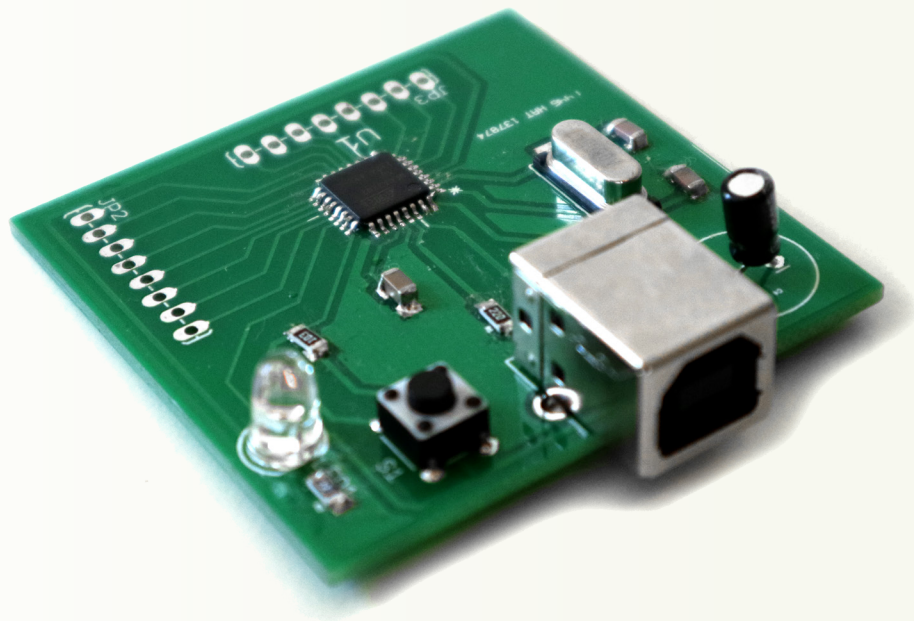


Building A Microcontroller Board From Scratch



By: Øyvind Nydal Dahl

Learn to build a microcontroller board with USB
that you can make at home using
standard hobby tools.

INTRODUCTION

This is a five-part microcontroller tutorial series that I wrote for [my blog](#) to show what it takes to build a microcontroller circuit from scratch.

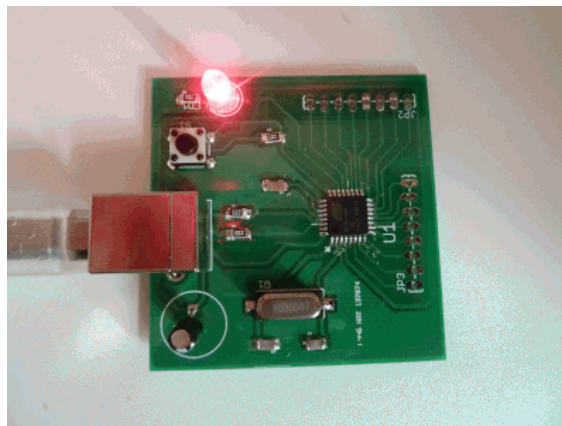
Throughout the tutorial, I will show you the steps you need to take to build your very own microcontroller circuit. You will then be able to use this circuit to build a blinking lamp, a robot, an automatic cat-feeder or whatever idea you want to build.

My goal is to make a circuit that is as simple as possible, and which requires no external programmers or debuggers. You should be able to just plug it into a USB port on your computer and program it.

I have not planned this out in any way. I am just going to build it, and write about the process. Hopefully we'll end up with a usable circuit.

In the part of the microcontroller tutorial, I'll start from scratch. I want to explain what a microcontroller is, in very simple terms. I want to get everyone on board before we dive into making the circuit.

At the end, you'll learn to build this:



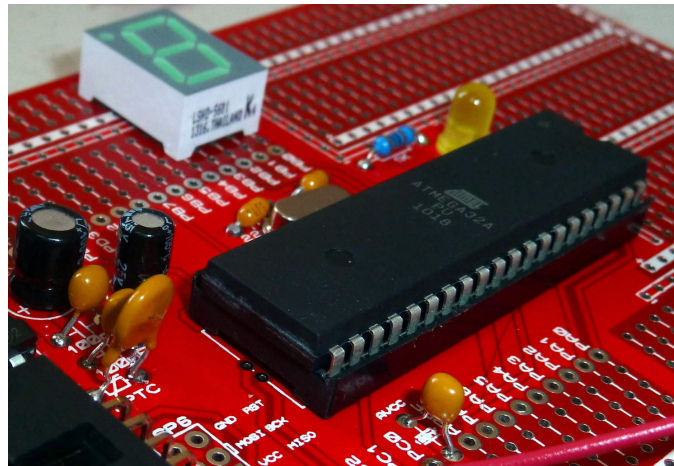
Part 1:

What Is A Microcontroller?

