APPLICATION NOTE 5030

Designing a Proximity Sensor into a Handheld Touch-Screen Application

By: Ilya Veygman, Strategic Applications Engineer
Nov 30, 2011

Abstract: This application note discusses the advantages of using the Maxim MAX44000 proximity sensors in handheld touch-screen applications. It then addresses a few application considerations. These include reducing crosstalk, improving noise immunity, and reducing the load on the application processor in the system. Sample C code for threshold hysteresis is available in the appendix.

Why Use Proximity Sensors?

Touch screens can be found in various handheld electronic devices, not the least of which are smartphones. While touch screens greatly improve a device’s functionality, they present several new challenges. This includes knowing how and when to interpret interaction with the touch screen. For example, the screen must interpret how to react when the phone is against a user’s face; inadvertent contact with the ear or cheek can be wrongly interpreted as desired user input.

The most popular way to get around this problem—and simultaneously add to the device’s capabilities—is to integrate a proximity sensor into the phone. This sensor can turn off the touch screen when the proximity reading is sufficiently large and the user is on the phone.

Devices containing infrared-based proximity sensors with digital outputs, such as the MAX44000, make this task relatively simple, compared to discrete solutions.

Key Benefits of Maxim's Proximity Sensors

Maxim’s proximity sensors offer several advantages. For one, the infrared emitter is set up in a current-sinking rather than a current-sourcing configuration. Thus the user can select, within reason, the supply voltage used on the LED, allowing for optimization between LED performance and power usage (Figure 1).

Because the MAX44000 family is I²C-capable, it is extremely easy to integrate the sensors into most embedded systems via this versatile bus. In addition, the devices support a hardware interrupt line. These two features ensure that the sensor can be seamlessly placed into most handheld devices, while minimizing the amount of processor resources needed to service information from the sensor.

Furthermore, Maxim’s proximity sensors have additional functionality built in. For example, the MAX44000 integrates an ambient light sensor and proximity into a single 6-pin IC. Solutions such as these eliminate the need for multiple sensors to accomplish all
light-sensor tasks.

Application Considerations

The MAX44000 comes in a tiny, 2mm x 2mm x 0.6mm UDFN-Opto package, allowing the user to save room in space-sensitive applications. Additionally, the sensor provides circuitry to drive an LED, but the end user must ultimately supply the emitter diode. This circuit can drive from 0mA to 110mA through the emitter diode in a sinking configuration, which saves the need for external circuitry to perform this task.

![Figure 2. The MAX44000 with a bypassed emitter.](image)

Care should be taken with this feature, especially at higher drive currents. Specifically, these short, high-current pulses may generate some noise around the MAX44000, due to current spikes in the power supply line. There are two ways to get around this: either use a decoupling capacitor with the emitter diode or completely isolate the power supply of the MAX44000 from that of the emitter diode. The decoupling capacitor has the advantage of being cheaper, but the disadvantage of having to be located very close to the MAX44000 and emitter diode. The end user should try this solution first before moving on to the alternative, as it generally works sufficiently well. Figure 2 shows both solutions simultaneously, though this may prove to be overkill.

One must also take into careful consideration the effects of glass on the proximity sensor. Most, if not all, smartphones have glass covering the front of the phone. In some phones, this glass is black. This glass has two key effects on the light sensor. First, the intensity of incident light onto the ambient light sensor section of the IC is attenuated, which should be taken into account. Second, some crosstalk may be introduced, by way of emitted light from the LED being partially reflected from the glass and back to the sensor (Figure 3).

![Figure 3. Illustration of crosstalk—no baffle.](image)

This can be mitigated in several ways. One option is to place an optical baffle between the emitter and receiver (Figure 4), which will greatly reduce the amount of optical power that can reach the sensor. Another is to keep both the emitter and receiver as close to the glass as possible, and to ensure that the circuit board is nonreflective.

![Figure 4. Example of a simple optical baffle.](image)

Considerations with the On/Off Threshold
A common challenge with integrating the proximity sensor into the system is correctly selecting the proximity threshold at which the screen is turned on or off during a call. The "trigger point" must be set to ensure that it will not have many false positives while maximizing the number of use cases that it can support. For example, users with light-colored hair who hold the phone in such a way that the proximity sensor faces their hair will reflect far more signal than a user with dark-colored hair.

The MAX44000’s proximity sensor has excellent sensitivity (2.7nW/cm²/LSB) for a standard, 850nm IR emitter. This means the MAX44000 can not only perform well behind black glass but also detect users with dark hair. In addition to this, the MAX44000’s proximity sensor rejects up to 100,000 lux of DC sunlight, improving its performance in outdoor conditions.

One last thing to consider is the application of hysteresis to this sensor. The reason one would want to do this is the same reason one would add hysteresis to a comparator circuit. Namely, if the input signal is hovering around the threshold, any noise can cause undesired, random switching of the output signal. The same applies to a proximity sensor.

