

# ArduDIY

---

First Edition

Prabodh Sakhardande  
*Sugar Plum Labs*  
prabodh.sakhardande@gmail.com



## Contents

Introduction.....	1
How and Where to Use This Board .....	3
Getting Started .....	5
What You Will Need: .....	5
Contents .....	8
How to go about .....	9
Arduino .....	10
The Microcontroller: .....	13
What makes a Microcontroller Arduino! .....	16
ArduDIY.....	17
Converter: .....	20
ArduDIY Board:.....	22
Components .....	29
Circuit Working .....	41
Soldering.....	45
Testing .....	57
Visual Checking:.....	57
Testing Using the DMM:.....	58
Turn the board on: .....	58
Plugging in the Atmega328: .....	59
Software & Programming.....	61
Let's start Programming!!! .....	69
Arduino Basics .....	70
Troubleshooting .....	75
Rule No. 1: .....	75

VCC-GND show a short circuit:.....	75
Power LED (red) does not glow when you switch on the board:	76
CP2102 Driver Problems: .....	76
ATmega328 not getting programmed:.....	77
Peripheral of ATmega328 not working: .....	78
Code compilation Errors:.....	78
The Arduino Bootloader .....	80
Making Your Own Arduino .....	82
A few initial notes: .....	82
Step 1:.....	83
Step 2:.....	83
Step 3:.....	83
Step 4:.....	83
Step 5:.....	83
Future Work & Some DIY Projects.....	86

# Introduction

---

Ok so the fact that you are reading this means that you must be interested in embedded technology. Well at least remotely interested (hopefully). Or you just must be curious what all this is about. Whatever your reason be, thank you for reading and I urge you to go on till the end. Just for the reason that there is a chance at the end of this, you might learn something new. If there is anything I have learnt in all these years it is we must always learn from everything around us. In the words of Steve Jobs “Stay Hungry Stay Foolish”.

Throughout this the only thing I expect from you is the desire to learn. Please do keep in mind that this is not some local author textbook or some big reference book. I shall be keeping to the point and hence you must read through the whole thing. Every line and word.

At the end of this, I promise you will have learnt something. For some it may not be technical knowledge. You shall learn the correct approach to things. This is something that is rarely taught in our education system. Though this is the one most important skillset we need in the future.

Knowledge is not something that should be kept locked up and sold. Most of what I have learnt throughout the years is thanks to people who were kind enough to put their work online free of cost. Many workshops only give a feel of doing stuff. All the things are provided by them. Students don't actually get in depth experience and what they learn is only superficial. While our current engineering education system does not even do that. In India, engineering is only based on how much you can learn by heart. It's like “raat sakte ho, chalo paas ho

gaye". Even if one tries to actually learn there is basically no point as everyone knows questions are going to come from the local author book. "Yaar marks toh future mei bhaut important hai na..." The whole thing is very frustrating to those who actually want to learn and do something.

The second reason for making this is that the grasping speeds of everyone is different. I could never pay attention in lectures as my attention span is a whole complete of ten minutes. So I believe freedom is necessary. Freedom to go as one wishes, at the pace one wishes. In the end what matters is how much you learn.

The main aim of this project, for me is to reduce the prototyping time. Every year there are so many students with amazing project ideas. But our current system is such that information is not readily available and our college syllabus does not teach anything practical at all. So when students start with projects, a lot of time is wasted in doing things from scratch. When the hardware finally is ready the semester ends and the innovative ideas are lost. In a country like India with so much talent this should not happen. Ideas being lost or forgotten is a huge mistake on our part.

What I am trying to do through this kit, is reduce the time required to get the hardware ready. This will enable us to directly work on our ideas in the Application Domain. Keep in mind, this is just the beginning. You are about to unlock the path to a very interesting domain of engineering. The possibilities of what you can do are limitless. Finally, the joy we get when we make something and it works is amazing!

# Foreword

---

Some Rights Reserved.

You are free to change this hardware, duplicate it, and make your own for non-commercial purposes.

Every kit has been tested. The only active component ATmega 328 has been thoroughly tested by programming and bootloading each IC before packing. The passive components cannot be tested individually before packing. If there are any errors in those we sincerely apologise. Due to the small cost replacing the passive components won't be an issue.

In case of errors while implementing this kit we urge you to first try things out yourself. Due to the small scale nature and direct involvement in making each kit, it is highly unlikely that a defective piece gets sold.

You can always call up the team at Sugar Plum Labs for queries advice or even if you get a random urge to discuss the string-loop theory, our contact details are given in the kit.

Any sort of suggestions, scope for improvement are welcome.

If you are interested in working with us, then we welcome you with open arms.

Special thanks to Sparkfun, Adafruit for pioneering the open hardware movement and obviously Arduino for simplifying our life.

# How and Where to Use This Board

---

This board has been designed for rapid prototyping, tinkering and experimenting. The nature of this board is such that after you finish making it and testing it for the first time, it has a very high reliability. It has been specifically made such that you can make any change to it physically as you wish (for example adding extra pins, headers, specific connections, etc.).

All components on this boards are in the DIP (Dual in Line) package so that they are easily solderable and also desolderable. Hence now even if some component gets damaged while experimenting you can change only that specific component and continue as good as new. This is a great advantage as compared to the Arduino Uno we get in the market which if damaged cannot be easily repaired.

As this board has a very high reliability it can be used as a starting point in all your projects.

When we start with new projects we need some reliable generalised hardware upon which we base the further project. Then using this generalised hardware we test all our project ideas rapidly. As it is a tested and reliable hardware time is not wasted in debugging of the hardware. Now once all the parts of the project are tested we can replace the generalised hardware with a specific one. What this does is it provides us with an interface we want, it eliminates the need of jumpers, connectors, breadboards and thus greatly reduces size and complexity.

So now whenever some nice idea comes in your head, just take out this board and start making!



# Getting Started

---

## What You Will Need:

To get through with this you will need some time, dedication and a certain amount of patience.

Along with that you will need 10% luck, 20% skill, 15% concentrated power of will, 5% pleasure, 50% pain and..... (For those who did not get the reference please ignore... sorry)

So on a serious note you will be needing the following things. If you are not sure about all this and want to just get an idea, you can borrow most of the needed stuff, colleges also have these.

If you are even slightly serious about making stuff and DIY electronics then I strongly suggest you get yourself this equipment.

- Digital Multimeter

There is a wide variety available ranging from 100 rupee ones to 5000 rupee industrial standard. For most part the 100 rupee one is sufficient. But if you are considering doing more projects like this I would advise you to invest a bit more and get something a bit better around 300 – 500 rupees.

This is the one we usually get at a hundred bucks.



Function Knob.

You have options like DC Voltmeter, AC Voltmeter and Ammeter

Insert probes here.  
Check which function is being used and choose hole accordingly.

Always disconnect the probes before changing the mode of the DMM. Suppose you are measuring voltage and suddenly change the mode to Ammeter, your system will get short circuited and damaged!

- Soldering Gun

The soldering gun we most easily get in these areas is the yellow coloured Solderon soldering iron (25 Watt). This is completely sufficient for all DIY projects and is even capable of SMD soldering.

Most soldering guns come with a flat ringed metal tip. For SMD work you can use the pointed tip. There is a different type of tip available known as a coated tip. It is about 3 times the cost of a normal tip but gives superior performance and lasts a lot longer. If you are going to do a lot of soldering work give it a try.



- Solder Metal and Flux

Solder metal for obvious purposes. The flux is optional but it makes soldering a whole lot easier.



- Wire Stripper and similar tools.  
This again is optional depending upon your project.

Other than the above stuff all that you need for the ArduDIY is provided in this kit.

# Contents

---

Component Name	Value	Quantity
Resistor	4k7	3
Ceramic Capacitor	0.1uF	6
	22pF	2
Electrolytic Capacitor	47uF	1
	100uF	1
Diode	1N4007	2
LED	3mm	2
Crystal	16MHz	1
7805	-	1
Reset Switch	-	1
Slider Switch	-	1
IC Base	28 Pin	1
Battery Connector	-	1
Single Strand Wire	-	1
Bergstrip	-	
PCB Mount Relimate	6 pin	1
	2 pin	1
Wired Relimate Connector	6 pin	1
Bootloaded ATmega328	P-PU	1
Box Header	10 Pin	1
ArduDIY PCB	-	1
Converter Board	CP2102	1

# How to go about

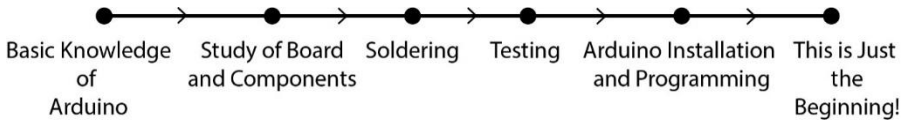
---

So, the right approach is to read this and side by side and practically perform.

What we have to do is widely distributed in Soldering, testing and Programming.

This book has been made in such a manner that the reader can follow the link and progress.

So just follow this book and you should be good.



Consider about half an hour for the basic knowledge and study of board part and one to two hours for soldering and another hour for the software and testing part. You could do it all at once or in breaks as you feel convenient.

# Arduino

---

Arduino is an open source prototyping platform. It consists of both hardware as well as software. This means that Arduino has defined the hardware (microcontrollers) which can be used and also provides us with the Arduino IDE where we write the code. Now this is a huge advantage as the whole system comes under one roof. Earlier we had to use one brand of microcontrollers, a second software to write the code, a separate compiler and a fourth software to upload the code. Arduino does all these things for us. That is one of the reasons it is so popular.

The Arduino hardware and software was designed for artists, designers, hobbyists, hackers, newbies, and anyone interested in creating interactive objects or environments. Arduino can interact with buttons, LEDs, motors, speakers, GPS units, cameras, the internet, and even your smart-phone or your TV!

Now open source means that the code developed is free for everyone to view and edit. The outcome of this is that thousands of talented coders are working on Arduino as we speak. The whole system is constantly developing and everyone gets the benefits. The whole community is friendly and all questions are answered on the forums. Now as this is open source a lot of libraries are readily available. A library is a set of codes specific to some application. Suppose you want to run a servo motor, to write the actual code can be quite cumbersome so we use a library developed for servo motors. This library contains all functions required in one package.

Arduino programming is done in a simplified version of C++, one just needs to know the basics and then can start programming right away.

What Arduino does is it greatly simplifies the whole code writing process. This is especially useful for people like us, who are more interested in implementing things rather than dealing with the whole computer part of it.

How does it simplify the code? Let me give you an example.

Suppose we want to run a motor with variable rpm. For this we need the PWM signal from the microcontroller. Now in traditional approach the code would be:

#### Traditional Code

```
int main()
{
    TCCR0=0b01101001;
    DDRB=0xFF;
    TCNT0=0;
    OCR0=0;
    int i,flag=0;
    delay(1);

    while(1)
    {
        OCR0=50;
    }
}
```

#### Arduino

```
void setup() {
    pinMode(9, OUTPUT);
}

void loop() {
    analogWrite(9, 50);
}
```

YES!! All that was reduced to only two lines!  
And this is just the most basic example, you will realise how great Arduino is once you start using it.

Now you must be thinking “How is this possible!? Is it really this easy?” Well yes. It is so because hundreds of people have already put in effort. It’s not like the code required has magically shrunk. There is a code behind what we write and that is huge. See what the people at Arduino did is that they created standard libraries and functions for everyone to use. So the extra lines you see in the first code are actually there but they are being used by us through the functions defined by the creators of Arduino.

## Just to give you a Perspective, this is the Arduino Header

```
#ifndef Arduino_h
#define Arduino_h

#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

#include <avr/pgmspace.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#include "binary.h"

#ifdef __cplusplus
extern "C" {
#endif

void yield(void);

#define HIGH 0x1
#define LOW 0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2

#define PI 3.1415926535897932384626433832795
#define HALF_PI 1.5707963267948966192313216916396
#define TWO_PI 6.283185307179586476925286766599
#define DEG_TO_RAD 0.017453292519943295769236907
#define RAD_TO_DEG 57.29577951308232087679815481
#define EULER 2.718281828459045235360287471352

#define SERIAL 0x0
#define DISPLAY 0x1

#define LSBFIRST 0
#define MSBFIRST 1

#define CHANGE 1
#define FALLING 2
#define RISING 3

#ifdef __cplusplus
#if defined(__AVR_ATtiny24__) || defined(__AVR_ATtiny45__)
#define DEFAULT 0
#define INTERNAL 1
#define INTERNAL2 2
#else
#define DEFAULT 0
#define INTERNAL 1
#define INTERNAL2 2
#define INTERNAL3 3
#endif
#else
#define DEFAULT 1
#define INTERNAL 0
#define INTERNAL2 0
#endif

// undefine stdlib's abs if encountered
#ifdef abs
#undef abs
#endif

#define min(a,b) ((a)<=(b)?(a):(b))
#define max(a,b) ((a)>=(b)?(a):(b))
#define abs(x) ((x)>0?(x):-(x))
#define constrain(val,min,max) ((val)<min?min:(val)>max?max:(val))
#define round(x) ((x)>0?long((x)+0.5):long((x)-0.5))
#define radians(deg) ((deg)*DEG_TO_RAD)
#define degrees(rad) ((rad)*RAD_TO_DEG)
#define sq(x) ((x)*(x))

#define interrupts() sei()
#define noInterrupts() cli()

#define clockCyclesToMicroseconds(c) ((c) * CPU / 1000000)
#define clockCyclesToMicroseconds(a) ((a) / clockCyclesPerMicrosecond())
#define microsecondsToClockCycles(a) ((a) * clockCyclesPerMicrosecond())

#define lowByte(w) ((uint8_t) ((w) & 0xff))
#define highByte(w) ((uint8_t) ((w) >> 8))

#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) : bitClear(value, bit))

#endif

#ifdef __cplusplus
}
#endif

// This comes from the pins_*.c file for the active board configuration.
#define analogInPinToBit(P) (P)

// On the ATmega1280, the addresses of some of the port registers are
// greater than 255, so we can't store them in uint8_t's.
extern const uint16_t PROGMEM digital_pins_to_bit_PGM[];
extern const uint16_t PROGMEM port_to_input_PGM[];
extern const uint16_t PROGMEM port_to_output_PGM[];
extern const uint8_t PROGMEM digital_pin_to_port_PGM[];
extern const uint8_t PROGMEM digital_pin_to_bit_PGM[];
extern const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[];
extern const uint8_t PROGMEM digital_pin_to_timer_PGM[];

// Get the bit location within the hardware port of the given virtual pin.
// This comes from the pins_*.c file for the active board configuration.
#define digitalPinToBit(P) (P)

// These perform slightly better as macros compared to inline functions
#define digitalPinToPort(P) (pgm_read_byte(digital_pin_to_port_PGM + P))
#define digitalPinToBitMask(P) (pgm_read_byte(digital_pin_to_bit_mask_PGM + P))
#define digitalPinToTimer(P) (pgm_read_byte(digital_pin_to_timer_PGM + P))
#define analogInPinToBit(P) (P)
#define digitalWritePinValue(P) (volatile uint8_t *) pgm_read_word(port_to_output_PGM + P)
#define pinModeRegister(P) (volatile uint8_t *) pgm_read_word(port_to_output_PGM + P)
#define pinModeRegister(P) (volatile uint8_t *) pgm_read_word(port_to_output_PGM + P)

#define NOT_A_PIN 0
#define NOT_A_PORT 0

#define NOT_AN_INTERRUPT -1

#include "pins_arduino.h"

#endif
```

If you want to actually read it, just open “arduino.h” located in the program files.



## The Microcontroller:

Microcontrollers are everywhere! From our watches to our phone, in our cars, our homes. Almost each and every electronics appliance available today has the microcontroller at its heart.

So what are these microcontrollers and why are they so popular?

You must have read the age old definition that “A microprocessor does not have RAM, ROM, I/O or memory and these have to be added externally while a microcontroller has memory, I/O, RAM and ROM on the same chip”

Let’s understand this a bit better.

Initially there were analog circuits in electronics. This was a complex field and only people with lab coats and years of experience could work in it. Then the transistor was discovered and came the era of digital circuits. Large scale fabrication began and prices were lowered. Multifunction integrated circuits were introduced and this put never before available technology in the hands of the common man (well basically hobbyists, enthusiasts, engineers, makers).

Then, someone suddenly put a question: Why should not we make a universal component? A cheap, universal integrated circuit that could be programmed and used in any field of electronics, device or wherever needed?