I loved learning about microcontrollers when I was studying. It meant I could start taking advantage of microcontrollers in my electronics projects. It felt like with that this knowledge, I was unstoppable. I could build anything!

And it is true. Microcontrollers are powerful components. They let you write programs to control your electronics. Combine this knowledge with building your own circuit boards, and you can make amazing things.

By using a microcontroller in your project you will have access to a vast amount of functionality from the tips of your (programming) fingers.



You can think of a microcontroller like a tiny computer. You can connect things, like a small display, some buttons, a motor and some sensors. And you can put programs onto it and run them.

What Can You Do With A Microcontroller?

Oh, where do I begin?

There are so many things you can do with a microcontroller. You could build a robot. Or an MP3-player. Or a cellphone. Or a door-lock that unlocks your door automatically when you enter a code on your smartphone.



Recyclomaten: A recycling machine controlled by a microcontroller circuit I designed.

The possibilities are endless!

Let's say you want to build a robot. You can connect an infrared sensor to use as vision for the robot. And you can connect a motor with some wheels to make it move.

Now, all you have to do is to make a program that reads from the infrared sensor and controls the motor. In your code, you can make sure the robot stops if it sees something in front of it, and make it turn to either left or right before continuing.

When you know how to build microcontroller circuits, there are almost no limits to what you can do! And by following this microcontroller tutorial, you'll learn to use microcontrollers in your own projects.

A Closer Look At A Microcontroller

A microcontroller has several pins. The simplest ones have around 8 pins, while more advanced microcontrollers can have hundreds of pins. Most of these pins are so-called input and output pins. And by using these pins, the microcontroller can interact with the outside world.

The microcontroller doesn't do anything by itself. You need to tell it what to do, by making a program that you load into it. This is called *programming* the microcontroller.

From the program you write, you can control the input and output pins. So if you connect for example a [Light-Emitting Diode \(LED\)](#), you will be able to switch the light on and off from your program.

An input pin could be used to check if a button connected to it has been pushed. Or to read the temperature from a temperature sensor.

In your program, you will be able to make decisions based on the input. So you can make a program that will start to blink a light if the temperature goes above or under a certain level.

Put this into your beer-brewing room and you will get a visual alarm if the temperature for brewing is not right.

Programming a Microcontroller

Programming a microcontroller can seem a bit tricky because there are many confusing choices to make. I remember how I felt in the beginning. With all the available compilers, IDE's, programmers and programming methods – no wonder you get confused!

So let's break it down. These are the three steps necessary to program a microcontroller:

1. Write code
2. Compile your code to machine code
3. Upload the machine code to your microcontroller

What exactly to do at each step varies from microcontroller to microcontroller. But don't worry – I'll be guiding you through the exact steps needed when we get there.

Next Step

It's time to find a microcontroller and get to work. Finding a microcontroller isn't necessarily as easy as you would like it to be. There are probably 58 billion different ones.

Ok, maybe a little less.

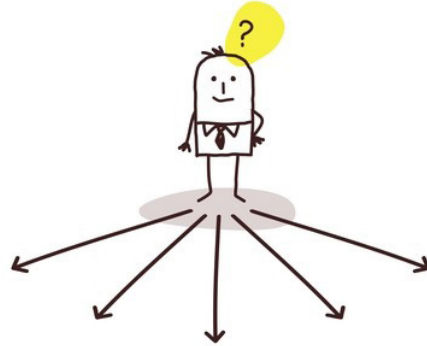
But I have some tips up my sleeve that will make it easier to choose. That's what you'll learn in the next part of this tutorial.

If you have any questions about what you've learned in this part, head on over to the following page and write your question in the comment section:

<https://build-electronic-circuits.com/microcontroller-tutorial-part1>

Part 2: How To Choose a Microcontroller

In the previous part of the microcontroller tutorial series, you learned the basics of microcontrollers. The goal of this tutorial is to show how to build a microcontroller circuit that is as simple as possible. So simple that you can make it at home.



Next up in this tutorial is choosing a microcontroller. This can be confusing! At least if you don't know what you are doing. But after reading this second part, I hope it will become clear.

The Differences Between Different Microcontrollers

There are a gazillion different microcontrollers. But what is really important for your project?

When you have the answer to this questions – everything becomes much simpler. So let's look at some of the main differences between microcontrollers.

Number of bits

You can find 8-bit, 16-bit and 32-bit microcontrollers. These numbers refer to the size of the databus. In simple and practical terms, a larger databus can do more heavy calculations.

The 8-bit microcontroller is the most commonly used by hobbyists. In general, the 8-bit microcontroller has fewer pins, so that it's easier to solder. And it is usually easier to program too.

