My_LCD Serial Backpack

Using the ATtiny2313 AVR Microcontroller
With the
HD44780 Text Based LCD Chip Set

By Carl W. Livingston
microcarl@roadrunner.com
Copyright 2007 by Carl W. Livingston
This article may be freely distributed in an unmodified form.
The software may be used for personal, noncommercial applications only.

Warning!

There are no reverse polarity protection diodes on the My_LCD Serial Backpack module. Connecting the power source in reverse will damage this module.

Introduction:

This document describes a serial interface for controlling a standard text based LCD which uses the HD44780 LCD controller chipset. The interface is controlled with an Atmel ATtiny2313 AVR microcontroller.

The My_LCD controller uses the ATtiny2313 as an RS232 based serial *Pass-thru* controller that provides power up initialization and establishes the necessary timing for the HD44780 text based LCD controller chipset. In addition, the ATtiny2313 firmware controls an optional LED backlight built into some LCDS to provide background illumination of the LCD information being displayed.

The Design Goals:

I had several goals for the My_LCD Serial Backpack:

1. Manufacturing cost of $15.00 or less.
2. Reliable operation.
3. Ease of use – Power, RS232 and GND connections only.
3. Jumper selectable 9600, 19.2K, 38.4K & 115.2K BAUD rates.
5. Use the ATtiny2313 microcontroller.
6. Single sided printed circuit board construction.
7. Direct access to all HD44780 chipset commands.

Additional Commands:

The following table summarizes three additional commands beyond those recognized by the HD44780 chipset. These commands are decoded and executed by the Ttiny2313 microcontroller.

My_LCD Serial Backpack Control Commands		
Command	Response	Function
LED_ON	Decimal 252 (0xFC)	Turn on LCD backlight
LED_OFF	Decimal 253 (0xFD)	Turn off LCD backlight
CMD_MODE	Decimal 254 (0xFE)	LCD command delimiter to the HD44780 controller

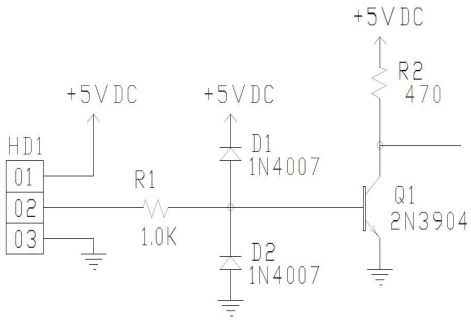
CMD_MODE (decimal 254) is interpreted by the My_LCD controller as an indication to the serial backpack that the LCD module is to enter into the command mode. CMD_MODE and its associated parameter will be passed directly to the LCD display. The LCD module executes the requested command following the receipt of the associated CMD_MODE parameter and terminates automatically, immediately after the LCD module executes the requested.

LED_ON (decimal 252) is interpreted by the My_LCD controller as a request to turn on the LED backlight.

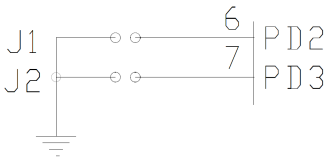
LED_OFF (decimal 253) is interpreted by the My_LCD controller as a request to turn off the LED backlight.

Technical Description:

Upon initial powering of the system (Host controller, LCD display and My_LCD serial backpack) there is a minimum 40 millisecond delay required for the LCD and the My_LCD serial backpack to properly initialize. If this 40 millisecond delay is not complied with, the LCD will have to be cleared by the host controller.



The schematic fragment on the left is the RS232 input interface. Transistor Q1 performs the logic inversion required to convert RS232 to the TTL signal requirements of the ATtiny2313. Diodes D1 & D2 protect Q1 from excessive +/- drive voltage used in the RS232 electrical specification. Resistor R1 limits the drive current of transistor Q1 and also protects D1 & D2 from over current.



To the right are the BAUD rate selection jumpers. J1 & J2 are installed with *Zero Ohm* 1206 SMT shorting resistors. The ATtiny2313 internal *Pull-Up* resistors associated with PD2 & PD3 are turned on. The My_LCD module comes with J1 & J2 *Zero Ohm* resistors installed. To select a different BAUD rate; other than the default 115.2K BAUD, replace the shorting resistor(s) with 0.1uF ceramic capacitors in the 1206 SMT package. This will enhance noise immunity of these inputs. For BAUD rate jumper selection, see the table below.

BAUD Rate Jumper Setting Schedule		
J1	J2	BAUD Rate
SHORT	SHORT	115.2K BAUD, N,8,1 (default)
OPEN	SHORT	38.4K BAUD, N,8,1
SHORT	OPEN	19.2K BAUD, N,8,1
OPEN	OPEN	9600 BAUD, N,8,1

Note: <u>To change the BAUD rate of the My_LCD serial backpack module, remove power from the serial backpack module, change the appropriate J1 & J2 BAUD rate selection jumpers, then apply power to the serial backpack module</u>. The BAUD rate setting of J1 & J2 are determined during the ATtiny2313 initialization only! Once power is applied to the serial backpack module and initialization is complete the BAUD rate is fixed for that particular power cycle. The BAUD rate can not be altered by software commands. When changing the BAUD rate to another Baud Rate, replace the *Zero Ohm* 1206 SMT resistor with a

0.1uF 1206 SMT ceramic capacitor to enhance noise immunity on the BAUD rate selection inputs, J1 & J2.

For LCDs with a built in LED backlight, the My_LCD serial backpack is fitted with a 33 Ohm current limiting resistor, R5. Most of the text based LCD display modules that have LED backlights that I have used have operated in the range of 40mA to 80mA, though I have heard that there are some LED backlights that operate on substantially higher forward operating current.

While the My_LCD serial backpack is compatible with the HD44780 chipset and many of the available clones, the interface will not work with the electroluminescent type backlighting. In the event of no backlight or the presences of an electroluminescent backlight, remove header pins 15 and 16 from the My_LCD serial backpack. You will need to control the electroluminescent backlight power supply according to its particular manufacturer's requirements.

The equation for determining the proper value of backlight LED current limiting resistor is:

$$R_{led} = \frac{Vcc - V_{led}}{I_{led}}$$

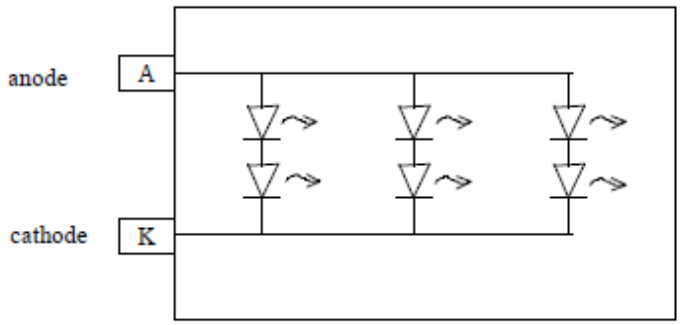
The backlight LEDs used in the LCD display modules that I have been using operate on approximately 60mA with a forward voltage drop of about 3 volts.

$$R_{led} = \frac{5v - 3v}{0.06} = 33.3 \text{ Ohms}$$

The value of the current limiting resistor, R5, shown on the My_LCD module schematic is 33 Ohms. The value you may require will depend on the particular LED backlight that is supplied with the LCD module that you will be using.

I have included an excerpt from the Optrex datasheet regarding treatment of the LED backlight.

In general, LED Backlit modules include an array of LED's positioned behind the LCD panel.

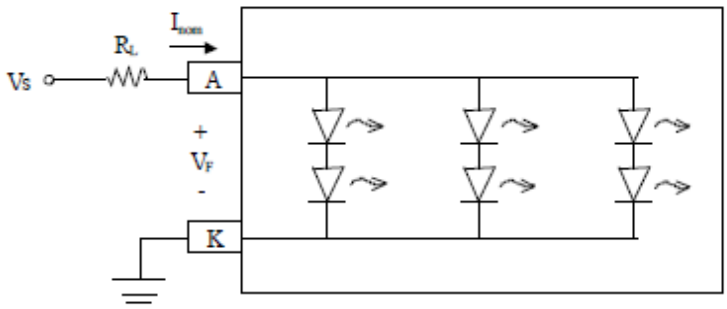


Optrex specifies the absolute maximum current for the LED array in the detailed module specification.

Also specified is the LED Forward Voltage at optimum current level.

