A Touch-Sense Technique that works on any MCU

(without special on-chip hardware support)

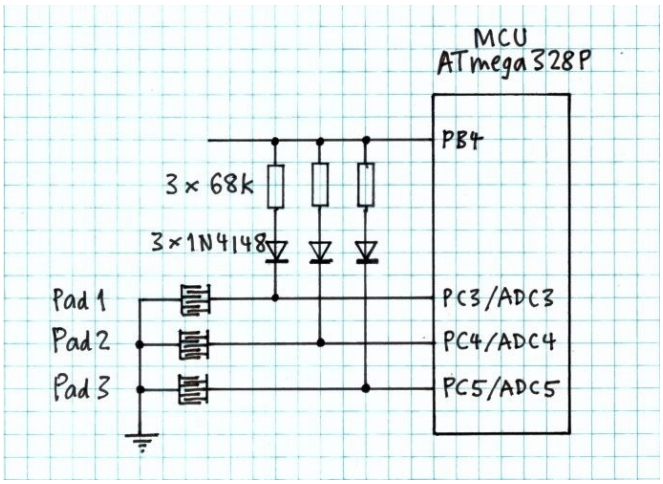
by Michael J. Bauer, B.E.(Elec)

Overview

Some micro-controller devices have on-chip hardware to support reading of capacitive touch-pads. These devices provide "automatic" measurement of capacitance between an I/O pin and earth (GND). However, a change in capacitance on an (analog) input pin can be detected quite easily without any special on-chip wizardry. The technique presented here relies on the measurement of capacitor (touch-pad) charge rate, or to be more precise, on the measurement of the voltage on the capacitor after a fixed charge time. While a pad is touched, its effective capacitance is higher; the charge rate will be lower and so the end voltage will be lower.

Principle

The micro-controller must have I/O pins which can be configured as either digital (GPIO) or analog (ADC inputs). One such pin is required for each touch input. The number of touch-pads may be increased above the number of available ADC inputs by using an external analog multiplexer IC. One digital output is required to provide a "current source". Actually, it's a voltage source, but a high-value resistor (one for each touch input) turns it into a current source! The schematic below shows 3 touch-pads wired to a microcontroller.

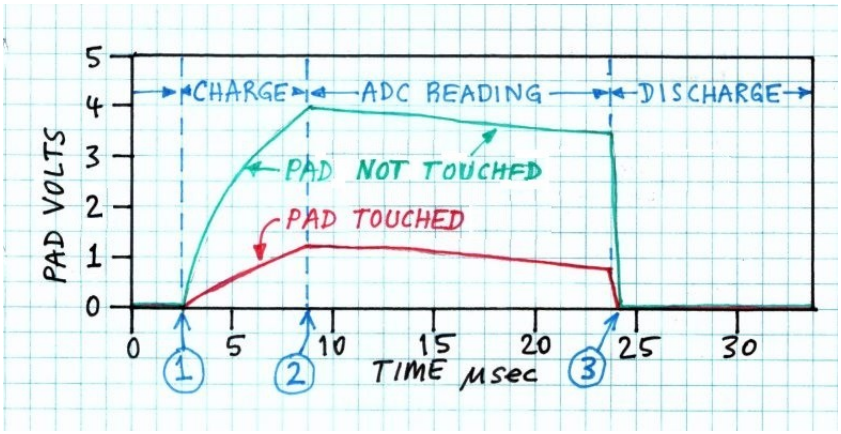


Touch-pads may be etched into a PCB with signal and ground traces intermeshed to form a capacitor. When touched with a finger, the capacitance between the signal and ground traces will increase. Alternatively, touch-pads may be just bare metal objects such as a screw head, rivet or piece of metallic tape. The human body acts a large capacitance to earth when it contacts a touch-pad. But touch-sense is more effective if the body is connected somehow to the micro-controller system ground (GND).

Here's how it works… Initially, the "current source" drive (PB4) is switched off and the touch-pad pins (PC3, PC4, PC5) are configured as outputs and set low (0V). This discharges the pads. Each pad is "serviced" in turn, one by one. At the start of the service routine, the pad I/O pin is configured as an analog input. The pad is charged from the current source (resistor and diode) for a fixed time duration, typically a few microseconds.

