PROJECT: Multilevel Audible Probe
LOCATION: Bulgaria
PAGE: 14

EE TIPS: Electronic Product Simulation & Testing
LOCATION: United States
PAGE: 60

INSIGHT: Efficient Low-Power Electronics
LOCATION: United States
PAGE: 64

CIRCUIT CELLAR

THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION

MAY 2012
ISSUE 262

MEASUREMENT & SENSORS

Implement Cap-Touch Sensors

Sensors & Image Stabilization

DIY Multipurpose Interface for Embedded Systems

Embedded Security: Threat Prevention & System Protection

"SNAP" Projects

PLUS

Life in Transmission

The Exciting Endeavors of a Communications Pro

// Operating Sonar Systems
// Working in Fiber Optics
// Developing IR Technology
// And More

05>
7 25274 75349 9
$7.50 U.S. ($8.50 Canada)
www.circuitcellar.com

by Matt Oppenheim (United Kingdom)

Audio-Enhanced Touch Sensors

Using a touch sensor, you can trigger audio tags on electronic devices. Here you learn how to couple capacitive-touch sensors with an Android device.

Adding touch sensors that trigger audio tags to the controls of everyday objects can help visually impaired users operate an unfamiliar device. One solution is to couple capacitive-touch sensors with an Android device.

I built a scalable, 16-channel touch sensor that interfaces with a PC or an Android smartphone. I installed these in some audio players to help visually impaired users. When the user touches a control, an audio tag is triggered that tells the user what the button does before it performs the function.

I used Microchip Technology's mTouch capacitance touch-sensor technology to instigate the touch sensors. I built a clip-on light-emitting diode (LED) board to show which channels are active and a battery board that powers the assembly from a single AAA battery. Photo 1 shows the completed boards and how they are mounted on a CD player.

Two implementations of the touch sensor are presented. The first uses a PC or laptop to produce the audio tags. The second is a fully portable system using an Android smartphone to produce and optionally record the audio tags.

This article explains how this technology works and how I built the touch-sensor boards. Firmware using MikroElekronica's mikroC PRO complier and software development using Java are also described.

CAPACITIVE-TOUCH SENSING

During my annual run around campus I got to thinking about how capacitive-touch sensors work. Bear with me in this analogy. On the road surface I get a firm rebound and trundle along with the low but constant rate of energy expenditure needed to maintain my pace. Then I head onto the soggy playing fields and my feet begin to trace a much more ragged path. To maintain the same pace, I would have to work harder. But I can't, as I am already maxed out. However, my stopwatch continues its merciless count unaffected by how much I have slowed down. The analogy with how capacitive-touch sensors work is that the user adds capacitance to a line directly connected to an oscillator—the capacitance-sensing oscillator (CSO). This capacitance acts in the same way that a soggy surface acts on me as I run—it slows down the oscillator. If the oscillator could add more current, it could maintain its pace, just as if I could add more energy to my running, I could maintain my pace. But the capacitive-touch sensor line is designed to be fixed current, so it will slow down. The user has access to a register to set this current to one of three preset values to enable a range of different size touchpads. By monitoring the frequency of the CSO using an isolated internal counter (which corresponds to my wrist watch when I am running), you can measure that the CSO slows down as it is loaded. Similarly, when the finger is removed from the touch sensor, the decrease in

Photo 1a—Two daisy-chained Microchip Technology mTouch boards with a battery board providing the power and LED boards showing the channel status. b—A single mTouch board attached to a CD player with an LED board attached. c—This is a close up of the buttons on the CD player. The thin laminated wire connects to the mTouch board and triggers the detector when touched.

