ADNS-5030

High Resolution Performance Optimization

Application Note 5300

Introduction

The Avago Technologies ADNS-5030 small form factor optical mouse sensor is designed for optimum navigation performance in cordless mouse applications. This entry level mouse sensor is capable of high-speed motion detection – up to 14 ips velocity and 2 G acceleration. Frame rate and power saving features are controlled internally to optimize surface tracking and battery life performance.

Unlike previous low power sensors, the ADNS-5030 is the first compact optical mouse sensor to feature an enhanced resolution of up to 1000 counts per inch (cpi). At this setting, users may find it difficult to control mouse motion at low speeds because this setting has greater movement sensitivity. In other words, you can see your PC cursor move dramatically even with the slightest nudge.

To overcome such movements, this application note recommends a switching method that improves ADNS-5030 performance, especially during low speeds at the highest resolution setting.

The Switching Method

The switching method is a set of algorithm calculations that changes the resolution of the optical mouse based on its motion speed. As the sensor's resolution is enhanced up to 1000 cpi, most mouse manufacturers will design their mouse to be set at the maximum. However, at low speeds, when users try to draw something or even locate the mouse cursor, a lower sensitivity is preferred.

It is recommended that there should be a motion threshold to determine whether a low or high resolution should be used. Avago Technologies recommends that the resolution should be updated frequently as the motion evaluation is done to improve user experience. Table 1 shows an example of the motion threshold for this implementation. Users will need to determine the number of pixels (as the threshold) since this depends very much on each motion evaluation rate[1].

Example 1 (Table 1) is with single stage switching at 2 ips.

Table 1. Motion threshold table to determine resolution set for Example 1

Speed (ips)	Operation	Effective Resolution (cpi)
<2.0	Write 0x00 to Address 0x0d	500
>2.0	Write 0x01 to Address 0x0d	1000

There is also another alternative way to enhance the user experience. It involves two or more stages of resolution change, as shown in Example 2 (Table 2):

Table 2. Motion threshold table to determine resolution set for Example 2

Speed (ips)	**Operation**	**Effective Resolution (cpi)**
< 1.5	Write 0x00 to Address 0x0d	500
1.5 - 2.0	Write 0x00 to Address 0x0d	500 * 3/2 = 750
> 2.0	Write 0x01 to Address 0x0d	1000

As mentioned earlier, there is an algorithm that can be used to determine the motion threshold. This is known as Common Resolution Estimation (C-RES).

The following discussion is based on Example 1 only.

Note 1. Motion evaluation rate refers to the frequency of checking whether the accumulated pixel movements have exceeded the threshold or not.

Common Resolution Estimation (C-RES)

The Common Resolution Estimation (C-RES) takes into consideration the total absolute values of delta-x and delta-y since the last movement. In this implementation algorithm, the values in delta-x and delta-y (pixels) are summed up to estimate the current mouse speed. By using this set of algorithms, a common resolution setting is used for both x and y movements. It is recommended that the motion evaluation rate is smaller than the USB polling rate. This is to ensure a smoother resolution transition, enabling more fluid cursor tracking. The next section describes how Z1 is determined.

