MAXIM

MAX1254 Evaluation Kit/Evaluation System

Evaluate: MAX1254/MAX1253

General Description

The MAX1254 evaluation system (EV system) consists of a MAX1254 evaluation kit (EV kit) and a Maxim 68HC16MODULE-DIP microcontroller (µC) module. The MAX1254 is a stand-alone, 10-channel, 12-bit system monitor ADC with internal reference. The evaluation software runs under Windows® 95/98/2000/XP, providing a convenient user interface to exercise the features of the MAX1254.

Order the complete EV system (MAX1254EVC16) for comprehensive evaluation of the MAX1254 using a personal computer. Order the EV kit (MAX1254EVKIT) if the 68HC16MODULE-DIP module has already been purchased with a Maxim EV system, or for custom use in other µC-based systems.

This system can also evaluate the MAX1253. Contact the factory for free samples of the product.

MAX1254 Stand-Alone EV Kit

The MAX1254 EV kit provides a proven PC board layout to facilitate evaluation of the MAX1254. It must be interfaced to appropriate timing signals for proper operation. Connect 6-20V and ground return to terminal block TB1 (Figure 10). Refer to the MAX1254 data sheet for timing requirements.

MAX1254 EV System

The MAX1254 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.

MAX1254EVC16 Component List

PART	QTY	DESCRIPTION
MAX1254EVKIT	1	MAX1254 EV kit
68HC16MODULE-DIP	1	68HC16 µC module

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
MAX1254EVKIT	0°C to +70°C	User supplied
MAX1254EVC16	0°C to +70°C	Windows software

Note: The MAX1254 evaluation software is designed for use with the complete evaluation system (MAX1254EVC16), which includes the 68HC16MODULE-DIP module together with the MAX1254EVKIT. If the MAX1254 evaluation software will not be used, the MAX1254EVKIT board can be purchased by itself, without the microcontroller.

MAX1254EV KIT Component List

DESIGNATION	QTY	DESCRIPTION
C1, C2	2	0.1µF ceramic capacitors (1206) Murata GRM319R71H104K
C3, C4	2	0.01µF, 10V min X7R ceramic capacitors (0805) Murata GRM216R71H103K Taiyo Yuden UMK212B103KQ
C5	1	10µF ±20%, 10V X7R ceramic capacitor (1210) Taiyo Yuden LMK325BJ106MN TDK C3225X7R1C106M TDK C3225X5R1A106M
C6–C13	8	Open (0805) (reserved for 0.01µF 10V min X7R ceramic capacitors)
H1, H2	2	9-pin headers
J1	1	2 x 20 right angle socket, Methode (Adam Tech) RS2R-40G SamTec SSW-120-02-S-D-RA

Windows is a registered trademark of Microsoft Corp.

MAXIM

For pricing, delivery, and ordering information, please contact Maxim/Dallas Direct! at 1-888-629-4642, or visit Maxim's website at www.maxim-ic.com.

MAX1254EV KIT Component List (continued)

DESIGNATION	QTY	DESCRIPTION
JU1:1-2, JU2, JU3:1-2, JU4, JU5, JU6:1-2	6	Shunts
JU1, JU3, JU6	3	3-pin headers
JU2, JU4, JU5	3	2-pin headers
Q1	1	2N3904 transistor (SOT23) Central Semiconductor CMPT3904 Diodes Incorporated MMBT3904-7 Fairchild MMBT3904 Motorola MMBT3904LT1

DESIGNATION	QTY	DESCRIPTION
R1, R3	0	Open, shorted by PC board trace (1206)
R2, R4	0	Open (1206)
R5, R6	2	10kΩ ±5% resistors (1206)
TB1	1	2-circuit terminal block
U1	1	MAX1254BEUE
U2	1	MAX1615EUK-T (SOT23-5)
U3, U4	2	MAX1840EUB or MAX1841EUB
U5	1	MAX6033AAUT25-T (SOT23-6)
None	1	MAX1254EVKIT PC board

Component Suppliers

SUPPLIER	PHONE	FAX	WEBSITE
Central Semiconductor	631-435-1100	631-435-1824	www.centralsemi.com
Murata	770-436-1300	770-436-3030	www.murata.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: *Please indicate you are using the MAX1254 when contacting these component suppliers.*

Quick Start

Required Equipment

Before you begin, you need the following equipment:

- MAX1254EVC16 (contains MAX1254EVKIT board and 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 1-2 position, JU2 is closed, JU3 is in 1-2 position, JU4 is open, JU5 is open, and JU6 is in 1-2 position (Tables 1–4).

2) Carefully connect the boards by aligning the 40-pin header of the MAX1254 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 +7V to +20V DC 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 MAX1254 program by opening the 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) To initialize the MAX1254 for the evaluation kit board, press function key F10. This enables the internal reference, with AIN0 and AIN1 for differential temperature measurement, enables AIN2 through AIN5 for single-ended voltage inputs, and disables AIN6 and AIN7.

MAXIM

MAX1254 Evaluation Kit/Evaluation System

10) Apply an input signal in the 0 to V_{REF} (4.096V) range between AIN2 and GND. Observe the readout on the screen.

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

__Detailed Description of Software

Main Window

The evaluation software's main window shows a block diagram of the MAX1253/MAX1254, with many clickable features. Each feature can be operated from the block diagram tab sheet or from the other tab sheets (see Figure 1.)

Press function key F1 at any time to return to the block diagram tab sheet. Press function key F2 to pop up a window containing the most recent data readings. The software reads the data registers every 300ms, unless disabled by unchecking **Poll All Data Registers** under the **Device** menu (see Figure 3.)

At startup, the evaluation software reads the device configuration. To initialize the MAX1254 for the evaluation kit board, press function key F10. This enables the internal reference, with AIN0 and AIN1 for differential temperature measurement, enables AIN2 through AIN5 for single-ended voltage inputs, and disables AIN6 and AIN7 (see Figure 2.)

The **Data Regs** tab sheet provides read and write access to all channel data registers. Individual measurements must be triggered unless scanning mode has been enabled. The data write commands are useful for preloading nominal initial values into the channel averaging filters. The **Reference Voltage** value only affects the displayed voltages, not the raw measurement codes (see Figure 4.)

The **Global Registers** tab sheet covers the global configuration registers in three groups: channel enable, input configuration, and setup. The **Input Configuration** tab sheet determines how each pair of inputs is connected. The **Setup Register** tab sheet configures internal or external reference voltage, interrupt output level, and scanning. All three of these subgroups require clicking the **Apply** button to write the new configuration into the device (see Figures 5, 6, 7.)

The INT pin status is normally shown in the status bar, unless disabled by unchecking **Poll INT Pin** under the **Device** menu.

The **Alarm Register** tab sheet reads and clears the alarm register. Alarm fault status is normally displayed in the status bar, unless disabled by unchecking **Poll Alarm Status** under the **Device** menu (see Figure 8.)

The **Channel Configuration** tab sheet configures each channel's fault detection and averaging. After changing any channel's settings, click the **Apply** button to write the new configuration into the device (see Figure 9.)

Graph Window

To view recently measured data, drop down the **View** menu and choose **Graph**. Data may be viewed as a time sequence plot, a histogram plot, or as a table of raw numbers. See Table 5 for available graph commands.

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**. 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 into a file in comma-delimited or tab-delimited format. Line numbers and a descriptive header line are optional.

Graph channel 0 plots the internal temperature of the MAX1254. Channel 1 is V_{DD}. Channels 2 through 9 are MAX1254 AIN0 through AIN7.

Evaluating the MAX1253

With power off, replace U1 with a MAX1253BEUA. Follow the *Quick Start* instructions, except in step 1 select V_{DD} = 3V by setting JU1 to 2-3 position. Tell the software the reference voltage is 2.5V.

Diagnostics Window

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

Evaluate: MAX1254/MAX1253

Table 1. Jumper JU1 (V$_{DD}$ Voltage Selection)

JU1 SHUNT POSITION	V$_{DD}$ (V)	FUNCTION
1-2*	5	Normal operation with U1 = MAX1254
2-3	3	Normal operation with U1 = MAX1253
Open	Unspecified	Do not operate kit with JU1 open

*Indicates default configuration.

Table 2. Jumpers JU2 and JU3 (Remote Temperature Sensor)

JU2 SHUNT POSITION	JU3 SHUNT POSITION	AIN0, 1 INPUT CONFIGURATION (GLOBAL REGISTERS)	FUNCTION
Closed*	1-2*	110	Configure AIN0 and AIN1 for differential remote temperature sense. AIN1 is return connection.
Closed	2-3	010	Configure AIN0 for single-ended remote temperature sense. AIN1 can be used for voltage input.
Closed	2-3	011	Configure AIN0 for single-ended remote temperature sense. AIN1 can be used for an off-board temperature-sensing transistor.
Closed	Open	Invalid	Do not operate kit in this configuration
Open	1-2	Invalid	Do not operate kit in this configuration
Open	2-3	Invalid	Do not operate kit in this configuration
Open	Open	000, 001, 010, 011, 100, 101, 110	AIN0 and AIN1 are available for voltage inputs or off-board remote temperature sensing.

*Indicates default configuration.

