MAX1499 EV Kit Component List (continued)

REFERENCE	QTY	DESCRIPTION
JU10–JU14	5	3 pins
JU1–JU9	9	2 pins
R1	1	133kΩ 1% resistor (1206)
R2, R12	2	100kΩ 1% resistors (1206)
R3–R7	5	1kΩ 5% resistors (1206)
R8, R9	0	Do not install—shorted trace on PC board (1206)
R10	1	500kΩ potentiometer
R11	1	24kΩ 5% resistor (1206)
R13, R14	2	10Ω 5% resistors (1206)
TB1	1	0.200in two-circuit screw terminal block
TP1–TP4	4	8 pins
U1	1	MAX1499ECJ
U2	1	MAX1659ESA
U3, U4	2	MAX1840EUB or MAX1841EUB
U5	1	MAX6062AEUR-T, FZFY
—	1	PC board, MAX1499 EV kit
—	13	Shunts

Quick Start

Required Equipment

Before you begin, you need the following equipment:

- MAX1499EVC16 (contains MAX1499EVKIT board and 68HC16MODULE-DIP)
- DC power supply, +7VDC to +20VDC at 0.5A
- Windows 95/98/2000/XP computer with an available serial (COM) port
- 9-pin I/O extension cable

Procedure

Do not turn on the power until all connections are made.

1) Ensure that JU1–JU8 and JU10–JU14 have shunts installed, and that JU9 is open. See the jumper settings in Table 2.
2) Carefully connect the boards by aligning the 40-pin header of the MAX1499 EV kit with the 40-pin connector of the 68HC16MODULE-DIP module. Gently press them together. The two boards should be flush against one another.
3) Connect a +7VDC to +20VDC power source to the µC module at the terminal block located next to the on/off switch, along the top edge of the µC module. Observe the polarity marked on the board.
4) Connect a cable from the computer's serial port to the µC module. If using a 9-pin serial port, use a straight-through, 9-pin female-to-male cable. If the only available serial port uses a 25-pin connector, a standard 25-pin to 9-pin adapter is required. The EV kit software checks the modem status lines (CTS, DSR, and DCD) to confirm that the correct port has been selected.
5) Install the evaluation software on your computer by running the INSTALL.EXE program on the disk. The program files are copied and icons are created for them in the Windows Start menu.
6) Turn on the power supply.
7) Start the MAX1499 program by opening its icon in the Start menu.
8) The program will prompt you to connect the µC module and to turn its power on. Slide SW1 to the ON position. Select the correct serial port, and click OK. The program automatically downloads its software to the module.

Component Suppliers

SUPPLIER	PHONE	FAX	WEBSITE
Kingbright Corporation	909-468-0500 (ext 126)	909-468-0505	www.kingbright.com
Taiyo Yuden	800-348-2496	847-925-0899	www.t-yuden.com
TDK	847-803-6100	847-390-4405	www.component.tdk.com

Note: *Indicate you are using the MAX1499 when contacting these component suppliers.*

9) Apply an input signal in the range -2V to +2V between AIN+ and AIN-. Observe the readout on the screen.

10) To view a graph of the measurements, pull down the **View** menu and click **Graph**.

Detailed Description of Software

Measurement

The **Measurement** tab of the evaluation software mimics the behavior of a digital voltmeter (DVM). The status bits are polled approximately once per second. Whenever the **Data** status bit is one, the ADC result register is read and displayed as **Analog Input Code**. The MAX1499 also displays the result on its own LED display.

The EV kit is not a complete DVM. Additional input scaling and protection circuitry might be required.

Whenever the **Measurement** tab is activated, the software offers to clear the **spi/adc** and **seg_sel** control bits to zero if they are not already clear.

Math Processing

The evaluation software implements several math functions found in physical systems. Whenever the **Math** tab is activated, the software offers to set the **spi/adc** control bit to one if it is not already set. The software also offers to clear the **seg_sel** control bit to zero if it is not already clear.

The evaluation software intercepts the ADC result prior to display, calculating a new LED display value whenever the **Measurement** or **Math** tab is active and the **spi/adc** control bit is set to one. Math results are graphed as channel one data, alongside the raw ADC result as channel zero data.

The **Type K Thermocouple** function can be used along with a suitable cold junction connection to convert a type K thermocouple's measured Seebeck voltage into temperature in degrees centigrade. The **a0** coefficient 230 represents a cold junction temperature of +23°C.

Control Register

The **Control Register** tab provides access to all control register bits. Drop down the appropriate combo-box and then click **write**.

Limit Registers, ADC Offset, ADC Result, LED Display, and Peak

The **Results, Displays, Limits** tab provides access to the two's-complement data registers. Each register has a **read** button and a **write** button, except for **ADC RESULT1, ADC RESULT2**, and **PEAK RESULT**, which are read only.

Reading the **ADC RESULT1** or **ADC RESULT2** register automatically updates the LED display, regardless of the **seg_sel** control register setting.

Writing to the ADC OFFSET register affects **ADC RESULT1** and **ADC RESULT2**, regardless of the **offset_cal1** control register setting.

LED Segment Registers

The **LED Segments** tab lets the user turn individual LED segments on and off by clicking them with the mouse.

Whenever the **LED Segments** tab is activated, the software offers to set the **seg_sel** control bit to one if it is not already set.

The **Write LED Text** button translates a text string into approximate seven-segment characters, and then writes the character patterns to the LED display.

Graph

The evaluation software has two options for graphing data. A graph of recent data can be displayed by selecting the **View** menu and then **Graph**. Data can be viewed as a time sequence plot, a histogram plot, or as a table of raw numbers. To control the size and timing of the data runs, activate the sampling tool by clicking the main window's **Collect Samples** button.

Sampled data can be saved to a file in comma-delimited or tab-delimited format. Line numbers and a descriptive header line are optional.

Channel zero plots raw 16-bit ADC result data. Channel one plots LED display data, if math processing is enabled. If extended resolution is enabled, channel two plots raw 20-bit ADC result data.

Diagnostics Window

The diagnostics window is used for factory testing prior to shipping the evaluation kit. It is not intended for customer use.

Detailed Description of Hardware

The MAX1499 device under test (U1) is a low-power, 4.5-digit ADC with integrated LED display drivers. The MAX6062 (U5) provides on-board 2.048V reference voltage. See the MAX1499 EV kit schematic in Figure 7 and refer to the MAX1499 data sheet.

The EV kit includes a MAX1659 high-current 5V linear regulator (U2) and a set of MAX1840/MAX1841 level shifters (U3 and U4) to support the use of 3V logic.

Table 1. Graph Tool Buttons

TOOL	FUNCTION
	Show the entire available input range.
	Expand the graph data to fill the window.
	Move the view left or right.
	Move the view up or down.
	Expand or contract the x-axis.
	Expand or contract the y-axis.
	Load data from a file.
	Save data to a file.
	Option to write a header line when saving data.
	Option to write line numbers when saving data.
	View code vs. time plot.
	View histogram plot (cumulative frequency of each code).
	View table.
Min	Show minimum in tabular view.
Max	Show maximum in tabular view.
Span	Show span in tabular view. Span = maximum - minimum.
N	Show number of samples in tabular view.

TOOL	FUNCTION
Sum(x)	Show sum of the samples in tabular view.
Sum(x*x)	Show sum of the squares of the samples in tabular view.
Mean	Show arithmetic mean in tabular view: $\text{Mean} = \frac{\Sigma(x)}{n}$
StdDev	Show standard deviation in tabular view: $\text{Standard deviation} = \sqrt{\frac{n\Sigma\left(x^2\right) - \left(\Sigma x\right)^2}{(n-1)n}}$
Rms	Show root of the mean of the squares (RMS) in tabular view: $\text{RMS} = \sqrt{\frac{\Sigma\left(x^2\right)}{n}}$
0	Channel 0 enable (16-bit ADC result).
1	Channel 1 enable (math result).
2	Channel 2 enable (20-bit ADC result).

Table 2. Jumper Functions Table

JUMPER	SHUNT POSITION	FUNCTION
JU1	Closed*	LED displays are powered by U2.
	Open	VLED must be supplied by an external power source.
JU2	Closed*	VDISP = GND.
	Open	Apply VDISP voltage at VDISP pad.
JU3	Closed*	Banana jack AIN+ connects to AIN+ input pin.
	Open	Insert custom filtering between JU3 pins 1 and 2.
JU4	Closed*	Banana jack AIN- connects to AIN- input pin.
	Open	Insert custom filtering between JU4 pins 1 and 2.
JU5	Closed*	REF- = GND.
	Open	REF- must be provided by user.
JU6	Closed*	REF+ = 2.048V from U5, MAX6062.
	Open	REF+ must be provided by user.
JU7	Closed*	REF+ is bypassed by C6.
	Open	C6 is disconnected.
JU8	Closed*	GLED return current flows to DV_{DD}.
	Open	GLED return current flows to external power source.
JU9	Closed	LED_EN = low; LED displays are blanked.
	Open*	LED_EN = high; LED displays are enabled.
JU10–JU14	1-2*	Display color = red.
	2-3	Display color = green.

