Application Note
Interfacing an ezLCD30x to a Stellaris LM3S811

Daniel Valvano
University of Texas at Austin

http://users.ece.utexas.edu/~valvano/arm/ezLCD_811.zip

Purpose

The purpose of posting this example is to assist engineers who are interfacing an ezLCD30x to a microcontroller and to a Stellaris microcontroller in particular. For more information about the Texas Instruments LM3S811, search "LM3S811" on www.ti.com. This code was tested both on the EKK-LM3S811 development board an on a custom PCB also using an LM3S811.

For more information about the UART, ADC, PWM, GPIO, and PLL code in this example, refer to the book "Embedded Systems: Real Time Interfacing to the Arm Cortex M3", ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011. See http://users.ece.utexas.edu/~valvano/arm/outline.htm

Copyright

Software Files included in the ezLCD_811.zip

Recorder2.uvproj	Keil uVision4 project
Recorder2.c	Main program for the testing interface
ezLCD.c ezLCD.h	High-level interface to ezLCD
UART2.c UART2.h	Low-level serial interface to ezLCD
pll.c pll.h	Phase-lock-loop to run at 50 MHz
pwm.c pwm.h	Output to speaker on PD0 (ezLCD buttons beep)
ADCT0ATrigger.c	Timer-triggered ADC sampling
fifo.h	First in first out queue for data streaming
startup.s	Reset and interrupt vectors

How to run it on an EKK-LM3S811 development board

1) Connect (set `#define UART 1` in UART2.c)

```
U1Rx/PD2 serial from ezLCD-301 Rx  Pin 7
U1Tx/PD3 serial to   ezLCD-301 Tx  Pin 8
3.3V        to       ezLCD-301 Vcc Pin 16
Ground     between   ezLCD-301 Gnd Pin 15
PD0 to speaker
ADC1 to analog input 0 to 3V
```

2) Open project **Recorder2.uvproj** in Keil uVision4

3) In **Recorder.c** set the compile options

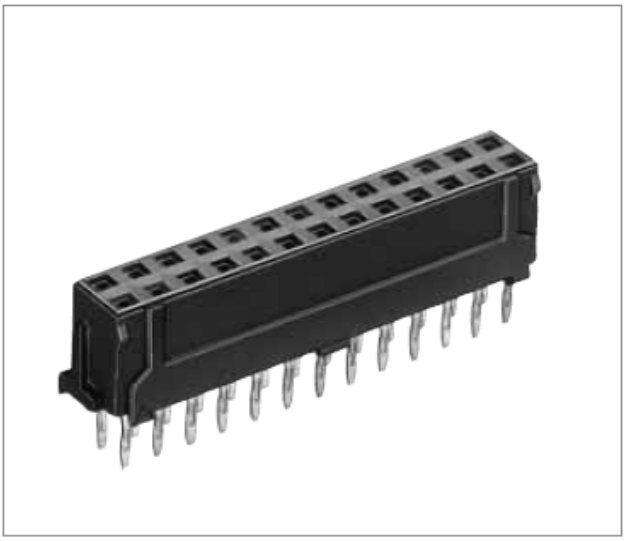
```
// The test file has four modes:
// 0: ADC test suite with buttons
// 1: multiple point speed test suite
// 2: sine wave plot speed test suite
// 3: time entry keypad
#define RECORDER2MODE           3
// Do you want to skip the test patterns?
// 0: do not skip the test patterns
// 1: skip the test patterns
#define RECORDER2SKIPPATTERNS   1
```

4) Build, download, and run

Other files

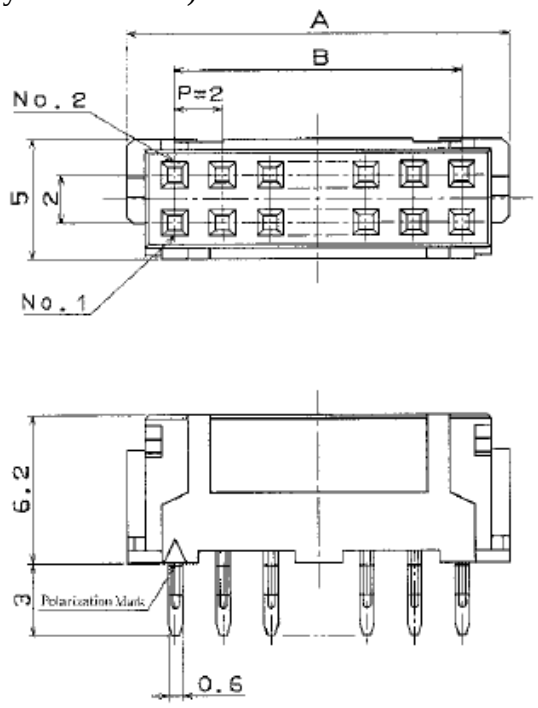
ezLCD.sch ezLCDsch.pdf	Circuit diagram in PCBartist
ezLCD.pcb ezLCDpcb.pdf	Board layout in PCBartist
recorder2.xls	Spread sheet to create test sequences
cat.gif	400 by 240 pixel picture (Isaac the cat)
DF11-16DS-2DSA.pdf	16-pin connector between PCB and ezLCD Digikey H10188-ND, Hirose DF11-16DS-2DSA
PacTec_XP.pdf	PCB/ezLCD fits into this PacTec box 81491-510-000 or 85292-510-000

Hirose DF11-16DS-2DSA Female connector on PCB (Digikey H10188-ND)

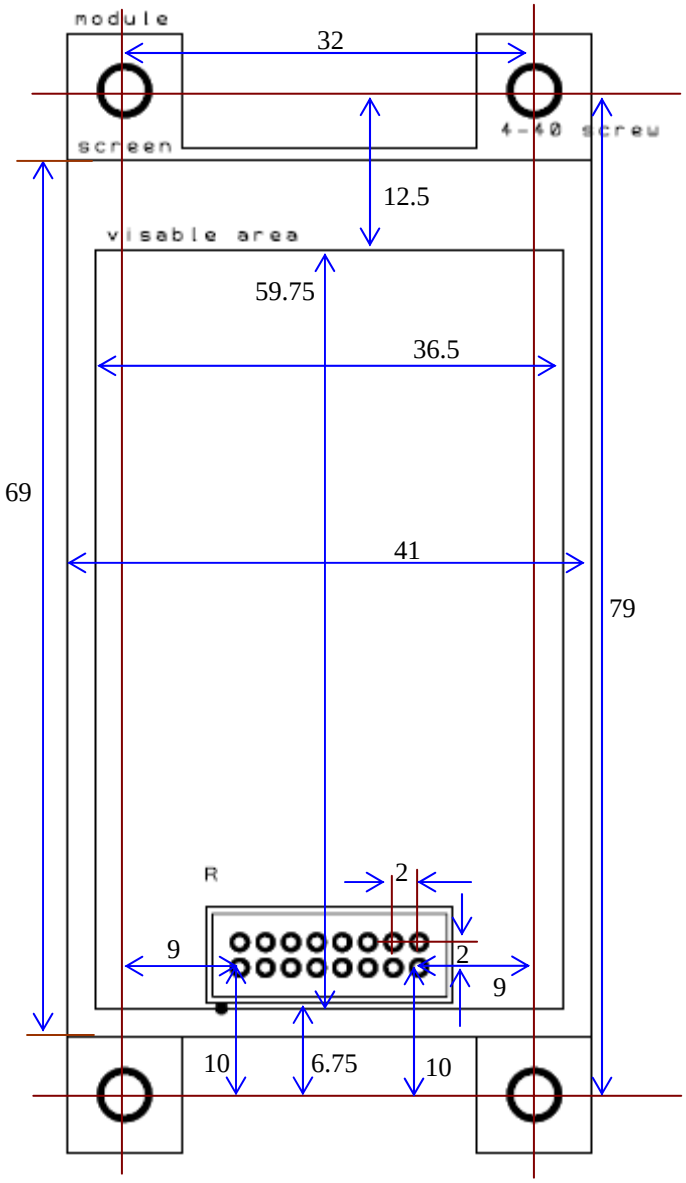


●Board Through-hole Diameter: $\phi 0.8^{+0.1}_{0}$

●Kink Prefixed Effective Diameter: $\phi 0.8^{+0.05}_{0}$

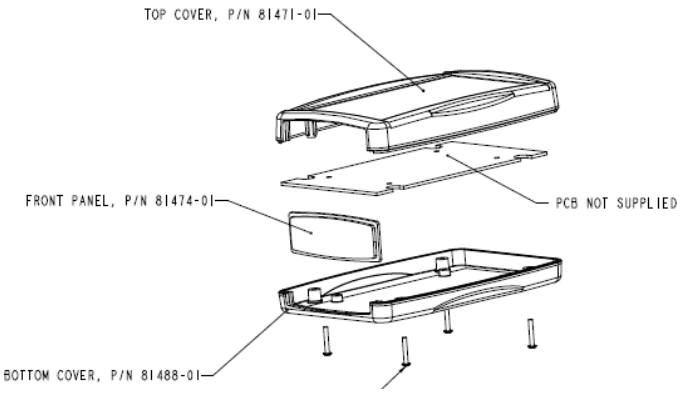


ezLCD layout (all dimensions are in mm) Visible area is used to cut out window in enclosure

