Keywords: MAX14001, MAX14002, Isolated ADC, SPI, Binary Input, Distribution Automation

APPLICATION NOTE 6636

# GUIDELINES TO PROGRAMMING THE MAX14001/MAX14002 CONFIGURABLE, ISOLATED ADCS

*Abstract: The MAX14001/MAX14002 are isolated, single-channel ADCs optimized for configurable binary input applications. A microcontroller compatible serial peripheral interface (SPI) provides access to many advanced features. This application note provides example C-code implementing setup, monitoring, and diagnostic functions in the microcontroller.*

## Introduction

The MAX14001 and MAX14002 are 10-bit analog-digital converters (ADCs) with a 3.75kV$_{RMS}$ isolated serial peripheral interface (SPI). Additional features include a programmable magnitude comparator, programmable inrush current for cleaning relay contacts, and programmable input bias current to optimize power dissipation while reducing capacitively coupled input noise. The ADC and all field-side circuits are powered by an integrated, isolated, DC-DC converter that allows field-side functionality to be verified even in the absence of input signals or other field-side supply. This makes the MAX14001 and MAX14002 ideally suited for high density, multirange, individually isolated, binary input modules. This application note presents a series of functions to provide a faster and proven solution to programming the MAX14001 and MAX14002. They are written in C and should prove easy to port over to any common microcontroller. For detailed information regarding the MAX14001/MAX14002 pins, operating modes, and control registers, refer to the MAX14001/MAX14002 data sheet.

## MAX14001/MAX14002 SPI

MAX14001/MAX14002 SPI commands are 16 bits long. The structure is shown in **Table 1**. SPI mode for the MAX14001/MAX14002 has some features that are different from the default SPI settings in most SPI masters, such as microcontrollers or FPGAs, which the software engineer must be aware of when writing code for the MAX14001/MAX14002. Specifically, the first bit clocked into the SDI port is D[0], the data LSB. Usually, SPI masters clock out the data MSB first, so the microcontroller or FPGA must reverse data prior to outputting it to the MAX14001/MAX14002 SDI pin. Another potential difference is the Control bit; for the MAX14001/MAX14002 the function is W/R, meaning Read = 0 and Write = 1.

**Table 1. MAX14001/MAX14002 SPI Command Structure**

| ADDRESS | CONTROL | DATA |
|---|---|---|
| 5-bits A[4:0], MSB to LSB | W/R bit, Read = 0, Write = 1 | 10-bits D[9:0], MSB to LSB |

Full details of SPI read and write cycles, along with register tables, instructions, and a recommended configuration flowchart, can be found in the MAX14001/MAX14002 data sheet.

**Figure 1** shows the main function blocks of the MAX14001/MAX14002. Essentially, there are four main parts to the devices:

- SAR ADC with Internal Voltage Reference –The main function is to convert the analog data that can be read using the SPI or that can be used to signal exceeding predefined voltage levels using the Binary Comparator.
- Logic Side Interface–SPI port for accessing all devices registers and hardware flags for COUT and FAULT.
- µPower DC-DC–Provides field-side power to MAX14001 and MAX14002 internal blocks and to field-side DC-DC output pin VDDF.
- GATE, IFET, ISET–Features used to control external FET and inrush/bias current.

Not all functions need to be used, depending upon the end application for the MAX14001/MAX14002.

