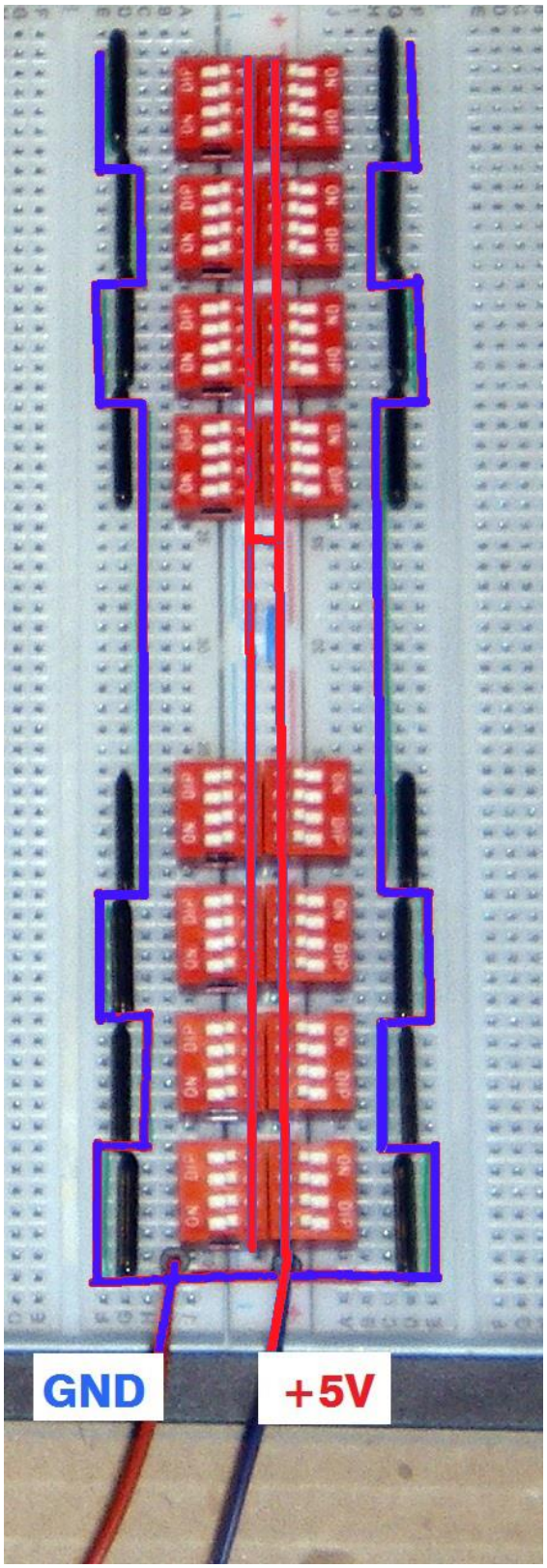
DIP/DIL Switch-based 8-Byte ROM via daisy-chained 4051 multiplexers and Arduino



Originally conceived as a tribute to the hand-sewn Core Rope memories of the Apollo Guidance Computer, my idea was successively simplified to "find a way to store a short character string in a series of DIP switches, then retrieve and display that text via an Arduino UNO".

I was inspired by (or better: I plagiarized) the project described here: https://forum.arduino.cc/index.php?topic=446780.0 : the prototype is based on a very simple circuit and nine 8-channel multiplexers in a "MUX daisy chain" arrangement (one multiplexer drives the other eight). Considering that each 4051 IC can manage up to 8 lines, the total number of bits in the ROM will be 64 (8 Byte), with each bit set independently via its own dip switch. A separate 4051 multiplexer reads one by one the bits in each Byte, the remaining one selects which Byte to read.

The ROM will be tested by storing 8 Byte of text as ASCII characters, that will be displayed by Arduino on a 2 line LCD display.

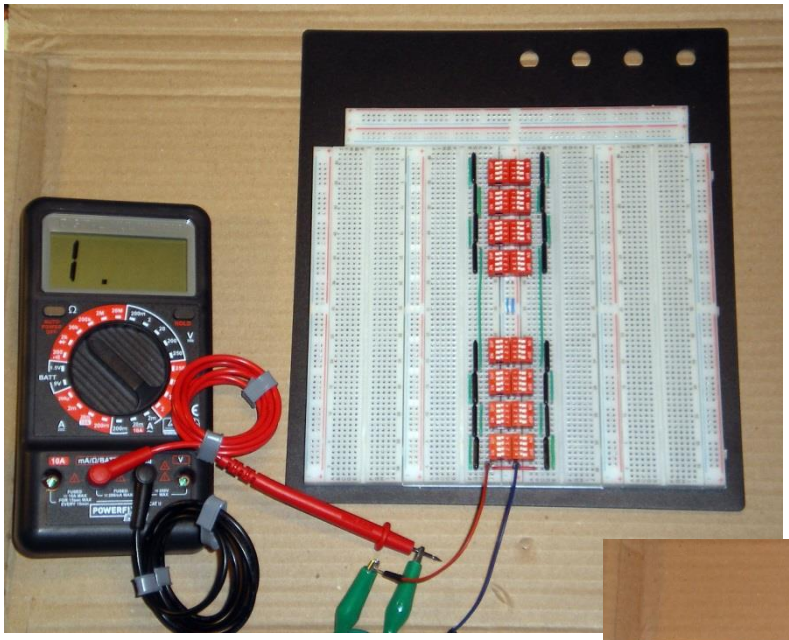
For each Byte, two banks of 4-bit DIP switches are provided. The choice of 4-switch packages has been dictated only by economic convenience (...a sale).