In this tutorial, we'll be using an 8-bit microcontroller.

Memory, IO and peripherals

Different microcontrollers have different amount of memory, number of Input/Output (IO) pins and peripherals.

Peripherals are extra functions added to the microcontroller. It could be Analog-to-Digital conversion, USB interface, PWM, or SPI communication. If you are not familiar with these terms, don't worry – you'll learn them when you need them.

AVR vs PIC

The two most common microcontroller-brands for hobbyists are probably AVR from Atmel and PIC from Microchip. AVR is the type of microcontroller used on the [Arduino](#).

I have used AVR a lot and I think it's a really good choice of microcontroller. The PIC is said to be really good too, but I have been so happy with the AVR that I never actually tried the PIC.

Finding a Microcontroller For Your Circuit

Now it's time to start making some decisions. Should you go for the AVR from Atmel or the PIC from Microchip?

Since I have used the AVR many times before, I'm just going to go for AVR again. I know this will save me time and energy. This is very common among electronics designers – to make their decision based on what they have experience with.

Writing Down Our Requirements

Ok, we have narrowed it down to 8-bit AVR chips. What else do we need?

My goal with this, is to build a microcontroller circuit that is as simple as possible. I want a simple circuit that I can plug into the USB of my computer to program it. So these will be our requirements:

- Programmable through USB
- As few components as possible
- Possible to solder at home

There are several ways to program the microcontroller through USB.

A common method, used on some Arduino boards, is to add a "USB to serial"-chip to the circuit. The problem with this approach is that it increases the number of components on the board.

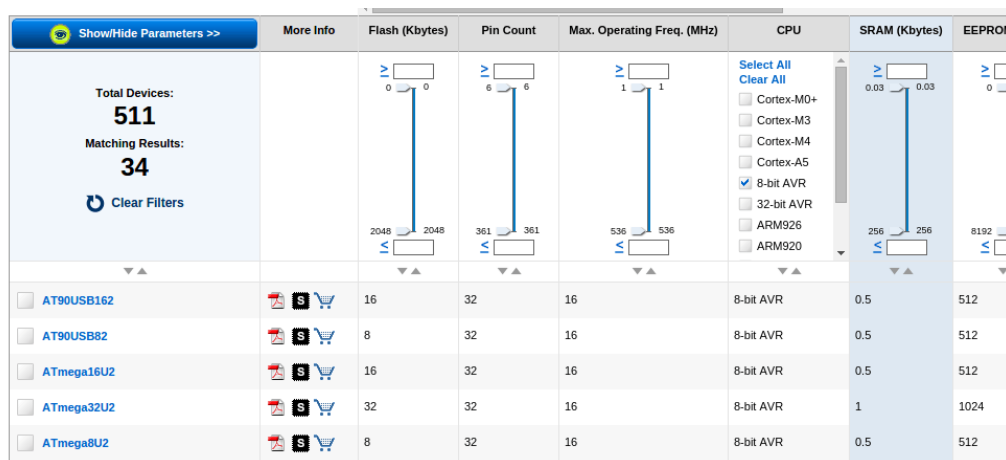
Another method, is to find a microcontroller that has USB interface integrated onto it. Since this matches our wish for fewer components, we'll go for this option. Specifically, we need a USB device interface.

And we have to make sure the microcontroller comes pre-loaded with a bootloader that makes it possible to program it through USB.

The last requirement is that it should be possible to solder the circuit at home. So we want to find a microcontroller with as few pins as possible. Less pins = easier to solder.

Using the Microcontroller Selector

Atmel have made a very useful tool for selecting a microcontroller ([find it here](#)).



	Flash (Kbytes)	Pin Count	Max. Operating Freq. (MHz)	CPU	SRAM (Kbytes)	EEPROM
Total Devices: 511						
Matching Results: 34						
Clear Filters						
AT90USB162	16	32	16	8-bit AVR	0.5	512
AT90USB82	8	32	16	8-bit AVR	0.5	512
ATmega16U2	16	32	16	8-bit AVR	0.5	512
ATmega32U2	32	32	16	8-bit AVR	1	1024
ATmega8U2	8	32	16	8-bit AVR	0.5	512

All we got to do is to insert our requirements:

- In the «CPU»-field – we select only the «AVR 8-bit»
- In the «USB»-field – we select only the ones with «USB device»

Then, we sort our results by pin count.

We get 5 results that has 32 pins. None with less pins. So we'll narrow it down to these 5 options.

But there is one more requirement that we need to take into account: For us to be able to program the chip through USB, the chip needs to be pre-programmed with a bootloader from the factory. The bootloader is called DFU Bootloader.

Information on which ones are pre-programmed with the DFU bootloader was a bit hidden. But I found a document which lists the pre-programmed devices on the right. From [this document](#), we can see that the Atmega8U2 does NOT have the bootloader pre-programmed.