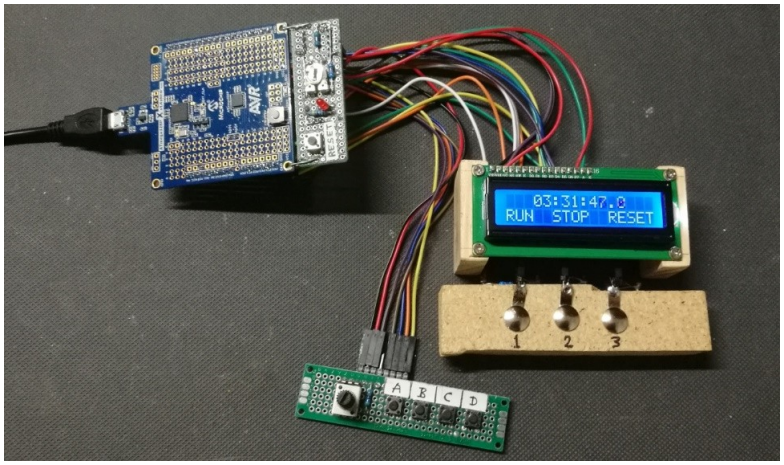
At the end of the charge time, the current source is switched off. The diode prevents discharge while the pad voltage is measured by the ADC. When the ADC conversion is complete, the result is read out and the pad I/O pin is re-configured as a digital output set LOW to discharge the pad.

This diagram shows the pad signal voltage versus time during the reading sequence…



The green trace is the pad voltage with the pad not touched. The pad capacitance is very low, so it charges rapidly and reaches a relatively high voltage. At timing point 2, the current source is switched off and the ADC conversion sequence is started. There will be a slow discharge due to current leakage in the diode and in the ADC inputs. This is not a problem because the ADC samples the voltage at the start of conversion (point 2) and holds it constant during conversion. At the end of the conversion sequence (point 3), the ADC reading is read out and saved in an array. Finally, the touch-pad is discharged, in preparation for a subsequent reading.

Depending on your specific application requirements, the interval between calling the touch-pad "service routine" may be anywhere in the range 100 microseconds up to 5 milliseconds, or more. Execution time of the service routine itself is typically under 30 microseconds for each touch input. Thus, the processing overhead to service the touch-pads is very low. Interrupts must be disabled during execution of the service routine because the timing is critical. For most embedded applications, a delay of under 30µs occurring once every millisecond (more or less) would not be regarded as "process blocking".



Test and demo setup – AVR 'X-mini' board

Test and Demo App

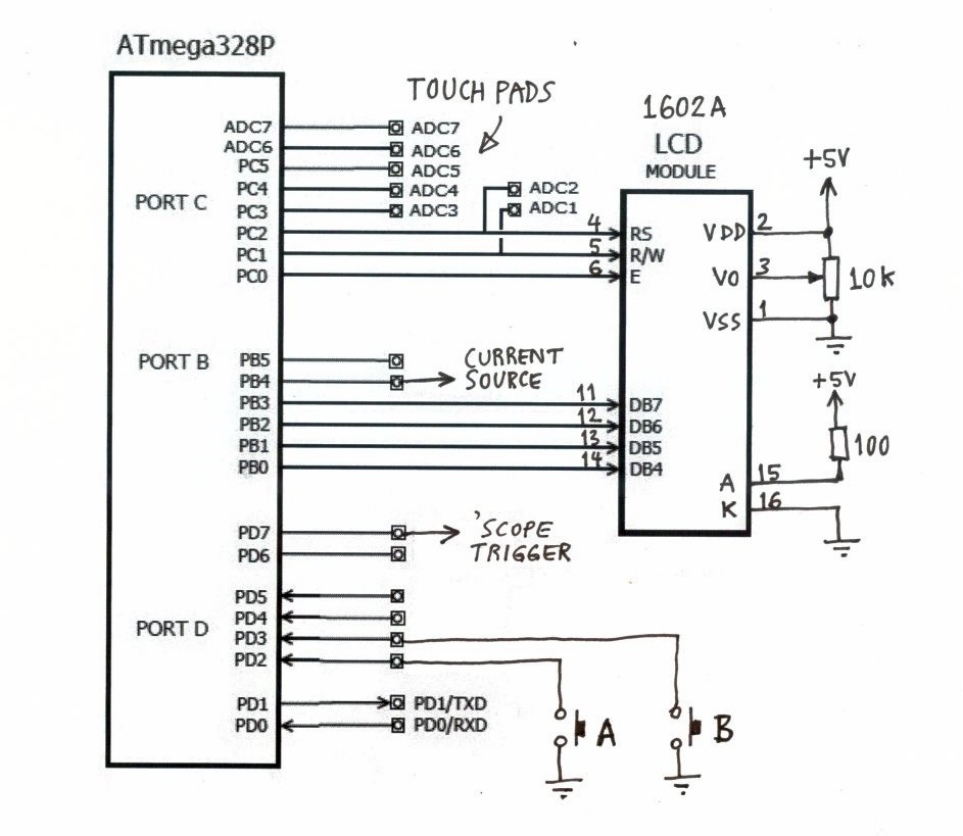
The touch-sense technique has been tested successfully on a "low-end" 8-bit AVR micro-controller – Atmel ATmega328P – as found in many popular development boards including the Arduino Uno R3 and Nano, and Microchip's ATmega328P (AVR) 'X-Mini' board. I chose the AVR "X-Mini" because it has an on-board programming tool, it is very cheap (~A$15) and readily available from major suppliers, but primarily because I already had one in my junk box!

