19-3145; Rev 0; 1/04

MAX1494 Evaluation Kit/ Evaluation System

General Description

The MAX1494 evaluation system (EV system) consists of a MAX1494 evaluation kit (EV kit) and a Maxim 68HC16MODULE-DIP microcontroller (µC) module. The MAX1494 is a low-power, 4.5-digit, analog-to-digital converter (ADC) with integrated liquid crystal display (LCD) drivers. The evaluation software runs under Windows® 95/98/2000/XP, providing a user interface to exercise the features of the MAX1494.

Order the complete EV system (MAX1494EVC16) for a comprehensive evaluation of the MAX1494 using a personal computer. Order the EV kit (MAX1494EVKIT) if the 68HC16MODULE-DIP module has been purchased with a previous Maxim EV system, or for custom use in other µC-based systems.

This system can also evaluate the MAX1493CCJ and MAX1495CCJ. Contact the factory for free samples of these products. See the *Detailed Description of Hardware* section.

MAX1494 EV Kit

The MAX1494 EV kit provides a proven PC board layout to facilitate evaluation of the MAX1494. It must be interfaced to appropriate timing signals for proper operation. Connect 6V to 26VDC and ground return to terminal block TB1 (see Figure 7). Refer to the MAX1494 data sheet for timing requirements.

MAX1494 EV System

The MAX1494 EV system operates from a user-supplied 7VDC to 20VDC power supply. The evaluation software runs under Windows 95/98/2000/XP on a PC, interfacing to the EV system board through the computer's serial communications port. See the *Quick Start* section for setup and operating instructions.

Features

- ♦ **Proven PC Board Layout**
- ♦ **Complete Evaluation System**
- ♦ **Convenient On-Board Test Points**
- ♦ **Data-Logging Software**
- ♦ **Fully Assembled and Tested**

Ordering Information

PART	TEMP RANGE	INTERFACE TYPE
MAX1494EVKIT	0°C to +70°C	User supplied
MAX1494EVC16	0°C to +70°C	Windows software

Note: *The MAX1494 evaluation software is designed for use with the complete evaluation system (MAX1494EVC16). The MAX1494EVC16 includes the 68HC16MODULE-DIP module together with the MAX1494EVKIT. If the MAX1494 evaluation software will not be used, the MAX1494EVKIT board can be purchased by itself, without the µC.*

MAX1494EV16 System Component List

PART	QTY	DESCRIPTION
MAX1494EVKIT	1	MAX1494 EV kit
68HC16MODULE-DIP	1	68HC16 µC module

MAX1494EVKIT Component List

DESIGNATION	QTY	DESCRIPTION
C1, C2	2	10µF ±20%, 10V X7R ceramic capacitors (1210) TDK C3225X7R1C106M
C3–C6	4	0.47µF ±10%, 16V X7R ceramic capacitors (0805) TDK C2012X7R1C474K
C7, C8, C9	3	0.1µF ±20%, 16V X7R ceramic capacitors (0603) TDK C1608X7R1C104K
CLK	1	BNC 50Ω PC board vertical mount A/D ELECTRONICS 580-002-00
FB1	1	Ferrite bead (0805) Murata BLM21AH102SN1
J1	1	2 x 20 right angle socket, SamTec SSW-120-02-S-D-RA Methode Electronics RS2R-40-G
JU1	1	3-pin header
JU1–JU6	6	Shunts
JU2–JU6	5	2-pin headers
LCD1	1	Triplexed liquid crystal display (LCD), ICL7129 type DCI Inc. 04-0925-00 or Varitronix VIM-503-DP-FC-S-HV

MAX1494EVKIT Component List (continued)

DESIGNATION	QTY	DESCRIPTION
LCD1 (2 rows)	2	15-pin socket strips
R1	1	133kΩ ±1% resistor (1206)
R2	1	100kΩ ±1% resistor (1206)
R3–R7	5	1kΩ ±5% resistors (1206)
TP1–TP4	4	8-pin headers
U1	1	MAX1494CCJ (32-pin TQFP)
U2	1	MAX1615EUK-T
U3, U4	2	MAX1840EUB or MAX1841EUB
U5	1	MAX6062AEUR-T
AIN+, AIN-, REF+, REF-	4	Noninsulated banana jacks Mouser 530-108-0740-1
None	1	MAX1494 EV kit PC board

Quick Start

Required Equipment

Before you begin, you will need the following equipment:

- MAX1494EVC16 (contains the MAX1494EVKIT board and the 68HC16MODULE-DIP)
- DC power supply, +7VDC to +20VDC at 0.25A
- 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 is in the 1-2 position and JU2–JU6 have shunts installed. See Table 2 (Jumper Settings).
2) Carefully connect the boards by aligning the 40-pin header of the MAX1494 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, 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 MAX1494 program by opening its icon in the Start Menu.
8) The program prompts you to connect the µC module and 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.
9) Apply an input signal in the -2V to +2V range 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 MAX1494 also displays the result on its own LCD.

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

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

Math Processing

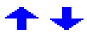
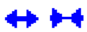
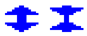
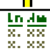
The evaluation software implements several math functions found in physical systems. Whenever the **Math** tab is activated, the software offers to set the **spi/adc**

Component Suppliers

SUPPLIER	PHONE	FAX	WEBSITE
TDK	847-803-6100	847-390-4405	www.component.tdk.com

Note: *Please indicate you are using the MAX1494 when contacting component suppliers*

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

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 LCD 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 23.0°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, LCD, 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 LCD, 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.

LCD Segment Registers

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

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

Table 2. Jumper Functions

JUMPER	SHUNT POSITION	FUNCTION
JU1	1-2*	DV_{DD} = +5V.
JU1	2-3	DV_{DD} = +3V.
JU2	Closed*	V_{DISP} = GND.
JU2	Open	Apply V_{DISP} voltage at VDISP pad.
JU3	Closed*	Banana jack AIN+ connects to AIN+ input pin.
JU3	Open	Insert custom filtering between JU3 pins 1 and 2.
JU4	Closed*	Banana jack AIN- connects to AIN- input pin.
JU4	Open	Insert custom filtering between JU4 pins 1 and 2.
JU5	Closed*	REF- = GND.
JU5	Open	REF- must be provided by user.
JU6	Closed*	REF+ = +2.048V from U5, MAX6062.
JU6	Open	REF+ must be provided by user.

**Indicates default configuration*

LCDs with only 12 segment lines (such as the VIM503) do not support **hold** or **peak** annunciators, however, the device and the kit do support **hold** or **peak** annunciators.

The **Write LCD Text** button translates a text string into approximate 7-segment characters, and then writes the character patterns to the LCD.

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 LCD 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 EV kit. It is not intended for customer use.

