APPLICATION NOTE 7163

HOW TO PROGRAM MAX22000 CONFIGURABLE ANALOG IO

By: Konrad Scheuer

Abstract:
The MAX22000 is a configurable analog IO device. It supports a 1-channel universal voltage/current input output (IO) together with an RTD or thermocouple input as an industry-standard 4-terminal interface. Alternatively, it can be used to create two-channel differential analog inputs or six-channels of single-ended analog inputs. A microcontroller-compatible serial peripheral interface (SPI) provides access to many advanced features. This application note provides example C-code implementation including setup, monitoring, and diagnostic functions.

Introduction

The MAX22000 integrates a 24-bit ADC, an 18-bit DAC, and an analog front-end (AFE) to create a software-configurable IO that supports all standard industrial analog interfaces: -10V to +10V analog input or output, -20mA to +20mA analog input or output, as well as an RTD or thermocouple input for temperature measurement. When used as an analog input (AI), the device supports either differential input channels with two single-ended inputs or up to six single-ended input channels. Additionally, one differential input channel (AI5 and AI6) has an integrated low-noise Programmable Gain Amplifier (PGA) designed for thermocouple or RTD measurements.

If the analog output (AO) is used, then it needs one differential pair of analog inputs in current-output mode, or one single-ended analog input channel in voltage-output mode.

This application note presents a series of functions to provide a faster and proven solution to programming the MAX22000 (**Figure 1**). They are written in C and should prove easy to port over to any common microcontroller. For detailed information regarding the MAX22000 pins, operating modes, and control registers, refer to the MAX22000 data sheet.

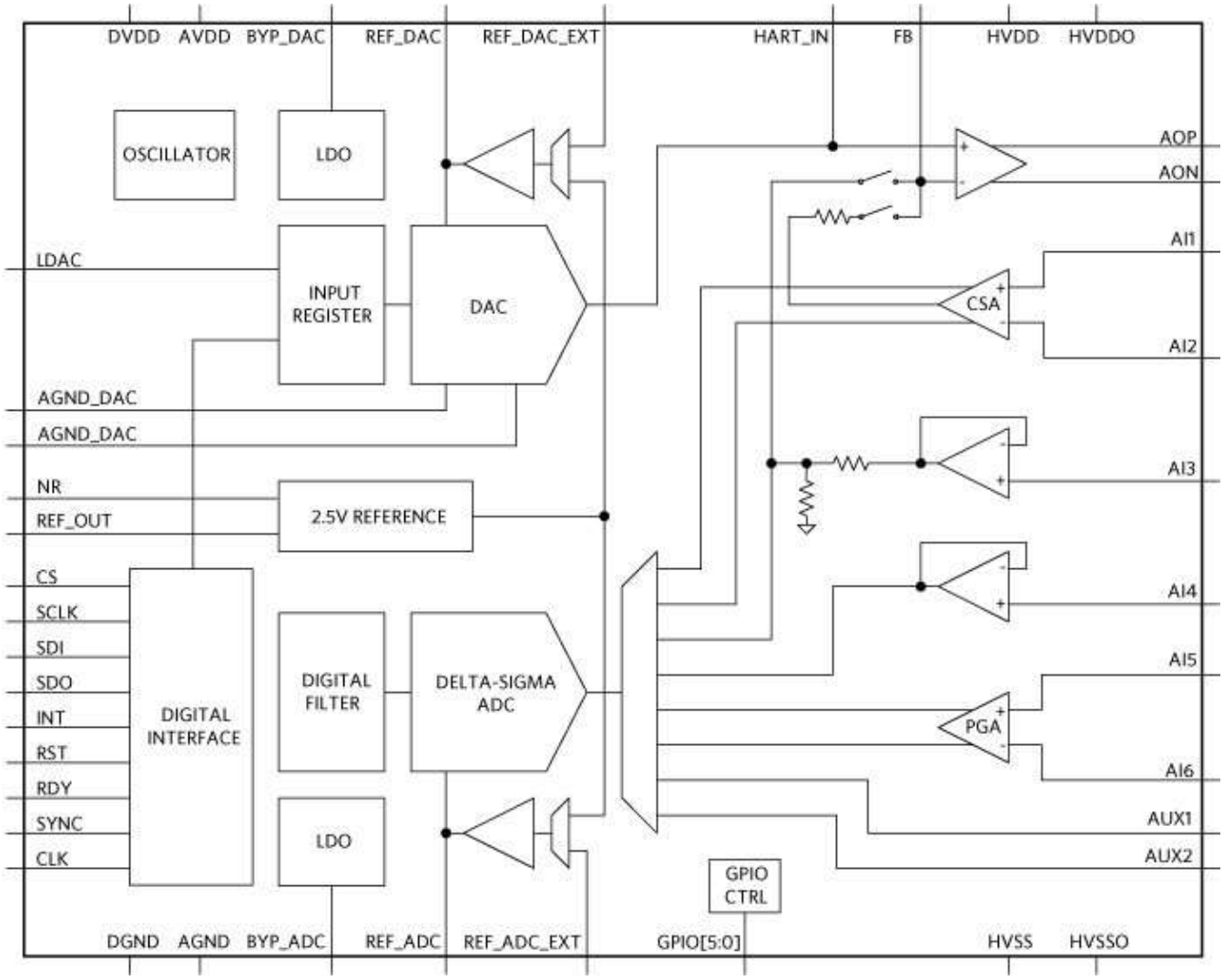

Figure 1. MAX22000 Functional Diagram

MAX22000 SPI

The MAX22000 SPI commands are 32 bits long (8-bit instruction + 24-bit data) with CRC disabled, and if CRC is enabled this adds 8 more bits for the CRC8; for details about CRC calculation, please see Application Note 7072: Guidelines to Implement CRC Programming for the MAX22000 Configurable Analog IO, which shows CRC8 encoding in more detail. The SPI command structure is shown in **Table 1**. SPI mode for the MAX22000 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. MAX22000 SPI Command Structure

Full details of SPI read and write cycles, along with register tables and instructions, can be found in the MAX22000 data sheet.