So that leaves us with 4 options.

Making a Decision

Since all of these seem to fit our requirements, it doesn't matter which one we choose. So let's choose the one that has the most amount of flash. More flash means that we can store bigger programs on the chip.

Atmega32U2 with 32kB of flash is the winner!

Before we move on, it's a good idea to check that the chip is available in our preferred store. I usually buy all my stuff online from [one of these stores](#).

And at the moment, I can see that the chip is available several places.

Next Step

Now we have chosen a microcontroller and are ready to start designing our circuit. Because of the microcontroller we chose, this might turn out to be a pretty simple task. But we'll see about that in the next part...

If you have any questions about what you've learned in this part, head on over to the following page and write your question in the comment section:

<https://build-electronic-circuits.com/microcontroller-tutorial-part2>

Part 3:

How To Design a Microcontroller Circuit

In the first part of this tutorial we looked at what a microcontroller is. We saw that a microcontroller is like a small computer, and that you can use it to build amazing things like cell phones or even your own handheld game-console.

Then in part two, we looked at different types of microcontrollers, and we chose one for our purpose. We chose the [ATmega32U2](#) because we can program it through USB and it is reasonably easy to solder at home.

Now, we are going to design our circuit from scratch. So we need to design a circuit diagram with all the components and connections necessary to make our circuit work.

Let's build this thing!

What Do We Need?

How to decide which components to connect to our microcontroller?

Let's think about it. We need power for the chip, otherwise it won't work. We need a USB connection, because we are going to program it through USB. And we need some physical pins where we can easily connect things to our circuit and test it, like an LED.

So, we need:

- Power
- USB Connection
- Pin Connections

The Datasheet

To figure out exactly which components we need, and how to connect them, we look in the [datasheet of the chip](#). The datasheet is a comprehensive document with a lot of technical data on how the microcontroller works and how you can control different parts of it.

If you are not used to reading a datasheet, it might feel a bit overwhelming. But after looking at a few datasheets, you will sooner or later start to understand how it is laid out.

It is not necessary to read the datasheet from start to the end. You only need to read the parts that are interesting for what you want to make. So if you want to make a timer with your microcontroller, you read the timer section. If you want to use the UART, you look up the UART section.

One thing to notice – in the datasheet of the ATmega32U2, the table of contents is placed at the end.

Power and USB

We can choose if we want the circuit to be powered by the USB cable or with an extra power cable.

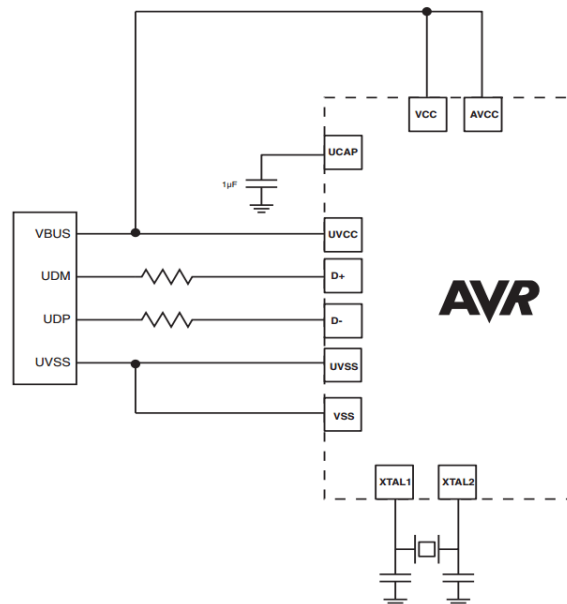
To keep things simple and with less components, let's make this a USB powered device. This means it will only work when it is connected to the USB.

Which components do we need to do this? To find out, let's look in the USB-section of the datasheet, and see what we find.

USB Module Powering Options

From the table of contents at the end of the datasheet, we find

a section called USB Module Powering Options at page 186. Here we can find the following figure that show us how to connect the USB section of the chip to make it powered by USB.



*How to connect the ATmega32U2 as a USB powered 5V device.
(Source:Atmel.com)*

From Design guidelines on page 189 we learn that the resistors should be 22 Ohm resistors, and that it is also highly recommended to have a 10 μ F capacitor on the VBUS line. So let's add the 10 μ F capacitor too.

Crystal

In the image above, we can see that there is also a crystal and two capacitors connected to the XTAL1 and XTAL2 pins. Why is this?

A microcontroller needs a clock to work. Most microcontrollers have an internal RC-oscillator that creates a clock signal. And so do the ATmega32U2. But the thing is that the USB part of the microcontroller is not able to run from that internal clock. So to make USB work, we need an external crystal.