Table 3. Jumpers JU4 and JU5 (Reference Voltage)

JU4 SHUNT POSITION	JU5 SHUNT POSITION	REFERENCE MODE (GLOBAL REGISTERS, SETUP)	FUNCTION
Closed	Closed	00	Use on-board external reference U5 and C2 bypass capacitor. Do not use internal reference mode. EV kit on-board MAX6033 reference voltage = 2.5V.
Open	Closed	00	Use an off-board external reference with on-board bypass capacitor C2. Do not use internal reference mode.
Open*	Open*	01, 10	Use internal reference with C2 disconnected. MAX1254 internal reference voltage = 4.096V. MAX1253 internal reference voltage = 2.5 V.

*Indicates default configuration.

MAXIM

MAX1254 Evaluation Kit/Evaluation System

Table 4. Jumper JU6 (INT Output)

JU6 SHUNT POSITION	INT MODES (GLOBAL REGISTERS, SETUP)	FUNCTION
1-2*	00	INT is pulled up to VDD
2-3	01	INT is pulled down to GND
Open	10, 11	No pullup or pulldown on INT output

*Indicates default configuration.

Table 5. 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

| TOOL	FUNCTION
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 (MAX1254 internal temperature)
1	Channel 1 enable (MAX1254 supply voltage VDD)
2	Channel 2 enable (MAX1254 input AIN0)
3	Channel 3 enable (MAX1254 input AIN1)
4	Channel 4 enable (MAX1254 input AIN2)
5	Channel 5 enable (MAX1254 input AIN3)
6	Channel 6 enable (MAX1254 input AIN4)
7	Channel 7 enable (MAX1254 input AIN5)
8	Channel 8 enable (MAX1254 input AIN6)
9	Channel 9 enable (MAX1254 input AIN7)

MAXIM

Evaluate: MAX1254/MAX1253
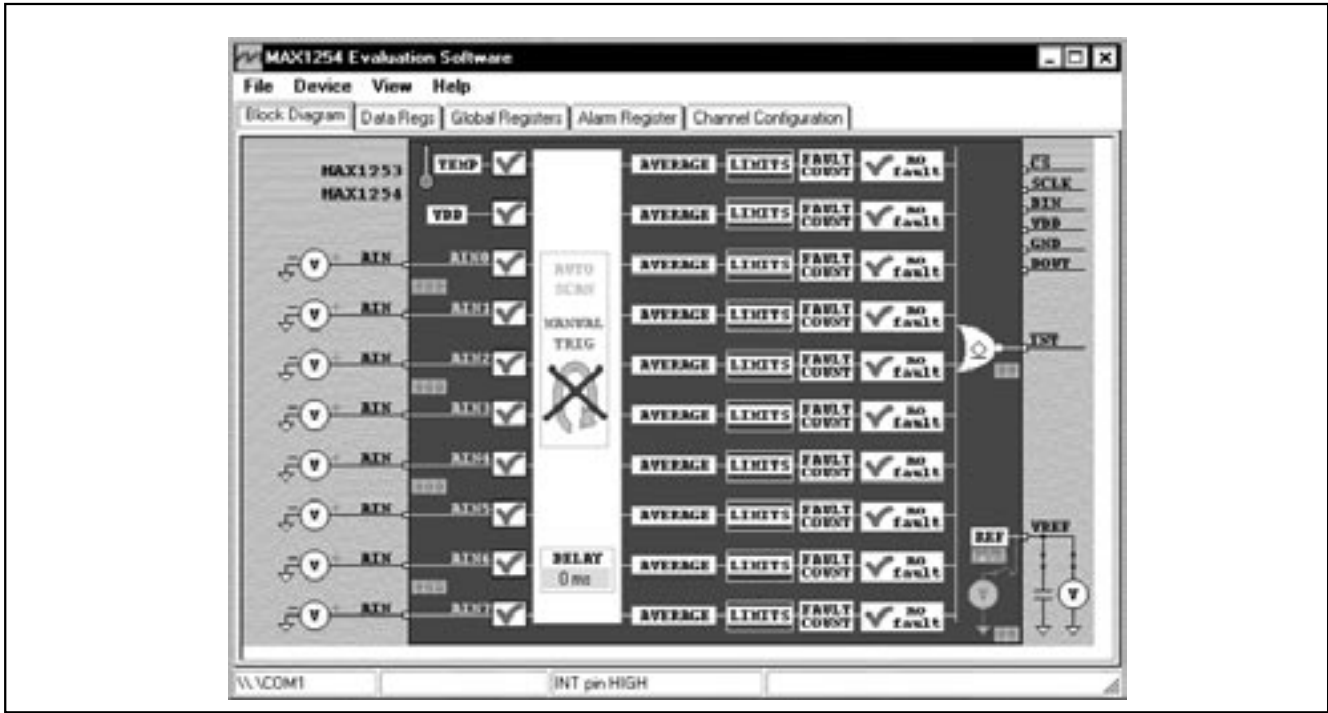
Figure 1. Initial Configuration (Power-On Defaults for MAX1254)
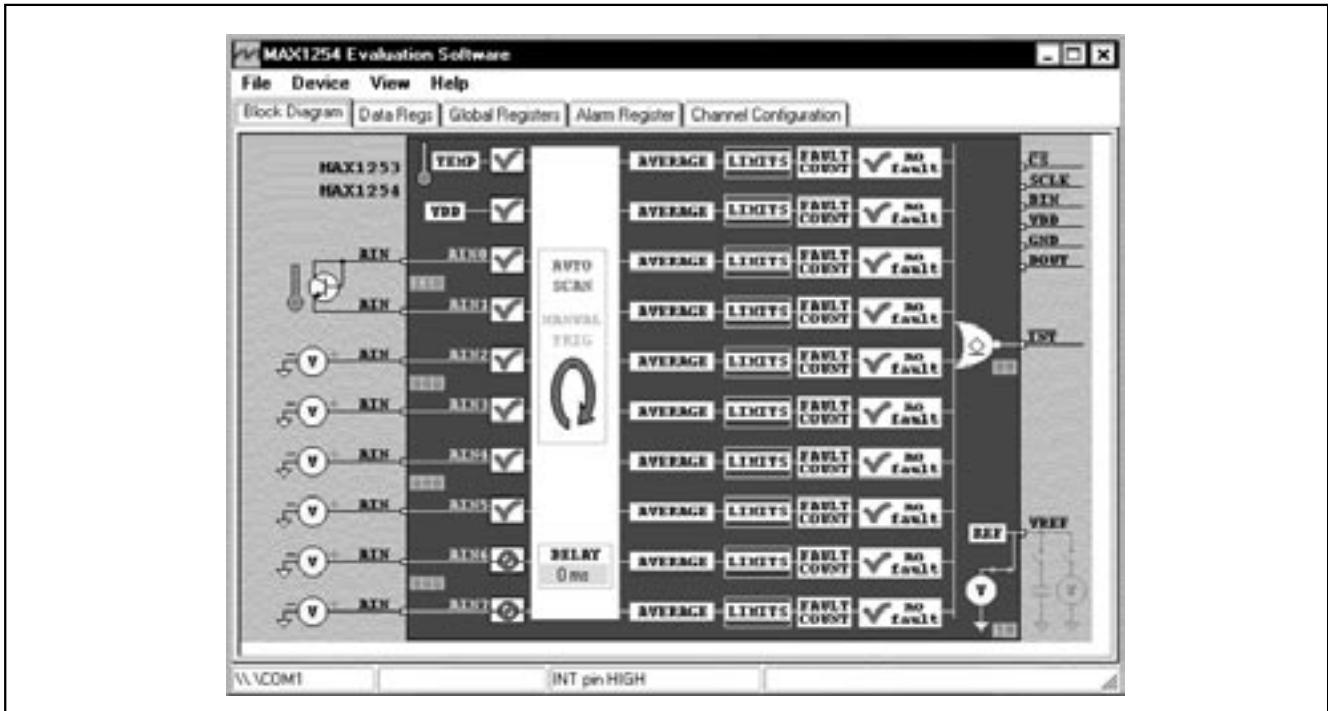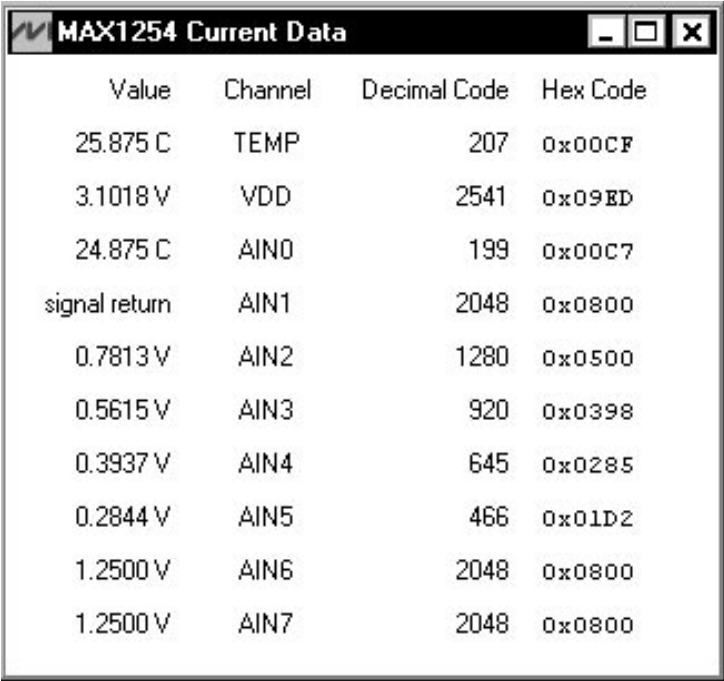
Figure 2. Block Diagram Showing Device Configuration After Pressing Function Key F10
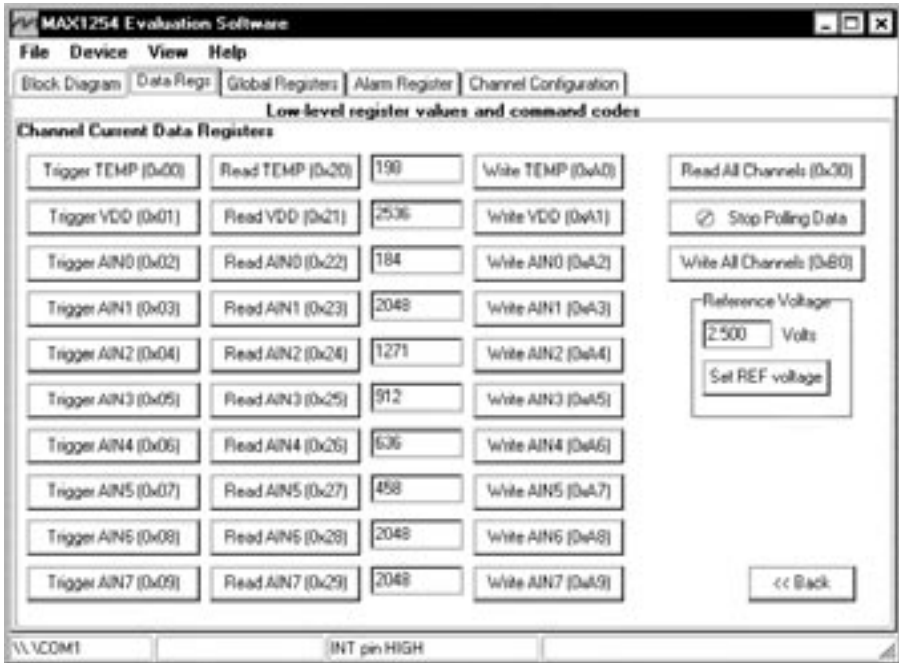
ϺΛXΙϺ

