ROBOTICS & CNC / ROBOTICS

Lock-style Solenoid - 12VDC

PRODUCT ID: 1512

DESCRIPTION -

Solenoids are basically electromagnets: they are made of a big coil of copper wire with an armature (a slug of metal) in the middle. When the coil is energized, the slug is pulled into the center of the coil. This makes the solenoid able to pull from one end.

This solenoid in particular is nice and strong, and has a slug with a slanted cut and a good mounting bracket. It's basically an electronic lock, designed for a basic cabinet or safe or door. Normally the lock is active so you can't open the door because the solenoid slug is in the way. It does not use any power in this state. When 9-12VDC is applied, the slug pulls in so it doesn't stick out anymore and the door can be opened.

The solenoids come with the slanted slug as shown above, but you can open it with the two Phillips-head screws and turn it around so its rotated 90, 180 or 270 degrees so that it matches the door you want to use it with.

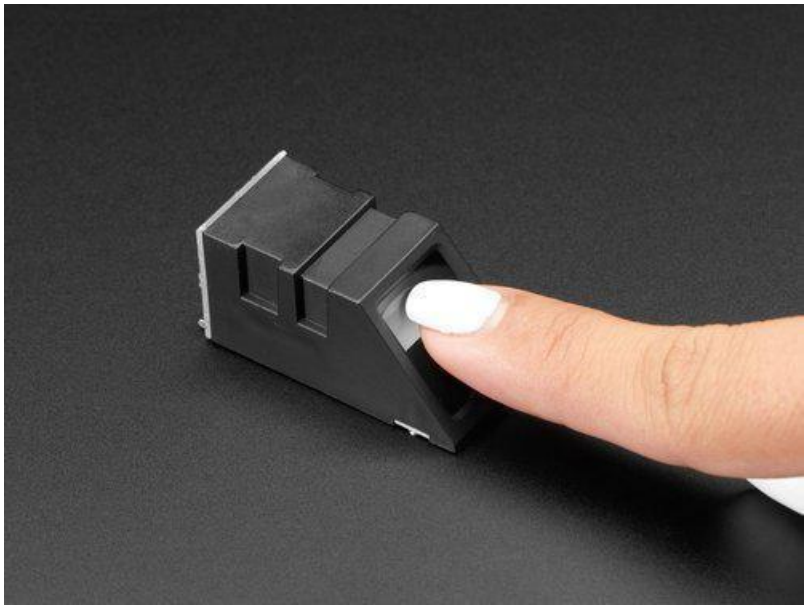
To drive a solenoid you will a power transistor and a diode, check this diagram for how to wire it to an Arduino or other microcontroller. You will need a fairly good power supply to drive a solenoid, as a lot of current will rush into the solenoid to charge up the electro-magnet, about 500mA, so don't try to power it with a 9V

battery!

adafruit learning system

Adafruit Optical Fingerprint Sensor

Created by lady ada



Last updated on 2018-11-14 05:50:07 PM UTC

Guide Contents

Overview

Secure your project with biometrics - this all-in-one optical fingerprint sensor will make adding fingerprint detection and verification super simple. These modules are typically used in safes - there's a high powered DSP chip that does the image rendering, calculation, feature-finding and searching. Connect to any microcontroller or system with TTL serial, and send packets of data to take photos, detect prints, hash and search. You can also enroll new fingers directly - up to 162 finger prints can be stored in the onboard FLASH memory.

We like this particular sensor because not only is it easy to use, it also comes with fairly straight-forward Windows software that makes testing the module simple - you can even enroll using the software and see an image of the fingerprint on your computer screen. But, of course, we wouldn't leave you a datasheet and a "good luck!" - we wrote a full Arduino library so that you can get running in under 10 minutes. The library can enroll and search so its perfect for any project (https://adafru.it/aRz). We've also written a detailed tutorial on wiring and use (https://adafru.it/clz). This is by far the best fingerprint sensor you can get.

- **Supply voltage:** 3.6 - 6.0VDC
- **Operating current:** 120mA max
- **Peak current:** 150mA max
- **Fingerprint imaging time:** <1.0 seconds
- **Window area:** 14mm x 18mm
- **Signature file:** 256 bytes
- **Template file:** 512 bytes
- **Storage capacity:** 162 templates
- **Safety ratings** (1-5 low to high safety)
- **False Acceptance Rate:** <0.001% (Security level 3)
- **False Reject Rate:** <1.0% (Security level 3)
- **Interface:** TTL Serial
- **Baud rate:** 9600, 19200, 28800, 38400, 57600 (default is 57600)
- **Working temperature rating:** -20C to +50C
- **Working humidy:** 40%-85% RH

- **Full Dimensions:** 56 x 20 x 21.5mm
- **Exposed Dimensions** (when placed in box): 21mm x 21mm x 21mm triangular
- **Weight:** 20 grams

Enrolling vs. Searching

There are basically two requirements for using the optical fingerprint sensor. First is you'll need to **enroll** fingerprints - that means assigning ID #'s to each print so you can query them later. Once you've enrolled all your prints, you can easily 'search' the sensor, asking it to identify which ID (if any) is currently being photographed.

You can enroll using the Windows software (easiest and neat because it shows you the photograph of the print) or with the Arduino sketch (good for when you don't have a Windows machine handy or for on-the-road enrolling).

Enrolling New Users with Windows

The easiest way to enroll a new fingerprint is to use the Windows software. The interface/test software is unfortunately windows-only *but* you only need to use it once to enroll, to get the fingerprint you want stored in the module.

First up, you'll want to connect the sensor to the computer via a USB-serial converter. The easiest way to do this is to connect it directly to the USB/Serial converter in the Arduino. To do this, you'll need to upload a 'blank sketch' this one works well for "traditional" Arduinos, like the Uno and the Mega:

```
//    this sketch will allow you to bypass the Atmega chip
//    and connect the fingerprint sensor directly to the USB/Serial
//    chip converter.

//    Red connects to +5V
//    Black connects to Ground
//    White goes to Digital 0
//    Green goes to Digital 1

void setup() {}
void loop() {}
```

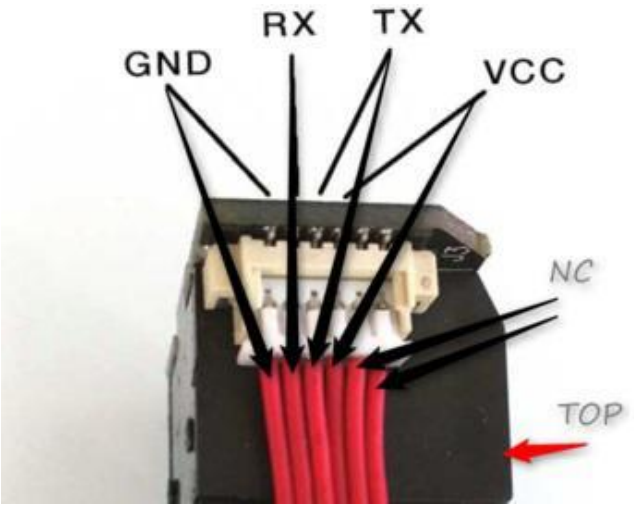
> The "blank" sketch won't work for "native USB" based Arduinos like the Leonardo, Micro, Zero, etc! Use the Leo_passthru sketch instead!

If you're using a Leonardo, Micro, Yun, Zero, or other native-USB device like ATSAMD21 or ATmega32U4-based controller, use the Leo_passthru sketch instead of the "blank" sketch.

```
 //Leo_passthru
 //    Allows Leonardo to pass serial data between fingerprint reader and Windows.
//
 //    Red connects to +5V
 //    Black connects to Ground
 //    Green goes to Digital 0
 //    White goes to Digital 1

 void setup() {
    //    put your setup code here, to run once:
    Serial1.begin(57600); Serial.begin(57600);

 }

 void loop() {

    while (Serial.available())
       Serial1.write(Serial.read());
    while (Serial1.available())
       Serial.write(Serial1.read());
 }
```

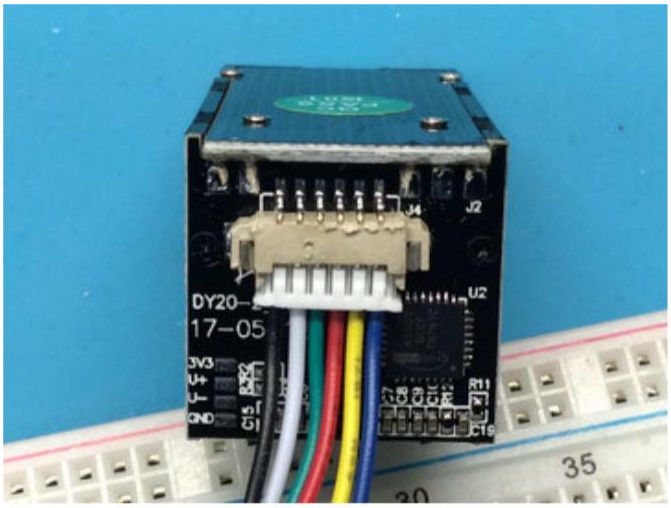
Wire up the sensor as described in the sketch comments **after** uploading the sketch. Since the sensor wires are so thin and short, we stripped the wire a bit and melted some solder on so it made better contact but you may want to solder the wires to header or similar if you're not getting good contact. When you plug in the power, you may see the LED

blink to indicate the sensor is working.

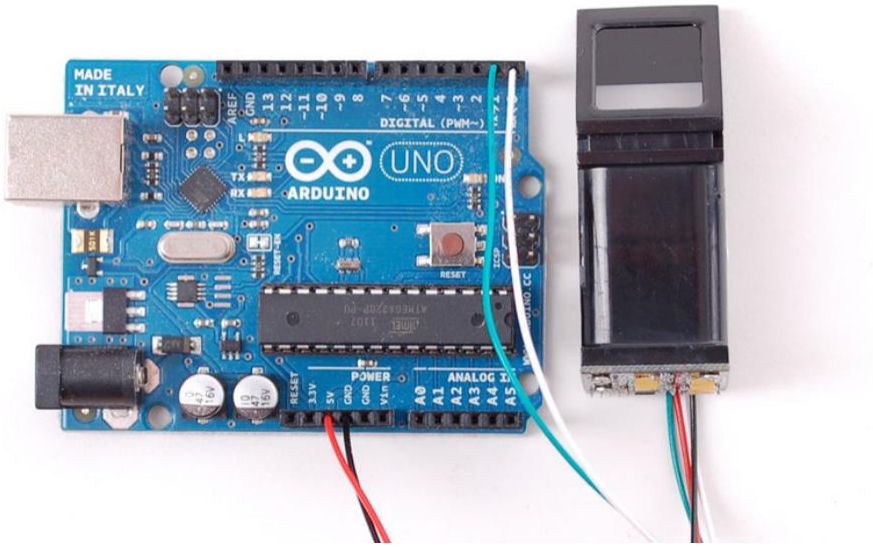


If your sensor has all the same-color wires, The first wire from the left is ground, then the two data pins, then power. You'll have to cut, strip and solder the wires.

