Learning arduino with the Magnolia

By @raspofabs

Table of Contents

Learning arduino with the Magnolia

This gitbook is a guide for beginners who have their hands on a Magnolia board from the L.A. Robotics club, originally developed as a Kickstarter campaign to produce an Arduino inspired robotics starter kit for a very meagre sum of $5 per board.

This guide is for those who already have their board, have an FTDI board, and a USB cable.

The lessons cover the very basics of Arduino, and hopefully also give good foundations in electronics that will help you search and learn more easily on your own.

In addition to basic tutorials on arduino related functions, there are also lessons on working around the limitations of the Magnolia board (such as not having PWM, or having only 6 IO ports.)

The original project: https://www.kickstarter.com/projects/annikaskywalker/microprocessor-about-the-cost-and-size-of-a-pack-o

Magnolia is an Open Hardware project, the schematics are available here http://www.annikaobrien.com/magnolia

A subreddit has formed: http://www.reddit.com/r/MagnoliaBoard

Basic requirements

Basic requirements

Hardware

In addition to the Magnolia, you're going to need a 5V happy LED. A lot of 3V+ LEDs can handle the 5V from an arduino, but lower voltage ones tend to shine bright and short when powered up. It's best if you prepare an LED with a 100Ohm resister in series (tied to one leg). You're going to need a push button switch of some sort, a high value 2kOhm-2MOhm resistor, and for later lessons there will be a need for other components.

- a whole bunch of LEDs
- resistors (100Ohm, 2kOhm and 4kOhm)
- prototyping board (optional, but really useful) and jumper cables.
- 8-bit shift register (HC595)
- photoresistor (light sensitive resistor)
- hobby servo
- piezo or other low resistance speaker
- 8-bit Parallel to serial (shift in) (CD4021)

Software

You're going to need to download and install the arduino IDE software from Arduino.cc

Once you have that, you need to plug the FTDI into your computer to find out what you need to do tto fixup any driver issues with talking to serial tty devices.

Windows 7 needs to use drivers that come with the arduino software. If they are not installed correctly on first attempt, then try these steps.

- open windows device manager
- look for FTDI / USB Serial / FTRS232 something like that with a warning icon
- reinstall the driver, browse to where arduino installed (C:\program files (x86)\arduino\drivers\...)
- Sometimes you will need to do this for two devices, not just the one, and the second one doesn't turn up until the first one has working drivers.

Linux needs to set your user as permitted to talk to tty and dialout.

- sudo usermod -a -G dialout
- sudo usermod -a -G tty

First steps

First steps

We're going to take one of the example sketches that comes with the arduino software and adjust it to match the Magnolia. The main thing to take from this lesson is that there is no pin **13** on the Magnolia, only analog pins, namely **A0** - **A5**.

Basic setup

Set up your magnolia by connecting the FTDI to the board, then connect the USB cable to the FTDI thingy.

Take an LED and stick the two legs in **GND** and **VCC**. It should light up. If not, try it the other way around.

Take note of what the LED looks like, and which way round it goes, they only work one way round. Normally, the long leg is positive, or *anode*. Inside the casing, you can see the larger internal part (the base) is attached to the shorter leg. The big metal inside is the negative or *cathode*.

You're now ready to pull those legs apart a bit and put the positive (anode) into one of the unmarked slots on the Magnolia. The unmarked slots are the same as the analogue input pins on a regular Arduino. **A0** is the pin nearest the markings for **GND** and **VCC**, **A5** is at the other end.

If you hold up your Magnolia board with the FTDI and USB cable off to the right, then this grid is the names of the 3x6 holes on the left side of the board.

```
+----------------------------+
|GND VCC A5 +----------------+|
|GND VCC A4 |  ATMega328     ||
|GND VCC A3 +----------------+| -> FTDI
|GND VCC A2                  |
|GND VCC A1      Magnolia    |
|GND VCC A0        Board     |
+----------------------------+
```

If you drop the anode into **A0** and the cathode into one of the **GND** pins, then the LED should not be lit up

Modifying an example

Now to the coding. Open up the arduino IDE, then open the blink example `file` > `examples` > `01.basics` > `blink` and look for the line: int led = 13; and change the `13` to `A0`. After this, the sketch should now be able to blink your LED. You need to upload it.