Evaluate: MAX1254/MAX1253

Figure 3. Current Data Panel Shown By Pressing Function Key F2

Figure 4. Data Register Screen Shot
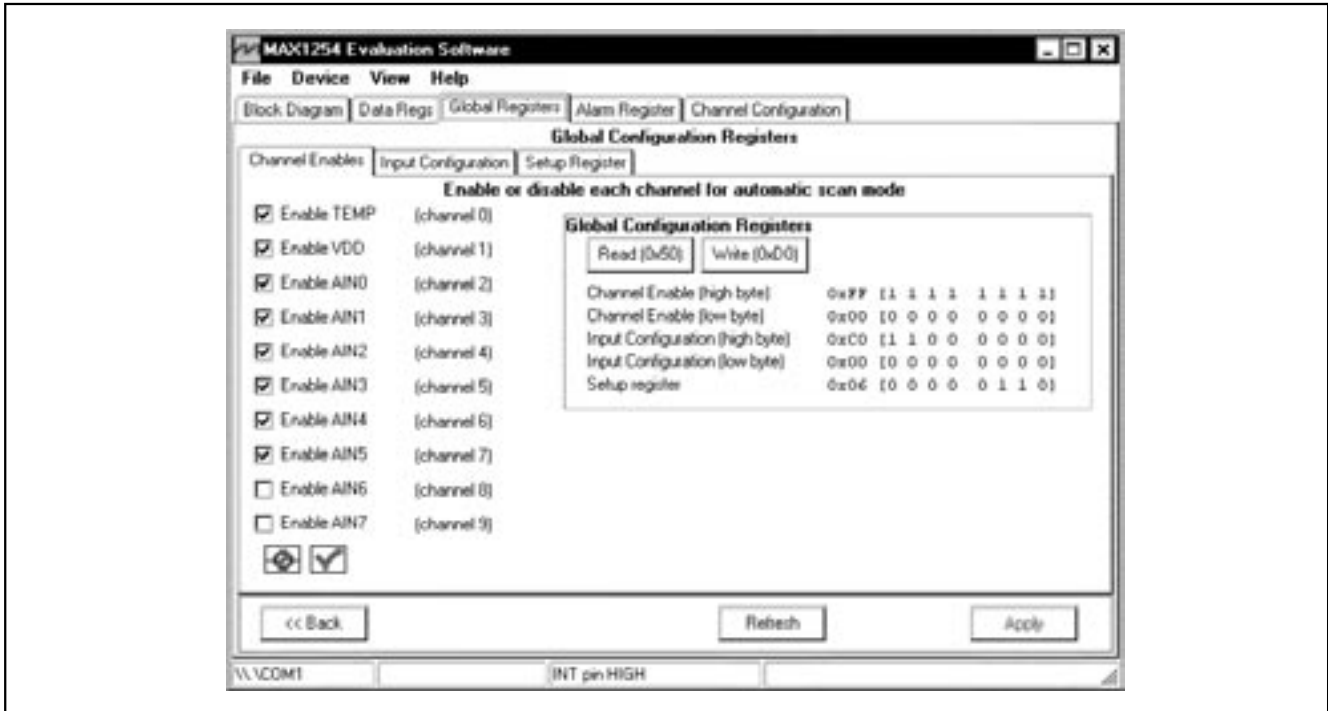
Figure 5. Global Register—Channel Enable Screen Shot
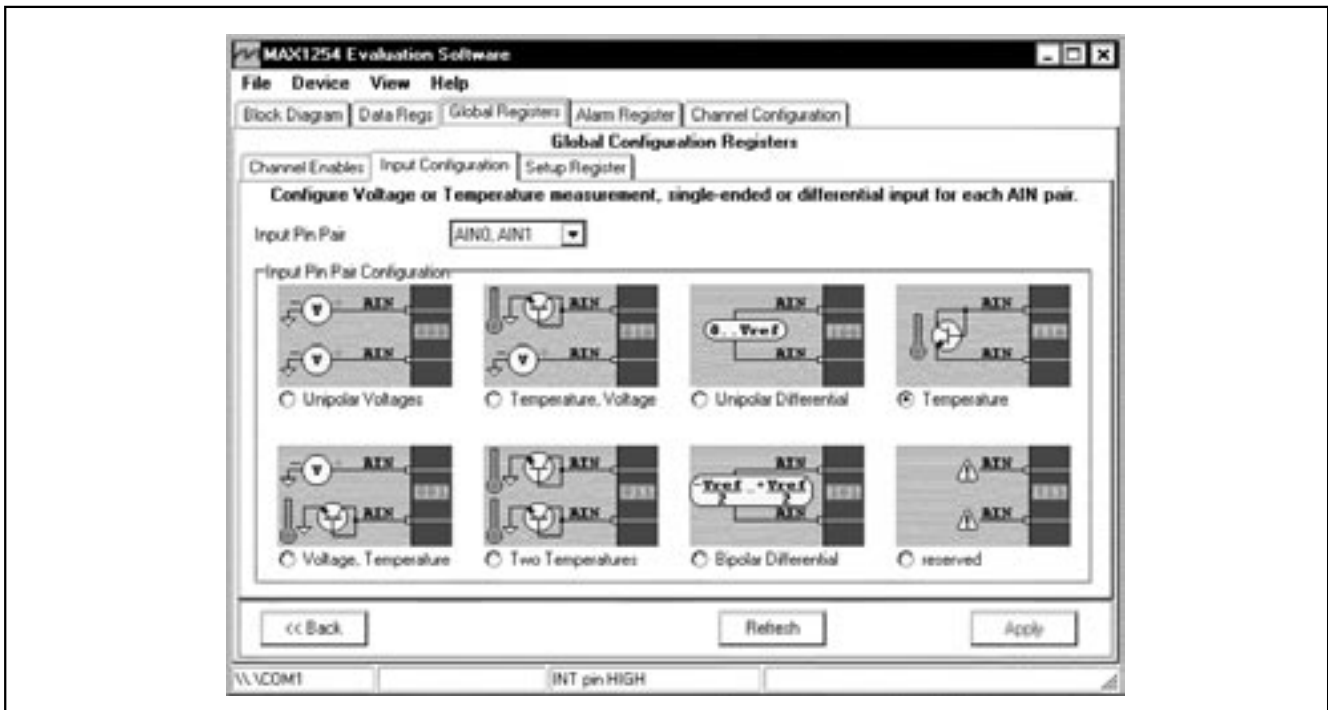
Figure 6. Global Register—Input Configuration Screen Shot

