Parts: MAX14915, MAX14916, MAX14915EVKIT, MAX14916EVKIT

Meta Keywords: MAX14915, MAX14916, High-Side Switch, 8-Channel Digital Output, CRC, SPI, programming, C-Code Example

Meta Description: This application note contains guidelines to implement the MAX14915 8-Channel High-Side Switch into the microcontroller code using C-Code Examples.

Abstract: *The MAX14915 is an 8 Channel High-Side Switch. It supports 8-channels driving 1A. A microcontroller-compatible serial peripheral interface (SPI) provides access to many diagnostic features. This application note provides example C-code implementation including setup, monitoring, and diagnostic functions.*

APPLICATION NOTE 7506

How to Program the MAX14915/16 - 8 Channel High-Side Switch

Introduction

The MAX14915/16 integrate 8 high side switches and provide options for features such as number of output channels, current limiting, and diagnostic capabilities. The MAX14915 is an octal high-side switch with extensive diagnostic features and 1A (typical) per channel current limit. The MAX14916 can be configured as an octal high-side switch (1A current limiting) or a quad high-side switch (2A current limiting). The devices feature individual channel thermal protection, internal clamps for fast inductive demagnetization, and open-wire detection for On and Off state. Also, there is an integrated light-emitting diode (LED) matrix that displays status (On/Off) as well as fault conditions for each individual channel.

This application note presents a series of functions to provide an easy and proven solution to programming the MAX14915/6 (**Figure 1**). They are written in C and should prove easy to port over to any common microcontroller. For detailed information regarding the MAX14915 pins, operating modes, and control registers, refer to the MAX14915 data sheet. For detailed information about MAX14916 pins, operating modes, and control registers, refer to the MAX14916 data sheet.

This application note mainly shows the programming of the MAX14915, but the technique is very similar for MAX14916.