Example: V_F @ I_{nom} [mA]

A current limiting resistor should be added in series to limit the current to the LED assembly. The resistor can be calculated as follows:

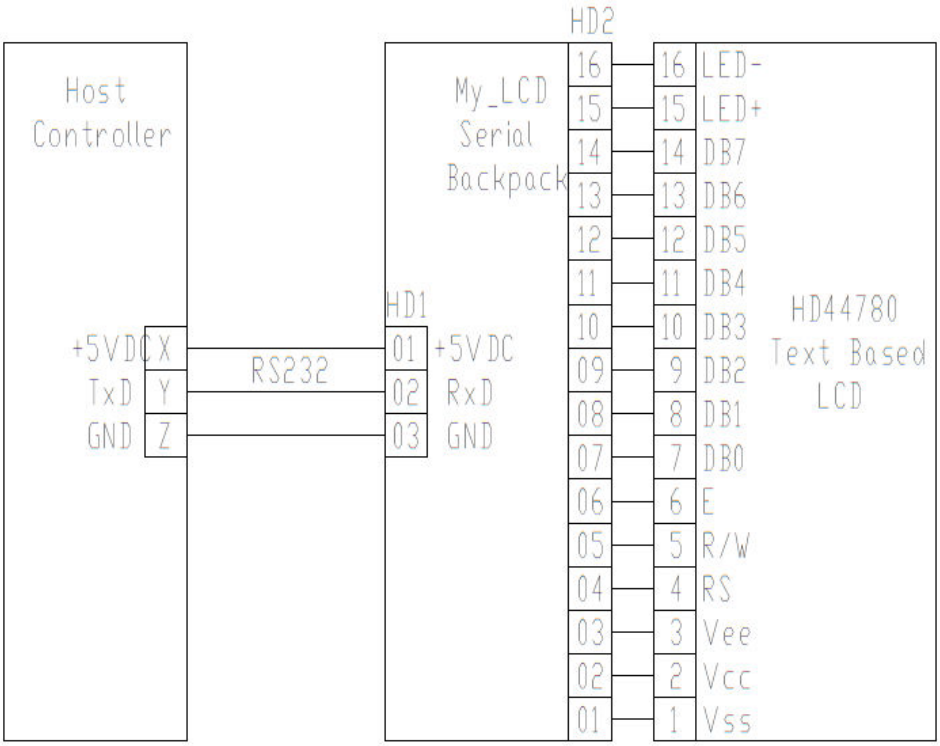


$$R_L = \frac{V_S - V_F}{I_{nom}} \quad \text{[ohm]}$$

Note: I_{nom} can be found in the V_F rating of the LCD module specification.

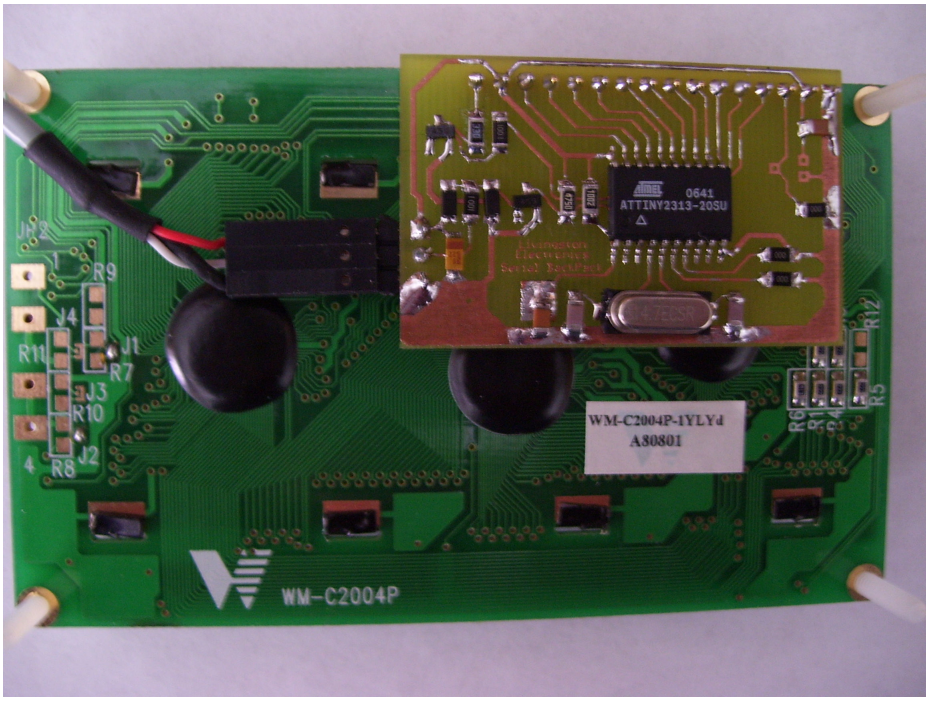
Electrical Connections:

Interfacing the My_LCD serial backpack is straight forward. Connections are made from +5 VDC, RS232 TxD and GND (5 volt common) on the host system, to HD1-01 (+5 VDC), HD1-02 (RS232 RxD) and HD1-03 (GND) connections on the My_LCD serial backpack module. The interconnect diagram is shown below.



Warning repeated!

There are no reverse polarity protection diodes on the My_LCD Serial Backpack module. Connecting the power source in reverse will damage this module.



Note the connector wiring in the picture above. The Red wire is +5 VDC, the White wire is RxD and the Black wire is GND (5 VDC common).

HD44780 LCD Chipset Instruction Reference:

The HD44780 instruction set follows as a quick reference.

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

AVR_DEV.c

```
#include <iom8535v.h>
#include <stdio.h>

#include "putchar.c"
#include "getchar.c"

#define UART_TRANSMIT_ON()    UCSRB |= 0x8
#define UART_TRANSMIT_OFF()   UCSRB &= ~0x8
#define UART_RECEIVE_ON()     UCSRB |= 0x10
#define UART_RECEIVE_OFF()    UCSRB &= ~0x10

// Serial Backpack command delimiter
#define CMD_MODE 0xFE

// Display CLEAR text command
#define DSP_CLR_CMD 0x01
// Display HOME cursor command
#define DSP_HOME_CMD 0x02

// Display ENTRY MODE command & parameters
#define DSP_ENTRY_CMD 0x04
#define CUR_DIR 0x02
#define CUR_MOVE 0x01

// Display CONTROL MODE & parameters
#define DSP_CON_CMD 0x08
#define DSP_ON 0x04
#define CUR_ON 0x02
#define CUR_BLINK 0x01

// Display CURSOR CONTROL & parameters
#define DSP_CUR_CMD 0x10
#define DSP_SHIFT 0x08
#define DISPLAY_L_R 0x04

// Display LINE POSITION addressing
#define LINE1 0x80
#define LINE2 0xC0
#define LINE3 0x94
#define LINE4 0xD4

// Display DATA & CHARACTER GENERATOR select bits
#define DD_RAM 0x80
#define CG_RAM 0x40

// Custom Character DDRAM location addressing
#define DDRAM_START_ADD_0 0x00
#define DDRAM_START_ADD_1 0x01
#define DDRAM_START_ADD_2 0x02
#define DDRAM_START_ADD_3 0x03
#define DDRAM_START_ADD_4 0x04
#define DDRAM_START_ADD_5 0x05
#define DDRAM_START_ADD_6 0x06
#define DDRAM_START_ADD_7 0x07
```

```
// Custom Character CGRAM Starting addresses
#define CGRAM_START_ADD_0 0x40
#define CGRAM_START_ADD_1 0x48
#define CGRAM_START_ADD_2 0x50
#define CGRAM_START_ADD_3 0x58
#define CGRAM_START_ADD_4 0x60
#define CGRAM_START_ADD_5 0x68
#define CGRAM_START_ADD_6 0x70
#define CGRAM_START_ADD_7 0x78

// LED enable switch input
#define LED 4
// ASCII control code (0xFC) turns on LED
#define LED_ON 252
// ASCII control code (0xFD) turns on LED
#define LED_OFF 253

#define BannerMessage1 "My_LCD"
#define BannerMessage2 "Serial  Backpack"
#define BannerMessage3 "By"
#define BannerMessage4 "Carl W. Livingston"

void MakeCustomChar (const char*, char);
void delay(unsigned int);
void clear_disp(void);
void display_banner(void);
void timer1(void);
void uart0_init(void);
void init_devices(void);

// Smiley face
const char Smiley_face[] = {// Smiley face
                                        0b00001110,
                                        0b00011111,
                                        0b00000000,
                                        0b00001010,
                                        0b00000100,
                                        0b00010001,
                                        0b00001110,
                                          0b00000000};
const char Thin_cross[] = {// Thin cross
                                          0b00010001,
                                    0b00000000,
                                    0b00011111,
                                    0b00010101,
                                    0b00011011,
                                    0b00001110,
                                    0b00010001,
                                    0b00000000};
const char Thick_cross[] = {// Thick cross
                                          0b00000000,
                                    0b00001110,
                                    0b00011011,
                                    0b00010001,
                                    0b00011011,
                                    0b00001110,
```

```
                                    0b00000000,
                                    0b00000000};
const char Music_note[] = {// Music note
                                          0b00000011,
                                    0b00000010,
                                    0b00000111,
                                    0b00000010,
                                    0b00011110,
                                    0b00010010,
                                    0b00011110,
                                    0b00000000};
const char Loop_Antenna[] = {// Loop Antenna
                                          0b00010001,
                                    0b00010001,
                                    0b00011111,
                                    0b00010101,
                                    0b00010101,
                                          0b00000100,
                                    0b00001110,
                                    0b00000000};
const char Lh_Yagi_Antenna[] = {// Lh Yagi Antenna
                                          0b00010000,
                                    0b00010100,
                                    0b00010101,
                                    0b00011111,
                                    0b00010101,
                                    0b00010100,
                                    0b00010000,
                                    0b00000000};
const char Diamond[] = {// Diamond
                                          0b00000000,
                                    0b00011111,
                                    0b00010001,
                                    0b00010101,
                                    0b00010001,
                                    0b00011111,
                                    0b00000000,
                                    0b00000000};
const char Rh_Yagi_Antenna[] = {// Rh Yagi Antenna
                                          0b00000001,
                                    0b00000101,
                                    0b00010101,
                                    0b00011111,
                                    0b00010101,
                                    0b00000101,
                                    0b00000001,
                                    0b00000000};

int   DelayCounter;
unsigned long int DisplayCounter = 0;

void main()
{
      init_devices(); // Initialize the required Mega8535 devices

      delay(40); // Wait for LCD to boot up
```

