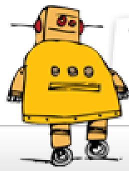


Getting Started with the ATMega328P

by **JayconSystems** on March 18, 2016

Table of Contents

Getting Started with the ATMega328P . . . 1

Intro: Getting Started with the ATMega328P . . . 2

Step 1: Application for the ATMega . . . 2

Step 2: Materials . . . 3

Step 3: Set-up . . . 3

Step 4: Set-up - continued . . . 3

Step 5: Set-up - continued . . . 4

Step 6: Set-up - continued . . . 5

Step 7: Set-up - continued . . . 5

Step 8: Set-up - continued . . . 6

Step 9: Test the Set-up . . . 6

Step 10: Understanding Hex Codes . . . 7

Step 11: Understanding Hex Codes - continued . . . 7

Step 12: Understanding Hex Codes - continued . . . 7

Step 13: Code to Blink the LED . . . 7

Step 14: Create the makefile . . . 7

Step 15: Make Our LED Blink . . . 8

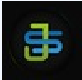
Step 16: Explaining the Code . . . 8

Step 17: In Conclusion..... . . . 9

Related Instructables . . . 9

Advertisements . . . 9

Comments . . . 9


Author:JayconSystems Jaycon Systems, LLC
We are tinkerers, inventors, geeks, students, teachers, hobbyist, engineers, and dreamers. We provide the building blocks to your projects!

Intro: Getting Started with the ATMega328P

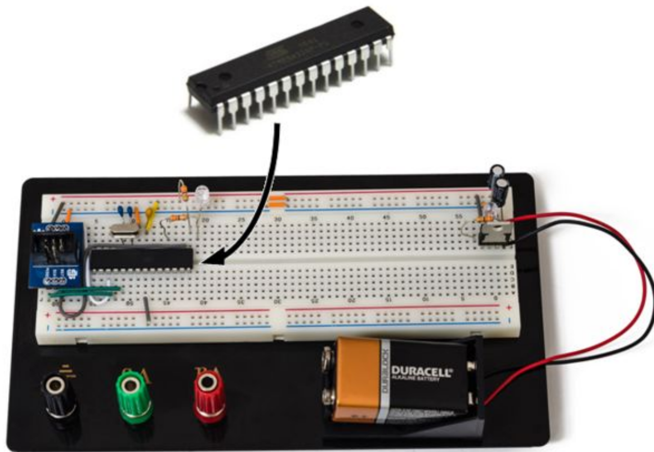
In the Internet of Things movement, people across the globe are connecting their stuff – TVs, pets, even houseplants - to the internet and transmitting all sorts of data.

If you're going to be a part of that movement, or want to dabble in creative prototyping on a budget, it's important to get to know our little friend:

The ATMega.

The real benefit of using this microcontroller is that it's only $4 US, whereas many other micro-controllers are 10X that price. It can also be easily programmed in the universal programming language, C++. The ATMega is also equipped with a decent amount of memory for any project.

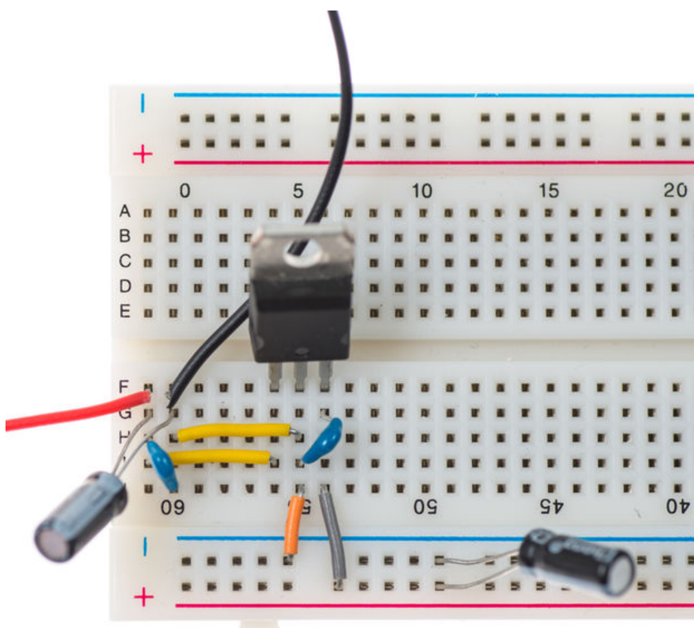
Photo by Will Carlson.



