Application Note – AN5013-20 Firmware coding

AS5013

Two-dimensional Magnetic Position Sensor with Digital Coordinates output

Table of Contents

1. General Description

This application describe the firmware of two typical applications using a microcontroller with an AS5013 based EasyPoint™ module.

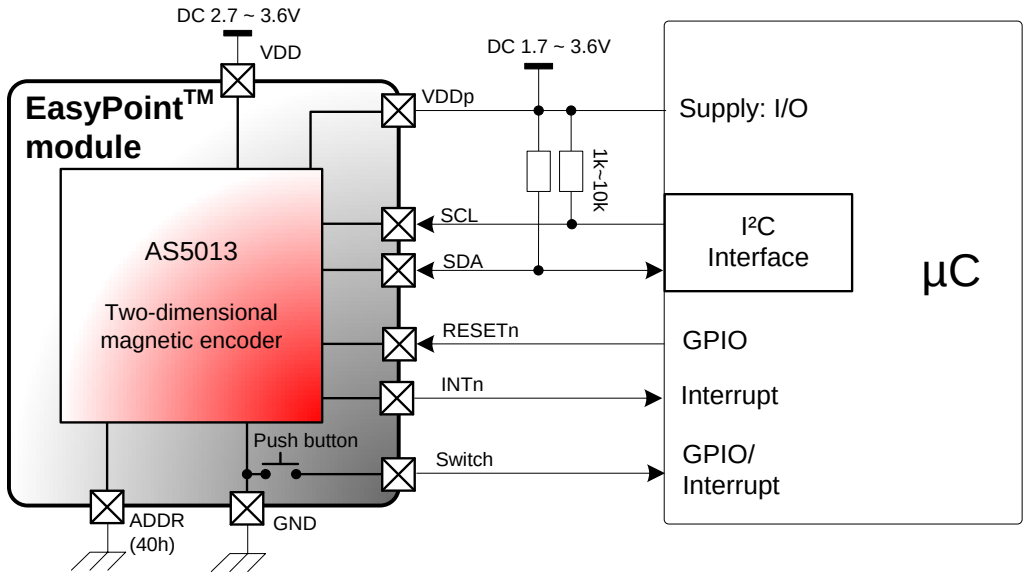
The battery powered application is suitable for battery powered handheld devices as mobile phones, portable GPS, Netbooks, remote controls, wireless joysticks. In idle state, when the knob of the joystick is released, the module is in Low Power mode, with a slow readout rate (max. 320ms). The interrupt is configured in motion detection mode. If the knob is moved above a programmable threshold, the AS5013 sends an interrupt to the MCU to e.g. wake it up. The MCU configures the AS5013 with a higher readout rate for faster reaction (20ms, 40ms, or using any readout rate synchronously from the microcontroller in Idle mode).

When the knob has been released, the AS5013 will be configured back with a slower readout rate and motion detection mode after some delay, for a better battery life.

The wire powered application is suitable for any other application where current consumption is not an issue. The AS5013 is configured with a permanent high readout rate (20ms, 40ms, or using any readout rate synchronously from the microcontroller in Idle mode). The source code is much easier and requires less MCU resource.

The source codes presented in this application note are written for C8051 MCU, but can be easily adapted to any other MCU having an external interrupt input.