Also, I much prefer Microchip/Atmel Studio IDE for AVR and SAM, compared to the Arduino IDE. The Test & Demo program could be migrated to Arduino IDE, but the MCU pin allocations (for the 1602A LCD module in particular) are incompatible with common Arduino code libraries. Hence, you would need to extract the source code for some peripheral functions from the AVR X-mini library to import into your Arduino "sketch".

If you want to run the Test & Demo program on an Arduino Uno or Nano board, the pre-built object code (hex file) can be programmed into the MCU without needing Microchip/Atmel Studio, or Arduino IDE, and without any additional programming tool. (See Appendix A.)

Whatever hardware platform and software development tools you are using for your application, the touch-sense technique can be understood by examining the code in the Test & Demo program, in particular the Touch-Sense "Service Routine". Just read the comments!

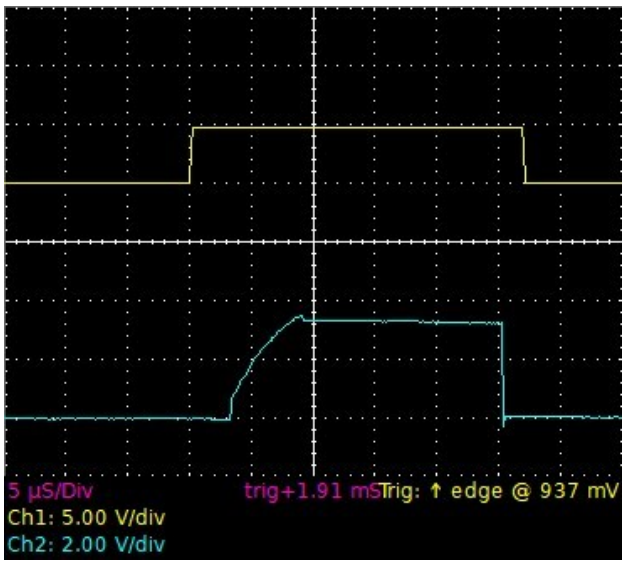
If you want to replicate the Test & Demo application on a compatible AVR hardware platform, connect the LCD and other peripherals as shown in this schematic:



