19-3324; Rev 1; 5/07

MAX1258 Evaluation Kit/Evaluation System

General Description

The MAX1258 evaluation system (EV system) consists of a MAX1258 evaluation kit (EV kit), Maxim 68HC16MODULE-DIP microcontroller (µC) module, and USBTO232. The MAX1258 offers multichannel, 12-bit, analog-to-digital converters (ADCs); a temperature sensor; octal 12-bit, digital-to-analog converters (DACs); and configurable general-purpose I/O ports (GPIOs). The evaluation software runs under Windows® 98/2000/XP, providing a handy user interface to exercise the features of the MAX1258.

Order the complete EV system (MAX1258EVC16) for a comprehensive evaluation of the MAX1258 using a PC. Order the EV kit (MAX1258EVKIT) if the 68HC16MODULE module has already been purchased with a previous Maxim EV system, or for custom use in other µC-based systems.

This system can also evaluate the MAX1057/MAX1058/MAX1257. See the Detailed Description of Hardware section for more details. Contact the factory for free samples of these products.

MAX1258 Stand-Alone EV Kit

The MAX1258 EV kit provides a proven printed-circuit board (PCB) layout to facilitate evaluation of the MAX1258. It must be interfaced to appropriate timing signals for proper operation. Power the on-board MAX1615 low dropout (LDO) by connecting a user-supplied 6VDC to 28VDC power supply and ground return to terminal block TB1. See Figure 7 and refer to the MAX1258 IC data sheet for timing requirements.

MAX1258 EV System

The MAX1258 EV system operates from a user-supplied 7VDC to 20VDC power supply. The evaluation software runs under Windows 98/2000/XP on a PC, interfacing to the EV system board through the computer's serial communications port (virtual COM port). See the Quick Start section for setup and operating instructions.

Features

- ♦ **Proven PCB Layout**
- ♦ **Complete Evaluation System**
- ♦ **Convenient On-Board Test Points**
- ♦ **Data-Logging Software**
- ♦ **Fully Assembled and Tested**
- ♦ **EV Kit Software Supports Windows 98/2000/XP with RS-232/COM Port**
- ♦ **EV Kit Software Supports Windows 2000/XP with USB Port**

Ordering Information

PART	TEMP RANGE	INTERFACE TYPE
MAX1258EVKIT	0°C to +70°C	User supplied
MAX1258EVC16	0°C to +70°C	Windows software

Note: *The MAX1258 evaluation software is designed for use with the complete evaluation system MAX1258EVC16 (includes 68HC16MODULE-DIP module, USBTO232, and MAX1258EVKIT.) If the MAX1258 evaluation software will not be used, the MAX1258EVKIT board can be purchased by itself, without the µC.*

Component Lists

MAX1258 EV System

PART	QTY	DESCRIPTION
MAX1258EVKIT	1	MAX1258 evaluation kit
68HC16MODULE-DIP	1	68HC16 µC module
USBTO232+	1	USB-to-COM port adapter board

+Denotes lead-free and RoHS-compliant.

Component Suppliers

SUPPLIER	PHONE	FAX	WEBSITE
Johanson Dielectric	818-364-9800	818-364-6100	www.johanson-caps.com
Murata Mfg. Co., Ltd.	770-436-1300	770-436-3030	www.murata.com
Panasonic Corp.	714-373-7366	714-737-7323	www.panasonic.com
Taiyo Yuden	800-348-2496	847-925-0899	www.t-yuden.com
TDK Corp.	847-803-6100	847-390-4405	www.component.tdk.com

Note: *Indicate that you are using the MAX1258 when contacting these component suppliers.*

Windows is a registered trademark of Microsoft Corporation.

Component Lists (continued)

MAX1258 EV Kit

DESIGNATION	QTY	DESCRIPTION
C1, C14, C19, C21	4	0.1µF ±10%, 16V X7R ceramic capacitors (0805) Murata GRM219R71C104K Johanson 250R15W104KV4Z TDK C2012X7R1C104K-0.85
C2–C13, C15, C16, C23	15	0.01µF ±10%, ceramic capacitors (0603) Taiyo Yuden UMK107B103KZ TDK C1608X7R1H103K Murata GRM188R71H103K
C17	1	470pF ±10%, 50V X7R ceramic capacitor (0603) Taiyo Yuden UMK107B471KZ TDK C1608X7R1H471K Murata GRM188R71H471K
C18	1	100pF ±5%, 50V C0G ceramic capacitor (0603) Murata GRM1885C1H101J Taiyo Yuden UMK107CH101JZ TDK C1608C0G1H101J
C20, C22	2	1µF ±10%, 10V X7R ceramic capacitors (0805) Murata GRM21BR71A105K Taiyo Yuden LMK212BJ105KG TDK C2012X7R1A105K
C24, C25, C26	3	10µF ±20%, 6.3V X5R ceramic capacitors (0805) TDK C2012X5R0J106M Taiyo Yuden JMK212BJ106MG Panasonic ECJ2FB0J106M

DESIGNATION	QTY	DESCRIPTION
H1–H4	4	12 pins
H5, H6, H7	3	2 x 4 dual-row header pins
H8	1	2 x 5 dual-row header pin
J1	1	2 x 20 right-angle socket
JU1, JU2	2	3 pins
JU4, JU5	2	2 pins
R18	1	100kΩ ±5% resistor (1206)
R19	1	10kΩ ±5% resistor (1206)
R1–R17, R22, R23	19	10Ω ±5% resistors (1206)
R20	1	510Ω ±5% resistor (1206)
R21	1	2kΩ ±5% resistor (1206)
TB1	1	Two-circuit terminal block
U1	1	MAX1258BETM (48-pin TQFN, 7mm x 7mm)
U2	1	MAX1615EUK-T (SOT23-5) (Top Mark: ABZD)
U3, U4	2	MAX1840EUB (µMAX®-10) or MAX1841EUB
—	6	Shunts (JU1: 1-2, JU2: 2-3, JU4: 1-2, JU5: 1-2, H8: 1-2, H8: 9-10)
—	1	PCB: MAX1258 Evaluation Kit

µMAX is a registered trademark of Maxim Integrated Products, Inc.

Quick Start

Recommended Equipment (USB Port/PC Connection Option)

Before beginning, the following equipment is needed:

- MAX1258 EV system:
  - MAX1258 EV kit
  - 68HC16MODULE-DIP
  - USBTO232 (USB cable included)
- DC power supply, +7V to +20VDC at 0.25A
- A user-supplied Windows 2000/XP computer with an available USB port to connect to the USBTO232

Note: In the following sections, software-related items are identified by bolding. Text in **bold** refers to items directly from the EV kit software. Text in **bold and underlined** refers to items directly from the Windows 2000/XP operating system.

Procedure

The MAX1284 EV kit is fully assembled and tested. Follow the steps below to verify board operation. **Caution: Do not turn on the power until all connections are completed.**

1) Visit the Maxim website (www.maxim-ic.com) to download the latest version of the USBTO232 User Guide. Follow the steps in the USBTO232 User Guide *Quick Start* section and return to step 2 of this *Quick Start* section when finished.
2) Ensure that jumper JU1 is in the 5V position, jumper JU2 is in the 1-2 position, and jumper JU5 is closed. Jumper JU4 should be open. Jumper JU3 is closed unless previously cut apart. See Tables 2–6.

3) Carefully connect the boards by aligning the 40-pin header of the MAX1258 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.
4) Connect a +7V to +20VDC power source to the µC module at the terminal block (J2) located next to the ON/OFF switch (SW1), along the top-edge of the µC module. Observe the polarity marked on the board.
5) Connect the USBTO232 board to the 68HC16MODULE-DIP module if you have not done so already.
6) The MAX1258 EV kit software should have already been downloaded and installed in the USBTO232 Quick Start.
7) Start the MAX1258 program by opening its icon in the **Start | Programs** menu.
8) Turn on the power supply and slide SW1 to the ON position on the 68HC16MODULE-DIP module. Press the **OK** button to automatically connect and download the file KIT1258.C16 to the module.
9) Apply an input signal to AIN0 and click **Perform Action**. Observe the readout on the screen. To perform the action repeatedly, check the **every 200ms** checkbox.
10) To view a graph of the measurements, pull down the **View** menu and click **Graph**.
11) Bring up the **DAC Outputs** tab, select action **1111 cccc cccc 001x Power On Selected Channels**, and click **Perform Action**.
12) Select action **1100 Write and Load OUT1-OUT8**, set OUT1 code to 2048, and click **Perform Action**. Observe that the voltage on OUT1 is now midscale (2.048V, assuming VREF = 4.096V).
13) Bring up the **GPIO Pins** tab. Set GPIOA0 to **High Output**, set GPIOB0 to **Low Output**, set GPIOC3 to **Input**, and click **Write Output Pins**. Observe that the A0 pin is now logic-high and the B0 pin is now logic-low.
14) Connect GPIO pin C3 to logic-high (A0) and click **Read Input Pins**. Observe that the software indicates that C3 is high.
15) Connect GPIO pin C3 to logic-low (B0) and click **Read Input Pins**. Observe that the software indicates that C3 is low.

Recommended Equipment (RS-232-to-COM Port/PC Connection Option)

Before beginning, the following equipment is needed:

- MAX1258 EV system:
  MAX1258 EV kit
  68HC16MODULE-DIP
- DC power supply, +7V to +20VDC at 0.25A
- A user-supplied Windows 98/2000/XP computer with an available serial (COM) port
- 9-pin I/O extension cable

Note: In the following sections, software-related items are identified by bolding. Text in **bold** refers to items directly from the EV kit software. Text in **bold and underlined** refers to items directly from the Windows 98/2000/XP operating system.

Procedure

The MAX1284 EV kit is fully assembled and tested. Follow the steps below to verify board operation. **Caution: Do not turn on the power until all connections are completed.**

1) Visit the Maxim website (www.maxim-ic.com/evkit-software) to download the latest version of the EV kit software. Save the EV kit software to a temporary folder and uncompress the file (if it is a .zip file).
2) Install the MAX1258 EV kit software on your computer by running the INSTALL.EXE program. The program files are copied and icons are created for them in the Windows **Start | Programs** menu.
3) Ensure that jumper JU1 is in the 5V position, jumper JU2 is in the 1-2 position, and jumper JU5 is closed. Jumper JU4 should be open. Jumper JU3 is closed unless previously cut apart. See Tables 2–6.
4) Carefully connect the boards by aligning the 40-pin header of the MAX1258 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.
5) Connect a +7V to +20VDC power source to the µC module at the terminal block (J2) located next to the ON/OFF switch (SW1), along the top-edge of the µC module. Observe the polarity marked on the board.
6) 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.

7) Start the MAX1258 program by opening its icon in the **Start | Programs** menu.
8) Turn on the power supply and slide SW1 to the ON position on the 68HC16MODULE-DIP module. Press the **OK** button to automatically connect and download the file KIT1258.C16 to the module.
9) Apply an input signal to AIN0 and click **Perform Action**. Observe the readout on the screen. To perform the action repeatedly, check the **every 200ms** checkbox.
10) To view a graph of the measurements, pull down the **View** menu and click **Graph**.
11) Bring up the **DAC Outputs** tab, select action **1111 cccc cccc 001x Power On Selected Channels**, and click **Perform Action**.
12) Select action **1100 Write and Load OUT1-OUT8**, set OUT1 code to 2048, and click **Perform Action**. Observe that the voltage on OUT1 is now midscale (2.048V, assuming VREF = 4.096V).
13) Bring up the **GPIO Pins** tab. Set GPIOA0 to **High Output**, set GPIOB0 to **Low Output**, set GPIOC3 to **Input**, and click **Write Output Pins**. Observe that the A0 pin is now logic-high and the B0 pin is now logic-low.
14) Connect GPIO pin C3 to logic-high (A0) and click **Read Input Pins**. Observe that the software indicates that C3 is high.
15) Connect GPIO pin C3 to logic low (B0) and click **Read Input Pins**. Observe that the software indicates that C3 is low.

Detailed Description of Software

The main window of the evaluation software configures the data converter and measures the analog inputs. Under **Action**, select a scanning sequence, repetitive conversions, or a single conversion. Each selected channel's measurement results are displayed in the corresponding **Measurement Results** field.

The **Action** setting **read single channel repeatedly** requires making a selection under **Repetition** to determine how many times the selected channel should be measured.

Use **Averaging** to summarize a series of measurements results on each selected channel as an arithmetic mean value. **Repetition** can be used together with **Averaging** to summarize a large number of measurement results as a small number of sample means.

The **Low Level Interface Details** panel shows the most recent low-level register write. A summary of what has been written to each register is available under the **Low-level registers** tab.

The AIN14 and AIN15 channels are automatically skipped if their alternate functions are selected in the setup register. The evaluation software updates its display to show or hide these channels whenever the alternate functions are enabled or disabled.

The **Setup** tab configures the alternate functions for the AIN14 and AIN15 pins, and also configures adjacent channels as differential input pairs.

The **Low-level registers** tab summarizes the commands that create the active configuration. The **Reset All Registers** checkbox resets these software-shadowed register values and sends the reset command to the MAX1258, optionally configuring slow mode and bandgap mode.

DAC Outputs

The **DAC Outputs** tab in the main window controls the analog output pins.

To power DAC channels prior to use, select action **1111 cccc cccc 001x Power On Selected Channels**. Verify that the checkboxes are set and click **Perform Action**. Jumper JU2 defines the initial states of the DAC output pins.

To reset all DAC outputs to zero, select action **0001 0... Reset all DACs to 000 (zero scale)** and click **Perform Action**.

To reset all DAC outputs to full scale, select action **0001 1... Reset all DACs to FFF (full scale)** and click **Perform Action**.

Note: At power-on the DAC expects an external reference on REF1. Refer to the MAX1258 IC data sheet.

To write and load multiple DAC channels, select action **1100 Write and Load OUT1-OUT8**, type the desired output code value (0 to 4095) into the **OUT1** edit field, and click **Perform Action**. The voltage on the OUT1–OUT8 pins immediately changes to the new value.

To write a single DAC output (for example OUT5), select action **0110 Write OUT5**. Type the desired output code value (0 to 4095) into the **OUT5** edit field and click **Perform Action**. Unless jumper JU5 is closed (asserting the $\overline{\text{LDAC}}$ pin low), a separate load command is required to update the pin voltage from the input register. Load OUT5 by selecting action **1110 cccc cccc xxxx Load**

Selected Channels, checking the Load checkbox for channel 5, and clicking **Perform Action**.

GPIO Pins

The main window's **GPIO Pins** tab configures, writes, and reads the general-purpose digital input/output pins.

Each GPIO pin has a drop-down combo box that configures **High Output**, **Low Output**, **Input**, or **open-drain pull-down** modes for that pin. Select output mode and click **Write Output Pins**. Read pins by clicking **Read Input Pins**.

Sampling

Measurement data can be sampled in external clock mode. From the **Setup** tab, set **Clock Mode** to **0111xxxx ext clock**. Then, return to the **Measurement** tab and click **Get Samples**.

Graph Window

To view recently measured data, drop down the **View** menu and choose **Graph**. Data can be viewed as a time sequence plot, a histogram plot, or as a table of raw numbers (see Figure 6). See Table 1 for available graph commands.

