19-3324; Rev 0; 6/04

MAXIM

MAX1258の評価キット/評価システム

概要

MAX1258評価システム(EVシステム)は、MAX1258評価キット(EVキット)及びマキシム68HC16MODULE-DIPマイクロコントローラ(µC)モジュールから構成されています。MAX1258は多チャネル、12ビットのアナログ-ディジタルコンバータ(ADC)、温度センサ、8チャネル、12ビットのディジタル-アナログコンバータ(DAC)、及び構成を変更可能な汎用I/Oポート(GPIO)を提供します。評価ソフトウェアは、Windows® 95/98/2000/XPで動作しMAX1258の機能を実行するために便利なユーザインタフェースを提供します。

PCを利用してMAX1258の総合的な評価を行う場合には、完全なEVシステム(MAX1258EVC16)をご注文ください。既に他のEVシステムとともに68HC16MODULEモジュールを購入されている場合、または他のマイクロコントローラベースのカスタム利用の場合には、EVキット(MAX1258EVKIT)をご注文ください。

このシステムを用いるとMAX1057/MAX1058/MAX1257を評価することも可能です。詳細は、「ハードウェアの詳細」の項を参照してください。これらのICの無料サンプルについては、お問い合わせください。

MAX1258スタンドアロンEVキット

MAX1258EVキットはMAX1258を容易に評価するための実証済のプリント基板レイアウトを提供します。キットの正常な動作のためには適切なタイミング信号とインタフェースを取る必要があります。ユーザ供給の6VDC～28VDC電源とグランドリターンを端子ブロックTB1に接続して、ボードに搭載している低ドロップアウト(LDO)のMAX1615に、給電してください(図7参照)。また、タイミング要件に関しては、MAX1258のデータシートを参照してください。

MAX1258 EVシステム

MAX1258 EVシステムはユーザ供給の7VDC～20VDC電源で動作します。評価用ソフトウェアは、IBM PCのWindows 95/98/2000/XPの上で動作し、コンピュータのシリアル通信ポートを通してEVシステムボードとインタフェースを取ります。セットアップと操作方法については「クイックスタート」の項を参照してください。

特長

- ♦ 実証済のプリント基板レイアウト
- ♦ 完全な評価システム
- ♦ ボードに搭載した便利なテストポイント
- ♦ データロギング用ソフトウェア
- ♦ 完全実装及び試験済

型番

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

注：MAX1258評価ソフトウェアは、完全な評価システムMAX1258EVC16(MAX1258EVKITと組み合わせた68HC16MODULE-DIPモジュールが含まれています)で使われるように設計されています。もしMAX1258評価ソフトウェアを使わない場合は、µCの無いMAX1258EVKITボード単体で購入することも可能です。

MAX1258EVC16システム部品リスト

PART	QTY	DESCRIPTION
MAX1258EVKIT	1	MAX1258 evaluation kit
68HC16MODULE-DIP	1	68HC16 µC module

部品サプライヤ

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

注: 部品サプライヤに連絡する際は、MAX1258を使用している旨を伝えてください。

WindowsはMicrosoft Corporationの登録商標です。

MAX1258EVキット部品リスト

REFERENCE	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
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, ABZD, SOT23-5
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	PC board, MAX1258 EV kit

クイックスタート

必要な装置

評価を始める前に以下に示す装置を用意してください。

- マキシムMAX1258EVC16(MAX1258EVKITボードと68HC16MODULE-DIPからなる)
- 0.25Aを供給可能な+7V～+20VDC電源
- シリアル(COM)ポートが使用可能なWindows 95/98/2000/XP PC
- 9ピンI/O(シリアル)用の延長ケーブル

手順

以下に示す、すべての接続を完了するまでは電源を投入しないでください。

1) JU1が5Vの位置、JU2が1-2の位置、そしてJU5が閉じられていることを確認してください。JU4はオープンとしてください。ジャンパ位置JU3は、以前に切り離されていなければ閉じられています。表2-6を参照してください。

2) MAX1258 EVキットの40ピンヘッダと68HC16MODULE-DIPモジュールの40ピンコネクタを整列させて注意深く接続してください。このためには静かにお互いを押し付けます。2つのボードはお互いにぴったりくっついていなければなりません。

3) µCの端子ブロックへ+7V～+20VのDC電源を接続してください。端子ブロックは、ON/OFFスイッチの隣でµCモジュールの上面エッジに沿った位置にあります。基板上に記された極性マークをよく見てください。

4) コンピュータのシリアルポートからµCモジュールにケーブルを接続してください。9ピンのシリアルポートを使う場合は、ストレート型のオス-メスケーブルを使用してください。25ピンコネクタのシリアルポートしか使えない場合、25ピン-9ピンの標準変換アダプタを必要とします。EVキットのソフトウェアは、正しいポートが接続されていることを確認するために、モデムのステータスライン(CTS、DSR、DCD)をチェックします。

5) フロッピーディスクにあるINSTALL.EXEプログラムを走らせてコンピュータに評価ソフトウェアをインストールしてください。プログラムファイルがコピー

され、Windowsのスタートメニューにアイコンが作られます。

6) 電源をオンしてください。

7) スタートメニューのアイコンを開いて、MAX1258プログラムを開始してください。

8) プログラムは、μCモジュールの接続と、その電源をオンすることを促します。SW1をON位置にスライドしてください。正しいシリアルポートを選択し、OKをクリックしてください。するとプログラムは、自動的に、そのソフトウェアをμCモジュールにダウンロードします。

9) 入力信号をAIN0に接続して、「Perform Action」をクリックしてください。スクリーン上にそれが読み出されることを確認してください。動作を繰り返すためには、「every200ms」ボックスをチェックしてください。

10)測定結果のグラフを見るには、「View」メニューをプルダウンして、「Graph」をクリックしてください。

11)「DAC Outputs」タブを選択して、アクション「1111 cccc cccc 001 x Power On Selected Channels」を選択して、「Perform Action」をクリックしてください。

12)アクション「1100 Write and Load OUT0-OUT7」を選択して、OUT1コードを2048にセットし、「Perform Action」をクリックしてください。OUT0の電圧が、中間スケール(V_{REF} = 4.096Vの場合、2.048V)になることを確認してください。

13)「GPIO Pins」タブを選択してください。GPIOA0を「High Output」に設定し、GPIOB0を「Low Output」に設定し、GPIOC3を入力に設定して、「Write Output Pins」をクリックしてください。A0端子がロジックハイで、B0端子がロジックローになることを確認してください。

14)GPIO端子C3をロジックハイ(A0)に接続して、「Read Input Pins」をクリックしてください。C3がハイであることを、ソフトウェアが示しているか確認してください。

15)GPIO端子C3をロジックロー(B0)に接続して、「Read Input Pins」をクリックしてください。C3がローであることを、ソフトウェアが示しているか確認してください。

ソフトウェアの詳細

評価ソフトウェアの主ウィンドウは、データコンバータの設定を行い、アナログ入力電圧を測定します。「Action」を使って走査シーケンス、繰返し変換、または単一変換を選択してください。各選択されたチャネルの測定結果は、対応する「Measurement Results」フィールドに表示されます。

「read single channel repeatedly」設定の「Action」では、選択したチャネルを何回、繰り返して測定するかを決定するための「Repetition」を選択してマークする必要があります。

選択した各測定チャネルの複数回の測定結果の算術平均値を得るには「Averaging」を用いてください。「Repetition」は、「Averaging」と組み合わせて、多数の測定結果を、少数のサンプル平均として要約することに使えます。

