micah carrick

Django, Python, GTK+, PHP, and a pinch of salt.

g+1 3 Tweet 4 Like 10

Aug 14, 2012

AVR Tutorial

A Hobbyist's Guide to Hacking 8-Bit AVR Microcontrollers

< Introduction Table of Contents AVR C Programming Basics >

Getting Started: Blinking an LED

Blinking an LED is the "Hello World" of programming microcontrollers. It is a great way to work through the entire development process and make sure all your tools are in working order.

In this chapter you will wire up a light emitting diode (LED) to an AVR on a breadboard, write a C program to blink that LED, and then program that code into the AVR.

2.1 List of Materials

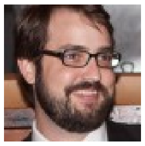
The following items are needed to complete the Blink LED project. Links are provided to purchase the items at Jameco Electronics. See the Tools & Equipment section for a general list of items used throughout this tutorial series.

- One of the following AVR Microcontrollers: ATmega328, ATmega168, ATmega88, or ATmega48. An older ATmega8 or similar may also work but has not been tested.
- An ISP AVR Programmer that is supported by avrdude.
- Solderless breadboard
- Hookup wire
- 10kΩ resistor (1), 470Ω resistor (1)
- 0.1uF capacitor (1)
- LED (1)

If you have a AVR microcontroller that has been previously used then it's "fuse bits" may have been modified. You will need to restore it's fuses to factory defaults to complete this tutorial.

2.2 Wiring The Circuit

The schematic diagram for the Blink LED project is shown in Figure 2.1. A photo of the circuit wired up on a breadboard is shown in Figure 2.2.



Hello, my name is Micah and I write software. I love Python, GTK+, Linux, and robots.

Email Facebook Twitter

Github RSS Feed

LinkedIn Stack Overflow

Need some work done? Hire me!