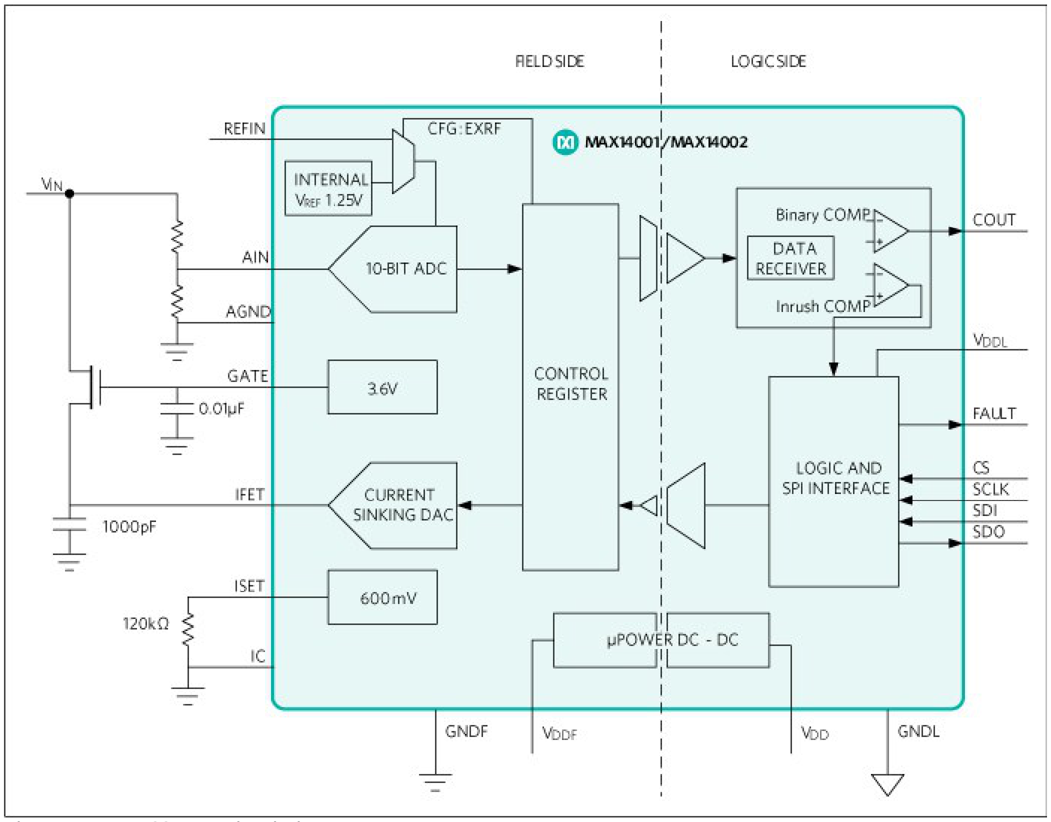

*Figure 1. MAX14001 Functional Diagram*

## MAX14001/MAX14002 – Application Examples

**Configurable, Multirange Binary Input with Wetting Current Control**

The MAX14001/MAX14002 are designed to support applications in protection relays and similar high-voltage binary input applications. Configurable features support an external high-voltage FET that passes small current or wetting pulses used to clean relay contacts within the monitored equipment. A typical application circuit is shown in **Figure 2**. The targeted application is Distribution Automation, and the MAX14001/MAX14002's ADC monitors a DC or AC voltage (typically in the range 24V to 300V) passed through HV Relay/Circuit Breaker. The MAX14001/MAX14002's Configurable Comparator Thresholds are used to support different full-scale system voltages. In this circuit example the GATE voltage from the MAX14001 is used to bias external power FET; the configurable current DAC (IFET) regulates the bias current to clamp the capacitively coupled noise on the HV relay/circuit breaker and to control the inrush current, which is used to clean the oxides from mechanical relays.

