STM32F3 Discovery - developing and debugging on Linux with Eclipse

Posted on March 5, 2013 by admin

Features

- Using free GCC toolchain.
- Makefile based project, either standalone or with Eclipse.
- Flashing program.
- Supports development with the STM32F3 Discovery Kit Firmware.
- Debugging with a standalone gdb or from Eclipse.

Some quick facts about the Discovery board

- STM32F3 Discovery board home page
- User Manual
- STM32F303VCT6 microcontroller (32-bit ARM Cortex-M4F core, 256 KB Flash, 48 KB RAM)
- 3D accelerometer and 3D magnetometer LSM303DLHC
- 3-axis digital MEMS gyroscope L3GD20
- 10 LEDs
- Two push buttons (user and reset)
- USB USER with Mini-B connector

1. Install GCC toolchain

Create a directory for the toolchain and other tools so we don't mess up our system.

```
mkdir ~/stm32f3
```

You can use any directory you like but then you would need to configure the path in the Makefile.common and the stlink_server.sh but on that later.

Install some dependencies:

```
apt-get install flex bison libgmp3-dev libmpfr-dev \
libncurses5-devlibmpc-dev autoconf texinfo \
build-essential libftdi-dev libusb-1.0-0-dev git
```

... and now the main toolchain. The following will download/update the latest version of the summo arm toolchain, build it and install in your home directory in **~/stm32f3/**

```
git clone https://github.com/esden/summon-arm-toolchain
cd summon-arm-toolchain
./summon-arm-toolchain PREFIX=~/stm32f3/
```

It appears the summon makefile has some synchronization issues when running on multicore machine (the build script runs make -j <num_of_your_cores>) and the building process stops on a few errors due to non existing object files. Easy fix is to run the ./summo-arm-toolchain command again after the error.

The toolchain will get installed in the directory **~/stm32f3** For example to run gcc you can do this:

```
~/stm32f3/bin/arm-none-eabi-gcc
```

2. Install stlink

This is necessary for flashing your code on the board and for remote debugging with gdb

```
git clone https://github.com/texane/stlink stlink.git
cd stlink.git
./configure --prefix=$HOME/stm32f3/
make
make install

sudo cp *.rules /etc/udev/rules.d
sudo restart udev
```

You can test that the stlink works by connecting the board to your computer over USB and run

```
~/stm32f3/bin/st-util
```

The response should be:

```
2013-03-05T11:43:48 INFO src/stlink-usb.c: -- exit_dfu_mode
2013-03-05T11:43:48 INFO src/stlink-common.c: Loading device
parameters....
2013-03-05T11:43:48 INFO src/stlink-common.c: Device connected
is: F3 device, id 0x10036422
2013-03-05T11:43:48 INFO src/stlink-common.c: SRAM size:
0xa000 bytes (40 KiB), Flash: 0x40000 bytes (256 KiB) in
pages of 2048 bytes
Chip ID is 00000422, Core ID is 2ba01477.
KARL - should read back as 0x03, not 60 02 00 00
init watchpoints
Listening at *:4242...
```

3. Install the STM32F3 Discovery Kit Firmware

Download the STM32F3 Discovery Kit Firmware and uncompress it to the **~/stm32f3** directory. You should get this:

```
~/stm32f3/STM32F3-Discovery_FW_V1.1.0
```

4. Setup the project template

Download https://electroncastle.googlecode.com/files/stm32f3_template.zip and uncompress it to any directory you like, for example:

```
~/stm_projects/
```

There are three directories:

blinky - a demo project that blinks with the 10 LEDs on the board
template - a blank project as a base for your new projects
common - all necessary stuff to build and flash the stm32f3 projects with the stm32f3 firmware kit.