Step 1: Application for the ATMega

Applications for the ATMega continue to grow across the global tech sphere. Today, it's mostly used in simple machines to receive, interpret, and output information. You may have seen the ATMega used in small machines like RC cars and robots. It can make them autonomous and allow these devices to get from point A to point B on their own.Thus, for its size and its cost, this is a powerful little device. Jaycon Systems is here to equip you with the know-how to put it to use!

For our demo, we'll use the basic 5V power supply created by founder and friend, Jay. Above you can see a picture of the circuit.



Step 2: Materials

Hardware:

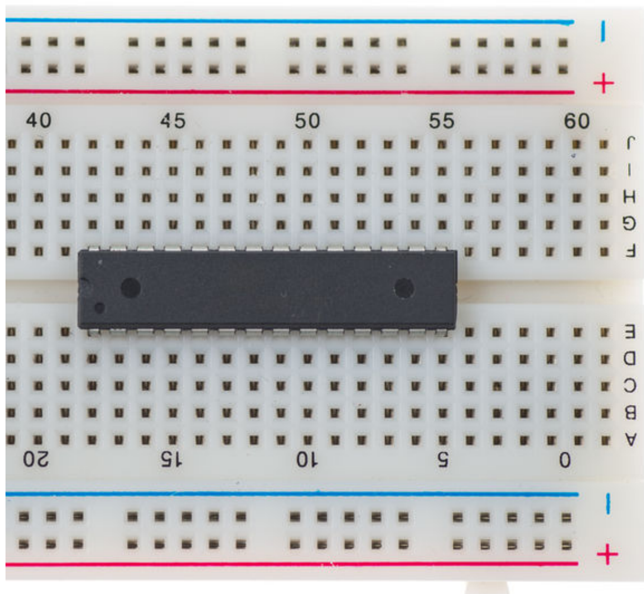
(2) 0.1 µF 50V Ceramic Capacitor
(1) 0.1 µF Ceramic Capacitor
(1) 16MHz Quartz Crystal Clock
(1) AVR ICSP Programming Adaptor
(1) ATMega328P
(1) LED 5mm (any color)
(2) 330Ohm Resistors
(1) USBTiny AVR Programmer
Jumper Wires

Software:

WinAVR [specifically, Programmer’s Notepad]

Step 3: Set-up

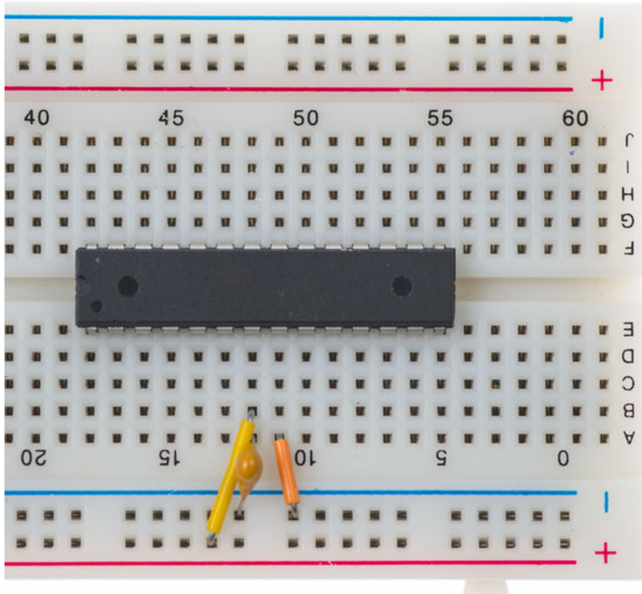
The ATMega328P is a microcontroller with 23 IO pins, two 8-bit internal clocks, and 32kB of flash memory.
To begin, notice the notch that indicates the direction of the chip — and the dot that indicates Pin 1. All pins that follow suit are in regular numerical order. If you are having trouble with the layout, check out the ATMega328P Datasheet for more information regarding the connections, and the ATMega's abilities.



Step 4: Set-up - continued

Add a wire from Power to Pin 7 (Power) and a wire from Pin 8 to Ground.

