Open Sesame! A SMS-controlled door lock

Created by lady ada

https://learn.adafruit.com/open-sesame-a-sms-controlled-door-lock

Last updated on 2023-08-29 02:57:38 PM EDT
Table of Contents

Overview ........................................ 3
Parts ............................................. 5
Wiring ........................................... 6
  • Wiring for the FONA & Metro mini
  • Wiring for the TB6612 Motor Driver & Metro Mini
  • Wiring from TB6612 Motor Driver to Lockitron
Software .......................................... 8
  • Understanding the code in detail
  • Setup and initialization
  • Setup() function
  • Loop()
  • moveMotor helper
Final fit .......................................... 19
  • Freewiring
  • Fitting
Overview

Wouldn’t it be cool if, instead of having to give your dog-sitter a key to your place, you could just text your home to let her in? Now we build it ourselves using the Lockitron lock module body, [Metro Mini](https://www.adafruit.com/products/1164) (tiny Arduino-compatible) and a [FONA 800](https://www.adafruit.com/products/1145) cellular module.

Adafruit has discontinued carrying the Lockitron. To build this guide, you will need to source your own lock parts.

Normally, a Lockitron has a built-in WiFi system for locking & unlocking via their [App](https://www.adafruit.com/products/1144). Since they have redesigned their enclosure, we now have the 'guts' of the Lockitron available for hacking. There's just enough space inside for some tasty Adafruit electronics.
In this tutorial you'll learn how to listen to SMS messages on a FONA cellular module and use that to move a motor.

Please note this project is tested and working but there's a lot of optimizations we didn't add, to keep it simple! For example, it doesn't (but could): check the sender phone number to make sure it's really you, have low power functionality for long term use, listen to the rotation-detection switches to verify that the lock has turned when the motor is activated, check and notify you when the battery gets low, etc! This is a fun project and a good intermediate-advanced maker electronics project, but it's not designed to protect your home and possessions!
Parts

Adafruit has discontinued carrying the Lockitron module (perhaps they were end of life). You will have to source your own lock parts. Please use this tutorial as a guide as to how you might construct a similar project.

You'll need a bunch of parts for this project! You can noodle around with most of the parts. If you want to fit all the parts in the Lockitron enclosure, you'll also want a 500mA battery (Some experimentation may be required to get a 500mA battery that fits). The battery may have to go outside the box!

- Lockitron ()
- Metro Mini 328 - The microcontroller ()
- FONA 800 w/uFL antenna (http://adafru.it/1946)
- Sticker antenna ()
- SIM Card ()
- TB6612 motor controller breakout ()
- 500mA lipoly battery ()
- Silicone cover stranded wires ()

©Adafruit Industries
Since this isn't a beginner tutorial, please familiarize yourself with the usage of the Metro Mini (make sure you can upload code from the Arduino IDE, check the serial monitor) and FONA 800 (get your SIM card, verify you can use the FONA with the example Arduino library and examples to connect to the cellular network and send an SMS, etc) and TB6612 motor driver.

**Wiring**

We'll start by making a version that fits on a breadboard. In the end we did fit the whole thing into the Lockitron casing but its a bit of work and for most people, they might want to keep the parts outside so they can be poked-at and modified.
Wiring for the FONA & Metro Mini

- BAT from FONA to VIN on Metro Mini - this will let the metro mini run off of the same lipoly battery the FONA runs off of - red wire. This is technically undervolting it but the metro mini is happy to run at a lower voltage of ~3.7V!
- GND from FONA to GND on Metro Mini - common ground - black wire
- RST (reset) from FONA to #5 on the Metro Mini - this is so the Metro can reset the FONA when it first starts up - blue wire
- KEY on FONA to GND - this will keep the FONA on all the time - black wire
- RI (Ring Interrupt) on FONA to #2 on Metro Mini - will let FONA alert the Metro when a text message comes in - yellow wire
- TX on FONA to #3 on Metro Mini - data sent from FONA to Metro - white wire
- RX on FONA to #4 on Metro Mini - data sent to FONA from Metro - green wire
- Vio on FONA to 5V on Metro Mini - will tell the FONA to use the same logic level as the Metro when sending/receiving data. - red wire
- 5V on FONA (near the micro USB connector) to USB on Metro Mini - will let you also use the Metro’s USB connection to recharge the FONA battery - white wire