Address	Control	Data
7-bits A[6:0], MSB to LSB	R/W bit, Read = 1, Write = 0	24-bits D[23:0], MSB to LSB

Figure 1 shows the main function blocks of the MAX22000. Essentially, there are four main parts to the device:

- Sigma-delta ADC with an internal voltage reference – the main function is to convert the analog data that can be read using the SPI.
- DAC with internal voltage reference – the main function is to convert digital data to an analog voltage.
- AFE with multiplexer – the main function is to select channels and switch modes (i.e., current/voltage).
- Logic-side interface – SPI port for accessing all device registers and hardware flags for diagnostic.

MAX22000—Application Examples Configurable, Multi-Range Analog Input/Output

The MAX22000 is designed to support industrial applications in end equipment such as programmable logic controllers (PLCs) that require configurable analog I/O. A typical application circuit is shown in **Figure 2**.

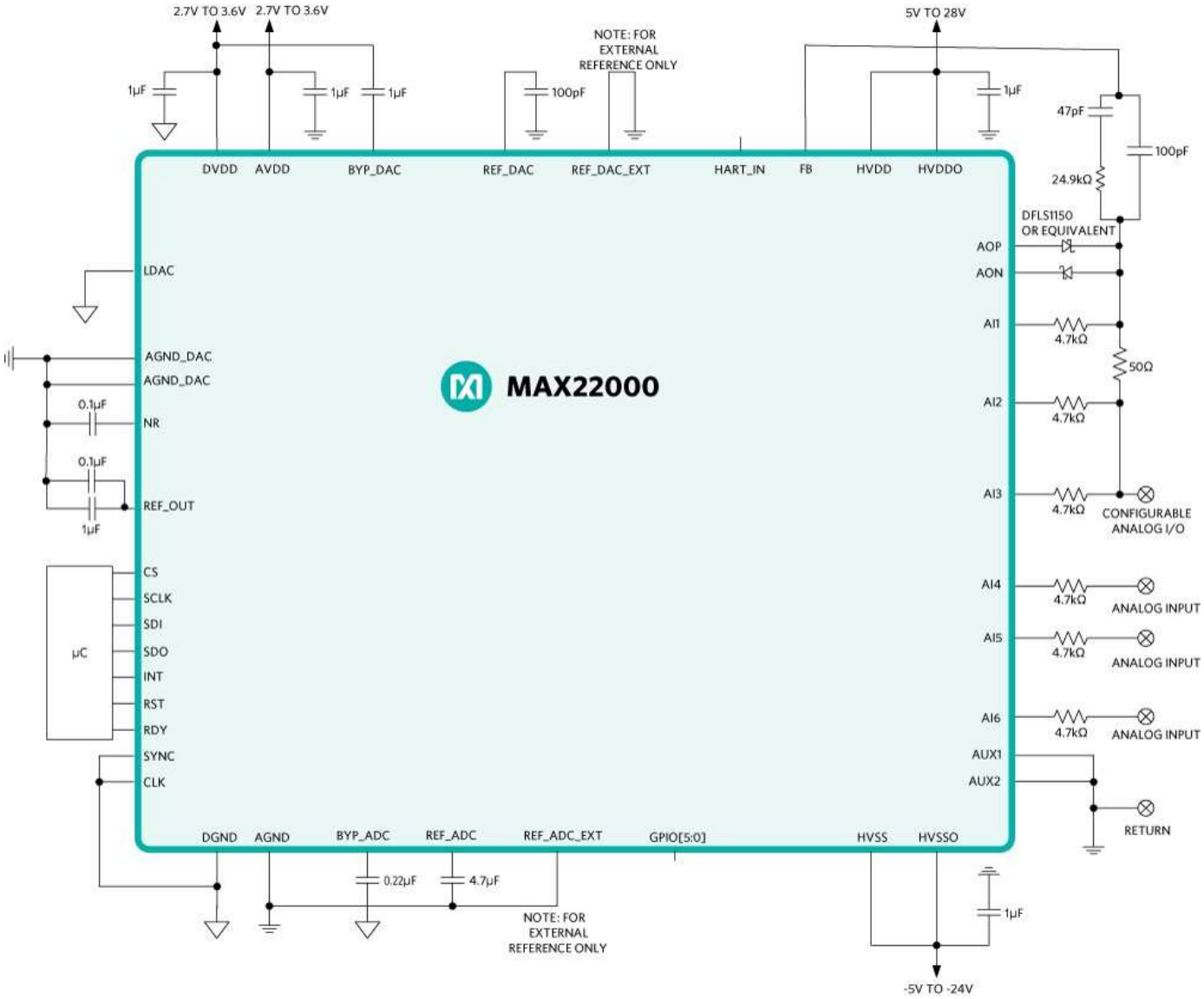

Figure 2. Configurable Analog IO + 3 Single-Ended Analog Voltage Inputs.

