Ghetto Programming: Getting started with AVR microprocessors on the cheap.

by The Real Elliot on November 16, 2006

Table of Contents

intro: Ghetto Programming: Getting started with AVR microprocessors on the cheap.

Update Aug 24, 2007: I've gone USB!

http://www.instructibles.com/id/EDRQZ56F5LD8KDX/ is a better picture of what I'm using now, and is a real improvement. That said, the basics here are still applicable, especially all the software stuff.

Microprocessors are so cheap these days. If only there were a way to program them up just as cheaply...

wavy dream-sequence lines

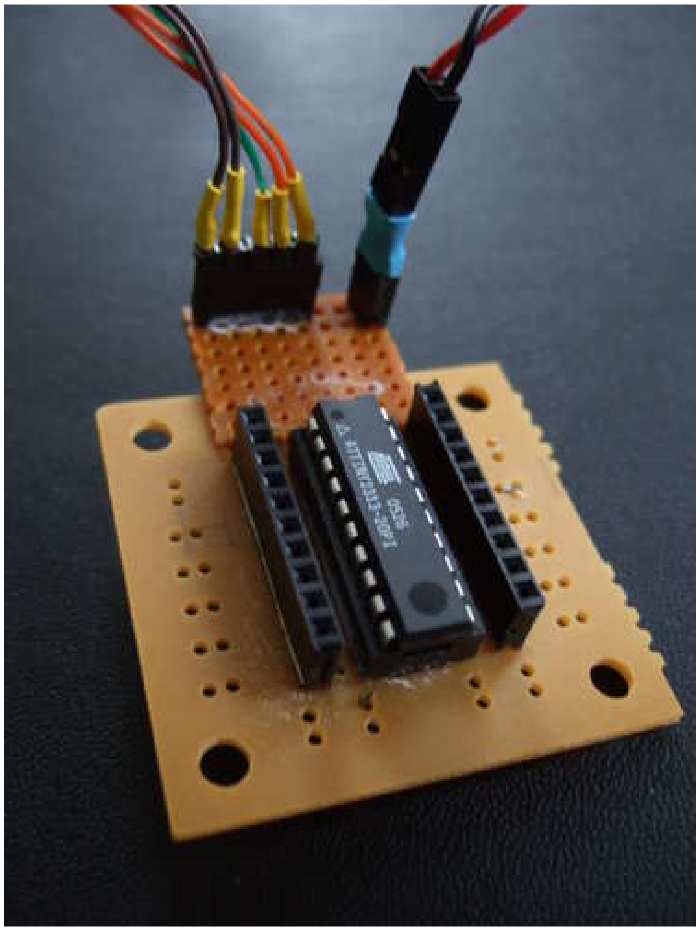
In this instructable, find out how to build up a complete AVR microprocessor toolchain: compiler, programmer software, programmer hardware, and some simple demos to get your feet wet.

From there, it's just a hop, skip, and a jump to world domination.

The endpoint is not quite as swanky as Atmel's suite, but it's gonna run you about $150 less and take only a little more work to get it set up.

This instructable is based on the Atmel ATtiny 2313 chip, mostly because it's one of the smaller chips (in size) while still being beefy enough to do most anything. And at $3 a pop (non-bulk), they don't break the bank.

That said, most of the steps are applicable across the AVR family, so you'll be able to re-use most everything when your programming needs outgrow the ATtiny and you reach for the $8-$12 ATmegas.



step 1: Go Order Parts!

Necessary Parts List (stuff to buy or scrounge):

```
Atmel ATtiny2313 chip:       $2.50 - $3
Socket for chip:             $1
Parallel port connector (DB25):       $3 at Radio Shack or $0.95 at Sparkfun
Header Pins:                 $1-$2
             (at least one strip each male & female, maybe 2 female)
Some LEDs and resistors:     $1
```

= Around $10, even if you go deluxe. Half that if you can find an old parallel port cable.

Add-ons

A pushbutton switch or two: $1.

A piezo speaker: $1-$5
Light sensitive photocells: $2-$6
Breadboard for making complex circuits: $8-$10(?)

Other Stuff You Oughta Have

Computer: The older the better because it needs a parallel port.
Hookup wire, solder, soldering iron
Super-duper glue
Source of ~5v DC: Batteries will work, and old computer power supplies are perfect.

I get a lot of stuff from Sparkfun because they're
fast, reliable, and fairly priced. They carry AVR chips (but seem to be out
of ATtiny13's at the moment). They've got everything on the "Necessary"
list in one place, nice website.

If you're gonna be ordering a lot of chips, you can get a deal from Digi-key or similar. For instance: ATtiny 2313 for $2.36 each. They've only got 27,853 more in stock, so order quick. (Make sure you get the DIP form-factor.)

Go order stuff now, and we can set up the software side while you wait.

The waiting is the hardest part.

step 2: Get and Install Software: Linux Version

If you run Windows, skip this page. Or install Linux, then come back. :)

The Linux AVR software toolchain used is:
1) an editor of your choice (emacs, gedit, kate, etc)
2) AVR-GCC compiler and libraries
3) AVRdude programmer
4) A nice Makefile to tie it all together

May 2008 Edit: Ubuntu has packages for the whole toolchain. Your installation is now as easy as: "sudo apt-get install avrdude avr-libc binutils-avr gcc-avr". Bam! The versions that come with 8.04 are good all around. Woot Ubuntu!

If you want to do it "by hand", the script below automates most of the work for you. Note that as of May 2008, it's a year old, and doesn't support the newer ATTinyx5 series chips. You can mess around with updated versions of all the software is you'd like. (It's saved with a .txt extension b/c Instructables wants it. Doesn't matter.)

You're going to have to edit the first command in the install script depending on whether you're using an apt-based package system or a yum-based one. Just uncomment the relevant line. It needs to make sure you've got some packages which are required for the compilation stage.

Have a read through the script to make sure it's not doing anything stupid, then get root and run it and you should be golden.

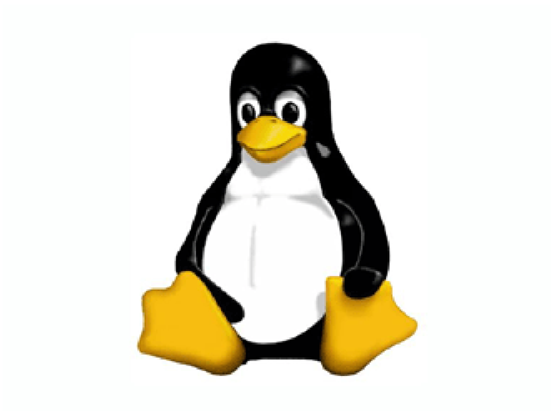
It will create a directory /usr/local/AVR and install all the AVR tools there.