Uploading a sketch

The Arduino IDE has two round buttons (one a tick, the other an arrow) for checking your code, and uploading your code. This sketch should compile fine, so no need to check it compiles (especially as it will do it as part of the upload anyway) but something you do need to do is tell it what type of arduino you're using.

from the `tools` menu, select the `board` that matches yours. In this case the `Arduino Duemilanove w/ ATmega328`. Once you've done this you won't need to change the board unless you get another and different Arduino.

Hit the upload button (the arrow), and you should see the IDE say that it is compiling, then when it hits uploading, you should see the lights on your FTDI / Magnolia light up as the program is transferred to the chip. Once the upload is complete, the LED should start to blink on and off with a one second delay.

Multiple lights

Multiple lights

Adding more is the next step. We're going to go over what you need to do to modify and extend an Arduino sketch.

The code

reopen and modify the blink sketch like you did in lesson 1 if you have shut down the IDE, then where it now says

```
int led = A0;
```

you need to add more lines:

```
int led = A0;
int led2 = A1; // second LED
int led3 = A2; // third LED
```

and you need to set the pin mode for where we're going to connect the LEDs, so go to `void setup()`, and modify it.

```
pinMode(led, OUTPUT);
pinMode(led2, OUTPUT);
pinMode(led3, OUTPUT);
```

and finally you need to modify the content of `void loop()` from this:

```
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
```

to this:

```
digitalWrite(led, HIGH); // turn on LED1
delay(1000);
digitalWrite(led, LOW); // then turn it off
digitalWrite(led2, HIGH); // and turn on LED2
delay(1000);
digitalWrite(led2, LOW); // then turn it off
digitalWrite(led3, HIGH); // and turn on LED3
delay(1000);
digitalWrite(led3, LOW); // then turn it off
```

Notice we don't add a delay between setting `led` to `LOW` and setting `led2` to `HIGH`. This is because we want the second LED to light up as soon as the first one has gone out, making it look like the light is moving from one LED to another.

Wiring

Add another two LEDs with their anodes (positive legs) in **A1** and **A2**. Upload the sketch and you should see the three LEDs light up in sequence.

The take-away

Whenever you add a new output to your sketch, you need to make sure you set the `pinMode` for the pin you are going to be using. Usually, you want to setup some `int` values for the pins so it is a little easier to know what each pin is for. You have to use `digitalWrite` to set whether the pin is outputting a **GND** (`LOW`) or **VCC** (`HIGH`) value.

Finally

We're limited to only having 6 LEDs when powering them directly from the pins. There are a number of different ways around this limit, but the one we will be exploring later involves adding another chip to the setup, a shift register, and that will let us power 8 (or even more) LEDs from only three pins.

Buttons

Buttons

To make interactive things, you need inputs. The most basic of which is a simple push button. This lesson shows you how to add a button, read its value, and react to it. In addition, we're going to learn an important lesson about circuits.

The code

Firstly, we again start with the modified blink sketch. We will modify the first section:

```
int led = A0;
int button = A1;
```

We don't really need to modify the `setup()` section, as Arduino sets all pins to `INPUT` by default, but if it didn't we would do this:

```
pinMode(led, OUTPUT);
pinMode(button, INPUT);
```

Then we will modify `loop()` to react to our button press:

```
if( digitalRead(button) )
{
   digitalWrite(led, HIGH);
}
else
{
   digitalWrite(led, LOW);
}
```

`if( ... )` is a keyword that tells the processor that it should do the thing following if the value is not zero, else do the thing in the else. In this case, the `digitalRead` makes the processor set the led to on or off based on the value of the pin `button`.

Wiring

You now need to wire up your Magnolia so the button is connected to **A1**. To do this, you will need to put your switch on a prototype board, and connect jumper wires from the **VCC** and **A1** to the two legs of the switch.

Buttons with only two pins need to be put in so that the legs are not on the same row (rows are already connected inside the prototyping board). Buttons with four or five pins have connected pairs of legs (five pin buttons have a dead leg that is meant for stability). Find out which pairs of legs are already connected by using the switch in the board and power an LED.

First put the button onto the prototyping board, designate one side of the button **B0** and the other **B1**.

- Make a connection between **VCC** and **B0** (using a jumper from the Magnolia to the row in which **B0** resides.)
- Make a connection between **A1** and **B1** (again using a jumper from the Magnolia to the row in whch **B1** resides.)