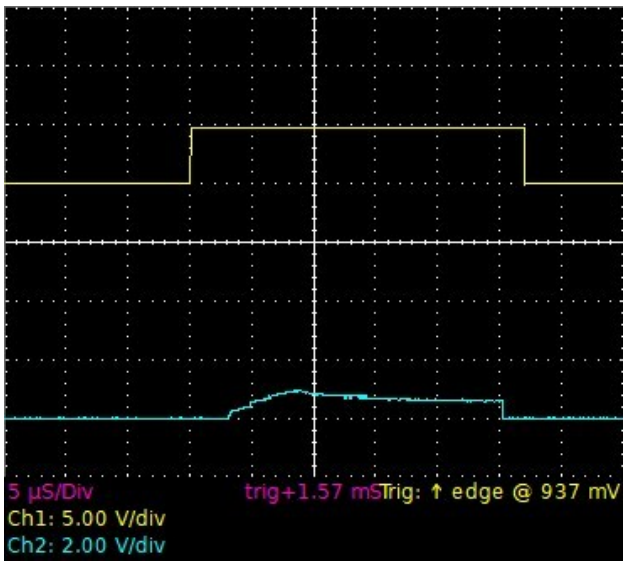
Two push-button switches (labelled 'A' and 'B') are provided for user input in case the touch-pads don't work initially. (The on/off voltage threshold may need adjustment.) The touch-pads are wired as shown on page 1. If your application doesn't need the LCD or I2C bus (PC4, PC5), then all (8) Port C pins may be used for touch-pads or other analog inputs.

Oscilloscope screen captures obtained with the Test & Demo app

The top (yellow) trace is a test-point output signal. This signal is set HIGH at the start of the service routine and LOW at the end, just before the function returns. The test-point output serves two purposes: (1) 'scope trigger and (2) measurement of SR execution time.



Result when pad not touched



Result when pad touched

The bottom (blue) trace is the touch-pad signal (ADC input). In these tests, the charge time was set to 6 microseconds. The current-source resistor value (68k) and charge time were chosen such that the pad signal reaches almost (but not quite) the ADC reference voltage (+5V) when the pad is not touched. This results in optimum touch sensitivity.

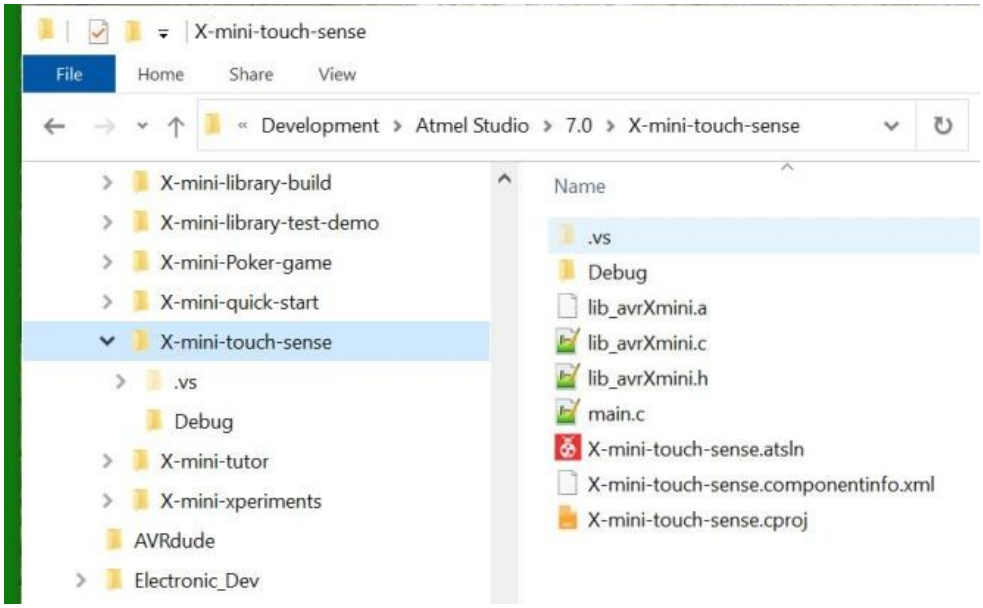
The screen on the right shows what happens when the pad is touched. The charge rate is much slower, so the voltage on the pad at the end of the charge time is lower. The software defines a voltage somewhere in between to be the "touch ON" threshold.

Note that the ADC conversion time is about 15 microseconds. This accounts for most of the execution time of the service routine. The ATmega328P ADC clock rate may be increased to reduce the SR execution time. There is a compromise between ADC speed and conversion accuracy, of course. However, the ADC clock rate used in the test program (2MHz) gives quite acceptable precision, so perhaps it could be increased further. (You could try 4MHz.)