**Asterisk indicates default configuration.*

Table 3. Stand-Alone Interface Pin Functions

U1 PIN	MAX1499 FUNCTION	MAX1498 FUNCTION
8	CLK	INTREF
9	$\overline{EOC}$	RANGE
10	$\overline{CS}$	DPSET1
11	DIN	DPSET2
12	SCLK	PEAK
13	DOUT	HOLD
30	LOWBATT	DPON

Evaluating the MAX1498

The MAX1499EVKIT supports stand-alone operation of the MAX1498; however, the evaluation software cannot be used because there is no µC interface.

The MAX1498 is the standalone version of the MAX1499. Refer to MAX1498/MAX1499 data sheet. Request a free sample of MAX1498ECJ.

1) The MAX1499EVKIT must be disconnected from the 68HC16MODULE.
2) With power disconnected, replace U1 with the MAX1498.
3) Connect DC power supply at terminal block TB1.
4) Turn on the power supply. The LED display should begin indicating measurement data.

After replacing U1 with the MAX1498, some of the pin functions are different. See Table 3.

Example Code

Listings 1 shows the variable declarations needed in the EV kit software. Listing 2 contains the functions used in the EV kit software.

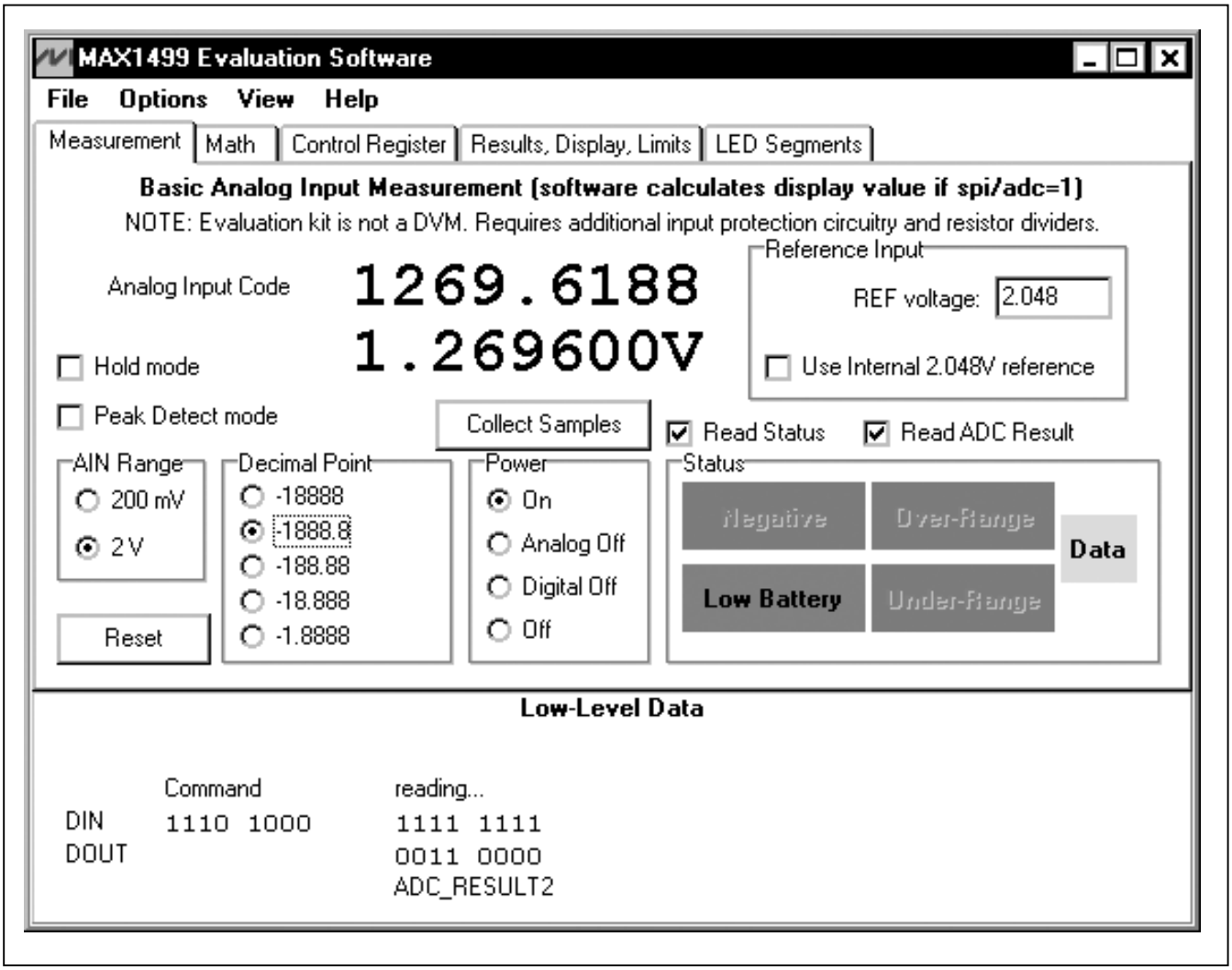

Figure 1. MAX1499 Evaluation Software—Measurement Tab

