Seeing (Infra)Red

by Tom Cantrell

A new Vishay smart IR proximity sensor eases integration into a wide range of designs. The low-cost evaluation kit makes it even easier.

Thanks to Moore's Law, electronic gadgets continue to boost their IQs. With ever more MIPS and megabytes of software under the hood, designers have unprecedented processing power on tap. But that's only part of the answer when it comes to adding compelling new product features. The ability to crunch data is good, but only if we can capture interesting data to crunch in the first place.

For instance, these days touch sensing is all the rage. So what do designers do for an encore? One obvious embellishment is to give devices the ability to sense proximity. Enter the Vishay VCNL4000 IR (infrared) proximity sensor (see Photo 1).

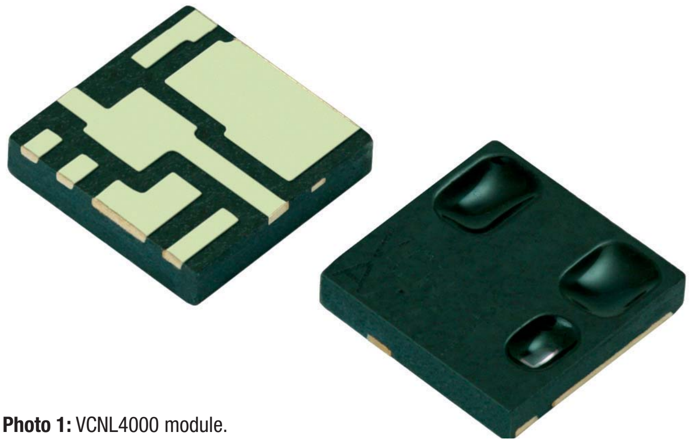


Photo 1: VCNL4000 module.

The idea of using IR to sense proximity isn't new and the concept is easy to understand. Just use an IR LED to illuminate the area of interest and then measure the amount of reflected light with an IR photodiode. The higher the reflection, the closer (and/or larger, and/or more reflective) an object is.

Going one step further, the VCNL4000 also integrates a visible wavelength photodiode for ambient light sensing (see Figure 1). The combination of IR proximity detection and visible light sensing makes the module a natural for handheld gadgets. For example, in a cell phone, the visible light detector could be used to automatically adjust the display's backlight level. Meanwhile, the IR proximity feature could be used to detect that the phone is being held to the user's ear, at which point the display could be completely turned off to conserve power.

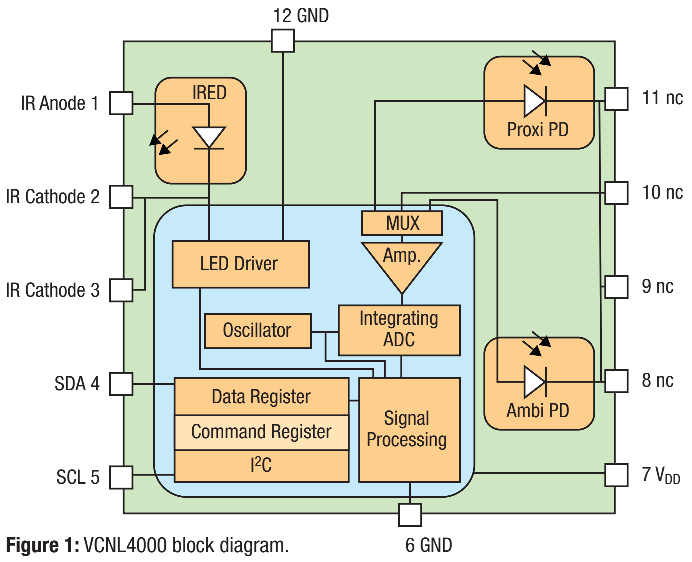


Figure 1: VCNL4000 block diagram.

Roll your own?

Sure you can. All it takes is an MCU, an IR LED, an IR photodiode, a visible light photodiode and a bit of software, right? It sounds easy, but as usual, the devil is in the details. In fact, the embellishments required to achieve a truly accurate and robust implementation are non-trivial and can easily nickel and dime a seemingly simple solution to death.

An easy way to see this is to take a look at the VCNL4000's features and consider what it would take for a back-of-the-napkin design to match them.

For instance, you need a couple of A/D channels to measure the photodiode outputs, right? No biggie, since even the bluest of blue-collar MCUs comes with an ADC these days. The only problem is that the ADC on a typical MCU can't match the 16-bit resolution of the Vishay part. Oh well, I guess you need to add a high-resolution ADC chip. Don't forget some op-amps to get the most out of the photodiodes. While you're at it, throw in a transistor to drive the IR LED at up to 200 mA, far beyond the drive capability of an MCU GPIO pin.