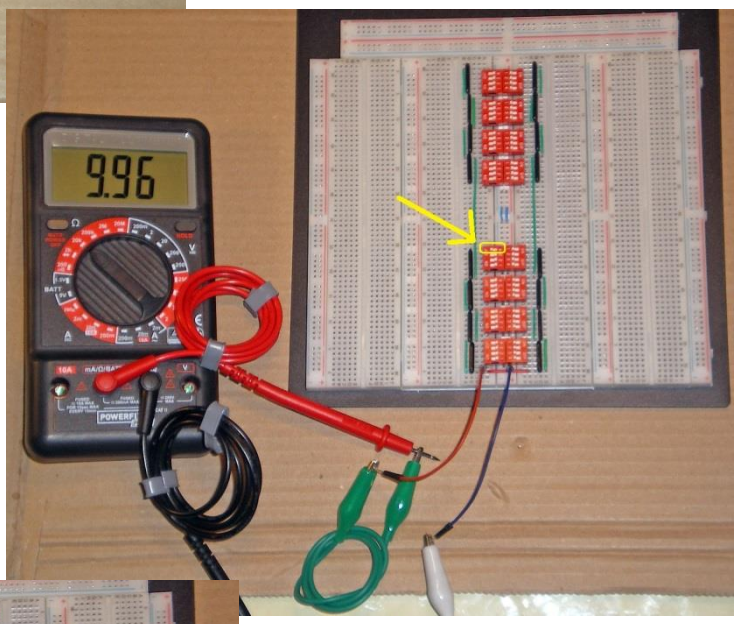
Although I expected that - to allow correct voltage reading by non-sinking IC's - the ROM banks should be independently powered by a separate 5V DC supply, it became apparent that equally successful readings can be obtained by powering the inner ROM circuit and the outer circuit with multiplexers, by two independent +5V/GND pins on the same Arduino.

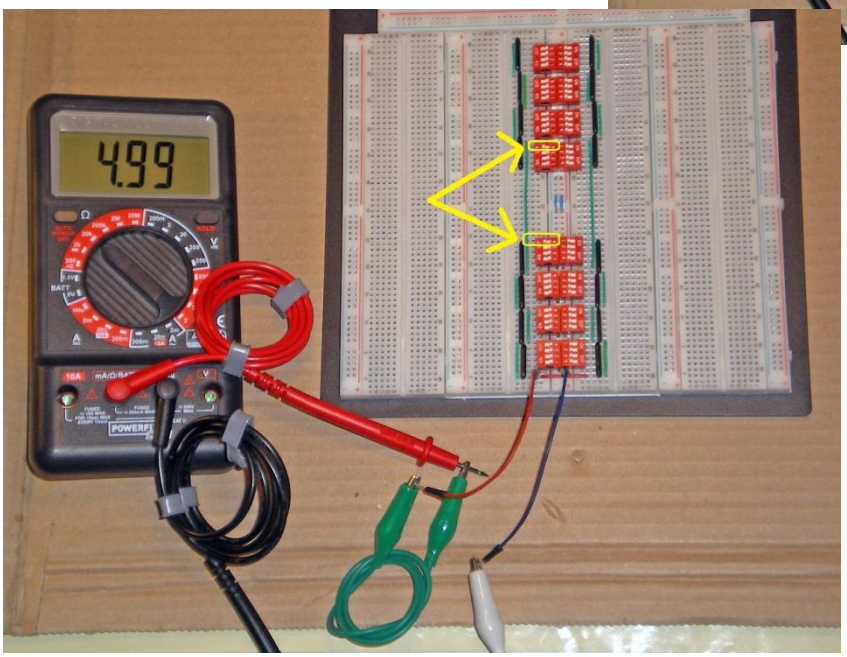
To make wiring easier, bussed 10KΩ resistive network (5 pins +common, also a bargain: in fact, I would have needed just 4 pins+common) have been used.

The 4096 points breadboard has been exploited in an unusual way: in particular, the two power rails along the midline have been connected and used as common exit for all the DIP switches.This arrangement ruled out the possibility to power each bank the easy way, via one connection per resistive network to one of the breadboard power rails. The GND line was propagated as illustrated in the picture at left. Perhaps you'll find immediatly the traces of an huge mistake on my part: the correct circuit has +5V connected to the inner ring, grounded resistor chains on the outer ring, and each closed switch lets the current flow and is read as "high".

Inner circuit continuity was tested very easily: with all switch open, the circuit as a whole is open. Closing any single switch gives a reading of 10 KΩ, closing any two gives a reading of 5 KΩ, and so on.



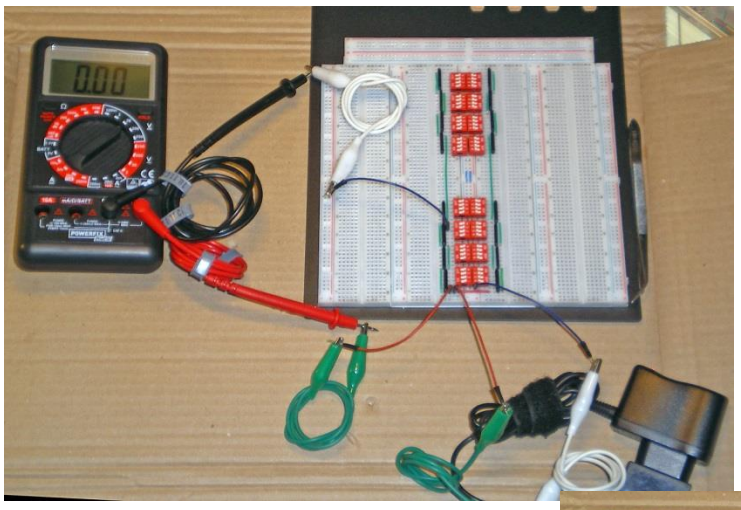




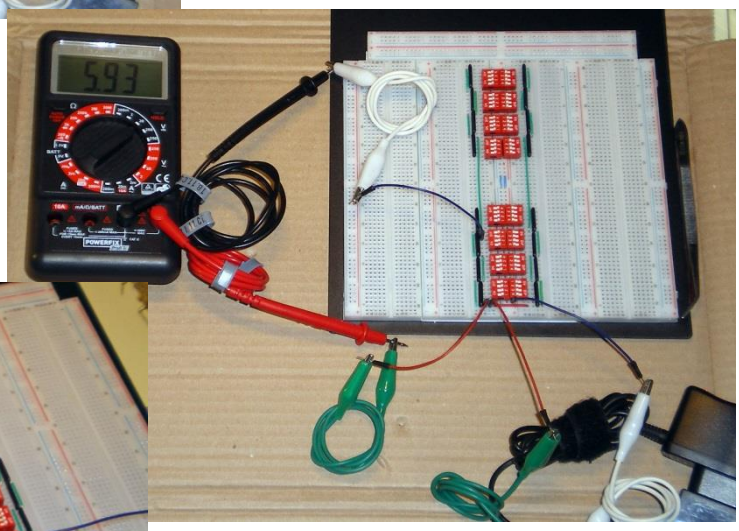
Considering that the whole project is about scanning bit by bit the ROM, to get an "high" reading for each closed switch (high bit), I need a circuit where the voltage is affected as little as possible by the number of closed switches. Lacking basic notions, I needed to check the voltage in different ROM configurations.