「Low-Level Interface Details」のパネルは、最新のローレベルレジスタの内容を示します。各レジスタに書き込まれた要約は、「Low-level registers」タブで利用できます。

AIN14とAIN15チャネルは、もしその代替機能がセットアップレジスタで選択された場合には、自動的にスキップされます。評価ソフトウェアは、代替機能がイネーブルまたはディセーブルされた場合は、これらのチャネルの表示/非表示を更新します。

「Setup」タブで、AIN14とAIN15端子の代替機能を設定し、また、隣接チャネルを差動入力ペアとして設定します。

「Low-level registers」タブには、アクティブな構成を作り出すコマンド群が要約されています。「Reset All Registers」ボタンは、これらのソフトウェアでシャドウされたレジスタ値をリセットし、MAX1258にリセットコマンドを送出します。オプションとして、スローモードとバンドギャップモードを構成する方法があります。

DAC出力

メインウィンドウにある「DAC Outputs」タブは、アナログ出力を制御します。

使用する前にDACチャネルに給電するためには、アクション「1111 cccc cccc 001x Power On Selected Channels」を選択してください。チェックボックスが設定されていることを確認して、「Perform Action」をクリックしてください。ジャンパJU2は、DAC端子の初期状態を決定します。

すべてのDAC出力をゼロにリセットするには、アクション「0001 0... Reset all DACs to 000(zero scale)」を選択して、「Perform Action」をクリックしてください。

すべてのDAC出力をフルスケールにリセットするには、アクション「0001 1... Reset all DACs to FFF(full scale)」を選択して、「Perform Action」をクリックしてください。

注：電源投入時には、DACはREF1端子に外部リファレンスが入力されていることを前提にしています。この点に関しては、MAX1258データシートを参照してください。

複数のDACチャネルに書込みを行い、出力させるためには、アクション「1100 Write and Load OUT0-Out7」を選択して、必要とする出力コード値(0～4095)を「OUT0」エディットフィールドにタイプイン

して、「Perform Action」をクリックしてください。OUT0～OUT7端子の電圧は、即座に新しい値に変化します。

1個のDACに書き込む(例としてOUT5)には、アクション「0110 Write OUT5」を選択してください。必要とするコード値(0～4095)を「OUT5」エディットフィールドにタイプインして、「Perform Action」をクリックしてください。ジャンパJU5が閉じられていなければ(LDAC端子がロー)、入力レジスタから端子電圧を更新するためには、別の書込みコマンドが必要です。アクション「1110 cccc cccc xxxx Load Selected Channels」を選択して、チャネル5のロードチェックボックスをチェックし、「Perform Action」をクリックすることによって、OUT5に書き込んでください。

GPIO端子

メインウィンドウの「GPIO Pins」のタブは、汎用のディジタル入力/出力端子の設定、書込み、読込みをします。

各GPIO端子は、その端子を「High Output」、「Low Output」、「Input」、または「Open-Drain Pull-Down」モードに設定するドロップダウンのコンボボックスを持っています。出力モードを選択して「Write Output Pins」をクリックしてください。「Read Input Pins」をクリックして端子の状態を読み込みます。

サンプリング

測定データを、外部クロックモードを用いてサンプルすることができます。「Setup」タブから「Clock Mode」を、「0111xxxx ext clock」に設定してください。その後、「Measurement」タブに戻り、「Get Samples」をクリックしてください。

グラフウィンドウ

最新の測定データを見るには、「View」メニューをドロップダウンして、「graph」を選択してください。データは時間シーケンス図、ヒストグラム図、または原データの表として見ることができます(図6を参照)。使うことができるグラフコマンドは表1を参照してください。

診断用ウィンドウ

診断用ウィンドウは、評価キットを出荷する前に行われる工場での試験で使われ、ユーザが使用することは意図されていません。

ハードウェアの詳細

テストされるMAX1258デバイス(U1)は、マルチチャネル12ビットADC、1個の温度センサ、8チャネル12ビットDAC、及び構成を変更可能なGPIOを提供します。抵抗R1～R16とコンデンサC1～C16は、各入力に対する一次のローパスアンチエイリアスフィルタを構成します。コンデンサC17は、U1の電源バイパス用です。図7とMAX1258のデータシートを参照してください。

EVキットはMAX1615 3V/5Vリニアレギュレータ(U2)と1セットのMAX1840/MAX1841レベルシフタ(U3とU4)を備え、5VのμCと3VのMAX1257の使用をサポートします。

MAX1257の評価

MAX1257はMAX1258の3Vバージョンです。無料サンプルは、MAX1257BETMをリクエストしてください。U1をMAX1257と交換し、JU1のシャントを3V位置に移動してください。そしてソフトウェアの「options」メニューで、「reference = 2.500V」を選択してください。

MAX1057の評価

MAX1057はMAX1257の3V、10ビットバージョンです。無料サンプルはMAX1057BETMをリクエストしてください。U1をMAX1057と交換してJU1のシャントを3Vの位置に移動してください。ソフトウェアの「Options」メニューで、「reference = 2.500V」を選択してください。

評価用のソフトウェアは12ビットのデータを期待していますが、MAX1057は10ビットの有意データを提供します。最上位ビット(MSB)の位置は同じ位置なので、ソフトウェアが表示するコード値は、実際の測定値の4倍となります。再生される電圧値は変わりません。その「Options」メニューを選択して、「Sub-LSBs」を2に設定して、グラフウィンドウを調整してください。

MAX1058の評価

MAX1058はMAX1258の10ビットバージョンです。無料サンプルはMAX1058BETMをリクエストしてください。U1をMAX1058に置き換え、JU1の位置を5V位置に移動してください。

評価用のソフトウェアは12ビットのデータを期待していますが、MAX1058は10ビットの有意のデータを提供します。最上位ビット(MSB)の位置は同じ位置なので、ソフトウェアが表示するコード値は、実際の測定値の4倍となります。再生される電圧値は変わりません。その「Options」メニューを選択して、「Sub-LSBs」を2に設定して、グラフウィンドウを調整してください。

外部リファレンスの使用

すべてのADCとDACは変換を実行するためにリファレンスが必要です。MAX1258のADC用には内部リファレンスを使用しながら、DACに対してはシングルエンドの外部リファレンスを使用することが可能です。これがデフォルトモードです。電源をオフにして、DACの外部リファレンスをREF1に接続してください。その後電源をオンにし、評価用のソフトウェアを動作させてください。「Setup」

タブにより、「Reference Input」を、「01xx10xx Pin 48=AIN14, ADCREF=Internal, DACREF=REF1」と設定してください。JU4にシャントを装着して、ボードに搭載したREF1のバイパスコンデンサC14を接続してください。ソフトウェアがDACのコード値に対応する電圧を計算する場合は、常にその計算は、ユーザが用意した「REF1 pin voltage」の値に依存します。

ADCとDACの両方に対して内部リファレンスを使う場合は、「Setup」タブを使い、「Reference Input to 01xx00xx Pin 48=AIN14, ADCREF=Internal, DACREF=Internal」と設定してください。内部リファレンスを使う場合は、JU4はオープンにしてください。

DACとADCに対して、別々のシングルエンドリファレンスを印加することが可能です。電源をオフにして、DAC外部リファレンスをREF1に、ADC外部リファレンスをREF2に接続してください。その後システムに給電して、評価用ソフトウェアを動作させてください。「Setup」タブにより、「Reference Input」を、「01xx01xx Pin48= REF2, ADCREF=REF2, DACREF=REF1」と設定してください。JU4にシャントを装着して、ボードに搭載したREF1のバイパスコンデンサC14を接続してください。ソフトウェアがDACのコード値に対応する電圧を計算する場合は、常にその計算は、ユーザが用意した「REF1 pin voltage」の値に依存します。ソフトウェアがADCのコード値に対応する電圧を計算する場合は、常にその計算はユーザが用意した「REF2 pin voltage」の値に依存します。