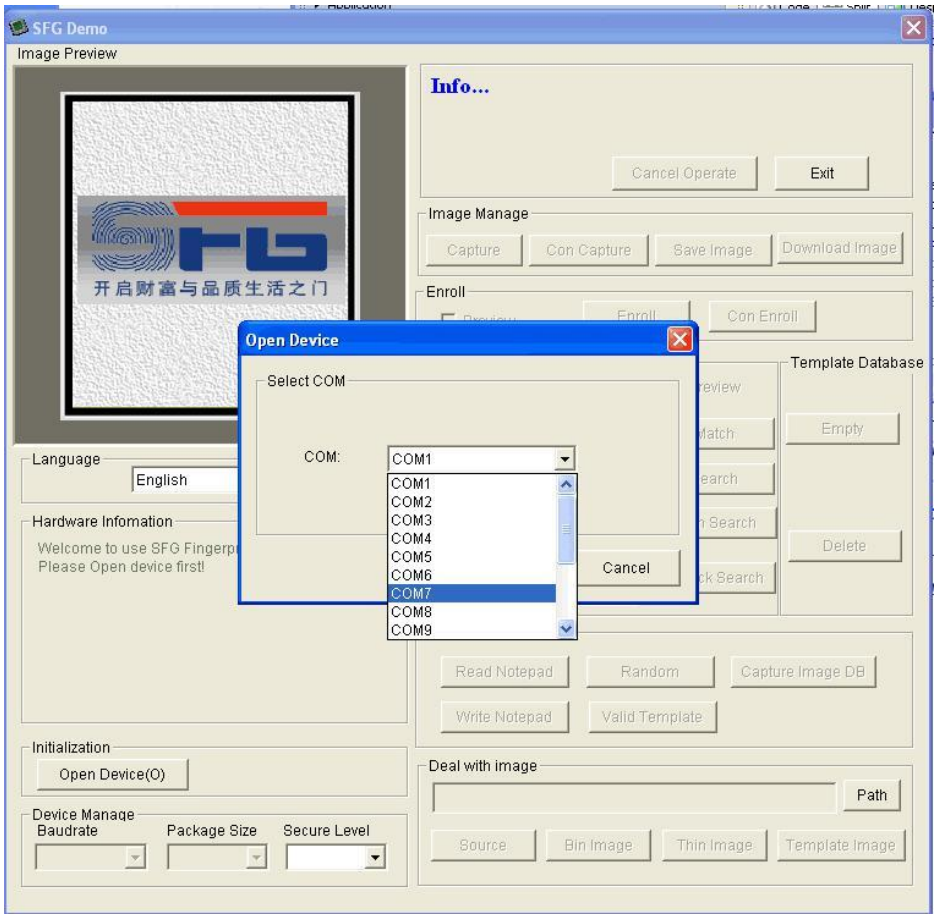
RX is the same as the White wire
TX is the same as the Green wire



If your sensor has different wires, The first wire from the left should be the black wire ground, then the two data pins, RX is the white wire, TX is the green wire then the red power wire. You'll have to cut, strip and solder the wires.

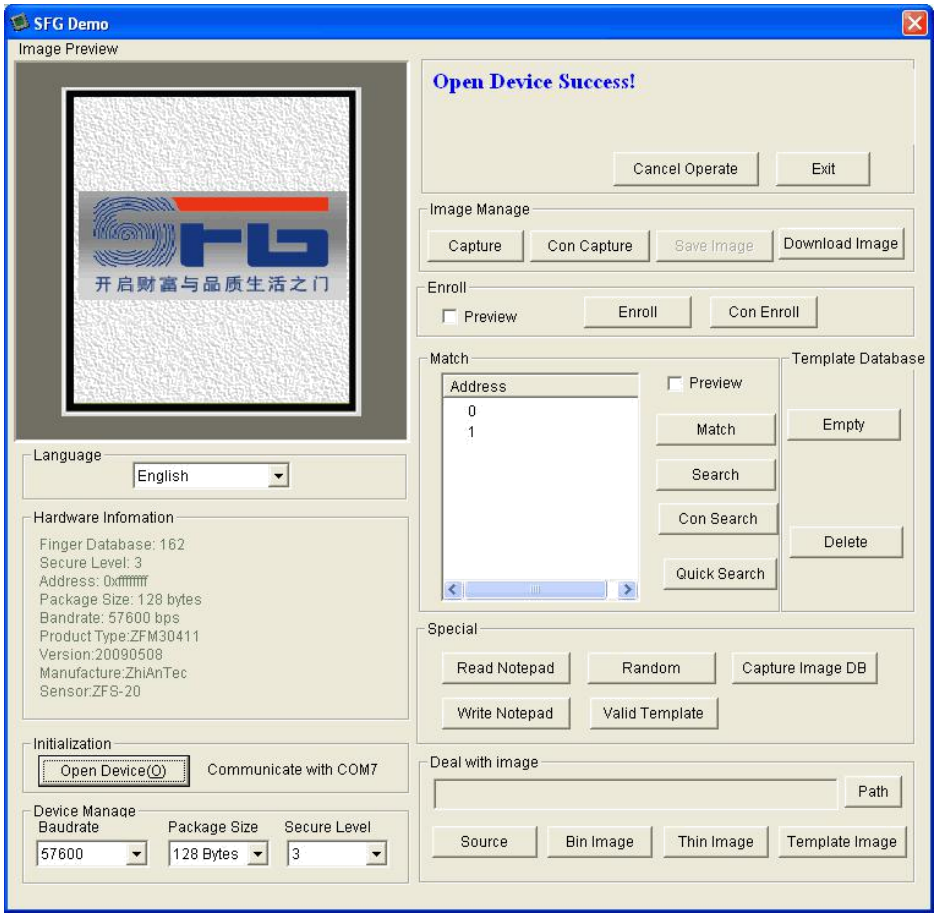


Start up the SFGDemo software and click **Open Device** from the bottom left corner. Select the **COM port** used by the Arduino.



And press OK when done. You should see the following, with a blue success message and some device statistics in the bottom corner. You can change the baud rate in the bottom left hand corner, as well as the "security level" (how sensitive it is) but we suggest leaving those alone until you have everything running and you want to experiment. They should default to 57600 baud and security level 3 so set them if they're wrong

If you get an error when you Open Device, check your wiring, try swapping the RX and TX wires on the sensor, that's a common mixup!



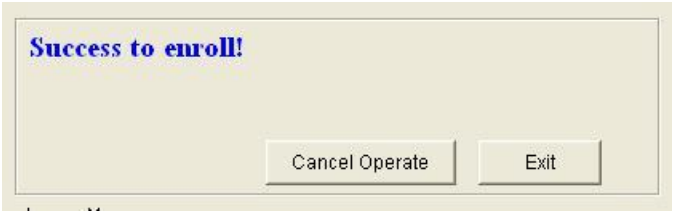
Lets enroll a new finger! Click the **Preview** checkbox and press the **Enroll** button next to it (**Con Enroll** means 'Continuous' enroll, which you may want to do if you have many fingers to enroll). When the box comes up, enter in the ID # you want to use. You can use up to 162 ID numbers.

The software will ask you to press the finger to the sensor

You can then see a preview (if you cliecked the preview checkbox) of the fingerprint.

You will then have to repeat the process, to get a second clean print. Use the same finger!

On success you will get a notice.



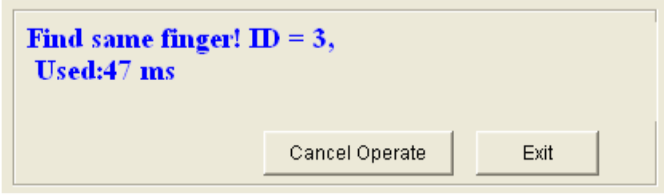
If there's a problem such as a bad print or image, you'll have to do it again.

Searching with the Software

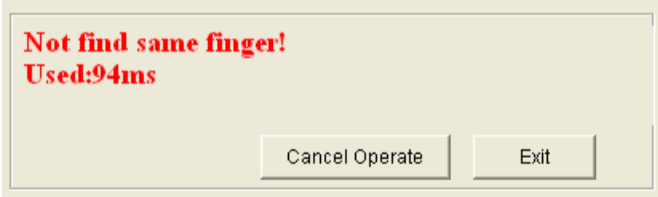
Once you have the finger enrolled, it's a good idea to do a quick test to make sure it can be found in the database. Click on the **Search** button on the right hand side.

When prompted, press a different/same finger to the sensor.

If it is the same finger, you should get a match with the ID #



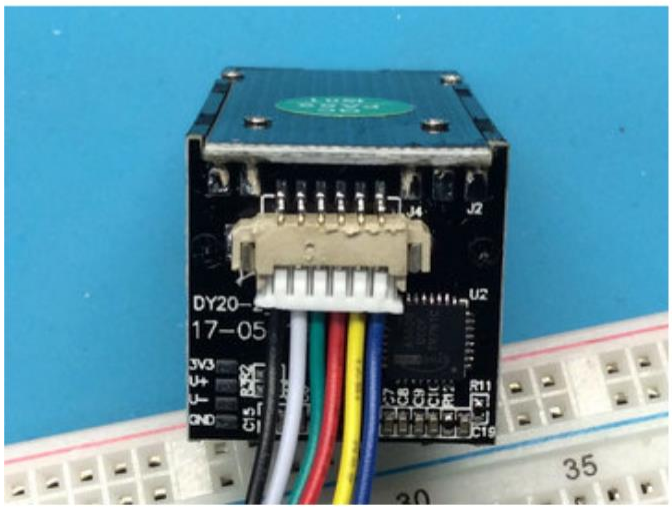
If it is not a finger in the database, you will get a failure notice.



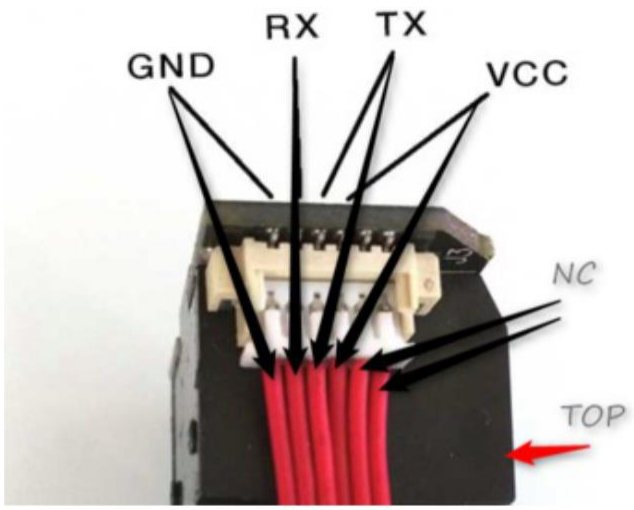
Wiring for use with Arduino

Once you've tested the sensor, you can now use it within a sketch to verify a fingerprint. We'll need to rewire the sensor. Disconnect the green and white wires and plug the green wire into digital **2** and the white wire to digital **3**. (For ESP8266 use **4** & **5**, for devices with Hardware UART use **0** & **1**)

> It is normal for the sensor to blink the LED quickly once powered, after that the LED may stay off until you've started to request data from it



If your sensor has different wires, The first wire from the left should be the black wire ground, then the two data pins, RX is the white wire, TX is the green wire then the red power wire. You'll have to cut, strip and solder the wires.



If your sensor has all the same-color wires, The first wire from the left is ground, then the two data pins, then power. You'll have to cut, strip and solder the wires.