Detailed Description of Hardware

The MAX1494 device under test (U1) is a low-power, 4.5-digit ADC with integrated LCD drivers. The MAX6062 (U5) provides on-board +2.048V reference voltage. See Figure 7, and refer to the MAX1494 data sheet.

Table 3. Stand-Alone Interface Pin Functions

U1 PIN	MAX1494 FUNCTION	MAX1493/MAX1495 FUNCTION
7	$\overline{EOC}$	RANGE
8	$\overline{CS}$	DPSET1
9	DIN	DPSET2
10	SCLK	PEAK
11	DOUT	HOLD
28	VDISP	DPON
30	CLK	INTREF

The EV kit includes a MAX1615 +3V/+5V linear regulator (U2) and a set of MAX1840/MAX1841 level shifters (U3 and U4) to support using the +3V MAX1494 with the +5V µC.

Evaluating the MAX1493/MAX1495

The MAX1494EVKIT supports stand-alone operation of the MAX1493/MAX1495. However, the evaluation software cannot be used because there is no microprocessor interface on these stand-alone devices.

The MAX1493 is the stand-alone version of the MAX1494. The MAX1495 is similar to the MAX1493, but it has the ability to enable offset calibration on demand. Refer to the MAX1491/MAX1493/MAX1495 data sheet. Request a free sample of MAX1493CCJ or MAX1495CCJ.

1) The MAX1494EVKIT must be disconnected from the 68HC16MODULE.

2) With power disconnected, replace U1 with the MAX1493 or the MAX1495. After replacing U1 with the MAX1493 or MAX1495, some of the pin functions are different. See Table 3.
3) Ensure that jumper JU1 selects the +3V or +5V logic level, as desired.
4) Connect DC power supply at terminal block TB1.
5) Turn on the power supply. The LCD should begin indicating measurement data.

Troubleshooting

Problem: Peak detect mode does not work below 19,487 counts.

This is a limitation of the MAX1494. Refer to the MAX1494 data sheet.

Problem: Startup delay

The MAX1494 requires approximately 2 seconds to initiate at power-up.