Ok, so when microcontrollers first came out there was a huge demand. But the problem was that everyone had different needs. One wanted to make a washing machine, another wanted home automation and a third just wanted to turn lights on and off. The manufacturers were in a fix. Up till now ICs used to be very specific. How could they fulfil everyone’s needs? So they came up with a very generic design. Multiple functions were bundled up in a single package. But to provide so many functions there would have to be a lot of pins (pins are the

only source of input/output/communication/etc. on a microcontroller). So they came up with the ingenious idea of multiplexing many functions on a single pin. Now the same pin could be used as an input, output, analog converter, communication, interrupt. Hence the user could use whatever he wanted and just ignore the rest. The same microcontrollers were now open for everyone to use as they wanted. Thus was formed the modern day microcontroller with incredible power to the user.

So basically a microcontroller is a combination of I/O, ROM, RAM and other function on the same chip. But how exactly is this useful? Ok, suppose you are a washing machine manufacturer. You need something to work as the brain of your machine. You have two broad options, a microcontroller and a microprocessor. If you use a microprocessor you will have to add all of its peripherals and this will increase your size and cost (keep in mind the highest priority in industry is cost saving, whatever they tell you). So you go for the microcontroller. Now the need in a washing machine is very specific. You have some buttons as inputs, and motors as outputs and a few sensors to monitor water levels and other such stuff. So how to decide which microcontroller to use? The first thing is we look at all the functions we need. We have input pins for the buttons, output pins for the display, PWM pins to run the motors and Analog to Digital converter pins for the sensor input. Once we have this we decide on the manufacturer. This is based on the comfort level in using them. Some common manufacturers are Texas Instruments, Atmel, PIC, Philips (NXP), and Renesas. Although the basic coding concept remains the same each manufacturer uses a different toolkit. This brings up the comfort part. Atmel, NCP and PIC are easy to use and also have a large support base of users. Texas on the other hand can be harder without prior knowledge.

Arduino exclusively runs on Atmel microcontrollers. Most of these are from the Atmega series. Few of the ICs used are Atmega 8/88, Atmega 328, and Atmega 2560. Thought the architecture and operating

frequency is similar they differ in number of pins and functions and the flash memory (flash memory is where we store the code).

The boards we will be making shall be a version of Arduino UNO. The technical name of our board is a derivative of Arduino Duemilanove. The controller used in these boards is the Atmega 328.

These are its specifications:

Microcontroller	Atmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6 (ADC)
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 2 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Our board differs from the Arduino Uno in the way code is uploaded onto the IC. The Uno uses another controller (Atmega 88) to act as the interface for uploading code. We shall be using the CP2102 to act as an interface between our computer and the Arduino Board.

## What makes a Microcontroller Arduino!

If you buy an ATmega328 from the market and plug it into your board it won't work. The board wouldn't respond at all. That's because the microcontroller we bought from the market isn't programmed to behave as an Arduino. Arduino has defined set parameters for communication between the IDE and the microcontroller. The microcontroller should be initially programmed such that it can interface with the Arduino IDE before we actually start using it as an Arduino.

What we actually do when we program a microcontroller to behave as an Arduino is change its bootloader to the "Arduino Bootloader".

What is a bootloader?

The bootloader is the little program that runs when you turn the Arduino on, or press the reset button. Its main function is to wait for the Arduino software on your computer to send it a new program for the Arduino, which it then writes to the memory on the Arduino. This is important, because normally you need a special device to program the Arduino. The bootloader is what enables you to program the Arduino using just the USB cable.

Without the Arduino bootloader we would not have been able to program the microcontroller so easily.

Thus the Arduino Bootloader is what makes our ATmega 328 an Arduino.

# ArduDIY

---

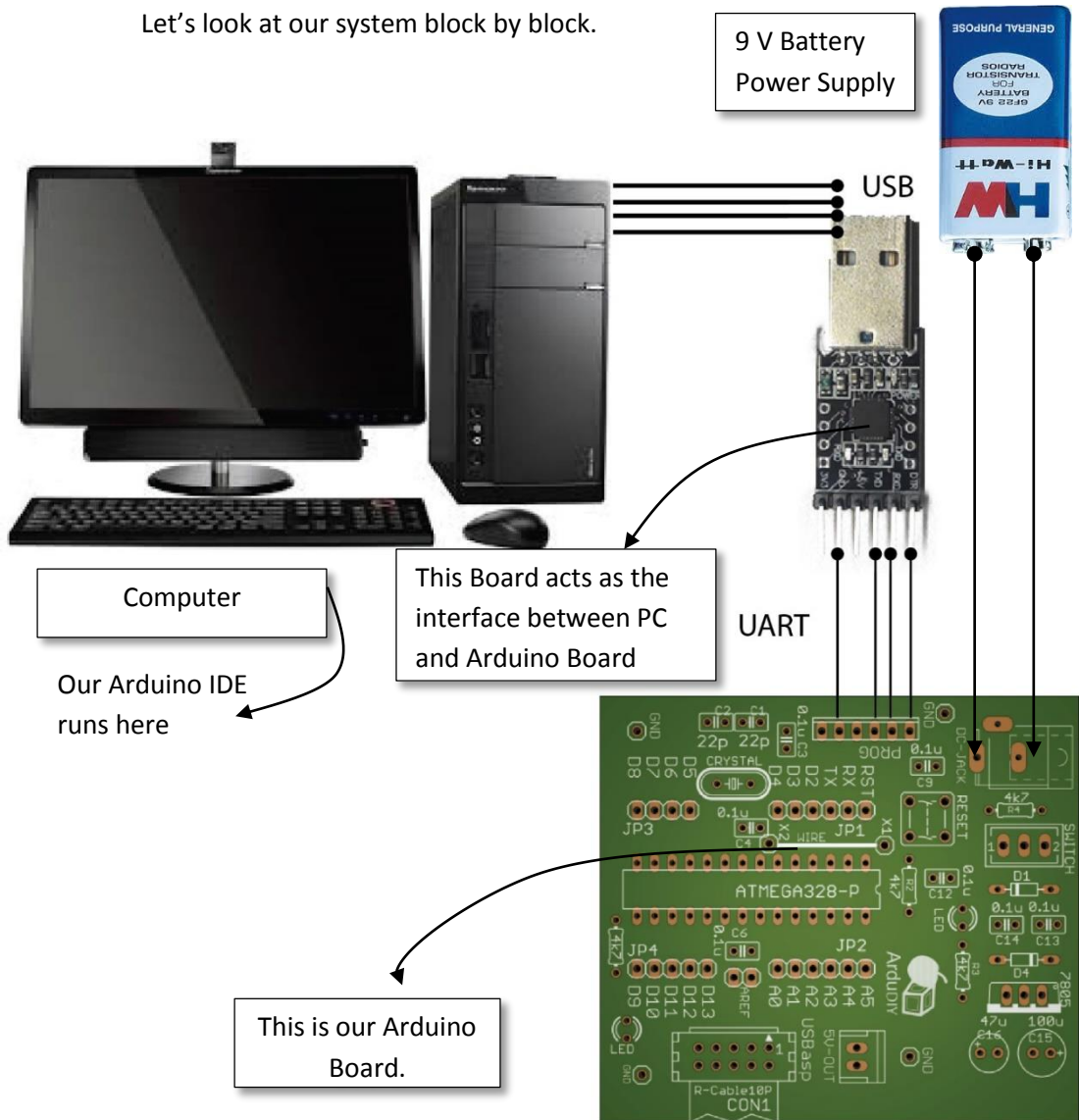
So now that we have gotten ourselves acquainted with microcontrollers and Arduino lets start on our current project the ArduDIY.

ArduDIY stands for Arduino Do-It-Yourself (yeah I know you may have guessed it already). The only premade thing provided to you is the PCB (Printed Circuit Board) everything else shall be done by you. Also towards the end of this you shall be able to make an Arduino Board Completely from scratch. This shall be beneficial when you have to make projects where specialised board designs are required, or in some cases where prefabricated PCBs are not allowed.

This board is not pin compatible with the Arduino Boards. Due to this you will not be able to use Arduino shields but I urge you to try developing your very own. The pins of this board are breadboard and 0 pcb compatible so go ahead make what you want!

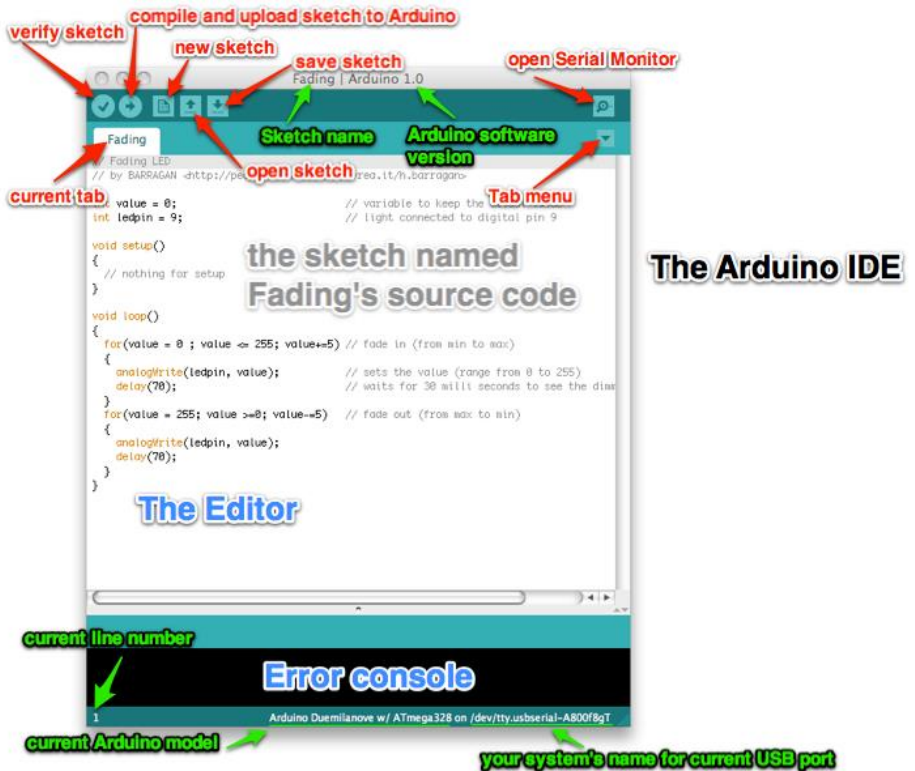
The reason this board was not made pin compatible with the Arduino is so that users could develop a better understanding of how the board actually works and is routed. This will help when you make your own Arduino board.

Let's look at our system block by block.



## Computer:

This will run the Arduino IDE (Integrated Development Environment) it looks like this



We write our code here and use this to compile and upload the code. We can also communicate with the Arduino Board using the serial monitor of the IDE. It uses the USB port of the computer. The console displays status messages and also warnings and errors.

We write the code in the text editor part of this program. In Arduino our code file is called a sketch and saved with the extension ".ino"

## Converter:

Our board and the Arduino IDE running on the computer need to communicate. Now in hardware communication is done with the help of protocols. Think of protocols as languages of hardware communication. Protocols define the way communication should take place and also the data that can be sent and received (Similar to grammar and words in a language). Some commonly used protocols are RS-232, USB, UART, USART, and SPI. When two people are talking if one speaks English and the other speaks French the conversation will be pointless, similarly for two boards to communicate they must support the same protocol. The USB protocol is somewhat of a high level protocol, the IC we are using (Atmega328) does not support it so we need device to act as a translator in between our computer and our board. This is the purpose of our converter board. The board communicates with our computer using USB protocol and with our controller using UART (universal asynchronous receiver/transmitter) protocol. It is powered using the 5V supply on our Computer but we shall not be using the same supply to power our ArduDIY board. The reason for this is that if something goes wrong in our circuit it can damage the Serial Port on the computer.

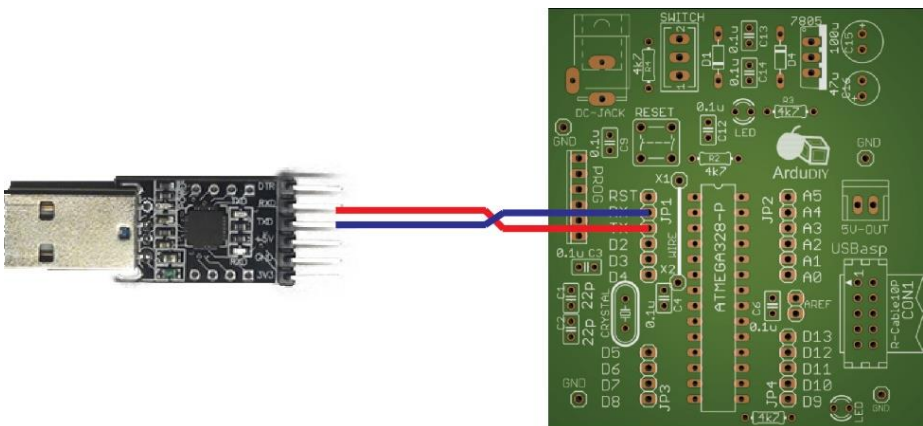
Now this converter board is universal. You can use it with any other board and any other microcontroller as well. It is a serial to USB converter. The signal voltage of this board is 3.3V, it is compatible with all ATmega microcontrollers.



It's most important use is that we shall be using it as a programmer to upload our program to the Arduino. You plug it into your computer/laptop directly and from the pins that come out the ones we need are:

DTR	Data Terminal Ready	It is a handshake signal which initialises communication.
RXD	Receive	Serial Data receive pin.
TXD	Transmit	Serial Data transmit pin.
GND	Ground	Signals are defined with respect to GROUND. Hence for two ICs to communicate they must understand each other's signals and thus their grounds must be common.

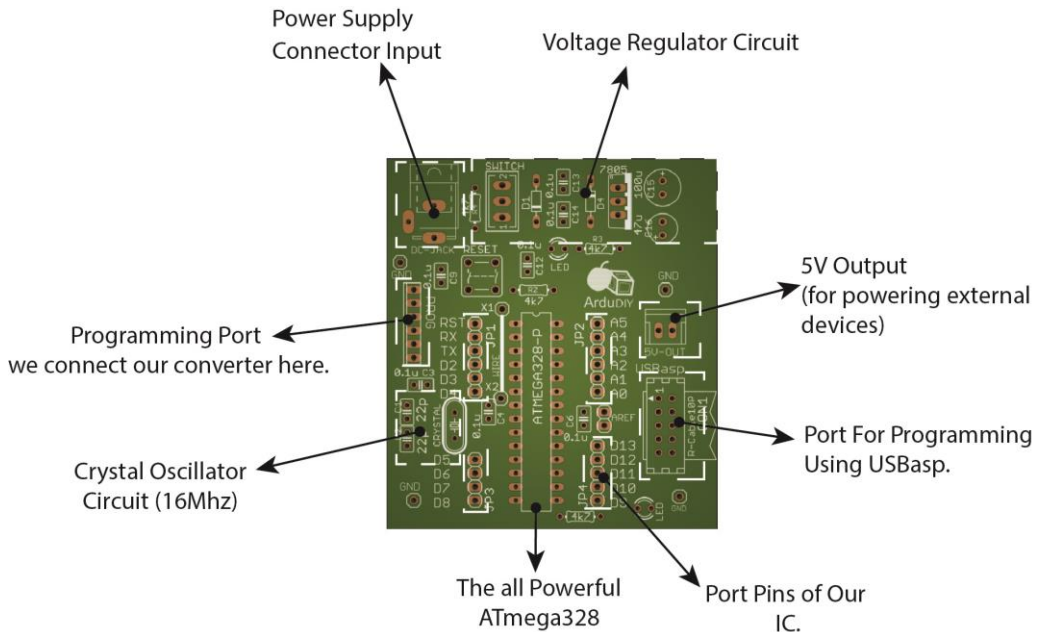
And important thing to note is that the RX of our converter is the TX of our Arduino (Atmega328) and the TX of our converter is the RX of the Atmega328. UART is a serial protocol so the data transmitted by one chip is received by the other on the same line. Hence this configuration.



## ArduDIY Board:

Now let's look at our Arduino board. On a layout level it can be split into the following parts.

We look at it this way as it makes understanding it easier.



Now every board has a corresponding schematic. Whenever we make a printed circuit board usually we have to create two files. The schematic (.sch) and the board (.brd) file. The schematic file must be created first.

The schematic file defines all the components that shall be used and the connection between all the pins of the components. Take for example we want to make a board with an AND gate.