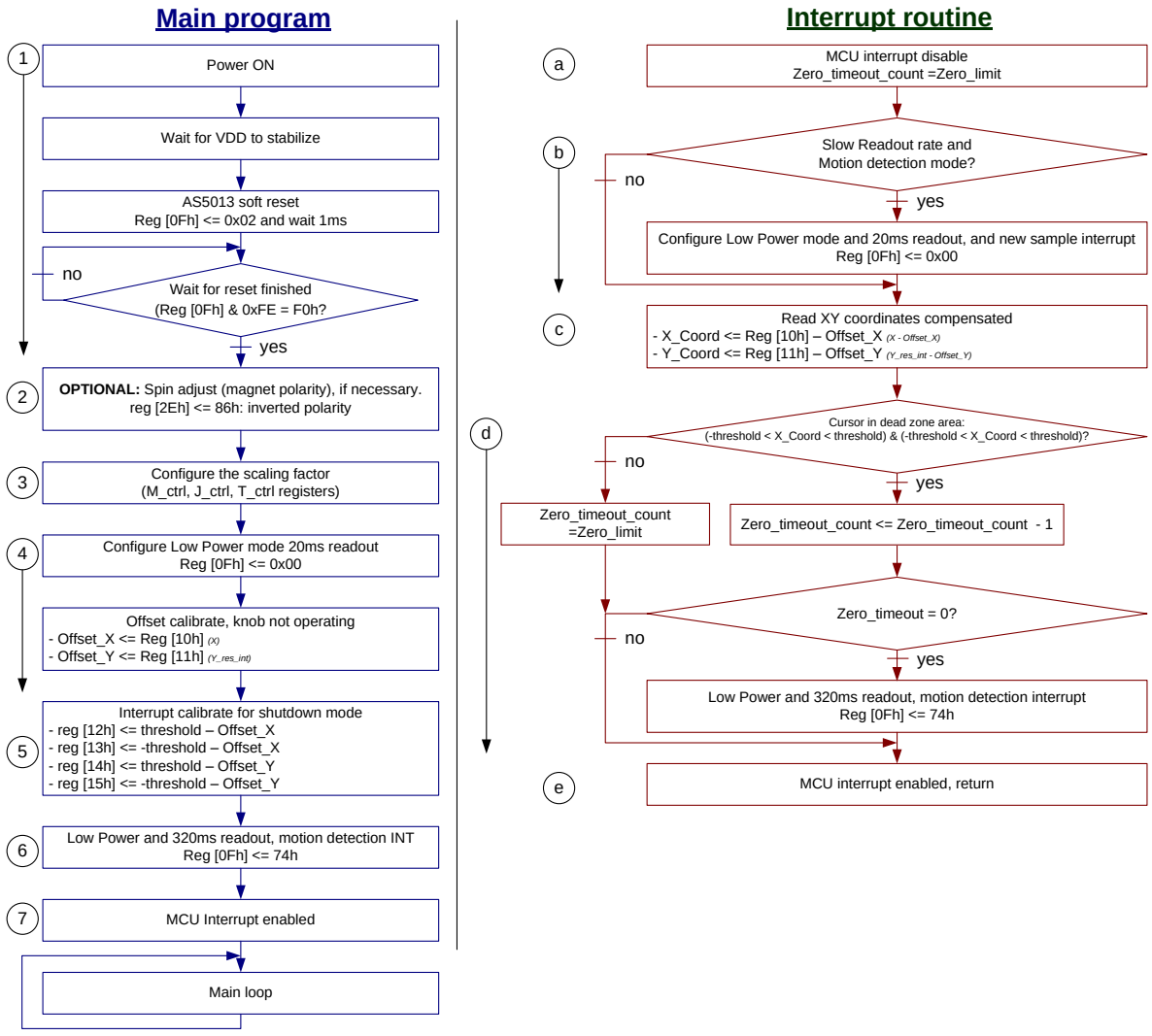
Figure 1:
EasyPointTM module connected to an MCU



2. Battery powered application (mobile devices)

For a typical battery powered application, one of the main issue is the battery life when the device is in sleep mode. The following example describes the procedure how to configure the power modes and how to read the coordinates.