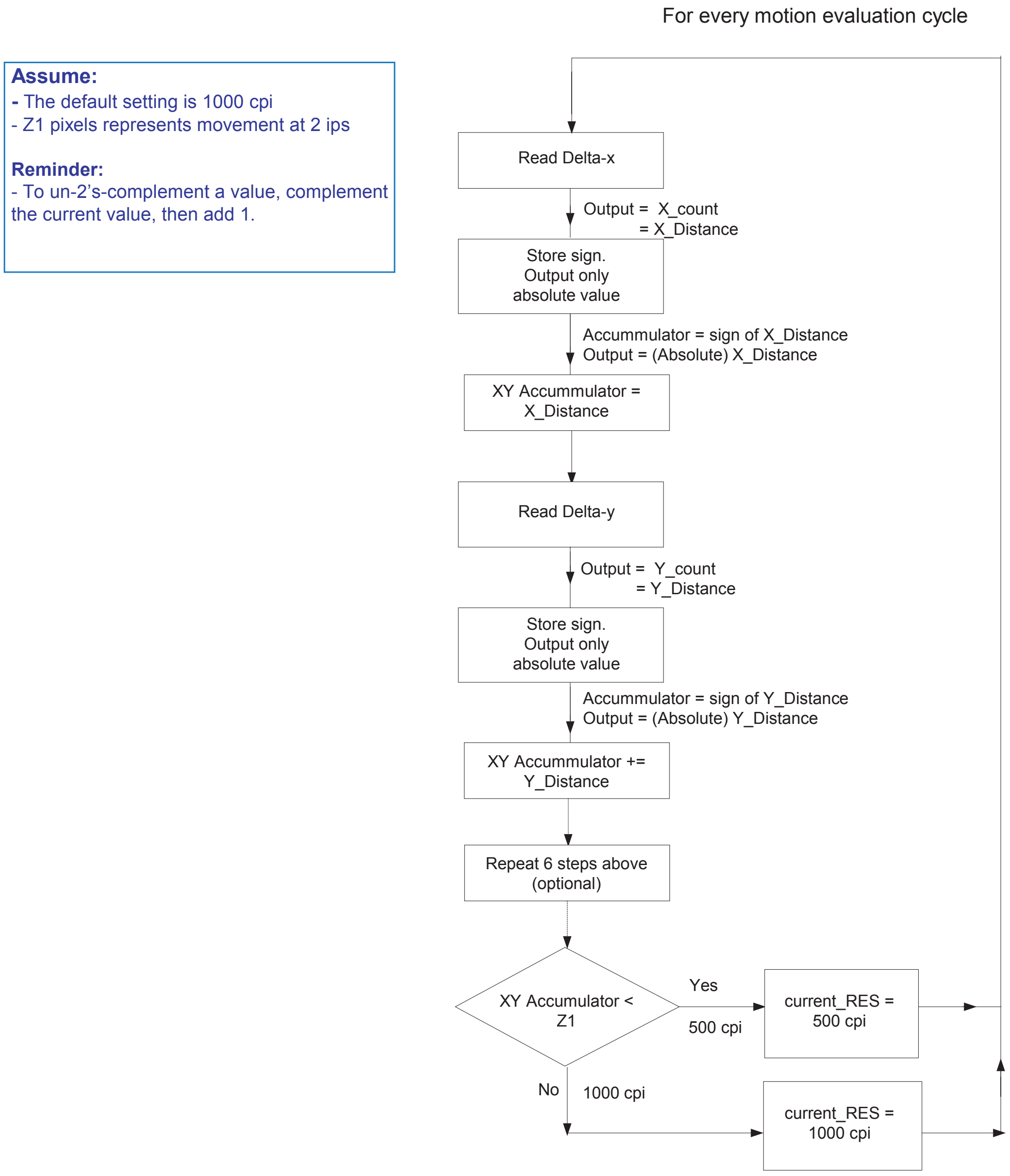
For every motion evaluation cycle
Assume:
- The default setting is 1000 cpi
- Z1 pixels represents movement at 2 ips
Reminder:
- To un-2's-complement a value, complement the current value, then add 1.
Read Delta-x
Output = X_count = X_Distance
Store sign. Output only absolute value
Accummulator = sign of X_Distance
Output = (Absolute) X_Distance
XY Accummulator = X_Distance
Read Delta-y
Output = Y_count = Y_Distance
Store sign. Output only absolute value
Accummulator = sign of Y_Distance
Output = (Absolute) Y_Distance
XY Accummulator += Y_Distance
Repeat 6 steps above (optional)
XY Accumulator < Z1
Yes
500 cpi
current_RES = 500 cpi
No
1000 cpi
current_RES = 1000 cpi

Figure 1. Common Resolution Estimation (C-RES) implementation flowchart

Guidelines to Determine Z1

To determine the Z1 value, the motion evaluation rate needs to be used. Avago Technologies recommends the evaluation rate should be done 2-4 times more frequently than the USB polling rate. This will avoid drastic resolution changes which may annoy mouse users. Also, the main advantage of this is to have the USB report movements with an "averaged" resolution value, especially when the user moves the mice repeatedly below and over the speed threshold.

Here are formulas to use as a reference when implementing this technique:

Pixel movement rate (counts/sec) = **resolution** (counts/inches) * **movement speed** (inches/sec)

Assuming that the speed threshold to switch to 1000 counts/inch is 2 inches/sec:

Pixel movement rate = 2000 counts/sec

Here is where we need to take into consideration the motion evaluation rate:

If the motion evaluation rate = 2 ms

Pixel movement at one direction, Z0 = (2000 * 2)/1000 = 4 counts

If the motion evaluation rate = 3 ms

Pixel movement at one direction, Z0 = (2000 * 3)/1000 = 6 counts

If the motion evaluation rate = 4 ms

Pixel movement at one direction, Z0 = (2000 * 4)/1000 = 8 counts

Since the list of formulas is just a guide, it is recommended that users customize the firmware design (especially the evaluation rate and speed threshold) according to their preference and application.