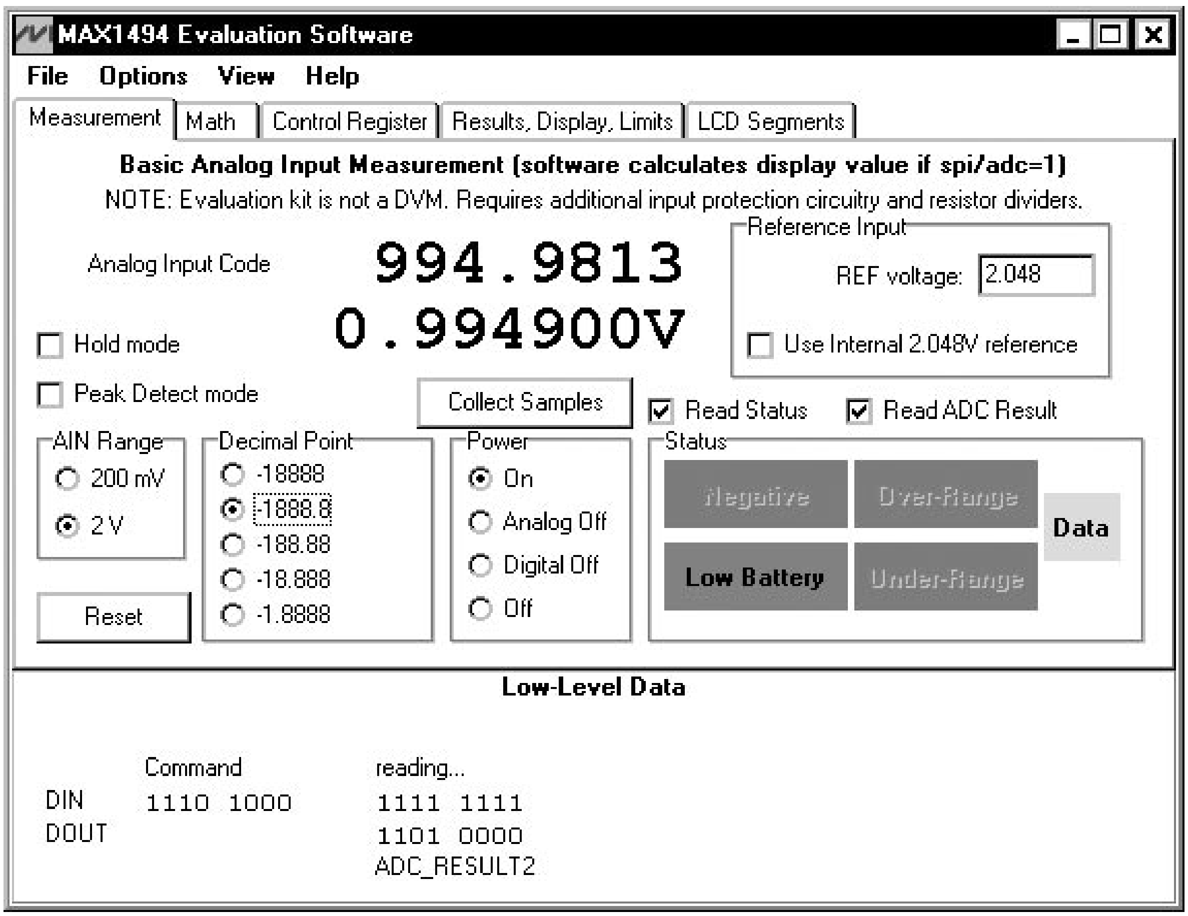


Figure 1. Main Window—MAX1494 Evaluation Software

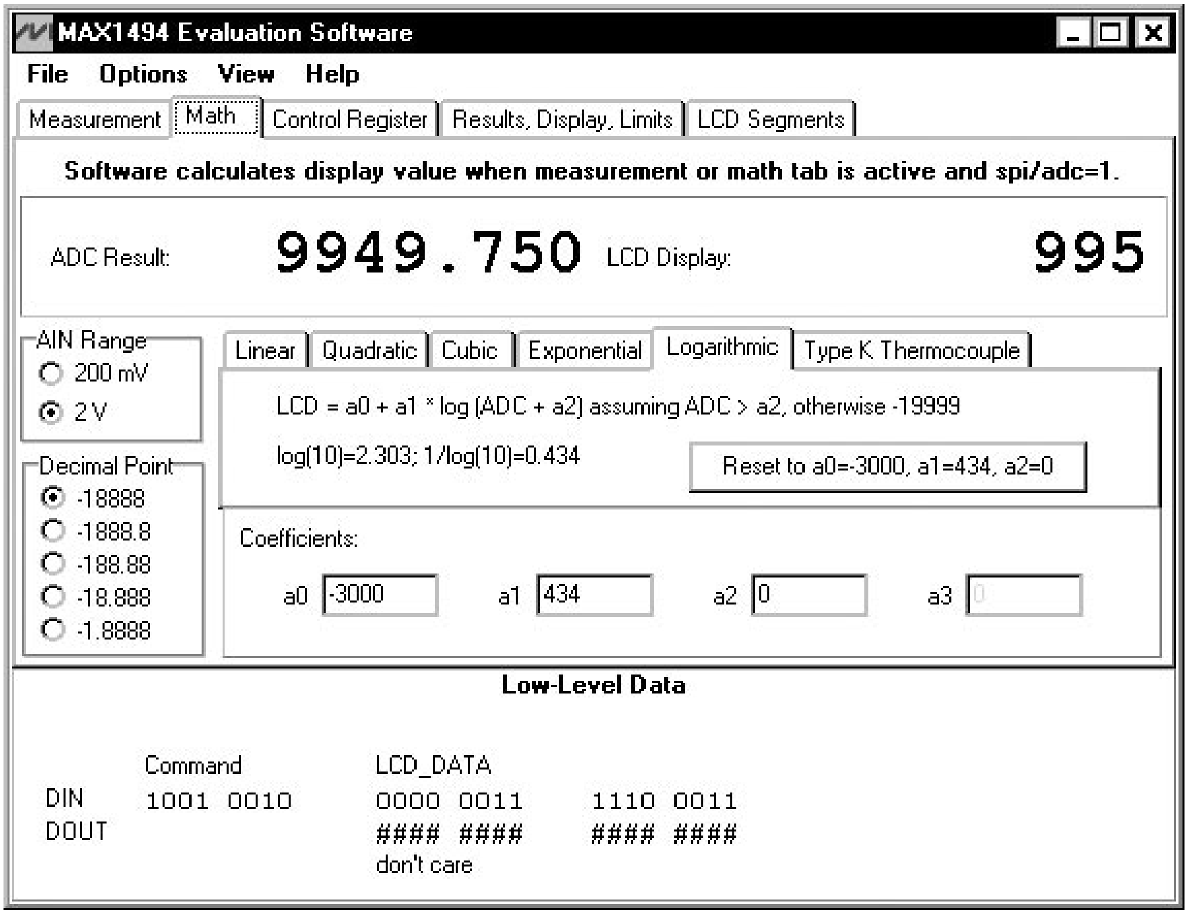


Figure 2. Math Tab—MAX1494 Evaluation Software