Figure 2:
EasyPointTM module connected to an MCU

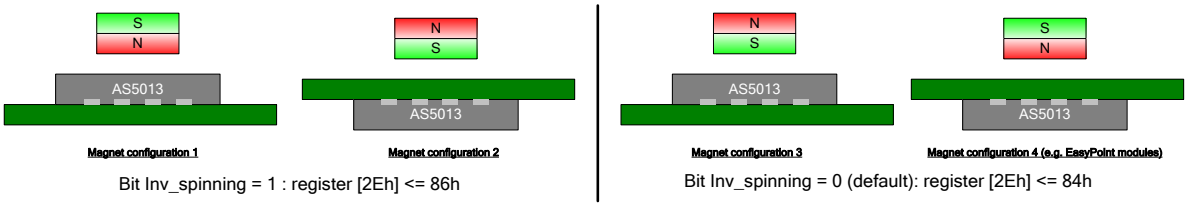


Main Program

1. After a power up of the whole system, the host must reset the AS5013 (I²C command), and wait until the encoder is ready (reset finished).

2. Configure the magnet polarity. The magnet can be placed in four different positions, as represented on Figure 3.

Figure 3:
Magnet configuration



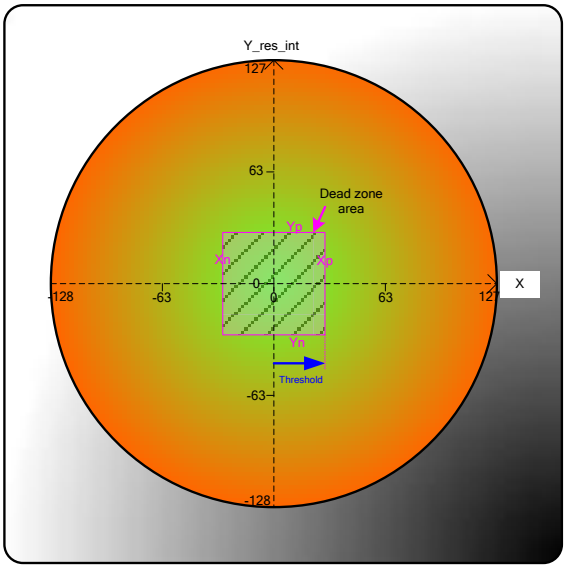
3. Configure the M_ctrl, J_ctrl, T_ctrl scaling registers, depending on the knob travel distance (0.5mm, 1mm, 2mm radius).

   N40P112 Module, 1mm knob travel: M_ctrl=0x00 (default), J_ctrl=0x06 (default), T_ctrl=0x0D.

4. Configure the AS5013 to *Low Power mode, 20ms readout rate and Interrupt for sample finished* and read the XY coordinates when the knob is in the center. At that moment the knob must not be operated. The resulting X and Y values will be used as offset values to compensate the XY coordinates in normal operation mode. This compensation is necessary due to any mechanical adjustment issue.

5. Configure the four interrupt threshold registers Xp Xn Yp Yn to determine a dead zone area. In *Low Power with Interrupt in motion detection mode*, an interrupt is generated when the magnet moves above the dead zone area. This feature can wake up a microcontroller. The values applied to those four registers are the coordinates of the borders. In that case, the offset is applied, that the area is centered on the zero position. The size of the zone defined by the variable “Threshold”, which is the threshold for the four directions, see Figure 4.

Figure 4:
Dead zone representation



6. Configure the AS5013 to Low Power mode with a slow readout rate (e.g. 320ms) and motion detection.

7. Enable the interrupt input of the MCU and go to the main loop, waiting for an interrupt from the AS5013 if the knob is moved away from the center.

Interrupt routine

a) The knob of the module has been moved above the dead zone area, an interrupt has been generated and the MCU enters into the interrupt routine.

b) Disable the interrupt of the MCU.

c) If the AS5013 is still in slow readout mode (e.g. 320ms) with motion detection interrupt, configure a faster readout rate with new sample interrupt mode in order to have a better movement precision and get an interrupt each time a new coordinate is available.

d) Read the XY coordinates [10h] and [11h] and apply the offset calculated in the main program during initialization. Those resulting XY coordinates compensated with the offset, can be now used for the main application.

