

Capacitive touch switch uses CPLD

Rafael Camarota, Altera, San Jose, CA

Capacitive touch switches work by measuring the change in capacitance of a PCB (printed-circuit-board) pattern depending on the placement of a user's finger over a sensing pad. Capacitive switches are becoming popular because they are less expensive than mechanical switches. Using the features of an Altera (www.altera.com) MAX IIZ CPLD (complex-programmable-logic device), you can implement a touch-switch decoder with no external components. The touch sensor employs an 8-mm-diameter sensing pad on the PCB using the solder mask as a dielectric. The circuit decodes a single switch, but you could use the approach for multiple switches, and it has programmable sensing thresholds that allow for different PCB layouts and dielectrics.

Figure 1 shows a simple circuit with no external components other than the capacitive-switch layout on the PCB. A basic touch-switch PCB layout is on the left. It comprises only an 8-mm copper circle surrounded by copper that connects to ground. The dashed line shows that the center sensor connects to the CPLD using a via and a backside copper trace. A solder mask acting as a dielectric covers the center sensor and ground. The PCB touch sensor becomes a variable capacitor, C_{TOUCH} .

The variable capacitor is part of a relaxation oscillator. The CPLD has a built-in weak pullup resistor on each I/O pin. C_{TOUCH} and the weak pullup resistor create an RC circuit. If the PINOSC (pin-oscillator) signal is low, the I/O pin will be low, making the D input to the PINOSC LPM (library-of-parameterized-modules) register

low. LPM blocks come from the Quartus II LPM.

The register and other logic in the circuit use a free-running, 4.4-MHz internal oscillator, ALTUFM oscillator, as a clock. On the rising edge of the clock, PINOSC goes low, making the buffer-driving pin go to a high-impedance state. The weak pullup resistor slowly makes the pin voltage rise based on an RC time constant. Not touching the switch causes it to have the lowest capacitance and fastest rise time. Touching the switch causes it to have the highest capacitance and the slowest rise time. The pin-I/O buffer uses the Schmitt-trigger option of the CPLD to reduce the noise sensitivity of

DIs Inside

70 Bit-shifting method performs fast integer multiplying by fractions in C

72 RS-232-to-TTL converter tests UARTs with a PC

74 Hot-swap circuit allows two computers to monitor an RS-232 channel

76 Improved laser-diode-clamp circuit protects against overvoltages

► What are your design problems and solutions? Publish them here and receive \$150! Send your Design Ideas to edndesignideas@reedbusiness.com.

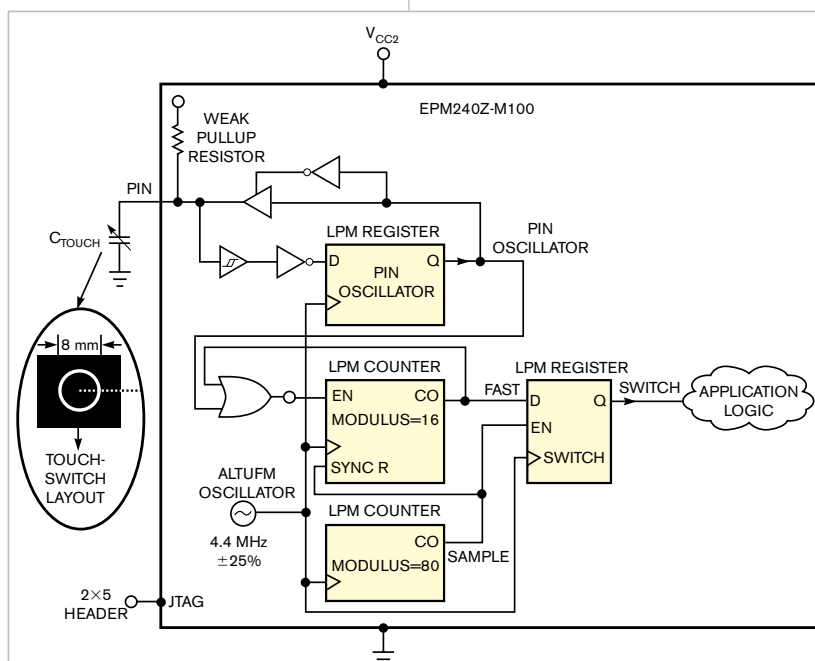


Figure 1 This capacitive-touch-switch decoder uses only a MAX IIZ CPLD and no external components other than the capacitive switch.

the slow-rising pin signal. Once the pin node reaches the high-voltage threshold, the D input of the PINOSC registers a zero. On the next clock edge, the PINOSC signal goes low, driving the pin node low for one full clock cycle. This PINOSC circuit oscillates at two fundamental frequencies, depending on the state of the touch capacitor. Putting the register into the oscillator loop reduces noise and makes the oscillator stable and synchronous with the decoding logic. The PINOSC period is always a multiple of 1/4.4 MHz or the frequency of the internal oscillator.