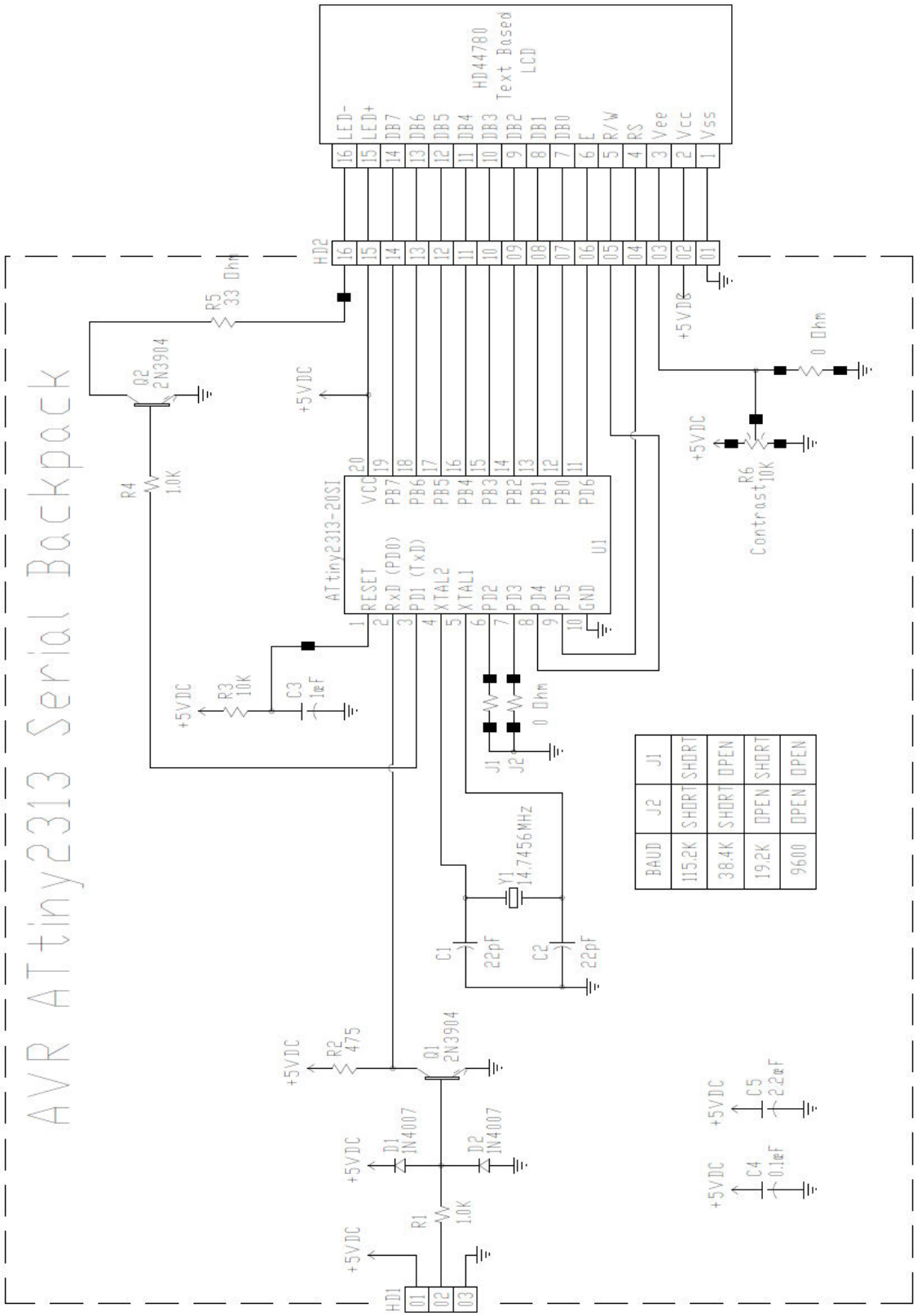
```
      UART_TRANSMIT_ON(); // Turn on the USART transmitter

    MakeCustomChar (Smiley_face, CGRAM_START_ADD_0);
      MakeCustomChar (Lh_Yagi_Antenna, CGRAM_START_ADD_1);
      MakeCustomChar (Diamond, CGRAM_START_ADD_2);
      MakeCustomChar (Rh_Yagi_Antenna, CGRAM_START_ADD_3);
    MakeCustomChar (Thin_cross, CGRAM_START_ADD_4);
      MakeCustomChar (Thick_cross, CGRAM_START_ADD_5);
      MakeCustomChar (Music_note, CGRAM_START_ADD_6);
      MakeCustomChar (Loop_Antenna, CGRAM_START_ADD_7);

      if (PINB & (1<<LED)) // If LED enable switch is active,
            putchar(LED_ON); // Turn on the LED

      // Put the banner message on the display
      clear_disp(); // Clear the display for the banner message
      display_banner(); // Display the banner message
      delay(1000); // Display banner message for 1 second
      clear_disp(); // Clear the display for incoming data

      while(1)
      {
       PORTB ^= (1<<PORTB0); // Sync Pulse for testing

       putchar(CMD_MODE); // Put the LCD in command mode
       putchar(LINE2);  // Set the LCD cursor position
       printf("%8lu", DisplayCounter);

       // Write the custom characters to the display
       putchar(CMD_MODE); // Put the LCD in command mode
       putchar(LINE1);  // Set the LCD cursor position
       putchar(DDRAM_START_ADD_0); // Display custom character 0
       putchar(DDRAM_START_ADD_1); // Display custom character 1
       putchar(DDRAM_START_ADD_2); // Display custom character 2
       putchar(DDRAM_START_ADD_3); // Display custom character 3
       putchar(DDRAM_START_ADD_4); // Display custom character 4
       putchar(DDRAM_START_ADD_5); // Display custom character 5
       putchar(DDRAM_START_ADD_6); // Display custom character 6
       putchar(DDRAM_START_ADD_7); // Display custom character 7

       if (PINB & (1<<LED)) // If LED enable switch is active,
            putchar(LED_ON); // Turn on the LED
       else putchar(LED_OFF); // Else, turn off the LED
      }
}

void MakeCustomChar (const char *Symbol, char Address) {
    // Set up the custom characters
      char n = NULL;
      putchar(CMD_MODE); // Put the LCD in command mode
      putchar(Address); // Set the display to CGRAM mode, address 0
      // Make the custom characters
      while (n < 8) {
              putchar(Symbol[n++]); // Send current character to the
display
      }
}
```

```
// Integer delay routine
// 65.535 seconds maximum
// All things wait on this delay function to term
void delay(unsigned int t)
{
       DelayCounter = t;
       while(DelayCounter != 0);
}

// Clear the display
void clear_disp(void)
{
      putchar(CMD_MODE); // Put the LCD in command mode
      putchar(DSP_CLR_CMD); // Send the clear the LCD display
      delay(30);  // Wait 30 miliseconds
}

// Print the banner message
void display_banner(void)
{
      putchar(CMD_MODE); // Put the LCD in command mode
      putchar(LINE1+5); // Set the LCD cursor position
      printf("%s", BannerMessage1); // Display line 1 of the banner
message

      putchar(CMD_MODE); // Put the LCD in command mode
      putchar(LINE2+0); // Set the LCD cursor position

      printf("%s", BannerMessage2); // Display line 2 of the banner
message

      putchar(CMD_MODE); // Put the LCD in command mode
      putchar(LINE3+9); // Set the LCD cursor position
      printf("%s", BannerMessage3); // Display line 3 of the banner
message

      putchar(CMD_MODE); // Put the LCD in command mode
      putchar(LINE4+1); // Set the LCD cursor position
      printf("%s", BannerMessage4); // Display line 4 of the banner
message
}

// Timer 1 Intrrupt Service Routine
// Call one time per milisecond
#pragma interrupt_handler timer1:7
 void timer1(void)
  {
//      PORTB ^= (1<<PORTB0);
        DelayCounter -= 1; // Decrement the delay counter
        DisplayCounter++;  // Increment the LCD display test counter
  }

void port_init(void)
{
 PORTA = NULL;
 DDRA  = NULL;
```