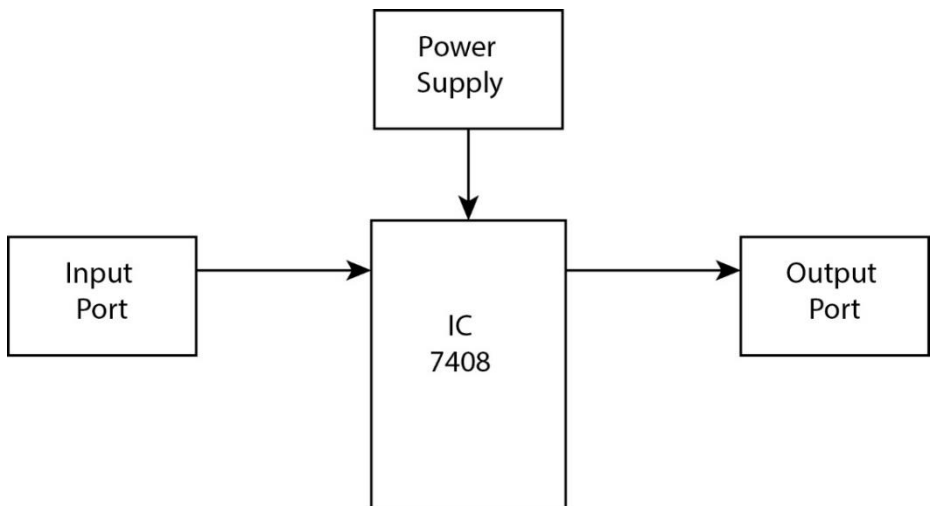
Before starting, the first thing to do is look at the datasheet of the IC 7408. This is a quad two input AND gate, that means there are four two input AND gates in this package. It may look huge and scary at first but believe me, as you read more and more datasheets it will get easier you

will realise exactly which parameters to search in a datasheet and soon you will be able to understand a whole datasheet in just minutes.

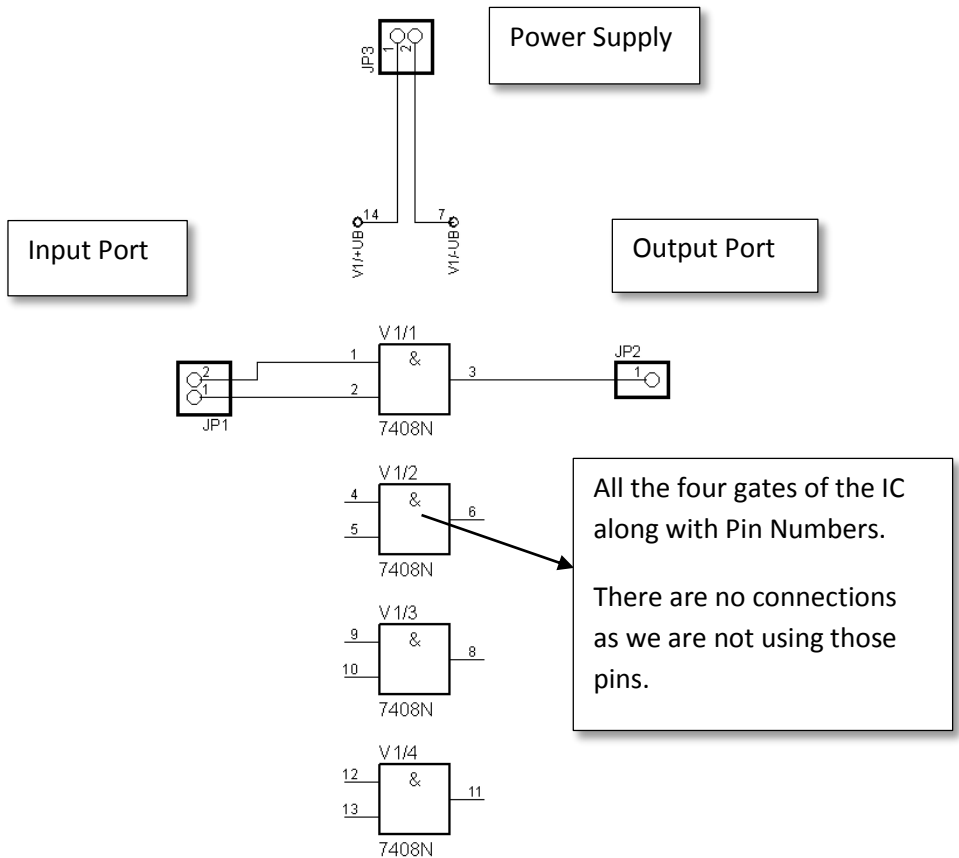
So the parameters we shall look at now in the datasheet are:

Logic Diagram	To understand what type the gate is.
Pin Layout	This is important for routing.
Power Supply	Power supply must always be within permissible limits.
Logic Voltage	This is the input/output voltage the pins support.

So now that we have studied the electrical characteristics of the IC we are using, let's start with the schematic. Let's look at what all we will be needing on the board we make.



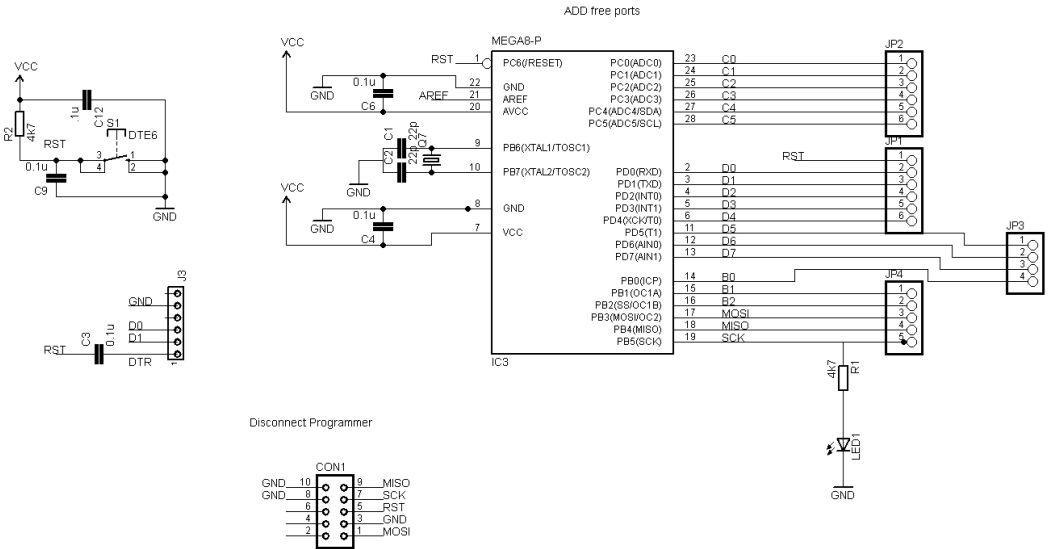
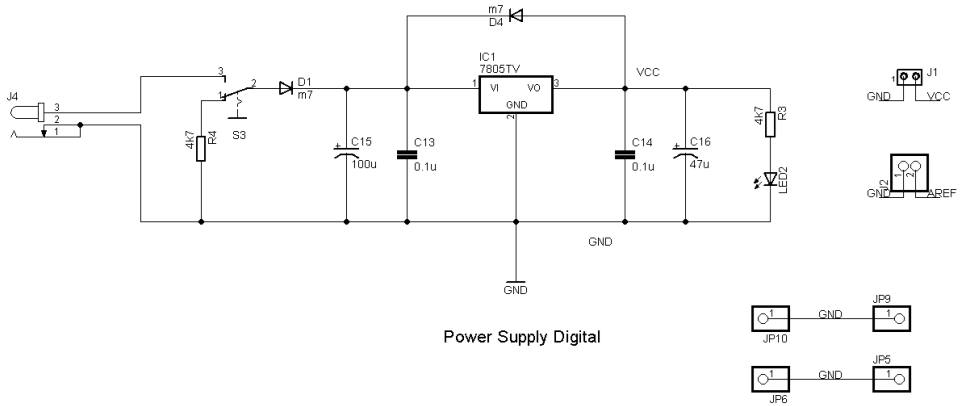
Now let's put this in schematic format (a little prior working knowledge of eagle is needed. I shall soon be making a tutorial on eagle basics.)



You can see the lines represent just connections and not the actual physical layout.

So here we have just defined all the components needed along with the connections for a simple AND gate board. This was just to get you accustomed to the design process.

Now that were familiar let's look at the schematic of our ArduDIY Board:

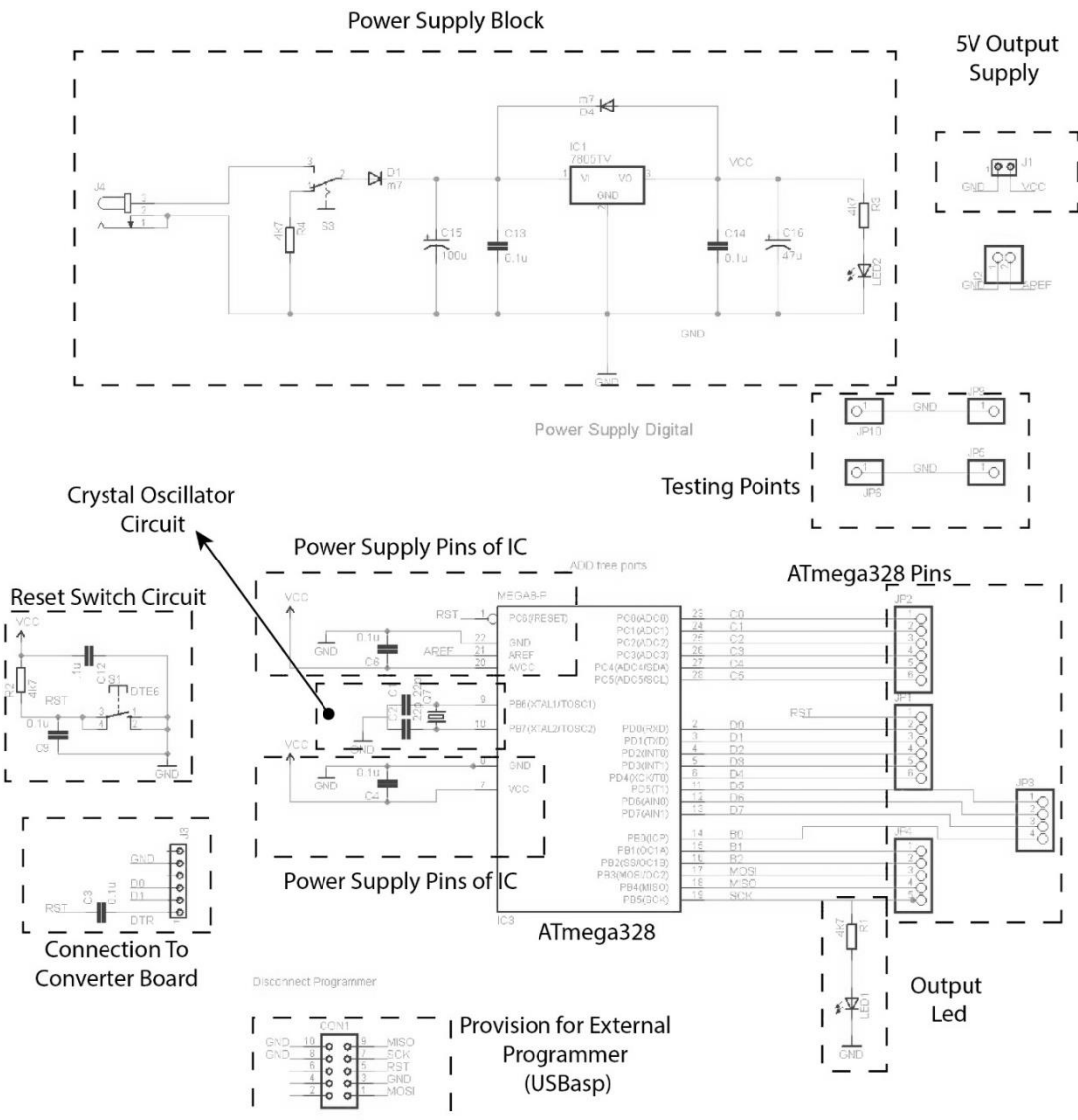


Have a look at this and try to understand whatever you can. Every component is named. Look at where it is connected and see if you can understand why it is connected there. Labels are used to make the wiring in this schematic easier. Labelling means that in the schematic all same named wires (or nets) are connected. So all labels saying "VCC" are actually connected. When designing you can use the show command of eagle to observe the complete connection.

At the centre is our microcontroller (Atmega328) and there are 28 pins coming out of this package and every pin has a specific purpose. If we want the controller to work as it should every pin must be properly connected as per instructions given in the datasheet. Also take the PCB in your hand and see where each component in the schematic is placed. Try to locate all components. (This is important in order to understand routing of the board)

Ok so once you have had a good and through look at the above circuit you can go ahead.

Here is the same schematic with blocks based on function. Check how many of them you guessed correctly.







# Components

---

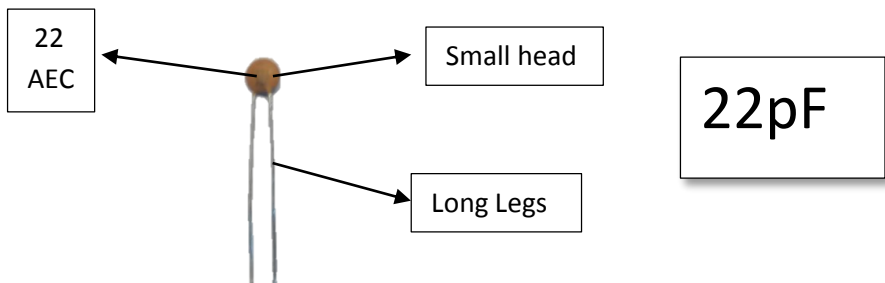
Let's have a look at all the components present in here so that it shall be easier to solder the board.

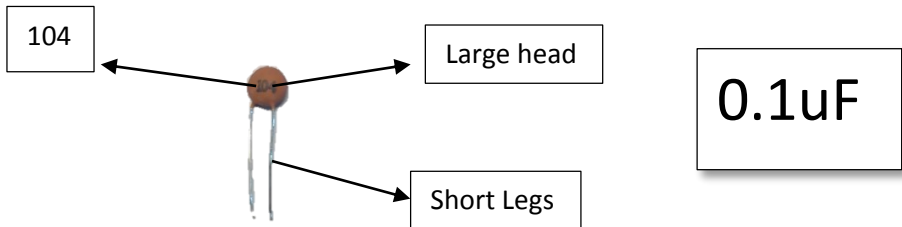
**Resistor:-** 4k7 or 4700Ohms.



**Ceramic Capacitor:-**

Ok we have ceramic capacitors of two different values. Please take care that you identify both correctly as it is very important during soldering. A mix up here can be very hard to debug later.





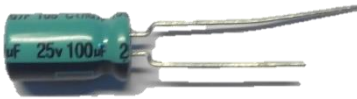
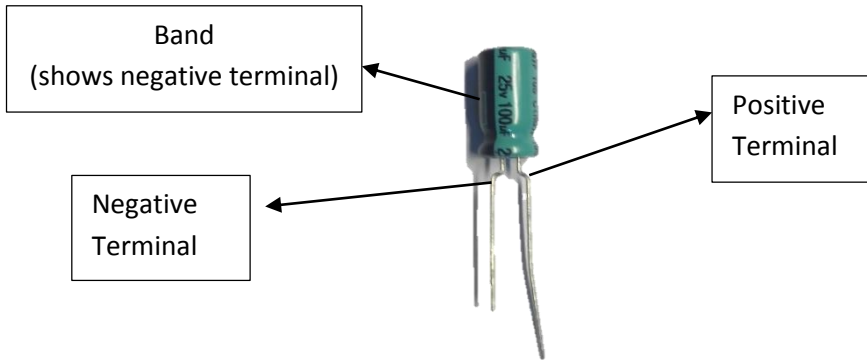
### Electrolytic Capacitor:-

Here again we have two different values. But these are easier to spot as the values are written on the capacitor itself.

Along with capacitance value these capacitors also vary based on voltage value.

But when dealing with electrolytic capacitors please do keep in mind that they have polarity. **IF POLARITY IS NOT CONNECTED PROPERLY THE CAPACITOR MAY EXPLODE.**

The terminal of the capacitor with a band above it is the negative terminal.