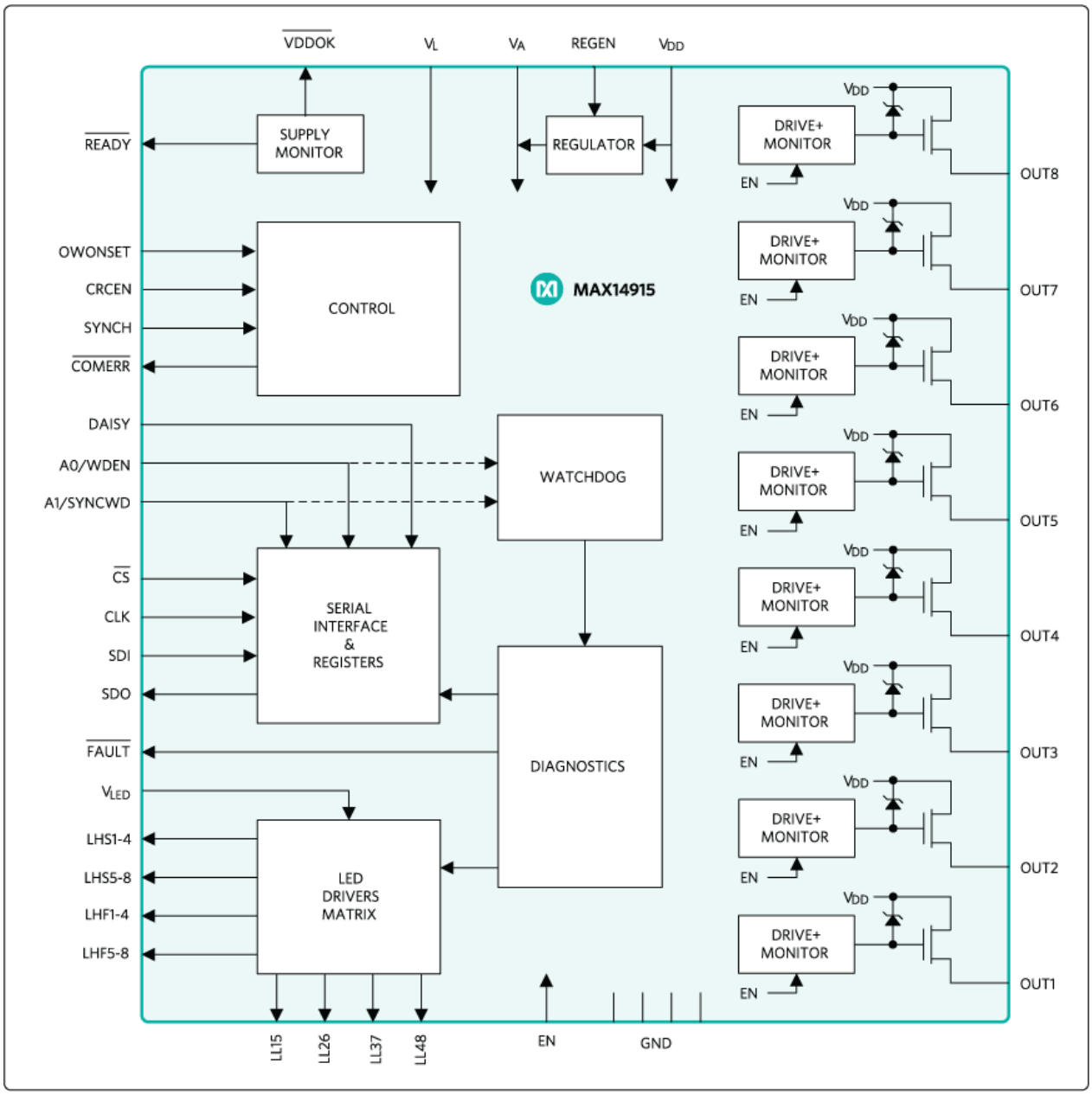

Figure 1. MAX14915 Functional Diagram.

MAX14915 SPI

The MAX14915 SPI commands are 16 bits long (8-bit instruction + 8-bit data), with CRC disabled, and if CRC is enabled this adds 8 more bits for the CRC8. The 2 MSBs of the Command-Byte are Chip address bits, these allow 4 chips to share the same Chip Select (CS) pin. The SPI command structure is shown in Table 1. SPI mode for the MAX14915 is CPOL = 0 (CLK idle = 0), CPHA = 0 (rising/first edge samples the data), the data/commands need to be clocked in MSB first.

Table 1. MAX14915 SPI Command Structure

CHIP-ADDRESS	BURST Bit	REGISTER-ADDRESS	R/W	DATA
2-bits A[1:0]	1 bit BRST[0]	4-bits R[3:0] MSB to LSB	1 bit RW[0]	8-bits D[7:0] MSB to LSB

For more details on SPI read and write cycles, along with register tables and instructions, refer to the MAX14915 and MAX14916 data sheet.

Figure 1 shows the main function blocks of the MAX14915. Essentially, there are 8 output channel high-side drivers, watchdog, and diagnostics, a 16 LED (4x4) Matrix driver, and the logic-side interface (SPI port) to access all device registers and hardware flags for diagnostic.

MAX14915 – Code Application Examples

The MAX14915 is designed to support industrial applications in end equipment such as programmable logic controllers (PLCs) that require multiple high-side switches. A typical application circuit that supports 16 channels group isolated, uses a single MAX14483 digital isolator is shown in **Figure 2**.

