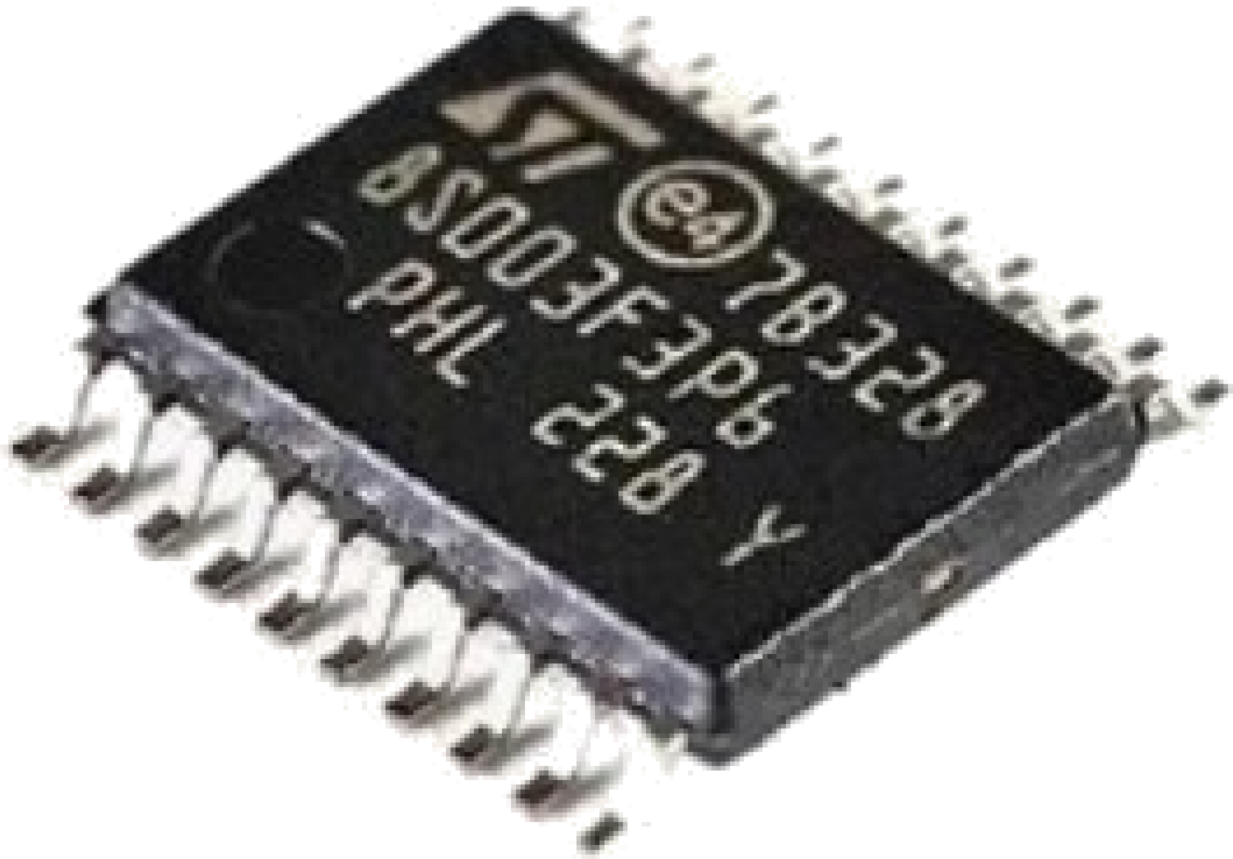
Hamza BENDALI BRAHAM FOLLOW

Getting started with STM8 Development Tools on GNU/LINUX

Published Oct 17, 2019

> *This guide was written for (and on) **Linux Arch, Ubuntu, Mint.** I have not tested the procedure on other Linux architectures or distros, Users of other distros may find that some of the instructions don't work verbatim. Adapt as needed.*



STM8 Series of micro-controllers by ST Microelectronics are dirt cheap and powerful at the same time. Their processing power is nothing short as of Arduino, while at the same time their power consumption is much less. These properties makes STM8 micros a great choice for many hobby and serious projects. This tutorial you will learn how to setup a developing and programming environment for STM8 on Linux based systems.

Here are the required tools that I used in this tutorial:

- **VS-Code** (Visual Studio Code) advanced text editor.
- **SDCC** Compiler v3.5.0 or higher.
- **STM8 Standard Peripherals Library [SPL]**, patched for SDCC.
- **ST-LINK / STM8FLASH** to write your compiled code into the micro-controller.
- **STM8-GDB / OPENOCD** for debugging.
- **ST's STM8 Evaluation board** or you can get away with el-cheapo chinese boards which are going for around 2$ including shipping!.

