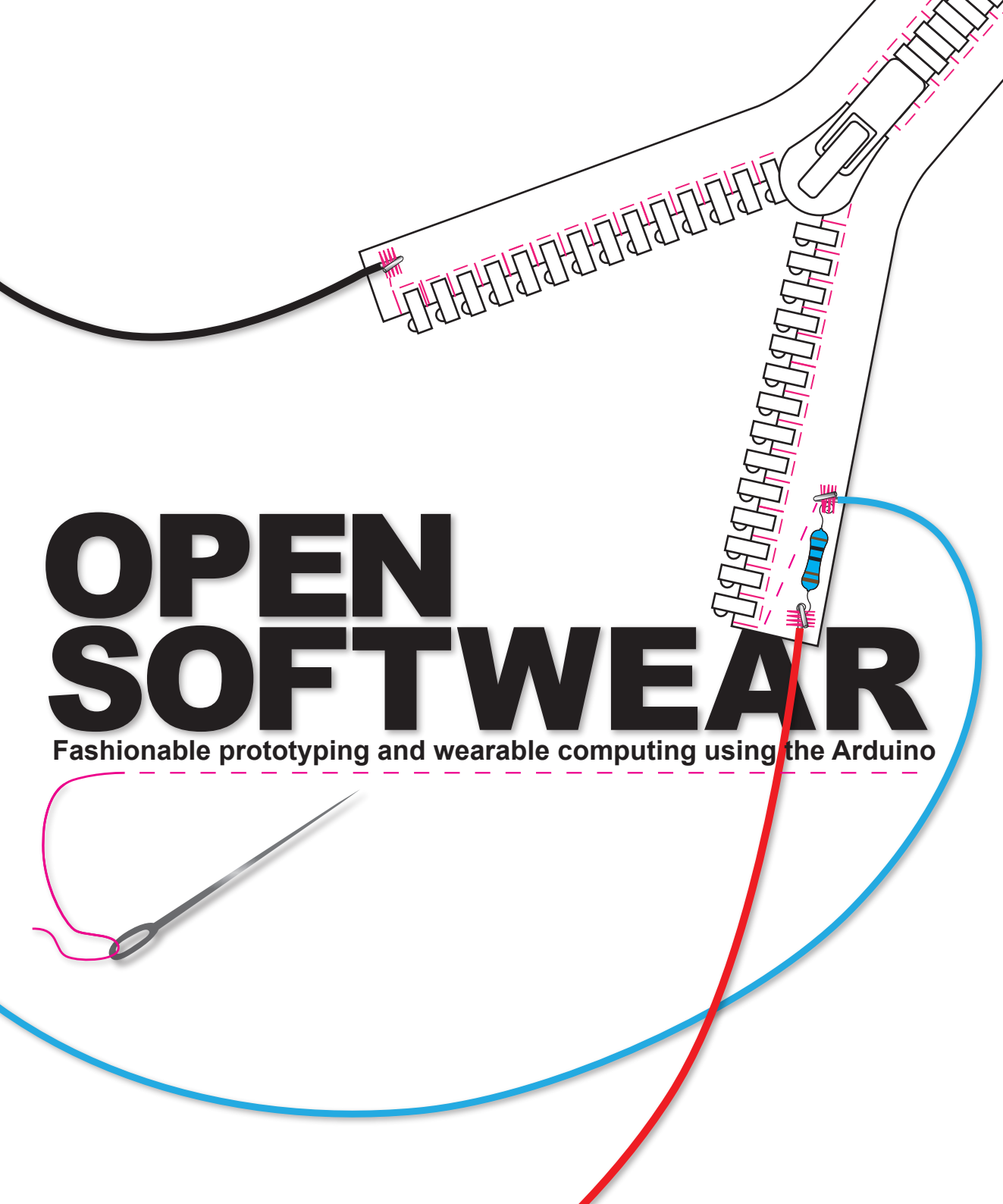


OPEN SOFTWEAR

Fashionable prototyping and wearable computing using the Arduino

Open softwear

fashionable prototyping and wearable computing using the Arduino

Preface

The content of this book is inspired by the teachings of the physical prototyping laboratory in the school of art and communication, at the University of Malmö.

The physical prototyping laboratory, run by David Cuartielles, has some of the longest running university courses based on the Arduino platform where Arduino has been a active part of the curriculum in Fashion, body and technology, Light Installation and in the Interaction programs at both bachelors and masters level, since 2005.

Until resent years students of the fashion body and technology course have been introduced to physical prototyping in a old fashion "hard" way. Earlier focus has been on transferring technology into the context of fashion and wearable computing.

However, in recent years, steps have been taken to implement technology into the context of fashion in a "softer" way. Prior to this, the experience from teachers at physical prototyping laboratory has been that students with an interest in fashion and wearable computing have had a hard time transferring the standard physical prototyping knowledge into prototype development of a wearable character.

When searching for suitable material to base our new approach to the field we soon realized that the available information was quiet limited and most of the material was of a "arts and craft" character. We a strong believer int the DIY movement and still think there is much to learn form the "arts and craft" materials out there but I think that a course at university level should be able to offer more complex approach than what can be learned for "do it your self guides".

Form my earlier experience of both attending and teaching the normal prototyping courses K3 Malmö my opinion is that the Arduino platform used, is that it is one of the best prototyping platforms currently available for two simple reasons. The price and the community. Both reasons are connected to the philosophy of the

Arduino and the Arduino is quite unique since it is both open software and hardware.

The goal with Arduino was to create a prototyping platform for designers to be able to realize there ideas by them selfs. And the openness in the Arduinos own design means that anyone is free to make modification to both the hardware and software or even produce and sell the actual boards. This in turn means that the price of the Arduino is available at a very affordable level since anyone is allowed to compete in the manufacturing.

The price in turn is also one of the main reasons to large number of users and its these users that is the Arduino community. This is a community that share a common love for prototyping and share the Arduinos philosophy of openness which not only means that there is a lot of information and help to be found. It also means that Arduino community also is one of the most current and rapid in pushing the evolution of prototyping forward. With this in mid the question has never been to move away from the Arduino when approaching the field of fashion and technology but rather how can we approach the field in a "softer way" using the Arduino.

Most of the development in the course has focused on comparing projects from active people in the field of fashion and wearable computing to the teachings at the University to find softer alternatives to the normal "hard" components. The goal has been to use the same basic principles used in "hard technology" and implementing them in a "soft" way to help the understanding of technology in terms that could be considered more naturally for people approaching physical prototyping from a textile background.

With the use of the Arduino I also hope to add what I consider one missing key feature that is missing in the "art and craft" moments approach to fashion and technology and that is the possibility of computing information.

This book isn't solely aimed at people with an academic interest in the field of fashion and technology and wearable computing but should also be considered as a start up guide for any one with a general interest in the subject.

It has been my aim to follow the teaching philosophy of David Cuartielles and the physical prototyping team at K3 Malmö. It is a simple philosophy that can best be compared to punk rock. Punk rock took the approach that you don't have to know everything about music to play music. If you know three basic accords that is enough to make a song. The same goes for prototyping. Once you know a couple of basic programming commands and how to connect something simple like a LED, then you can start building. Knowledge about prototyping comes from doing, not reading. But still you need some basic accords to play and I hope that the following chapters will get you started.

Tony

Contents

Part one: Basics

Introduction to wearable computing.

Chapter 1: Introduction

Prototyping with the Arduino

The Arduino prototyping platform is based on a simple work progress of using inputs and outputs. The inputs are usually some form of button, switch or sensor. Buttons are usually only on and off but with sensor you can measure your environment in a variety of ways. Sound, movement, temperature and light can all be processed by the Arduino and if you can think of any other behaviors that you want to measure, chances are that there already exists a sensor for it.

In the same way that you can connect a large amount of inputs, you can control a large scale of outputs. An output can be anything from light, movement, heat to more complex outputs like sending an SMS or even turning a TV off on the other side of the world. In most cases there already exists a technical solution for your prototype, you just have to find them.

Since the Arduino at heart is a micro controller, everything you connect to it has to be electrical. The good thing is that nearly everything can be translated into a electrical signal. For example, when a human is touched by something a signal is sent to the brain that there is a sensation somewhere on the body. In the same way we can make an electric line from the Arduino and back to it. If something breaks this line of electricity a signal is sent back to the brain of the Arduino to tell it that there is no electricity in the line.

Hacking: save money, learn more

Once you have started to play around with making your own electric prototypes you will soon realize that a lot of electronic components costs a fare amount of money. It's therefore most people interested in electric prototyping have their other foot in hardware hacking.

Hardware hacking is also known as tinkering and it describes the activity of breaking commercial electrical products apart just to see "what makes them tick".

Tinkering isn't just a good way to learn more about how electric

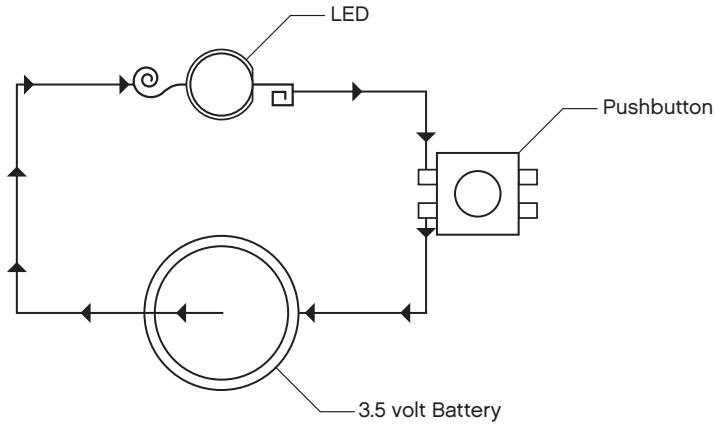
things work but it's also an efficient way to save money. A lot of components you may need for your projects can be found in what's normally considered junk. An old printer has a motor that might still be functional, an old phone has nice batteries and small vibrators and cheap electric toys are often goldmines.

There's no right or wrong way of hacking and tinkering which also means theres no official instructions on how to do it. But the internet is full of information and tips on tinkering and the Arduino www.arduino.cc/playground, www.makezine.com and www.instructables.com are great resources to get you started.

How electricity works

There are a few things you need to know about electricity to make your own electronic prototypes. First of all electricity always need to go back to where it came from to make a circuit.

In the following example we have connected a LED to a battery with a switch:



The power from the battery will travel through the cable into the LED in one leg and out the other down to the button and back to the battery.

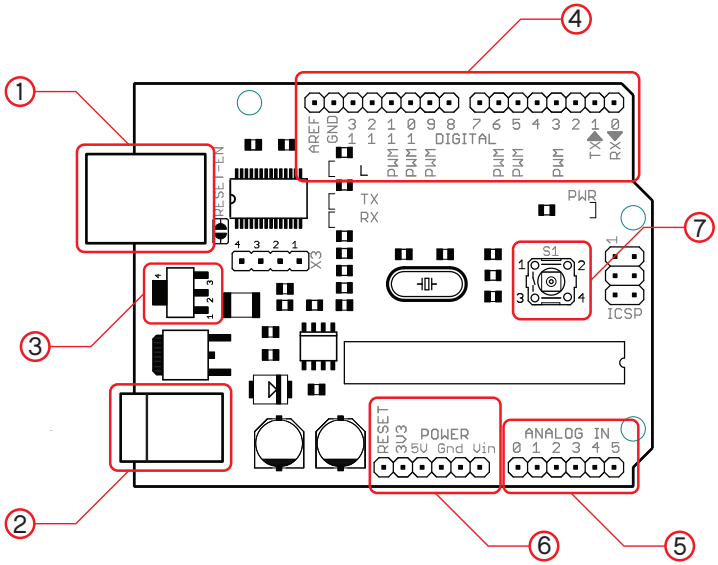
Buttons normally connect a small piece of metal with another one when you push it. If the button isn't pushed nothing will happen but when you push it you make a connection with the metal plates and the power from the battery can travel back to where it came from. Once it comes back to the battery the LED lights up. In the above example there is a 3.3V battery and a LED that can handle 3.3V of electricity. If we connected a 9V battery instead the LED would burn. This is because electricity have different types of volt and ampere and it travels with different resistance.

Imagine electricity like water. The speed of the water would be the same as the volt and the amount of water traveling would be the same as the amperes. Lets say we let our water through a garden hose then the garden hose would be the same as the resistance for electricity. So to connect the LED to 9V battery would be like pushing to much water to fast through a garden hose. If the garden hose cant let all the water through the hose will burst.
It's very rare that that things explode when making prototypes with the Arduino since most prototypes are made with such low voltage it's hard to even consider them harmful. But still it's never a good idea to connect more power to something than it can handle since there's a good chance you'll brake it. So always follow the recommended power restrictions for you components.

Chapter 2: Hardware

Arduino

The Arduino is an open source micro controller board used for electronic prototyping. The Arduino can receive data from sensors used to collect information in its soundings and it can be used to control other electronic components as lights, motors and more. There are varieties of Arduinos available and the most common one is the latest version of the standard Arduino board which looks like this:



The most important parts on the Arduino board high lighted in red:

1: USB connector
2: Power connector
3: Automatic power switch
4: Digital pins
5: Analog pins
6: Power pins
7: Reset switch

The USB connector (1) is used for connecting your Arduino board to your computer. While connected the Arduino board will be powered from the USB cable and while connected you can upload code and you can communicate from and to your Arduino board.

The power connector (2) is used when you don't want to power your Arduino with the USB cable. Instead you can use a normal transformer (power adapter) in the range from 6V to 24V.

Note:
Although the Arduino has an on board power regulator, make shure to never connect a power source that is larger than 24V. Chances are that you will destroy your Arduino board.

The Arduino can also run on batteries. On Arduinos earlier than the Duemilanove version you need to manually switch power source on the Arduino. This is done by switching the plastic jumper (3) located between the USB connector and the power connector. If you want to power the Arduino with the USB you put the jumper over the two pins closest to the USB connector and if you want an external power source you put the jumper over the two pins closest to the power connector. On versions later than the Duemilanove the Arduino automatically choses the power source.

There are 13 digital pins (4) on the Arduino board and these can be used as both inputs and outputs depending on how you set theme in your program:

The analog pins (5) work only as input (this is not completely true since you can reprogram these pins to digital ones but this requires some Arduino knowledge) but can handle a larger range of incoming information then what the digital pins can:

Note:
GND is short for ground.

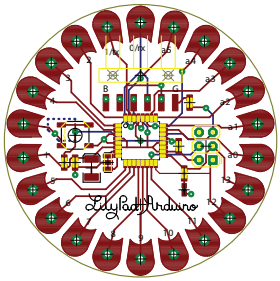
To the left of the analog pins you will find the power pins (6). From here you can pull either 3.3V or 5V. The pin named vin will give you whatever is connected to the power jack. If you have 12V connected to the power jack, you will be able to pull the same from this pin. Here you also can find two GND pins.

The reset switch (7) is used to reset any program on the Arduino to start from its beginning. On Arduinos older than the Diecimila version of the Arduino, the reset button needs to be pushed every time you try to upload code.

For the examples in this book we have chosen to use the standard Arduino board since it is our opinion that this is the best board for prototyping. When you are close to finalizing you prototypes it can be useful to migrate to one of the other smaller boards to save space. These board work the same way and any program written for a standard Arduino board will work on all other types of Arduinos. The following are two other examples of Arduino boards available.

LilyPad

The LilyPad Arduino is a micro controller board designed for wearables and e-textiles. It can be sewn to fabric and similarly mounted power supplies, sensors and actuators with conductive thread. The board is based on the ATmega168V chip (the low-power version of the ATmega168) The LilyPad Arduino was designed and developed by Leah Buechley and SparkFun Electronics.

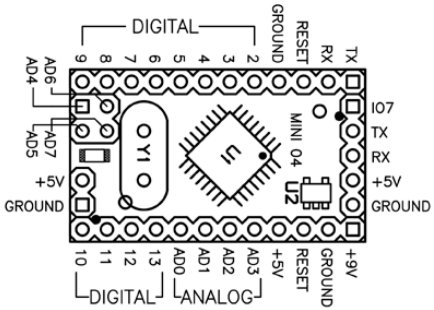


Note:
ATmega 168 is an electronic integrated circuit microcontroller produced by the Atmel corporation. It has the basic Atmel AVR instruction set. For more information about Atmel and the ATmega chip family visit the Atmel site:
www.atmel.com

Note:
To learn more about the hardware of the standard Arduino board or for information on other Arduino boards visit the Arduino site:
www.arduino.cc. ard.

Arduino mini

The Arduino Mini is a small micro controller board based on the ATmega168, intended for use on breadboards and when space is at a premium. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 8 analog inputs, and a 16 MHz crystal oscillator. It can be programmed with the mini USB adapter or other USB or RS232 to TTL serial adapter (www.arduino.cc). If you remove the male pins from this board it can be sewn to a piece of fabric using conductive thread.

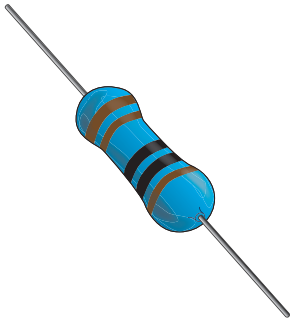


Basic electronic components for soft prototyping

• Conductive thread

This type of thread looks like normal grey thread but it has the possibility to carry currents for power and signals. The conductive thread can be used instead of normal cables to power your electronics and it is very suitable for working with "soft" prototyping. Conductive threads come in different thicknesses and with different resistances. The resistance in the thread lowers the pressure of the voltage in the power supply you are using. The resistance of the thread is normally calculated per meter so remember the simple principle that more thread equals more resistance.

• Resistors



A resistor is an electronic component designed to oppose an electric current by producing a voltage drop between its terminals in proportion to the current. Resistance is always measured in ohm and can also be presented as the symbol Ω. The most common multipliers for resistance calculations is:

kilo ohm which is the same as a 1'000 Ω

megohm which is the same as 1'000'000 Ω

All resistors are color coded. Resistors can have 4, 5 or 6 colder bands and it is these bands that tell you the resistance of the resistor. The first three bands are digits that follow this color digit scheme:

black	0
brown	1
red	2
orange	3
yellow	4
green	5
blue	6
purple	7
gray	8
white	9

If the three bands are brown, red and blue this would translated to 126. The forth band is the multiplier. You multiple the first three digit by this band and then you get the resistance of your resistor. The fourth band follows the below color scheme:

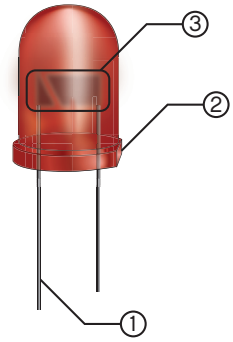
silver	0.01
gold	0.1
black	1
brown	10
red	100
orange	1k
yellow	10k
green	100k
blue	1M
purple	10M

The fifth band is the tolerance of the resistor and the sixth band is used for temperature coefficient. It's hard to learn resistor calculations by heart so it recommended to use a online resistor calculator to be certain. If you google "resistor calculator" you will find a lot of them.

Note:
The third digit is not used on four band resistors. Four band resistor only use two digits, the third band is the multiplier and the fourth is the tolerance.

• LED

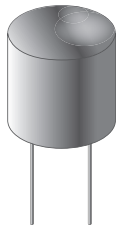
A light emitting diode (LED) is a semi conducting diode that lights up when electric current is applied in the forward direction of the LED. There are three different ways of telling the forward direction of a LED. The first is to look at the legs of the LED. There will be one that is longer. This long leg will be the one connected to where the power comes from and the short leg needs to be connected to the ground of the power source(1). The second way to tell what way of a LED is to look at the shape of the plastic bubble(2). The lower rim of the bubble should have one side that is flat and it is the leg of that side that should go to you power and the other side should go to ground. The third way is to hold a LED up to a light and have a look inside. There will be two pegs, one small and one big(3). The leg that goes to the small peg is the one that should be connected to the ground and the leg on the big peg should be connected to power.