Wiring for the TB6612 Motor Driver & Metro Mini

- V+ (not Vm!) on TB6612 connects to 5V on Metro Mini - will tell the TB6612 to use the same logic level as the Metro - red wire. Technically this will be 3.7V or there abouts but that's fine
- GND on TB6612 to GND on Metro Mini - common ground - black wire
- PWMB on TB6612 to A1 on Metro Mini - motor direction control 2 - white wire
- BIN2 on TB6612 to A2 on Metro Mini - motor direction control 2 - yellow wire
- BIN1 on TB6612 to A3 on Metro Mini - motor direction control 1 - blue wire

Wiring from TB6612 Motor Driver to Lockitron

- Red wire from battery pack connects to V+ on TB6612 (we recommend the ’3.5mm terminal block’ spot)
- Black wire from battery pack connects to GND on TB6612 (we recommend the ’3.5mm terminal block’ spot)
- Red wire from the motor connects to MOTORB on TB6612 (last pin)
- Black wire from the motor connects to MOTORB on TB6612 (2nd-to-last pin)
Software

Before you upload the Open Sesame sketch, it's a good idea to make sure you can use the Metro Mini with the FONA 800 module. Check out this tutorial on using the FONA, make sure to try out the FONAtest sketch with the updated pin defines:

```
#define FONA_TX 3
#define FONA_RX 4
#define FONA_RST 5
```

Once you have that working nicely, and for example you have tested sending/receiving SMS messages, continue by downloading the Open Sesame sketch.

See the project's files on GitHub

```
// SPDX-FileCopyrightText: 2019 Anne Barela for Adafruit Industries
// SPDX-License-Identifier: MIT

#include <SoftwareSerial.h>
#include "Adafruit_FONA.h"

#define FONA_TX 3
#define FONA_RX 4
#define FONA_RST 5
#define FONA_RI 2
#define MOTOR_PWM A1
#define MOTOR_1 A2
#define MOTOR_2 A3
#define LED 13
#define OPEN true
#define CLOSE false
#define BUSYWAIT 5000  // milliseconds

// this is a large buffer for replies
char replybuffer[255];

// or comment this out & use a hardware serial port like Serial1 (see below)
```
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);

Adafruit_FONA fona = Adafruit_FONA(FONA_RST);

uint8_t readline(char *buff, uint8_t maxbuff, uint16_t timeout = 0);

boolean fonainit(void) {
  Serial.println(F("Initializing....(May take 3 seconds)"));

  digitalWrite(LED, HIGH);
  delay(100);
  digitalWrite(LED, LOW);
  delay(100);
  digitalWrite(LED, HIGH);
  delay(100);
  digitalWrite(LED, LOW);
  delay(100);

  // make it slow so its easy to read!
  fonaSS.begin(4800); // if you're using software serial
  //Serial1.begin(4800); // if you're using hardware serial

  // See if the FONA is responding
  if (! fona.begin(fonaSS)) { // can also try fona.begin(Serial1)
    Serial.println(F("Couldn't find FONA"));
    return false;
  }
  Serial.println(F("FONA is OK"));
  return true;
}

void setup() {
  while (!Serial); // useful for Leonardo/Micro, remove in production!

  // set LED output for debugging
  pinMode(LED, OUTPUT);

  // set up motor pins, but turn them off
  pinMode(MOTOR_1, OUTPUT);
  digitalWrite(MOTOR_1, LOW);
  pinMode(MOTOR_2, OUTPUT);
  digitalWrite(MOTOR_2, LOW);
  pinMode(MOTOR_PWM, OUTPUT);
  digitalWrite(MOTOR_PWM, LOW);

  Serial.begin(115200);
  Serial.println(F("FONA basic test"));

  while (! fonainit()) {
    delay(5000);
  }

  // Print SIM card IMEI number.
  char imei[15] = {0}; // MUST use a 16 character buffer for IMEI!
  uint8_t imeiLen = fona.getIMEI(imei);
  if (imeiLen > 0) {
    Serial.print("SIM card IMEI: "); Serial.println(imei);
  }

  pinMode(FONA_RI, INPUT);
  digitalWrite(FONA_RI, HIGH); // turn on pullup on RI
  // turn on RI pin change on incoming SMS!
  fona.sendCheckReply(F("AT+CFGRI=1"), F("OK"));
}

int8_t lastsmsnum = 0;

void loop() {
digitalWrite(LED, HIGH);
delay(100);
digitalWrite(LED, LOW);

while (fona.getNetworkStatus() != 1) {
    Serial.println("Waiting for cell connection");
    delay(2000);
}