The last thing the script does is to add avr-gcc to your path so that you can run avr-gcc directly by adding a line to the tail of your /etc/profile.

The other file, setupParport.txt, is just a (very) simple script to set up your parallel port for user use. If you're savvy, you can add the same commands to a boot script so that it enables the parallel port every time you boot up. Otherwise, just run it by hand.

Websites with similar, but different and possibly outdated, procedures include: Psychogenic.com, AVRWiki, and this (old) Linux Focus article.

Alternatively, I hear cdk4AVR has a complete set of compiler tools in RPM format. I haven't tried it, but it could save you the 30 min of compiling time on a slow, old laptop.



File Downloads

avr-gcc_installer.txt (1 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'avr-gcc_installer.txt']

setupParport.txt (71 bytes)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'setupParport.txt']

step 3: Get and Install Software: Windows Version

The MS Windows software toolchain is based on the same compiler as Linux one, but with some optional extra GUI friendliness.

WinAVR is the compiler and programmer software. It also includes a nice programming editor, Programmer's Notepad.

Download and install it. It's that easy. Almost.

As the last step in setup, you'll need to make the parallel port user-accessible. WinAVR includes a program for you to do this. If you accepted the default installation options, run C:\winAVR\bin\install_giveio.bat by double-clicking on it. A window will flash up and down, and then you're done.

step 4: Make Programmer Cable

The cable you're going to make is a "Direct AVR Parallel Access" or DAPA cable.

I think I got the pinouts from somewhere else, but this site has a nice schematic of the parallel port pins for your reference.

Mine goes something like this:

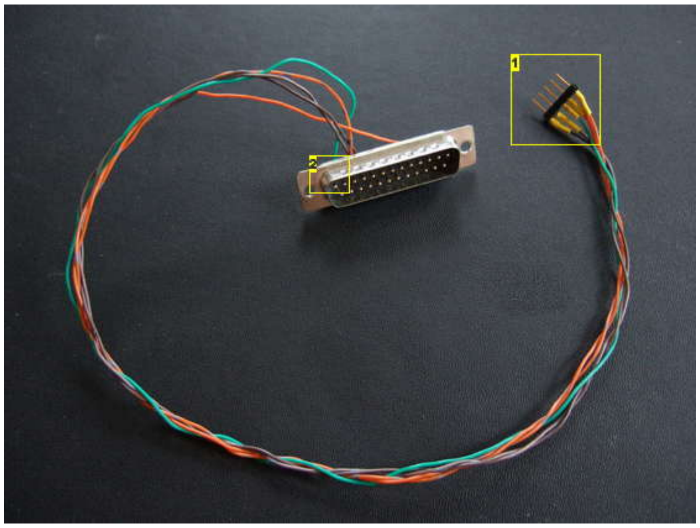
```
Parallel Port    AVR Function  Color
2                MOSI          Orange/Grey
11               MISO          Orange
1                SCK           Green
16               RESET         Brown
18            GND           Brown/Grey
```

Only tricky bit here: Note that pin 1 (SCK) is on the upper-right hand side when you are looking at the solder pins from the back. It's upper-left when you're looking at it here, and in the circuit diagram.

Also, the guy's website above has ground connected to 20 and 21 while mine (and others) use 18. Many of the pins connect to ground, and it doesn't matter which of them you pick, as long as you get ground.

If you look around the web, you'll find that most people put resistors in either the cable or the cradle (next step) to protect their computer's parallel port from excessive voltages on the AVR chip for use when programming it in-circuit. We will be using strictly 5v here, so there's no such worry, and I leave them out for simplicity.

However, if your chip ever comes near >5v, DO NOT USE THIS CABLE WITHOUT RESISTORS! A computer with a burnt-out parallel port is no fun. That said, I've been using it without incident for 6 months now.



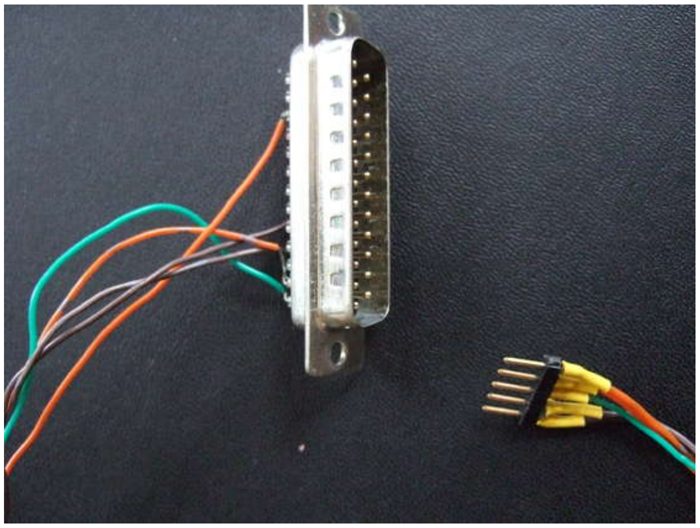


Image Notes
1. Top to bottom: MOSI, MISO, SCK, RESET, GND
2. Pin 1 Here

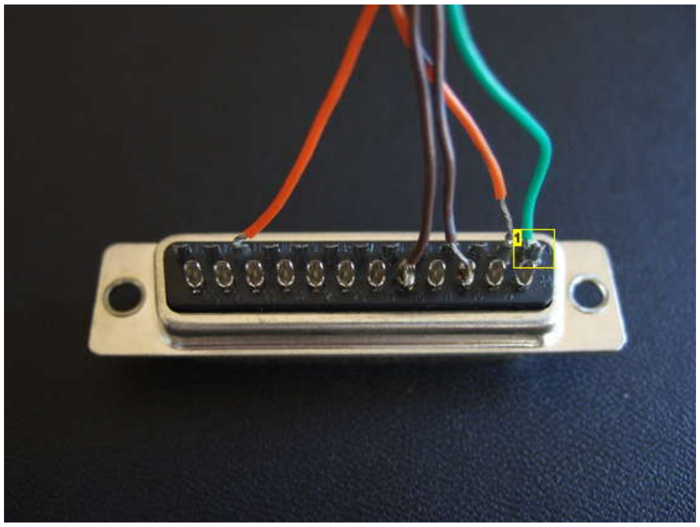


Image Notes
1. Pin 1: Reset

step 5: Make the programming cradle

Now use the 20-pin socket and female headers to make a cradle which connects the pins from the cable to the right pins on the chip.