Exactly what kind of crystal we need, we can find at page 30 in the datasheet. Here, the capacitor values are also listed.

Pin Connections and LED



We want to be able to connect stuff to our microcontroller circuit. So, we'll add 16 physical pins that are connected to the I/O pins PBo-PB7 and PDo-PD7. This makes it easy to connect something to our circuit.

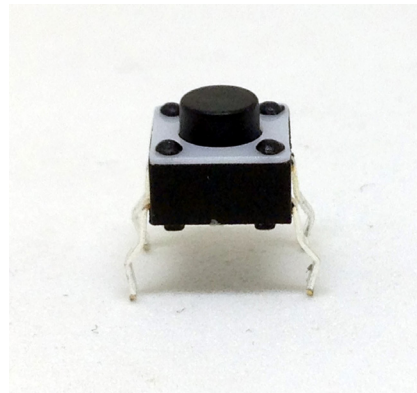
Another thing we should add, is an LED. This way, we can easily check if the circuit and our code works. To control the current through the LED, we also add a [current limiting resistor](#) in series with it.

Push-Button for Reset

It is very common to have a reset button on a microcontroller circuit. This makes it easy to reset the microcontroller without having to unplug the USB cable. From page 47 of the datasheet we can read:

"The MCU is reset when a low level is present on the RESET pin for longer than the minimum pulse length".

So we need to make sure the reset pin is normally pulled high, and only pull it low when the reset button is pushed.



*A standard pushbutton
with four legs*

Which value should this resistor have? A good value is around 10k Ohm.

If we combine all the parts that we've discussed above, we'll end up with the following circuit diagram.



Next Step

It's all coming together now. All we have to do now is to create a circuit board. So that's exactly what we'll do in the next part of the tutorial.

If you have any questions about what you've learned in this part, head on over to the following page and write your question in the comment section:

<https://build-electronic-circuits.com/microcontroller-tutorial-part3>

Part 4: Creating a Circuit Board

Let's get ready to design the circuit board!

We are now in the fourth part of the microcontroller tutorial. We have a microcontroller circuit diagram ready. And it's time to make a circuit board.

I love this part. This is the "magical" step that takes the idea we started with and turns it into something real.

But let's recap. So far we have learned:

- Part 1: What is a microcontroller?
- Part 2: How to choose your microcontroller?
- Part 3: How to design a circuit diagram for your microcontroller?

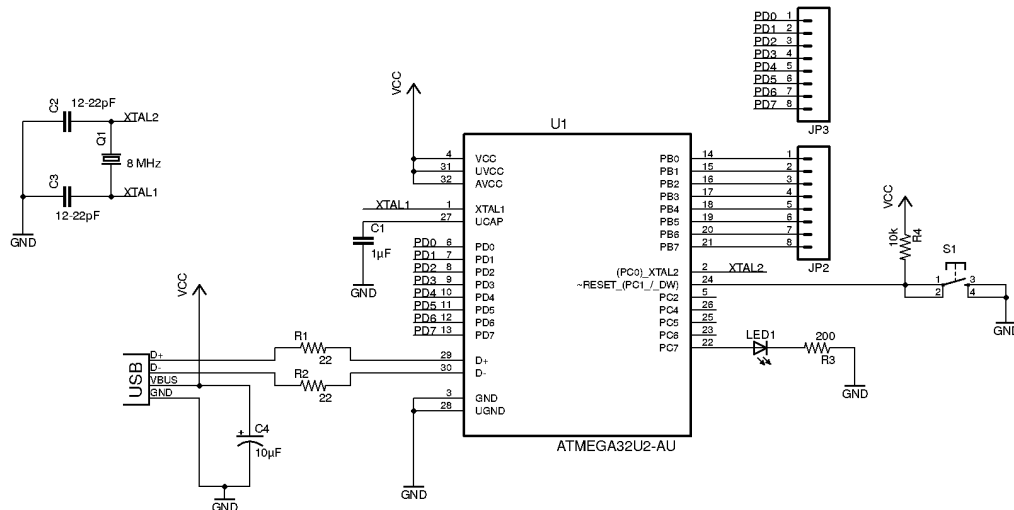
In this part we are going to create a circuit board for our circuit, then get this board created in one way or another. This can be done in several ways, as we'll see later.

To design our circuit board we'll use [Cadsoft Eagle](#). It's available in a free version and works on Windows, Mac and Linux.