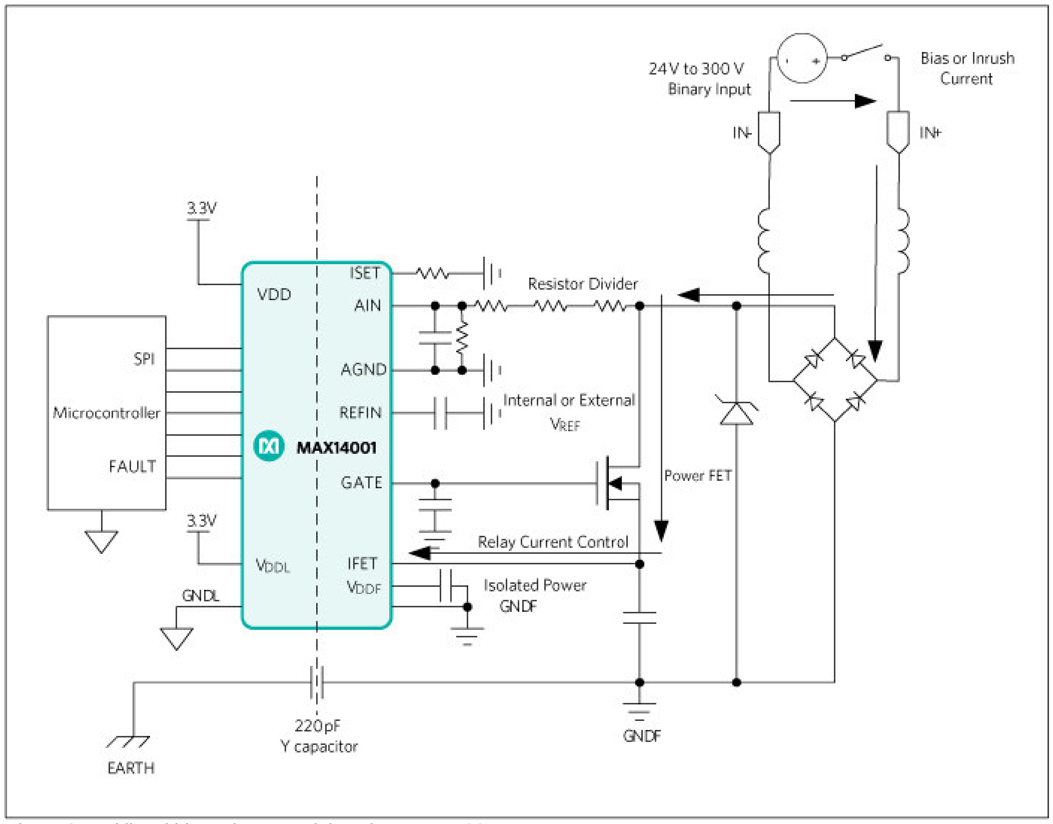

*Figure 2. Multilevel binary input module using MAX14001.*

This application shows a single MAX14001, but multiple MAX14001s and MAX14002s can be daisy-chained to support a multichannel solution. In this application mode, full use is made of the features within the IC, which requires access to all the registers. Refer to the configuration flowchart (Figure 12 in the MAX14001/MAX14002 data sheet) for the Register Programming after Power-On Reset.

**Easy-to-Use Isolated ADC**

**Figure 3** illustrates how easy it is to use the MAX14001 or MAX14002 as a simple isolated ADC, taking advantage of their integrated isolated DC-DC to design systems, which require no field-side power supply. The use of the device in this mode is greatly simplified, with just a few decisions to make that impact the initial configuration programming:

- Configuration is set using the CFG register: select Voltage Reference and decide whether to use the internal voltage reference or an external device.
- Read unfiltered ADC data (ADC register) or filtered ADC data (FADC register).
- Set upper and lower comparator thresholds by programming the THL and THU registers.
- Interrogate the FAULT register to determine the cause of the FAULT pin asserting. Set the bits in the FLTEN register to enable or disable the fault sources.
- All other registers INRR, INRT, INRP, and all the “verification” registers can be ignored for “simple ADC mode”, which does not use the other features of the MAX14001 or MAX14002.

The MAX14002 is a more cost-effective selection than the MAX14001 in the application if no inrush or bias current features are needed. The INRR, INRT, and INRP registers are read-only in the MAX14002. Refer to the MAX14001/MAX14002 data sheet for the difference between MAX14001 and MAX14002.