Categories

Android Programming

Django

Game Programming

GNOME

GTK+ Programming

Linux

Python Programming

Robotics & Electronics

Web Development

Tutorials

Autotools Tutorial for Python and GTK+

AVR Tutorial

GTK+ 2 / Glade Tutorial

Recent Posts

No Package Error with Jhbuild in Ubuntu 12.10 64

Show git Branch in Shell Prompt

JSON Validation and Formatting in Gedit

Raspberry Pi WiFi

Django 1.5, Python 3.3, and Virtual Environments

Django Contact Form with reCAPTCHA

Notes on Serving Django Apps with uWSGI

Getting an IP Address in Django behind an Nginx Proxy

SSL behind an Nginx Reverse Proxy in Django 1.4

Using libgdx without Eclipse

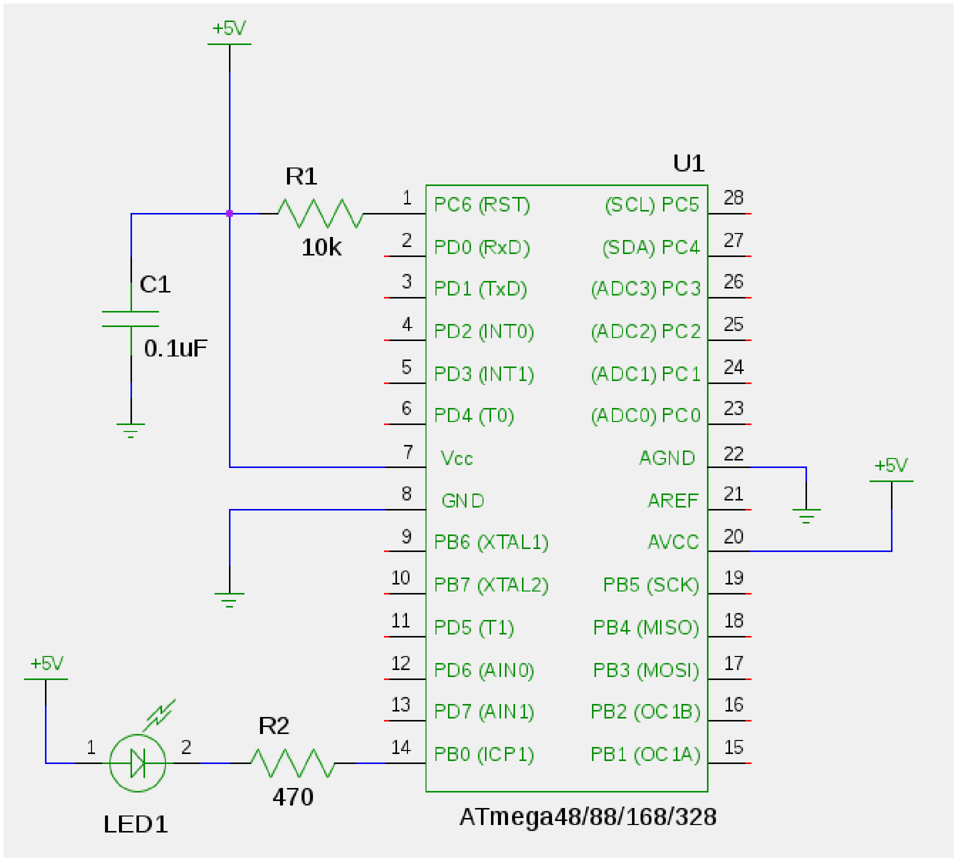


Figure 2.1 - The Blink LED schematic diagram

If you are not familiar with wiring integrated circuits (ICs), the AVR chip have a little notch at one end to designate the "top". The pin numbering starts at 1 on the top left, progress squentially down the left side, and then back up the right side as shown in the schematic above.

A 10kΩ "pull-up resister" (`R1`) is connected to the `RESET` pin to keep the pin in a digital high state.

A 470Ω resistor (`R2`) and an LED (`LED1`) are connected to pin `PB0`. The resistor is a "current limiting" resistor to limit the current (mA) passing through the LED so that it doesn't burn out. When `PB0` goes low (0V) the LED will turn on. When `PB0` goes high (+5V) the LED turns off.

A 0.1uF "bypass capacitor" (`C1`) helps prevent noise or "ripple" on the +5V line from effecting the microcontroller.

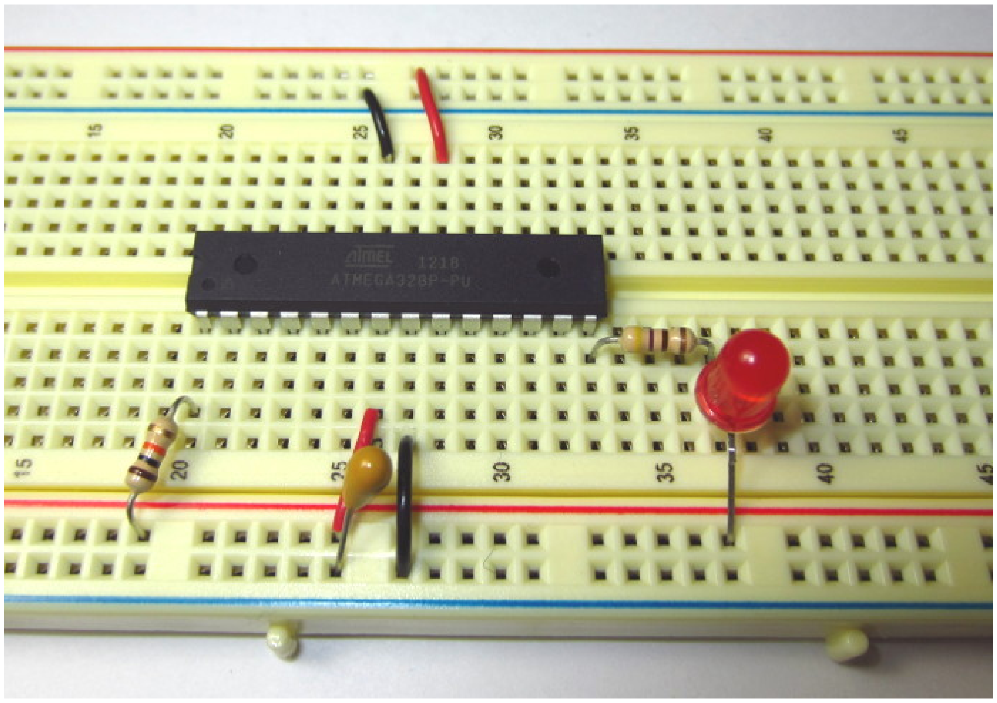


Figure 2.2 - Close-up View of the Blink LED circuit on a breadboard.