The terminal labeled 'Universal Analog I/O' is the software-configurable and fully flexible analog IO port, which in conjunction with the terminal labeled 'Return' provides either:

- Current Input ±20mA (nominal) full-scale range is ±25mA.
- Current Output ±20mA (nominal) full-scale range is ±25mA.
- Voltage Input ±10V (nominal), full-scale range is ±12.5V.
- Voltage Output ±10V (nominal), full-scale range is ±12.5V.

To simplify systems which require galvanic isolation, the MAX22000 supports six logic-level GPIOs (GPIO[5:0]), which can be used in case external components (MUXes/FETs/power supplies) need to be switched or digital signals need to be read back through the isolation barrier.

Source Code

This application note provides C source code examples, essentially providing driver functions to access the multiple registers within the MAX22000 for configuration, control, and diagnostic features. All software has been implemented and tested using MAX22000 EVkit.

Globals to allow easy channel/mode selection:

```
        public enum Register_address
        {
            GEN_PROD          = 0x00,
            GEN_REV           = 0x01,
            GEN_CNFG          = 0x02,
            GEN_CHNL_CTRL     = 0x03,
            GEN_GPIO_CTRL     = 0x04,
            GEN_GPI_INT       = 0x05,
            GEN_GPI_DATA      = 0x06,
            GEN_INT           = 0x07,
            GEN_INTEN         = 0x08,
            GEN_PWR_CTRL      = 0x09,
            GEN_TST_MODE_1    = 0x1a,
            GEN_TST_ENTRY     = 0x1c,
            GEN_TST69         = 0x1d,
            DCHNL_CMD         = 0x20,
            DCHNL_STA         = 0x21,
            DCHNL_CTRL1       = 0x22,
            DCHNL_CTRL2       = 0x23,
            DCHNL_DATA        = 0x24,
            DCHNL_N_SEL       = 0x25,
            DCHNL_N_SOC       = 0x26,
            DCHNL_N_SGC       = 0x27,
            AO_DATA_WR        = 0x40,
            AO_OFFSET_CORR_WR = 0x41,
            AO_GAIN_CORR_WR   = 0x42,
            AO_CNFG_WR        = 0x43,
            AO_DATA_RD        = 0x44,
            AO_OFFSET_CORR_RD = 0x45,
            AO_GAIN_CORR_RD   = 0x46,
            AO_STA_RD         = 0x47,
        };
```

```
            // DAC is 18bit, full-range = 262144; Half because bipolar: 131072, Theoretical facto
r = V(range) / half-range
            public const double phy_A0_25V_factor  = (double) 25 / (double) 262144;    //0.000095
367431641;
            public const double phy_A0_25V_offset  = -131072;


            public const double phy_A0_12V_factor  = (double) 25 / (double) 262144;    //0.000095
367431641;
            public const double phy_A0_12V_offset  = 0;


            // DAC is 18bit, full-range = 262144; Theoretical factor = mA(range) / half-range (bi
polar)
            public const double phy_A0_25mA_factor = (double) 50 / (double) 262144;   // 0.000190
73486633;
            public const double phy_A0_25mA_offset = 0;
            public const double phy_A0_2mA_factor  = (double) 5  / (double) 262144;   // 0.000019
073486633;
            public const double phy_A0_2mA_offset  = 0;



        public enum Channel_select
        {
            AI1_SE_b12V         = 0x00,
            AI2_SE_b12V         = 0x01,
            AI1_2_diff_b1V      = 0x02,
            AI3_SE_b12V         = 0x03,
            AI4_SE_b12V         = 0x04,
            AI3_4_diff_b25V     = 0x05,
            AI5_SE_b12V         = 0x06,
            AI6_SE_b12V         = 0x07,
            AI5_6_diff_b25V     = 0x08,
            AI5_SE_b0p125V      = 0x09,
            AI5_SE_b0p250V      = 0x0a,
            AI5_SE_b0p500V      = 0x0b,
            AI5_SE_b2p500V      = 0x0c,
            AI6_SE_b0p125V      = 0x0d,
            AI6_SE_b0p250V      = 0x0e,
            AI6_SE_b0p500V      = 0x0f,
            AI6_SE_b2p500V      = 0x10,
            AI5_6_diff_b0p125V  = 0x11,
            AI5_6_diff_b0p250V  = 0x12,
            AI5_6_diff_b0p500V  = 0x13,
            AI5_6_diff_b2p500V  = 0x14,
```

```
            AUX1_SE_u2V         = 0x15,
            AUX2_SE_u2V         = 0x16,
            AUX1_2_diff_b2V     = 0x17,
        };

        public enum A0ut_Mode
        {
            high_impedance = 0,
            A0_25V         = 1,
            A0_12V         = 2,
            A0_6V          = 3,
            A0_1V          = 4,
            A0_0p6V        = 5,
            A0_25mA        = 6,
            A0_12mA        = 7,
            A0_2mA         = 8,
            A0_1mA         = 9,
            out_of_range1  = 10
        }

public enum MAX22000_CI0_Mode
{
    Current_Input,
    Voltage_Input,
    Current_0utput,
    Voltage_0utput,
    RTD_Input,
    PGA_Input,
    PGA_Input_with_Current_sourcing,
    0ff
};
//*********************************************************************
//*
//* Function: MAX22000_read_register
//* Description: Read one Register from MAX22000
//*
//* Input: Register-Address (take from definitions in header-file)
//* 0utput: 24bit register content
//*
//* if CRC is enabled, then crc8-Command is required
//*
```

```
//*********************************************************************/
public UInt32 MAX22000EVKIT_read_register(Register_address address)
{
    UInt32 result = 0;
    UInt32 CRC_read = 0;

    if (CRC_Enabled == false)
    {
        max22000_port.SPI_CSOEnable();
        max22000_port.SPI_W_transaction_8( (address << 1) + 0x01 );
        result = max22000_port.SPI_R_transaction_24();
        max22000_port.SPI_CSODisable();
    }
    else
    {
        max22000_port.SPI_CSOEnable();
        max22000_port.SPI_W_transaction_8( (address << 1) + 0x01 );
        result    = max22000_port.SPI_R_transaction_24();
        CRC_read  = max22000_port.SPI_R_transaction_8();  // read the CRC
        max22000_port.SPI_CSODisable();

        // calculate and check...
        byte CRC_TX1    = (address << 1) + 0x01;   // TX byte (was sent)
        byte CRC_RX1    = (result >>16)  & 0xff;   // 1st RX byte
        byte CRC_RX2    = (result >>8 )  & 0xff;   // 2nd RX byte
        byte CRC_RX3    = (result     )  & 0xff;   // 3rd RX byte
        byte CRC_Calc = crc8(CRC_TX1, CRC_RX1, CRC_RX2, CRC_RX3);

         if (CRC_Calc != CRC_read) printf("CRC read from MAX22000 is incorrect.");
    }

    return result;
}

//*********************************************************************
//*
//* Function: MAX22000_write_register