It took a little time for me to understand how I could I check the voltage. Then I grasped that, to let the current flow through my tester only in case a switch was closed, I needed to let the current from the power supply pass through the tester, with the tester's common ground connected uphill from the switch that I wanted to test.

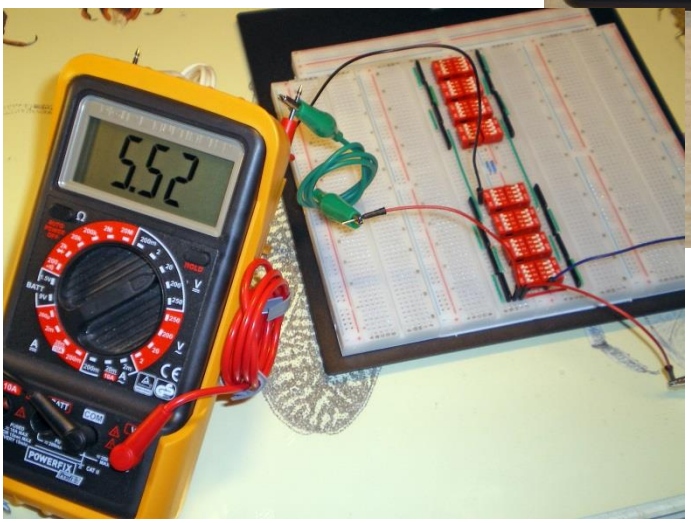
Quite obviously, any open switch wouldn't let the current flow. Each additional closed switch apparently causes a voltage drop of just a few mV. This conclusion may determine whether or not to use digital or analog Arduino ports.