100uF/25V

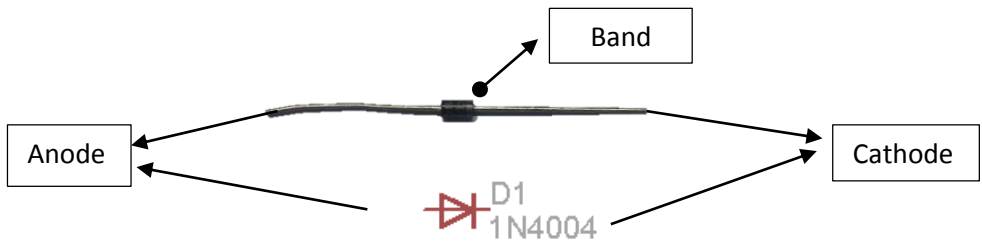


47uF/25V



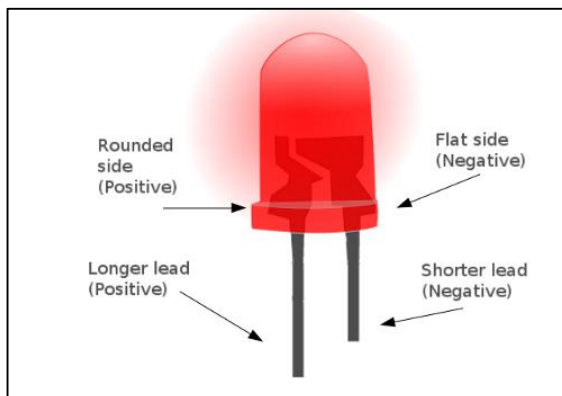
**Diode:** - We use the 1N4007 diode. This is also a device with polarity and must be soldered with care.

Notice the silver/white band on one side. That terminal is the cathode or the negative terminal.



**LEDs:** - Light Emitting Diodes. As they are a type of diodes they also have polarity and must be connected as per their polarity.

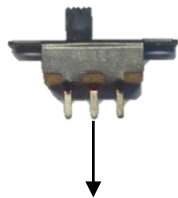
You can identify the terminals of an LED by looking at the length of the terminals.



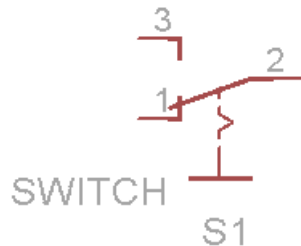
We shall be using two LEDs, Red for Power and Green as the indicator Led connected to the Arduino Pin.



**Slider Switch:** - This is the switch we shall use to power ON and OFF the board. It is a single Pole single throw switch.



This terminal is connected to the other two based on the position of the pole.



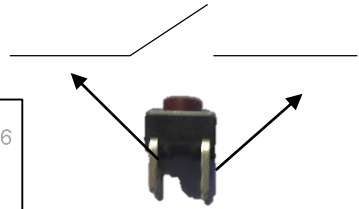
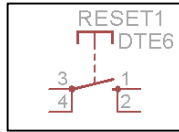
**Reset Switch:** - This is the button switch we shall be using to reset the microcontroller when it is on.

Please use a DMM in continuity mode to understand properly before Soldering.

These are two sets of switches



Side View



Front View

This is one individual set of switch.

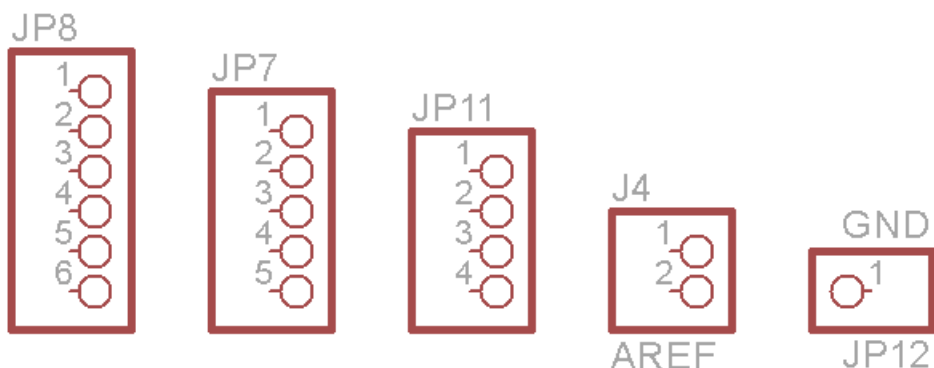
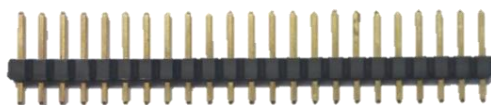
**Crystal:** - The Crystal we use in our oscillator circuit to provide clock to the microcontroller.

We use a 16 MHz Cut Crystal. It does not have polarity so you can connect it either way.



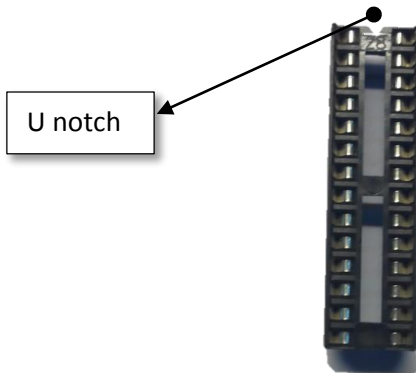
CRYSTAL1 

**Bergstrip:** - This is the connector we use to interface the pins. It is a male type Bergstrip of 2.54mm pitch. We use it in sets of 4,5,6,1 i.e. 1x4, 1x5, 1x6, 1x1. You can use a hand cutter to break apart the whole strip into smaller parts.

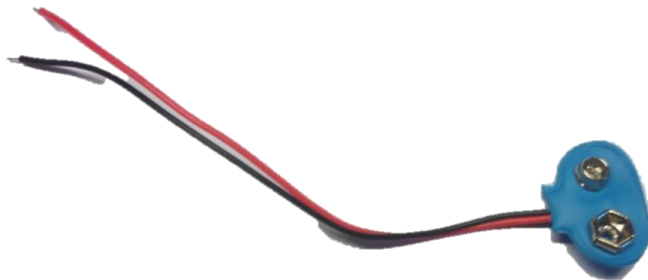


**IC Base:** - We shall use a 28 pin IC base to fit the microcontroller into. We do not directly solder the microcontroller as the heat of soldering may damage the pins and many times in prototyping we need to change the microcontroller IC. Now if something happens to the microcontroller we replace just the controller IC without replacing any other components on the board.

**The U type notch on the IC base should match the one shown on the PCB.**



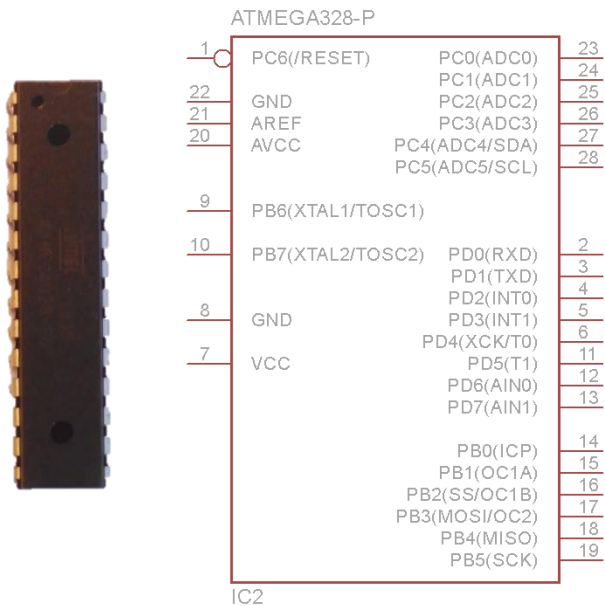
**Battery Connector:** - We shall use this to connect our board to a 9V battery. The red wire is positive of the battery and the black negative. We will solder these wires directly onto the board.



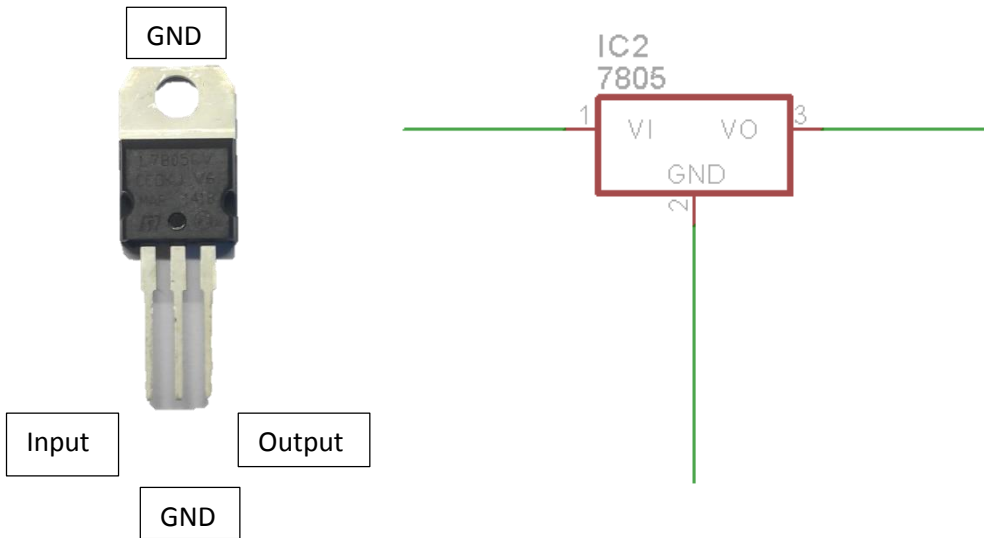


**Microcontroller:** - And finally we have the heart of our board, the all-powerful Atmega328P.

Please take special care with this IC, it can be damaged due to electrostatic discharge (from our fingers).



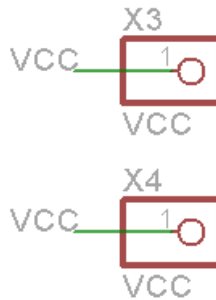
**Voltage Regulator:** - The voltage regulator we use is the 7805. It is a linear voltage regulator. The output voltage is 5 Volts. The IC is available in many forms of which we will be using the DIP package. Minimal external circuit is required with this IC (just a few capacitors). It is one of the most widely used series of voltage regulators.



**Single Strand Wire:** - As the PCB we use is single sided, if two connections cross we need to take one of them over another. We use a wire to do this. A single strand wire is preferable when making such connections.

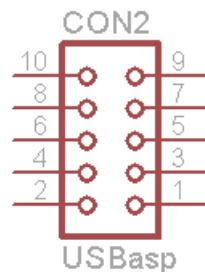
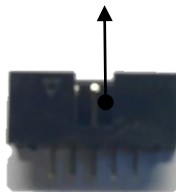


As there is no direct method to add wire in Eagle, I have used these two pins to represent the wire.



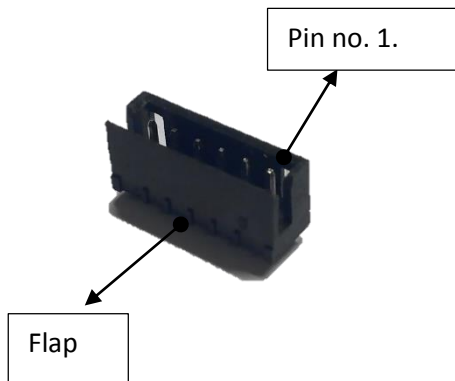
**Box Header:** - This is the 10 pin box header. It is used to use an external programmer such as the USBasp when burning the bootloader or programming.

There is an arrow here which denotes pin no. 1.



**Relimate:** - We shall use the Relimate connector to connect our board to the converter board.

The Relimate connector fits only in one direction so take care when soldering this. Notice the flap of the connector, we shall use that as an identifier while soldering.



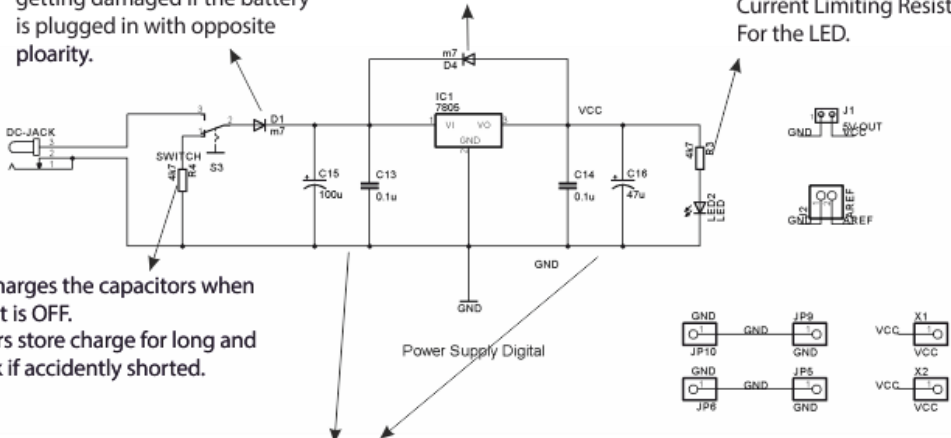
# Circuit Working

Now that we know what component is what, let's look at the circuit a little more in detail and understand its working.

**Protection Diode.**  
This prevents the circuit from getting damaged if the battery is plugged in with opposite polarity.

**Protection Diode.**  
Provides a path across the voltage regulator for reverse voltages. Thus protecting the regulator in case of sudden reverse voltage.

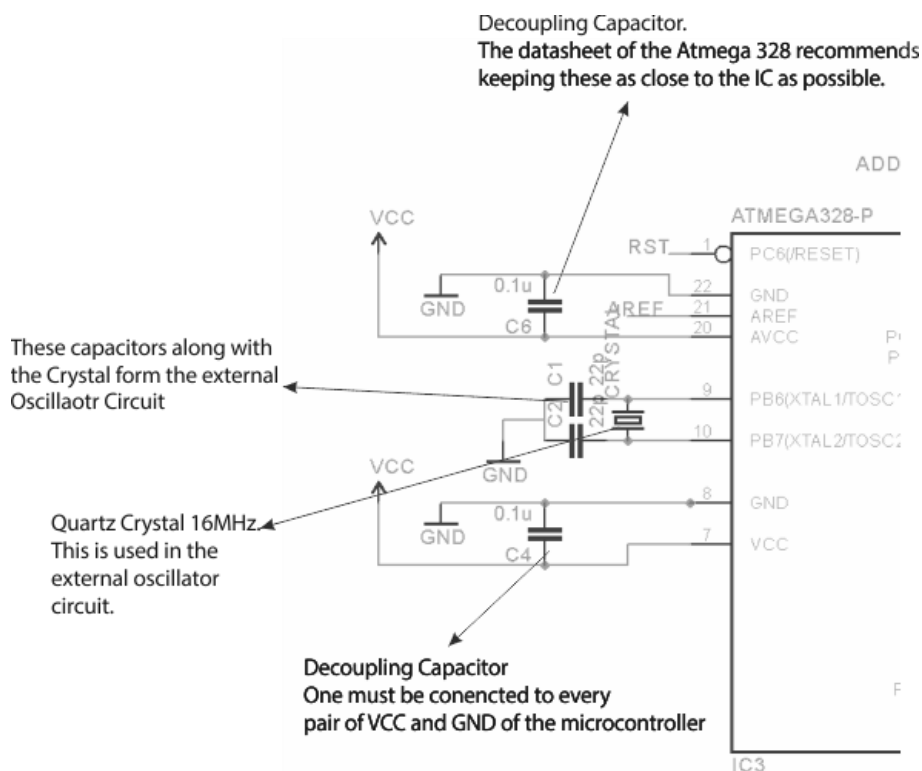
**Current Limiting Resistor.**  
For the LED.



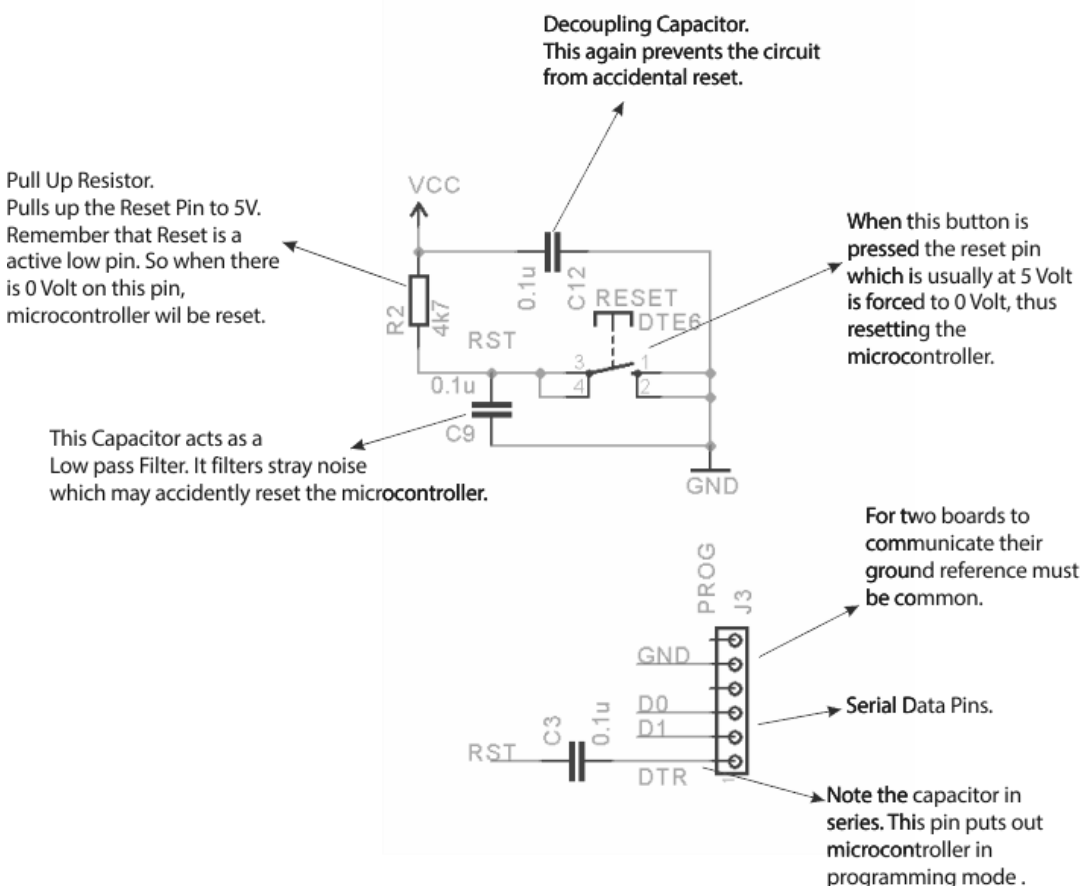
This discharges the capacitors when the circuit is OFF. Capacitors store charge for long and can spark if accidentally shorted.