Then, add the 0.1µF Ceramic Capacitor to Pin 7 and Ground.



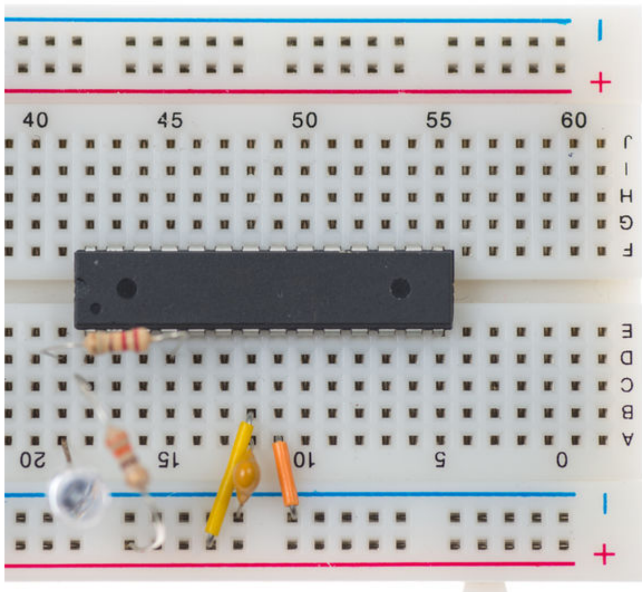
Step 5: Set-up - continued

Add a LED to the first available row next to Pin 1 that is not already being used by the ATMega.

Add a 330 Ohm Resistor from Pin 1 to Power. Pin 1 is the reset pin.

Then, add another 330 Ohm Resistor from Pin 4 to the row that the LED is connected to.
You have now powered the LED.

Note: The power applied from the Resistor to Pin 1 controls what “LOW” should be so the ATMega doesn’t constantly reset itself. The LED also needs the resistor so the ATMega doesn’t kill the LED.



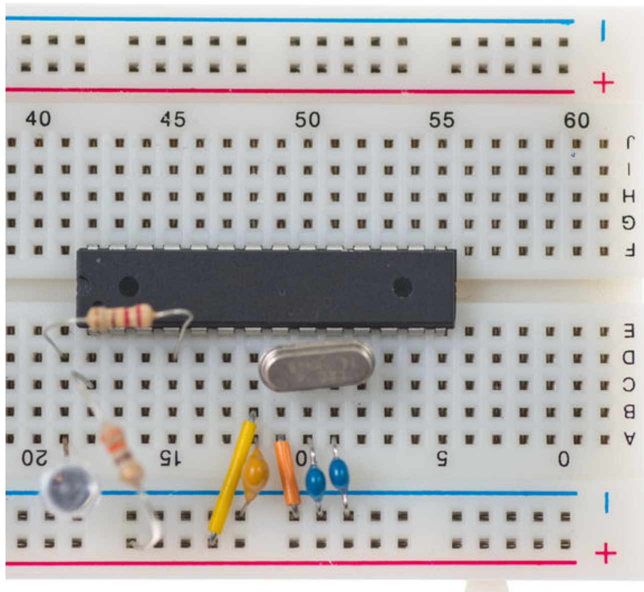
Step 6: Set-up - continued

The ATMega's clock is slow and unreliable because it's not constant.

Let's add a crystal clock that will speed it up and make the ATMega more reliable.

Add the Quartz crystal clock to Pins 9 and 10. Then, add a 0.1 µF 50V Ceramic Capacitor from Pin 9 to Ground, and another one from Pin 10 to Ground.

(**Note:** It's important to **not power the ATMega** with the clock already installed **unless** it has been flashed first.)