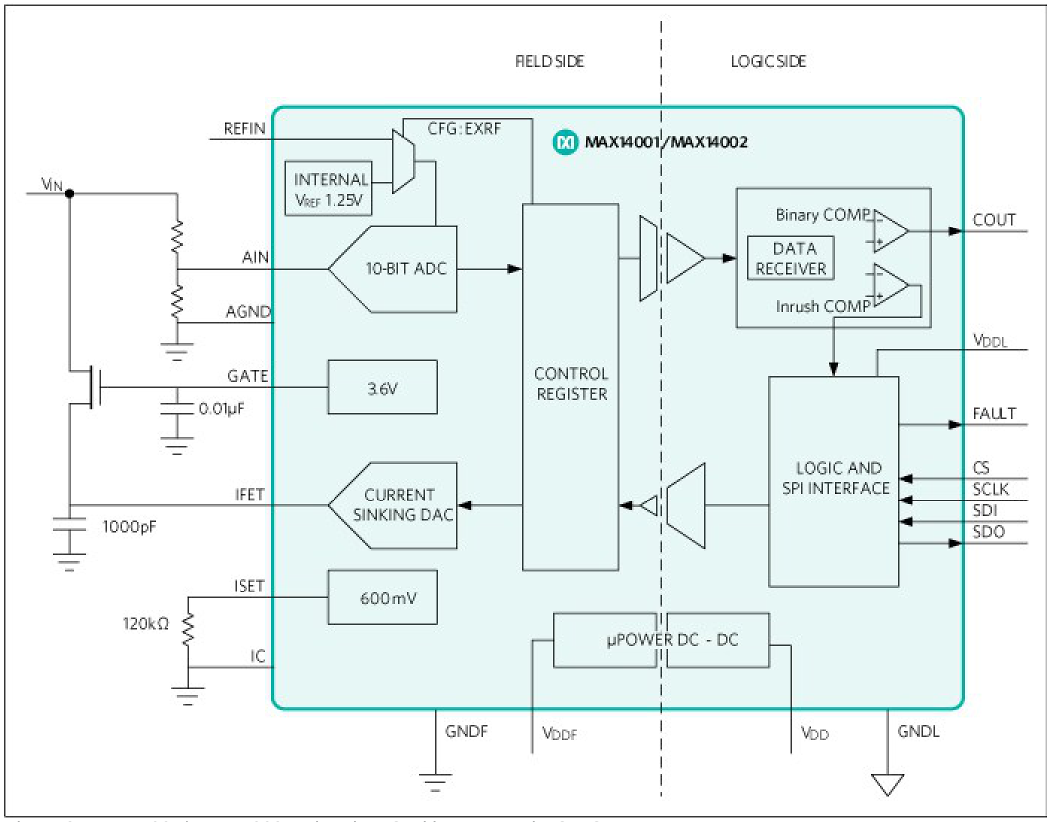

*Figure 3. MAX14001/MAX14002 Isolated ADC with Integrated DC-DC*

## Source Code

This application note provides C source code examples, essentially providing driver functions to access the multiple registers within the MAX14001/MAX14002 for configuration, control, and diagnostic features.

```
  // local variables for registers

UInt16 reg_FLAGS = 0x000;

UInt16 reg_FLTEN = 0x000;

UInt16 reg_THU = 0x000;

UInt16 reg_THL = 0x000;

UInt16 reg_ INRR = 0x000;

UInt16 reg_ INRT = 0x000;

UInt16 reg_ INRP = 0x000;

UInt16 reg_CFG = 0x000;

UInt16 reg_ ENBL = 0x000;

UInt16 reg_WVR = 0x000;

bool fault_pin = false; // means no fault

#region Basic calls to setup and read first results of MAX14001

  // Initialize device

MAX14001_ Initialize();
```

```
  // read the 2 ADC registers

UInt16 ADC_value_ new = MAX14001_read(MAX14001_ADC_adr);

UInt16 FADC_value_new = MAX14001_read(MAX14001_FADC_adr);

#endregion

#region MAX14001 commands

private void MAX14001_ Initialize()

{

  MAX14001_ write(0x0c, 0x294); // enable register write

  MAX14001_Reg_ReadAll_Click ( null, null ); // reads all registers and stores results in local variables

  // Write the values back to their appropriate verification registers to clear the MV Fault

  MAX14001_write (0x10 + MAX14001_FLTEN_adr, reg_FLTEN);

  MAX14001_write (0x10 + MAX14001_THU_ adr, reg_THU);

  MAX14001_write (0x10 + MAX14001_THL_ adr, reg_THL);

  MAX14001_write (0x10 + MAX14001_INRR_ adr, reg _INRR);

  MAX14001_write (0x10 + MAX14001_INRT_ adr, reg _INRT);

  MAX14001_write (0x10 + MAX14001_INRP_ adr, reg _INRP);

  MAX14001_write (0x10 + MAX14001_CFG_ adr, reg_CFG);

  MAX14001_write (0x10 + MAX14001_ENBL_ adr, reg _ENBL);

}

private void MAX14001_Reg_ReadAll_ Click( object sender, EventArgs e)

{

  reg_FLTEN = 0x3ff & MAX14001_read (MAX14001_FLTEN_adr);

  reg_THU = 0x3ff & MAX14001_read (MAX14001_THU_adr);

  reg_THL = 0x3ff & MAX14001_read (MAX14001_THL_adr);

  reg_ INRR = 0x3ff & MAX14001_read (MAX14001_INRR_adr);

  reg_ INRT = 0x3ff & MAX14001_read (MAX14001_INRT_adr);

  reg_ INRP = 0x3ff & MAX14001_read (MAX14001_INRP_adr);

  reg_CFG = 0x3ff & MAX14001_read (MAX14001_CFG_adr);

  reg_ ENBL = 0x3ff & MAX14001_read (MAX14001_ENBL_adr);

  reg_WVR = 0x3ff & MAX14001_read (MAX14001_WVR_adr);

}

private void Read_Fault_Register_btn_ Click ( object sender, EventArgs e)

{

  // read Error flags to clear them

  MAX14001_read (MAX14001_FLAGS_adr);
```

```
      // read Error flags to get the new value
      MAX14001_read (MAX14001_FLAGS_adr);
      reg_FLAGS = 0x3ff & MAX14001EVAL_USB2PMB_adapter.GPI_ read(1); // read FAULT pin
      update_FAULT_ LEDs ();
    }
    private void FullReset_btn_ Click ( object sender, EventArgs e)
    {
      MAX14001_ write(0x0b, 0x80); // writes ACT register RSET bit
