

## Learning About Capacitive Proximity Sensing By Building A Theremin

by Todd Leshner, Cypress Semiconductor

Capacitive proximity sensing is a method to determine how close two items are to each other. In many cases, it is used in user interfaces to measure the distance a finger is from a sensor. Capacitive sensing is useful in many applications because it provides an input method with no moving parts.

Often when learning a new technology, it helps to have a project in mind that is simple enough that the majority of design and troubleshooting is directly related to implementing the technology. It can also help keep an engineer's drive going when the project is interesting and exciting. To this end, this TechNote describes how to make a rudimentary Theremin – a musical instrument with two controls: pitch (frequency) and volume (amplitude) – to illustrate the basics of working with capacitive sensing. At the end of this TechNote, you'll find a list of parts, several reference sources, and a link for downloading the software you'll need.

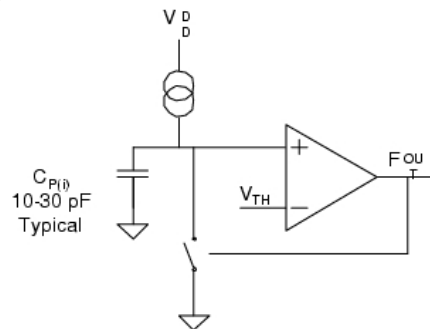
### Theremin Basics

In its simplest form, a Theremin creates a periodic waveform which is driven to an amplifier and, subsequently, a speaker. There are two controls: pitch and volume. The pitch input controls the waveform frequency while the volume input controls the amplitude. Traditionally, each input is an antenna connected to an RF resonant circuit. As a hand comes near the antenna, capacitance is added to the circuit, modifying its frequency. For good information about how the frequency is used, see these links:

<http://mrr3000gt.mystarband.net/MRT/index.htm>

<http://www.Thereminvox.com/article/articleview/15/2/2/>

### Capacitive Sensing Basics



**Fig. 1: Relaxation Oscillator**

Rather than trying to model the beat frequency method, I used a method of capacitance measurement sometimes used in conjunction with microcontrollers. A CSR (capacitive sensor relaxation oscillator) consists of a current programmable relaxation oscillator (fixed current source, reference voltage and a comparator), a multiplexer, and a digital counter (see Fig. 1).

Here's how to measure an absolute capacitance:

- Connect a small capacitor ( $C_p$ ) to one side of a comparator
- Charge  $C_p$  with a constant current until the positive node of the comparator is greater than the reference voltage on the negative node
- Once the comparator trips, short  $C_p$  to ground – removing all charge
- Measure how long it took to charge  $C_p$  to the reference voltage
- Convert charge time to capacitance

Charge time is converted to capacitance using the relationship between voltage across and current through a capacitor:

$$i(t) = C_p \frac{dV(t)}{dt}$$

Solving for  $C_p$ :

$$C_p = \frac{i(t)dt}{dV(t)}$$

$i$  is from a constant current source.  $V$  is a constant comparator reference voltage, so:

$$C_p = \frac{it}{V}$$

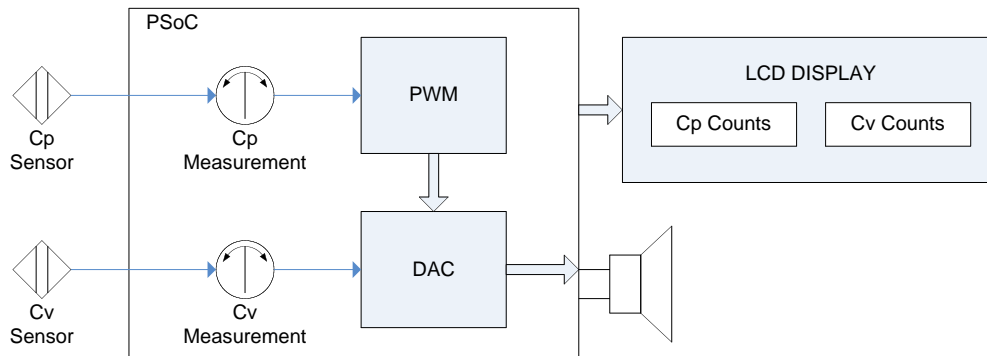
Because of the small capacitances involved, it is sometimes easier to perform relative capacitance measurements. This is where a multiplexer comes in. Connect a known reference capacitor ( $C_r$ ) to one input of the multiplexer and the sensing node ( $C_s$ ) to the other. First for  $C_r$ , then for  $C_s$ , measure how long it takes to charge the capacitors to the reference voltage. If  $C_s$  happens to be one-half the value of  $C_r$ , it will take half as long to charge  $C_s$  than it will to charge  $C_r$ .