Prepare required tools

Install VS-Code

Visual Studio Code is a cross-platform, free and open-source (licensed under the MIT License) text editor developed by Microsoft and is extensible using extensions, which can be browsed from within the text editor itself (via its extension gallery) or from https://marketplace.visualstudio.com/VSCode. While open-source, a proprietary build (licensed under an End-User License Agreement) provided by Microsoft is available and used as the basis for the visual-studio-code-bin [**AUR**] package (for an explanation of the mixed licensing, see this GitHub comment).

Installation

```
# Arch linux
$ yaourt -S visual-studio-code-bin

# Ubuntu and Mint linux
$ wget https://go.microsoft.com/fwlink/?LinkID=760868 -O vscode.deb
$ sudo dpkg -i vscode.deb
```

Usege

```
$ code
```

press (Ctrl + Shift + X) then search and install the folowing extention:

- C/C++ for Visual Studio Code
- C++ Intellisense
- hexdump for VSCode
- vscode-devdocs

Install SDCC

Download and install SDCC Snapshot Builds for more optimisation from SourceForge

```
# download the latest version
$ wget https://sourceforge.net/projects/sdcc/files/snapshot_builds/amd64-unknown-

$ tar -xjf ./sdcc-snapshot-amd64-unknown-linux2.5-20200113-11515.tar.bz2
$ cd sdcc
$ sudo cp -r * /usr/local
```

> *SDCC SourceForge*
> *Documentation SDCC Manual PDF*

Download STM8 Standard Peripherals Library [SPL]

SDCC supports STM8, but for licensing reasons (booo, ST!), the Standard Peripheral Library (SPL) is missing.

Someone developed a patch that makes the SPL compatible with SDCC, available here: SPL_2.2.0_SDCC_patch. There's an AUR package that attempts to install it in the SDCC libraries folder (aur/stm8-spl-sdcc), but alas the zip with the SPL files is login & EULA-click protected (booo again, ST!).

Programmer

ST-Link

ST-Link programmer or clone used to write your compiled code (Firmware) into the micro-controller.
For the programmer, you need one that support SWIM (Single Wire Interface Module) mode. You can (recommended) go with the original debugger of STMicroelectronics which is ST-Link V2 (you can get this one second hand as low as 20
)orifyouarereallywanttogoeconomical,youcangetawaywiththefakeoneswichcostyouunder1
(please note that these cheap debuggers only support software mode, which works fine, and do not give you full functionality and speed of the genuine debuggers of ST itself). or you can build your own Open source Stlink Tools.

> *ST-LINK/V2-1 firmware upgrade STSW-LINK007.*
>
> *Open source version of the STMicroelectronics Stlink Tools here*
>
> *Black Magic Probe, Open Source JTAG & SWD GNU Debugger and Programmer here and here*

STM8FLASH
it was the only program that's able to communicate through the SWIM interface of ST-LINKs to upload compiled code (Firmware) into the micro-controller.

> *libusb-1.0-0-dev is required to compile stm8flash*

Install **stm8flash** from [AUR] package

```
## Arch linux
$ yaourt -S aur/stm8flash-git

## Ubuntu, Mint Linux
## Install from src git repo
$ git clone https://github.com/vdudouyt/stm8flash.git
$ cd stm8flash
$ make
$ sudo make install
```

> *GitHub opensource software distributed on vdudouyt/stm8flash*

USB Troubleshooting
To solve USB device acquring write access problem

> *libusb: error [_get_usbfs_fd] libusb couldn't open USB device /dev/bus/usb/003/004: Permission denied*
> *libusb: error [_get_usbfs_fd] libusb requires write access to USB device nodes.*
> *Could not open USB device.*

create files with content:

49-stlinkv1.rules

```
# stm32 discovery boards, with onboard st/linkv1
# ie, STM32VL.

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="3744", \
    MODE="660", GROUP="plugdev", TAG+="uaccess", \
    SYMLINK+="stlinkv1_%n"
```

49-stlinkv2.rules

```
# stm32 discovery boards, with onboard st/linkv2
# ie, STM32L, STM32F4.

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="3748", \
    MODE="660", GROUP="plugdev", TAG+="uaccess", \
    SYMLINK+="stlinkv2_%n"
```

49-stlinkv2-1.rules

```
# stm32 nucleo boards, with onboard st/linkv2-1
# ie, STM32F0, STM32F4.

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="374b", \
    MODE="660", GROUP="plugdev", TAG+="uaccess", \
    SYMLINK+="stlinkv2-1_%n"

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="3752", \
    MODE="660", GROUP="plugdev", TAG+="uaccess", \
    SYMLINK+="stlinkv2-1_%n"
```