2.3 Powering The Circuit

The schematic shows a DC power source of +5V to power the microcontroller, however, it's up to you to provide that +5V. One of the most common ways hobbyists use provide +5V to projects is to use a 9V battery with a 7805 voltage regulator IC to drop the voltage down to +5V.

I prefer to use 4 rechargable AA batteries (4 batteries x 1.2V ea. = 4.8V). You can pick up rechargable AA batteries and chargers just about anywhere these days. You can wire them up to you breadboard using a 4-AA Battery Holder along with a 9V Battery Clip as shown in figure 2.3.

Of course, having a +5V power source in your "lab" (whatever that may be) is invaluable. Hobbyists often get +5V out of a wall outlet by using an expensive lab power supply (which is awesome if you can afford it), using a +5V AC/DC wall adaptor, using the +5V line of an ATX computer power supply, or tapping the +5V line from a USB bus or USB charger.

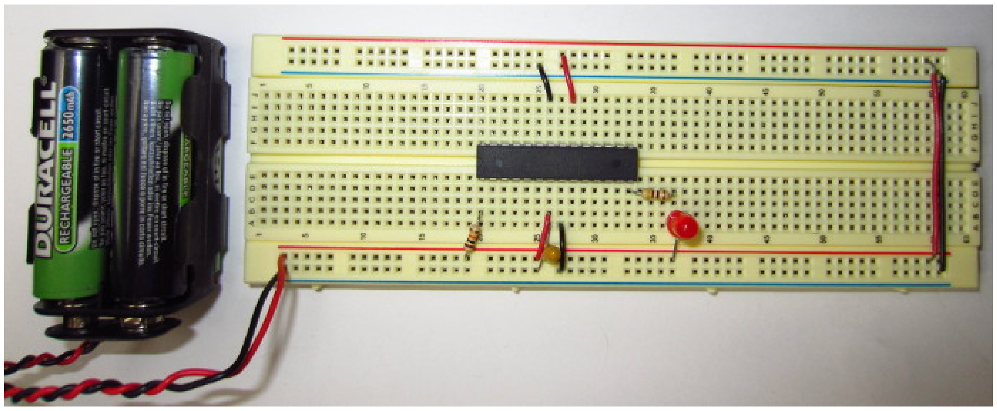


Figure 2.3 - The Blink LED circuit on a breadboard

2.5 Compiling the C Program

The C program must be compiled into an Intel HEX formated file. This is the format that is passed to `avrdude` which tells the AVR programmer what to program into the microcontroller. The HEX file is created in two steps.

First, `avr-gcc` will be used to compile the C program `blink.c` into a binary ELF (**E**xecutable and **L**inkable **F**ormat) object file. This new file will be named `blink.elf`.

```
avr-gcc -mmcu=atmega328p -Wall -Os -o blink.elf blink.c
```

- `-mmcu`, tells the compiler which AVR microcontroller the code is being compiled for. If you are not using an `ATmega328P` then you will need to change it to the AVR that you are using.
- `-Wall` turns on all reasonable compiler warnings which will help make sure you're writing good code. Any time you see a warning you will want to investigate what it means.
- `-Os` is the optimization flag which tells the compiler to optimize the code for efficient space utilization.
- `-o` argument specifies `blink.elf` as the output filename.

Next, `avr-objcopy` copies specific sections of binary data from the `blink.elf` file into a new Intel HEX format file. The new file will be named `blink.hex`.

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

- `-j` specifies Memory Sections to copy from the ELF file. It is used twice, once to copy the `.text` section and once for the `.data` sections. The `.text` section contains the machine instructions which make up the program. The `.data` section contains various static data used in the program.
- `-O ihex` option specifies Intel HEX as the output format.
- `blink.elf` is passed as the input file and `blink.hex` is specified as the output file.

The Blink LED program doesn't actually use the `.data` section, but most projects will. Including it now won't hurt anything.

The `blink.hex` file is now ready to be programmed into the AVR.

2.4 The Blink LED Program

As was described in Overview of the Development Process, the "firmware" which tells the AVR microcontroller what to do is written in C. The GNU Toolchain is used to compile that code into a file formatted for programming into the AVR.

