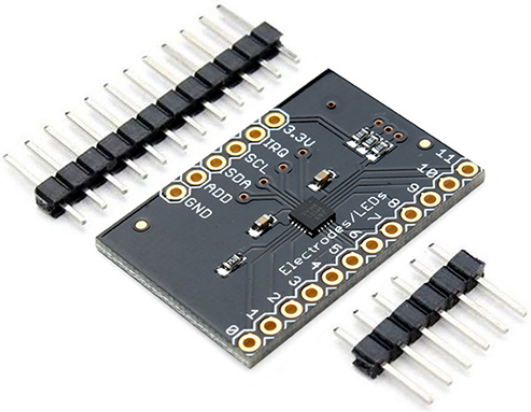
Arduino Module Capacitive Touch Controller
Model:MPR121
User's Manual



MPR121 Overview:

If you are interested in adding the 'magic' of touch to control your electronics project, a capacitive touch sensor might be the way to go. This hookup guide will show you how to use the MPR121QR2 sensor.
The MPR121QR2 is a capacitive touch sensor controller that makes it very easy to integrate capacitive touch sensing into your project. It communicates via I²C, and works by measuring the capacitance of twelve electrode points. When an object comes close to the electrode connector, the measured capacitance changes. This signals the MPR121 that something has touched a 'button'. The IC is also capable of driving LEDs or basic GPIO functionality on electrode pins 4 through 11, giving you a lot of freedom for setting up your project. The sensor works from 1.6V to 3.3V. The sensor isn't very current-hungry, drawing only around 29 µA when sampling every 16 milliseconds.

Materials:

To work through this tutorial, you are going to need one of the three versions of the MPR121 sensor:

- MPR121 Capacitive Touch Sensor Breakout Board
- Touch Shield
- MPR121 Capacitive Touch Keypad

You will also want a soldering iron, some hookup wires and a microcontroller capable of I²C communication. For our examples, we will be using an Arduino Uno. You will also need some kind of material to act as a capacitive sensing surface (also known as an electrode, which is not to be confused with the character Electrode). Generally, aluminum foil works well. However, you could also use coins, conductive paint, or copper tape.

Suggested Reading:

The MPR121 is very easy to get started using, especially with the example code. However, if you haven't worked with Arduino previously or aren't familiar with I²C communication, you should check out the tutorials below.

- What is an Arduino
- I²C Communication
- How to Use a Breadboard
- How to Solder

Capacitive Touch Sensor Breakout Board:

The Capacitive Touch Sensor Breakout Board is the most versatile option of the three MPR121 products. You can wire it up to any kind of electrode you want, and, as it is a simple breakout board, does not have a particular microcontroller footprint it favors.

The breakout board has 4 pins that need to be connected to your microcontroller at a minimum to get communication going: the power lines and the I²C lines. However, for our example, we are going to be also connecting the IRQ pin to more easily detect a change on one of the electrodes.

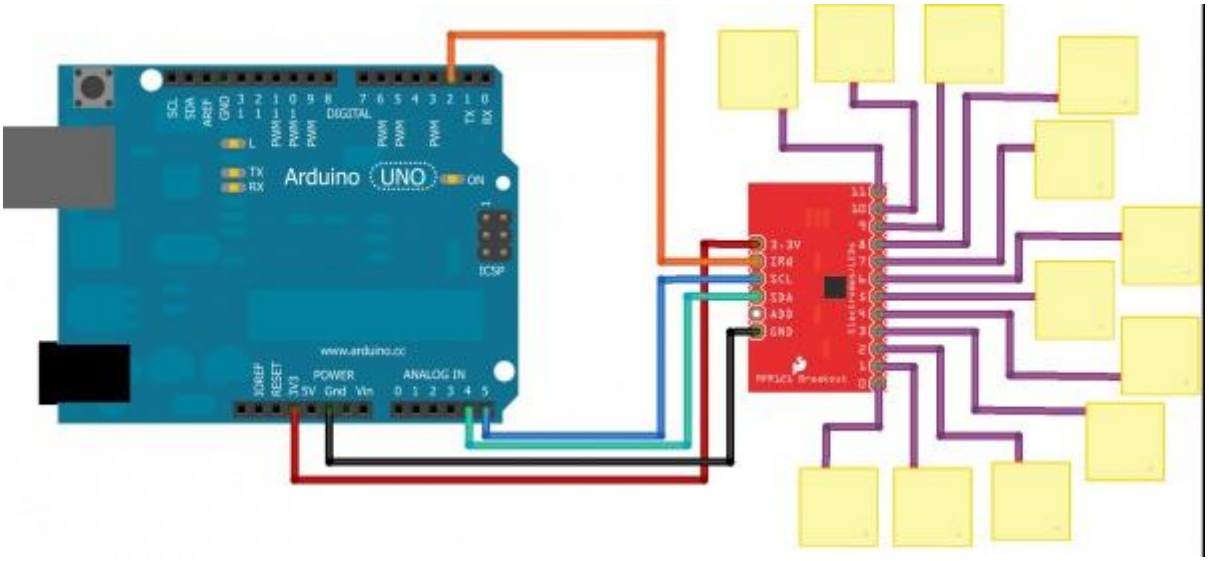
Connections:

MPR121 Breakout → Arduino Uno

- 3.3V → 3.3V
- SCL → A5
- SDA → A4
- GND → GND
- IRQ → D2

You will also want to connect the Electrode/LED pins to your electrode material you selected previously. You will want to make sure you have a good, solid connection between your material and your board, so make sure you thoroughly solder your connections.

Check out diagram below for how your connections should look. The yellow squares represent whatever material you decide to use for your electrodes.



Communicating with the Breakout Board:

To communicate with your breakout board, you will need the Arduino sketch available as a zip file here. Alternatively, you can also find the most up-to-date firmware for working with the breakout board available on GitHub. Let's take a look and see exactly what the code is doing.

```
COPY CODE

    #include "mpr121.h"
    #include <Wire.h>

    int irqpin = 2;  // Digital 2
    boolean touchStates[12]; //to keep track of the previous touch states
```

In this first section of the code, the MPR121 library and the Wire library are initialized. The Wire library makes I²C communication easy to use on the Arduino. The sketch also defines digital pin 2 as the IRQ pin connection, and creates 12 instances of the boolean variable touchStates.

For the second section of the code, we define the `irqpin` as an INPUT, telling the Arduino to monitor the digital signal coming in over that pin. Serial communication is also started at 9600 bps, s well as the Wire and mpr121 libraries.

```
COPY CODE

    void setup(){
        pinMode(irqpin, INPUT);
        digitalWrite(irqpin, HIGH); //enable pullup resistor

        Serial.begin(9600);
        Wire.begin();

        mpr121_setup();
    }
```

The main loop of the code is incredibly simple, as it only calls a single function.

```
COPY CODE

    void loop(){
        readTouchInputs();
    }
```

The function is actually described in the next section of the code. The Arduino requests the electrode states from the sensor in the first section, and the least significant bits and most significant bits are defined for the sensor.

```
COPY CODE

void readTouchInputs(){
  if(!checkInterrupt()){

    //read the touch state from the MPR121
    Wire.requestFrom(0x5A,2);

    byte LSB = Wire.read();
    byte MSB = Wire.read();
```

```
    uint16_t touched = ((MSB << 8) | LSB); //16bits that make up the touch states


    for (int i=0; i < 12; i++){  // Check what electrodes were pressed
      if(touched & (1<<i)){

        if(touchStates[i] == 0){
          //pin i was just touched
          Serial.print("pin ");
          Serial.print(i);
          Serial.println(" was just touched");

        }else if(touchStates[i] == 1){
          //pin i is still being touched
        }

        touchStates[i] = 1;
      }else{
        if(touchStates[i] == 1){
          Serial.print("pin ");
          Serial.print(i);
          Serial.println(" is no longer being touched");

          //pin i is no longer being touched
       }

        touchStates[i] = 0;
      }

    }

  }
}
```

The Arduino scans each electrode and prints out a message over serial if an electrode is triggered as being touched. The Arduino will then print out a message as soon as the electrode is no longer being touched.