RX is the same as the White wire
TX is the same as the Green wire

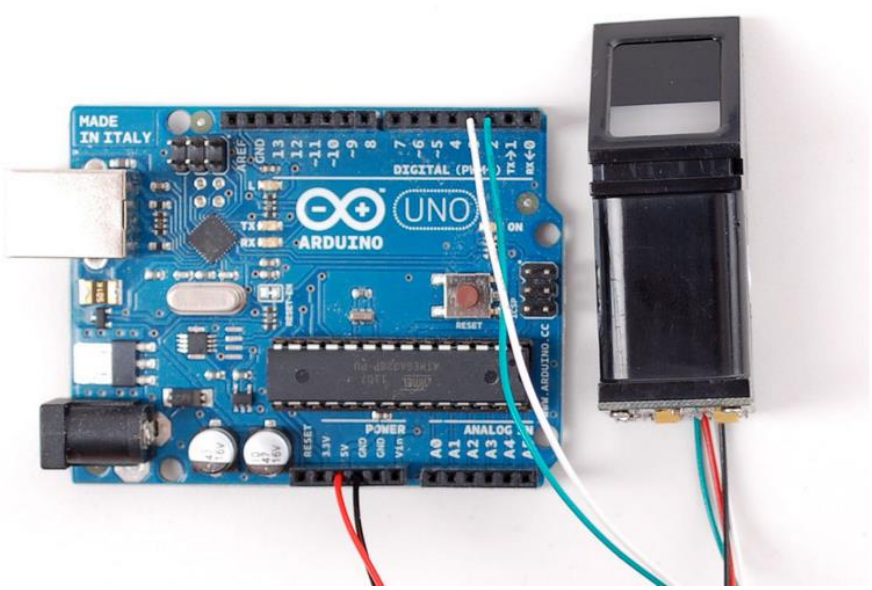
Arduino UNO & Compatible Wiring

This example sketch uses pins **2** and **3** for software serial (on ATmega328P type boards by default) - Not all boards support Software Serial on all pins so check board documentation! For example on ESP8266 we used **4** & **5**

You can power the sensor from **3.3V** or **5V**

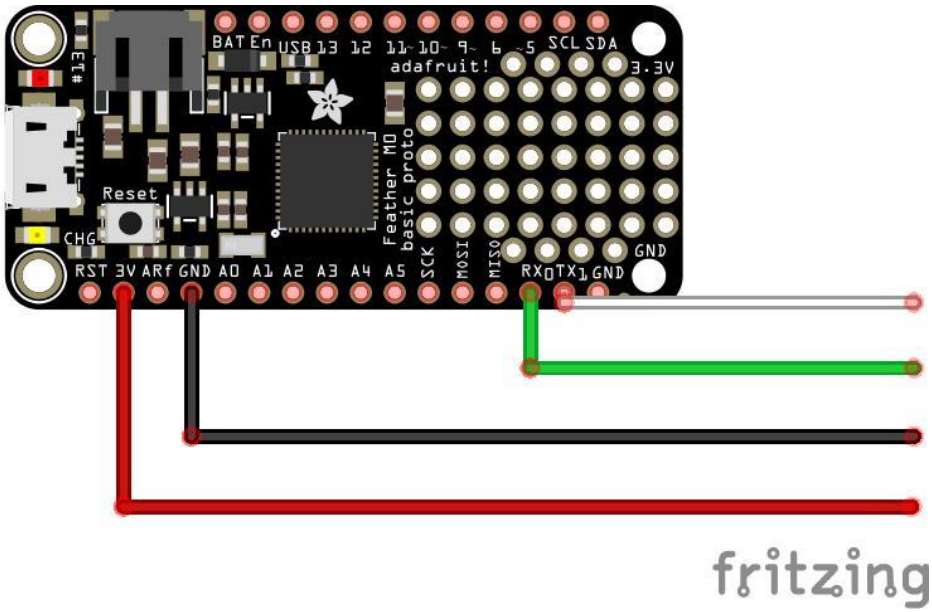
> In the diagrams below we show the wires plugged directly into the Arduino. However, this does not work well because the wires are so thin and they dont make contact. You should solder thicker solid core wires to each

wire, to make good contact



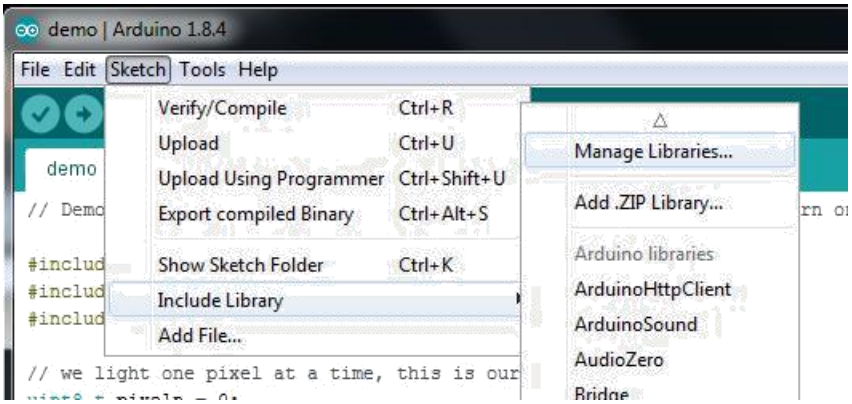
Hardware Serial Wiring

If you have a device with hardware serial, you should use that instead. Often this is pins #0 and #1



Next, you'll need to install the Adafruit Fingerprint sensor library (also available from github) (https://adafru.it/aRz).

Open up the Arduino Library Manager:



Type in **Fingerprint** until you see the **Adafruit Fingerprint library** show up!

Click Install! That's it. Now you should be able to select the **File→Examples→Adafruit_Fingerprint→fingerprint** example sketch.

Soft & Hard Serial

By default the sketch uses software serial (Arduino UNO & compatibles). If you are using a device with Hardware Serial, e.g does not have a USB-Serial converter chip, use that instead! Usually those are on pins 0 & 1

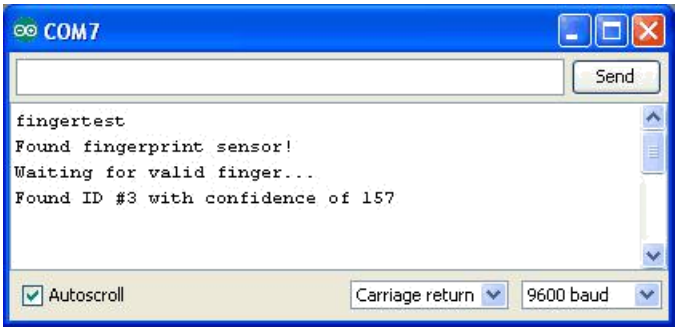
```
//   On Leonardo/Micro or others with hardware serial, use those! #0 is green wire, #1 is white
//   uncomment this line:
#define mySerial Serial1

//   For UNO and others without hardware serial, we must use software serial...
//   pin #2 is IN from sensor (GREEN wire)
//   pin #3 is OUT from arduino  (WHITE wire)
//   comment these two lines if using hardware serial
//#include <SoftwareSerial.h>
//SoftwareSerial mySerial(2, 3);
```

If necessary, uncomment/comment lines for hardware serial support

Upload

Upload it to your Arduino as usual. Open up the serial monitor at 9600 baud and when prompted place your finger against the sensor that was already enrolled.

You should see the following:



The 'confidence' is a score number (from 0 to 255) that indicates how good of a match the print is, higher is better. Note that if it matches at all, that means the sensor is pretty confident so you don't have to pay attention to the confidence number unless it makes sense for high security applications.

Of course you have to have **enrolled** a fingerprint first! If you did this using the Windows program, that's good to go. If you have not yet, you should at least have gotten a `Found fingerprint sensor!` printout. You can go ahead to the next step to enroll fingerprints.

If you get `Did not find fingerprint sensor :(` check your wiring, maybe swap the RX and TX wire as that's a common issue

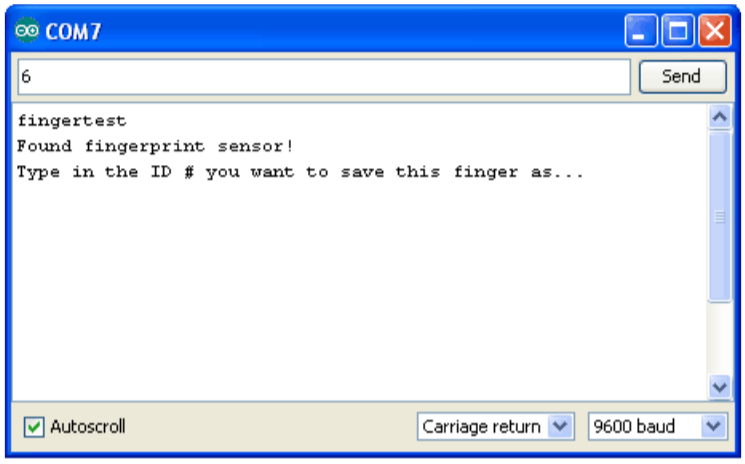
If you want to have a more detailed report, change the **loop()** to run **getFingerprintID()** instead of **getFingerprintIDez()** - that will give you a detailed report of exactly what the sensor is detecting at each point of the search process.

Enrolling with Arduino

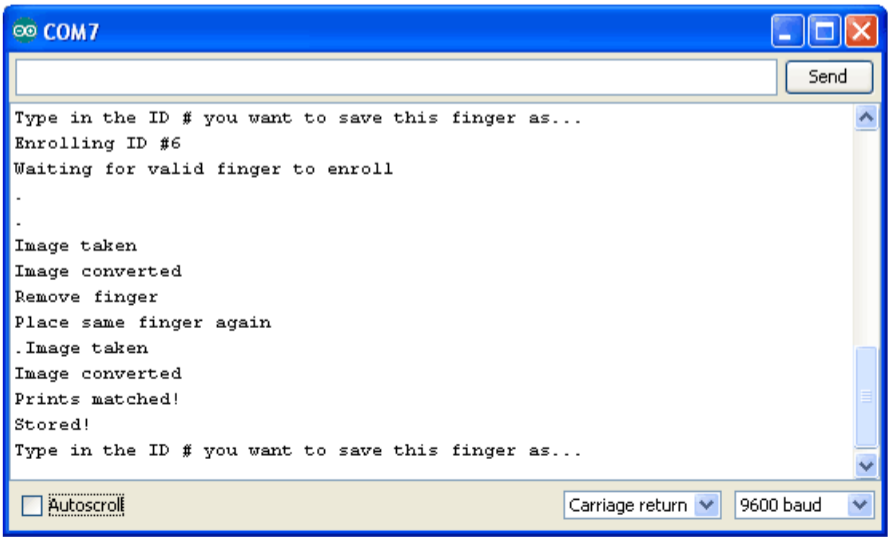
We did put together a simple sketch for enrolling a new finger via Arduino - its not as easy to use as the Windows program but it does work!

Run the **File→Examples→Adafruit_Fingerprint→enroll** sketch and upload it to the Arduino, use the same wiring as above.

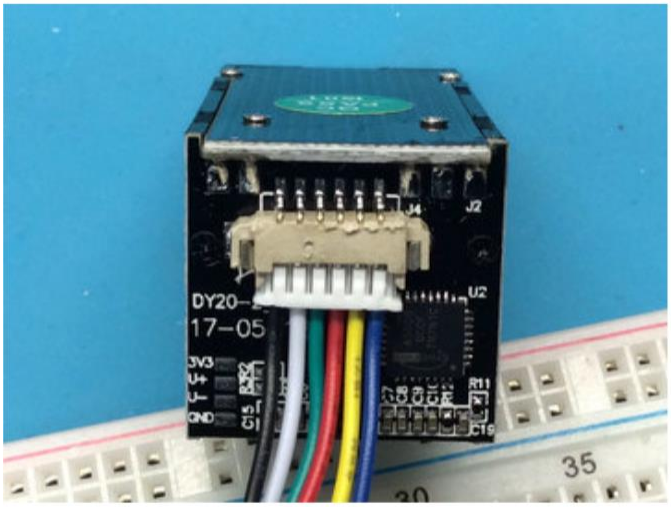
When you open up the serial monitor, it will ask for you to type in the ID to enroll - use the box up top to type in a number and click Send.



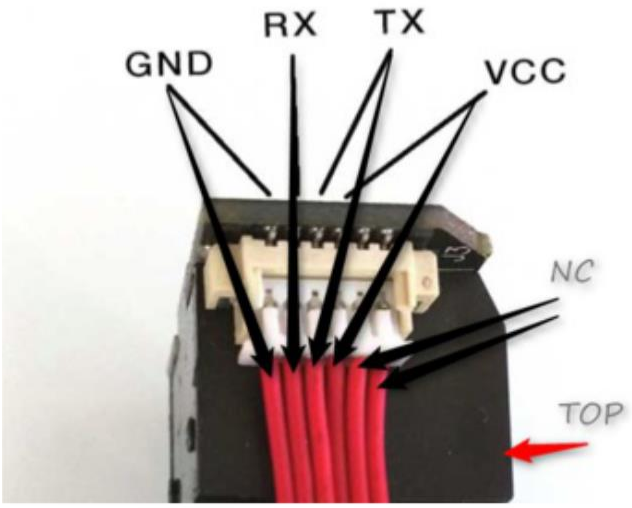
Then go through the enrollment process as indicated. When it has successfully enrolled a finger, it will print **Stored!**



Don't forget to do a search test when you're done enrolling to make sure its all good!