But that's just the start. A key feature of the VCNL4000 is that it modulates the IR LED at high frequency, up to 3.125 MHz, and bandpass filters the raw IR return. Focusing purely on the reflection helps cancel the influence of ambient and interfering IR sources, such as fluorescent light ballasts. Now your roll-your-own design needs some real help, maybe a PLD and a filter chip? Or step on up to a much bigger-ticket MCU/DSP.

While the IR proximity feature gets the headlines, the VCNL4000 visible light sensor is no slouch. Notably, it includes the ability to automatically take an average of up to 128 light readings. This helps filter out external and internal interference—such as 60/120 Hz interference and converter noise floor interference, respectively—to deliver a more sensitive and accurate measurement both night and day from 0.2 to 13,000 lux.

Power tripping

Early on in your roll-your-own musings, it might have seemed that a mini-me MCU could not only easily handle the job, but would have plenty of time to kick back and take a nap between readings. By now, you can see that you need beefier silicon, and it's going to be pretty busy with all the high-frequency stuff going on, such as modulation, filtering, averaging and bit-banging. For instance, matching the VCNL4000 ambient light averaging capability calls for up to 128 A/D conversions, or in the case of a digital light sensor, like an I^{2}C, thousands of clock edges. Not to mention all the associated CPU cycles. That means power consumption is going up, so toss in some spare change for a bigger battery.

You might think that, when talking about driving the LED with up to 200 mA, any solution is going to be a power hog. But, in fact, the VCNL4000's power consumption is surprisingly low due to the sparse duty cycle of LED illumination and optimized signal processing.

Let's say you want to take ten proximity readings per second with 100 mA LED drive current. As you can see in Figure 2, during each 100 ms sample time, the LED is only illuminated for some 70 µs. As a result, the total average power consumption of the LED and VCNL4000 is under 100 µA. The ambient light sensor is similarly green. For example, at ten samples per second with each sample comprising an average of eight readings, average power consumption is once again under 100 µA.

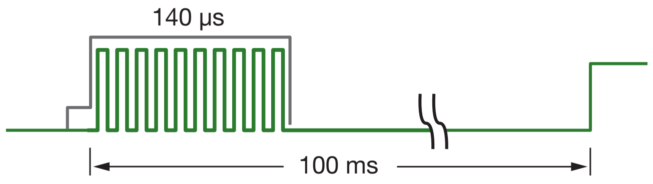


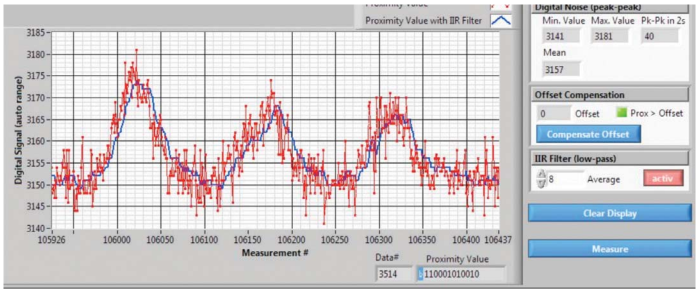
Figure 2: Proximity measurement timing.

I'm a big fan of the latest and greatest MCUs and fully respect the ability of clever designers to wring the most out of them. Without actually going through a design, I can't completely rule out the possibility that a multi-part lash-up might come close to matching the capabilities of the VCNL4000. But don't forget the optical aspects. With a roll-your-own design, you'll need to come up with an optically isolated packaging scheme that avoids internal crosstalk, such as reflections inside the box. No matter how clever a discrete design is, it will surely consume more board space than the super-tiny and thin Vishay module at 3.95 mm x 3.95 mm x 0.75 mm.

Vishay makes it easy to kick the tires with a low-cost evaluation kit (see Photo 2) comprising a VCNL4000 carrier board, a USB interface, and Windows-based dashboard software. The carrier board has a 10-pin header that plugs into the USB adapter, which plugs into your PC.

The dashboard software (see Photo 3) features real-time, strip-chart plotting of the sensor output with the ability to point-and-click configure various options such as LED current, sample frequency, offset compensation, and filtering, as well as read and write the VCNL4000 registers directly.

Photo 2: VCNL4000 evaluation kit.



Digital Noise (peak-peak)
Min. Value 3141
Max. Value 3181
Pk-Pk in 2s 40
Mean 3157
Offset Compensation
0 Offset
🟩 Prox > Offset
IIR Filter (low-pass)
8 Average
activ
Data# 3514
Proximity Value 110001010010

Photo 3: VCNL4000 evaluation kit software screen shot.

Light show

The Vishay approach of splitting the VCNL4000 carrier board and the USB interface is handy because it allows you to use the carrier board by itself for prototyping and experiments.