//* Description: Write one Register to MAX22000
//*
//* Input: Register-Address (take from definitions in header-file)
//*        24bit data (new register content)
```

```
//*
//*********************************************************************/
public UInt32 MAX22000EVKIT_write_register(Register_address address, UInt32 data)
{
    if (CRC_Enabled == false)
    {
        max22000_port.SPI_CSOEnable();
        max22000_port.SPI_W_transaction_8( (ushort) ( ((byte)address << 1) )  );
        max22000_port.SPI_W_transaction_24(data);
        max22000_port.SPI_CSODisable();
    }
    else
    {
        byte CRC_TX1 = (address << 1);
        byte CRC_TX2 = ((data> >16) & 0xff);
        byte CRC_TX3 = ((data >>8 ) & 0xff);
        byte CRC_TX4 = ( data       & 0xff);

        byte CRC_Calc = crc8(CRC_TX1, CRC_TX2, CRC_TX3, CRC_TX4);

        max22000_port.SPI_CSOEnable();
        max22000_port.SPI_W_transaction_8( address << 1 );
        max22000_port.SPI_W_transaction_24(data);
        max22000_port.SPI_W_transaction_8( CRC_Calc );
        max22000_port.SPI_CSODisable();
    }
}

// *********************************************************************
//
// Function: MAX22000_CI0_Setup
// Description: Sets up MAX22000 for one of the CI0 Modes
//              Assuming HW is connected like the standard application diagram
//
// Input: Desired Mode
// Output: None (MAX22000 will be setup by this routine)
//
// *********************************************************************
uint32_t MAX22000_CI0_Setup (MAX22000_CI0_Mode mode)
{
          uint32_t ADC_result = 0;
```

```
    switch (mode)
    {
        case MAX22000_CI0_Mode.Current_Input:
            // set calibration factors for Current Input
            MAX22000_write_register(MAX22000_DCHNL_CMD, 0x100000);        // Stop any potentially
running conversions
            MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000000);    // No channel selected
            MAX22000_write_register(MAX22000_DCHNL_N_SEL,   0x000003);    // Select CH AI1-2 diff
calibration factor
            MAX22000_write_register(MAX22000_DCHNL_N_S0C,   0xFFFFE1);    // Write Offset (regula
r low-side)
            MAX22000_write_register(MAX22000_DCHNL_N_SGC,   0xBD934B);    // Write Gain   (regula
r low-side)
            // END restore calibration

            uint32_t new_GEN_CNFG = 0;
            new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x008000;          // AI1-2 diff (CSA-Mode
)
            new_GEN_CNFG = (new_GEN_CNFG & 0xf0ffff) + 0x020000;          // A0 voltage output mo
de +/- 12.5V range
             MAX22000_write_register(MAX22000_GEN_CNFG,     new_GEN_CNFG); // Write new_GEN_CNFG_
value
             MAX22000_write_register(MAX22000_A0_DATA_WR,   0x000000);     // Write A0 Voltage to
0V so current can flow

                              // Prepare ADC
            MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000300);    // AI1-2 diff channel s
elected

            MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single cy
cle conversions
            MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use co
efficients, ...

            MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000300);    // select AI1-2 diff mo
de
            MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000); // Start conversion (10sps
)    (0x340000 = 50sps)

            ADC_result = MAX22000EVKIT_read_register(MAX22000_DCHNL_DATA);// read Data
            break;

        case MAX22000_CI0_Mode.Current_Output:
            // Set DAC calibration
            MAX22000_write_register(MAX22000_A0_OFFSET_CORR_WR, 0x00F5C0);
            MAX22000_write_register(MAX22000_A0_GAIN_CORR_WR,   0xFA3E80);
```

```
            // END restore calibration


            // Set A0 Mode (Register 0x02: GEN_CNFG)
            uint32_t new_GEN_CNFG = 0;
            new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x008000;          // AI1-2 diff (CSA-Mode
)
            new_GEN_CNFG = (new_GEN_CNFG & 0xf0ffff) + 0x060000;          // A0 current output mo
de +/- 25mA range
            new_GEN_CNFG = (new_GEN_CNFG & 0xefffff) + 0x000000;          // Set 4-wire Mode
            new_GEN_CNFG = (new_GEN_CNFG & 0xffefff) + 0x001000;          // enable AI3 SE
            new_GEN_CNFG = (new_GEN_CNFG & 0xbfffff) + 0x000000;              // Internal Referen
ce
             MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNFG_
value


             MAX22000_write_register(MAX22000_GEN_PWR_CTRL, 0x000000);  // Normal operation (espe
cially make sure GEN_PD=0


            // Set Hex value / physical Value
             MAX22000_write_register(MAX22000_A0_DATA_WR,   0);//new_A0_value<<6);     // Write n
ew DAC value
            break;


        case MAX22000_CI0_Mode.Voltage_Input:
            // restore calibration
            MAX22000_write_register(MAX22000_DCHNL_CMD, 0x100000);        // Stop any potentially
running conversions
            MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000000);    // No channel selected
            MAX22000_write_register(MAX22000_DCHNL_N_SEL,   0x000003);    // Select CH AI3 SE cal
ibration factor
            MAX22000_write_register(MAX22000_DCHNL_N_SOC,   0xFFFFE1);    // Write Offset
            MAX22000_write_register(MAX22000_DCHNL_N_SGC,   0xBD934B);    // Write Gain
            // END restore calibration


            uint32_t new_GEN_CNFG = 0;
            new_GEN_CNFG = (new_GEN_CNFG & 0xffefff) + 0x001000;              // AI3 SE enabled


            new_GEN_CNFG = (new_GEN_CNFG & 0xf0ffff) + 0x000000;          // A0 High-Impedance mo
de
            new_GEN_CNFG = (new_GEN_CNFG & 0xefffff) + 0x000000;          // Set 4-wire Mode
            new_GEN_CNFG = (new_GEN_CNFG & 0xbfffff) + 0x000000;              // Internal Referen
ce
            MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNFG_v
alue


            // Select Internal REFs, set all channels to single-ended leave as is
```

```
            MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single cy
cle conversions
            MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use co
efficients, ...


            MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000400);    // Select Channel AI3 /
Single Ended


            //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversion (