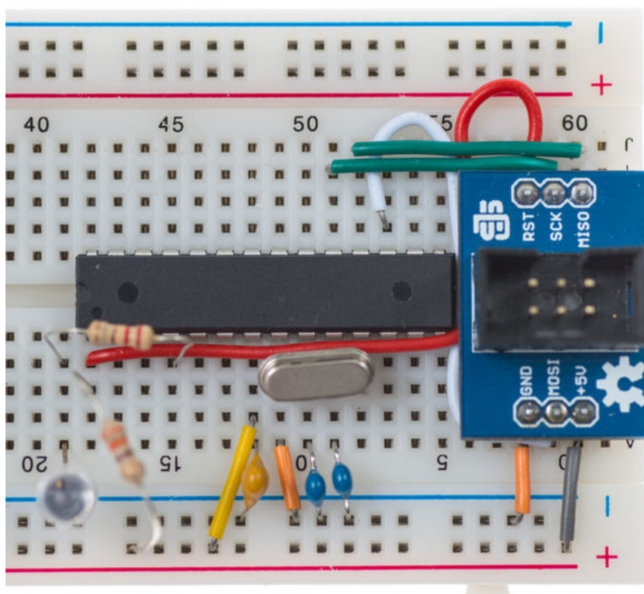


Step 7: Set-up - continued

You now have a working circuit, and need the programmer.

Solder the programming adapter together and place it on the board.

Make sure that you place parts in the correct direction, and be careful handling hot tools.



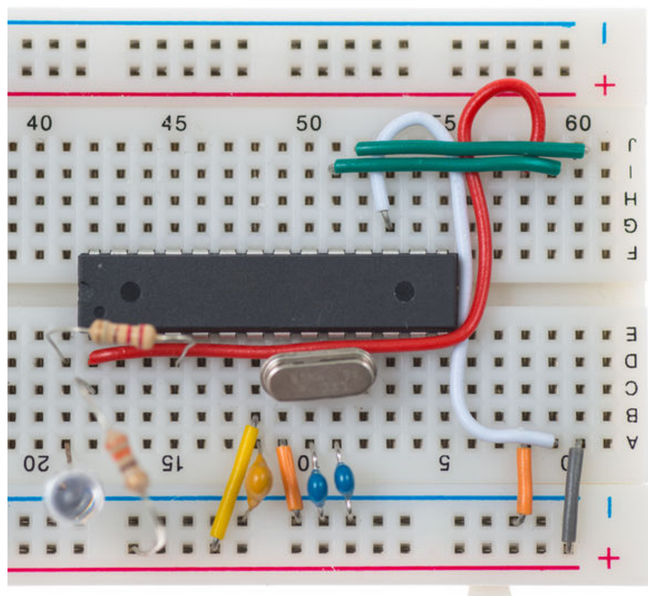
Step 8: Set-up - continued

The next step is to wire the programming adapter.
You can take any path to wire it in, as long as you use the correct pins.

Optional: If you are going to power your ATMega with the programming adapter, then wire from the “GND” and “+5V” pins on the adapter, across to the power rails.

Here is the Pin map for wiring the rest of the adapter: 17 to MOSI; RST to 1; 19 to SCK; and 18 to MISO.

The USBTiny has a small switch on it labeled “NO POWER” and “POWER TARGET”.
To power the ATMega from the USBTiny, set it to “POWER TARGET”.
Not interested? Simply set it to “NO POWER” to avoid unintentional destruction.



Step 9: Test the Set-up

Now that we have the board completely built, let’s program it so we can make the LED blink.
Once it blinks, you know that you have properly built the board.

Ready for the next steps?
First, we need to make the ATMega use the clock that was just installed.
Download and install WinAVR from this page.

WinAVR is a full suite with a compiler, programmer, debugger, and more!

Use these for the USBTiny. It will include Programmer’s Notepad, which is what we are going to use to program the ATMega, the AVRDude, and will burn fuses and act as a backup for programming the ATMega.

Find “Run” on your computer, type in “cmd” and click “OK”.

Type “avrdude -c usbtiny -p m328p -B 25 -U lfuse:w:0xFF:m -U hfuse:w:0xDE:m -U efuse:w:0x05:m”

-C identifies the programmer “usbtiny”

-P identifies the chip being programed “m328p” (short for ATMega328p)

-B sets the clock rate. We are setting it to 25 because the current clock is much slower than the clock on the programmer.

-U Is a memory operation, lfuse selects the low fuse, w tells the program to write it, and the hex code (0x##) is the fuse value.

This is repeated for the high fuse and extended fuse.

Step 10: Understanding Hex Codes

Hex codes may look intimidating, but they are really just counting, using an extended list from 0–15, starting with 0–9, and then continuing with a-f (filling the 10–15 places). The 0x in front of each code is how the software knows that it is reading a new value, because it never uses “0x”.