The switch decoder counts the period of 16 PINOSC cycles and compares it with a known time period. If 16 or more cycles happen in less than the sample period, it means that no one is touching the switch. If fewer than 16 cycles happen in the sample period, it means that someone is touching the switch, and the PINOSC oscillation becomes slower. The lower LPM counter sets the sample period.

For example, the sample signal was active once every 80 clock cycles in a prototype (**Figure 2**). The upper LPM counter measures the period of 16 PI-

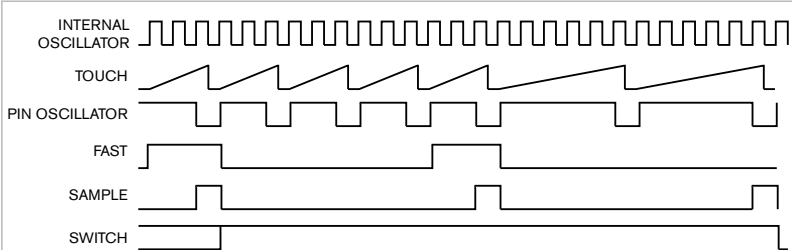



Figure 2 Representative waveforms that Figure 1 illustrates have a top-counter modulus of 3; a bottom-counter modulus of 12; and pin-oscillator periods of 3 and 6, respectively.

NOSC cycles. After 16 cycles, the fast signal goes high and stays high until the sample signal resets it. The fast signal is a one in the prototype when 16 cycles occur in fewer than 80 cycles, making the fast signal a one when the sample signal is a one. When the sample signal is a one, the fast value clocks into the switch-LPM register. The switch-signal value updates every sample cycle with the current capacitive switch state. When you touch the switch, PINOSC is slow, and the fast signal remains a zero when the sample signal is a one, making the switch output zero. In the prototype design, the PINOSC period was three clock cycles when someone

touched it and nine cycles when no one touched it. The switch threshold was five cycles. Therefore, the lower LPM-counter modulus was $5 \times 16 = 80$. You can use any value from four to eight, but four is too sensitive, and eight does not work for small fingers; hence, five is the best value. The upper LPM-counter modulus affects noise sensitivity. The larger the count, the more the circuit averages the period of oscillation. A low modulus makes the circuit more sensitive to random system noise. The five-cycle sensing point also allows margin for the $\pm 25\%$ variation among parts of the internal oscillator frequency. **EDN**

Bit-shifting method performs fast integer multiplying by fractions in C

Aaron Lager, Panamax Furman, Santa Rosa, CA

 This Design Idea presents a method for fast integer multiplying and multiplying by fractions. What can you do when you lack access to a hardware multiplier or MAC (multiply/accumulate) function and you need to multiply by something other than a power of two? One option is to include the math.h function and just sling around the multiplication operator and watch your code bloat and slow to a crawl. Option two is to get fancy with bit shifting. The general idea is to find powers of two, including zero, that you can add to achieve the multiplier you need. This method works because of the distributive prop-

erties of multiplication. Using the distributive properties of multiplication, you can, for example, rearrange the problem of: $12 \times 12 = 144 \rightarrow (4 + 8) \times 12 = 144 \rightarrow (12 \times 4) + (12 \times 8) = 144$. This version is amenable to implementation in C code because four and eight are powers of two. To implement the multiplications, you use the exponent of the power-of-two representation for your code as an integer shift. Because $4 = 2^2$ and $8 = 2^3$, you use two and three as your shift factors.

For example, multiply the variable foo by 12 to get 144: `BYTE foo=12; foo=((foo<<3)+(foo<<2))`. Left-shifting by three is the same as mul-

tiplying by eight, and left-shifting by two is the same as multiplying by four. Another example is multiplying by six: $6 \times 10 = 60 \rightarrow (2 + 4) \times 10 = 60 \rightarrow (2 \times 10) + (4 \times 10) = 60$. `BYTE foo=10; foo=((foo<<1)+(foo<<2))`. Left-shifting by one is the same as multiplying by two, and left-shifting by two is the same as multiplying by four.

Using this same theory of distribution, you can also perform fractional multiplication or division. This method creates rounding errors just like dividing integers by values that are not powers of two does with math.h functions and the division operator.

