these functions eliminating the need to deal with the hardware's nuts and bolts details. The easy to use function calls allow an application to quickly gain the benefits of a GLCD. The library comes with complete well commented source code, installation instructions and a batch file for easy creation of a link library. Helpful tips are also included as well as a demo program that uses all the features of the library. Well commented header (.h) files are also part of the package. Consult the ***GLCD Library Reference*** for more details about the specific function calls. The ***Installation Guide*** covers compiling the source code, installation, header files, modifications, creating a library and how to use the functions

in your project. GLCD_Demo.c is a good example of how the GLCD functions are called in an actual application. ***Schematics*** presents helpful circuit diagrams that cover all the details of wiring and several example CMOS logic interfacing methods.

MUNIAC, LLC
scott@muniac.com