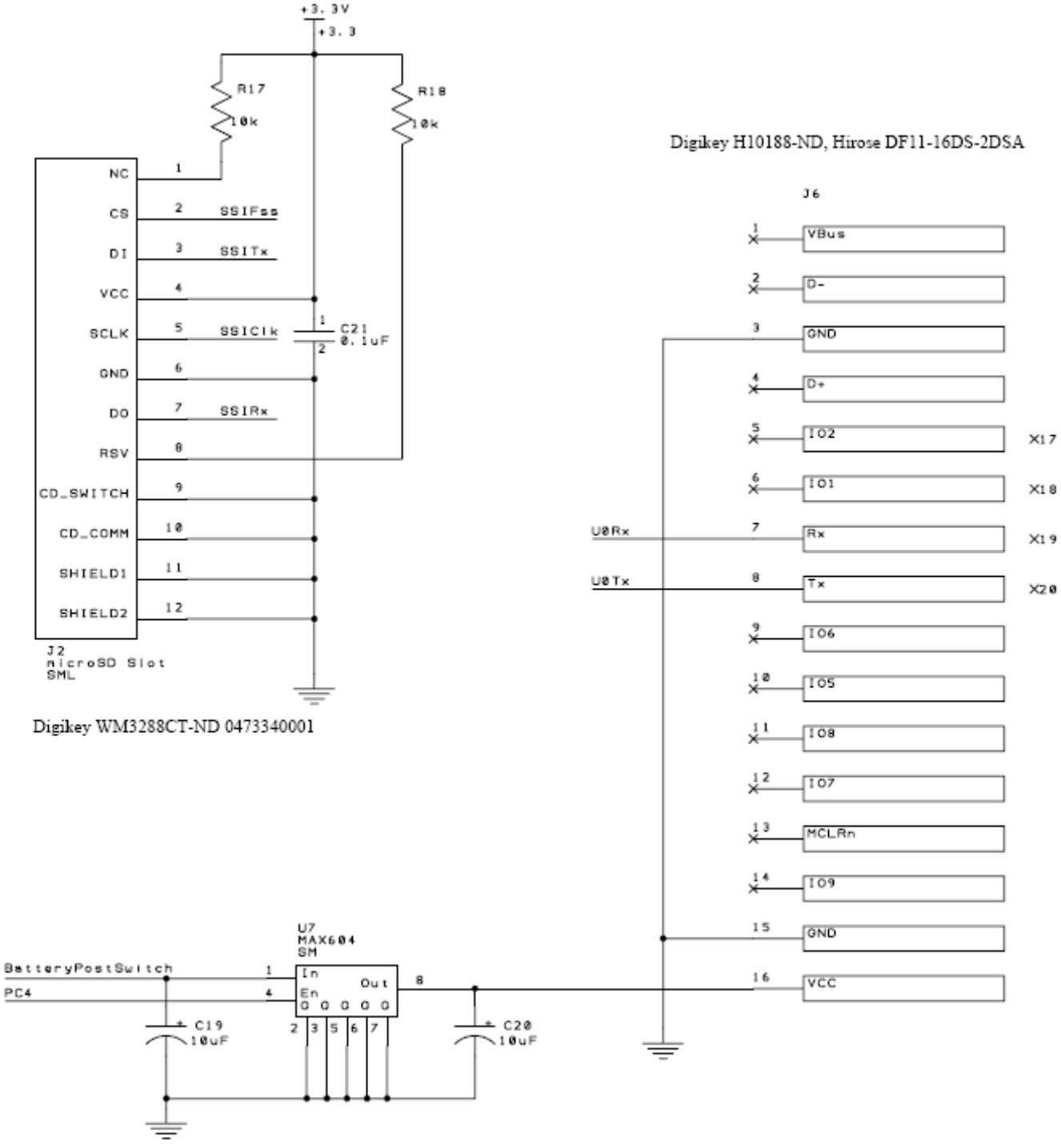


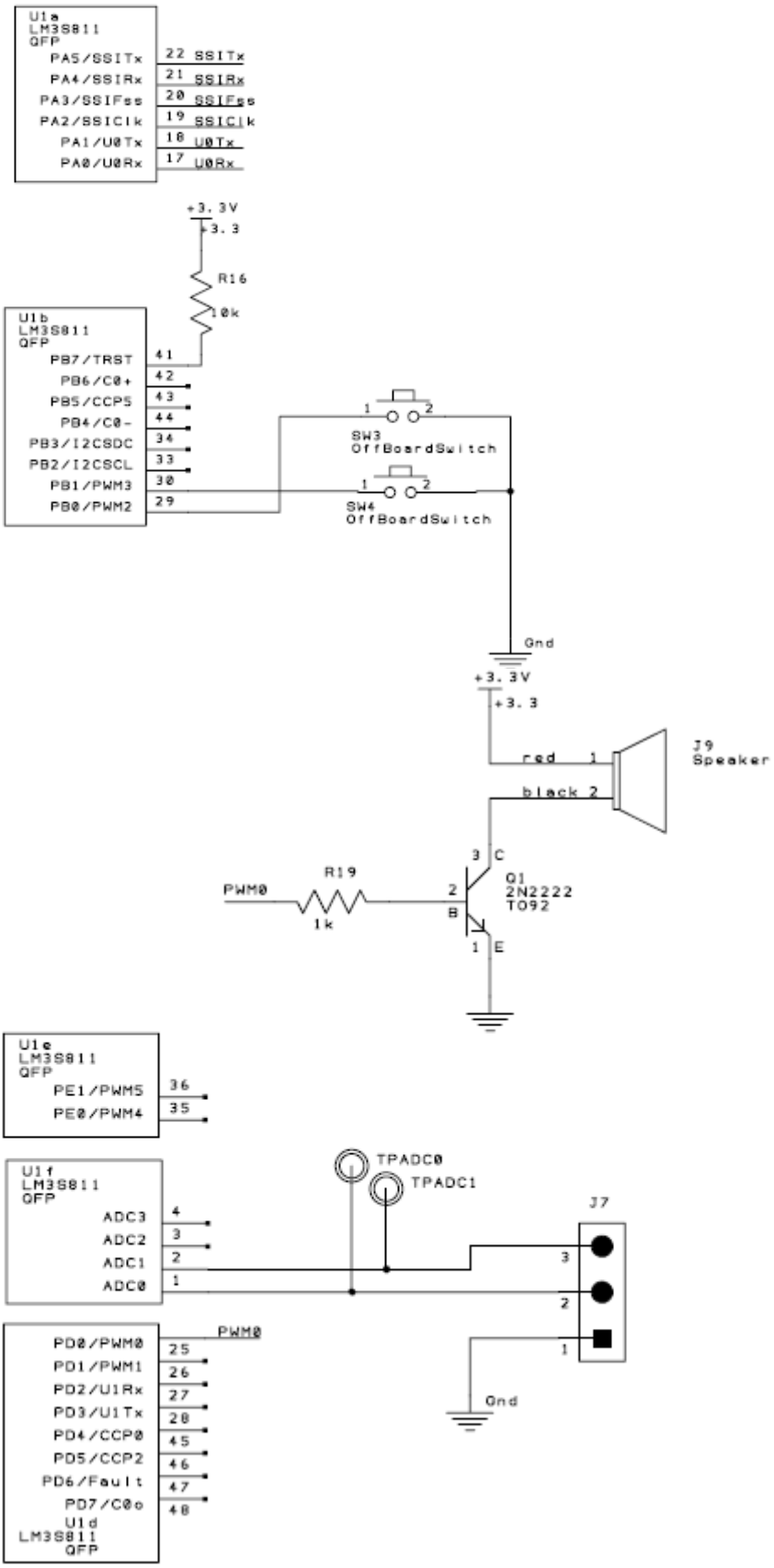
Visible area is 36.5 by 59.75 mm
Screen is 41 by 69 mm
4-40 holes are 32 by 79 mm apart
4-40 hole to pin 9 mm over 10 mm up,
measured between hole centers
4-40 holes are drilled at 3mm diameter
(Diameter=(screw number)*13+60mil,
so a 4-40 screw = 4*13+60=112 mil, plus
8 mil tolerance = 120mil = 3mm)
Pins are 2 mm apart (0.7mm holes)
Visible area is 6.75 mm from bottom holes
Visible area is 12.5 mm from top holes