Left: all switches closed, no current flow in any line

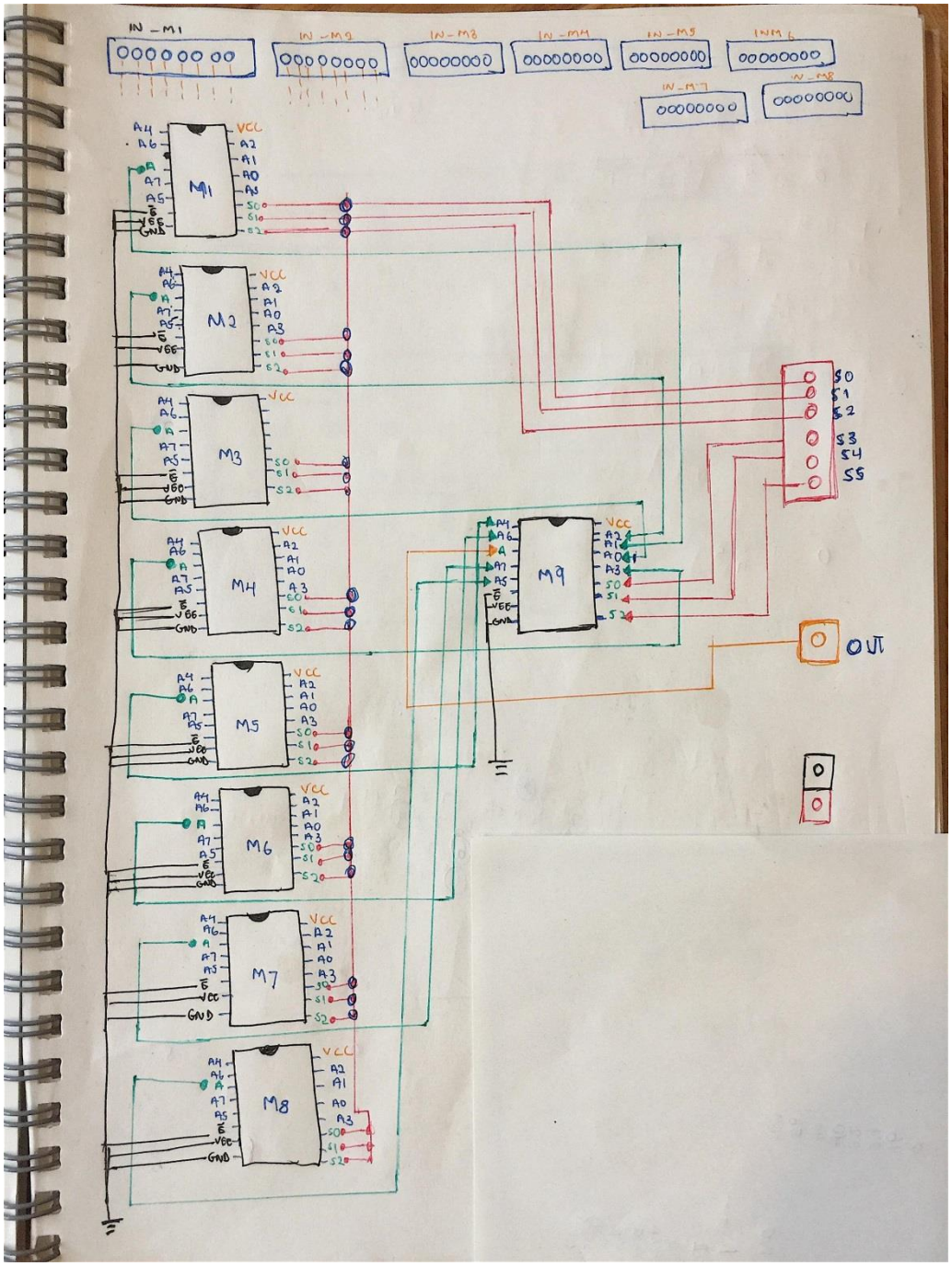


Right: one switch closed, 5.93 V reading



Left: all switches closed, 5.52 V reading in any line

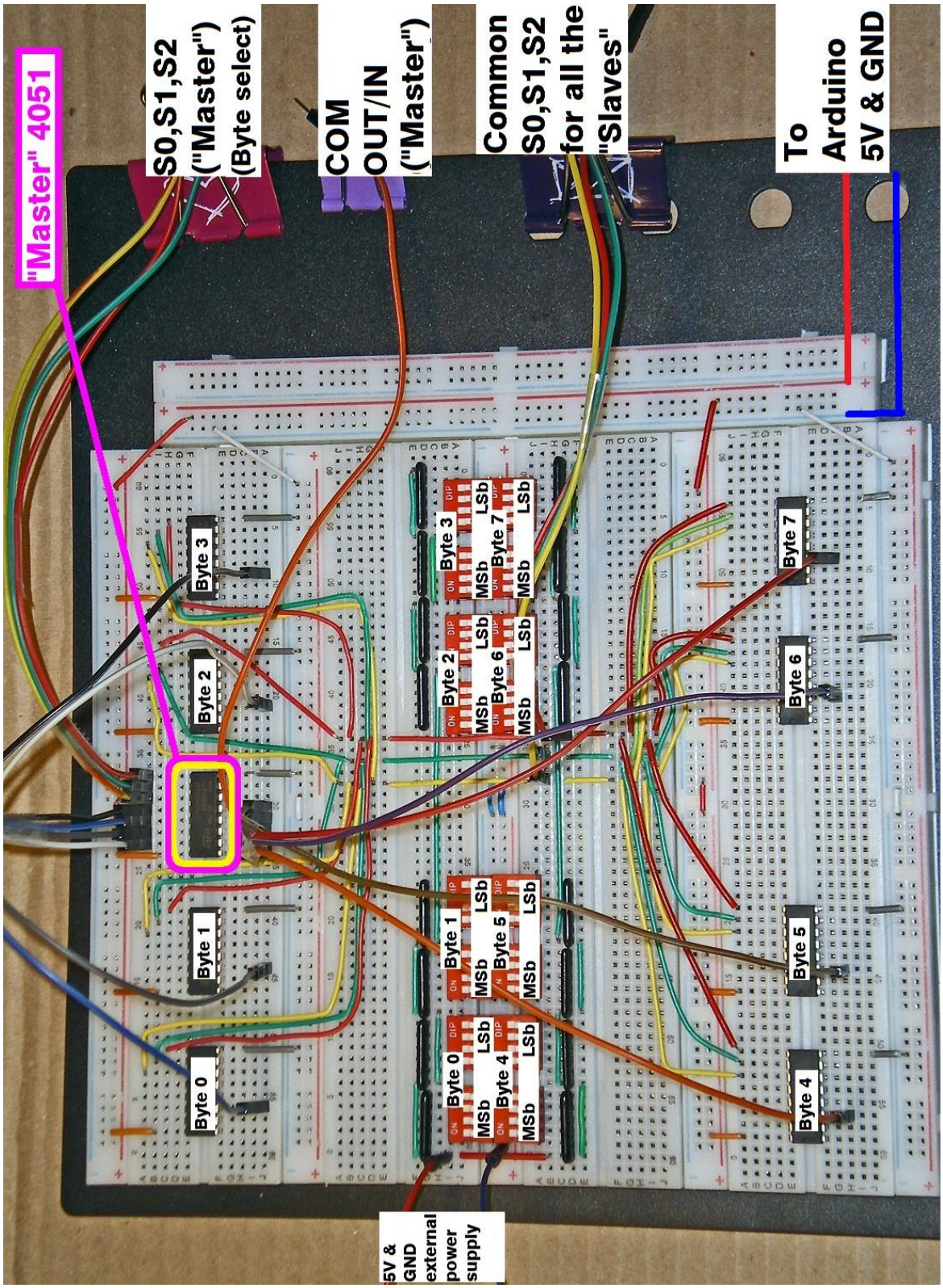
The "control stage" (here also cited as "outer circuit") by daisy-chained multiplexers was inspired by this schematics by PaulRB from the following Arduino forum post: https://forum.arduino.cc/index.php?topic=446780.0