ADCには差動の外部リファレンスを印加しDACにはシングルエンドの外部リファレンスを印加することができます。電源をオフにして、DACリファレンスをREF1

表1. グラフツールボタン

ツール	機　能
	すべての使用可能な入力レンジを示します。
	表示ウィンドウ枠一杯にグラフを拡大して表示します。
	表示を左右に移動します。
	表示を上下に移動します。
	X軸を拡大または縮小します。
	Y軸を拡大または縮小します。
	データをファイルからロードします。
	データをファイルに保存します。
	データを保存するとき、ヘッダラインを書き込むオプションです。
	データを保存するとき、ライン数を書き込むオプションです。
	データをコード 対 時間の変化として表示します。
	ヒストグラム表示を行います(各コード値の累積頻度)。
	テーブルを参照します。
Min	最小値を表形式で示します。
Max	最大値を表形式で示します。
Span	スパンを表形式で示します。 スパン = 最大値－最小値
N	サンプル数を表形式で示します。
Sum(x)	サンプルの総和を表形式で示します。
Sum(x*x)	サンプルの二乗和を表形式で示します。
Mean	算術平均値を表形式で示します。 $\text{Mean} = \frac{\Sigma(x)}{n}$

ツール	機　能
StdDev	標準偏差を表形式で示します。 $\text{標準偏差} = \sqrt{\frac{n\sum_{n}\left(x^2\right)-\left(\sum_{n}x\right)^2}{(n-1)n}}$
Rms	二乗平均の平方根(RMS)を表形式で示します。 $\text{RMS} = \sqrt{\frac{\Sigma\left(x^2\right)}{n}}$
0	チャネル0イネーブル
1	チャネル1イネーブル
2	チャネル2イネーブル
3	チャネル3イネーブル
4	チャネル4イネーブル
5	チャネル5イネーブル
6	チャネル6イネーブル
7	チャネル7イネーブル
8	チャネル8イネーブル
9	チャネル9イネーブル
10	チャネル10イネーブル
11	チャネル11イネーブル
12	チャネル12イネーブル
13	チャネル13イネーブル
14	チャネル14イネーブル
15	チャネル15イネーブル
16	チャネル16イネーブル(温度)

表2. ジャンパJU1(V_{DD}電圧の選択)

シャントの位置	V_{DD}の電圧	機　能
1-2*	5V	U1 = MAX1058またはMAX1258とした通常の動作
2-3	3V	U1 = MAX1057またはMAX1257とした通常の動作
開(Open)	未定義	JU1をオープンとしてキットを動作させないでください。

*デフォルト設定

表3. ジャンパJU2(RES_SEL)

シャントの位置	RES_SELの状態	機　能
1-2	ハイ(High)	電源が最初に投入されたとき、DAC出力は内部の100ｋΩの抵抗器を通してREF1に接続されます。すべてのDAC入力レジスタは0xFFFに設定されます。
2-3*	ロー(Low)	電源が最初に投入されたとき、DAC出力は内部の100kΩの抵抗器を通してAGNDに接続されます。すべてのDAC入力レジスタは0x000に設定されます。
開(Open)	駆動されない	JU2をオープンにしたまま電源を投入しないでください。

*デフォルト設定

表4. オプションのジャンパJU3 (AIN15の別機能)

JU3の状態	U1端子1の接続状態	機　能
閉(Closed)*	µCモジュールのJ1端子29に接続される(レベルシフタ経由)。	U1端子1 = $\overline{CNVST}$変換スタートコマンド。AIN15のパッドは無接続のままとしてください。
開(Open)	AIN15のパッドに接続される。	U1の端子1 = AIN15アナログ入力。信号源をAIN15のパッドに接続してください。

*デフォルト設定

に接続し、ADCリファレンスをREF1とREF2の間にREF1 > REF2となるように接続してください。その後でシステムに給電して、評価ソフトウェアを動作させてください。「Setup」タブを選択して、「Reference Input」を、「01xx11xx Pin 48=REF2, ADCREF=REF1-REF2, DACREF=REF1」と設定してください。JU4にシャントを装着して、ボードに搭載したREF1のバイパスコンデンサC14を接続してください。

表5. ジャンパJU4(REF1バイパス)

シャントの位置	REF1のバイパスコンデンサ	機　能
開(Open)	ユーザが用意	内部リファレンスを使う場合はJU4をオープンのままとしてください。
閉(Closed)*	C14がREF1をバイパス	外部リファレンスを使う場合はJU4を閉じてください。

*デフォルト設定

表6. ジャンパJU5($\overline{LDAC}$)

シャントの位置	$\overline{LDAC}$の状態	機　能
開(Open)	ハイ(High)	SPI経由のコマンド送信によりDAC出力を更新します。
閉(Closed)*	ロー(Low)	DAC入力レジスタのデータがDAC出力レジスタに転送されます。

*デフォルト設定

ソフトウエアがDACのコード値に対応する電圧を計算する場合は常に、その計算はユーザが用意した「REF1 pin voltage」と「REF2 pin voltage」の値に依存します。

トラブルシューティング

障害：出力を測定することができない。システムはゼロ電圧表示となっているか、または測定できないようだ。

解：V_{DD}の供給電圧をチェックしてください。ディジタルボルトメータを用いてリファレンス電圧をチェックしてください。オシロスコープを使って変換開始信号がストローブされるか確認してください。

障害：測定値に誤りがある、不安定、または不正確。

解：ディジタルボルトメータを用いてリファレンス電圧をチェックしてください。オシロスコープでノイズを観測してください。ノイズ測定のプロービングはオシロスコープのグランドリターンを可能な限り短くして、可能ならば、0.5インチ(10mm)以下としてください。

障害：トランスデューサを測定するとき、許容できない大きい誤差がある。

解：大部分の信号源はMAX1258のアナログ入力に直接、接続することができますが、ハイインピーダンスの信号源(300Ω以上)の場合は入力バッファを必要とするかもしれません。補足時間(内部クロックモード01)を長くしてセットリング誤差をチェックしてください。必要に応じてMAX4430をハイインピーダンス信号源のバッファとして使ってください。詳しくは、MAX1258のデータシートを参照してください。