Evaluate: MAX1254/MAX1253
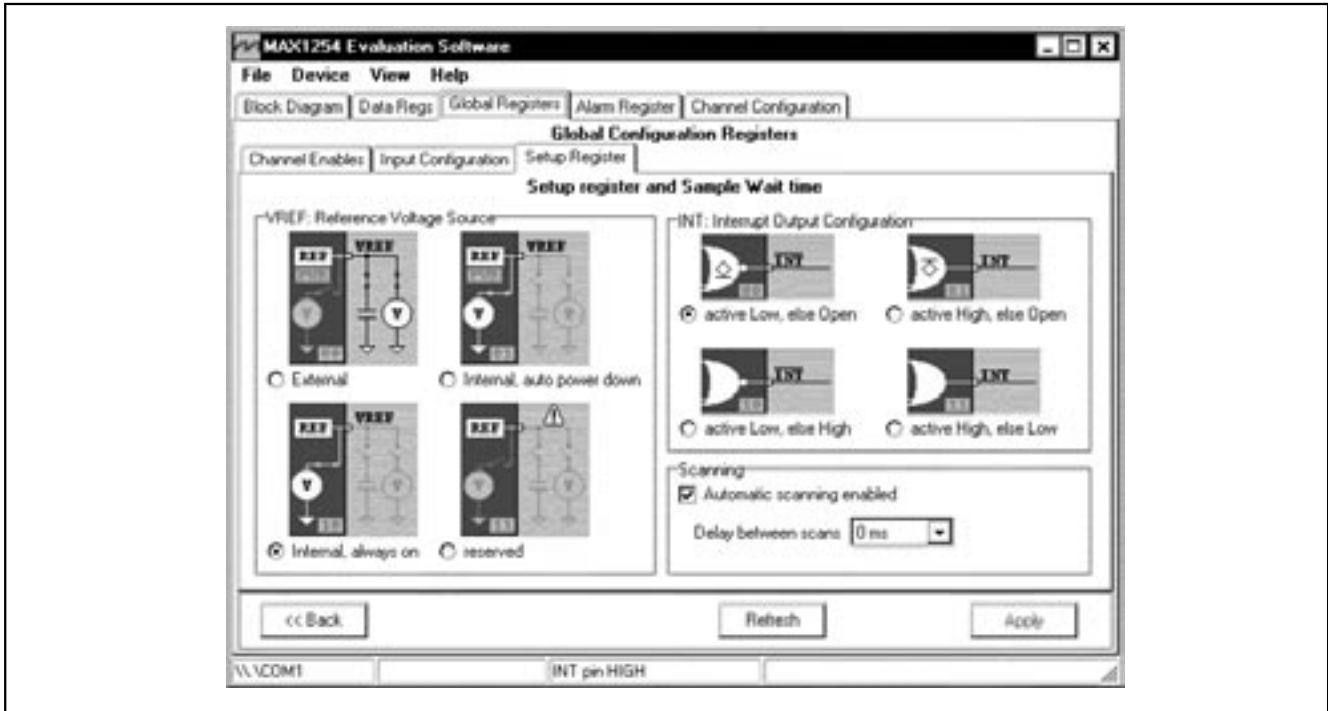
Figure 7. Global Register—Setup Register Screen Shot
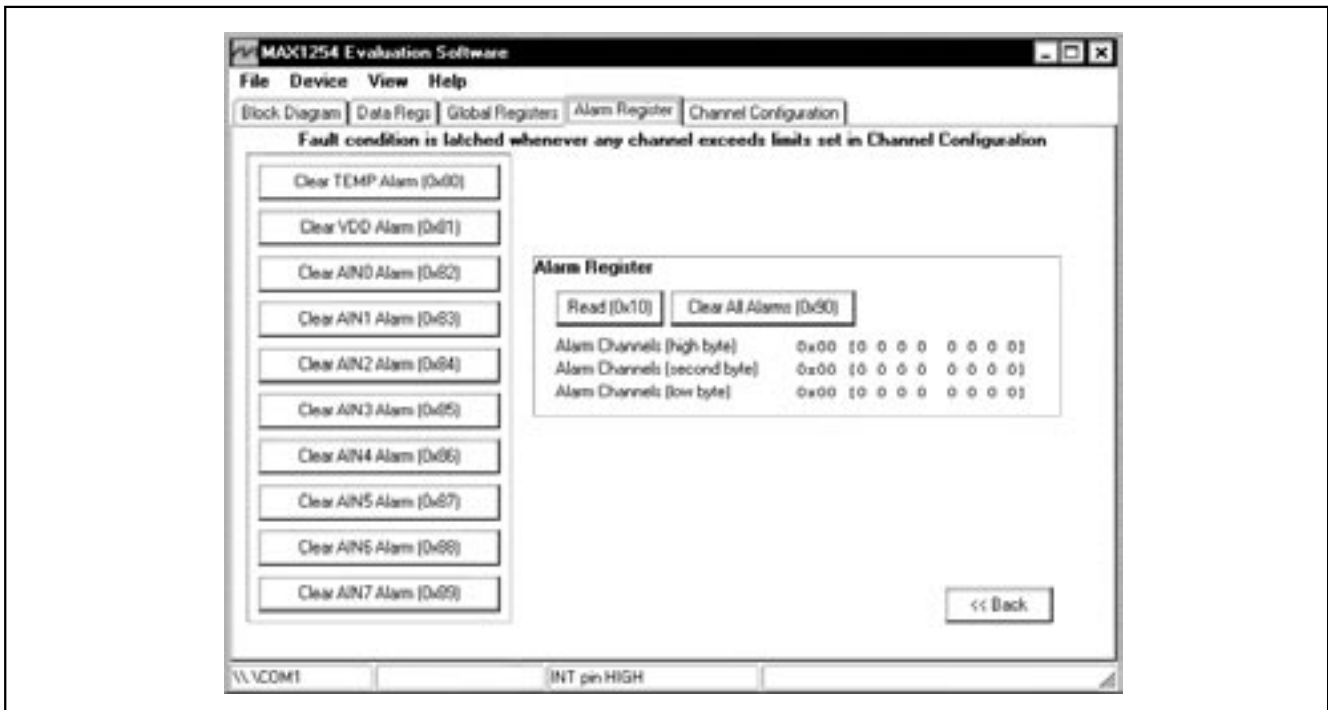
Figure 8. Alarm Register Screen Shot

Evaluate: MAX1254/MAX1253
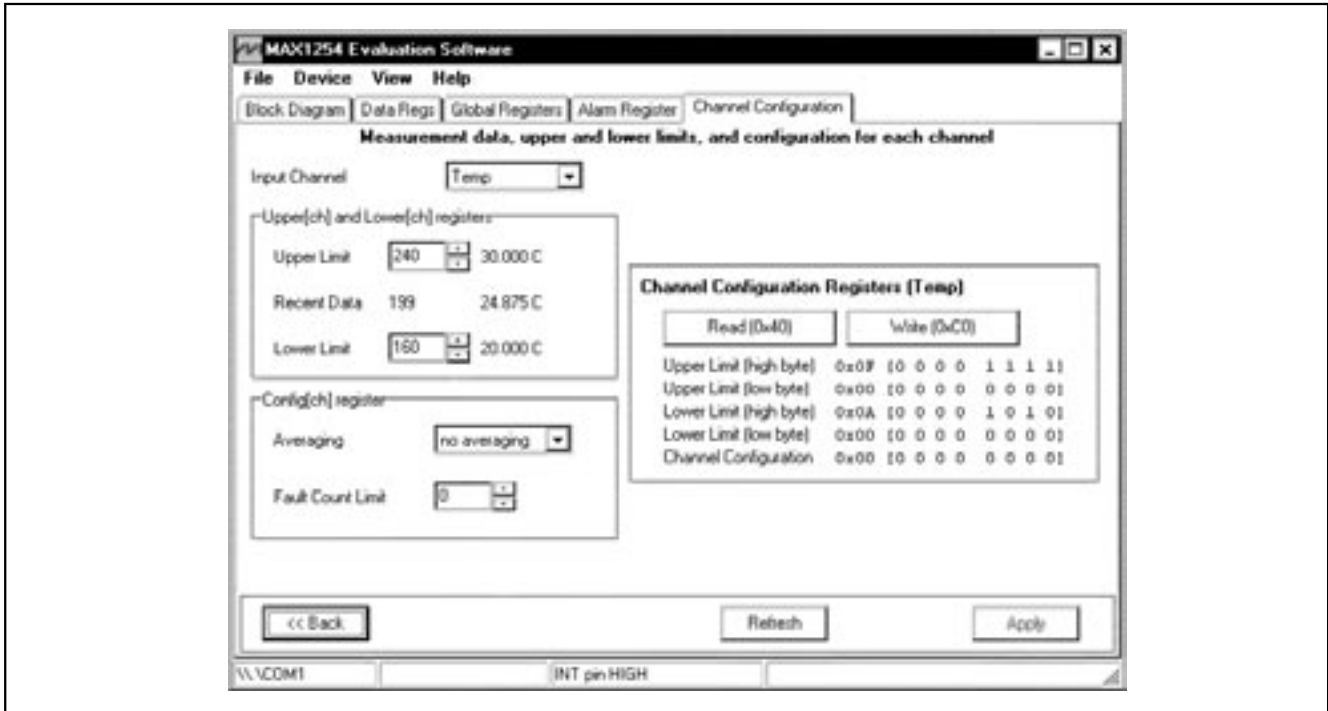
Figure 9. Channel Configuration Screen Shot