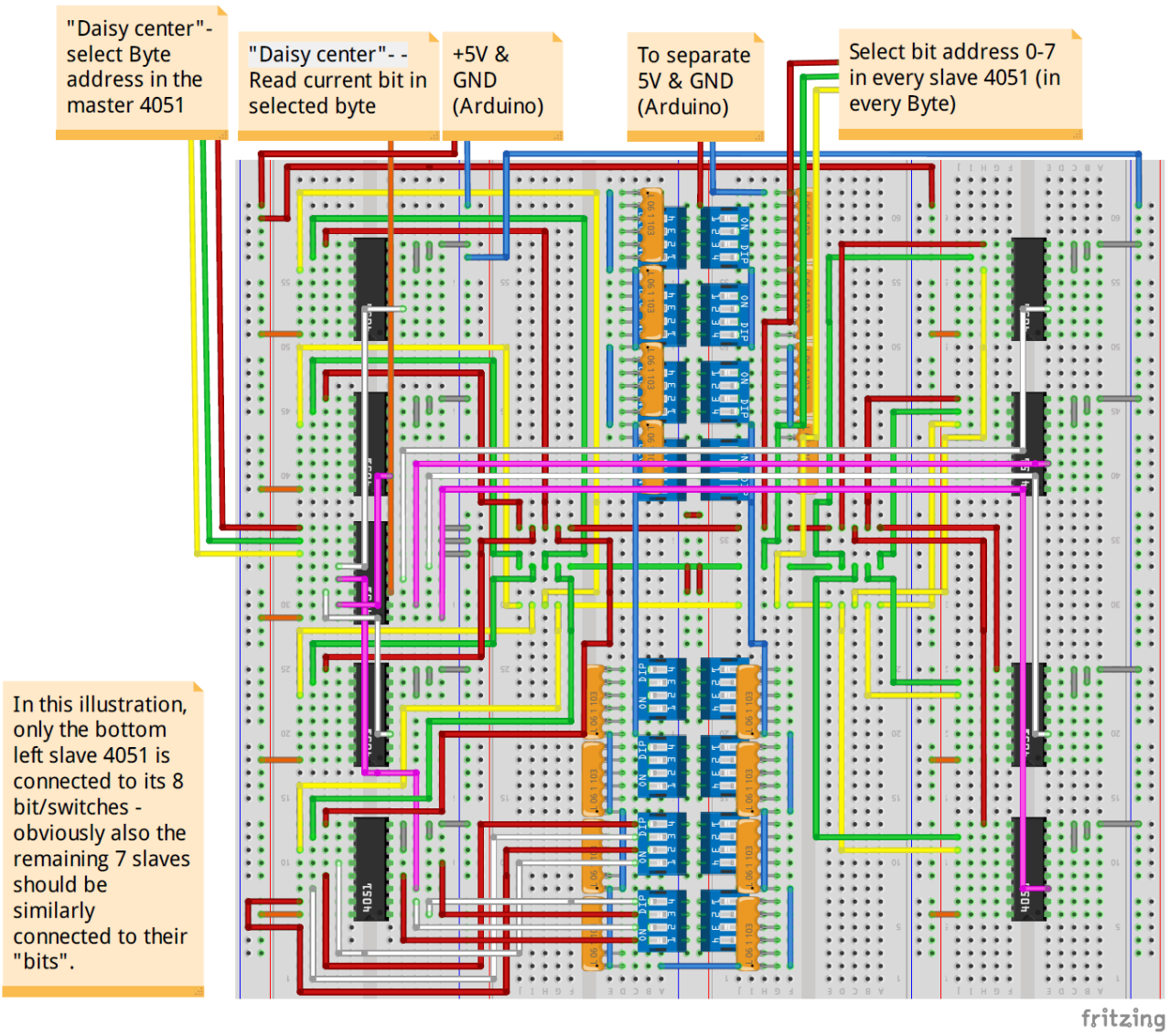
where Arduino is used to control 64 lines in a two-step process:

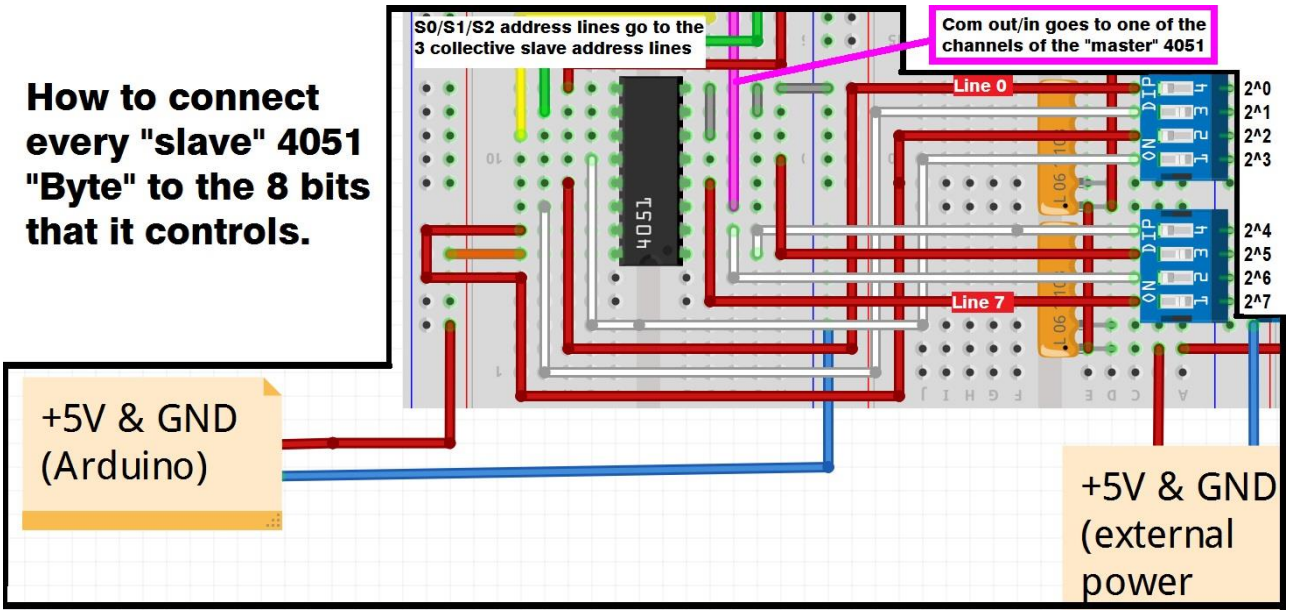
1) (using the three address lines of the "master") select one of the "slave" 4051 IC's (one per "Byte") via the three lines S0/S1/S2 of the "master" 4051
2) (using the three address lines shared by all the "slaves") considering that all the S0, the S1 and the S2 pins of the "Bytes" are in common, regardless of which "Byte" is selected, it suffices to cycle through the eight available addresses: only the selected "Byte" will answer, and make a reading for each bit
3) go to 1) and select the next "Byte"