The most common LEDs are powered in the vicinity of 3.3V and 20mA. Note that the Arduino will always supply 5V from the digital pins. This is why we will use a resistor of 220 Ω to lower the power so we won't burn the LED. To be sure what power your LED needs, have a look at the data sheet for your LED. This information is often available with the electric components you buy. If you need to calculate what resistance to use for your LED I recommend you use an online resistor calculator.

• Conductive fabric

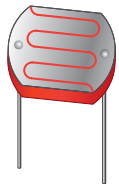
Conductive fabrics are normally a combination of highly conductive metals and lightweight fabrics and is often used as a shielding material. Conductive fabrics have the ability to conduct electricity.



• Tilt sensor

A tilt sensor is a sensor that can detect if a object is tilting to one side or another. The cheapest kind of tilt sensor can also be used as a measurement for movement. The drawback of this kind of tilt sensor is that they work as a pushbutton so they cant be used to tell which direction an object is tilting or how much, only that the object is tilting.

Inside the tilt sensor there is a small metal ball inside a metal casing. Once the ball touches the sides of the metal casing it will complete a circuit and we can read it from the Arduino board.

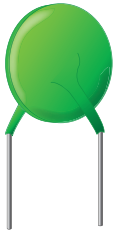


• LDR sensor

LDR stands for Light Dependent Resistor and is also known as a photoresistor. An LDR is made of a high resistance semiconductor. The LDR is similar to a normal resistor with the exception that normal resistor have fixed values and the LDR:s resistance is dependent on light in its vicinity. It is very hard to use an LDR to determine an exact amount of light in a given setting but they are good enough to use for determining light in a broader sense - if its dark or light.

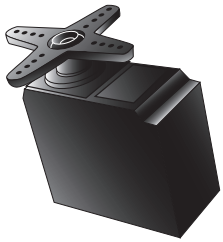
• NTC sensor

NTC stands for Negative Temperature Coefficient and is also know as a thermistor. A thermistor is a type of resistor that changes its resistance according to temperature. It's hard to use a thermistor to tell exact temperature but they're still good enough to make estimates if something is cold or hot.



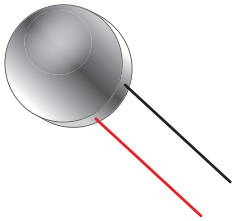
• Motors

If you want to move something you probably need a motor of some sort. A motor is more or less an actuator that turns electricity into movement. While working with hidden wearable prototypes motors can become and issue if you need a lot of force. Most motors follow the simple principle "the bigger the force, the bigger the motor". There are still lots of small motor that could be relevant and with some creativity motors can be nicely integrated in your prototypes.



There are three main types of motors: DC motors, Servo motors and stepper motors.

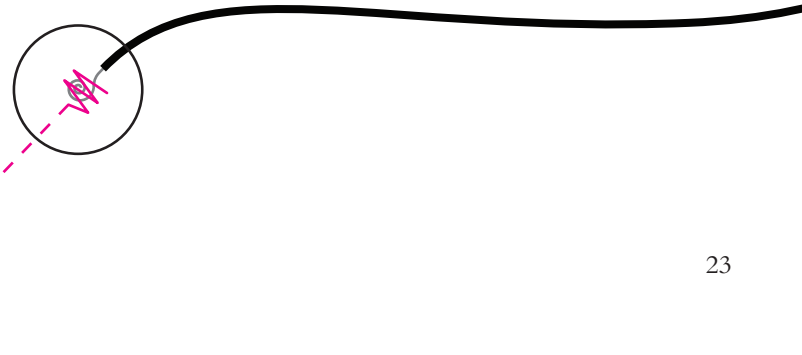
We have not included any examples of stepper motors in this book. Though stepper motors are good for moving in full rotation and in steps, they're not appropriate to implement into wearable technology since they are both heavy for there size and quite bulky in their shape. If you are considering using motors it's recommended that you find either a DC motor or a Servo motor that fits your prototype.



• Wires

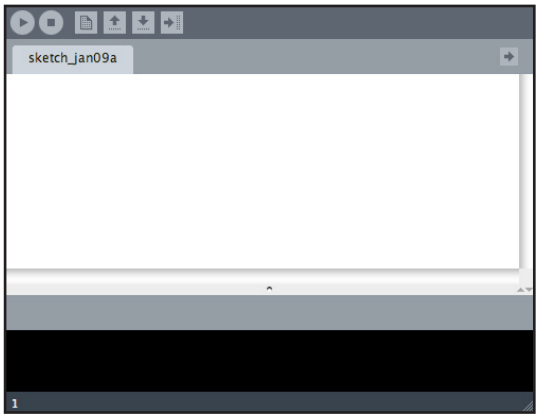
Wires are thin conductive metal threads placed inside a plastic casing, they come in a wide range of sizes and colors. It's good prototyping behaviour to color code your wires so you consistenly use the same colors.. The common color use is GND = black, PWR = red and then the pins can have any other color.

Note:
When we're using wires it's good prototyping behaviour to curl the exposed metal core to make it easier to sew in place.

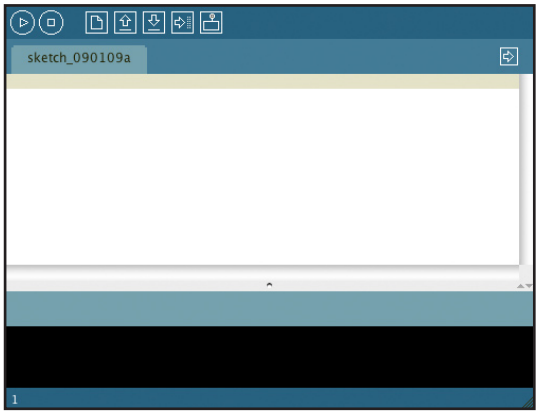


Chapter 3: Software

The software used to write programs for your Arduino is called the Arduino IDE (Integrated Development Environment). The Arduino IDE is based on another open source programing language and program called Processing used for programing images, animations and computer interactions. The Arduino IDE looks very similar to the Processing IDE:



The above is the Processing IDE and below you can see the Arduino IDE:



Even the Arduino coding language is molded after the Processing language. The Arduino language is based on easy to use commands and every time you press the upload button in the Arduino IDE it will translate you program in to C code so the Arduino board can understand your program. The language is built this way since C programming is quite hard to use for first time programmers.

Installing the software

The software can be found on the www.arduino.cc site under "downloads". Once you have found the download page choose the right version for your operating system. When you have downloaded the software, uncompress it and put the Arduino folder on your desktop.

• For XP users

After you have done all the steps in previous section, take your Arduino board and connect to your computer with the USB cable. Now Windows will tell you that it found a new USB device and it needs the drivers. Redirects the program to the drivers folder inside your Arduino folder and install the drivers:

desktop/Arduino/drivers/

After you have done this one time Windows will ask you the yet another time for new drivers. Redirect the program once again to the same folder and install one more time.

• For Vista users

Go to www.ftdichip.com and in the "drivers" section locate the D2XX driver and download it, double click on it and install the drivers. Then connect your Arduino to the computer with a USB cable. Vista will tell you that it found a new device and that it needs the driver for it. Redirect it to the drivers folder inside the Arduino folder:

desktop/Arduino/drivers/

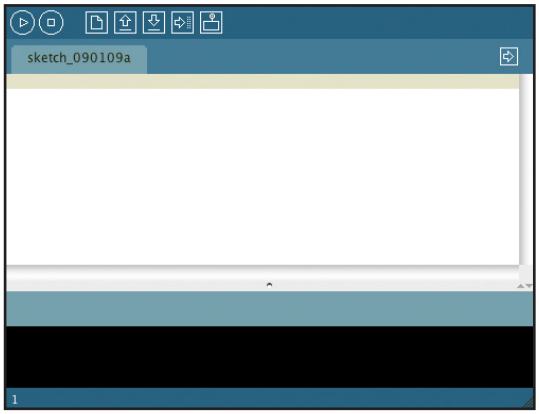
After you have done this one time Windows will ask you the yet another time for new drivers. Redirect the program once again to the same driver and install one more time.

• For OSX users

Open the Arduino folder and then the drivers folder. Choose the right driver for your processor and double click on it. This will install the drivers needed and it will force you to restart your computer.

Chapter 4: Using the IDE

The IDE consists of two large spaces, one white and one black. The white space is where you will write your program. Note that anything you write in here will be considered as code if you don't "comment it". To learn more on how to hide text in your code turn to page ?.



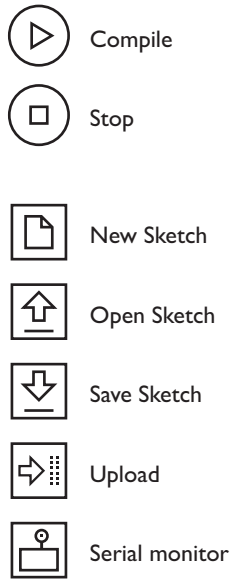
The black space is where you will receive error and confirmation messages. Above the white space you will find the IDE buttons. With these buttons you control most of the actions in the IDE.

The first one is the compile button. This button makes a check of your program to see if there's any logical errors in your code.

The second button is the stop button. This button is used to turn off the serial monitor. The compiler will take as much time as it need but don't worry, compiling usually only takes a few seconds depending one how big your program is.

The third button is the new sketch button. In the Arduino IDE all programs you open or write are called sketches. The new sketch button will open a new sketch for you, but will first ask if you want to save your present sketch.

The IDE buttons:



Note:
The IDE only makes a logical check and can't determine if the program corresponds to what you want the program to do.

Once you started to compile you can't stop the compilation with the stop button.

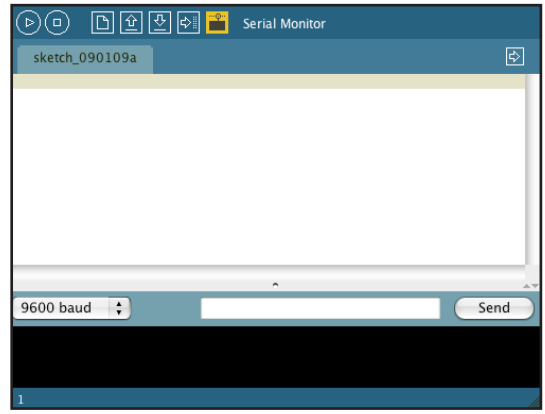
The fourth button is the open sketch button. This button opens the sketches folder and here you can choose to open already saved sketches.

The fifth button is the save button. This button saves the present sketches in the folder named sketchebook.

The sixth button is the upload button. This button will upload the present program to your Arduino board assuming there is no errors in your code. The upload button will first try to compile your code and if it finds any errors it will stop compiling and a error message will appear in the black window of the IDE, telling you what the problem is and the IDE will highlight the line of code that is causing the problem.

The last button is the serial monitor button. This button will open your serial monitor in the black space at the bottom of the IDE. Some times it can get confusing to tell if the serial monitor is open or not. The thing that tells the serial monitor apart from the normal black space is that when the serial monitor is open a bar appears with a drop down menu, a send button and a message box. To turn the monitor off use the stop button.

At the top of the IDE you will find drop down menus as in any other program.



In the file menu you will find all the functions of the buttons, your sketchbook folder and preferences. In the edit menu you can find the functions and commands for undo, redo, cut, copy, paste, select

all, find and find next. In the sketch menu you can verify/compile your code, stop, import libraries, show the sketchbook folder and add new file.

The two most important parts in the tools menu are the board and serial port. The board option is where you select your type of Arduino board. In the serial port you select which USB port you have connected your Arduino board. The easiest way to determine which port your board is connected to, since more the one port can sow up in the menu, is to unplug you board and look which ports are connected. Then you plug your board back again and the new port that appearer in the list is your Arduino board.

Uploading code

To test if your installation of the software went well open the blink example code found in:

file/ sketchbook/ examples/ digital/ blink.

Once you have the code present make sure you have the right board type and serial port selected in the tools menu. Push the upload button and if everything goes without problems, the LED on the board next to pin 13 should start to blink on and off with a one second delay.

Part two: Examples

How to realise your concepts into prototypes.

Examples

The following part of the book is the examples section. Here we will present a collection of examples on how to create and use both input and output devices and how to interface them with an Arduino board.

These examples are not in any way finished prototypes but should be considered as inspirational construction solutions and programing techniques that can be useful for fashionable and wearable prototypes.

This section will start of with more simple examples, gradually turning to more complex construction and techniques. All examples start with a list of all components needed for them and all examples are based on components and materials that should be supplied by most electronic and electrical hobbyists stores. Some materials can be tricky too find like conductive fabrics and thread. Use the internet to find suppliers near you or compare online stores to find materials at the best prices.

Chapter 5: Using digital Pins

As explained in chapter 2 the digital pins on the Arduino have only two modes, either they are On or Off. The actual command for these two states of the Arduino are 1 for on or 0 for off and that is why we call theme digital pins. Normally we use the constants HIGH and LOW since this makes it easier to read code compared to using 0 and 1. Remember that the digital pin always gives the output power of 5V when high and 0V when LOW. Don't connect anything straight to the digital pin if you don't know that it can handle 5V.

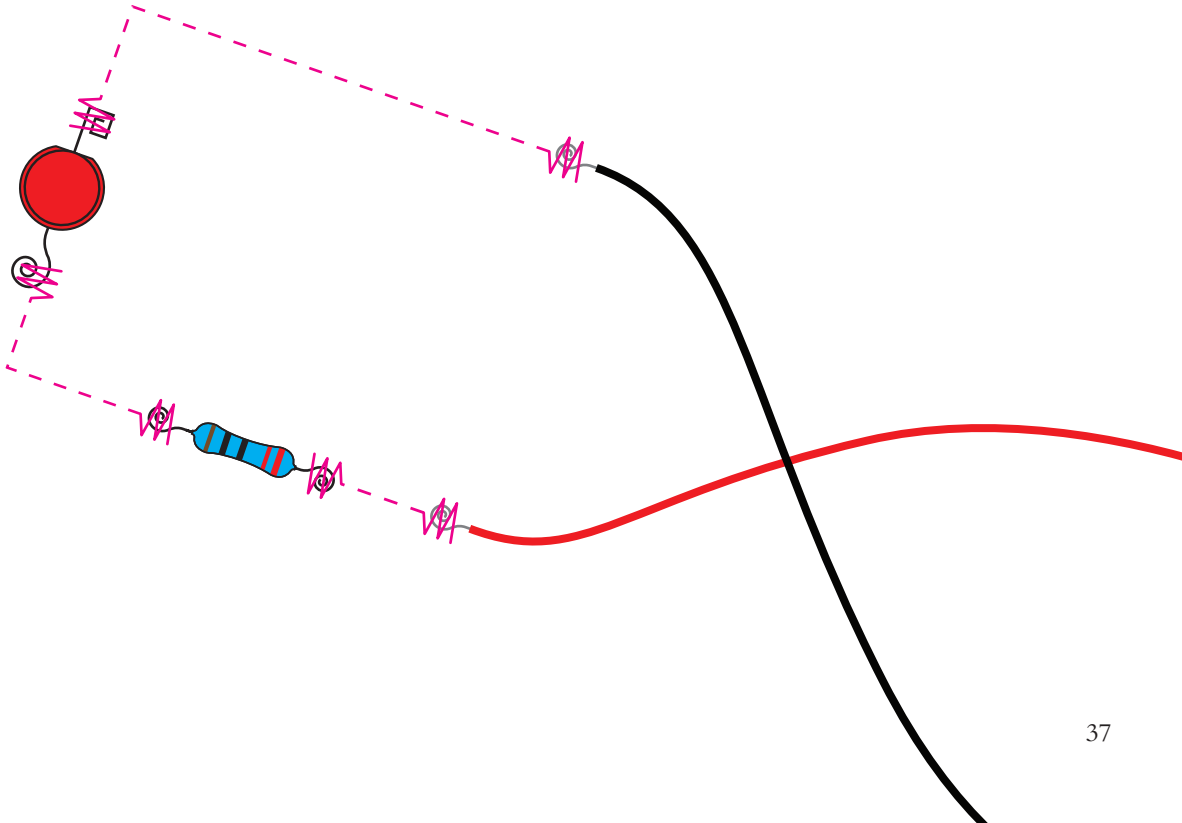
Part 1: Soft prototyping with LEDs

To make a soft prototype with LEDs we need:

- Some conductive thread
- One LED
- One 220 Ω resistor
- Non-conductive fabric

• Sewing a LED

This drawing illustrates how to sew your circuit on a piece of fabric. You can use any piece of fabric as long as it is not conductive:



In this example we have made circles on the legs of the resistor and both circles and squares of the legs on the LED. This isn't only for cosmetic reasons, it's also practical - it makes it easier to sew the components in place and the different shapes is also good for keeping track of the long and short leg of the LED. For the voltage and ground connection we have used cables, sewn in place with conductive thread. This is a good technique to use while testing your prototype and in final prototypes is recommended that you sew your Arduino board on to the fabric and make straight connections with conductive thread to your components.

• Coding a Blinking LED

Before you connect your Arduino we can test that this code works and that there is no damage to you Arduino board. The Arduino has a small built in LED next to your digital pin 13 that it's connected to pin 13 so with the following code example it should start to blink:

```
int ledPin = 13;
/* an integer variable for the LED connected on digi-
tal pin 13 */

void setup(){
   pinMode(ledPin,OUTPUT);
   /* sets the ledPin as output */
}

void loop(){
   digitalWrite(ledPin,HIGH);
   /* turns on the ledPin */

   delay(1000);
   /* wait for one second */

   digitalWrite(ledPin,LOW);
   /* turns the ledPin off */

   delay(1000);
   /* wait for one second */
}
```

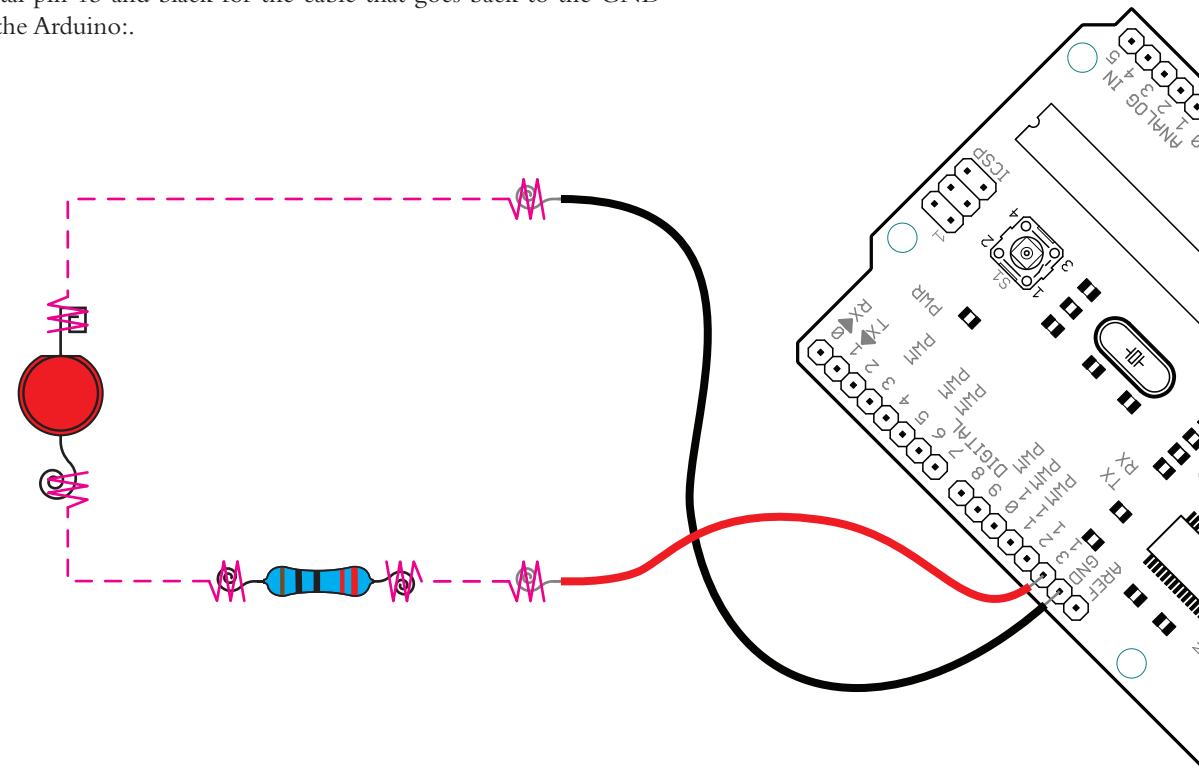
This program will turn the LED on for one second then off for one second before starting over again.