The first thing to do is superglue all the headers and the socket to the circuit board. That way, it's easier to solder to. You can even make an extension for the header like I did if you need more room on your circuit board.

The wiring is as follows:

```
Cable     ATtiny2313 Pin
MOSI          17
MISO          18
SCK           19
RESET         1
GND           10
```

Get yourself the short ATtiny2313 overview to double-check the pinouts.

And remember: you're soldering up the wires on the underneath, and it's mirror-image. It might help to mark where pin 1 is on the bottom side before soldering. (I did it wrong once. Once.)

That said, the wiring is very simple. Just pin to pin, and then connect all the header pins to the closest pins on the socket. Fortunately for me, this pre-printed circuit board from Radio Shack did the trick.

The cradle is versatile too. See the last picture for my ATtiny13 programmer cradle. I have another for an ATMega8 too.

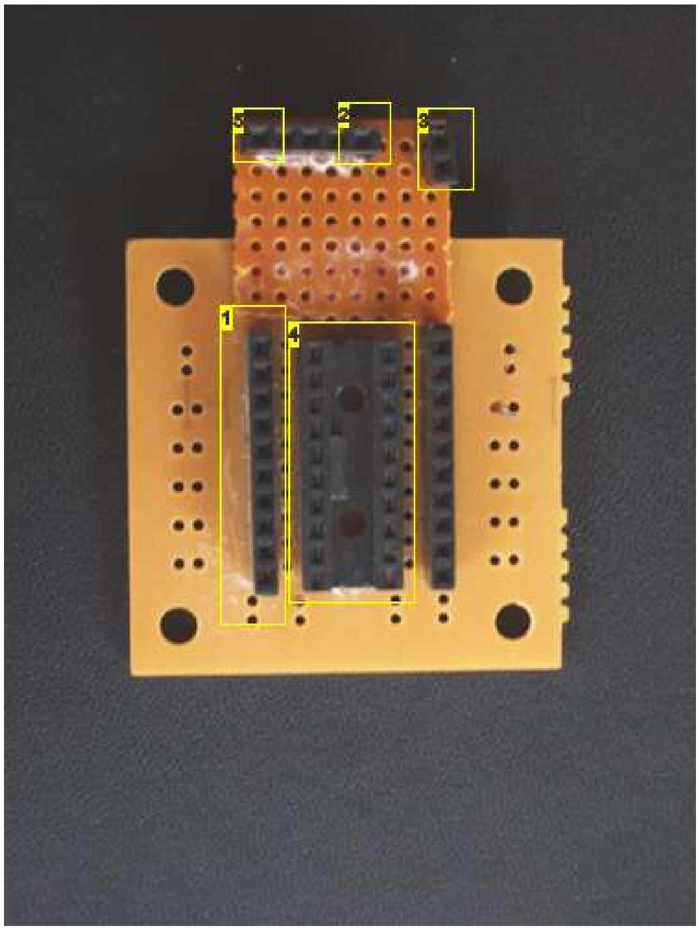


Image Notes
1. These connect directly to the AVR pins to make them easily accessible later.
2. MOSI
3. 5v and ground. Top to bottom.
4. Socket for AVR
5. GND