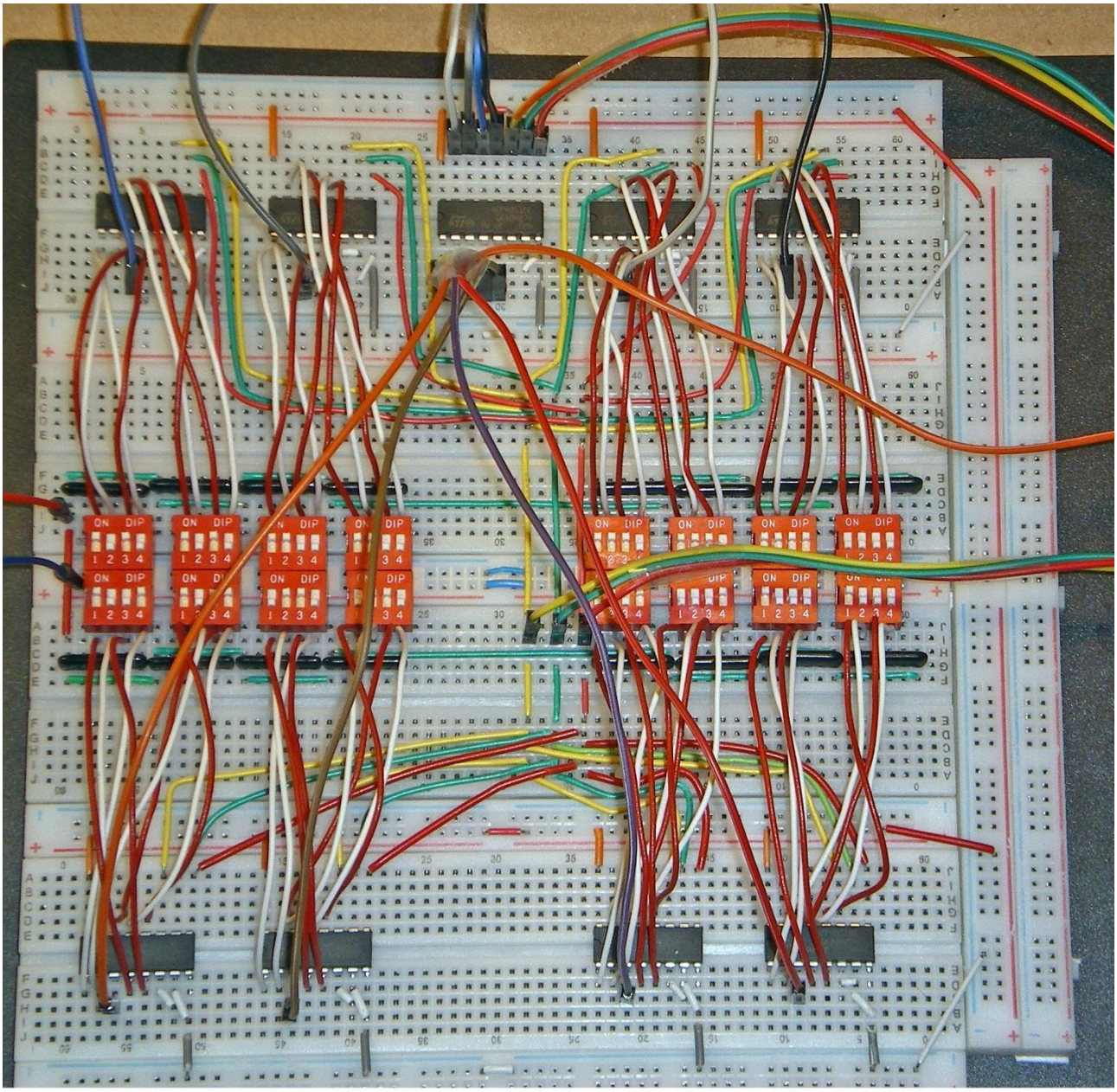
I breadboarded the schematics as follows: the photo and the Fritzing prototype do not include the lines between each "slave" 4051 (Byte) and its 8 DIP switches ("bits"), that are obviously needed to make the prototype work but that would make the picture unreadable. Also the connection to ground of pin 6&7 in each 4051 are missing from the photo, as well as the "bridges" to restore continuity of the topmost horizontal power rails, interrupted at midway.



Breadboard Prototype before connecting each "slave" 4051 (Byte) to its eight switches (bits)


"Daisy center"- select Byte address in the master 4051
"Daisy center"- - Read current bit in selected byte
+5V & GND (Arduino)
To separate 5V & GND (Arduino)
Select bit address 0-7 in every slave 4051 (in every Byte)
In this illustration, only the bottom left slave 4051 is connected to its 8 bit/switches - obviously also the remaining 7 slaves should be similarly connected to their "bits".
fritzing


How to connect every "slave" 4051 "Byte" to the 8 bits that it controls.
S0/S1/S2 address lines go to the 3 collective slave address lines
Com out/in goes to one of the channels of the "master" 4051
Line 0
Line 7
2^0
2^1
2^2
2^3
2^4
2^5
2^6
2^7
+5V & GND (Arduino)
+5V & GND (external power