You can jump straight to the **blinky** directory build it and upload on your board.

```
cd ~/stm_projects/blinky
make
make flash
```

To create a new blank project copy the **template** directory, for example

```
cd ~/stm_projects/
cp -a template test
cd test
make
make flash
```

A few quick notes on using drivers from the STM32F3 firmware kit

1. If you want to use, for example GPIO driver, go to the project makefile and add the corresponding driver file and its dependencies. In the case of the GPIO it would be:

```
OBJS+= \
 stm32f30x_gpio.o \
 stm32f3_discovery.o \
 stm32f30x_rcc.o \
 stm32f30x_exti.o \
 stm32f30x_syscfg.o \
 stm32f30x_misc.o
```

2. To handle interrupts you have to add you handler to the stm32f30x_it.c For example to get an interrupt for sys ticks you need to call you own function from the

```
void SysTick_Handler(void)
{
     TimingDelay_Decrement();
}
```

The **SysTick_Handler()** interrupt is used in the blinky project so check that if you like.

5. Coding with the Eclipse

You will need Eclipse with CDT. If you already have it fine if not get it from here http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/junosr2

1. Open any workspace you like, or create a new one in the **~/stm_projects**
2. Go to **File -> New -> Project -> C/C++ -> Makefile Project with Existing Code**
3. If you Eclipse doesn't recognize that the project needs C/C++ profile go to: **Window -> Open Perspective -> C/C++**
4. Now you can build the project with **Ctrl-B**

6. Debugging with stlink and Eclipse

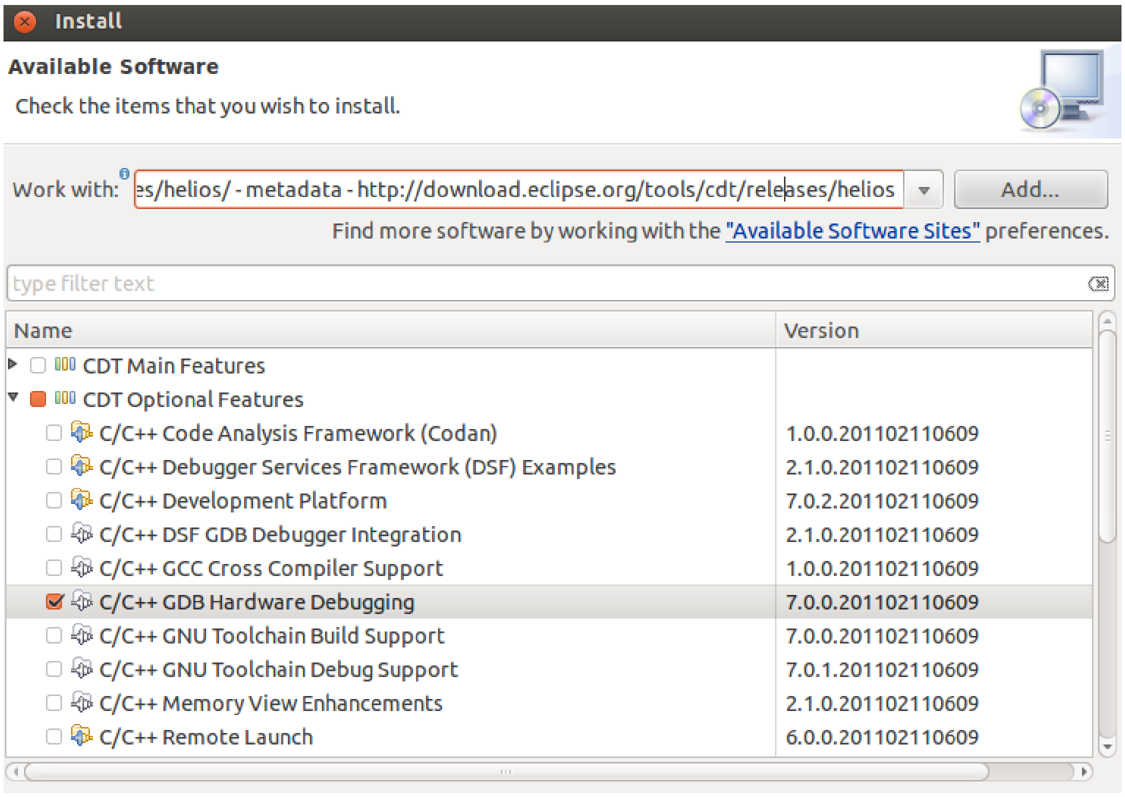
Install the Eclipse according to the previous step. On top of that you will need to install GDB Hardware Debugger