One example is $2.5 \times 10 = 25 \rightarrow (2 + 0.5) \times 10 = 25 \rightarrow (2 \times 10) + (0.5 \times 10) = 25$. The result is `((foo<<1)+(foo>>1))`. Left-shifting by one is the same as multiplying by two, and right-shifting by one is the same as dividing by two or multiplying by 0.5. Another example is

$3.125 \times 80 = 250 \rightarrow (2 + 1 + 0.125) \times 80 = 250 \rightarrow (2 \times 80) + (1 \times 80) + (0.125 \times 80) = 250$. The result is $((foo < 1) + foo + (foo > 3))$. Left-shifting by one is the same as multiplying by two, multiplying by one is the same as adding the multiplicand once to the result, and right-shifting by three is the same as dividing by eight or multiplying by 0.125. A third example is $2.625 \times 80 = 210 \rightarrow (2 + 0.5 + 0.125) \times 80 =$


$210 \rightarrow (2 \times 80) + (0.5 \times 80) + (0.125 \times 80) = 210$. The result is $((foo < 1) + (foo > 1) + (foo > 3))$. Left-shifting by one is the same as multiplying by two, right-shifting by one is the same as dividing by two or multiplying by 0.5, and right-shifting by three is the same as dividing by eight or multiplying by 0.125.

All of these examples take up less space and are faster than calling the

standard 8x8-multiply function or division function from most standard math libraries. Also, you should note that, if the result of the variable you are multiplying can ever exceed 8 bits, then you should use a word function that can store 16 bits of your result, and you should use casting on the outer parentheses. The result is $(word)((foo < 1) + (foo > 1) + (foo > 3))$. **EDN**

RS-232-to-TTL converter tests UARTs with a PC

Matthieu Bienvenüe, Malissard, France

 You often need an RS-232-to-TTL adapter for debugging or testing UARTs using a computer. But most of these adapters require an external power-supply adapter to power up the RS-232 transceiver. This external adapter increases the number of cables on your desk and uses no flow-control signals. This Design Idea describes how you can use these signals as power sources. It uses the RTS (re-

quest-to-send) and DTR (data-terminal-ready) signals, which provide a positive voltage when you open the PC's COM port (**Figure 1**). The voltage on those pins can differ from one computer to another but is generally higher than 6V, which is sufficient to power the adapter.

A standard RS-232 MAX3232 line driver from Maxim (www.maxim-ic.com) performs the TTL-to-RS-232

conversion. The MAX3232 accepts a 5 or 3.3V supply voltage, which is switch-selectable using S_1 . D_1 and D_2 block the negative voltage that occurs when the COM port is closed. Q_1 , R_3 , S_1 , and zener diodes D_3 and D_4 form a simple voltage regulator. LED₁ signals that the COM port is open. R_1 , R_5 , and R_6 protect the circuit under test and the line driver. The use of a pull-up resistor for R_2 avoids the need for an open input. This circuit has successfully undergone testing with a laptop computer, which provides a 6V power supply. The circuit works well at speeds as high as 115,200 bps. **EDN**