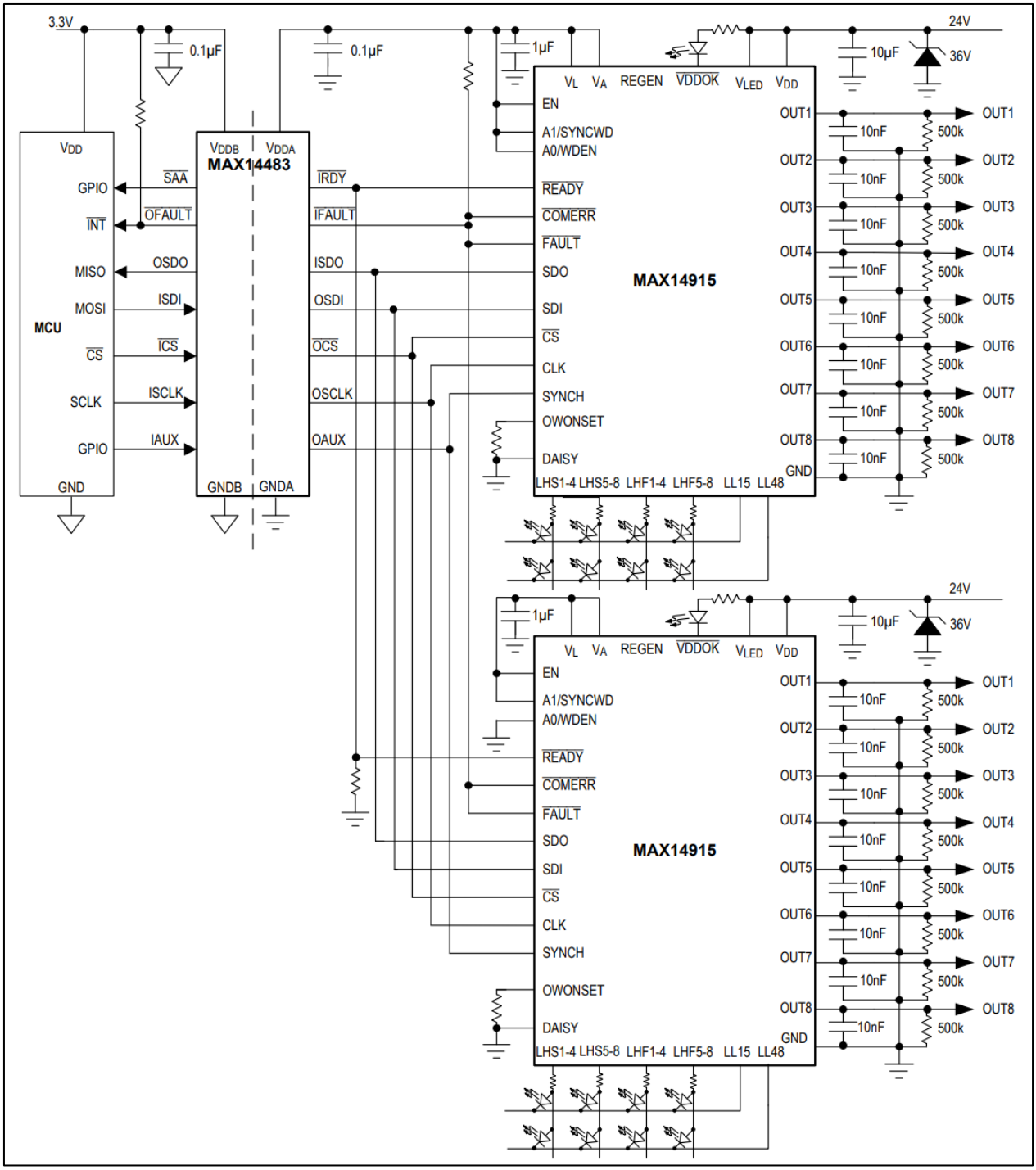

Figure 2. 16 Channels High-Side Switch group isolated.

Each individual output can drive up to 1A, and has individual open-wire, overcurrent and over temperature diagnostics.

To simplify the systems which require galvanic isolation, the MAX14915 supports both daisy-chain and Addressed SPI mode.

Note: Some diagnostics features are only available in direct SPI or addressed SPI mode, but not in daisy-chain mode. For more details, refer to the MAX14915 datasheet.

Source Code

This application note provides C source code examples, essentially providing driver functions to access the multiple registers within the MAX14915 for configuration, control, and diagnostic features. All software has been implemented and tested using MAX14915 EV-Kit. Customers should use the functions in this document as reference only and design their own firmware/software based on their own microcontroller and hardware implementation in their application.

Register Address definitions:

```
#define REG_SetOUT       0x00
#define REG_SetFLED      0x01
#define REG_SetSLED      0x02
#define REG_Interupt     0x03
#define REG_OvlChF       0x04
#define REG_CurrLimF     0x05
#define REG_OwOffChF     0x06
#define REG_OwOnChF      0x07
#define REG_ShtVDDChF    0x08
#define REG_GlobalErr    0x09
#define REG_OwOffEn      0x0a
#define REG_OwOnEn       0x0b
#define REG_ShtVddEn     0x0c
#define REG_Config1      0x0d
#define REG_Config2      0x0e
#define REG_Mask         0x0f
```

Other definitions:

```
#define OUT1             0x01
#define OUT2             0x02
#define OUT3             0x04
#define OUT4             0x08
#define OUT5             0x10
#define OUT6             0x20
#define OUT7             0x40
#define OUT8             0x80

#define GLB_ShrtVDD      0x20
#define GLB_OWOnF        0x10
#define GLB_OWOffF       0x08
#define GLB_CurrLim      0x04
#define GLB_OverLdF      0x02
#define GLB_GloblF       0x01
```

```
bool CRC_Enabled = true;            // if pin CRCEN = high, then this must be true
//bool CRC_Enabled = false;           // if pin CRCEN = low, then this must be false
const int CRC_ERROR = 0xffff;
```

This Test-Routine waits for MAX14915 until it doesn't report any faults, for example after power-up MAX14915 reports a power-on reset condition. Next, the MAX14915 will be initialized so that it can report any faults and as part of a typical power-on self-test routine, a binary counter runs on the 8 channels and prints the fault-conditions.

```
void test()
{

  uint8_t ADR = 0;                  // MAX14915 Chip-Address Allows multiple Chips with 1 Chip-Select
  uint16_t loop_count = 0;

  // Check status, only continue if All OK.
  uint16_t fault = 0xffff;
  while (fault != 0)                // Halt until MAX14915 has no faults
  {
          fault = MAX14915_Init(ADR);
          if ( (fault) != 0) MAX14915_Identify_Error(ADR, fault);  // check for errors if any bit is set
  }

  // Writes a binary counter to the 8 Ports
  // Checks for any errors and prints them
  while (1)
  {
          printf("Writing %i\r\n", loop_count);

          uint16_t w_result = MAX14915_write_register(ADR, REG_SetOUT, loop_count);
          w_result = w_result & 0x3fff;                                    // ignore the 2 MSBs, they're not used

          printf("W-Result 0x%04x\r\n", w_result);
          if ( (w_result) != 0) MAX14915_Identify_Error(ADR, w_result);    // only check if any bit is set

          delay_ms(100);
          loop_count++;
          if (loop_count > 255) loop_count = 0;
  }
}
```

