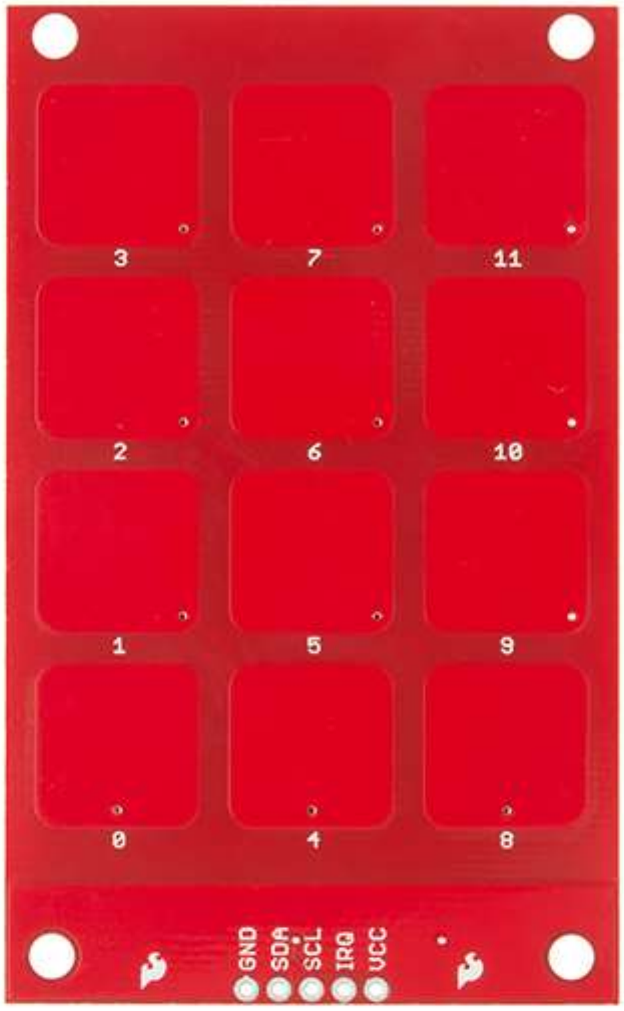
Capacitive Touch Keypad



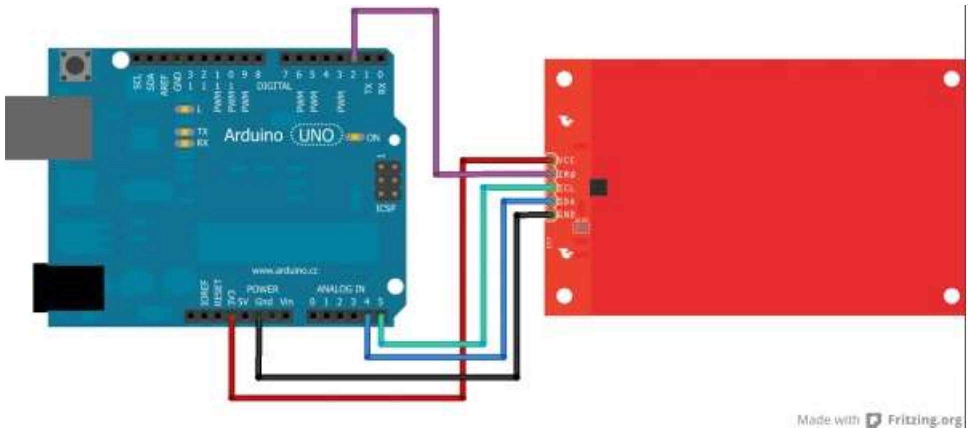
The keypad has 5 lines that need to be connected to your microcontroller, including the power lines, the IRQ line and the I^2C lines.

Connections

MPR121 Keypad → Arduino Uno

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

Check out the Fritzing diagram below to verify your connections.



This diagram shows the bottom side of the keypad, in order to show you the location of the I^{2}C address jumper. In this diagram, the jumper is set to 0, but if you need to have 2 keypads on the same I^{2}C bus, you can change one of them to be set to 1.

Now that you've got your keypad hooked up properly, let's get communicating with it!

Communicating with the Keypad

In order to communicate with your keypad, you'll want to download the example sketch available here. Alternatively, you can check for the most up-to-date firmware available in the GitHub repository.

This sketch is designed to allow the keypad to function as a phone keypad. Check out the pin mapping below to see how the numbered pads will print over the Serial terminal.