Connecting your favorite MCU to the VCNL4000 requires just five connections: the two-wire I^{2}C interface (SCL, SDA), ground, power for the logic (V_{DD}), and a separate power supply for the LED (IR Anode). While the power supplies are specified as 2.5 V to 3.6 V, the I^{2}C bus can handle 1.7 V to 5.5 V for connection with virtually any MCU.

Note that the evaluation kit powers both the logic and the LED from a single power supply. In your own application, make sure the high LED current pulses (up to 200 mA) don't glitch the power for the rest of your logic. If it's an issue you could, for example, power the LED directly from a battery and your digital logic from a regulated supply.

I was anxious to wire the gadget up, but the module's fine-pitch, 0.05-in. connector had me scratching my head. I didn't want to solder wires directly to the module, so I did the right thing and ordered a matching socket. Being no brain surgeon, I was pretty proud of myself when I was able to tack wires to the socket. But the applause didn't last long. As soon as I tried to plug in the Vishay module, the pins on the socket fell right out.

Eventually I was able, barely, to get four clip leads—SDA, SCL, GND, and combined LED and logic power—connected to the VCNL4000 module, making for a bit of a cumbersome lash-up (see Photo 4). The folks at Vishay might consider tossing in an adapter or a breakout board with more room for connections.

Once I got the clip leads to stay put, it was easy enough to bring the connections over to a Parallax Inc. SBC (single-board computer) based on that company's novel Propeller multicore MCU. It's a setup that's especially useful for experimenting and prototyping, since the edit-compile-download cycle takes just seconds and the company hosts a large and ever-growing library of user-contributed software objects. For example, instead of having to write an I^{2}C driver, I just downloaded a "Basic_I^{2}C_Driver" from the Parallax website.

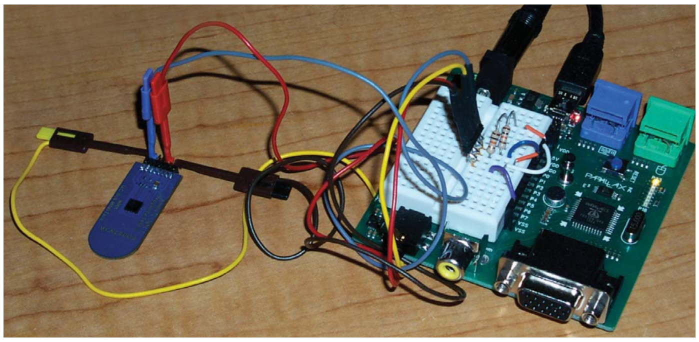
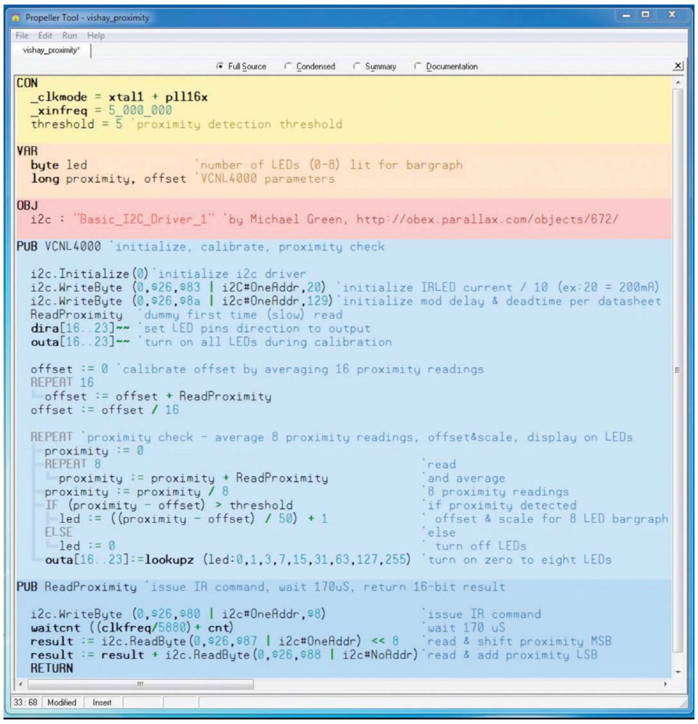


Photo 4: VCNL4000 connected to Parallax SBC.

Thanks to the module's built-in averaging and offset compensation, taking an ambient light reading is trivially easy. First, you configure the part with your chosen options, including the number and timing of samples to average and whether or not to perform automatic offset compensation. After that it's a simple matter of issuing the command and then spinning on a busy bit, or inserting a software delay, until the operation completes some 1 to 100 ms later, depending on the number and timing of samples being averaged. Now you can read the result where each count, or LSB (least significant byte), corresponds to 0.2 lux (to get an absolute lux reading, divide the offset-compensated 16-bit result by 5). Just remember that the level of illumination perceived by the human eye for a given lux level varies a bit depending on the nature of the light source, such as incandescent, fluorescent, or sunlight.