Hex:	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Value:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Step 11: Understanding Hex Codes - continued

What happens if you want to count a number bigger than 15?
Hex, like normal math, just counts up, rounding back to 1 followed by a 0; so 10 means 16, 11 means 17, and so on, like normal counting, but with more digits.

So 0xd3 just says 211, and if you want to say 75, just write 0x4b.

Hex:	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
Value:	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Step 12: Understanding Hex Codes - continued

This sets the necessary fuses the chip needs for the clock without killing itself. Once it is done, we are going to use Programmer’s Notepad to allow you to do a lot more with the chip than what you can do using Arduino.

Once the fuses have been burned, download and install WinAVR from this page. WinAVR is a full suite with a compiler, programmer, debugger, and more! We will use these for the USBTiny. It will include Programmer’s Notepad, which is what we are going to use to program the ATMega.

Step 13: Code to Blink the LED

Now we are going to create a .c file, which will contain the code that will blink the LED, and a makefile to specify what microcontroller we are using among other things.

Open Programmers Notepad and type in the following code:

```
#include <avr/io.h> // This contains the definitions of the terms used
#include <util/delay.h> // This contains the definition of delay function

void main()
{
DDRD = 0b00000100; // Port D2 (Pin 4 in the ATmega) made output
PORTD = 0b00000000; // Turn LED off

while(1)
{
PORTD = 0b00000100; //Turn LED on
_delay_ms(200); // delay of 200 millisecond
PORTD = 0b00000000; //Turn LED off
_delay_ms(200); // delay of 200 millisecond
}
}
```

Now save the file as led.c (you can choose a different name if you want as long as it has the .c extension).

It is recommended to create a new folder to save all your WinAVR projects.

For example, we created a folder in the desktop called AVR. The led.c file is then saved inside a folder called LED_blink within the AVR folder.

Step 14: Create the makefile

The next step is to create the makefile.

Open Mfile and follow these steps:

1. Click MakeFile -> Enable Editing of Makefile. This will allow you to modify the text within the Makefile.

2. Click MakeFile -> Main file name. This will open a small window in which you need to type the name of the file you created using Programmers Notepad without the .c extension. In this example we simply typed led and then clicked OK.

3. Since we are using a 16MHZ Quartz Crystal Clock we need to specify the processor frequency. This is done by changing F_CPU = 8000000 to F_CPU = 16000000 (we are only changing the last F_CPU that does not have a “#.” Leave the other ones as they are).

4. Click MakeFile -> MCU type -> ATMega -> atmega328p. This specifies the microcontroller that we are using.

5. Click MakeFile -> Port -> USB.

6. Scroll to the section” Programming Options (avrdude)” and change AVRDUDE_PROGRAMMER to USBtiny.

7. Save the file in the same folder where you saved the led.c file (Desktop\AVR\LED_Blink). Make sure you don’t change the name of the file (Makefile). Leave it as it is.

Step 15: Make Our LED Blink

Now we can upload the code to make our LED blink. Open the led.c file and do the following:

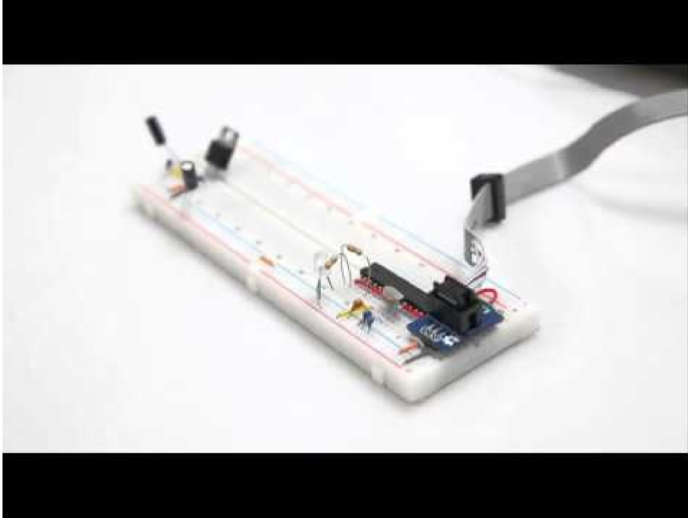
1.Click Tools -> Make All

2.Click Tools -> Make Clean

3.Click Tools -> Program

You can now disconnect the AVR ICSP Programming Adapter and use the 5V power supply. Your LED should be blinking now, like in the video above.