49-stlinkv3.rules

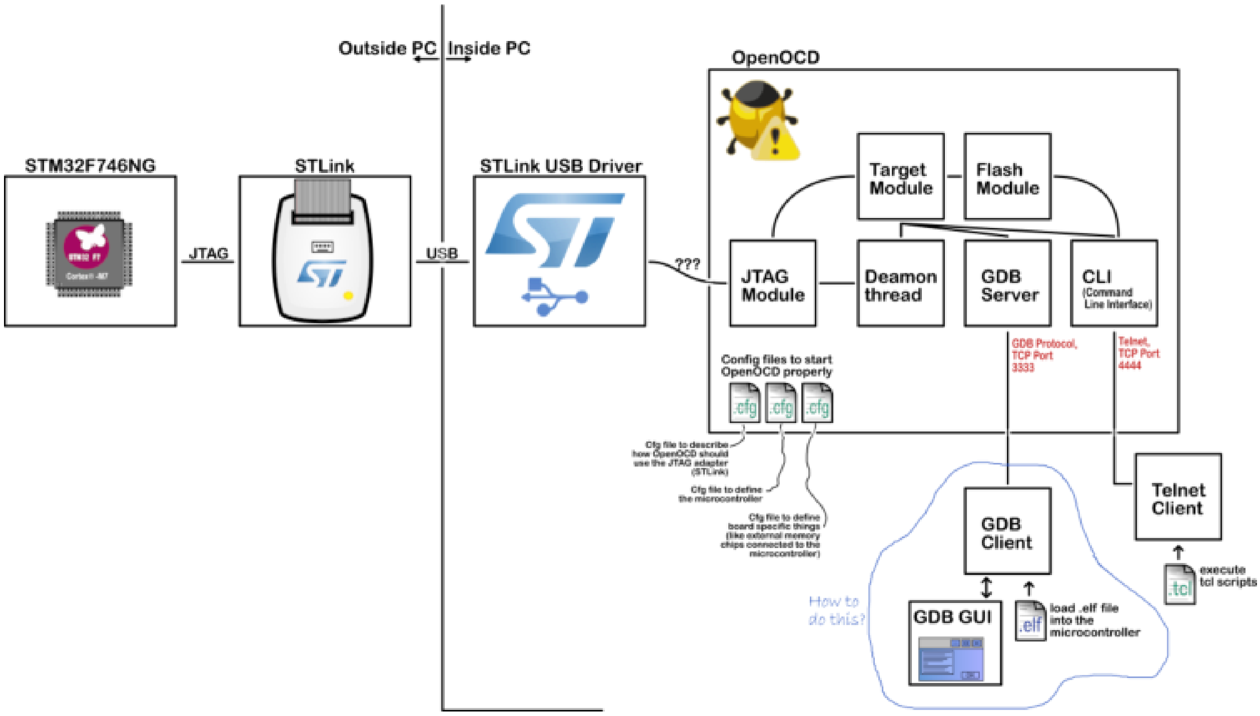
```
# stlink-v3 boards (standalone and embedded) in usbloader mode and standard (debu

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="374d", \
    MODE="660", GROUP="plugdev", TAG+="uaccess", \
    SYMLINK+="stlinkv3loader_%n"

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="374e", \
    MODE="660", GROUP="plugdev", TAG+="uaccess", \
    SYMLINK+="stlinkv3_%n"

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="374f", \
    MODE="660", GROUP="plugdev", TAG+="uaccess", \
    SYMLINK+="stlinkv3_%n"

SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="3753", \
    MODE="660", GROUP="plugdev", TAG+="uaccess", \
    SYMLINK+="stlinkv3_%n"
```

Create **49-stlinkv1.rules**, **49-stlinkv2.rules**, **49-stlinkv2-1.rules**, **49-stlinkv3.rules** and copy it in **/etc/udev/rules.d/**, Then reload **udevadm**

```
$ sudo cp *.rules /etc/udev/rules.d/
$ sudo udevadm control --reload-rules && sudo udevadm trigger
```

> *Note that a file is provided for ST-Link/V1 (idProduct=3744) despite most toolsets do not support it.*

GDB Debugger



GDB offers extensive facilities for tracing and altering the execution of programs. The user can monitor and modify the values of programs' internal variables, and even call functions independently of the program's normal behavior.