Uploading and seeing something strange

When you upload the program, you will notice that the LED turns on before you touch the button. Sometimes it will flicker on and off a few times, but if you press the button, the LED will stay on constantly. This is because the switch connects the **VCC** to **A1** when the button is pressed, but there is

nothing connected to **A1** when the button is not pressed.

In this detached state, reading values from **A1** is very nearly random. Sometimes the voltage goes high just because moving your hand near the button increases the capacitance of the wire. Try playing with it for a bit before we fix the problem.

Fixing a floating input

Try moving the jumper from **B1**, to **GND**. Doing this makes the LED stay turned off. This is because the input **A1** is getting a strong grounding signal. The grounding signal doesn't have to be very strong, but it has to be there otherwise the wire acts like a kind of aerial or capacitor, picking up stray charge and triggering the **A1** seemingly at random.

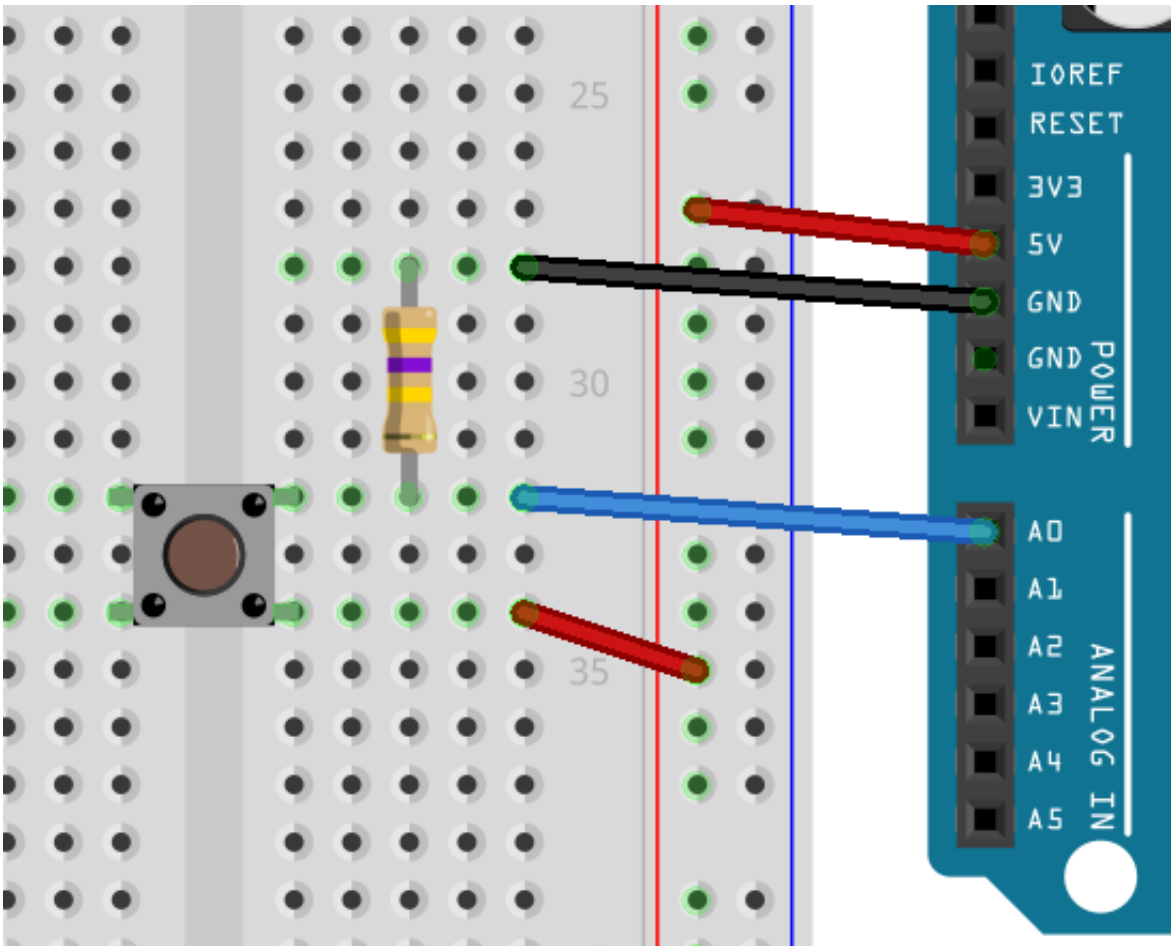
When you want a wire to register a high when a button is pressed, you need to connect **B0** to the **VCC**, and **B1** to **A1**, but to make a wire register a low, you also need to connect **A1** to the **GND** so that when the button isn't pressed, the **A1** pin is *dragged down* towards **GND**.