      MAX14001_Reg_ReadAll_Click( null, null);
    }
    private void RegisterReset_btn_ Click ( object sender, EventArgs e)
    {
      MAX14001_ write(0x0b, 0x40); // writes ACT register SRES bit
      MAX14001_Reg_ReadAll_Click( null, null);
    }
    private void Trigger_Inrush_Pulse_btn_ Click ( object sender, EventArgs e)
    {
      // Trigger inrush current Register 0x0b bit 9
      MAX14001_ write(0x0b, 0x200); // writes ACT register INPLS bit
      MAX14001_Reg_ReadAll_Click( null, null);
    }
private void FLTen_cb_ CheckedChanged ( object sender, EventArgs e)
    {
      reg_FLTEN = 0;
      if (FLTen_00_ADC_ cb.Checked) reg_FLTEN += 0x02;
      if (FLTen_01_INRD_ cb.Checked) reg_FLTEN += 0x04;
      if (FLTen_02_SPI_ cb.Checked) reg_FLTEN += 0x08;
      if (FLTen_03_COM_ cb.Checked) reg_FLTEN += 0x10;
      if (FLTen_04_CRCL_ cb.Checked) reg_FLTEN += 0x20;
      if (FLTen_05_CRCF_ cb.Checked) reg_FLTEN += 0x40;
      if (FLTen_06_FET_ cb.Checked) reg_FLTEN += 0x80;
      if (FLTen_07_MVL_ cb.Checked) reg_FLTEN += 0x100;
      MAX14001_ write(MAX14001_FLTEN_adr, reg_FLTEN);
      // Write the values back to their appropriate verification registers
      MAX14001_write (0x10 + MAX14001_FLTEN_adr, reg_FLTEN);
```

```
  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cb_FAST_mode_ Click ( object sender, EventArgs e)

{

  if ( cb_FAST_ mode.Checked) reg_CFG |= 0x02;

  else reg_CFG &= 0xfd;

  MAX14001_ write(MAX14001_CFG_adr, reg_CFG);

  // Write the values back to their appropriate verification registers

  MAX14001_write (0x10 + MAX14001_CFG_ adr, reg_CFG);

  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cb_Use_Unfiltered_ADC_IRAW_ Click ( object sender, EventArgs e)

{

  if ( cb_Use_Unfiltered_ADC_IRAW.Checked) reg_CFG |= 0x01;

  else reg_CFG &= 0xfe;

  MAX14001_ write(MAX14001_CFG_adr, reg_CFG);

  // Write the values back to their appropriate verification registers

  MAX14001_write (0x10 + MAX14001_CFG_ adr, reg_CFG);

  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cb_Enable_Input_Current_ENA_ Click ( object sender, EventArgs e)

{

  if ( cb_Enable_Input_Current_ENA.Checked) reg_ENBL  |= 0x10;

  else reg_ENBL &= 0xef;

  MAX14001_ write(MAX14001_ENBL_adr, reg_ENBL);

  // Write the values back to their appropriate verification registers

  MAX14001_write (0x10 + MAX14001_ENBL_ adr, reg _ENBL);

  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cb_EXRF_ Click ( object sender, EventArgs e)

{

  if ( cb_EXRF.Checked) reg_CFG |= 0x20;

  else reg_CFG &= 0xdf;

  MAX14001_ write(MAX14001_CFG_adr, reg_CFG);

  // Write the values back to their appropriate verification registers
  MAX14001_write (0x10 + MAX14001_CFG_ adr, reg _CFG);

  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cb_EXTI_ Click ( object sender, EventArgs e)

{