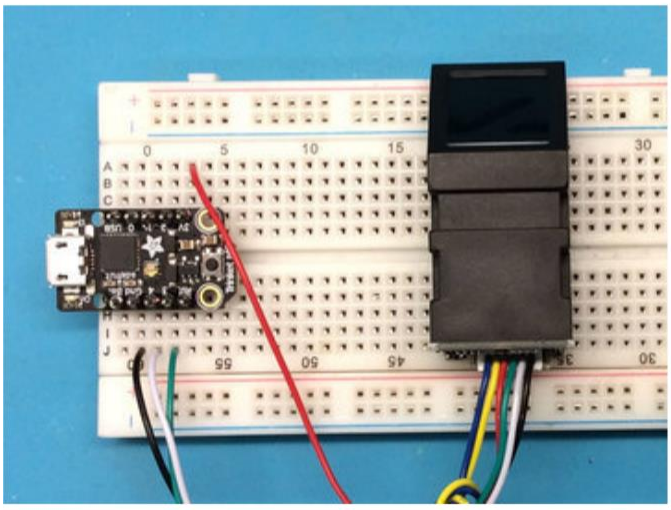


If your sensor has different wires, The first wire from the left should be the black wire ground, then the two data pins, RX is the white wire, TX is the green wire then the red power wire. You'll have to cut, strip and solder the wires.



If your sensor has all the same-color wires, The first wire from the left is ground, then the two data pins, then power. You'll have to cut, strip and solder the wires.

RX is the same as the White wire
TX is the same as the Green wire



Every CircuitPython board has a hardware UART. Check the product page or look for **RX** and **TX** written on the board. Remember that the RX from the sensor goes to the TX on the board! If you have problems try swapping them, its a common mistake

Installing Library

To use the Fingerprint sensor you'll need to install the Adafruit CircuitPython Fingerprint (https://adafru.it/C4A) library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU) for both express and non-express boards.

Remember for non -express boards like the Trinket M0 or Feather M0 Basic, you'll need to manually install the necessary libraries from the bundle:

- **adafruit_fingerprint.mpy**

You can also download the **adafruit_fingerprint.mpy** from its releases page on Github (https://adafru.it/C4B).

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_fingerprint.mpy** file copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz)so you are at the CircuitPython **>>>** prompt.

Usage

To demonstrate the usage of the sensor, we'll use the example python script included with the library. This sensor is fairly complex so its hard to run it just from the REPL.

Once you've installed the library, run this **main.py** example on your CircuitPython board.

```
import time
import board
import busio
from digitalio import DigitalInOut, Direction
import adafruit_fingerprint

led = DigitalInOut(board.D13)
led.direction = Direction.OUTPUT

uart = busio.UART(board.TX, board.RX, baudrate=57600)

#  If using with a computer such as Linux/RaspberryPi, Mac, Windows...
#import serial
#uart = serial.Serial("/dev/ttyUSB0", baudrate=57600, timeout=1)

finger = adafruit_fingerprint.Adafruit_Fingerprint(uart)

##################################################


def get_fingerprint():
    """Get a finger print image, template it, and see if it matches!"""
    print("Waiting for image...")
    while finger.get_image() != adafruit_fingerprint.OK:
        pass
    print("Templating...")
    if finger.image_2_tz(1) != adafruit_fingerprint.OK:
```

```
            return False
        print("Searching...")
        if finger.finger_fast_search() != adafruit_fingerprint.OK:
            return False
        return True

#  pylint: disable=too-many-branches def
get_fingerprint_detail():
        """Get a finger print image, template it, and see if it matches!
        This time, print out each error instead of just returning on failure"""
        print("Getting image...", end="", flush=True)
        i = finger.get_image()
        if i == adafruit_fingerprint.OK:
            print("Image taken")
        else:
            if i == adafruit_fingerprint.NOFINGER:
                print("No finger detected")
            elif i == adafruit_fingerprint.IMAGEFAIL:
                print("Imaging error")
            else:
                print("Other error")
            return False

        print("Templating...", end="", flush=True)
        i = finger.image_2_tz(1)
        if i == adafruit_fingerprint.OK:
            print("Templated")
        else:
            if i == adafruit_fingerprint.IMAGEMESS:
                print("Image too messy")
            elif i == adafruit_fingerprint.FEATUREFAIL:
                print("Could not identify features")
            elif i == adafruit_fingerprint.INVALIDIMAGE:
                print("Image invalid")
            else:
                print("Other error")
            return False

        print("Searching...", end="", flush=True)
        i    = finger.finger_fast_search()
        # pylint: disable=no-else-return
        # This block needs to be refactored when it can be tested. if i ==
        adafruit_fingerprint.OK:
            print("Found fingerprint!") return
            True
        else:
            if i == adafruit_fingerprint.NOTFOUND: print("No
                match found")
            else:
                print("Other error") return
            False

#  pylint: disable=too-many-statements
def enroll_finger(location):
        """Take a 2 finger images and template it, then store in 'location'"""
        for fingerimg in range(1, 3):
            if fingerimg == 1:
                print("Place finger on sensor...", end="", flush=True)
            else:
                print("Place same finger again...", end="", flush=True)
```

```
        while True:
                i = finger.get_image()
                if i == adafruit_fingerprint.OK:
                        print("Image taken")
                        break
                elif i == adafruit_fingerprint.NOFINGER:
                        print(".", end="", flush=True)
                elif i == adafruit_fingerprint.IMAGEFAIL:
                        print("Imaging error")
                        return False
                else:
                        print("Other error")
                        return False

        print("Templating...", end="", flush=True)
        i   =  finger.image_2_tz(fingerimg)  if  i  ==
        adafruit_fingerprint.OK:
                print("Templated")
        else:
                if i == adafruit_fingerprint.IMAGEMESS:
                        print("Image too messy")
                elif i == adafruit_fingerprint.FEATUREFAIL: print("Could
                        not identify features")
                elif i == adafruit_fingerprint.INVALIDIMAGE: print("Image
                        invalid")
                else:
                        print("Other error") return
                False

        if fingerimg == 1:
                print("Remove finger")
                time.sleep(1)
                while i != adafruit_fingerprint.NOFINGER:
                        i = finger.get_image()

print("Creating model...", end="", flush=True)
i = finger.create_model()
if i == adafruit_fingerprint.OK:
        print("Created")
else:
        if i == adafruit_fingerprint.ENROLLMISMATCH:
                print("Prints did not match")
        else:
                print("Other error")
        return False

print("Storing model #%d..." % location, end="", flush=True)
i   =  finger.store_model(location)  if  i  ==
adafruit_fingerprint.OK:
        print("Stored")
else:
        if i == adafruit_fingerprint.BADLOCATION: print("Bad
                storage location")
        elif i == adafruit_fingerprint.FLASHERR: print("Flash
                storage error")
        else:
                print("Other error") return
        False
```

```
    return True


##################################################

def get_num():
    """Use input() to get a valid number from 1 to 127. Retry till success!"""
    i = 0
    while (i > 127) or (i < 1):
        try:
            i   = int(input("Enter ID # from 1-127: ")) except
        ValueError:
            pass
    return i


while True:
    print("----------------")
            if finger.read_templates() != adafruit_fingerprint.OK:
        raise RuntimeError('Failed to read templates')
    print("Fingerprint templates:", finger.templates)
    print("e) enroll print")
    print("f) find print")
    print("d) delete print")
    print("----------------")
    c = input("> ")

    if c == 'e':
        enroll_finger(get_num())
    if c == 'f':
        if get_fingerprint():
            print("Detected #", finger.finger_id, "with confidence", finger.confidence)
        else:
            print("Finger not found")
    if c == 'd':
        if finger.delete_model(get_num()) == adafruit_fingerprint.OK:
            print("Deleted!")
        else:
            print("Failed to delete")
```

It's fairly long but it will help you set-up and test your sensor!

When you first start up, you should get something like this:

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
main.py output:
----------------
Fingerprint templates: [2]
e) enroll print
f) find print
d) delete print
----------------
>
```

If you get an error like `RuntimeError: Failed to read data from sensor` it means something went wrong - check your wiring and baud rate!

This menu system is fairly simple, you have three things you can do

- Enroll print - you will use your finger to take images and 'store' the model in the sensor
- Find print - determine whether a fingerprint is known and stored
- Delete print - clear out a model

Enrolling Prints

Enrolling a finger print is easy. Type **e** to start the process. You'll need to select a location. The sensor can store up to 127 print locations. Pick a valid number, then place your finger *twice* to enroll.

```
Fingerprint templates: [2]
e) enroll print
f) find print
d) delete print
----------------
> e
Enter ID # from 1-127: 0
Enter ID # from 1-127: 199
Enter ID # from 1-127: 5
Place finger on sensor.................Image taken
Templating...Templated
Remove finger
Place same finger again....Image taken
Templating...Templated
Creating model...Created
Storing model #5...Stored
----------------
Fingerprint templates: [2, 5]
```

Note that after success, the **Fingerprint templates: [...]** printout will include the new template id.

If an error occurs, the sensor will give you an error, such as if the two prints don't match, or if it failed to store or generate a model:

```
> e
Enter ID # from 1-127: 4
Place finger on sensor..............Image taken
Templating...Templated
Remove finger
Place same finger again......Image taken
Templating...Templated
Creating model...Prints did not match
```

Finding Prints

Once you've enrolled fingerprints you can then test them. Run the **f**ind command, and try various fingers! Once the fingerprint id identified it will tell you the location number, in this case **#5**

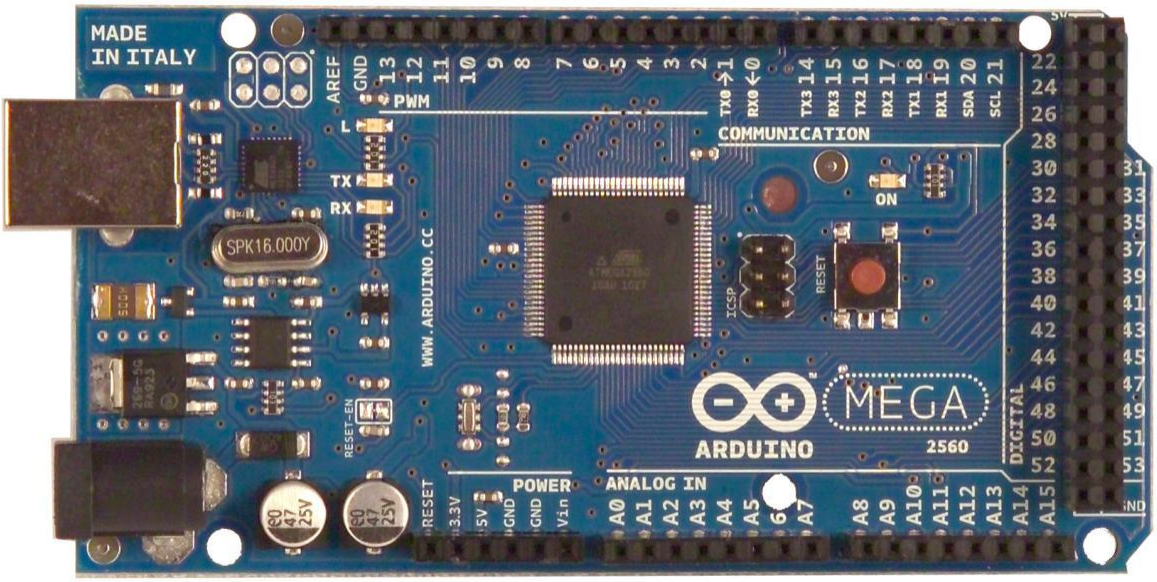
```
----------------
> f
Waiting for image...
Templating...
Searching...
Finger not found
----------------
Fingerprint templates: [2, 5]
e) enroll print
f) find print
d) delete print
----------------
> f
Waiting for image...
Templating...
Searching...
[illegible] 5 with confidence 102
```

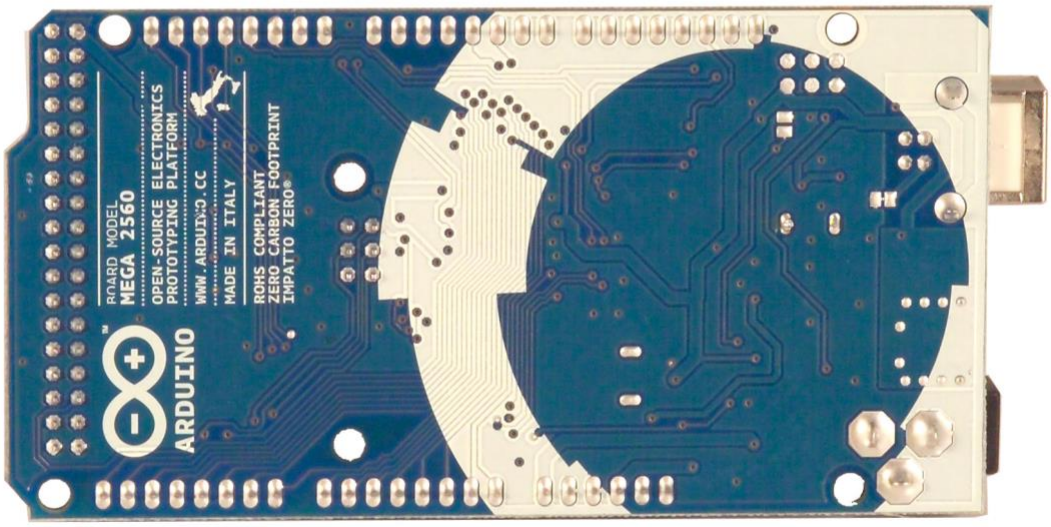
Deleting Fingerprints

If you made a mistake you can remove fingerprint models from the database. For example, here's how to delete #5. Note the **Fingerprint templates: [...]** printout changes!

