

# NerdKits

**ATmega328P Upgrade Nano-Guide**

ATmega328P Upgrade Nano-Guide  
by Humberto Evans & Michael F. Robbins  
rev 836-10590nt2mc9f3rb8p7jzs

Copyright 2009 Humberto Evans &  
Michael F. Robbins.

Unauthorized copying, transmittal,  
reproduction, or redistribution, in any  
form including electronic and paper, is  
expressly forbidden. The owners  
specifically forbid you from posting this  
material to a website or a P2P network,  
regardless of whether or not this is  
done for personal gain.

## When would the ATmega328P be useful to me?

The ATmega168 is a powerful chip and has been used in USB NerdKits for over a year. It has been and continues to be a great microcontroller platform to get started with microcontrollers and electronics, and The NerdKits Guide helps our customers get up and running with this chip and on their way to building their own microcontroller-based projects.

However, for some users, there are constraints which might be eased by moving to the ATmega328P microcontroller. In particular, the memory spaces on this chip are larger than on the ATmega168.

	<u>ATmega168</u>	<u>ATmega328P</u>	
Flash Memory*	16	32	kilobytes
SRAM	1024	2048	bytes
EEPROM	512	1024	bytes

\* In both cases, 2kB of flash is used for the bootloader itself, leaving either 14kB or 30kB available for your program.

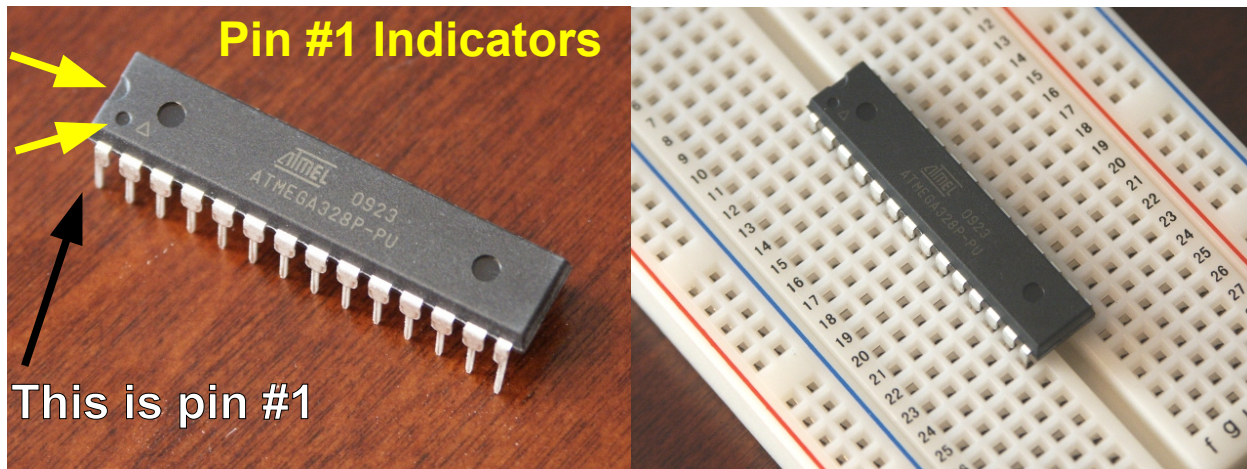
The ATmega168 with the NerdKits bootloader has  $16\text{kB} - 2\text{kB} = 14\text{kB}$  available to store your program. (The 2kB is for the bootloader itself.) This refers to flash memory, which is a type of reprogrammable memory that retains its information even when power is disconnected, and it's what you are writing to when you run “make” in your various NerdKits projects. While 14kB can represent quite a bit of code, when you include various components like floating point math libraries, printf/scanf functions, and LCD handling, it is possible to exceed this limit. The ATmega328P has  $32\text{kB} - 2\text{kB} = 30\text{kB}$  of flash memory available for your program – more than double that available to you in the ATmega168 – which may allow you to do more complicated projects with this chip.

Additionally, the ATmega328P has 2kB of SRAM, which is the temporary memory used to store your computations and data while your project is running. This is double the 1kB present in the ATmega168. While some programs need only a handful of RAM storage space, there are others in which it's important to keep long data buffers, or a long string of samples, in which case the additional RAM space may be particularly useful.