Once you have written the program in your Arduino IDE press the verify button and the message: "Done compiling" should appear in the black window. If there is no compiling errors make sure you have the right com-port and board type selected in the tools menu.

The next step is to upload the code to your Arduino board by pressing the upload button. There are two LED:s on the Arduino named RX and TX. Ever time you try to upload a program to the Arduino these two LED:s should start to flicker on and off. If they don't, have a look in your tools menu and check that the right com-port and board type is selected. If there where no errors in uploading the program is now stored in the Arduinos memory and will stay there until you replace it with a new one and after 5 seconds the program will start.

If the LED next to pin 13 on the board starts to blink it's safe to assume that the program was correctly uploaded and that there's nothing wrong with your Arduino board. This LED blink program is good to use if you experience problems while making prototypes and want exclude hardware failure from your debugging list.

Now it's safe to connect the Arduino to your soft circuit. In the following drawing we have used red for the cable that goes to the digital pin 13 and black for the cable that goes back to the GND on the Arduino:.



This is commonly how you color code your cables while working with electronics. Red is used to mark a cable that connects to the power out and black are used to mark cables as ground cables. All of the following examples used in this book will follow the same color scheme. Note that red cables does not mean that they are always connected to the digital pins. In this example we are switching 5V on and of on the digital pins and that is why we used red to mark the cable.

• Coding a fading a LED

In the previous example we turned an LED on and off. As you may have noticed the LED was set to maximum on (5V reduced to with a resistor) to a complete off (0V). As I mentioned before the digital pins have only two modes, HIGH and LOW. But the pigital pins 3, 5, 6, 9, 10 and 11 has a special function called PWM. With the PWM mode we can transformer to 5V output into a range of 255 possible levels. To learn more about the PWM function turn to page ?. In the following example we are going to test how we fade a led up and down:

Note:
To make the code more viewable we use tabs to clarify the code trees.

```
int ledPin = 5;
/* connect your led to digital pin 5 */

void setup(){
   pinMode( ledPin , OUTPUT );
   /* declare ledPin as an OUTPUT */
}

void loop(){
   for(int i = 0; i < 255; i++){
   /* as long as i is smaller than 255, increases i by
   one */

     analogWrite( ledPin , i );
     /* fade the led to i */

     delay(30);
     /* small pause so we can see each step */
   }

   for(int i = 0; i > 0; i--){
   /* as long as i is bigger than 0, decreases i by
   one */

     analogWrite(ledPin,i);
     /* fade the led to i */

     delay(30);
     /* small pause so we can see each step*/
   }
}
```

In the above example we are using the command analogWrite() which is the command for PWM on the LED. We are also using two for loops, the first one starts counting from 0 and up to 255. For every loop it makes it increases the voltage in the LED. Instead of going from 0V to 5V in one instance it goes from 0V to 5V in 255 steps which makes the LED fade. The second for loop does the opposite of the first one, it goes from 5V back to 0V in 255 steps. The delay of 30 milliseconds gives us some time to notice the steps. Now you can connect the LED as we did in previous example, upload the code to your Arduino board and enjoy the fading. You should be able to see the LED fade up and down again.

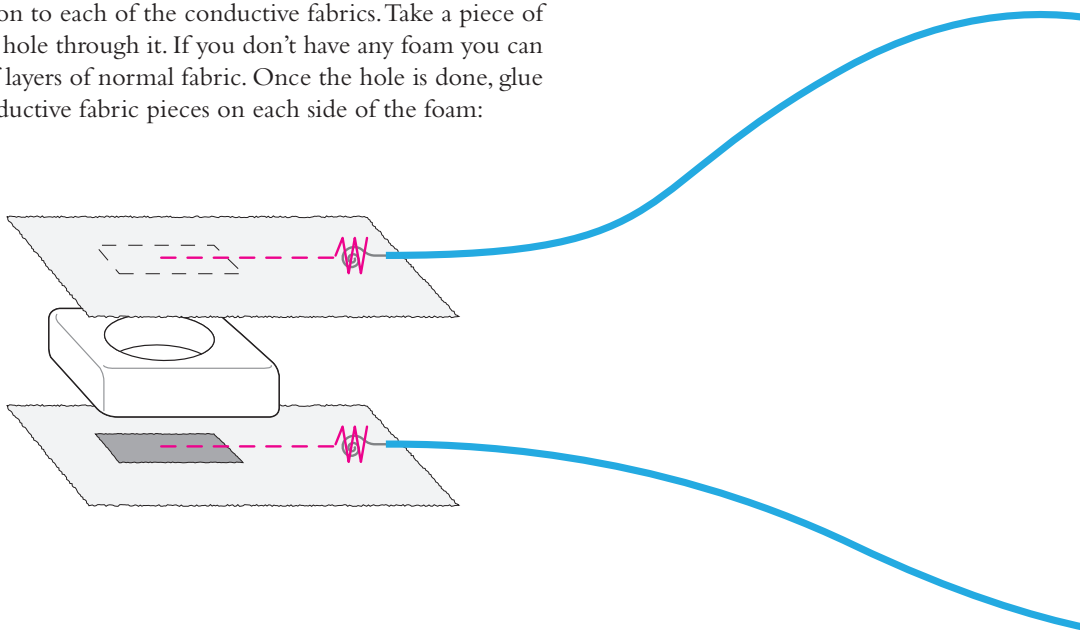
Part 2: Soft push button

There are different ways of making your own soft push buttons. All of them are based on the same principle of creating a circuit from and back to Arduino with a breaking point somewhere in the circuit. A braking point is where we can reconnect the circuit so we can determine if the soft pushbutton is active (pushed) or not.

To make a soft push button we need:

- Conductive fabric
- Non-conductive fabric
- One piece of foam
- Conductive thread
- Cables
- Fabric glue

To make this soft push button start by cutting out two small pieces of conductive fabric and glue them on two separate pieces of non-conductive fabric. Then sew a cable at the end of both pieces and sew a connection to each of the conductive fabrics. Take a piece of foam and cut a hole through it. If you don't have any foam you can use a couple of layers of normal fabric. Once the hole is done, glue one of the conductive fabric pieces on each side of the foam:



When you have the push button ready we can get started with the program:

Note:
In the setup we turn 5V on to the button. Every time we push the button we redirect the power to GND and if we read the status of the push button pin it will tell us LOW since there is no power in the pin at that moment.

```
int myButton = 4;
/* declare digital 4 on the Arduino as myButton */

int ledPin = 13;
/* declare digital 13 on the Arduino as ledPin */

void setup(){
   pinMode(ledPin,OUTPUT);
   /* set ledPin as an OUTPUT */

   pinMode(myButton,INPUT);
   /* set myButton as an INPUT */

   digitalWrite(myButton,HIGH);
   /* activate internal resistor on pin 4 */
}

void loop(){
   if(digitalRead(myButton) == LOW ){
   /*check if the button is pushed */

     digitalWrite(ledPin,HIGH);
     /* if the button is pushed, light led  */

   }else{
     digitalWrite(ledPin,LOW);
     /* if the button isn't pushed, turn off led */

   }
}
```

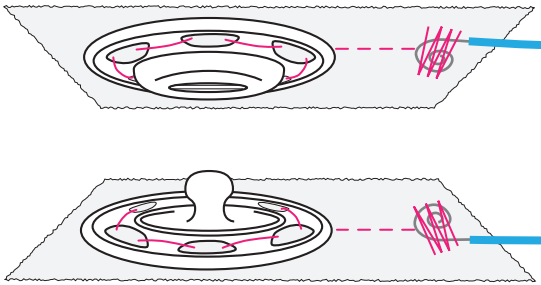
Now we can upload the program to our Arduino board and connect the push button. One end connects to our digital pin 4 and the other one is connected to GND. The above program will check if the push button is pushed and light up the LED next to digital pin 13. If the button isn't pushed the LED will remain off.

Part 3: Hidden push button

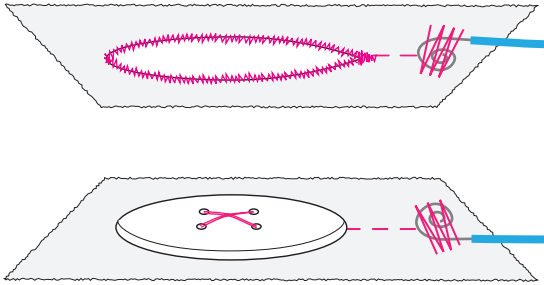
To make hidden push button we need:

- Sew-on metal snap button, or
- Sew-on metallic button, or
- Jeans button
- Non-conductive fabric
- Conductive thread
- Cables

The following are two simpler ways of making hidden push buttons, appropriate when you need a discrete input hidden in garments. You can use the same code as above to try them out. The first one is made with two sew-on metal snap buttons that can be found in any sewing shop. You sew the buttons in place using conductive thread and connect two cables at the end.



The last hidden button is made like a normal button. The trick is to yet again to use conductive thread to sew the button in place and then sew around the button hole using the same thread. Most modern sewing machines have a preprogrammed function for making button holes and will work perfectly for this. Just load you machine with conductive thread and cut the hole in the fabric. This example works the same way either with a normal sew-on button or a jeans button.



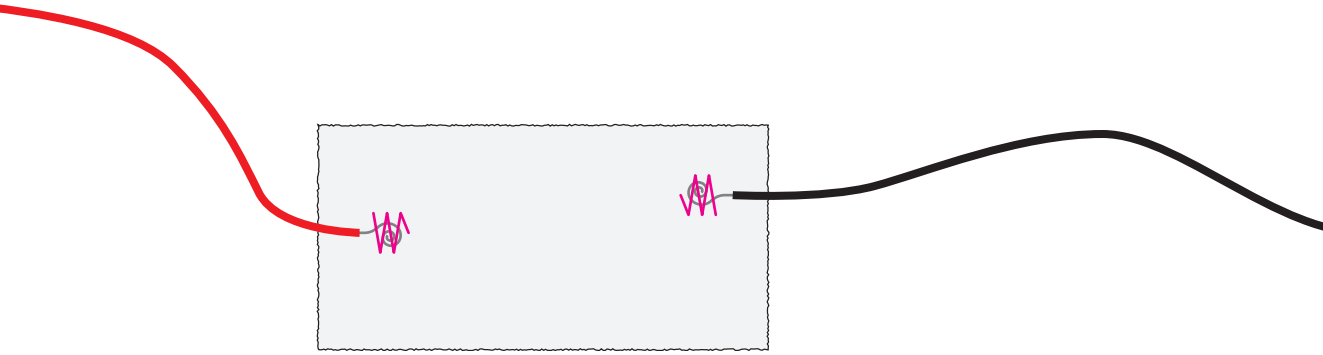
In these examples we have put cables at the end to make it easier for rapid testing. For finalizing prototypes it works as well to sew each button with conductive thread all the way back to your Arduino.

To make a soft speaker we need:

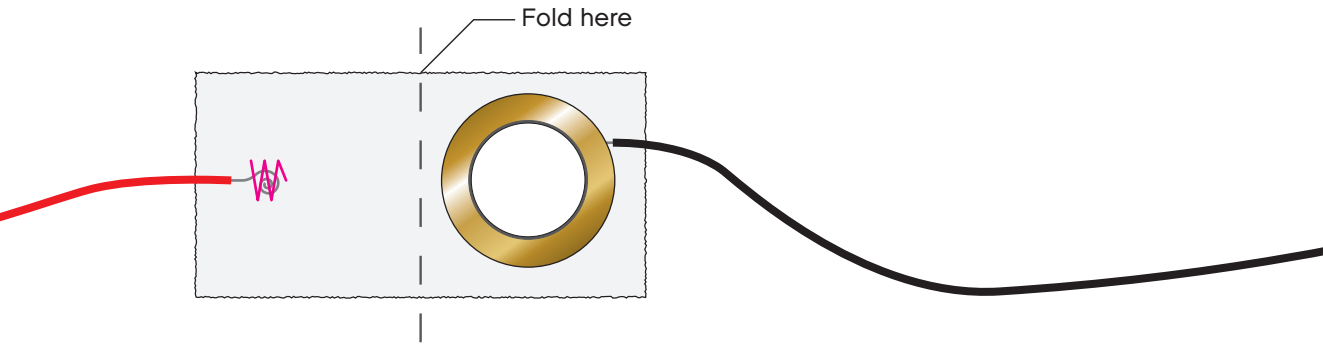
- One piezo speaker
- Non-conductive fabric
- Conductive thread
- Two wires

Part 4: Sound

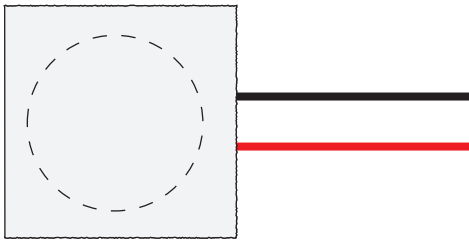
The cheapest way of making sound with the Arduino is to use a piezo speaker. Piezoelectricity speaks of the ability in material to respond to applied mechanical stress. A piezo speaker consists of two metal plates and when electricity is applied to the piezo speaker it will make the metal plates attract and repel generating quick vibration which in turn will generate sound. Piezo speakers are available in a large range of shapes and sizes. In the example we are only going to use the membrane of the piezo speaker. You can buy the piezo speaker membrane in most well sorted electronic shops or you can break the plastic casing of a full piezo speaker. If you choose to break one be careful not to bend the actual membrane while taking the speaker apart. Once you have the membrane start preparing a piece of fabric:



Attach two cables 1/3 of the way on each side of the fabric. Stitch the end of the cable without the covering plastic with conductive thread and then attach the remaining part of cable to the fabric with normal thread. Do not use conductive thread when you stitch the rest of the cable to the fabric since this will create a short circuit once we put the piezo speaker membrane in place. The stitchings with normal thread are only meant to hold the cables in place. Place the element over one side and fold the other side over it and stitch everything together with normal thread. It's good to stitch around the membrane since it need to be quite snug for the cables to connect on booth sides of the membrane.



Now we have a piezo speaker that is a bit softer than a normal one and we can get started on some programs to generate sounds. To make the vibrations that generates sound we will pulse the piezo with power. To make these pulses fast enough to generate vibrations we cant use the normal delay() since this pause isn't quick enough. Instead we will use delayMicroseconds(). In the following example we will generate a vibration that creates the tone A:



```
int piezoPin = 9;
/* plug the piezo to pin 9 */

void setup() {
   pinMode(outputPin OUTPUT);
   /* declare the piezo pin as an output */
}

void loop() {
   digitalWrite(outputPin,HIGH);
   delayMicroseconds(1136);
   /* the time here will determine the tone */

   digitalWrite(outputPin,LOW);
   delayMicroseconds(1136);
   /* the time here will determine the tone */

}
```

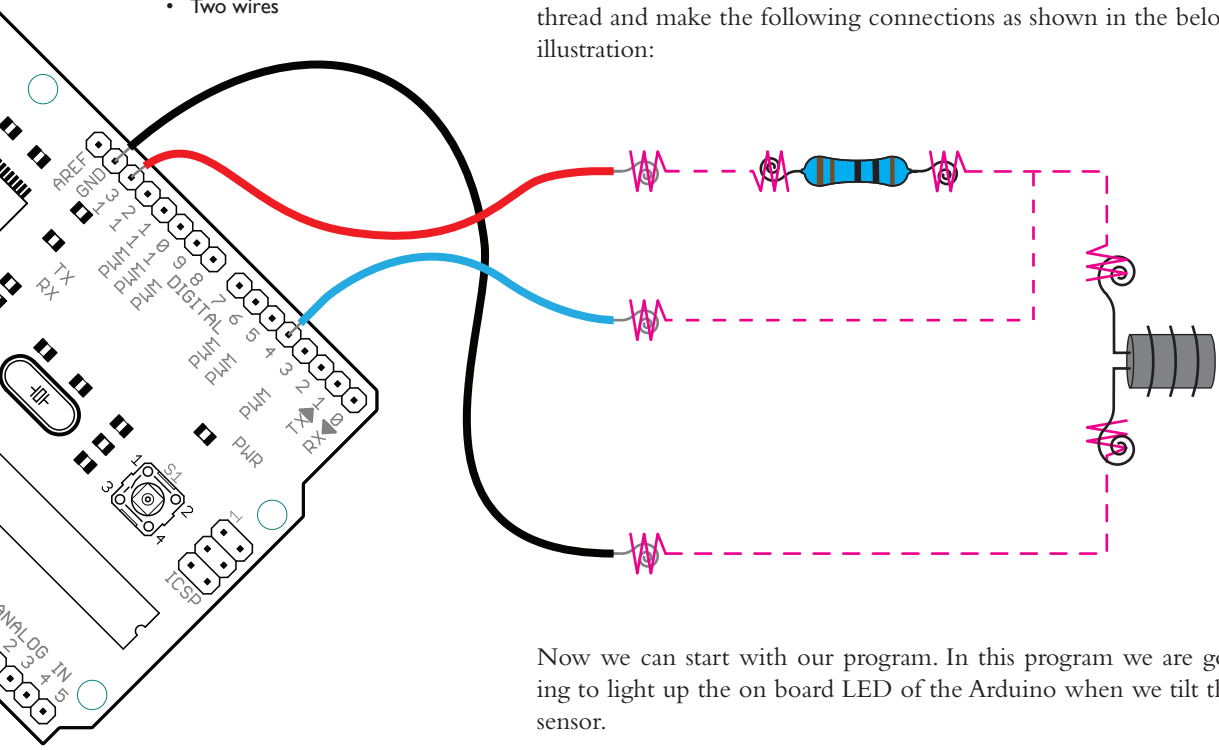
In the above example we turn the 5V to the piezo on and off and in between we make a pause of 1136 microseconds. This pause is what determines what tone is generated. If for instance we pause for 1911 microseconds we would instead get the tone C.