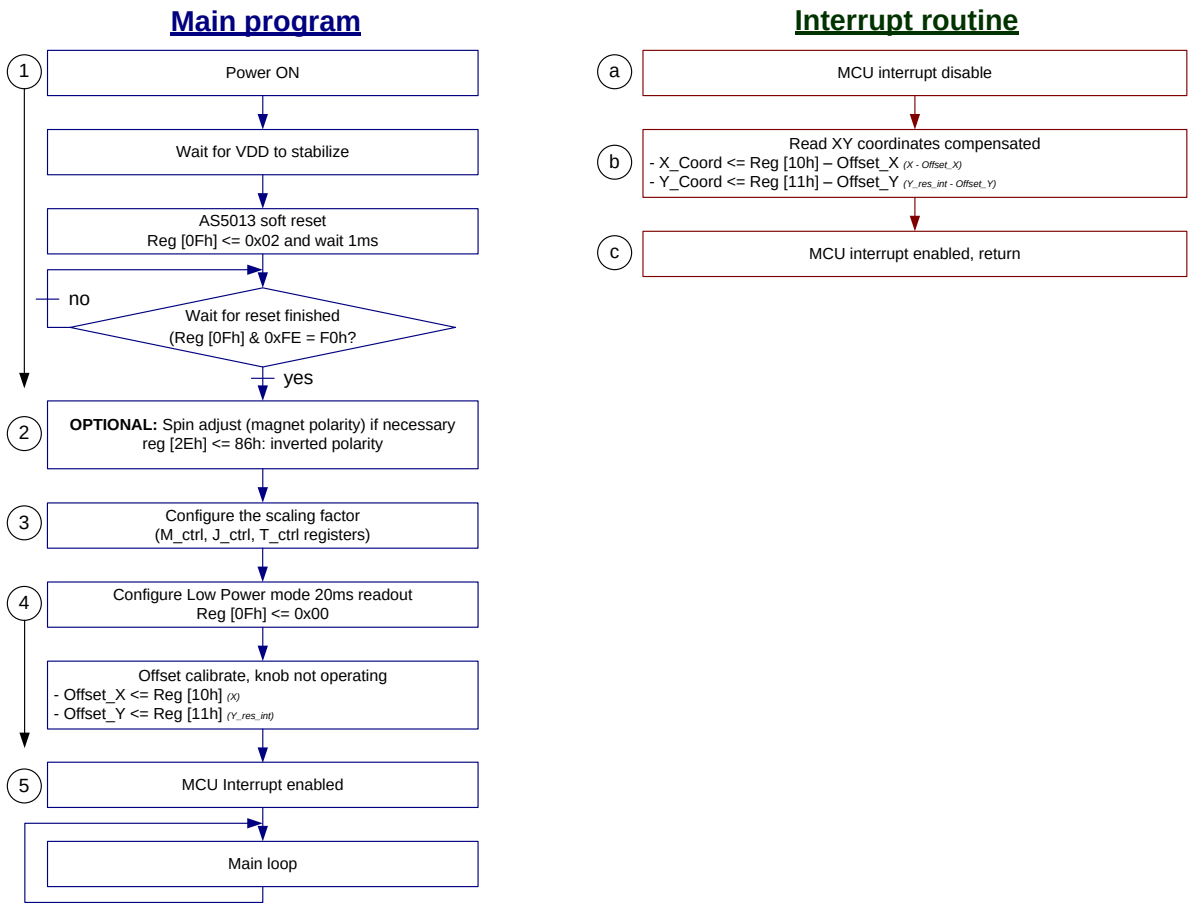
e) If the XY coordinates are in the dead zone area, a counter will increment until a value defined by the user. This value is the number of interrupts (20ms between each interrupt in that case) before the AS5013 is configured back to a slower readout rate and motion detection interrupt. That means that the knob has been released by the user a certain time ago.

f) The interrupt of the MCU should be now enabled, the interrupt routine is finished.