Finally, the ATmega328P is essentially a drop-in replacement for the ATmega168. It has the same pin layout and requires only a few changes to your development environment set-up, described in the next few pages. When loaded with the NerdKits bootloader, it works with the same USB-Serial type programmer that you've used with your ATmega168. No additional parts are necessary to move to the ATmega328P besides the chip itself.

## Step 1: Physically switching to the ATmega328P

On your breadboard, you simply need to physically remove the ATmega168 and then insert the ATmega328P in the same position, with power off.



**When removing the old chip and inserting the new one, be careful not to bend the leads.** They are quite fragile and can bend or even break off. Removal is most easily accomplished with a small screwdriver or a pen/pencil. Put it between one end of the chip and the breadboard, and pry the chip up slightly. Switch to the other side and repeat, trying to keep the chip level to prevent the chip leads from bending much as you go. When inserting the new chip, you may want to first make sure the two rows of pins are parallel by gently pushing the entire row against a flat surface, as the chips often come from the factory with a somewhat outward-pointing angle.

**Make sure you pay attention to where pin #1 is located.** This is indicated in two places on the top of the plastic chip package: first, a half-circle at the end of the chip where pin #1 and pin #28 are located, and nearby you'll see a smaller circular imprint off to one side, indicating pin #1. Finally, you can verify this by holding the chip so that you can read the "ATmega328P" text on the top normally (upright). In this position, pin #1 is located at the bottom left. Now that you've identified pin #1, you need to match it to pin #1 of the ATmega168 that is being replaced. When we ship NerdKits, this is generally placed in row #11 of the breadboard, and if everything is already wired up, you'll find that this node is connected to +5V (the red rail). **Please double-check that pin #1 is in the correct place before you apply power.**

**Now, set the programming/run mode switch to run (so that microcontroller pin 14 is left floating), and then apply power to your breadboard.** If your LCD is connected, you'll see a message indicating that your ATmega328P is running! If your LCD is not connected, then please just continue along with these instructions and see if you are able to program the chip. If your LCD is connected but you were unable to see this message, please confirm that your chip is in run mode, not programming mode, and then if issues persist please contact us at [support@nerdkits.com](mailto:support@nerdkits.com), or simply try continuing along with the instructions.

## Step 2a: Modify Makefiles

Next, you must make two changes to your Makefile in order to inform your development tools about the new chip type. Open the Makefile in your favorite text editor. First, find the line that begins “GCCFLAGS”, and specifically the option