To make a soft speaker we need:

- One 1k Ω resistor
- One tilt sensor
- Non-conductive fabric
- Conductive thread
- Two wires

Part 5: Tilt sensor

Start by attaching the tilt sensor and a 1kΩ resistor to a piece of fabric. Sew a few loops over the tilt sensor to keep in place with some normal thread. Stitch three cables to the fabric using conductive thread and make the following connections as shown in the below illustration:



Now we can start with our program. In this program we are going to light up the on board LED of the Arduino when we tilt the sensor.

```
int myTilt = 4;
/* declare pin for tilt sensor */

int ledPin = 13;
/* declare pin for on board led */

int tiltStatus = 0;
/* declare variable to store the status of the tilt
sensor */

void setup(){
   pinMode(myTilt,INPUT);
   /* declar tilt pin as INPUT */

   pinMode(ledPin,OUTPUT);
   /* declar LED pin as OUTPUT */

}
```

```
void loop(){
   tiltStatus = digitalRead(myTilt);
   /* read tilt pin and store the value */

   if(tiltStatus == HIGH){
   /* check if tilt sensor is tilted */

     digitalWrite(ledPin,HIGH);
     /* turn of LED if tilt sensor is tilted */

   }else{
     digitalWrite(ledPin,LOW);
     /* turn off LED in every other case */

   }
}
```

Once you are done with your code, upload it to the Arduino board and connect your tilt sensor as shown in the illustration on the previous page.

Now when you tilt the fabric the LED next to pin 13 on the Arduino board should light up. As described in the section on the tilt sensor in chapter 2 page 16, you can use this type of tilt sensor to detect movement. It's a matter of making a estimate of how many hits you get from the tilt sensor in a certain amount of time. Let's say that the tilt sensor goes from on and off 2 times every second, then its safe to assume that the garment the tilt sensor is placed on, is moving.

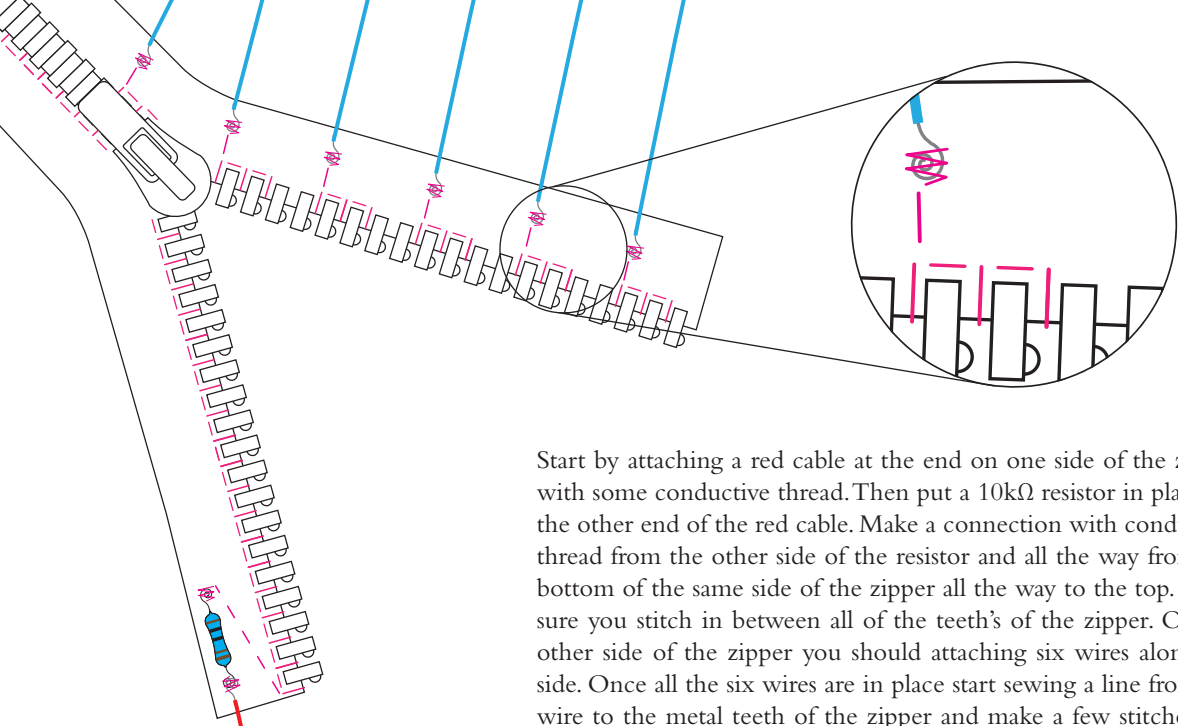
Part 6: The digital zipper

For this example you will need:

- Normal zipper in metal
- Conductive thread
- 10K resistor

There are two ways we can use a normal metal zipper to make an input sensor. The difference between them is in how we read them from the Arduino. In this chapter we are going to read our zipper the digital way.

The advantage of using this zipper compared to the analog zipper on page?, is that the digital will be more precise in it's reading but the disadvantage is that we will get a much lower range of values to read. The range will depend on how many digital pins you want to use. In this example we are going to use six of our digital pins on the Arduino.



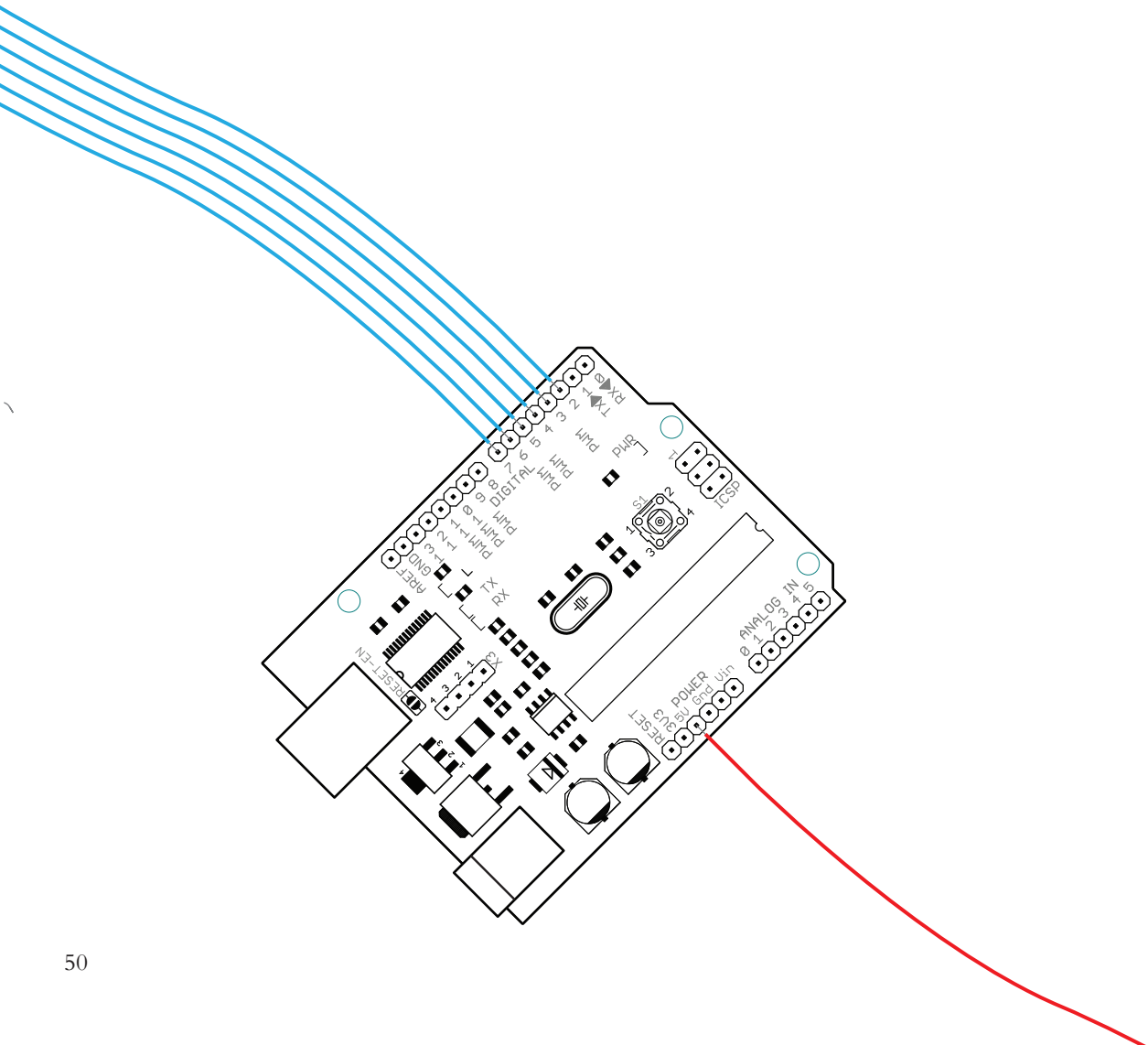
Start by attaching a red cable at the end on one side of the zipper with some conductive thread. Then put a 10kΩ resistor in place on the other end of the red cable. Make a connection with conductive thread from the other side of the resistor and all the way from the bottom of the same side of the zipper all the way to the top. Make sure you stitch in between all of the teeth's of the zipper. On the other side of the zipper you should attaching six wires along the side. Once all the six wires are in place start sewing a line from the wire to the metal teeth of the zipper and make a few stitches between the teeth, upwards, on the zipper.

Make sure to leave a gap between the stitchings from one wire to the next one. None of the six wires should be connected to one another with conductive thread. Once the zipper is stitched up we can start with our code:

```
int pin2 = 2;          // declaration of digital pin
int pin3 = 3;          // declaration of digital pin
int pin4 = 4;          // declaration of digital pin
int pin5 = 5;          // declaration of digital pin
int pin6 = 6;          // declaration of digital pin
int pin7 = 7;          // declaration of digital pin
int pin8 = 8;          // declaration of digital pin

void setup(){
   pinMode(pin2,INPUT);
   /* set the mode of the pin to an INPUT */

   pinMode(pin3,INPUT);
   /* set the mode of the pin to an INPUT */

   pinMode(pin4,INPUT);
   /* set the mode of the pin to an INPUT */

   pinMode(pin5,INPUT);
   /* set the mode of the pin to an INPUT */

   pinMode(pin6,INPUT);
   /* set the mode of the pin to an INPUT */
```

```
   pinMode(pin7,INPUT);
   /* set the mode of the pin to an INPUT */

   Serial.begin(9600);
   /* start serial communication and set the communi
   cation speed to 9600 baud */
}

void loop(){
   if(digitalRead(pin2) == HIGH){
   /* check if the pin is HIGH */

     Serial.println(2);
   /* if it is, send the number of that pin */

   }

   if( digitalRead(pin3) == HIGH){
   /* check if the pin is HIGH */

     Serial.println(3);
     /* if it is, send the number of that pin */

   }

   if(digitalRead(pin4) == HIGH){
   /* check if the pin is HIGH */

     Serial.println(4);
     /* if it is, send the number of that pin */

   }

   if( digitalRead(pin5) == HIGH){
   /* check if the pin is HIGH */

     Serial.println(5);
     /* if it is, send the number of that pin */

   }

   if( digitalRead(pin6) == HIGH){
   /* check if the pin is HIGH */

     Serial.println(6);
     /* if it is, send the number of that pin */

   }

   if(digitalRead(pin7) == HIGH){
   /* check if the pin is HIGH */

     Serial.println(7);
     /* if it is, send the number of that pin */

   }

   delay(200);
   /* make a pause */
}
```

Upload the code to your Arduino board and connect the zipper. The red cable goes to 5V pin on the board and the rest of the wires should be connected to digital pins 2 to 7. Once everything is in place you can open your serial monitor and test your zipper. By moving it up and down the Arduino should give you number from 2 to 7. What we are doing here is to measure where the zipper head is located since it connects the 5V to a digital pin, thus sending the current through that pin, making the value HIGH and then, in the code, transforming that information into a number, telling us which one of the pins that is communicating.



Chapter 6: Using analog pins

As said earlier in this book, the real world is not digital and sometimes you can't transfer changes in our environment into digital readings. For example temperature does not change from only hot to cold, it changes within a range of different values and normally these changes occur slowly over time. This is why we often use analog sensors to read environmental parameters like temperature, light and motion. This resulting information is stored as sequential digital data. Since the Arduino can't handle information like humans, we need to translate analog information in a way that the Arduino can understand it.

Analog sensors can transform data in our environment into a voltage value between 0V and 5V. These values are different from the HIGH and LOW that the digital pins use. For the digital pins HIGH and LOW means respectively 5V and 0V and nothing else. But the analog pins can make sense of values like 0.3V, 1.6V, 3,2V and so on.
The resolution in between a max and min value differs from microprocessors. Arduino can only distinguish 1024 levels in the range of 0V to 5V.

Note:
When we do analog reading the Arduino starts counting at 0 so you will never get a reading that is over 1023.

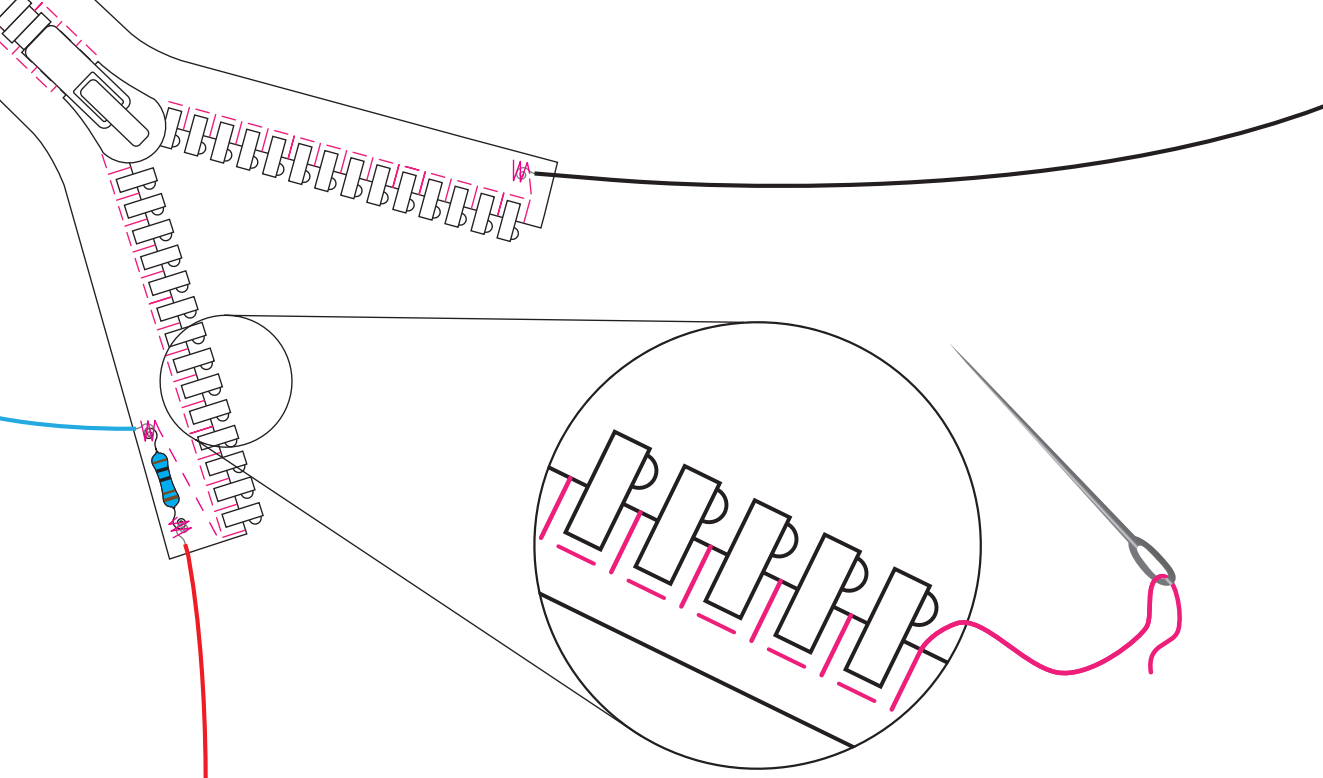
Part 1: The analog zipper

The analog zipper is different from the digital one in the earlier chapter. The digital zipper can only be HIGH or LOW (0V or 5V). The analog zipper on the other hand can give you a range of values in between 0V and 5V. To make a analog zipper you need a normal zipper, some conductive thread and a 10kΩ resistor.

To make an analog zipper we need:

- One metal zipper
- Conductive thread
- One 10kΩ resistor
- Two wires

Start off by sewing the resistor in place. Then we sew one red wire to one end of the resistor and a blue one to the other end of the resistor. The colors of the cables are used so you will remember what cable goes where. From the same place as the blue cable start sewing in between every tooth of the zipper. When we reach the end of one side of the zipper do the same on the other side and back down again.



Make sure the thread is intact all the way round the zipper or it will not work. Once you have reached the end of the zipper on the other side attach a black cable and sew it into place. Now our zipper is done and we can start writing our code to read what kind of values the zipper will give. To be able to show what kind of values the zipper gives us we will need to use serial communication so that the Arduino can report back to the computer and show us the values in the serial monitor:

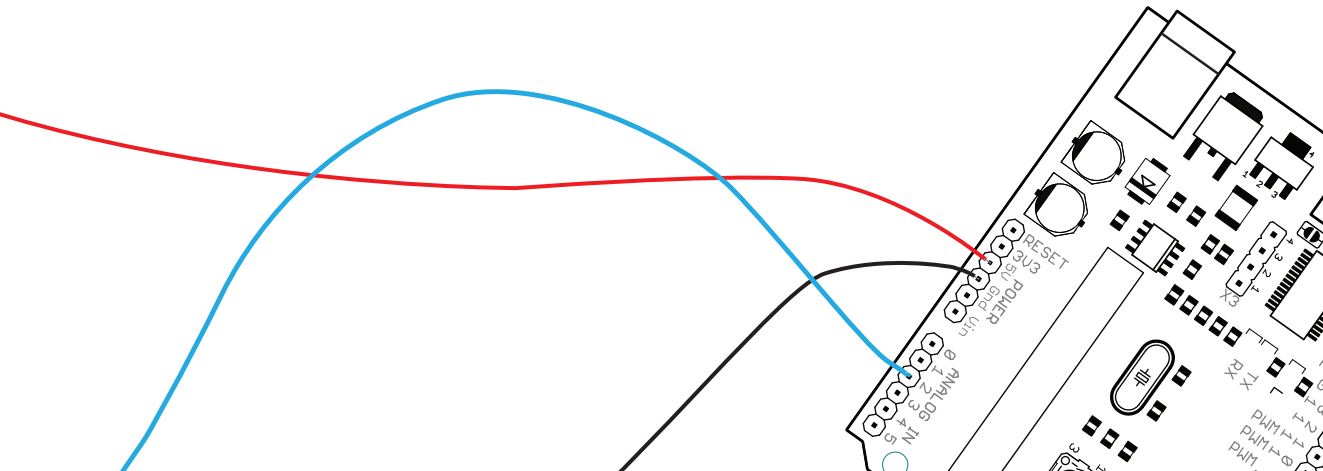
```
int analogPin = 2
/* here we declare that we want to use analog pin 2
and call it analogPin */

int myZipper = 0;
/* here we have declared a integer that will act as a
temporary variable so we can store the values that the
zipper will give us */

void setup(){
   Serial.begin(9600);
   /* this will setup your communication and set the
   communication to 9600 baud */

}
```

```
void loop(){
   myZipper = analogRead(analogPin);
   /* here we do an analog reading on analogPin and
   save the value in the varible myZipper */

   Serial.println(myZipper)
   /* this command will send the value stored in myZ
   ipper and send it over the serial port and then
   make a break. */

   delay(200);
//pause for 200 milliseconds
}
```

Note:
We don't have to declare the mode for the analog pin as we would have to do if it was a digital pin.

The first thing we do in our void loop() is to make an analog reading on our analog pin 2. The value returned by this command is saved in the variable myZipper. After the reading of the analog pin we print this value over the serial port and anything connected to the Arduino (for instance; a computer) will receive whatever was stored in myZipper. The last thing we do is adding a delay in our program since even if the Arduino is small it still can communicate faster than a normal computer so the delay lets your computer keep up with the Arduino.