These are the functions needed to initialize and use the MAX14915, covering register read and write operations, CRC code generation and decode, and error flag polling.

```
//*****************************************************************
//*
//* Function: MAX14915_Init
//* Description: Initialize MAX14915
//*
//* Input:
//*         Chip_Address:     Chip address selected by pins A1 and A0
//*
//* Output, none
//*         Initializes the device after power-up
//*
//* if CRC is enabled, then crc5 is sent accordingly
//*
//*****************************************************************/
uint16_t MAX14915_Init(uint8_t Chip_Address)
{
          // ********************************************************************************
          // *   D07   *   D06   *   D05   *   D04   **   D03   *   D02   *   D01   *   D00   *
          // * ComErrM *SuplErrM *  VddOKM * ShtVddM **  OWOnM  *  OWOffM * OWOffM  * OverLdM *
          // *    1    *    0    *    0    *    0    **    0    *    0    *    0    *    0    *
          // ********************************************************************************
          // -> 0x80   // All warnings enabled, just Watchdog off
          uint16_t result = MAX14915_write_register(Chip_Address, REG_Mask, 0x80);

          // ********************************************************************************
          // *   D07   *   D06   *   D05   *   D04   **   D03   *   D02   *   D01   *   D00   *
          // * LEDCLim * FLatchEn*FiltrLong*FFilterEn**FLEDStr1 *FLEDStr0 * SLEDSet * FLEDSet *
          // *    0    *    1    *    0    *    1    **    0    *    0    *    0    *    0    *
          // ********************************************************************************
          // -> 0x50   // Enable Status and Fault LEDs
                       // Keep Fault-Latch and Filter-Blanking on
          result |= MAX14915_write_register(Chip_Address, REG_Config1, 0x50);

          // enable Open-Wire when off detection on all Channels
          // this only works if a load is connected on all channels
          // Maybe should be handled by the Application?
```

```
        result |= MAX14915_write_register(Chip_Address, REG_OwOffEn, 0xff);

        // enable Open-Wire when on detection on all Channels
        // this only works if a load is connected on all channels
        // Maybe should be handled by the Application?
        result |= MAX14915_write_register(Chip_Address, REG_OwOnEn, 0xff);

        // enable Short to VDD detection on all Channels
        // this only works if a load is connected on all channels
        // Maybe should be handled by the Application?
        result |= MAX14915_write_register(Chip_Address, REG_ShtVddEn, 0xff);


        return result & 0x3fff; // discard the 2 MSBs, since they're unused
}


//*****************************************************************
//*
//* Function: MAX14915_write_register
//* Description: Write one Register to MAX14915
//*
//* Input:
//*         Chip_Address:     Chip address selected by pins A1 and A0
//*         Register-Address  take from definitions in header-file
//*         data              Data to be written into the register
//*
//* Output, 16bit result:
//*         MSB 8-bits = Global Error uint8_t
//*         LSB 8-bits = Fault bits F8 to F1
//*
//* if CRC is enabled, then crc5 is sent accordingly
//*
//*****************************************************************/
uint16_t MAX14915_write_register(uint8_t Chip_Address, uint8_t Register_Address, uint8_t data)
{
    // Construct the SPI uint8_t to transmit
    uint8_t  CHIP_ADR = (uint8_t) (Chip_Address<<6);       // 2 MSBs are the Chip Address (A1 and A0)
    uint8_t  BRST     = 0;                                // 0 = no burst read; 1 = burst read  -> 0x20 to set this bit
    uint8_t  R_ADR    = (uint8_t) (Register_Address<<1);   // Register Address

    uint8_t  command  = (uint8_t) (CHIP_ADR | BRST | R_ADR | 1); // construct SPI Command uint8_t. LSB = 1 for Write-
Command
    // END Construct the SPI uint8_t to transmit

    MAX14915_CS_Low();
    // Write Command-uint8_t, Receive the Global Error uint8_t at the same time