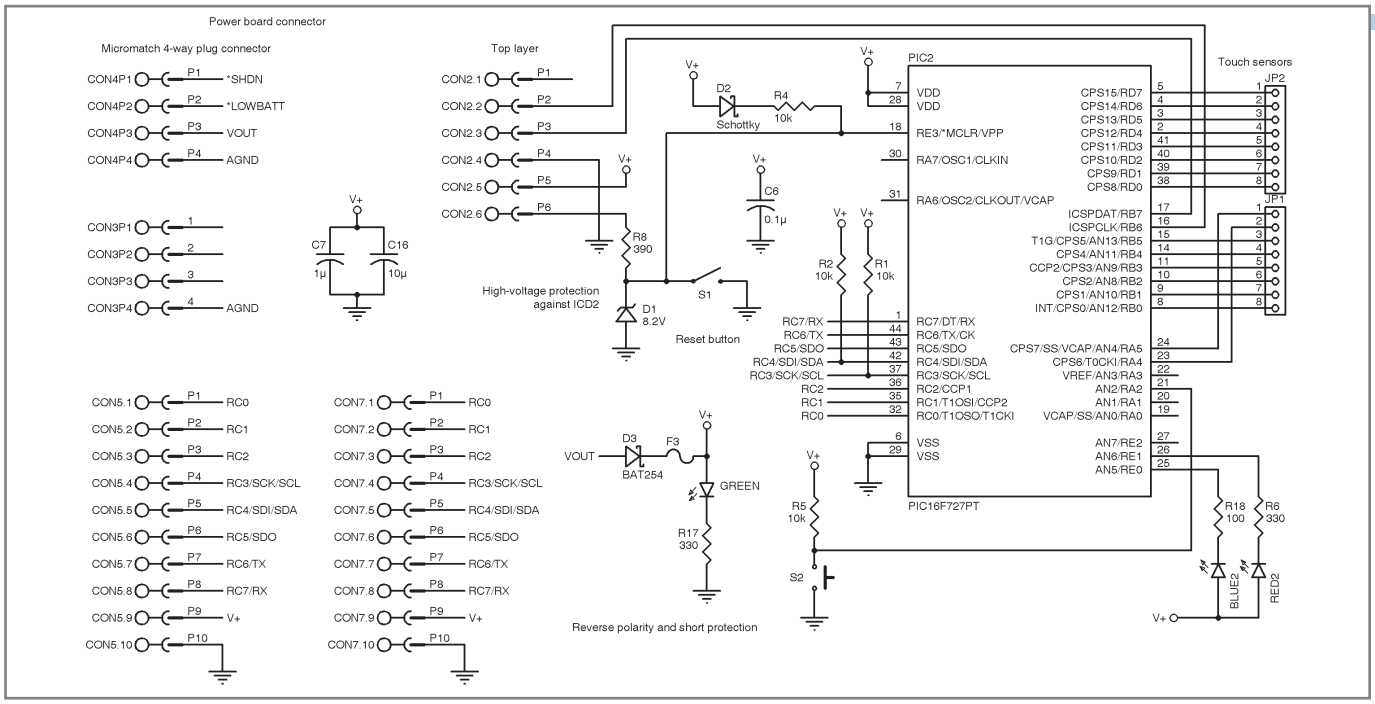

Figure 1—The main touch sensor board features a Microchip Technology PIC16F727 microcontroller.

capacitance will cause the CSO to speed back up again.

mTOUCH BOARD

In my article, "Multichannel Touch Sensors: Implement Scalable Capacitive Touch Sensing" (*Circuit Cellar* 234, 2010), I described a scalable touch-sensor system I built based on Atmel's QProx QT1103 capacitive-touch sensor chip. Microchip brought out a range of microcontrollers after I finished that project, which makes adding touch sensors a bit more straightforward for the homebrew engineer. Microchip calls the technology mTouch. You have to set up the touch sensors in firmware, which is a bit less convenient than using the dedicated QProx solution. But, having a microcontroller wrapped around the touch sensor makes adding communications, flashing LEDs, and button controls easy to implement. Plus, the chips come with legs which makes home assembly a lot less painful than the legless QFN32 package available in the QT1103!

My main mTouch board is shown in Figure 1 and Figure 2. I used Microchip's PIC16F727 microcontroller, which has 16 channels available to configure as touch sensors. A single-capacitance CSO is multiplexed between these channels.