Now we can upload our code to the Arduino board. Once the code is uploaded we plug the red cable into the port that says 5V, the black cable to one of the GND ports and the last cable to our analog pin 2. If you declare your analogPin in your program as anything else than 2 you will have to put the yellow cable in the corresponding analog port:



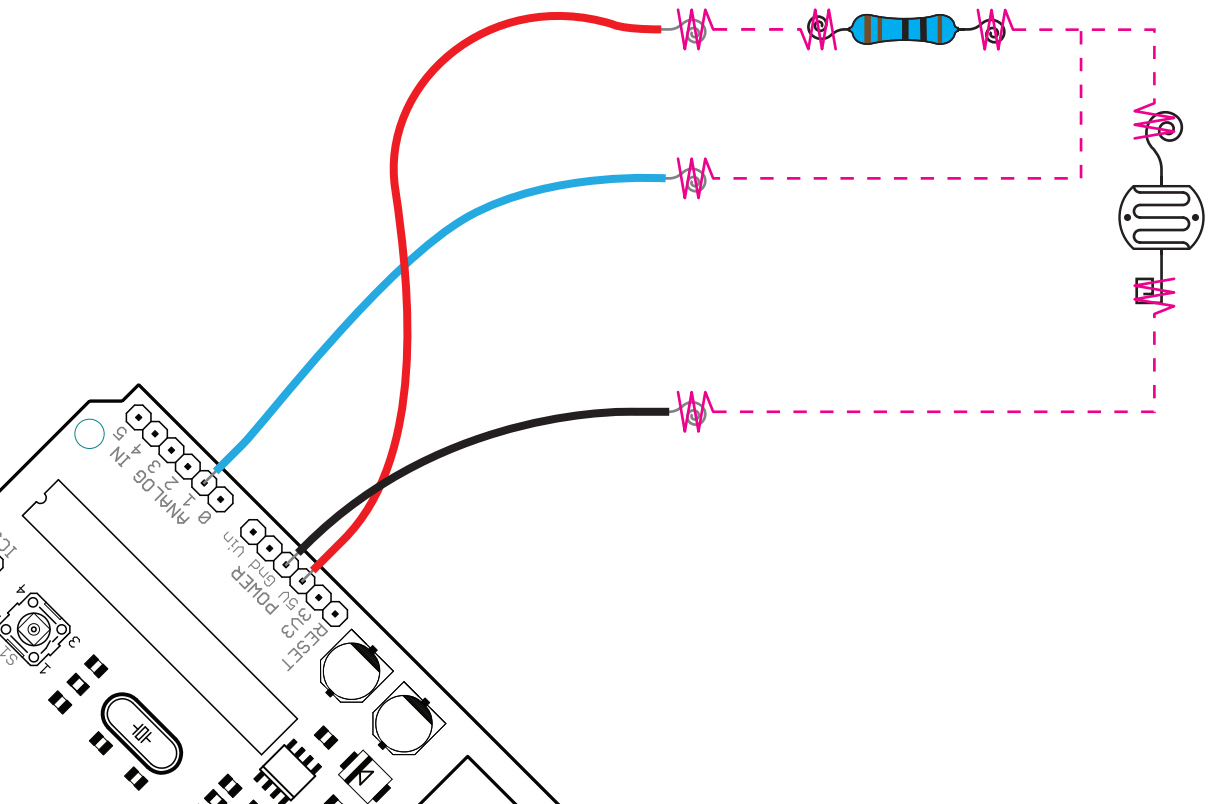
Once we have uploaded the code and connected your cables we can open our serial monitor, which is the button on the far left in your Arduino IDE. Now values should appear in our serial monitor and if we open and close the zipper the values should change. If you can't see any numbers in your monitor, make sure the communication speed set in the Arduino IDE is the same as in your program (9600 baud in this example). Each zipper will give you different values since making an analog zipper isn't an exact science yet, the slightest difference will affect the result. This form of home made analog sensor is still good enough to be used for most prototypes that need a hidden input device.

Part 2: Using an LDR

For the following example you will need:

- One LDR
- One 10k resistor
- Conductive thread
- Three wires

The following drawing illustrates how to sew the LDR and resistor onto a piece of fabric and where the wires should connect to the arduino.:



The resistor in the example is used as what is called a "pull up resistor". We sometimes use pull up resistors to ensure we won't get 5V straight back in the Arduino. A pull up resistor is the same as a normal resistor but we use them to lower the output voltage from a power source. Connect a red cable to one of the legs of the resistor and to 5V on the Arduino. In between the resistor and the LDR we put a cable that connects to our analog pin on the Arduino. The same connection goes to one leg of the LDR. It doesn't matter which leg you connect it to. From the other leg of the LDR, sew a cable that connects to GND on the Arduino:

When you have connected everything lets try the following code and see what values the LDR will give us:

```
int analogPin = 2;
/* the analog pin we are using on the Arduino */

int myLDR = 0;
/* temporary variable to store the LDR value */

void setup(){
   Serial.begin(9600);
/* setting up communication and communication speed */

}

void loop(){
   myLDR = analogRead(analogPin);
   /* read the value from ther LDR and store it */

   Serial.print(myLDR);
   /* print the value stored in myLDR */

   delay(200);
}
```

In the above code example we read the value, store it and print it back to the computer. Don't forget to open your serial monitor in the Arduino IDE and set it to 9600 baud to be able to see the value printed from the Arduino. Once you get everything to work, try covering the LDR with your hand and check what value the Arduino prints back. Remember this value and try the next code example.

```
int analogPin = 2;
/* the analog pin we are using on the Arduino */

int myLDR = 0;
/* temporary variable to store the the LDR value */

int myDarkNumber = 100;
/* the threshold for dark */

int ledPin = 13;

void setup(){
   Serial.begin(9600);
   /* setting up communication and speed */

   pinMode(ledPin,OUTPUT);
   /* declare ledPin as OUTPUT */

}

void loop(){
   myLDR = analogRead(analogPin);
   /* read the value from the LDR and store it */

   if (myLDR <= myDarkNumber){
     digitalWrite(ledPin,HIGH);
   }else{
     digitalWrite(ledPin,LOW);
   }
}
```

In this example the variable named "myDarkNumber" is the value you got back from the previous example when you covered the LDR with your hand. In my case it is 100 but you should change this number to your value. This program will read the value from the LDR and compare it to your threshold variable (myDarkNumber) and if the LDR is below or equal to this threshold the internal LED on the Arduino board will light up, if not the LED will remain off.

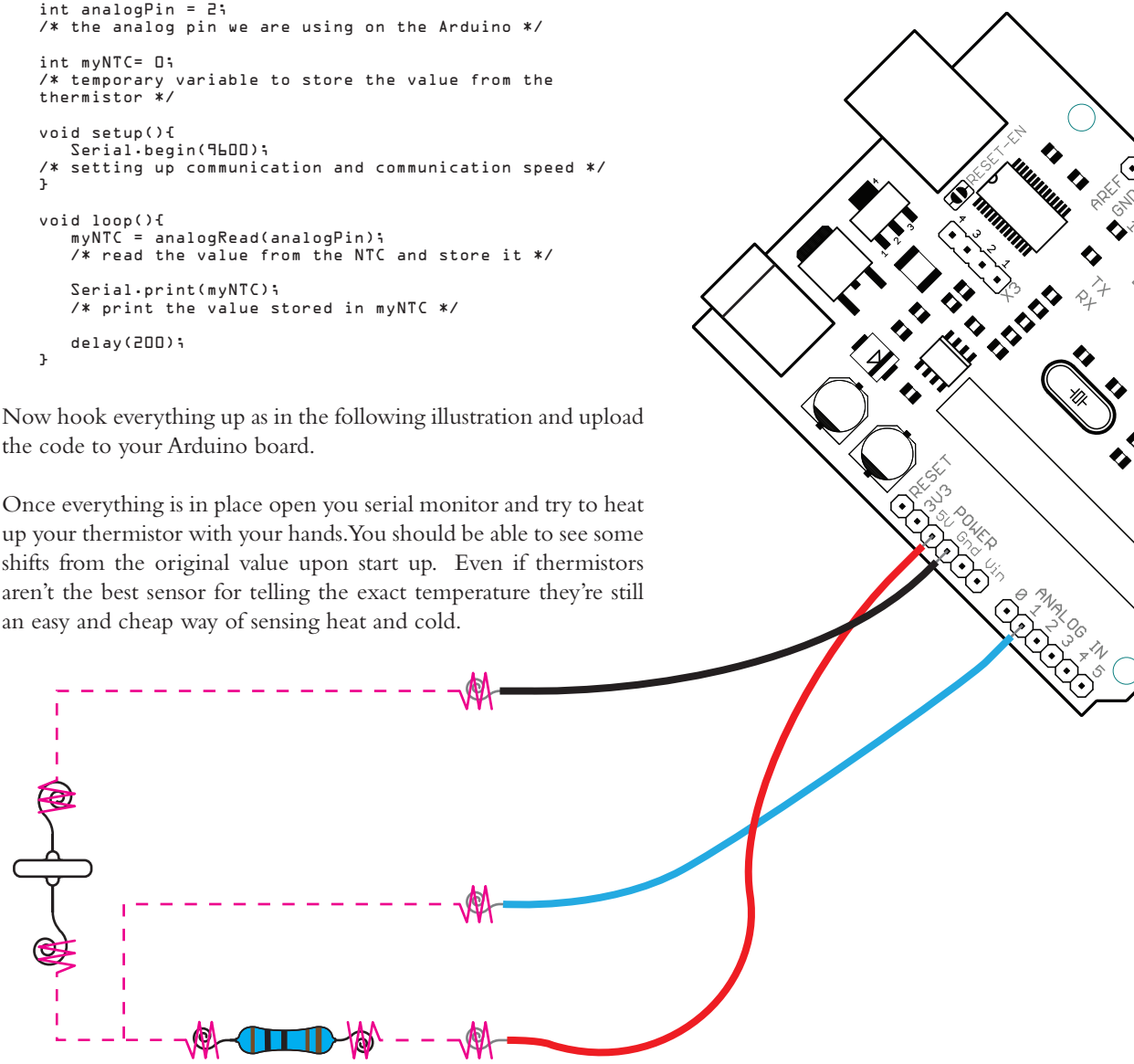
Part 3: Using an NTC (Thermistor)

The following code to read the thermistor is the same as for the LDR which works fine since they're both analog sensors:

```
int analogPin = 2;
/* the analog pin we are using on the Arduino */

int myNTC= 0;
/* temporary variable to store the value from the
thermistor */

void setup(){
   Serial.begin(9600);
/* setting up communication and communication speed */
}

void loop(){
   myNTC = analogRead(analogPin);
   /* read the value from the NTC and store it */

   Serial.print(myNTC);
   /* print the value stored in myNTC */

   delay(200);
}
```

Now hook everything up as in the following illustration and upload the code to your Arduino board.

Once everything is in place open you serial monitor and try to heat up your thermistor with your hands.You should be able to see some shifts from the original value upon start up. Even if thermistors aren't the best sensor for telling the exact temperature they're still an easy and cheap way of sensing heat and cold.



Chapter 7: Moving stuff

• DC motors

These types of motor are usually found in toys and devices that don't need a lot of accuracy in terms of angle and speed. A DC motor runs freely when power is applied and it's hard to control the speed and position of them but with some extra circuit and components this can be improved. The good thing with DC motor is that some can be found very cheap and they are often used is cheep electrical device that has some kind of motion. Hacking for a DC motor is quite easy since it only uses two cables. Another good thing about DC motors are that some of them can be powered with less then 2.5V.

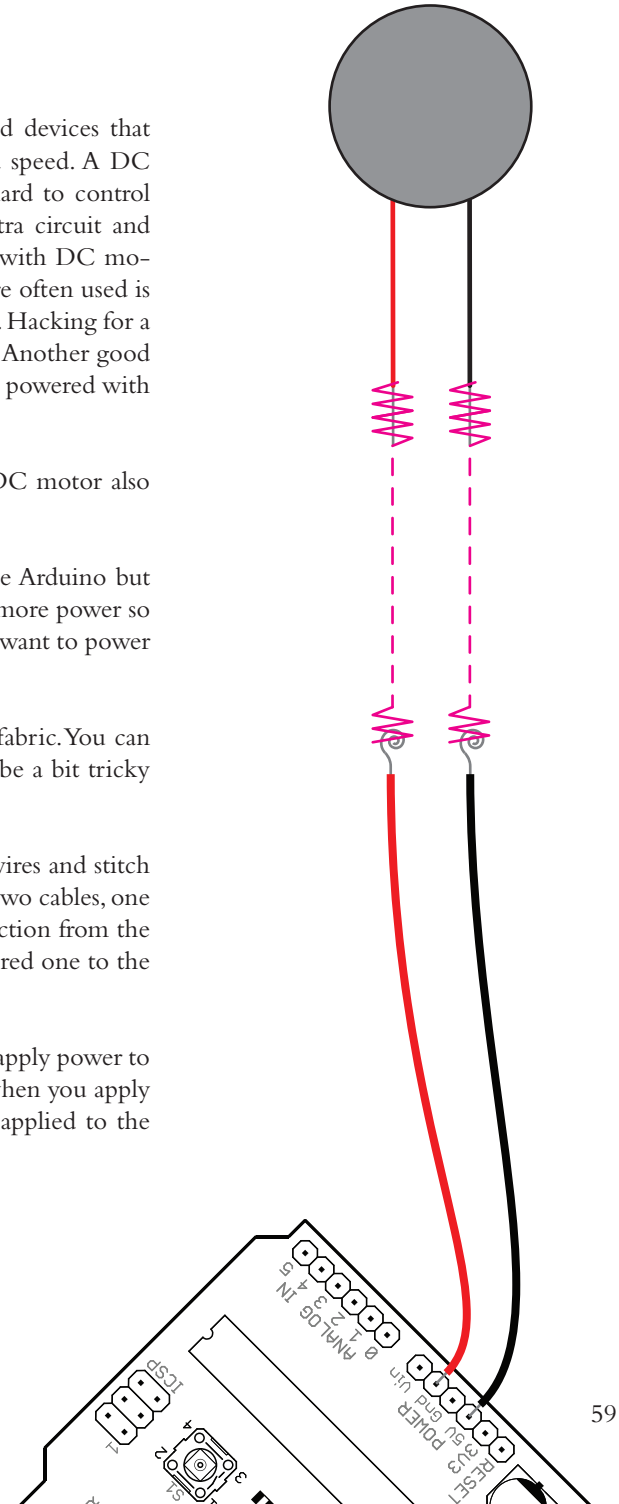
The following example shows how to use a small DC motor also known as a mini vibrator.

This can be run straight from the digital pins on the Arduino but beware that with other DC motors you might need more power so be sure to use appropriate resistor or transistor if you want to power it with external power.

To sew a mini vibrator start to glue it to a piece of fabric. You can also sew over it using normal thread but this might be a bit tricky since most mini vibrators are round.

Once the vibrator is in place, cut the plastic of the wires and stitch over theme with some conductive thread. Now sew two cables, one black and one red to your fabric and make a connection from the black one to one cable on the vibrator and from the red one to the other one:

With a DC motor it doesn't matter which cable you apply power to and which is connected to GND. The difference is when you apply power to on cable the motor spins one way and if applied to the other side it will spin the other way.



Now when we have everything in place lets write the following program:

```
int motoPin = 5;

void setup(){
   pinMode(motoPin,OUTPUT);
}

void loop(){
   digitalWrite(motoPin,HIGH);
   delay(1000);
   digitalWrite(motoPin,LOW);
   delay(1000);
}
```

This is a simple program to turn the vibrator on for one second and off for one second. It's also possible to turn the vibration speed up and down gradually using PWM:

```
int motoPin = 5;
/* connect your led to digital pin 5 */

void setup(){
   pinMode(motoPin,OUTPUT);
   /* declare ledPin as an OUTPUT */
}

void loop(){
   for(int i = 0; i < 255; i++){
   /* as long as i is smaller than 255, increases i by
   one */

     analogWrite(motoPin ,i);
     /* fade the speed of the vibrator with i */

     delay(30);
     /* small pause so we can see each step */

   }

   for(int i = 0; i > 0; i--){
   /* as long as i is bigger than 0, decreases i by
   one */

     analogWrite(motoPin,i);
     /* fade the speed of the vibrator with i */

     delay(30);
     /* small pause so we can see each step */

   }
}
```

The above example will turn the vibrator slowly up to the maximum speed and back again.

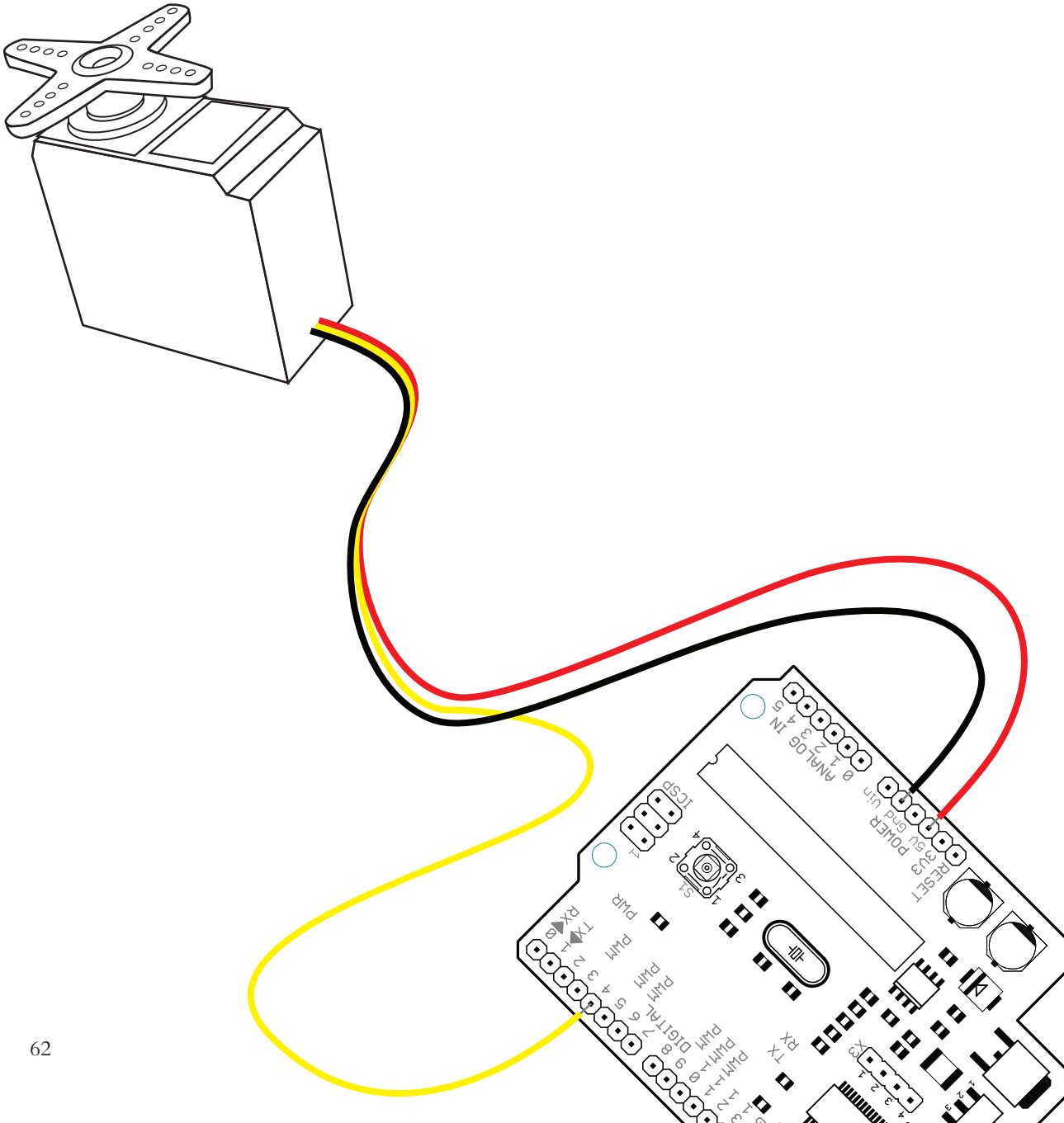
• Servo motors

There are two types of servo motors, normal servo and continuous rotation servos. The normal ones are controlled with PWM and rotate to a position depending on the pulse width. The normal servos can only rotate from 0 degrees to 180 degrees. The continuous rotation servos keep on rotating when pulse with power. The most common continuous rotation servos from Parallax rotate in one direction when pulsed with power with a delay above 1500 microseconds and the other way when pulse with a delay below 1500 microseconds.