    uint8_t GLOB_ERROR = MAX14915_SPI_RW_8bit(command);
    // Write Data-uint8_t, Receive the Individual Fault-Bits at the same time
    uint8_t FAULT_BITS = MAX14915_SPI_RW_8bit(data);

    if (CRC_Enabled == true)
    {
        uint8_t crc5_checksum;
        crc5_checksum = MAX14915_crc5_encode_2byte(command, data);
        MAX14915_SPI_RW_8bit(crc5_checksum);
    }
    MAX14915_CS_High();

    return (uint16_t) ((GLOB_ERROR << 8) | FAULT_BITS);;
}



//*****************************************************************
//*
//* Function: MAX14915_read_register
//* Description: Read one Register from MAX14915
//*
//* Input:
//*         Chip_Address:     Chip address selected by pins A1 and A0
//*         Register-Address  take from definitions in header-file
//*         data              Data to be written into the register
//*
//*
//* Output, 16bit result:
//*         MSB 8-bits = Global Error uint8_t
//*         LSB 8-bits = Register content 8-bits
//*
//* if CRC is enabled and there was a CRC error, then Output is CRC_ERROR
```

```
//*
//*****************************************************************/
uint16_t MAX14915_read_register (uint8_t Chip_Address, uint8_t Register_Address)
{
    // Construct the SPI uint8_t to transmit
    uint8_t  CHIP_ADR = (uint8_t) (Chip_Address<<6);       // 2 MSBs are the Chip Address (A1 and A0)
    uint8_t  BRST     = 0;                                 // 0 = no burst read; 1 = burst read  -> 0x20 to set this bit
    uint8_t  R_ADR    = (uint8_t) (Register_Address<<1);   // Register Address

    uint8_t  command  = (uint8_t) (CHIP_ADR | BRST | R_ADR); // construct SPI Command uint8_t. LSB = 0 for Read-Command
    // END Construct the SPI uint8_t to transmit

    MAX14915_CS_Low();
    // Write Command-uint8_t, Receive the Global Error uint8_t at the same time
    uint8_t GLOB_ERROR = MAX14915_SPI_RW_8bit(command);
    GLOB_ERROR = GLOB_ERROR & 0x3f;                     // Discard the 2 MSBs, they're not used

    // Write Data-uint8_t, Receive the Individual Fault-Bits at the same time
    uint8_t data       = MAX14915_SPI_RW_8bit(0);

    if (CRC_Enabled == true)
    {
        uint8_t crc5_read;
        uint8_t crc5_calc;

        crc5_calc = MAX14915_crc5_decode_2byte(GLOB_ERROR, data);
         crc5_read = MAX14915_SPI_RW_8bit(MAX14915_crc5_encode_2byte(command,0));

         if (crc5_read != crc5_calc) System_Handle_CRC_Error(Chip_Address);
    }
    MAX14915_CS_High();

    return (uint16_t) ((GLOB_ERROR << 8) | data);
}


//*****************************************************************
//*
//* Function: MAX14915_crc5encode_2byte
//* Description: Generates CRC5 byte for 2 byte-Command
//*                       this is needed for WRITE commands
//*
//* Input:
//*         byte1:     Usually Command Byte
//*         byte2:     Byte written to MAX14915
//*
//* Output, 8bit result:
//*         CRC5 of the 2 Bytes
//*
//*****************************************************************/
uint8_t MAX14915_crc5_encode_2byte(uint8_t byte1, uint8_t byte2)
{
    uint8_t crc5_start = 0x1f;
    uint8_t crc5_poly  = 0x15;
    uint8_t crc_result = crc5_start;

    // byte1
    for (int i=0; i<8; i++)
    {
        if( ((( byte1>>(7-i) )&0x01) ^ ((crc_result & 0x10)>>4)) > 0  )
        {
            crc_result = (uint8_t) (crc5_poly ^ ((crc_result<<1) & 0x1f));
        }
        else
        {
            crc_result = (uint8_t)((crc_result<<1) & 0x1f);
        }
    }
    // END byte1

    // byte2
    for (int i=0; i<8; i++)
    {
        if( ((( byte2>>(7-i) )&0x01) ^ ((crc_result & 0x10)>>4)) > 0  )
        {
            crc_result = (uint8_t) (crc5_poly ^ ((crc_result<<1) & 0x1f));
        }
        else
        {
            crc_result = (uint8_t)((crc_result<<1) & 0x1f);
```

```
        }
    }
    // END byte2

    // 3 extra bits set to zero
    uint8_t uint8_t3=0x00;
    for (int i=0; i<3; i++)
    {
        if( ((( uint8_t3>>(7-i) )&0x01) ^ ((crc_result & 0x10)>>4)) > 0  )
        {
            crc_result = (uint8_t) (crc5_poly ^ ((crc_result<<1) & 0x1f));
        }
        else
        {
            crc_result = (uint8_t)((crc_result<<1) & 0x1f);