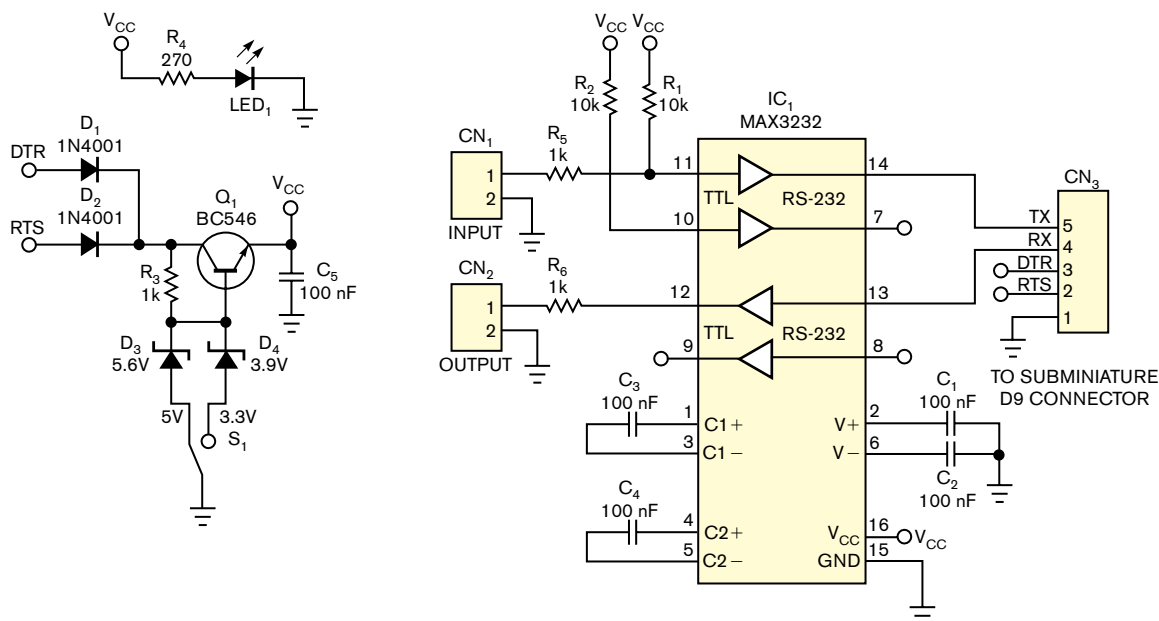


Figure 1 An RS-232-to-TTL converter uses the unused DTR and RTS outputs of a PC's COM port to self-power the circuit.

Hot-swap circuit allows two computers to monitor an RS-232 channel

Jeff Patterson, All Weather Inc, Sacramento, CA

The hot-swap serial-interface circuit in **Figure 1** allows two computers to see all of the communication between each computer and each device on the communication network for that serial port. This circuit allows each computer to determine what the other is doing

and receive all of the data from the peripheral device. Only one device can transmit at a time; otherwise, the transmitted data becomes corrupted. This circuit allows two computers in a hot-swap configuration to know when to become the master computer. When the master computer fails,

the slave computer stops receiving the data requests that the master supplies, and the slave then becomes the master. This approach allows for computer redundancy in applications in which a master computer that is communicating with an RS-232 device must always be operating. When you replace the failed computer, it “hears” that a master computer is communicating with the device and operates in slave mode while waiting for the current master to fail.

This circuit allows two DTE (data-terminal-equipment) computers to use one DCE (data-communications-equipment) RS-232 peripheral device. This device is usually a communication interface, such as a UHF radio or an RS-232-to-RS-485 converter. The board requires 9 to 15V dc to operate. You must provide this voltage on Pin 9 of the peripheral RS-232 device.

The transmitted RS-232 signal from the peripheral device converts to a TTL signal through level converter IC₃ and feeds into an AND gate. The output of this AND gate feeds into two inputs of another level converter, IC₄. These RS-232 outputs travel to the input lines (Pin 2) of the two monitoring computers. When one of the computers transmits on Pin 3 of its serial port, its output converts to TTL levels with IC₄. The TTL-converted outputs of both computer serial ports feed into an AND gate. The default, or off, level for a computer serial port is -12V dc. The level converter inverts the signal as part of the conversion to TTL levels. This action makes the default a high level going to the AND gate, allowing the data on the other input of the AND gate to pass to the output of the AND gate.

The output of this AND gate goes to the second input of the AND gate that receives the output of the peripheral device as well as the input into the level converter going to the input of the peripheral device at Pin 3. This action enables the output of one of the two computers to return to the computer that transmitted the data as well as to the other computer and the peripheral device. **EDN**

