# Using STM32 discovery kits with open source tools

STLINK development team

# Contents

# 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

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
$> make
```

It includes:

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

# 4 Building and running a program

A simple LED blinking example is provided in the example directory. It is built using:

```
cd stlink.git/example/blink ;
PATH=$TOOLCHAIN_PATH/bin:$PATH make
```

This builds three files, one for each of the Discovery boards currently available, linked to run from SRAM. (So no risk of overwriting anything you didn't mean to) These blink examples can safely be used to 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
```

From GDB, connect to the server using:

```
$> target extended localhost:4242
```

By default, the program was linked such that the base address is 0x20000000. From the architecture memory map, GDB knows this address belongs to SRAM. To load the program in SRAM, simply use:

```
$> # Choose one as appropriate for your Discovery kit
$> load blink_32L.elf | load blink_32VL.elf | load blink_F4.elf
```

GDB automatically set the PC register to the correct value, 0x20000000 in this case. Then, you can run the program using:

```
$> continue
```

All the LEDs on the board should now be blinking in time (those leds are near the user and reset buttons).

## 5 Reading and writing to flash

Flash memory reading and writing is done by a separate tool. A binary running in flash is assumed to be linked against address 0x8000000. The flash tool is then used as shown below:

```
# change to the flash tool directory
$> cd stlink.git/flash ;

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

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

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

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

# 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:

```
$> disassemble 0x20000001
```

## 6.2 libstm32l_discovery

The repository includes the STM32L discovery library source code from ST original firmware packages, available here:

```
http://www.st.com/internet/evalboard/product/250990.jsp#FIRMWARE
```

It is built using:

```
$> cd stlink.git/example/libstm32l_discovery/build
$> make
```

An example using the library can be built using:

```
$> cd stlink.git/example/lcd
$> make
```

# 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