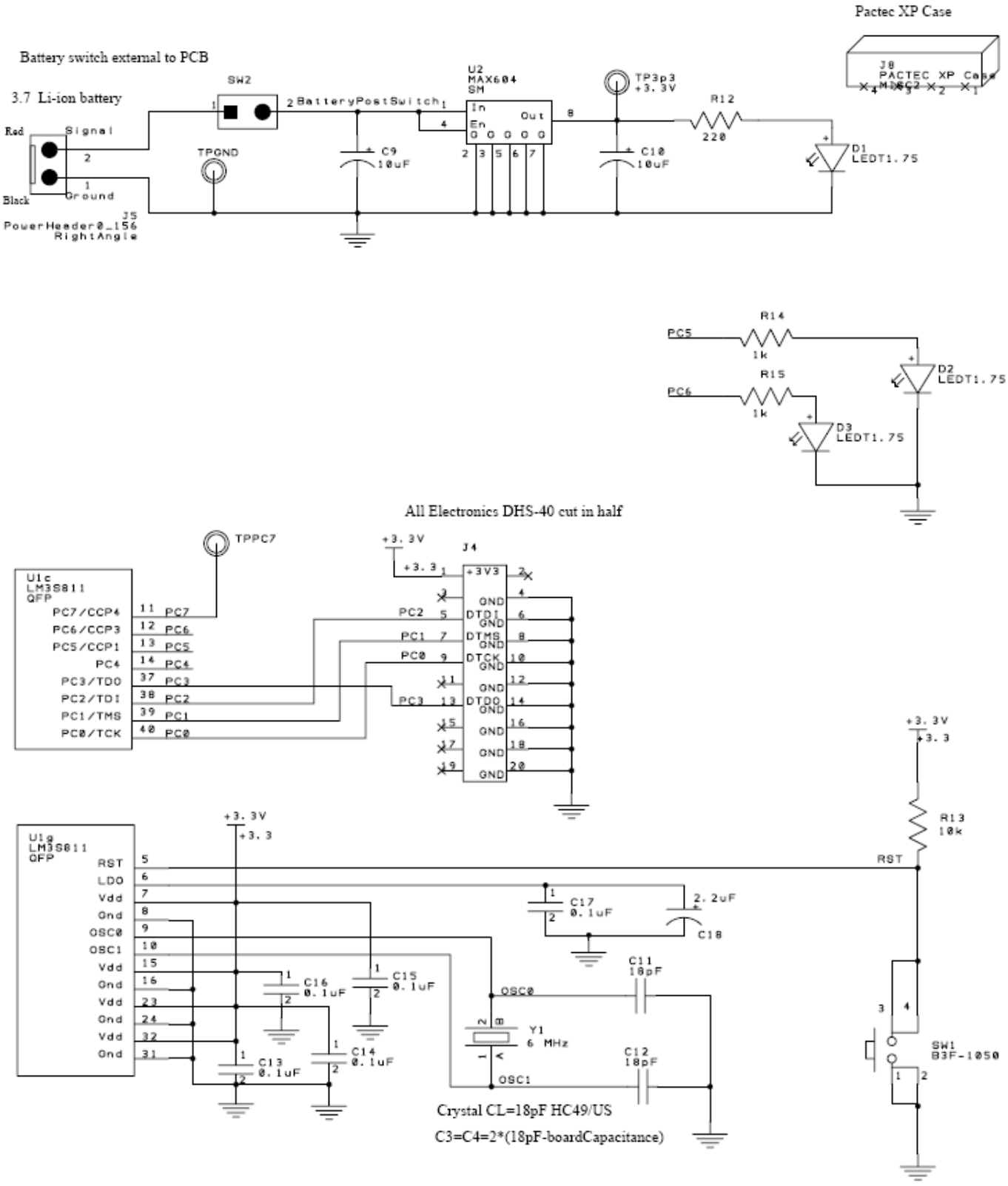
PacTec XP box



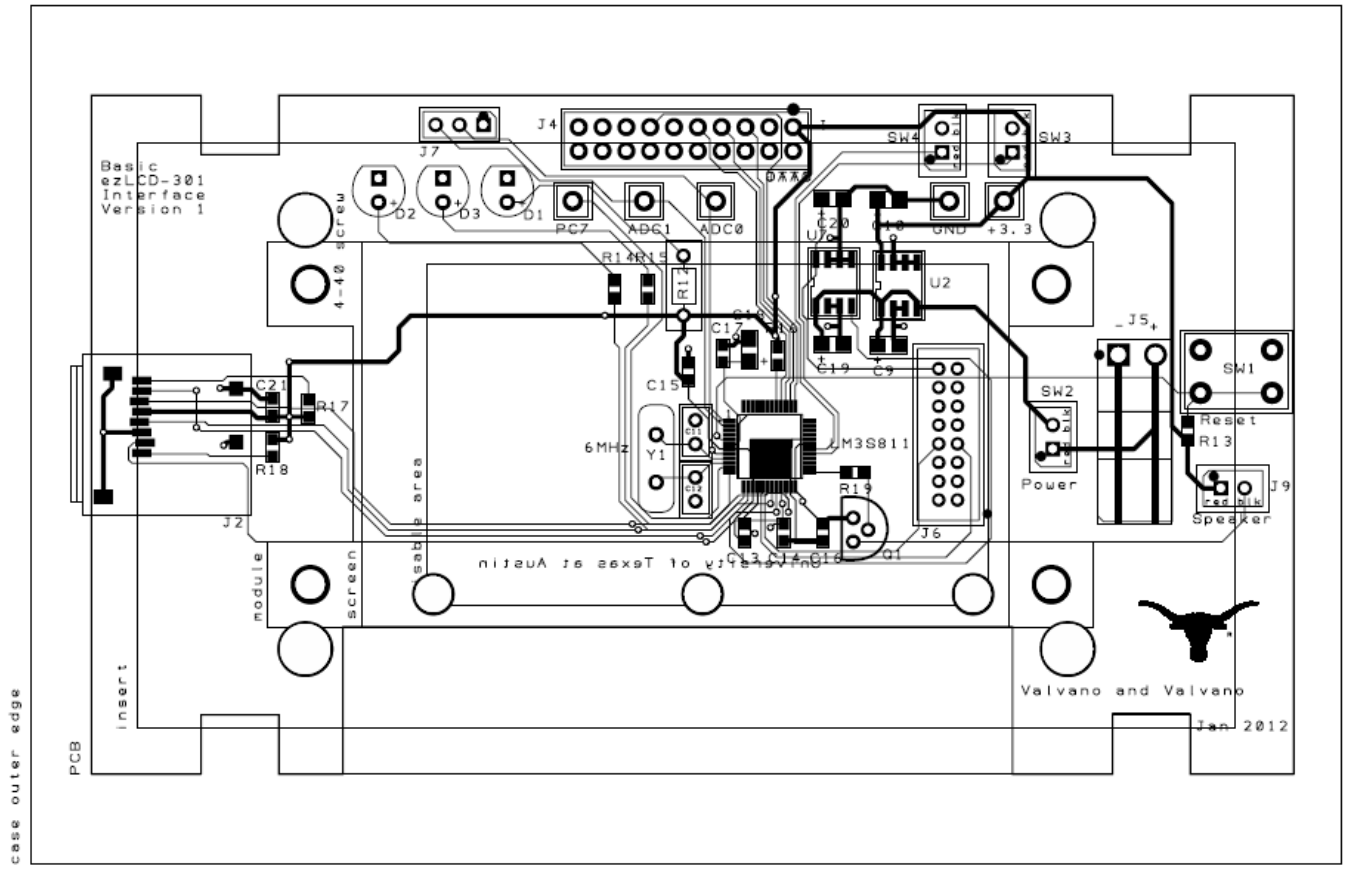
Circuit Diagram (see ezLCD.sch ezLCDsch.pdf for all the details). Drawn in PCBartist.
Set **PC4** high to apply power to ezLCD, set **PC4** low when in low-power sleep mode