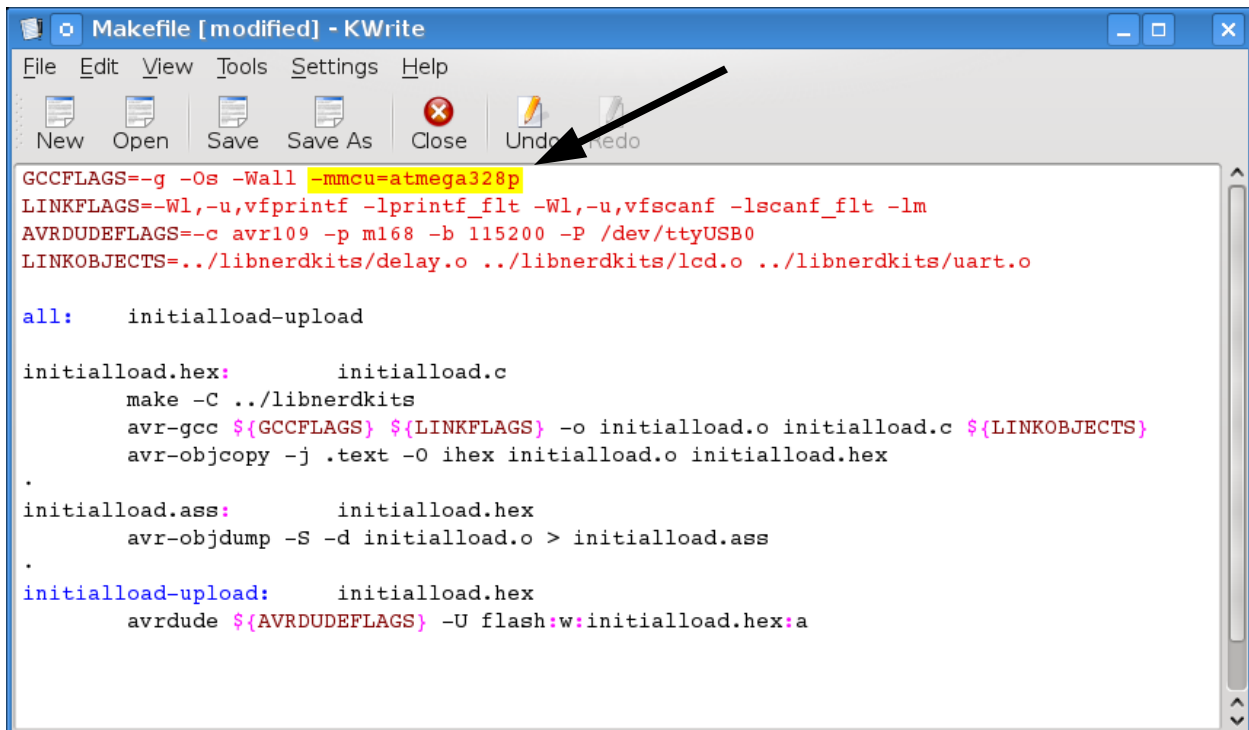
```
-mmcu=atmega168
```

Change this option to

```
-mmcu=atmega328p
```

This tells the compiler that you're using the new chip, and that it should take into account the newly available flash and RAM space.

Also repeat this step for the Makefile that is present in the libnerdkits directory.



```
Makefile [modified] - KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo
GCCFLAGS=-g -Os -Wall -mmcu=atmega328p
LINKFLAGS=-Wl,-u,vfprintf -lprintf_flt -Wl,-u,vfscanf -lscanf_flt -lm
AVRDUDEFLAGS=-c avr109 -p m168 -b 115200 -P /dev/ttyUSB0
LINKOBJECTS=../libnerdkits/delay.o ../libnerdkits/lcd.o ../libnerdkits/uart.o

all:    initialload-upload

initialload.hex:    initialload.c
    make -C ../libnerdkits
    avr-gcc ${GCCFLAGS} ${LINKFLAGS} -o initialload.o initialload.c ${LINKOBJECTS}
    avr-objcopy -j .text -O ihex initialload.o initialload.hex
.
initialload.ass:    initialload.hex
    avr-objdump -S -d initialload.o > initialload.ass
.
initialload-upload:    initialload.hex
    avrdude ${AVRDUDEFLAGS} -U flash:w:initialload.hex:a
```

## Step 2b: Modify Makefiles

Then, while still editing the Makefile, find the line that begins “AVRDUDEFLAGS”, and specifically the option

