Driving “Dumb” Graphic LCD Panels with PIC Controller

Donnaware International LLP

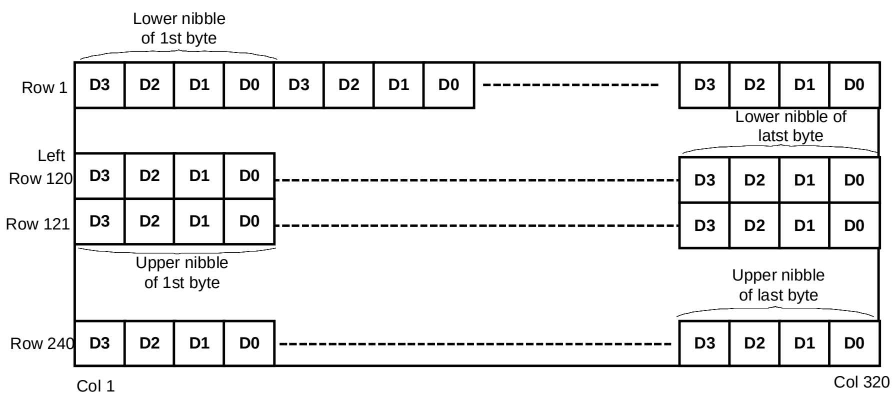
Introduction:

Many graphic LCD panels do not come with an intelligent controller, and in general, to use them with something like a PIC microprocessor you would need a controller chip like the Epson SED1335 or better. Some Graphic LCD panels come with an on board controller, but that type typically cost quite a bit more (in the $50 to $100 range).

However, with a little clever coding, we can drive a cheapie “dumb” LCD panel directly with a PIC16F876 (with just a small bit of processing power to spare). The panel used here is the INMP001 available from Vetco Electronics (http://www.vetco.com/) for only $4.99 ea. This is a 320x240 pixel monochrome panel with EL backlight. This unit has on board EL power control and LCD bias generator so it runs off a single ended +5Vdc supply and +6 to +12VDC for the backlight.

Theory of operation:

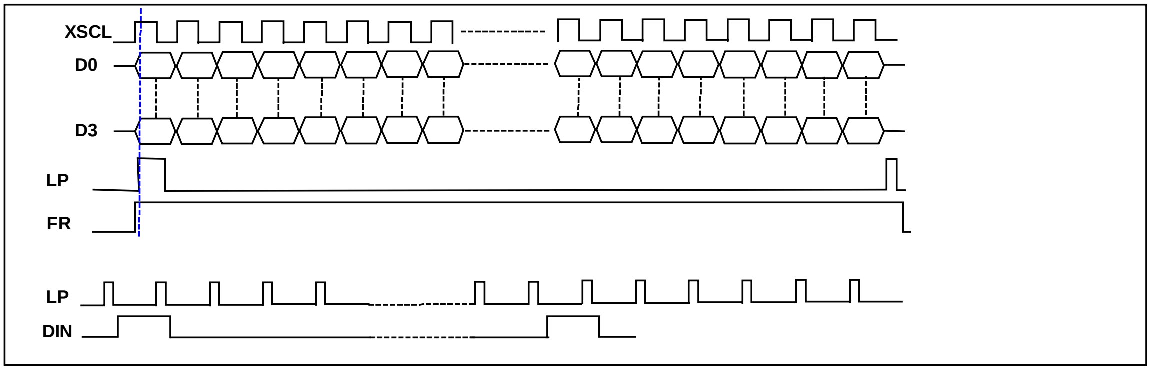
First, a little about the theory behind graphic LCD’s; For an LCD pixel to maintain the “on” state, it has to be constantly refreshed (just like a CRT screen). So the driver circuitry has to constantly scan through the entire display at a refresh rate usually something in the 30-70HZ range to be flicker free.



The way this works for this particular panel (and this is a fairly common type of interface) the screen is accessed by using a 4 bits data interface. The horizontal part of the screen is formed by these four bits as shown above. So for a 320 pixel row you only need 80 lines. The screen layout is shown above.

The pin out of the panel is shown on the spec sheet in Appendix B (these specs were obtained by examining the chips on the back of the LCD, I was unable to find the actual spec sheets from who ever the manufacturer is). The interface is made up of the data bits referred to above, plus there is an X Clock (XSCL) to clock in the horizontal data, a Line Latch (LP) pulse, and a vertical frame pulse (DIN) and an AC Waveform.

The timing diagram below shows the relationship of these signals. This information is also supplied in the spec sheet shown in Appendix B.



The difficulty in driving a panel like this is in generating the XSCL signals and providing valid data on the D0-D3 data lines at a rate fast enough to keep up with the timing requirements of the display.

Timing Calculations:

To achieve a frame rate that is fast enough to avoid flicker, we need at least 30 frames per second (faster is usually better, up to a point, too fast and the display will be blank (no chance of that with a 20Mhz PIC). This display has a 320x240 format, but we actually clock in a nibble at a time in the horizontal direction as mentioned above. So we really only have to scan 80 nibbles to get 1 scan line. So let's assume we use a section of code such as this

```
HLoop: Bcf          CAS             // CAS Low to clock column
       Bsf          CAS             // CAS High
       Decfsz       DRAMAddr,F      // Decrement x counter/DRAM Address
       Goto         HLoop           // Repeat until done
```

We increment 1 nibble in the horizontal direction and then strobe the Column address strobe line of the DRAM chip (which it turns out, we can also use this same line to clock the data into the LCD, see circuit diagram on next page). We will simply create a loop to toggle the CAS line 80 times while decrementing the address register to create the subroutine to make 1 horizontal scan line.

Assuming we are using a 20Mhz crystal with our PIC, that actually yield an instruction rate of 5Mhz (the crystal is always divided by 4 on the PIC to get the instruction speed). So the actual time to do 1 line is as follows:

Line time = 1/5Mhz * 4 Instructions * 80 nibbles => 64us

Now to compute the frame rate, we just take the above and multiply it by the number of lines. Now there is also obviously some over head for looping and so on, but this is a good approximation.

