LINEAR TECHNOLOGY

MARCH 2007 VOLUME XVII NUMBER 1

IN THIS ISSUE…

COVER ARTICLE

DESIGN FEATURES

DESIGN IDEAS

LINEAR TECHNOLOGY

Easy Drive Delta-Sigma ADCs Deliver Powerful Features *and* Reduce Design Effort

by Mark Thoren

Introduction

Easy Drive™ delta-sigma ADCs are rich in features but easy to use. The Easy Drive feature simplifies or eliminates active amplification or filtering at the inputs. Even the software interface is significantly less complicated than other ADCs (see sidebar on page 6). Overall, much of the traditional complexity around an ADC, such as external components and software timing, is simply gone, saving significant design time.

Table 1 lists the features of the 18 available Easy Drive devices, including 1-, 4- or 16-channel versions with I^{2}C or SPI interfaces. The 24-bit devices suit very high performance applications, while 16-bit devices are more general-purpose. A programmable gain amplifier (PGA) is available on the 16-bit devices for intermediate requirements or where several input ranges need to be accommodated.

Easy Drive Technology Simplifies Measurement of High Impedance Sensors

Delta-Sigma ADCs, with their high accuracy and high noise immunity, are ideal for directly measuring many

continued on page 3

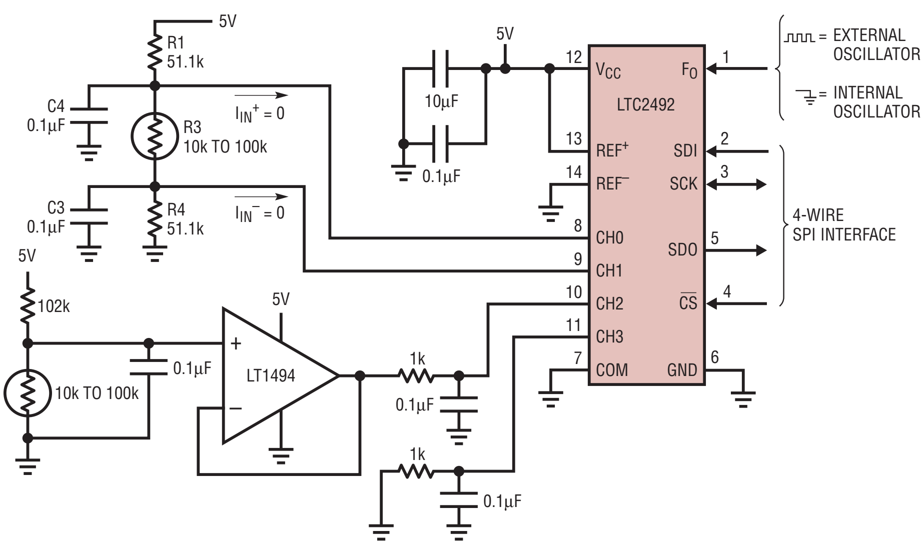

Figure 1. Easy Drive ADCs simplify measurement of high impedance sensors.

Easy Drive, continued from page 1

types of sensors. Nevertheless, input sampling currents can overwhelm high source impedances or low bandwidth, micropower signal conditioning circuits. Easy Drive solves this problem by balancing the input currents, thus simplifying or eliminating the need for signal conditioning circuits.

A common application for a delta-sigma ADC is thermistor measurement. Figure 1 shows two examples of thermistor digitization benefiting from Easy Drive technology. The first circuit (applied to input channels CH0 and CH1) uses two equal reference resistors that set the input common mode voltage equal to the reference common mode voltage and balance the differential input source resistance. If reference resistors R1 and R4 are exactly equal, the input current is zero and no errors result. If these resistors have a 1% tolerance, the maximum error in measured resistance is 1.6Ω due to a shift in common mode voltage, far less than the 1% error of the reference resistors themselves. No amplifier is required, making this an ideal solution in micropower applications.

Input sampling currents can overwhelm high source impedances or low-bandwidth, micropower signal conditioning circuits. Easy Drive solves this problem by balancing the input currents, thus simplifying or eliminating the need for signal conditioning circuits.