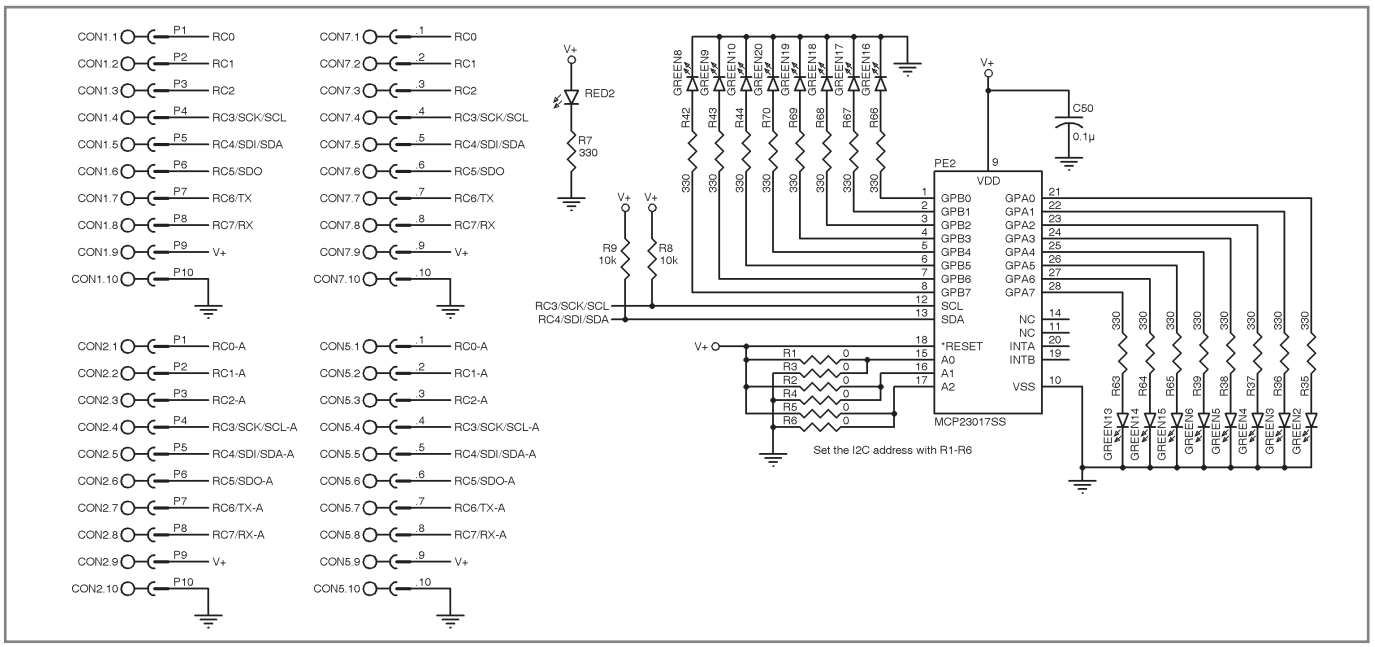

Figure 2—The LED board indicates which touch sensor channels are active.

The 44-pin TQFP version of the chip is not too tricky to hand-solder. Lightly flex a scalpel blade against each pin after tacking it down to verify that the solder joint is solid. The chip's datasheet recommends protecting the MCLR line against overvoltage protection as the Microchip MPLAB ICD 2 in-circuit debugger/programmer can take it dangerously high during programming. This is provided by the Zener diode D1 and resistor R8.

Two LEDs connected to the PIC enable simple debugging flashes and an "I'm alive" signal during start-up. I placed a Reset button on the *MCLR line. The firmware recalibrates itself each time the board is reset. This means if you twist up the touch sensor wires and they start to randomly trigger all you have to do is reset the board and you are good to go again. Of course, you can just toggle the power switch to do this as well. I guess I'm trying to justify habit—I've always put a Reset button on my microcontroller boards.

I designed and laid out my two-layer printed circuit board (PCB) using Eagle v5.60. I'm too stingy to pay for silkscreen, so I put my text onto the copper layers. I have learned the hard way. Always print the version of board onto the PCB to avoid later confusion.

Listing 1—In the `while` loop, each capacitive-touch sensor channel is repeatedly configured until TMR0 overflows. The time taken to do this is compared each time the loop repeats. A change indicates that a touch event has occurred.

```
while(1) {
   channelNumber = startChannel;
   reset_system();
   while (channelNumber <= lastChannel){  // scan through each touch channel.
      configurePort(channelNumber);// repeated until timer overflows
      if (interrupt_alarm) { // s/w interrupt alarm bit set by ISR
         alarmInterrupt(channelNumber); // handle interrupt flag
         reset_system(); // Start again with a clean slate
         channelNumber++ ;
       } // end if
   } // end while
   update_channel_status(); //  updates channel status register
   //noChange is a flag for when channel status is changed.
   if (noChange == false) { // changed i/p, so change LEDs
      mcp_activechannels(); // work out which LEDs to light on board
      mcp_update(LEDboard, MCPportA, MCPportB);// activate LEDs on board with status
      noChange = true; // reset flag
   } // end if
}   // end while(1)
```

CONNECTORS

I use a six-way Tyco Electronics Micro-MaTch header to connect the ICD2–CON2 shown in Figure 1. I like Tyco's Micro-MaTch connectors as they are keyed and give a positive click when connected. They are available in a range of styles: through-pin, surface-mount, and crimp-on connectors for ribbon cable. The ubiquitous 0.1" unkeyed header pins commonly used for connecting programmers just ask you to connect your programmer the wrong way around and potentially connect the board's ground to your power rail! A pair of four-way Micro-MaTch connectors (CON3 and CON4) enables me to clip on a DC-to-DC converter board with an AAA battery to power the assembly. The pair of 10-way connectors enables access to the I²C, SPI, UART, power, and three signal lines. As both of these connectors share the same signals, I can use them to daisy chain several boards together using the corresponding crimp-on headers and a length of ribbon cable (see Photo 1a). An LED board clips across these connectors to show which of the channels are active.