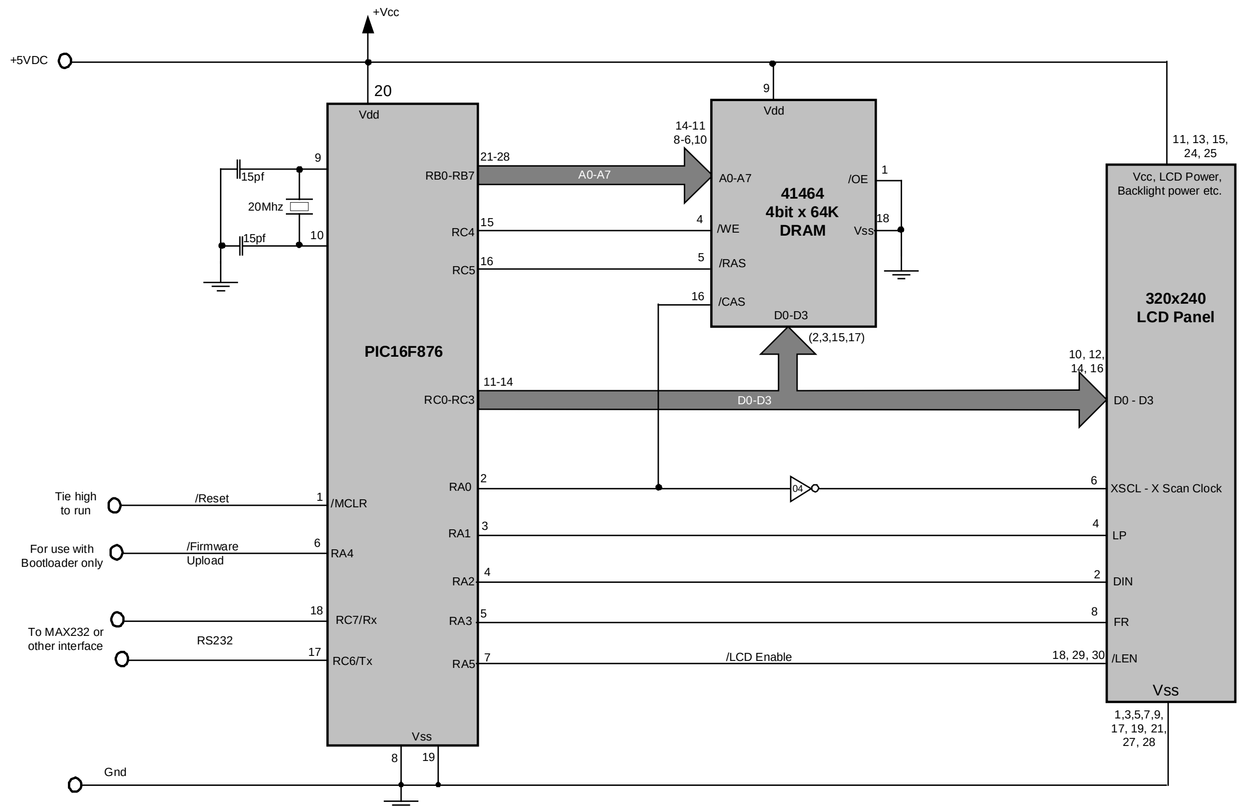
Frame Rate = 240 * (64us + 6us processing overhead) = 16.8 ms
= 16.8ms/frame => ~60hz rate (NO Flicker, yeah !)

This approach provides an adequate frame rate with no flicker and there should be enough processor time between frames to provide some processing (just a little).

Circuit Description:

To make use of one of these “dumb” LCD displays, you really need to add some video RAM of some type, otherwise, you have to constantly be computing or deriving the data to send to the LCD. By using the RAM, you simply write the image data to the RAM and then the PIC just sits there looping through all the cells of the RAM reading it back out to the LCD to keep the screen active.

The simplest way to solve the problem is to use Dynamic RAM. Now, I know what everyone always says “gee what about refresh ?”. Well, the beauty part of this application is that you do not have to worry about it because as long as you continue to read through all the memory addresses over and over again, that does the job. With all modern DRAM’s, they have a hidden refresh that is done automatically any time you read or write to the DRAM. In this application, we are constantly looping through all the addresses, so there is no need to concern ourselves with any type of refresh routines.



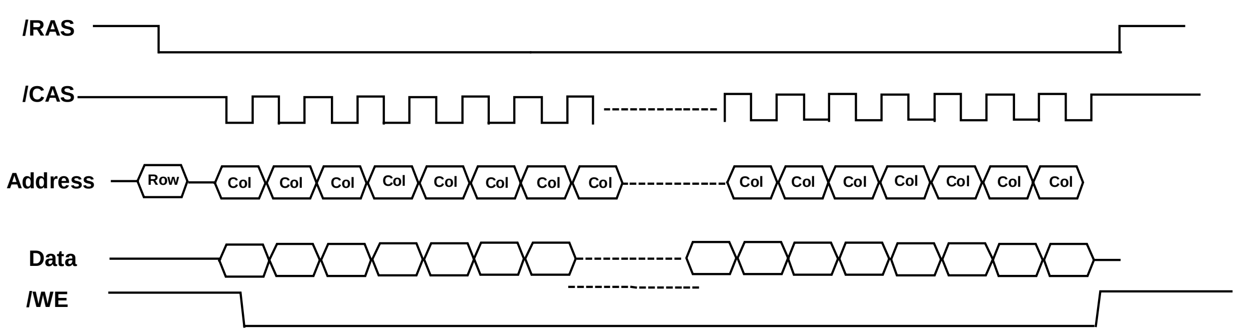
The diagram above shows how to hook up DRAM to a PIC. A single 64Kx 4 bit DRAM (eg. HM41464) is used as image storage (just like a VGA controller). In fact, the chip I used (HM41464) is very common on older VGA cards. I pulled these off of an old ISA type VGA in my junk box, in this case the “expansion” chips were even socketed so it was easy to pull them. (Full schematic provided in appendix A)

The CAS line of the DRAM is simply inverted to form the XSCL (Horizontal scan clock) for the LCD. The LP line is pulsed high at the end of each line to latch the data into the LCD. The DIN line is pulsed high at the beginning of each frame to indicate the start of the vertical scan. The FR is the AC waveform to drive the LCD. This waveform alternates high and low on alternating frames. This tells the LCD to switch polarities ever other frame to prevent frying the LCD crystals (if you leave the same polarity on LCD crystals too long, it destroys them).