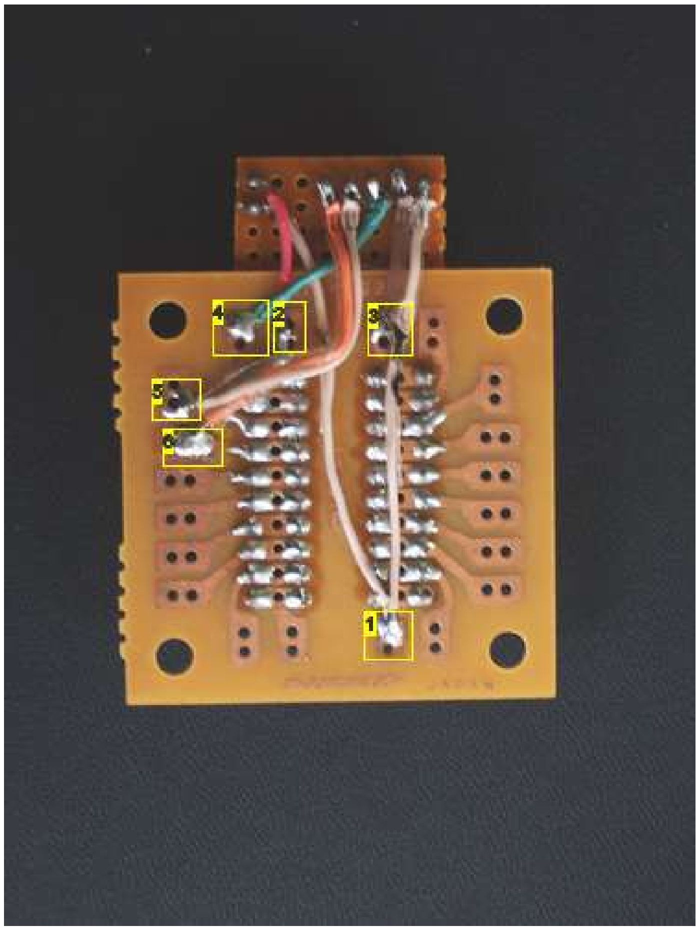
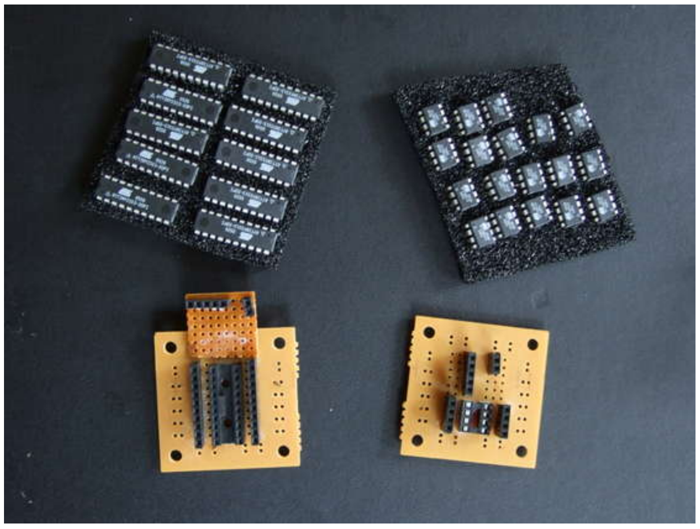
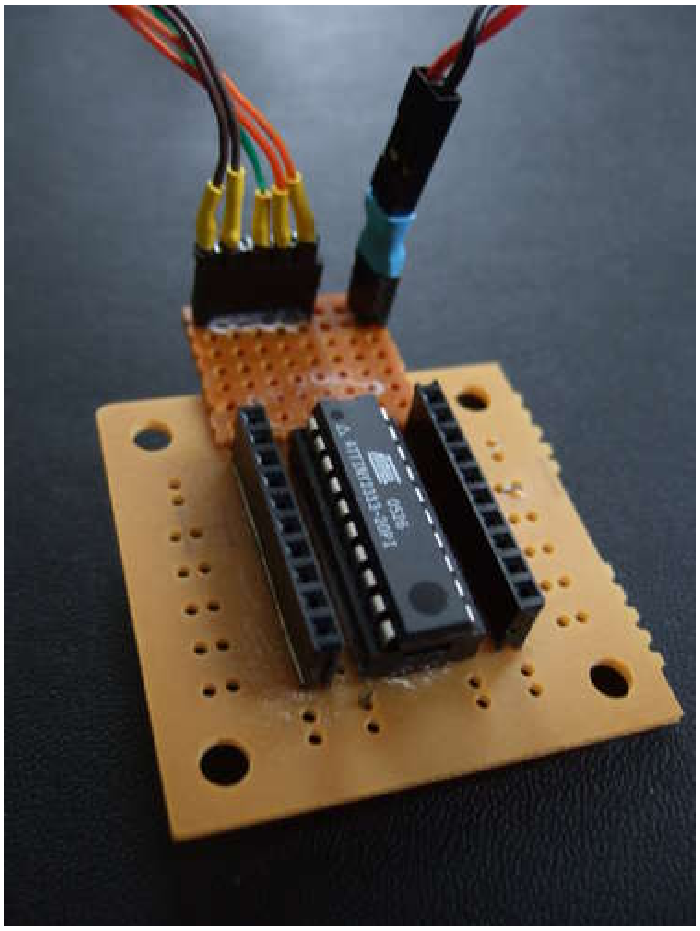
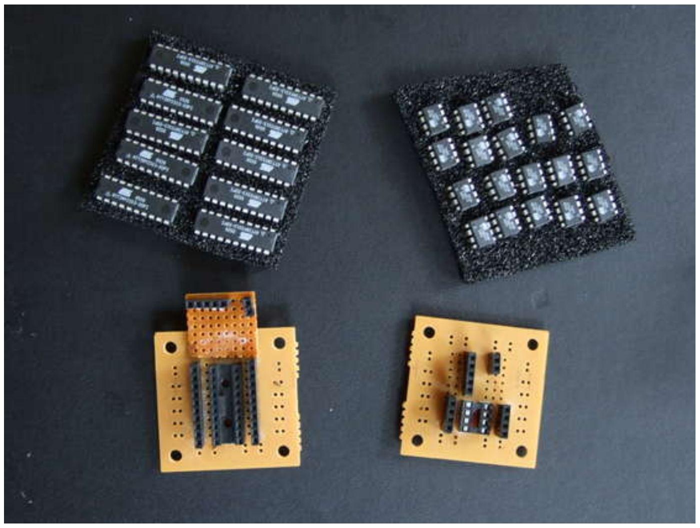


Image Notes
1. Ground
2. 5v
3. Reset
4. Clock
5. MISO
6. MOSI





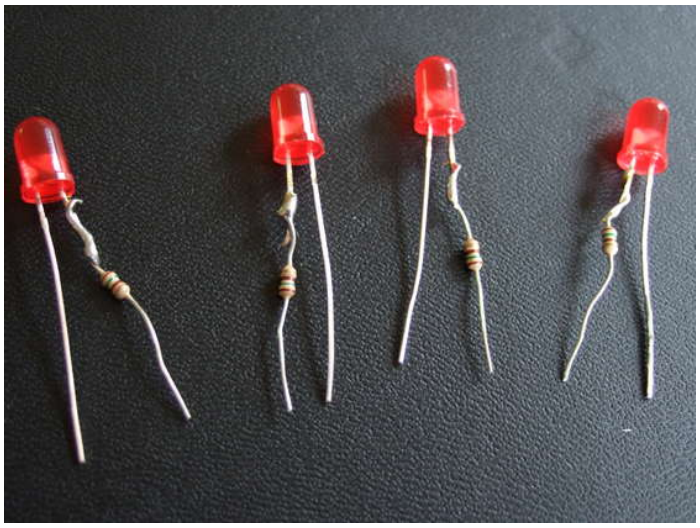


step 6: Make some Blinkenlights

An integral part of the Ghetto Development Environment is the plug-in LED.

The AVR chips can all source a fair amount of juice (50mA or more), which is enough to burn out your standard red LED, so it's a good idea to put protective resistors inline with your LEDs. Enter the Ghetto Blinkenlight.

To make them, simply solder a smallish (150 ohm) resistor to the negative lead of an LED, then you can just plug it straight into 5v and it won't burn out. (5 - 1.4) v / 150ohm = 24mA, which is just about right. Make a few -- I made eight. We only use one here.



step 7: Set up the programming project

Linux and Windows, the procedure for starting a new project is basically the same.

1) Make a new directory. Here, call it LED_Demo or something.
2) Copy your Makefile into that directory.
3) Download the code (LED_Demo.c) into the directory, or start writing new code from scratch.
4) Edit the Makefile to reflect the chip you're using and the name of the project.

Then you're ready to program.

Editing the Makefile

Windows: The Makefile's set up for you. All you have to do to make the Makefile work is remove the ".txt" from the end of the filename. Open it up in Wordpad or Programmer's Notepad anyway, just to have a look at the options you can change later.

Linux: Rename the Makefile.txt to Makefile. Then open the Makefile up in an editor. Un-comment the line that has "/dev/parport0" and comment out (with a #) the line that has "lpt1".