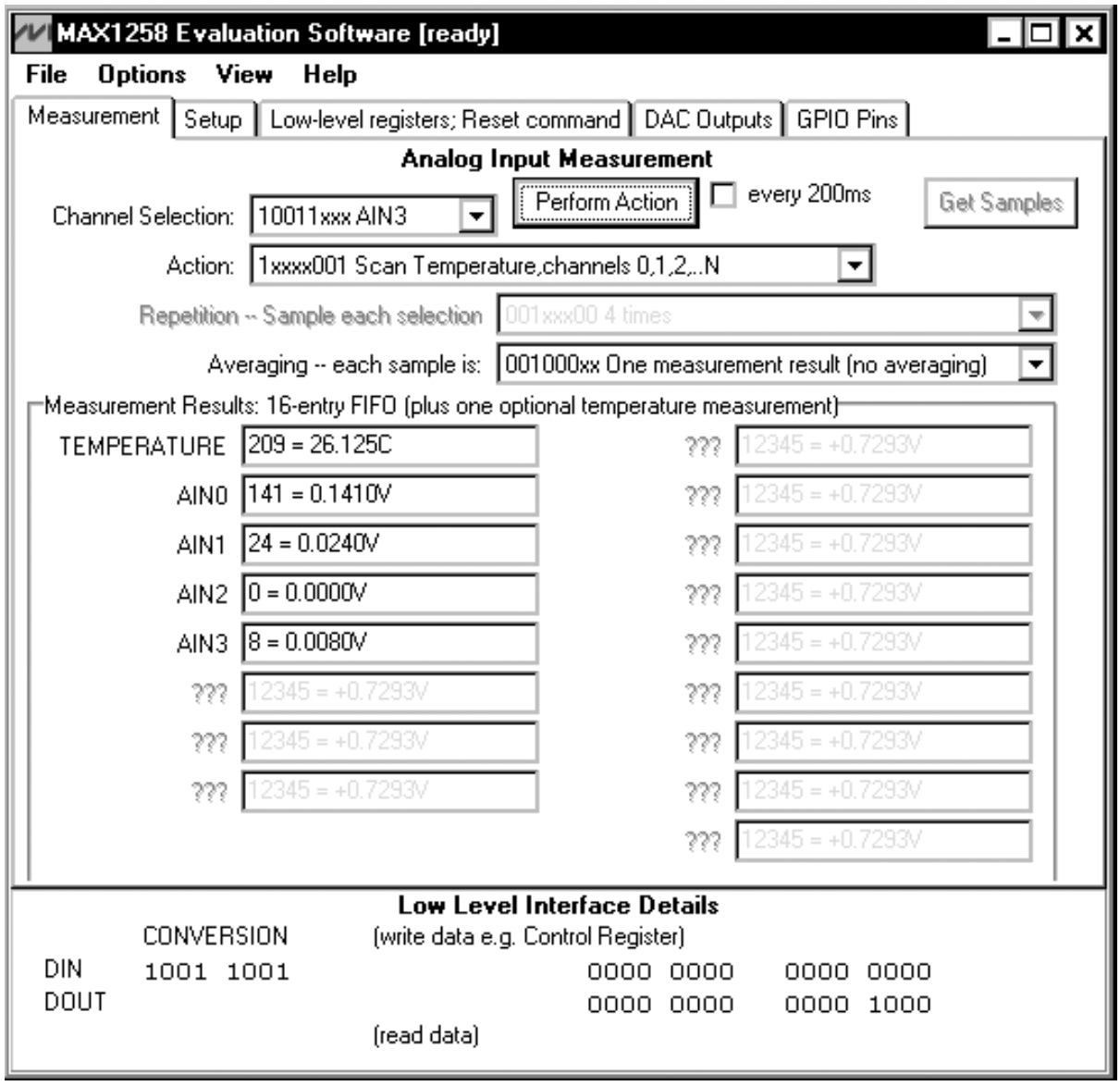

図1．MAX1258評価ソフトウェアのメインウィンドウ：データコンバータの設定とアナログ入力の測定をします。

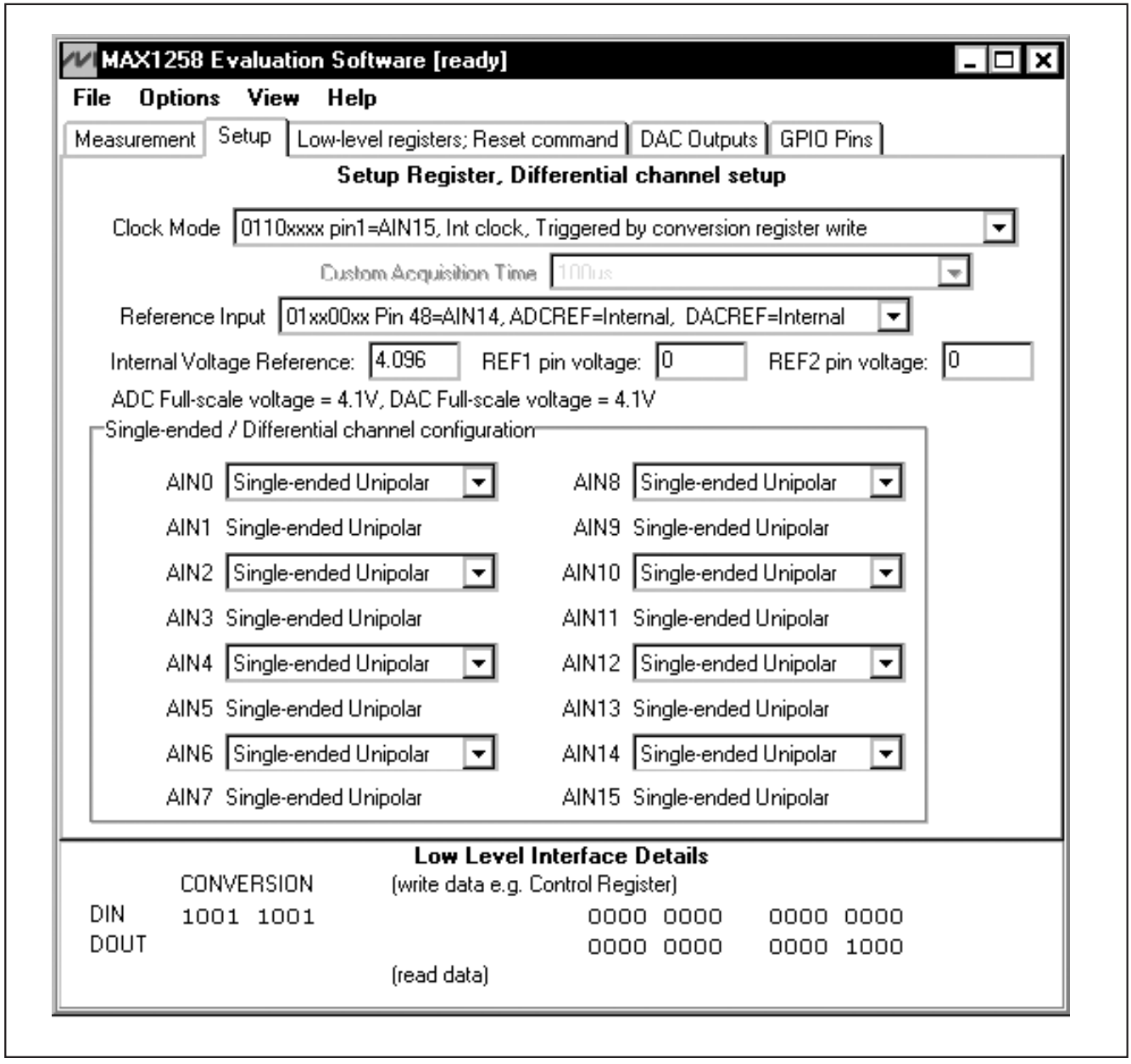

図2. メインウィンドウのセットアップタブ：AIN14とAIN15に対して代替機能を設定