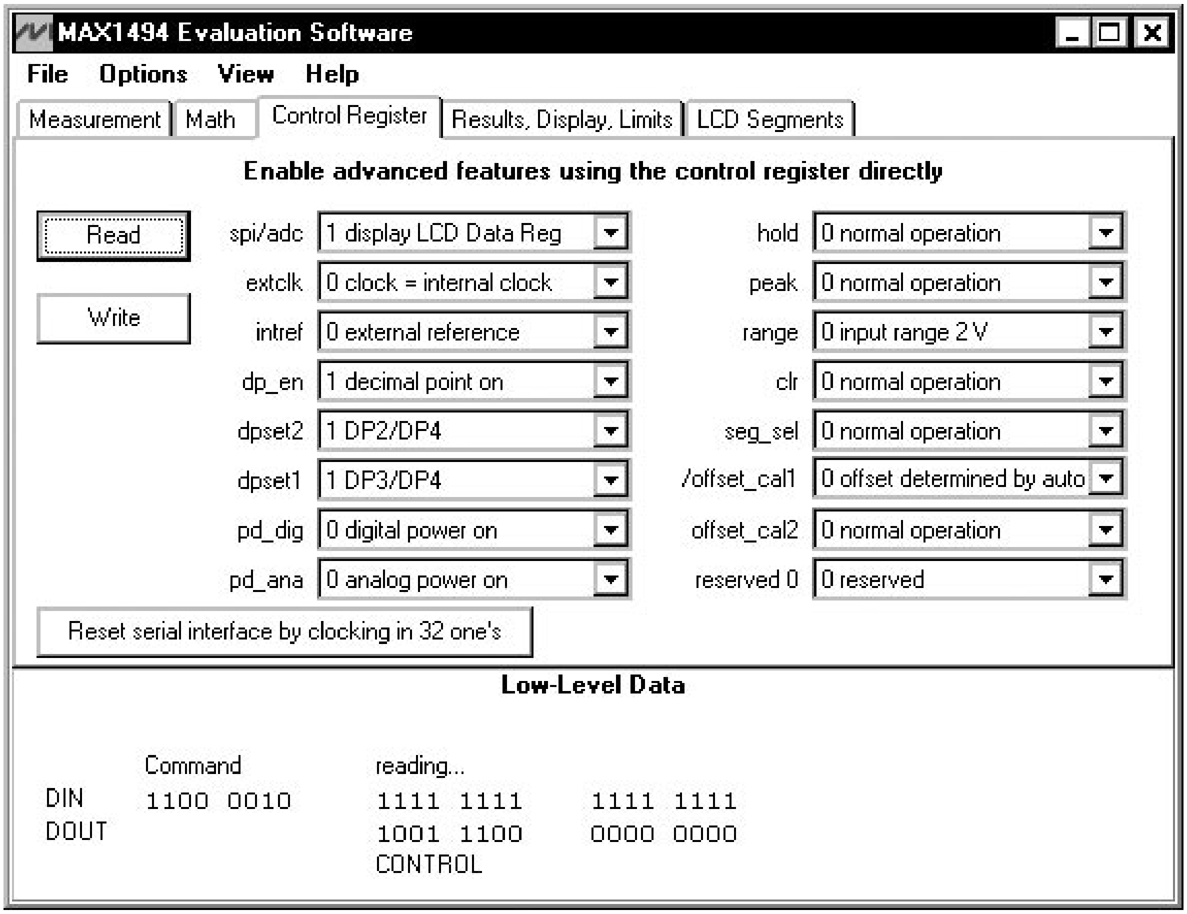


Figure 3. Control Register Tab—MAX1494 Evaluation Software

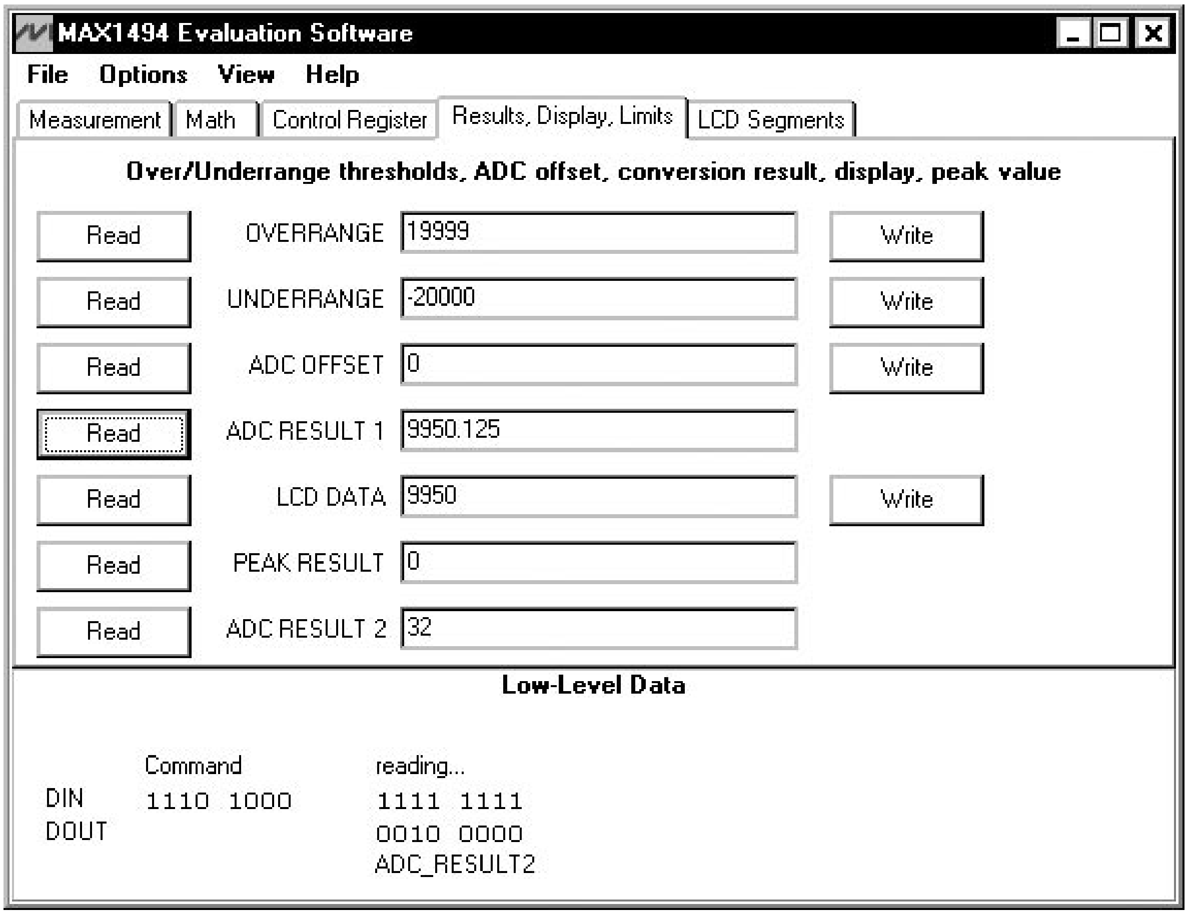


Figure 4. Results, Display, Limits Tab—MAX1494 Evaluation Software