MAXIM

MAX1254 Evaluation Kit/Evaluation System
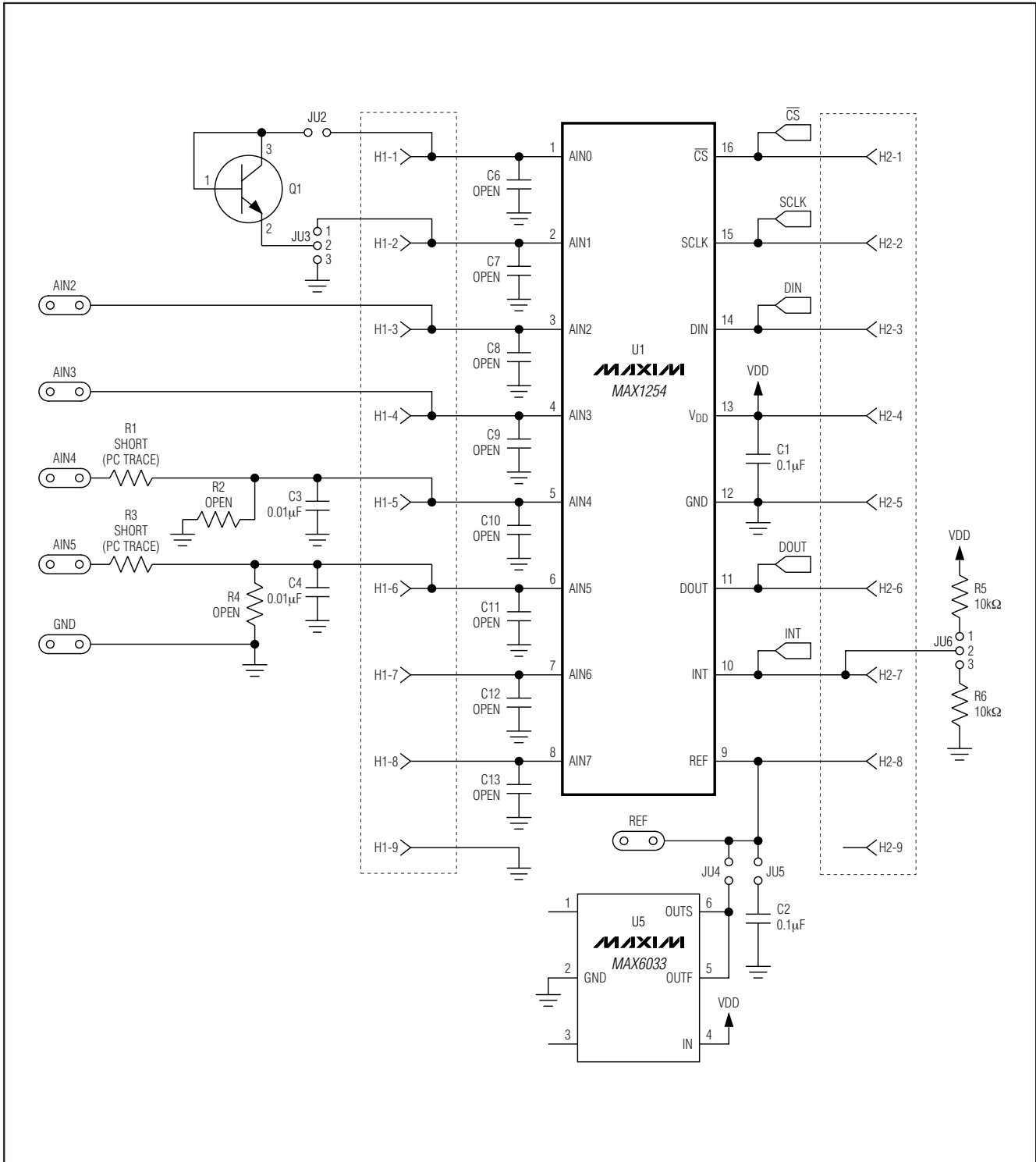
Figure 10a. MAX1254 EV Kit Schematic (Sheet 1 of 2)
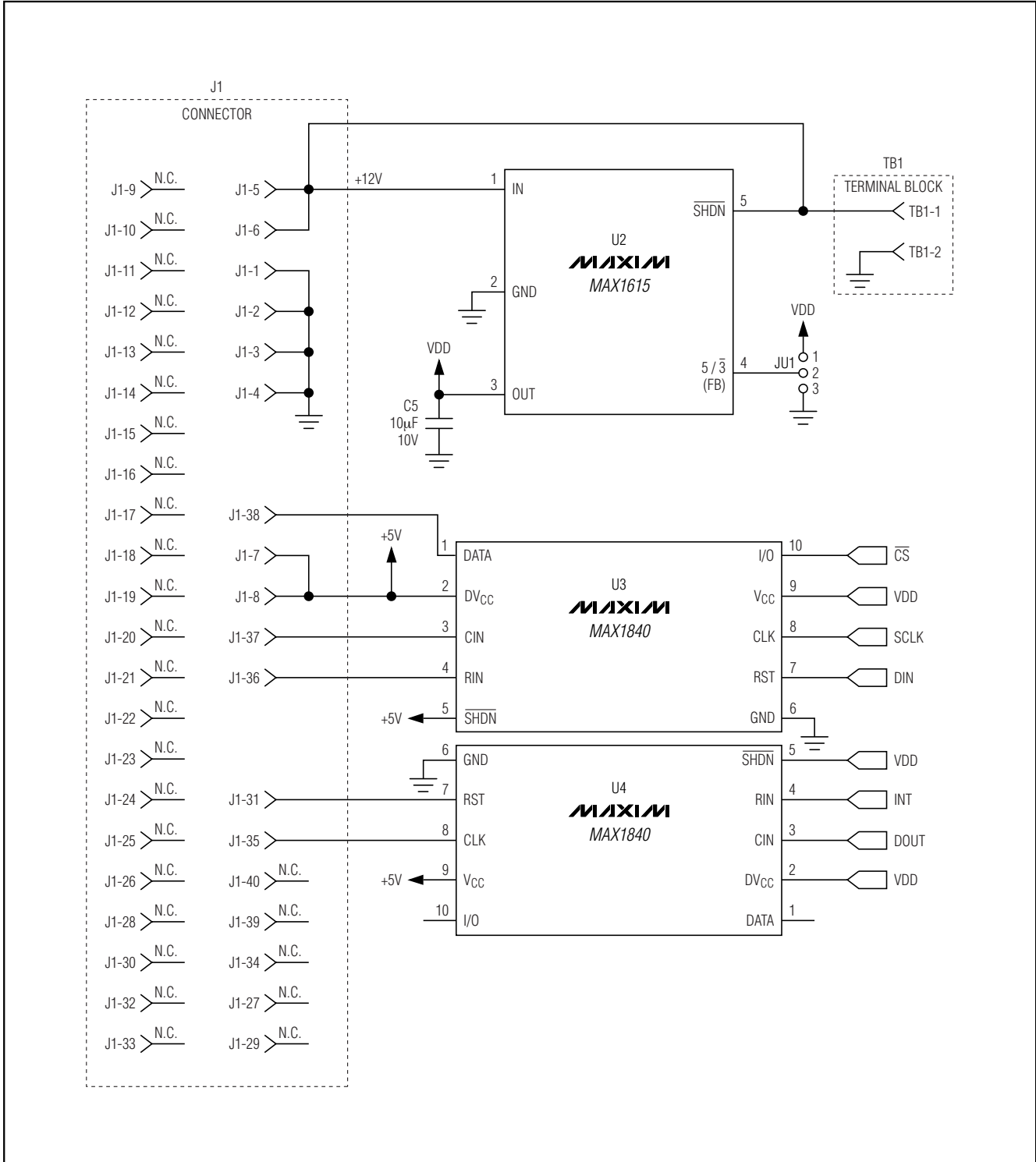
Figure 10b. MAX1254 EV Kit Schematic (Sheet 2 of 2)