Here is the code snippet of the frame scan code to "light" the display:

```
//----------------------------------------------------------------
//  Make 1 Frame:
//----------------------------------------------------------------
#asm
            Goto        DoFrame             // Go do the frame
            //----------------------------------------------------------------
            //  Make 1 horizontal Line:
            //----------------------------------------------------------------
HorzLine:   Movlw       SCANW               // Get LCD Width (decimal 80, 1/4 of 320)
            Movwf       DRAMAddr            // Set DRAM Address lines
CasLoop:    Bcf         CAS                 // CAS Low to clock column
            Bsf         CAS                 // CAS High
            Decfsz      DRAMAddr,F          // Decrement x counter
            Goto        CasLoop             // If not then continune on
            Return                          // Return to caller
            //----------------------------------------------------------------
            //  Start of Do Frame Routine:
            //----------------------------------------------------------------
DoFrame:    Movlw       0x08                // Frame bit, AC Waveform
            Xorwf       PORTA,F             // Toggle Frame Bit
            //----------------------------------------------------------------
            // First line is special, clicks in the frame start bit
            //----------------------------------------------------------------
            Movlw       SCANH               // Get LCD Data Height (240)
            Movwf       r_count             // Save as row address
            Movwf       DRAMAddr            // Output it to the DRAM Address lines
            Bcf         RAS                 // Pulse RAS to DRAM, leave down for page mode
            Call        HorzLine            // Do 1 Horz line
            Bsf         RAS                 // Done with Load RAS to DRAM
            Decf        r_count,F           // Decrement row counter
            Bsf         XLP                 // Horizontal Sync Pulse
            Bsf         DIN                 // Vertical sync low to signal start of frame
            Bcf         XLP                 // Horizontal Sync Pulse
            Bcf         DIN                 // Vertical sync low to signal start of frame
            //----------------------------------------------------------------
            // Frame Loop
            //----------------------------------------------------------------
FrameLoop:  Movf        r_count,W           // Get the current row address
            Movwf       DRAMAddr            // Output it to the DRAM Address lines
            Bcf         RAS                 // Load RAS to DRAM, leave down for page mode
            Call        HorzLine            // Do 1 horz line
            Bsf         RAS                 // Done with Load RAS to DRAM
            Bsf         XLP                 // Horizontal Sync Pulse
            Bcf         XLP                 // Horizontal Sync Pulse
            Decfsz      r_count,F           // Decrement row counter
            Goto        FrameLoop           // If not done, then continune on
            //----------------------------------------------------------------
            // Exit Routine
            //----------------------------------------------------------------
#endasm
```

Reading and writing data to the DRAM is really very straight forward. You simply put the data on the data pins and then toggle in the Row and Column addresses. By examining the spec sheet of the DRAM chips we find another really helpful feature, called page mode. Page Mode DRAM means that we can assert the /RAS line and then sequentially read or write to the column addresses within that Row or "Page". The following timing diagram is an example of how that works:



The next thing is how to get the data into the DRAM. The simplest thing is to just load it between frames. This seems like it would be really slow, and it is, but for most applications, being able to change 4 bits once every 16ms is not that bad, most applications the display data is fairly static. For higher speed displays again, refer to the "AN-LG68 – Driving 640x480 LCD.pdf" app note.

To load the data, the same approach is used, the Row and Column addresses are strobed in, but in this case, the /WE lead is exerted first which puts the DRAM into write mode.

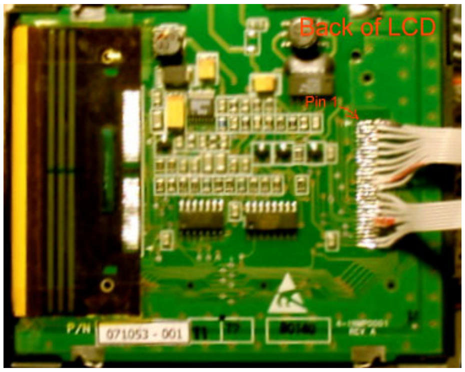
The following snippet of code shows how to access DRAM chips from the PIC. For this application, all code was developed using CCS C compiler, only the code above for refreshing the LCD screen is done in assembler (for maximum speed).

```
/*-------------------------------------------------------------------------*/
/*   Write 1 Nibble to DRAM                                                 */
/*-------------------------------------------------------------------------*/
void WriteNibble(int row, int col, int data)
{
    int value;

    Output_Low(DRAMWE);                  // DRAM Write Disabled
    set_tris_c(0b10000000);              // Port C  output

    Output_B(row);                       // Put row Address onto port B
    Output_Low(DRAMRAS);                 // DRAM Row Address Strobe

    Output_B(col);                       // Put Column address on port B
    value = Input_C() & 0xF0;            // Get Port C Value, mask off lower
    Output_C((data&0x0F) | value);       // Put data on port C
    Output_Low(DRAMCAS);                 // DRAM Column Address Strobe Disabled
    Output_High(DRAMCAS);                // DRAM Column Address Strobe Disabled

    Output_High(DRAMRAS);                // DRAM Row Address Strobe Disabled
    Output_High(DRAMWE);                 // DRAM Write Disabled
    set_tris_c(0b10001111);              // Port C input (hi-z)
}
```

In this example, the row address is clocked in and then the /WE line is held low will clocking in a number of column addresses while valid data is presented on the data lines. A similar sequence is used with the /WE line high read a byte. The /OE line is actually redundant since the outputs can be completely controlled with the /RAS and /CAS lines. In this application the /OE line is simply tied low.

Putting it all together:

To assemble the circuit, it is important to make the proper connections to the LCD. This LCD comes with a ribbon cable attached that has a funky connector on it, but that particular connector is hard to find. So I removed the cable and soldered directly to the board as shown in the picture to the right. Pin 1 had a small white arrow next to it. The pin out listed in Appendix B is in reference to this pin out, not the connector on the end of the funky ribbon cable.



After assembling the circuit, the final step is the complete code. The complete code listing is shown in Appendix D. In this case, the software is designed to simply check the status of the RS232 receive bit and if nothing is received, then it just keeps looping to refresh the screen. The only time it stops is to compute something if data comes in on the RS232 line.

The sample application will allow you to clear the screen and upload a BMP from your PC, set and get pixels. Donnaware Appnote's AN-L64 – Driving640x400.PDF, AN-256, ANL68 etc, show how to add the functions to draw lines, circles, fill and generate text. To really make this an effective controller, one would really need to set up an interrupt service routine to service the screen refresh while performing the graphics functions. This approach is illustrated in AN-LG68 and ANA0674 (for ECM-A0674 LCD).