Easy Drive also enables very low power, low bandwidth amplifiers to drive the input of the LTC2492. As shown in Figure 1, CH2 is driven by an LT1494. The LT1494 has excellent DC specs for an amplifier with 1.5µA supply current, offering maximum offset voltage of 150µV and an open loop gain of 100,000. However, its 2kHz bandwidth makes it unsuitable for driving conventional delta-sigma ADCs. Adding a 1kΩ, 0.1µF filter solves this problem by providing a charge reservoir that supplies the LTC2492 instantaneous current, while the 1kΩ resistor isolates the capacitive load from the LT1494. The input sampling current of conventional delta-sigma ADCs leads to DC errors as a result

Table 1. Complete Easy Drive delta-sigma family

Part Number	# Inputs	Interface	Bits	Temp	PGA	2×	Package
LTC2480	1	SPI	16	✓	✓	✓	3mm × 3mm DFN
LTC2481	1	I²C	16	✓	✓	✓	3mm × 3mm DFN
LTC2482	1	SPI	16				3mm × 3mm DFN
LTC2483	1	I²C	16				3mm × 3mm DFN
LTC2484	1	SPI	24	✓		✓	3mm × 3mm DFN
LTC2485	1	I²C	24	✓		✓	3mm × 3mm DFN
LTC2486	2/4	SPI	16	✓	✓	✓	3mm × 4mm 14DFN
LTC2487	2/4	I²C	16	✓	✓	✓	3mm × 4mm 14DFN
LTC2488	2/4	SPI	16				3mm × 4mm 14DFN
LTC2489	2/4	I²C	16				3mm × 4mm 14DFN
LTC2492	2/4	SPI	24	✓		✓	3mm × 4mm 14DFN
LTC2493	2/4	I²C	24	✓		✓	3mm × 4mm 14DFN
LTC2494	8/16	SPI	16	✓	✓	✓	5mm × 7mm QFN
LTC2495	8/16	I²C	16	✓	✓	✓	5mm × 7mm QFN
LTC2496	8/16	SPI	16				5mm × 7mm QFN
LTC2497	8/16	I²C	16				5mm × 7mm QFN
LTC2498	8/16	SPI	24	✓		✓	5mm × 7mm QFN
LTC2499	8/16	I²C	24	✓		✓	5mm × 7mm QFN

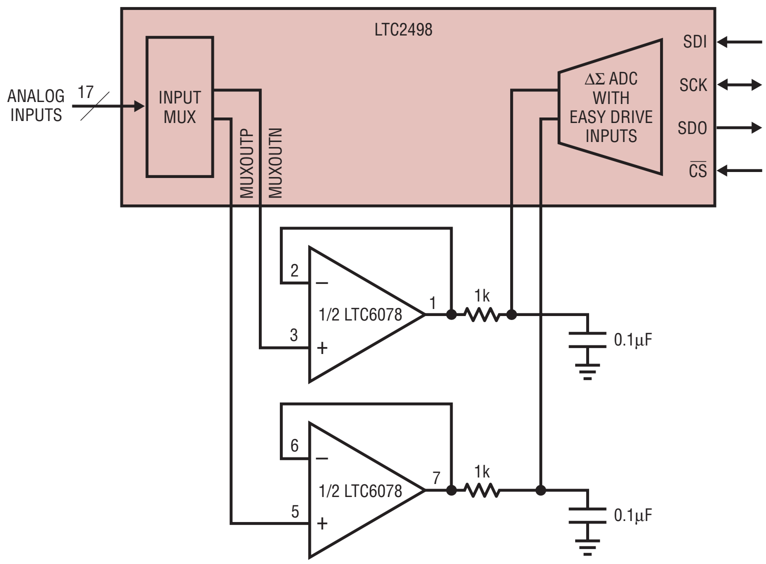

Figure 2. External buffers provide high impedance inputs and amplifier offsets are automatically cancelled.