```
----------------
Fingerprint templates: [2, 5]
e) enroll print
f) find print
d) delete print
----------------
> d
Enter ID # from 1-127: 5
Deleted!
----------------
Fingerprint templates: [2]
e) enroll print
f) find print
d) delete print
----------------
```

Arduino Mega 2560 Datasheet





Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Schematic & Reference Design

EAGLE files: arduino-mega2560-reference-design.zip

Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Power

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode() digitalWrite() and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX). Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2). These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
PWM: 0 to 13. Provide 8-bit PWM output with the analogWrite() function.
SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.
LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH

value, the LED is on, when the pin is LOW, it's off.

// **I2C: 20 (SDA) and 21 (SCL).** Support I2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I2C pins on the Duemilanove or Diecimila.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

// **AREF.** Reference voltage for the analog inputs. Used with analogReference().
// **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A SoftwareSerial library allows for serial communication on any of the Mega2560's digital pins.
The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. For SPI communication, use the SPI library.

Programming

The Arduino Mega can be programmed with the Arduino software (download). For details, see the reference and tutorials.
The ATmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It

communicates using the original STK500 protocol (reference, C header files).
You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these [redacted] for details.

Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.
This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened.
If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.
The Mega2560 contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega2560 has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega2560 PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega2560 is designed to be compatible with most shields designed for the Uno, Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega2560 and Duemilanove / Diecimila. *Please note that I^2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).*