Actual Keypad → Phone Keypad Printout

- 3 → 1
- 7 → 2
- 11 → 3
- 2 → 4
- 6 → 5
- 10 → 6
- 1 → 7
- 5 → 8
- 9 → 9
- 0 → *
- 4 → 0
- 8 → #

This is defined in the first section of the code, shown below. The I^{2}C address is also defined, along with setting digital pin 2 to be assigned as the IRQ pin connection point.

```
#include "mpr121.h"
#include "i2c.h"

#define MPR121_R 0xB5     // ADD pin is grounded
#define MPR121_W 0xB4    // So address is 0x5A
```

```
#define PHONE_DIGITS 10  // 10 digits in a phone number

// Match key inputs with electrode numbers
#define STAR 0
#define SEVEN 1
#define FOUR 2
#define ONE 3
#define ZERO 4
#define EIGHT 5
#define FIVE 6
#define TWO 7
#define POUND 8
#define NINE 9
#define SIX 10
#define THREE 11

int irqpin = 2;  // D2

uint16_t touchstatus;
char phoneNumber[PHONE_DIGITS];
```

The second section of the code is the basic initialization of the serial communication port at 9600 bps, as well as setting the IRQ pin as an input on the Arduino. It also starts the configuraton of the MPR121 IC.

```
void setup()
{
  pinMode(irqpin, INPUT);
  digitalWrite(irqpin, HIGH);

  Serial.begin(9600);
  DDRC |= 0b00010011;
  PORTC = 0b00110000;  // Pull-ups on I2C Bus
  i2cInit();

  delay(100);
  mpr121QuickConfig();
}
```

The main loop in the code simply scans the MPR121 electrodes and looks for a phone number to be entered on the keypad. This is the function `getPhoneNumber()`. The code then prints out the phone number dialed over the serial monitor.

```
void loop()
{
  getPhoneNumber();

  Serial.print("\nDialing... ");
  for (int i=0; i<PHONE_DIGITS; i++)
    Serial.print(phoneNumber[i]);

  while(1)
    ;
}
```

As you can see in the function loop for `getPhoneNumber()`, the Arduino checks the `touchstatus` register for each electrode and prints out the assigned value for any electrodes that are registering a touch. It also includes an error if multiple buttons are pressed simultaneously, printing over the serial monitor to tell the user to touch only one button.

```
void getPhoneNumber()
{
  int i = 0;
  int touchNumber;

  Serial.println("Please Enter a phone number...");

  while(i<PHONE_DIGITS)
  {
    while(checkInterrupt())
      ;
    touchNumber = 0;

    touchstatus = mpr121Read(0x01) << 8;
    touchstatus |= mpr121Read(0x00);

    for (int j=0; j<12; j++)  // Check how many electrodes were pressed
    {
      if ((touchstatus & (1<<j)))
        touchNumber++;
    }

    if (touchNumber == 1)
    {
      if (touchstatus & (1<<STAR))
        phoneNumber[i] = '*';
      else if (touchstatus & (1<<SEVEN))
        phoneNumber[i] = '7';
      else if (touchstatus & (1<<FOUR))
        phoneNumber[i] = '4';
      else if (touchstatus & (1<<ONE))
        phoneNumber[i] = '1';
      else if (touchstatus & (1<<ZERO))
        phoneNumber[i] = '0';
      else if (touchstatus & (1<<EIGHT))
        phoneNumber[i] = '8';
      else if (touchstatus & (1<<FIVE))
        phoneNumber[i] = '5';
      else if (touchstatus & (1<<TWO))
        phoneNumber[i] = '2';
      else if (touchstatus & (1<<POUND))
        phoneNumber[i] = '#';
      else if (touchstatus & (1<<NINE))
        phoneNumber[i] = '9';
      else if (touchstatus & (1<<SIX))
        phoneNumber[i] = '6';
      else if (touchstatus & (1<<THREE))
        phoneNumber[i] = '3';

      Serial.print(phoneNumber[i]);
      i++;
    }
    else if (touchNumber == 0)
      ;
    else
      Serial.println("Only touch ONE button!");
  }
}
```

The next section of the code simply steps through the I^{2}C communication to read data from the MPR121 IC.

```
byte mpr121Read(uint8_t address)
{
  byte data;

  i2cSendStart();
  i2cWaitForComplete();

  i2cSendByte(MPR121_W);    // write 0xB4
  i2cWaitForComplete();

  i2cSendByte(address); // write register address
  i2cWaitForComplete();

  i2cSendStart();

  i2cSendByte(MPR121_R);    // write 0xB5
  i2cWaitForComplete();
  i2cReceiveByte(TRUE);
  i2cWaitForComplete();

  data = i2cGetReceivedByte();  // Get MSB result
  i2cWaitForComplete();
  i2cSendStop();

  cbi(TWCR, TWEN);  // Disable TWI
  sbi(TWCR, TWEN);  // Enable TWI

  return data;
}
```