30sps)
            MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion (10
sps)
            break;


        case MAX22000_CI0_Mode.Voltage_Output:
            MAX22000_write_register(MAX22000_A0_OFFSET_CORR_WR, 0x00F5C0);
             MAX22000_write_register(MAX22000_A0_GAIN_CORR_WR,   0xFA3E80);
             // END restore calibration


             uint32_t new_GEN_CNFG = 0;
             new_GEN_CNFG = (new_GEN_CNFG & 0xf0ffff) + 0x020000;          // 10V   is 0b0010 -> 2
             new_GEN_CNFG = (new_GEN_CNFG & 0xeFffff) + 0x000000;           // Set 4-wire Mode
             new_GEN_CNFG = (new_GEN_CNFG & 0xbfffff) + 0x000000;        // Internal Reference
             MAX22000_write_register(MAX22000_GEN_CNFG, new_GEN_CNFG);     // Write new_GEN_CNFG_
value


            // Set Hex value / physical Value
            MAX22000_write_register(MAX22000_A0_DATA_WR,   0);//new_A0_value<<6);     // Write ne
w DAC value
            break;


        case MAX22000_CI0_Mode.Off:
                 // Make A0 high-impedance, Stop ADC, Disable all Amplifiers
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x100000);    // Stop any potenti
ally running conversions
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000000);    // No channel selec
ted
                MAX22000_write_register(MAX22000_GEN_CNFG,      0x000000);    // Write new_GEN_CN
FG_value
            break;


        case default:
                 // Make A0 high-impedance, Stop ADC, Disable all Amplifiers
                 MAX22000_write_register(MAX22000_DCHNL_CMD,     0x100000);    // Stop any potent
ially running conversions
```

```
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000000);    // No channel sele
cted
                MAX22000_write_register(MAX22000_GEN_CNFG,      0x000000);    // Write new_GEN_C
NFG_value
            break;
    }
}


// ***********************************************************************
//
// Function: MAX22000_ADC_Setup
// Description: Sets up MAX22000 for to read one of the ADC Channels in selected Mode
//              Assuming all ADCs (at least the selected one) is open / connected to a voltage so
urce
//
// Input:  Desired ADC-Channel +Mode
// Output: None (MAX22000 will be setup by this routine, Conversion will be started)
//
// ***********************************************************************
uint32_t MAX22000_ADC_Setup (ADC_CH_Mode CH_and_Mode)
{
          switch(CH_and_Mode)
          {
            case ADC_CH_Mode.AI1_SE:
                // Setup Channel 1 for Single Ended and continuous sampling
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x006000;           // make AI1 and AI2
SE active
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000100);    // Select Channel AI
1 / Single Ended
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;


            case ADC_CH_Mode.AI2_SE:
                // Setup Channel 2 for Single Ended and continuous sampling
```

```
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x006000;          // make AI1 and AI2
SE active
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000200);    // Select Channel AI
1 / Single Ended
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;


            case ADC_CH_Mode.AI3_SE:
                // Setup Channel 3 for Single Ended and continuous sampling
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x001800;          // make AI3 and AI4
SE active
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000400);    // Select Channel AI
3 / Single Ended
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;


            case ADC_CH_Mode.AI4_SE:
                // Setup Channel 4 for Single Ended and continuous sampling
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x000800;          // make AI4 SE activ
e
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
```

```
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000500);    // Select Channel AI
4 / Single Ended
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;


            case ADC_CH_Mode.AI12_DIFF:
                // Setup Channel 1-2 for Differeential and continuous sampling
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x008000;          // enable AI1-2 diff
(CSA) mode
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000300);    // Select Channel AI
1-2 / Differential
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;


             case ADC_CH_Mode.AI34_DIFF:
                // Setup Channel 3-4 for Differeential and continuous sampling
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x001800;          // enable AI3-AI4 di
ff
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000600);    // Select Channel AI
3-4 / Differential
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;
```

```
            case ADC_CH_Mode.AI56_DIFF:
                // Setup Channel 5-6 for Diffeeential and continuous sampling
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x000400;          // enable AI5-6 diff
+/- 25V range
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000900);// Select Channel AI5-6
/ Differential +/-25V
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;

            case ADC_CH_Mode.AI56_PGA_2p500V:
                // Setup Channel 5-6 for Diffeeential and continuous sampling (PGA Path +/-2.5V r
ange)
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x000400;   // enable AI5-6 diff +/- 2.
5V range (in PGA Mode)
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000c00); // Select Channel AI5-6
/ Differential PGA Path
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;

            case ADC_CH_Mode.AI56_PGA_0p500V:
                // Setup Channel 5-6 for Diffeeential and continuous sampling (PGA Path +/-0.5V r
ange)
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x000440;          // enable AI5-6 diff
+/- 0.5V range
```

```
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value

                // Select Internal REFs, set all channels to single-ended leave as is

                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions

                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...

                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000c00); // Select Channel AI5-6
/ Differential PGA Path

                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)