If we connected the output side of the button to **GND** the LED would stay off while the button wasn't pressed, but as soon as we pressed the button, it would short the whole thing out and we may even break stuff.

The solution is to make the **GND** connected to the output side of the button, but in such a way that it doesn't matter if we press the button. We don't have to be strongly grounded, so we can make that connection weaker. To do this, we add a rather high valued resistor. Usually, anything over 2000 Ohms will do, I prefer 470kOhm just to be sure I'm not wasting juice on the *pulldown resistor*.

Wire up the circuit **VCC** to button in, button out to **A1**, but also button out, via *pulldown resistor* to **GND**. Place your resistor on the prototyping board, and like with the button give the two legs names **R0** and **R1**.

- Make a connection between **VCC** and **B0**
- Make a connection between **A1** and **B1**
- Make a connection between **B1** and **R0**
- Make a connection between **R1** and **GND**



Now, when you run the program, the LED does not turn on and off randomly.

What we need to learn from this

Using switches can cause gaps in your circuits, so you need to make sure you have a circuit all the time, no dangling wires that go no-where. There are two ways to fix dangling wires on switches, the first is to use a `pulldown resistor`, like we did above. Doing this means that at no point is the input pin left unconnectd.

The second method is to use the other input mode of the chip, namely `INPUT_PULLUP`. In this mode things are a bit backward, you will want to connect the switch between the `INPUT_PULLUP` pin, and the **GND**. When the button is pressed, the button will drain causing a change in state for `digitalRead( ... )` on the input pin.

Fully understanding why it's doing what it's doing

Start by thinking of **A0** as being able to read the voltage difference between **GND** and whatever is plugged into it.

If you plug **A0** directly into **GND** then it will be measuring the voltage (potential difference) between **GND** and **GND**. This is zero.

Finally

Like with the limited resources stopping us from having more LEDs, you'll find that the Magnolia having only 6 IO ports makes having more than 6 buttons a very difficult thing. We can solve this problem too, but that will come in a later lesson about shift-in registers. These convert parallel data into serial data, meaning you can read from 8 inputs at once, using only three pins, just like you did with the shift register for extending the number of outputs.

Serial output

Serial output

When something goes wrong, you need to debug it. Because the Arduino sketches aren't running on your PC, it's generally impossible to use a debugger, but you can still trace the cause of unexpected behaviour in your Magnolia by printing text back to your computer by making it talk to you over the cable. This lesson introduces the Serial Monitor, and how to add output to your sketch.

Code setup

Before you can communicate, the Magnolia needs to know the way in which it will do the communication. On a Magnolia, just like an Arduino proper, the only thing we need to do is tell `Serial` to `begin`, and how fast it should communicate. Open the blinkA0 sketch again, and add to your `setup()`

```
Serial.begin(9600);
```

And any time you start the sketch, the Magnolia will know to talk back to your FTDI, and in turn your PC, at a *baud rate* of 9600. That is, 9600 bits per second. That's plenty fine for most simple debugging.

To test it out, add to your `loop()`

```
Serial.println("Test");
```

And then upload.

You should see that every two seconds (after the LED has done a cycle), the RX light flashes on the FTDI / Magnolia board. Now open up the Serial Monitor. It's `tools` > `serial monitor`, and as long as it is set to 9600 baud, you should start to see some output.

Forms of output

`Serial` has a lot of different functions to output data. DEC, HEX, println, print, write.

Serial input

Serial input

Sometimes you need to command the Magnolia to do something from the PC. If you do, then Serial input is a great simple way to communicate with the board without using up any of the pins.

Code setup

Just like in writing, before you can communicate, the Magnolia needs `Serial.begin()` to be called.

Instead of `Serial.write` we're going to call `Serial.read`, which returns either a character, or -1 if there is nothing to read.

To test it out, add this to your `loop()`

```
int inchar = Serial.read();
if( inchar != -1 )
{
   Serial.print( "Recvd [" );
   Serial.print( inchar );
   Serial.print( "] = \"" );
   Serial.write( inchar );
   Serial.println( "\"" );
}
```

And then upload.