  if ( cb_EXTI.Checked) reg_CFG |= 0x10;

  else reg_CFG &= 0xef;

  MAX14001_ write(MAX14001_CFG_adr, reg_CFG);

  // Write the values back to their appropriate verification registers

  MAX14001_write (0x10 + MAX14001_CFG_ adr, reg _CFG);

  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cmb_TINR_ SelectionChangeCommitted ( object sender, EventArgs e)

{

  reg_INRP = (UInt16) ( reg_INRP & 0x3c3); // set the TINR bits all to 0

  reg_INRP = (UInt16) ((UInt 16)( cmb_TINR.SelectedIndex<<2) + reg_INRP);

  MAX14001_ write(MAX14001_INRP_adr, reg_INRP);

  // Write the values back to their appropriate verification registers

  MAX14001_write (0x10 + MAX14001_INRP_ adr, reg _INRP);

  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cmb_IINR_ SelectionChangeCommitted ( object sender, EventArgs e)

{

  reg_INRP = (UInt16) ( reg_INRP & 0x03f); // set the IINR bits all to 0

  reg_INRP = (UInt16) ((UInt 16)( cmb_IINR.SelectedIndex<<6) + reg_INRP);

  MAX14001_ write(MAX14001_INRP_adr, reg_INRP);

  // Write the values back to their appropriate verification registers

  MAX14001_write (0x10 + MAX14001_INRP_ adr, reg _INRP);

  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cmb_InrLim_ SelectionChangeCommitted ( object sender, EventArgs e)

{

  reg_INRP = (UInt16) ( reg_INRP & 0x3fc); // set the DU bits all to 0

  reg_INRP = (UInt16) ((UInt 16)( cmb_InrLim.SelectedIndex) + reg_INRP);

  MAX14001_ write(MAX14001_INRP_adr, reg_INRP);
  // Write the values back to their appropriate verification registers

  MAX14001_write (0x10 + MAX14001_INRP_ adr, reg _INRP);

  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cmb_IBIAS_ SelectionChangeCommitted ( object sender, EventArgs e)

{

  reg_CFG = (UInt16) ( reg_CFG & 0x03f); // set the IBIAS bits all to 0

  reg_CFG = (UInt16) ((UInt 16)( cmb_IBIAS.SelectedIndex<<6) + reg_CFG);

  MAX14001_ write(MAX14001_CFG_adr, reg_CFG);

  // Write the values back to their appropriate verification registers

  MAX14001_write (0x10 + MAX14001_CFG_ adr, reg_CFG);

  MAX14001_Reg_ReadAll_Click( null, null);

}

private void cb_FiltSet_ SelectionChangeCommitted ( object sender, EventArgs e)

{

  reg_CFG = (UInt16) ( reg_CFG & 0x3f3); // set the FT bits all to 0

  reg_CFG = (UInt16) ((UInt 16)( cb_FiltSet.SelectedIndex<<2) + reg_CFG);

  MAX14001_ write(MAX14001_CFG_adr, reg_CFG);

  // Write the values back to their appropriate verification registers

  MAX14001_write (0x10 + MAX14001_CFG_ adr, reg_CFG);

  MAX14001_Reg_ReadAll_Click( null, null);

}

#endregion

#region MAX14001 device driver (SPI interface)

  /// <summary>

  /// Below register access calls and register Name constants

  /// SPI-Driver for MAX14001

  /// </summary>

  ///

  // device has only 8 registers:

  /* Register_Name       Direction     */

  /*---------------------------------------------*/