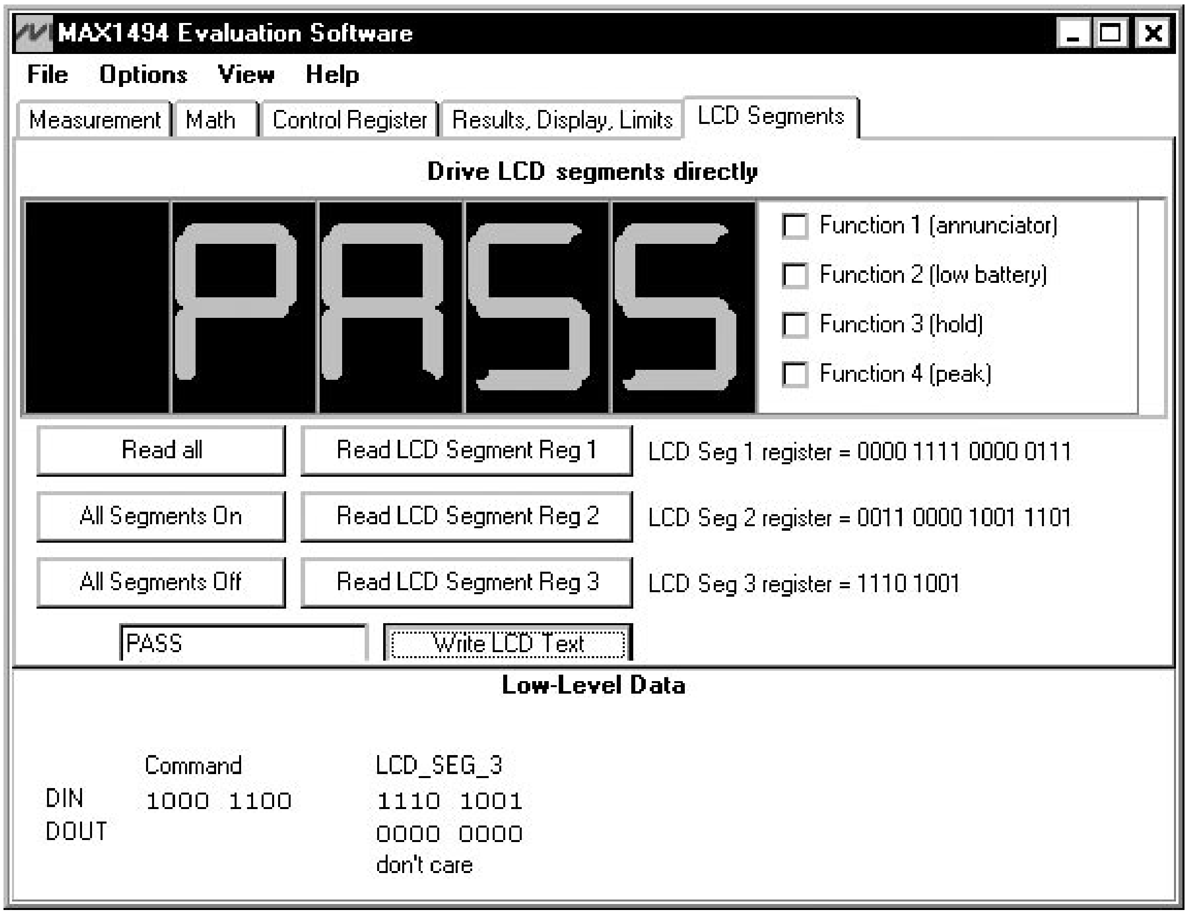


Figure 5. LCD Segments Tab—MAX1494 Evaluation Software

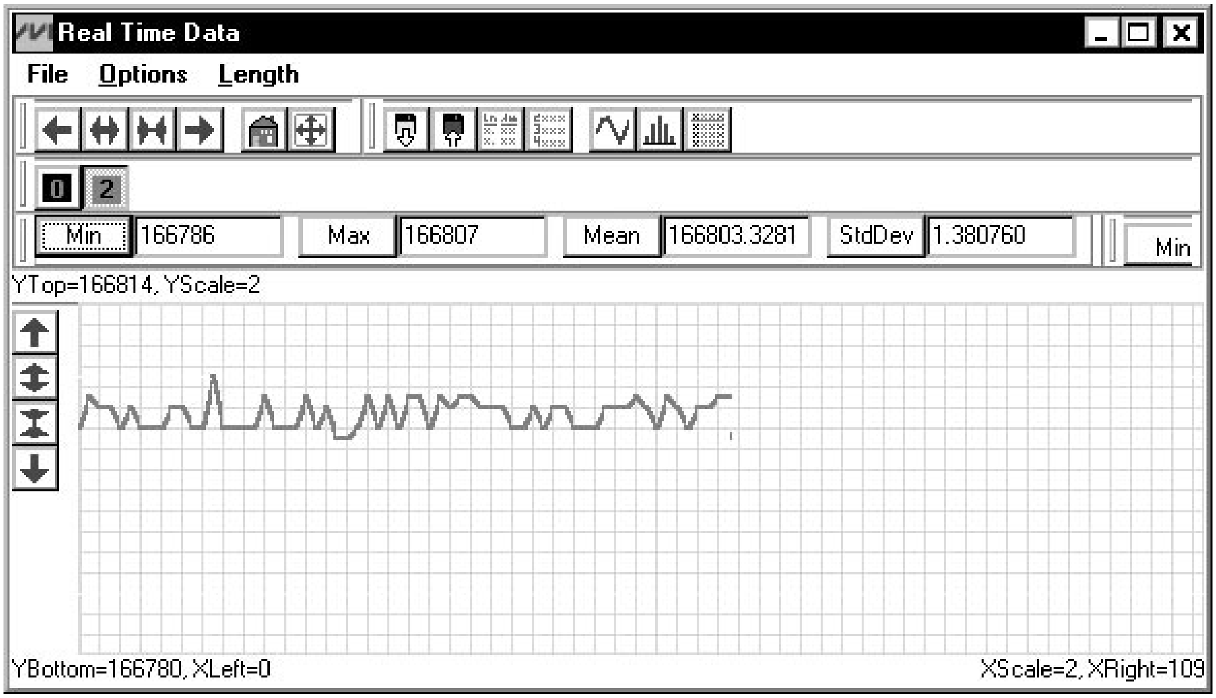


Figure 6. Graph Window—MAX1494 Evaluation Software