Analogue Input

Analogue Input

Often there will be a need for reading a value that isn't just `LOW` or `HIGH`, but something in between. Temperature, light level, humidity, or pressure, all come in a more continuous form than just yes and no. The Magnolia makes a decent enough sensor setup because all its pins are set up for reading analogue values.

Analog reading

Before we get to the code, an important thing to remember. Unlike digitalRead and digitalWrite, the analogueRead doesn't use `LOW` and `HIGH`, but a range from values from `0` to `1023`, so make sure to keep that in mind.

The code

We're going to be doing a nice and reactive example that uses the photoresistor. Set up a pin for the resistor like this.

```
int led = A0;
int photo = A2;

void setup()
{
   pinMode(led, OUTPUT);
}
```

Then we need to read the value before we use it in the loop function.

```
void loop()
{
   int value = analogRead(photo);

   // turn the light on for a bit
   digitalWrite(led,HIGH);
   delay( 10 );

   // off time depends on value
   digitalWrite(led,LOW);
   delay( value );
}
```

Note, the smaller `value` is, the shorter the delay, and this the faster the LED will flash.

Wiring

We need to wire up the LED for output, but also the photoresistor for input. Like with the button, we're going to ground the input line, but this time for a slightly different reason. Voltage across resistors is based on their resistance, and we will use that to make the input pin receive a differing voltage over different brightness reaching the photoresistor. First we wire up, then explain.

Put the photoresistor in the board and temporarily give the legs names **PH0** and **PH1**. Add a 2kOhm resistor to the board. Name the resistor legs **R0** and **R1**.

- *The resistor value is quite important, make sure it's around the 2000 Ohm range for this wiring up. (your kit has a 2k7Ohm)*
- **VCC** to **PH0**

- **PH1** to **R0**
- **PH1** to **A1**
- **R1** to **GND**
- **GND** to LED cathode (neg)
- **A0** to LED anode (pos)

Running

Now try the example. The LED light should be flashing. If you cover the photoresistor, the amount of time the LED is on for per cycle will increase.

Why ground?

If we ground with different values of resistor, the photoresistor will become sensitive to different levels of light. High value resistances make it less sensitve, allowing for sunlight levels. Low value resistances make it more sensitive, meaning you can detect low light level differences. This is because the grounding resistance pairs with the photoresistor resistance to create a voltage divider.

In a circuit, the current is the same the whole circuit around, but the voltage on the other hand, is not. Voltage is a measurement of potential difference, and a really obvious example is how we can add multiple batteries in a line to increase the voltage.

Two 1.5v batteries in a battery holder turn into a 3v battery because the voltages add up, but it's not like they have become two 3v batteries by touching, they are still two seperate 1.5v batteries than can still be tapped for power independently as well.

As two 1.5v batteries add up to 3v, so do two sources of resistance. We calculate what the voltage is at each resistance by dividing the voltage into two different ones based on the resistance of the two sources of resistance. In our circuit, the **GND** is 0v, and the **VCC** is 5v. To calculate the voltage between the two resistors, we can work it out from first principles and ohms law. Or we can just use this formula:

```
middle volts = R2 / ( R1 + R2 ) * overall volts
```

Here, R2 is the 2k resistor, and R1 is the photoresistor. If R1 is very high (when it is dark, the photoresistor can measure in the megaohm range,) then R2 / (REALLY HIGH + R2) is going to be a very small value. If R1 is very low, then R2 / (almost zero + R2) is going to be almost 1.

This is how we make inputs work with potentiometers (sliders and knobs, and other variable resistors). We ground one side, power the other, then catch a voltage off the middle pin.

If you have a unpredictable analogue input (such as a photoresistor) you can add a variable resistor to the mix so you can tweak the lower and upper range, effectively giving you a hardware trimming. To do this, attach **VCC** to the live end of the variable resistor, and then attach the R2 to the middle pin of the variable resistor. This way you can adjust the value of R2 to fine tune the mid range.

Take away

Always ground things that are going to be used as inputs.

Filesystem

Filesystem

The ATMega328p doesn't have a hard drive attached, and even plugging in an SDCARD is a small project of its own, but the little microcontroller does have a very small amount of storage in the form of EEPROM, Electronically Erasable, Programmable Read-only memory. 1024 bytes to be precise.

We can use this to store something that is meant to survive across power outages. We will use it to store the number of times the Magnolia has been booted up. Useless, but a good starting point for any ideas you might have as it includes reading and writing.