        }
    }
    // END 3 extra bits set to zero

    return crc_result;
}


//*****************************************************************
//*
//* Function: MAX14915_crc5_decode_2byte
//* Description: Decodes   CRC5 byte for 2 byte-Command
//*                        this is needed for READ commands
//*
//* Input:
//*         byte1:     Usually Command Byte
//*         byte2:     byte read from MAX14915
//*
//* Output, 8bit result:
//*         CRC5 of the 2 Bytes
//*
//*****************************************************************/
uint8_t MAX14915_crc5_decode_2byte(uint8_t byte1, uint8_t byte2)
{
         uint8_t crc5_start = 0x1f;
         uint8_t crc5_poly  = 0x15;
         uint8_t crc_result = crc5_start;

    // BYTE1
    for (int i=2; i<8; i++)
    {
        if( ((( byte1>>(7-i) )&0x01) ^ ((crc_result & 0x10)>>4)) > 0  )  // IF(XOR(C6;BITAND(D5;2^4)/2^4)
        {  // BITXOR($D$1;BITAND((D5*2);31))
            crc_result = (uint8_t) (crc5_poly ^ ((crc_result<<1) & 0x1f));
        }
        else
        {
            crc_result = (uint8_t)((crc_result<<1) & 0x1f);
        }
    }
    // END BYTE1

    // BYTE2
    for (int i=0; i<8; i++)
    {
        if( ((( byte2>>(7-i) )&0x01) ^ ((crc_result & 0x10)>>4)) > 0  )  // IF(XOR(C6;BITAND(D5;2^4)/2^4)
        {  // BITXOR($D$1;BITAND((D5*2);31))
            crc_result = (uint8_t) (crc5_poly ^ ((crc_result<<1) & 0x1f));
        }
        else
        {
            crc_result = (uint8_t)((crc_result<<1) & 0x1f);
        }
    }
    // END BYTE2

    // 3 extra bits set to zero
    uint8_t byte3=0x00;
    for (int i=0; i<3; i++)
    {
        if( ((( byte3>>(7-i) )&0x01) ^ ((crc_result & 0x10)>>4)) > 0  )  // IF(XOR(C6;BITAND(D5;2^4)/2^4)
        {  // BITXOR($D$1;BITAND((D5*2);31))
            crc_result = (uint8_t) (crc5_poly ^ ((crc_result<<1) & 0x1f));
        }
        else
```

```
        {
            crc_result = (uint8_t)((crc_result<<1) & 0x1f);
        }
    }
    // END 3 extra bits set to zero

    return crc_result;
}
```

```
//*****************************************************************
//*
//* Function: MAX14915_Identify_Error
//* Description: This should be called if one of the Global-Error bits
//*              or Fault bits is set
//*              This function identifies the error and calls
//*
//* Input:
//*         uint16_t:    MS-uint8_t = Global Errors from SPI transfer
//*                    LS-uint8_t = Channel Fault bits from SPI transfer
//*
//* Output:
//*         none, will call System Level function to handle the error
//*
//*****************************************************************/
void MAX14915_Identify_Error(uint8_t Chip_Address, uint16_t ERROR_INFO)
{
    uint8_t   GlobalERR  = (uint8_t) ((ERROR_INFO>>8) & 0xff); // MSB 8-bits = Global Error uint8_t
    uint8_t   CH_FAULT   = (uint8_t) ((ERROR_INFO)    & 0xff); // LSB 8-bits = Fault bits F8 to F1

    if ((GlobalERR & GLB_GloblF)  != 0)   System_Handle_Global_fault(Chip_Address);

    // If no Channel Fault, then no further checking is needed:
    if (CH_FAULT == 0) return;

    // if only one single fault exists, then Error-Type is already known by Global_Error_uint8_t
    if ( (CH_FAULT == OUT1) ||
         (CH_FAULT == OUT2) ||
         (CH_FAULT == OUT3) ||
         (CH_FAULT == OUT4) ||
         (CH_FAULT == OUT5) ||
         (CH_FAULT == OUT6) ||
         (CH_FAULT == OUT7) ||
         (CH_FAULT == OUT8) )
    {
          uint8_t ch_number = MAX14915_decode_channel_number(CH_FAULT);
        if ((GlobalERR & GLB_ShrtVDD) != 0)   System_Handle_Short_to_VDD_CH(ch_number, Chip_Address);
        if ((GlobalERR & GLB_OWOnF)   != 0)   System_Handle_OpenWire_CH(ch_number, Chip_Address);
        if ((GlobalERR & GLB_OWOffF)  != 0)   System_Handle_OpenWire_CH(ch_number, Chip_Address);
        if ((GlobalERR & GLB_CurrLim) != 0)   System_Handle_Current_Limit_CH(ch_number, Chip_Address);
        if ((GlobalERR & GLB_OverLdF) != 0)   System_Handle_Overload_CH(ch_number, Chip_Address);
    }
    else
    {   // there is more than 1 error present
        // so we need to check each channel

        // first read each Error Register:
        uint16_t OvlChF    = MAX14915_read_register (Chip_Address, REG_OvlChF);
        uint16_t CurrLim   = MAX14915_read_register (Chip_Address, REG_CurrLimF);
        uint16_t OwOffChF  = MAX14915_read_register (Chip_Address, REG_OwOffChF);
        uint16_t OwOnChF   = MAX14915_read_register (Chip_Address, REG_OwOnChF);
        uint16_t ShtVDDChF = MAX14915_read_register (Chip_Address, REG_ShtVDDChF);

        if ((OvlChF    & OUT1) != 0) System_Handle_Overload_CH(1, Chip_Address);
        if ((OvlChF    & OUT2) != 0) System_Handle_Overload_CH(2, Chip_Address);
        if ((OvlChF    & OUT3) != 0) System_Handle_Overload_CH(3, Chip_Address);
        if ((OvlChF    & OUT4) != 0) System_Handle_Overload_CH(4, Chip_Address);
        if ((OvlChF    & OUT5) != 0) System_Handle_Overload_CH(5, Chip_Address);
        if ((OvlChF    & OUT6) != 0) System_Handle_Overload_CH(6, Chip_Address);
        if ((OvlChF    & OUT7) != 0) System_Handle_Overload_CH(7, Chip_Address);
        if ((OvlChF    & OUT8) != 0) System_Handle_Overload_CH(8, Chip_Address);

        if ((CurrLim   & OUT1) != 0) System_Handle_Current_Limit_CH(1, Chip_Address);
        if ((CurrLim   & OUT2) != 0) System_Handle_Current_Limit_CH(2, Chip_Address);
        if ((CurrLim   & OUT3) != 0) System_Handle_Current_Limit_CH(3, Chip_Address);
        if ((CurrLim   & OUT4) != 0) System_Handle_Current_Limit_CH(4, Chip_Address);
        if ((CurrLim   & OUT5) != 0) System_Handle_Current_Limit_CH(5, Chip_Address);
        if ((CurrLim   & OUT6) != 0) System_Handle_Current_Limit_CH(6, Chip_Address);
        if ((CurrLim   & OUT7) != 0) System_Handle_Current_Limit_CH(7, Chip_Address);
        if ((CurrLim   & OUT8) != 0) System_Handle_Current_Limit_CH(8, Chip_Address);

        if ((OwOffChF  & OUT1) != 0) System_Handle_OpenWire_CH(1, Chip_Address);
        if ((OwOffChF  & OUT2) != 0) System_Handle_OpenWire_CH(2, Chip_Address);
        if ((OwOffChF  & OUT3) != 0) System_Handle_OpenWire_CH(3, Chip_Address);
        if ((OwOffChF  & OUT4) != 0) System_Handle_OpenWire_CH(4, Chip_Address);
        if ((OwOffChF  & OUT5) != 0) System_Handle_OpenWire_CH(5, Chip_Address);
        if ((OwOffChF  & OUT6) != 0) System_Handle_OpenWire_CH(6, Chip_Address);
        if ((OwOffChF  & OUT7) != 0) System_Handle_OpenWire_CH(7, Chip_Address);
        if ((OwOffChF  & OUT8) != 0) System_Handle_OpenWire_CH(8, Chip_Address);
```

```
        if ((OwOnChF   & OUT1) != 0) System_Handle_OpenWire_CH(1, Chip_Address);
        if ((OwOnChF   & OUT2) != 0) System_Handle_OpenWire_CH(2, Chip_Address);
        if ((OwOnChF   & OUT3) != 0) System_Handle_OpenWire_CH(3, Chip_Address);
        if ((OwOnChF   & OUT4) != 0) System_Handle_OpenWire_CH(4, Chip_Address);
        if ((OwOnChF   & OUT5) != 0) System_Handle_OpenWire_CH(5, Chip_Address);
        if ((OwOnChF   & OUT6) != 0) System_Handle_OpenWire_CH(6, Chip_Address);
        if ((OwOnChF   & OUT7) != 0) System_Handle_OpenWire_CH(7, Chip_Address);
        if ((OwOnChF   & OUT8) != 0) System_Handle_OpenWire_CH(8, Chip_Address);

        if ((ShtVDDChF & OUT1) != 0) System_Handle_Short_to_VDD_CH(1, Chip_Address);
        if ((ShtVDDChF & OUT2) != 0) System_Handle_Short_to_VDD_CH(2, Chip_Address);
        if ((ShtVDDChF & OUT3) != 0) System_Handle_Short_to_VDD_CH(3, Chip_Address);
        if ((ShtVDDChF & OUT4) != 0) System_Handle_Short_to_VDD_CH(4, Chip_Address);
        if ((ShtVDDChF & OUT5) != 0) System_Handle_Short_to_VDD_CH(5, Chip_Address);
        if ((ShtVDDChF & OUT6) != 0) System_Handle_Short_to_VDD_CH(6, Chip_Address);
        if ((ShtVDDChF & OUT7) != 0) System_Handle_Short_to_VDD_CH(7, Chip_Address);
        if ((ShtVDDChF & OUT8) != 0) System_Handle_Short_to_VDD_CH(8, Chip_Address);
    }
}

uint8_t MAX14915_decode_channel_number(uint8_t CH_binary)
{
    if ((CH_binary & OUT1) != 0) return 1;
    if ((CH_binary & OUT2) != 0) return 2;
    if ((CH_binary & OUT3) != 0) return 3;
    if ((CH_binary & OUT4) != 0) return 4;
    if ((CH_binary & OUT5) != 0) return 5;
    if ((CH_binary & OUT6) != 0) return 6;
    if ((CH_binary & OUT7) != 0) return 7;
    if ((CH_binary & OUT8) != 0) return 8;

    return 0;
}
```