// this is a 'busy wait' loop, we check if the interrupt
// pin went low, and after BUSYWAIT milliseconds break out to check
// manually for SMS's and connection status
// This would be a good place to 'sleep'
for (uint16_t i=0; i<BUSYWAIT; i++) {
    if (!digitalRead(FONA_RI)) {
        // RI pin went low, SMS received?
        Serial.println(F("RI went low"));
        break;
    }
    delay(1);
}

int8_t smsnum = fona.getNumSMS();
if (smsnum < 0) {
    Serial.println(F("Could not read # SMS"));
    return;
} else {
    Serial.println(smsnum);
    Serial.println(F(" SMS's on SIM card!"));
}

if (smsnum == 0) return;

// there's an SMS!
uint8_t n = 1;
while (true) {
    uint16_t smslen;
    char sender[25];
    Serial.print(F("\n\rReading SMS ") ); Serial.println(n);
    uint8_t len = fona.readSMS(n, replybuffer, 250, &smslen); // pass in buffer
    // and max len!
    // if the length is zero, its a special case where the index number is higher
    // so increase the max we'll look at!
    if (len == 0) {
        Serial.println(F("[empty slot]"));
        n++;
        continue;
    }
    if (!fona.getSMSSender(n, sender, sizeof(sender))) {
        // failed to get the sender?
        sender[0] = 0;
    }
    Serial.print(F("***** SMS ") ); Serial.println(n);
    Serial.print(" (") ; Serial.print(len); Serial.println(F(") bytes *****") );
    Serial.println(replybuffer);
    Serial.println(F("From: ") ); Serial.print(sender);
    Serial.println(F("*****") );

    if (strcasecmp(replybuffer, "open sesame") == 0) {
        // open the door!
        digitalWrite(LED, HIGH);
        moveMotor(OPEN, 800);
        digitalWrite(LED, LOW);
    }
    if (strcasecmp(replybuffer, "close cadabra") == 0) {
        // close the door!
        digitalWrite(LED, HIGH);
moveMotor(CLOSE, 800);
digitalWrite(LED, LOW);
}
delay(3000);
break;
} fona.deleteSMS(n);
delay(1000);
}

void moveMotor(boolean direction, uint16_t t) {
digitalWrite(MOTOR_PWM, HIGH);   // enable motor
if (direction) {
  digitalWrite(MOTOR_1, HIGH);
digitalWrite(MOTOR_2, LOW);
} else {
  digitalWrite(MOTOR_2, HIGH);
digitalWrite(MOTOR_1, LOW);
}
delay(t);
digitalWrite(MOTOR_1, LOW);
digitalWrite(MOTOR_2, LOW);
digitalWrite(MOTOR_PWM, LOW);    // disable motor
}

Before uploading code to your Metro, we recommend turning on FONA debugging mode where you can see all the messages sent/received, as it can make it easier to debug! Open up FONA.h in the sketchbook/libraries/Adafruit_FONA library and uncomment this line by removing the // and saving

//#define ADAFRUIT_FONA_DEBUG
Upload the code to your Metro Mini and open up the serial console to see the FONA brought up and connect to the network.

Once its on the network, it will constantly check for SMS messages. It does this in two ways, one it waits for a low pin message on FONA_RI and will instantly bail seeing a new SMS is received. It also manually queries the FONA every few seconds in case the RI pin didn’t fire for some reason.
Try sending a text message, you will see a few seconds after sending, the FONA will see the message and print out the message and the sender's number or email address.
It will then delete the message.

Finally, now that you know it can receive messages, try sending it "Open sesame" or "Close cadabra" to open or close the lock.

Understanding the code in detail

Even tho the code may be updated after this tutorial is written, here's the basic methods to the madness!