GPRS /GSM SIM900A MODEM

USER MANUAL

Contents

Overview

GSM GPRS SIM900A Modem

GSM/GPRS Modem-RS232 is built with Dual Band GSM/GPRS engine- SIM900A, works on frequencies 900/ 1800 MHz. The Modem is coming with RS232 interface, which allows you connect PC as well as microcontroller with RS232 Chip(MAX232). The baud rate is configurable from 9600-115200 through AT command. The GSM/GPRS Modem is having internal TCP/IP stack to enable you to connect with internet via GPRS. It is suitable for SMS, Voice as well as DATA transfer application in M2M interface. The onboard Regulated Power supply allows you to connect wide range unregulated power supply . Using this modem, you can make audio calls, SMS, Read SMS, attend the incoming calls and internet through simple AT commands

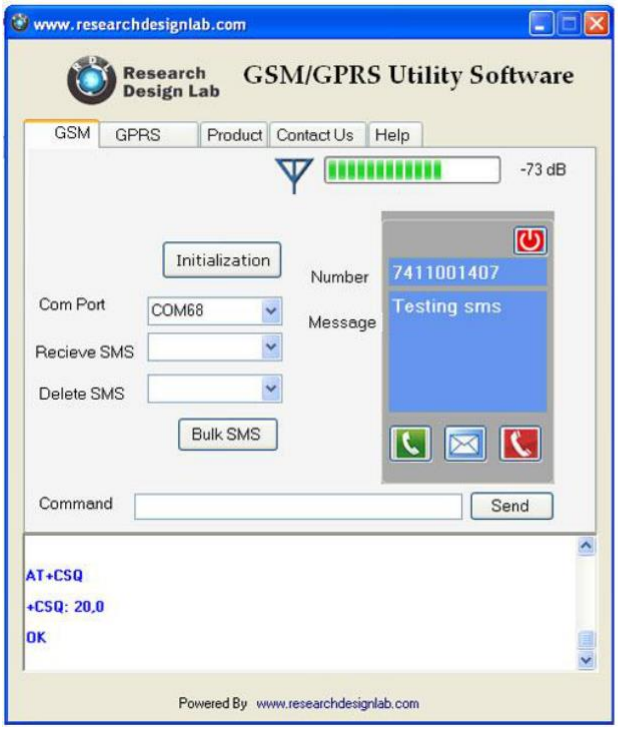
Features

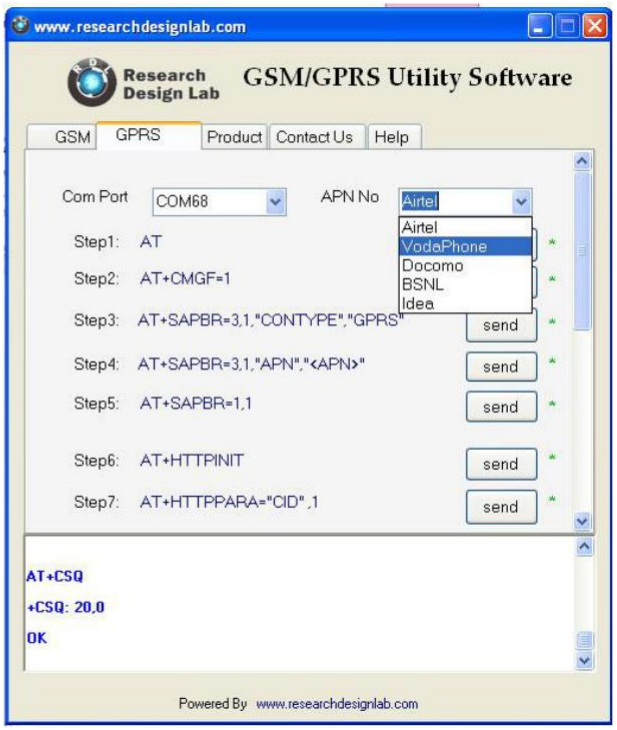
// Dual-Band GSM/GPRS 900/ 1800 MHz.
// RS232 interface for direct communication with computer or MCU kit.
// Configurable baud rate.
// Power controlled using 29302WU IC.
// ESD Compliance.
// Enable with MIC and SPeaker socket.
// With slid in SIM card tray.
// With Stub antenna and SMA connector.
// Input Voltage: 12V DC.

Datasheets

// AT Commands datasheet
https://drive.google.com/a/researchdesignlab.com/file/d/0BzrGD4zr88GnTkJwSll3dnhKbTg/edit?usp=sharing
// FTP Commands datasheet
https://drive.google.com/a/researchdesignlab.com/file/d/0BzrGD4zr88GnVkhacjUtY2tIU2c/edit?usp=sharing
// TCP/IP Commands datasheet
https://drive.google.com/a/researchdesignlab.com/file/d/0BzrGD4zr88GnUHRCQlJwUjdWTVU/edit?usp=sharing

GSM Utility Software





// Bulk Message sending
// AT command testing terminal
// Provides step by step GPRS setup

To download GSM/GPRS Utility software ,click on the link below

// https://docs.google.com/file/d/0BzrGD4zr88GnYll6dlFJT2NFY2s/edit
// http://www.4shared.com/file/rwyHmtGOba/GSM_GPRS_utility.html

Basic AT Commands for Testing

GSM AT Commands:

// TO CHECK THE MODEM: AT ↲
OK

// TO CHANGE SMS SENDING MODE: AT+CMGF=1 ↲
OK

// TO SEND NEW SMS:
AT+CMGS="MOBILE NO." ↲
<MESSAGE
{CTRL+Z}

// TO RECEIVE SMS
AT+CMGD=1 ↲ {to delete the message in buffer} AT+CMGR=1 ↲ {to receive first message AT+CMGR=1}
{to receive second message AT+CMGR=2 and so on}
+CMGL: 1,"REC READ","+85291234567",,"07/05/01,08:00:15+32",145,37 <MESSAGE

PREFERRED SMS MESSAGE STORAGE: AT+CPMS=? ↲
+CPMS: ("SM"),("SM"),("SM")
OK AT+CPMS? ↲
+CPMS: "SM",19,30,"SM",19,30,"SM",19,30

TO MAKE A VOICE CALL:

TO REDIAL LAST NO: ATDL ↲
TO RECEIVE INCOMING CALL: ATA ↲
TO HANGUP OR DISCONNECT A CALL: ATH ↲

TO SET A PARTICULAR BAUDRATE:

OPERATOR SELECTION: AT+COPS=? ↲
OK AT+COPS? ↲
+COPS: 0,0,"AirTel"
OK

- AT+CRC SET CELLULAR RESULT CODES FOR INCOMING CALL INDICATION: AT+CRC=? ↲

  +CRC: (0-1)
  OK AT+CRC? ↲
  +CRC: 0
  OK AT+CRC=1 ↲
  OK
  +CRING: VOICE

- READ OPERATOR NAMES.

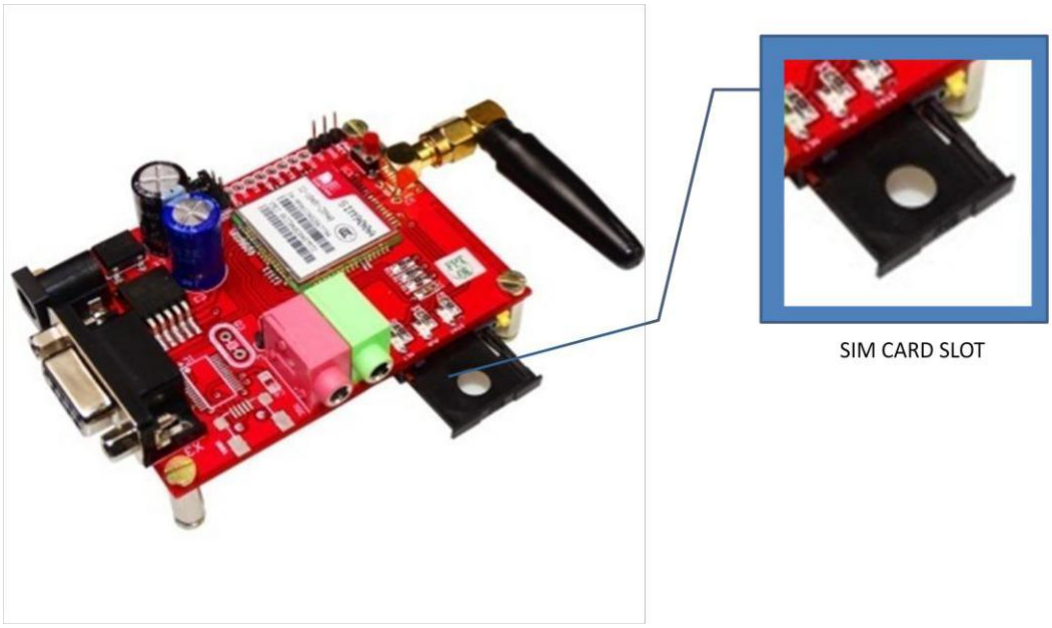
  AT+COPN=? ↲
  OK
  AT+COPN ↲
  +COPN: “472001″,”DHIMOBILE”
  +COPN: “60500
  +COPN: “502012″,”maxis mobile”
  +COPN:
  +COPN: “502013″,”TMTOUCH”
  +COPN
  +COPN: “502016″,”DiGi”
  +COPN: “502017″,”TIMECel”"
  +COPN: “502019″,”CELCOM GSM”

GPRS Commands:

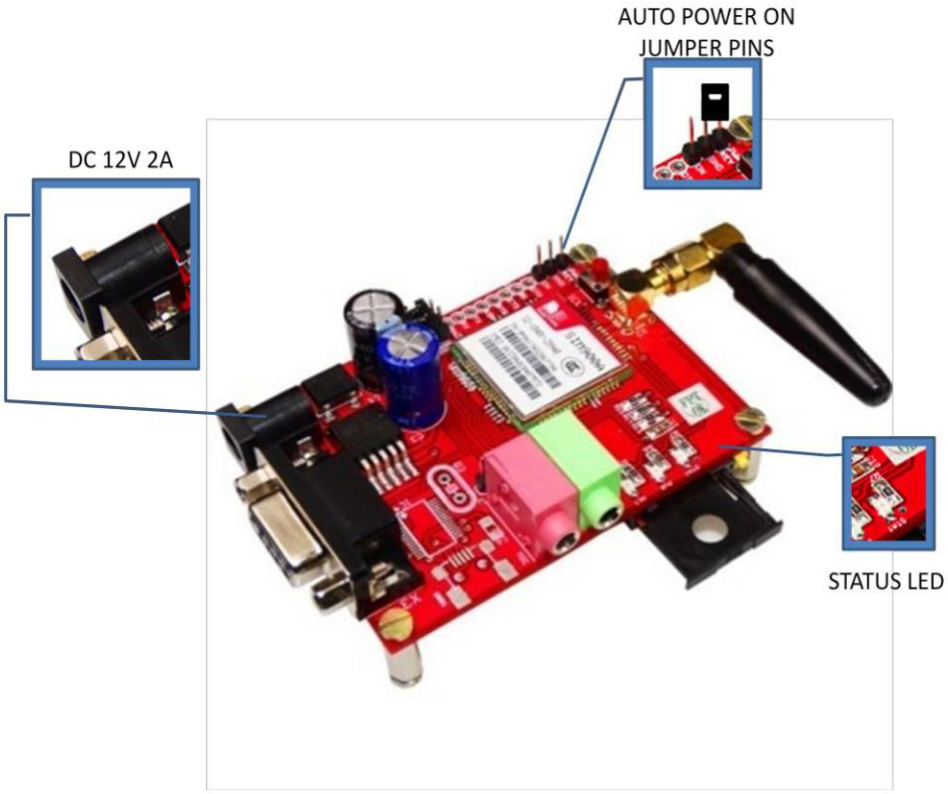
Command	Description
AT+CGATT ↲	ATTACH/DETACH FROM GPRS SERVICE
AT+CGDCONT ↲	DEFINE PDP CONTEXT
AT+CGQMIN ↲	QUALITY OF SERVICE PROFILE (MINIMUM ACCEPTABLE)
AT+CGQREQ ↲	QUALITY OF SERVICE PROFILE (REQUESTED)
AT+CGACT ↲	PDP CONTEXT ACTIVATE OR DEACTIVATE
AT+CGDATA ↲	ENTER DATA STATE
AT+CGPADDR ↲	SHOW PDP ADDRESS
AT+CGCLASS ↲	GPRS MOBILE STATION CLASS
AT+CGEREP ↲	CONTROL UNSOLICITED GPRS EVENT REPORTING
AT+CGREG ↲	NETWORK REGISTRATION STATUS
AT+CGSMS ↲	SELECT SERVICE FOR MO SMS MESSAGES
AT+CGCOUNT ↲	GPRS PACKET COUNTERS

MODULE SETUP

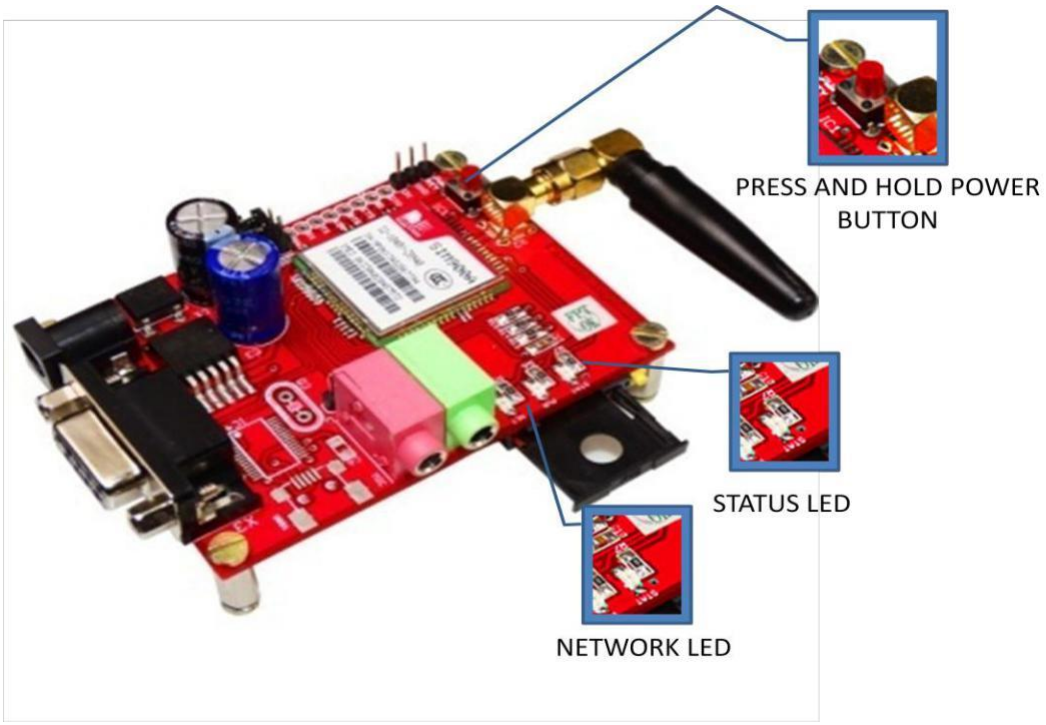
step 1 : Insert SIMcard into the SIM slot.



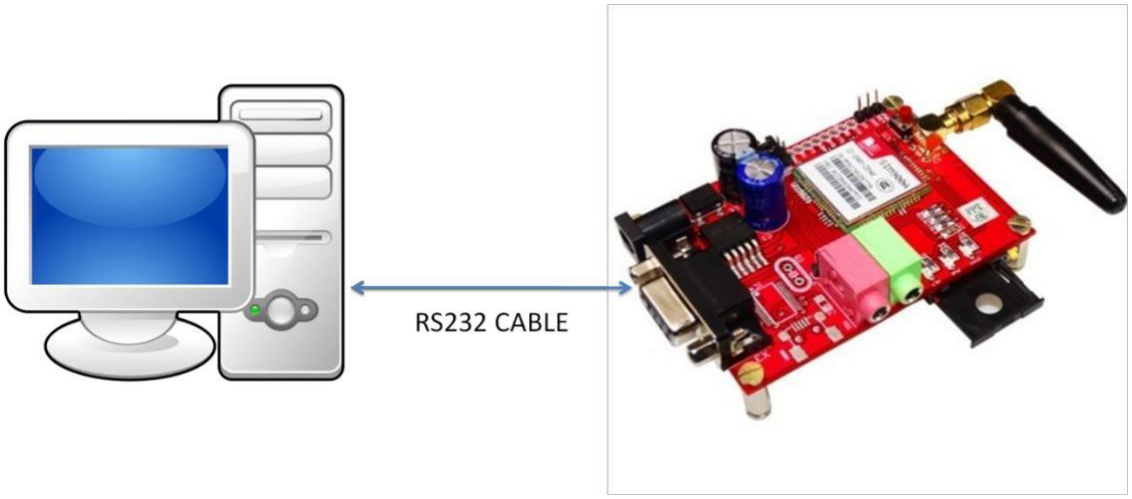
step 2 : Plug in 12V -2A DC power adapter, power led is lit (place jumper between PWRkey and on pin for only to turn ON automatically).



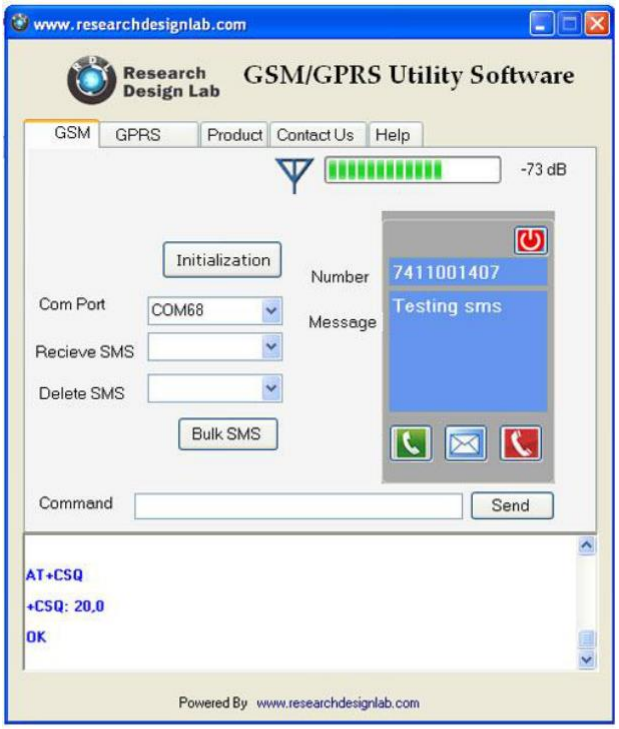
step 3 : Press and hold power button (To turn on manually without jumper)

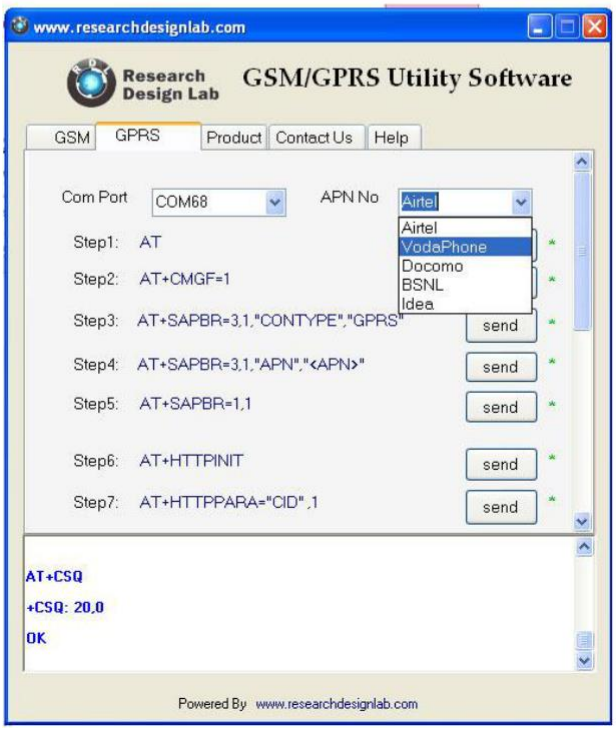


step 4 : Connect to PC through RS232 cable



step 5 : open GSM/GPRS utility software ,choose appropriate COM port and use AT commands listed in this manual for basic testing GPRS GSM/messaging and voice calling.





POWER MODES

Power down mode

SIM900A is set power down mode by "AT+CPOWD=0"

There are two methods for the module to enter into low current consumption status

Minimum Functionality Mode

Minimum functionality mode reduces the functionality of the module to a minimum and thus minimizes the current consumption to the lowest level.
If SIM900A has been set to minimum functionality by "AT+CFUN=0" If
SIM900A has been set to full functionality by "AT+CFUN=1"
If SIM900A is set "AT+CFUN=4" to disable both the above functionality.

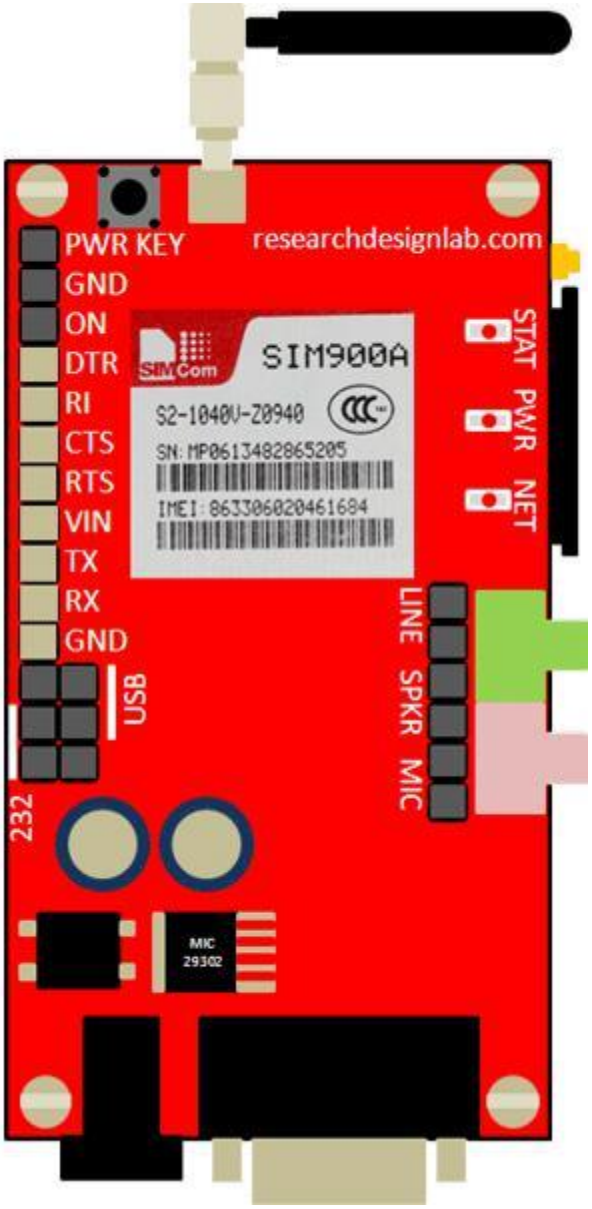
Sleep mode