3. Wire powered application

For some wired application, a higher current for the device shouldn't be an issue. The AS5013 can be permanently configured with a fast readout rate (e.g. 20ms, 40ms or externally triggered when Idle bit =1), and no power switch routine is necessary anymore. The source code and resources are greatly reduced.

Figure 5:
Wired operation application firmware structure



Main Program

1. After a power up of the whole system, the host must reset the AS5013 (I²C command), and wait until the encoder is ready (reset finished).

2. Configure the magnet polarity. The magnet can be placed in four different positions, as represented on Figure 3.

3. Configure the M_ctrl, J_ctrl, T_ctrl scaling registers, depending on the knob travel distance (0.5mm, 1mm, 2mm radius).

   N40P112 Module, 1mm knob travel: M_ctrl=0x00 (default), J_ctrl=0x06 (default), T_ctrl=0x0D.

4. Configure the AS5013 to *Low Power mode, 20ms readout rate and Interrupt for sample finished* and read the XY coordinates when the knob is centered. At that moment the knob must not be operated. The resulting X and Y values will be used as offset values to compensate the XY coordinates in normal operation mode. This compensation is necessary due to mechanical misalignment.

5. Enable the interrupt input of the MCU and go to the main loop, waiting for an interrupt from the AS5013 if the knob is moved away from the center.

Interrupt Routine

The knob of the module has been moved above the dead zone area, an interrupt has been generated and the MCU enters into the interrupt routine.

a) Disable the interrupt of the MCU.

b) Read the XY coordinates and apply the offset calculated in the main program during initialization. The last register which is read must do an interrupt reset. In that case read register [11h] Y_res_int. Those resulting XY coordinates compensated with the offset, can be now used for the main application.

c) The interrupt of the MCU should be now enabled, the interrupt routine is finished.

4. Firmware source code for battery powered application

Main Program

```
void main(void)
{
        System_Init ();          // Configure MCU GPIO, I²C
        IT0 = 1;                 // External Interrupt INT0 edge triggered
        IT01CF = 0x23;           // INT0 low active, INT0 on P0.0, where AS5013 INTn is connected
        EX0 = 1;                 // INT0 active

        AS5013_init ();          // AS5013 initialization

        I2C_Write8(0x40, 0x0F, 0x00);    // Configure AS5013 to low power mode 20ms, sample finished interrupt
        Offset_Calibrate ();             // Offset calibrate, knob not operating
        Interrupt_Calibrate ();          // Interrupt thresholds Xp Xn Yp Yn
        I2C_Write8(0x40, 0x0F, 0x74);    // Configure AS5013 to low power mode (320ms), motion detect interrupt
        Power_Mode = Motion_detect;      // Power_Mode is a global variable, used as a flag in the interrupt routine

        EA = 1;      // Global Interrupt enabled

        while (1)    // Main loop
        {
                     // Main program
        }
}
```

Interrupt_Calibrate(), Wakeup interrupt calibration

Adjust Xp Xn Yp Yn as a square around the center compensated by offset_X and offset_Y. This square has a dimension of 2*center_threshold.

```
void Interrupt_Calibrate (void)
{
        EA = 0;      // Global interrupt disable
        I2C_Write8(0x40, 0x12, center_threshold - offset_X );     // Xp configuration (right)
        I2C_Write8(0x40, 0x13, -center_threshold - offset_X);     // Xn configuration (left)
        I2C_Write8(0x40, 0x14, center_threshold - offset_Y);      // Yp configuration (up)
        I2C_Write8(0x40, 0x15, -center_threshold - offset_Y);     // Yn configuration (down)
        EA = 1;      // Global interrupt enable
}
```