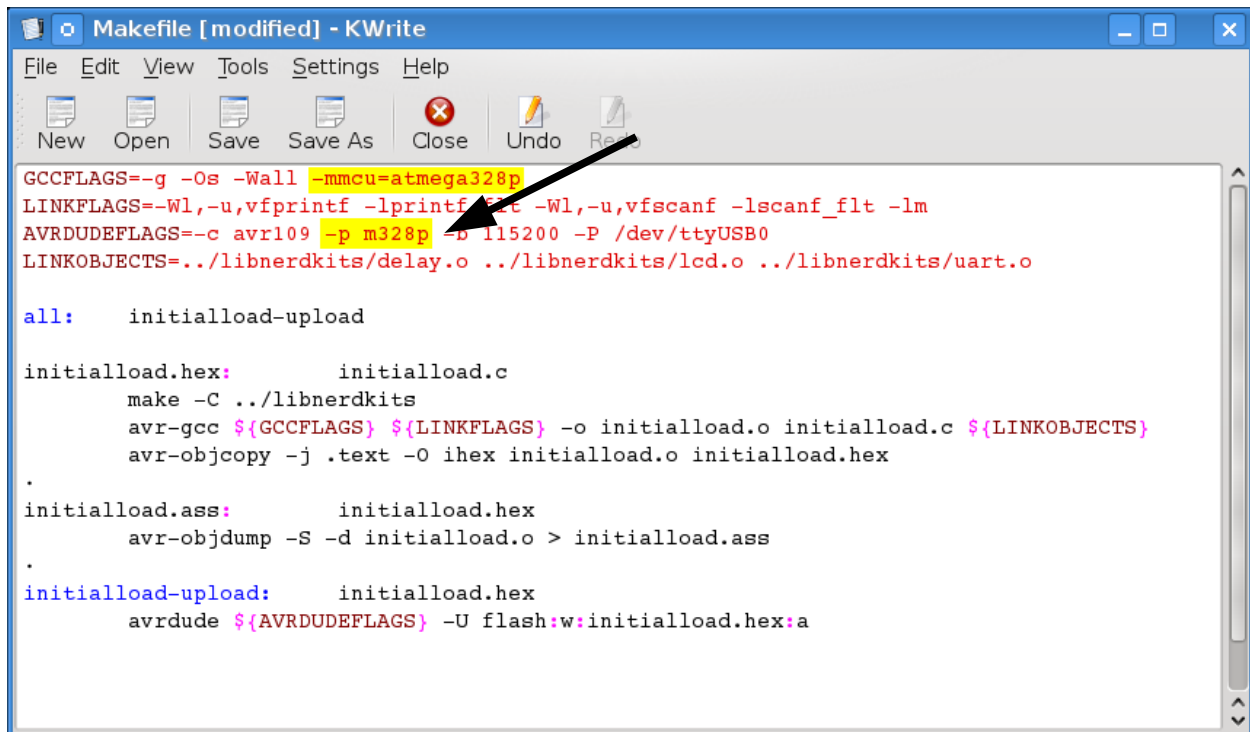
```
-p m168
```

Change this to read

```
-p m328p
```

This tells the avrdude program that it should expect to be communicating with the larger ATmega328P device. You can now save and close the Makefile.

You will need to apply these modifications to the Makefile for each project directory you plan on using with the ATmega328P chip.



```

Makefile [modified] - KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo
GCCFLAGS=-g -Os -Wall -mmcu=atmega328p
LINKFLAGS=-Wl,-u,vfprintf -lprintf_std -Wl,-u,vscanf -lscanf_flt -lm
AVRDUDEFLAGS=-c avr109 -p m328p -b 115200 -P /dev/ttyUSB0
LINKOBJECTS=../libnerdkits/delay.o ../libnerdkits/lcd.o ../libnerdkits/uart.o

all:    initialload-upload

initialload.hex:    initialload.c
    make -C ../libnerdkits
    avr-gcc ${GCCFLAGS} ${LINKFLAGS} -o initialload.o initialload.c ${LINKOBJECTS}
    avr-objcopy -j .text -O ihex initialload.o initialload.hex
.
initialload.ass:    initialload.hex
    avr-objdump -S -d initialload.o > initialload.ass
.
initialload-upload:    initialload.hex
    avrdude ${AVRDUDEFLAGS} -U flash:w:initialload.hex:a
  
```

### Step 3: Add new header file

Now, you'll need to download one extra file and place it inside your libnerdkits folder. Download this file here:

[http://www.nerdkits.com/files/Downloads/io\\_328p.h](http://www.nerdkits.com/files/Downloads/io_328p.h)

and put it in your “libnerdkits” directory.

To make your code reference these files, open up your code file and add the following include statement near the top of the file, along with the various other includes you might be using:

```
#include "../libnerdkits/io_328p.h"
```

This tells your code to take advantage of the definitions in that header file.

#### ***Why is this step necessary?***

This file contains some definitions that make it easy to work with the ATmega328P using the same code that you've used for the ATmega168, such as “PB0”. For some reason, the maintainers of the avr-libc library forgot to define these PB0, PB1, PB2... bits for the ATmega328P, even though they do so for all of the other chips in the library. If you look at our io\_328p.h file, you'll simply find definitions like:

```
#define PB0 0
#define PB1 1
...
```

The ATmega328P datasheet continues to show example code with these names, and we think it's a perfectly good notation, so we put together this quick header file which simply defines those names for PB0..PB7, and the same for PC $n$  and PD $n$ .

Hopefully, the upstream maintainers of avr-libc will fix this in a future version of avr-libc and this header file will become unnecessary, but for now it provides easy compatibility for your code.

## Step 4: Clear old compiled versions

The next step is to clear any old compiled output files to force the compiler to re-generate them with the new settings. In your project directory, remove any files ending in “.hex” or “.o”.

Also, go into the libnerdkits directory and remove any files ending in “.o”.

While the “make” tool is great for detecting when a source code file has changed and needs to be recompiled, it is not set up to handle the scenario where the Makefile itself is what has changed. That's why we have to force it to recompile your code here.