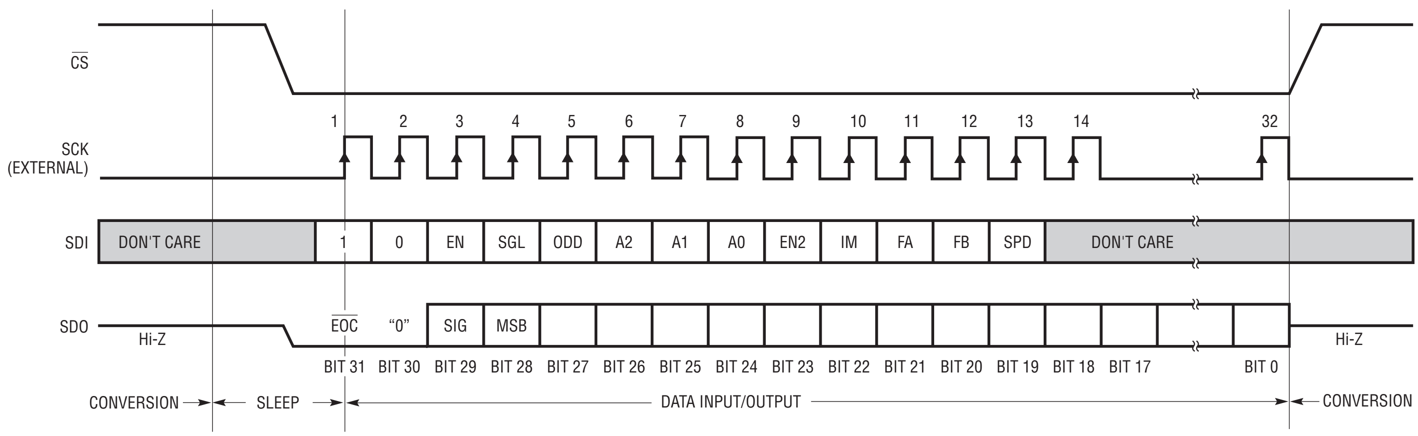

Figure 3. SPI interface, configuration and data output timing

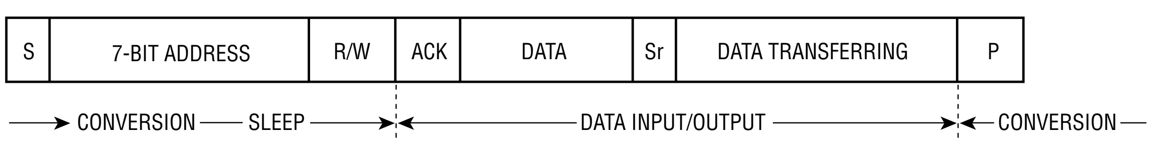

Figure 4. I²C conversion sequence

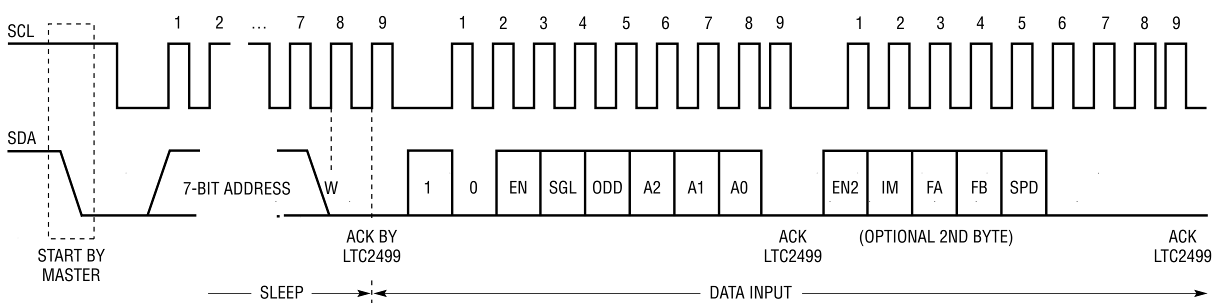

Figure 5. I²C configuration and data output timing

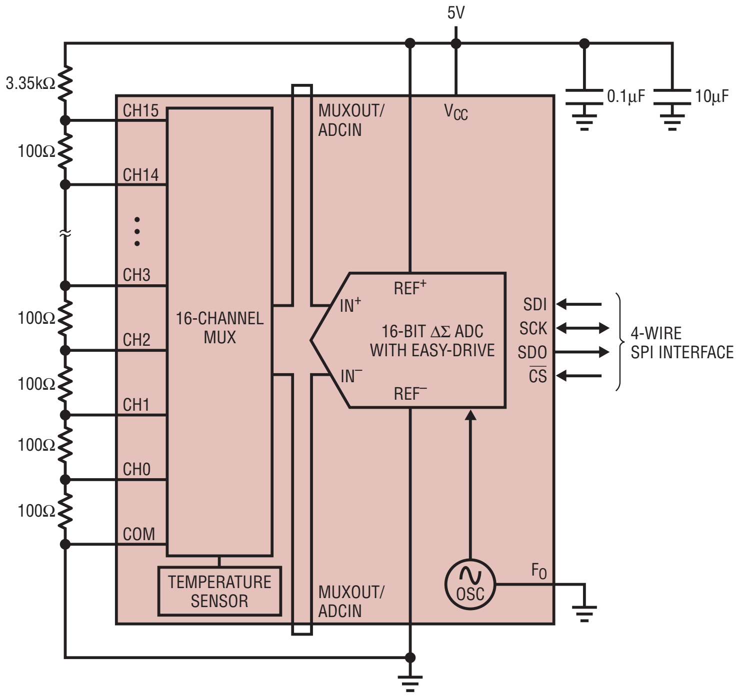

Figure 6. Use this setup to quickly sort out which SDI word is associated with each input channel.