MAXIM

Evaluate: MAX1254/MAX1253
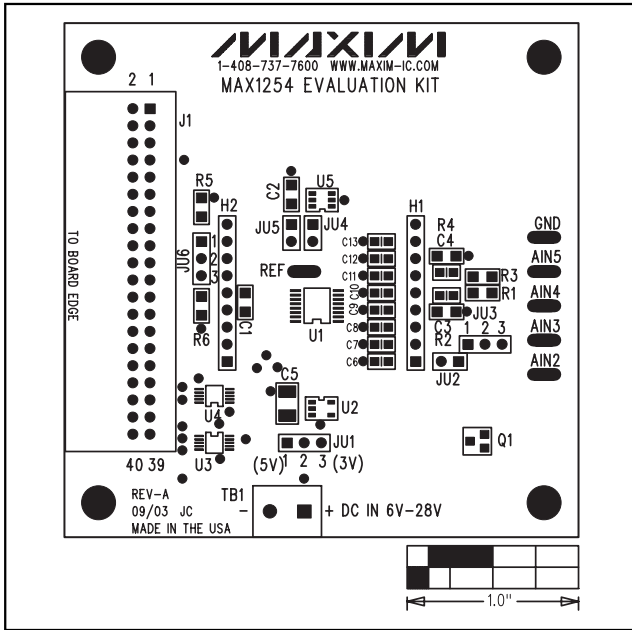
Figure 11. MAX1254 EV Kit Component Placement Guide—Component Side
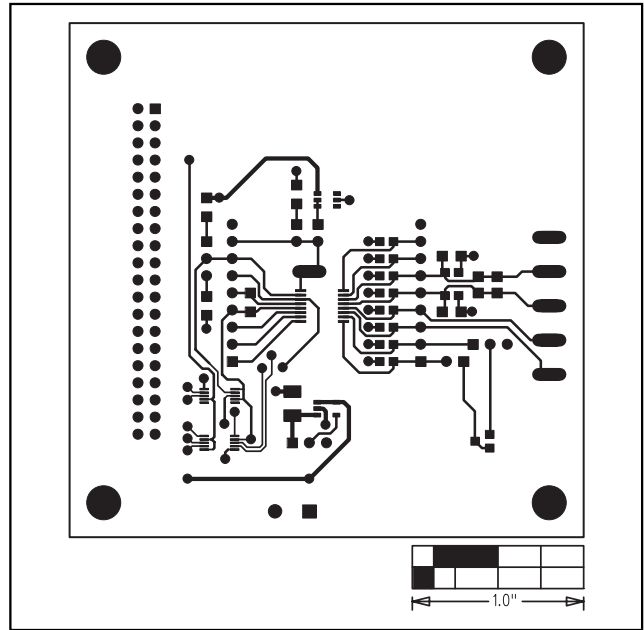
Figure 12. MAX1254 EV Kit PC Board Layout—Component Side
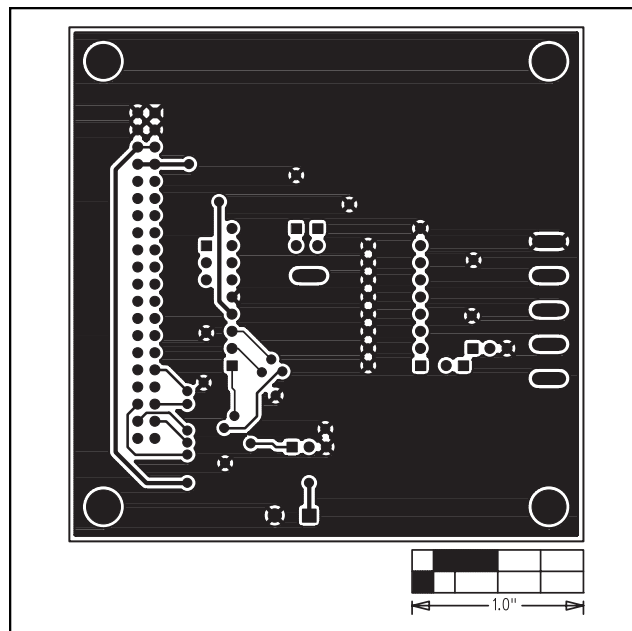
Figure 13. MAX1254 EV Kit PC Board Layout—Solder Side

```
// Drv1254.h
// MAX1254-specific driver.
// mku 09/16/2003
// (C) 2003 Maxim Integrated Products
// For use with Borland C++ Builder 3.0
//-----------------------------------------------------------------------
// Revision history:
// 09/16/2003: begin cleanup for publication
//-----------------------------------------------------------------------
#ifndef drv1254H
#define drv1254H
//-----------------------------------------------------------------------


//-----------------------------------------------------------------------
// 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[]);

//-------------------------------------------------
// MAX1254 command definitions
//
#define MAX1254_TRIGGER_CH          0x00
#define MAX1254_RD_ALARM            0x10
#define MAX1254_RD_DATA_CH          0x20
#define MAX1254_RD_DATA_ALL         0x30
#define MAX1254_RD_CONFIG_CH        0x40
#define MAX1254_RD_GLOBAL_CONFIG    0x50
#define MAX1254_RESERVED_0110       0x60
#define MAX1254_RESET               0x70
#define MAX1254_CLEAR_ALARM_CH      0x80
#define MAX1254_CLEAR_ALARM_ALL     0x90
#define MAX1254_WR_DATA_CH          0xA0
#define MAX1254_WR_DATA_ALL         0xB0
#define MAX1254_WR_CONFIG_CH        0xC0
#define MAX1254_WR_GLOBAL_CONFIG    0xD0
#define MAX1254_RESERVED_1110       0xE0
#define MAX1254_RESERVED_1111       0xF0


//-------------------------------------------------
// MAX1254 channel definitions
//
#define MAX1254_TEMP       0x00
#define MAX1254_VDD        0x01
#define MAX1254_AIN0       0x02
#define MAX1254_AIN1       0x03
#define MAX1254_AIN2       0x04
#define MAX1254_AIN3       0x05
#define MAX1254_AIN4       0x06
#define MAX1254_AIN5       0x07
#define MAX1254_AIN6       0x08
#define MAX1254_AIN7       0x09


//-------------------------------------------------
// MAX1254 Global Configuration: Channel Enable
//
#define MAX1254_GC_CHEN_TEMP      0x8000
#define MAX1254_GC_CHEN_VDD       0x4000
#define MAX1254_GC_CHEN_AIN0      0x2000
#define MAX1254_GC_CHEN_AIN1      0x1000
#define MAX1254_GC_CHEN_AIN2      0x0800
#define MAX1254_GC_CHEN_AIN3      0x0400
#define MAX1254_GC_CHEN_AIN4      0x0200
#define MAX1254_GC_CHEN_AIN5      0x0100
#define MAX1254_GC_CHEN_AIN6      0x0080
#define MAX1254_GC_CHEN_AIN7      0x0040
```

Listing 1 (Sheet 1 of 3)

MAXIM

```
//-------------------------------------------------
// MAX1254 Global Configuration: Setup Register
//
#define  MAX1254_GC_SETUP_DELAY_MASK        0xE0
#define  MAX1254_GC_SETUP_DELAY_0MS              0x00
#define  MAX1254_GC_SETUP_DELAY_001              0x20
#define  MAX1254_GC_SETUP_DELAY_010              0x40
#define  MAX1254_GC_SETUP_DELAY_011              0x60
#define  MAX1254_GC_SETUP_DELAY_100              0x80
#define  MAX1254_GC_SETUP_DELAY_101              0xA0
#define  MAX1254_GC_SETUP_DELAY_110              0xC0
#define  MAX1254_GC_SETUP_DELAY_111              0xE0
#define  MAX1254_GC_SETUP_INT_MASK          0x18
#define  MAX1254_GC_SETUP_INT_00                 0x00
#define  MAX1254_GC_SETUP_INT_01                 0x08
#define  MAX1254_GC_SETUP_INT_10                 0x10
#define  MAX1254_GC_SETUP_INT_11                 0x18
#define  MAX1254_GC_SETUP_AUTOSCAN          0x04
#define  MAX1254_GC_SETUP_REF_MASK          0x03
#define  MAX1254_GC_SETUP_REF_EXT                0x00
#define  MAX1254_GC_SETUP_REF_INT_AUTOPD         0x01
#define  MAX1254_GC_SETUP_REF_INT                0x02


//-------------------------------------------------
// MAX1254 Global Configuration: Input Configuration Register
//
#define  MAX1254_GC_INPUT01_MASK        0xE000
#define  MAX1254_GC_INPUT01_000                  0x0000
#define  MAX1254_GC_INPUT01_001                  0x2000
#define  MAX1254_GC_INPUT01_010                  0x4000
#define  MAX1254_GC_INPUT01_011                  0x6000
#define  MAX1254_GC_INPUT01_100                  0x8000
#define  MAX1254_GC_INPUT01_101                  0xA000
#define  MAX1254_GC_INPUT01_110                  0xC000
#define  MAX1254_GC_INPUT01_111                  0xE000
#define  MAX1254_GC_INPUT23_MASK        0x1C00
#define  MAX1254_GC_INPUT23_000                  0x0000
#define  MAX1254_GC_INPUT23_001                  0x0400
#define  MAX1254_GC_INPUT23_010                  0x0800
#define  MAX1254_GC_INPUT23_011                  0x0C00
#define  MAX1254_GC_INPUT23_100                  0x1000
#define  MAX1254_GC_INPUT23_101                  0x1400
#define  MAX1254_GC_INPUT23_110                  0x1800
#define  MAX1254_GC_INPUT23_111                  0x1C00
#define  MAX1254_GC_INPUT45_MASK        0x0380
#define  MAX1254_GC_INPUT45_000                  0x0000
#define  MAX1254_GC_INPUT45_001                  0x0080
#define  MAX1254_GC_INPUT45_010                  0x0100
#define  MAX1254_GC_INPUT45_011                  0x0180
#define  MAX1254_GC_INPUT45_100                  0x0200
#define  MAX1254_GC_INPUT45_101                  0x0280
#define  MAX1254_GC_INPUT45_110                  0x0300
#define  MAX1254_GC_INPUT45_111                  0x0380
#define  MAX1254_GC_INPUT67_MASK        0x0070
#define  MAX1254_GC_INPUT67_000                  0x0000
#define  MAX1254_GC_INPUT67_001                  0x0010
#define  MAX1254_GC_INPUT67_010                  0x0020
#define  MAX1254_GC_INPUT67_011                  0x0030
#define  MAX1254_GC_INPUT67_100                  0x0040
#define  MAX1254_GC_INPUT67_101                  0x0050
#define  MAX1254_GC_INPUT67_110                  0x0060
#define  MAX1254_GC_INPUT67_111                  0x0070


//-----------------------------------------------------------------------
class MAX1254
{
public:
    __fastcall MAX1254(void);

    // Reference voltage
    //
    double vref;

    // Channel identification
    //
    enum channel_id {
```