The code

Reading and writing from EEPROM isn't a basic feature of Arduino, so you need to include a header that makes this code compilable. Add this at the top.

```
#include <EEPROM.h>
```

We've still got to display something, so use the normal LED output code.

```
int led = A0;

void setup()
{
   pinMode(led,OUTPUT);
```

but we're going to do the body of our code in setup this time, so after setting the pinmode, add this.

```
   int address = 0;
   int bootCount = EEPROM.read(address);
   EEPROM.write(address, bootCount+1);

   LEDFlash(bootCount);

   // wait for ten seconds, then reset the bootcount
   delay(10000);
   EEPROM.write(address, 1);
   LEDFlash(4);
}
```

LEDFlash isn't an Arduino function, it's one we're going to write. Above the function setup, add this.

```
void LEDFlash( int n )
{
   for( int i = 0; i < n; ++i )
   {
      digitalWrite(led, HIGH);
      delay(50);
      digitalWrite(led, LOW);
      delay(200);
   }
}
```

Now we have a function we can call. You can use this function in other sketches by copy pasting it in.

We're not going to use the loop on this sketch, so make sure it looks like this.

```
void loop()
```

```
{
}
```

Now, when you upload, the first thing it will do is flash the number of times the magnolia has been powered up. Leave it for ten seconds, then it should flash again saying it's reset the boot count. Each time you unplug the USB and plug it back in again, then number of times it flashes should go up by one. Try it a few times to make sure. It doesn't matter how long you leave it unplugged, as EEPROM doesn't change when there is no power to the device.

Take away

There is a place to store data that you need between power failures, but it's very limited, only 1024 bytes. That's not enough for a desktop icon (they are normally at least 4k).

Servos

Servos

If we need to make something be in a particular position based on some code, such as the rudder of a boat, the steering angle of a car, the openness of a window, or the direction of a solar panel, then we can use a servo.

Servos provide a way to request a position from 0 - 180 degrees. The request is done using only one IO pin, so you can control up to six servos with the Magnolia without adding any other chips.

The code

No biggie for this one, just open up the example sketch found in `File` > `Examples` > `Servo` > `Sweep`

Change the pin the servo is attached to by changing this:

```
myservo.attach(9);
```

to this

```
myservo.attach(A0);
```

Now you can upload and wire it up.

Wiring

Servos have leads with three wires. One is **GND**, one is **VCC**, and the last is the signal wire, **SIG**.

- connect servo **GND** to **GND**
- connect servo **VCC** to **VCC**
- connect **SIG** to **A0**

As soon as you connect **VCC**, the servo will probably squeak. When powered, the servo will try to go to wherever it thinks it should for a moment, and that could be anywhere while it is powering up.

As soon as you connect **SIG** to **A0**, the servo should rush to wherever it is meant to be, then continue to follower the 0 - 180 - 0 degree sweep that the Magnolia is looping through.

Where to go?

You could try combining the light sensor lesson with the servo, and have the servo react to light levels.

Shift Registers

Shift Registers

To get around the limitations of the Magnolia's 6 IO pins, this lesson will show how to use shift registers to turn three pins into eight. This can be extended to many more outputs with chainable shift registers.

The shift register we are going to use is called the **HC595**, and the one in your kit is specifically an **SN74HC595** (normal one with legs so you can stick it in the prototyping board.)

```
Qb  1   U   16 Vcc
Qc  2      15 Qa
Qd  3      14 SER
Qe  4      13 OE(ground to enable)
Qf  5      12 RCLK
Qg  6      11 SRCLK
Qh  7      10 SRCLR(power to disable)
GND 8       9 Qh'
```

The code

The **HC595** has a 16 pins, 8 are outputs, the rest are **VCC**, **GND**, or other IO pins to control the chip. We will be using all eight output pins that are meant to be the parallel output to power our LEDs. To drive those outputs we will be connecting to the data and clock line for the internal registers, and pushing the internal registers to the output pins with the register clock. We're going to be hard wiring the enable and clear pins and not using the chaining pin to start with.