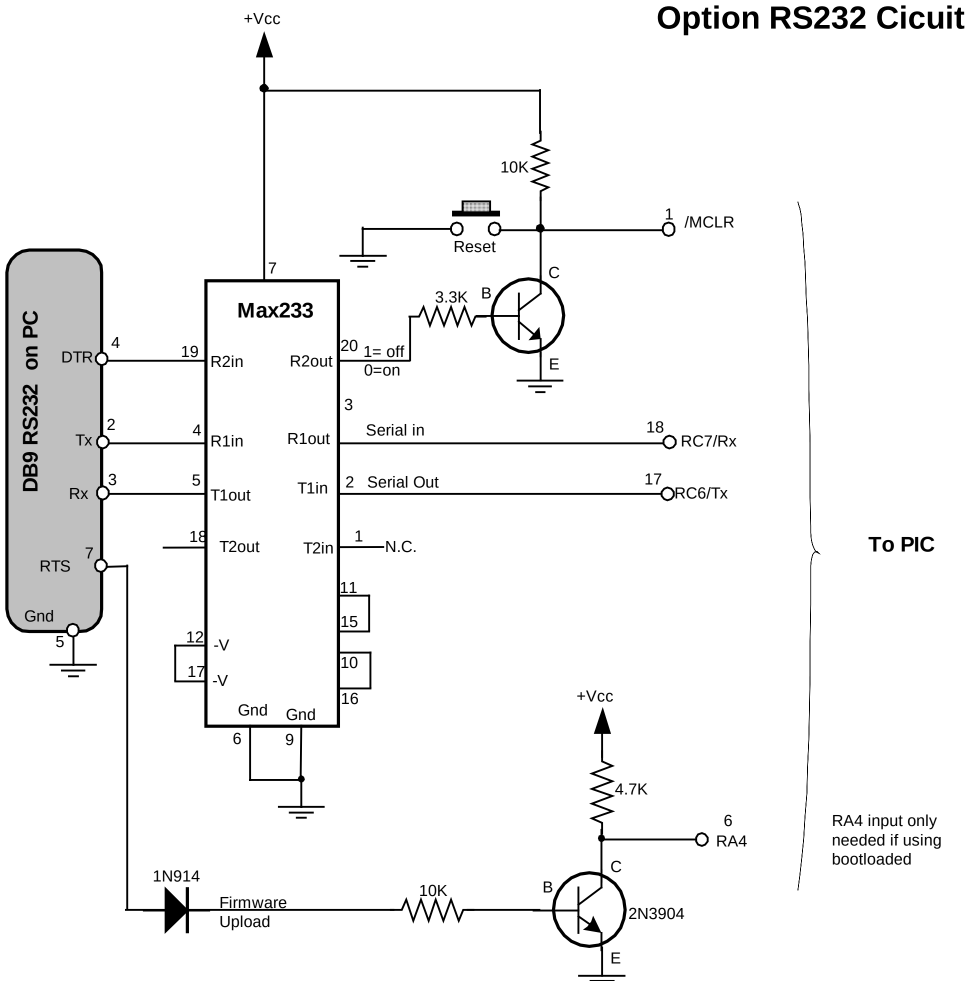
Notes regarding RS232:

This circuit uses TTL Level RS232 interface (DO NOT CONNECT DIRECTLY TO PC COM PORT !). To interface to the PC COM port, you need to add the good ole MAX232 . You also need to set the BRGH constant on the PIC to the appropriate value to match you PC settings. The best use for this circuit is to interface to another PIC, you can just make the Baud rate generator constants the same between the two and hook the RS232 up directly without bothering with the MAX232 level shifters. That way one PIC runs the display while the other can perform whatever functions your project calls out for.

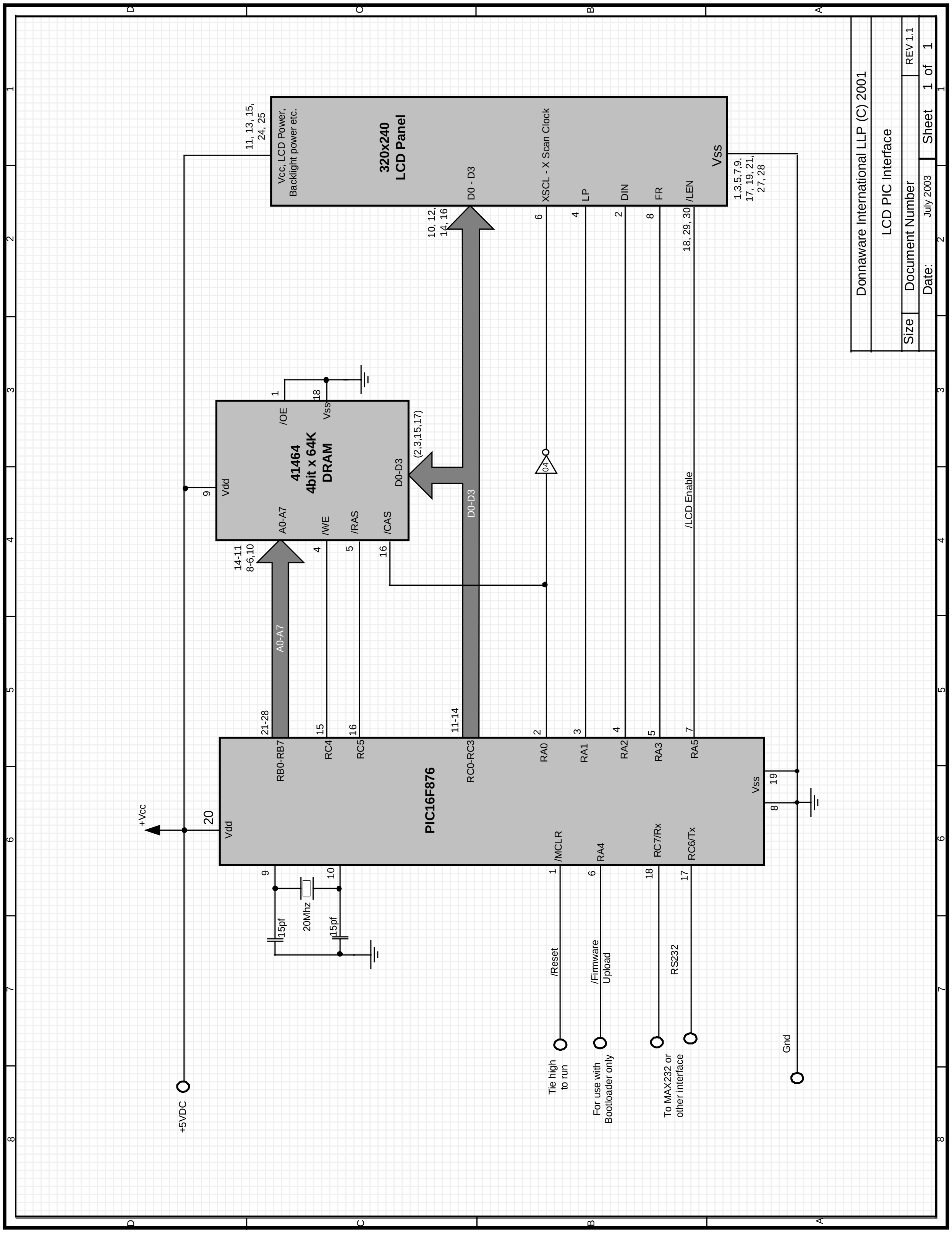
The attached code is set up for 250Kbaud rate which is good for interfacing to another MCU such as another PIC.