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: $\text{Mean} = \frac{\Sigma(x)}{n}$

TOOL	FUNCTION
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: $\text{RMS} = \sqrt{\frac{\Sigma\left(x^2\right)}{n}}$
	Channel 0 enable.
	Channel 1 enable.
	Channel 2 enable.
	Channel 3 enable.
	Channel 4 enable.
	Channel 5 enable.
	Channel 6 enable.
	Channel 7 enable.
	Channel 8 enable.
	Channel 9 enable.
	Channel 10 enable.
	Channel 11 enable.
	Channel 12 enable.
	Channel 13 enable.
	Channel 14 enable.
	Channel 15 enable.
	Channel 16 enable (temperature).

Table 2. Jumper JU1 (V_{DD} Voltage Selection)

SHUNT POSITION	V_{DD} VOLTAGE	FUNCTION
1-2*	5V	Normal operation with U1 = MAX1058 or MAX1258.
2-3	3V	Normal operation with U1 = MAX1057 or MAX1257.
Open	Unspecified	Do not operate kit with JU1 open.

**Default configuration.*

Table 3. Jumper JU2 (RES_SEL)

SHUNT POSITION	RES_SEL STATE	FUNCTION
1-2	High	When power is first applied, the DAC outputs are pulled to REF1 through internal 100kΩ resistors. All DAC input registers are set to 0xFFF.
2-3*	Low	When power is first applied, the DAC outputs are pulled to AGND through internal 100kΩ resistors. All DAC input registers are set to 0x000.
Open	Undriven	Do not power the kit with JU2 open.

**Default configuration.*

Table 4. Optional Jumper JU3 (AIN15 Alternate Function)

JU3 STATE	U1 PIN 1 CONNECTION	FUNCTION
Closed*	Connected to µC module J1 pin 29 (through level shifter).	U1 pin 1 = $\overline{CNVST}$ conversion start command. Leave AIN15 pad unconnected.
Open	Connected to AIN15 pad.	U1 pin 1 = AIN15 analog input. Connect signal source to AIN15 pad.

**Default configuration.*

Diagnostics Window

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

Table 5. Jumper JU4 (REF1 Bypass)

SHUNT POSITION	REF1 BYPASS CAPACITOR	FUNCTION
Open	User-provided	Leave JU4 open when using the internal reference.
Closed*	C14 bypasses REF1	Close JU4 when using an external reference.

**Default configuration.*

Table 6. Jumper JU5 ($\overline{LDAC}$)

SHUNT POSITION	$\overline{LDAC}$ STATE	FUNCTION
Open	High	Update DAC outputs by sending command through SPI.
Closed*	Low	DAC input registers are transferred to DAC output registers.

**Default configuration.*

Detailed Description of Hardware

The MAX1258 device under test (U1) offers a multichannel 12-bit ADC, a temperature sensor, an octal 12-bit DAC, and configurable GPIOs. Resistors R1–R16 and capacitors C1–C16 form single-pole lowpass antialiasing filters for each input. Capacitor C17 provides power-supply bypassing for U1. See Figure 7 and refer to the MAX1258 IC data sheet.

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 MAX1257 with the 5V µC.

Evaluating the MAX1257

The MAX1257 is the 3V version of the MAX1258. Request a free sample of the MAX1257BETM. Replace U1 with a MAX1257 and move the JU1 shunt to the 3V position. In the software's **Options** menu, select **reference = 2.500V**.

Evaluating the MAX1057

The MAX1057 is the 3V, 10-bit version of the MAX1257. Request a free sample of the MAX1057BETM. Replace U1 with a MAX1057 and move the JU1 shunt to the 3V position. In the software's **Options** menu, select **reference = 2.500V**.

The evaluation software expects 12 bits of data, but the MAX1057 only provides 10 bits of meaningful data. Because the most significant bits (MSBs) are aligned, the measurement code numbers reported by the soft-

ware are four times the actual measurement code number. Reconstructed voltage values are unaffected. Adjust the graph window by selecting its **Options** menu and setting **Sub-LSBs** to 2.

Evaluating the MAX1058

The MAX1058 is the 10-bit version of the MAX1258. Request a free sample of the MAX1058BETM. Replace U1 with a MAX1058 and move the JU1 shunt to the 5V position.

The evaluation software expects 12 bits of data, but the MAX1058 only provides 10 bits of meaningful data. Because the MSBs are aligned, the measurement code numbers reported by the software are four times the actual measurement code number. Reconstructed voltage values are unaffected. Adjust the graph window by selecting its **Options** menu and setting **Sub-LSBs** to 2.

Using an External Reference

All ADCs or DACs need a reference to perform conversions. A single-ended external reference can be applied for the MAX1258 DAC while using the internal reference for the ADC. This is the default mode. With the power off, connect the DAC external reference to REF1. Then power the system and run the evaluation software. Under the **Setup** tab, set the **Reference Input** to **01xx10xx Pin 48=AIN14**, **ADCREF=Internal**, **DACREF=REF1**. Connect on-board REF1 bypass capacitor C14 by installing a shunt on JU4. Whenever the software calculates a voltage that corresponds to a DAC code value, that calculation depends upon the user-provided **REF1 pin voltage** value.

To use the internal reference for both the ADC and the DAC, bring up the **Setup** tab and set **Reference Input** to **01xx00xx Pin 48=AIN14**, **ADCREF=Internal**, **DACREF=Internal**. Leave JU4 open when using the internal reference.

Independent single-ended references can be applied for both the DAC and the ADC. With the power off, connect the DAC external reference to REF1 and connect the ADC external reference to REF2. Then power the system and run the evaluation software. Under the **Setup** tab, set the **Reference Input** to **01xx01xx Pin 48=REF2**, **ADCREF=REF2**, **DACREF=REF1**. Connect on-board REF1 bypass capacitor C14 by installing a shunt on JU4. Whenever the software calculates a voltage that corresponds to a DAC code value, that calculation depends upon the user-provided **REF1 pin voltage** value. Whenever the software calculates a voltage that corresponds to an ADC code value, that calculation depends upon the user-provided **REF2 pin voltage** value.

A single-ended external reference can be applied for the DAC while a differential external reference is applied to the ADC. With the power off, connect the DAC external reference to REF1, and connect the ADC external reference between REF1 and REF2 such that REF1 > REF2. Then power the system and run the evaluation software. Under the **Setup** tab, set the **Reference Input** to **01xx11xx Pin 48=REF2**, **ADCREF=REF1-REF2**, **DACREF=REF1**. Connect on-board REF1 bypass capacitor C14 by installing a shunt on JU4. Whenever the software calculates a voltage that corresponds to a DAC code value, that calculation depends upon the user-provided **REF1 pin voltage** and **REF2 pin voltage** values.

Troubleshooting

Problem: No output measurement. System seems to report zero voltage or fails to make a measurement.

Solution: Check V_{DD} supply voltage. Check the reference voltage using a digital voltmeter. Use an oscilloscope to verify that the conversion-start signal is being strobed.

Problem: Measurements are erratic, unstable, or inaccurate.

Solution: Check the reference voltage using a digital voltmeter. Use an oscilloscope to check for noise. When probing for noise, keep the oscilloscope ground return lead as short as possible, preferably less than 0.5in (10mm).

Problem: Unacceptable errors when measuring a transducer.

Solution: Although most signal sources can be connected directly to the analog inputs of the MAX1258, some high-impedance signal sources (greater than 300Ω) may require an input buffer. Check for settling errors by increasing the acquisition time (internal clock mode 01). If necessary, use a MAX4430 to buffer high-impedance signal sources. Refer to the MAX1258 IC data sheet.

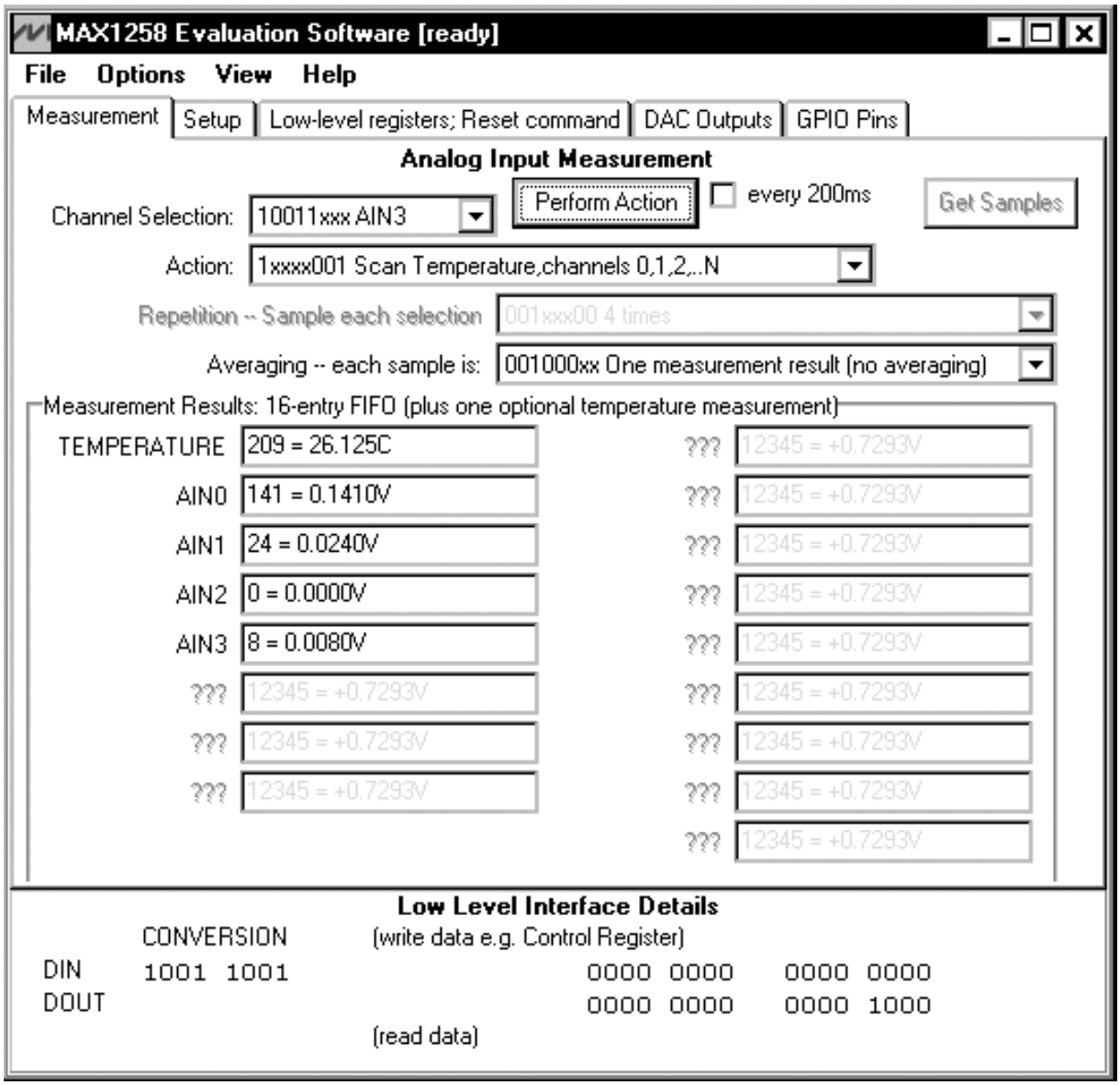

Figure 1. MAX1258 Evaluation Software's Main Window—Configures the Data Converter and Measures the Analog Inputs

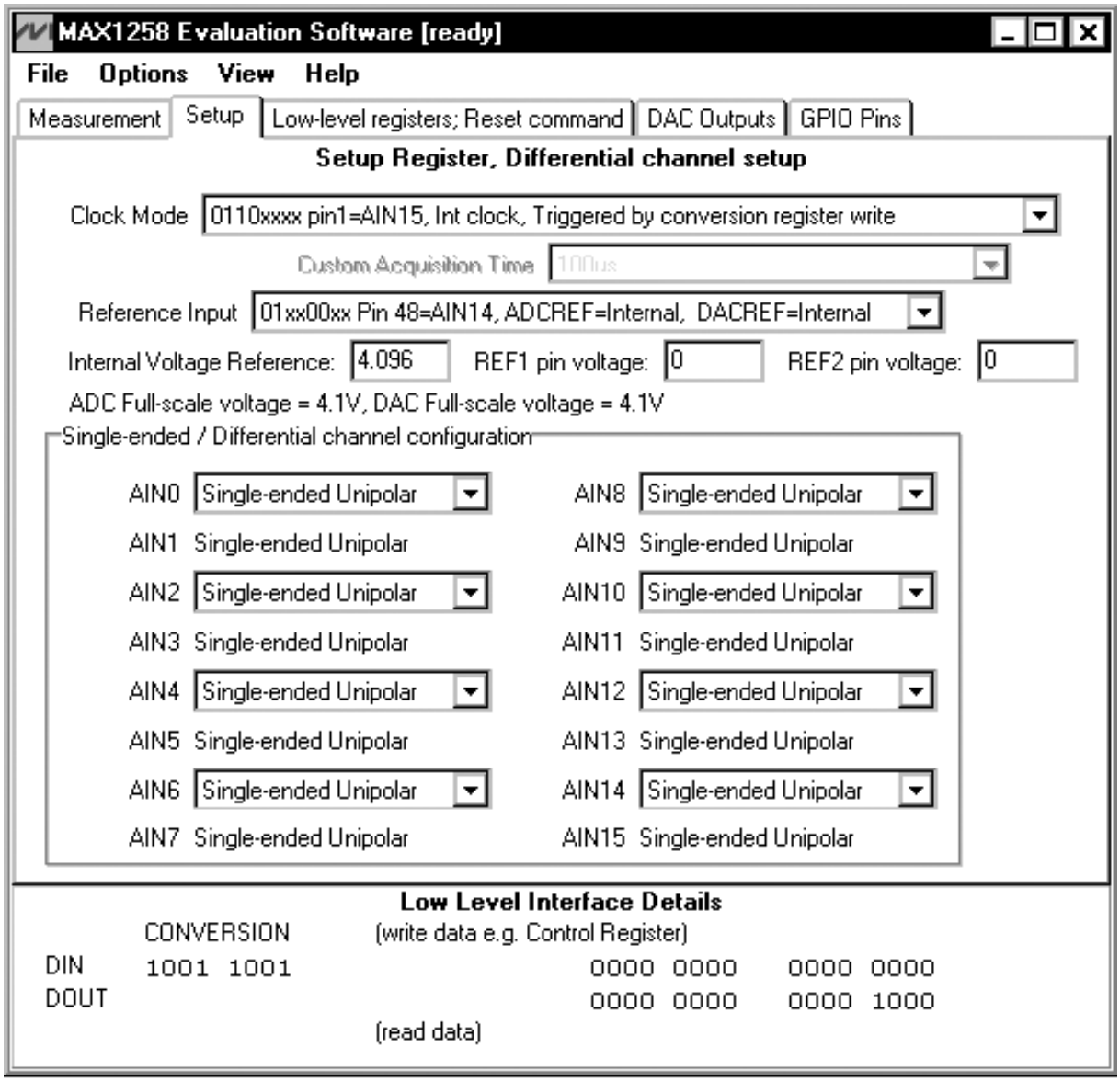

Figure 2. Setup Tab of Main Window—Configures the Alternate Functions for AIN14 and AIN15, and also Configures Adjacent Channels as Differential Input Pairs