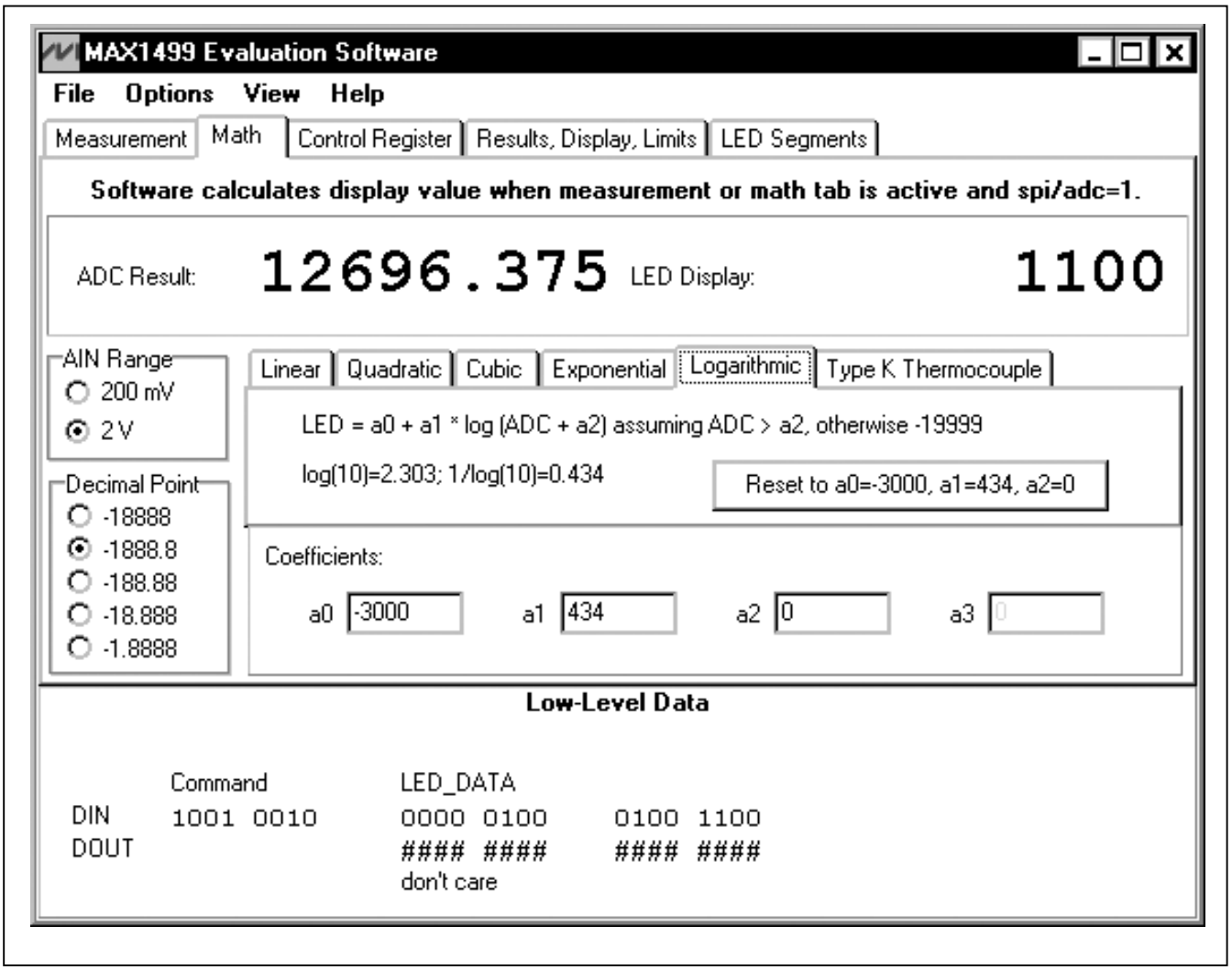

Figure 2. MAX1499 Evaluation Software—Math Tab

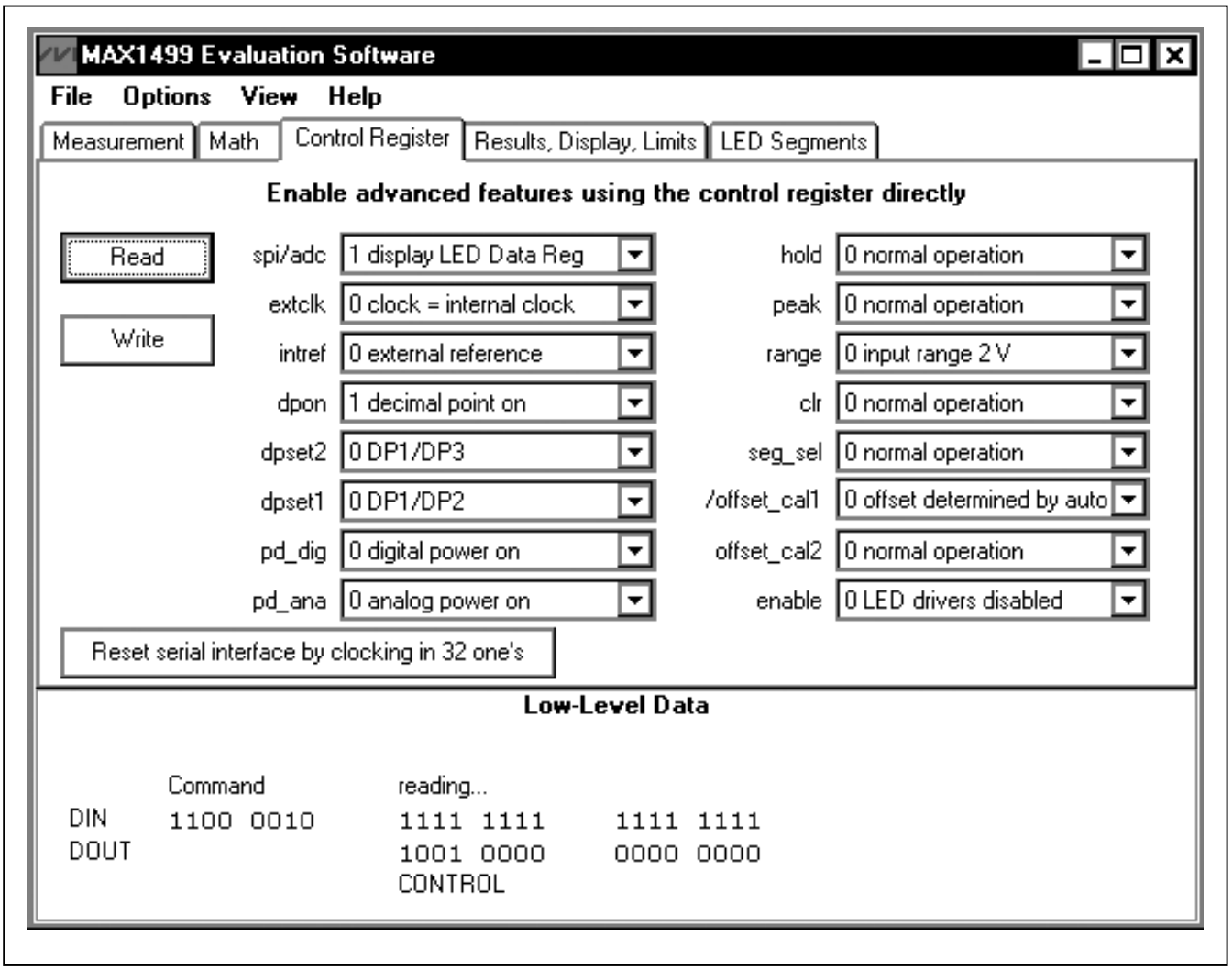

Figure 3. MAX1499 Evaluation Software—Control Register Tab

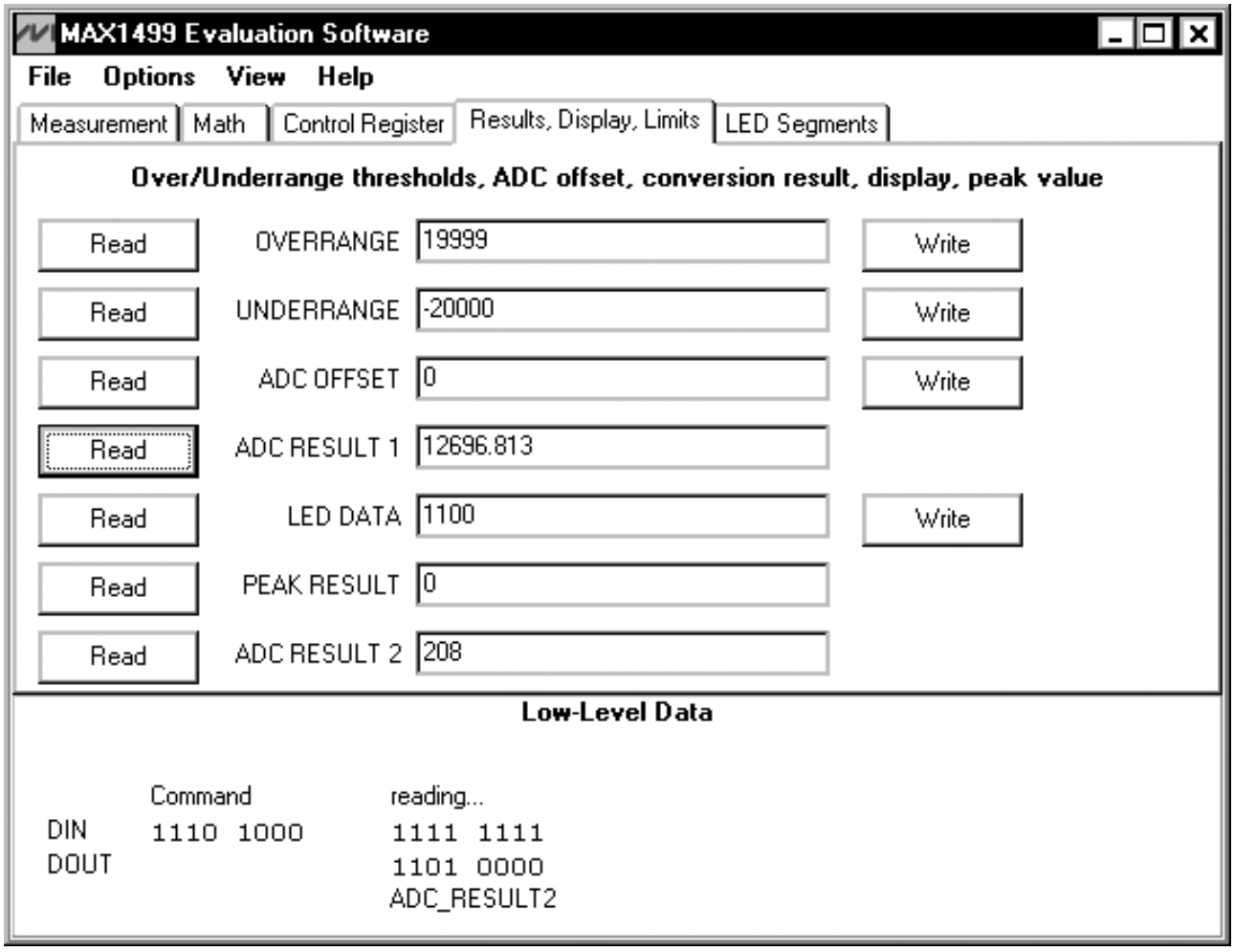

Figure 4. MAX1499 Evaluation Software—Results, Display, Limits Tab

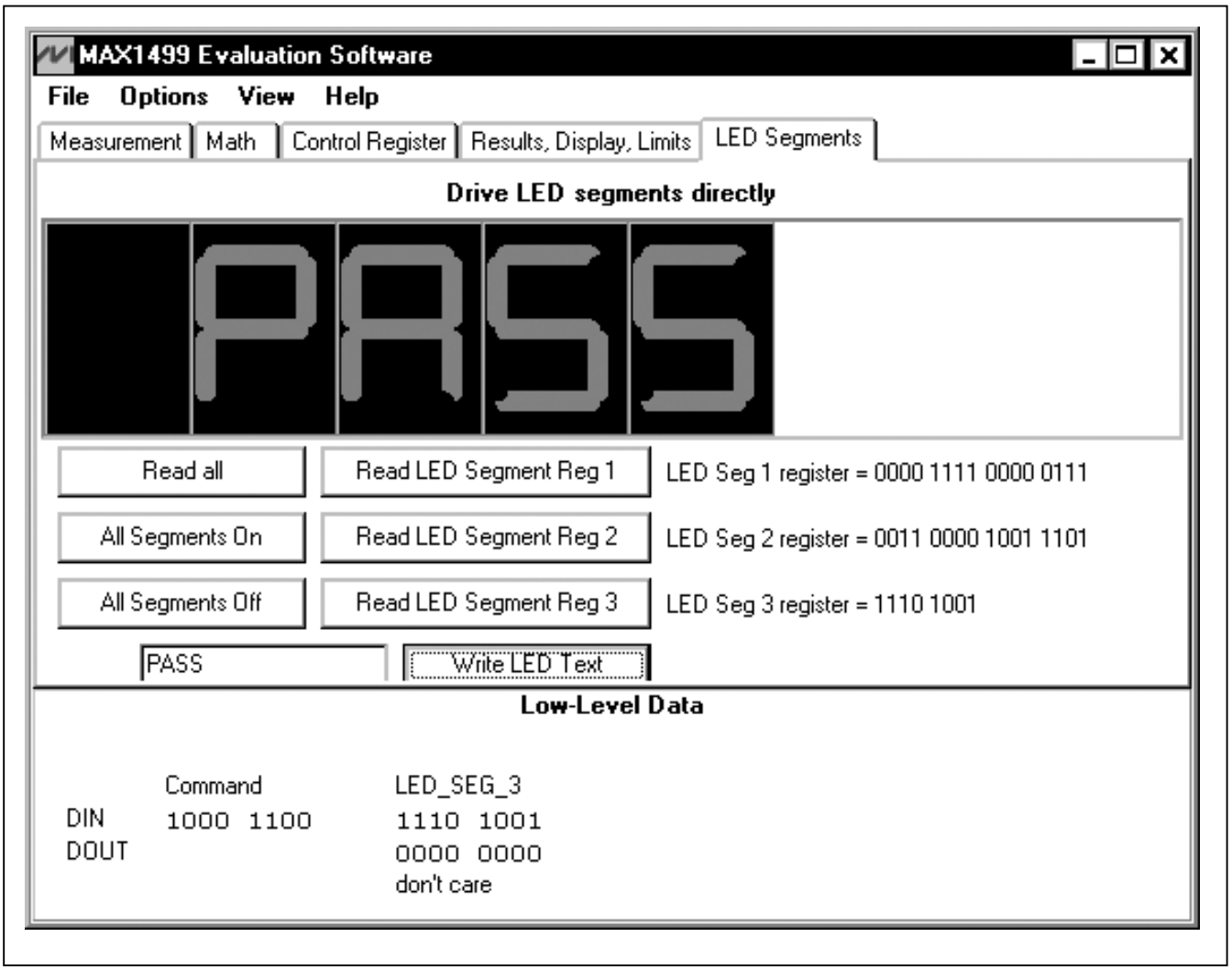

Figure 5. MAX1499 Evaluation Software—LED Segments Tab

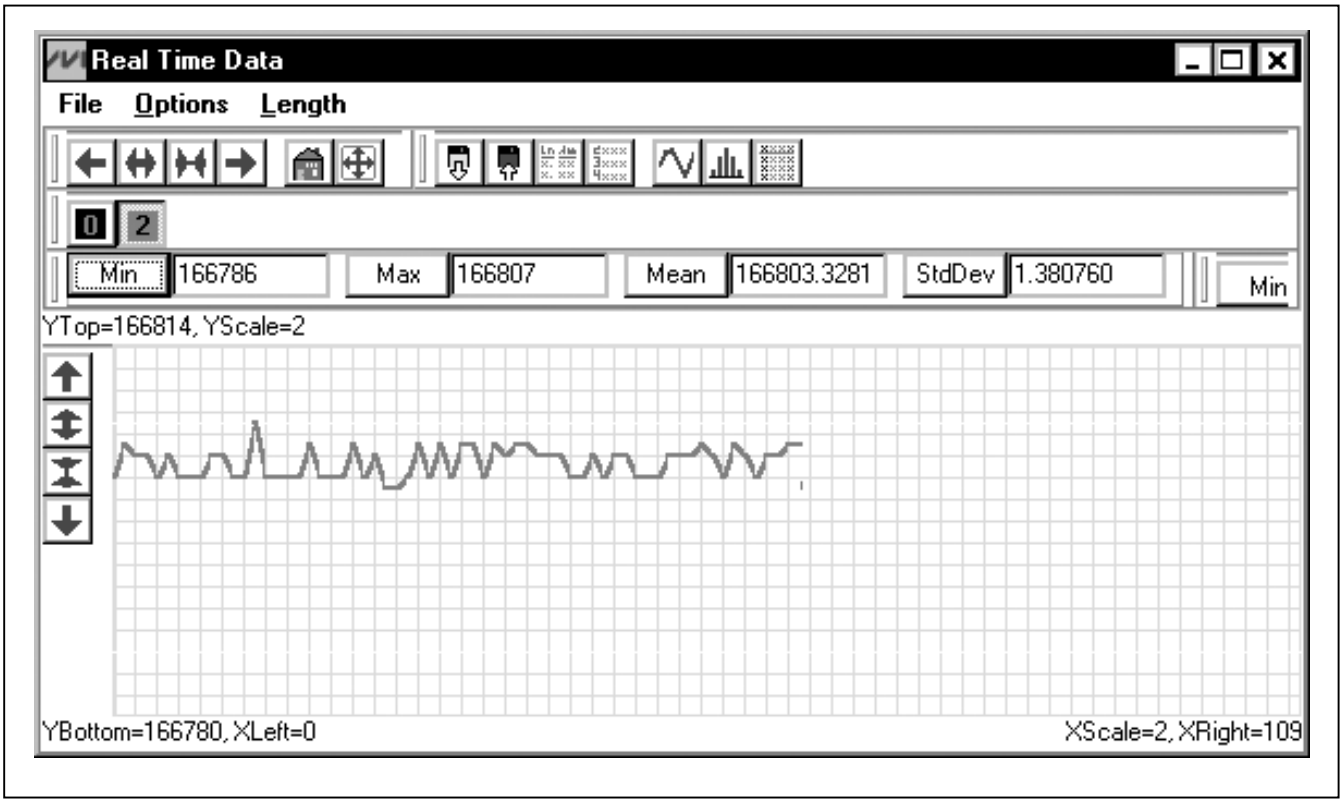

Figure 6. MAX1499 Evaluation Software—Graph

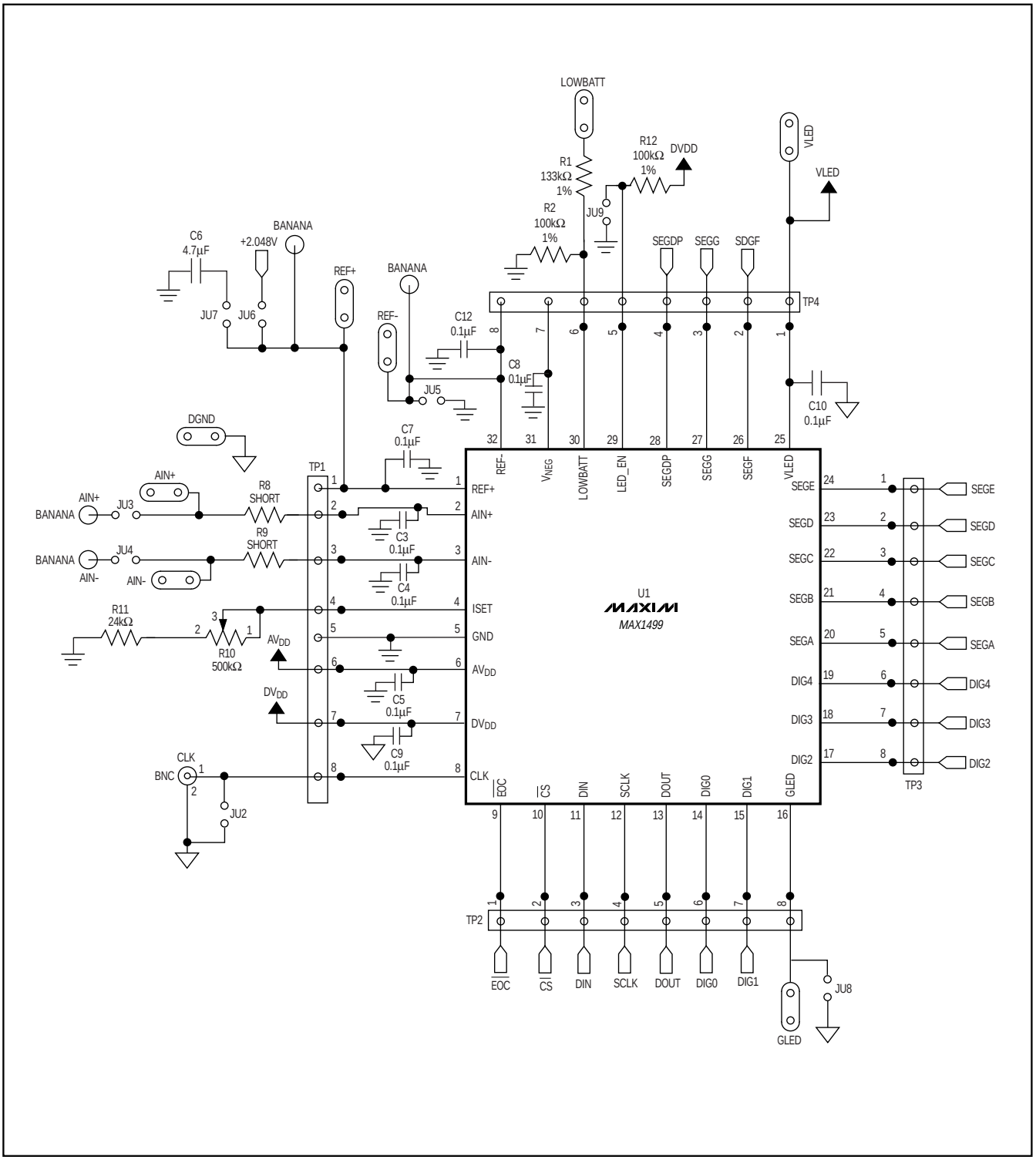

Figure 7a. MAX1499 EV Kit Schematic (Sheet 1 of 2)