  /* ADC       R/W     */  public const byte

  MAX14001_ADC_adr     =0x00;

  /* Filtered ADC       R/W     */  public const byte
  MAX14001_FADC_adr     =0x01;

  /* Error Flags       R/W     */  public const byte

  MAX14001_FLAGS_adr     = 0x02;

  /* FAULT Enable       R/W     */  public const byte

  MAX14001_FLTEN_adr     =0x03;

  /* Lower Threshold       R/W     */  public const byte

  MAX14001_THL_adr     =0x04;

  /* Upper Threshold       R/W     */  public const byte

  MAX14001_THU_adr     =0x05;

  /* Inrush Reset       R/W     */  public const byte

  MAX14001_INRR_adr     =0x06;

  /* Inrush Trigger       R/W     */  public const byte

  MAX14001_INRT_adr     =0x07;

  /* Inrush Pulse       R/W     */  public const byte

  MAX14001_INRP_adr     =0x08;

  /* Configuration       R/W     */  public const byte

  MAX14001_CFG_adr     =0x09;

  /* Enable       R/W     */  public const byte

  MAX14001_ENBL_adr     =0x0a;

  /* Write Verification       R/W     */  public const byte

  MAX14001_WVR_adr     =0x0c;

  // verification registers

  /* FAULT Enable verification       R/W     */  public const byte

MAX14001_FLTV_adr     =0x13;

  /* Lower Threshold verification       R/W     */  public const byte

MAX14001_THLV_adr     =0x14;

  /* Upper Threshold verification       R/W     */  public const byte

MAX14001_THUV_adr     =0x15;

  /* Inrush Reset verification       R/W     */  public const byte

MAX14001_INRRV_adr     =0x16;

  /* Inrush Trigger verification       R/W     */  public const byte

MAX14001_INRTV_adr     =0x17;

  /* Inrush Pulse verification       R/W     */  public const byte

MAX14001_INRPV_adr     =0x18;

  /* Configuration verification       R/W*/ public const byte
```

```
    MAX14001_CFGV_adr     =0x19;

      /* Enable verification       R/W     */  public const byte

    MAX14001_ENBLV_adr     =0x1a;

      void MAX14001_SPI_CS_ l()

      {

        MAX14001EVAL_USB2PMB_adapter.SPI_CS0 Enable();

      }

    void MAX14001_SPI_CS_ h()

      {

        MAX14001EVAL_USB2PMB_adapter.SPI_CS0 Disable();

      }

      // maximum 5MHz SPI clock for this device !!

      //*********************************************************************

      //*

      //*Function: MAX14001_read

      //* Description: Read one Register from MAX14001

      //*

      //* Input: Register-Address (take from definitions in header-file)

      //* Output: 8bit data-sample

      //*

      //*********************************************************************/

void MAX14001_read ( byte Register)

    {

      // 5-bit Address (MSB to LSB),        R/W bit (1=W/0=R), 10-bit Data (MSB to LSB)

      //  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0

      //  A4  A3  A2  A1  A0        R/W  D9  D8  D7  D6  D5  D4  D3  D2  D1  D0