When using chips with clock lines, we usually find they are triggered by positive-edge. This means the line transitions from a `LOW` to a `HIGH`. We'll start by adding this function to the top of our sketch so the main code can call it.

```
void trigger( int pin )
{
   // assume pin in low state
   // trigger the pos-edge
   digitalWrite(pin,HIGH);
   // reset to low
   digitalWrite(pin,LOW);
}
```

Inside the **HC595**, there are internal registers that can either be set high, or low. The internal registers ar called Qa' through Qh'. The external registers are called Qa through Qh. Notice the subtle name difference. When **RCLK** is triggered, the values of the external registers are overwritten by the internal ones. This means we can safely modify the values on the internal registers without worrying that anything is visible outside.

To modify the internal registers, we need to use **SER** and **SRCLK**. Every time the **SRCLK** is triggered, two things happen: The values in Qh' down to Qb' get overwritten by Qg' down to Qa', and then the value of **SER** is read, shunted into Qa'. This is why it is called a shift register. Every time you trigger **SRCLK** all the values are shifted along to the next Q' register.

To shift out all eight values, first we need a function to push one value out.

```
void put_one( int val )
{
   // set SER
   digitalWrite(SER, val);

   // trigger a shift
```

```
    trigger(SRCLK);
}
```

This sets the **SER** line, then triggers the **SRCLK**, meaning the values get shifted. At this point, the outputs Qa - Qh are still in whatever state they were in before, but Qa' - Qh' have been shifted, or updated.

Often we will use binary to feed into shift registers, so we will do that and write a function to take a byte (8 bits) and set all the shift register values in sequence, then trigger **RCLK** to update the Q registers (the external ones.)

```
void send_out( int bits )
{
    for( int i = 0; i < 8; ++i )
    {
        // send only the lowest bit
        put_one( val & 1 );

        // move the next highest bit down
        val >>= 1;
    }
    trigger(RCLK); // move Q' into Q
}
```

This function can now be used by our main loop and setup code.

```
int SER = A0;
int SRCLK = A1;
int RCLK = A2;

void setup()
{
    pinMode(SER, OUTPUT);
    pinMode(SRCLK, OUTPUT);
    pinMode(RCLK, OUTPUT);
}

int t;

void loop()
{
    send_out( 1 << (t&7) );
    delay(125);
}
```

This sketch, once we have wired up the lesson, should have a row of lights fire in sequence once per second.

Wiring

Because there are five wires from the Magnolia to the board, two of which are basic lines, w're going to start with some prototyping tips.

First, assume that we're going to take a lead from **GND** and **VCC** and drive the power rails on your prototyping board.

Any time the wiring requires a line from **VCC** or **GND** now, take it from these powered rails.

Place your shift register with the little dimple near the top, but a few rows down. Just so you have a little bit of space to work with. The top right pin (next to dimple) needs connecting to **VCC**, and the bottom left needs connecting to **GND**.