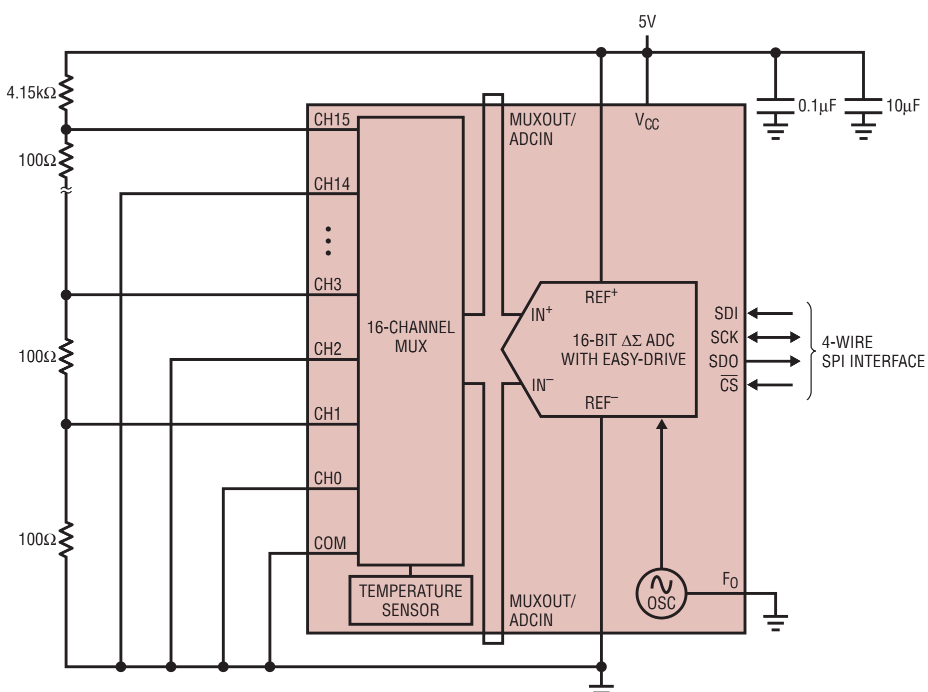

Figure 7. Use this setup to quickly sort out which SDI word is associated with each *differential* input channel.

of incomplete settling in the external RC network. Linear Technology's Easy Drive technology cancels the differential input current. By balancing the negative input (CH3) with a 1kΩ-0.1µF RC network, errors due to the common mode input current are cancelled.

Complete Easy Drive Delta-Sigma Family

Easy Drive ADCs are at home in a vast array of applications. The 24-bit, 16-channel LTC2498 with integrated temperature sensor is ideal for high performance data acquisition systems. It can directly digitize thermocouples without any signal conditioning and provide cold junction compensation. It can also directly measure low level strain gage outputs. At the same time it can handle industrial sensor voltages with the addition of a simple resistive divider—no active circuitry required.

The 16-bit, 16-channel devices are suitable for measuring voltages and currents on large circuit boards that have several high current supplies. Up to 16 ground referred measurements can be taken if the COM pin is grounded to a common point for all supplies. Using the inputs differentially (up to 8 differential input channels) allows high side sensing of current shunts as long as the shunt common mode voltage is less than or equal to the ADCs' supply voltage. Differential measurements also allow voltages to be sensed remotely, eliminating errors due to large ground currents.

Another big advantage of using a delta-sigma ADC for power supply measurements is the very strong rejection of noise and switching transients. The ADC's internal SINC4 filter, in conjunction with a simple 1-pole filter at the ADC input, is adequate to attenuate switching power supply noise below the ADC noise floor. What is left is an extremely accurate measurement of the DC value of the power supply voltage or current.