**Capacitor:**  
They store charge and are used here for filtering purposes. A capacitor connected in parallel prevents sudden change of the voltage across it thus regulating voltage and preventing noise.

This is the Power Supply Block. Its main function is to generate 5V regulated voltage to run our microcontroller and other peripherals.



This block contains the power supply pins and the oscillator circuit of our microcontroller. These are the most important pins required for our microcontroller to work.



This is the reset and programming connection circuit diagram.

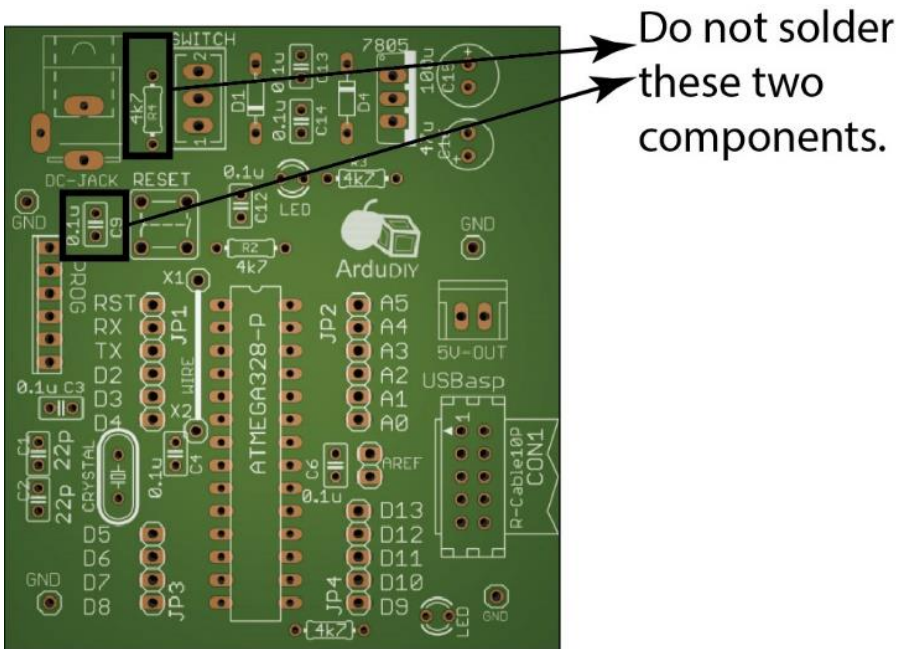
## Kindly Note

**DO NOT SOLDER CAPACITOR C9 and RESISTOR R4.**

It has been added to prevent stray noise from resetting the controller in noisy environments. But if this capacitor is added the board cannot be programmed using the converter board.

If you need this capacitor (due to problems of noise) you will need to program this board using the external programmer.

The resistor provides a continuous load to the power supply. But if this is used with a 9V battery it will drain fast.





# Soldering

---

Soldering is one of the most important tools at the disposal of a DIY enthusiast. It helps in rapid prototyping, repairing and building stuff from scratch.

Proper soldering is essential.

**If a joint is not soldered properly the board won't work as expected and debugging is very hard when there are 40 to 50 solder joints along with components, we can't find out if a component is faulty or the soldering is. Thus take utmost care while soldering and double check each joint.**

The tools we shall require for soldering have been listed earlier. Here is a list just to recheck.

1. Soldering Gun
2. Solder Metal
3. Wire Cutter
4. Flux (Optional though preferred)
5. Wet Sponge (Optional)

1. Soldering Gun

The soldering gun we most easily get in these areas is the yellow coloured Solderon soldering iron. This is completely sufficient for all DIY projects and is even capable of SMD soldering. If you are soldering more intricate stuff I recommend using a soldering station. The advantage with these is that they come with temperature setting, proper grounding for ESD protection and other such facilities which help in better soldering.

Please conduct proper research before buying a soldering station. As

far as possible do not buy Chinese replicas, they seem cheap but will give you problems within a month.

Most soldering guns come with a flat ringed metal tip. For SMD work you can use the pointed tip. There is a different type of tip available known as a coated tip. It is about 3 times the cost of a normal tip but gives superior performance and lasts a lot longer. If you are going to do a lot of soldering work give it a try.

Soldering is an art, every individual has his own soldering style. If you are practise soldering you too will come to realise this. You can make out who has soldered a board just by looking at it. I urge you to look at the basic techniques and then develop your own style. There are many options, you could go with whatever is the fastest or whatever seems the safest or anything!

## 2. Flux

What flux basically does is it removes the metal oxide layer which is formed due to high heat. This layer is the blackened stuff you see on your soldering gun and on the board. Oxidation eats away at the metal and comes in the way of a proper solder joint. It can give rise to dry solder. There are many types of flux like liquid flux, flux paste etc..

## 3. Solder Wire

Solder wire is easily available in most electronics shops. Observe the thickness before buying, if it seems too thick it may give problems melting properly.

## 4. Wet Sponge

If you have a soldering gun stand you must have noticed a small yellow thick sponge on the front. This is meant to clean the gun and must be wetted before use. Just rub the tip of your gun over the sponge. Do this with a swiping movement without maintaining contact for too long.

## **How to Solder**

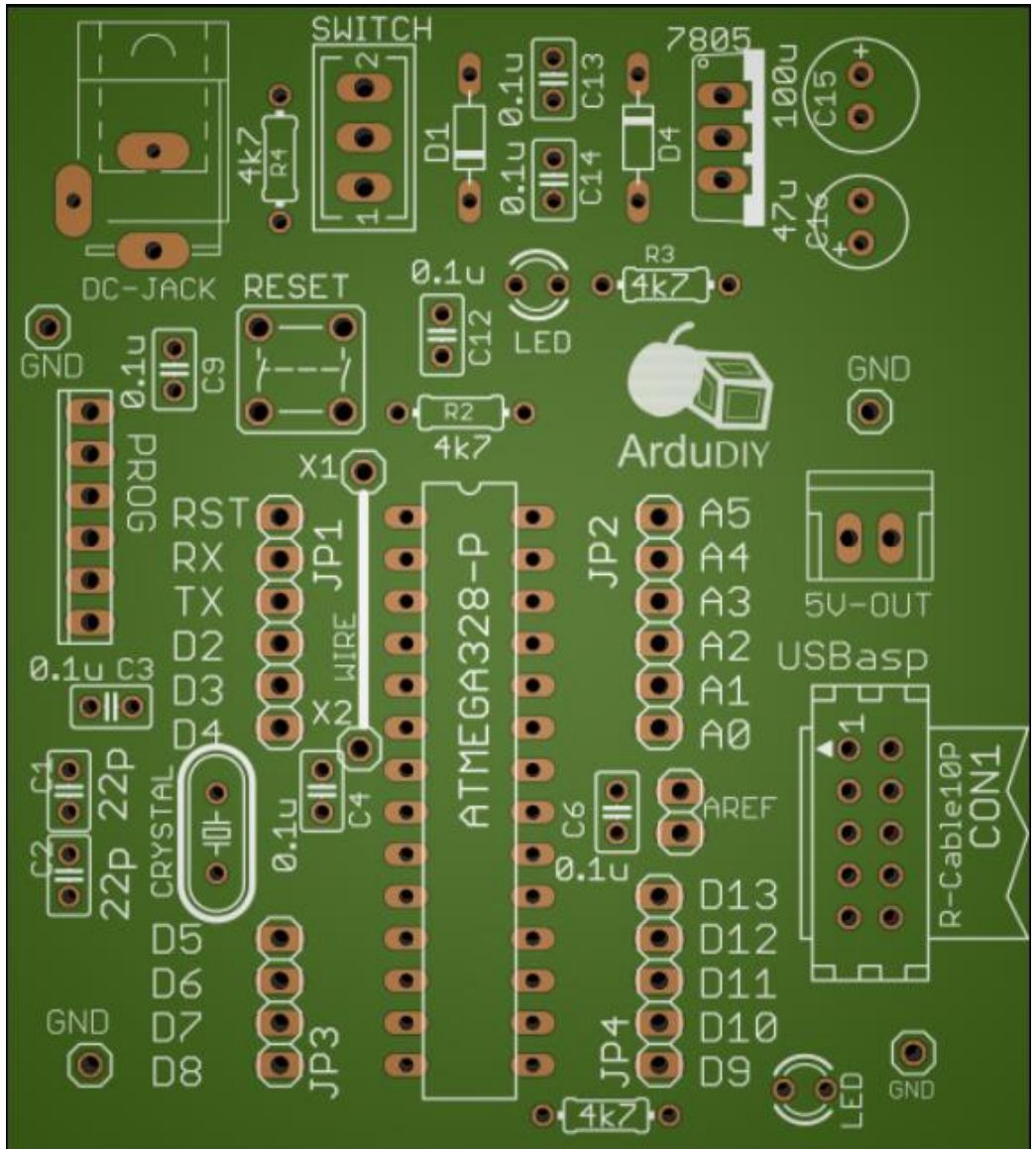
1. Check that your soldering gun is properly heated. (You should be able to feel its heat near your face from a distance of 10 cm.)
2. Place the component through the holes. Fasten it if necessary by bending the pins outwards.
3. Touch the soldering iron on the joint you want to solder. Use the flat tip so that it covers the most area.
4. After the joint is sufficiently heated, bring the solder metal into contact of the joint and gun.
5. The metal will melt and flow towards the joint.
6. When enough solder has melted into the joint take the soldering metal away and simultaneously gently lift the soldering gun UP.
7. After the solder has cooled check to see if it has stuck to the joint properly.
8. A perfectly soldered joint is conical in shape.
9. Clean your soldering gun by tapping it in flux and/or using the wet sponge.

## **After Soldering**

1. Let the joint cool a bit before moving anything.
2. Visually check the joint to see if the solder has held on to the joint correctly.  
I have observed that visual inspection is many times the best method of checking.
3. Use a DMM in continuity mode.

Now let's start with the soldering. We go step by step starting from the small components first.

Use this image as a reference. All components are labelled along with polarity where ever required.



Use that Image as a reference for locating where components are. It is an image of the bare PCB board without any components. You will notice that as you go on soldering components it gets harder to spot where you have to solder new components.

Go in this order for soldering the components, it will be easier this way. Tally the components with the quantity given. That way you will not miss out on any components.

Component Name	Value	Quantity
Resistor	4k7	3
Ceramic Capacitor	0.1uF	6
	22pF	2

**DO NOT SOLDER CAPACITOR C9 and RESISTOR R4.**

**For soldering the Electrolytic Capacitor, first identify which terminal is negative and which positive.**

**The PCB has a marking for the positive terminal.**

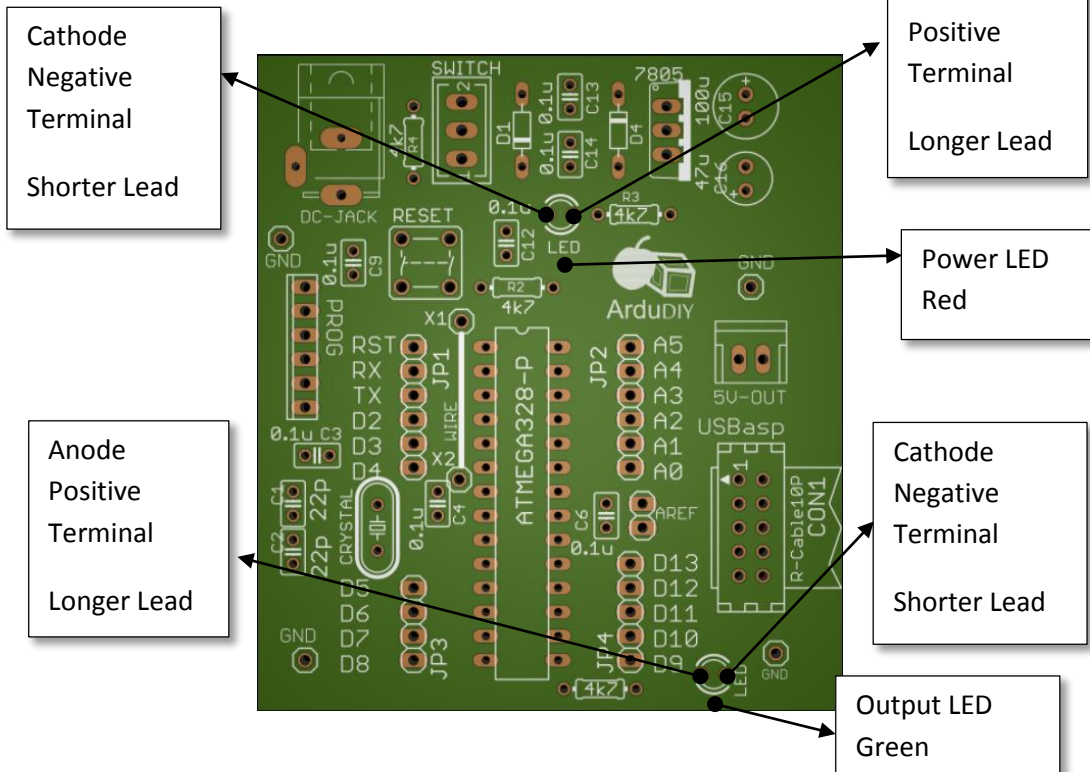
Electrolytic Capacitor	47uF	1
	100uF	1

**Similarly for the LEDs and diodes, first identify the anode and cathode.**

Diode	1N4007	2
LED	3mm	2

**The diodes have a clear marking to show the polarity (the side with the band is cathode or negative terminal).**

**The Led on the other hand has no such marking. Use this for reference.**



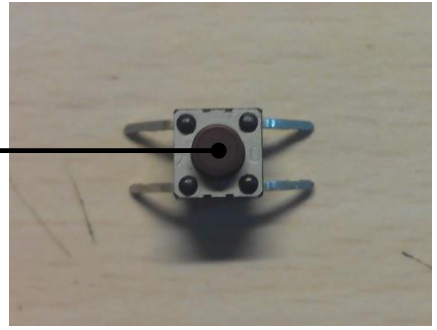
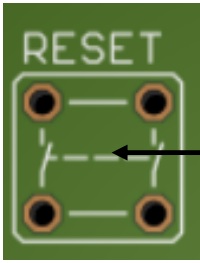
**Note that both LEDs have a different orientation.**

You can swap the led to a colour of your choice, Blue and white is a great combination.

Now we solder the Reset Switch. Look at the diagram on the board properly and then connect accordingly. Remember there are two sets of switches in our reset switch.

Solder it like this.

**Later keep in mind to check this with the DMM as, if this is not soldered in proper orientation, the board will continuously be in reset state and won't work.**



IC Base	28 Pin	1
---------	--------	---

For the IC base, place it on the board and support it from the other side with your fingers (careful so that you do not burn yourself). Now solder two pins diagonally. After that you can solder the rest of the IC base with ease.



Bergstrip	-	
-----------	---	--

Now let's solder the Bergstrip. Cut it out in pieces with a cutter. There is a technique to solder this easily.

Initially solder just the first pin and adjust the Bergstrip by touching the other side, so you do not burn yourself. Once a pin is soldered you can continue with the complete strip. After doing that just retouch the first pin you soldered again. The single pin (1x1) Bergstrip soldering can be a

bit tricky. Use someone’s help to hold the other side using forceps or a plier or if you are doing it yourself use the plastic housing to adjust the pin without touching it for too long.

Use the image of the completely soldered board to get where bergstrips must be soldered.