U1a
LM3S811
QFP
PA5/SSITx
PA4/SSIRx
PA3/SSIFss
PA2/SSIClk
PA1/U0Tx
PA0/U0Rx
22 SSITx
21 SSIRx
20 SSIFss
19 SSIClk
18 U0Tx
17 U0Rx
+3.3V
+3.3
R16
10k
U1b
LM3S811
QFP
PB7/TRST
PB6/C0+
PB5/CCP5
PB4/C0-
PB3/I2CSDA
PB2/I2CSCL
PB1/PWM3
PB0/PWM2
41
42
43
44
34
33
30
29
SW3
OffBoardSwitch
SW4
OffBoardSwitch
Gnd
+3.3V
+3.3
red
black
J9
Speaker
PWM0
R19
1k
Q1
2N2222
TO92
C
B
E
U1e
LM3S811
QFP
PE1/PWM5
PE0/PWM4
36
35
U1f
LM3S811
QFP
ADC3
ADC2
ADC1
ADC0
TPADC0
TPADC1
J7
PWM0
PD0/PWM0
PD1/PWM1
PD2/U1Rx
PD3/U1Tx
PD4/CCP0
PD5/CCP2
PD6/Fault
PD7/C0o
U1d
LM3S811
QFP
25
26
27
28
45
46
47
48
Gnd


Pactec XP Case
Battery switch external to PCB
3.7 Li-ion battery
SW2
BatteryPostSwitch
U2
MAX604
SM
TP3p3
+3.3V
R12
220
Red
Signal
Black
Ground
TPGND
C9
10uF
C10
10uF
D1
LEDT1.75
J5
PowerHeader0_156
RightAngle
R14
1k
R15
1k
D2
LEDT1.75
D3
LEDT1.75
PC5
PC6
All Electronics DHS-40 cut in half
TPPC7
U1c
LM3S811
QFP
PC7/CCP4
PC6/CCP3
PC5/CCP1
PC4
PC3/TDO
PC2/TDI
PC1/TMS
PC0/TCK
J4
+3V3
GND
DTDI
DTMS
DTCK
DTDO
U1g
LM3S811
QFP
RST
LDO
Vdd
Gnd
OSC0
OSC1
C16
0.1uF
C15
0.1uF
C13
0.1uF
C14
0.1uF
C17
0.1uF
2.2uF
C18
C11
18pF
C12
18pF
Y1
6 MHz
R13
10k
SW1
B3F-1050
Crystal CL=18pF HC49/US
C3=C4=2*(18pF-boardCapacitance)

PCB Layout Diagram (see ezLCD.pcb ezLCDpcb.pdf for most details)



If you want to build your own PCB, you can download my PCBartist library files, which include all these parts. http://users.ece.utexas.edu/~valvano/Starterfiles/PCBArtistLibrary.zip

General comments

The UART is running at 115,200 baud rate. Interrupts are enabled for both transmit and receive. The bus clock is 50 MHz clock. The UART protocol is 8-bit word length, no parity bits, one stop bit, FIFOs enabled. It was essential to place a logic analyzer on the transmit and receive pins and observe the serial traffic in order to figure out exactly what the ezLCD was sending.

The members of the Stellaris family of microcontrollers (LM3Sxxxx) have similar I/O ports. Translating this example to other members of this family involves changing the lowest level mapping defining the UART pins. For example, the ADC, UART2, and ezLCD drivers will run on a LM3S1968 and LM3S8962 without any changes at all. To get the main program to work on either of these two processors involves changing the PLL code, PWM, and the GPIO for the regular buttons.

This system as you can see also interfaced a Secure Digital Card. Measured data can be accumulated and stored on the SD Card for future retrieval at a desktop computer. For more information on this interface see Chapter 6 of Embedded Systems: Real-Time Operating Systems for the Arm® Cortex™-M3, Volume 3, 2012, Jonathan Valvano, ISBN: 978-1466468863. See http://users.ece.utexas.edu/~valvano/arm/outline3.htm.