I crimped a 10-way Micro-MaTch connector to a Future Technology Devices International (FTDI) UART-to-USB cable to enable communication between the PIC's UART and the PC. For this application, I only sent data one way, from the PIC, but I have tested two-way communications using this hardware in another project. Using the FTDI cable enabled detailed debugging messages to be displayed on the serial port terminal of the mikroC PRO compiler that I use to develop firmware. In the finished device, the cable enables the touch-sensor status to be sent to the PC to trigger an audio tag for each channel. The cable also enables the boards to be powered from the 5-V rail on the USB port.

Each of the 16 touch-sensor channels are routed directly to 0.1" header pins, eight on each side of the board. A corresponding strip of header sockets slides onto these pins, which enables the boards to be removed from the devices they are being used with. Thin laminated wire is soldered to the back of these sockets. The connectors and the laminated wire are shown in Photo 1b and Photo 1c. Thin laminated wire is soldered to the back of these sockets. These wires run to where the touch sensors are required. In my case, they run to the top of the radio and CD player controls. On the CD player, I made a little coil of wire on the area where I needed the touch sensor and stuck it down with a thin layer of clear adhesive. On the radio, I was able to use a needle to stitch the laminate wire through the rubber buttons.

LED BOARD

I made a clip-on LED board that indicates the active channels. The touch sensors run fine without the board attached, but the lights look great and are useful for checking that the touch pads are correctly connected.

All LEDs are not created equal. Check the brightness rating, expressed in millicandela (mcd), the candela being a unit of luminosity. You can get more bang for your buck or more mcd for your milliamp by paying a little more for your LEDs.

I used Microchip's MCP23017 I/O expander to drive the LEDs. This interfaces with the PIC16F727 using the I²C communication protocol. Prior to this project, I used the SPI version of the port expander as I thought the extra data line this protocol uses is somehow more robust. But Jeff Bachiochi's article, "Extend and Isolate the I²C Bus" (*Circuit Cellar* 233, 2009), made me reconsider. I²C only ties up two lines and is

Listing 2—Bit handling definitions and configuration and interrupt handling subroutines. The ISR deals with TMR0 overflowing. The complete listing on *Circuit Cellar*'s FTP site deals with other interrupts.

```
#define bit(num) (1 << num) // creates a bit mask
#define bit_set(v, m) ((v) |=(m)) // Sets the bit
// e.g. bit_set (PORTD, bit(0) | bit(1));
#define bit_clear(v, m) ((v) &=  ~(m)) // Clears the bit
#define bit_toggle(v, m) ((v) ^= (m)) // toggle the bit
#define bit_read(v, m) ((v) & (m))  // read a bit and see if it is set
#define bit_test(v,m)      ((v) && (m))
#define startChannel 0
#define lastChannel  15

// Configure tris & CPSCON1 settings for channel
void configurePort(char channel) {
   CPSCON1 = channel; // Set CSO to active channel
   //uart_write_short(CPSCON1); // debug info
   if (channel < 6) {
     TRISB = TRISBCapOscOn[channel]; // Channels 0-5 on portB
   } else if (channel < 8){
     TRISA = TRISACapOscOn[channel]; // Channels 6-7 on portA
   }else {
     TRISD = TRISDCapOscOn[channel]; // Channels 8-15 on portD
   }
}

void scan_channels(){ // get CSO count for each channel
   char channel;
   while (channel <= lastChannel){
     configurePort(channel);// repeated until timer overflows
     if (interrupt_alarm) { // s/w interrupt alarm bit set by ISR
       alarmInterrupt(channel); // handle interrupt flag
       reset_system(); // Start again with a clean slate
       channel++ ;
     }// end if
   } // end while
}// end scan_channels

void interrupt() { // Interrupt Service Routines
   INTCON.GIE = 0; // Global disable interrupts
   if (INTCON.T0IF) { //tmr0 overflow
     CPSCON0.CPSON = 0;   // Turn cap sense off - disable tmr0
     T1CON.TMR1ON = 0; //Turn tmr1 off
     INTCON.T0IF = 0; // Reset tmr0 interrupt flag
   } // end if
}// end interrupt()

void alarmInterrupt(short intchannel) { // deal with interrupts outside of ISR
   if bit_read(interrupt_alarm,t0Interrupt) { //tmr0 interrupt flag
     disable_interrupts(); // The LCD write will cause timer IF
     capCount[intchannel] = TMR1H; // CSO count from MSB of tmr1
     bit_clear(interrupt_alarm, t0Interrupt);  // clear alarm flag for tmr0 interrupt
   }
} // end alarmInterrupt
```

plenty fast enough to cope with this application. The chip can be given a unique I²C address by pulling the address lines A0–A2 high or low using 0-Ω resistors on pads R1–R6. Photo 1a shows two boards daisy chained together. Each has its own LED board with different I²C addresses. Naturally, you have to change the firmware for the mTouch board to match the LED board's I²C address! I defined the I²C addresses of the board's statements such as `#define board2 0x44` at the start of my code, then refer to `board2` throughout the rest of the code.