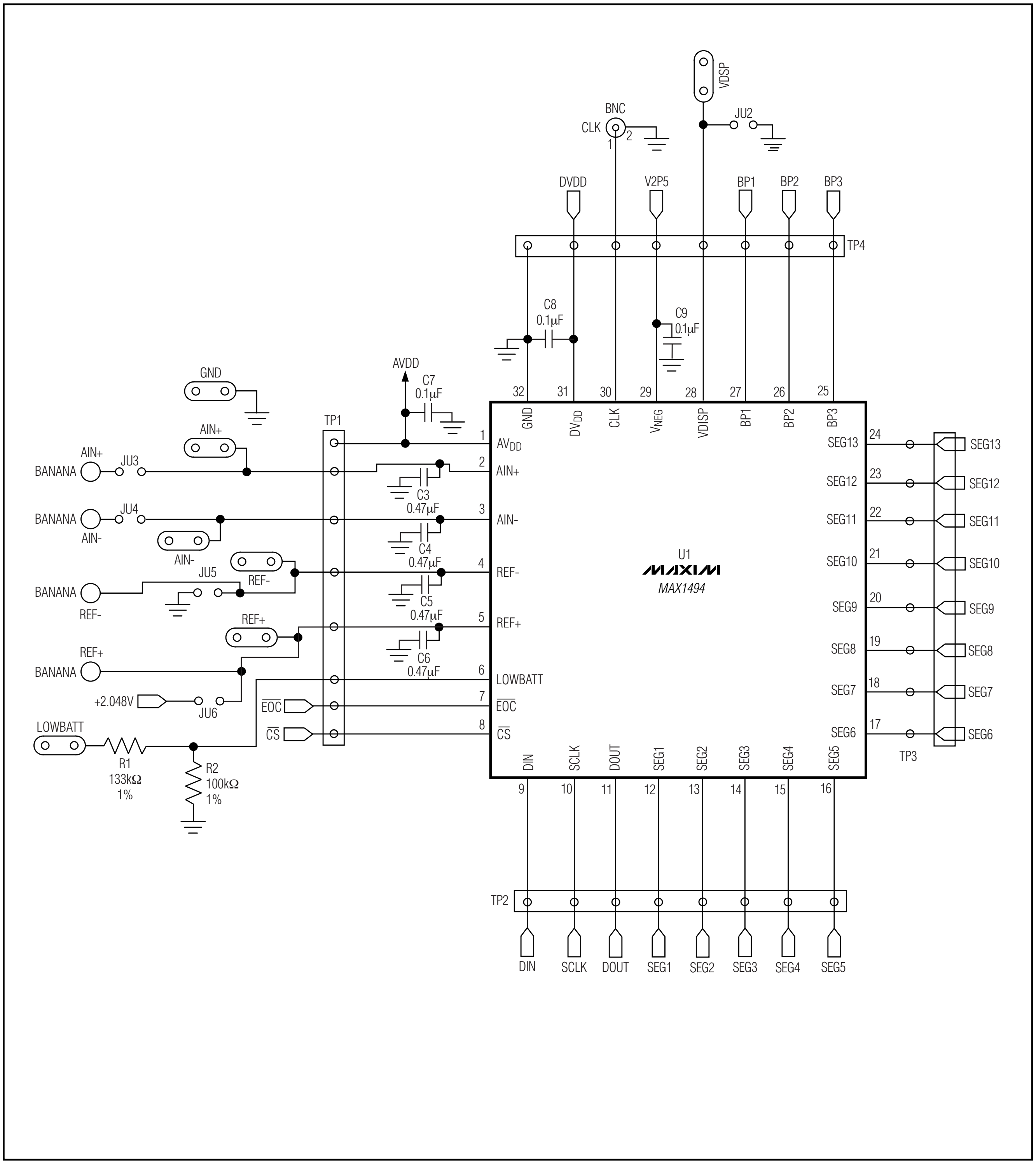


Figure 7a. MAX1494 EV Kit Schematic

MAX1494 Evaluation Kit/ Evaluation System

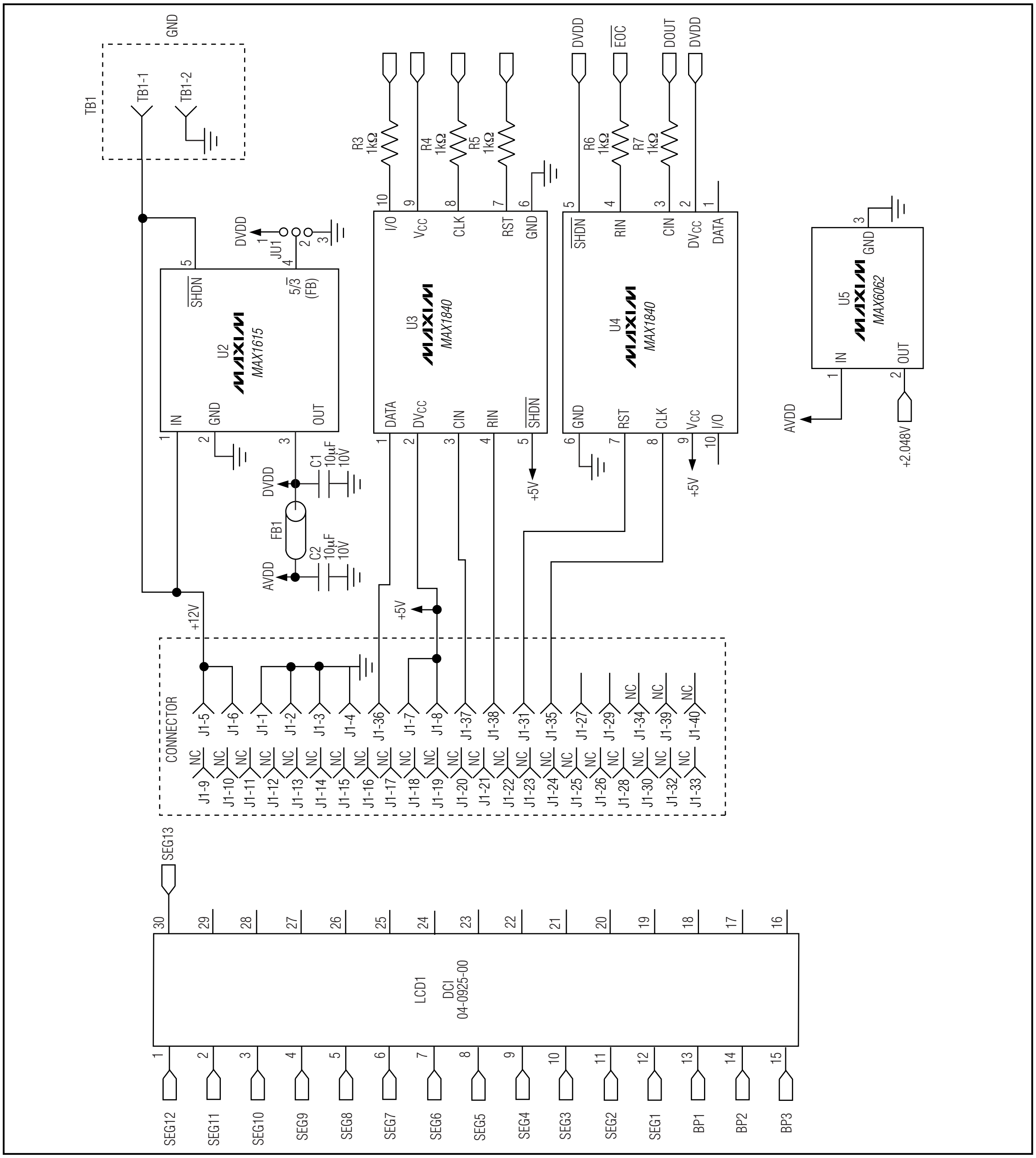


Figure 7b. MAX1494 EV Kit Schematic (continued)