Features The ezLCD.h file outlines the features of this interface

```
// ezLCD.h
// Runs on LM3S811
```

```
// Use UART to implement bidirectional data transfer to and from an
// ezLCD-301.  Provide functions to display a plot and graph data
// over time or distance.  Functions also send the proper strings to
// the ezLCD-301 to implement simple functions such as clearing the
// screen, moving the cursor, or drawing simple shapes.
// Daniel Valvano
// November 17, 2011

/*
 Copyright 2012 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */
// either (see #define UART in UART2.c)
// U1Rx/PD2 serial from ezLCD-301 Rx  Pin 7
// U1Tx/PD3 serial to   ezLCD-301 Tx  Pin 8
// 3.3V        to       ezLCD-301 Vcc Pin 16
// Ground     between   ezLCD-301 Gnd Pins 3,15
// or
// U0Rx/PA0 serial from ezLCD-301 Rx  Pin 7
// U0Tx/PA1 serial to   ezLCD-301 Tx  Pin 8
// 3.3V        to       ezLCD-301 Vcc Pin 16
// Ground     between   ezLCD-301 Gnd Pins 3,15

// The following is a table of preset colors and their index values.
// These values are the defaults and can be changed with the function
// ezLCD_SetColorID(colorID, red, green, blue);
// Basic Colors
#define BLACK            0
#define GRAY             1
#define SILVER           2
#define WHITE            3
#define RED              4
#define MAROON           5
#define YELLOW           6 //looks good on black background
#define OLIVE            7
#define LIME             8 //looks good on black background
#define GREEN            9
#define AQUA            10
#define TEAL            11
#define BLUE            12
#define NAVY            13
#define FUCHSIA         14
#define PURPLE          15
// Red Colors
#define INDIANRED       16
#define LIGHTCORAL      17
#define SALMON          18
#define DARKSALMON      19
#define LIGHTSALMON     20
```

```
#define RED2            21 //same as 4
#define CRIMSON         22
#define FIREBRICK       23
#define DARKRED         24
// Pink Colors
#define PINK            25
#define LIGHTPINK       26
#define HOTPINK         27
#define DEEPPINK        28
#define MEDIUMVIOLETRED 29
#define PALEVIOLETRED   30
// Orange Colors
#define LIGHTSALMON2    31 //same as 20
#define CORAL           32
#define TOMATO          33
#define ORANGERED       34
#define DARKORANGE      35
#define ORANGE          36
// Yellow Colors
#define GOLD            37
#define YELLOW2         38 //slightly different than 6
#define LIGHTYELLOW     39
#define LEMONCHIFFON    40
#define LIGHTGOLDENROD  41
#define PAPAYAWHIP      42
#define MOCCASIN        43
#define PEACHPUFF       44
#define PALEGOLDENROD   45
#define KHAKI           46
#define DARKKHAKI       47
// Purple Colors
#define LAVENDER        48
#define THISTLE         49
#define PLUM            50
#define VIOLET          51
#define ORCHID          52
#define FUCHSIA2        53 //same as 14
#define MEDIUMORCHID    54
#define MEDIUMPURPLE    55
#define BLUEVIOLET      56
#define DARKVIOLET      57
#define DARKORCHID      58
#define DARKMAGENTA     59
#define PURPLE2         60 //same as 15
#define INDIGO          61
#define DARKSLATEBLUE   62
#define SLATEBLUE       63
#define MEDIUMSLATEBLUE 64
// Green Colors
#define GREENYELLOW      65
#define CHARTREUSE       66
#define LAWNGREEN        67
#define LIME2            68 //same as 8
#define LIMEGREEN        69
#define PALEGREEN        70
#define LIGHTGREEN       71
#define MEDIUMSPRINGREEN 72
```

```
#define SPRINGGREEN      73
#define MEDIUMSEAGREEN   74
#define SEAGREEN         75
#define FORESTGREEN      76
#define GREEN2           77 //same as 9
#define DARKGREEN        78
#define YELLOWGREEN      79
#define OLIVEDRAB        80
#define OLIVE2           81 //same as 7
#define DARKOLIVEGREEN   82
#define MEDIUMAQUAMARINE 83
#define DARKSEAGREEN     84
#define LIGHTSEAGREEN    85
#define DARKCYAN         86
#define TEAL2            87 //same as 11
// Blue/Cyan Colors
#define AQUA2           88 //same as 10
#define CYAN            89
#define LIGHTCYAN       90
#define PALETURQUOISE   91
#define AQUAMARINE      92
#define TURQUOISE       93
#define MEDIUMTURQUOISE 94
#define DARKTURQUOISE   95
#define CADETBLUE       96
#define STEELBLUE       97
#define LIGHTSTEELBLUE  98
#define POWDERBLUE      99
#define LIGHTBLUE      100
#define SKYBLUE        101
#define LIGHTSKYBLUE   102
#define DEEPSKYBLUE    103
#define DODGERBLUE     104
#define CORNFLOWERBLUE 105
#define ROYALBLUE      106
#define BLUE2          107 //slightly different than 12
#define MEDIUMBLUE     108
#define DARKBLUE       109
#define NAVY2          110 //same as 13
#define MIDNIGHTBLUE   111
// Brown Colors
#define CORNSILK       112
#define BLANCHEDALMOND 113
#define BISQUE         114
#define NAVAJOWHITE    115
#define WHEAT          116
#define BURLYWOOD      117
#define TAN            118
#define ROSYBROWN      119
#define SANDYBROWN     120
#define GOLDENROD      121
#define DARKGOLDENROD  122
#define PERU           123
#define CHOCOLATE      124
#define SADDLEBROWN    125
#define SIENNA         126
#define BROWN          127
```

```
#define MAROON2        128 //same as 5
// White Colors
#define WHITE2         129 //same as 3
#define SNOW           130
#define HONEYDEW       131
#define MINTCREAM      132
#define AZURE          133
#define ALICEBLUE      134
#define GHOSTWHITE     135
#define WHITESMOKE     136
#define SEASHELL       137
#define BEIGE          138
#define OLDLACE        139
#define FLORALWHITE    140
#define IVORY          141
#define ANTIQUEWHITE   142
#define LINEN          143
#define LAVENDERBLUSH  144
#define MISTYROSE      145
// Gray Colors
#define GAINSBORO        146
#define LIGHTGRAY        147
#define SILVER2          148 //same as 2
#define DARKGRAY         149
#define GRAY2            150 //same as 1
#define DIMGRAY          151
#define LIGHTSLATEGRAY   152
#define SLATEGRAY        153
#define MEDIUMTURQUOISE2 154 //same as 94
#define DARKSLATEGREY    155
#define BLACK2           156 //same as 0
// Extra Colors
#define GRAY7          157
#define GRAY20         158
#define GRAY40         159
#define GRAY80         160
#define GRAY90         161
#define GRAY95         162
#define RED4           163 //slightly different than 4 and 21
#define FIREBRICK1     164 //slightly different than 23
#define DARKGREEN2     165 //same as 78
#define PALEGREEN2     166 //same as 70
#define LIGHTYELLOW2   167 //same as 39

// justification is according to the following:
// ****************************
// * LT         CT         RT *
// *                          *
// * LC         CC         RC *
// *                          *
// * LB         CB         RB *
// ****************************
enum justification {
  LT,             // left-top justified
  CT,             // center-top justified
  RT,             // right-top justified
  LC,             // left-center justified
```

```
  CC,             // center justified
  RC,             // right-center justified
  LB,             // left-bottom justified
  CB,             // center-bottom justified
  RB              // right-bottom justified
};

// button option is according to the following:
// 0=remove button; 1=draw button; 2=disable button;
// 3=press button (cosmetic); 4=toggle button (cosmetic)
enum buttonoption {
//  buttonRemove = 0,  // button response off (not supported after about firmware
V1.06)
  buttonDraw = 1,    // button response on
  buttonDisable = 2, // button response off
  buttonPress = 3,   // button response unchanged
  buttonToggle = 4   // button response unchanged
};

// button label text alignment is according to the following:
// 0=centered; 1=right; 2=left; 3=bottom; 4=top
enum buttonalign {
  buttonTxtCenter = 0,
  buttonTxtRight = 1,
  buttonTxtLeft = 2,
  buttonTxtBottom = 3,
  buttonTxtTop = 4   // may not be supported before about firmware V1.06
};

typedef struct ezLCD_Button_Type {
    char *label;            // string to be displayed on the button
    void (*callPress)(void);// user-define callback when this button is pressed
    void (*callRel)(void);  // user-define callback when this button is released
    enum buttonoption option;// button option
} ezLCD_Button_Obj;

//------------ezLCD_Init------------
// Initialize UART for 115,200 baud rate (assuming 50 MHz clock),
// 8 bit word length, no parity bits, one stop bit, FIFOs enabled.
// Input: none
// Output: none
void ezLCD_Init(void);

//------------ezLCD_BasicGraphInit------------
// Clears LCD and draws graph axes.
// Input: backgroundColor  color ID of background (0 to 199),
//        axisColor        color ID of axes and labels (0 to 199)
// Output: none
// Modifies: ezLCD drawing color, font, line width,
//  line type, text orientation, and cursor
// Assumes: "verbose" mode off
void ezLCD_BasicGraphInit(unsigned short backgroundColor, unsigned short axisColor);

//------------ezLCD_GraphInit------------
// Clears LCD and draws graph axes.  The axes marked with
// the specified number of tick marks and numbered with
// the values provided in two arrays (one for the X-axis
```

```
// and one for the Y-axis).  The length of the tick marks
// is given in number of pixels.  The axes can be labeled
// with two separate NULL-terminated strings.
// Suggestions:
// *It looks best if you limit to three characters on the
// X-axis numbers and two characters on the Y-axis numbers.
// *Insert spaces before the Y-axis label text if it does
// not look centered.