File Downloads

Makefile.txt (9 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Makefile.txt']

LED_Demo.c (451 bytes)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'LED_Demo.c']

step 8: Get your first demo circuit set up

Put the chip in the cradle. You'll notice that the pins all angle outwards a bit when the chips come from the factory. Bend them in (gently) on the table so that they're all even and parallel. Then it fits in the socket nicely.

Notice the alignment of the chip. The little dot marks pin 1. Is it where pin 1 should be?

Plug in your 5v power and the programmer cable. Plug the programming cable up to your computer.

Take a Ghetto Blinkenlight and put it between pins 8 and 10 so that the resistor (negative) is in pin 10, the ground for the chip.

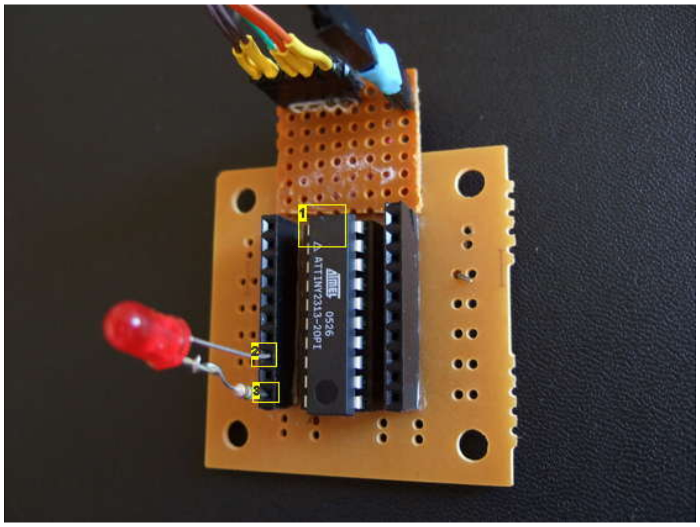
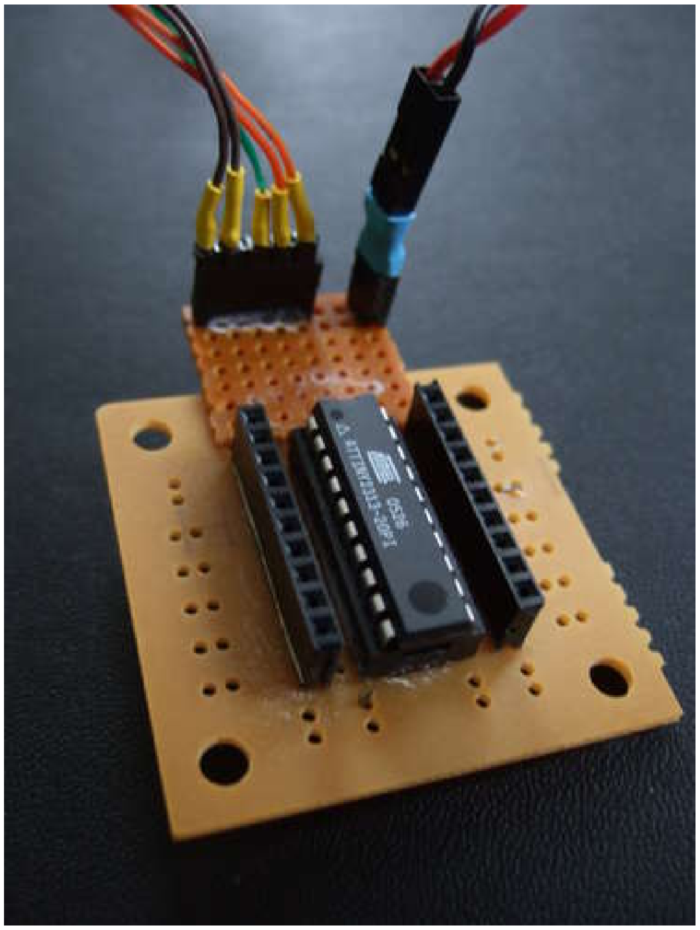
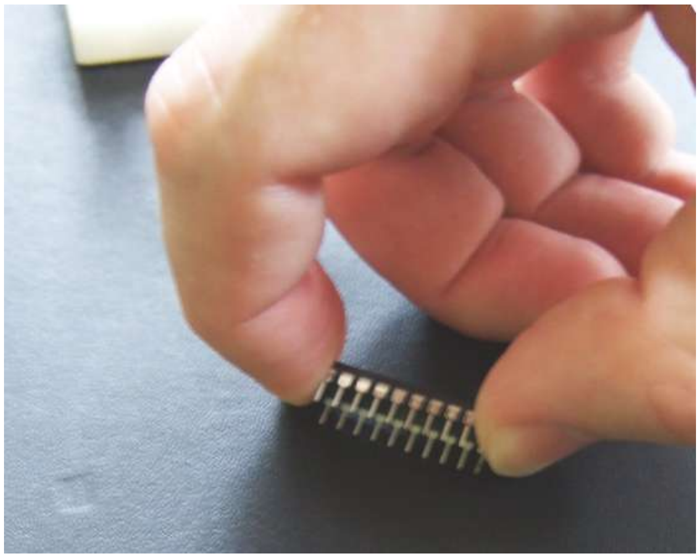


Image Notes
1. Small notch and round dot mark Pin 1.
2. Pin 8, Port D4
3. Pin 10, Ground





step 9: Program the Chip

Linux:
1) Get in the LED_Demo directory.
2) Cross your fingers.
3) Type "make"

Windows:
1) Open up LED_Demo.c with Programmer's Notepad.
2) Cross your fingers.
3) Select "Tools...Make Program"

Both:

Watch the log as the text goes scrolling by. There are two things you're looking out for here.

One is if the program compiled sucessfully. If there are no errors, it did. If it didn't, why? Did you mess around with my code?

Two is if the chip programmed sucessfully. For this, it writes the code in, then verifies the chip's memory. It should say "Contents Verified." If it says something about the parallel port or "giveio.sys", did you enable the parallel port back a couple steps ago?

Success? Yay! There's nothing like the sweet smell of blinking LED's in the morning.

step 10: Explaining the software

The code's pretty simple, as far as AVR code goes, but still displays a few tricks of the trade. If you're still basking in the glory of your blinking light, you can come back later.

The #include lines load up some of the extra functions and definitions in the AVR-libC suite. In particular, the delay function _delay_ms() is in delay.h. Interrupt.h has all the pin definitions which make life easier. You're always going to want to include it.

The program always starts from the function main(). In this simple case, it's the only function we have.