7805	-	1
Slider Switch	-	1

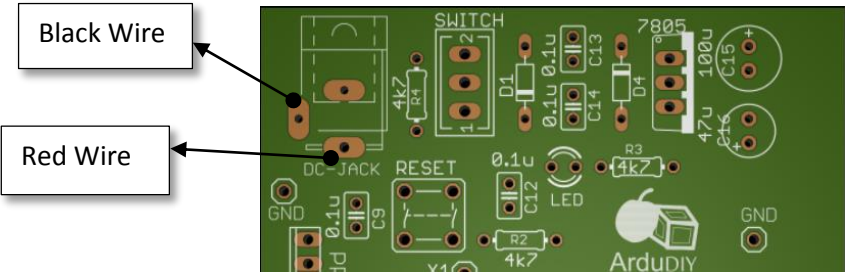
Ok almost there, we now solder the slider switch and the voltage regulator. For the regulator look at the indication on the board for proper orientation.

Battery Connector	-	1
-------------------	---	---

The battery connector wires need to be soldered onto the board. There is also a provision for soldering a DC Jack on the board, you can proceed with what you are comfortable with. Just put the wire through the slot and solder it on the bottom.

Look at the image below to get the positions for the red and black wires.

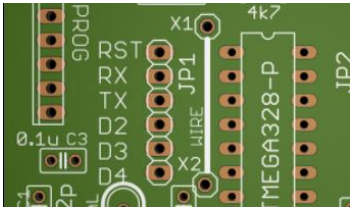
You can also solder a DC jack here in case you are using an adapter. As you can see the original component on the board is a DC Jack.





Single Strand Wire	-	1
--------------------	---	---

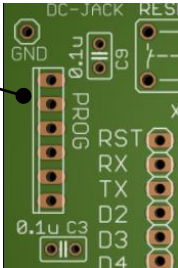
While soldering the wire, first cut the wire to proper length. An easy way to do this is to directly take a measurement by placing the wire on the board. Now strip the ends of the wire to expose the strand underneath. Once that is done place the wire in the two holes provided. Solder one end first, check, then solder the other point. As far as possible try to keep the wire as close/ flat to the PCB as possible. This is not only a good soldering practise but will also make the board neater and prevent the wire from coming in between other components.



PCB Mount Relimate	6 pin	1
	2 pin	1

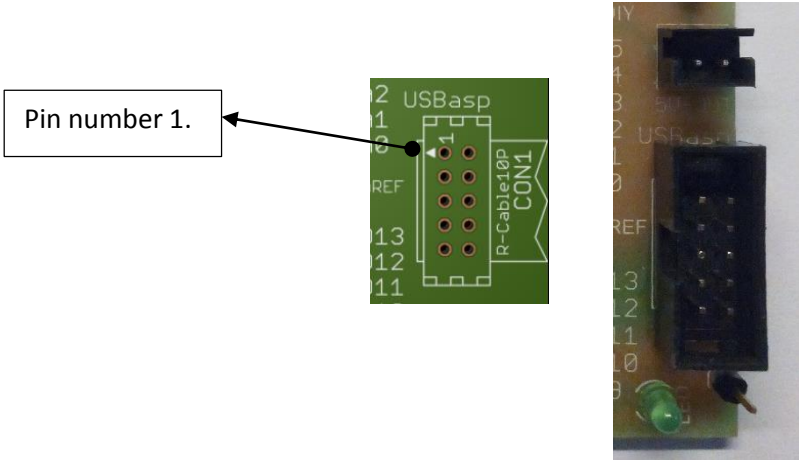
When soldering the Relimate we must take care that it is soldered in the proper orientation. First identify the flap. When soldering the flap must be outwards. Use the print on the PCB as reference.

This is the flap.  
It is facing the outer edge of the board.



Box Header	10 Pin	1
------------	--------	---

Almost there, while soldering the box header first identify pin number 1 on both the header and on the PCB. To identify pin number 1 read the section above where all components are introduced (It is shown by a small arrow on the header as well as the board).



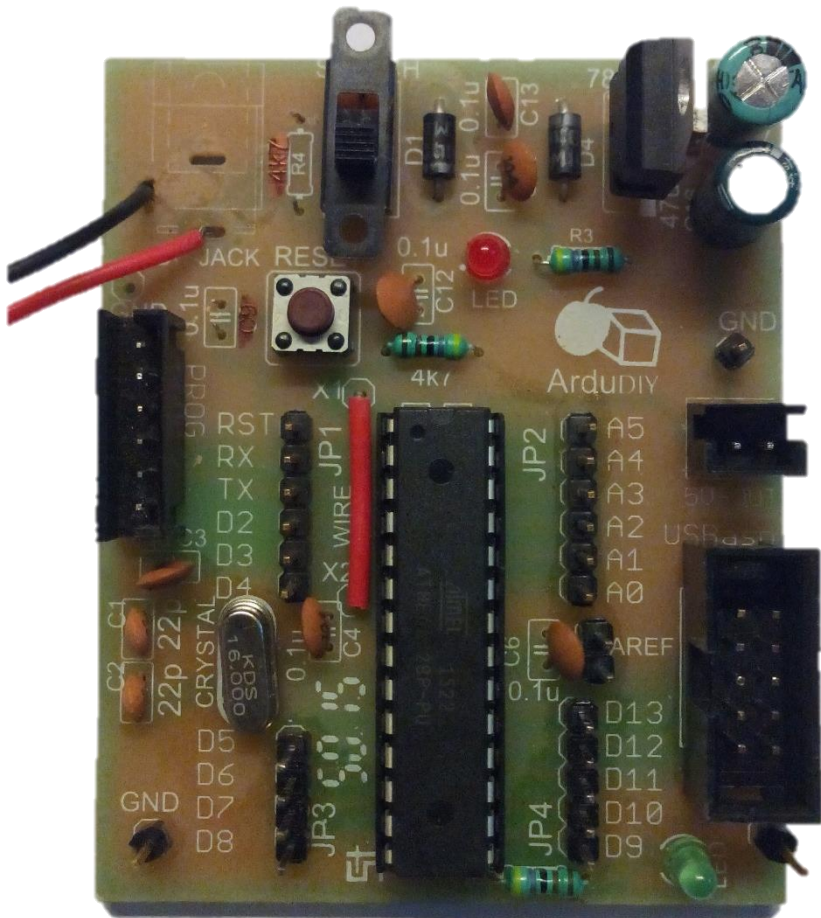
Crystal	16MHz	1
---------	-------	---

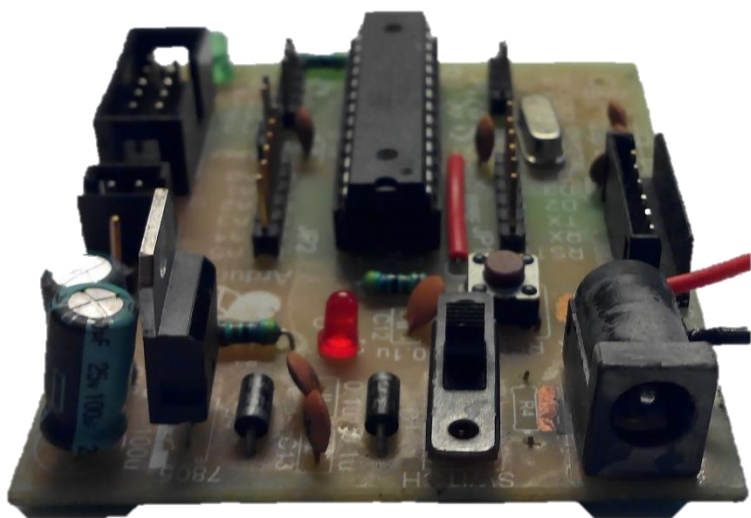
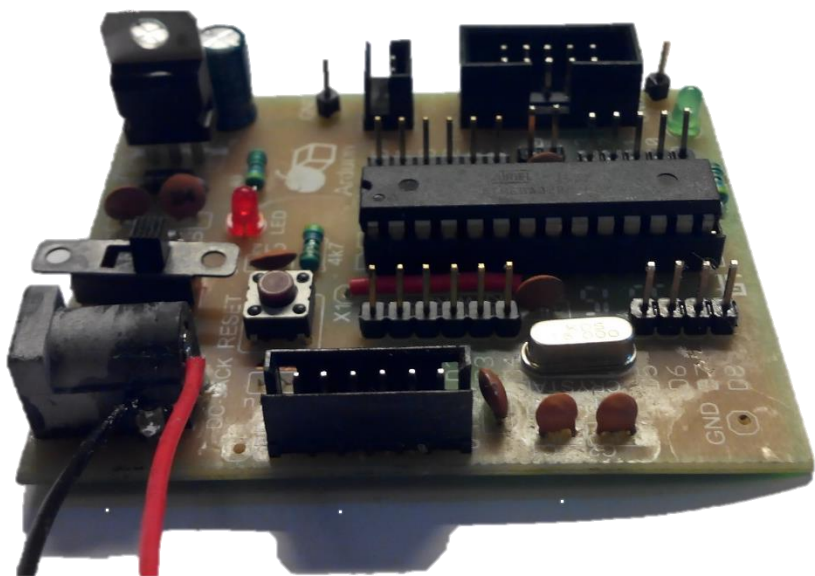
The crystal is the last component we solder. This is because the quartz crystal is quite sensitive. Do not hold your soldering gun to its points for too long while soldering. But do place it properly (pressing it onto the surface so that there is no space between the crystal and the PCB) and solder it thoroughly.

Now that we are done soldering, take a break and look upon what you have made ☺

Here are some reference images.

The DC jack is not included in this kit, you can solder it if you plan on powering this board using an adapter. The adapter can be 8V to 20V. Preferably 12V.





# Testing

---

Now that we are done soldering, before we actually start we need to perform some basic testing on the board. **Before we plug in the battery and the ATmega328.** This is done to prevent damage to components in case of any error. This is known as dry testing. A dry test means completely checking the board for soldering flaws, incorrect component flaws and connection errors without connecting the power supply.

This must always be done before using any soldered board.

Suppose due to some error the Vcc and Ground of a soldered board is short circuited (connected directly) now if we connect the supply without dry testing a large amount of current will flow through the board and it could damage components and even destroy the board due to excessive heating. Now that's a lot of pain due to a simple problem. That's why always Dry Test!

## Visual Checking:

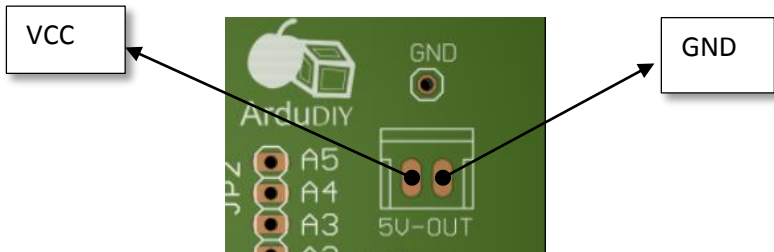
Turn the board around and have a good look at all the soldering points. Start from the top left corner and progress towards the right line by line till you reach the bottom right corner.

Keep a look out for any stray solder splashes or dots, see that every joint is soldered properly and has a conical shape. There should not be any irregular soldering and this leads to something known as dry solder and the whole component may come off after a few uses.

## Testing Using the DMM:

Put the DMM in continuity mode (the mode where the DMM beeps when the two probes are connected to each other). Now the first most important thing we check is the VCC and GND. We do not want the board to go poof the moment we turn it on. The best place to check this would be the 5V Output Port.

**Remember to hold the DMM there for a long while if it is showing a short. This happens due to the electrolytic capacitor we have connected. If the DMM beep does not stop even after 40 seconds, then we have a problem.**



These two must not be connected in any circumstance.

## Turn the board on:

**DO NOT PLUG THE Atmega 328 IN AT THIS POINT**

Connect the battery to its connector and turn the slider switch on. The power led should turn on. Everything else should be off. Carefully touch the voltage regulator gently. It should not be burning hot. If it is very hot immediately switch off the board and plug out the battery, there is a short circuit.

After you have turned it on and the regulator is not hot, wait for about 20 seconds. Again check that the regulator is not very hot (warm is ok). The red LED should be continuously on. If everything is alright we can proceed.

## Plugging in the Atmega328:

Now that we have tested most of the components, we plug in our controller. Gently plug the IC into the base. The U notch on the top of the IC should match the U on the IC base (there is a small one, observe carefully).

Be very gentle about this and take care that all pins are inside the base.

If you ever want to remove the IC from its base, do so with the help of a small flat screw driver. Gently lift one edge then the other edge, keep on doing this till the IC comes out. It must be done in this way to prevent bending of the IC pins.

**The ATmega328 provided is preloaded with the Arduino bootloader. Only ICs with the Arduino Bootloader can be used as an Arduino. That is why ICs bought from the market won't directly work unless you burn the bootloader.**

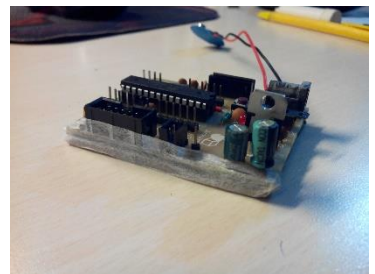
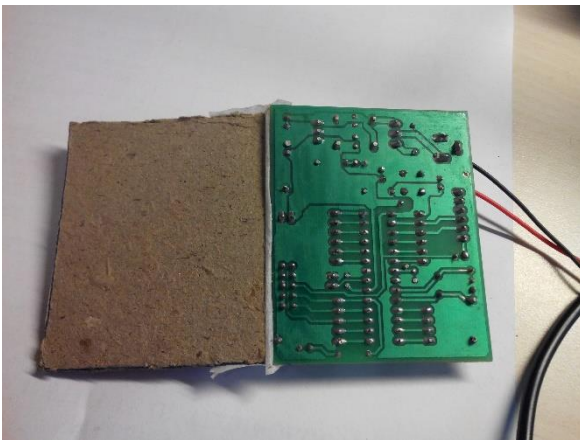
Now that our hardware side is ready, it's time to move on to the software.

## **ALWAYS COVER THE BOTTOM OF YOUR PCB WHEN USING.**

While using the ArduDIY for your projects, always keep the bottom side (this is the side you soldered on) covered. Many times we absent-mindedly keep the board on your work table as it is. Our work table is littered with wires and small pieces of metal, especially the small strands we get after cutting the legs of a resistor or capacitor. These can touch and connect two points on our board and short circuit the whole system damaging the board.

Sometimes we also need to check the bottom soldered part of the board, so it's not a good idea to permanently close off the bottom. The best solution is to use a thick cardboard and tape it on one side of your board. This way when using it the cardboard will cover the board and if needed the cardboard can be just folded off to access the board.

Take any cardboard of good thickness (covers of our old hardbound notebooks works well) mark the boundaries of the PCB, and cut away. Stick only one side with scotch tape.





# Software & Programming

Before we begin you shall need to install two things. Firstly the Arduino IDE and secondly the drivers for our converter board.

You can get the latest Arduino Software at [www.arduino.cc](http://www.arduino.cc)

On their homepage select the download tab. At the time of writing this Arduino 1.6.6 was the latest version, though all higher versions will also work.

Preferably use the Arduino Installer and not the ZIP file. It makes things easier.

Once you have downloaded, install Arduino.

Now we install the drivers for our software board. The converter IC on the board is CP2102. It is manufactured by Silicon Labs, we can obtain the drivers from their website.

Just Google “CP2102 driver” It should probably be the first link. This is the full link.

## CP210x USB to UART Bridge VCP Drivers

The CP210x USB to UART Bridge Virtual COM Port (VCP) drivers are required for device operation as a Virtual COM Port to facilitate host communication with CP210x products. These devices can also interface to a host using the USBXpress direct access driver. These drivers are static examples detailed in application note 197: The Serial Communications Guide for the CP210x, download an example below:

 [AN197: The Serial Communications Guide for the CP210x](#)

### Download Software

The CP210x Manufacturing DLL and Runtime DLL have been updated and must be used with v6.0 and later of the CP210x Windows VCP Driver. Application Note Software downloads affected are AN144SW.zip, AN205SW.zip and AN223SW.zip. If you are using a 5.x driver and need support you can download archived [Application Note Software](#).

### Download for Windows XP/Server 2003/Vista/7/8/8.1 (v6.7)

Platform	Software	Release Notes
 Windows XP/Server 2003 Vista/7/8/8.1	<a href="#">Download VCP (3.66 MB)</a>	<a href="#">Download VCP Revision History</a>

### Download for Windows 2K (v6.3a)

Platform	Software	Release Notes
----------	----------	---------------

 **Find Products Fast**

[Parametric Search](#)

[Cross-Reference Search](#)

 **Get Support & Tools**

[Software Downloads](#)

[Development Tools](#)

[Reference Designs](#)


[Documentation](#)

[Application Notes](#)

[Knowledgebase](#)

[Community](#)

[Training & Resources](#)

 **Need Help?**

[Technical Support](#)

[Contact Sales](#)

GET THE LATEST  
DOCUMENTATION  
UPDATES.