OpenOCD

Install **openocd** from **aur/openocd-git** for latest update to use **STM8** devices

```
# Arch linux
$ yaourt -S aur/openocd-git

# Ubuntu and mint linux
$ sudo apt install openocd
```

> *SourceForge OpenOCD.*

Download the latest stm8 binutils-gsb sources from Official site or direct from SourceForge. https://sourceforge.net/projects/stm8-binutils-gdb/files/stm8-binutils-gdb-sources-2018-03-04.tar.gz/download

Building the binaries is basically the process of downloading the sources and applying the patches. There are helper scripts to assist with the process.

Also note you need some libraries for TUI mode to work. Among those are ncursesw.

To download, patch and configure:

```
$ wget https://sourceforge.net/projects/stm8-binutils-gdb/files/stm8-binutils-gdb

$ tar -xf stm8-binutils-gdb-sources-2018-03-04.tar.gz
$ cd stm8-binutils-gdb-sources
$ ./patch_binutils.sh
$ ./configure_binutils.sh
```

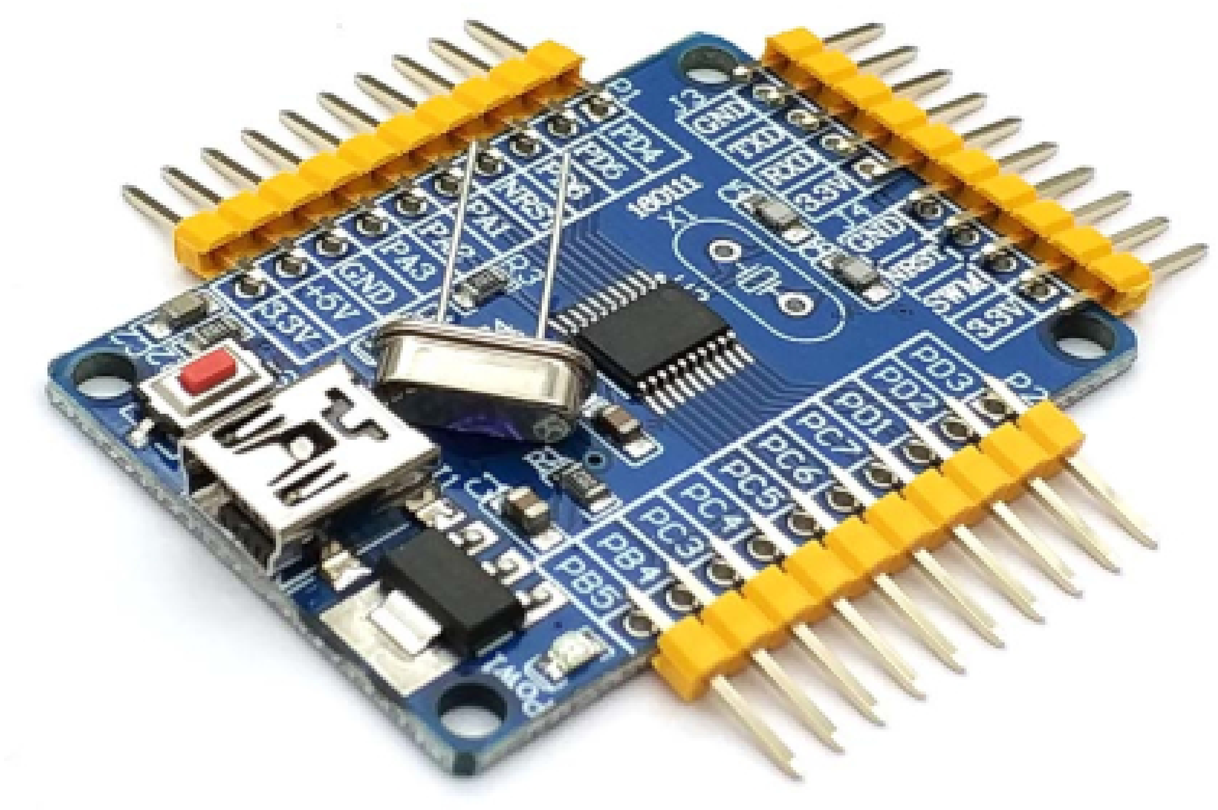
Next step is the regular building and install:

```
$ cd binutils-2.30
$ make
$ sudo make install
```

> *SourceForge stm8-binutils-gdb.*

ST's STM8 Discovery

The STM8S-DISCOVERY helps you to discover the STM8S features and to develop and share your own application. In my case i use STM8S003F3P6 STM8S Minimum System Development Board Module. It's about $1-$5 from AliExpress



Get Started

Your first code </>

Usually the first step toward learning development on a micro-controller is simply blinking a LED, as an analog to “Hello, world!” example used on PC programming languages. This time we will have a look into how to start programming and development on STMicroelectronics STM8 series of micro-controllers.