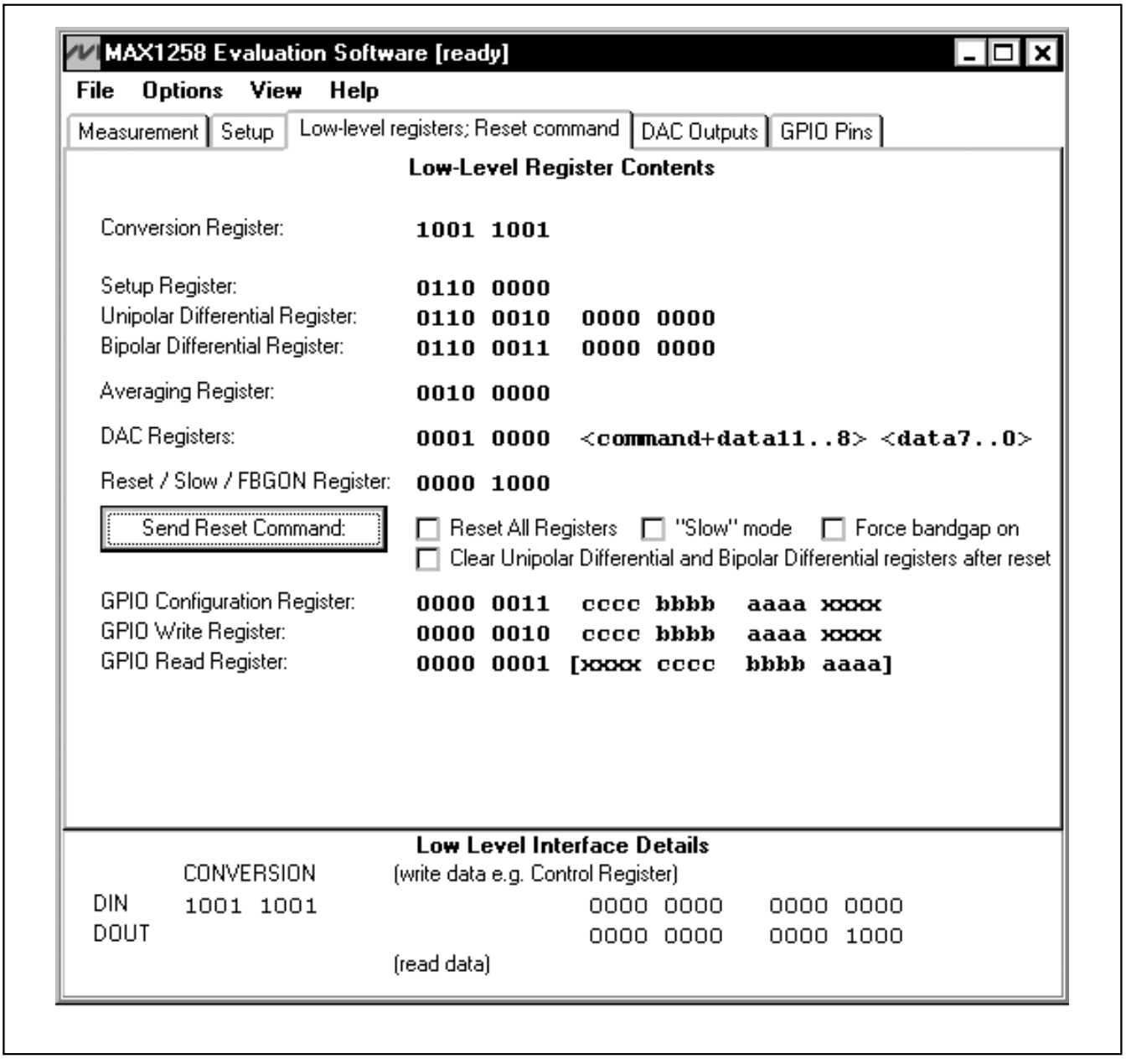

Figure 3. Low-Level Registers Tab of Main Window—Summarizes the Commands that Create the Active Configuration

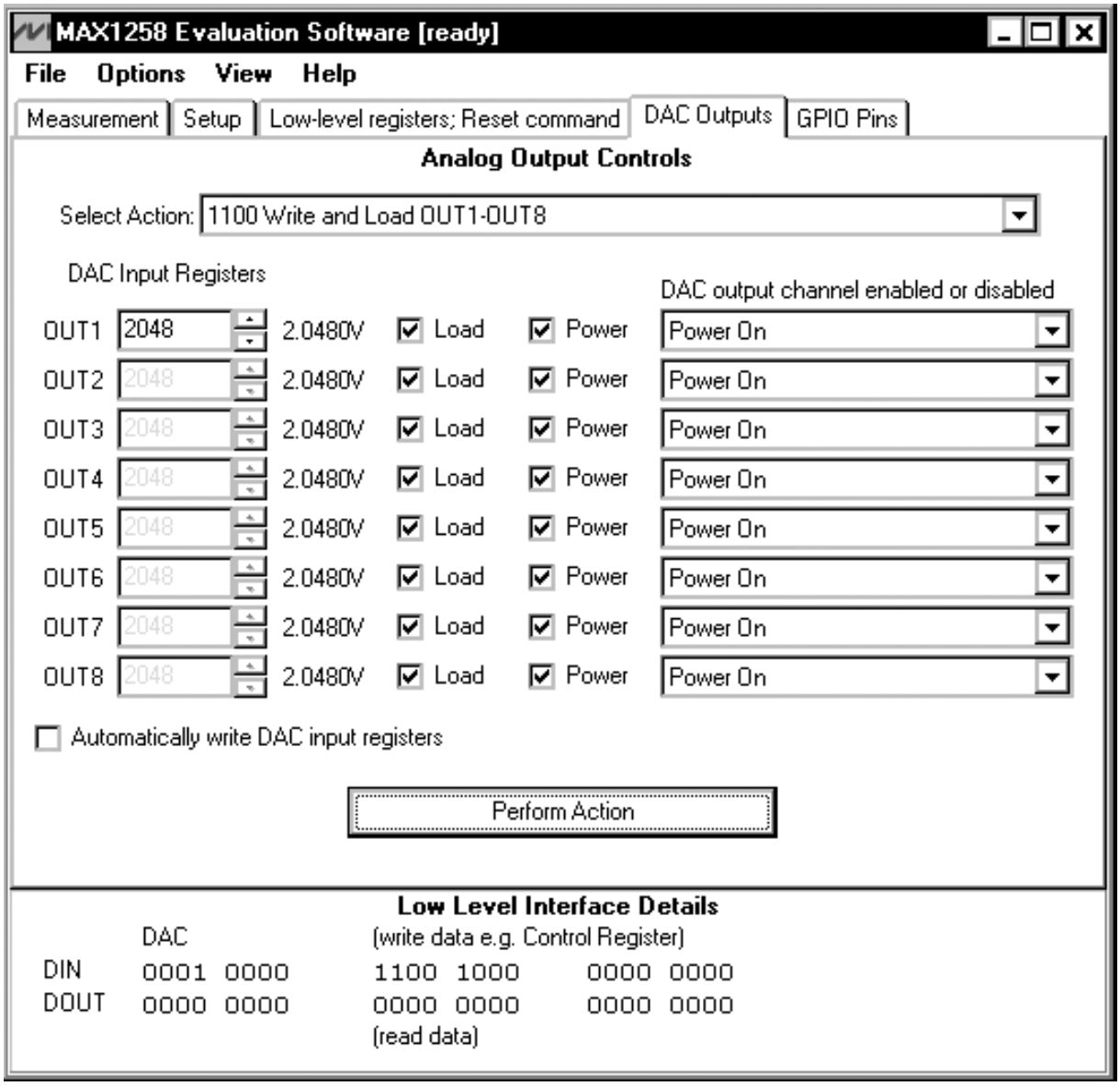

Figure 4. DAC Outputs Tab of Main Window—Controls the Analog Output Pins

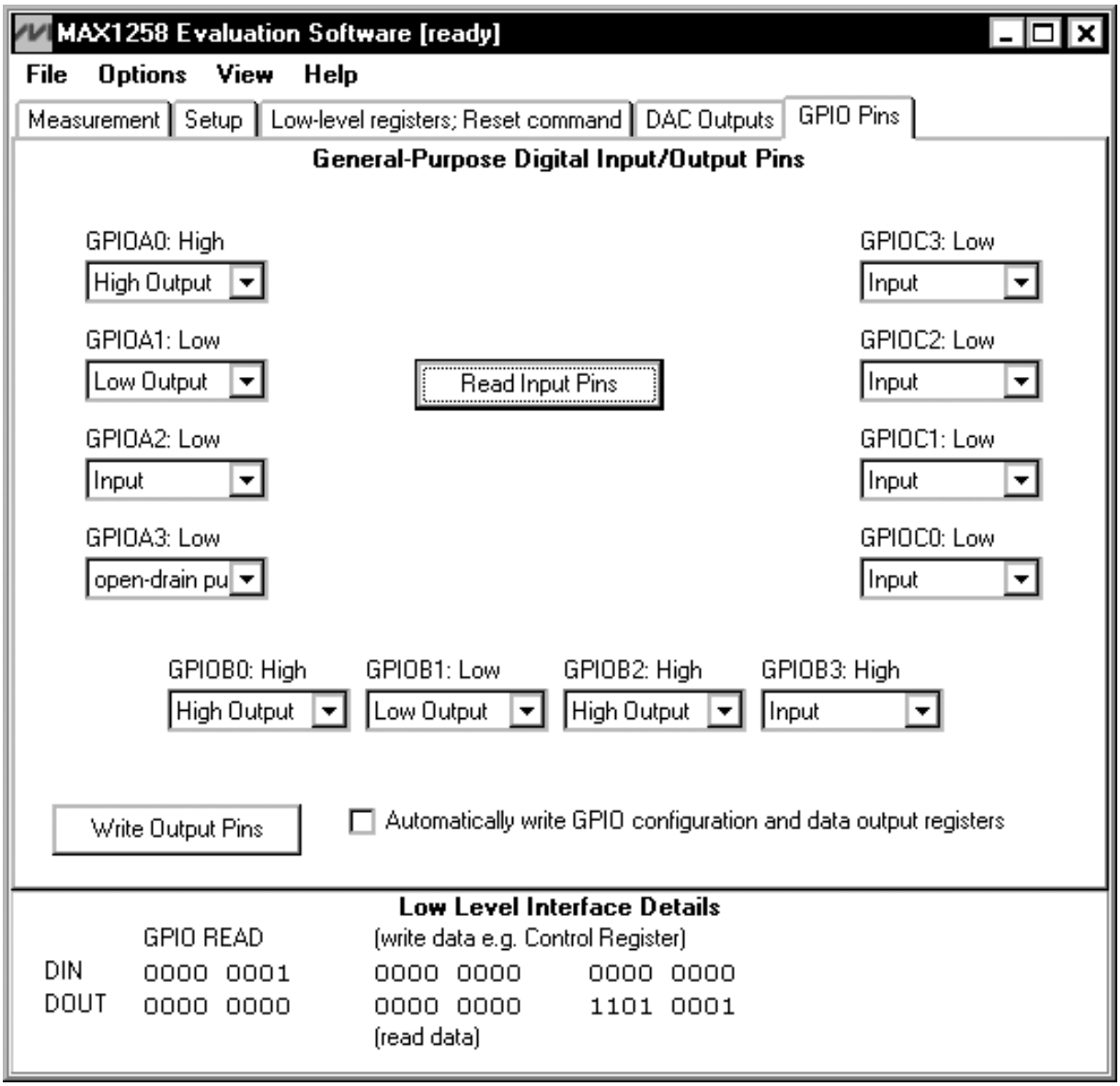

Figure 5. GPIO Pins Tab—Configures, Writes, and Reads the Digital GPIO Pins

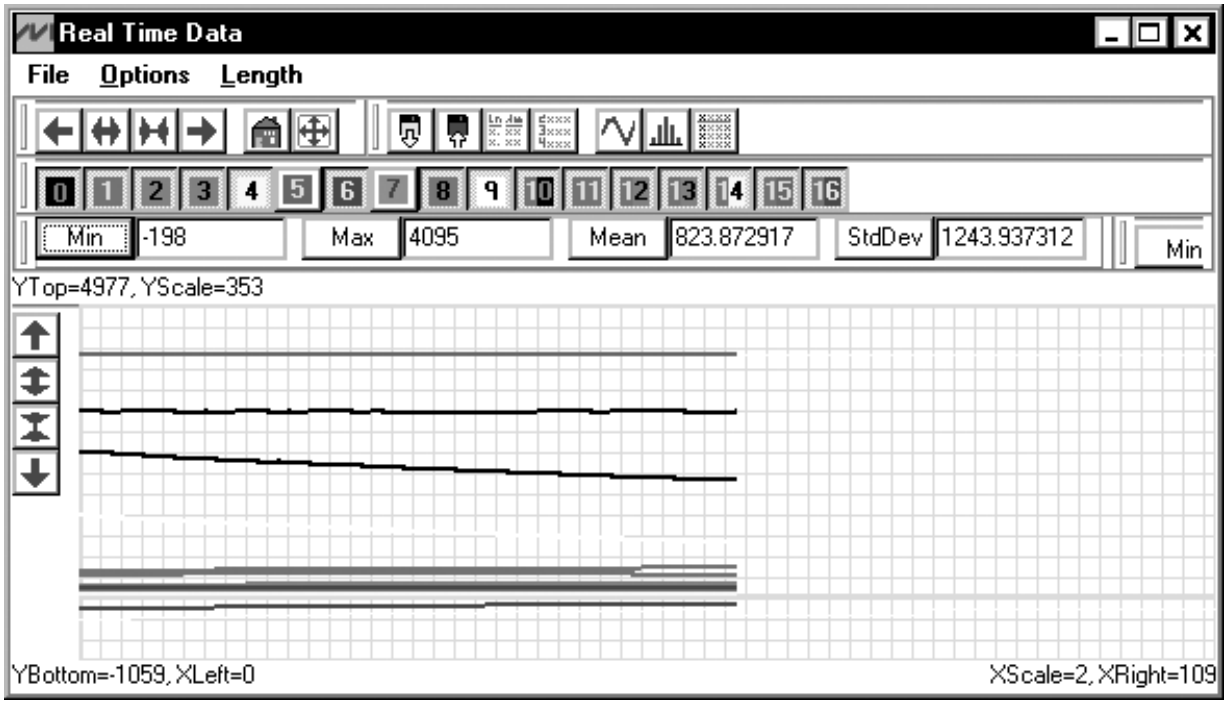

Figure 6. Real-Time Data and Sampled Data Graphs—Display Data as a Time Sequence, Histogram, or Table