                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)

                break;


            case ADC_CH_Mode.AI56_PGA_0p250V:

                // Setup Channel 5-6 for Differeential and continuous sampling (PGA Path +/-0.250V
range)

                uint32_t new_GEN_CNFG = 0;

                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x000480;          // enable AI5-6 diff
+/- 0.250V range

                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value

                // Select Internal REFs, set all channels to single-ended leave as is

                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions

                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...

                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000c00); // Select Channel AI5-6
/ Differential PGA Path

                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)

                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)

                break;


            case ADC_CH_Mode.AI56_PGA_0p125V:

                // Setup Channel 5-6 for Differeential and continuous sampling (PGA Path +/-0.250V
range)

                uint32_t new_GEN_CNFG = 0;

                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x0004c0;          // enable AI5-6 diff
+/- 0.125V range

                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CNF
G_value

                // Select Internal REFs, set all channels to single-ended leave as is

                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions

                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
```

```
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000c00); // Select Channel AI5-6
/ Differential PGA Path
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;

            case ADC_CH_Mode.AUX1_SE:
                // Setup Channel AUX1 for Single Ended and continuous sampling
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000d00);    // Select Channel AU
X1 / Single Ended
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;

            case ADC_CH_Mode.AUX2_SE:
                // Setup Channel AUX2 for Single Ended and continuous sampling
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000e00);    // Select Channel AU
X2 / Single Ended
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;

            case ADC_CH_Mode.AUX12_DIFF:
                // Setup Channel AUX1-2 for Differential and continuous sampling
                // Select Internal REFs, set all channels to single-ended leave as is
                MAX22000_write_register(MAX22000_DCHNL_CTRL1,   0x010000);    // Continuous Single
cycle conversions
                MAX22000_write_register(MAX22000_DCHNL_CTRL2,   0x000000);    // Internal OSC, use
coefficients, ...
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000f00);    // Select Channel AU
X1-2 / Differential
                //MAX22000_write_register(MAX22000_DCHNL_CMD,     0x330000);    // Start conversio
n (30sps)
```

```
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x310000);    // Start conversion
(10sps)
                break;


            case default:
                // Make A0 high-impedance, Stop ADC, Disable all Amplifiers
                MAX22000_write_register(MAX22000_DCHNL_CMD,     0x100000);    // Stop any potentia
lly running conversions
                MAX22000_write_register(MAX22000_GEN_CHNL_CTRL, 0x000000);    // No channel select
ed
                MAX22000_write_register(MAX22000_GEN_CNFG,      0x000000);    // Write new_GEN_CNF
G_value
                break;
          }
}


// ***********************************************************************
//
// Function: MAX22000_ADC_Read
// Description: Reads the currently selected and running ADC Channel
//              as setup per MAX22000_ADC_Setup
//
// Input: None
// Output: Current ADC Reading in LSB (24bit wide)
//
// ***********************************************************************
uint32_t MAX22000_ADC_Read  (void)
{
          uint32_t adc_result = 0;

          wait_for_RDYB(); // When RDYB pin is low, the ADC finished conversion.
          adc_result = MAX22000EVKIT_read_register(MAX22000_DCHNL_DATA);// read Data

          return adc_result;
}



// ***********************************************************************
//
// Function: MAX22000_DAC_Setup
// Description: Sets up the DAC for the selected Mode
//              after this the DAC can be updated with DAC_Set_LSB or DAC_Ser_PHY
```

```
//
// Input: DAC range
// Output: None, DAC in MAX22000 will be setup according to setting
//
// *********************************************************************
void MAX22000_DAC_Setup (DAC_Range range)
{
          switch (range)
          {
              case DAC_Range.A0_25V:
                 uint32_t new_GEN_CNFG = 0;
                 new_GEN_CNFG = (new_GEN_CNFG & 0xf0ffff) + 0x010000;    // A0_CNFG = 0001: A0 Cur
rent Mode, 25V setting
                 MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CN
FG_value
                 // Restore Calibration
                 MAX22000_write_register(MAX22000_A0_OFFSET_CORR_WR, 0x00F5C0);
                 MAX22000_write_register(MAX22000_A0_GAIN_CORR_WR,   0xFA3E80);
                 break;
              case DAC_Range.A0_12V:
                 uint32_t new_GEN_CNFG = 0;
                 new_GEN_CNFG = (new_GEN_CNFG & 0xf0ffff) + 0x020000;    // A0_CNFG = 0010: A0 Cur
rent Mode, 12.5V setting
                 MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CN
FG_value
                 // Restore Calibration
                 MAX22000_write_register(MAX22000_A0_OFFSET_CORR_WR, 0x00F5C0);
                 MAX22000_write_register(MAX22000_A0_GAIN_CORR_WR,   0xFA3E80);
                 break;

              case DAC_Range.A0_25mA:
                 uint32_t new_GEN_CNFG = 0;
                 new_GEN_CNFG = (new_GEN_CNFG & 0xf0ffff) + 0x060000;    // A0_CNFG = 0110: A0 Cur
rent Mode, 25mA setting
                 new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x008000;          // AI1-2 diff (CSA-
Mode)
                 MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CN
FG_value
                 // Restore Calibration
                 MAX22000_write_register(MAX22000_A0_OFFSET_CORR_WR, 0x00F5C0);