Listing 1 (Sheet 2 of 3)

Evaluate: MAX1254/MAX1253

```
        TEMP=0, VDD, AIN0, AIN1, AIN2, AIN3, AIN4, AIN5, AIN6, AIN7,
        num_channels
};

// Command to reset to power-on default settings
//
bool __fastcall Reset(void);

// Command to trigger a measurement on the selected channel
//
bool __fastcall Trigger(int channel_id);

// Channel Data Format
//
enum channel_format {
    RESERVED,                    /* skipped channel */
    SIGNAL_RETURN,               /* return path for differential signal pair */
    U12_VOLTS_DIV_2,         /* unsigned 12-bit voltage-divided-by-two */
    U12_VOLTS,                   /* unsigned 12-bit voltage */
    S11_VOLTS,                   /* signed 11-bit voltage */
    S8_CENTIGRADE_MUL_8,      /* signed 8-bit temperature (with 3 sub-lsbs) */
    num_formats
};
enum channel_format Channel_Format[num_channels];

// Recent measurement data for each channel
//
unsigned __int16 Channel_Data_uint16[num_channels];
bool __fastcall Read_Channel_Data(int channel_id);
bool __fastcall Write_Channel_Data(int channel_id);
bool __fastcall Read_All_Channel_Data(void);
bool __fastcall Write_All_Channel_Data(void);

const char* __fastcall Get_Channel_FormatString(int channel_id);
int __fastcall Get_Channel_int(int channel_id);
int __fastcall Get_Channel_int(int channel_id, unsigned __int16 code_u12);
double __fastcall Get_Channel_double(int channel_id, unsigned __int16 code_u12);

// Configuration settings for each channel
//
struct channel_config_t {
    unsigned __int16 UpperLimit;
    unsigned __int16 LowerLimit;
    unsigned __int8  AverageSize_FaultLimit;
};
 channel_config_t Channel_Config[num_channels];
bool __fastcall Read_Channel_Config(int channel_id);
bool __fastcall Write_Channel_Config(int channel_id);

// Global configuration settings
//
    struct {
    unsigned __int16 ChannelEnable;
    unsigned __int16 InputConfig;
    unsigned __int8  SetupReg;
} Global_Config;
bool __fastcall Read_Global_Config();
bool __fastcall Write_Global_Config();
bool __fastcall channel_is_enabled(int channel_id);

// Alarm fault status
//
unsigned __int8 Alarm_Reg_1;
unsigned __int8 Alarm_Reg_2;
unsigned __int8 Alarm_Reg_3;
bool __fastcall Read_Alarm();
bool __fastcall Clear_Alarm(int channel_id);
bool __fastcall Clear_All_Alarms();

};
//----------------------------------------------------------------------
#endif
```

Listing 1 (Sheet 3 of 3)

Evaluate: MAX1254/MAX1253

```
// Drv1254.cpp
// MAX1254-specific driver.
// mku 09/16/2003
// (C) 2003 Maxim Integrated Products
// For use with Borland C++ Builder 3.0
//-----------------------------------------------------------------------
// Revision history:
// 09/16/2003: cleanup for publication
//-----------------------------------------------------------------------

#include "drv1254.h"

//-----------------------------------------------------------------------
// To indicate failure using return value instead of C++ exception,
// define the preprocessor symbol THROW_EXCEPTIONS=0.
//
#ifndef THROW_EXCEPTIONS
#define THROW_EXCEPTIONS 1
#endif
//
// Functions that return a data value must indicate failure by either
// throwing a C++ exception, or by returning a value that could never happen.
#if THROW_EXCEPTIONS
#else
#define MAX1254_IMPOSSIBLE_DATA_VALUE 32000
#endif

//-----------------------------------------------------------------------
__fastcall MAX1254::MAX1254(void)
{
    vref = 2.500;

    for (int index = 0; index < num_channels; index++) {
        Channel_Format[index] = U12_VOLTS;
        Channel_Data_uint16[index] = 0;
        Channel_Config[index].UpperLimit = 0xFFFF;
        Channel_Config[index].LowerLimit = 0x0000;
        Channel_Config[index].AverageSize_FaultLimit = 0x00;
    }
    Channel_Format[TEMP] = S8_CENTIGRADE_MUL_8;
    Channel_Format[VDD] = U12_VOLTS_DIV_2;
    Alarm_Reg_1 = 0;
    Alarm_Reg_2 = 0;
    Alarm_Reg_3 = 0;
}
//-----------------------------------------------------------------------
bool __fastcall MAX1254::Reset(void)
{
    const unsigned __int8 mosi[] = { MAX1254_RESET };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    for (int index = 0; index < num_channels; index++) {
        Channel_Format[index] = U12_VOLTS;
        Channel_Data_uint16[index] = 0;
        Channel_Config[index].UpperLimit = 0xFFFF;
        Channel_Config[index].LowerLimit = 0x0000;
        Channel_Config[index].AverageSize_FaultLimit = 0x00;
    }
    Channel_Format[TEMP] = S8_CENTIGRADE_MUL_8;
    Channel_Format[VDD] = U12_VOLTS_DIV_2;
    Alarm_Reg_1 = 0;
    Alarm_Reg_2 = 0;
    Alarm_Reg_3 = 0;
    return result;
}
//-----------------------------------------------------------------------
bool __fastcall MAX1254::Trigger(int channel_id)
{
    if ((channel_id < 0) || (channel_id > num_channels)) {
#if THROW_EXCEPTIONS
        throw "illegal channel id in MAX1254::Trigger";
```

Listing 2 (Sheet 1 of 6)

Evaluate: MAX1254/MAX1253

```
#else
        return false;
#endif
    }
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1254_TRIGGER_CH+(channel_id&0x0F))
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    return result;
}
//-----------------------------------------------------------------------
bool __fastcall MAX1254::Read_Channel_Data(int channel_id)
{
    if ((channel_id < 0) || (channel_id > num_channels)) {
#if THROW_EXCEPTIONS
        throw "illegal channel id in MAX1254::Read_Channel_Data";
#else
        return false;
#endif
    }
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1254_RD_DATA_CH+(channel_id&0x0F)),
        0x00, 0x00
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    if (result) {
        Channel_Data_uint16[channel_id & 0x0F] =
            (miso_buf[1] * 0x100) + miso_buf[2];
    } else {
        // no channel data. Should we make something up?
    }
    return result;
}
//-----------------------------------------------------------------------
bool __fastcall MAX1254::Write_Channel_Data(int channel_id)
{
    if ((channel_id < 0) || (channel_id > num_channels)) {
#if THROW_EXCEPTIONS
        throw "illegal channel id in MAX1254::Write_Channel_Data";
#else
        return false;
#endif
    }
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1254_WR_DATA_CH+(channel_id&0x0F)),
        (unsigned __int8)(Channel_Data_uint16[channel_id] / 0x100),
        (unsigned __int8)(Channel_Data_uint16[channel_id] & 0x0FF),
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    return result;
}
//-----------------------------------------------------------------------
bool __fastcall MAX1254::Read_All_Channel_Data(void)
{
    const unsigned __int8 mosi[] = { MAX1254_RD_DATA_ALL,
        0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0
    }; // total of 21 bytes
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    if (result) {
        for (int index = 0; index < num_channels; index++) {
            Channel_Data_uint16[index] =
                (miso_buf[1+index*2] * 0x100) + miso_buf[2+index*2];
        }
    }
    return result;
}
//-----------------------------------------------------------------------
bool __fastcall MAX1254::Write_All_Channel_Data(void)
{
    const unsigned __int8 mosi[] = { MAX1254_WR_DATA_ALL,
```

Listing 2 (Sheet 2 of 6)

MAXIM

```
                    (unsigned __int8)(Channel_Data_uint16[0] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[0] & 0x0FF),
                    (unsigned __int8)(Channel_Data_uint16[1] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[1] & 0x0FF),
                    (unsigned __int8)(Channel_Data_uint16[2] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[2] & 0x0FF),
                    (unsigned __int8)(Channel_Data_uint16[3] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[3] & 0x0FF),
                    (unsigned __int8)(Channel_Data_uint16[4] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[4] & 0x0FF),
                    (unsigned __int8)(Channel_Data_uint16[5] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[5] & 0x0FF),
                    (unsigned __int8)(Channel_Data_uint16[6] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[6] & 0x0FF),
                    (unsigned __int8)(Channel_Data_uint16[7] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[7] & 0x0FF),
                    (unsigned __int8)(Channel_Data_uint16[8] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[8] & 0x0FF),
                    (unsigned __int8)(Channel_Data_uint16[9] / 0x100),
                    (unsigned __int8)(Channel_Data_uint16[9] & 0x0FF)
            }; // total of 21 bytes
            unsigned __int8 miso_buf[sizeof(mosi)];
            bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
            return result;
        }
    //-----------------------------------------------------------------------
    bool __fastcall MAX1254::Read_Channel_Config(int channel_id)
    {
            if ((channel_id < 0) || (channel_id > num_channels)) {
    #if THROW_EXCEPTIONS
                throw "illegal channel id in MAX1254::Read_Channel_Config";
    #else
                return false;
    #endif
            }
            const unsigned __int8 mosi[] = {
                (unsigned __int8)(MAX1254_RD_CONFIG_CH+(channel_id&0x0F)),
                0, 0, 0, 0, 0
            }; // total of 6 bytes
            unsigned __int8 miso_buf[sizeof(mosi)];
            bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
            if (result) {
                Channel_Config[channel_id].UpperLimit =
                    (miso_buf[1] * 0x100) + miso_buf[2];
                Channel_Config[channel_id].LowerLimit =
                    (miso_buf[3] * 0x100) + miso_buf[4];
                Channel_Config[channel_id].AverageSize_FaultLimit = miso_buf[5];
            }
            return result;
    }
    //-----------------------------------------------------------------------
    bool __fastcall MAX1254::Write_Channel_Config(int channel_id)
    {
            if ((channel_id < 0) || (channel_id > num_channels)) {
    #if THROW_EXCEPTIONS
                throw "illegal channel id in MAX1254::Write_Channel_Config";
    #else
                return false;
    #endif
            }
            unsigned __int8 mosi[] = {
                (unsigned __int8)(MAX1254_WR_CONFIG_CH+(channel_id&0x0F)),
                (unsigned __int8)(Channel_Config[channel_id].UpperLimit / 0x100),
                (unsigned __int8)(Channel_Config[channel_id].UpperLimit & 0x0FF),
                (unsigned __int8)(Channel_Config[channel_id].LowerLimit / 0x100),
                (unsigned __int8)(Channel_Config[channel_id].LowerLimit & 0x0FF),
                Channel_Config[channel_id].AverageSize_FaultLimit
            }; // total of 6 bytes
            unsigned __int8 miso_buf[sizeof(mosi)];
            bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
            return result;
    }
    //-----------------------------------------------------------------------
    bool __fastcall MAX1254::Read_Global_Config(void)
```