MAX1258 Evaluation Kit/Evaluation System

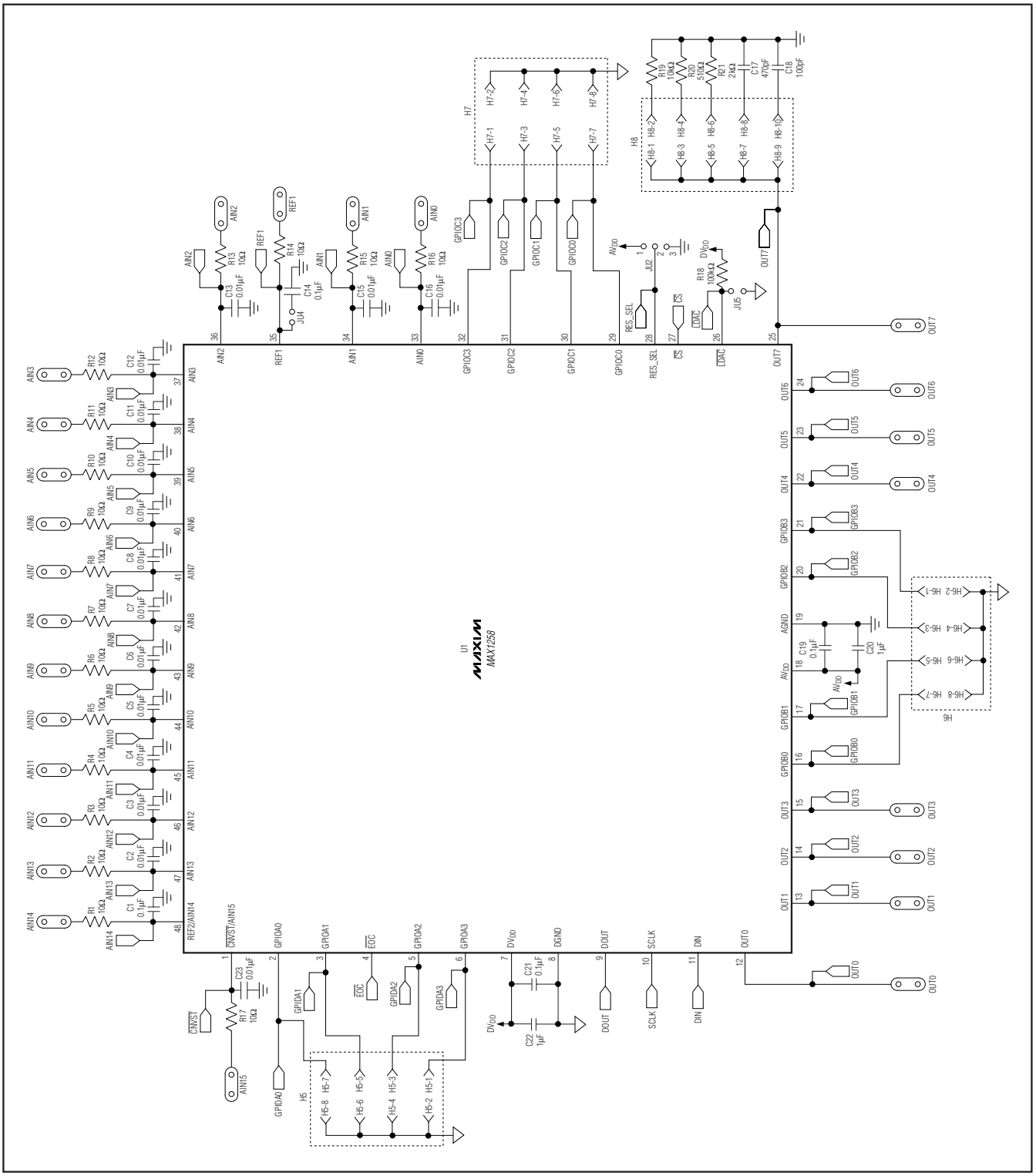

Figure 7a. MAX1258 EV Kit Schematic (Sheet 1 of 2)

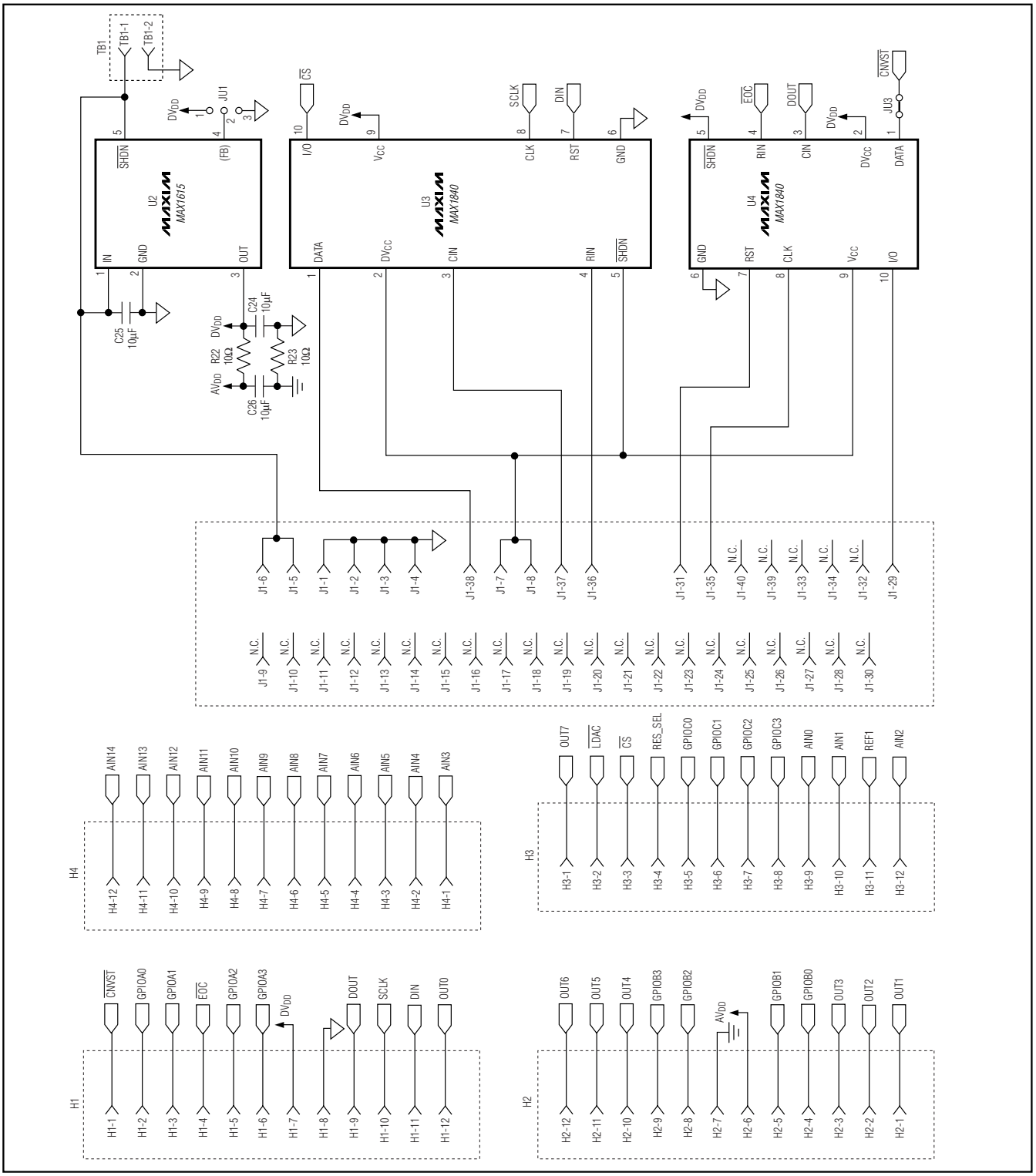

Figure 7b. MAX1258 EV Kit Schematic (Sheet 2 of 2)