Setup and initialization

Here's where we include the required libraries (Software Serial for chatting to the FONA, and the FONA library)
We also define the pins from the Metro to the FONA and TB6612 motor driver. You can change these around to whatever you like. Pin 13 is the onboard LED so you'll want to keep that.

```cpp
#include <SoftwareSerial.h>
#include "Adafruit_FONA.h"

#define FONA_TX 3
#define FONA_RX 4
#define FONA_RST 5
#define FONA_RI 2
#define MOTOR_PWM A1
#define MOTOR_1 A2
#define MOTOR_2 A3
#define LED 13
```

Other defines, we also define which way is 'open' and 'close' for the motor. If you want to have the motor go the opposite way, you can just swap these defines. We also have the amount to wait between querying the FONA for new SMS's.

```cpp
#define OPEN true
#define CLOSE false
#define BUSYWAIT 5000  // milliseconds
```

Next up, we have the buffer for storing SMS messages (they can be long!) and creating the FONA object. If you're using a Micro or Leonardo you could use Serial1 instead of software serial.

```cpp
// this is a large buffer for replies
char replybuffer[255];
// or comment this out & comment above & use a hardware serial port like Serial1 (see below)
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);
Adafruit_FONA fona = Adafruit_FONA(FONA_RST);
```

We have a helper function for initializing the FONA, which we'll call whenever we want to reset the FONA. Normally we do this in setup() only but if the FONA 'hangs' or 'stalls' you can always call this for a hard-reset.

```cpp
boolean fonainit(void) {
  Serial.println(F("Initializing....(May take 3 seconds)"));
  digitalWrite(LED, HIGH);
delay(100);
digitalWrite(LED, LOW);
delay(100);
digitalWrite(LED, HIGH);
delay(100);
digitalWrite(LED, LOW);
delay(100);
```

// make it slow so its easy to read!
void setup() {
    while (!Serial);  // useful for Leonardo/Micro, remove in production!

    // set LED output for debugging
    pinMode(LED, OUTPUT);

    // set up motor pins, but turn them off
    pinMode(MOTOR_1, OUTPUT);
    digitalWrite(MOTOR_1, LOW);
    pinMode(MOTOR_2, OUTPUT);
    digitalWrite(MOTOR_2, LOW);
    pinMode(MOTOR_PWM, OUTPUT);
    digitalWrite(MOTOR_PWM, LOW);

    Serial.begin(115200);
    Serial.println(F("Open Sesame! An SMS-controlled door lock"));
}

It then calls our helper to boot up the FONA, check the SIM card is connected, and then tells the FONA to alert us when an SMS comes in by pulsing the RI pin low with the AT+CFGRI=1 command. (The RI pin is set to an input as well)

while (! fonainit()) {
    delay(5000);
}

// Print SIM card IMEI number.
char imei[15] = {0}; // MUST use a 16 character buffer for IMEI!
uint8_t imeiLen = fona.getIMEI(imei);
if (imeiLen &gt; 0) {
    Serial.print("SIM card IMEI: "); Serial.println(imei);
}

pinMode(FONA_RI, INPUT);
digitalWrite(FONA_RI, HIGH); // turn on pullup on RI
// turn on RI pin change on incoming SMS!
cona.sendCheckReply(F("AT+CFGRI=1"), F("OK"));
Loop()

In the loop we start out by blinking the LED (lets us visually tell the loop is running) and checking the FONA is up and running and connected to the cellular network

```cpp
void loop() {
    digitalWrite(LED, HIGH);
    delay(100);
    digitalWrite(LED, LOW);
    while (fona.getNetworkStatus() != 1) {
        Serial.println("Waiting for cell connection");
        delay(2000);
    }
}
```

We then have the SMS waiting loop, we wait for either the RI pin to pulse low, or after 3 seconds, we manually check

```cpp
// this is a 'busy wait' loop, we check if the interrupt
// pin went low, and after BUSYWAIT milliseconds break out to check
// manually for SMS's and connection status
// This would be a good place to 'sleep'
for (uint16_t i=0; i<BUSYWAIT; i++) {
    if (! digitalRead(FONA_RI)) {
        // RI pin went low, SMS received?
        Serial.println(F("RI went low"));
        break;
    }
    delay(1);
}
```

then we check for how many SMS's are received! If we have 0 SMS's, we bail

```cpp
int8_t smsnum = fona.getNumSMS();
if (smsnum &lt; 0) {
    Serial.println(F("Could not read # SMS"));
    return;
} else {
    Serial.println(smsnum);
    Serial.println(F(" SMS's on SIM card!"));
}
if (smsnum == 0) return;
```