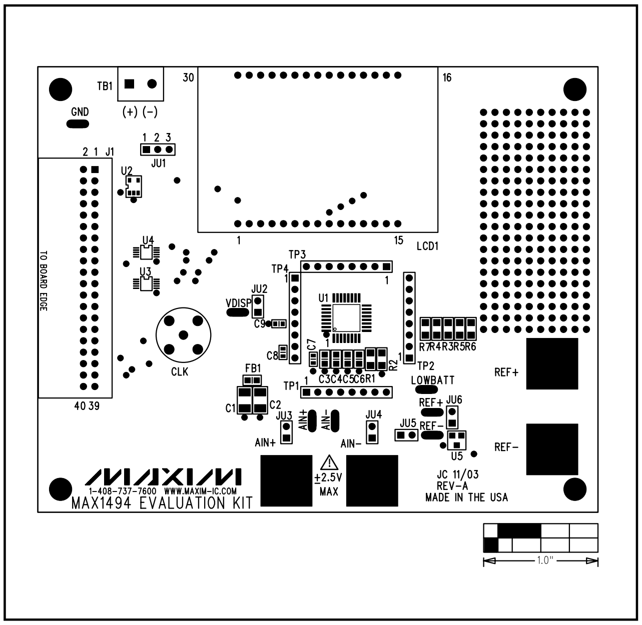


Figure 8. MAX1494 EV Kit Component Placement Guide—Top Silkscreen

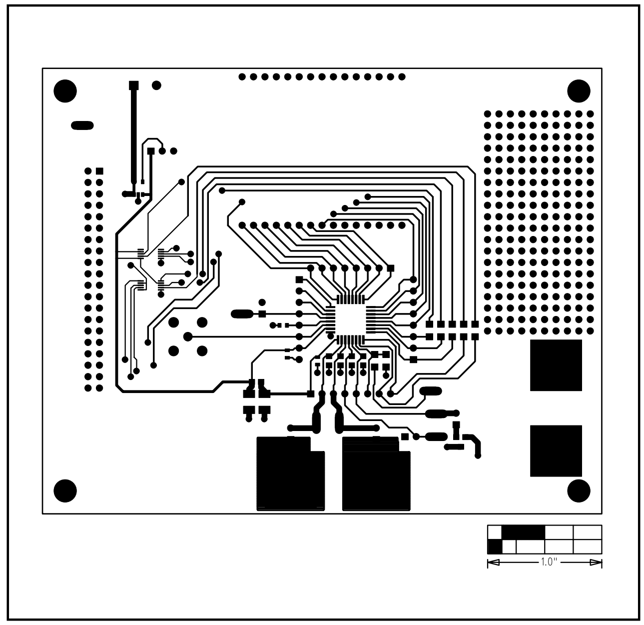


Figure 9. MAX1494 EV Kit PC Board Layout—Component Side

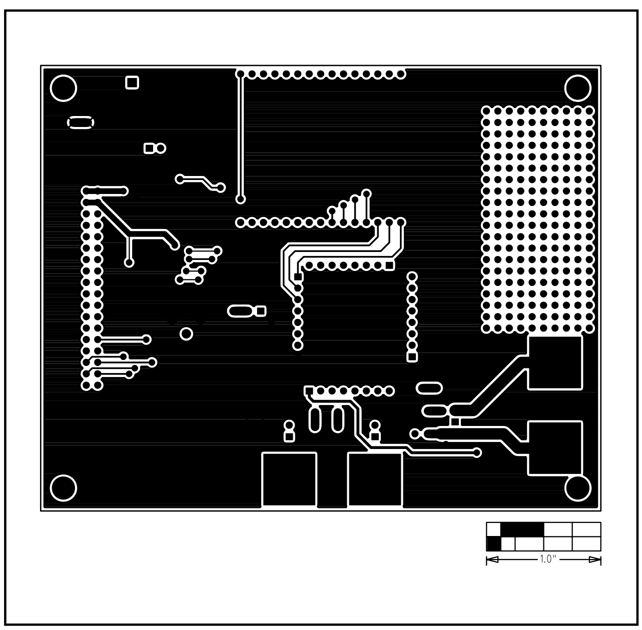


Figure 10. MAX1494 EV Kit PC Board Layout—Solder Side

```
// Drv1494.h
// MAX1494-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)
// 09/12/2003: add SPI_Transfer_After_EOC()
// 09/09/2003: add class MAX1494 dependent on external SPI_Interface()
// 08/13/2003: preliminary draft of reuseable code
//---------------------------------------------------------------------------
#ifndef drv1494H
#define drv1494H
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// 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 MAX1494_COMMS_START          0x80
#define MAX1494_COMMS_RW_MASK        0x40
#define MAX1494_COMMS_RW_WRITE           0x00
#define MAX1494_COMMS_RW_READ            0x40
#define MAX1494_COMMS_RS_MASK        0x3E
#define MAX1494_COMMS_RS_00000           0x00
#define MAX1494_COMMS_RS_STATUS          0x00
#define MAX1494_COMMS_RS_00001           0x02
#define MAX1494_COMMS_RS_CONTROL         0x02
#define MAX1494_COMMS_RS_00010           0x04
#define MAX1494_COMMS_RS_OVERRANGE       0x04
#define MAX1494_COMMS_RS_00011           0x06
#define MAX1494_COMMS_RS_UNDERRANGE      0x06
#define MAX1494_COMMS_RS_00100           0x08
#define MAX1494_COMMS_RS_LCD_SEG_1       0x08
#define MAX1494_COMMS_RS_00101           0x0A
#define MAX1494_COMMS_RS_LCD_SEG_2       0x0A
#define MAX1494_COMMS_RS_00110           0x0C
#define MAX1494_COMMS_RS_LCD_SEG_3       0x0C
#define MAX1494_COMMS_RS_00111           0x0E
#define MAX1494_COMMS_RS_ADC_OFFSET      0x0E
#define MAX1494_COMMS_RS_01000           0x10
#define MAX1494_COMMS_RS_ADC_RESULT1     0x10
#define MAX1494_COMMS_RS_01001           0x12
#define MAX1494_COMMS_RS_LCD_DATA        0x12
```

Figure 11. Listing 1 (Sheet 1 of 4)

```
#define MAX1494_COMMS_RS_01010          0x14
#define MAX1494_COMMS_RS_PEAK           0x14
#define MAX1494_COMMS_RS_10100          0x28
#define MAX1494_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
#define MAX1494_STATUS_POL_MASK         0x80
#define MAX1494_STATUS_POL_POSITIVE         0x00
#define MAX1494_STATUS_POL_NEGATIVE         0x80
#define MAX1494_STATUS_OVER_RANGE       0x40
#define MAX1494_STATUS_UNDER_RANGE      0x20
#define MAX1494_STATUS_LOW_BATTERY      0x10
#define MAX1494_STATUS_DATA_READY       0x08

//---------------------------------------------------------------------------
// Define the bits in the CONTROL register.
// SPI_ADC  EXTCLK  INTREF DP_EN  DPSET2  DPSET1    PD_DIG     PD_ANA
// HOLD    PEAK   RANGE  CLR  LCD    OFFSET_CAL1 OFFSET_CAL2  0
#define MAX1494_CONTROL_SPI_ADC         0x8000
#define MAX1494_CONTROL_EXTCLK          0x4000
#define MAX1494_CONTROL_INTREF          0x2000
#define MAX1494_CONTROL_DPMASK          0x1C00
#define MAX1494_CONTROL_DP_EN           0x1000
#define MAX1494_CONTROL_DPSET2          0x0800
#define MAX1494_CONTROL_DPSET1          0x0400
// (DPSET2 is the LSB and DPSET1 is the MSB)
#define MAX1494_CONTROL_DP1ON           0x1000 /* -1888.8 */
#define MAX1494_CONTROL_DP2ON           0x1800 /* -188.88 */
#define MAX1494_CONTROL_DP3ON           0x1400 /* -18.888 */
#define MAX1494_CONTROL_DP4ON           0x1C00 /* -1.8888 */
#define MAX1494_CONTROL_PD_DIG          0x0200
#define MAX1494_CONTORL_PD_ANA          0x0100
#define MAX1494_CONTROL_PD_ALL          0x0300
#define MAX1494_CONTROL_HOLD            0x0080
#define MAX1494_CONTROL_PEAK            0x0040
#define MAX1494_CONTROL_RANGE_200mV     0x0020
#define MAX1494_CONTROL_CLR             0x0010
#define MAX1494_CONTROL_SEG_SEL         0x0008
#define MAX1494_CONTROL_OFFSET_CAL1     0x0004
#define MAX1494_CONTROL_OFFSET_CAL2     0x0002

//---------------------------------------------------------------------------
// Define the bits in the LCD SEGMENT 1 register.
// A2 G2 D2 F2 E2 DP2 ANNUNCIATOR B1
// C1 A1 G1 D1 F1 E1 DP1 0
//
#define MAX1494_LCD_SEG1_A2           0x8000
#define MAX1494_LCD_SEG1_G2           0x4000
#define MAX1494_LCD_SEG1_D2           0x2000
#define MAX1494_LCD_SEG1_F2           0x1000
#define MAX1494_LCD_SEG1_E2           0x0800
#define MAX1494_LCD_SEG1_DP2          0x0400
#define MAX1494_LCD_SEG1_ANNUNCIATOR  0x0200
#define MAX1494_LCD_SEG1_B1           0x0100
#define MAX1494_LCD_SEG1_C1           0x0080
#define MAX1494_LCD_SEG1_A1           0x0040
#define MAX1494_LCD_SEG1_G1           0x0020
#define MAX1494_LCD_SEG1_D1           0x0010
#define MAX1494_LCD_SEG1_F1           0x0008
#define MAX1494_LCD_SEG1_E1           0x0004
#define MAX1494_LCD_SEG1_DP1          0x0002

//---------------------------------------------------------------------------
// Define the bits in the LCD SEGMENT 2 register.
// F4 E4 DP4 MINUS B3 C3 A3 G3
// D3 F3 E3 DP3 LOWBATT B2 C2 0
```