If you do want to hook up to you PC, here is an example circuit:

Option RS232 Cicuit



Appendix A: Circuit Diagram



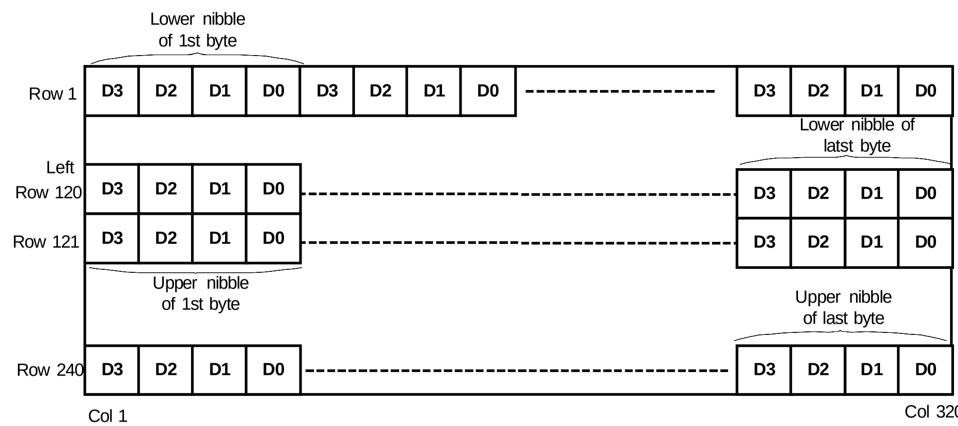
Appendix B: INMP001 Specs

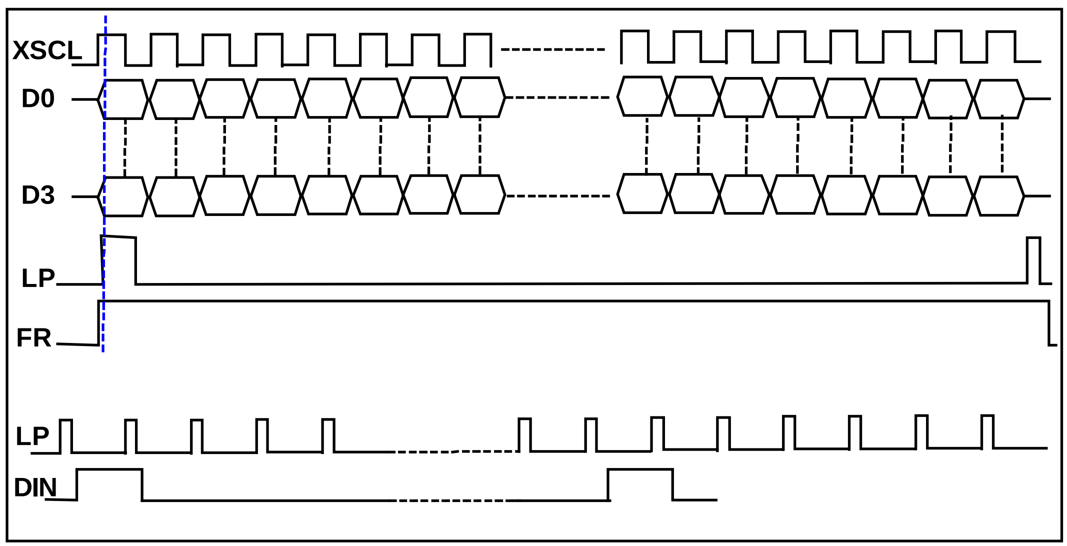
SPECIFICATIONS FOR LCD INMP0001 Panel

Resolution: 320x240 Pixels
Effective area: 9cm x 7cm"
Display Type: Positive Grey Transflexive
Backlight: Electroluminecent
(with built in HV supply)

Power Supply: Vdd = +5Vdc
LCD Power: +5 Vdc
(built in LCD bias generator)
Nominal Clock: 2.5 Mhz
Max Clock: 3.0 Mhz

Pin No.	Name	INMP0001 Panel
1	VSS	Gnd for Power Supply for LCD
2	DIN	Scan Start-up signal (Vert)
3	VSS	Gnd for Power Supply for LCD
4	LP	Latch Pulse for Horizontal
5	VSS	Gnd for Power Supply for LCD
6	XSCL	XSCL - X Scan Column Clock
7	VSS	Gnd for Power Supply for LCD
8	FR	Input AC Waveform for LCD
9	VSS	Gnd for Power Supply for LCD
10	D0	Display Data signal Bit 0
11	VDD	Power Supply for logic (+5v)
12	D1	Display Data signal Bit 1
13	VDD	Power Supply for logic (+5v)
14	D2	Display Data signal Bit 2
15	VDD	Power Supply for logic (+5v)
16	D3	Display Data signal Bit 3
17	VSS	Gnd for Power Supply for LCD
18	/LEN	LCD Enable
19	VSS	Gnd for Power Supply for LCD
20	CUP	Contrast Up
21	VSS	Gnd for Power Supply for LCD
22	CDN	Contrast Down
23	NC	No Connection
24	Vel	EL Backlight Power (+6 to12Vdc)
25	Vel	EL Backlight Power (+6 to12Vdc)
26	NC	No Connection
27	FGnd	Frame/Backlight Ground
28	FGnd	Frame/Backlight Ground
29	BLen	Backlight enable (active high)
30	BLbr	Backlight Brightness