し、また、隣接するチャネルを作動入力ペアとして設定します。

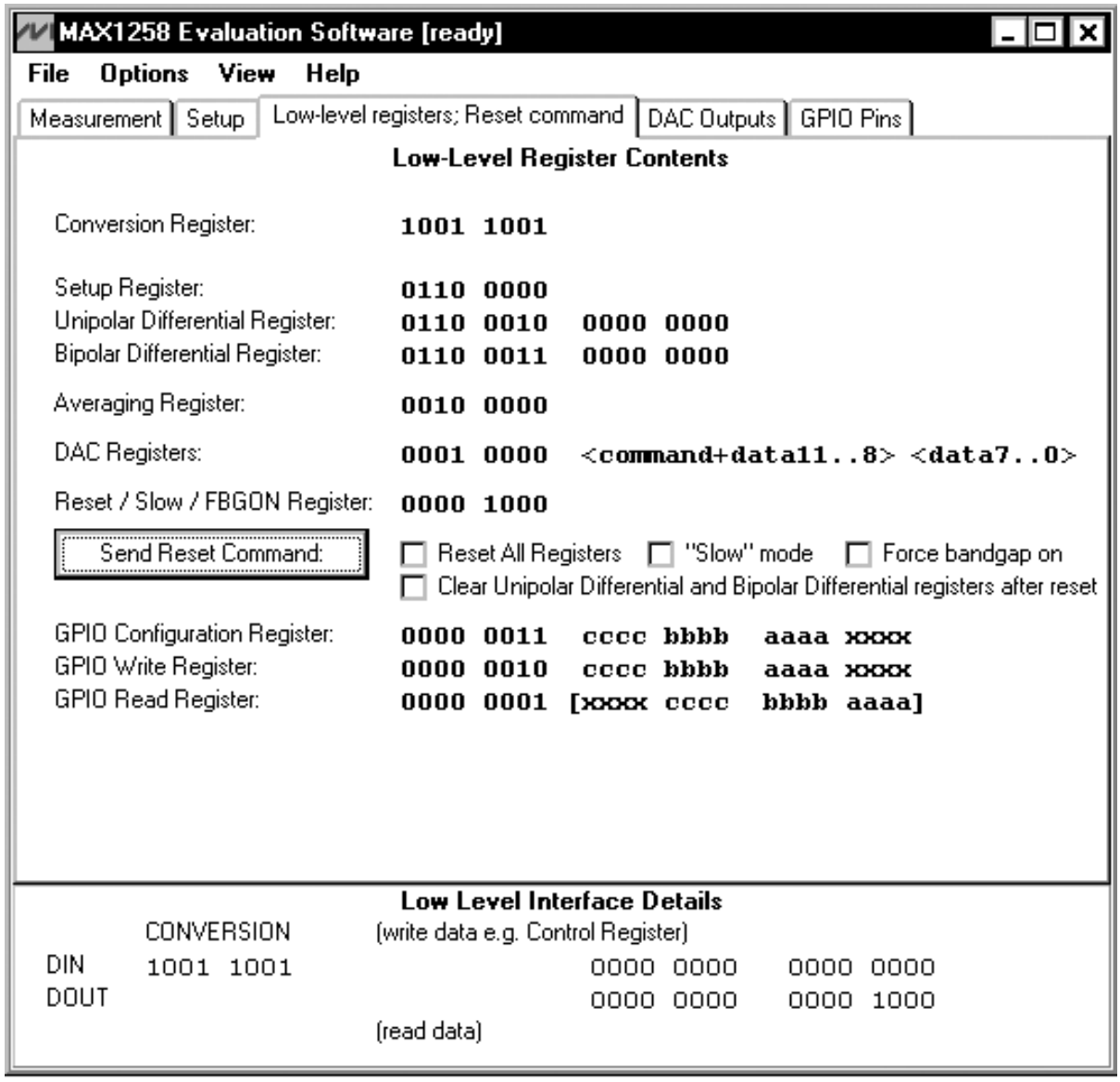

図3. メインウィンドウのローレベルレジスタタブ：アクティブコンフィギュレーションを作り出すコマンドを要約します。

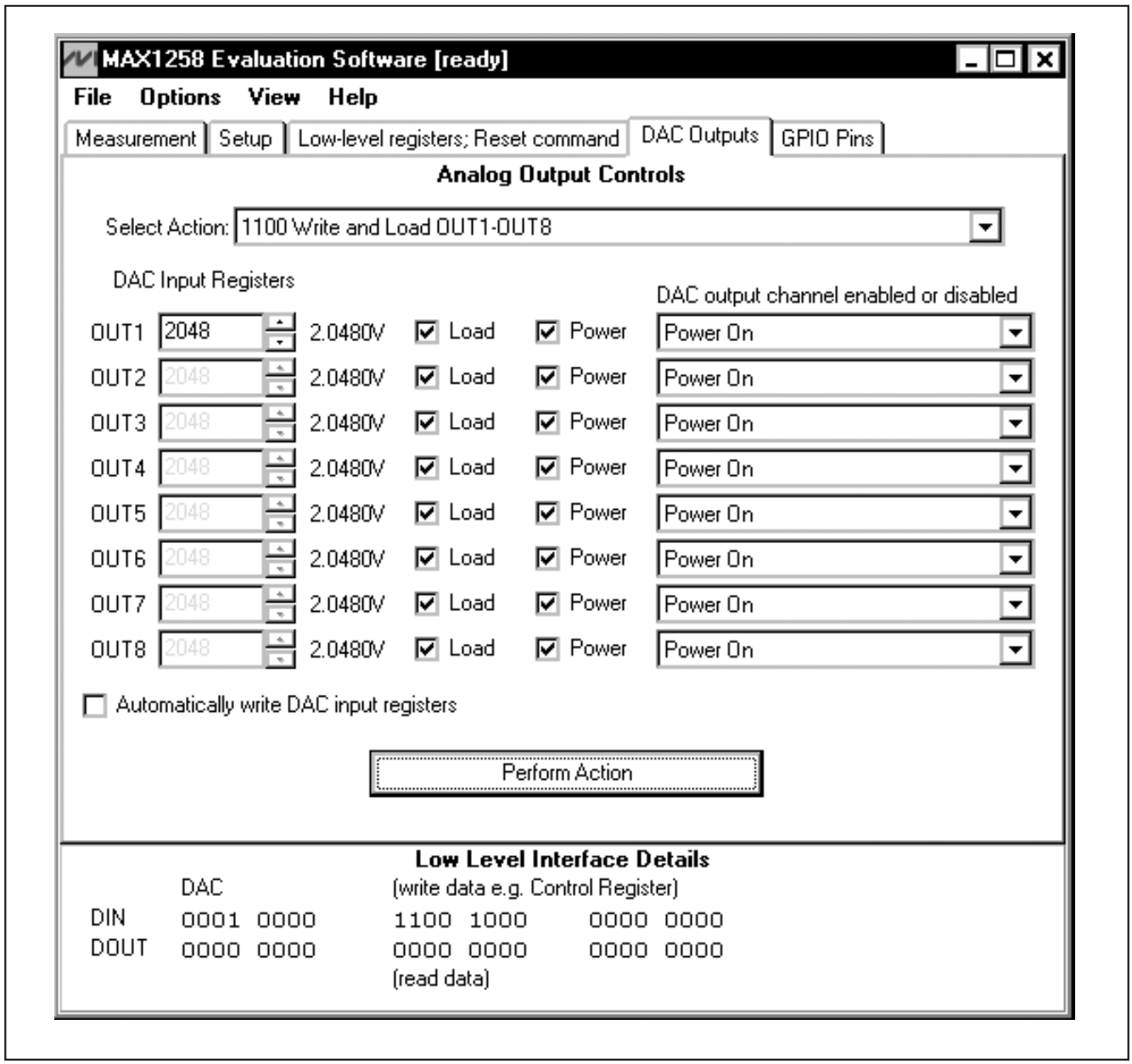