### ***Why is this step necessary?***

The entries in any Makefile follow a simple pattern:

```
target: dependencies
      commands
```

The “make” command looks at file modification times. If any of the files in the `dependencies` list have been changed more recently than the `target` itself, “make” knows that it needs to do something to re-generate an up-to-date `target`, and that something is to execute the `commands`. However, a modification of the Makefile itself will not trigger this by itself, so we have to manually remove the `target` files so that “make” is forced to re-generate them with our new settings.

### ***Command line tips***

If you aren't already familiar with using “wildcards” at the command line, this is an opportunity to practice using them.

In Windows, running

```
del *.hex *.o
```

or in Linux or Mac OS X,

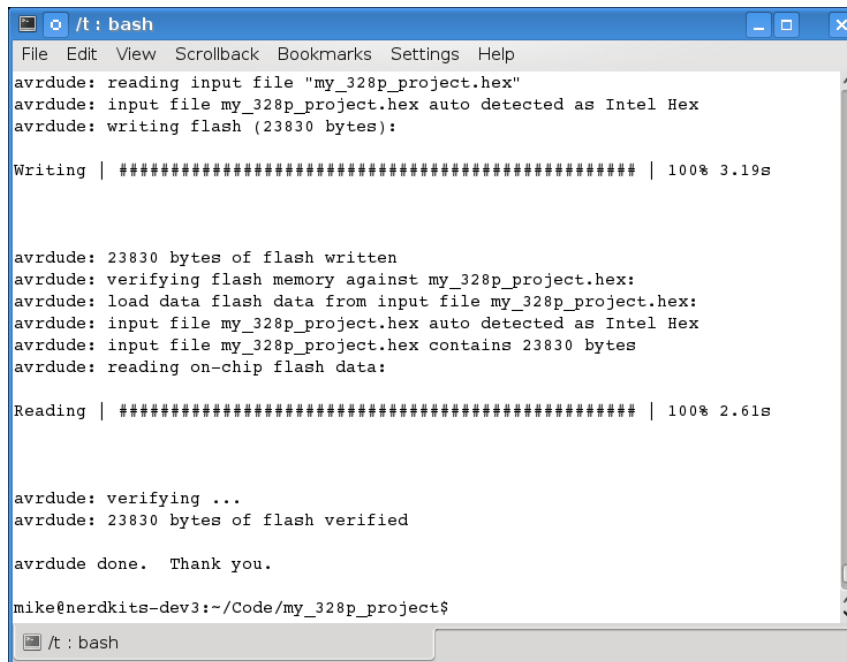
```
rm *.hex *.o
```

will delete all files in the current directory that end in “.hex” or “.o”. The shell allows the wildcard “\*” to match any file name, so “\*.hex” will match anything that ends with “.hex”.



## Step 5: Recompile and upload!

Finally, it's time to load your code onto the new chip. Set your run/program mode select switch to program mode, such that microcontroller pin #14 is connected to ground, and apply power to your breadboard. At the command line, go into your project directory and run "make". Your program will now be re-compiled for the ATmega328P processor, and will be loaded onto the chip!



```
/t : bash
File Edit View Scrollback Bookmarks Settings Help
avrdude: reading input file "my_328p_project.hex"
avrdude: input file my_328p_project.hex auto detected as Intel Hex
avrdude: writing flash (23830 bytes):

Writing | ##### | 100% 3.19s

avrdude: 23830 bytes of flash written
avrdude: verifying flash memory against my_328p_project.hex:
avrdude: load data flash data from input file my_328p_project.hex:
avrdude: input file my_328p_project.hex auto detected as Intel Hex
avrdude: input file my_328p_project.hex contains 23830 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 2.61s

avrdude: verifying ...
avrdude: 23830 bytes of flash verified

avrdude done. Thank you.

mike@nerdkits-dev3:~/Code/my_328p_project$
/t : bash
```

You can now download the ATmega328P datasheet:

<http://www.nerdkits.com/files/Downloads/Datasheets/ATmega328P.pdf>

and refer to this as you design your microcontroller projects. However, be aware that it is substantially equivalent to the ATmega168 datasheet, and register names and behaviors are essentially exactly identical in our experience. Your code should work 100% as it did previously, but you can now modify it to take advantage of larger memory spaces.

As always, please go to the NerdKits Forums at

<http://www.nerdkits.com/forum/>

and let us know if the new chip has given your project enhanced capabilities!