At this point you should have a working dev environment and can start experimenting with the board.

STM8S Reference Manual

stm8_blinky.c

```c
#include "stm8l.h"

#define Led_Init GPIO_Init(GPIOD, GPIO_PIN_1, GPIO_MODE_OUT_PP_LOW_FAST)
#define Led_ON   GPIO_WriteHigh    (GPIOD,GPIO_PIN_1)
#define Led_OFF  GPIO_WriteLow     (GPIOD,GPIO_PIN_1)
#define Led_TOG  GPIO_WriteReverse (GPIOD,GPIO_PIN_1)

void main(void)
{

   // Init LED Port, Pin
   Led_Init;

   // Set LED ON
   Led_ON;

   // Loop
   while(1){
      // Toggle LED ON/OFF
      Led_TOG;

      // White moment
      for(uint16_t d = 0; d<19000; d++){
         for(uint8_t c = 0; c<5; c++);
      }
   }
}
```

Build code

```
$ sdcc -lstm8 -mstm8 --opt-code-size --std-sdcc99 --nogcse --all-callee-saves --d
```

Wiring it up

Out of the factory, each board is flashed with a blinking demo, so you should see it start blinking as soon as you connect the USB. We won't be using the USB though, so unplugit again and prepare your ST-Link and the connection cable that came with it.

You'll also need to solder some headers to your STM8 board, at the very least the programming header (opposite of the USB). Both the dongle pins and the programming header are clearly labeled, so you shouldn't have any issues.

> *If you use clone st-link programmer, don't connect the 3V3 line from the dongle to the board while powering the board from USB. Technically nothing bad should happen, but you're connecting two LDOs in parallel and that's just a bad idea.*
>
> *Simply leave the 3V3 pin of the programming header unconnected in this case.*

First-time wipe

The board should start blinking immediately. The first step though will be to wipe the chip, since the factory-loaded firmware is read-protected and you can't do anything while it's locked down.

To unlock the chip, use the `-u` flasher option (for more info, run `stm8flash -h`):

```
$ stm8flash -c stlinkv2 -p stm8s003f3 -u
Determine OPT area
Unlocked device. Option bytes reset to default state.
Bytes written: 11
```

The board will stop flashing, you just bricked it. Oh no! But we'll fix that promptly.

Uploading firmware

You can now upload your own firmware using `make flash` , or if you downloaded the HEX file manually:

```
$ stm8flash -c stlinkv2 -p stm8s003f3 -s flash -w stm8_blinky.ihx
Determine FLASH area
Writing Intel hex file 655 bytes at 0x8000... OK
Bytes written: 655
```

Advenced

Debug

Compiling with sdcc and debug info:

```
$ sdcc -mstm8 led.c --out-fmt-elf --all-callee-saves --debug --verbose --stack-au
```

launch openoc

```
$ openocd -f /usr/share/openocd/scripts/interface/stlink.cfg -f /usr/share/openoc
```

or if you prefer the generic stm8s configuration (for medium size flash stm8s) replace **stm8s105.cfg** by **stm8s.cfg**

Currently config files for stm8s003, stm8s105 and stm8l152 are available.

Then start gdb:

```
$ stm8-gdb test.elf --tui
start
```

or if you prefer to load manually:

```
$ stm8-gdb test.elf --tui
target extended-remote localhost:3333
load
break main
continue
```

STM8 Microcontroller Linux St Link Opengdb

Enjoy this post? Give **Hamza BENDALI BRAHAM** a like if it's helpful.

1 SHARE

Hamza BENDALI BRAHAM

Enjoy this post?

1

1 Reply

Leave a reply

Bittu Bittu 2 years ago

sir why I am getting syntax error on compilation
sdcc -lstm8 -mstm8 --opt-code-size --std-sdcc99 --nogcse --all-callee-saves --debug --verbose --stack-auto --fverbose-asm --float-reent --no-peep -I./ -I./STM8S_StdPeriph_Driver/inc -D STM8S003 ./blinky.c

sdcc: Calling preprocessor...

Show more

Reply

Ufuk Şafak

How to run a Python script in Linux with SYSTEMD

Running a python script is easy, right?

```
python app.py
```

That's it! We are done here. Have a nice day 😃

Wait a second!
When you want that script to run automatically on startup, things get changed. You have some other options to accomplish that like crontab, rc.local etc. But for this blog, I am focusing on SYSTEMD.
DISCLAIMER: I am not experienced linux developer or user. This post is actually note to myself, because I forget very quickly 😃

What is SYSTEMD

READ MORE

Enjoy this post?

1