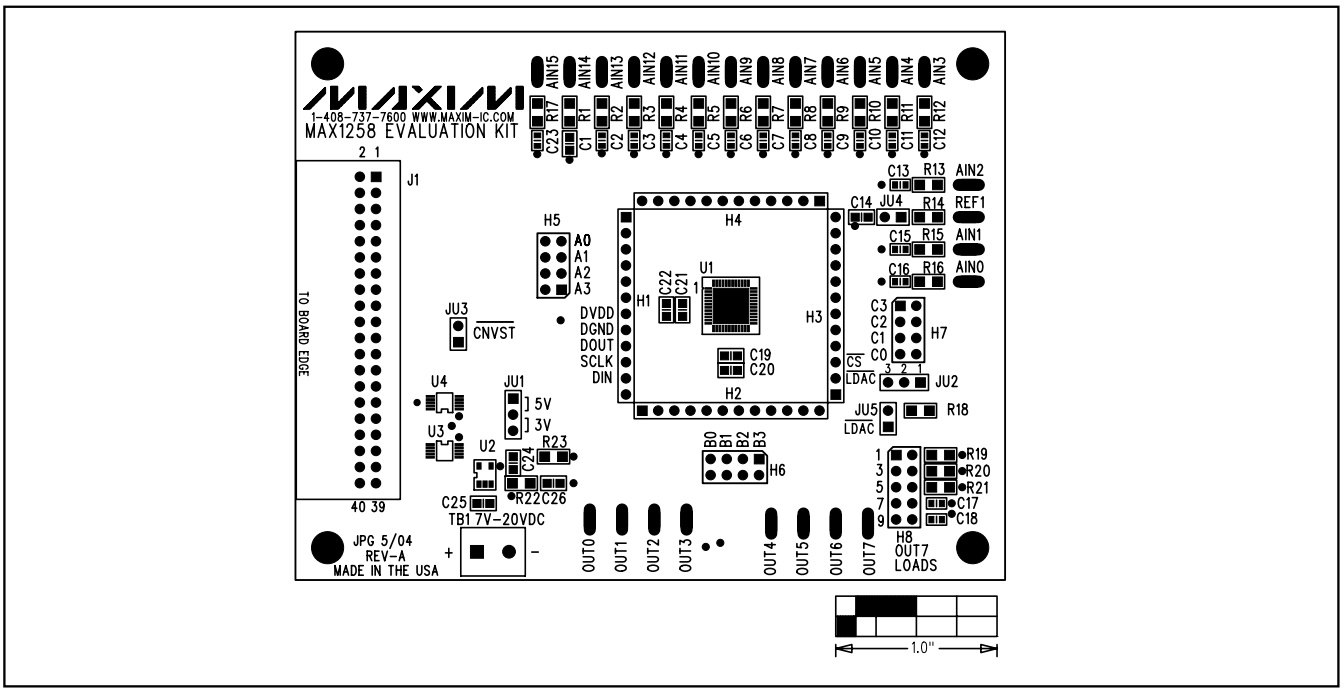

Figure 8. MAX1258 EV Kit Component Placement Guide—Component Side

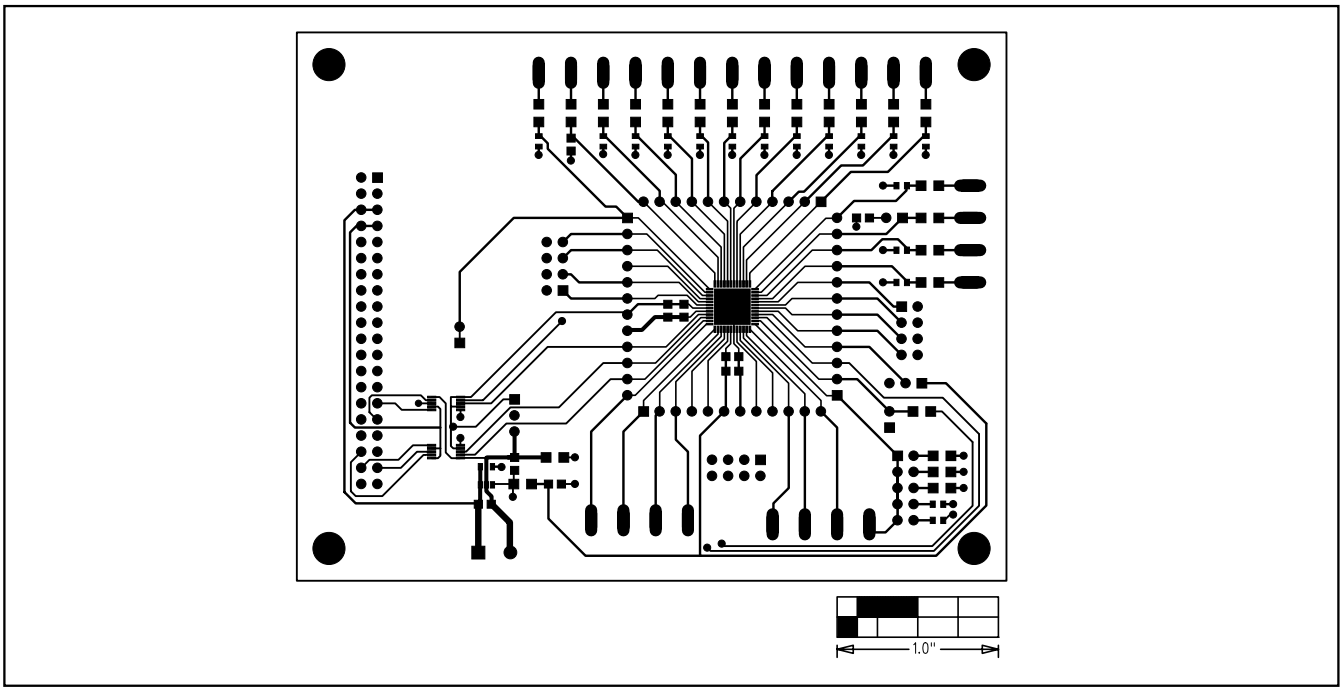

Figure 9. MAX1258 EV Kit PCB Layout—Component Side

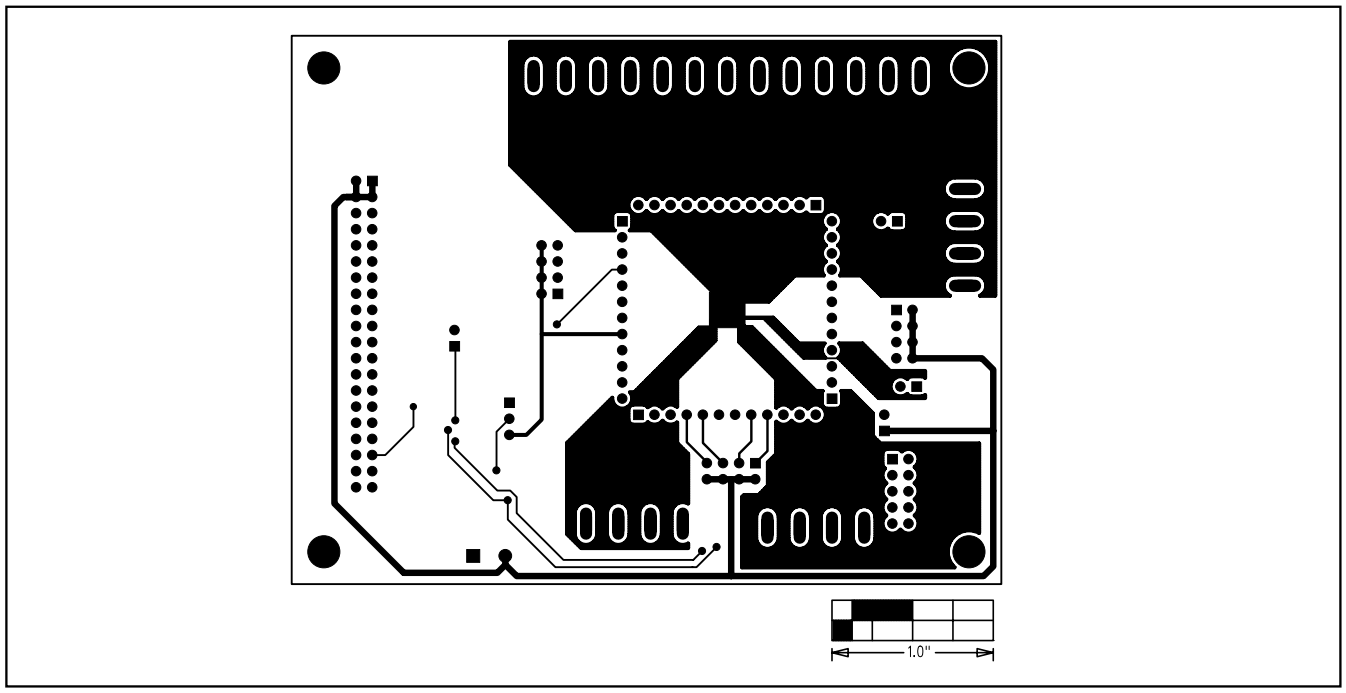

Figure 10. MAX1258 EV Kit PCB Layout—Solder Side

MAX1258 EV kit Listing 1 06/01/04 1
MAX1258EV listing1

```
// Drv1258.h
// MAX1258-specific driver.
// mku 04/07/2004
// (C) 2004 Maxim Integrated Products
//---------------------------------------------------
// Revision history (latest at top):
// =======================================
// __/__/2004: Initial Release as MAX1258 Version 1.0
// =======================================
//-------------------------------------------------------------------------
#ifndef DRV1258H
#define DRV1258H
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// 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
**
** 04/07/2004: master-in slave-out data from MAX1258 is delayed one clock cycle.
** When instructed to transfer n bytes, the hardware must generate
** n * 8 clock pulses. However, the first bit of miso_buf[] must be sampled
** concurrent with the SECOND bit of mosi[].
** The final bit of miso_buf[] does not get a clock pulse, instead the
** final state of the MAX1258 DOUT pin is sampled prior to negating CS.
*/
extern bool SPI_Transfer_MISO_Delayed(int byte_count,
    const unsigned __int8 mosi[], unsigned __int8 miso_buf[]);

// Read the state of the EOC pin until the pin is low
// or until a platform-specific timeout expires.
// On Exit:
//    Return value = true if EOC pin is low
//    Return value = false if timeout expires and EOC is sitll high
//
extern bool Wait_MAX1258_EOC_Low(void);

// Pulse the CONV pin low and then high.
//
extern void Pulse_MAX1258_CONV(void);

// Set acquisition time (when in clock mode 01)
// DelayString is of the form:
//    200us
//    500us
//    1ms
//    2ms
//    5ms
//    10ms
//    20ms
//    50ms
//    100ms
//    200ms
//    500ms
//    1s
extern bool Set_Acquisition_Time(const char* DelayString);
```

Listing 1 (Sheet 1 of 10)

```
MAX1258 EV kit Listing 1                06/01/04                                2
MAX1258EV listing1


//------------------------------------------------
// MAX1258 Conversion register
// 1xxx xxxx
#define MAX1258_CONV      0x80
//
// Power-on state:   1000 0000
#define MAX1258_CONV_POR     0x80
//
// Channel Selection
#define MAX1258_CONV_AIN00       0x80     /* 10000xxx AIN0 */
#define MAX1258_CONV_AIN01       0x88     /* 10001xxx AIN1 */
#define MAX1258_CONV_AIN02       0x90     /* 10010xxx AIN2 */
#define MAX1258_CONV_AIN03       0x98     /* 10011xxx AIN3 */
#define MAX1258_CONV_AIN04       0xA0     /* 10100xxx AIN4 */
#define MAX1258_CONV_AIN05       0xA8     /* 10101xxx AIN5 */
#define MAX1258_CONV_AIN06       0xB0     /* 10110xxx AIN6 */
#define MAX1258_CONV_AIN07       0xB8     /* 10111xxx AIN7 */
#define MAX1258_CONV_AIN08       0xC0     /* 11000xxx AIN8 */
#define MAX1258_CONV_AIN09       0xC8     /* 11001xxx AIN9 */
#define MAX1258_CONV_AIN10       0xD0     /* 11010xxx AIN10 */
#define MAX1258_CONV_AIN11       0xD8     /* 11011xxx AIN11 */
#define MAX1258_CONV_AIN12       0xE0     /* 11100xxx AIN12 */
#define MAX1258_CONV_AIN13       0xE8     /* 11101xxx AIN13 */
#define MAX1258_CONV_AIN14       0xF0     /* 11110xxx AIN14 */
#define MAX1258_CONV_AIN15       0xF8     /* 11111xxx AIN15 */
//
// Actions
#define MAX1258_CONV_SCAN_00_N          0x80     /* 1xxxx000 Scan 0,1,2,...N */
#define MAX1258_CONV_SCAN_T_00_N        0x81     /* 1xxxx001 Scan T,0,1,2,..N */
#define MAX1258_CONV_SCAN_N_15          0x82     /* 1xxxx010 Scan N,N+1,...,15 */
#define MAX1258_CONV_SCAN_T_N_15        0x83     /* 1xxxx011 Scan T,N,N+1,...,15 */
#define MAX1258_CONV_SINGLE_REPEAT      0x84     /* 1xxxx10x Read repeatedly */
#define MAX1258_CONV_SINGLE_READ        0x86     /* 1xxxx11x Read once */
//
#define MAX1258_ACTION_MASK             0x87     /* 1xxxx111 bits to test*/

//------------------------------------------------
// MAX1258 Setup register
// 01xx xx00
//
// Setup register may optionally be followed by
// one of the the differential configuration registers.
// 01xxxx10 followed by a second byte, selecting Unipolar-Differential inputs
// 01xxxx11 followed by a second byte, selecting Bipolar-Differential inputs
#define MAX1258_SETUP               0x40     /* 01xxxx00 no additional bytes */
#define MAX1258_SETUP_UNIDIFF       0x42     /* 01xxxx10 followed by another byte */
#define MAX1258_SETUP_BIPDIFF       0x43     /* 01xxxx11 followed by another byte */
//
// Power-on state:   0110 0000
#define MAX1258_SETUP_POR     0x60
//
// Clock Mode
// 0100xxxx pin16=CNVST, Int clock, Triggered by CNVST pulse
// 0101xxxx pin16=CNVST, Int clock, Triggered by CNVST pulses, custom Tacq
// 0110xxxx pin16=AIN15, Int clock, Triggered by conversion register write
// 0111xxxx pin16=AIN15, Ext clock, Triggered by conversion register write
#define MAX1258_SETUP_INTCLK_CNVST             0x40     /* 0100xxxx CNVST */
#define MAX1258_SETUP_INTCLK_CNVST_TACQ        0x50     /* 0101xxxx CNVST */
#define MAX1258_SETUP_INTCLK                   0x60     /* 0110xxxx AIN15 */
#define MAX1258_SETUP_EXTCLK                   0x70     /* 0111xxxx AIN15 */
//
// Reference Voltage
// MAX1258: 01xx00xx Pin 48=AIN14, ADCREF=Internal,  DACREF=Internal
// MAX1258: 01xx01xx Pin 48=REF2,  ADCREF=REF2,     DACREF=REF1
```

Listing 1 (Sheet 2 of 10)

```
MAX1258 EV kit Listing 1                06/01/04                                3
MAX1258EV listing1


// MAX1258: 01xx10xx Pin 48=AIN14, ADCREF=Internal, DACREF=REF1
// MAX1258: 01xx11xx Pin 48=REF2, ADCREF=REF1-REF2, DACREF=REF1
#define MAX1258_SETUP_REF00        0x40     /* 01xx00xx */
#define MAX1258_SETUP_INTREF_SLEEP        0x40     /* 01xx00xx Pin 48=AIN14 */
#define MAX1258_SETUP_REF01         0x44     /* 01xx01xx */
#define MAX1258_SETUP_EXTREF              0x44     /* 01xx01xx Pin 48=REF2 */
#define MAX1258_SETUP_REF10       0x48     /* 01xx10xx */
#define MAX1258_SETUP_INTREF_ACTIVE       0x48     /* 01xx10xx Pin 48=AIN14 */
#define MAX1258_SETUP_REF11         0x4C     /* 01xx11xx */
#define MAX1258_SETUP_EXTREF_DIFF         0x4C     /* 01xx11xx Pin 48=REF2 */
//
//
// MAX1258 Unipolar-Differential input pairs
// Byte Following MAX1258_SETUP_UNIDIFF
// 0 1 x x  x x 1 0  unidiff
//
// Power-on state:   0110 0010 0000 0000
#define MAX1258_SETUP_UNIDIF_POR       0x00
//
#define MAX1258_SETUP_UNIDIF0001       0x80
#define MAX1258_SETUP_UNIDIF0203       0x40
#define MAX1258_SETUP_UNIDIF0405       0x20
#define MAX1258_SETUP_UNIDIF0607       0x10
#define MAX1258_SETUP_UNIDIF0809       0x08
#define MAX1258_SETUP_UNIDIF1011       0x04
#define MAX1258_SETUP_UNIDIF1213       0x02
#define MAX1258_SETUP_UNIDIF1415       0x01
//
// MAX1258 Bipolar-Differential input pairs
// Byte Following MAX1258_SETUP_BIPDIFF
// 0 1 x x  x x 1 1  bipdiff
//
// Power-on state:   0110 0011 0000 0000
#define MAX1258_SETUP_BIPDIF_POR       0x00
//
#define MAX1258_SETUP_BIPDIF0001       0x80
#define MAX1258_SETUP_BIPDIF0203       0x40
#define MAX1258_SETUP_BIPDIF0405       0x20
#define MAX1258_SETUP_BIPDIF0607       0x10
#define MAX1258_SETUP_BIPDIF0809       0x08
#define MAX1258_SETUP_BIPDIF1011       0x04
#define MAX1258_SETUP_BIPDIF1213       0x02
#define MAX1258_SETUP_BIPDIF1415       0x01

//-------------------------------------------------
// MAX1258 Averaging register
// 0 0 1 x  x x x x
//
// Power-on state:   0010 0000
#define MAX1258_AVERAGE_POR       0x20
//
// Averaging
// 001000xx One measurement result (no averaging)
// 001100xx Mean of 4 measurement results
// 001101xx Mean of 8 measurement results
// 001110xx Mean of 16 measurement results
// 001111xx Mean of 32 measurement results
#define MAX1258_AVERAGE_1       0x20     /* 001000xx No averaging */
#define MAX1258_AVERAGE_4       0x30     /* 001100xx Mean of 4 measurements */
#define MAX1258_AVERAGE_8       0x34     /* 001101xx Mean of 8 measurements */
#define MAX1258_AVERAGE_16      0x38     /* 001110xx Mean of 16 measurements */
#define MAX1258_AVERAGE_32      0x3C     /* 001111xx Mean of 32 measurements */
//
// Repeat Count
```

Listing 1 (Sheet 3 of 10)

```
MAX1258 EV kit Listing 1                06/01/04                                4
MAX1258EV listing1

// Enabled by MAX1258_CONV_SINGLE_REPEAT 1xxxx10x
// Internal clock modes only
#define MAX1258_REPEAT_4        0x20    /* 001xxx00 4 times */
#define MAX1258_REPEAT_8        0x21    /* 001xxx01 8 times */
#define MAX1258_REPEAT_12       0x22    /* 001xxx10 12 times */
#define MAX1258_REPEAT_16       0x23    /* 001xxx11 16 times */

//-----------------------------------------------------
// MAX1258 Reset register (reset command)
// 0 0 0 0 1 <RESET> <SLOW> <FBGON>
//
// Reset all registers to their power-on default states
#define MAX1258_RESET_ALL        0x0C    /* 000011xx Reset All Registers */
//
// "Slow" mode
#define MAX1258_RESET_SLOW       0x0A    /* 0000101x "Slow" mode */
//
// Force bandgap and bias block to be turned on (and clear FIFO)
#define MAX1258_RESET_FBGON       0x09    /* 000010x1 Force bandgap on */

//-----------------------------------------------------
// MAX1258 GPIO (General-purpose Input/Output)
//
// MAX1220 has four GPIO pins
// MAX1221 has four GPIO pins
// MAX1257/MAX1258 have twelve GPIO pins
//
// To configure a GPIO pin for OUTPUT,
//     set the corresponding bit 1 in the configuration register.
//     Pin state is controlled by a bit in the write-data register.
//
// To configure a GPIO pin for INPUT,
//     set the corresponding bit 0 in the configuration register
//     and set the corresponding bit 1 in the write-data register.
//     Pin state is returned in a bit in the read-data register.
//
//-----------------------------------------------------
// MAX1258 GPIO Configuration register
// 0 0 0 0  0 0 1 1 <GPIO pin masks>
//
#define MAX1258_GPIO_CONFIG      0x03    /* 00000011 next 16 bits = GPIO configuration
data */
#define MAX1220_GPIO_CONFIG      0x03    /* 00000011 next 8 bits = GPIO configuration
data */
//-----------------------------------------------------
// MAX1258 GPIO Write register
// 0 0 0 0  0 0 1 0 <GPIO pin masks>
//
#define MAX1258_GPIO_WRITE      0x02    /* 00000010 next 16 bits = GPIO write data */
#define MAX1220_GPIO_WRITE      0x02    /* 00000010 next 8 bits = GPIO write data */
//-----------------------------------------------------
// MAX1258 GPIO Read register
// 0 0 0 0  0 0 0 1 <GPIO pin masks>
//
#define MAX1258_GPIO_READ      0x01    /* 00000001 next 16 bits = GPIO read data */
#define MAX1220_GPIO_READ      0x01    /* 00000001 next 8 bits = GPIO read data */
//-----------------------------------------------------
// MAX1257/MAX1258 GPIO pin WRITE/CONGIFURE mask bits
//     <GPIOC3> <GPIOC2> <GPIOC1> <GPIOC0> <GPIOB3> <GPIOB2> <GPIOB1> <GPIOB0>
//     <GPIOA3> <GPIOA2> <GPIOA1> <GPIOA0>  X X X X
//
// MAX1257/MAX1258 GPIO pin READ mask bits
```