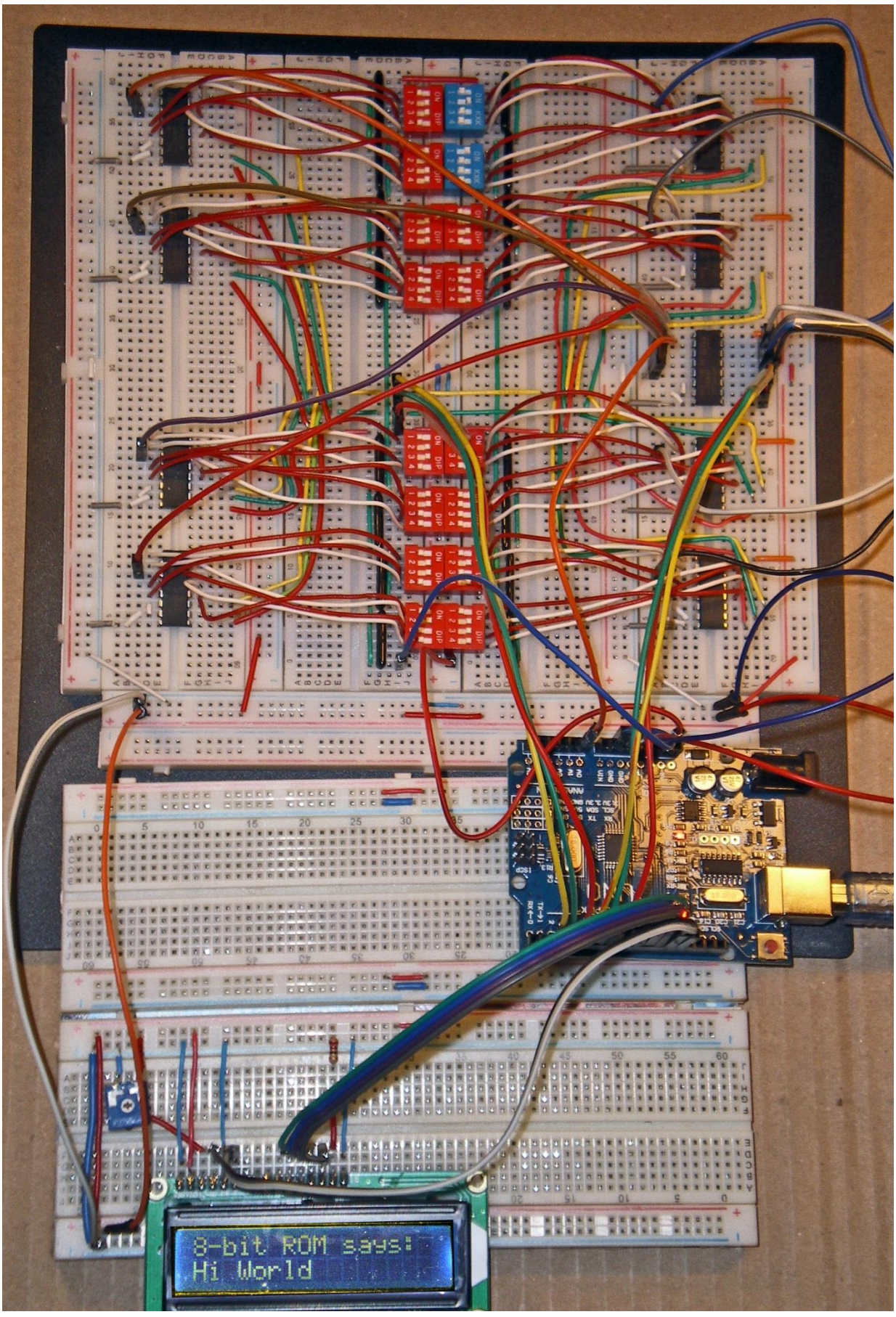
The full breadboard with "slave" 4051's connected to their respective "bits"

The picture above was taken with the inner circuit still attached to an external power supply with reversed polarity.
I want to state again that:

1) The dip switches should be powered with +5V running along the inner rails at the center of the breadboard, and the outer ring made of interconnected resistive networks.
2) Both the circuits (the one that powers the DIP switch array, and the one that powers and connects the daisy-chained multiplexers) may well be powered by the same Arduino, provided that a separate +5V/GND couple of pins is used for each circuit.

I add that I'm well aware that I could have performed a digital read of the switch status instead of an analog read. Analog read was a "parachute" solution when I still wasn't sure whether closing many switches would cause a voltage drop that would have prevented reaching the minimum voltage for digital HIGH. Even though the circuit now is safe in that respect, I retained the conservative analog approach.

Here is a picture of the fully assembled prototype, including an LCD display that shows the content of the 8-Byte ROM