The first command in main(), DDRD = _BV(PD4);, seems pretty cryptic, but here's what it's doing. DDRD is the Data Direction Register for port D. All the input/output pins are broken up into different ports for easier access. The one we're using happens to be in D. We need to enable the pin PD4 for output for the LED.

The DDRs are set up so that they have 8 bits, one for each pin in the port. You set a pin up for input by writing 0 to its bit in the register. You can set it to output by writing a 1. We want the contents of DDRD to be 00001000, or output only on pin 4 (read right to left). So how do we do this?

_BV(i) takes a number, i, and converts it to an 8-bit binary number where the i'th bit is a 1 and the rest are zeros. Just exactly what we need. And PD4 is the number corresponding to pin 4 on this port. So we set DDRD to _BV(PD4), and then pin 4 (and only pin 4) is set up for output -- blinking our LED.

The rest of the program repeats forever in a loop. It alternates between turning pin D4 on and off, with a delay in-between.

You can turn the individual pins on (and off) by writing a 1 (0) to the PORT register. The syntax is just like above with the DDR -- PORTD = _BV(PD4) sets the fourth pin in port D to 1.

The _delay_ms() function then waits for a bit. It may not be quite 1ms, though. Depending on what clock speed your chip is set at, it may be a lot faster. The timing's not critical here, so let's overlook that for now.

Finally, PORTD &= ~_BV(PD4); turns pin PD4 off without affecting the rest of the values in PORTD. Let's look in detail at how it does it.

_BV(PD4) creates a binary number with the 4th bit (from the right) as a 1 -- 00001000. "~" is the logical complement operator. It turns the 00001000 into 11110111. "&" is the bitwise "and" operator. It compares two bits, and returns a 1 if and only if both bits are 1. If either is 0, it returns a 0.

The "&=" in PORTD &= ~_BV(PD4) is a very common shortcut. It stands for PORTD = PORTD & ~_BV(PD4). This last command compares the current value of PORTD

(00001000) and the value (11110111) described above. The zero in the 4th place in ~_BV(PD4), when used with the & operator, always makes the 4th bit of the result = 0, effectively turning off bit PD4.

The 1s in the rest of ~_BV(PD4) make it so that the & operator doesn't clobber the rest of the contents of PORTD. Since the & returns 1 only if both inputs are 1, the remaining bits in PORTD are re-assigned whatever value they already have -- leaving them unchanged.

Could we have set PORTD = 0? Sure. Since we're only using the one pin in D, it would turn it off just fine. But it would have the side-effect of turning off _all_ the pins on port D, and it wouldn't have provided such a nice example. The bit-masking techniques (&= ~_BV() and its opposite, |= _BV()) are pretty useful to learn for chip-level programming.

The last part of the code, return(0), never gets reached. The while(1) command ensures that the chip is always going to be stuck in the while loop. I just included the return() command because the compiler complained when I didn't include it -- the main C function in a program is always supposed to have a return value, even if the chip will never get there.

That's a lot of programming for one day. Take some time to admire the simple beauties of the flashing LED.

step 11: The End & Web Resources

So there you go -- your first AVR application.

I'll post up some more tutorials and code showing off different aspects of the ATtiny2313. At least one for the Analog/Digital converter, some on Timers, and one for using its built-in serial interface.

In the meantime, if you want to learn more about the AVR chips, here's some good web resources:

AVR Freaks is the motherlode: A community of friendly users with a forum.

Cornell's EE476 class webpage is a tremendous source for info, and their final projects are a treasure-trove of crazy, cool project ideas, all well-documented.

Psychogenic.com is good for AVR & Linux.

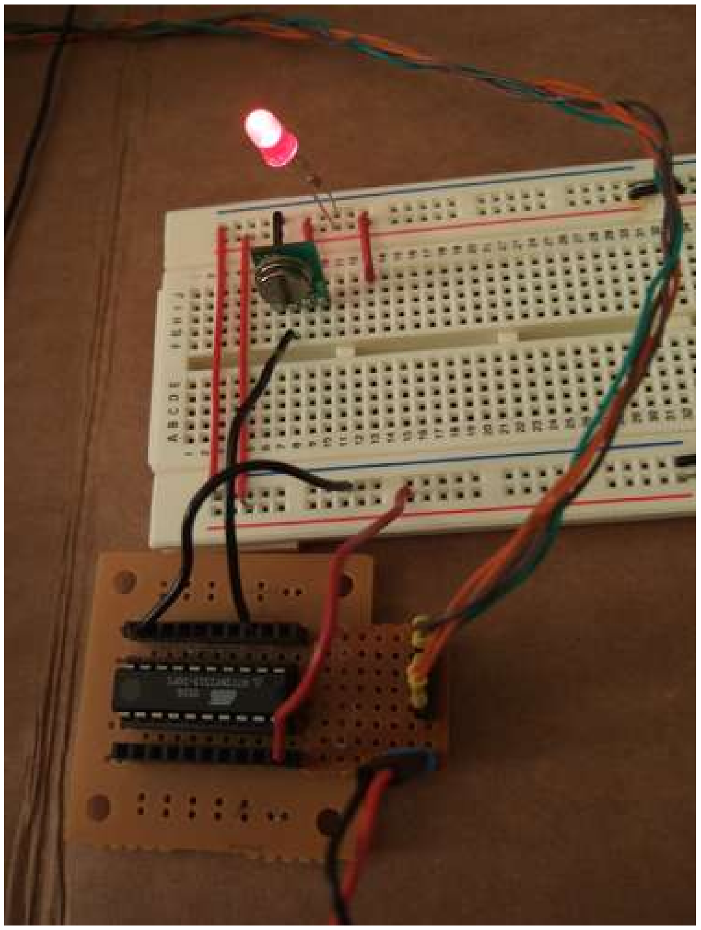
The AVR-libC demo program is not a bad one to learn from either, but maybe a bit advanced if you're just beginning with microprocessors or C.

Just found Building a USB Sensor Interface which has good instructions, and heads in the direction of USB connectivity.

Dec 2 Update: If you run Linux with KDE and want an integrated GUI environment, have a look at KontrollerLab. It's very similar-looking to AVRs AVR Studio for Windows.

Jan 2, 2007: AVR Tutor has a short getting-started-with-AVR tutorial written in C. AVR Beginners has a very complete tutorial, but it's based on assembly language instead of C, so I'd consider it an advanced tutorial. You might consider looking at them in order.