The last major section of the code defines the threshold values for each electrode. Each electrode must have a touch threshold and a release threshold for the Arduino to compare the current state of the electrode.

```
COPY CODE

void mpr121_setup(void){

  set_register(0x5A, ELE_CFG, 0x00);

  // Section A - Controls filtering when data is > baseline.
  set_register(0x5A, MHD_R, 0x01);
  set_register(0x5A, NHD_R, 0x01);
  set_register(0x5A, NCL_R, 0x00);
  set_register(0x5A, FDL_R, 0x00);

  // Section B - Controls filtering when data is < baseline.
  set_register(0x5A, MHD_F, 0x01);
```

```
  set_register(0x5A, NHD_F, 0x01);
  set_register(0x5A, NCL_F, 0xFF);
  set_register(0x5A, FDL_F, 0x02);

  // Section C - Sets touch and release thresholds for each electrode
  set_register(0x5A, ELE0_T, TOU_THRESH);
  set_register(0x5A, ELE0_R, REL_THRESH);

  set_register(0x5A, ELE1_T, TOU_THRESH);
  set_register(0x5A, ELE1_R, REL_THRESH);

  set_register(0x5A, ELE2_T, TOU_THRESH);
  set_register(0x5A, ELE2_R, REL_THRESH);

  set_register(0x5A, ELE3_T, TOU_THRESH);
  set_register(0x5A, ELE3_R, REL_THRESH);

  set_register(0x5A, ELE4_T, TOU_THRESH);
  set_register(0x5A, ELE4_R, REL_THRESH);

  set_register(0x5A, ELE5_T, TOU_THRESH);
  set_register(0x5A, ELE5_R, REL_THRESH);

  set_register(0x5A, ELE6_T, TOU_THRESH);
  set_register(0x5A, ELE6_R, REL_THRESH);

  set_register(0x5A, ELE7_T, TOU_THRESH);
  set_register(0x5A, ELE7_R, REL_THRESH);

  set_register(0x5A, ELE8_T, TOU_THRESH);
  set_register(0x5A, ELE8_R, REL_THRESH);

  set_register(0x5A, ELE9_T, TOU_THRESH);
  set_register(0x5A, ELE9_R, REL_THRESH);

  set_register(0x5A, ELE10_T, TOU_THRESH);
  set_register(0x5A, ELE10_R, REL_THRESH);

  set_register(0x5A, ELE11_T, TOU_THRESH);
  set_register(0x5A, ELE11_R, REL_THRESH);

  // Section D
  // Set the Filter Configuration
  // Set ESI2
  set_register(0x5A, FIL_CFG, 0x04);

  // Section E
  // Electrode Configuration
  // Set ELE_CFG to 0x00 to return to standby mode
  set_register(0x5A, ELE_CFG, 0x0C);  // Enables all 12 Electrodes


  // Section F
```

```
  // Enable Auto Config and auto Reconfig
  /*set_register(0x5A, ATO_CFG0, 0x0B);
  set_register(0x5A, ATO_CFGU, 0xC9);  // USL = (Vdd-0.7)/vdd*256 = 0xC9 @3.3V   set_register(0
x5A, ATO_CFGL, 0x82);  // LSL = 0.65*USL = 0x82 @3.3V
  set_register(0x5A, ATO_CFGT, 0xB5);*/  // Target = 0.9*USL = 0xB5 @3.3V

  set_register(0x5A, ELE_CFG, 0x0C);

}
```

It looks like a lot of code, but it simply repeating the procedure of setting the threshold values for each electrode pin.

The last two functions in the example sketch simply check the status of the `irqpin` to determine if the IC is signaling that an electrode has been touched. The very last function `set_register` simply runs the Arduino through the standard steps in the Wire library to write the registers to the IC.

```
COPY CODE

boolean checkInterrupt(void){
  return digitalRead(irqpin);
}


void set_register(int address, unsigned char r, unsigned char v){
    Wire.beginTransmission(address);
    Wire.write(r);
    Wire.write(v);
    Wire.endTransmission();
}
```