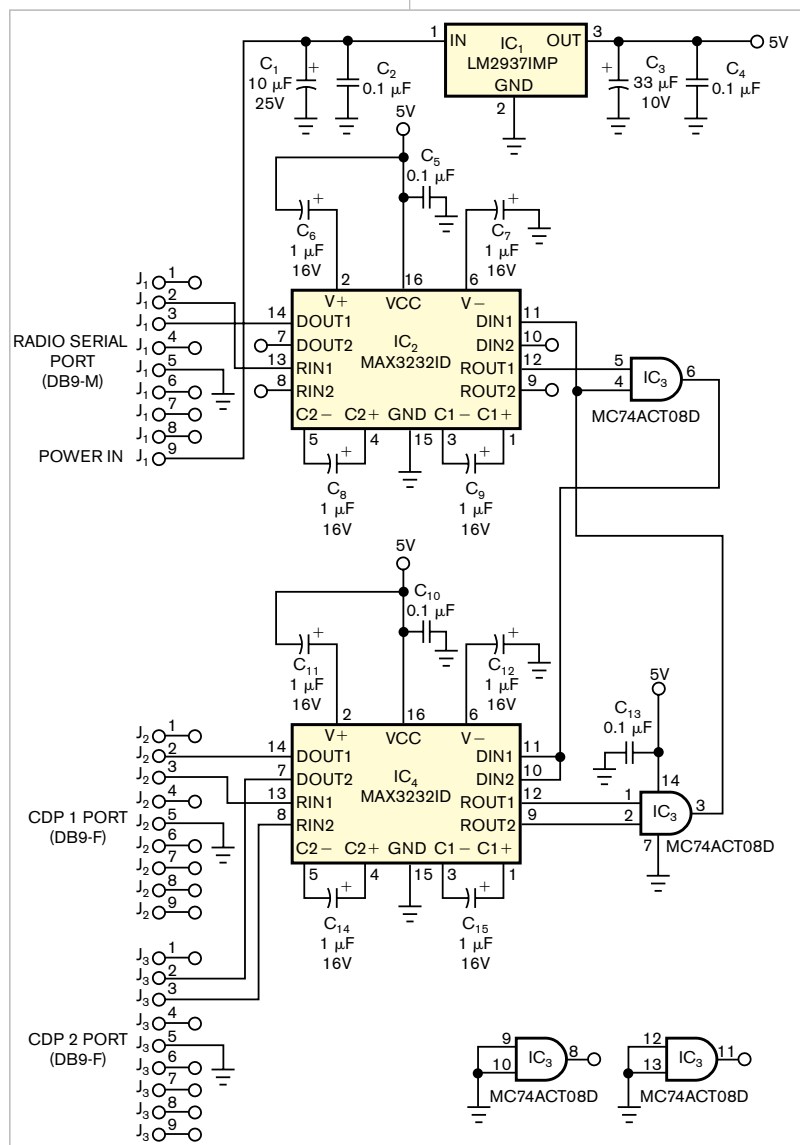


Figure 1 This hot-swap RS-232 interface allows two computers to monitor traffic on a computer's RS-232 port.

Improved laser-diode-clamp circuit protects against overvoltages

James Zannis, Baulne-en-Brie, France

Expensive semiconductor laser diodes have no tolerance for fast voltage or current transients. To minimize the risk of damage, a standard JFET-clamp circuit shorts the laser when there is no supply voltage, thus protecting it against such transients (Figure 1). When the negative supply rail comes up, the JFET turns off.

This circuit is effective for low-power laser diodes but may not be so for diodes with power dissipation greater than 150 mW. The maximum cutoff current of the JFET sets this limit. If it becomes necessary in an emergency to clamp the laser during normal operation, the selected JFET might not adequately shunt the current. Higher-cur-

rent JFETs are available but are more expensive and difficult to procure.

The circuit in Figure 2 avoids these deficiencies. It is similar to the standard JFET circuit but has a supplementary bipolar transistor that shunts most negative-going currents when the JFET is on. R_2 prevents the gate of Q_1 from floating, and R_3 ensures rapid turn-off of Q_2 . The 1N914 diode bypasses any positive-going transients. The RC circuit ensures an adequately slow response; therefore, the transition between on and off is smooth. **EDN**

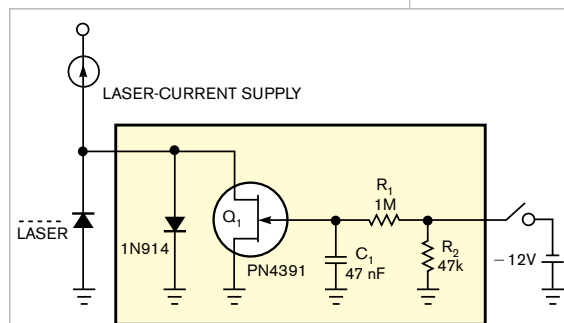


Figure 1 This circuit protects low-power laser diodes but is not suitable for higher-power laser diodes.

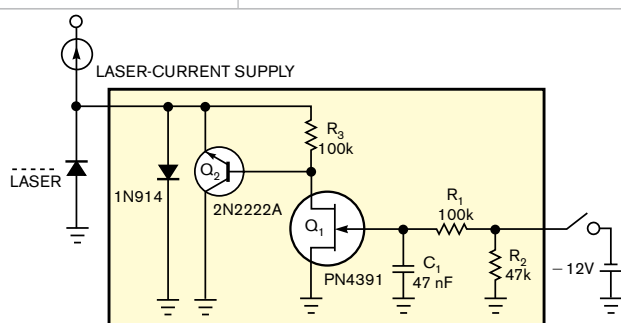


Figure 2 Adding a bipolar transistor to the circuit in Figure 1 allows the circuit to protect higher-power laser diodes.