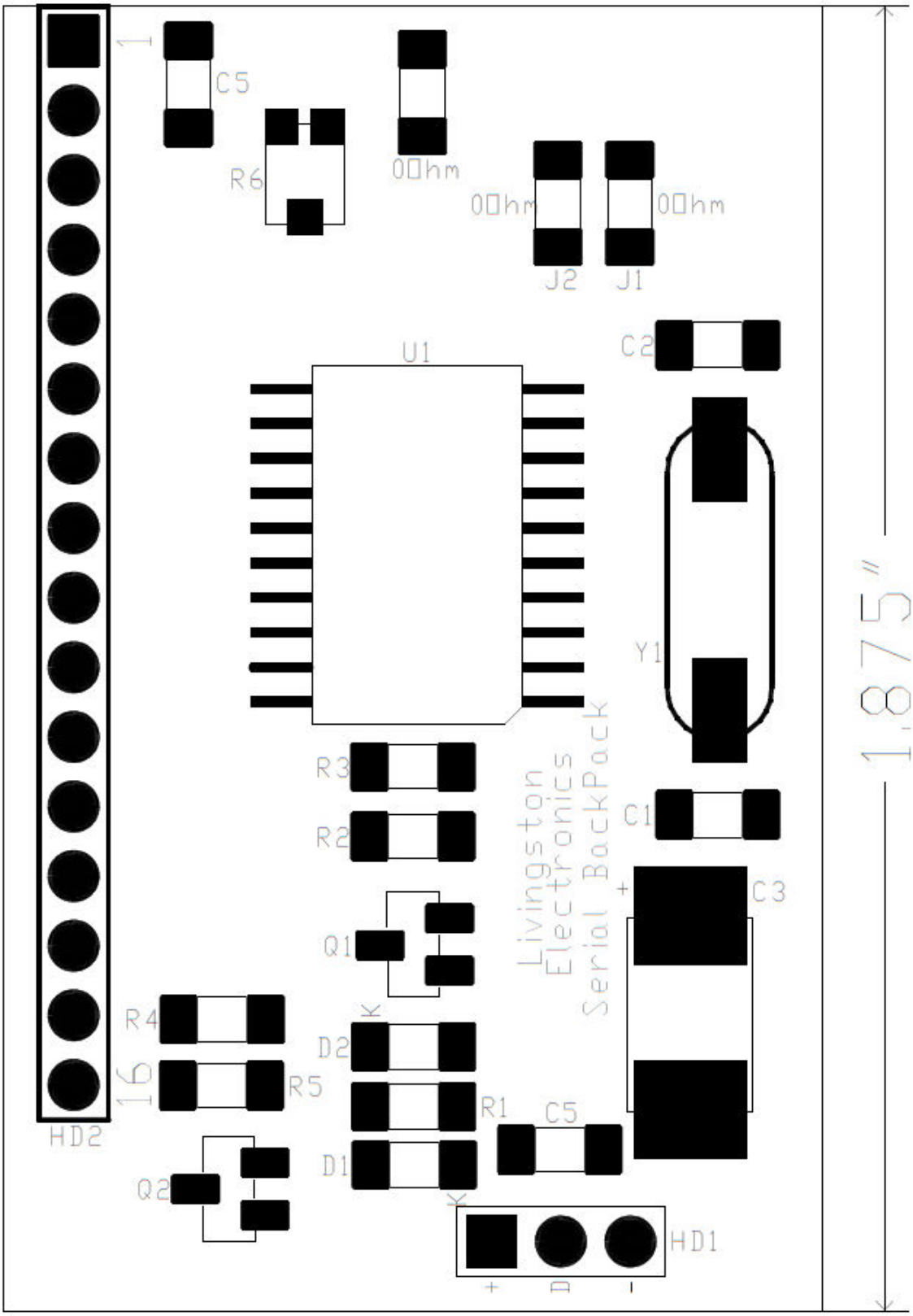
```
 PORTB = 0xFF;
 DDRB  = 0x01;
 PORTC = NULL;
 DDRC  = 0xFF;
 PORTD = NULL;
 DDRD  = 0xFF;
}

// TIMER1 initialize - prescale:64
// WGM: 0) Normal, TOP=0xFFFF
// Desired value: 1KHz
// Actual value:  1.000KHz (0.0%)
void timer1_init(void)
{
 TCCR1B = NULL;           // Stop Timer Counter 1
 TCNT1 = 0;               // Clear Timer Counter 1
 OCR1AH = NULL;           // We want a 1.000 Milisecond delay
resolution
 OCR1AL = 0x72;
 TCCR1A = NULL;           // Generate no outputs
 TCCR1B = 0x0B;           // Start Timer Counter 1 with OCR1A, CLK / 64
}

// BAUD rate definitions
// Actual baud rate = 9600 BAUD, (0.0% error) @ 7.3728MHz
// Actual baud rate = 19.2K BAUD, (0.0% error) @ 7.3728MHz
// Actual baud rate = 38.4K BAUD, (0.0% error) @ 7.3728MHz
// Actual baud rate = 115.2K BAUD, (0.0% error) @ 7.3728MHz
#define BAUD_4800 95
#define BAUD_9600 47
#define BAUD_14400 31
#define BAUD_19200 23
#define BAUD_28800 15
#define BAUD_38400 11
#define BAUD_57600 7
#define BAUD_76800 5
#define BAUD_115200 3

// Initialize UART0
void uart0_init(void)
{
 UCSRB = NULL; // Disable while setting baud rate
 UCSRA = NULL;
 UCSRC = (1<<URSEL) | 0x06;
 UBRRL = BAUD_115200; // Set baud rate lo
 UBRRH = NULL; // Set baud rate hi
 UCSRB = 0x18;
}

// Initialize required controller peripherals
void init_devices(void)
{
 asm("cli"); // Disable global interrupts
 port_init(); // Call the I/O port initialization
 timer1_init(); // Call the timer 1 initialization
 uart0_init(); // Call the IUSART initialization
```

```
 MCUCR = NULL;
 GICR  = NULL;
 TIMSK = 0x10; // Enable Timer 1 interrupt
 asm("sei"); // Re-enable global interrupts
 // Controller peripherals are now initialized
}
```