The C program for Blink LED is shown below.

```
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

int
main (void)
{
    DDRB |= _BV(DDB0);

    while(1)
    {
        PORTB ^= _BV(PB0);
        _delay_ms(500);
    }
}
```

This code configures the PB0 pin on the AVR microcontroller as a digital output. Then, it toggles the state of that pin every 500 ms. For now, just copy and paste this code into a file named `blink.c`.

2.7 Programming the AVR Microcontroller

Now for the moment of truth. The `blink.hex` file will now be programmed into the microcontroller using the `avrdude` program. Make sure the AVR programmer is connected to your circuit and to your computer and that the circuit has power. Then you can run the `avrdude` command:

```
avrdude -p m328p -c usbtiny -e -U flash:w:blink.hex
```

- `-p` specifies the AVR microcontroller part number being programmed. If you are not using an `ATmega328P` then you will need to specify the AVR you are using.
- `-c` specifies the AVR programmer you are using. If you are not using a USBtinyISP then you will need to specify the AVR programmer you are using.
- `-e` erases the chip before writing the new contents
- `-U flash:w:blink.hex` performs a memory operation: `flash` is the type of memory, `w` is for a "write" operation, and `blink.hex` is the name of the file to write. In other words, "write blink.hex to flash memory".

If the AVR microcontroller was successfully programmed then the LED should start blinking about once per second and the output of `avrdude` should look like the following:

```
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0

avrdude: Device signature = 0x1e950f
avrdude: erasing chip
avrdude: reading input file "blink.hex"
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: writing flash (200 bytes):

Writing | ################################################## | 100% 0



avrdude: 200 bytes of flash written
avrdude: verifying flash memory against blink.hex:
avrdude: load data flash data from input file blink.hex:
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: input file blink.hex contains 200 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0



avrdude: verifying ...
avrdude: 200 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Congratulations, you have now programmed an AVR microcontroller!

2.6 Connecting the AVR Programmer