THE SOFTWARE

Here is the Arduino source code:

```
/*

DIP/DIL Switch-based 8-Byte ROM via daisy-chained 4051 multiplexers and Arduino

Circuit and comments:
See http:_____________________________
Circuit is as illustrated here:_____________________________

 created 03 Feb 2019
 by Cesare Brizio
 modified ----

This example code is in the public domain.

Originally conceived as a tribute to the hand-sewn Core Rope memories of the
Apollo Guidance Computer,
my idea was successively simplified to “find a way to store a short character
string in a series of
DIP switches, then retrieve and display that text via an Arduino UNO”.

I was inspired by (or better: I plagiarized) the project described here:
https://forum.arduino.cc/index.php?topic=446780.0

I have 64 DIP switches, each one capable of storing a bit of information, in
eaght groups of eight switches
(closed = non-null voltage reading = binary 1 - opened = null voltage reading =
binary 0)

Each group of eight switches, that I will call a "Byte", is connected to the
eight I/O channels of a
separate "slave" 4051.

The three address lines S0/S1/S2 of all the slave multiplexers are common: one
S0, one S1, and one S2 address all
the slave multiplexers simultaneously

I am using a master multiplexer 4051 whose I/O channels are connected to the com
out/in of each slave multiplexer
Thus, by a two-step process:
a) select the "Byte" (=slave multiplexer) to read via the S0/S1/S2 of the master
4051
b) select which "bit" to read from each slave via the common S0/S1/S2 of the
slave 4051's
I can access one switch at a time and make individual reading.

As each Byte is meant to store an 8-bit ASCII character, I compose the Bytes by
the successive readings from
their bits, and translate them into ASCII values, that are added to the string.

The string obtained is displayed on an LCD display driven by Arduino.

By changing the open/closed DIP switch configuration, I can vary the displayed
string in real time.


  ==========================
  The circuit for LCD board:
  ==========================
 * LCD RS pin to digital pin 15
```

```
 * LCD Enable pin to digital pin 14
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * LCD VSS pin to ground
 * LCD VCC pin to 5V
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)

*/
// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// "Slave" 4051 ADDRESS PINS :
int slave_S0 = 2;
int slave_S1 = 3;
int slave_S2 = 4;

// "Master" 4051 ADDRESS PINS :
int master_S0 = 5;
int master_S1 = 6;
int master_S2 = 7;

//
int analogPin = A0;

int ReadVolt = 0;

int bitValue;

int asciiCode;

char asciiChar ="";

int powerOfTwo = 0;

char asciiString[8];



void setup() {
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    // Print a message to the LCD.
    lcd.print("8-bit ROM says: ");
    // CONFIGURE ADDRESS PINS
    pinMode(master_S0, OUTPUT);
    pinMode(master_S1, OUTPUT);
    pinMode(master_S2, OUTPUT);
    pinMode(slave_S0, OUTPUT);
    pinMode(slave_S1, OUTPUT);
    pinMode(slave_S2, OUTPUT);

  Serial.begin(9600);

}
```

```
void loop() {

    for( int x = 0; x < 8;  ++x )
          asciiString[x] = (char)0;


    // For each "Byte"
    for (int i = 0; i < 8; i++) {

        // Select current Byte
        cycle4051('M',i);

        delay(5);

        //Serial.print("Byte ");
        //Serial.print(i);
        //Serial.println(" readings:");
        //Serial.print(">");

        asciiCode=0;

        for (int j = 7; j >=0; j--) {

            // Select current bit
            cycle4051('S',j);

            delay(5);

            ReadVolt = analogRead(analogPin);

            if(ReadVolt>500) // check if current is running
            {
                bitValue=1; // Switch closed, bit high
                asciiCode = asciiCode + powerOfTwo;
            }
            else
            {
                bitValue=0; // Switch open, bit low
            }

            //Serial.print(bitValue);
            //Serial.print(ReadVolt);
            //Serial.print("|");

        }

        //Serial.print(asciiCode);
        //Serial.println("|");
        asciiString[i]=asciiCode;


    }
    //Serial.println("--- End series");
    //Serial.println(asciiString);

    for( int z = 0; z < 8;  ++z )
        {
        // (note: line 1 is the second row, since counting begins with 0):
        lcd.setCursor(z, 1);
        // print on the LCD the current character from the string:
        lcd.print(asciiString[z]);
```

```
        // print it also on the Serial Monitor
        Serial.print(asciiString[z]);
        }
    Serial.println("");
    delay(10);
}


void cycle4051(char master_slave, int stepNumber) {
     // It's very obvious that this subprogram could have been written in
     // a much more compact form. I prefer to use a more verbose style
     // for clarity, readability and ease of maintenance.
     // I do know that there are may valid alternatives to the stratagem of
     // using the powerOfTwo variable - the native POW() exponentiation function
     // that I could well have used has its drawbacks.
     switch (master_slave) {
        case 'M':
            //Next "Byte" (= next slave 4051)
            //Select the next slave 4051, connected to one of the channels
              of the master 4051
            switch (stepNumber) {
               case 0:
                   //Select Channel 0 of the master 4051
                   digitalWrite(master_S2, LOW);
                   digitalWrite(master_S1, LOW);
                   digitalWrite(master_S0, LOW);
                   break;
               case 1:
                   //Select Channel 1 of the master 4051
                   digitalWrite(master_S2, LOW);
                   digitalWrite(master_S1, LOW);
                   digitalWrite(master_S0, HIGH);
                   break;
               case 2:
                   //Select Channel 2 of the master 4051
                   digitalWrite(master_S2, LOW);
                   digitalWrite(master_S1, HIGH);
                   digitalWrite(master_S0, LOW);
                   break;
               case 3:
                   //Select Channel 3 of the master 4051
                   digitalWrite(master_S2, LOW);
                   digitalWrite(master_S1, HIGH);
                   digitalWrite(master_S0, HIGH);
                   break;
               case 4:
                   //Select Channel 4 of the master 4051
                   digitalWrite(master_S2, HIGH);
                   digitalWrite(master_S1, LOW);
                   digitalWrite(master_S0, LOW);
                   break;
               case 5:
                   //Select Channel 5 of the master 4051
                   digitalWrite(master_S2, HIGH);
                   digitalWrite(master_S1, LOW);
                   digitalWrite(master_S0, HIGH);
                   break;
               case 6:
                   //Select Channel 6 of the master 4051
                   digitalWrite(master_S2, HIGH);
                   digitalWrite(master_S1, HIGH);
                   digitalWrite(master_S0, LOW);
                   break;
```

```
                case 7:
                    //Select Channel 7 of the master 4051
                    digitalWrite(master_S2, HIGH);
                    digitalWrite(master_S1, HIGH);
                    digitalWrite(master_S0, HIGH);
                    break;
            }
            break;
        case 'S':
            //Next "bit" (= next channel in the currently selected slave 4051)
            //Select the next channel of the slave 4051, connected to one of the
dip switches
            switch (stepNumber) {
                case 0:
                    //Select Channel 0 of the slave 4051
                    digitalWrite(slave_S2, LOW);
                    digitalWrite(slave_S1, LOW);
                    digitalWrite(slave_S0, LOW);
                    powerOfTwo=1;
                    break;
                case 1:
                    //Select Channel 1 of the slave 4051
                    digitalWrite(slave_S2, LOW);
                    digitalWrite(slave_S1, LOW);
                    digitalWrite(slave_S0, HIGH);
                    powerOfTwo=2;
                    break;
                case 2:
                    //Select Channel 2 of the slave 4051
                    digitalWrite(slave_S2, LOW);
                    digitalWrite(slave_S1, HIGH);
                    digitalWrite(slave_S0, LOW);
                    powerOfTwo=4;
                    break;
                case 3:
                    //Select Channel 3 of the slave 4051
                    digitalWrite(slave_S2, LOW);
                    digitalWrite(slave_S1, HIGH);
                    digitalWrite(slave_S0, HIGH);
                    powerOfTwo=8;
                    break;
                case 4:
                    //Select Channel 4 of the slave 4051
                    digitalWrite(slave_S2, HIGH);
                    digitalWrite(slave_S1, LOW);
                    digitalWrite(slave_S0, LOW);
                    powerOfTwo=16;
                    break;
                case 5:
                    //Select Channel 5 of the slave 4051
                    digitalWrite(slave_S2, HIGH);
                    digitalWrite(slave_S1, LOW);
                    digitalWrite(slave_S0, HIGH);
                    powerOfTwo=32;
                    break;
                case 6:
                    //Select Channel 6 of the slave 4051
                    digitalWrite(slave_S2, HIGH);
                    digitalWrite(slave_S1, HIGH);
                    digitalWrite(slave_S0, LOW);
                    powerOfTwo=64;
                    break;
                case 7:
```

```
                    //Select Channel 7 of the slave 4051
                    digitalWrite(slave_S2, HIGH);
                    digitalWrite(slave_S1, HIGH);
                    digitalWrite(slave_S0, HIGH);
                    powerOfTwo=128;
                    break;
                }
                break;
    }


}
```

Cesare Brizio, 3 February 2019