Depending on your project you will have to use your imagination to hide a servo motor. In the following example we will only show how to control servo motors since there are no general ways of implementing servos in fabrics. Extending the wire length from the servo to Arduino is still possible by the use of conductive thread. The first example is a simple program that illustrates how to move a continuous servo motor forward and back:

```
int motoPin = 4;
/* digital pin for the servo */

void setup(){
   pinMode(motoPin,OUTPUT);
   /* declare digital pin as output */

}

void loop(){
   for(int i = 0; i < 100; i++);{
   /* loop 100 times so we can see the servo move */

     digitalWrite(motoPin,HIGH);
     delayMicroseconds(1850);
     /* the delay above 1500 to make the servo go one
     way */

     digitalWrite(motoPin,LOW);
     delayMicroseconds(1850);
   }

   for(int j = 0; j < 100; j++);{
   /* loop 100 times so we can see the servo move */

     digitalWrite(motoPin,HIGH);
     delayMicroseconds(1250);
     /* the delay below 1500 to make the servo go the
     other way */

     digitalWrite(motoPin,LOW);
     delayMicroseconds(1250);
   }
}
```

This program will make the continuous rotation servo go 100 steps one way then 100 steps the other way. Once your code is done, upload it to The Arduiono board and connect the motor like in the following illustration.



Chapter 8: Complex examples

Part 1: Oscillation with a Zipper

Oscillation is the variation in time of a central value or between two or more states. With an oscillator we can change the tone manually of our piezo speaker from the previous example.
The analog Zipper on page ? will work perfect as an oscillator since it gives us nice range of values that can be remapped to a range of different tones.

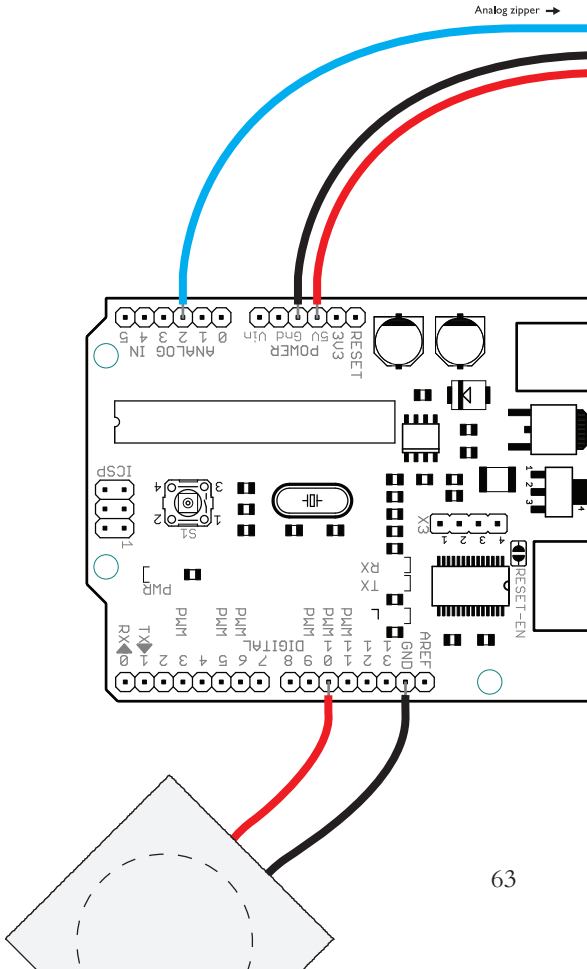
The following program will read the values from the zipper, remap these values and then use the value to set the tone to be played by the piezo speaker:

```
int piezoPin = 10;
/* declares the pin the piezo speaker is attached to
*/

int analogPin =  2;
/* declares the pin the zipper is attached to */

int myTemp = 0;
/* declares a temporary storage variable */

int myRemapValue = 0;
/* declares a variable for the remapped value */

void loop(){
   pinMode(piezoPin,OUTPUT);
   /* declare piezoPin as output */

}

void loop(){
   myTemp = analogRad(analogPin);
   /* read and store the value from the zipper */

   myRemapValue = map(myTemp,100,300,0,2000);
/* remapp the value from the zipper to fit within the
range of 0 through 2000 */

   digitalWrite(piezoPin,HIGH);
   /*send 5V to the piezo speaker */

   delayMicroseconds(myRemapValue);
   /* pause with the remapped value */

   digitalWrite(piezoPin,LOW);
   /* send 0V to the piezo speaker */

   delayMicroseconds(myRemapValue);
   /* pause with the remapped value again */

}
```

For this example you will need:

- The piezo speaker (from page ?)
- The analog zipper (from page ?)

Note:
You need to know what the maximum and minimum value the zipper gives you. In this example we will use a fictive range of 100 to 300.



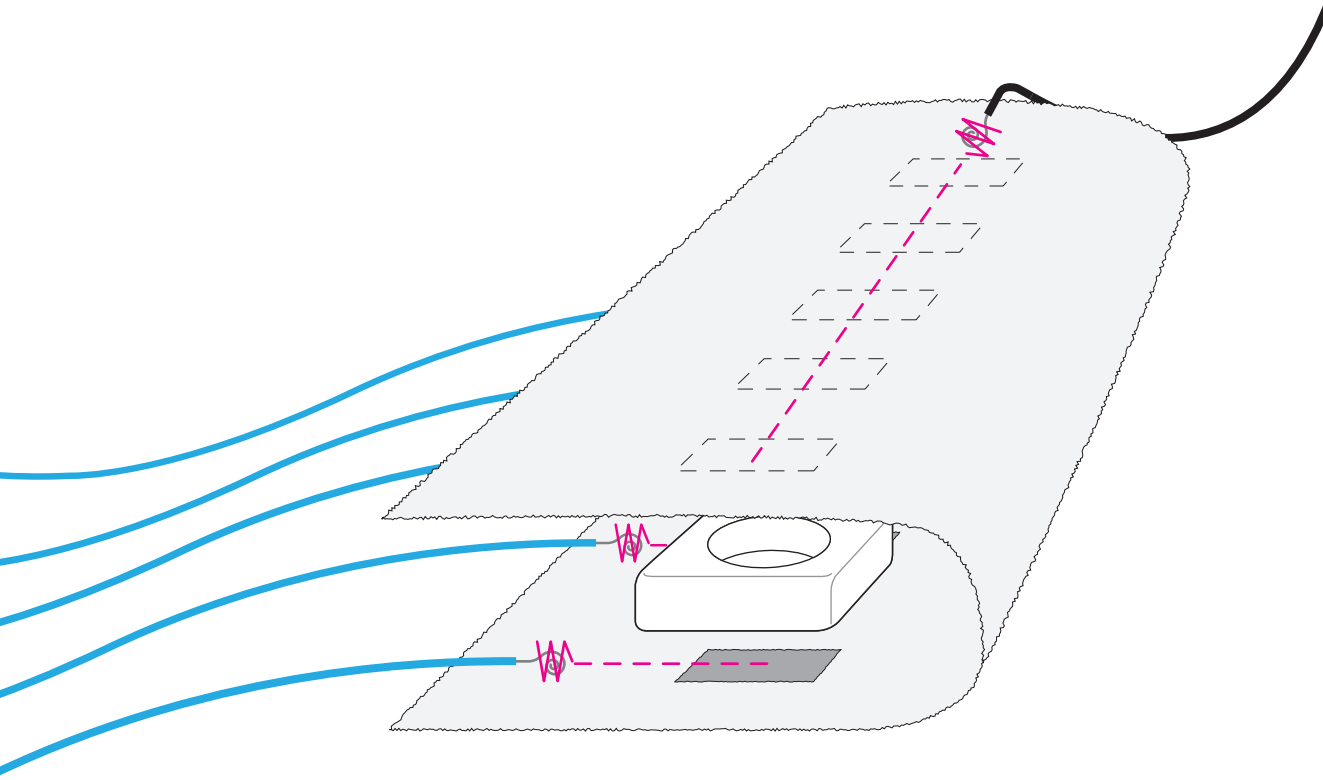
This example implements the use of the map() function. To learn more about the map() function turn to page ?. Your zipper might give values good enough to set some tones directly, but they will have much lower range than wanted since an analog sensor never gives a value above 1023. Once you're done with the code try hooking everything up to the Arduino board as shown in the illustration and test the zipper oscillator.

Part 2: The soft synthesizer

To make a soft synthesizer you will need:

- Five soft pushbuttons
- One piezo speaker
- Five wires
- Conductive thread
- Conductive fabric
- One piece of normal fabric

Use the same method of creating soft pushbutton as shown in the example on page ?. But for this example we will use one big piece of fabric to make place for all pushbuttons:



On one side of the buttons attach 5 cables and sew a connection to one piece of conductive fabric from one cable to another for all the pushbuttons. From the other side of the pushbutton to the other piece of conductive fabric we will sew a connection to a long line of thread and at the end of this thread we will connect a black wire. This will act as a ground connection for all the buttons since there is not enough GND ports on the Arduino for all the buttons.

Now that we have the soft keyboard part ready for our soft synthesizer we can start on our program:

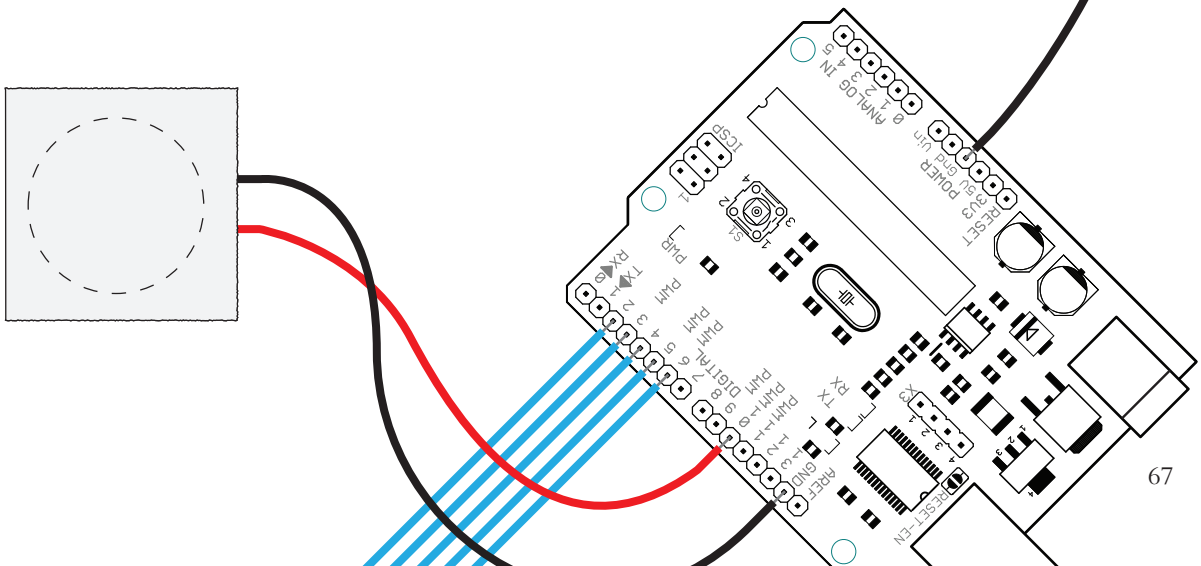
```
int piezoPin = 10 ;
/* declare the pin which the piezo speaker is attached
to */

int keyOne = 2;
/* declare the first soft button */

int keyTwo = 3;
/* declare the second soft button */

int keyThree = 4;
/* declare the third soft button */

int keyFour = 5;
/* declare the fourth soft button */

int keyFive = 6;
/* declare the fifth soft button */

void setup(){
   pinMode(piezoPin,OUTPUT);
   /* declare the piezopin as output */

   pinMode(keyOne,INPUT);
   /* declare the first soft button as input */

   pinMode(keyTwo,INPUT);
   /* declare the second soft button as input */

   pinMode(keyThree,INPUT);
   /* declare the third soft button as input */

   pinMode(keyFour,INPUT);
   /* declare the fourth soft button as input */

   pinMode(keyFive,INPUT);
   /* declare the fifth soft button as input */

   digitalWrite(keyOne,HIGH);
   /* send 5V to the first soft button */

   digitalWrite(keyTwo,HIGH);
   /* send 5V to the second soft button */

   digitalWrite(keyThree,HIGH);
   /* send 5V to the third soft button */

   digitalWrite(keyFour,HIGH);
   /* send 5V to the fourth soft button */
```

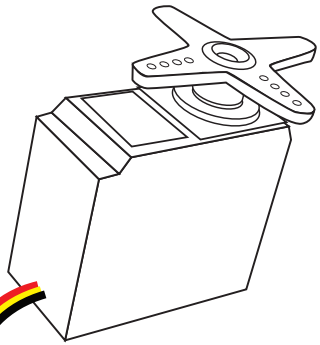
```
   digitalWrite(keyFive,HIGH);
   /* send 5V to the fifth soft button */

}

void loop(){
   if(digitalRead(keyOne) == LOW){
   /* tests if the first soft button is pushed */

     digitalWrite(piezoPin,HIGH);
     /* sends 5V to the piezo speaker */

     delayMicroseconds(1911);
     /* waits for 1911 microseconds (creates a C) */

     digitalWrite(piezoPin,LOW);
     /* stops sending 5V to the piezo speaker */

     delayMicroseconds(1911);
     /* waits for another 1911 microseconds */

   }

   if(digitalRead(keyTwo) == LOW){
   /* tests if the second soft button is pushed */

     digitalWrite(piezoPin,HIGH);
     /* sends 5V to the piezo speaker */

     delayMicroseconds(1703);
     /* waits for 1703 microseconds (creates a D) */

     digitalWrite(piezoPin,LOW);
     /* stops sending 5V to the piezo speaker */

     delayMicroseconds(1703);
     /* waits for another 1703 microseconds */

   }

   if(digitalRead(keyThree) == LOW){
   /* tests if the third soft button is pushed */

     digitalWrite(piezoPin,HIGH);
     /* sends 5V to the piezo speaker */

     delayMicroseconds(1517);
     /* waits for 1517 microseconds (creates an E) */

     digitalWrite(piezoPin,LOW);
     /* stops sending 5V to the piezo speaker */

     delayMicroseconds(1517);
     /* waits for another 1517 microseconds */

   }

   if(digitalRead keyFour) == LOW){
   /* tests if the fourth soft button is pushed */

     digitalWrite(piezoPin,HIGH);
     /* sends 5V to the piezo speaker */

     delayMicroseconds(1432);
     /* waits for 1432 microseconds (creates an F) */
```

```
        digitalWrite(piezoPin,LOW);
        /* stops sending 5V to the piezo speaker */

        delayMicroseconds(1432);
        /* waits for another 1432 microseconds */
      }

      if (digitalRead(keyFive) == LOW){
      /* tests if the fifth soft button is pushed */

        digitalWrite(piezoPin,HIGH);
        /* sends 5V to the piezo speaker */

        delayMicroseconds(1276);
        /* waits for 1276 microseconds (creates a G) */

        digitalWrite(piezoPin,LOW);
        /* stops sending 5V to the piezo speaker */

        delayMicroseconds(1276);
        /* waits for another 1276 microseconds */

      }
  }
```

This program check all the buttons as soon as one of them is pushed it will start to play the tone of that key. In this example we are going to use the tones C, D, E, F and G. At the end of this chapter you can find a chart of tone and their corresponding delay time. Once you have the program ready, upload it to your Arduino board and connect your soft keyboard and piezo speaker as in the following illustration:

Note:
The five blue cables are connected to digital pins 2 through 6 on the Arduino board. The black cable is connected to GND. The black cable from the soft piezo speaker is also connected to one of the GND pins and the red cable is connected to digital pin 10.





The piezo speaker could also be implemented into the keyboard fabric to make your design nicer.

Chart of tones and there delay time

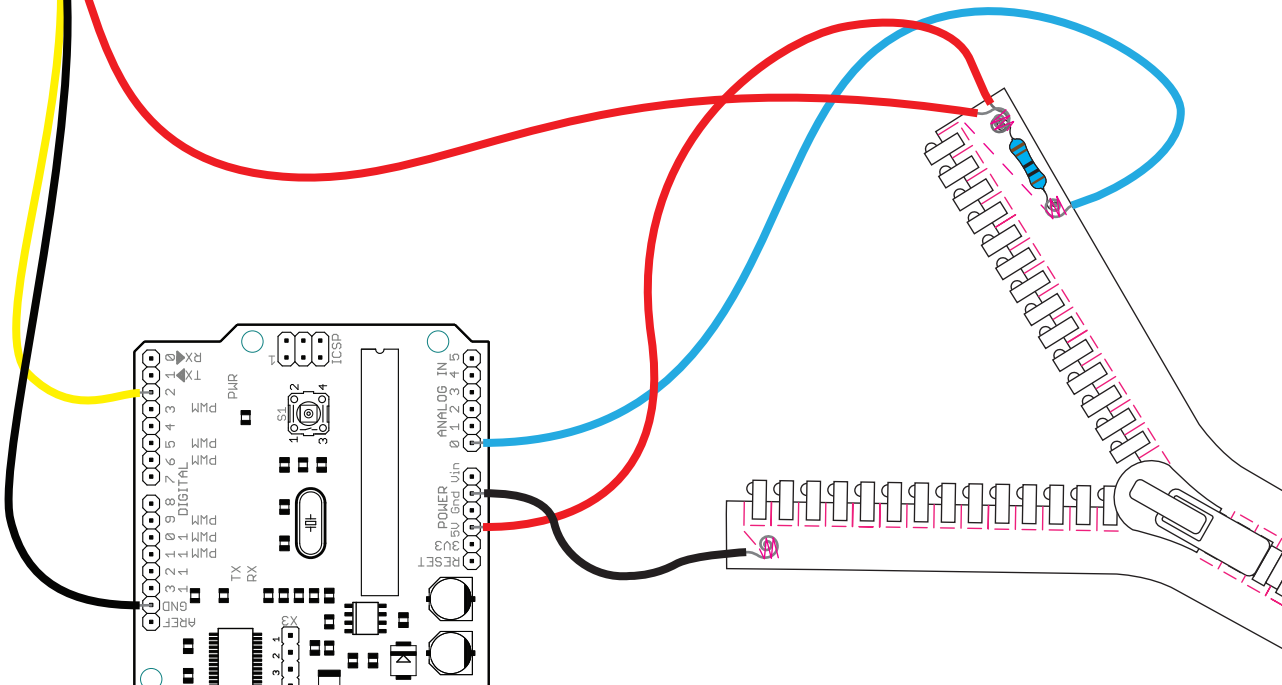
tone	delay
C	1911
D	1703
E	1517
F	1432
G	1276
A	1136
B	1012
C	0956

To control a servo with a zipper you will need:

- The analog zipper (page ?)
- One normal servo motor

Part 3: Controlling a normal servo with a zipper

In this example we will use the analog zipper from page ? To control the rotation of a normal servo motor. The following illustration show how to connect both the servo and the zipper:



To write the program you first have to know the maximum and minimum value of your zipper. Turn to page ? If you don't remember how to read values from a analog sensor.
The minimum and maximum value from the zipper will be remapped in our program using the map() function to the minimum and maximum rotation value of the servo motor:

```
int servoPin = 2;
/* digital pin for the servo motor */

int servoMin = 500;
/* minimum servo position */

int servoMax = 2500;
/* maximum servo position */

int pulse = 0;
/* amount to pulse the servo */

long lastPulse = 0;
/* the time in milliseconds of the last pulse */

int refreshTime = 20;
/* the time needed in between pulses */

int myZipper = 0;
/* the value returned from the analog sensor */

int analogPin = 0;
/* the analog pin that the sensor's on */

void setup() {
   pinMode(servoPin,OUTPUT);
   /* Set servo pin as an output pin */

   pulse = servoMin;
   /* Set the motor position value to the minimum */

   Serial.begin(9600);
}

void loop() {
   myZipper = analogRead(analogPin) ;
   /* read the analog input */

   pulse = map(myZipper,0,1023,servoMin,servoMax);
   /* remapp the value from zipper to a range between
   minPulse and maxPulse */

   if (millis() - lastPulse >= refreshTime) {
   /* if millis (which is the amount of milliseconds
   the arduino has been powered on) minus the value
   from "lastPulse" is larger than or equal to the
   value in "refreshTime" then this piece of code is
   triggered */

     digitalWrite(servoPin,HIGH);
     /* Turn the motor on */

     delayMicroseconds(pulse);
     /* Length of the pulse sets the motor position */

     digitalWrite(servoPin,LOW);
```

```
        /* Turn the motor off */

        lastPulse = millis();
        /* save the time of the last pulse */
      }
  }
```

This program will read the value from the zipper and remap this value into a pulse that will be used to set the position of the motor. Once you are done with your code upload it to your board and try to move the zipper to move the servo motor. In this program you have to add the maximum and minimum value of your zipper in the map function " map(myZipper,"your zipper min" ,"your zipper max", 500, 2500); ".