The AVR microcontroller will be programmed, in it's circuit, right where it is on the breadboard. This is known as **I**n **S**ystem **P**rogramming or ISP. Connecting the AVR Programmer's ISP cable to the microcontroller on the breadboard can be a little confusing at first, so bear with me.

There are 2 standard ISP connections. A 10-pin and a 6-pin connection. The AVR Programmer typically has a 6 or 10 pin ribbon cable comming out of it. This cable is meant to be connected to what's known as a "shrouded header" on a PCB (printed circuit board). The ISP cable does not directly plug directly into a breadboard without some help.

The cheap and simple way to connect the ISP cable to the breadboard is to simply stick hookup wire into the ISP cable connector and then to the corresponding pins on the AVR microcontroller.

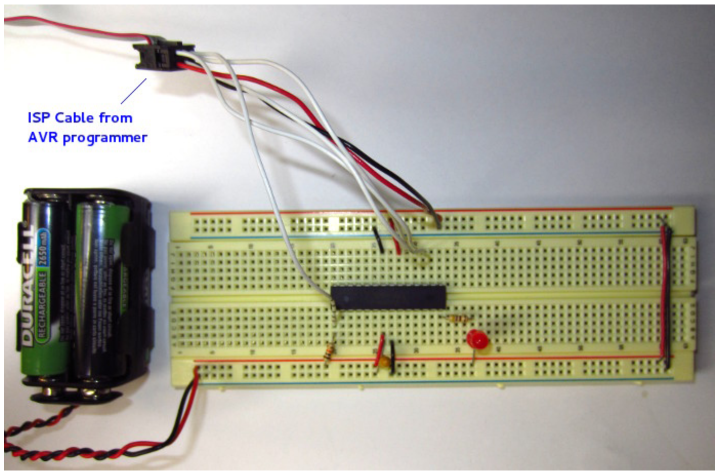


Figure 2.4 - ISP cable wired to breadboard using hookup wire

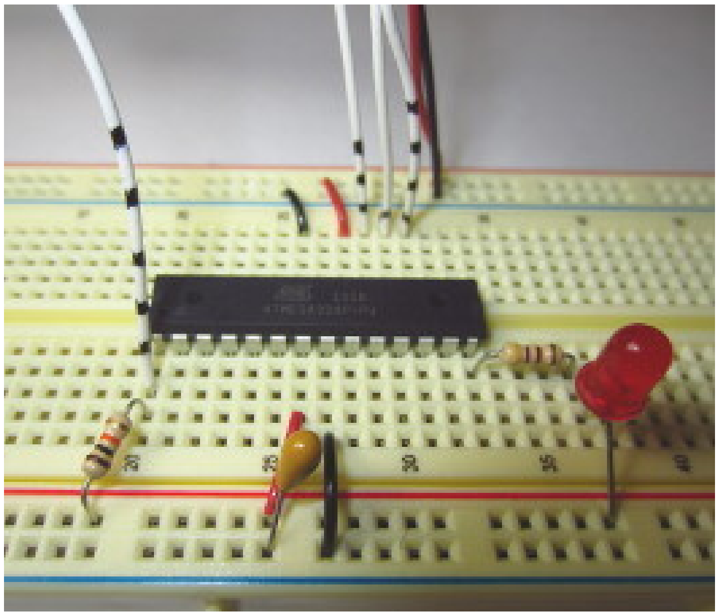


Figure 2.5 - Close-up view of wires (white) coming from ISP cable

In order to connect the ISP cable, a pinout diagram is used to determine which pins of the ISP connection correspond to which pins on the AVR microcontroller. The pinouts for the 6 and 10 pin ISP connectors are shown in Figure 2.6. The square pin in the pinout depicts `pin 1`. The red conductor on a ribbon cable also depicts `pin 1`.

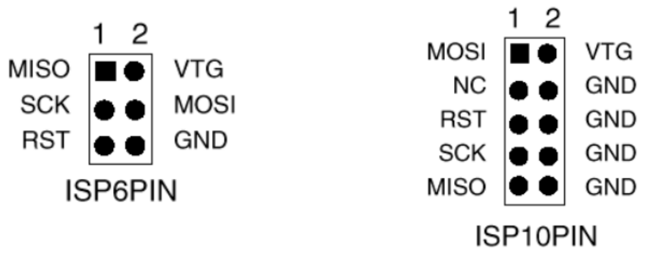


Figure 2.6 - 6-pin and 10-pin AVR ISP pinouts

The trick to this is that the pinout is for the header, not the cable. In other words, it depicts the pins of the header that the cable would plug into, as would be seen from above. If you are looking into the plug-end of the cable, you are looking at it flipped over (as if from underneath the PCB) as shown in Figure 2.7. It's easy to inverse the connections if you do not understand the pinout diagram.

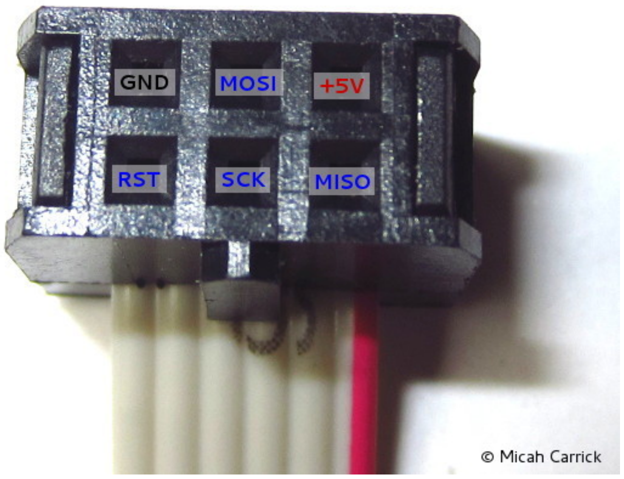


Figure 2.7 - Labeled ISP conductors looking into plug-end of a 6-pin cable

Using Figure 2.7 as a reference, connect hookup wire from the ISP cable to the corresponding pins of the AVR microcontroller on the breadboard.

ISP Conductor	AVR Pin
+5	Pin 7 (anywhere on the +V line)
GND	Pin 8 (anywhere on the ground line)
RST	Pin 1
MOSI	Pin 17
MISO	Pin 18
SCK	Pin 19

The pinout for the AVR microcontroller on page 2 of the datasheet was used to determine the ISP pins on the AVR microcontroller.

2.8 Troubleshooting

If the AVR programmer's ISP cable is not properly connected to the AVR microcontroller or the circuit does not have power, avrdude will show an error like this:

```
avrdude: initialization failed, rc=-1
         Double check connections and try again, or use -F to overrid
         this check.
```

Next: AVR C Programming Basics >

Did you enjoy **Getting Started: Blinking an LED?** If you would like to help support my work, A donation of a buck or two would be very much appreciated.

Donate

AROUND THE WEB WHAT'S THIS?

Men's Life & Health
Controversy Over New Steroid Alternative

Inc.
13 Common Email Phrases--and What They Really Mean

ALSO ON MICAH CARRICK

Show git Branch in Shell Prompt 2 comments

Django 1.5, Python 3.3, and Virtual Environments 1 comment

JSON Validation and Formatting in Gedit 1 comment

WiFi on the Raspberry Pi 4 comments

17 Comments Micah Carrick Login

Sort by Newest Share Favorite

Join the discussion…

nick • a month ago
need avr isp usb Driver for avr studio
• Reply • Share ›

Ahmad 'ukon' Syukron • 5 months ago
thank you for the tutorial,

I'm using your schematic picture for my tutorial in Indonesian, if you don't mind, (please let me know if you do, I will put it down).
• Reply • Share ›

jamesbeedy • 8 months ago
Any ideas on how we can alter this to automate the downloading and installing of required 3rd party python packages needed by the application. I'm thinking a call to pip and reference to a requirements.txt........but where?
• Reply • Share ›

jamesbeedy • 8 months ago
Thanks for the great tutorial!
I wrote a simple python script to automate the processes you describe here at:
https://github.com/jamesbeedy/...
I referenced you and linked to this page as my inspiration.

Thanks again!
• Reply • Share ›

san • 8 months ago
In section 3.1, you use 'hello' as a subdir of 'src', but use 'helloworld' as a subdir in later sections. In section 3.4, the second 'src/Makefile.am' subheading should be 'src/hello/Makefile.am' or 'src/helloworld/Makefile.am'.
• Reply • Share ›

Ade Malsasa Akbar • 10 months ago
Thank you, although I don't use your tutorial at all, but I am very glad now to see my Ubuntu (using avrdude) can download .hex program (using usbasp) to ATMEGA chip. Now I understand the command avrdude -p m328p -c usbtiny -e -U flash:w:blink.hex you have hentioned above. Now I save this page for my further learning. Thank you.
• Reply • Share ›

Brian Killian • 10 months ago

Thank you for this excellent tutorial. In section 3.1, you use 'hello' as a subdir of 'src', but use 'helloworld' as a subdir in later sections. In section 3.4, the second 'src/Makefile.am' subheading should be 'src/hello/Makefile.am' or 'src/helloworld/Makefile.am'.

1 • Reply • Share ›

akoskovacs • 10 months ago

The best AVR begginer tutorial ever! Thank you so much!

• Reply • Share ›

ashith babu • 10 months ago

thank u very much sir ..i have programmed my atmega8 succesfully..very usefull tutorial..

1 • Reply • Share ›

Shokhrukh Shomurat • a year ago

Thank you for Tutotial!!! I followed the steps and everything worked great!

• Reply • Share ›

Diego Espirito Santo • a year ago

I finished this tutorial perfectly but The led is not blinking...
does anyone have an idea of what's the problem?

1 • Reply • Share ›

Guest • 2 years ago

Hmmm, looks like having the Makefile as a dependency of the application ensures that changes in the Makefile.am will cause the Makefile to be rebuilt before the application is built. That's pretty important given that changes in the Makefile will likely effect how the application is built :)

• Reply • Share ›

me ➦ Guest • 2 years ago

woops, tried to delete my comment so I could put it in a thread with my original question. Guess 'delete' is code for 'repost as Guest' on this Disqus thing. Awesome.

• Reply • Share ›

me • 2 years ago

Great info. One detail that's still a mystery to me: why does the make rule for your application include the Makefile as a dependency? I realize the automake documentation uses this exact pattern but they don't say why this is necessary. I've found that having 'hello-world' depend on only the template 'hello-world.in' works for my purposes. Is there an edge case I may run into by omitting the 'Makefile'?

• Reply • Share ›

Jonathan • 2 years ago

This was an amazing and very helpful tutorial. Thank you so much!

• Reply • Share ›

VENKATA AMRUTH POLAVARAPU 11BE • 2 years ago

my red led is not blinking which is present on my microcontroller, it blinked previously when i used it but now its not

• Reply • Share ›

studyembedded • 2 years ago

A good tutorial...thanks

• Reply • Share ›

Subscribe Add Disqus to your site Privacy