The single channel LTC2482 is ideal for cost sensitive applications such as portable medical devices and consumer products. Don't be fooled by its relatively low cost—it is essentially a perfect 16-bit ADC that shares the same 600nV input noise floor as the 24-bit parts. This means it would also be ideal for a 4½ digit handheld or bench-top voltmeter with a ±1 count linearity specification.

Automatic Offset Calibration of External Buffers/Amplifiers

In addition to the Easy Drive input current cancellation, the 16-channel Easy Drive ADCs allow an external amplifier to be inserted between the multiplexer output and the ADC input (see Figure 2). This is useful in applications where balanced source impedances are not possible or where the source impedance is very high. One pair of external buffers/amplifers can be shared between all 17 analog inputs. The LTC2498 performs an internal offset calibration every conversion cycle in order to remove the offset and drift of the ADC. This calibration is performed through a combination of front end switching and digital processing. Since the external amplifier is placed between the multiplexer and the ADC, it is inside this correction loop. This results in automatic removal of the offset and offset drift of the external amplifer.

The LTC6078 is an excellent amplifier for this function. It operates with supply voltages as low as 2.7V and its voltage noise level is a low $18\text{nV}/\sqrt{\text{Hz}}$. The LTC2498's Easy Drive inputs allow an RC network to be added directly to the output of the LTC6078. The capacitor reduces the magnitude of the current spikes seen at the input to the ADC and the resistor isolates the capacitor load from the op amp output enabling stable operation.

Software Interface

The simplicity of the analog interfacing requirements of Linear Technology's Easy Drive ADCs is matched by the simplicity of their serial interface. The No Latency architecture eliminates the annoyance of having to discard readings after switching channels on the multichannel devices. The start of conversion is directly controlled by the serial interface, so external signal conditioning or sensor excitation can be switched in at the proper