Figure 11. Listing 1 (Sheet 2 of 4)

```
//
#define MAX1494_LCD_SEG2_F4         0x8000
#define MAX1494_LCD_SEG2_E4      0x4000
#define MAX1494_LCD_SEG2_DP4     0x2000
#define MAX1494_LCD_SEG2_MINUS     0x1000
#define MAX1494_LCD_SEG2_B3         0x0800
#define MAX1494_LCD_SEG2_C3         0x0400
#define MAX1494_LCD_SEG2_A3         0x0200
#define MAX1494_LCD_SEG2_G3         0x0100
#define MAX1494_LCD_SEG2_D3         0x0080
#define MAX1494_LCD_SEG2_F3         0x0040
#define MAX1494_LCD_SEG2_E3      0x0020
#define MAX1494_LCD_SEG2_DP3     0x0010
#define MAX1494_LCD_SEG2_LOWBATT    0x0008
#define MAX1494_LCD_SEG2_B2         0x0004
#define MAX1494_LCD_SEG2_C2         0x0002

//--------------------------------------------------------------------------
// Define the bits in the LCD SEGMENT 3 register.
// ?PEAK? ?HOLD? BC5 B4 C4 A4 G4 D4
//
#define MAX1494_LCD_SEG3_PEAK         0x80
#define MAX1494_LCD_SEG3_HOLD         0x40
#define MAX1494_LCD_SEG3_BC5         0x20
#define MAX1494_LCD_SEG3_B4             0x10
#define MAX1494_LCD_SEG3_C4             0x08
#define MAX1494_LCD_SEG3_A4             0x04
#define MAX1494_LCD_SEG3_G4             0x02
#define MAX1494_LCD_SEG3_D4             0x01

//--------------------------------------------------------------------------
class MAX1494
{
public:
    MAX1494(void);

    // Enumerated type describing the register select bits.
    enum RegisterSelect_t {
        RS_STATUS       = MAX1494_COMMS_RS_STATUS,
        RS_CONTROL      = MAX1494_COMMS_RS_CONTROL,
        RS_OVERRANGE    = MAX1494_COMMS_RS_OVERRANGE,
        RS_UNDERRANGE   = MAX1494_COMMS_RS_UNDERRANGE,
        RS_LCD_SEG_1    = MAX1494_COMMS_RS_LCD_SEG_1,
        RS_LCD_SEG_2    = MAX1494_COMMS_RS_LCD_SEG_2,
        RS_LCD_SEG_3    = MAX1494_COMMS_RS_LCD_SEG_3,
        RS_ADC_OFFSET   = MAX1494_COMMS_RS_ADC_OFFSET,
        RS_ADC_RESULT1  = MAX1494_COMMS_RS_ADC_RESULT1,
        RS_LCD_DATA     = MAX1494_COMMS_RS_LCD_DATA,
        RS_PEAK         = MAX1494_COMMS_RS_PEAK,
        RS_ADC_RESULT2  = MAX1494_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 DP_EN DPSET2 DPSET1 PD_DIG PD_ANA
    // HOLD PEAK RANGE CLR LCD OFFSET_CAL1 OFFSET_CAL2 0
    int CONTROL_REG;
```

Figure 11. Listing 1 (Sheet 3 of 4)

```
    //
    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);

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

};

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

Figure 11. Listing 1 (Sheet 4 of 4)

```
// Drv1494.cpp
// MAX1494-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 MAX1494 dependent on external SPI_Interface()
// 08/13/2003: preliminary draft of reuseable code

#include "drv1494.h"

//---------------------------------------------------------------------------
MAX1494::MAX1494(void)
{
    vref = 2.048;
    extended_resolution = false;
}
//---------------------------------------------------------------------------
bool MAX1494::Read_STATUS(void)
{
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1494_COMMS_START |
            MAX1494_COMMS_RW_READ | MAX1494_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 MAX1494::Write_CONTROL(int data)
{
    data = data & 0xFFFF;       // validate the data
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1494_COMMS_START |
            MAX1494_COMMS_RW_WRITE | MAX1494_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 &=~ MAX1494_CONTROL_CLR;
    return result;
}
//---------------------------------------------------------------------------
bool MAX1494::Read_CONTROL(void)
{
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1494_COMMS_START |
            MAX1494_COMMS_RW_READ | MAX1494_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
    }
```

Figure 12. Listing 2 (Sheet 1 of 4)

```
        return result;
}
//---------------------------------------------------------------------------
bool MAX1494::Read_ADC_RESULT1(void)
{
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1494_COMMS_START |
            MAX1494_COMMS_RW_READ | MAX1494_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 MAX1494::Read_ADC_RESULT2(void)
{
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1494_COMMS_START |
            MAX1494_COMMS_RW_READ | MAX1494_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 MAX1494::Read_DATA(void)
{
    // Read the DATA register
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1494_COMMS_START |
            MAX1494_COMMS_RW_READ | MAX1494_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)(MAX1494_COMMS_START |
                MAX1494_COMMS_RW_READ | MAX1494_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
```

Figure 12. Listing 2 (Sheet 2 of 4)

```
            }
            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;
        }
        return DATA_REG;
    }
    //---------------------------------------------------------------------------
    double MAX1494::Voltage(void)
    {
        if ((CONTROL_REG & MAX1494_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 MAX1494::Write_2s_complement(int reg, int data)
    {
        // Write one of the 2's complement registers
        reg = (reg & MAX1494_COMMS_RS_MASK);
        data = data & 0xFFFF;        // validate the data

        const unsigned __int8 mosi[] = {
            (unsigned __int8)(MAX1494_COMMS_START | MAX1494_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 MAX1494::Read_2s_complement(int reg)
    {
        // Read one of the 2's complement registers
        reg = (reg & MAX1494_COMMS_RS_MASK);

        const unsigned __int8 mosi[] = {
            (unsigned __int8)(MAX1494_COMMS_START | MAX1494_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 MAX1494::Write_8bit_reg(int reg, int data)
    {
        // Write one of the 8 bit registers
        reg = (reg & MAX1494_COMMS_RS_MASK);
        const unsigned __int8 mosi[] = {
```

Figure 12. Listing 2 (Sheet 3 of 4)

```
            (unsigned __int8)(MAX1494_COMMS_START | MAX1494_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 MAX1494::Read_8bit_reg(int reg)
{
        // Read one of the 8 bit registers
        reg = (reg & MAX1494_COMMS_RS_MASK);
        const unsigned __int8 mosi[] = {
            (unsigned __int8)(MAX1494_COMMS_START | MAX1494_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;
}
//---------------------------------------------------------------------------
```

Figure 12. Listing 2 (Sheet 4 of 4)