Appendix C: Code Listing

```
/*--------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------*/
/*  INMP0001.H -- USB PIC LCD Controller                                    */
/*  DonnaWare International LLP Copyright (2001) All Rights Reserved        */
/*--------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------*/
#include <16F876.h>                     // Proceesor definitions
#device  *=16 adc=8                     // Use 16 bit pointers, Use 8 bit ADC values
#opt     5                              // Use medium optimization
#use     delay(clock=20000000)          // Set device crystal speed
//--------------------------------------------------------------------------
//  Set up microcontroller fuses
//  These are ignored when used with bootloader
//--------------------------------------------------------------------------
#fuses   NOWDT, HS, NOPUT, NOPROTECT, NOBROWNOUT,  NOCPD, NOWRT, NODEBUG
//--------------------------------------------------------------------------
//  Set up baud rate
//--------------------------------------------------------------------------
#use     rs232(baud=250000,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
//--------------------------------------------------------------------------
//  Use fast I/O for all Ports
//--------------------------------------------------------------------------
#use     fast_io(A)     // Use Fast I/O for Port A
#use     fast_io(B)     // Use Fast I/O for Port B
#use     fast_io(C)     // Use Fast I/O for Port C

/*--------------------------------------------------------------------------*/
/*  Include Libs                                                            */
/*--------------------------------------------------------------------------*/
#include <stdlib.h>

/*------------------------------------------------------------------------------*/
/*------------------------------------------------------------------------------*/
/*  Font Definitions                                                            */
/*------------------------------------------------------------------------------*/
/*------------------------------------------------------------------------------*/
#ORG    0x1A00, 0x1FFF {}       // Reserve for Chargen
#define FONTSTART 0x1A00        // Start of Font area
#define FONTEND   0x1EFF        // End of Font Area
#define FONTLEN   0x0480        // Font Table Legth for 12x12 font

/*--------------------------------------------------------------------------*/
/*  Command Strings:                                                        */
/*   Cmd   Bytes    Format              Description                         */
/*   ---   -----    ------------        ----------------------------        */
/*   'O'     1       'O'                Turn LCD on (Enable)                */
/*   'o'     1       'o'                Turn LCD off (Disable)              */
/*   'W'     4       'W' 0xNN 0xNN 0xdd Write Data Directly to DRAM         */
/*   'R'     3       'R' 0xNN 0xNN      Read Data Directly to DRAM          */
/*   'v'     1       'v'                Return firmware version             */
/*   'Z'     1       'Z'                Self Test Command                   */
/*   'C'     2       'C' 0xdd           Clear Screen                        */
/*   'B'     1 +...  'B' ........       Load BMP file                       */
/*   'p'     4       'p' x, y, c        Set 1 pixel                         */
/*--------------------------------------------------------------------------*/
#define CMD_ONN     'O'                 // Turn LCD On
#define CMD_OFF     'o'                 // Turn LCD Off
#define CMD_WRD     'W'                 // Write data
#define CMD_RDD     'R'                 // Read  data
#define CMD_VER     'v'                 // Get Version
#define CMD_TST     'Z'                 // Test Routine
/*------------------------------------------------------------------------------*/
/* Graphics commands:  All commands are valid ASCII characters with the Upper   */
/* or lower case used to denote the color                                       */
/*------------------------------------------------------------------------------*/
#define WHITE        0                  // Set this based on type of LCD Display
#define BLACK        1                  // Black=0 for FTN type display
#define CMD_CLR     'C'                 // Clear Screen
#define CMD_BMP     'B'                 // Load BMP Screen
#define CMD_PIX     'p'                 // Set Pixel
#define CMD_GET     'g'                 // Get Pixel
#define CMD_LIN     'l'                 // Set Line
#define CMD_BOX     'b'                 // Set Box
#define CMD_SFR     's'                 // Solid filled rectangle
#define CMD_CIR     'e'                 // Set Circle
#define CMD_FLF     'f'                 // Set Flood Fill
#define CMD_ULF     'U'                 // Upload Font
#define CMD_CHR     'a'                 // Print Character at specified location
```

```
#define CMD_TXT     't'                 // Print Text at specified location

/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
/*   Control pin definition:                                                      */
/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
#byte       Port_B      = 6         // Define Port B Variable location
#byte       Port_C      = 7         // Define Port C Variable location
#define     Data0       PIN_C0      // RC0 - Data line 0
#define     Data1       PIN_C1      // RC1 - Data line 1
#define     Data2       PIN_C2      // RC2 - Data line 2
#define     Data3       PIN_C3      // RC3 - Data line 3

#define     DRAMWE      PIN_C4      // RC4 - DRAM Write Enable
#define     DRAMRAS     PIN_C5      // RC5 - DRAM RAS
#define     DRAMCAS     PIN_A0      // RA0 - DRAM CAS & XSCLK

#define     LCD_XSCL    PIN_A0      // RA0 - DRAM CAS & XSCLK
#define     LCD_LP      PIN_A1      // RA1 - LCD Horizontal Latch Pulse
#define     LCD_DIN     PIN_A2      // RA2 - LCD Vertical Data In
#define     LCD_FR      PIN_A3      // RA3 - LCD Vertical Frame Sync

#define     FIRMWARE    PIN_A4      // RA4 - Firmware upload signal
#define     LCDEnable   PIN_A5      // RA5 - LCD Enable (Active high)

/*--------------------------------------------------------------------------------*/
/*  General Definintions and Macros                                               */
/*--------------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------------*/
/*  Global Variables                                                              */
/*--------------------------------------------------------------------------------*/


/*--------------------------------------------------------------------------------*/
/*  Function Prototypes                                                           */
/*--------------------------------------------------------------------------------*/
void ShowBanner(void);

/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
/*  Processor Register Definitions                                                */
/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
#define STATUS      0x03            // Status Register
#define ZEROBIT     0x02            // Zero Flag
#define PORTA       0x05            // Port A
#define PORTB       0x06            // Port B
#define PORTC       0x07            // Port C
#define RCREG       0x1A            // Receive Register
#define TRISA       0x85            // Port A
#define TRISB       0x86            // Port B
#define TRISC       0x87            // Port C
#define PIR1        0x0C            // Peripheral Interupt Register 1
#define RP0         5               // Register Pointer
#define RP1         6               // Register Pointer
#define RCIF        5               // Receive interupt flag

/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
/*  L C D    C O N T R O L    R O U T I N E S:                                    */
/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
/*  LCD Display sizes                                                             */
/*--------------------------------------------------------------------------------*/
#define     SCANW       0x50                // Screen Width in Nibbles (320/4)
