Using STM32 discovery kits with open source tools

STLINK development team

Contents

1 Overview **3**

2 Installing a GNU toolchain **4**

3 Installing STLINK **5**

4 Using the GDB server **6**

5 Building and flashing a program **8**

6 Notes **9**

7 References **10**

1 Overview

This guide details the use of STMicroelectronics STM32 discovery kits in an open source environment.

2 Installing a GNU toolchain

Any toolchain supporting the cortex m3 should do. You can find the necessary to install such a toolchain here:

```
https://github.com/esden/summon-arm-toolchain
```

Details for the installation are provided in the topmost README file. This documentation assumes the toolchains is installed in a $TOOLCHAIN_PATH.

3 Installing STLINK

STLINK is open source software to program and debug ST's STM32 Discovery kits. Those kits have an onboard chip that translates USB commands sent by the host PC into JTAG/SWD commands. This chip is called STLINK, (yes, isn't that confusing? suggest a better name!) and comes in 2 versions (STLINK v1 and v2). From a software point of view, those versions differ only in the transport layer used to communicate (v1 uses SCSI passthru commands, while v2 uses raw USB). From a user point of view, they are identical.

Before continuing, the following dependencies must be met:

- libusb-1.0
- pkg-config
- autotools

STLINK should run on any system meeting the above constraints.

The STLINK software source code is retrieved using:

```
$> git clone https://github.com/texane/stlink stlink.git
```

Everything can be built from the top directory:

```
$> cd stlink.git
$> ./autogen.sh
$> ./configure
$> make
```

It includes:

- a communication library (stlink.git/libstlink.a),
- a GDB server (stlink.git/st-util),
- a flash manipulation tool (stlink.git/st-flash).

4 Using the GDB server

This assumes you have got the libopencm3 project downloaded in [ocm3]. The libopencm3 project has some good, reliable examples for each of the Discovery boards.

Even if you don't plan on using libopencm3, the examples they provide will help you verify that:

- Your installed toolchain is capable of compiling for cortex M3/M4 targets
- stlink is functional
- Your arm-none-eabi-gdb is functional
- Your board is functional

A GDB server must be started to interact with the STM32. Depending on the discovery kit you are using, you must run one of the 2 commands:

```
# STM32VL discovery kit (onboard ST-link)
$> ./st-util --stlinkv1

# STM32L or STM32F4 discovery kit (onboard ST-link/V2)
$> ./st-util

# Full help for other options (listen port, version)
$> ./st-util --help
```

Then, GDB can be used to interact with the kit:

```
$> $TOOLCHAIN_PATH/bin/arm-none-eabi-gdb example_file.elf
```

From GDB, connect to the server using:

```
(gdb) target extended localhost:4242
```

GDB has memory maps for as many chips as it knows about, and will load your project into either flash or SRAM based on how the project was linked. Linking projects to boot from SRAM is beyond the scope of this document.

Because of these built in memory maps, after specifying the .elf at the command line, now we can simply "load" the target:

```
(gdb) load
```

st-util will load all sections into their appropriate addresses, and ”correctly” set the PC register. So, to run your freshly loaded program, simply ”continue”

```
(gdb) continue
```

Your program should now be running, and, if you used one of the blinking examples from libopencm3, the LEDs on the board should be blinking for you.

5 Building and flashing a program

If you want to simply flash binary files to arbitrary sections of memory, or read arbitary addresses of memory out to a binary file, use the st-flash tool, as shown below:

```
# stlinkv1 command to read 4096 from flash into out.bin
$> ./st-flash read v1 out.bin 0x8000000 4096

# stlinkv2 command
$> ./st-flash read out.bin 0x8000000 4096

# stlinkv1 command to write the file in.bin into flash
$> ./st-flash write v1 in.bin 0x8000000

# stlinkv2 command
$> ./st-flash write in.bin 0x8000000
```

Of course, you can use this instead of the gdb server, if you prefer. Just remember to use the ".bin" image, rather than the .elf file.

```
# write blink.bin into FLASH
$> [sudo] ./st-flash write fancy_blink.bin 0x08000000
```

Upon reset, the board LEDs should be blinking.

6 Notes

6.1 Disassembling THUMB code in GDB

By default, the disassemble command in GDB operates in ARM mode. The programs running on CORTEX-M3 are compiled in THUMB mode. To correctly disassemble them under GDB, uses an odd address. For instance, if you want to disassemble the code at 0x20000000, use:

```
(gdb) disassemble 0x20000001
```

7 References

- http://www.st.com/internet/mcu/product/248823.jsp
  documentation related to the STM32L mcu
- http://www.st.com/internet/evalboard/product/250990.jsp
  documentation related to the STM32L discovery kit
- http://www.libopencm3.org
  libopencm3, a project providing a firmware library, with solid examples for Cortex M3, M4 and M0 processors from any vendor.