      // construct the dataword to send on SPI:

      UInt16 data_to_send = (UInt16) (( ((UInt16)Register) <<11) & 0xf800 );

      // reverse the bits

      data_to_send = reverse_uint16( data_to_send);

      MAX14001_SPI_CS_ l();

      MAX14001EVAL_USB2PMB_adapter.SPI_W_transaction_16( data_to_send);

      MAX14001EVAL_USB2PMB_adapter.SPI_RW_transaction_16(data_to_send);

      MAX14001_SPI_CS_ l();

    }
```

```
//***********************************************************************
//*
//* Function: MAX14001_write
//* Description: Write one Register to MAX14001
//*
//* Input: Register-Address (take from definitions in header-file)
//* Output: 8bit data-sample
//*
//**********************************************************************/
void MAX14001_write ( byte Register, UInt16 data)
{
  // 5-bit Address (MSB to LSB),        R/W bit (1=W/0=R), 10-bit Data (MSB to LSB)
  //  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
  //  A4  A3  A2  A1  A0         R/W  D9  D8  D7  D6  D5  D4  D3  D2  D1  D0
  // construct the dataword to send on SPI:
  UInt16 data_to_send = (UInt16) ( (( ((UInt16)Register) <<11) & 0xf800 ) + data + 1024); // 1024 sets write bit
  // reverse the bits
  data_to_send = reverse_uint16( data_to_send);
  MAX14001_SPI_CS_ l();
  MAX14001EVAL_USB2PMB_adapter.SPI_W_transaction_16 ( data_to_send);
  MAX14001_SPI_CS_ l();
}
UInt16 reverse_uint16(UInt16 x)
{
  UInt16 y= (UInt16) ((((x >> 0) & 1) << 15) |
  (((x >> 1) & 1 ) << 14) |
  (((x >> 2) & 1 ) << 13) |
  (((x >> 3) & 1 ) << 12) |
  (((x >> 4) & 1 ) << 11) |
  (((x >> 5) & 1 ) << 10) |
  (((x >> 6) & 1 ) << 9) |
  (((x >> 7) & 1 ) << 8) |
  (((x >> 8) & 1 ) << 7) |
  (((x >> 9) & 1 ) << 6) |
  (((x >> 10) & 1) < < 5) |
      (((x >> 11) & 1) < < 4) |
      (((x >> 12) & 1) < < 3) |
      (((x >> 13) & 1) < < 2) |
      (((x >> 14) & 1) < < 1) |
      (((x >> 15) & 1) < < 0));
      return y;
    }
    #endregion
```

## Conclusion

This application note has shown how the MAX14001 and MAX14002 have two different use cases, which require limited or full use of the control registers within the device. This code was tested using MAX14001PMB# and MAX14001EVSYS# as well as the corresponding USB2PMB2# adapter and Munich GUI. By taking advantage of the C code examples in this application note, the engineer has a proven solution to quickly and easily implement an interface between popular microcontrollers and the MAX14001/MAX14002.

| Related Parts | | |
|---|---|---|
| MAX14001 | Configurable, Isolated 10-bit ADCs for Multi-Range Binary Inputs | Free Samples |
| MAX14001EVSYS | Evaluation Kit for the MAX14001/MAX14002 | |
| MAX14001PMB | Evaluation Kit for the MAX14001 | |
| MAX14002 | Configurable, Isolated 10-bit ADCs for Multi-Range Binary Inputs | Free Samples |
| USB2PMB2 | Adapter Board for the Munich (USB2PMB2#) | |