#define     SCANH       0xF0                // Screen Height (240)
#define     DATAW       0x50                // Screen Width in Nibbles (320/4)
#define     DATAH       0xF0                // Screen Height (240)
#define     WIDTH       320                 // Display Width
#define     HEIGHT      240                 // Display Height

/*--------------------------------------------------------------------------------*/
/*  Definitions for Assembler section                                             */
/*--------------------------------------------------------------------------------*/
#define     DWE             PORTC,4         // RC4 - DRAM Write Enable
#define     RAS             PORTC,5         // RC5 - DRAM RAS line
#define     CAS             PORTA,0         // RA0 - DRAM CAS line

#define     XSCL            PORTA,0         // RA0 - DRAM CAS line
```

```
#define     XLP             PORTA,1         // RA1 - Horizontal Sync Line
#define     DIN             PORTA,2         // RA2 - Frame Sync Line
#define     FRM             PORTA,3         // RA3 - Veritcal Sync Line
#define     FrameBit        0x08            // Bit 4

#define     DRAMAddr        PORTB           // Port B - Address lines
#define     DRAMData        PORTC           // Port C - Data Lines
#define     DataDir         TRISC           // Tris C - Data Direction Register

/*-------------------------------------------------------------------------------*/
/*  LCD Frame Loop:                                                              */
/*  Loop to make 1 Frame by looping through the DRAM addresses and output to LCD*/
/*  Vertical Scan Lines: 160 Lower half then another 160 for upper              */
/*                                                                               */
/* Frame ___|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX|__________          */
/*              _____                                _____                      */
/* DIN     ____|     |______....240 lines..._______|     |____________          */
/*               _    _    _        _    _     _                                 */
/* XLP     ______|¯|___|¯|___|¯|__...___|¯|___|¯|____|¯|______________          */
/*                                                                               */
/* Row output by rolling through all the column addresses using DRAM Page       */
/* mode. Clock data into the LCD panel at the same time                         */
/* Horizonal Timing, Horizonal Dots 320, loaded 1 nibble at a time.             */
/*                __                                 __                          */
/* XLP     _______|¯¯|_______...320 Pixels...__________|¯¯|____________         */
/* CAS/    _   _   _   _   _   __....__   _   _   _   _   _   _   _             */
/* XSCL    ¯|_|¯|_|¯|_|¯|_|¯|_|¯......¯|_|¯|_|¯|_|¯|_|¯|_|¯|_|¯|_|¯|_          */
/*        _ ____ ____ ____ ____ ____ ____ ____ ____ ____ ____ ____ ____         */
/* Data   _|____|____|____|____|____|____|____|____|____|____|____|____         */
/*                                                                               */
/*-------------------------------------------------------------------------------*/
void Do1Frame(void)
{
            int  r_count;                   // Row count byte
#asm                                        // This section in assembler for speed
            //-------------------------------------------------------------
            //-------------------------------------------------------------
            //  Make 1 Frame:
            //-------------------------------------------------------------
            //-------------------------------------------------------------
            Goto        DoFrame             // Go do the frame
            //-------------------------------------------------------------
            //  Make 1 horizontal Line:
            //-------------------------------------------------------------
HorzLine:   Movlw       SCANW               // Get LCD Width (decimal 80, 1/4 of 320)
            Movwf       DRAMAddr            // Set DRAM Address lines
CasLoop:    Bcf         CAS                 // CAS Low to clock column
            Bsf         CAS                 // CAS High
            Decfsz      DRAMAddr,F          // Decrement x counter
            Goto        CasLoop             // If not then continune on
            return
            //-------------------------------------------------------------
            //-------------------------------------------------------------
            //  Start of Do Frame Routine:
            //-------------------------------------------------------------
            //-------------------------------------------------------------
DoFrame:    Movlw       0x08                // Frame bit
            Xorwf       PORTA,F             // Toggle Frame Bit
            //-------------------------------------------------------------
            // First line is special, clicks in the fram start bit
            //-------------------------------------------------------------
            Movlw       SCANH               // Get LCD Data Height
            Movwf       r_count             // Get the current row address
            Movwf       DRAMAddr            // Output it to the DRAM Address lines
            Bcf         RAS                 // Load RAS to DRAM, leave down for page mode
            Call        HorzLine            // Do 1 horz line
            Bsf         RAS                 // Done with Load RAS to DRAM
            Decf        r_count,F           // Decrement row counter
            Bsf         XLP                 // Horizontal Sync Pulse
            Bsf         DIN                 // Vertical sync low to signal start of frame
            Bcf         XLP                 // Horizontal Sync Pulse
            Bcf         DIN                 // Vertical sync low to signal start of frame
            //-------------------------------------------------------------
            // Frame Loop
            //-------------------------------------------------------------
FrameLoop:  Movf        r_count,W           // Get the current row address
            Movwf       DRAMAddr            // Output it to the DRAM Address lines
            Bcf         RAS                 // Load RAS to DRAM, leave down for page mode
            Call        HorzLine            // Do 1 horz line
            Bsf         RAS                 // Done with Load RAS to DRAM
            Bsf         XLP                 // Horizontal Sync Pulse
```

```
            Bcf         XLP                 // Horizontal Sync Pulse
            Decfsz      r_count,F           // Decrement row counter
            Goto        FrameLoop           // If not done, then continune on
            //------------------------------------------------------------------
            // Exit Routine
            //------------------------------------------------------------------
#endasm
}

/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
/*  D R A M    C O N T R O L    R O U T I N E S:                                   */
/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------*/
/*  Get char with no echo Function (keep frame alive while waiting             */
/*------------------------------------------------------------------------------*/
int getchd(void)
{
    int ch;

    while(!kbhit()) Do1Frame();             // Do the frame routine until char received
    ch = getch();                           // Get the character
    return(ch);                             // Return the value
}

/*------------------------------------------------------------------------------*/
/*  Get char with echo Function (keep frame alive while waiting                */
/*------------------------------------------------------------------------------*/
int getche(void)
{
    int ch;

    while(!kbhit()) Do1Frame();             // Do the frame routine until char received
    ch = getch();                           // Get the character
    putchar(ch);                            // Echo it back
    return(ch);                             // Return the value
}

/*--------------------------------------------------------------------------------*/