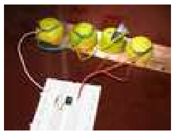
The photo is of me using the ghetto dev kit to run a 2400 baud wireless transmitter. Takes only a couple minutes to run a few wires to the breadboard, and off you go!



Related Instructables

Help: An Absolute Beginner's Guide to 8-Bit AVR Programming- AVR Dragon by PopSci

Getting started with ubuntu and the AVR dragon by }{itch


Tiny AVR Microcontroller Runs on a Fruit Battery by Gadre

Cellphone operated Robot by Nitin Sharma

An Electronic, Battery-less Dice (video) by Gadre

Electronic Tic-Tac-Toe with RGB LEDs (video) by cedtlab

Annoying Beeper by samirsky

How to use the Dragon Rider 500 with your AVR Dragon by barney_1

Comments

50 comments **Add Comment** **view all 266 comments**

ice877 says: Jan 9, 2009. 4:46 PM **REPLY**
your instructables have been such a help for getting started in avr but i am having a problem with the buttons. ima send you a pm asking you a few questions if you dont mind :D

p2otoole says: Dec 27, 2008. 12:00 PM **REPLY**
How is it question
How is it that I need an STK200/300 ISP programmer(AVR) *(bought and paid for)* to program an Atmega32, while this system can use the straight thru connection from PC parallel port to AT2313? Will my STK200/300 ISP programmer work on this system ?

dannytemmerman says: Dec 25, 2008. 1:58 AM **REPLY**
Can i also use a ATMega8 instead of the ATtiny2313?

Nightmare Incarnate says: Dec 22, 2008. 3:03 PM **REPLY**
Sweet, I finally got it working!

Does your Instructable mention that, unless the makefile is edited, the programming cable must be removed in order for the microcontroller to run? I was stumped by that for months.

luisnperez says: Nov 18, 2008. 8:02 AM **REPLY**
Wait a minute! So you program the chip to do something (flashing the LED) and then you DO NOT NEED the PC anymore to run the programmed chip? in other words, just switch the power and the LED will flash in loop?
Please be more specific for newbies like me!

rearthur2003 says: Nov 25, 2008. 10:48 AM **REPLY**
yes thats all in short the microcontroller is a mini computer

luisnperez says: Nov 19, 2008. 11:45 PM **REPLY**
Frustration! doesn't work , message:
*** No rule to make target `LED_Demo.o'. Stop
The programminf needs more detail, specially you mentioned to modify the txt file, but what line?
In windows there is no "Tools...Make Program"

rearthur2003 says: Nov 25, 2008. 10:48 AM **REPLY**
he should be using linux with the avrdude for windows user you can download winavr or the avr studio4

luisnperez says: Nov 18, 2008. 6:13 AM **REPLY**
I'm using the AVR to run a mini-step-motor, How do I run 4 motors using the same AVR? Any ideas?

Where is the usb version ?

sysadmn says: Nov 21, 2006. 8:40 AM **REPLY**
Stoopit newbie question - is the negative lead of the LED the longer or shorter one? Does it matter which lead you use if you are consistent? I haven't done hardware in 25 years! It's great to get started again.

Koil_1 says: Nov 17, 2008. 9:40 AM **REPLY**
the ground is usually the the one attached to the larger piece in the LED. That is if the lens is clear enough to see the actual parts inside. I've noticed that not all LEDs have a flat spot on the rim. Further more some are square which kinda defeats the flat spot thing. Seriously though, it might be a noob question but it wasn't stupid. The only stupid question is the one you don't ask. Don't be so hard on yourself OK.

The Real Elliot says: Nov 23, 2006. 12:53 AM **REPLY**
When they come out of the box, longer is positive.

Something I just learned recently -- the negative has a slight flattening on the round disc at the bottom of the plastic lens. Useful after you've trimmed the leads.

My convention is to put the resistor on the negative end, but it doesn't really matter as long as the resistor's in the circuit somewhere.

And you can really just try it out. If there's light, it's right.

comwiz says: Mar 4, 2008. 4:28 PM **REPLY**
Where did you get the AVR socket? I can't find it on SparkFun's website.

Koil_1 says: Nov 17, 2008. 9:33 AM **REPLY**
try looking at MouserMouser's website. They have a lot of different prototyping componants. They also have no minimum order which is really nice.

The Real Elliot says: Mar 5, 2008. 10:58 AM **REPLY**
The socket is just a standard DIP socket for most of the AVR's.

The exceptions are the 28-pin+ ones (Mega 8 and up) which use the slightly rarer thin-DIP standard. For those, I just use two 14-DIPs stacked close to each other and it works great and saves a few pennies.

Sigsaucer647 says: Nov 16, 2008. 10:55 AM **REPLY**
hey, just wonderin, what can these things do?
like what would you use 'em for?

The Real Elliot says: Nov 16, 2008. 3:28 PM **REPLY**
What can't they do is more like it -- too many things to list here. Have a look at Cornell's EE Class Page or this link for inspiration, among many, many others.

botmaster 10 says: Oct 5, 2008. 1:30 PM **REPLY**
Could I use THISinstead? (I already bought one today)

JZ Price says: Nov 16, 2008. 9:38 AM **REPLY**
Rs232 serial

The Real Elliot says: Nov 16, 2008. 3:20 PM **REPLY**
Like the other guy said, that's a serial connector. There _are_ serial-port programmers, and you can google around for them.

That's not a bad way to get started, but not as simple as the parallel port.

disturbedreaper says: Oct 19, 2008. 4:49 PM **REPLY**
can i do this with a usb instead because i do not have that port on my laptop thanx.

Derinsleep says: Nov 9, 2008. 8:28 AM **REPLY**
you should then use the upgraded version with usb

purpulhaze says: Oct 21, 2008. 1:17 PM **REPLY**
Is tapping into the usb as a power source safe for the chip? I know you posted 5v but I believe usb is rated a lil over 5v.

The Real Elliot says: Oct 21, 2008. 2:38 PM **REPLY**
Yup. Totally safe. I do it all the time, for hours at a time. Chip doesn't even get a little hot.

junits15 says: Oct 19, 2008. 7:34 AM **REPLY**
do i have to use this kind of microcontroler for this? or can i use something else?

nex_otaku says: Oct 14, 2008. 6:47 AM **REPLY**
I got the LED blinking now, that was some expirience... I'm totally new to micros and electronic stuff.
Thanks for the great instructable!
I've encountered two problems, and posting here for future newbies like me:

1) I took power supply from PC (without it being connected to the motherboard, just only to power line), and the current was too high. Chip din't die, it just switched off. Spent hours to find out what is wrong, so be awared!
I bought special power, now it runs flawlessly.