I placed the same 10-way port expander sockets the LED board clips onto on the top side of the LED board. This enables you to connect the stack to a PC using the FTDI cable or to daisy chain several boards together with the LED boards attached.

FIRMWARE

I developed the firmware for the PIC16F727 using the mikroC PRO compiler v4.60 programming environment which is a free download. I have yet to exceed the demo version's size limit. The final code is available on the *Circuit Cellar* FTP site. I based my code on the example C code supplied by Microchip for the mTouch technology.

Once the various registers are initialized, the program enters an endless while loop—which is a standard way of writing microcontroller code (see Listing 1). In this loop, each of the touch-sensor channels is connected to the CSO in turn. The CSO clocks the 8-bit timer register, TMR0, until it overflows. This triggers an interrupt that is handled by the interrupt service routine (ISR) called `interrupt()`. This resets TMR0 and sets the flag

interrupt_alarm to be true (see Listing 2). The program then branches to the subroutine alarmInterrupt (channelNumber). I could have put the code from this subroutine into the ISR, but good practice is to keep the ISR as short as practical—in case another interrupt comes along while you are already in the ISR. This could put you into a situation where you never manage to get out of the ISR! There are a couple of other possible interrupt situations, such as the UART triggering an interrupt if you set up two-way communication. These are dealt with in the complete listing on the FTP site.

So, touching a channel will cause the CSO to slow down while it is connected to that channel. As the CSO clocks the 8-bit counter TMR0, this will take longer to overflow. Meanwhile the 16-bit counter TMR1 is being clocked by the internal oscillator in the PIC. The rate of increase of TMR1 is unaffected by touching the channel, so it will have a higher value than usual when TMR0 overflows. If TMR1 starts to overflow instead of the smaller TMR0, this would be a reason to change the current setting bits, which are called CPSRNG0 and CPSRNG1, in the CPSCON0 register. This controls what fixed current is available to the CSO, which in turn controls how much the CSO will slow when it is loaded with the extra capacitance of a finger touch.

In alarmInterrupt, the most-significant byte of TMR1, is stored in an array as capCount[intchannel], which represents the capacitance of that channel. If there is an increase in this value, the channel is flagged as having been touched. Similarly, once touched, if there is a decrease, the channel is flagged as no longer being touched.

Note the definitions of bit_set, bit_clear, bit_read, and bit_test at the start of Listing 2. I got this idea from "Bit Flipping Tutorial: An Uncomplicated Guide to Controlling MCU Functionality" by Eric Weddington (*Circuit Cellar* 180, 2005).

I use the Microchip PICkit2 programmer/debugger for my programming. The standalone interface it uses can be configured to automatically load a hex file whenever you recompile your code. This makes it easy to use your choice of C compiler. I made a programming cable by crimping a six-way Micro-MaTch plug onto some ribbon cable, then soldering the other end of the cable to standard 0.1" header pins, which slots into the PICkit2. You can power the board via the programmer, which can be a useful option while debugging your code—in the unlikely event your code ever needs debugging.

My code works. The Microchip example has a time-averaging function to compensate for changing atmospherics starting to trigger the detectors. This may be something to incorporate at a later date if I start having problems.

JAVA GUI

As mentioned, I have two setups for the boards. For some of my research on human-computer interaction, I need to connect the boards to a PC to accurately log timing information.

I developed a graphical user interface (GUI) on my PC that displays the status of each channel and plays an audio tag whenever a touch sensor triggers. Luckily, I work in a computer science research lab, so I trade soldering for Java tuition. I use the freely available Eclipse programming environment for Java development. Oracle, which now develops Java, provides some excellent Java tutorials online.

A screenshot of the GUI is shown in Photo 2. There is a button control for each channel that lights up as a channel is triggered. You can also click on the button to test the audio clip that is associated with that channel. A record of when each channel is triggered and released is displayed and can be logged to a text file. The sound can be toggled on and off. As you move your finger from one touch sensor to another, the audio from the previous channel is killed to prevent a confusing cacophony of overlapping sound tags.

I used the gnu.io library to service the virtual com port set up by the FTDI driver. This enables me to treat the interface as a serial port. The audio clips are all .wav snippets that are held in a hashtable structure. I used Swing components to build the interface.

To make the .wav files, I recorded my own voice using High Criteria's Total Recorder audio and video recording software program. I used the same program to edit the files to remove any dead space at the start of the recording, so there is immediate audio as soon one of the sensors is touched. The full Java listings are on *Circuit Cellar*'s FTP site.