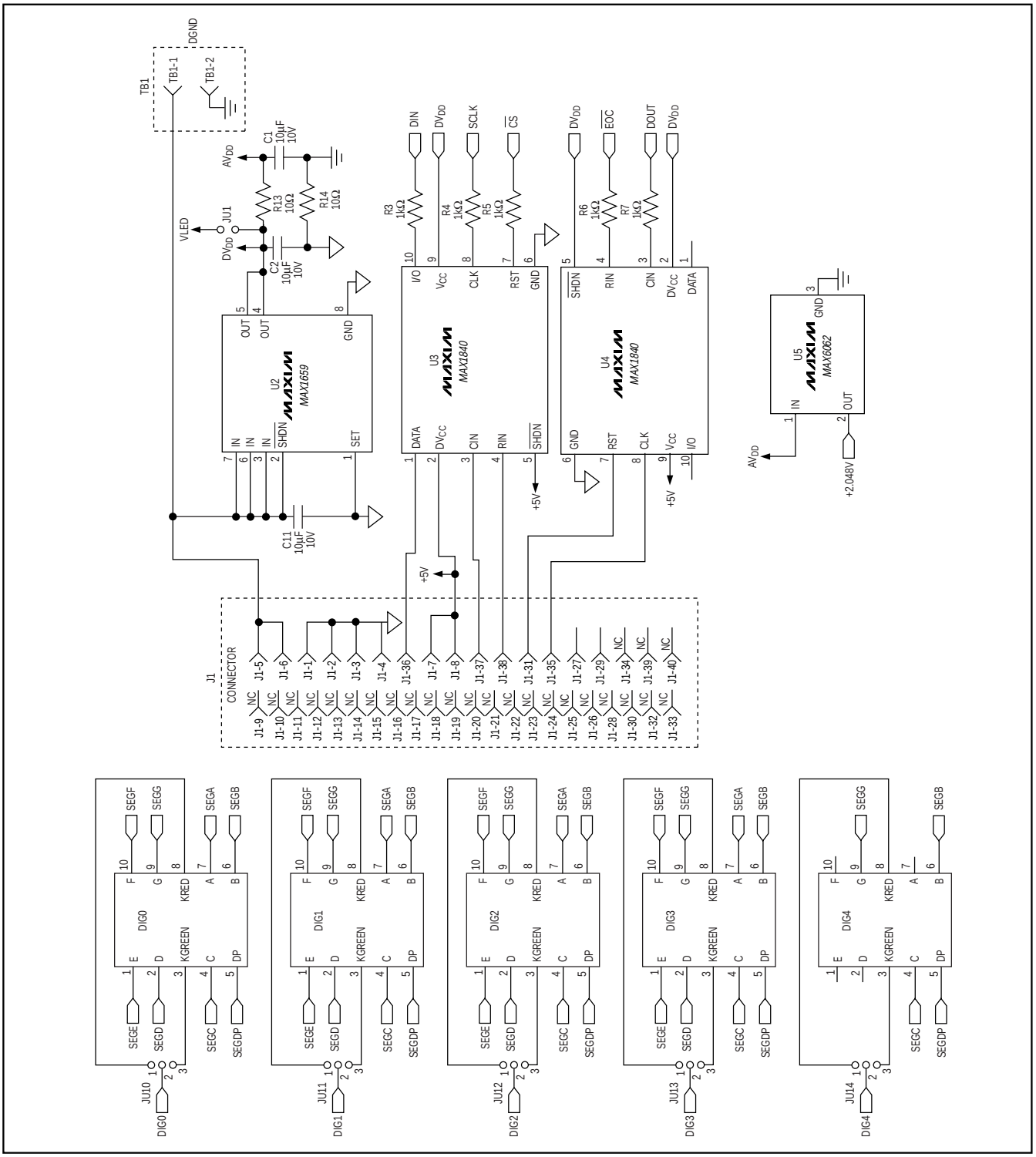

Figure 7b. MAX1499 EV Kit Schematic (Sheet 2 of 2)