Listing 2 (Sheet 3 of 6)

Evaluate: MAX1254/MAX1253

```cpp
{
    const unsigned __int8 mosi[] = {
        MAX1254_RD_GLOBAL_CONFIG, 0, 0, 0, 0, 0
    }; // total of 6 bytes
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    if (result) {
        Global_Config.ChannelEnable = (miso_buf[1] * 0x100) + miso_buf[2];
        Global_Config.InputConfig = (miso_buf[3] * 0x100) + miso_buf[4];
        Global_Config.SetupReg = miso_buf[5];
    }
    return result;
}
//----------------------------------------------------------------------
bool __fastcall MAX1254::Write_Global_Config(void)
{
    unsigned __int8 mosi[] = { MAX1254_WR_GLOBAL_CONFIG,
        (unsigned __int8)(Global_Config.ChannelEnable / 0x100),
        (unsigned __int8)(Global_Config.ChannelEnable & 0x0FF),
        (unsigned __int8)(Global_Config.InputConfig / 0x100),
        (unsigned __int8)(Global_Config.InputConfig & 0x0FF),
        Global_Config.SetupReg
    }; // total of 6 bytes
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    return result;
}
//----------------------------------------------------------------------
bool __fastcall MAX1254::channel_is_enabled(int channel_id)
{
    switch(channel_id) {
    case TEMP:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_TEMP);
    case VDD:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_VDD);
    case AIN0:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_AIN0);
    case AIN1:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_AIN1);
    case AIN2:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_AIN2);
    case AIN3:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_AIN3);
    case AIN4:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_AIN4);
    case AIN5:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_AIN5);
    case AIN6:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_AIN6);
    case AIN7:
        return (Global_Config.ChannelEnable & MAX1254_GC_CHEN_AIN7);
    }
    return false;
}
//----------------------------------------------------------------------
bool __fastcall MAX1254::Read_Alarm(void)
{
    const unsigned __int8 mosi[] = { MAX1254_RD_ALARM, 0, 0, 0 };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    if (result) {
        Alarm_Reg_1 = miso_buf[1];
        Alarm_Reg_2 = miso_buf[2];
        Alarm_Reg_3 = miso_buf[3];
    }
    return result;
}
//----------------------------------------------------------------------
bool __fastcall MAX1254::Clear_Alarm(int channel_id)
{
    if ((channel_id < 0) || (channel_id > num_channels)) {
#if THROW_EXCEPTIONS
        throw "illegal channel id in MAX1254::Clear_Alarm";
#else
```

Listing 2 (Sheet 4 of 6)

MAXIM

```
            return false;
#endif
    }
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1254_CLEAR_ALARM_CH+(channel_id&0x0F))
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    return result;
}
//----------------------------------------------------------------------
bool __fastcall MAX1254::Clear_All_Alarms(void)
{
    const unsigned __int8 mosi[] = { MAX1254_CLEAR_ALARM_ALL };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer(sizeof(mosi), mosi, miso_buf);
    Alarm_Reg_1 = 0;
    Alarm_Reg_2 = 0;
    Alarm_Reg_3 = 0;
    return result;
}
//----------------------------------------------------------------------
const char* __fastcall MAX1254::Get_Channel_FormatString(int channel_id)
{
    if ((channel_id < 0) || (channel_id > num_channels)) {
#if THROW_EXCEPTIONS
        throw "illegal channel id in MAX1254::Get_Channel_Value";
#else
        return "illegal channel id in MAX1254::Get_Channel_Value";
#endif
    }
    switch(Channel_Format[channel_id]) {
    case S8_CENTIGRADE_MUL_8:       return "%1.3f C";
    case U12_VOLTS_DIV_2:           return "%1.4f V";
    case U12_VOLTS:                 return "%1.4f V";
    case S11_VOLTS:                 return "%1.4f V";
    case SIGNAL_RETURN:             return "signal return";
    }
    return "(%f?)";
}
//----------------------------------------------------------------------
int __fastcall MAX1254::Get_Channel_int(int channel_id)
{
    if ((channel_id < 0) || (channel_id > num_channels)) {
#if THROW_EXCEPTIONS
        throw "illegal channel id in MAX1254::Get_Channel_int";
#else
        return MAX1254_IMPOSSIBLE_DATA_VALUE;
#endif
    }
    switch(Channel_Format[channel_id]) {
    case S8_CENTIGRADE_MUL_8: {
            signed __int16 adc_code = Channel_Data_uint16[channel_id] / 16;
            if (adc_code > 2047) {
                adc_code = adc_code - 4096;
            }
            return adc_code;
        }
    case U12_VOLTS_DIV_2:
    case U12_VOLTS: {
            unsigned __int16 adc_code = Channel_Data_uint16[channel_id] / 16;
            return adc_code;
        }
    case S11_VOLTS: {
            signed __int16 adc_code = Channel_Data_uint16[channel_id] / 16;
            if (adc_code > 2047) {
                adc_code = adc_code - 4096;
            }
            return adc_code;
        }
    }
    return Channel_Data_uint16[channel_id] / 16;
}
//----------------------------------------------------------------------
int __fastcall MAX1254::Get_Channel_int(int channel_id, unsigned __int16 code_u12)
```

Listing 2 (Sheet 5 of 6)

Evaluate: MAX1254/MAX1253

![MAXIM logo] _____ 21

Evaluate: MAX1254/MAX1253

```
{
    if ((channel_id < 0) || (channel_id > num_channels)) {
#if THROW_EXCEPTIONS
        throw "illegal channel id in MAX1254::Get_Channel_int";
#else
        return MAX1254_IMPOSSIBLE_DATA_VALUE;
#endif
    }
    switch(Channel_Format[channel_id]) {
    case S8_CENTIGRADE_MUL_8: {
            signed __int16 adc_code = code_u12;
            if (adc_code > 2047) {
                adc_code = adc_code - 4096;
            }
            return adc_code;
        }
    case U12_VOLTS_DIV_2:
    case U12_VOLTS: {
            unsigned __int16 adc_code = code_u12;
            return adc_code;
        }
    case S11_VOLTS: {
            signed __int16 adc_code = code_u12;
            if (adc_code > 2047) {
                adc_code = adc_code - 4096;
            }
            return adc_code;
        }
    }
    return code_u12;
}
//-----------------------------------------------------------------------
double __fastcall MAX1254::Get_Channel_double(int channel_id, unsigned __int16
code_u12)
{
    if ((channel_id < 0) || (channel_id > num_channels)) {
#if THROW_EXCEPTIONS
        throw "illegal channel id in MAX1254::Get_Channel_double";
#else
        return MAX1254_IMPOSSIBLE_DATA_VALUE;
#endif
    }
    switch(Channel_Format[channel_id]) {
    case S8_CENTIGRADE_MUL_8: {
            signed __int16 adc_code = code_u12;
            if (adc_code > 2047) {
                adc_code = adc_code - 4096;
            }
            return adc_code / 8.0;
        }
    case MAX1254::U12_VOLTS_DIV_2:
    case MAX1254::U12_VOLTS: {
            unsigned __int16 adc_code = code_u12;
            double adc_voltage = adc_code * vref / 4096.0;
            if (Channel_Format[channel_id] == U12_VOLTS_DIV_2) {
                adc_voltage = adc_voltage * 2;
            }
            return adc_voltage;
        }
    case MAX1254::S11_VOLTS: {
            signed __int16 adc_code = code_u12;
            if (adc_code > 2047) {
                adc_code = adc_code - 4096;
            }
            double adc_voltage = adc_code * vref / 4096.0;
            return adc_voltage;
        }
    }
    return code_u12;
}
//-----------------------------------------------------------------------
```

Listing 2 (Sheet 6 of 6)