Interrupt routine

The following code is the main interrupt routine called by the AS5013 INT output.

```
void AS5013_interrupt (void) interrupt 0
{
        #define center_threshold 25                  // Dead zone amplitude around the centre; See Note 1 below.
        #define Zero_time_delay 50                   // Delay before shutting down the AS5013, if the knob stays in the center

        int x_coord, y_coord;                        // Final XY coordinates
        int delay_counter = Zero_time_delay;         // Zero_time_delay is a time constant, before the module goes in low readout
                                                     rate when in center: (20ms * Zero_time_delay)
        EA=0;                                        // Disable global interrupts

        // If AS5013 in Shutdown Mode (80 ms), configure it to Low Power Mode (20ms), for better time response:
        If (Power_Mode = Motion_detect)    // Power_Mode is a flag (global variable), initialized in main()
        {
```

```
                I2C_Write8(0x40, 0x0F, 0x00);       // Configure low power mode 20ms, sample finished interrupt Power_Mode =
                Sample_finish;              // Set the actual power mode flag
        }

        x_coord = (int) I2C_Read8(0x40, 0x10); // Read X position
        y_coord = (int) I2C_Read8(0x40, 0x11);  // Read Y position with interrupt reset

        // Add the X and Y offset for correct recentering. offset_X and offset_Y are global variables calculated in the function Offset_Calibrate(),
        called in main() routine.
        x_coord += offset_X;                    // Add offset_x to x_coord, then it is the final compensated X value
        y_coord += offset_Y;    // Add offset_y to y_coord, then it is the final compensated Y value

        // Clip x_coord and y_coord values into the range (-128..+127). It can happen because of the offset added.
        if (x_coord > 127)                      x_coord = 127;
        else (x_coord < -128)   x_coord = -128;

        if (y_coord > 127)                      y_coord = 127;
        else (y_coord < -128)   y_coord = -128;

        // If the knob is near the center, in center_threshold area:
        if ((x_coord >-center_threshold) && (x_coord <center_threshold) &&
           (y_coord >-center_threshold) && (y_coord <center_threshold))
        {
                delay_counter--;            // Increment the delay counter before configuring AS5013 back to Shutdown

                If (delay_counter == 0) // If the delay is over (20ms * Zero_time_delay)
                {
                        I2C_Write8(0x40, 0x0F, 0x74); // Configure low power mode (320ms), motion detect INT
                        Power_Mode = Motion_detect;     // Set the actual power mode flag
                }
                 x_coord = 0;   // If needed, the XY coordinates around the center in the dead zone can be set to 0 to…
                 y_coord = 0;                //…avoid e.g. a cursor to move by itself when the knob is released and XY not exactly 0,0
        }
        else                    // Knob not in the center, the user is moving it
{               // Reset delay counter, no need to put AS5013 to low readout rate and motion detection mode for the moment
                delay_counter = delay_counter == Zero_time_delay;                       }

        EA=1;                                   // Enable global interrupts
}
```

Note 1: The “center_threshold “ that defines the dead zone around the center depends on the mechanics that is used for positioning the magnet. Modules with higher recentering force will need smaller dead zones. The EasyPoint™ modules may have in some cases a tolerance of re-centering of 10% of its travel. It is higly recommended to choose a bigger dead zone to avoid accidential system wake-up.

5. Firmware source code for wire powered application

Main Program

```
void main(void)
{
        System_Init ();         // Configure MCU GPIO, I²C
        IT0 = 1;                // External Interrupt INT0 edge triggered
        IT01CF = 0x23;          // INT0 low active, INT0 on P0.0, where AS5013 INTn is connected
        EX0 = 1;                // INT0 active

        AS5013_init ();         // AS5013 initialization

        I2C_Write8(0x40, 0x0F, 0x00);       // Configure AS5013 to low power mode 20ms, sample finished interrupt
        Offset_Calibrate ();                // Offset calibrate, knob not operating

        EA = 1;     // Global Interrupt enabled

        while (1)   // Main loop
        {
                    // Main program
        }
}
```