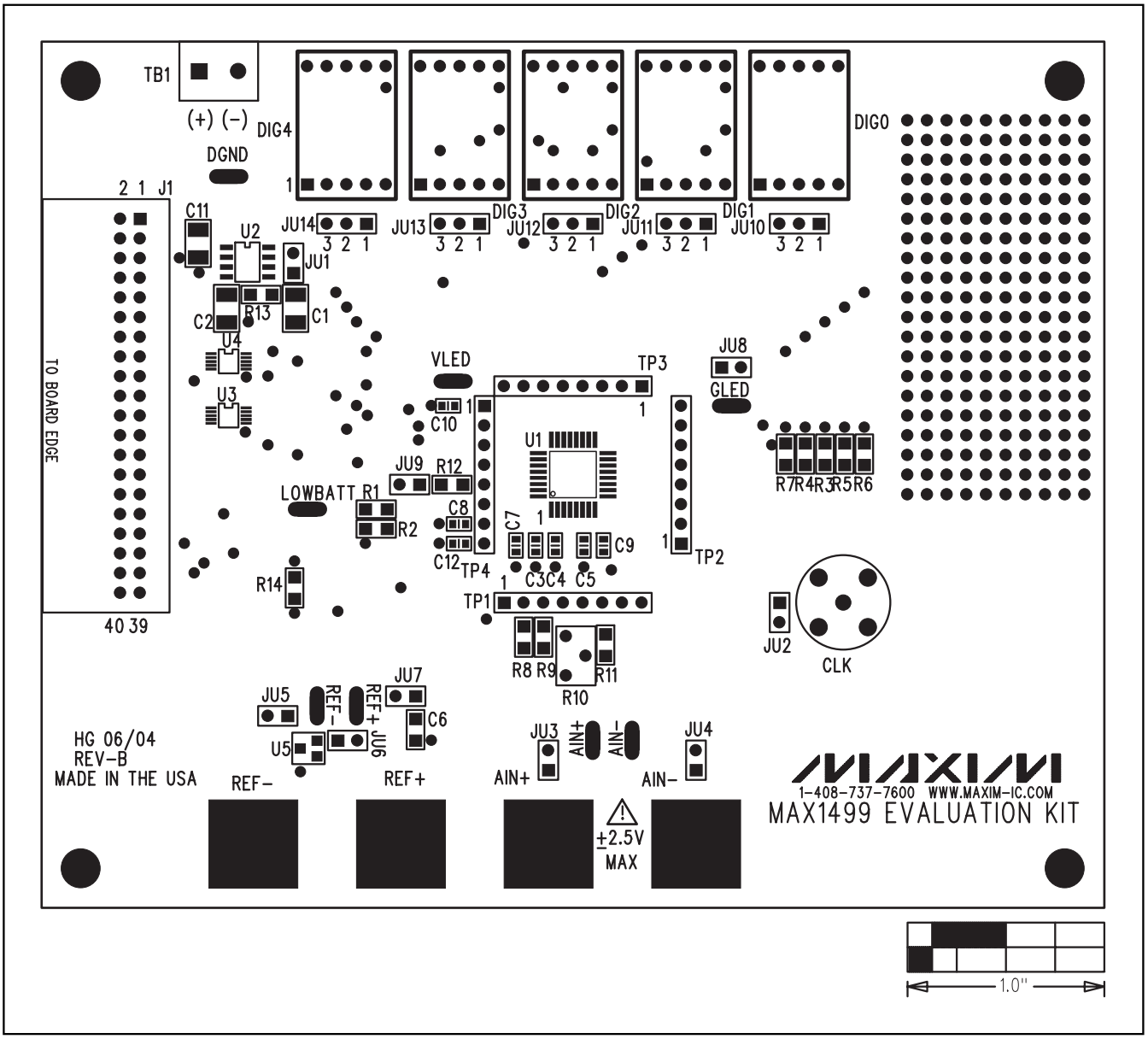

Figure 8. MAX1499 EV Kit Component Placement Guide—Component Side

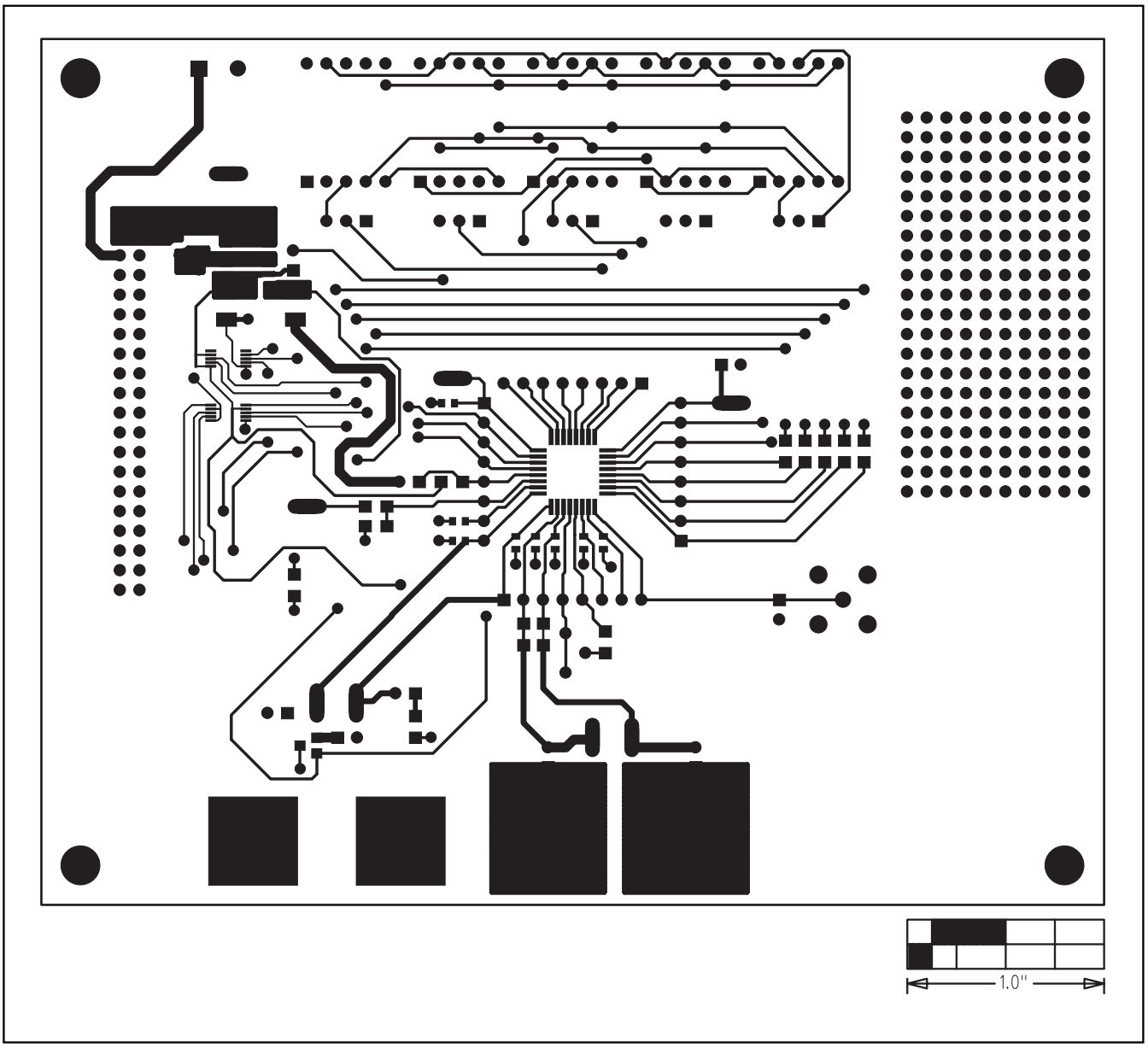

Figure 9. MAX1499 EV Kit PC Board Layout—Component Side

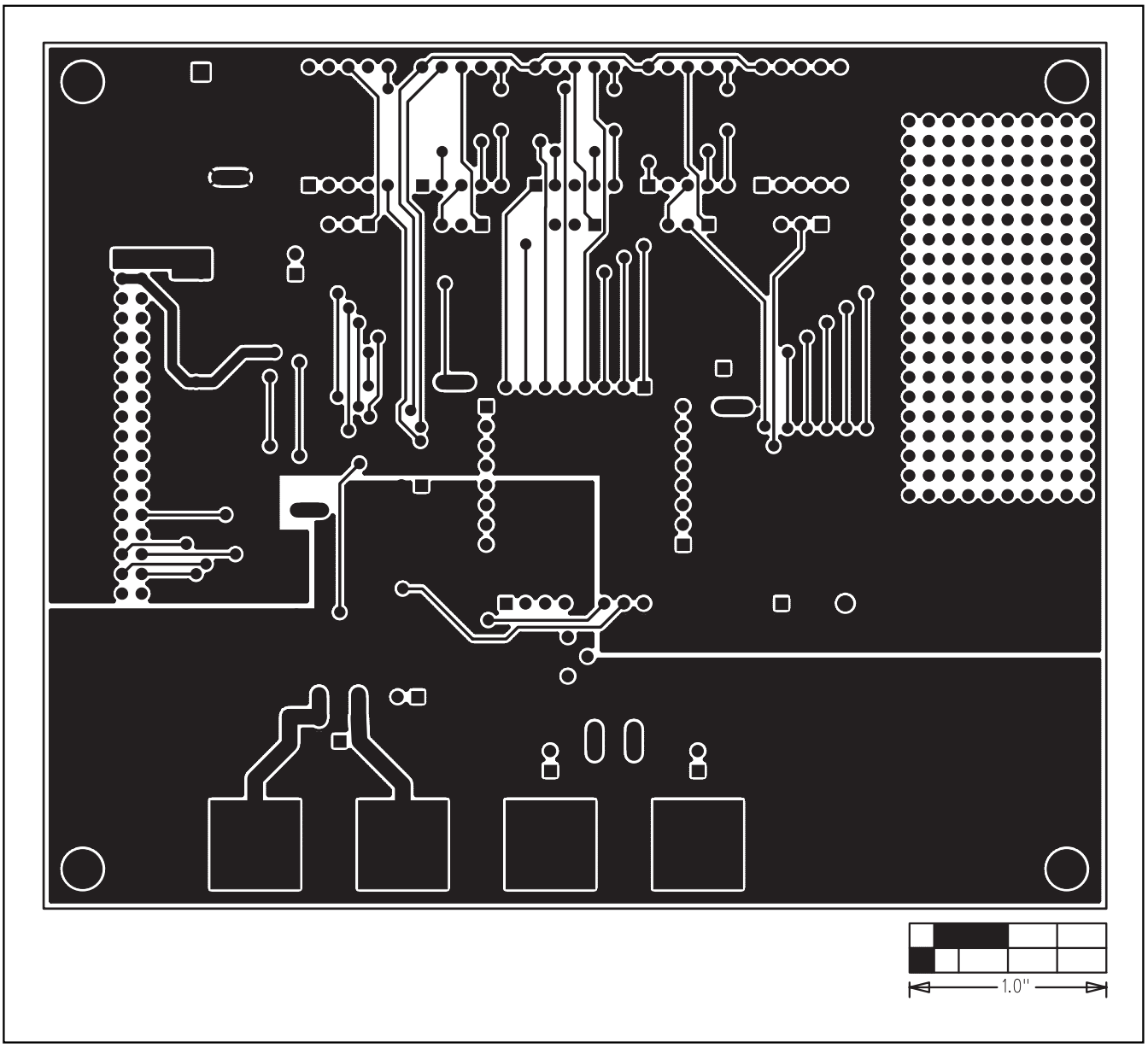

Figure 10. MAX1499 EV Kit PC Board Layout—Solder Side

```
// Drv1499.h
// MAX1499-specific driver.
// mku 01/07/2004
// (C) 2004 Maxim Integrated Products
// For use with Borland C++ Builder 3.0
//---------------------------------------------------------------------------
// Revision history:
// 01/07/2004: modify drv 1 4 9 4 driver to become drv1499
// 12/04/2003: fix indentation
// 09/15/2003: add double Voltage(void)
// 09/12/2003: add SPI_Transfer_After_EOC()
// 09/09/2003: add class MAX1499 dependent on external SPI_Interface()
// 08/13/2003: preliminary draft of reusable code
//---------------------------------------------------------------------------
#ifndef drv1499H
#define drv1499H
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// The following interface protocols must be provided by
// the appropriate low-level interface code.
//

/* SPI interface:
**       byte_count = transfer length
**       mosi[] = array of master-out, slave-in data bytes
**       miso_buf[] = receive buffer for master-in, slave-out data bytes
*/
extern bool SPI_Transfer(int byte_count,
    const unsigned __int8 mosi[], unsigned __int8 miso_buf[]);

/* SPI interface, with data transfer immediately after EOC is asserted:
**       byte_count = transfer length
**       mosi[] = array of master-out, slave-in data bytes
**       miso_buf[] = receive buffer for master-in, slave-out data bytes
*/
extern bool SPI_Transfer_After_EOC(int byte_count,
    const unsigned __int8 mosi[], unsigned __int8 miso_buf[]);

//---------------------------------------------------------------------------
// Define the bits in the COMMS register.
// START R/W RS4 RS3 RS2 RS1 RS0 0
#define MAX1499_COMMS_START          0x80
#define MAX1499_COMMS_RW_MASK        0x40
#define MAX1499_COMMS_RW_WRITE           0x00
#define MAX1499_COMMS_RW_READ            0x40
#define MAX1499_COMMS_RS_MASK        0x3E
#define MAX1499_COMMS_RS_00000           0x00
#define MAX1499_COMMS_RS_STATUS          0x00
#define MAX1499_COMMS_RS_00001           0x02
#define MAX1499_COMMS_RS_CONTROL         0x02
#define MAX1499_COMMS_RS_00010           0x04
#define MAX1499_COMMS_RS_OVERRANGE       0x04
#define MAX1499_COMMS_RS_00011           0x06
#define MAX1499_COMMS_RS_UNDERRANGE      0x06
#define MAX1499_COMMS_RS_00100           0x08
#define MAX1499_COMMS_RS_LED_SEG_1       0x08
#define MAX1499_COMMS_RS_00101           0x0A
#define MAX1499_COMMS_RS_LED_SEG_2       0x0A
#define MAX1499_COMMS_RS_00110           0x0C
#define MAX1499_COMMS_RS_LED_SEG_3       0x0C
#define MAX1499_COMMS_RS_00111           0x0E
#define MAX1499_COMMS_RS_ADC_OFFSET      0x0E
#define MAX1499_COMMS_RS_01000           0x10
#define MAX1499_COMMS_RS_ADC_RESULT1     0x10
#define MAX1499_COMMS_RS_01001           0x12
#define MAX1499_COMMS_RS_LED_DATA        0x12
#define MAX1499_COMMS_RS_01010           0x14
#define MAX1499_COMMS_RS_PEAK            0x14
#define MAX1499_COMMS_RS_10100           0x28
#define MAX1499_COMMS_RS_ADC_RESULT2     0x28

//---------------------------------------------------------------------------
// Define the bits in the STATUS register.
// POL OVR_RNG UNDR_RNG LOW_BATT ADD(data available) 0 0 0
```

Listing 1 (Sheet 1 of 4)

```
#define MAX1499_STATUS_POL_MASK          0x80
#define MAX1499_STATUS_POL_POSITIVE          0x00
#define MAX1499_STATUS_POL_NEGATIVE          0x80
#define MAX1499_STATUS_OVER_RANGE        0x40
#define MAX1499_STATUS_UNDER_RANGE       0x20
#define MAX1499_STATUS_LOW_BATTERY       0x10
#define MAX1499_STATUS_DATA_READY        0x08

//-------------------------------------------------------------------------
// Define the bits in the CONTROL register.
// SPI_ADC  EXTCLK  INTREF DPON  DPSET2  DPSET1    PD_DIG    PD_ANA
// HOLD    PEAK   RANGE  CLR  LED   OFFSET_CAL1  OFFSET_CAL2  ENABLE
#define MAX1499_CONTROL_SPI_ADC          0x8000
#define MAX1499_CONTROL_EXTCLK           0x4000
#define MAX1499_CONTROL_INTREF           0x2000
#define MAX1499_CONTROL_DPMASK           0x1C00
#define MAX1499_CONTROL_DPON             0x1000
#define MAX1499_CONTROL_DPSET2           0x0800
#define MAX1499_CONTROL_DPSET1           0x0400
// (DPSET2 is the LSB and DPSET1 is the MSB)
#define MAX1499_CONTROL_DP1ON            0x1000 /* -1888.8 */
#define MAX1499_CONTROL_DP2ON            0x1800 /* -188.88 */
#define MAX1499_CONTROL_DP3ON            0x1400 /* -18.888 */
#define MAX1499_CONTROL_DP4ON            0x1C00 /* -1.8888 */
#define MAX1499_CONTROL_PD_DIG           0x0200