Sample Code Drivers for Basic Communications with the LTC2448 and LTC2449

```
// Make sure this structure applies in the
// context of the following functions.
struct fourbytes  // Define structure of four consecutive bytes
   {              // To allow byte access to a 32-bit int or float.
   int8 te0;      //
   int8 te1;      // The make32() function in some compilers will
   int8 te2;      // also work, but a union of 4 bytes and a 32-bit int
   int8 te3;      // is more portable because it is standard C.
   }

// Some defines for I2C communication
#define READ      0x01    // bitwise OR with address for read or write
#define WRITE     0x00


/*****************************************************************
Blocking version of read_LTC2498() function. When called,
it will wait for the LTC2498 to finish converting and then
read data. The longest this function should ever take to return
is the maximum conversion time of the LTC2498. It is a good
idea to use a watchdog when your program has blocking functions
like this.

The spi_readwrite() function simultaneously reads and writes
an 8-bit byte from the SPI port. Most compilers that support
processors that have a hardware SPI port have a similar function.

As a starting point, configure the SPI port for data transitions
on the falling clock edge, valid on the rising edge.

Arguments:  channel - channel to program for the next conversion
            config - configuration bits for next conversion

Returns:    32 bit word from the LTC2498 when the conversion finishes
*****************************************************************/
Int32 read_LTC2498(char channel, char config);
        {
   // Create a union of the four-byte structure and a 32-bit
   // signed integer.
   union// adc_code.bits32    all 32 bits
      {                          // adc_code.by.te0    byte 0
      signed int32 bits32;       // adc_code.by.te1    byte 1
      struct fourbytes by;       // adc_code.by.te2    byte 2
      } adc_code;                // adc_code.by.te3    byte 3

        output_low(CS_);   // Lower Chip Select, enabling serial port
        while(input(SDO)); // Wait for SDO to go low. You can also put a
                        // timeout here in case something bad happens
        adc_code.by.te3 = spi_readwrite(channel);
        adc_code.by.te2 = spi_readwrite(config);
   adc_code.by.te1 = spi_readwrite(0);
   adc_code.by.te0 = spi_readwrite(0);
   return adc_code.bits32;
   } // end of read_LTC2498()

/*****************************************************************
Non-blocking version of read_LTC2498() function. When called,
it will see if the LTC2498 has finished converting. If so,
data will be read and returned. If not, zero will be returned.
Since all zeros is NOT a valid code from the LTC2498, the calling
program can ignore the return result if zero.

Arguments:  channel - channel to program for the next conversion
            config - configuration bits for next conversion

Returns: 32 bit word from the LTC2498 if conversion is done,
         zero if not.
*****************************************************************/

Int32 read_LTC2498(char channel, char config);
        {
   // Create a union of the four byte structure and a 32 bit
   // signed integer.
   union// adc_code.bits32    all 32 bits
      {                          // adc_code.by.te0    byte 0
      signed int32 bits32;       // adc_code.by.te1    byte 1
      struct fourbytes by;       // adc_code.by.te2    byte 2
      } adc_code;                // adc_code.by.te3    byte 3
```

```
        output_low(CS_);   // Lower Chip Select, enabling serial port
        while(input(SDO)); // Wait for SDO to go low. You can also put a
                           // timeout here in case something bad happens
        adc_code.by.te3 = spi_readwrite(channel);
        adc_code.by.te2 = spi_readwrite(config);
   adc_code.by.te1 = spi_readwrite(0);
   adc_code.by.te0 = spi_readwrite(0);
   return adc_code.bits32;
   } // end of read_LTC2498()


/*****************************************************************
Non-blocking read_LTC2499() function.

the i2c_xxxx() functions do the following:

void i2c_start(void):   generate an i2c start or repeat start condition
void i2c_stop(void):    generate an i2c stop condition
char i2c_read(boolean): return 8 bit i2c data while generating
                        an ack or nack
boolean i2c_write():    send 8 bit i2c data and return ack or
                        nack from slave device

These functions are very compiler specific, and can use either a
hardware i2c port or software emulation of an i2c port. This example
uses software emulation.

A good starting point when porting to other processors is to write
your own i2c functions. Note that each processor has its own way of
configuring the i2c port, and different compilers may or may not have
built-in functions for the i2c port.

Arguments:  addr - LTC2499 I2C address
            channel - channel to program for the next conversion
            config - configuration bits for next conversion

Returns:    32 bit word from the LTC2499 if conversion is done,
            zero if not.
*******************************************************************/


signed int32 read_LTC2499(char addr, char channel, char config)
   {
   union                         // adc_code.bits32    all 32 bits
      {                          // adc_code.by.te0    byte 0
      signed int32 bits32;       // adc_code.by.te1    byte 1
      struct fourbytes by;       // adc_code.by.te2    byte 2
      } adc_code;                // adc_code.by.te3    byte 3

// Start communication with LTC2481:
      i2c_start();
      if(i2c_write(addr | WRITE))// If no acknowledge, return zero
         {
         i2c_stop();
         return 0;
         }
      i2c_write(channel);
      i2c_write(config);
      i2c_start();
      i2c_write(addr | READ);
      adc_code.by.te3 = i2c_read();
      adc_code.by.te2 = i2c_read();
      adc_code.by.te1 = i2c_read();
      adc_code.by.te0 = i2c_read();
      i2c_stop();
   return adc_code.bits32;
   } // End of read_LTC2499()


/*****************************************************************
Note: you can create a non-blocking version of this function
by repeatedly attempting to write the LTC2499 address, sending
a stop condition if there is no acknowledge to keep the bus free.
When the LTC2499 acknowledges, read the data and return.
*****************************************************************/
```

time. The implicit offset and gain calibration that takes place in every conversion eliminates the need for complicated internal register set or calibration cycles. Communication for both the SPI and I^{2}C interface parts is a simple read/write operation where data from one conversion is read as the configuration for the next channel is programmed into the ADC.

Figure 3 shows the data input/output operation for the LTC2498. This is the SPI-interface ADC with the most channels and features—other SPI parts have similar interfaces.

Figure 4 shows the data input/output operation for the LTC2499. Likewise, this is the most feature-laden I^{2}C part—other I^{2}C parts have similar interfaces. Figure 5 shows the details of writing the channel and configuration to the input registers.

To help the software/firmware designer get started, see the sidebar for C code drivers for basic communications with the LTC2448 and LTC2449. These functions can be easily ported to any C compiler and can easily be adapted to the other Easy Drive ADCs.

Try this Trick!

While the Easy Drive serial interface is easy to program—just read the data for sample N while programming the channel for sample N+1—it can still be tricky to figure out what was just read when looking at a microcontroller's registers through a debugger. Here is a hardware trick that can significantly reduce code design headaches. Figure 6 shows a simple circuit that applies a known voltage to each single-ended input. With the values shown, CH0 has a voltage of 101mV, CH1 202mV, and so on up to CH15, which produces 1.616V. Figure 7 shows the equivalent circuit for differential inputs. Use this setup to quickly sort out which SDI word is associated with each input channel.