PORTABLE ANDROID VERSION

The touch-sensor circuitry can be discretely stuck onto the back of the devices I am enhancing. Having a

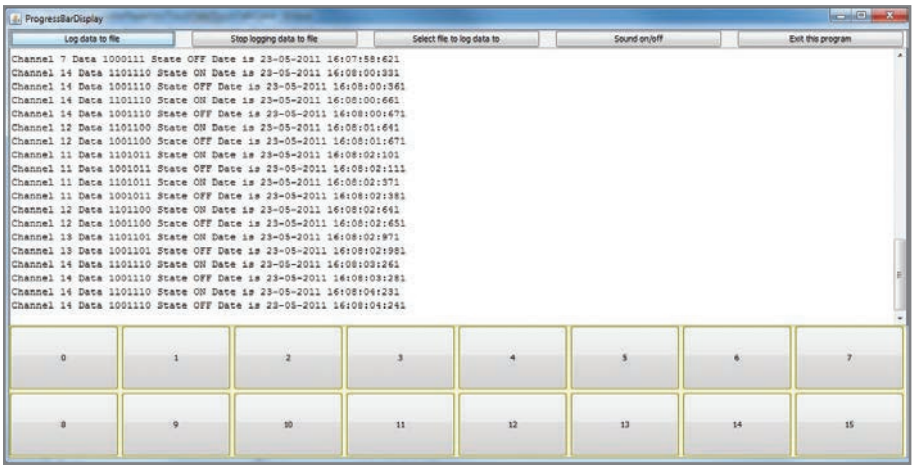

Photo 2—The Java GUI for the mTouch board. Each button represents a channel. The channel statuses are updated in the text area.

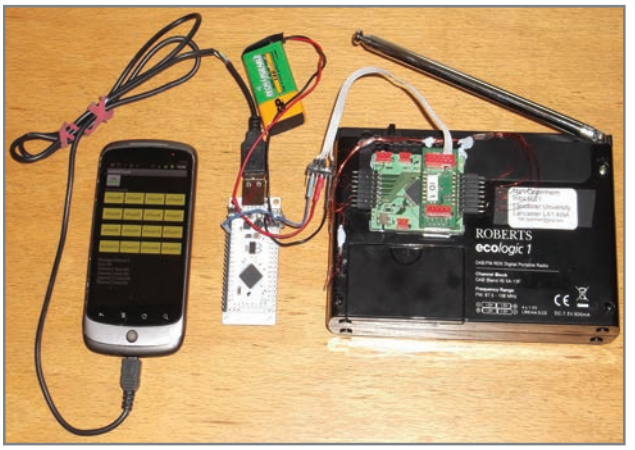

Photo 3—The portable touch-sensor assembly. The touch-sensor boards are mounted on the back of a digital radio, connected to a IOIO board and a Nexus One smartphone. The Android interface is displayed on the phone.

portable system for generating and recording the audio tags means I can leave an enhanced device with a visually impaired user until he or she has learned the layout of the controls. Then I can disconnect the touch-sensor boards and take away the extra hardware. The sensor wires that have been added to the device can stay in place with no effect on the usability of the player.

I started by prototyping a board based on a Nuvoton Technology ISD ChipCorder speech recorder IC. But the quality of sound was never going to be great due to the low bit-rate sampling. The recording capacity on the chips is adequate for the implementation shown in this article, but I have plans to daisy chain a number of boards together to add tags to a keyboard. I settled on using an Android phone as it already has a speaker, a microphone, and a lot of memory to store as many audio tags as I could ever need. To add audio to the handful of devices I am looking at enhancing, buying a few second-hand phones is not going to be more expensive than paying for a few custom-made boards.

Detailing how to program an Android phone is an article in itself. I used Java running on the Eclipse development platform with a plug-in produced by Google. Luckily, the Java skills I acquired through programming the interface on the PC could be reused for programming the Android phone. The Android interface is shown in Photo 3. Activating a touch sensor causes the corresponding button to flash and the audio tag to be triggered. If you need more volume than the phone can deliver, it is easy to add a portable speaker.

Audio tags can be directly recorded using the phone's built-in microphone or preloaded into memory. I found that preloading the audio files provided the best quality. I used a separate program to generate audio tags using Google's text-to-speech library.

But how could I get the data from the touch-sensor boards into the phone? This is where the IOIO (pronounced "yo-yo") board developed by

SparkFun Electronics and Ytai Ben-Tsvi, a Google engineer, came to the rescue. The board uses a Microchip PIC24FJ256DA206 microcontroller from the PIC24F family, which has USB on-the-go built in, so it acts as a USB host. The phone is hosted by the board. The IOIO comes preloaded with firmware that enables you to easily control the interfaces on the PIC24 (UART, SPI, I²C, etc.) directly from the phone. Adding the IOIO library to your Eclipse project enables you to access these interfaces.