These functions show the diagnostic features of the MAX14915. In this example, the error condition is only printed. The real applications may have to take further actions:

```
void System_Handle_Overload_CH(uint8_t Channel, uint8_t Chip_Address)
{
    // "Channel" is overloaded
    // System needs to handle over-load on "Channel"
    // Print to the User?
    // Just switch off?
    printf("MAX14915 %i - Channel %i is overloaded\r\n", Chip_Address, Channel);

    // Application actions

    MAX14915_read_register (Chip_Address, REG_OvlChF);  // read to clear the flag
                                                        // if error persists, the flag
                                                        // will come back immediately
}

void System_Handle_Current_Limit_CH(uint8_t Channel, uint8_t Chip_Address)
{
    // "Channel" is in current limiting mode
    // Does the system have to do anything in this case?
    // Print to the User?
    // Just switch off?
    printf("MAX14915 %i - Channel %i is in Current Limiting mode\r\n", Chip_Address, Channel);

    // Application actions

    MAX14915_read_register (Chip_Address, REG_CurrLimF); // read to clear the flag
                                                         // if error persists, the flag
                                                         // will come back immediately
}

void System_Handle_OpenWire_CH(uint8_t Channel, uint8_t Chip_Address)
{
    // "Channel" is open wire
    // Does the system have to do anything in this case?
    // Print to the User?
    printf("MAX14915 %i - Channel %i is Open Wire\r\n", Chip_Address, Channel);

    // Application actions

    MAX14915_read_register (Chip_Address, REG_OwOnChF);
```

```
    MAX14915_read_register (Chip_Address, REG_OwOffChF); // read to clear the flag
                                                         // if error persists, the flag
                                                         // will come back immediately
}

void System_Handle_Short_to_VDD_CH(uint8_t Channel, uint8_t Chip_Address)
{
    // "Channel" is shorted to VDD
    // Does the system have to do anything in this case?
    // Print to the User?
    // Just switch off?
    printf("MAX14915 %i - Channel %i is shorted to VDD\r\n", Chip_Address, Channel);

    // Application actions

    MAX14915_read_register (Chip_Address, REG_ShtVDDChF); // read to clear the flag
                                                          // if error persists, the flag
                                                          // will come back immediately
}

void System_Handle_Global_fault(uint8_t Chip_Address)
{
         // read Global Error Register
    uint16_t global_error = MAX14915_read_register (Chip_Address, REG_GlobalErr);

    if ((global_error & 0x02) != 0) printf("MAX14915 %i Logic Undervoltage\r\n", Chip_Address);
    if ((global_error & 0x04) != 0) printf("MAX14915 %i VDD not good (below 16V)\r\n", Chip_Address);
    if ((global_error & 0x08) != 0) printf("MAX14915 %i VDD warning (below 13V)\r\n", Chip_Address);
    if ((global_error & 0x10) != 0) printf("MAX14915 %i VDD Undervoltage (below 8V)\r\n", Chip_Address);
    if ((global_error & 0x20) != 0) printf("MAX14915 %i Thermal Shutdown\r\n", Chip_Address);
    if ((global_error & 0x40) != 0) printf("MAX14915 %i SYNCH Error\r\n", Chip_Address);
    if ((global_error & 0x80) != 0) printf("MAX14915 %i Watchdog Error\r\n", Chip_Address);

    if ((global_error & 0x01) != 0)
         {
         // After a power-on reset all registers are in reset state
         // -> we need to re-initialize to our settings
         MAX14915_Init(Chip_Address);
         printf("MAX14915 %i Power-up Reset detected\r\n", Chip_Address);
         }
}


void System_Handle_CRC_Error(uint8_t Chip_Address)
{
    // There was a CRC error
    printf("MAX14915 %i - CRC Error!\r\n", Chip_Address);

    // The Application needs to react here

    MAX14915_read_register (Chip_Address, REG_GlobalErr); // read to clear the flag
                                                          // if error persists, the flag
                                                          // will come back immediately
}
```

Conclusion

This application note shows how to program the MAX14915 to drive outputs and diagnose fault-conditions. This code is tested by using MAX14915EVKIT. An engineer can implement an interface quickly and easily between popular microcontrollers and the MAX14915 by using C-code examples mentioned in this application note.