Firmware Development under Microchip/Atmel Studio IDE

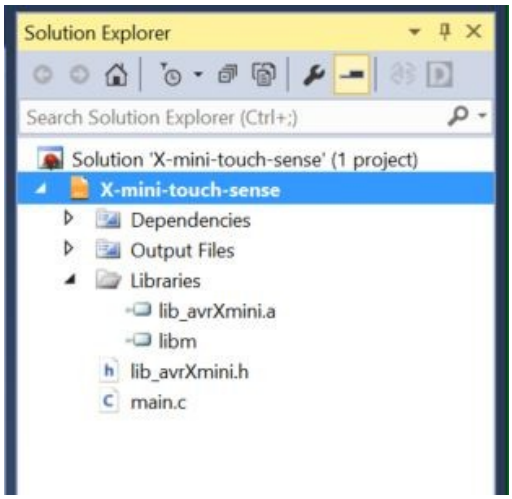
Should you decide to develop your own application using Microchip/Atmel Studio IDE, assuming you have not used it before, a good place to start is to re-build the Test & Demo program. Once you have the app running on your target platform, you can then modify and extend the source code for your own purposes.

First, download the project files (including source code) from GitHub. Create a project folder on your computer's local drive and copy all the files into that folder. The directory structure should look like this example…



From the Start Page in Atmel Studio, select "Open Existing Project". Navigate to your project folder and click on the file named "**X-mini-touch-sense.atsln**". This is easier than creating a New Project and will ensure that the project has all the required components loaded, in particular the peripheral library files, "lib_avrXmini.*".

Further down the track, if you wish to create a new project using the X-mini library, be sure to attach the library files correctly. This can be checked in the "Solution Explorer" panel on the right side of the IDE editor window, as shown here:



Download links

[Test and Demo Program - Project Files (source code, etc)]

[Microchip-Atmel Studio IDE for AVR and SAM Devices]

Appendix A – How to Program Arduino Uno or Nano in Microchip Studio

The firmware was developed using Microchip Studio for AVR and SAM Devices (formerly Atmel Studio). One of many reasons for choosing this IDE (Integrated Development Environment) instead of Arduino is that the project hardware design is not compatible with available Arduino code libraries. In particular, the 1602A LCD interface scheme (MCU I/O pin assignment) is not supported by any Arduino library (that I am aware of).

Programming the target device (ATmega328P MCU) can be achieved without Arduino IDE and without any hardware programming tool. The Uno and Nano boards have an on-board USB-serial "bridge" IC and a flash-resident AVR bootloader. A Windows PC application called "**AVRdude**" talks to the bootloader via USB to program firmware into the MCU flash memory.

Hence you need to download some files to run "**avrdude**" on Windows. The best place to download the files is GitHub, here: https://github.com/mariusgreuel/avrdude/releases. There should be 3 distribution files: "avrdude.exe", "avrdude.conf" and "avrdude.pdb". Copy these files to a new folder named "**AVRdude**" on your PC local drive, in the "root directory" (C:\).

Connect your Uno/Nano board to a USB port on your PC. Open Windows "Device Manager" utility and click on "Ports (COM & LPT)". You should see the Uno/Nano USB-serial device listed. Note the number of the associated "COM" port. Be aware that the COM port number may change from time to time. Remember to check the allocated COM port when re-connecting the Uno or Nano board to your PC.

How to create a (software) "Programming Tool" in Microchip Studio

Click in the menu "**Tools/External tools**".

You should see a dialog box asking for some parameters, as follows...

In **Title**, write: `Program Nano` or any other name you prefer.

In **Command**, write: `C:\AVRdude\avrdude.exe`

In **Arguments**, write (all on one line):

```
-C "C:\AVRdude\avrdude.conf" -p atmega328p -c arduino -P COM# -b 115200
-U flash:w:"$(ProjectDir)Debug\$(TargetName).hex":i
```

Replace COM# (in the Arguments field) with the actual COM port your Nano board is connected to, as found in Windows Device Manager. (Example: COM4)

Tick the box: "Use output window". Click OK.

Done... You should see a new option "**Program Nano**" in the **Tools** menu.

After your program code is built, it can be programmed into the Nano board simply by clicking on the item "Program Nano" in the Tools menu.

Note: Some cheap Chinese Nano board clones use a non-standard Baud rate for the serial bootloader, typically 57600 baud. If Microchip Studio outputs an error message when running the programming tool, try replacing "`115200`" with "`57600`" in the Arguments field.