[Register today](#)



<https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx>

The download page looks like this. Download the Drivers and install them.

Once you have installed the drivers you can plug in the converter board into your USB port.

You will get a notification and the PC will auto detect. Now we have to know which COM port has been assigned to our converter board.

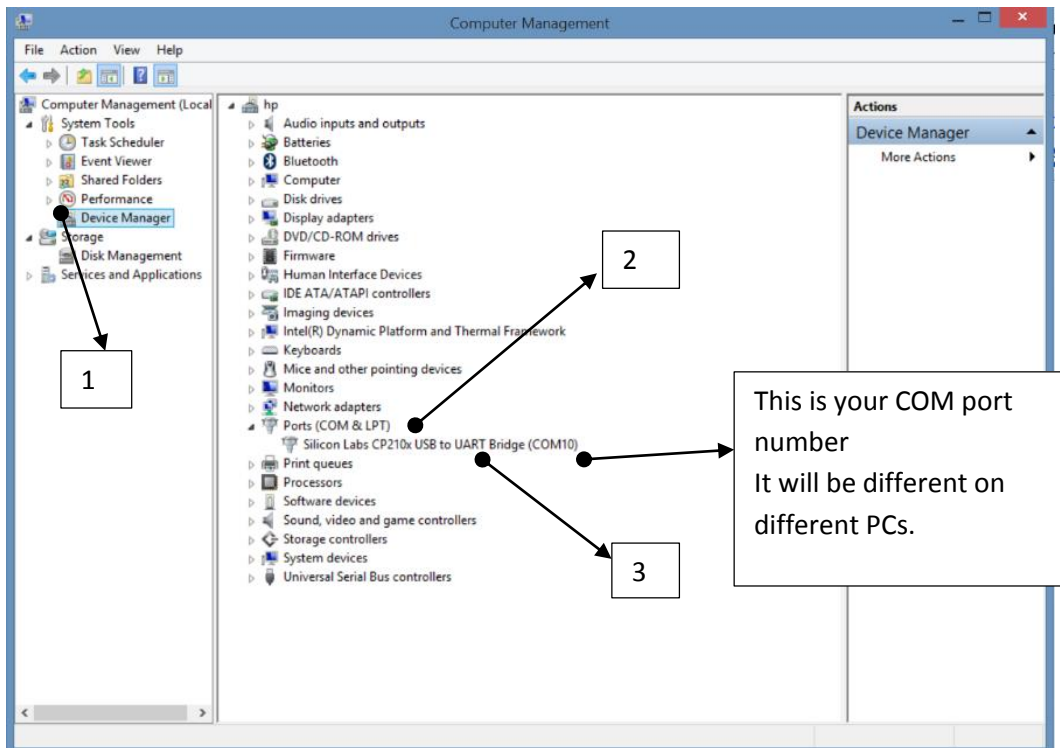
(Basically how this works is that for any I/O device we connect to our PC, Windows assigns it a COM port number. To use this port we have to know the COM port number. )

To view the COM port on Windows 7 and above. First plug in the Converter Board into the USB. Then right click on “My Computer” and click on “Manage”.

Then click on “Device Manager” on the left. After then click on “Ports (COM & LPT)” on the list that appears on the right.

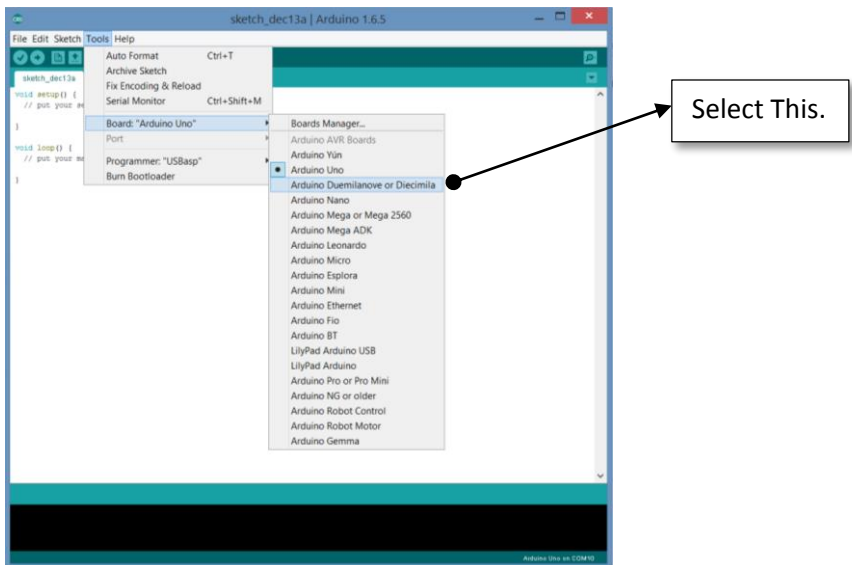
You will see the connected converter board along with the COM port name in brackets.

If “Silicon Labs CP210x USB to UART Bridge” cannot be seen here the drivers have not installed properly.

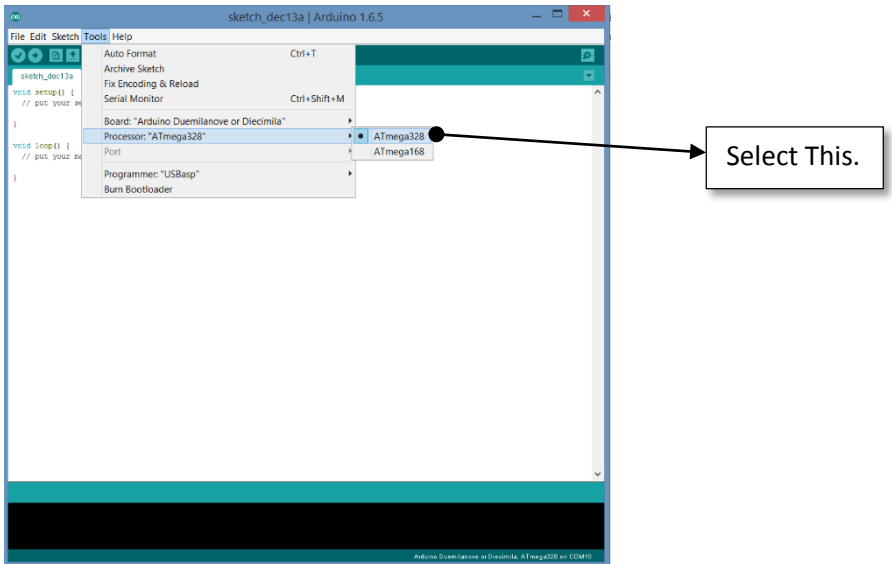


So now that we know our COM port number, open the Arduino IDE. Before we can start programming, we will need to set some settings in the Arduino software.

Go to Tools -> Board -> And select Arduino Duemilanove or Diecimila

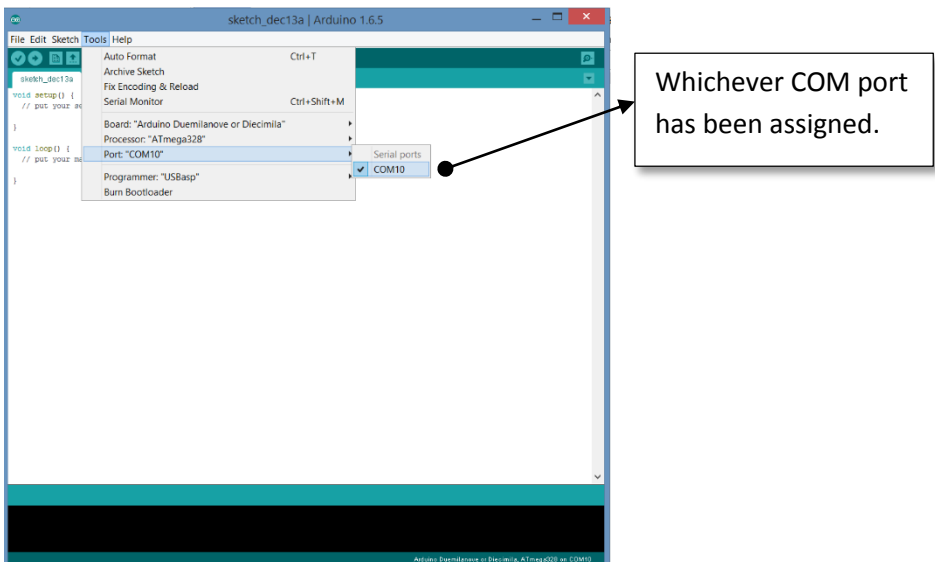


Now go to Tools -> Processor and select ATmega328



Lastly go to Tools -> Port -> Now select the COM port of our converter board.

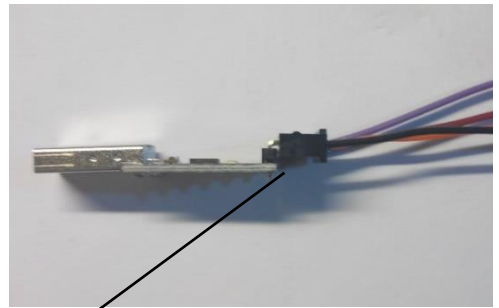
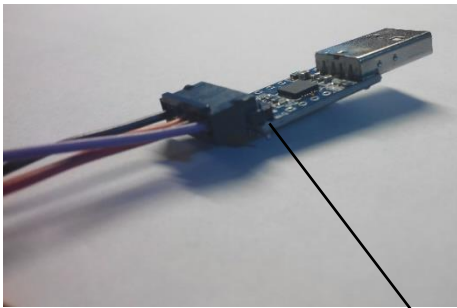
The Converter board must be plugged in before doing this!



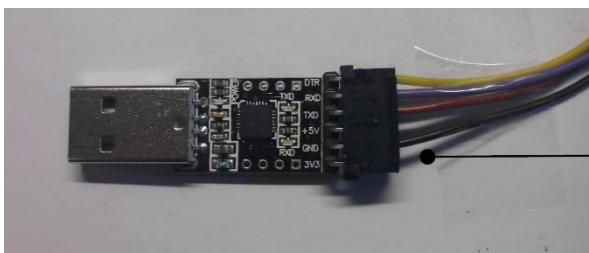
Now Connect the Convert Board to our Arduino Board. We use the given Relimate Wire to do this.

On our Arduino Board this wire goes inside only on way. On the Converter Board there are bergstrips, so the Relimate can be inserted both ways. Hence care must be taken. Look at the way we plug in the Relimate.

Basically the flap on the Relimate should be downwards.

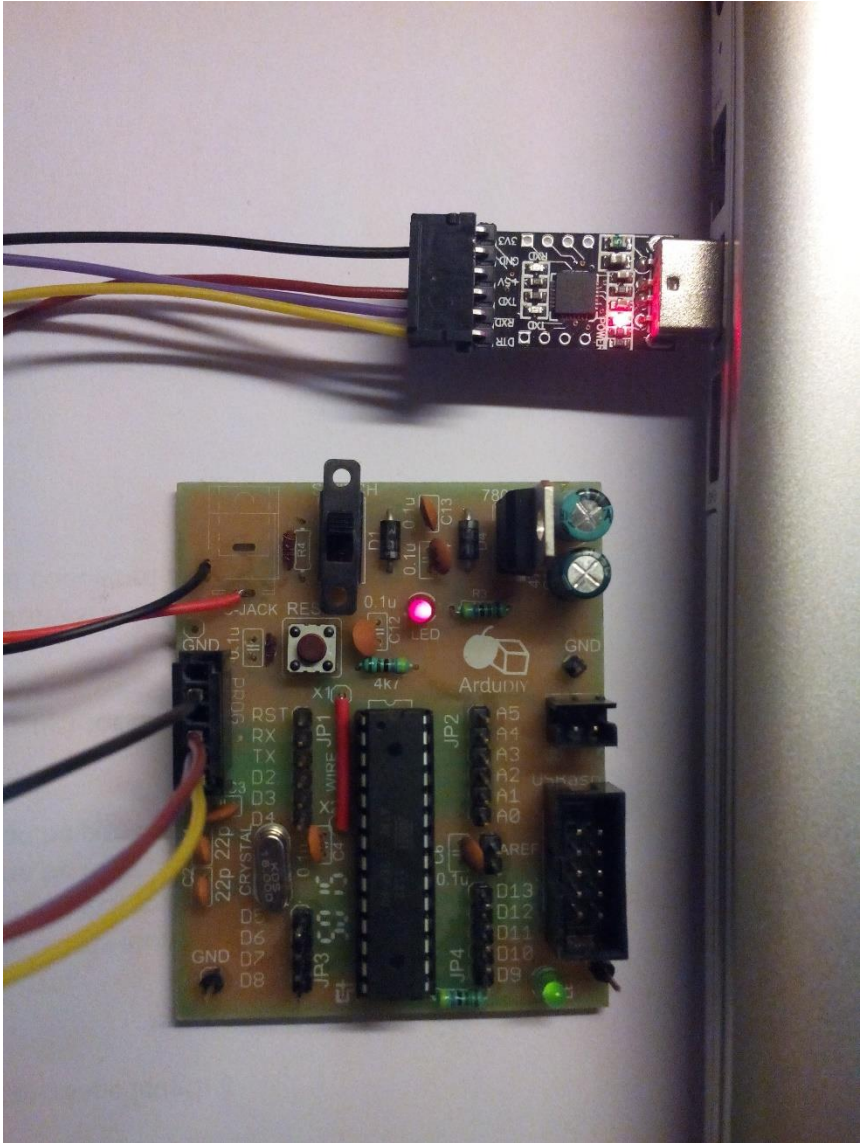


Flap



Observe here that the Black wire corresponds to GND on the CP2102 converter board.

Once you have connected the Relimate, the whole arrangement should look like this:



Just a little tip: when you want to remove the Relimate wire from the Arduino board, do not just pull it out. Doing that will reduce the life of your Relimate. The proper method to remove the Relimate is using your finger pull the Relimate in the direction of the flap while pulling it upwards. It should easily come off this way.



## Let's start Programming!!!

You must know about “Hello World” it is the first program we try out whenever learning a new language.

“Blink” is the Arduino equivalent of Hello World. What this code does is it starts blinking the connected led of the Arduino Board. Let us burn this Code onto our ArduDIY.

Go to File -> Examples -> Basics -> Blink

This shall open the code in a new Arduino Window.

Now Click on the Verify Icon. What this does is it compiles the code (sketch). If there are any errors in the code it does not compile and we get an error.

A sketch can be compiled even when the Arduino Board is not connected.

Now Click on the Upload Icon. This again compiles the code but now after compiling, the code is burned onto the ATmega328 of our board.

You will get a Done Compiling message. If there is some error look at the troubleshooting section.

If everything works correctly the Green LED on our board should start blinking on/off.

Congratulations, you just ran your first code on the Arduino!!!

This is just the beginning and the sky is the limit!

Everyday people are doing something new using the Arduino, take inspiration from them, and join them!

Keep in mind you will have to periodically change the Battery. Change it immediately if the board stops working or LEDs start glowing dim. The battery voltage should never drop below 7V.

# Arduino Basics

---

Arduino has been designed such that it is very easy to learn for new users. Before starting just brush up on the basics of C/C++. This is essential to understand the Arduino code.

The main important concepts you must cover are: syntax, data types, functions (in detail), condition statements (if-else, while, do while etc.), structures, classes (for advanced users and library manipulation). Most of the Arduino code is created using structures and classes. Hence we need proper knowledge of these and also how to access them (for example the dot operator used to access functions of objects of a particular class).

Once you know C/C++ you will understand part of the code immediately when you read it. A big help is that Arduino functions are very well named. Just take care which letters are capital and which small. Let us look at the code you just ran.

```
/*  
Blink  
Turns on an LED on for one second, then off for one second, repeatedly.
```

This part defines the code example. And gives a brief explanation.

Most Arduinos have an on-board LED you can control. On the Uno and Leonardo, it is attached to digital pin 13. If you're unsure what pin the on-board LED is connected to on your Arduino model, check the documentation at <http://www.arduino.cc>

This example code is in the public domain.

```
modified 8 May 2014  
by Scott Fitzgerald  
*/
```

Before we use any pin, we must first define how we are using it. Here we define a digital pin as OUTPUT

```
// the setup function runs once when you press reset or power the board  
void setup() {  
  // initialize digital pin 13 as an output.  
  pinMode(13, OUTPUT);  
}
```

Once we define the pin we can use it.

```
// the loop function runs over and over again forever  
void loop() {  
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)  
  delay(1000);             // wait for a second  
  digitalWrite(13, LOW);  // turn the LED off by making the voltage LOW  
  delay(1000);             // wait for a second  
}
```

When you read this example code you will see how well it is commented. Arduino code examples are the best way to learn Arduino.