#define MAX1499_CONTROL_PD_ANA           0x0100
#define MAX1499_CONTROL_PD_ALL           0x0300
#define MAX1499_CONTROL_HOLD             0x0080
#define MAX1499_CONTROL_PEAK             0x0040
#define MAX1499_CONTROL_RANGE_200mV      0x0020
#define MAX1499_CONTROL_CLR              0x0010
#define MAX1499_CONTROL_SEG_SEL          0x0008
#define MAX1499_CONTROL_OFFSET_CAL1      0x0004
#define MAX1499_CONTROL_OFFSET_CAL2      0x0002
#define MAX1499_CONTROL_ENABLE           0x0001

//-------------------------------------------------------------------------
// Define the bits in the LED SEGMENT 1 register.
// A2 G2 D2 F2 E2 DP2 0 B1
// C1 A1 G1 D1 F1 E1 DP1 0
//
#define MAX1499_LED_SEG1_A1                0x8000
#define MAX1499_LED_SEG1_G1                0x4000
#define MAX1499_LED_SEG1_D1                0x2000
#define MAX1499_LED_SEG1_F1                0x1000
#define MAX1499_LED_SEG1_E1                0x0800
#define MAX1499_LED_SEG1_DP2               0x0400
#define MAX1499_LED_SEG1_B0                0x0100
#define MAX1499_LED_SEG1_C0                0x0080
#define MAX1499_LED_SEG1_A0                0x0040
#define MAX1499_LED_SEG1_G0                0x0020
#define MAX1499_LED_SEG1_D0                0x0010
#define MAX1499_LED_SEG1_F0                0x0008
#define MAX1499_LED_SEG1_E0                0x0004
#define MAX1499_LED_SEG1_DP1               0x0002

//-------------------------------------------------------------------------
// Define the bits in the LED SEGMENT 2 register.
// F4 E4 DP4 G4 B3 C3 A3 G3
// D3 F3 E3 DP3 0 B2 C2 0
//
#define MAX1499_LED_SEG2_F3           0x8000
#define MAX1499_LED_SEG2_E3           0x4000
#define MAX1499_LED_SEG2_DP4          0x2000
#define MAX1499_LED_SEG2_MINUS          0x1000  /* TODO: is this documented? minus sign
*/
#define MAX1499_LED_SEG2_B2           0x0800
#define MAX1499_LED_SEG2_C2           0x0400
#define MAX1499_LED_SEG2_A2           0x0200
#define MAX1499_LED_SEG2_G2           0x0100
#define MAX1499_LED_SEG2_D2           0x0080
#define MAX1499_LED_SEG2_F2           0x0040
#define MAX1499_LED_SEG2_E2           0x0020
```

Listing 1 (Sheet 2 of 4)

```
#define MAX1499_LED_SEG2_DP3          0x0010
#define MAX1499_LED_SEG2_B1           0x0004
#define MAX1499_LED_SEG2_C1           0x0002

//-------------------------------------------------------------------------
// Define the bits in the LED SEGMENT 3 register.
// 0 0 BC5 B4 C4 A4 G4 D4
//
#define MAX1499_LED_SEG3_BC_          0x20
#define MAX1499_LED_SEG3_B3           0x10
#define MAX1499_LED_SEG3_C3           0x08
#define MAX1499_LED_SEG3_A3           0x04
#define MAX1499_LED_SEG3_G3           0x02
#define MAX1499_LED_SEG3_D3           0x01

//-------------------------------------------------------------------------
class MAX1499
{
public:
     MAX1499(void);

     // Enumerated type describing the register select bits.
     enum RegisterSelect_t {
          RS_STATUS        = MAX1499_COMMS_RS_STATUS,
          RS_CONTROL       = MAX1499_COMMS_RS_CONTROL,
          RS_OVERRANGE     = MAX1499_COMMS_RS_OVERRANGE,
          RS_UNDERRANGE    = MAX1499_COMMS_RS_UNDERRANGE,
          RS_LED_SEG_1     = MAX1499_COMMS_RS_LED_SEG_1,
          RS_LED_SEG_2     = MAX1499_COMMS_RS_LED_SEG_2,
          RS_LED_SEG_3     = MAX1499_COMMS_RS_LED_SEG_3,
          RS_ADC_OFFSET    = MAX1499_COMMS_RS_ADC_OFFSET,
          RS_ADC_RESULT1   = MAX1499_COMMS_RS_ADC_RESULT1,
          RS_LED_DATA      = MAX1499_COMMS_RS_LED_DATA,
          RS_PEAK          = MAX1499_COMMS_RS_PEAK,
          RS_ADC_RESULT2   = MAX1499_COMMS_RS_ADC_RESULT2
     };

     // Reference voltage
     //
     double vref;

     //---------------------------------------
     // Status Register
     // POL OVR_RNG UNDR_RNG LOW_BATT ADD(data available) 0 0 0
     int STATUS_REG;
     //
     bool Read_STATUS(void);

     //---------------------------------------
     // Control Register
     // SPI_ADC  EXTCLK  INTREF DPON  DPSET2  DPSET1    PD_DIG     PD_ANA
     // HOLD     PEAK    RANGE  CLR  LED    OFFSET_CAL1  OFFSET_CAL2  ENABLE
     int CONTROL_REG;
     //
     bool Write_CONTROL(int data);
     bool Read_CONTROL(void);

     //---------------------------------------
     // Data Registers
     int ADC_RESULT1;
     unsigned int ADC_RESULT2;
     //
     bool Read_ADC_RESULT1(void);
     bool Read_ADC_RESULT2(void);
     long int DATA_REG;   // 16-bit or 24-bit result from A/D converter
     bool extended_resolution;
     long Read_DATA(void);
     double Voltage(void);

     //---------------------------------------
     // Other registers, having 16-bit 2's complement data format
     bool Write_2s_complement(int reg, int data);
     int Read_2s_complement(int reg);
```

