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, $C_{\text{TOUTCH}}$.

The variable capacitor is part of a relaxation oscillator. The CPLD has a built-in weak pullup resistor on each I/O pin, $C_{\text{TOUTCH}}$, 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 the pin.

Figure 1 This capacitive-touch-switch decoder uses only a MAX IIZ CPLD and no external components other than the capacitive switch.
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 stick 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 properties of multiplication. Using the distributive properties of multiplication, you can, for example, rearrange the problem of: 12×12=144→(4×8)×12=144→(12×4)+(12×8)=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=2² and 8=2³, 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 multiplying by eight, and left-shifting by two is the same as multiplying by four. Another example is multiplying by six: 6×10=60→(2+4)×10=60→(2×10)+(4×10)=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 2.5×10=25→(2+0.5)×10=25→(2×10)+(0.5×10)=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
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 (request-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 $S_1$, $D_3$, and $D_4$, and zener diodes $D_3$ and $D_4$ form a simple voltage regulator. LED signals that the COM port is open. $R_s$, $R_p$, and $R_1$ protect the circuit under test and the line driver. The use of a pull-up resistor for $R_s$ 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.

![Figure 1](image-url) 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 IC2 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 IC3. 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.
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 mA. 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-current 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.