I looked at some other solutions for interfacing my phone with the board, which gives greater control of the interface board by directly programming the board's firmware. But I couldn't get them to be as reliable as the IOIO board. I'm probably doing something wrong and I hope to return to the more advanced boards for a future project. The IOIO board proved to be a simple and reliable solution.

One of the stumbling blocks in using Android phones for data collection is that the phone will try to recharge itself whenever you connect it to a USB port. This means being able to supply up to 500 mA, which is a tall order for a battery-powered portable device. The way around this is to solder in a 1-kΩ resistor to the voltage line (red wire) in the USB cable that you use to connect the phone to your interface board. The phone will connect, but it won't recharge itself through the connection. Thanks to *Dr. Monk's DIY Electronics Blog* for putting this information onto the blog.

The full assembly, which shows the touch-sensor board mounted on the back of a digital radio connected to the IOIO board and a Nexus One Android phone is shown in Photo 3.

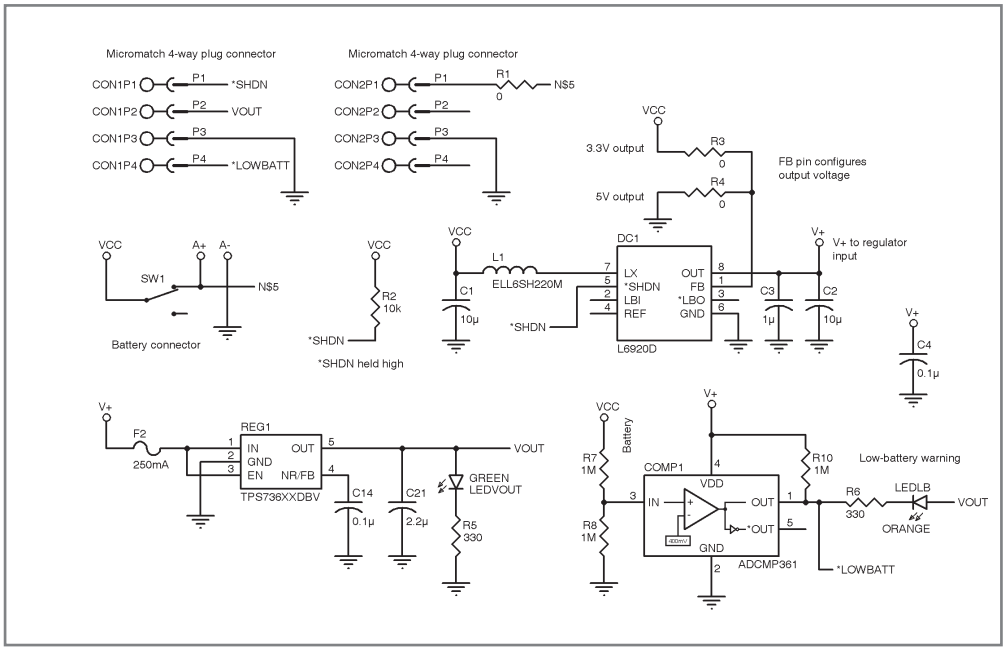

Figure 3—DC-to-DC converter board with a selectable output of 3.3 V or 5 V

board when it is directly connected to a USB port. The portable version runs from a 9-V battery that is connected to the IOIO board.

I built a small DC-to-DC converter board that clips directly onto the touch-sensor board. This is useful for demos involving only the touch sensor and LED board (see Photo 1a). Running your finger along the pins of

POWER

"Power is nothing without control." But it is also hard to control anything without power! The FTDI cable provides power to the mTouch

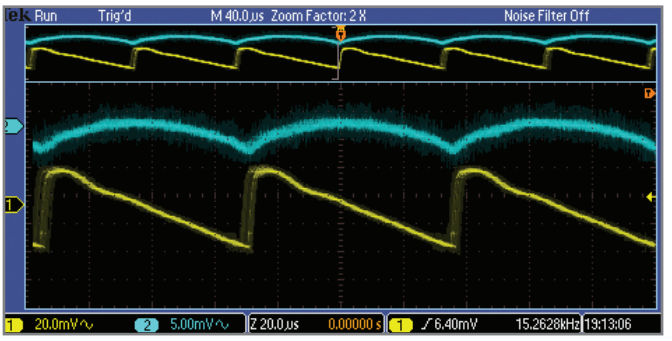

Photo 4—A 3-V DC-to-DC converter: regulated output on top, DC-DC output on bottom. Note the different traces on the scales.