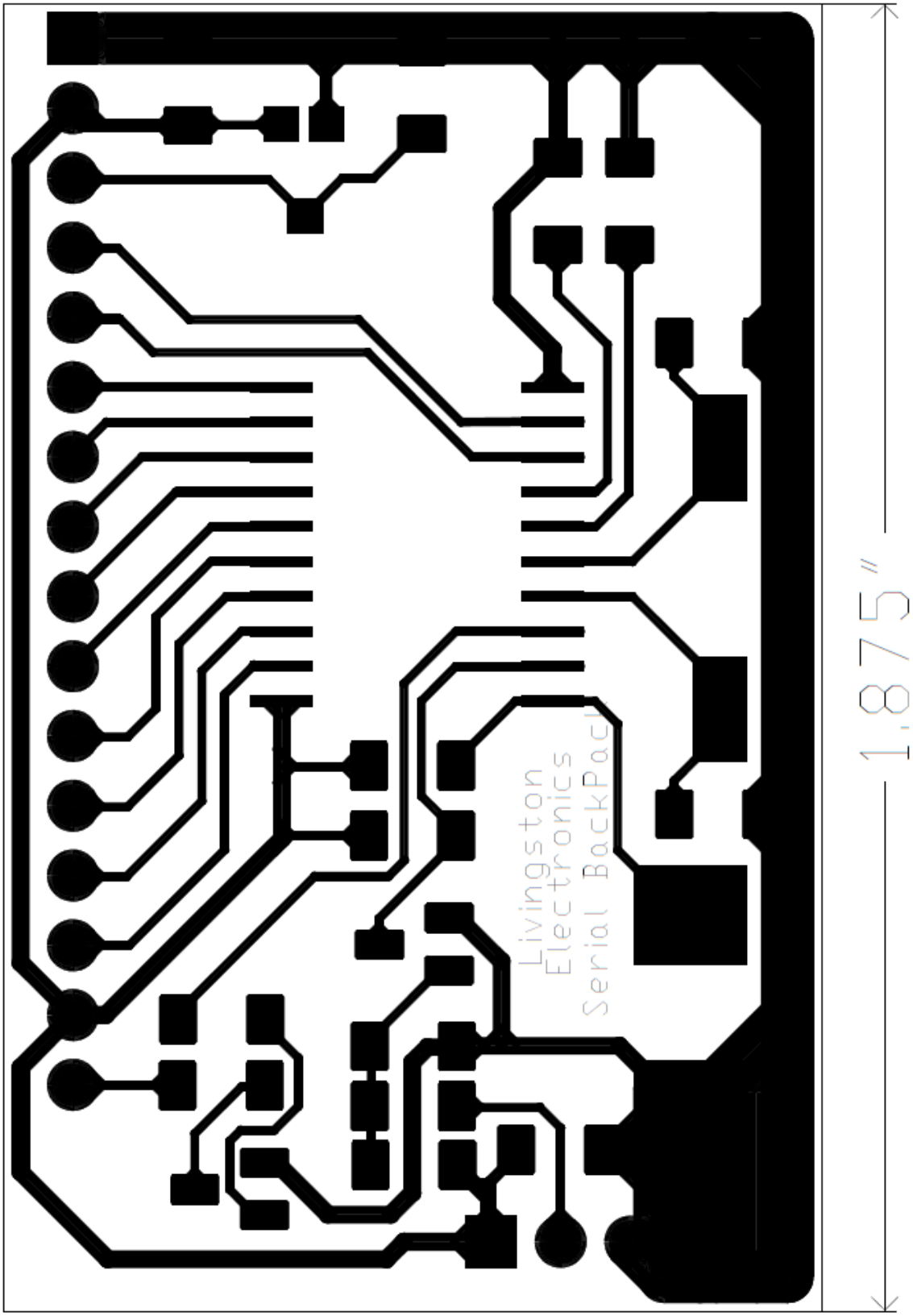
The Full My_LCD Serial Backpack Module Schematic:



Component Layout:



Foil Artwork:



Parts List:

Designation	Value	Part Number	Cost Ea.	Description
All components are Digi-Key Part Numbers				
R1	1.00K Ohm	311-1.00FCT-ND	$0.088	1.0K Ohm, 1/4W, 1% Resistor, 1206 SMD
R2	475 Ohm	311-475FCT-ND	$0.088	475 Ohm, 1/4W, 1% Resistor, 1206 SMD
R3	10.00K Ohm	311-10.0KFCT-ND	$0.088	10.0K Ohm, 1/4W, 1% Resistor, 1206 SMD
R4	1.00K Ohm	311-1.00FCT-ND	$0.088	1.0K Ohm, 1/4W, 1% Resistor, 1206 SMD
R5	33.2 Ohm	311-33.2FCT-ND	$0.088	33 Ohm, 1/4W, 1% Resistor, 1206 SMD
R6	10K Ohm	TC73X-103ECT-ND	$0.20	10K Ohm Single Turn Potentiometer, 3mm SMD
Zero Ohm	Zero Ohm	311-0.0ERCT-ND	$0.078	Zero Ohm Resistor, 1/4W 1%, 1206 SMD
Zero Ohm	Zero Ohm	311-0.0ERCT-ND	$0.078	Zero Ohm Resistor, 1/4W 1%, 1206 SMD
Zero Ohm	Zero Ohm	311-0.0ERCT-ND	$0.078	Zero Ohm Resistor, 1/4W 1%, 1206 SMD
Zero Ohm	Zero Ohm	311-0.0ERCT-ND	$0.078	Zero Ohm Resistor, 1/4W 1%, 1206 SMD
C1	22pF	399-1196-1-ND	$0.499	22pF Ceramic Capacitor, 1206 SMD
C2	22pF	399-1196-1-ND	$0.499	22pF Ceramic Capacitor, 1206 SMD
C3	0.1uF	399-1250-1-ND	$0.253	0.1uF Ceramic Capacitor, 1206 SMD
C4	0.1uF	399-1250-1-ND	$0.253	0.1uF Ceramic Capacitor, 1206 SMD
C5	2.2uF	511-1469-1-ND	$0.33	2.2uF Tantalum Capacitor, 16V 20% 1206 SMD
CX	0.1uF	399-1250-1-ND	$0.253	0.1uF Ceramic Capacitor, 1206 SMD
CY	0.1uF	399-1250-1-ND	$0.253	0.1uF Ceramic Capacitor, 1206 SMD
D1	BAV3004W-7	BAV3004WDICT	$0.45	Diode Switch, 350V, 400MW, SOD-123
D2	BAV3004W-7	BAV3004WDICT	$0.45	Diode Switch, 350V, 400MW, SOD-123
Q1	FMMT459TA	FMMT459TACT-ND	$0.71	Transistor Switch, NPN, 450V, 150ma, SOT-23
Q2	FMMT459TA	FMMT459TACT-ND	$0.71	Transistor Switch, NPN, 450V, 150ma, SOT-23
Y1	14.7456MHz	XC692CT-ND	$0.73	14.7456MHz Quartz Crystal, SMT
U1	ATtiny2313-20SU	ATTINY2313-20SU-ND	$2.25	Atmel ATtiny2313 20MHz Microcontroller, SMD
HD1	3 PIN Header	929835-01-36-ND	$1.82	Connector Header, 100 Single R/A 36Pos
HD2	14 PIN Header	929834-02-36-ND	$1.80	Connector Header, 100 Single STR 36Pos
PCB1	PCB	My_LCD	$5.00	Single Sided Printed Circuit Board
		TOTAL	$17.172	

Note: *Capacitors CX & CY are spare capacitors to be used to replace J1 & J2 when selecting alternate BAUD rates.*
Prices were taken on February 11, 2007. Prices are single item pricing

Example using the Mega8535 and the C Programming Language:

This example code uses an AVR Atmega8535 microcontroller as a host device operating on a *Home-Brew* controller board running with a crystal frequency of 7.3728MHz. The user of the My_LCD Serial Backpack can use any microcontroller that provides an *On-Board* USART serial communications channel to communicate with the My_LCD serial backpack.

The C compiler used was the ImageCraft ICCAVR v7.xx compiler. Note that ImageCraft allows a 30 day trial period where there are no restriction on compiler functionality. After the initial 30 day trial period, the maximum code that can be compiled is around 1 Kilo-Byte. This code exceeds that 1 Kilo-Byte post 30 day trial

demonstration limitation and will not compile unless you have a licensed version of the compiler.

This code should not be too much effort to convert to another compiler manufacturer. If you use a GCC based C compiler, IAR, CodeVision, WinAVR, etc., the data-type and register access syntax for that particular C compiler will have to be adhered to, as well.

When using a substitute AVR microcontroller, the appropriate interrupt and USART differences will need to be taken into account. In addition, register names for that flavor AVR microcontroller will need to be considered.