// *Use a light foreground and a dark background, such as
// AQUA on BLACK, LIME on BLACK, YELLOW on BLACK,
// PINK on BLACK, or WHITE on BLACK
// Input: backgroundColor  color ID of background (0 to 199),
//        axisColor        color ID of axes and labels (0 to 199),
//        xPt              pointer to array of X-axis labels,
//        numX             number of X-axis labels,
//        yPt              pointer to array of Y-axis labels,
//        numY             number of Y-axis labels,
//        tickLength       axis tick length (pixels),
//        xLabel           NULL terminated string X-axis label,
//        yLabel           NULL terminated string Y-axis label
// Output: none
// Modifies: ezLCD drawing color, font, line width,
//  line type, text orientation, and cursor
// Assumes: "verbose" mode off
void ezLCD_GraphInit(unsigned short backgroundColor, unsigned short axisColor,
                     const unsigned short *xPt, unsigned short numX,
                     const unsigned short *yPt, unsigned short numY,
                     unsigned short tickLength, char *xLabel,
                     char *yLabel);

//------------ezLCD_GraphPoint------------
// Graphs a point and then increments the X-position of the
// next point to be plotted.  Assume that ezLCD_GraphInit()
// has already been called.  Once the cursor exceeds the end
// of the X-axis, the next point is graphed starting on the
// leftmost position in the graph area.
// Input: data        10-bit data point to be plotted,
//        pointColor  color ID of point (0 to 199)
// Output: none
// Modifies: ezLCD drawing color and cursor
// Assumes: line width=1 (1 pixel wide), line type=0 (solid)
//  "verbose" mode off
// NOTE: takes about 30 ms at 19,200 baud rate and "verbose" off.
void ezLCD_GraphPoint(unsigned short data, unsigned short pointColor);

//------------ezLCD_GraphPointNoInc------------
// Graphs a point without incrementing the X-position of the
// next point to be plotted.  Assume that ezLCD_GraphInit()
// has already been called.  Once the cursor exceeds the end
// of the X-axis, the next point is graphed starting on the
// leftmost position in the graph area.
// Input: data        10-bit data point to be plotted,
//        pointColor  color ID of point (0 to 199)
// Output: none
// Modifies: ezLCD drawing color
// Assumes: "verbose" mode off
void ezLCD_GraphPointNoInc(unsigned short data, unsigned short pointColor);
```

```
//------------ezLCD_GraphPointNoIncNoColor------------
// Graphs a point without incrementing the X-position of the
// next point to be plotted.  Assume that ezLCD_GraphInit()
// has already been called.  Once the cursor exceeds the end
// of the X-axis, the next point is graphed starting on the
// leftmost position in the graph area.  In this case, the
// drawing color is assumed to be correct for the point(s)
// to be plotted.  This function is intended for more
// efficient plotting of many pre-calculated values.
// Structure your program like this:
// ezLCD_SetColor(equation1Color);
// for(i=0; i<340; i=i+1){
//   ezLCD_GraphPointNoIncNoColor(equation1Values[i]);
//   ezLCD_GraphIncrement();
// }
// ezLCD_SetColor(equation2Color);
// for(i=0; i<340; i=i+1){
//   ezLCD_GraphPointNoIncNoColor(equation2Values[i]);
//   ezLCD_GraphIncrement();
// }
// Input: data  10-bit data point to be plotted
// Output: none
// Assumes: "verbose" mode off
// NOTE: takes about 8 ms at 19,200 baud rate and "verbose" off.
void ezLCD_GraphPointNoIncNoColor(unsigned short data);

//------------ezLCD_GraphIncrement------------
// Increment the X-position of the next point to be plotted.
// Previously plotted data is not removed.  If you want to
// "sweep" across the plot from left to right and then
// automatically start over from the left, structure your
// program like this:
// ezLCD_GraphPointNoInc(data1, color1);
// ezLCD_GraphPointNoInc(data2, color2);
// ezLCD_GraphPointNoInc(data3, color3);
// ezLCD_GraphPoint(dataLast, colorLast);
// If you want to display all recorded data points without
// ever removing any, structure your program like this:
// ezLCD_GraphPointNoInc(data1, color1);
// ezLCD_GraphPointNoInc(data2, color2);
// ezLCD_GraphPointNoInc(data3, color3);
// ezLCD_GraphPointNoInc(dataLast, colorLast);
// ezLCD_GraphIncrement();
// Input: none
// Output: none
void ezLCD_GraphIncrement(void);

//------------ezLCD_Reset------------
// Send "RESET" command to ezLCD-301.  On startup, the macro
// "startup.ezm" should be run from the Flash.
// Input: none
// Output: none
// Modifies: ezLCD reset
void ezLCD_Reset(void);

//------------ezLCD_ClearScreen------------
```

```
// Send "CLS" command to ezLCD-301.  The screen will be
// solidly filled with the specified color.  Any buttons
// will be disabled.
// Input: clearColor  color ID to use to fill the screen (0 to 199)
// Output: none
// Assumes: "verbose" mode off
void ezLCD_ClearScreen(unsigned short clearColor);

//------------ezLCD_Ping------------
// Test the ezLCD-301 using the "PING" command.
// In "verbose" mode, the response is "Pong".
// Otherwise, the response is "0\r".
// Input: none
// Output: 0 if no error; 1 otherwise
// Assumes: "verbose" mode off
int ezLCD_Ping(void);

//------------ezLCD_Verbose------------
// Send "VERBOSE" command to ezLCD-301.  When in verbose
// mode, the ezLCD-301 replies to the microcontroller
// through the UART with a greater volume of text after
// each successful command.  Although this can be useful
// for debugging, the text is generally meaningless for
// stand-alone applications.  Verbose mode also greatly
// slows down the speed at which commands are executed,
// since the biggest bottleneck is usually the baud rate
// of the UART.  Verbose mode affects synchronization.
// Most of the functions in this file are written using
// UART_OutCommand(), which assumes that the ezLCD-301
// is not in verbose mode.  UART_OutCommandV() should be
// used when certain that verbose mode is enabled.
// Input: enable  new verbose setting to change to
//                (0=disable verbose or 1=enable verbose)
// Output: none
void ezLCD_Verbose(unsigned short enable);

//------------ezLCD_SetLight------------
// Send "LIGHT X" command to ezLCD-301.  The LED back-
// light is set to X% of maximum brightness.  The
// default is 75%.
// Input: newLight  new LED brighness (0 to 100) percent
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetLight(unsigned short newLight);

//------------ezLCD_GetLight------------
// Send "LIGHT" command to ezLCD-301.  Return the
// current LED backlight value as X% of maximum
// brightness.  The default is 75%.
// In "verbose" mode, the response is "Get Light: XXX\r".
// Otherwise, the response is "XXX\r".
// Results will be unpredictable if response is not
// in this format.
// Input: none
// Output: current LED brighness (0 to 100) percent
// Assumes: "verbose" mode off
unsigned short ezLCD_GetLight(void);
```

```
//------------ezLCD_SetColor------------
// Send "COLOR X" command to ezLCD-301.  The next thing
// that is drawn will be the color of the specified
// color ID.  An uninitialized color ID may be black.
// Input: newColor  new color ID of drawing color (0 to 199)
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetColor(unsigned short newColor);

//------------ezLCD_GetColor------------
// Send "COLOR" command to ezLCD-301.  Return the
// current drawing color values.
// In "verbose" mode, the response is
// "Get Color: R=XX G=XX B=XX\r".
// Otherwise, the response is "XX XX XX\r".
// Results will be unpredictable if response is not
// in this format.
// Input: none
// Output: red, green, and blue values of the drawing color
// Assumes: "verbose" mode off
void ezLCD_GetColor(unsigned short *red, unsigned short *green, unsigned short
*blue);

//------------ezLCD_SetColorID------------
// Send "COLORID index R G B" command to ezLCD-301.  The
// specified red, green, and blue values will be stored
// in the specified color ID.  To see this color, use
// "ezLCD_ClearScreen(colorID);".  If the ezLCD-301 loses
// power, all custom colors may return to default.
// Input: colorID  destination color ID (16 to 199) (recommend 169 to 199),
//        red      new red value,
//        green,   new green value,
//        blue     new blue value
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetColorID(unsigned short colorID, unsigned short red, unsigned short
green, unsigned short blue);

//------------ezLCD_GetColorID------------
// Send "COLORID index" command to ezLCD-301.  Return
// the specified color ID's red, green, and blue values.
// In "verbose" mode, the response to "COLORID 5" is
// "Get Color ID 5: R=16 G=0 B=0\r".
// Otherwise, the response is "16 0 0\r".
// Results will be unpredictable if response is not
// in this format.
// Input: colorID  color index number to look at (0 to 199)
// Output: red, green, and blue values of specified color ID
// Assumes: "verbose" mode off
// WARNING: this command may not work
//  Specifically, "COLORID 180 83 120 179" sets color 180
//  to "IBM blue".  However, "COLORID 180" does not
//  return R=83, G=120, B=179.  It returns
//  R=10, G=30, B=22, which is a totally different color.
void ezLCD_GetColorID(unsigned short colorID, unsigned short *red, unsigned short
*green, unsigned short *blue);
```

```
//------------ezLCD_SetFontStr------------
// Send "FONT \"font\"" command to ezLCD-301.  This sets