2) lpt1, lpt2 and lpt3 in a Makefile doesn't reflect actual LPT1, LPT2 and LPT3 in Windows! It stands for output address, like 0x3BC(this for lpt3).
See what output address is **your** LPT, select appropriate name and write it in Makefile.

lpt1 0x378
lpt2 0x278
lpt3 0x3BC

ComradeLynx says: Sep 14, 2008. 12:30 PM **REPLY**
Can i use ATMEL ATTINY2313V-10PU ?? ??

Is it possible to program traffic lights from LEDs, but replace the LEDs with relays so it will control a real traffic light by relays? Does it need to be connected with computer to work?

Sorry for nooby questions, but i just discovered microcontrollers :P

frontier says: Sep 15, 2008. 3:53 PM **REPLY**
yes that is possible, all you need is a relay that will clap with the 5v power the avr outputs.
and nope, it dosent have to be connected to a computer to work, only when you program it, afterwards you can throw your computer out the window if you please, and it'll keep working.

frontier says: Sep 15, 2008. 2:58 PM **REPLY**
got a power question here.
for the 5v regulated power supply the chip and this board needs. would the 5v from a usb port do?

I'm thinking taking a usb extension cord, plugging one end into my computer, and stripping the end off the other and using the +v and gnd to plug into this board.

would this burn out the avr chip or my usb port?

twenglish1 says: Sep 5, 2008. 9:22 PM **REPLY**
hey got a problem i just had this working on an attiny13 and i got some attiny2313 and i made a new programmer for them(the same way you made yours) but i keep getting the error: device not responding. i have power hooked up to it and im pretty sure the chip is not dead any idea what the problem is??

doctek says: Sep 6, 2008. 2:54 PM **REPLY**
You did make sure to change the processor type in the Makefile, right?

Just checking the obvious.

twenglish1 says: Sep 6, 2008. 7:07 PM REPLY
i think thats what i forgot to do ill let you know

Jeebiss says: Sep 1, 2008. 8:18 PM REPLY
So, I was looking at a 6 pin avr, i think tiny13.
I was wondering if that would still work with this set up.
And, what do MOSI, MISO, SCK. RESET, stand for?

The Real Elliot says: Sep 2, 2008. 9:27 AM REPLY
Off to to manufacturer's datasheets you go!

Tiny 13 Pinouts are on page 2, and you can read all about it in the serial programming section.

In short, though, they're the names/functions of the various lines that are used in the serial communication between chip and programmer.

MOSI = Master Out, Slave In. MISO = the opposite. SCK is the Serial ClocK. And I have know idea why I typed reset in all-caps.

Jeebiss says: Sep 3, 2008. 6:02 AM REPLY
So, I would just use the same pins on the parallel conenctor and just align them with the tiny 13's, MOSI, MISO, and so on?

The Real Elliot says: Sep 3, 2008. 10:49 AM REPLY
Yup. (Honestly, the cradle's a little bit fancy.)

You can skip all that and just connect the wires from the parallel port directly to a breadboard that has your chip in it. I should really take photos of doing this and post them up. Or perhaps you could?

Jeebiss says: Sep 5, 2008. 5:12 PM REPLY
I would if I had a bread board.
I plan to do something similar to yours.

The Real Elliot says: Sep 5, 2008. 8:15 PM REPLY
Cool. It does kinda work like a mini breadboard. For extra fancy-points, you could also add a whole row of headers dedicated to +Vcc and to ground, which would make hooking up components even easier...

But yeah, the basic idea of the cradle is just to get the parallel port lines to the right pins on the chip.

KT Gadget says: Sep 1, 2008. 12:04 AM REPLY
one question is the smaller board the same as the bigger one? cause i see the pins are in a different arrangement. just checking on the diagram for the connections.

btw nice ible im gonna make this

---

digitalrox says: Aug 31, 2008. 9:47 AM REPLY
Thanks a lot this is super useful :D

---

snuffybox says: Aug 25, 2008. 7:14 PM REPLY
Yes!!!!!!!!!!!!!!Blinking LED!!!!
I got it to work!!!!!

---

wee_man says: Aug 1, 2008. 3:23 AM REPLY
Hello now i need to ask will this work with any 20pin micro controller because i cant find any avr ones where i live

cheers

The Real Elliot says: Aug 2, 2008. 10:58 PM REPLY
Not sure, I've only ever used the AVR's.

For it to work, you'd need two essential things. One: a chip that allows for flashing its memory over a regular serial interface. Two: a program for your computer that'll put out the necessary serial messages to do the flashing.

---

wee_man says: Aug 3, 2008. 1:34 AM REPLY

ok but i think i will try and find some AVR's or i might have to order some. Which might be a pain but with AVR's being so common it might be easyer to program it

solidacid says: Jul 11, 2008. 4:46 AM REPLY

OK, i figured out where to solder +5V to and i got it to program the chip successfully but, no flashing :(any ideas what i might be doing wrong?

The Real Elliot says: Jul 17, 2008. 8:43 PM REPLY

Hmmm...

Things to check:

1) Does the LED work? Do you have the polarity right? Plug it in (with resistors) to a known-good 5v source to check. If you've run it without resistors at voltages higher than 2-3v, it might have burnt out.

2) Did you enable the pins for output? If you have substituted PDx for PBx, make sure you've changed the DDRx = _BV(PDx) part of the code (where x is whatever pin you've got the LED on).

3) Are you sure it's programming? If you can test with avrdude from the command-line, it's pretty helpful. I usually use something like "avrdude.exe -p t2313 -c dapa -P com1 -n -v" to test (on Windows).

solidacid says: Jul 19, 2008. 12:33 AM REPLY

1)the LED works, and I tried other LEDs to be sure.

2)I copypasta'd you code

3)I did program it from the command-line, it replied that the programming was successful

twenglish1 says: Jul 22, 2008. 1:23 PM REPLY

what chip are you using

solidacid says: Jul 15, 2008. 3:51 PM REPLY

I'm using a ATtiny2313 like the tutorial

twenglish1 says: Jul 12, 2008. 5:54 PM REPLY

what avr are you using i used an attiny13 and i needed to change the code a little

view all 266 comments