The command “make all” compiles and checks all the software from the two tabs, while “program” programs the ATMega with the newly compiled software. “Make clean” removes all previously created temporary files. It gets you ready for an entire new compile. This is nice because it allows you to compile the software once, and then program multiple chips, one after the other, without going through the whole process again! Trust me, you will be thankful once you start making bigger programs.



Step 16: Explaining the Code

#include <avr/io.h> declares the appropriate Input/Output definitions. For instance, including this header file allows us to use “DDRD” to set port D as an output.

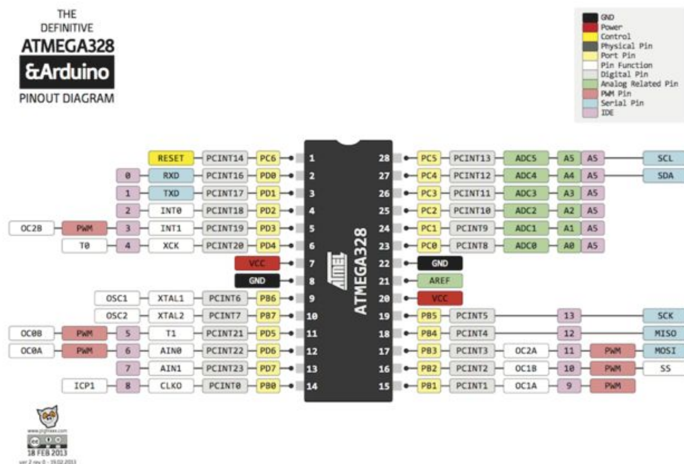
#include <util/delay.h> declares the basic busy-wait functions. This allows us to use the function _delay_ms() to create a delay in millisecond. The argument of the function states the number of milliseconds that we want to wait.

You may notice a semicolon (;) after every line of code. The semicolon tells the program that this is the end of the command. The reason you need this is because when the program reads it, it doesn’t see multiple lines from when you hit “Enter” or “Space”. Instead, it sees it all as one continuous, massive line of code, so it needs the semicolon to know when to start interpreting a different command.

“while(1)” is a simple way of making a continuous loop. The “while” command repeats everything inside the “{ }” immediately after it as long as the statement in the “( )” is true. Because it has been set to “1”, with no real variables or math, the statement will always be true.

The phrase “PORTD = 0b000000100;” translates into “Turn on Port D, number 2”, which corresponds to pin 4 of our microcontroller as seen in the picture above.

Each bit number represents a pin in a port. As you can see Port D has 8 pins: D0 -D7. So saying “PORTD = 0b00000100” is really this “PORTD = 0bD7D6D5D4D3D2D1D0.” Placing a one in one of the port’s pins drives the pin high, while placing a zero drives the pin low.



Step 17: In Conclusion.....

You may need something smaller, or need something to handle more data; otherwise, the Mega can do anything that your average project would require. It's rare that anything will overpower the ATMega.

For micro projects, or wearable electronics, turn to the AT Tiny.

If you have any questions about this tutorial, don't hesitate to post a comment, shoot us an email, or post it in our forum!

When you're ready to start your project, check Jaycon Systems online store for your component needs. While you're on our website, check out the other great tutorials we have available, and, if you have not already, the other Instructables on our profile.

Thanks for reading!

Let us know what you create!

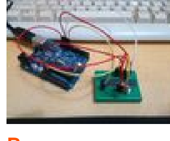
Related Instructables

Programming AVR With Arduino As ISP Without Bootloader and External Crystal by abhra0897

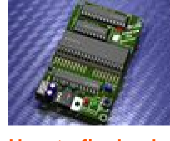
How to change fuse bits of AVR Atmega328p - 8bit microcontroller using Arduino by smandal13

Parallel AVR programming board by Milen

Build a Complete AVR System and Play Mastermind! by nevdull



Burn BootLoader into Atmega328P using Arduino Diecimila by whalescwh



How to fix dead atmega and attiny avr chips by manekinen

Comments