図4. メインウィンドウのDAC出力タブ：アナログ出力端子を制御します。

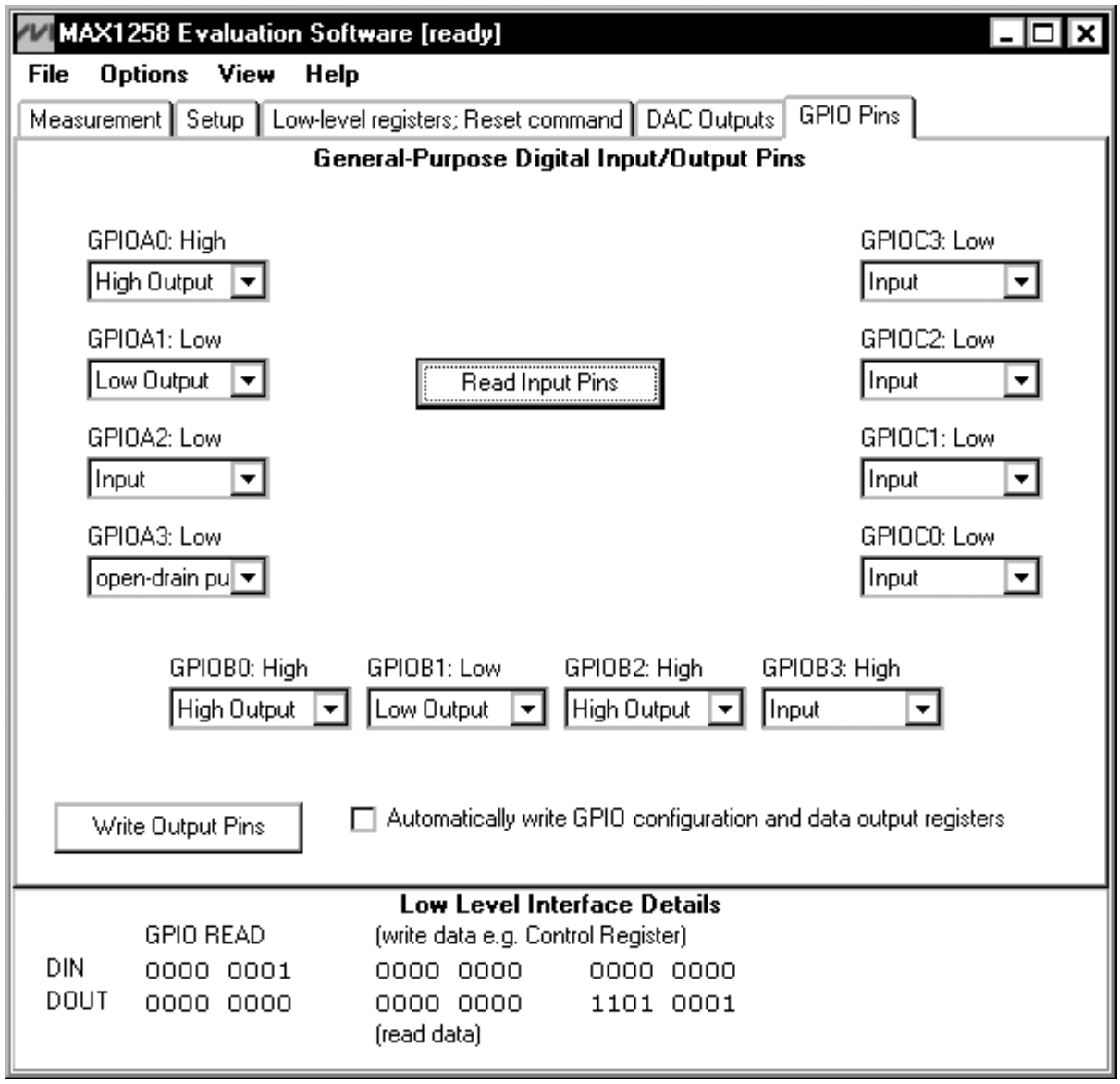

図5. GPIO端子タブ：ディジタルGPIO端子の設定、書込み、または読出しをします。

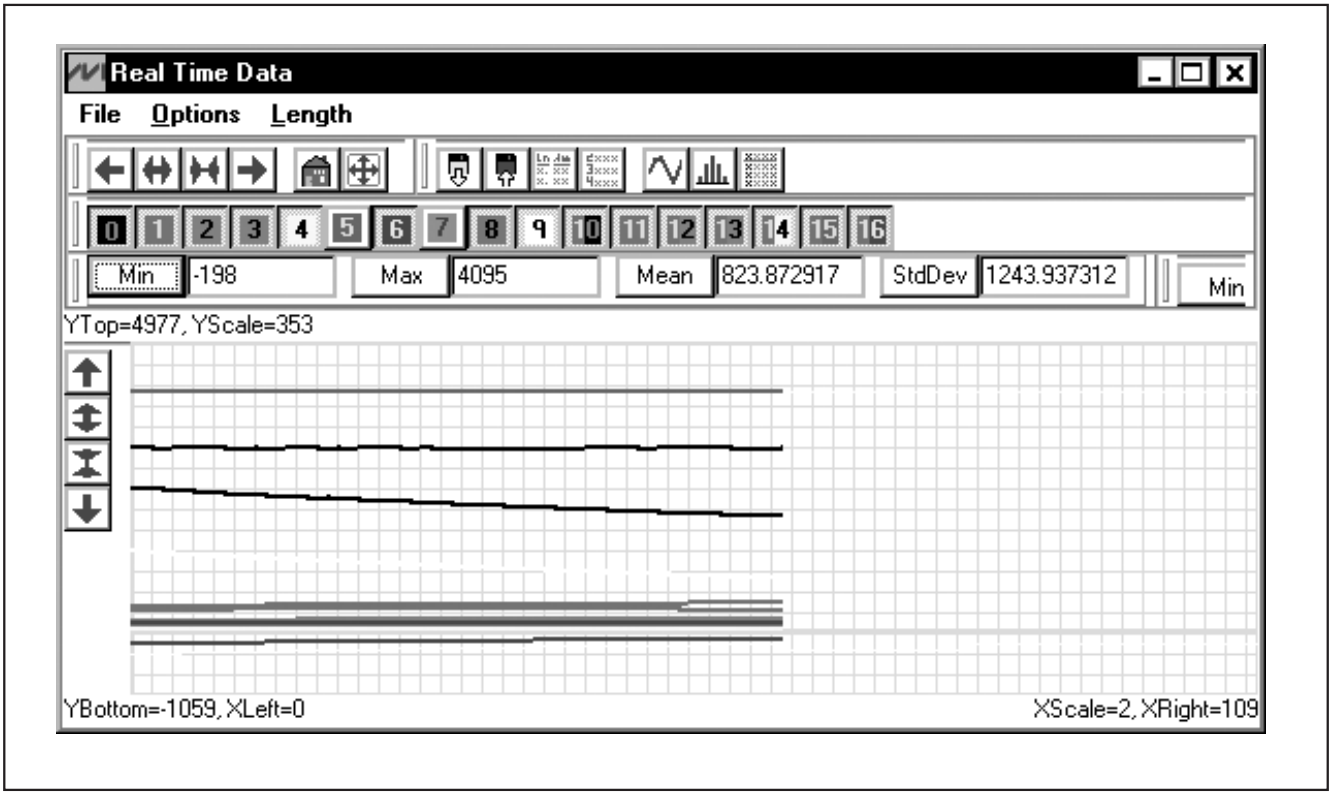