Listing 1 (Sheet 3 of 4)

```
        //---------------------------------------
        // Other registers, having 8 bit data format
        bool Write_8bit_reg(int reg, int data);
        int Read_8bit_reg(int reg);

};

//---------------------------------------------------------------------------
#endif
```

Listing 1 (Sheet 4 of 4)

```
// Drv1499.cpp
// MAX1499-specific driver.
// mku 09/15/2003
// (C) 2003 Maxim Integrated Products
// For use with Borland C++ Builder 3.0
//---------------------------------------------------------------------------
// Revision history:
// 09/15/2003: add double Voltage(void)
// 09/09/2003: add class MAX1499 dependent on external SPI_Interface()
// 08/13/2003: preliminary draft of reuseable code

#include "drv1499.h"

//---------------------------------------------------------------------------
MAX1499::MAX1499(void)
{
    vref = 2.048;
    extended_resolution = false;
}
//---------------------------------------------------------------------------
bool MAX1499::Read_STATUS(void)
{
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1499_COMMS_START |
            MAX1499_COMMS_RW_READ | MAX1499_COMMS_RS_STATUS),
        (unsigned __int8)(0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    if (result) {
        int data = miso_buf[1];
        STATUS_REG = data;            // remember the value we just received
    }
    return result;
}
//---------------------------------------------------------------------------
bool MAX1499::Write_CONTROL(int data)
{
    data = data & 0xFFFF;        // validate the data
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1499_COMMS_START |
            MAX1499_COMMS_RW_WRITE | MAX1499_COMMS_RS_CONTROL),
        (unsigned __int8)( (data >> 8) & 0xFF),
        (unsigned __int8)( data & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    CONTROL_REG = data;             // remember the value we just wrote
    // The CLR bit is self-clearing, and should not be kept high.
    CONTROL_REG &=~ MAX1499_CONTROL_CLR;
    return result;
}
//---------------------------------------------------------------------------
bool MAX1499::Read_CONTROL(void)
{
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1499_COMMS_START |
            MAX1499_COMMS_RW_READ | MAX1499_COMMS_RS_CONTROL),
        (unsigned __int8)(0xFF),
        (unsigned __int8)(0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    if (result) {
        int data = miso_buf[1] * 0x100 + miso_buf[2];
        CONTROL_REG = data;            // remember the value we just wrote
    }
    return result;
}
//---------------------------------------------------------------------------
bool MAX1499::Read_ADC_RESULT1(void)
{
    const unsigned __int8 mosi[] = {
```

Listing 2 (Sheet 1 of 4)

```
            (unsigned __int8)(MAX1499_COMMS_START |
                MAX1499_COMMS_RW_READ | MAX1499_COMMS_RS_ADC_RESULT1),
            (unsigned __int8)(0xFF),
            (unsigned __int8)(0xFF)
        };
        unsigned __int8 miso_buf[sizeof(mosi)];
        bool result = SPI_Transfer_After_EOC(sizeof(mosi), mosi, miso_buf);
        if (result) {
            ADC_RESULT1 = (miso_buf[1] * 0x100L) + miso_buf[2];
            long data = (miso_buf[1] * 0x100L) + miso_buf[2];
            if (data >= 32768) {
                data -= 65536;
            }
            DATA_REG = data;            // remember the value we just received
        }
        return result;
    }
    //---------------------------------------------------------------------------
    bool MAX1499::Read_ADC_RESULT2(void)
    {
        const unsigned __int8 mosi[] = {
            (unsigned __int8)(MAX1499_COMMS_START |
                MAX1499_COMMS_RW_READ | MAX1499_COMMS_RS_ADC_RESULT2),
            (unsigned __int8)(0xFF)
        };
        unsigned __int8 miso_buf[sizeof(mosi)];
        bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
        if (result) {
            ADC_RESULT2 = miso_buf[1];
            long data_24 = ((long)ADC_RESULT1 * 0x100L) + ADC_RESULT2;
            DATA_REG = data_24;
        }
        return result;
    }
    //---------------------------------------------------------------------------
    long MAX1499::Read_DATA(void)
    {
        // Read the DATA register
        const unsigned __int8 mosi[] = {
            (unsigned __int8)(MAX1499_COMMS_START |
                MAX1499_COMMS_RW_READ | MAX1499_COMMS_RS_ADC_RESULT1),
            (unsigned __int8)(0xFF),
            (unsigned __int8)(0xFF)
        };
        unsigned __int8 miso_buf[sizeof(mosi)];
        if (SPI_Transfer_After_EOC(sizeof(mosi), mosi, miso_buf) == false) {
            return 0; // failure
        }
        ADC_RESULT1 = (miso_buf[1] * 0x100L) + miso_buf[2];
        long data = (miso_buf[1] * 0x100L) + miso_buf[2];
        if (data >= 32768) {
            data -= 65536;
        }
        DATA_REG = data;            // remember the value we just received
        if (extended_resolution) {
            // Read the ADC_RESULT2 register
            const unsigned __int8 mosi[] = {
                (unsigned __int8)(MAX1499_COMMS_START |
                    MAX1499_COMMS_RW_READ | MAX1499_COMMS_RS_ADC_RESULT2),
                (unsigned __int8)(0xFF)
            };
            unsigned __int8 miso_buf[sizeof(mosi)];
            if (SPI_Transfer(sizeof(mosi), mosi, miso_buf) == false) {
                return 0; // failure
            }
            ADC_RESULT2 = miso_buf[1];
            long data_24 = ((long)ADC_RESULT1 * 0x100L) + ADC_RESULT2;
            double data_16 = data_24 / 256.0;
            if (data_16 >= 32768) {
                data_16 = data_16 - 65536;
            }
            DATA_REG = data_24;
```

Listing 2 (Sheet 2 of 4)

```
    }
    return DATA_REG;
}
//---------------------------------------------------------------------
double MAX1499::Voltage(void)
{
    if ((CONTROL_REG & MAX1499_CONTROL_RANGE_200mV) == 0) {
        // Input range 2V
        return DATA_REG * (vref / 2.048) * 10e-6 * 10;
    } else {
        // Input range 200mV
        return DATA_REG * (vref / 2.048) * 10e-6;
    }
}
//---------------------------------------------------------------------
bool MAX1499::Write_2s_complement(int reg, int data)
{
    // Write one of the 2's complement registers
    reg = (reg & MAX1499_COMMS_RS_MASK);
    data = data & 0xFFFF;        // validate the data

    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1499_COMMS_START | MAX1499_COMMS_RW_WRITE | reg),
        (unsigned __int8)((data >> 8) & 0xFF),
        (unsigned __int8)(data & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    return result;
}
//---------------------------------------------------------------------
int MAX1499::Read_2s_complement(int reg)
{
    // Read one of the 2's complement registers
    reg = (reg & MAX1499_COMMS_RS_MASK);

    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1499_COMMS_START | MAX1499_COMMS_RW_READ | reg),
        (unsigned __int8)(0xFF),
        (unsigned __int8)(0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    if (result == false) {
        return 0; // failure
    }
    int data = miso_buf[1] * 0x100 + miso_buf[2];
    if (data >= 32768) {
        data -= 65536;
    }
    if (data >= 32768) {
        data -= 65536;
    }
    return data;
}
//---------------------------------------------------------------------
bool MAX1499::Write_8bit_reg(int reg, int data)
{
    // Write one of the 8 bit registers
    reg = (reg & MAX1499_COMMS_RS_MASK);
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1499_COMMS_START | MAX1499_COMMS_RW_WRITE | reg),
        (unsigned __int8)(data & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    return result;
}
//---------------------------------------------------------------------
int MAX1499::Read_8bit_reg(int reg)
{
    // Read one of the 8 bit registers
    reg = (reg & MAX1499_COMMS_RS_MASK);
```

Listing 2 (Sheet 3 of 4)

```
        const unsigned __int8 mosi[] = {
            (unsigned __int8)(MAX1499_COMMS_START | MAX1499_COMMS_RW_READ | reg),
            (unsigned __int8)(0xFF)
        };
        unsigned __int8 miso_buf[sizeof(mosi)];
        bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
        if (result == false) {
            return 0; // failure
        }
        int data = miso_buf[1];
        return data;
    }
    //---------------------------------------------------------------------
```

Listing 2 (Sheet 4 of 4)