The function `mpr121Write()` is then defined, which again, steps through the I^{2}C process of writing commands to the MPR121 sensor.

```
void mpr121Write(unsigned char address, unsigned char data)
{
  i2cSendStart();
  i2cWaitForComplete();

  i2cSendByte(MPR121_W);// write 0xB4
  i2cWaitForComplete();

  i2cSendByte(address);    // write register address
  i2cWaitForComplete();

  i2cSendByte(data);
  i2cWaitForComplete();

  i2cSendStop();
}
```

The `mpr121QuickConfig()` function is then defined. In this function, all 12 of the electrodes are enabled, and the touch and release thresholds for all of the sensors are set. The filtering registers are also configured.

```
void mpr121QuickConfig(void)
{
  // Section A
  // This group controls filtering when data is > baseline.
  mpr121Write(MHD_R, 0x01);
  mpr121Write(NHD_R, 0x01);
  mpr121Write(NCL_R, 0x00);
  mpr121Write(FDL_R, 0x00);

  // Section B
  // This group controls filtering when data is < baseline.
  mpr121Write(MHD_F, 0x01);
  mpr121Write(NHD_F, 0x01);
  mpr121Write(NCL_F, 0xFF);
  mpr121Write(FDL_F, 0x02);

  // Section C
  // This group sets touch and release thresholds for each electrode
  mpr121Write(ELE0_T, TOU_THRESH);
  mpr121Write(ELE0_R, REL_THRESH);
  mpr121Write(ELE1_T, TOU_THRESH);
  mpr121Write(ELE1_R, REL_THRESH);
  mpr121Write(ELE2_T, TOU_THRESH);
  mpr121Write(ELE2_R, REL_THRESH);
  mpr121Write(ELE3_T, TOU_THRESH);
  mpr121Write(ELE3_R, REL_THRESH);
  mpr121Write(ELE4_T, TOU_THRESH);
  mpr121Write(ELE4_R, REL_THRESH);
  mpr121Write(ELE5_T, TOU_THRESH);
  mpr121Write(ELE5_R, REL_THRESH);
  mpr121Write(ELE6_T, TOU_THRESH);
  mpr121Write(ELE6_R, REL_THRESH);
  mpr121Write(ELE7_T, TOU_THRESH);
  mpr121Write(ELE7_R, REL_THRESH);
  mpr121Write(ELE8_T, TOU_THRESH);
  mpr121Write(ELE8_R, REL_THRESH);
  mpr121Write(ELE9_T, TOU_THRESH);
  mpr121Write(ELE9_R, REL_THRESH);
  mpr121Write(ELE10_T, TOU_THRESH);
  mpr121Write(ELE10_R, REL_THRESH);
  mpr121Write(ELE11_T, TOU_THRESH);
  mpr121Write(ELE11_R, REL_THRESH);

  // Section D
  // Set the Filter Configuration
  // Set ESI2
  mpr121Write(FIL_CFG, 0x04);

  // Section E
  // Electrode Configuration
  // Enable 6 Electrodes and set to run mode
  // Set ELE_CFG to 0x00 to return to standby mode
  mpr121Write(ELE_CFG, 0x0C);    // Enables all 12 Electrodes
  //mpr121Write(ELE_CFG, 0x06);     // Enable first 6 electrodes

  // Section F
  // Enable Auto Config and auto Reconfig
  /*mpr121Write(ATO_CFG0, 0x0B);
  mpr121Write(ATO_CFGU, 0xC9);  // USL = (Vdd-0.7)/vdd*256 = 0xC9 @3.3V
mpr121Write(ATO_CFGL, 0x82);  // LSL = 0.65*USL = 0x82 @3.3V
  mpr121Write(ATO_CFGT, 0xB5);*/    // Target = 0.9*USL = 0xB5 @3.3V
}
```

The final section of the sketch simply reads the IRQ pin for an indication that an electrode has been pressed. It then returns either a 1 or 0 as the value for `checkInterrupt()` variable.

```
byte checkInterrupt(void)
{
  if(digitalRead(irqpin))
    return 1;

  return 0;
}
```

Now that you know how the keypad works and communicates with the Arduino, you can start customizing your project and going further.