Consider the simple (and power-inefficient) case where we periodically poll the sensor in software. If the counts are above some value and the screen is on, then the screen is turned off. Otherwise, if the screen was already off, then it is turned back on. This might seem acceptable at first glance, but the user may hold the device in such a way that the counts will hover around this threshold, causing the screen to turn on and off undesirably.

One way around this is to set the hysteresis in software. For example, if the transition of the touch screen from "on" to "off" occurs at or above 150 counts (assuming the proximity sensor is being run in 8-bit mode), then the transition from "off" back to "on" should occur only after the counts fall to or below 135. In addition, it is useful to ensure that these respective levels persist for a certain amount of time. This further reduces the chances of noise triggering undesired behavior by acting as a sort of crude lowpass filter.

The MAX44000 supports such a scheme in its internal registers:

<table>
<thead>
<tr>
<th>REGISTER</th>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
<th>REGISTER ADDRESS</th>
<th>POWER-ON RESET STATE</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold Persist Timer</td>
<td>PRXPST[1:0]</td>
<td>ALSPST[1:0]</td>
<td>0x0A</td>
<td>0x00</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROX Threshold —High Byte</td>
<td>ABOVE</td>
<td>PRXTHR[13:8]</td>
<td>0x0B</td>
<td>0x00</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROX Threshold —Low Byte</td>
<td>PRXTHR[7:0]</td>
<td>0x0C</td>
<td>0x00</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If interrupts are enabled (register 0x01, bits 1:0 for proximity and ALS), then these registers can set the chip so it can run without the need to constantly poll the sensor over I²C. Use bits 2 and 3 from register 0x0A to set the delay before an interrupt, as described earlier. This delay can be 1, 4, 8, or 16 consecutive samples past the threshold. Registers 0x0B and 0x0C set the threshold, as well as whether the interrupt is triggered by the counts being above or below the threshold.

Some sample code that implements the hysteresis scheme and this interrupt approach is available in the appendix. When doing reads or writes over the I²C bus, it should be noted that this particular part does not autoincrement register addresses in the case of multiple read/write. This should be done manually in software, although this is generally a trivial implementation. It is critical that care is taken when reading data across multiple registers in an I²C-compatible device, so that errors do not occur. For more information, see application note 5033, "The Importance of Being Earnest (About Reading from an ADC on the I²C Interface.)"

Appendix A: Sample Code for Threshold Hysteresis

```c
#define MAX44000_ADDR 0x94
#define INT_STATUS_REG 0x00
#define OFF_THRESHOLD 4600
#define OFF_DELAY 1
#define ON_THRESHOLD 4000
#define ON_DELAY 3
```
uint8 screenStatus;  // 0 means off, 1 means on

/*
 * i2cWriteBytes()
 * Arguments:
 * uint8 address - device address
 * uint8 start_reg - register where the first byte is written
 * uint8 *data - data to write
 * uint8 nbytes - number of bytes to write
 * Consecutively writes several bytes to some i2c device starting at some
 * specified address -- implemented elsewhere
 */
void i2cWriteBytes(uint8 address,uint8 start_reg,uint8 *data,uint8 nbytes);

/*
 * MAX44000InterruptHandler()
 * The example code below implements an interrupt handler when the INT pin of the MAX44000, assuming that the MAX44000's
 * proximity sensor is set up for 14-bit mode and interrupts have been enabled. Furthermore, assume that screenStatus was
 * initialized at some point to 1 or 0. Please consult the data sheet's register descriptions for more details.
 */
void MAX44000InterruptHandler() {
    uint8 i2cData[3];
    i2cRead1Byte(MAX44000_ADDR,INT_STATUS_REG,&i2cData);
    if (i2cData&0x01 != 0)
        return; // check to make sure interrupt really fired
        // this simultaneously clears the interrupt flag
    if (screenStatus) {
        i2cData[0] = ON_DELAY;
        i2cData[1] = ON_THRESHOLD >> 8 & 0xBF; // set ABOVE = 0
        i2cData[2] = ON_THRESHOLD & 0xFF;
    } else {
        i2cData[0] = OFF_DELAY;
        i2cData[1] = OFF_THRESHOLD >> 8 | 0x40; // set ABOVE = 1
        i2cData[2] = OFF_THRESHOLD & 0xFF;
    }
    // set the new threshold depending on what the screen status was
    // set the delay and threshold after each interrupt
    i2cWriteBytes(MAX44000_ADDR,0x0A,i2cData,3);
    return;
} // MAX44000InterruptHandler

Related Parts

| MAX44000 | Ambient and Infrared Proximity Sensor | Free Samples |

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 5030: http://www.maximintegrated.com/an5030
APPLICATION NOTE 5030, AN5030, AN 5030, APP5030, Appnote5030, Appnote 5030
Copyright © by Maxim Integrated Products
Additional Legal Notices: http://www.maximintegrated.com/legal