                 MAX22000_write_register(MAX22000_A0_GAIN_CORR_WR,   0xFA3E80);
                 break;

              case DAC_Range.A0_2mA:
```

```
                uint32_t new_GEN_CNFG = 0;
                new_GEN_CNFG = (new_GEN_CNFG & 0xf0ffff) + 0x080000;    // A0_CNFG = 1000: A0 Cur
rent Mode, 2.5mA setting
                new_GEN_CNFG = (new_GEN_CNFG & 0xff1fff) + 0x008000;          // AI1-2 diff (CSA-
Mode)
                MAX22000_write_register(MAX22000_GEN_CNFG,   new_GEN_CNFG);   // Write new_GEN_CN
FG_value
                // Restore Calibration
                MAX22000_write_register(MAX22000_A0_OFFSET_CORR_WR, 0x00F5C0);
                MAX22000_write_register(MAX22000_A0_GAIN_CORR_WR,   0xFA3E80);
                break;

             default:
                // In case invalid range select, do nothing
                break;
          }
}

// ***********************************************************************
//
// Function: MAX22000_DAC_Set_LSB
// Description: Writes a new LSB value to the DAC,
//              assuming it is already setup in a specific mode, use DAC_Setup first
//              If LDAC-pin is high, it must be toggled after setting up update the output
//
// Input: new DAC value in LSB
// Output: None
//
// ***********************************************************************
void     MAX22000_DAC_Set_LSB   (uint32_t data)
{
          // DAC must be setup before using this function
          // Below will simply write the new Value to the DAC
          MAX22000_write_register(MAX22000_A0_DATA_WR,   data);
}

// ***********************************************************************
//
// Function: MAX22000_DAC_Set_PHY
// Description: Writes a new PHY value (Volt or mA) to the DAC,
//              assuming it is already setup in a specific mode, use DAC_Setup first
//              If LDAC-pin is high, it must be toggled after setting up update the output
```

```
//
// Input: new DAC value in physical value (either Volt or Miliampere, NOT AMPERE)
// Output: None
//
// ********************************************************************
void     MAX22000_DAC_Set_PHY (float volt_V_or_current_mA, DAC_Range range)
{
          // DAC must be setup before using this function
          // Calculate new LSB Value
          uint32_t DAC_LSB_value = 0;

          switch (range)
          {
             case DAC_Range.A0_25V:
                if (volt_V_or_current_mA < 25)
                { DAC_LSB_value = 0x1ffff + ((volt_V_or_current_mA / (phy_A0_25V_factor))) + phy_A
0_25V_offset + 1;  }
                else
                { DAC_LSB_value = -0x1ffff + ((volt_V_or_current_mA / (phy_A0_25V_factor)) + phy_A
0_25V_offset - 0); }
                break;

             case DAC_Range.A0_12V:
                if (volt_V_or_current_mA < 0)
                { DAC_LSB_value = 0x3ffff - ((-volt_V_or_current_mA / (phy_A0_12V_factor))) + phy_
A0_12V_offset + 1; }
                else
                { DAC_LSB_value = ((volt_V_or_current_mA / (phy_A0_12V_factor)) + phy_A0_12V_offse
t - 0); }
                break;



             case DAC_Range.A0_25mA:
                if (volt_V_or_current_mA < 0)
                { DAC_LSB_value = 0x3ffff - ((-volt_V_or_current_mA / (phy_A0_25mA_factor))) + phy
_A0_25mA_offset + 1; }
                else
                { DAC_LSB_value = ((volt_V_or_current_mA / (phy_A0_25mA_factor)) + phy_A0_25mA_off
set - 0);  }
                break;

             case DAC_Range.A0_2mA:
                if (volt_V_or_current_mA < 0)
```

```
                { DAC_LSB_value = 0x3ffff - ((-volt_V_or_current_mA / (phy_A0_2mA_factor))) + phy_
A0_2mA_offset + 1; }
                else
                { DAC_LSB_value = ((volt_V_or_current_mA / (phy_A0_2mA_factor)) + phy_A0_2mA_offse
t - 0); }
                break;


             default:
                DAC_LSB_value = 0; // default means non-existend range selected
                break;
          }


// ********************************************************************
//
// Function: MAX22000_GPIO_Setup
// Description: Sets up all 6 GPIO Pins, bit0=GPIO0, bit1=GPIO1, ...
//               Since the command includes everything Enable/Disable as well as
//               GPIO Direction, this function is faster than GP0_Set
//               because it doesn't have to read back the setup from the part
//
// Input:  GPIO_enable (byte)    Bit0 = GPIO0, Bit1 = GPIO1, ... (0 = Off,   1 = On)
//         GPIO_direction (byte) Bit0 = GPIO0, Bit1 = GPIO1, ... (0 = Input, 1 = Output)
//         GP0_Setting (byte)    Bit0 = GPIO0, Bit1 = GPIO1, ... (0 = Low,   1 = High)
// Output: None
//
// ********************************************************************
void MAX22000_GPIO_Setup (uint8_t GPIO_enable, uint8_t GPIO_direction,  uint8_t GP0_Setting)
{
    uint32_t new_gpio_value = ((GPIO_enable & 0x3f)<<16) + ((GPIO_direction & 0x3f)<<8) + (GP0_Se
tting & 0x3f);
    MAX22000_write_register(MAX22000_GEN_GPIO_CTRL,   new_gpio_value);   // Write new_GEN_CNFG_va
lue
}


// ********************************************************************
//
// Function: MAX22000_GP0_Set
// Description: Sets GP0s high or low, bit0=GPIO0, bit1=GPIO1, ...
//               GP0s must be setup and enabled prior this use MAX22000_GPIO_Setup
//
// Input: GP0 Setting, bit0=GPIO0, bit1=GPIO1, ... (0 = Low, 1 = High)
// Output: None
```

```
//

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

void MAX22000_GP0_Set (uint8_t GP0_Setting)

{

    uint32_t gpio_setup = MAX22000EVKIT_read_register(MAX22000_GEN_GPI0_CTRL);  // read Setup

    gpio_setup = gpio_setup & 0xffff00;                                         // Mask out previ
ous GP0 settings

    MAX22000_write_register(MAX22000_GEN_GPI0_CTRL,   gpio_setup);              // Write new_GEN_
CNFG_value

}


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

//

// Function: MAX22000_GPI_Get

// Description: Gets all GPI readings high or low, bit0=GPI00, bit1=GPI01, ...

//              GPIs must be setup and enabled prior this use MAX22000_GPI0_Setup

//

// Input: None

// Output: GPI Setting, bit0=GPI00, bit1=GPI01, ... (0 = Low, 1 = High)

//

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

uint8_t  MAX22000_GPI_Get (void)

{

    uint32_t gpi_result = MAX22000EVKIT_read_register(MAX22000_GEN_GPI_DATA);  // read GPI Data

    return   gpi_result & 0x3f;

}
```