This code works for 1x16, 2x16, 4x16, 2x20, 4x20, 2x40 & 4x40 text based LCD displays using the HD44780 chipset.

The LED backlight control switch is monitored via PORTB, pin 4 (PB:4).

Timer1is used as a 1 millisecond interrupt driven timer that establishes the necessary delays used in this example code.

My_LCD Firmware:

```
// My_LCD.c

/*
Two port, 8 BIT LCD control program
Writen by Carl W. Livingston
microcarl@roadrunner.com
AVR Freaks member - microcarl
January 02, 2007
*/

/*************************************************************/
/*************************************************************/
#include <iot2313v.h>
//#include <iom16v.h>
//#include <iom32v.h>
/*************************************************************/
/*************************************************************/

/*    //// From the LCD perspective \\\\
            RxD --> PORTD:0
            LED <-- PORTD:1 // High = ON, Low = OFF
            J_1 --> PORTD:2 // BAUD rate select
            J_2 --> PORTD:3 // BAUD rate select
            LCD:R/W <-- PORTD:4
            LCD:RS <-- PORTD:5
            LCD:E <-- PORTD:6
            N/A <-> PORTD:7

            LCD:R/W <-- PORTD:4
            LCD:RS <-- PORTD:5
            LCD:E <-- PORTD:6
            LCD:Vee <-- CONTRAST
            LCD:DB0 <-- PORTB:0
            LCD:DB1 <-- PORTB:1
            LCD:DB2 <-- PORTB:2
            LCD:DB3 <-- PORTB:3
            LCD:DB4 <-- 4.7K Ohm <-- PORTx:4
            LCD:DB5 <-- 4.7K Ohm <-- PORTx:5
            LCD:DB6 <-- 4.7K Ohm <-- PORTx:6
            LCD:DB7 <-- 4.7K Ohm <-- PORTx:7

      //// From the I/O PORTx perspective \\\\
            PORTD:0 <--RxD
            PORTD:1 --> LED // High = ON, Low = OFF
            PORTD:2 <-- J_1 // BAUD rate select
            PORTD:3 <-- J_2 // BAUD rate select
            PORTD:4 --> LCD:R/W
            PORTD:5 --> LCD:RS
            PORTD:6 --> LCD:E
            PORTD:7 <-> N/A

            PORTB:0 --> LCD:DB0
            PORTB:1 --> LCD:DB1
            PORTB:2 --> LCD:DB2
```

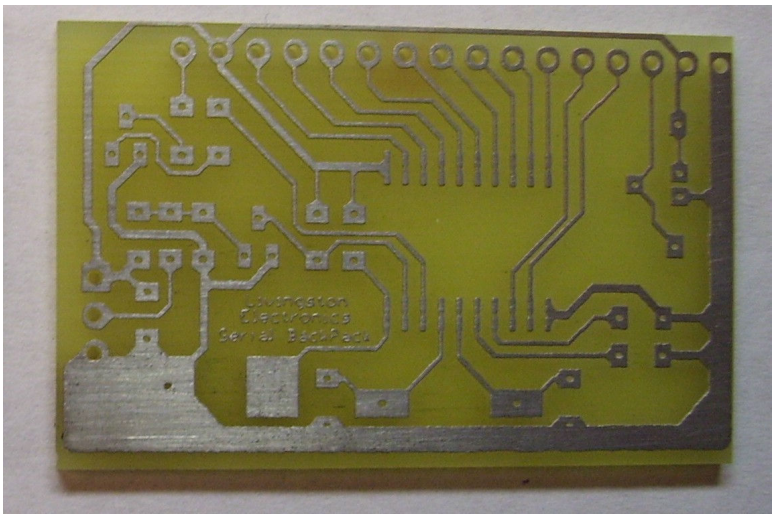
```
            PORTB:3 --> LCD:DB3
            PORTB:4 --> 4.7K Ohm --> LCD:DB4
            PORTB:5 --> 4.7K Ohm --> LCD:DB5
            PORTB:6 --> 4.7K Ohm --> LCD:DB6
            PORTB:7 --> 4.7K Ohm --> LCD:DB7
*/

/*************************************************************/
/*************************************************************/
// If you want to use a different I/O port for LCD control & data,
// do it here!!!
#define LCD_DATA_OUT PORTB
#define LCD_DATA_IN PINB
#define LCD_DATA_DDR DDRB

#define LCD_CONTROL_OUT PORTD
#define LCD_CONTROL_IN PIND
#define LCD_CONTROL_DDR DDRD

// Define LCD Read/Write as PORTx, 0x10;
#define LCD_RW 4
// Define LCD Register Select as PORTx, 0x20;
#define LCD_RS 5
// Define LCD Enable as PORTx, 0x40;
#define LCD_E 6
/*************************************************************/
/*************************************************************/

// LCD busy status bit
#define LCD_BUSY 7
// LED control bit
#define LED 1
// BAID rate control bits
#define J_1 2
#define J_2 3
#define CGRAM 6

// Turn on power to the display, no cursor
#define     PWR_ON      0x0C
// Set 8 data bits
#define     DATA_8      0x30
// Set 8 data bits, 4 display lines
#define     LINE_8x4    0x38
// Clear display command
#define CLR_DSP 0x01
// Character generator RAM command
#define REG_MODE 0xFE

#define NULL 0x00

// ASCII control code (0xFC) turns on LED
#define LED_SW_ON 252
// ASCII control code (0xFD) turns off LED
#define LED_SW_OFF 253

void MakeCustomChar (const char *, char);
void LCD_Delay (unsigned int d);
```

```
void LCD_INIT (void);
void LCD_PutCmd (char);
void LCD_PutChar (char);
char LCD_BusyWait (void);
void USART_Init (void);
char USART_BAUD (void);

char USART_DATA = NULL;

void main (void) {
      LCD_INIT ();

      // Initialize the AVR USART
      USART_Init ();

      while(1) {
      while (!(UCSRA & (1 << RXC))); // RXC1: USART Recieve Complete
            USART_DATA = UDR; // Get USART data


            if (USART_DATA == LED_SW_OFF)
                  LCD_CONTROL_OUT &= ~(1<<LED);
            else if (USART_DATA == LED_SW_ON)
                  LCD_CONTROL_OUT |= (1<<LED);
            else if (USART_DATA == REG_MODE) {
               while (!(UCSRA & (1 << RXC))); // RXC1: USART Recieve
Complete
               LCD_PutCmd (UDR); // Send LCD command character
            }
            else LCD_PutChar (USART_DATA); // Send LCD data character


      }
}

// Clock cycle = 67nS @ 14.7456MHz
// Delay resolution ~ 1uS @ 14.7456MHz
void LCD_Delay (unsigned int d) {
       while (d-- != 0);
      asm("nop");
}

void LCD_INIT (void) {
      // Set J_2:J_1 PULL-UPS active
      LCD_CONTROL_OUT = (1<<J_2) | (1<<J_1);
      // Set LCD_control J_2:J_1 to inputs
      LCD_CONTROL_DDR = 0xF2;

    // Delay resolution ~ 1uS @ 14.7456MHz
      LCD_Delay (15000); // Need 15mS delay for LCD to power up
      LCD_CONTROL_OUT |= (1<<LCD_RS); // Set LCD_RS HIGH

      // Initialize the AVR controller I/O
      LCD_DATA_DDR = 0xFF; // Set LCD_DATA_OUT as all outputs
      LCD_DATA_OUT = 0x00; // Set LCD_DATA_OUT to logic low

      // Initialize the LCD controller
```

```
        LCD_CONTROL_OUT &= ~(1<<LCD_RS);

        LCD_PutChar (DATA_8);
        // Delay resolution ~ 1uS @ 14.7456MHz
        LCD_Delay (4100); // Need 4.1mS delay

        LCD_PutChar (DATA_8);
        // Delay resolution ~ 1uS @ 14.7456MHz
        LCD_Delay (100); // Need 100uS delay

        LCD_PutChar (DATA_8);
        LCD_CONTROL_OUT |= (1<<LCD_RS);

        // The display will be out into 8 bit data mode here
        LCD_PutCmd (LINE_8x4); // Set 8 bit data, 4 display lines
        LCD_PutCmd (PWR_ON); // Power up the display
        LCD_PutCmd (CLR_DSP); // Power up the display
}

void LCD_PutCmd (char Cmd) {
        LCD_CONTROL_OUT &= ~(1<<LCD_RS);
        LCD_BusyWait();
        LCD_DATA_OUT = Cmd;
        asm ("nop");
        LCD_CONTROL_OUT |= (1<<LCD_E);
        asm ("nop"); // PWeh must be 230nS minimum
        asm ("nop"); // nop = 67nS @ 14.7456MHz
        asm ("nop");
        asm ("nop"); // Here, E = 271nS @ 14.7456MHz
        LCD_CONTROL_OUT &= ~(1<<LCD_E);
        asm ("nop");
        LCD_CONTROL_OUT |= (1<<LCD_RS);
}

void LCD_PutChar (char c) {
        LCD_CONTROL_OUT &= ~(1<<LCD_RS);
        LCD_BusyWait();
        LCD_CONTROL_OUT |= (1<<LCD_RS);
        LCD_DATA_OUT = c;
        asm ("nop");
        LCD_CONTROL_OUT |= (1<<LCD_E);
        asm ("nop"); // PWeh must be 230nS minimum
        asm ("nop"); // nop = 67nS @ 14.7456MHz
        asm ("nop");
        asm ("nop"); // Here, E = 271nS @ 14.7456MHz
        LCD_CONTROL_OUT &= ~(1<<LCD_E);
}

char LCD_BusyWait (void) {
        unsigned char LCDStatus;
        LCD_DATA_DDR = 0x00; // Set LCD data port to inputs
        LCD_CONTROL_OUT |= (1<<LCD_RW);
        asm ("nop");
        do {
           LCD_CONTROL_OUT |= (1<<LCD_E);
           asm ("nop"); // PWeh must be 230nS minimum
           asm ("nop"); // nop = 67nS @ 14.7456MHz
```

```
        asm ("nop");
        asm ("nop"); // Here, E = 271nS @ 14.7456MHz
        LCDStatus = LCD_DATA_IN;
        LCD_CONTROL_OUT &= ~(1<<LCD_E);
      } while ((LCDStatus & (1<<LCD_BUSY)) == (1<<LCD_BUSY));
      LCD_CONTROL_OUT &= ~(1<<LCD_RW);
      LCD_DATA_DDR |= 0xFF; // Set LCD data port to outputs
      return (LCDStatus);
}

// Initialize UART0
void USART_Init(void)
{
 UCSRB = NULL; // Disable while setting baud rate
 UCSRA = NULL;
 UCSRC = (1<<UCSZ1) | (1<<UCSZ0); // 8 bit data
 UBRRL = USART_BAUD (); // Set baud rate
 UBRRH = NULL; // Set baud rate hi
 UCSRB = (1<<RXEN); // RXEN = Enable
}

// Actual baud rate = 9600 BAUD, (0.0% error) @ 14.7456MHz
// Actual baud rate = 19.2K BAUD, (0.0% error) @ 14.7456MHz
// Actual baud rate = 38.4K BAUD, (0.0% error) @ 14.7456MHz
// Actual baud rate = 115.2K BAUD, (0.0% error) @ 14.7456MHz
#define BAUD_4800 191
#define BAUD_9600 95
#define BAUD_14400 63
#define BAUD_19200 47
#define BAUD_28800 31
#define BAUD_38400 23
#define BAUD_57600 15
#define BAUD_76800 11
#define BAUD_115200 7

char USART_BAUD (void) {
      char BaudSelectJumpersValue;
      static char BaudLookupTable[] = {BAUD_115200, BAUD_38400,
BAUD_19200, BAUD_9600};

      // Get BAUD rate jumper settings
      BaudSelectJumpersValue  = LCD_CONTROL_IN;
      // Mask off unwanted bits
      BaudSelectJumpersValue &= (1<<J_2) | (1<<J_1);
      // J_2 = PD.3, J_1 = PD.2
      // This is two bits too far to the left and the array index will
      // increment by 4, instead of 1.
      // Shift J_2 & J_1 right by two bit positions for proper array
indexing
      BaudSelectJumpersValue = (BaudSelectJumpersValue >> J_1);

      return BaudLookupTable[BaudSelectJumpersValue];
}
```