Listing 1 (Sheet 4 of 10)

```
MAX1258 EV kit Listing 1                06/01/04                                5
MAX1258EV listing1

//    X X X X  <GPIOC3> <GPIOC2> <GPIOC1> <GPIOC0>
//    <GPIOB3> <GPIOB2> <GPIOB1> <GPIOB0> <GPIOA3> <GPIOA2> <GPIOA1> <GPIOA0>
//
// For READ, the GPIOB3..GPIOB0 pins migrate to the third byte.
//
#define MAX1258_GPIO_WRC3      0x8000     /* 1xxxxxxx xxxxxxxx */
#define MAX1258_GPIO_WRC2      0x4000     /* x1xxxxxx xxxxxxxx */
#define MAX1258_GPIO_WRC1      0x2000     /* xx1xxxxx xxxxxxxx */
#define MAX1258_GPIO_WRC0      0x1000     /* xxx1xxxx xxxxxxxx */
#define MAX1258_GPIO_WRB3      0x0800     /* xxxx1xxx xxxxxxxx */
#define MAX1258_GPIO_WRB2      0x0400     /* xxxxx1xx xxxxxxxx */
#define MAX1258_GPIO_WRB1      0x0200     /* xxxxxx1x xxxxxxxx */
#define MAX1258_GPIO_WRB0      0x0100     /* xxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_WRA3      0x0080     /* xxxxxxxx 1xxxxxxx */
#define MAX1258_GPIO_WRA2      0x0040     /* xxxxxxxx x1xxxxxx */
#define MAX1258_GPIO_WRA1      0x0020     /* xxxxxxxx xx1xxxxx */
#define MAX1258_GPIO_WRA0      0x0010     /* xxxxxxxx xxx1xxxx */
//
#define MAX1258_GPIO_RDC3      0x0800     /* xxxx1xxx xxxxxxxx */
#define MAX1258_GPIO_RDC2      0x0400     /* xxxxx1xx xxxxxxxx */
#define MAX1258_GPIO_RDC1      0x0200     /* xxxxxx1x xxxxxxxx */
#define MAX1258_GPIO_RDC0      0x0100     /* xxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_RDB3      0x0080     /* xxxxxxxx 1xxxxxxx */
#define MAX1258_GPIO_RDB2      0x0040     /* xxxxxxxx x1xxxxxx */
#define MAX1258_GPIO_RDB1      0x0020     /* xxxxxxxx xx1xxxxx */
#define MAX1258_GPIO_RDB0      0x0010     /* xxxxxxxx xxx1xxxx */
#define MAX1258_GPIO_RDA3      0x0008     /* xxxxxxxx xxxx1xxx */
#define MAX1258_GPIO_RDA2      0x0004     /* xxxxxxxx xxxxx1xx */
#define MAX1258_GPIO_RDA1      0x0002     /* xxxxxxxx xxxxxx1x */
#define MAX1258_GPIO_RDA0      0x0001     /* xxxxxxxx xxxxxxx1 */
//
//------------------------------------------------
// GPIO pin masks for all GPIO commands
// MAX1220/MAX1221 GPIO pin WRITE/CONGIFURE mask bits
//    <GPIOC3> <GPIOC2> <GPIOA1> <GPIOA0>  X X X X
//
// MAX1220/MAX1221 GPIO pin READ mask bits
//    X X X X  <GPIOC3> <GPIOC2> <GPIOA1> <GPIOA0>
//
#define MAX1220_GPIO_WRC1      0x80     /* 1xxxxxxx */
#define MAX1220_GPIO_WRC0      0x40     /* x1xxxxxx */
#define MAX1220_GPIO_WRA1      0x20     /* xx1xxxxx */
#define MAX1220_GPIO_WRA0      0x10     /* xxx1xxxx */
//
#define MAX1220_GPIO_RDC1      0x08     /* xxxx1xxx */
#define MAX1220_GPIO_RDC0      0x04     /* xxxxx1xx */
#define MAX1220_GPIO_RDA1      0x02     /* xxxxxx1x */
#define MAX1220_GPIO_RDA0      0x01     /* xxxxxxx1 */
//
//------------------------------------------------


//------------------------------------------------
// MAX1258 DAC Select register
// 0 0 0 1  x x x x
//     <C3>  <C2>  <C1>  <C0>  <D11> <D10> <D09> <D08>
//    <D07> <D06> <D05> <D04>  <D03> <D02> <D01> <D00>
//
// This 8-bit preamble is followed by 4-bit command and 12-bit data,
// described in the next table.
//
#define MAX1258_DAC      0x10     /* 0001xxxx next 16 bits = DAC command and data */
//
// Because the DAC requires sending 3 bytes, the following constants
// use the prefix MAX1258_DACc_ for the second byte (command byte)
```

Listing 1 (Sheet 5 of 10)

MAX1258 EV kit Listing 1 06/01/04 6
MAX1258EV listing1

```
// and the prefix MAX1258_DACd_ for the third byte (data byte).
//
// A complete DAC command requires a 3-byte SPI transfer.
//
//---------------------------------------------------
// MAX1258 DAC commands
// <C3> <C2> <C1> <C0> <D11..D0 = 0>
//
#define MAX1258_DACc_NOP         0x00  /* 0000 no operation */
//
// DAC reset commands
// 0 0 0 1 0 x x x x x x x x x x x x  // reset all input and DAC registers to 0x000
// 0 0 0 1 1 x x x x x x x x x x x x  // reset all input and DAC registers to 0xFFF
#define MAX1258_DACc_RESET_000   0x10  /* 00010xxx reset to 0x000 */
#define MAX1258_DACc_RESET_FFF   0x18  /* 00011xxx reset to 0xFFF */
//
// DAC input register write commands (not updating DAC outputs)
#define MAX1258_DACc_WRITE1      0x20  /* 0010 write input register 1 */
#define MAX1258_DACc_WRITE2      0x30  /* 0011 write input register 2 */
#define MAX1258_DACc_WRITE3      0x40  /* 0100 write input register 3 */
#define MAX1258_DACc_WRITE4      0x50  /* 0101 write input register 4 */
#define MAX1258_DACc_WRITE5      0x60  /* 0110 write input register 5 */
#define MAX1258_DACc_WRITE6      0x70  /* 0111 write input register 6 */
#define MAX1258_DACc_WRITE7      0x80  /* 1000 write input register 7 */
#define MAX1258_DACc_WRITE8      0x90  /* 1001 write input register 8 */
//
// DAC input register and DAC write-through commands (updating DAC outputs)
#define MAX1258_DACc_WRITE14LOAD     0xA0  /* 1010 write input registers 1-4 and DAC
registers 1-4 */
#define MAX1258_DACc_WRITE58LOAD     0xB0  /* 1011 write input registers 5-8 and DAC
registers 5-8 */
#define MAX1258_DACc_WRITE18LOAD     0xC0  /* 1100 write input registers 1-8 and DAC
registers 1-8 */
//
// DAC multiple input register write commands (not updating DAC outputs)
#define MAX1258_DACc_WRITE18     0xD0  /* 1101 write input registers 1-8 */
//
// DAC load commands
// 1 1 1 0 x x x x x x x 1 x x x x  // load DAC 1 from input register 1
// 1 1 1 0 x x x x x x 1 x x x x x  // load DAC 2 from input register 2
// 1 1 1 0 x x x x x 1 x x x x x x  // load DAC 3 from input register 3
// 1 1 1 0 x x x x 1 x x x x x x x  // load DAC 4 from input register 4
// 1 1 1 0 x x x 1 x x x x x x x x  // load DAC 5 from input register 5
// 1 1 1 0 x x 1 x x x x x x x x x  // load DAC 6 from input register 6
// 1 1 1 0 x 1 x x x x x x x x x x  // load DAC 7 from input register 7
// 1 1 1 0 1 x x x x x x x x x x x  // load DAC 8 from input register 8
#define MAX1258_DACc_LOAD    0xE0  /* 1110 load DAC registers from input registers,
masked... */
#define MAX1258_DACc_CH8      0x08     /* xxxx10000000xxxx DAC 8 */
#define MAX1258_DACc_CH7      0x04     /* xxxx01000000xxxx DAC 7 */
#define MAX1258_DACc_CH6      0x02     /* xxxx00100000xxxx DAC 6 */
#define MAX1258_DACc_CH5      0x01     /* xxxx00010000xxxx DAC 5 */
#define MAX1258_DACd_CH4      0x80     /* xxxx00001000xxxx DAC 4 */
#define MAX1258_DACd_CH3      0x40     /* xxxx00000100xxxx DAC 3 */
#define MAX1258_DACd_CH2      0x20     /* xxxx00000010xxxx DAC 2 */
#define MAX1258_DACd_CH1      0x10     /* xxxx00000001xxxx DAC 1 */
#define MAX1258_DACd_CH1234  0xF0     /* xxxx00001111xxxx DAC 1..4 */
#define MAX1258_DACc_CH5678  0x0F     /* xxxx11110000xxxx DAC 5..8 */
//
//---------------------------------------------------
// DAC Power-up and Power-down commands
// All of these power-up/power-down commands operate on individual DAC buffers.
//
#define MAX1258_DACc_PWR    0xF0  /* 1111 power-up or power-down DAC channels*/
```

Listing 1 (Sheet 6 of 10)

```
MAX1258 EV kit Listing 1                06/01/04                                7
MAX1258EV listing1


//
// DAC power-up/power-down channel select mask bits:
// 1 1 1 1 <dac8> <dac7> <dac6> <dac5> <dac4> <dac3> <dac2> <dac1> x x x x
//  (note: 0 = no change in DAC power-on state)
//
// DAC power-up/power-down command bits:
// 1 1 1 1 d d d d  d d d d 0 0 1 x  // power up selected DAC buffers
// 1 1 1 1 d d d d  d d d d 0 1 0 x  // power off selected DAC buffers, high impedance output
// 1 1 1 1 d d d d  d d d d 1 0 0 x  // power off selected DAC buffers, 1kohm to AGND
// 1 1 1 1 d d d d  d d d d 0 0 0 x  // power off selected DAC buffers, 100kohm to AGND
// 1 1 1 1 d d d d  d d d d 1 1 1 x  // power off selected DAC buffers, 100kohm to V(REF1)
#define MAX1258_DACd_PWR_ON               0x02  /* d4d3d2d1 001x power ON selected
channels */
#define MAX1258_DACd_PWR_OFF              0x04  /* d4d3d2d1 010x power OFF, high
impedance */
#define MAX1258_DACd_PWR_OFF_1K_AGND      0x08  /* d4d3d2d1 100x power OFF, 1kohm to
AGND */
#define MAX1258_DACd_PWR_OFF_100K_AGND    0x00  /* d4d3d2d1 000x power OFF, 100kohm to
AGND */
#define MAX1258_DACd_PWR_OFF_100K_VREF    0x0F  /* d4d3d2d1 111x power OFF, 100kohm to
V(REF1) */


//------------------------------------------------
// Enumerated type defining the meaning of each of the
// MAX1258's FIFO data slots.
typedef enum {
    //
    // Unused FIFO slot; meaningless data.
     UNDEFINED = 0,
    //
    // Temperature measurement.
    // The scan modes always place temperature data
    // at the head of the FIFO.
     TEMPERATURE,
    //
    // Single-ended unipolar analog inputs.
    // Code 0x0000 = minimum voltage
    // Code 0x0FFF = maximum voltage
    // Note that AIN14 and AIN15 pins have optional alternate functions.
     UNIAIN00, UNIAIN01, UNIAIN02, UNIAIN03,
     UNIAIN04, UNIAIN05, UNIAIN06, UNIAIN07,
     UNIAIN08, UNIAIN09, UNIAIN10, UNIAIN11,
     UNIAIN12, UNIAIN13, UNIAIN14, UNIAIN15,
    //
    // Unipolar differential input pairs.
    // Code 0x0000 = minimum voltage
    // Code 0x0FFF = maximum voltage
     UNIDIF0001, UNIDIF0203, UNIDIF0405, UNIDIF0607,
     UNIDIF0809, UNIDIF1011, UNIDIF1213, UNIDIF1415,
    //
    // Bipolar differential input pairs.
    // Code 0x07FF = maximum voltage
    // Code 0x0000 = zero volts
    // Code 0x0800 = minimum voltage
     BIPDIF0001, BIPDIF0203, BIPDIF0405, BIPDIF0607,
     BIPDIF0809, BIPDIF1011, BIPDIF1213, BIPDIF1415,
    //
     NUM_FIFO_ENTRY_TYPES
} MAX1258_fifo_entry_t;

//------------------------------------------------
// Enumerated type defining each pair of MAX1258 inputs
```

Listing 1 (Sheet 7 of 10)

MAX1258 EV kit Listing 1 06/01/04 8
MAX1258EV listing1

```
// as single-ended or differential
typedef enum {
     SINGLE_ENDED = 0,
     UNIPOLAR_DIFFERENTIAL,
     BIPOLAR_DIFFERENTIAL
} MAX1258_channel_config_t;

//-----------------------------------------------
// C++ class representing the state of a MAX1258
class MAX1258
{
public:
     // MAX1258 registers cannot be read,
     // so keep track of the register values here.
     int conversion_register;
     int new_conversion_register;
     int setup_register;
     int setup_unidiff_register;
     int setup_bipdiff_register;
     int averaging_register;
     //
     int reset_register;

     int gpio_config;
     int gpio_data;
     //
     int dac_input[8];

     // The reference voltage is used to calculate the input voltage
     // represented by each measurement.
     double Vintref;       // internal reference voltage
     double VpinREF1;      // REF1 pin voltage
     double VpinREF2;      // REF2/AIN14 pin voltage
     double VrefDAC(void);    // DAC full-scale voltage depends on setup register
     double VrefADC(void);    // ADC full-scale voltage depends on setup register

     int adc_code[17];

     double DAC_Voltage(int code) {
         return VrefDAC() * code / 4096;
     };

     // Constructor for class MAX1258.
     MAX1258(void);

     // Write a value to one of the part's registers.
     bool Write_Conversion(int value);
     bool Write_Setup(int value);
     bool Write_Setup_UniDiff(int value, int unidiff);
     bool Write_Setup_BipDiff(int value, int bipdiff);
     bool Write_Averaging(int value);
     bool Write_Reset(int value);
     int Read_Data(int index);
     int Read_Data_Trigger_Next_Conversion(int index);
     bool Read_Multiple_Data_Channels(int count);

     // Input configuration
     // Array InputPairConfig[] determines whether each pair
     // of input channels is configured as two single-ended inputs,
     // a unipolar differential pair, or a bipolar differential pair.
     MAX1258_channel_config_t  InputPairConfig[8];
     //
     // Member function to figure out the values of InputPairConfig
     // based on the MAX1258 register values.
     // Call this function afer setting setup_unidiff_register and setup_bipdiff_register
     // before using the values of InputPairConfig[].
```

Listing 1 (Sheet 8 of 10)