// the active font to the specified font name.  To send a
// font index number instead, use ezLCD_SetFontID().
// Input: newFont  NULL terminated string name of new font
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetFontStr(char *newFont);

//------------ezLCD_SetFontID------------
// Send "FONT index" command to ezLCD-301.  This sets the
// active font to the specified index number.  To send a
// font string name instead, use ezLCD_SetFontStr().
// Input: fontID  new font index number
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetFontID(unsigned short fontID);

//------------ezLCD_SetFontW------------
// Send "FONTW widget \"font\"" command to ezLCD-301.
// This sets the font to be used with the specified
// widget.  Widgets are used to draw button objects,
// and this function must be called to attach a font
// to any widgets that will be used.
// Widgets seem to return to default when the ezLCD
// loses power, although this function still must be
// called.
// Input: widgetIndex    font widget ID,
//        newWidgetFont  NULL terminated string name of font
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetFontW(unsigned short widgetIndex, char *newWidgetFont);

//------------ezLCD_SetFontO------------
// Send "FONTO value" command to ezLCD-301.  This
// sets the font orientation to 0 (horizontal) or
// 1 (vertical).  Sending other values may have
// unpredictable results.
// Input: newFontO  new font orientation value
//                  (0=horizontal or 1=vertical)
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetFontO(unsigned short newFontO);

//------------ezLCD_GetFontO------------
// Send "FONTO" command to ezLCD-301.  This gets
// the current font orientation as 0 (horizontal)
// or 1 (vertical).
// In "verbose" mode, the response is
// "Font Orient: Vertical\r" or
// "Font Orient: Horizontal\r".
// Otherwise, the response is "1\r" or "0\r".
// Return 2 if response is not in this format.
// Input: none
// Output: current font orientation value
// Assumes: "verbose" mode off
```

```
unsigned short ezLCD_GetFontO(void);

//------------ezLCD_SetLinewidth------------
// Send "LINEWIDTH value" command to ezLCD-301.
// This sets the line width to 1 or 3 pixels wide.
// Line width applies to regular lines, the box,
// and the circle.
// Sending other values may have unpredictable
// results.
// Input: newLineWidth  new line width (pixels) (1 or 3)
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetLinewidth(unsigned short newLinewidth);

//------------ezLCD_GetLinewidth------------
// Send "LINEWIDTH" command to ezLCD-301.  This
// gets the current line width as 1 or 3 pixels
// wide.
// In "verbose" mode, the response is
// "Get Line Width: 3\r".
// Otherwise, the response is "3\r".
// Results will be unpredictable if response is not
// in this format.
// Input: none
// Output: current line width (pixels)
// Assumes: "verbose" mode off
unsigned short ezLCD_GetLinewidth(void);

//------------ezLCD_SetLinetype------------
// Send "LINETYPE value" command to ezLCD-301.
// This sets the line type by inserting the
// specified number of pixels between line
// segments, creating a dotted or dashed line.
// For a solid line, specify 0 pixel gap.
// Line type applies to regular lines, the box,
// and the circle.
// Input: newLineType  new number of pixels between line segments
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetLinetype(unsigned short newLinetype);

//------------ezLCD_GetLinetype------------
// Send "LINETYPE" command to ezLCD-301.  This
// gets the current line type as the number of
// pixels between line segments.
// In "verbose" mode, the response is
// "Get Line Type: 5\r".
// Otherwise, the response is "5\r".
// Results will be unpredictable if response is not
// in this format.
// Input: none
// Output: current number of pixels between line segments
// Assumes: "verbose" mode off
unsigned short ezLCD_GetLinetype(void);

//------------ezLCD_SetCursor------------
// Send "XY x y" command to ezLCD-301.  This sets
```

```
// the drawing cursor to the specified X and Y
// values if they are legal.
// Input: newX  new X value of the cursor (0 to 399),
//        newY  new Y value of the cursor (0 to 239)
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetCursor(unsigned short newX, unsigned short newY);

//------------ezLCD_GetCursor------------
// Send "XY" command to ezLCD-301.  This gets the
// drawing cursor's X and Y values.
// In "verbose" mode, the response is
// "Get XY: X=50 Y=50\r".
// Otherwise, the response is "50 50\r".
// Results will be unpredictable if response is not
// in this format.
// Input: none
// Output: x  current X value of the cursor,
//         y  current Y value of the cursor
// Assumes: "verbose" mode off
void ezLCD_GetCursor(unsigned short *x, unsigned short *y);

//------------ezLCD_Plot------------
// Send "PLOT" command to ezLCD-301.  This plots a
// pixel at the drawing cursor's X and Y values in
// the current color.
// Input: none
// Output: none
// Assumes: "verbose" mode off
void ezLCD_Plot(void);

//------------ezLCD_PlotXY------------
// Send "PLOT x y" command to ezLCD-301.  This
// plots a pixel at the specified X and Y values
// in the current color.
// Input: ptX  pixel X value (0 to 399),
//        ptY  pixel Y value (0 to 239)
// Output: none
// Assumes: "verbose" mode off
void ezLCD_PlotXY(unsigned short ptX, unsigned short ptY);

//------------ezLCD_Line------------
// Send "LINE x y" command to ezLCD-301.  This
// draws a line from the drawing cursor's X and
// Y values to the specified X and Y values.
// Input: lineX  X value of line endpoint (0 to 399),
//        lineY  Y value of line endpoint (0 to 239)
// Output: none
// Assumes: "verbose" mode off
void ezLCD_Line(unsigned short lineX, unsigned short lineY);

//------------ezLCD_Box------------
// Send "BOX w h fill" command to ezLCD-301.
// This draws a box with its top left corner at
// the drawing cursor's X and Y values with the
// specified width and height (in pixels).  A
// non-zero value of "fill" fills the box.
```

```
// If the line type (number of pixels between line
// segments) is not zero and the box is not filled,
// the box's perimeter will be dotted.
// Input: width  box width (pixels) (1 to 400),
//        height box height (pixels) (1 to 240),
//        fill   non-zero for filled box
// Output: none
// Assumes: "verbose" mode off
void ezLCD_Box(unsigned short width, unsigned short height, unsigned short fill);

//------------ezLCD_Circle------------
// Send "CIRCLE radius fill" command to ezLCD-301.
// This draws a circle centered at the drawing
// cursor's X and Y values with the specified
// radius (in pixels).  A non-zero value of "fill"
// fills the circle.
// If the line type (number of pixels between line
// segments) is not zero and the circle is not filled,
// the circle's perimeter will be dotted.
// Input: radius  circle radius (pixels),
//        fill    non-zero for filled circle
// Output: none
// Assumes: "verbose" mode off
void ezLCD_Circle(unsigned short radius, unsigned short fill);

//------------ezLCD_Arc------------
// Send "ARC radius start end" command to ezLCD-301.
// This draws an arc centered at the drawing
// cursor's X and Y values with the specified
// radius (in pixels) and the specified start and
// end angles (in degrees).  An angle of 0 degrees
// is in the 3:00 position.  An angle of 90 degrees
// is in the 6:00 position.
// Input: radius  arc radius (pixels),
//        start   arc starting angle (0 to end),
//        end     arc ending angle (start to 359)
// Output: none
// Assumes: "verbose" mode off
// WARNING: In firmware versions 1.11 and earlier,
//  this command sometimes changes the drawing cursor's
//  X and Y values when the line width is set to 3.
//  The line width is always set to 1.
void ezLCD_Arc(unsigned short radius, unsigned short start, unsigned short end);

//------------ezLCD_Pie------------
// Send "PIE radius start end" command to ezLCD-301.
// This draws a pie slice centered at the drawing
// cursor's X and Y values with the specified
// radius (in pixels) and the specified start and
// end angles (in degrees).  An angle of 0 degrees
// is in the 3:00 position.  An angle of 90 degrees
// is in the 6:00 position.
// If the line type (number of pixels between line
// segments) is not zero, the pie slice will look
// like a slice of a target.
// Input: radius  pie slice radius (pixels),
//        start   slice starting angle (0 to end),
```

```
//        end     slice ending angle (start to 359)
// Output: none
// Assumes: "verbose" mode off
void ezLCD_Pie(unsigned short radius, unsigned short start, unsigned short end);

//------------ezLCD_PrintStr------------
// Send "PRINT string" command to ezLCD-301.  This
// prints the NULL-terminated string at the drawing
// cursor's X and Y values with the pre-configured
// font, color, and text orientation.  The drawing
// cursor is located at the top left corner of the
// first character to be printed.  The drawing cursor
// is then moved to the end of the printed string.
// If you wish to continue printing more text on the
// same line, simply call ezLCD_PrintStr() again
// without needing to move the drawing cursor.
// Input: text   NULL terminated string to print
// Output: none
// Assumes: "verbose" mode off
// Modifies: ezLCD drawing cursor