Conclusion

The suggested switching method is intended to help mouse manufacturers and designers overcome the over-sensitivity of the mouse cursor while operating with the maximum resolution setting. However, there may be more alternatives in terms of the speed threshold selection and number of switching stages which designers can adjust to optimize user experience. It is recommended that a microcontroller function be used to enable or disable the algorithm easily.

Appendix

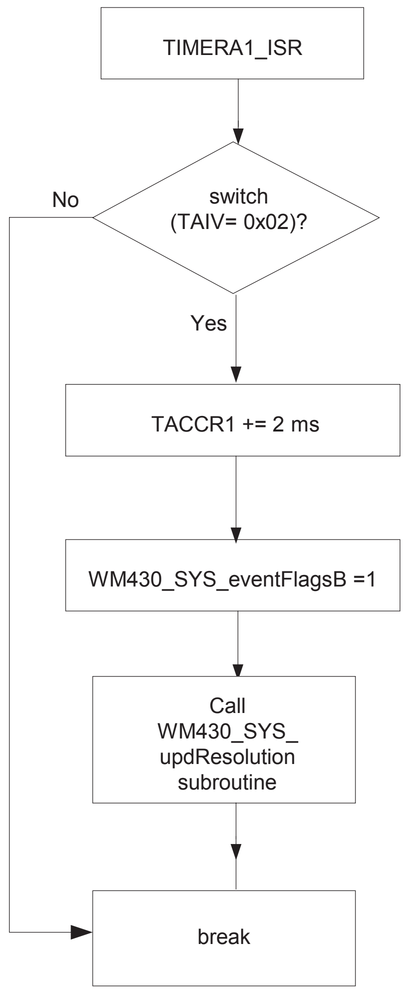

Figure 2. Example of implementation in ADNK-5033-TN24 – TIMERA1_ISR in "wm430_system.c"

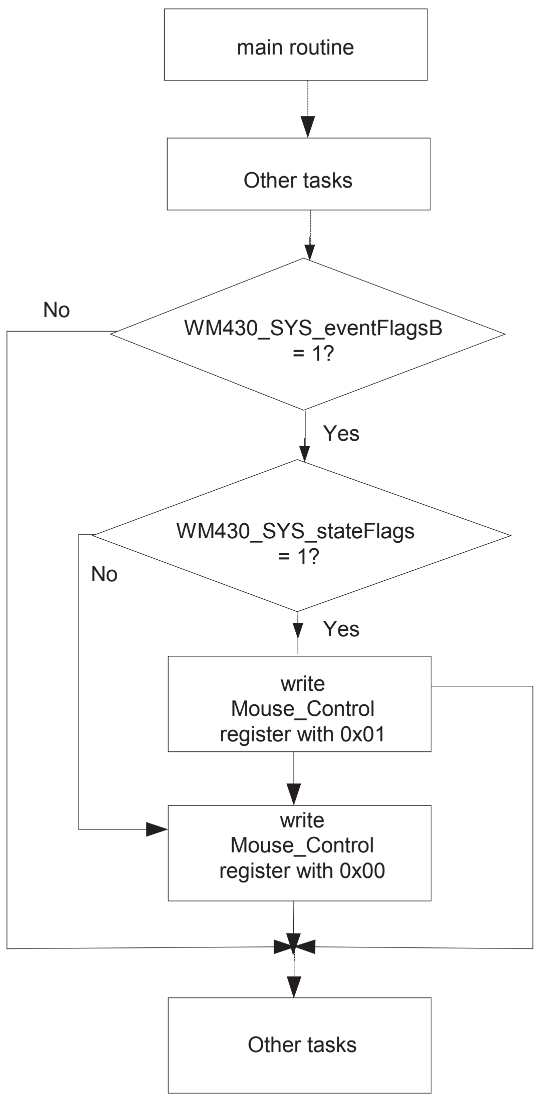
main routine
Other tasks
WM430_SYS_eventFlagsB = 1?
No
Yes
WM430_SYS_stateFlags = 1?
No
Yes
write Mouse_Control register with 0x01
write Mouse_Control register with 0x00
Other tasks

Figure 3. Example of implementation in ADNK-5033-TN24 – Main routine in "wm430_system.c"