MAX1258 EV kit Listing 1 06/01/04 9
MAX1258EV listing1

```
    void Update_InputPairConfig(void);

    // The MAX1258 has a FIFO data buffer with
    // 16 entries (plus an optional temperature measurement).
    // Array FIFO_meaning[] determines what to do with
    // each successive word read from the MAX1258.
     MAX1258_fifo_entry_t  FIFO_meaning[17];
    //
    // Member function to figure out the values of FIFO_meaning
    // based on the MAX1258 register values.
    // Call this function afer setting conversion_register, setup_register, and averaging_register
    // before using the values of FIFO_meaning[].
    void Update_FIFO_meaning(void);

    int channel (MAX1258_fifo_entry_t meaning) {
        if ((UNIAIN00 <= meaning) && (meaning <= UNIAIN15)) {
            return (meaning - UNIAIN00);
        } else
        if ((UNIDIF0001 <= meaning) && (meaning <= UNIDIF1415)) {
            return (meaning - UNIDIF0001) * 2;
        } else
        if ((BIPDIF0001 <= meaning) && (meaning <= BIPDIF1415)) {
            return (meaning - BIPDIF0001) * 2;
        } else
        if (meaning == TEMPERATURE) {
            return 16;
        } else {
            return 0;
        }
    };

    // General-Purpose Input/Output pin functions
    //
    // Configure the specified GPIO pins as outputs without changing the other pins
    bool GPIO_Outputs(int pins_mask);
    //
    // Configure the specified GPIO pins as inputs without changing the other pins
    bool GPIO_Inputs(int pins_mask);
    //
    // Read the specified GPIO pins
    int GPIO_Read(int pins_mask);
    //
    // Write the specified GPIO output pins high, all other output pins low.
    bool GPIO_Write(int pins_mask);
    //
    // Write the specified GPIO pins high without changing the other pins
    bool GPIO_Set(int pins_mask);
    //
    // Write the specified GPIO pins low without changing the other pins
    bool GPIO_Clear(int pins_mask);

    // DAC Analog Outputs
    //
#define MAX1258_MASK_CH1 0x10
#define MAX1258_MASK_CH2 0x20
#define MAX1258_MASK_CH3 0x40
#define MAX1258_MASK_CH4 0x80
#define MAX1258_MASK_CH5 0x01
#define MAX1258_MASK_CH6 0x02
#define MAX1258_MASK_CH7 0x04
#define MAX1258_MASK_CH8 0x08
    //
    // Send the DAC prefix with the no-operation command
    // MAX1258_DACc_NOP
    bool DAC_No_Operation(void);
```

Listing 1 (Sheet 9 of 10)

MAX1258 EV kit Listing 1 06/01/04 10
MAX1258EV listing1

```
    //
    // Reset all DAC channels to all minimum output (all 0's)
    // MAX1258_DACc_RESET_000
    bool DAC_Reset_All_000(void);
    //
    // Reset all DAC channels to all maximum output (all 1's)
    // MAX1258_DACc_RESET_FFF
    bool DAC_Reset_All_FFF(void);
    //
    // Write the specified DAC channel(s) input register, without changing the output value
    // MAX1258_DACc_WRITE1 .. MAX1258_DACc_WRITE8
    bool DAC_Write(int channel_number_12345678, int value);
    //
    // Write the specified value to DAC input registers channel 1-4 and update the outputs
    // MAX1258_DACc_WRITE14LOAD
    bool DAC_Write_Load_CH1234(int value);
    //
    // Write the specified value to DAC input registers channel 5-8 and update the outputs
    // MAX1258_DACc_WRITE58LOAD
    bool DAC_Write_Load_CH5678(int value);
    //
    // Write the specified value to DAC input registers channel 1-8 and update the outputs
    // MAX1258_DACc_WRITE18LOAD
    bool DAC_Write_Load_All(int value);
    //
    // Write the specified value to DAC input registers channel 1-8
    // MAX1258_DACc_WRITE18
    bool DAC_Write_All(int value);
    //
    // Update the specified DAC channel(s) output value from the corresponding input register, changing the output
value
    // MAX1258_DACc_LOAD
    bool DAC_Load_channel(int channel_number_12345678);
    bool DAC_Load_channels(int channel_mask);
    //
    // Power-on the specified DAC channel(s) without changing the others
    // MAX1258_DACd_PWR_ON
    bool DAC_PowerOn_channels(int channel_mask);
    //
    // Power-off the specified DAC channel(s) high-impedance without changing the others
    // MAX1258_DACd_PWR_OFF
    bool DAC_PowerOff_HiZ_channels(int channel_mask);
    //
    // Power-off the specified DAC channel(s) VREF-100kohm without changing the others
    // MAX1258_DACd_PWR_OFF_100K_VREF
    bool DAC_PowerOff_Vref100k_channels(int channel_mask);
    //
    // Power-off the specified DAC channel(s) GND-100kohm without changing the others
    // MAX1258_DACd_PWR_OFF_100K_AGND
    bool DAC_PowerOff_Gnd100k_channels(int channel_mask);
    //
    // Power-off the specified DAC channel(s) GND-1kohm without changing the others
    // MAX1258_DACd_PWR_OFF_1K_AGND
    bool DAC_PowerOff_Gnd1k_channels(int channel_mask);


};

//---------------------------------------------------------------------------
#endif // DRV1258H
```

Listing 1 (Sheet 10 of 10)

MAX1258 EV kit Listing 2 06/01/04 1
MAX1258EV listing2

```
// Drv1258.cpp
// MAX1258-specific driver.
// mku 04/07/2004
// (C) 2004 Maxim Integrated Products
// For use with Borland C++ Builder 3.0
//---------------------------------------------------------------------------
// Revision history:
// ======================================
// __/__/2004: Initial Release as MAX1258 Version 1.0
// ======================================
//---------------------------------------------------------------------------

#include "Drv1258.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 MAX1258_IMPOSSIBLE_DATA_VALUE 32000
#endif
//---------------------------------------------------------------------------
MAX1258::MAX1258(void)
{
     conversion_register = MAX1258_CONV_POR;
     setup_register = MAX1258_SETUP_POR;
     setup_unidiff_register = MAX1258_SETUP_UNIDIF_POR;
     setup_bipdiff_register = MAX1258_SETUP_BIPDIF_POR;
     averaging_register = MAX1258_AVERAGE_POR;
    //~ Vref = 2.500;
     Vintref = 4.096;
     VpinREF1 = 0;
     VpinREF2 = 0;
     for (int index = 0; index < 8; index++) {
          InputPairConfig[index] = SINGLE_ENDED;
     }
     for (int index = 0; index < 17; index++) {
          FIFO_meaning[index] = UNDEFINED;
     }
}
//---------------------------------------------------------------------------
bool MAX1258::Write_Conversion(int value)
{
    // NOTE: the act of writing to the conversion register
    // has the effect of triggering one or more conversions
    // if the clock mode is 10 or 11.

    value = value &~ 0x00;  // x x x x  x x x x
    value = value |  0x80;  // 1 x x x  x x x x
                            // 1 x x x  x x x x

    const unsigned __int8 mosi[] = {
        (unsigned __int8)(value)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
         conversion_register = value;
}
```

Listing 2 (Sheet 1 of 14)

MAX1258 EV kit Listing 2 06/01/04 2
MAX1258EV listing2

```
    return result;
}
//---------------------------------------------------------------------
bool MAX1258::Write_Setup(int value)
{
    value = value &~ 0x83;  //0xxx xx00
    value = value |  0x40;  //x1xx xxxx
                            //01xx xx00

    const unsigned __int8 mosi[] = {
        (unsigned __int8)(value)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        setup_register = value;
        Update_FIFO_meaning();
    }
    return result;
}
//---------------------------------------------------------------------
double MAX1258::VrefDAC(void)
{
    switch (setup_register & MAX1258_SETUP_REF11) {
    case MAX1258_SETUP_REF00:
        return Vintref;
    case MAX1258_SETUP_REF01:
        return VpinREF1;
    case MAX1258_SETUP_REF10:
        return VpinREF1;
    case MAX1258_SETUP_REF11:
        return VpinREF1;
    }
    return Vintref;
}
//---------------------------------------------------------------------
double MAX1258::VrefADC(void)
{
    switch (setup_register & MAX1258_SETUP_REF11) {
    case MAX1258_SETUP_REF00:
        return Vintref;
    case MAX1258_SETUP_REF01:
        return VpinREF2;
    case MAX1258_SETUP_REF10:
        return Vintref;
    case MAX1258_SETUP_REF11:
        return VpinREF1-VpinREF2;
    }
    return Vintref;
}
//---------------------------------------------------------------------
bool MAX1258::Write_Setup_UniDiff(int value, int unidiff)
{
    value = value &~ 0x81;  //0xxx xxx0
    value = value |  0x42;  //x1xx xx1x
                            //01xx xx10 unidiff
    //01xx xx10
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(value),
        (unsigned __int8)(unidiff)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        setup_register = value;
        setup_unidiff_register = unidiff;
        Update_InputPairConfig();
        Update_FIFO_meaning();
    }
```

Listing 2 (Sheet 2 of 14)

MAX1258 EV kit Listing 2 06/01/04 3
MAX1258EV listing2

```
    return result;
}
//---------------------------------------------------------------------
bool MAX1258::Write_Setup_BipDiff(int value, int bipdiff)
{
    value = value &~ 0x80;  //0xxx xxxx
    value = value |  0x43;  //x1xx xx11
                            //01xx xx11 bipdiff
    //01xx xx10
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(value),
        (unsigned __int8)(bipdiff)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        setup_register = value;
        setup_bipdiff_register = bipdiff;
        Update_InputPairConfig();
        Update_FIFO_meaning();
    }
    return result;
}
//---------------------------------------------------------------------
bool MAX1258::Write_Averaging(int value)
{
    value = value &~ 0xC0;  //00xx xxxx
    value = value |  0x20;  //xx1x xxxx
                            //001x xxxx

    const unsigned __int8 mosi[] = {
        (unsigned __int8)(value)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        averaging_register = value;
        Update_FIFO_meaning();
    }
    return result;
}
//---------------------------------------------------------------------
bool MAX1258::Write_Reset(int value)
{
    value = value &~ 0xF0;  //0000 xxxx
    value = value |  0x08;  //xxxx 1xxx
                            //0000 1xxx
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(value)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
    }
    return result;
}
//---------------------------------------------------------------------
int MAX1258::Read_Data(int index)
{
    if ((index < 0) || (index >= 17)) {
#if THROW_EXCEPTIONS
        throw "illegal channel index in MAX1258::Read_Data";
#else
        return MAX1254_IMPOSSIBLE_DATA_VALUE;
#endif
    }
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(0),
```

Listing 2 (Sheet 3 of 14)

MAX1258 EV kit Listing 2 06/01/04 4
MAX1258EV listing2

```
        (unsigned __int8)(0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        int data16 = (miso_buf[0] * 0x100) + miso_buf[1];
        adc_code[index] = data16;
        return data16;
    }
#if THROW_EXCEPTIONS
    throw "SPI_Transfer failed in MAX1258::Read_Data";
#else
    return MAX1258_IMPOSSIBLE_DATA_VALUE;
#endif
}
//---------------------------------------------------------------------
int MAX1258::Read_Data_Trigger_Next_Conversion(int index)
{
    if ((index < 0) || (index >= 17)) {
#if THROW_EXCEPTIONS
        throw "illegal channel index in MAX1258::Read_Data_Trigger_Next_Conversion";
#else
        return MAX1254_IMPOSSIBLE_DATA_VALUE;
#endif
    }
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(0),
        (unsigned __int8)(conversion_register)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        int data16 = (miso_buf[0] * 0x100) + miso_buf[1];
        adc_code[index] = data16;
        return data16;
    }
#if THROW_EXCEPTIONS
    throw "SPI_Transfer failed in MAX1258::Read_Data_Trigger_Next_Conversion";
#else
    return MAX1258_IMPOSSIBLE_DATA_VALUE;
#endif
}
//---------------------------------------------------------------------
bool MAX1258::Read_Multiple_Data_Channels(int count)
{
    switch(setup_register & 0x30)
    {
    case 0x00: // 0100xxxx pin16=CNVST, Int clock, Triggered by CNVST pulse
            // Clock mode 00:
            //    Pulse CNVST pin
            //    Wait for EOC low
            //    issue command "R" to read each 16-bit value from the FIFO
             //
            // Update configuration register value only if necessary.
             if (new_conversion_register != conversion_register) {
                 Write_Conversion(new_conversion_register);
             }
            // Trigger conversion by pulsing the CNVST pin.
              Pulse_MAX1258_CONV();
             if (Wait_MAX1258_EOC_Low()) {
                 for (int index = 0; index < count; index++) {
                     Read_Data(index);
                 }
             }
        return true;
    case 0x10: // 0101xxxx pin16=CNVST, Int clock, Triggered by CNVST pulses, custom Tacq
        // Clock mode 01:
        //    For each requested channel,
```

Listing 2 (Sheet 4 of 14)

```
        //      Drive CNVST pin low
        //      Delay for the acquisition time (run this delay on 68HC16?)
        //      Drive CNVST pin high
        //      Wait for EOC low
        //    issue command "R" to read each 16-bit value from the FIFO
         //
        // Update configuration register value only if necessary.
         if (new_conversion_register != conversion_register) {
              Write_Conversion(new_conversion_register);
         }
        // Trigger conversion by pulsing the CNVST pin for each channel.
         for (int index = 0; index < 17; index++) {
             if (FIFO_meaning[index] != UNDEFINED)
             {
                 Pulse_MAX1258_CONV(); // TODO: pulse width sets acquisition time for each channel
                  if (Wait_MAX1258_EOC_Low()) {
                       Read_Data(index);
                  }
             }
         }
         return true;
    case 0x20: // 0110xxxx pin16=AIN15, Int clock, Triggered by conversion register write
         {
             // Clock mode 10:
             //    Write to the converion register 1xxx xxxx using command "Wxx"
             //    Wait for EOC low
             //    issue command "R" to read each 16-bit value from the FIFO
              //
             // Trigger conversion by writing the conversion register.
              Write_Conversion(new_conversion_register);
              if (Wait_MAX1258_EOC_Low()) {
                  for (int index = 0; index < count; index++) {
                       Read_Data(index);
                  }
              }
         }
         return true;
    case 0x30: // 0111xxxx pin16=AIN15, Ext clock, Triggered by conversion register write
         // Clock mode 11:
         //    Write to the converion register 1xxx xxxx using command "Wxx"
         //    issue command "R" to read 16-bit value
         //
         // Trigger conversion by writing the conversion register.
          Write_Conversion(new_conversion_register);
         // Clock mode 11 does not generate an EOC pulse.
           Read_Data_Trigger_Next_Conversion(0);
         return true;
    }
    return false; // TODO: this code was optimized into machine language file KIT1258.ASM
    // to increase data throughput. Need to convert back to portable C code,
    // and probably #ifdef it back to the optimized 68HC16-specific program.
}
//---------------------------------------------------------------------------
void MAX1258::Update_InputPairConfig(void)
{
    for (int index = 0; index < 8; index++) {
         InputPairConfig[index] = SINGLE_ENDED;
    }

    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0001)
         InputPairConfig[0] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0203)
         InputPairConfig[1] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0405)
         InputPairConfig[2] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0607)
```

Listing 2 (Sheet 5 of 14)