Designing Schematics For Our Microcontroller Circuit

The first thing we need to do, is to put our schematic design into Eagle's schematic editor. If you are not familiar with this process, check out this tutorial: [How to create schematics with Eagle](#)

In the previous part of this microcontroller tutorial, we decided on which components to use and how to connect them. Here is the circuit diagram we ended up with:



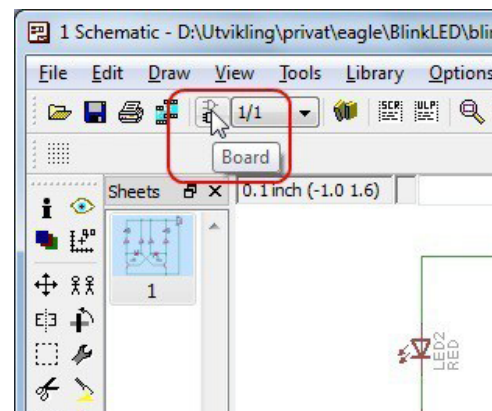
The ATmega32U2 microcontroller, is not in Eagle's default library. I could have designed my own custom component, but to save time I used a library that I found here: <https://github.com/civanovici/roduino/tree/master/eagle/eagleLibrary>

For the USB connector, I used one from Sparkfun's library here: <https://github.com/sparkfun/SparkFun-Eagle-Libraries>

Designing Our Board Layout

The next step is to design the board.

In Eagle, we can click on the «Board» button in the toolbar to open the design editor. If no board design exists for your schematics, you will be asked if you want to create one.



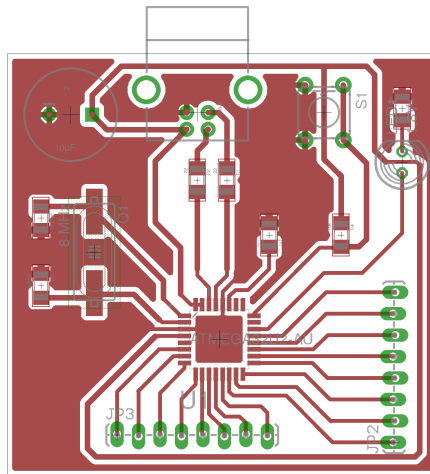
Answer «Yes» to this.

.....

I always start out by defining my board size. I know that I can get really cheap prototypes if I stick to 5cm x 5cm (1.9685 in x 1.9685 in), so I will set my board size to this.

Now it's time to place the components onto the board and draw the connections. In this design, I wanted to draw only on one side, so that it would be easier to mill or etch the board – just in case I wanted to do this later.

If you're not familiar with drawing a board layout, check out my PCB design tutorial here: <https://www.build-electronic-circuits.com/pcb-design-tutorial/>



.....

In my opinion, the easiest way is to use a PCB manufacturer. It takes a bit of time to wait for your board, but at least everything will be done properly. This does not have to be expensive. You can order 10 boards for \$10 at some places. And the cheapest shipping option can be as low as a few dollars.

I ended up ordering my boards from [seeedstudio.com](https://www.seeedstudio.com). Because I wanted to get my boards fast to continue the tutorial, I paid for the fastest shipping option. Hopefully I will get it very soon.

To see exactly how you can prepare an order of cheap PCB prototypes, check out [this video](#).

Getting Components

A board without components isn't very interesting. So we need to get our hands on some components too.

There are many stores that sell components online, [check out my recommendations here](#).

Complete Parts List

Part	Description	Value	Package
C1	Capacitor	1 μ F	SMD 1206
C2, C3	Capacitor	12-22pF	SMD 1206
C4	Capacitor Polarized	10 μ F	Through-hole
JP1	USB Connector	USB Type B Receptacle	Through-hole
JP2, JP3	Header 8 pin		Through-hole
LED1	Light Emitting Diode	1.8V	Through-hole
Q1	Crystal	8 MHz	SMD C49UP
R1, R2	Resistor	22 Ohm	SMD 1206
R3	Resistor	200 Ohm	SMD 1206
R4	Resistor	10k Ohm	SMD 1206
S1	Momentary Switch		Through-hole
U1	Microcontroller	ATmega32U2	TQFP-32

Now we just need to sit back and wait for the boards and components to arrive.

Next Step

In the next part we'll solder and program the circuit. Then we'll finally get to see if the circuit works or not. Let's hope it does!

If you have any questions about what you've learned in this part, head on over to the following page and write your question in the comment section:

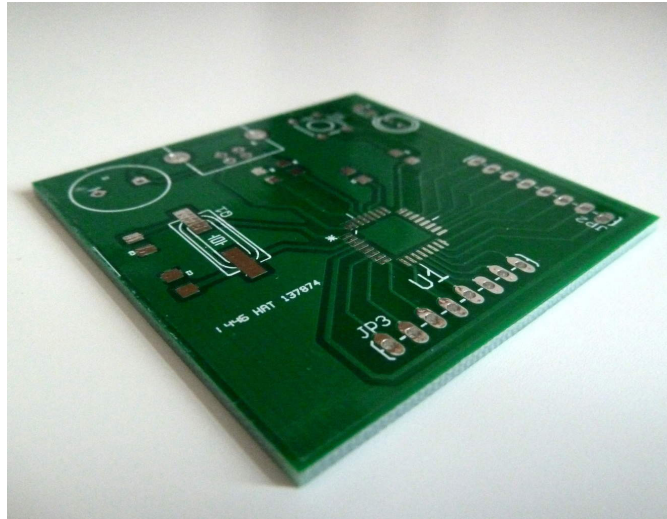
[https://build-electronic-circuits.com/microcontroller-tutorial-part 4](https://build-electronic-circuits.com/microcontroller-tutorial-part-4)