図6. リアルタイムデータとサンプリングデータのグラフ：時間シーケンス、ヒストグラム、または表形式でデータを表示します。

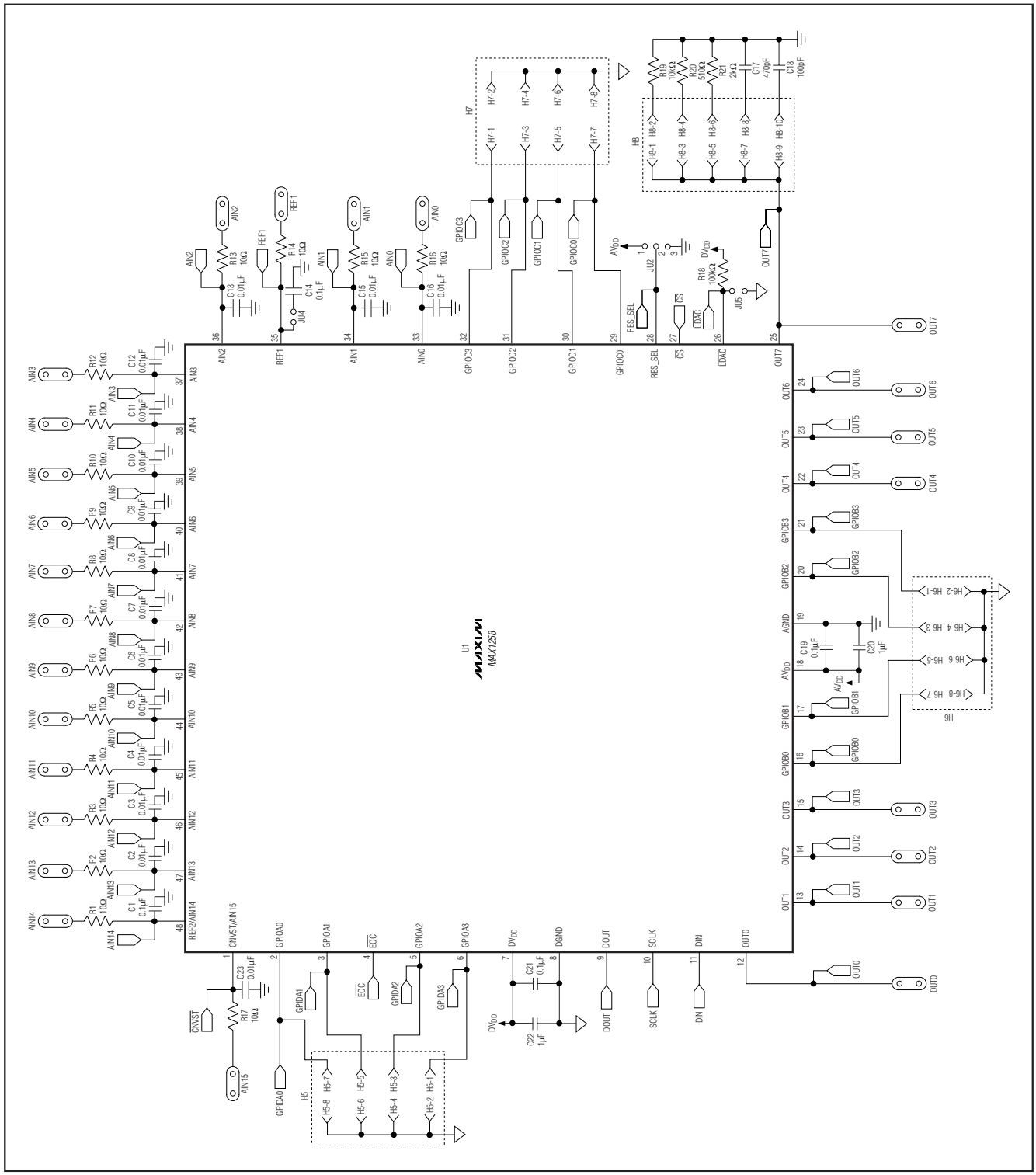

図7a. MAX1258 EVキットの回路図(シート1/2)

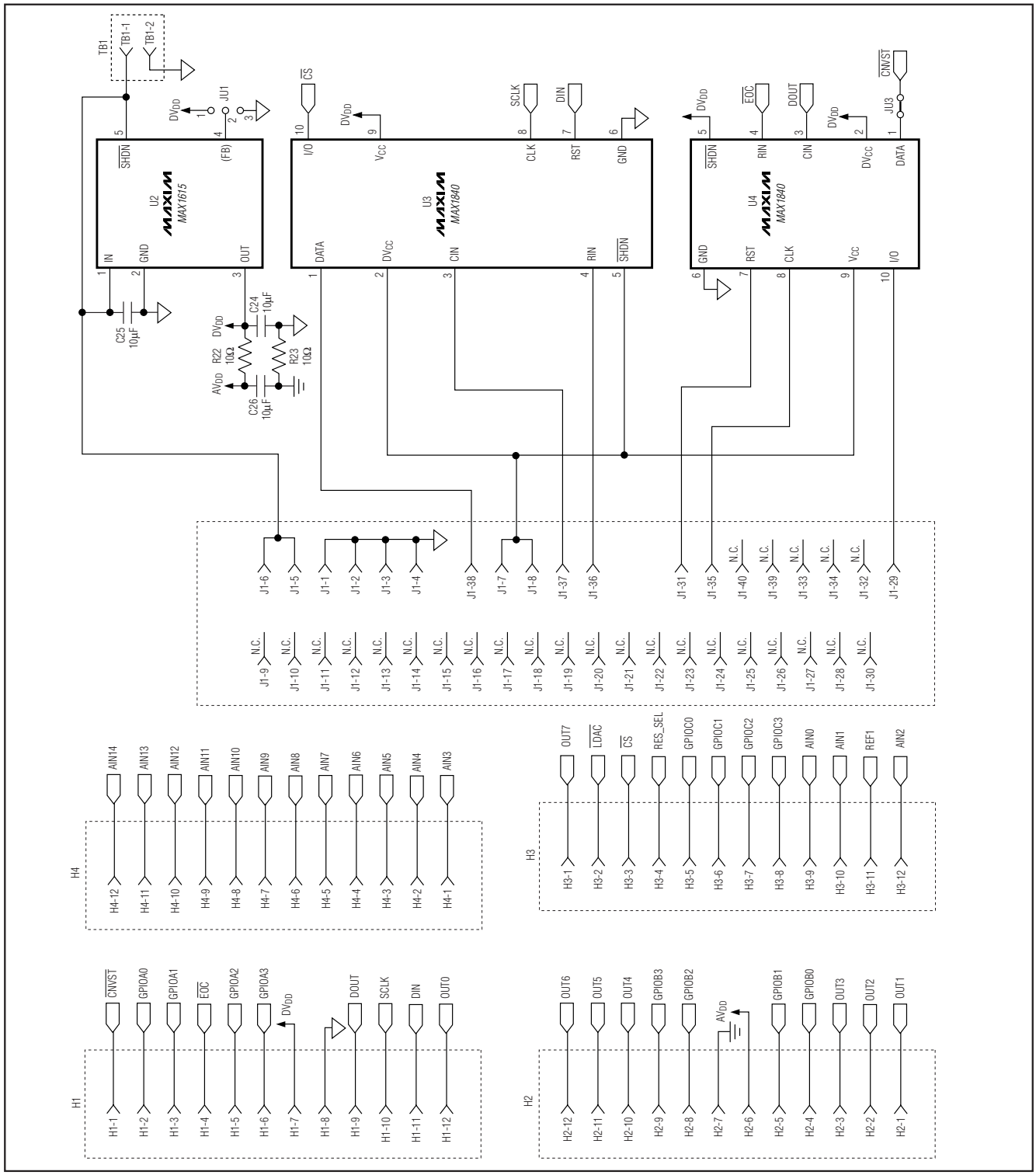

図7b. MAX1258 EVキットの回路図(シート2/2)