void ezLCD_PrintStr(char *text);

//------------ezLCD_PrintStrJustified------------
// Send "PRINT string justification" command to
// ezLCD-301.  This prints the NULL-terminated string
// at the drawing cursor's X and Y values with the
// pre-configured font, color, and text orientation
// at the specified justification.  The drawing cursor
// is then moved to the end of the printed string.
// If you wish to continue printing more text on the
// same line, simply call ezLCD_PrintStrJustified()
// again with the same justification code without
// needing to move the drawing cursor.
// justification is according to the following:
// ****************************
// * LT         CT         RT *
// *                          *
// * LC         CC         RC *
// *                          *
// * LB         CB         RB *
// ****************************
// Input: text   NULL terminated string to print,
//        jcode  justification code
//               (LT, CT, RT, LC, CC, RC, LB, CB, or RB)
// Output: none
// Assumes: "verbose" mode off
// Modifies: ezLCD drawing cursor
void ezLCD_PrintStrJustified(char *text, enum justification jcode);

//------------ezLCD_PrintUDec------------
// Send "PRINT string" command to ezLCD-301.  This
// prints the 16-bit number at the drawing cursor's
// X and Y values with the pre-configured font, color,
// and text orientation.  The drawing cursor is
// located at the top left corner of the first
// character to be printed.  The drawing cursor is
// then moved to the end of the printed string.  If
```

```
// you wish to continue printing more text on the same
// line, simply call ezLCD_PrintUDec() again without
// needing to move the drawing cursor.
// Input: n      16-bit number to print
// Output: none
// Assumes: "verbose" mode off
// Modifies: ezLCD drawing cursor
void ezLCD_PrintUDec(unsigned short n);

//------------ezLCD_PrintUDecJustified------------
// Send "PRINT string justification" command to
// ezLCD-301.  This prints the 16-bit number at the
// drawing cursor's X and Y values with the
// pre-configured font, color, and text orientation
// at the specified justification.  The drawing cursor
// is then moved to the end of the printed string.
// If you wish to continue printing more text on the
// same line, simply call ezLCD_PrintUDecJustified()
// again with the same justification code without
// needing to move the drawing cursor.
// justification is according to the following:
// ****************************
// * LT         CT         RT *
// *                          *
// * LC         CC         RC *
// *                          *
// * LB         CB         RB *
// ****************************
// Input: n      16-bit number to print,
//        jcode  justification code
//               (LT, CT, RT, LC, CC, RC, LB, CB, or RB)
// Output: none
// Assumes: "verbose" mode off
// Modifies: ezLCD drawing cursor
void ezLCD_PrintUDecJustified(unsigned short n, enum justification jcode);

//------------ezLCD_SetThemeID------------
// Send "THEME <parameters>" command to ezLCD-301.  This
// sets the specified theme to the specified parameters.
// Themes are used to draw buttons.  Themes may return to
// default values if the ezLCD-301 loses power.  Default
// values as of firmware version 1.01 are the following:
// THEME 0 1 2 0 0 0 3 3 3 3         '(white/black)
// THEME 1 155 152 3 3 3 0 0 0 0     '(black/white)
// THEME 2 5 20 3 3 3 4 4 4 4        '(red/white)
// THEME 3 9 3 0 0 0 8 8 8 8         '(green/black)
// THEME 4 7 3 0 0 0 6 6 6 6         '(yellow/black)
// THEME 5 126 118 3 3 3 35 35 35 35 '(orange/white)
// THEME 6 111 106 3 3 3 12 12 12 12 '(blue/white)
// THEME 7 58 48 3 3 3 14 14 14 14   '(pink/white)
// Input: themeID             theme ID number (0 to 15),
//        embossDkColor       color ID of button shadow (lower right edge),
//        embossLtColor       color ID of button highlight (upper left edge),
//        textColorRest       color ID of button text at rest,
//        textColorTouched    color ID of button text when touched,
//        textColorDisabled   color ID of button text when disabled,
//        buttonColorRest     color ID of button face at rest,
```

```
//        buttonColorTouched  color ID of button face when touched,
//        buttonColorDisabled color ID of button face when disabled,
//        commonBGColor       color ID of common background color (possibly used when
button set to "remove")
//        widgetIndex         font widget ID of button text (not supported before
firmware V1.06)
// Output: none
// Assumes: "verbose" mode off
void ezLCD_SetThemeID(unsigned short themeID, unsigned short embossDkColor,
                      unsigned short embossLtColor, unsigned short textColorRest,
                      unsigned short textColorTouched, unsigned short
textColorDisabled,
                      unsigned short buttonColorRest, unsigned short
buttonColorTouched,
                      unsigned short buttonColorDisabled, unsigned short
commonBGColor,
                      unsigned short widgetIndex);

//------------ezLCD_Button------------
// Send "BUTTON id X Y w h option align radius theme text"
// command to ezLCD-301.  This creates a button with its
// top left corner at the specified X and Y values with
// the specified width and height (in pixels).  The option
// is one of the following:
// buttonDraw, buttonDisable, buttonPress, buttonToggle
// The label text alignment is one of the following:
// buttonTxtCenter, buttonTxtRight, buttonTxtLeft,
// buttonTxtBottom, buttonTxtTop
// The radius is the amount of roundness in the button's
// corners (in pixels).  A radius of 0 creates a square
// button.  A radius of half the width or height creates
// a round or hotdog shaped button.  The theme index sets
// the appearance of the button to the specified theme.
// See ezLCD_SetThemeID().  The text is the text label on
// the button.  Finally, the pressFunc and releaseFunc are
// pointers to functions which are called when the button
// is pressed or released. Try to keep these functions
// short, since they will be called from the UART
// interrupt, and delays may cause the UART to miss data
// from the ezLCD-301.  Buttons still work if they are
// covered by other graphics, but when the button is
// pressed, it is automatically re-drawn, so the graphics
// may need to be re-drawn also.  All buttons are lost
// when the screen is cleared.
// Input: id          button ID number (0 to MAXBUTTONS),
//        x           top left corner X value (0 to 399),
//        y           top left corner Y value (0 to 239),
//        width       button width (pixels) (1 to 400),
//        height      button height (pixels) (1 to 240),
//        option      button option code
//                    (buttonDraw, buttonDisable,
//                     buttonPress, or buttonToggle),
//        align       button text alignment code
//                    (buttonTxtCenter, buttonTxtRight,
//                     buttonTxtLeft, buttonTxtBottom,
//                     or buttonTxtTop),
//        radius      corner smoothing radius (pixels)
```

```
//                    (0 to width/2) or (0 to height/2)
//                    whichever is smaller,
//        theme       button theme index (0 to 15),
//        textLabel   NULL terminated string to label button,
//        pressFunc   function to call when button is pressed,
//        releaseFunc function to call when button is released
// Output: none
// Assumes: "verbose" mode off
// Modifies: ezLCD string variable
void ezLCD_Button(unsigned char id, unsigned short x, unsigned short y,
                  unsigned short width, unsigned short height,
                  enum buttonoption option, enum buttonalign align,
                  unsigned short radius, unsigned short theme,
                  char *textLabel, void (*pressFunc)(void),
                  void (*releaseFunc)(void));

//------------ezLCD_ButtonDefaultAction------------
// Default action for button responses.  Currently does
// nothing.
// Input: none
// Output: none
void ezLCD_ButtonDefaultAction(void);

//------------ezLCD_ButtonPress------------
// Function used by external programs to call a button's
// "callPress" function.  Before the function is called,
// the button is checked that it is active and the function
// is checked that it is not NULL.
// Input: id    button ID number of pressed button (1 to MAXBUTTONS-1)
// Output: none
void ezLCD_ButtonPress(unsigned char id);

//------------ezLCD_ButtonRelease------------
// Function used by external programs to call a button's
// "callRel" function.  Before the function is called,
// the button is checked that it is active and the function
// is checked that it is not NULL.
// Input: id    button ID number of pressed button (1 to MAXBUTTONS-1)
// Output: none
void ezLCD_ButtonRelease(unsigned char id);

//------------ezLCD_Picture------------
// Send "PICTURE 0 0 3 name" command to ezLCD-301.  This
// loads the .JPG, .GIF, or .BMP image and displays it
// in the center of the screen.  Plug the ezLCD-301 into
// the USB port of a computer, load it like a removeable
// drive, and copy the image into the ezLCD-301's 4 MB
// flash memory in the directory "EZUSER\IMAGES".
// Input: name   NULL-terminated image file name
// Output: none
// Assumes: "verbose" mode off
void ezLCD_Picture(char *name);
```

Enjoy
Daniel Valvano