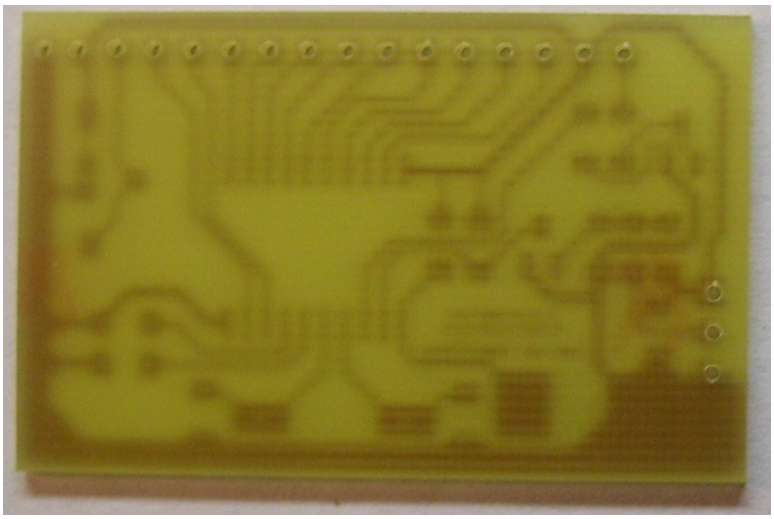
Blank PCB, Foil (Component) Side:



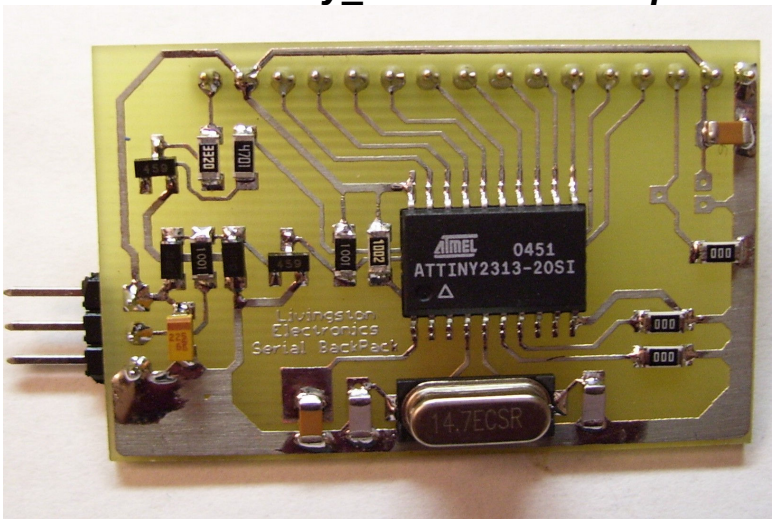
The printed circuit board is made out of FR4 Glass filled resin laminated with 2Oz copper clad.

Blank PCB, Blank (Header) Side:

Using surface mount components and careful trace layout, the back side of the printed circuit board has no components present that might otherwise interfere with, and short against any components present on the LCD module.



The Assembled My_LCD Serial Backpack Module:

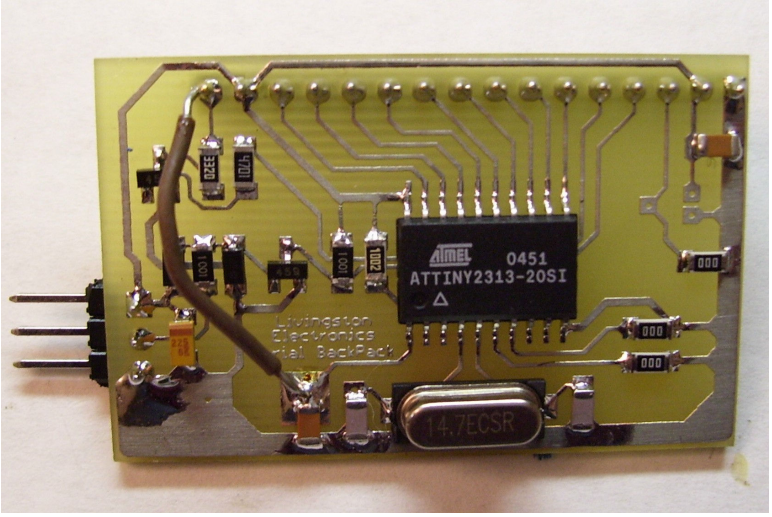


This is the fully assembled My_LCD Serial Backpack. The three header (HD1) pins protruding from the Left side of the module are +5 VDC Power (Top), RS232 In (Center) and, 5 VDC GND (Bottom). The three empty pads on the