One approach to follow while learning Arduino is, first decide what you want to do (this depends what you was running using the Arduino) then search for code example available. The examples folder in Arduino itself contains a lot of code samples. Understand these code examples and

then write your own code. DO NOT copy paste. Understand and write it yourself.

The coding convention followed by Arduino is a bit different. It is used throughout Arduino and if you are going to publish your code online it is recommended that you stick to the Arduino convention of writing. The most important thing in Arduino is **PINS ARE NOT ACCESSED BY THE ACTUAL PIN NUMBER OF THE MICROCONTROLLER**. Those who are familiar with embedded coding will remember that we always used to access registers (PORT, Data Direction) using the actual pin numbers. Instead the Arduino developers have come up with its own pin mapping and naming configuration. This widely divides the pins as Digital and Analog. Now we don't have to use registers at all, Arduino takes care of this for us. Digital pins are those which can input and output digital values. Along with this they also have pin specific functions. PWM pins are a part of digital pins. Please note that while using PWM, you will have to use only those pins which are capable of producing a PWM output. The analog pins are those which have ADC (Analog to Digital Converter Capabilities). A thing to note here is that all Analog pins can be used as digital pins but digital pins cannot be used as analog pins.

Another thing you will notice when you start reading Arduino code is the efficient use of macros and identifiers. So suppose you are using a red coloured hitswitch in your project, now this hitswitch is connected on pin 7 on the Arduino. So when we write the code whenever we want to access anything related to this hitswitch we use it like

```
digitalRead(7);
```

```
or pinMode(7);
```

Obviously we will use these functions multiple times at different places in our code. Later we want to use pin number 7 as a Led indicator. Now what? We will have to sit and change every instance of 7 to something else. That is a pain. So what we do in the beginning define the pin number eg;

```
int red_hitswitch = 7;
```

Now wherever we want to access this pin we just replace it with red\_hitswitch.

```
digitalRead(red_hitswitch);
or pinMode(red_hitswitch);
```

This makes reading and writing the code easier and now even if we want to change the pin assignment in the future we just change the definition and nothing else.

This is the pin mapping Arduino uses:

## ATmega168/328 Pin Mapping

Arduino function					Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MISO, MOSI, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Let's go over the basics of what you will require to get started. Now for a microcontroller as we saw earlier, every pin has multiple functions. Some of the important ones are Digital Pin (Input or Output), Analog to Digital Conversion, PWM. Arduino has specific functions defined for all of these. Let's look at them.

`pinMode (Pin_Number, INPUT/OUTPUT)` – This defines the pin as either an input or output. This has to be done prior to using that pin.

`digitalWrite(Pin_Number, HIGH/LOW)` – This is the function to write a digital value to a pin. Digital value means that the voltage seen on the pins can only be high (5V) or low (0V).

`digitalRead(Pin_Number)` – Through this function we can read a voltage level on the given pin. As this is digital the value can be only HIGH (5V) or LOW (0V). This function returns the value which is read hence it must be assigned to a variable.

e.g.; `buttonState = digitalRead(buttonPin);`

`analogWrite(Pin_Number, PWM_Value)` – The analog write function lets us output PWM on a particular pin. Please note that only pins with PWM capabilities can be used.

On the Arduino UNO (and our board) these are **(3, 5, 6, 9, 10, 11)**.

`analogRead(Pin_Number)` – This is the function to read the analog value present on the ADC pins of the microcontroller. This function returns an integer value hence it must be assigned to a variable.

It only works with the ADC pins i.e. (A0-A5).

e.g.; `sensorValue = analogRead(sensorPin);`

These are just the very basics. Everything about using Arduino is available on the net. Let google be your teacher. If you have any doubts or queries, just google it.

# Troubleshooting

---

Ok, it happens many times in electronics that first time things don't work out as we planned them to. At such times remember persistence is the key. NEVER GIVE UP. You will find the problem. Just keep in mind, you will learn a whole lot more while debugging.

A lot of care has been taken in designing this whole kit so reliability is high, but due to the nature of this it may happen that things could go wrong in the components or the DIY process. Stop worrying and let's start debugging.

Debugging is a step by step process. Start with all the known problems, then try to find one key problem and work on a solution. The most irritating problems are when multiple issues overlap and we don't know what is causing what. At such times one must keep cool, and work patiently one problem at a time.

## Rule No. 1:

Try to turn the whole thing off, close the program, wait for some time and turn it on again.

Believe me this works. Mainly in the Arduino IDE and programming.

## VCC-GND show a short circuit:

1. Have you checked holding the DMM for more than 30sec?
2. Visually Check there is no solder blob or spot on the board which is shorting it.
3. Check with the reference Diagram that all soldered components have correct polarity and are in the proper position.

Power LED (red) does not glow when you switch on the board:

1. Check Battery Voltage.
2. Have you performed a dry test? Check VCC-GND again.
3. See if the Voltage Regulator (7805) is getting an input voltage (should be around 6V to 8V) and if there is 5V output voltage present on the output terminal of the 7805.
4. If there is 5V present on the output, check if the LED is soldered correctly along with the resistor on the side.
5. If there is no voltage at input, check all diodes, resistors and the battery connector.

### CP2102 Driver Problems:

1. Are you getting the COM port?
2. Are the drivers you installed correct? Check the link. We need CP210x Drivers.
3. Have you installed any third party software that may interfere with installation.
4. Remove all other USBs and try again.
5. Try again on a different PC.

There is a test known as echo test to check if the converter board is running correctly.

Short the TX and RX on the Converter Board. Now connect it and use some serial monitor or terminal software like “RealTerm” or “TerraTerm”. Now open this and select the COM port of the converter board. Now if the board is working correctly any text you type should be displayed on the screen. And the boxes beside TX and RX should simultaneously glow (Realterm).



## ATmega328 not getting programmed:

1. What is the error message?
2. Is the ATmega328 a bootloaded IC? The one with the sticker provided with this board is previously bootloaded. Any other IC from the market will not be bootloaded.
3. Is the Converter Board and the Arduino board connected properly (look again at orientation of the connector)?
4. Is the reset switch properly connected? Check voltage on pin number 1 when the board is in on state. If it is anything other than 5V the board is continuously being reset.
5. Check if the converter Board is working properly (using the method mentioned above).
6. Are the 22pF capacitors on the board of correct value and are they soldered correctly.
7. Check if the supply pins of the IC are getting 5V.
8. Check for proper soldering of capacitors C1, C2, C9 and resistor R2.
9. Try replacing the crystal oscillator.

### **Does the Arduino Software get stuck in “Uploading” when you start to upload the code?**

#### **Note that big codes need time to get uploaded.**

Your Arduino got programmed the first couple of times then suddenly stopped getting programmed. The LEDs on the Converter board just blink for a second when you press upload and then nothing?

This is a common problem when there are issues with the bootloader.

It occurs due to errors while programming an IC (loss of connection during programming) or if you program using some other source (external programmer).

The solution to this is to burn the bootloader on your ATmega328 again.

**Please Note:**

ATmega328 from the Arduino Uno will not work directly on this board and vice versa. You will have to burn the bootloader again when changing the ATmega in between these two boards. More on this is given later.

### Peripheral of ATmega328 not working:

1. Is the example code “blink” working or did it work the first time.
2. Check the code.
3. The pins of the IC may have been damaged due to incorrect use.
4. Try another bootloaded IC (a one with a sticker).

### Code compilation Errors:

1. Go through the code to find any syntax or logical errors.

The Arduino console is not very great at identifying errors in the code and will usually return something in a complicated language.

A great help is the line number where the error occurs is usually shown by the console.

Many errors we make are silly mistakes while typing.

2. A good way to identify the problem is to google it and look at forums.

There has to be someone in the world who has had a problem similar to yours.

You may not find the exact same problem as yours, but there should be something similar.

Look at all the available solutions and see if you can adapt them

to your problem in any way.

If you find nothing, post your problem on the forum. There are a lot of people willing to help.

# The Arduino Bootloader

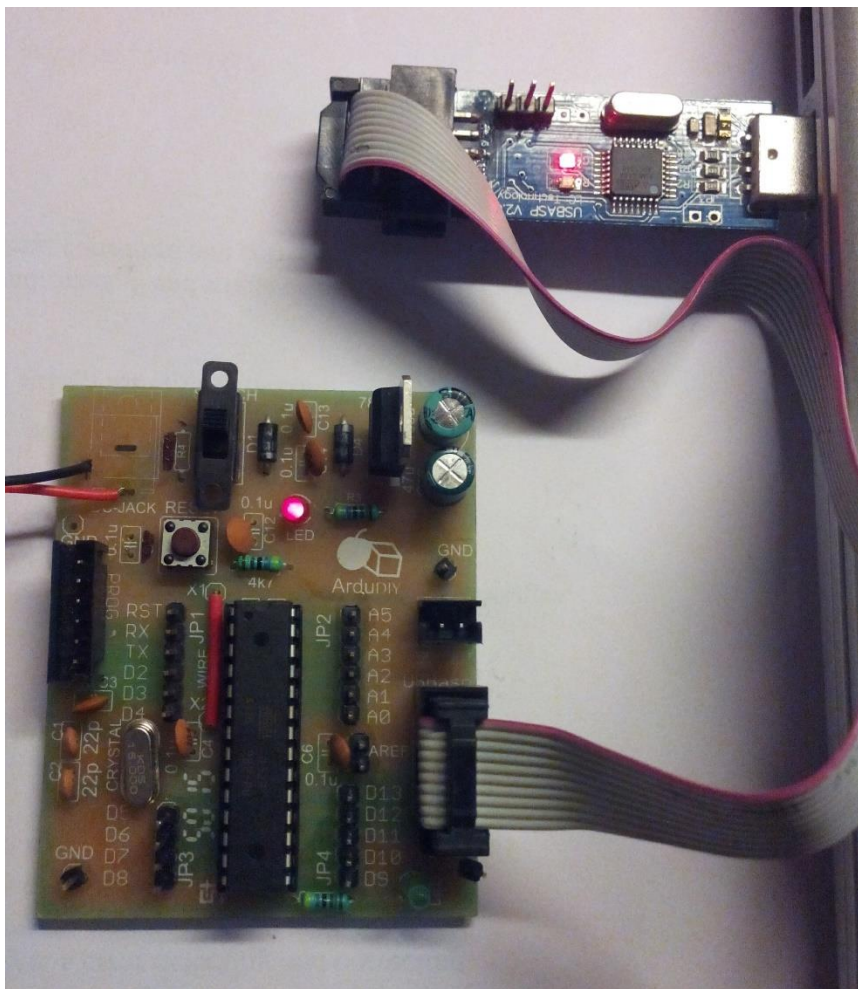
---

The bootloader is basically a .hex file that runs when you turn on the board. It is very similar to the BIOS that runs on your PC. It does two things. First, it looks around to see if the computer is trying to program it. If it is, it grabs the program from the computer and uploads it into the ICs memory (in a specific location so as not to overwrite the bootloader). That is why when you try to upload code, the Arduino IDE resets the chip. This basically turns the IC off and back on again so the bootloader can start running again. If the computer isn't trying to upload code, it tells the chip to run the code that's already stored in memory. Once it locates and runs your program, the Arduino continuously loops through the program and does so as long as the board has power.

If you are building your own Arduino, or need to replace the IC, you will need to install the bootloader. You may also have a bad bootloader and need to reinstall the bootloader. There are also cases where you've put your board in a weird setting and reinstalling the bootloader and getting it back to factory settings is the easiest way to fix it. We've seen boards where people have turned off the serial port meaning that there is no way to upload code to the board, while there may be other ways to fix this, reinstalling the bootloader is probably the quickest and easiest.

We use a dedicated external programmer the USBasp for burning the bootloader. A port for direct connection with the USBasp has been provided on the ArduDIY. You can also use this to upload code to the Arduino using the "Upload Using Programmer" option.

You will need to buy an USBasp separately. Don't worry its widely available.



# Making Your Own Arduino

---

## A few initial notes:

Ok now that you are armed with a fair amount of knowledge on Arduino and its working, let's get to the part where you make your own.

Use the schematic given above as a reference.

The essential components for the Arduino are power supply block, power supply connection to micro controller, crystal oscillator circuit and reset pull up. Do not change without proper study.

With the rest, feel free to tweak it as per your application. This will give you a greater control over the PCB, and also make the whole thing smaller.

If you are using a new IC brought directly from the market, first burn the bootloader using a programmer like the USBasp. A port for the USBasp has been provided on the ArduDIY board. You can use this to burn the bootloader and then use the IC elsewhere.

For programming, you can use the programmer or any USB to Serial Converter like the one provided in this kit. Just make sure the connections are correct.

USBasp Programmer



### Step 1:

Identify what all you shall need. Think of your complete application in detail, this includes all the pins, ports and other components.

Now assign proper pins as per their use e.g. (GPIO, UART, PWM).

As a good practise always keep extra pins for future expansion and give multiple ground pins.

### Step 2:

Now add the circuits for power supply, crystal oscillator and pull up for reset.

For now just add blocks on paper with the relative location as to where you shall be placing them.

### Step 3:

Now think of the actual routes or connections. If you are using a software like eagle, go ahead with the board. If you are going to make this on a general purpose PCB (O-PCB) then it is recommended that you think about the routes first on paper.

### Step 4:

Start fabrication.

Once you are done with fabrication, it is highly recommended that you perform proper testing as given in this booklet.

### Step 5:

To run Arduino you shall need an IC with the Arduino bootloader burnt. Now you can directly buy a IC with the Arduino bootloader pre-programmed or you can burn it yourself.

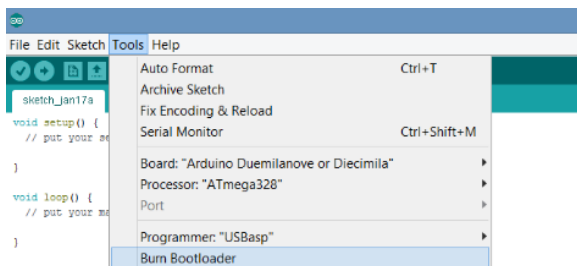
If you want to avoid the trouble of bootloading the IC yourself, just buy a bootloaded IC from me.

To do this you shall need a dedicated programmer such as the USBasp. The ArduDIY board provides a port to which the USBasp can be directly connected thus removing the need for jumpers.

To use the USBasp you will have to install its drivers. They are available on [www.fischl.de/usbasp/](http://www.fischl.de/usbasp/)

Installing drivers for Windows8 and above is slightly complicated. The complete procedure is given on <http://letsmakerobots.com/node/36841>. This link is also available on the USBasp driver page at the fischl website. It has been tried and works.

Once you install your drivers, just connect your USBasp to the Arduino and your PC, then use option burn bootloader.



Before burning the bootloader select the board on which you are going to use the ATmega. You can also make ATmega328's meant for the Arduino Uno by selecting Arduino Uno under Boards before burning bootloader. For IC's meant for the ArduDIY, just select the same



options you use for programming i.e. Arduino Duemilanove and processor ATmega328

The programmer leds will flash and in some time you will get the done burning message. Now your ATmega is an Arduino. You can now remove the IC from your ArduDIY board and use it on any other custom board you have designed.

# Future Work & Some DIY Projects

---

A few projects to start with which you may find interesting...

## **Arduino with Keypad Input, Display**

Being able to reliably use a keypad goes a long way in all DIY projects

## **Battery Tester**

Use the built in ADC capability of the Arduino to test any battery below 5V.

## **LED Cube**

This is one of the most interesting projects you can take up. You will learn a lot of basics of electronics and programming.

For starters you can try a 4x4x4 cube and then scale up.

All required components will be easily available at electronics stores.

## **Water Level Indicator**

You can use existing sensors by interfacing them with your Arduino or even build your own sensors. This will be a good practise of electronic circuit skills.

These are just some random ideas that came to my mind. There are literally thousands of interesting projects out there with the Arduino.

*Explore*

*Make*

*Create*

# Disclaimer

---

This product ArduDIY is provided as is without any guarantee or warranty. In association with this product Sugar Plum Labs takes no responsibility for any damage caused while using or soldering of the product by the customer himself. Also in case of damage or particular defect in the above mentioned product there shall be no refund allowed unless after a check on individual case basis within one week from date of purchase.

## Note:

As above mentioned regarding the defects or the damage, the main components inclined towards the said things are passive which do not damage easily and are replaceable. The only main active device is the microcontroller and that has been checked prior to packing under the knowledge of individual experts related with the above mentioned product.