the touch-sensor boards makes for a nice little light show to demonstrate your technology. The battery board runs from a single AA or AAA battery and uses an STMicroelectronics L6920D step-up converter (DC1 in Figure 3) to boost the voltage. The output of the converter can be set to either 3.3 or 5 V by connecting the feedback (FB) pin to the output (OUT) pin or to ground. This is set by populating either R3 or R4 in Figure 3 with a 0-Ω resistor. As with all DC-to-DC converters, there is an appreciable ripple on the output. Touch-sensor circuits can be sensitive to a ripple in the voltage rail, so I use the 5-V output option and an additional regulator stage to reduce this ripple (REG1 on Figure 3). Photo 4 shows the ripple from the converter and the smoother output from the 3.3-V regulator. Note the different scales on the traces. Clearly, the regulator significantly reduces the ripple. This ability is listed as power supply rejection ratio (PSRR) on the component's datasheet. During my testing, I used the smooth 5-V USB power rail available from the FTDI cable to power the device. But having the clip-on battery board enables me to show off the touch sensor and LED boards at meetings. The smoothed output from the regulator will be useful for future analog designs I have in the pipeline.

The touch sensor and LED boards are happy to run at either 3.3 V from the regulated DC-to-DC converter or 5 V from the USB port. If you are using the IOIO board, use the 3.3-V pin on the IOIO board to connect to the touch-sensor board's +V rail.

TESTING & FUTURE WORK

The initial application of these boards was to add audio tags to the controls of a CD player, a cassette player, and a digital radio to help the visually impaired learn the layout of an unfamiliar device. When the controls are touched, an audio tag states the function (e.g., play or stop) before the button is pushed.

The devices have been tested by visually impaired volunteers who like the idea. As expected, the technology is most useful with the more complex digital radio as it has the most controls. Wiring up a complete computer or music keyboard is a practical idea as the boards can be daisy chained together and the Android device has the hardware to cope. The Android code can easily be extended to cope with the extra channels.

My hope is that manufacturers will adopt the concept of adding touch sensors and audio tags to the controls of devices geared toward the visually impaired community. The Roberts Symphony CD player I used for testing is one of the few CD players with high visibility and tactile buttons manufactured specifically for the visually impaired. But, for people who can't see the buttons, the added audio tags make it even easier to use.

Author's note: The Lancaster University Faculty of Science and Technology supported this work through a faculty grant. Andrew Greaves and Robert Hardy of InfoLab21 supplied invaluable help by coaching my Java.

Matt Oppenheim (matt.oppenheim@gmail.com) is a geophysicist for Polarcus Ltd. who works onboard seismic survey ships. He spends his time onshore at InfoLab21, Lancaster University, working in the Embedded Interactive Systems group. By nature a hardware specialist, Matt realizes that software is a necessary evil.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2012/262.

RESOURCES

Android Developers, http://developer.android.com/index.html.

J. Bachiochi, "Extend and Isolate the I²C BUS," *Circuit Cellar* 233, 2009.

Eclipse Foundation, Inc., www.eclipse.org.

Dr. Monk's DIY Electronics Blog, "Open-Source Hardware, Arduino, IOIO, General DIY Electronic Construction, Reviews, Projects, How-Tos, and Recipes," 2011, http://srmonk.blogspot.com/2011/06/android-open-accessory-without charging.html.

M. Oppenheim, "Multichannel Touch Sensors: Implement Scalable Capacitive Touch Sensing," *Circuit Cellar* 234, 2010.

———, InfoLab21, Lancaster University, "Touch and Speech Enhanced Cassette Player to Help the Visually Impaired," www.youtube.com/watch?v=Z98IRhTjz30.

Oracle, "The Java Tutorials," 2011, http://download.oracle.com/javase/tutorial.

Vogella Blog, "Android Tutorials," www.vogella.de/android.html.

E. Weddington, "Bit Flipping Tutorial: An Uncomplicated Guide to Controlling MCU Functionality," *Circuit Cellar* 180, 2005.

SOURCES

Eagle PCB Software
CadSoft, Inc. | www.cadsoftusa.com

USB TTL Serial cables
Future Technology Devices International (FTDI) Ltd. | www.ftdichip.com

Total Recorder software
High Criteria | www.highcriteria.com

mTouch Capacitive touch evaluation kit, PIC16F727 microcontroller, MPLAB ICD 2 in-circuit debugger/programmer, MCP23017 I/O expander, PICkit2 programmer/debugger, and PIC24FJ256DA206 microcontroller
Microchip Technology, Inc. | www.microchip.com

mikroC PRO compiler
MikroElektronika | www.mikroe.com

ISD ChipCorder speech recorder
Nuvoton Technology Corp. | www.nuvoton.com

IOIO board
SparkFun Electronics | www.sparkfun.com

L6920D Step-up converter
STMicroelectronics | www.st.com

Micro-MaTch Connectors
Tyco Electronics Corp. | www.te.com