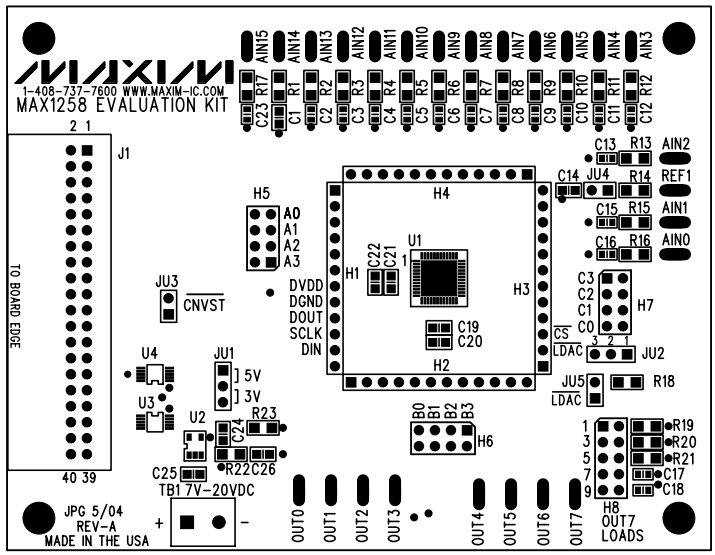

図8. MAX1258 EVキットの部品配置ガイド：部品面

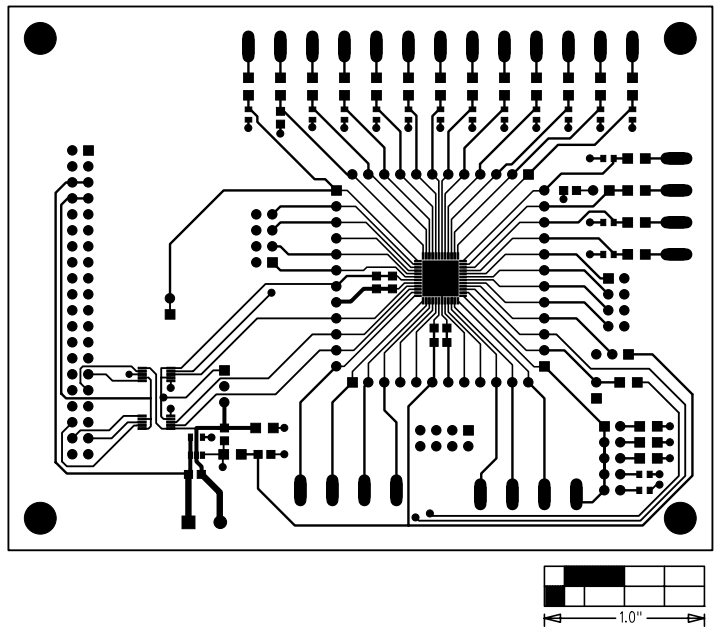

図9. MAX1258 EVキットのプリント基板レイアウト：部品面

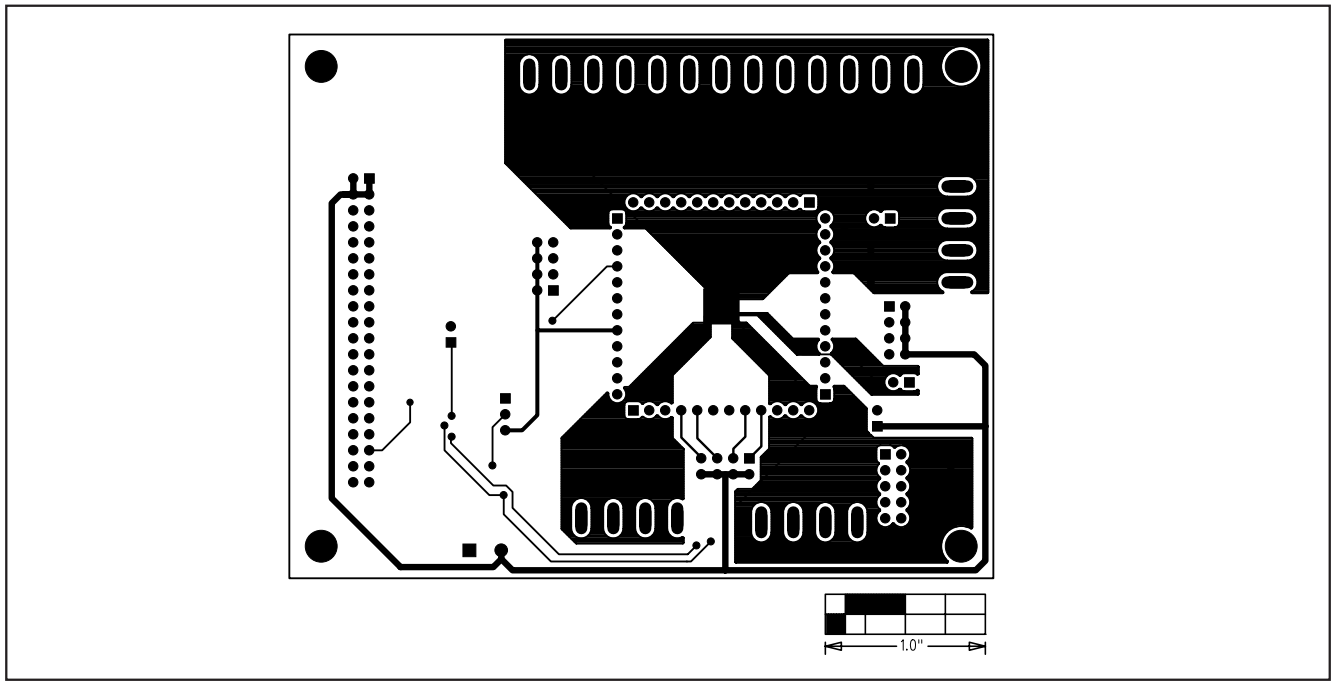

図10. MAX1258 EVキットのプリント基板レイアウト：半田面

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);
```

リスト1(シート1/10)

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
```

リスト1(シート2/10)

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
```

リスト1(シート3/10)

```
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
```

リスト1(シート4/10)

```
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)
```

リスト1(シート5/10)

```
// 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  // reset all input and DAC registers to 0x000
// 0 0 0 1  1 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*/
```

リスト1(シート6/10)

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
```

リスト1(シート7/10)

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[].
```

リスト1(シート8/10)

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);
```

リスト1(シート9/10)

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
```

リスト1(シート10/10)

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;
}
```

リスト2(シート1/14)

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();
    }
```

リスト2(シート2/14)

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),
```

リスト2(シート3/14)

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,
```

リスト2(シート4/14)

MAX1258 EV kit Listing 2 06/01/04 5
MAX1258EV listing2

```
        //      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)
```

リスト2(シート5/14)

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) {
```

リスト2(シート6/14)

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;
```

リスト2(シート7/14)

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;
```

リスト2(シート8/14)

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)
```

リスト2(シート9/14)

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)
    };
```

リスト2(シート10/14)

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)];
```

リスト2(シート11/14)

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;
```

リスト2(シート12/14)

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;
```

リスト2(シート13/14)

MAX1258 EV kit Listing 2 06/01/04 14
MAX1258EV listing2

```
    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;
}
//---------------------------------------------------------------------------
```

リスト2(シート14/14)

マキシム・ジャパン株式会社
〒169-0051東京都新宿区西早稲田3-30-16(ホリゾン1ビル)
TEL. (03)3232-6141 FAX. (03)3232-6149