The IR proximity feature takes a little more work, since MCU software has to perform the averaging and offset compensation. Let's walk through the program shown in Photo 5.

After initializing the module, the baseline offset is established by taking the average of 16 proximity readings. Obviously, this calibration phase depends on nothing being in proximity in order to establish an accurate baseline.



```
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  threshold = 5 'proximity detection threshold

VAR
  byte led                'number of LEDs (0-8) lit for bargraph
  long proximity, offset 'VCNL4000 parameters

OBJ
  i2c : "Basic_I2C_Driver_1" 'by Michael Green, http://obex.parallax.com/objects/672/

PUB VCNL4000 'initialize, calibrate, proximity check

  i2c.Initialize(0)'initialize i2c driver
  i2c.WriteByte (0,$26,$83 | i2C#OneAddr,20) 'initialize IRLED current / 10 (ex:20 = 200mA)
  i2c.WriteByte (0,$26,$8a | i2c#OneAddr,129)'initialize mod delay & deadtime per datasheet
  ReadProximity  'dummy first time (slow) read
  dira[16..23]~~ 'set LED pins direction to output
  outa[16..23]~~ 'turn on all LEDs during calibration

  offset := 0 'calibrate offset by averaging 16 proximity readings
  REPEAT 16
    offset := offset + ReadProximity
  offset := offset / 16

  REPEAT 'proximity check - average 8 proximity readings, offset&scale, display on LEDs
    proximity := 0
    REPEAT 8                                                'read
      proximity := proximity + ReadProximity                'and average
    proximity := proximity / 8                              '8 proximity readings
    IF (proximity - offset) > threshold                     'if proximity detected
      led := ((proximity - offset) / 50) + 1                ' offset & scale for 8 LED bargraph
    ELSE                                                    'else
      led := 0                                              ' turn off LEDs
    outa[16..23]:=lookupz (led:0,1,3,7,15,31,63,127,255) 'turn on zero to eight LEDs

PUB ReadProximity 'issue IR command, wait 170uS, return 16-bit result

  i2c.WriteByte (0,$26,$80 | i2c#OneAddr,$8)               'issue IR command
  waitcnt ((clkfreq/5880)+ cnt)                             'wait 170 uS
  result := i2c.ReadByte(0,$26,$87 | i2c#OneAddr) << 8     'read & shift proximity MSB
  result := result + i2c.ReadByte(0,$26,$88 | i2c#NoAddr)'read & add proximity LSB
  RETURN
```

Photo 5: IR proximity sensing program screenshot.

Subsequently, the program performs checks for proximity by taking the average of eight readings, subtracting the baseline offset, and comparing the result against an application-specific threshold. In general, you can reduce the threshold for proximity detection, i.e. increase the range, by increasing the number of samples averaged to reduce noise. Finally, the result, or distance, is scaled for bar graph display on the SBC's eight LEDs.

When contemplating the specifics of your application, keep in mind that sensitivity, or threshold, and calibration strategy are intertwined. For instance, in the coarsest "yes/no" proximity applications with a large swing, where a high threshold is okay, factory or manual calibration may suffice. But for the highest-sensitivity, low-threshold applications, more frequent calibration may be called for, ideally just prior to each reading.

For instance, with my demo setup I noticed what appeared to be a bit of temperature drift, with the offset slowly increasing from power-up. The VCNL4000 datasheet isn't explicit in this regard, but there is a graph that indicates temperature drift varies across the full -40 to +85°C range, depending on the configured LED current. Though only on the order of one percent or so, it is enough to require periodic recalibration in the highest-sensitivity applications.

Software embellishments might include dynamically calibrating the baseline offset by adjusting it based on history. The exact nature of the algorithm used depends on application specifics, such as the relationship between the sampling rate and the proximity rate of change. Be careful to confirm that your algorithm is robust and avoids bugs such as "baseline creep" and "stuck key."

Since an individual IR proximity reading only takes 170 microseconds, faster than the I^2C transaction required to read the busy bit, I just included a delay for command completion in the software. The only "gotcha" is that an extra 400 microsecond delay is required for the first reading after initialization. Rather than slowing down every reading, I dealt with this special case by performing a dummy proximity reading at the beginning of the program.

The bottom line is that the VCNL4000 worked as advertised. For example, my demo setup could reliably detect the presence of my hand from a distance of almost 8 in., or 200 mm. Within range, the high resolution of the device, supplemented with software signal processing such as noise filtering, calibration and linearization makes possible very precise distance measurements. At the same time, the 16-bit dynamic range has enough headroom to deal with real-world vagaries, such as a scratched or dirty window.

Sources

Parallax Propeller multicore 32-bit MCU and Demo Board – www.parallax.com

Vishay VCNL4000 IR proximity and ambient light sensor – www.vishay.com