The final piece of information is how to convert charge time to a digital value usable by firmware. Counters on a microprocessor can be used to count how many clock cycles it takes for the charge to reach a threshold voltage. The higher the capacitance, the higher the count will be.

### **Final Product Integration**

All that remains is to allow the pitch input capacitance ( $C_p$ ) to control the output frequency, and the volume input capacitance ( $C_v$ ) to control the output amplitude. To keep all the circuitry on one chip, I routed an 8-bit PWM through an 8 bit-DAC.  $C_p$  was scaled to an 8-bit value and used for the compare value of the PWM.  $C_v$  was scaled to an 8-bit value and used for the DAC setting. Then I attached a 32  $\Omega$  speaker, and there was sound!

Fig. 2 shows the system block diagram.



**Fig. 2: System Block Diagram**

## Troubleshooting And Experimentation

One of the most useful tools for troubleshooting and experimentation is an LCD. Even a simple 2 x 16 character display can provide a vast window into the internal workings of a system. I used the display to periodically update the counter values. Changing the shape of the sensor wire immediately caused a change in the number of counts on the display. This made it a simple process for me to easily try out other sensors.

Capacitive proximity sensing is an extremely useful and versatile technology that does not have to be complex to implement, as has been shown by this example application. By keeping the project simple and focused, engineers can quickly learn the basics of capacitive sensing so that they can implement it in larger, more complex systems.

The following is the component list used to build this system:

### Hardware

- PSoC with two CapSense inputs – one for pitch, one for volume  
I chose an 8C24x94 pod because it has a 28-pin DIP footprint (see this link for details on the part <http://www.cypress.com/?rID=3472>). However, the less expensive CY8C24794, CY8C24894 or CY8C24994 QFN parts can be used just as well (see this link for details <http://www.cypress.com/?rID=17853>)
- Capacitive Sensors  
I used two 2-inch breadboard jumper wires. They plug into the connectors that are built into the 8C24x94 pod. Bending them into circles changed their sensitivity
- PSoCEval1 Development board with LCD display: By using an LCD, I avoided the need for an ICE (In-Circuit Emulator). An LCD can prove very useful for experimentation and debugging (See this link for details on the LCD <http://www.cypress.com/?rID=2541>)
- MiniProg to program the PSoC: comes with PSoCEval1 Board

## Software

- PSoCDesigner 4.4, with included CSR User Module. Version 5.0 should work just as well: Click here <http://www.cypress.com/?rID=34517> for more information and a download link

## **How To Obtain And Use The Configuration Files**

Here is a link to the configuration files: <http://www.cypress.com/?rID=37467> To use these files:

- 1- Extract them to your PC then open the *psocTheremin.SOC* file with PSoCDesigner. Everything but the API files is in main.c.
- 2- To compile the binary, choose from the menu : *Config* → *Generate Application*, then select *Build* → *Build*
- 3- Hook up your MiniProg to your PC with a USB cable and to your pod or evaluation board with the five pin header
- 4- Now choose Program → *Program Part*. The program will start running after programming completes. If not, toggle the power in the Programmer application, or detach the programmer and apply power to the eval board
- 5- Values will show up on the LCD
- 6- If you run into any difficulty, the PSoC Developer CapSense Forum: <http://www.psocdeveloper.com/forums/viewforum.php?f=22> is a good place to post your questions, or to search the database to see if they have already been answered

## **References**

Dave Van Ess video on getting started with CapSense: <http://www.cypress.com/?id=1295>  
CSR User Module documentation (included in PSoCDesigner)

## **About The Author**

Todd Leshar has been testing, troubleshooting and fixing ICs, PCBs and embedded systems since 1999. He earned a BSEE at Colorado Technical University and is a senior systems engineer at Cypress Semiconductor. Todd can be contacted at [tlx@cypress.com](mailto:tlx@cypress.com)