Interrupt routine

The following code is the main interrupt routine called every 20ms by the AS5013 INT output.

```
void AS5013_interrupt (void) interrupt 0
{
        int x_coord, y_coord;   // Final XY coordinates

        EA=0;       // Disable global interrupt

        x_coord = (int) I2C_Read8(0x40, 0x10); // Read X position
        y_coord = (int) I2C_Read8(0x40, 0x11);  // Read Y position with interrupt reset

        // Add the X and Y offset for correct recentering. offset_X and offset_Y are global variables calculated in the function Offset_Calibrate(),
        called in main() routine.
        x_coord += offset_X;                // Add offset_x to x_coord, then it is the final compensated X value
        y_coord += offset_Y;    // Add offset_y to y_coord, then it is the final compensated Y value

        // Clip x_coord and y_coord values into the range (-128..+127). It can happen because of the offset added.
        if (x_coord > 127)                  x_coord = 127;
        else (x_coord < -128)   x_coord = -128;

        if (y_coord > 127)                  y_coord = 127;
        else (y_coord < -128)   y_coord = -128;

        EA=1;       // Enable global interrupt
}
```

6. Common Routines

AS5013_init (), initialization routine

```
void AS5013_init (void)
{
	unsigned char Reset_status = 0;

	Delay_ms(1);                                                    // Wait 1ms, to be sure the power supply is stable
	I2C_Write8(0x40, 0x0F, 0x02);                  // Reset AS5013 enabled

	Delay_ms(1);                                                    // Wait 1ms
	while (Reset_status != 0xF0)          // Check the reset sequence is finished. The register [0Fh] goes back to its
	{                                                  // default value F0h by itself)
		Reset_status = I2C_Read8(0x40, 0x0F) & 0xFE;
	}
	I2C_Write8(0x40, 0x2D, 0x0D);                  // Configure the scaling factor (resolution) of XY registers
                                                                        // 0x0D is for 1mm displacement (e.g. N40P112 modules)
                                                                        // N35P112:0x06        (142.8% Scaler)
                                                                        // N40P112:0x0D        (71.5% Scaler)
                                                                        // N50P111:0x16        (43.6% Scaler)

	// I2C_Write8(0x40, 0x2E,0x86);                                 // Optional: Invert spin (magnet polarity), if necessary, see
                                                                        // AS5013 Datasheet
}
```

Offset_Calibrate (), Zero point calibration at power up

This routine calculates offset_X and offset_Y values (global variables called in the interrupt routine) to compensate mechanical misalignment. This routine must be called one time in main(), the knob should be released.

```
void Offset_Calibrate (void)
{
	char i;
	int x_cal=0, y_cal=0;     // offset_X and offset_Y are global variables, char type

	EA = 0;                                  // Global interrupt disable

	Delay_ms(1);                                    // Wait 1ms

	I2C_Read8(0x40, 0x11);                 // Flush the actual coordinate to reset the interrupt, and get the newest XY values

	for (i=0; i<32; i++)                  // We read 32 times the coordinates and we make an average value
	{
		while (INTn);                           // Wait for INTn active, to read the next (polling here, no need of an INT vector)
		x_cal += (signed char) I2C_Read8(0x40, 0x10); // Read X position
		y_cal += (signed char) I2C_Read8(0x40, 0x11);  // Read Y position with interrupt reset
	Delay_ms(1);                            // Wait 1ms
	}
	offset_X = -(x_cal>>5);  // Divide the sum by 32 to get an average of offset_X (global variable, used in interrupt)
	offset_Y = -(y_cal>>5);  // Divide the sum by 32 to get an average of offset_Y (global variable, used in interrupt)

	EA = 1; // Global interrupt on
}
```

7. Contactless Push Button

It is possible to emulate a dome switch (push button) function with the AS5013, by reading the magnetic field from the five hall elements and applying an algorithm on them.

The value of the middle hall element C5 increases when the button is pushed vertically, i.e. the magnet gets closer and the magnetic field increases on the hall element. This simple measure can be used for detecting a push state by comparing it to a fixed threshold value, but if an external parasitic magnetic field appears near the joystick module, the magnetic field detected by C5 will increase too (offset), and can be detected as a push state.