/*  Clear Screen Routine:                                                          */
/*  Interates through the DRAM and loads "data" value into all locations.         */
/*--------------------------------------------------------------------------------*/
void ClearScreen(int data)
{
    int value, row, col;

    set_tris_c(0b10000000);             // Port C  output
    Output_Low(DRAMWE);                 // DRAM Write Disabled

    //----------------------------------------------------
    // First do upper & lower half of screen at same time
    //----------------------------------------------------
    for(row = DATAH; row > 0 ; row--) {
        Output_B(row);                          // Put row Address onto port B
        Output_Low(DRAMRAS);                    // DRAM Row Address Strobe
        for(col = DATAW; col >0 ; col--) {
            value = Input_C() & 0xF0;           // Get Port C Value, mask off lower
            Output_B(col);                      // Put Column address on port B
            Output_C((swap(data)&0x0F) | value); // Put data on port C
            Output_Low(DRAMCAS);                // DRAM Column Address Strobe Disabled
            Output_High(DRAMCAS);               // DRAM Column Address Strobe Disabled

            Output_B(col--);                    // Put Column address on port B
            Output_C((swap(data)&0x0F) | value); // Put lower data on port C
            Output_Low(DRAMCAS);                // DRAM Column Address Strobe Disabled
            Output_High(DRAMCAS);               // DRAM Column Address Strobe Disabled

        }
        Output_High(DRAMRAS);                   // DRAM Row Address Strobe Disabled
    }
    Output_High(DRAMWE);                // DRAM Write Disabled
    set_tris_c(0b10001111);             // Port C input (hi-z)
}

/*--------------------------------------------------------------------------------*/
/*  Load BMP Routine:    Interates through the DRAM and loads data into all       */
/*  locations from input comming from the RS232 line.  This loads the screen.    */
/*--------------------------------------------------------------------------------*/
void LoadBMP(void)
```

```
{
    int value, data, row, col;

    set_tris_c(0b10000000);             // Port C  output
    Output_Low(DRAMWE);                 // DRAM Write Disabled
    for(row = DATAH; row >0; row--) {
        Output_B(row);                          // Put row Address onto port B
        Output_Low(DRAMRAS);                    // DRAM Row Address Strobe
        for(col = DATAW; col > 0; col--) {
            data = getch();                     // Get data From RS232 port
            Output_B(col);                      // Put Column address on port B
            value = Input_C() & 0xF0;           // Get Port C Value, mask off lower
            Output_C((data&0x0F) | value);      // Put upper data on port C
            Output_Low(DRAMCAS);                // DRAM Column Address Strobe Disabled
            Output_High(DRAMCAS);               // DRAM Column Address Strobe Disabled
/*
            Output_B(col++);                    // Put Column address on port B
            Output_C((swap(data)&0x0F) | value); // Put lower data on port C
            Output_Low(DRAMCAS);                // DRAM Column Address Strobe Disabled
            Output_High(DRAMCAS);               // DRAM Column Address Strobe Disabled
*/
        }
        Output_High(DRAMRAS);               // DRAM Row Address Strobe Disabled
    }
    Output_High(DRAMWE);                // DRAM Write Disabled
    set_tris_c(0b10001111);             // Port C input (hi-z)
}

/*------------------------------------------------------------------------------*/
/*   Self Test Routine                                                          */
/*------------------------------------------------------------------------------*/
void TestFunc(int data)
{

    Do1Frame();

}

/*------------------------------------------------------------------------------*/
/*------------------------------------------------------------------------------*/
/*  G R A P H I C S    F U N C T I O N S:                                       */
/*------------------------------------------------------------------------------*/
/*------------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------*/
/*   Read 1 Nibble from DRAM                                                        */
/*------------------------------------------------------------------------------*/
int ReadNibble(int row, int col)
{
    int nibble;

    Output_High(DRAMWE);                // DRAM Write Disabled
    set_tris_c(0b10001111);             // Port C input

    Output_B(row);                      // Put row Address onto port B
    Output_Low(DRAMRAS);                // DRAM Row Address Strobe

    Output_B(col);                      // Put Column address on port B
    Output_Low(DRAMCAS);                // DRAM Column Address Strobe Disabled
    nibble = Input_C() & 0x0F;          // Get data off port C & mask off upper bits
    Output_High(DRAMCAS);               // DRAM Column Address Strobe Disabled

    Output_High(DRAMRAS);               // DRAM Row Address Strobe Disabled
    return(nibble);                     // return answer
}
/*------------------------------------------------------------------------------*/
/*   Write 1 Nibble to DRAM                                                     */
/*------------------------------------------------------------------------------*/
void WriteNibble(int row, int col, int data)
{
    int value;

    Output_Low(DRAMWE);                 // DRAM Write Disabled
    set_tris_c(0b10000000);             // Port C  output

    Output_B(row);                      // Put row Address onto port B
    Output_Low(DRAMRAS);                // DRAM Row Address Strobe

    Output_B(col);                      // Put Column address on port B
    value = Input_C() & 0xF0;           // Get Port C Value, mask off lower
    Output_C((data&0x0F) | value);      // Put data on port C
```

```
    Output_Low(DRAMCAS);                // DRAM Column Address Strobe Disabled
    Output_High(DRAMCAS);               // DRAM Column Address Strobe Disabled

    Output_High(DRAMRAS);               // DRAM Row Address Strobe Disabled
    Output_High(DRAMWE);                // DRAM Write Disabled
    set_tris_c(0b10001111);             // Port C input (hi-z)
}

/*-------------------------------------------------------------------------------*/
/*  Set 1 Pixel:                                                                 */
/*  This routine sets  one pixel located at x,y (0-319,0-239) to the color of    */
/*  px, the color of the pixel, either black or white.  This of course depends  */
/*  on the type of display, for an FTN a 1 makes a white pixel, for STN oposite */
/*-------------------------------------------------------------------------------*/
void SetPixel(long x, long y, int1 px)
{
    int  ra, ca;
    int  data, p;

    if(x > (WIDTH-1))  x = 0;
    if(y > (HEIGHT-1)) y = 0;

    ca = x/4;
    ra = y;
    data = ReadNibble(ra, ca);

    p = 0x08>>(x%4);
    if(px) data |=  p;
    else   data &= ~p;

    WriteNibble(ra, ca, data);
}
/*-------------------------------------------------------------------------------*/
/*  Get 1 Pixel:                                                                 */
/*  This routine gets the color of one pixel located at x,y (0-319,0-239).      */
/*  The color of the pixel, either black or white is returned. This again       */
/*  depends on the type of display. This routine is used by the flood file      */
/*  routine                                                                      */
/*-------------------------------------------------------------------------------*/
int1 GetPixel(long x, long y)
{
    int  ra, ca, data;
    int1 p;

    if(x > (WIDTH-1))  x = 0;
    if(y > (HEIGHT-1)) y = 0;

    ca = x/4;
    ra = y;

    data = ReadNibble(ra, ca);
    p = (int1)(data>>(3-(x%4)));
    return(p);
}
/*-------------------------------------------------------------------------------*/
/*  End .h
/*-------------------------------------------------------------------------------*/
```