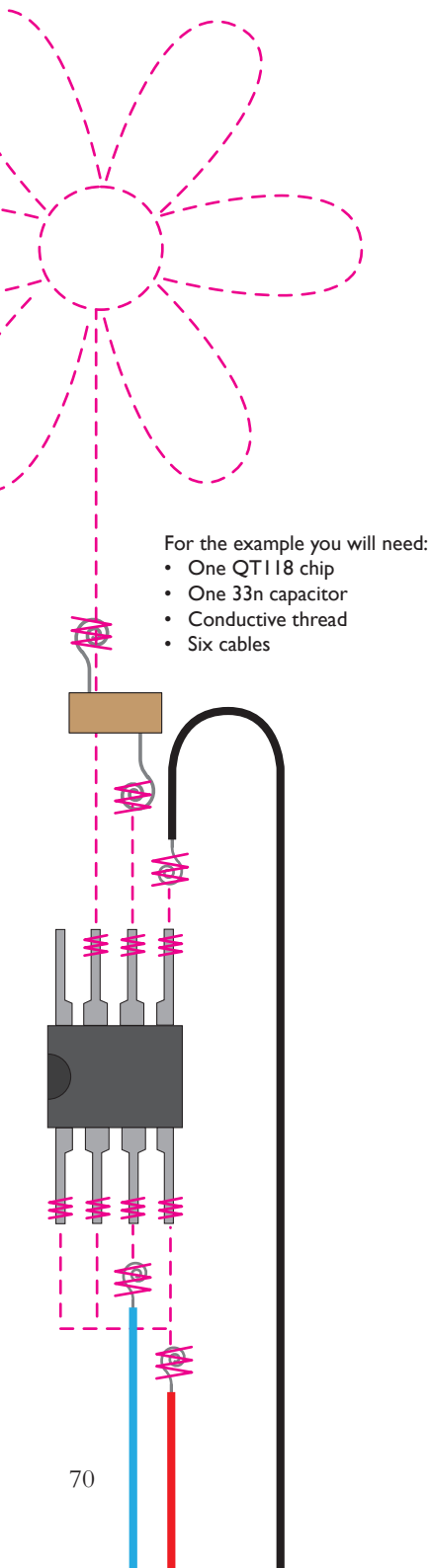
Part 4: Touch sensitive embroidery

In this example we are going to make an embroidery touch sensitive which will make the embroidery act like a button.

To make embroideries we need to use a extra chip. The QT118 chip is a touch sensor that is self-contained digital IC and capable of detecting near-proximity or touch. There different types of QT chips and the QT118 we are going to use can only handle one touch sensitive input. The QT chip works like a on off switch and you can extend the touch sensitivity from the chip with any conductive material like a wire or conductive thread. The following illustration shows the legs of the QT118 chip and where they need to be connected to:

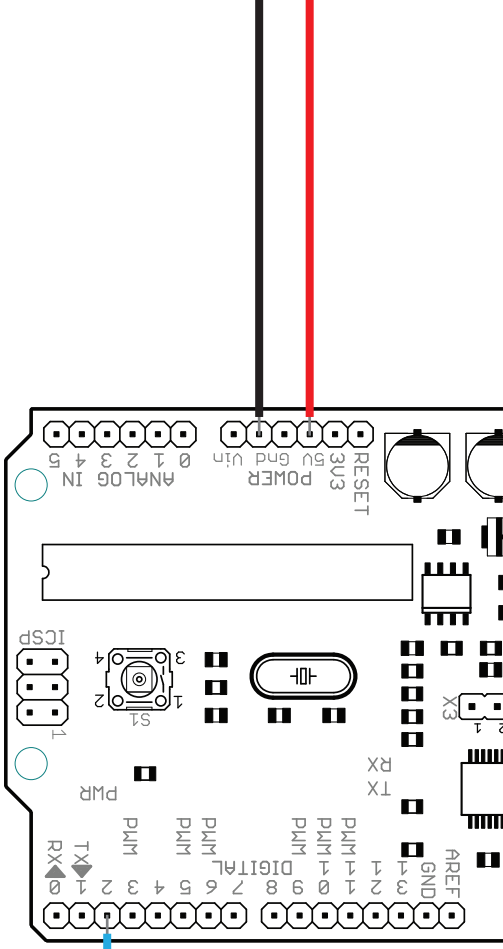
To make an embroidery touch sensitive we need to add a capacitor. The capacitor will even out the signal coming from the QT chip into our embroidery to be able to generate a reference value. The QT chip will use this reference value to compare the signal when to embroidery is touched to when it's not. The following illustration shows how to connect the capacitor to the chip and where you can start with your embroidery:

All parts that you want to be touch sensitive have to be connected to the thread that connects to the pin of the chip. Connect the following wires to the chip.



Once all connections to the chip has been made we can start writing our program. The QT118 will send a signal of 5V when it is touched. We can read this signal on both the digital and analog but if you have available digital pins it doesn't make sense to use analog pins since the signal will only be read as either 0 (the chip is not touched) or 1023 (when the chip is touched). The following program is a simple sketch on how to light up the on board LED when the embroidery is touched:

```
int ledPin = 13;
/* on board LED on the Arduino */

int touchChip = 2;
/* connect output signal from touch chip to pin 2 */

void setup(){
   pinMode(ledPin ,OUPUT) ;
   /* declare ledPin as OUTPUT */

   pinMode(touchChip,INPUT) ;
   /* declare touchChip as INPUT */

}

void setup(){
   if (digitalRead(touchChip) == HIGH){
   /* check if the embroidery is touched */

     digitalWrite(ledPin,HIGH )
     /* if it is, light up the LED */

   }else{
     digitalWrite(ledPin,LOW);
     /* if not, turn the LED off */

   }
}
```



Once the code is done we can upload it to the board. Then we connect the embroidery with the chip as in the illustration.

The embroidery will be sensitive from both side so if you want to be able to wear a touch sensitive embroidery you need to put some protective fabric on the inside of you garment. You may need to experiment with some fabrics to find one that is thick enough for you embroidery and make sure that the fabric you are using isn't conductive.

Part three: Coding

How to write programs for the Arduino.

Chapter 9: Writing Programs

An Arduino program is made of code written in a coding language called C. When writing code it's important to keep in mind that the Arduino doesn't function as a human being, it doesn't reason. If someone pronounces a word incorrectly you will probably still under-stand the meaning - the Arduino however will not. If you misspell a command while writing code in the Arduino IDE, it won't understand what you are trying to do.

Another thing to keep in mind is that the Arduino is logical but not rational. It does not know what you want to do, it only does what you tell it to do.

Basic structure

When we are writing code in the Arduino IDE we use a basic three step structure.

- variable declaration
- the setup()
- the loop()

The setup() and the loop() are essential for writing a functioning program. As you become more confident in writing code for the Arduino you will realize that it isn't mandatory to write code using these steps. However it is a useful way of structuring since it makes the program easier to overview which helps when you're looking for errors in the code. The following is an example of a simple program that turns the on board LED of the Arduino on and off:

```
int ledPin = 13
void setup(){
   digitalWrite(ledPin,OUTPUT);
}

void loop(){
   digitalWrite(ledPin,HIGH);
   delay(1000);
   digitalWrite(ledPin,LOW);
   delay(1000);
}
```

Variables

Note:
It's always good to put your variables in the beginning of the code.

A variable is like a container for something else. Let's say you want to read a sensor of some kind. This sensor will give you a value, most certainly in the form of number. If you want to use this value in more the one part of your program you can store this value in a variable to save time. Before we can store the value we need to declare the variable, this means we tell the program that the variable exists and what kind of variable it is.

We have to name the variable. The name could be anything but it is a good idea to give your variables names that are logical. When you overview your program it will be easier to determine what kind of value is stored in a variable named "tempSensor". It might be harder to remember what is hidden in variables with names like "banana", "peter" or "supercalifragilisticexpialidocious". More on this is explained in the Variable types and Declaration section on page ?.

Void setup

The first thing the Arduino does when it starts is to look for the void setup(). This is one of the essential steps of a functional program. The void setup() is the part that initializes the mode of different parts of the Arduino for instance the mode of your pin or to setup communication speed. The void setup() only runs once upon startup and will not run again until the Arduino is powered off and back on again.

The following is an example of a void setup that sets the mode of a pin as an output.

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

Void loop

The void loop() is the second essential step to a functional program. This is where the actions of your program takes place. As the name

suggests this part runs over and over again. In an Arduino program all the code is executed line by line. After the Arduino leaves the void setup() upon startup it looks for the void loop() and enters it. It then starts to do everything that is in your code from the start to bottom of the void loop(). When it reaches the bottom of the void loop() it simply starts over again which allows the program to change. The following is the void loop() from the example at the start of the chapter:

```
void loop(){
   digitalWrite(ledPin,HIGH);
   delay(1000);
   digitalWrite(ledPin,LOW);
   delay(1000);
}
```

This void loop() contains code that will turn the pin with the same number as the variable named "ledPin" on. The next line in the code makes a delay and then turns the same pin off and then makes a new delay. If we were to run this code in an Arduino with a LED connected to the pin with the same number as the variable named "ledPin" it would blink the LED on and off with one second delay until we turn the power off.

Brackets

Brackets are used to define the beginning and ending of certain parts of code. There are two types of brackets used when writing code for the Arduino. The first ones are the left and right parenthesis ( ) and are normally just called brackets. These brackets are used when we are writing functions inside our programs. They are used as a space for sending a variable somewhere else within your program. It is also possible to have a function with the brackets empty but it is still necessary to put them after your functions name or the Arduino will give you a compiling error. An example of functions that use empty brackets are the void setup() and void loop(). To learn more about functions read under Functions chapter on page ?.

The second type of brackets are the braces or curly brackets {}. These are used to show the start and end of a function. Without these brackets the Arduino will not be able to know where the function begins and ends and what to consider the next piece of the code. One common place for these brackets is the void setup() function.

```
void setup(){
   //The code in the function goes in here.
}
```

Semicolon

Semicolons are one of the most important parts of writing code for the Arduino and one of the most easy to forget. They are used to separate the different lines of code in your program and tells the Arduino where your command ends. The following example is how you declare a variable with the proper use of semicolons:

```
int myNumber = 15;
```

The semicolon ends the command and we have created an Integer variable with the name "myNumber" and this variable will have the value 15. If you forget a semicolon in your code the Arduino IDE will let you know so by giving you a compiling error when you try to compile the code and the IDE will highlight the line of code with the missing semicolon.

Commenting code

Sometimes it can be useful to put notes or write comments inside your code for yourself or for someone else. If you write text inside your program the Arduino will think its code and will try to execute what is written. If what is written is something the Arduino doesn't understand it will give you an error. There are two ways you can write messages in your code and hide it from the Arduino. The first one is to use double slash // in front of any message. This will hide the message from the Arduino but still leave it visible for your eyes. The following is an example of a message hidden inside a void loop():

```
void loop(){
   digitalWrite(ledPin,HIGH); // turns the led on
   delay(1000);               // wait for some time
   digitalWrite(ledPin,LOW);  // turns the led off
   delay(1000);               // wait a bit more
}
```

Note:
// only works for messages and notes no longer then one line.

If you want to hide messages longer then one line you have to use /* and */. To mark the beginning of a message to be hidden you use /* and to mark the end of the message you use the */. This will hide the entire message. The following is a example of how to hide a block of text inside a void loop():

```
void loop(){
/* this code will first turn a led on
then it will wait for some time
after that it will turn the led off
and then wait again. */

digitalWrite(ledPin,HIGH);
delay(1000);
digitalWrite(ledPin,LOW);
delay(1000);
}
```

Variables types and declarations

Giving a variable a value is also referred to as declaring a variable. Declaring a variable means that you give a variable both a type, name and a value.

```
int myNumber = 14;
```

In the example above the "int" is the type, the "myNumber" is the name and 14 is the value. Note that you always have to give a variable a value when you declare it. Say that I want to save a value from a sensor in my program but I cant read the value from the sensor when we declare at start of our program outside the void loop();. Then you just give it a temporary value of 0 when you declare your variable at the start of the code like the following example:

```
int mySensor = 0;
```

There are two possible ways of declaring a variable. If we declare them at the start of our program before the void setup() they will be what is called a global variable. A global variable can be accessed

by every part of your program. The opposite is called a local variable and that is a variable that can only be used inside the function where it is declared. The following example shows a variable named ledPin that is global:

```
int ledPin = 13;
void setup(){
   digitalWrite(ledPin,OUTPUT);
}

void loop(){
   digitalWrite(ledPin,HIGH);
   delay(1000);
   digitalWrite(ledPin,LOW);
   delay(1000);
}
```

The variable ledPin in visible in the entire program and when it's used in the void loop(); it will be recognized as integer variable with the value 13. If we however wrote the same program as follows:

```
void setup(){
   int ledPin = 13;
   digitalWrite(ledPin,OUTPUT);
}

void loop(){
   digitalWrite(ledPin,HIGH);
   delay(1000);
   digitalWrite(ledPin,LOW);
   delay(1000);
}
```

The program would give you an error, tell you it cant find the variable ledPin that you are trying to use in the void loop(); since it is declared, and therefore hidden inside, the void setup();.
In some programs it can be useful to use local variables but in most cases it's best to leave them as global variables and therefore declaring them before the void setup();.

Once you have a variable with a value you can reassign a new value to your variable but note that this will erase the preexisting value contained in your variable. Lets say we have a variable called myNumber and we have declared it as following:

```
int myNumber = 14;
```

If you later in your program want to give "myNumber" a new value you do so as following;

```
myNumber = 56;
```

This will empty the number 14 from the variable "myNumber" and replace it with the number 56. When we reassigned a new value to our variable we did not add an "int" before this line of code, like we did when we declared the variable. This is because you only have to declare the type of your variable once in your program. It is possible to change the value of a variable but it is not possible to change the type of your variable.

Types

So far we have been talking about "int" which is short for Integer the most common type of variable when writing programs for the Arduino. The common variables are:

• Int

Integers are used for storing data that is a number without any decimals points and they store a 16-bit value with the range of 32767 to – 32767. This means an Integer variable takes 16bits of the Arduinos memory and we have the possibility to use any number between 32767 and – 32767.

```
int myNumber = 1234;
```

• Long

In most cases the length of an Integer will work but in some cases we need to be able to store variables that are longer than the maximum size of an Integer and then we use long. A long is the extended data type for integers without decimal points and stores a 32bit value with the range of 2147483647 to – 2147483647. This means a long variable takes 32bits of the Arduinos memory and we have the possibility to use any number between 2147483647 and – 2147483647.

Note:
Using longs can fill up the memory of the Arduino so only use them when needed.

```
long myBigNumber = 90000;
```

• Byte

To save memory space in the Arduino it can be useful to store variables as bytes. A byte is a 8bit numerical value without decimal point with the range of 0 to 255. This means a byte variable takes 8 bits of the Arduinos memory and we have the possibility to use any number between 0 and 255.

```
BYTE mySmallNumber = 150;
```

• Float

The only data type that can save numbers with decimal points is the float. A float has a greater resolution than Integers and they are stored as a 32bit value with the range of 3.4028235E+38 to – 3.4028235E+38. This means that you can save a number with decimal point but only within the range of 3.4028235E+38 to – 3.4028235E+38. Declaring variables as float takes a lot of space in the Arduino. Using float is also much slower than using Integers since the Arduino needs more time to do calculations with float.

```
float mydecimalNumber = 2.33;
```

• Arrays

Sometimes it can be useful to store a collection of values and then we need to use an array. All values stored in a array will be stored with an index number and you collect a certain value by referring to the index number. Arrays need to be declared in the same way as we declare variables with a type, name and the collection of values. The following example shows how to declare an integer array with six different values:

```
int myArray[] = {1, 2, 3, 4, 5, 6};
```

Note that arrays start counting from 0. This means that the first position in an array is 0. In the example above the number 1 is store in the first position in the array and if we were to call for this value we would have to do it as:

```
myNumber = myArray[0];
```

This would save the value of the first position of the array which is 1, in our variable myNumber. In the opposite way we can store values in the array by referring to a position in the array:

```
myArray[0] = 23;
```

This would save the number 23 on position 0 in the array.
If you know you will be using a lot of numbers and want to store them in an array but you don't know what values there are going to be you can still declare an array with a certain amount of empty places like the following:

```
int myArray[5];
```

This will create an array with 6 empty positions since an array start counting from 0.

Doing math

As the Arduino is a small computer it can do mathematical operations. The Arduino can handle the most common arithmetic operators like addition, subtraction, multiplication and division.

```
myValue = 1 + 1;
/* this will store the number 2 in myValue: */

myValue = 4 - 2;
/* this will store the number 2 in myValue */

myValue = 3 * 4;
/* this will store the number 12 in myValue */

myValue = 6 / 2;
/* this will store the number 3 in myValue */
```

If you are using Integers for doing mathematical operation it can't handle decimal point as the float are the only type of variable that can do this. In other words if we where to divide 10 by 6 it would give us 1 as the result. Doing to large mathematical operations could also result in a overflow of the memory since all types of variables have a specific maximum size. Also be aware that calculating with large numbers will slow the Arduino down.