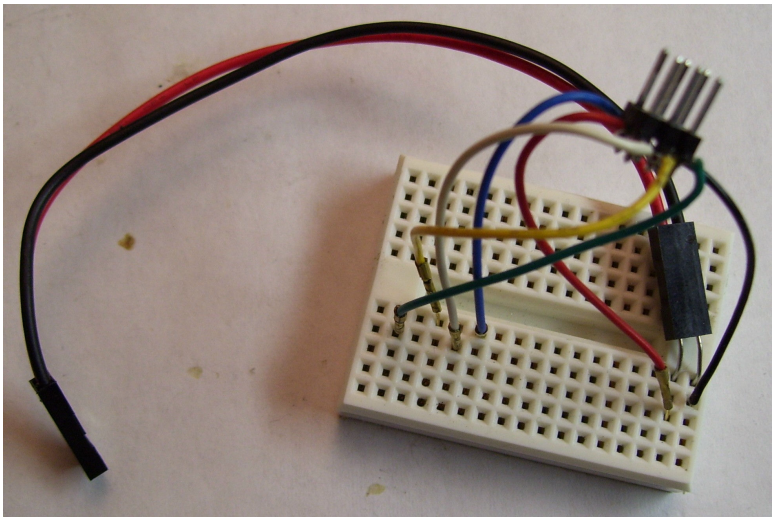
Right side of the PCB are for an optional SMD potentiometer to adjust the LCD contrast.

Adding A Temporary Jumper Between RESET And Pin 16:

To program the My_LCD serial Backpack, one temporary jumper is needed to bring the microcontroller RESET signal out to pin 16 of header HD2. This temporary jumper will be removed, after the My_LCD serial Backpack is programmed.



Wiring a Proto-Board To Program The My_LCD Serial Backpack:



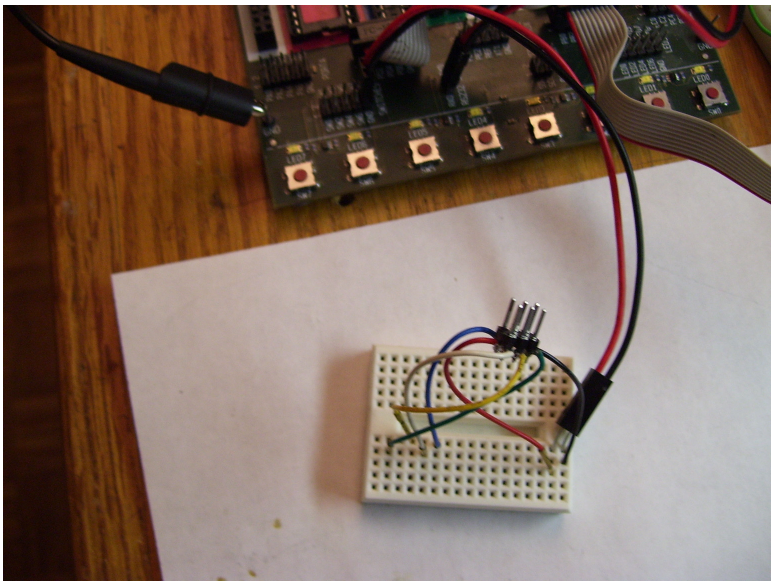
A small 18 row Proto-Board is used to plug the 16 pin header HD2 on the My_LCD Serial Backpack for programming. Care must be taken to connect the wires coming from the 6 pin ISP header to the proper header pins on the Serial Backpack module.

ISP Programming Connections				
ISP	Function	HD1 Connections	Tiny2313	Color
Pin 1	MISO	HD2-13	Pin 18	White
Pin 2	VCC	HD2-2	Pin 20	Red
Pin 3	SCK	HD2-14	Pin 19	Yellow
Pin 4	MOSI	HD2-12	Pin 17	Blue
Pin 5	RESET	HD2-16	Pin 1	Green
Pin 6	GND	HD2-1	Pin 10	Black

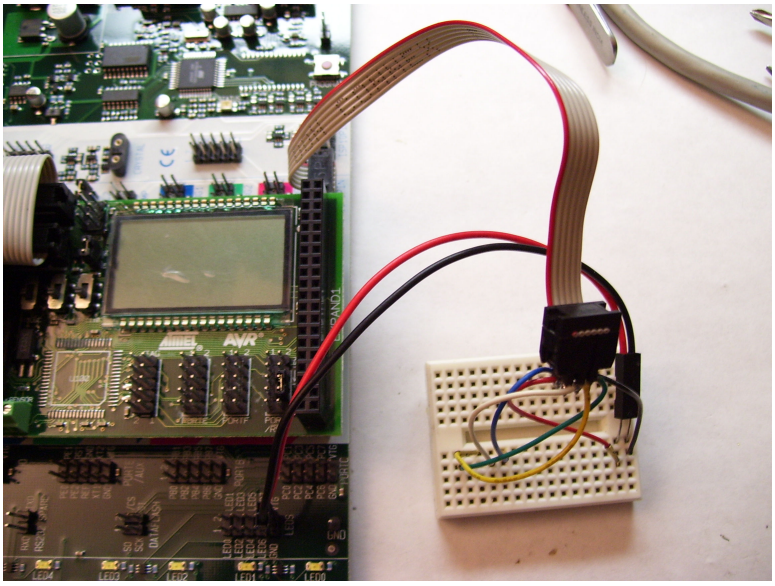
Connecting Power to the Programming Proto-Board:

The picture on the right shows VCC and GND connected to VGT on one of the PORT headers on the STK500 development board. VGT is set to 5.0 VDC.

Pay close attention to the polarity of VGT and the way it is connected to the proto-board. If connected backwards, it will more then likely destroy the My_LCD serial backpack.



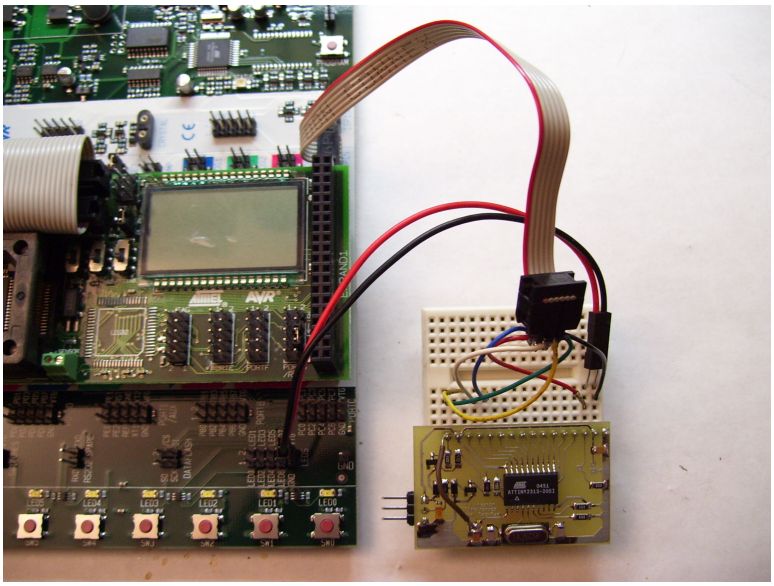
Connecting the ISP Programming Cable:



This picture shows the 6 pin ISP cable connected to the temporary programming proto-board and the STK500 ISP programming header.

Be very attentive to the proper orientation of the ISP plug on both, the STK500 ISP header and the make-shift ISP header used to connect the pins to the proto-board.

Mounting The My_LCD Serial Backpack To The Proto-Board:



Here the My_LCD serial backpack is plugged into the proto-board.

Check all wiring against the datasheet and the table above before applying power.

Recheck all of the wiring and make sure that you can verify each and every wire for proper electrical connection. A few more minutes spent on this phase of the programming setup will save you many hours of grief if the programming setup is powered up with a miss-wire.

Once all of the wiring is verified, the programming of the My_LCD serial backpack can be performed. When the My_LCD has been successfully programmed, remove the My_LCD serial backpack module from the proto-board and remove the temporary jumper wire from pin 16 and RESET. You are now ready to mate the My_LCD serial backpack to the LCD display. Enjoy!