If there is an SMS available, we go thru all the slots looking for it. SMS's are stored in slots but if you get a 2 SMS's and delete the first one, the second one doesn't "move up" a slot, so you have to check each one until you find the SMS you're looking for. Thats why we have the n++ and continue if we get an "[empty slot]"

```cpp
// there's an SMS!
uint8_t n = 1;
while (true) {
    uint16_t smslen;
    char sender[25];
```
Serial.print(F("\n\rReading SMS ")); Serial.println(n);
uint8_t len = fona.readSMS(n, replybuffer, 250, &smslen); // pass in buffer and max len!
// if the length is zero, its a special case where the index number is higher
// so increase the max we'll look at!
if (len == 0) {
  Serial.println(F("[empty slot]"));
  n++;
  continue;
}
if (! fona.getSMSSender(n, sender, sizeof(sender))) {
  // failed to get the sender?
  sender[0] = 0;
}

OK now we have the SMS message read and also know the sender. Print them out!

Serial.print(F("***** SMS #")); Serial.print(n);
Serial.print(" ("); Serial.print(len); Serial.println(F(" bytes *****"));
Serial.println(replybuffer);
Serial.print(F("From: "); Serial.println(sender);
Serial.println(F("*****"));

Now we can do something based on the SMS text message. We use strcasecmp for a case-insensitive comparison. You can also adapt this code to look for substrings or parse out values, we're keeping it simple! One command for opening, one for closing. Afterwards, we delay 3 seconds and restart the loop.

if (strcasecmp(replybuffer, "open sesame") == 0) {
  // open the door!
  digitalWrite(LED, HIGH);
  moveMotor(OPEN, 800);
  digitalWrite(LED, LOW);
}
if (strcasecmp(replybuffer, "close cadabra") == 0) {
  // close the door!
  digitalWrite(LED, HIGH);
  moveMotor(CLOSE, 800);
  digitalWrite(LED, LOW);
}
delay(3000);
break;
}

Finally we delete the SMS, wait another second, and restart the loop

fona.deleteSMS(n);
delay(1000);
moveMotor helper

At the very end we have our motor helper. This is a basic function that turns on the PWM pin to 'activate' the motor (its only active for the movement, then we turn off the motor to conserve power) and then turn it forward or backwards for t milliseconds.

```c
void moveMotor(boolean direction, uint16_t t) {
  digitalWrite(MOTOR_PWM, HIGH);   // enable motor
  if (direction) {
    digitalWrite(MOTOR_1, HIGH);
    digitalWrite(MOTOR_2, LOW);
  } else {
    digitalWrite(MOTOR_2, HIGH);
    digitalWrite(MOTOR_1, LOW);
  }
  delay(t);
  digitalWrite(MOTOR_1, LOW);
  digitalWrite(MOTOR_2, LOW);
  digitalWrite(MOTOR_PWM, LOW);  // disable motor
}
```

Final fit

OK now that it works great on a breadboard, time to turn it into a single packaged project! The Lockitron body has just enough space to fit the FONA, Metro Mini, and battery. I did need to trim some plastic off the battery holder and be a little careful with placement but it did fit!

Freewiring

The breadboard will definitely not fit inside the enclosure, so instead of breadboarding it, we use these nice silicone cover wires which are very flexible and strong (). Begin by freewiring the Metro Mini to the FONA, and go through all the tests making sure the Open Sesame sketch still works.
Then add in the TB6612 motor driver, you'll need to solder directly to the wires in the Lockitron.

Run the test again, making sure the motor moves, etc!

Fitting

Then you can start fitting in parts. We covered the solder points on the bottom of the breakouts with Gaffer's tape to protect them since they're all sort of squeezed into that corner spot!
Start with the motor driver, it goes right into the cavity next to the AA holder. Don’t forget to cover the battery terminals so they don’t short against the motor driver!

The Metro mini fits alongside next to the motor driver, with the MicroUSB jack pointing towards the corner (with a little filing and gluing it could be a programming slot!) and then the FONA slides in right next to it too. We removed a little plastic board-slot jig that was used for the original PCB to get the FONA to sit flat.

To fit the battery in, we cut off a bit of the battery housing plastic and slid the battery alongside the battery casing. The antenna is then stickered onto the battery case covering which means there’s no metal around it to attenuate signal.
Then cover it up with the plastic casing! Try it again and deploy onto a local door