You can also do what is called compound assignments which is when you do mathematical operations to variables. The following are examples of the different compound assignments you can do to variables:

```
x++
/* this will increase x by one and it's the same as
writing x = x + 1 */

x--
/* this will decrease x by one and it's the same as
writing x = x-  1 */

x += y
/* this will increase x by y and it's the same as
writing x = x + y */

x -= y
/* this will decrease x by y and it's the same as
writing x = x - y */

x *= y
/* this will multiply x by y and it's the same as
writing x = x * y */

x /= y
/* this will divide x by y and it's the same as writ-
ing x = x / y */
```

• Map

Lets say you have a sensor that only gives a range of values from 50 to 200 and what you need is a range from 0 to 500. Then the map function could come in handy. The map function re-maps a rang of values to another range of values:

```
myVariable = map(mySensor,50,200,0,500);
//sensMin, sensMax, desiredMin, desiredMax
```

In the above example we are using the map function to store a value in myVariable. The value comes from mySensor and the 50 and 200 marks the min and max value from our sensor. The 0 and 500 are the desired range we want in stead. The map function will automatically re-map the values in the rang of 50 to 200 that we get from mySensor to a value in the range of 0 to 500. Note that if you want to use the map function you have to know the range of the thing you want to re-map. To find out more on how to read values have a look at on page ? In the ? section.

• Random(max)

The random command will return a random value in the range from 0 to the max value you put inside the parenthesis. To be able to use this value you have to save it in a variable:

```
myVariable = random(5);
```

This will save a random number in myVariable ranging from 0 to 4 or you can use the random command directly while making comparisons

```
if (3 = random(5); ){
   doSomething;
}
```

The random command will only return a value between 0 and the max value you put in the parenthesis, it will never return the actual max number.

• Random(min,max)

If you want a random number that starts from a higher range than 0 to something else you will have to add your desired minimum value:

```
myVariable = random(200,300);
```

Note:
This example will only return a value in the range between 200 and 300. It will never return 200 or 300.

Logical comparisons

If you want to make comparisons in your program you have to use one of the possible comparison operators. We use them to compare variables to either other variables or other constants to make statements that can either be true or false.

• Equals equals ==

== is used to compare if something is equal to something else. A statement using == will only be true if something is exactly the same as something else:

```
x == y
/* x is exactly the same as y */
```

• Differ from !=

!= is used to compare if something is not equal to something else. A statement using != will only be false if something is exactly the same as something else:

```
x != y
/* x is different from y */
```

Note:
The "more than" and "less than" symbols can easily be mixed up. Always remember that the closed end points towards the smaller value.

• Less than <

< is used to compare if something is less than something else. A statement using < will only be true if something is smaller than something else:

```
x < y
/* x is smaller than y */
```

• More than >

> is used to compare if something is larger than something else. A statement using > will only be true if something is bigger than something else:

```
x > y
/* x is bigger than y */
```

• Less or eqauls <=

<= is used to compare if something is less or equal to something else. A statement using <= will only be true if something is smaller or the same as something else:

```
x <= y
/* x is smaller or equal to y */
```

• More or equals >=

>= is used to compare if something is larger or equal to something else. A statement using >= will only be true if something is bigger or the same as something else:

```
x >= y
/* x is bigger or equal to y */
```

Logical Operators

The logical operators are used when you need two or more statements in the same sentence and these can be true or false. There are three different kind of logical operators that can be used.

• And &&

This is used to determine if two or more statements are true. If not all statements are true then the sentence will be false. For something to be true using two or more statements in the same sentence, all statements need to be met but for something to be false only one of the statements need to fail its requirements:

```
x < y && y > 5
/* x is smaller than y and y is bigger than 5 */
```

• Or ll

This is used to determine if something or something else is true. If only one statement is true then the sentence will still be true:

```
x < y || y > 5
/* x is smaller than y or y is bigger than 5. Note:
both statements can be true */
```

• Not !

This is used to determine if something is not true. If the the statement is not met then the sentence will always be true:

```
!x==5
/* x is not equal to 5 */
```

Note:
All Arduinos constants are always written with capital letters while programming.

Constants

Constants are parts of the code language the Arduino uses that have predefined values. They are used to make it easier to read the code in your program.

• True and False

True and false are what is called boolean constants and they define if something is, or is not, at a logical level. Any number can be used as a boolean operator. 200 can be used as an operator and if a variable has the value and we compare it to 200 this will generate a true:

```
boolean myBoolean = true;
```

It can also be written with a number:

```
int myNbrBoolean = 1;
```

In the first case we would need to compare myBoolean to another boolean and in the second case we would need to compare myNbrBoolean to another int.

• High and Low

HIGH and LOW are used to set the state of the digital pin which only have these two states. HIGH means the same as ON or that we are turning on 5 volt on our digital pin. It's also the same as a logical 1. LOW means the same as OFF or that there is 0 volt on our digital pin. It's also the same as a logical 0:

```
digitalWrite(ledPin,HIGH);
```

This can also be written with numbers:

```
digitalWrite(ledPin,1);
```

• Input and Output

INPUT and OUTPUT are used when we declare the mode of our digital pin and there is only these two modes for the digital pins:

```
pinMode(12,OUTPUT);
```

If something happens and what to do

Let's say you are working on a prototype and you are measuring distance. Now when something comes with a certain range from your object you want something to happen. This is when the if-statement comes in handy.

• If

An if-statement is like a test the Arduino can do to determine if something is true or false. An if-statement looks like the following example:

```
if (myVariable>myOtherVariabel){
   doSomething;
}
```

Note:
If is always spelled with small letters while writing code. Don't forget the curly brackets to mark the start and end of the if-function.

In this example we ask the question if myVariable is bigger than myOtherVariable and if it is so then the program will jump inside the if-function and execute the code. If the statement is not true it will skip this part of code. In the above example we are comparing variables but we can compare constants as well:

```
if (buttonPin==HIGH){
   doSomething;
}
```

In this example we ask the question -if buttonPin is HIGH then you doSomething and if it's not skip what ever is inside the if-function.

Keep in mind not to forget to use == when you are making comparisons. If you use a single = you will not compare something to something else but you will declare the value of something else to something:

```
if (buttonPin=HIGH){
   doSomething;
}
```

The above examples shows the wrong way of writing an if-statement. This examples would declare buttonPin as HIGH and would not check if buttonPin is HIGH.

• If else

Now lets say you make a check using an if-statement and your condition is not true and you know what you want to do if the first thing does not happen. Then you you can connect an else-statement to your if statement:

```
if (myVariable>myOtherVariabel){
   doSomething;
} else {
   doAnotherThing;
}
```

The above example works like an "either-or" statements. Either myVariable is bigger than myOtherVariable and you doSomething or myVariable isn't bigger than myOtherVariable and you doThis. Don't forget to use new curly bracket to mark where the else part starts and ends.

You can add as many else conditions to an if-statement as you want but if you add more than one else you have to write all except the last one as else if followed by a new condition:

```
if (myVariable>myOtherVariabel){
   doSomthing;
} else if (myVariable<100){
   doAnotherThing;
} else {
   doTheLastThing;
}
```

If myVariable in the above example isn't bigger then myOtherVariable then ask if myVariable is smaller than 100 and if this isn't true either doTheLastThing.

For

For-loops are used when you want to repeat a part of code a certain amount of time. The for-loop always has three parts in the parenthesis and it is the initialization of the counter, the condition to end the for loop and the increasing of your counter:

```
for (int i=0; i<200; i++) {
   doSomething;
}
```

In this example int i=0; is the initialization of the counter for the for loop. Here we say that we want a counter with the name i and we want it to start counting from 0. When we are done with the first part of the loop we end it with a semicolon and this goes for the second part as well. The second part is i<200 which is our condition for ending the for loop and in our case this means when i becomes bigger than 200 we want to stop looping. The last part is our increasing of the counter for every time the loop starts. i++ will increase the counter by one every time the loop restarts. The first time the counter i will be 0 the next time it will be 1 and so on. When i hits 199 the condition for ending the for loop will be met and the program will exit the loop to carry on with the code written after the curly bracket that marks the end of the loop.

While

A while loop will keep on looping until the condition inside its parenthesis becomes false. If a while loop is used there has to be something within the while loop that can make some increasing or change or the while loop will never end:

```
while (myVariable>100) {
   doSomething;
}
```

This example tests if myVariable is smaller than 100. If it is then it will start looping. But there is nothing inside the loop that changes myVariable since the first time we checked so this loop would continue to doSomething forever.

In the following example we have added a reading from a sensor:

```
while (myVariable>100) {
   doSomething;
   myVariable = readSensor;
}
```

Now every time it makes a loop it will first doSomething and then save the value from readSensor in myVariable. If the value save in myVariable is above 100 the while loop will stop and carry on with the code written after the curly bracket that marks the end of the loop. Note that if you are using sensors to break a while loop make sure the sensor will give you a value that is above the threshold of your condition in the while loop.

Note:
If you are using a pin as an INPUT make sure the signal is never more then 5v or you will burn your Arduino board.

The Digital Pins

These pins are the 0 to 13 pins on your Arduino and they are called digital because they can only handle information in 0 or 1. If you want to use a digital pin for something the first thing you have to do is to set the mode of the pin. This is always done in the void setup().

The command for setting the mode of a pin is pinMode() and is used as follows:

```
pinMode(pin,OUTPUT);
```

"pin" in this example is a variable with the value that corresponds to the number next to the physical pin on your Arduino board. OUTPUT is the the desired mode of your pin. The digital pins has only two modes, OUTPUT and INPUT. If you declare a pin as an OUTPUT you can only use it to either turn 5V on the pin or turn off to 0V. If you declare your pin as an INPUT you can only use it to read if there is 5V coming in to the pin or if it 0V on the pin:

```
digitalWrite(pin,value)
```

To turn your digital pin on and off you need to use the digitalWrite() command. In the parenthesis you always need to state what pin you want to use and what value you want to give it:

```
digitalWrite(pin,HIGH);
```

This will turn the pin to HIGH which will enable it to send 5V. If you write LOW instead of HIGH we will turn the pin back to 0V. Note that until you turn your digital pin to HIGH the default value is LOW after setting the mode of the pin. If you have a look at your Arduino board you'll also see that digital pin 0 and 1 are marked as RX and TX. These two port are reserved for serial communication and should not be used since it will put the Arduino in standby mode until a signal is received.

•DigitalRead(pin)

The digitalRead() command reads the status of a pin and returns either a HIGH if there is 5V coming into the pin or LOW if there is 0V on the pin:

```
digitalRead(pin);
```

To be able to use this status for something you will need to save it in a variable:

```
myVariable = digitalRead(pin);
```

If you want to make a comparison you can write the command directly in a statement:

```
if (digitalRead(pin)==LOW){
   doSomething;
}
```

Analog pins

The analog pins work differently from the digital. We mentioned that the digital pins only handles information in 1 or 0 which is the same as HIGH and LOW or 0V and 5V. However, in the real world we don't measure everything in zeros and ones so the Arduino has six special pins called analog pins that makes a mathematical calculation on the voltage from a range of 0 to 1023. Analog pins don't have to be mode declared since they're only used as inputs.

Note:
If you are making comparisons with analog readings the value from a analog pin can never be above 1023.

• Analog read (pin)

To read the value on an analog pin you have to use the analogRead() command and refer to what pin you want to read:

```
analogRead(pin);
```

As with the digital pin you have to save this value in a variable to be able to use it

```
myVariable = analogRead(pin);
```

You can use the command directly to make comparisons

```
if (analogRead(pin)>500){
   doSomething;
}
```

• Analog write (pin,value)

The digital pins can only be HIGH and LOW which means the same as either there is 5V on the digital pins or 0V. But digital pin 3, 5, 6, 9, 10, and 11 has a special function called analogWrite(). With this function is possible to send a pseudo-analog value to these special digital pins. This is also called pulse with modulation (PWM):

```
analogWrite(pin,value);
```

The value in this example can be anything from 0 to 255. If you where to write 0 this would mean the same as setting the pin to LOW and 255 is the same as HIGH. But with the analogWrite() you get 255 steps in between HIGH and LOW so for example:

```
analogWrite(pin,127);
```

This would be the same as sending out 2.5V on your digital pin. Compared to the digitalWrite() which shift from 0V to 5V in an instance, with the analogWrite() you can make a slower transition from 0V to 5V. Note that the analogWrite() only works on the digital pins marked PWM (3, 5, 6, 9, 10, and 11) and not on the analog pins.

Using time

The Arduino is a small but powerful computer and can make 1000000 calculations per second. Wen you are making prototypes you may not want to execute at this lightning speed. Then you're going to have to tell the Arduino to slow down every now and then.

• Delay

The delay command is used to make a pause in your program. This command counts in milliseconds and you put your desired pause time inside the parenthesis like the following example:

```
delay(1000);
```

This delay will make a pause in your program with one second.

• Count milliseconds

This command will return how many milliseconds that has passed since the Arduino started the current running program. To be able to use this value you have to save it in a variable:

Note:
The value of millis() will reset itself to 0 after about 9 hours.

```
myVariable =  millis();
```

You can use it directly to make time comparisons:

```
if (myAlarmTime == millis();){
   ringAlarm;
}
```

Communication with other devices

To be able to communicate with other electronic devices you will have to enable the communication ports on your Arduino. The Arduino can both communicate with your computer or with other electronic devices that use the serial communication protocol.
The digital pin 0 and 1 on the Arduino is reserved for serial communication with other devices and you should avoid using these two port for anything else since this can interfere with the Arduino trying to run you program.

• Serial begin

To enable the Arduino for communication you use the command Serial.begin(). This command is always used in the void setup() and nowhere else. Inside the parenthesis you have to put your desired communication in bits per second which is also known as baud and the available speeds are 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200:

```
void setup() {
   Serial.begin(9600);
}
```

This will open the serial port and set the communication rate to 9600 baud.

• Serial println

This command will print whatever you put inside the parenthesis. To print integers you just have to type them inside the parenthesis:

```
Serial.println(12345);
```

However printing characters and string of character needs to be quoted. If you want to send a single character you use single quotation marks ‘ ‘:

```
Serial.println('C');
```

The above code line will send the character C over the serial port. If you want to print a string of characters like a message you have to use normal quotation marks “ “:

```
Serial.println("Hello from Arduino");
```

• Serial println

The Serial.print() works the same as the Serial.prinln with the exception that Serial.println() is followed by a automatic carriage return and a line feed. If something is sent over the serial port with Serial.println(1) the message would be show like the following in your monitor:

```
1
1
1
1
1
```

The line feed added by the Serial.println() is the same as pushing enter after every time you send something so it ends up on a new line. If we would send Serial.print(1) over the serial port it would be received like the following:

```
111111111111111111111111111111111111
```

Note that if you are communicating with other electronic devices make sure that you print your data in the right formation. Sometimes the carriage return and line feed added by Serial.println() can interfere with the communication. Information about the right communication protocol for individual electronic devices can be found in the data sheet for that device.

• Special case printing

Sometimes you will need to send information in different formats and then you have to add this format to your print:

```
Serial.println(message,format);
```

The available formats are decimals, hexadecimal, octal, binary and bytes and are used as the following examples show:

Note:
A small b, which is used in these examples has the ASCII-code "79".

```
Serial.print(b,DEC);
/* This will print 79 as a ASCII-encoded decimal which
is "79" */

Serial.print(b,HEX);
/* This will print 79 as a ASCII-encoded hexadecimal
which is "4F" */

Serial.print(b,OCT);
/* This will print 79 as a ASCII-encoded octal which
is "117" */

Serial.print(b,BIN);
/* This will print 79 as a ASCII-encoded binary code
which is "1001111" */

Serial.print(b,BYTE);
/* This will print 79 as a ASCII-encoded bytes which
is "O" */
```

The ASCII (American Standard Code for Information Interchange) encoding is a standard way of encoding text numerically. Even if you write numbers in your code it's not actually numbers you are writing, it's the text representation of numbers as same as "one" is the word representation of 1.

Epilogue

We have now reached the end of this book and hopefully the beginning of your own future fashionable and wearable prototypes. We hope that this book has given you a basic insight into the world of wearable computing and that the examples in the book will work as a foundation for your own progress within the field.

However a straight transition from this book to prototyping on your own might be hard. But fortunately you are not alone in the world of electronic prototyping. As we said at the beginning of this book the Arduino is not only hardware and software. It is also a huge community of people, both professionals and hobbyists with a interest in physical prototyping.

A large portion of these people come together at the Arduino playground (www.arduino.cc/playground) where they both help each other out with all kinds of problems or just to share ideas and show off new constructions.

Instructables (www.instructables.com) is another great online community for DIY (do it yourself) people. The users not only create there own elecrtronic DIY guides but also "how to do" on almost any subject you can think of. Instructables is one of the best and biggest sources of inspiration and we can sincerely recommend it to any one interested in fashionable and wearable computing.

Another good contender to Instructables is the Make blog and community (www.makezine.com) which also has a large group of skilled people tied to it and is a great recourse to keep you updated on what is happening in the field of prototyping.

If you are looking for communities strictly devoted to the field of fashion and technology then the Fashioning tech blog (www.fashioningtech.com) is a nice place to start. They also have a growing community of users with a forum where they share ideas and help each other out.

Even if a book might be a good start and introduction, it will never beat the internet in its extensive and up to date information. But it's our hope that this book will give you the knowledge needed to be able to make use of all the information you can find online.

There's more good online resources and to include them in this book would probably require a few extra chapter. The list above are all good starting points for your own online information library.

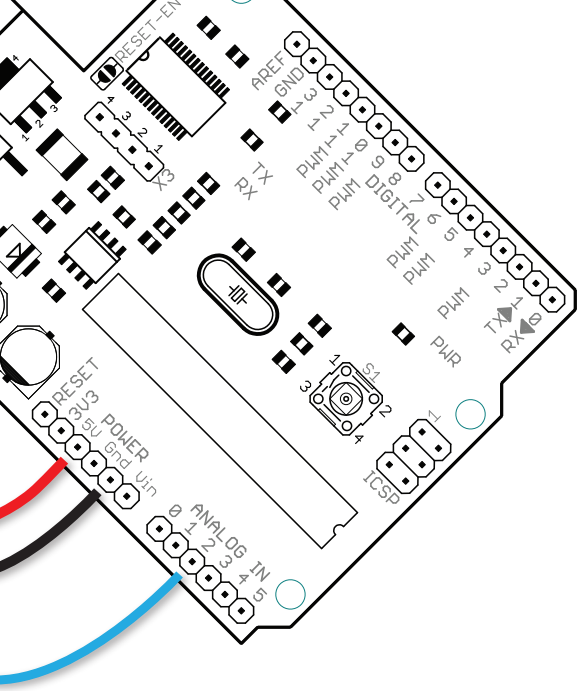
The most important thing you should keep in mind while working with electronic prototyping is to have fun and don't be scared to try things out. But still beware that you are working with electricity so a simple principle like, if you are not sure about something make proper research until you are confident before plugging stuff together, it might save you money.

Be safe, not stupid and happy prototyping.

Acknowledgments

This book has been inspired by Brian Evans booklet
Arduino programming notebook, First ed, 2007
and
Getting Started with Arduino, Thired ed, 2008by Massimo Banzi
Thanks also goes to
the physical prototyping laboratory at Malmö University, Faculty of Art, Culture and communication (k3)
the guys at the critical design studio 1scale1, Malmö
and special thanks to the Arduino team for there work with the Arduino, Tom Igo, Dave Mellis, David Cuartielles, Massimo Banzi and Gianluca Martino

Index


RESET-EN
AREF
GND
PWM~13
~12
PWM~11
~10
~9
8
PWM DIGITAL
7
6
PWM~5
4
PWM~3
2
TX→1
RX←0
TX
RX
RESET
3V3
5V
POWER
GND
GND
Vin
ANALOG IN
0
1
2
3
4
5
PWR
ICSP