We can control SIM900A module to enter or exit the SLEEP mode in customer applications through DTR signal. When DTR is in high level and there is no on air and hardware interrupt (such as GPIO interrupt or data on serial port), SIM900A will enter SLEEP mode automatically. In this mode, SIM900A can still receive paging or SMS from network but the serial port is not accessible.

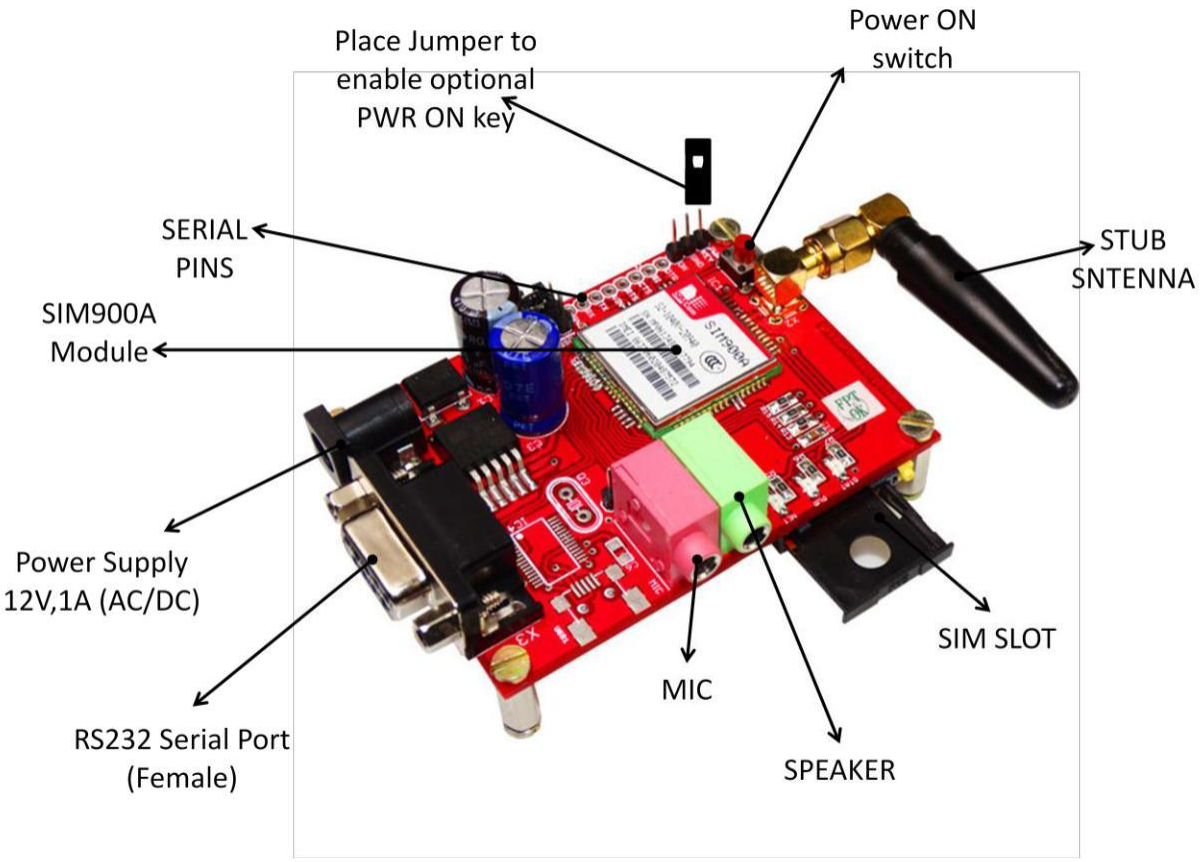
Wake up SIM900A from sleep mode

- Enable DTR pin to wake up SIM900A. If DTR pin is pulled down to a low level
- This signal will wake up SIM900A from power saving mode. The serial port will be active after DTR changed to low level for about 50ms.
- Receiving a voice or data call from network to wake up SIM900A.
- Receiving a SMS from network to wake up SIM900A.

PINS OF GSM SIM900A Modem

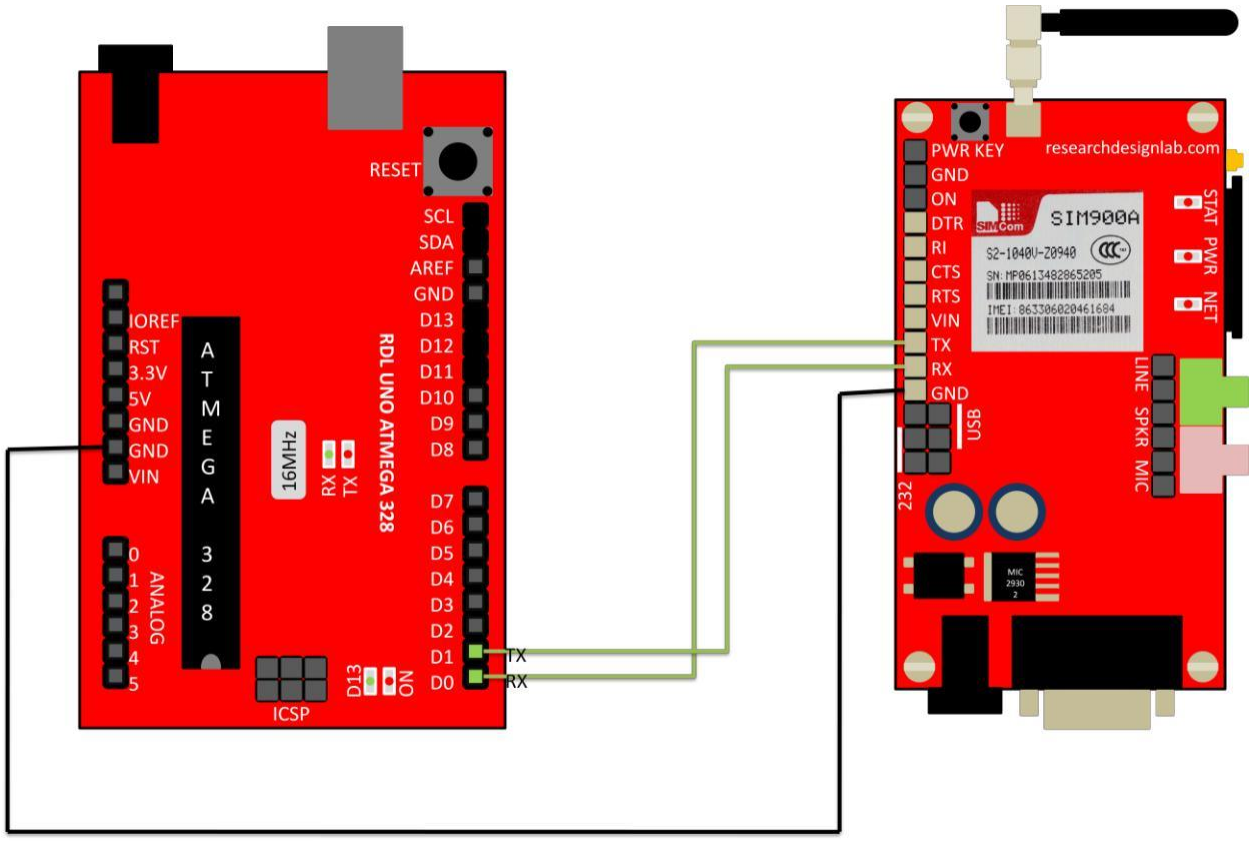


NARATION OF GSM SIM900A MODEM

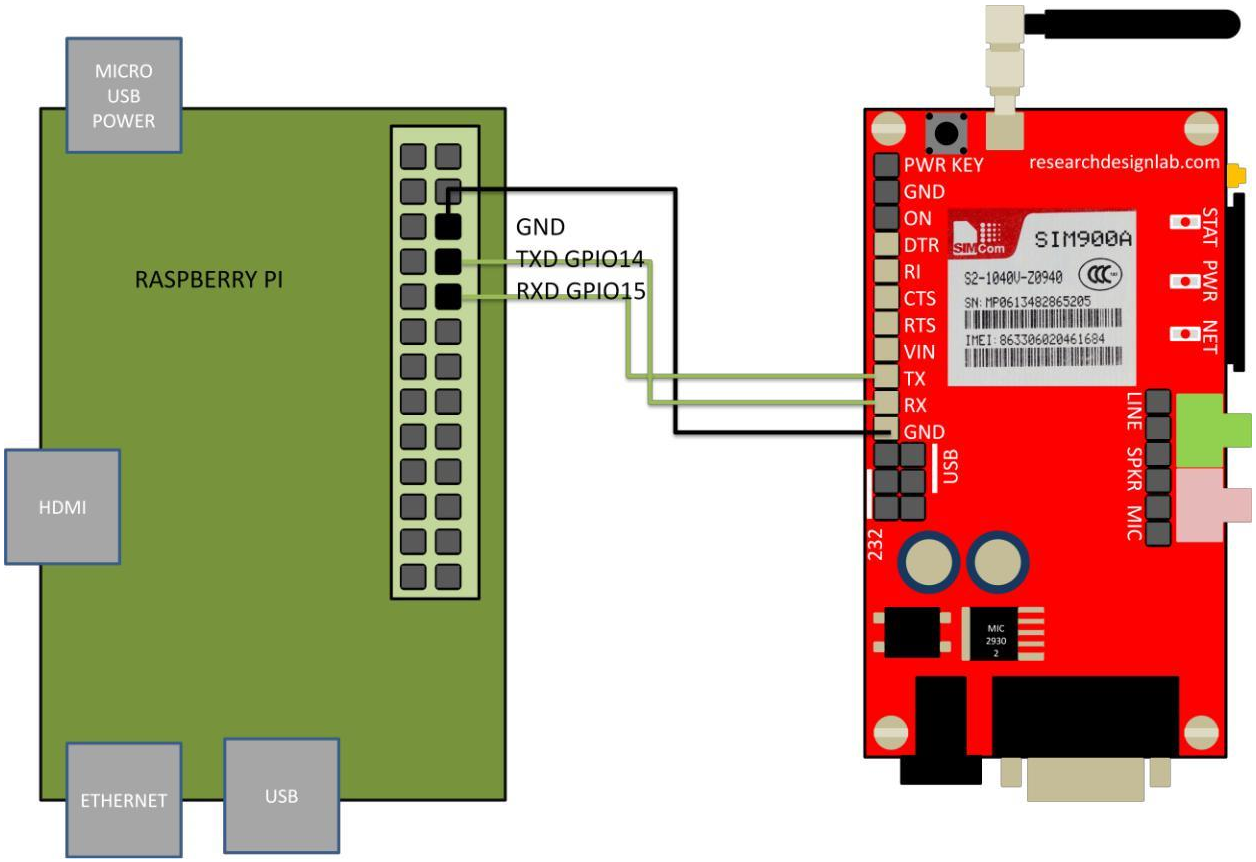


BLOCK DIAGRAMS

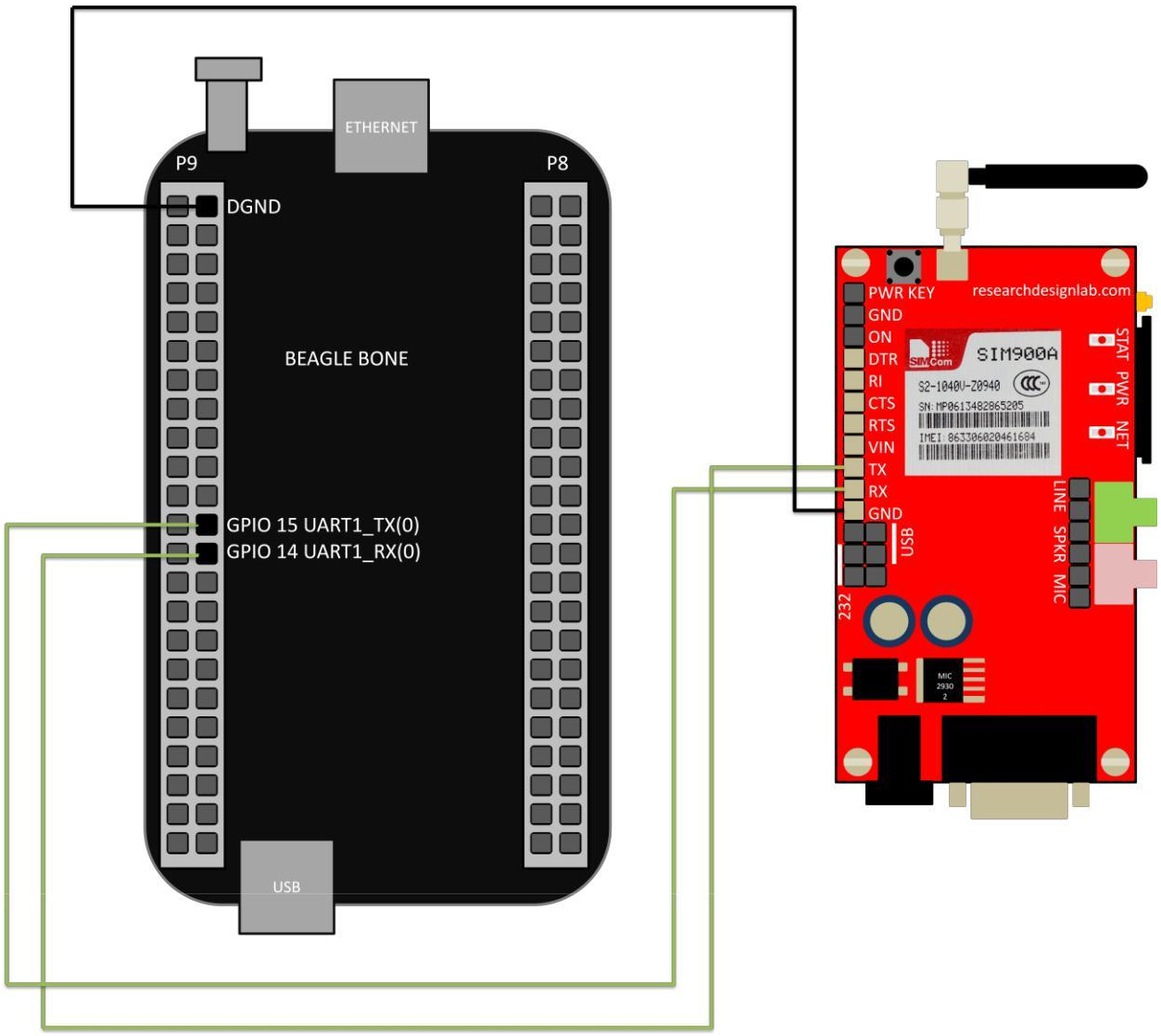
INTERFACING UNO AND GSM SHIELD



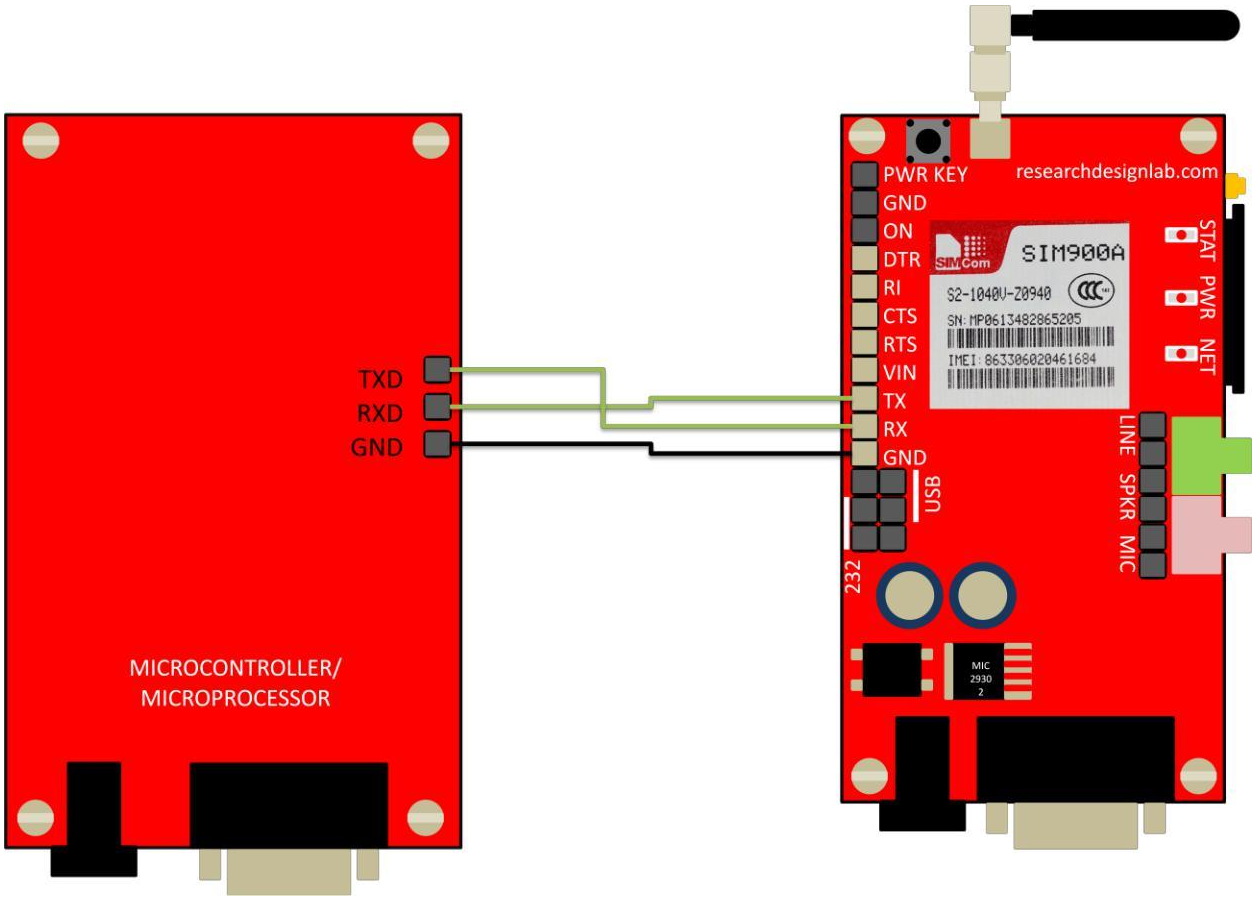
INTERFACING RASPBERRY AND GSM SHIELD



INTERFACING BEAGLEBOARD AND GSM SHIELD



INTERFACING MICROCONTROLLER WITH GSM SHIELD



CODES

ARM CODE
http://researchdesignlab.com/gsm-modem-arm-code

ATMEL CODE
http://researchdesignlab.com/gsm-modem-atmel-code

PIC CODE
http://forum.researchdesignlab.com/GSM%20SIM900/PIC/SIM900.c

ARDUNIO CODE
http://researchdesignlab.com/arduino-gsm2-code

RASPBERRY PI CODE
SENDING CODE

http://researchdesignlab.com/gsm-raspberry-code

RECEIVING CODE

http://researchdesignlab.com/gsm-raspberry-receiving-code.html

BEAGLEBONE CODE
SENDING CODE

http://researchdesignlab.com/gsm-beaglebone-send-code

RECEIVING CODE

http://researchdesignlab.com/gsm-beaglebone-receiving-code.html

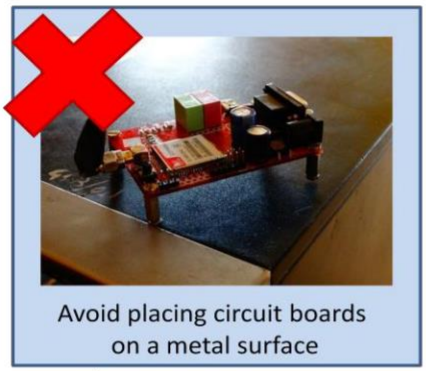
MSP430 CODE
http://forum.researchdesignlab.com/MSP430/MSP/GSM.zip

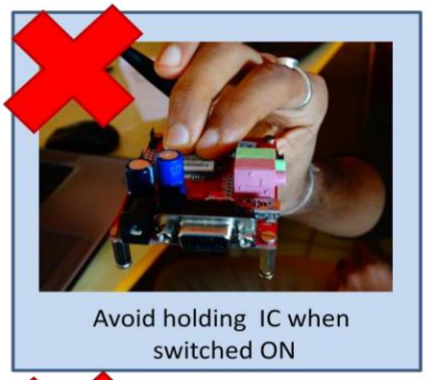
GSM POWER SAVING ATMEL CODE
http://researchdesignlab.com/gsm-power-atmel-code.html

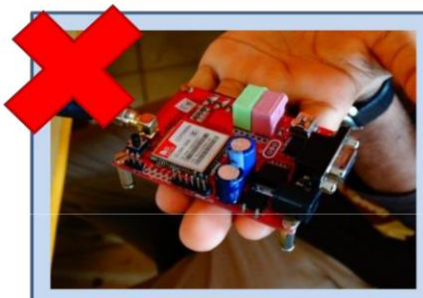
GSM POWER SAVING PIC CODE
http://researchdesignlab.com/gsm-power-pic-code.html

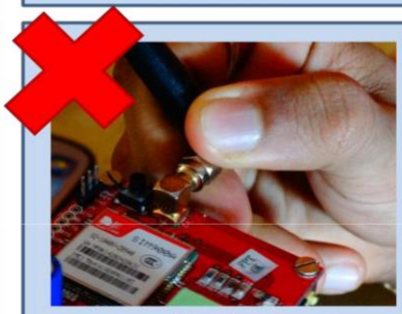
MODULE HANDLING

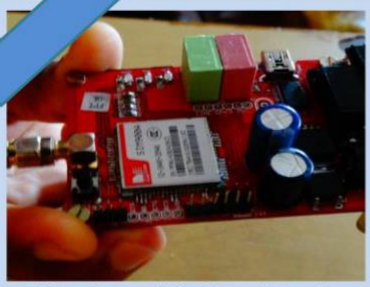
DO'S AND DONT'S


Avoid placing circuit boards on a metal surface


Avoid holding IC when switched ON


Avoid placing circuit boards n your palm


Avoid holding circuit with component


Hold edges while handling the circuit boards


If possible use anti static glovse