Part 5: Soldering and Programming

We are now at part 5, the final part of the microcontroller tutorial. Up until now we have learned:

- What a microcontroller is
- How to select a microcontroller
- How to design a circuit with the microcontroller
- How to create a circuit board from our circuit

I have just received the boards I ordered in the previous part, and today we are going to solder the board and program it.



The finished circuit board

Soldering The Board

To solder the board – I am going to use my old Ersa 30 soldering iron. The tip of it is a bit big, so it's really not the ideal tool to use. But it's what I have on my desk right now.

And it's also a way for me to show you that you don't need any fancy equipment to make this circuit. You can make this circuit at home.



Because I wanted to put everything on one side, I chose to use mostly SMD (Surface Mount Device) components when I designed the circuit board. So to solder this, I'm going to use the techniques from my [smd soldering](#) article.

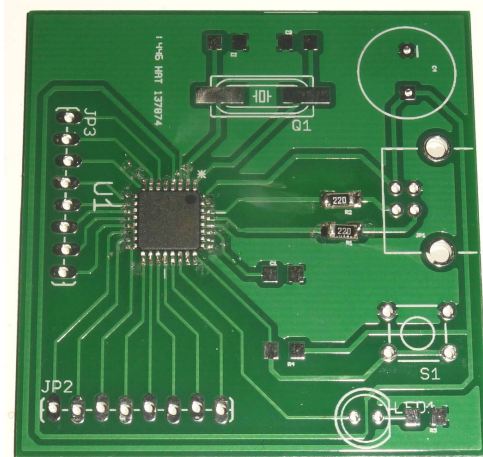
Soldering the Microcontroller Chip

Since the microcontroller chip was the most difficult thing to solder, I started out with that one.

First, I added some solder to one corner pad. I placed the chip carefully, using a pair of tweezers. And I made sure that all the pins were placed correctly over their pads. Then, while holding the chip in place with the tweezers, I placed the tip of my soldering iron onto the pin and pad of the corner where I already added solder –making the solder melt.

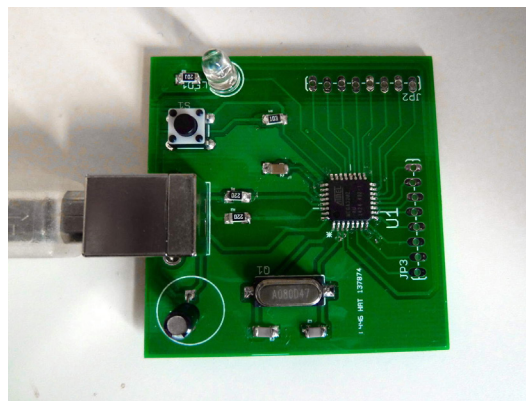
I removed the tip, and let the solder cool for a second. The chip was in place. Now, all I needed to do was to apply a bit of solder onto each of the pins, to make them stick to the pads on the board.

This was a clumsy process with the thick tip of my soldering iron. But, by keeping my cool and being patient – I was able to solder all the pins onto their pads. As you can see from the picture, soldering with the large-tip soldering iron was a bit messy. But it doesn't matter, as long as it works.



Soldering the Other Components

After I'd managed to solder the chip, the other components were easy. I might not have gotten them perfectly aligned, but it wasn't that bad either.



Testing the Circuit

The ATmega32U2 chip comes with a pre-programmed boot-loader that should make it appear as a USB device when plugged into a computer.

After everything was soldered, I *should* have inspected the solder joints closely with a [USB Microscope](#) or something. This is smart because if there are any short-circuits caused by a tiny little solder blob somewhere, it could damage the circuit.

But I didn't have one nearby – and I was super excited to see if it worked. I'm kind of like a child waiting to open my Christmas presents in this situation. I can't always make myself do what is smart to do. I'm just too excited to see if it works.

Instead I just looked extra carefully at the USB pins, to see if they at least seemed to be properly soldered. Then I plugged it into my USB-port and...

...nothing happened.

I was a bit disappointed for a brief second. Until I realized that nothing was supposed to happen. I only had one LED on the circuit, and it was connected to an IO pin. And at this point there was no code to control the LED.