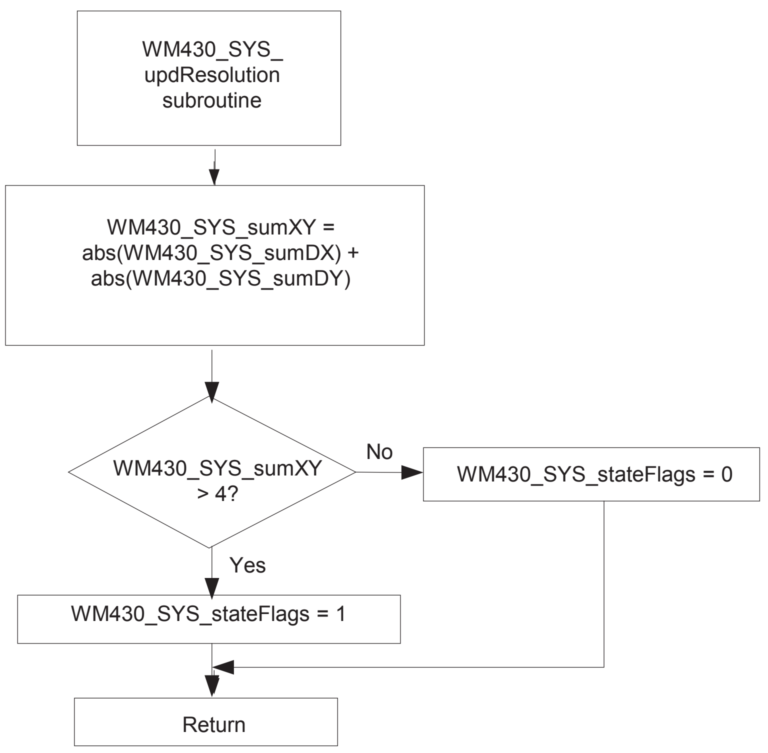

Figure 4. Example of implementation in ADNK-5033-TN24 – WM430_SYS_updResolution subroutine in "wm430_system.c"

File: “wm430_system.c”

```
==================================================================================
__interrupt void TIMERA1_ISR(void);
// Usage:      Executed whenever a Timer_A1 interrupt is generated
// Parameters:  none
// Returns:     nothing
==================================================================================
#pragma vector=TIMERA1_VECTOR
__interrupt void TIMERA1_ISR(void)
{
  switch(TAIV)
  {
    case 0x02:  // TACCR1 CCIFG
      TACCR1 += TIMERA1_SMCLK_002MSEC;      // Set up next periodic interrupt for TACCR1
      WM430_SYS_eventFlags |= EVENT_TACCR1INT;  // Signal the need for processing events at TACCR1 periods
      WM430_SYS_eventFlagsB |= EVENT_RESOLUTIONUPD; // Check and update resolution
      WM430_SYS_updResolution();  // Update sensor resolution
      break;
…
…
  }
}
```

File: “wm430_system.c”

```
==================================================================================
Void main (void);
// Usage:      main routine which executes all mouse buttons and sensor related operation
// Parameters:  none
// Returns:     nothing
==================================================================================
void main(void)
{
…
…
      // Common Resolution Estimation(C-RES) implementation
      if(WM430_SYS_eventFlagsB & EVENT_RESOLUTIONUPD)   // Check and update resolution
      {
        if(WM430_SYS_stateFlags & STATE_RES1000CPI)
        {
             SENSOR_writeRegister(ADNS5030_MOUSECTRL_ADDR, ADNS5030_RES1000CPI);  // Configuration register data
(1000CPI)
        }
        else
        {
              SENSOR_writeRegister(ADNS5030_MOUSECTRL_ADDR, ADNS5030_RES500CPI);    // Configuration register data
(500CPI)
        }
        WM430_SYS_eventFlagsB &= ~EVENT_RESOLUTIONUPD;  // Clear update resolution flag
      }
…
…
}
```

File: “wm430_system.c”

```
===========================================================================
void WM430_SYS_updResolution(void);
// Usage:       Update sensor resolution for performance optimization
// Parameters:  none
// Returns:     nothing
===========================================================================
void WM430_SYS_updResolution(void)
{
    WM430_SYS_sumXY = abs(WM430_SYS_sumDX);
    WM430_SYS_sumXY += abs(WM430_SYS_sumDY);

    if(WM430_SYS_sumXY > ADNS5030_PIXELMOV_THRESHOLD)
    {
      WM430_SYS_stateFlags |= STATE_RES1000CPI;   // 1000cpi
    }
    else
    {
      WM430_SYS_stateFlags &= ~STATE_RES1000CPI;  // 500cpi
    }
}
```