CRC Function

Please note, the below CRC function is described in more detail in Application Note 7072, but since some of above functions depend on CRC, the code is provided here as well for more convenience.

```
        public byte crc8(byte BYTE1, byte BYTE2, byte BYTE3, byte BYTE4)

        {

            byte crc8_start = 0x00;

            byte crc8_poly  = 0x8c; // rotated 0x31, which is our polinomial

            byte crc_result = crc8_start;


            // BYTE1

            for (int i=0; i<8; i++)

            {

                if( ( (( BYTE1>>i ) ^ (crc_result) ) & 0x01 ) > 0 )

                { crc_result = (byte) (crc8_poly ^ crc_result>>1 );  }
```

```
            else
            { crc_result = (byte) (crc_result>>1);              }
        }

        // BYTE2
        for (int i=0; i<8; i++)
        {
            if( ( (( BYTE2>>i ) ^ (crc_result) ) & 0x01 ) > 0 )
            { crc_result = (byte) (crc8_poly ^ crc_result>>1 );  }
            else
            { crc_result = (byte) (crc_result>>1);              }
        }

        // BYTE3
        for (int i=0; i<8; i++)
        {
            if( ( (( BYTE3>>i ) ^ (crc_result) ) & 0x01 ) > 0 )
            { crc_result = (byte) (crc8_poly ^ crc_result>>1 );  }
            else
            { crc_result = (byte) (crc_result>>1);              }
        }

        // BYTE4
        for (int i=0; i<8; i++)
        {
            if( ( (( BYTE4>>i ) ^ (crc_result) ) & 0x01 ) > 0 )
            { crc_result = (byte) (crc8_poly ^ crc_result>>1 );  }
            else
            { crc_result = (byte) (crc_result>>1);              }
        }

        crc8_2_for_testing(BYTE1, BYTE2, BYTE3, BYTE4);

        return crc_result;
    }
```