So what I needed to check was if it showed up as a USB device on my computer. And it did! Wohooooo!!

Programming the Microcontroller Circuit

Now that I knew the USB-part was working, it was time to program the circuit with some code.

I've written about [microcontroller programming](#) before.

The process is:

1. Create program code
2. Compile code into machine code
3. Upload machine code onto board

Program code

To make a simple test, I created a blink-LED code. It does nothing more than blink the LED on the board.

Here is the code I used:

```
#define F_CPU 1000000 // The chip runs at 1 MHz as default
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRC = (1<<PC7); //Sets the direction of the PC7 to output
    PORTC = (1<<PC7); //Sets PC7 high
    while(1)
    {
        _delay_ms(500); //Wait 500 milliseconds
        PORTC &= ~(1<<PC7); //Turn LED off
        _delay_ms(500); //Wait 500 milliseconds
        PORTC |= (1<<PC7); //Turn LED on
    }
    return 0;
}
```

Compile the program

I saved the code in a file called blink-led.c . Then, I used a tool called avr-gcc to compile the code. Because I am using a Linux machine with Ubuntu, this is very easy to do (for Windows, check out [Win-AVR](#)):

First, install the application by opening a terminal window and typing:

```
sudo apt-get install avr-gcc
```

Then you can compile it to machine code by typing in these two commands:

```
avr-gcc -mmcu=atmega32u2 -Os blink-led.c -o blink-led.out
```

```
avr-objcopy -j .text -j .data -O ihex blink-led.out blink-led.hex
```

Now you have a file – blink-led.hex – that you can upload to the microcontroller.

You can find more information on the commands [here](#).

Upload Program Onto Board

To get the program onto the board I used [dfu-programmer](#). To install it, use the following command (on Ubuntu):

```
sudo apt-get install dfu-programmer
```

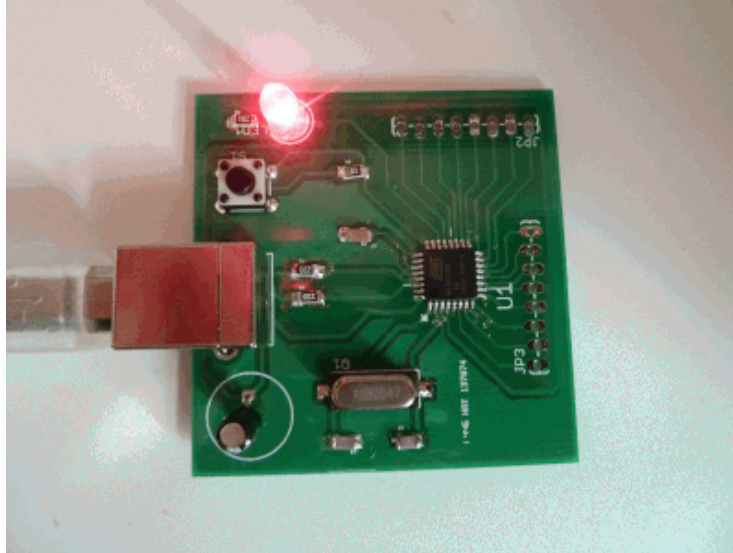
The first thing to do is to erase the old memory:

```
sudo dfu-programmer atmega32u2 erase
```

Then I flashed the microcontroller with the blink-led program:

.....
`sudo dfu-programmer atmega32u2 flash blink-led.hex`

I unplugged it from the USB, then connected it again... And it worked!



To finish this part of the microcontroller tutorial I've burned my fingers, I've doubted myself, and I've received a surprise bill from the customs office.

But all in all, I'm very happy with the result. I made it work. And I love the feeling I get when I make something work!

If you have any questions about what you've learned in this part, head on over to the following page and write your question in the comment section:

<https://build-electronic-circuits.com/microcontroller-tutorial-part5>

Part 6:

Learn to build your own ideas with electronics

Congratulations, you've made it all the way to the end!

If you're new to electronics, you probably did not understand the details of how to do everything in the tutorial, and that's okay. The fact that you read all the way through means you have the interest needed to get really good at electronics.

I'm constantly creating new learning material to help people build their own ideas with electronics. Whether you want to build an automatic cat door for your house or the next must-have gadget for the world, there are some steps you need to take to build up the skills and knowledge necessary to build your own inventions with electronics.

To help people from all backgrounds learn these steps, I've created Ohmify. It's an online electronics builders club with lots of courses and resources to learn electronics such as:

- 25+ courses on skills such as soldering, Arduino programming, basic electronics and more.
- Step-by-step instructions for cool projects like a robot, music player, phone charger and the Atari Punk Console
- Forum for discussing ideas, asking questions and getting un-stuck with your projects.

Learn more about Ohmify here:

<https://ohmify.com/join/>