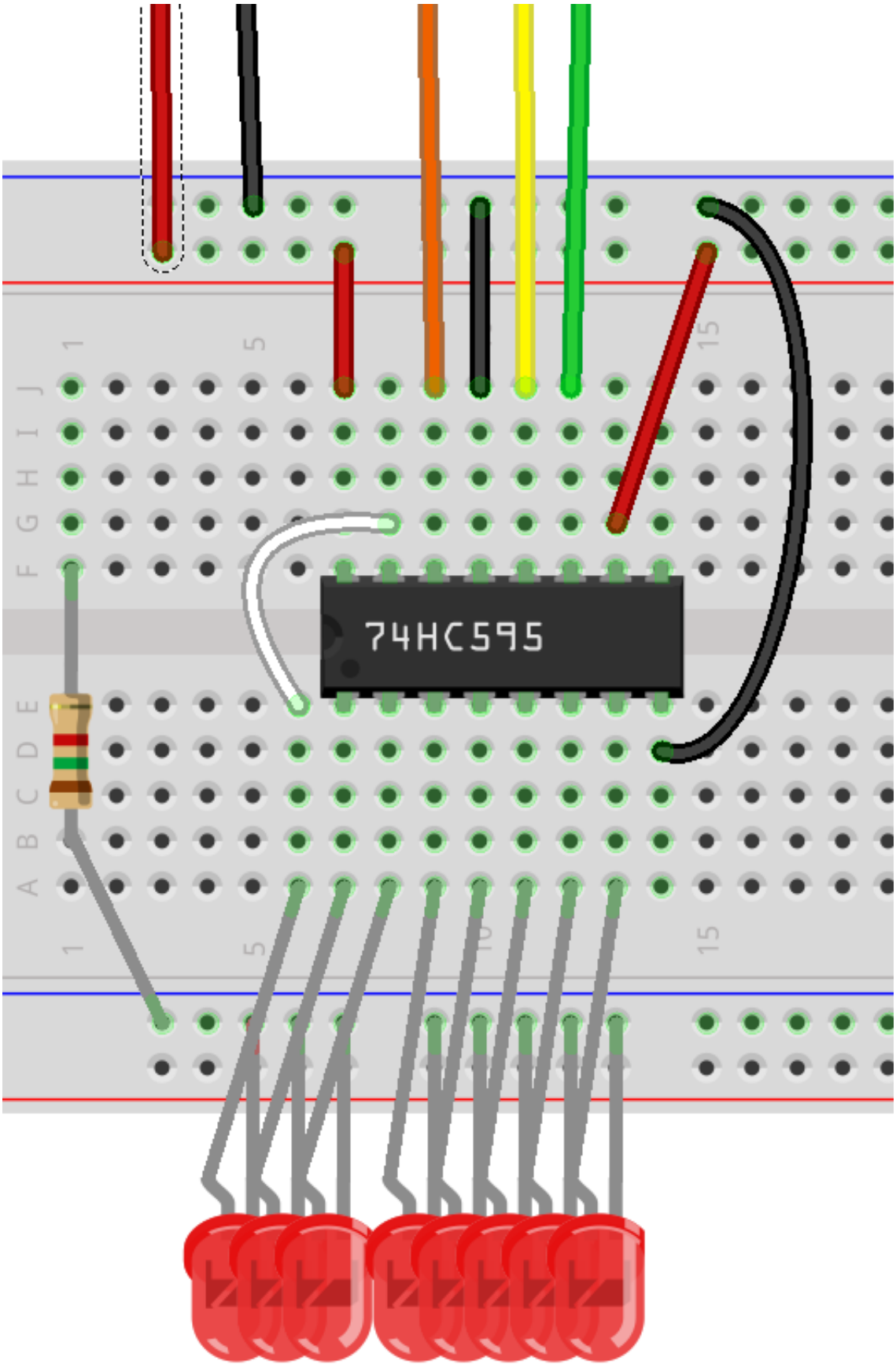
To hard wire the chip as enabled, wire **GND** to pin 13 (three down from **VCC**). To stop the chip from self clearing, power **SRCLR** with **VCC**. **SRCLR** is pin 10, one up from bottom right.

Take three long leads and connect **SER** (pin 14) to **A0**, **SRCLK** (pin 12) to **A1**, and **RCLK** (pin 11) to **A2**.

Take a jumper and jump between Qa (pin 15), and the row above Qb. This is so you can have all your LEDs plugged straight into the 8 rows starting above Qb.

Insert all your LEDs anode by the chip. All the cathodes should link to one of the two remaining rails. This rail will have a 2k resistor just to make sure the overall current drop doesn't drain the Magnolia. The 2k resistor should then be linked to **GND**.

Upload!



More outputs

Pin 9 is high when Qh' is high. This means that if there are two chips, then chip 1 Qh' can be attached to chip 2 **SER**, The **SRCLK** and **RCLK** can be wired common, then you have a 16bit shift-register. This sequence of adding more chips eventually runs out of juice, but not so quickly that it matters for most hobby projects.

Multiple Inputs

Multiple Inputs

Shift registers come as input types as well as output types. To get more inputs than six into your Magnolia, whilst reducing the number of pins used to do it, we use a shift-in register, or a parallel to serial convertor.

The basic function of the shift-in register is the same as the shift register, the master (Magnolia) triggers the serial clock to shift the registers inside the chip. In both these chips, one pin can push data into the chip. In the shift-in register, there are three output pins, but we're not going to use all three, think of them as being convenient optional additions.

In the shift-register, we had a latch pin that when triggered, took the internal registers Q' and put them out to the external registers Q. In the shift-in register, we have internal registers Q, and when we trigger the latch pin, the external registers P overwrite them.

In this way, we can request the input pins are put into the internal buffer (Q), and then read each pin at our leisure (triggering the **SERCLK** to request the next value.)

The code

Wiring

More inputs

Just as with shift registers, we can chain shift-in registers. As mentioned at the start of this lesson, both chips have serial input, and that means that you can tie **Q8** to **SER** on the next chip down. Make the **SERCLK** common, and the same with **PAR**, and you have all you need to run 16 inputs with the same number of IO pins from the Magnolia.