In order to avoid any risk of unwanted “push” state, it is necessary to remove the external magnetic parasitic field. This can be done by measuring the magnetic field from the surrounding hall elements C1..C4, do an average, and subtract from the C5 value. This is the Delta value in the following source code.

When the magnet gets closer to the IC (magnet in the middle region), C5 value will increase, and C1..C4 values increase as well but with a lower amplitude. The magnetic field peak value is in the middle of the IC, over C5.

Comparing this Delta value to a fixed threshold allows detecting a push state.

The Delta amplitude between a push state and a released state is about 10 when used with an N35, N40 or N50 EasyPointTM module, having a vertical magnet displacement of 400µm.

```
void main()
{
        const short Push_threshold = 5;       // Threshold value to detect a pushed button
        if (calculate_Delta() > Push_threshold)
        {
                Push_LED = 1;           // Button is pushed
        }
        else
        {
                Push _LED = 0;          // Button is NOT pushed
        }
}

short calculate_Delta (void)
{
        short       C1, C2, C3, C4, C5,
                    C1n, C1p, C2n, C2p, C3n, C3p, C4n, C4p, C5n, C5p,
                    Delta;
        unsigned char buf[20];

        buf[0] = 0x16;                      // Start address pointer (c4_neg), and do a multibyte write
        i2c_read_write(I2C_WRITE(0x40), buf, 1);            // Write address pointer 16h
        i2c_read_write(I2C_READ(0x40), buf, 20);            // Read 20 bytes from there (C1..C5)

        C4n = (buf[0] << 8) | buf[1];
        C4p = (buf[2] << 8) | buf[3];

        C3n = (buf[4] << 8)  | buf[5];
```

```
C3p = (buf[6] << 8)  | buf[7];

C2n = (buf[8] << 8) | buf[9];
C2p = (buf[10] << 8)  | buf[11];

C1n = (buf[12] << 8) | buf[13];
C1p = (buf[14] << 8) | buf[15];

C5n = (buf[16] << 8) | buf[17];
C5p = (buf[18] << 8) | buf[19];

C1 = (C1p-C1n)>>5;      C2 = (C2p-C2n)>>5;
C3 = (C3p-C3n)>>5;      C4 = (C4p-C4n)>>5;
C5 = (C5p-C5n)>>5;

// Read middle hall sensor (C5), remove the surrounding magnetic fields (C1, C2, C3, C4) for better immunity to external magnetic fields

Delta = C5 - ((C1 + C2 + C3 + C4)>>2);
if (delta < 0) delta = 0;    // Limit delta to positive values only (security)

return delta;
}
```

8. Ordering Information

Table 1:
Ordering Information

Ordering Code	Description	comments
AS5013-DB-2	AS5013 Demo Kit	AS5013 Demo board in gamepad-shape

Copyright

Disclaimer

Devices sold by ams AG are covered by the warranty and patent indemnification provisions appearing in its Term of Sale. ams AG makes no warranty, express, statutory, implied, or by description regarding the information set forth herein or regarding the freedom of the described devices from patent infringement. ams AG reserves the right to change specifications and prices at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with ams AG for current information.

This product is intended for use in normal commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or life sustaining equipment are specifically not recommended without additional processing by ams AG for each application. For shipments of less than 100 parts the manufacturing flow might show deviations from the standard production flow, such as test flow or test location.

The information furnished here by ams AG is believed to be correct and accurate. However, ams AG shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interruption of business or indirect, special, incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of ams AG rendering of technical or other services.

Contact Information

Headquarters

ams AG
Tobelbader Strasse 30
8141 Unterpremstaetten
Austria
T. +43 (0) 3136 500 0
For Sales Offices, Distributors and Representatives, please visit:
http://www.ams.com/contact