MAX1258 EV kit Listing 2 06/01/04 6
MAX1258EV listing2

```
        InputPairConfig[3] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0809)
        InputPairConfig[4] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF1011)
        InputPairConfig[5] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF1213)
        InputPairConfig[6] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF1415)
        InputPairConfig[7] = BIPOLAR_DIFFERENTIAL;

    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0001)
        InputPairConfig[0] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0203)
        InputPairConfig[1] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0405)
        InputPairConfig[2] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0607)
        InputPairConfig[3] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0809)
        InputPairConfig[4] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF1011)
        InputPairConfig[5] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF1213)
        InputPairConfig[6] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF1415)
        InputPairConfig[7] = UNIPOLAR_DIFFERENTIAL;

}
//---------------------------------------------------------------
void MAX1258::Update_FIFO_meaning(void)
{
    int conversion_register = new_conversion_register;

    for (int index = 0; index < 17; index++) {
        FIFO_meaning[index] = UNDEFINED;
    }

    int channel_field = (conversion_register >> 3) & 0x0F;
    int channel_index = channel_field;
    int repeat_count = 4;
    switch(averaging_register & 0xE3) {
    case MAX1258_REPEAT_4: // 001xxx00 4 times
        repeat_count = 4;
        break;
    case MAX1258_REPEAT_8: // 001xxx01 8 times
        repeat_count = 8;
        break;
    case MAX1258_REPEAT_12: // 001xxx10 12 times
        repeat_count = 12;
        break;
    case MAX1258_REPEAT_16: // 001xxx11 16 times
        repeat_count = 16;
        break;
    }

    MAX1258_fifo_entry_t meaning =
        (MAX1258_fifo_entry_t)(UNIAIN00 + channel_field);
    if (InputPairConfig[channel_field / 2] == BIPOLAR_DIFFERENTIAL) {
        meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + channel_field/2);
    } else if (InputPairConfig[channel_field / 2] == UNIPOLAR_DIFFERENTIAL) {
        meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + channel_field/2);
    }
    int index = 0;
    switch(conversion_register & MAX1258_ACTION_MASK) {
    case MAX1258_CONV_SCAN_00_N:
        meaning = UNIAIN00;
        channel_index = 0;
        while(channel_index <= channel_field) {
            int pair_index = channel_index / 2;
            if (InputPairConfig[pair_index] == BIPOLAR_DIFFERENTIAL) {
```

Listing 2 (Sheet 6 of 14)

MAX1258 EV kit Listing 2 06/01/04 7
MAX1258EV listing2

```
                meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + pair_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 2;
            } else if (InputPairConfig[pair_index] == UNIPOLAR_DIFFERENTIAL) {
                meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + pair_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 2;
            } else {
                meaning = (MAX1258_fifo_entry_t)(UNIAIN00 + channel_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 1;
            }
        }
        break;
    case MAX1258_CONV_SCAN_T_00_N:
         FIFO_meaning[index++] = TEMPERATURE;
        meaning = UNIAIN00;
        channel_index = 0;
        while(channel_index <= channel_field) {
            int pair_index = channel_index / 2;
            if (InputPairConfig[pair_index] == BIPOLAR_DIFFERENTIAL) {
                meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + pair_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 2;
            } else if (InputPairConfig[pair_index] == UNIPOLAR_DIFFERENTIAL) {
                meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + pair_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 2;
            } else {
                meaning = (MAX1258_fifo_entry_t)(UNIAIN00 + channel_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 1;
            }
        }
        break;
    case MAX1258_CONV_SCAN_N_15:
        while (index < 17) {
            int pair_index = channel_index / 2;
            if (InputPairConfig[pair_index] == BIPOLAR_DIFFERENTIAL) {
                meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + pair_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 2;
            } else if (InputPairConfig[pair_index] == UNIPOLAR_DIFFERENTIAL) {
                meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + pair_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 2;
            } else {
                meaning = (MAX1258_fifo_entry_t)(UNIAIN00 + channel_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 1;
            }
            if (meaning == UNIAIN15) break;
            if (meaning == UNIDIF1415) break;
            if (meaning == BIPDIF1415) break;
        }
        break;
    case MAX1258_CONV_SCAN_T_N_15:
         FIFO_meaning[index++] = TEMPERATURE;
        while (index < 17) {
            int pair_index = channel_index / 2;
            if (InputPairConfig[pair_index] == BIPOLAR_DIFFERENTIAL) {
                meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + pair_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 2;
            } else if (InputPairConfig[pair_index] == UNIPOLAR_DIFFERENTIAL) {
                meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + pair_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 2;
            } else {
                meaning = (MAX1258_fifo_entry_t)(UNIAIN00 + channel_index);
                FIFO_meaning[index++] = meaning;
                channel_index = channel_index + 1;
```

Listing 2 (Sheet 7 of 14)

MAX1258 EV kit Listing 2 06/01/04 8
MAX1258EV listing2

```
            }
            if (meaning == UNIAIN15) break;
            if (meaning == UNIDIF1415) break;
            if (meaning == BIPDIF1415) break;
        }
        break;
    case MAX1258_CONV_SINGLE_REPEAT:
        while (index < repeat_count) {
            FIFO_meaning[index++] = meaning;
        }
        break;
    case MAX1258_CONV_SINGLE_READ:
        FIFO_meaning[index++] = meaning;
        break;
    }

    /* Check for setups where AIN14-AIN15 are used for an alternate function */
    if ((setup_register & 0xE0) == MAX1258_SETUP_INTCLK_CNVST) {
        /* 0100xxxx AIN15 alternate function as CNVST input */
        /* 0101xxxx AIN15 alternate function as CNVST input */
        for (int index = 0; index < 17; index++) {
            meaning = FIFO_meaning[index];
            if (meaning == UNIAIN15)
                FIFO_meaning[index] = UNDEFINED;
            if (meaning == UNIDIF1415)
                FIFO_meaning[index] = UNDEFINED;
            if (meaning == BIPDIF1415)
                FIFO_meaning[index] = UNDEFINED;
        }
    }

    // MAX1258 AIN14 alternate pin function in mode 01xx01xx */
#if 1
    switch (setup_register & 0xCC) {
    case MAX1258_SETUP_EXTREF:
    case MAX1258_SETUP_EXTREF_DIFF:
        /* AIN14 alternate function as REF2 input */
        for (int index = 0; index < 17; index++) {
            meaning = FIFO_meaning[index];
            if (meaning == UNIAIN14)
                FIFO_meaning[index] = UNDEFINED;
            if (meaning == UNIDIF1415)
                FIFO_meaning[index] = UNDEFINED;
            if (meaning == BIPDIF1415)
                FIFO_meaning[index] = UNDEFINED;
        }
        break;
    default:
        break;
    }
#else
    if ((setup_register & 0xCC) == MAX1258_SETUP_EXTREF_DIFF) {
        /* MAX1231: 01xx11xx AIN14 alternate function as REF- input */
        for (int index = 0; index < 17; index++) {
            meaning = FIFO_meaning[index];
            if (meaning == UNIAIN14)
                FIFO_meaning[index] = UNDEFINED;
            if (meaning == UNIDIF1415)
                FIFO_meaning[index] = UNDEFINED;
            if (meaning == BIPDIF1415)
                FIFO_meaning[index] = UNDEFINED;
        }
    }
#endif

}
//---------------------------------------------------------------------------
bool MAX1258::GPIO_Outputs(int pins_mask)
{
    gpio_config = gpio_config | pins_mask;
```

Listing 2 (Sheet 8 of 14)

MAX1258 EV kit Listing 2 06/01/04 9
MAX1258EV listing2

```
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_GPIO_CONFIG),
        (unsigned __int8)(gpio_config / 0x100),
        (unsigned __int8)(gpio_config & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        return true; // success
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::GPIO_Inputs(int pins_mask)
{
    //~ To configure a pin for input requires writing BOTH
    //~ a MAX1258_GPIO_CONFIG with bit = 0 and also
    //~ a MAX1258_GPIO_WRITE with bit = 1.
    //~ Writing CONFIG = 0 and WRITE = 0 will configure the pin for "open-drain pull-down" mode. Not input.

    gpio_config = gpio_config &~ pins_mask;
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_GPIO_CONFIG),
        (unsigned __int8)(gpio_config / 0x100),
        (unsigned __int8)(gpio_config & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        return GPIO_Write(gpio_data | pins_mask);
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
int MAX1258::GPIO_Read(int pins_mask)
{
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_GPIO_READ),
        (unsigned __int8)(0),
        (unsigned __int8)(0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        unsigned __int16 pins = miso_buf[1] * 0x100 + miso_buf[2];
        return (pins & pins_mask);
    } // else operation failed
    return 0;
}
//---------------------------------------------------------------------
bool MAX1258::GPIO_Write(int pins_mask)
{
    gpio_data = pins_mask;
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_GPIO_WRITE),
        (unsigned __int8)(gpio_data / 0x100),
        (unsigned __int8)(gpio_data & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        return true; // success
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::GPIO_Set(int pins_mask)
```

Listing 2 (Sheet 9 of 14)

MAX1258 EV kit Listing 2 06/01/04 10
MAX1258EV listing2

```
{
    return GPIO_Write(gpio_data | pins_mask);
}
//---------------------------------------------------------------------
bool MAX1258::GPIO_Clear(int pins_mask)
{
    return GPIO_Write(gpio_data &~ pins_mask);
}
//---------------------------------------------------------------------
bool MAX1258::DAC_No_Operation(void)
{
    // Send the DAC prefix with the no-operation command
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_NOP),
        (unsigned __int8)(0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_Reset_All_000(void)
{
    // Reset all DAC channels to all minimum output (all 0's)
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_RESET_000),
        (unsigned __int8)(0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_Reset_All_FFF(void)
{
    // Reset all DAC channels to all maximum output (all 1's)
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_RESET_FFF),
        (unsigned __int8)(0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_Write(int channel_number_12345678, int DAC_value)
{
    // Write the specified DAC channel(s) input register, without changing the output value
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(DAC_value >> 8 & 0x0F),
        (unsigned __int8)(DAC_value & 0xFF)
    };
```

Listing 2 (Sheet 10 of 14)

MAX1258 EV kit Listing 2 06/01/04 11
MAX1258EV listing2

```
    switch (channel_number_12345678) {
    case 1: mosi[1] |= MAX1258_DACc_WRITE1; break;
    case 2: mosi[1] |= MAX1258_DACc_WRITE2; break;
    case 3: mosi[1] |= MAX1258_DACc_WRITE3; break;
    case 4: mosi[1] |= MAX1258_DACc_WRITE4; break;
    case 5: mosi[1] |= MAX1258_DACc_WRITE5; break;
    case 6: mosi[1] |= MAX1258_DACc_WRITE6; break;
    case 7: mosi[1] |= MAX1258_DACc_WRITE7; break;
    case 8: mosi[1] |= MAX1258_DACc_WRITE8; break;
    default:
        return false; // invalid channel mask
    }
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_Load_channel(int channel_number_12345678)
{
    // Update the specified DAC channel(s) output value from the corresponding input register, changing the output
value
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_LOAD),
        (unsigned __int8)(0)
    };
    switch (channel_number_12345678) {
    case 1: mosi[2] |= MAX1258_DACd_CH1; break;
    case 2: mosi[2] |= MAX1258_DACd_CH2; break;
    case 3: mosi[2] |= MAX1258_DACd_CH3; break;
    case 4: mosi[2] |= MAX1258_DACd_CH4; break;
    case 5: mosi[1] |= MAX1258_DACc_CH5; break;
    case 6: mosi[1] |= MAX1258_DACc_CH6; break;
    case 7: mosi[1] |= MAX1258_DACc_CH7; break;
    case 8: mosi[1] |= MAX1258_DACc_CH8; break;
    default:
        return false; // invalid channel mask
    }
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_Load_channels(int channel_mask)
{
    // Update the specified DAC channel(s) output value from the corresponding input register, changing the output
value
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_LOAD),
        (unsigned __int8)(0)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
```

Listing 2 (Sheet 11 of 14)

MAX1258 EV kit Listing 2 06/01/04 12
MAX1258EV listing2

```
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_Write_Load_CH1234(int value)
{
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)((value >> 8 & 0x0F) | MAX1258_DACc_WRITE14LOAD),
        (unsigned __int8)(value & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_Write_Load_CH5678(int value)
{
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)((value >> 8 & 0x0F) | MAX1258_DACc_WRITE58LOAD),
        (unsigned __int8)(value & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_Write_Load_All(int value)
{
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)((value >> 8 & 0x0F) | MAX1258_DACc_WRITE18LOAD),
        (unsigned __int8)(value & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_Write_All(int value)
{
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)((value >> 8 & 0x0F) | MAX1258_DACc_WRITE18),
        (unsigned __int8)(value & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
```

Listing 2 (Sheet 12 of 14)

MAX1258 EV kit Listing 2 06/01/04 13
MAX1258EV listing2

```
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------------
bool MAX1258::DAC_PowerOn_channels(int channel_mask)
{
    // Power-on the specified DAC channel(s) without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_PWR),
        (unsigned __int8)(MAX1258_DACd_PWR_ON)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------------
bool MAX1258::DAC_PowerOff_HiZ_channels(int channel_mask)
{
    // Power-off the specified DAC channel(s) high-impedance without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_PWR),
        (unsigned __int8)(MAX1258_DACd_PWR_OFF)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------------
bool MAX1258::DAC_PowerOff_Vref100k_channels(int channel_mask)
{
    // Power-off the specified DAC channel(s) VREF-100kohm without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_PWR),
        (unsigned __int8)(MAX1258_DACd_PWR_OFF_100K_VREF)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
```

Listing 2 (Sheet 13 of 14)

```
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_PowerOff_Gnd100k_channels(int channel_mask)
{
    // Power-off the specified DAC channel(s) GND-100kohm without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_PWR),
        (unsigned __int8)(MAX1258_DACd_PWR_OFF_100K_AGND)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
bool MAX1258::DAC_PowerOff_Gnd1k_channels(int channel_mask)
{
    // Power-off the specified DAC channel(s) GND-1kohm without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_PWR),
        (unsigned __int8)(MAX1258_DACd_PWR_OFF_1K_AGND)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//---------------------------------------------------------------------
```

Listing 2 (Sheet 14 of 14)

Revision History

Pages changed at Rev 1: 1, 2, 3, 6, 13–16, 40

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for Data Conversion IC Development Tools ***category:***

Click to view products by Maxim ***manufacturer:***

Other Similar products are found below :

EVAL-AD7265EDZ EVAL-AD7719EBZ EVAL-AD7767-1EDZ EVAL-AD7995EBZ AD9211-200EBZ AD9251-20EBZ AD9251-65EBZ AD9613-170EBZ AD9629-20EBZ AD9716-DPG2-EBZ AD9737A-EBZ AD9993-EBZ DAC8555EVM EVAL-AD5061EBZ EVAL-AD5062EBZ EVAL-AD5443-DBRDZ EVAL-AD5570SDZ EVAL-AD7992EBZ EVAL-AD7994EBZ AD9119-MIX-EBZ AD9233-125EBZ AD9629-80EBZ AD9650-80EBZ AD9767-EBZ DAC8531EVM LM96080EB/NOPB EVAL-AD5445SDZ EVAL-AD5660EBZ EVAL-AD7685SDZ EVAL-AD7687SDZ MAX5318PMB# MAX1246EVL11-QSOP MAX117EVKIT-DIP DC2365A-C DC2795A-B DC2795A-A DAC088S085EB/NOPB SIM8909-EVB-KIT 82635ASRCDVKHV 961443 DC1466B-B EVAL-AD5413SDZ ADC12D1600RB/NOPB 1083 RFPDK FOR CMT2X5X TS7003DB TSC2014EVM-PDK MOD-USB3G KDC5514EVALZ 650201392G ISL28005FH-100EVAL1Z