Help -> Install New Software

Paste the following link into the "Work With" edit box:

```
http://download.eclipse.org/tools/cdt/releases/helios
```

After a few seconds you should see a list of tools for this repository. Open the **CDT Optional Features** and select the **C/C++ GDB Hardware Debugging**



Work with: es/helios/ - metadata - http://download.eclipse.org/tools/cdt/releases/helios

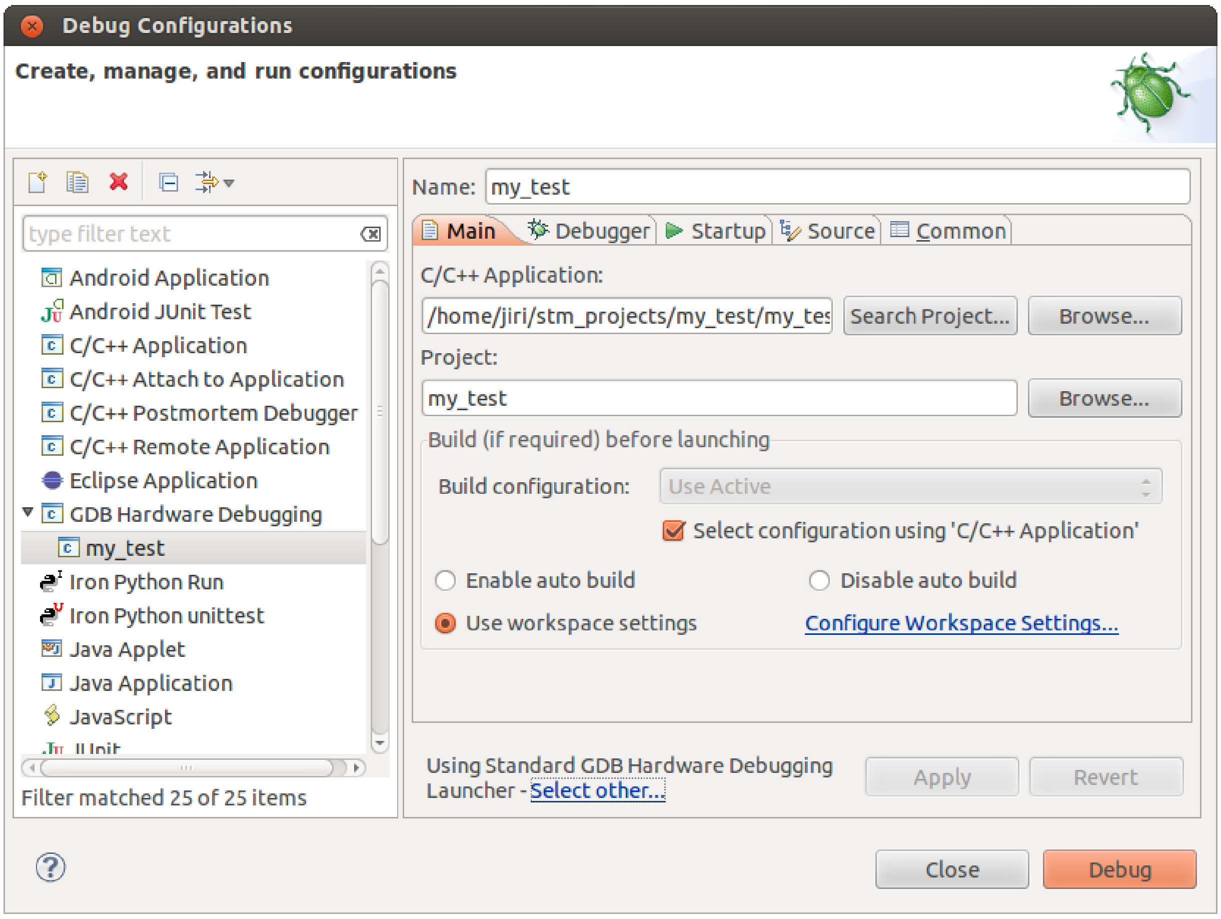
Find more software by working with the "Available Software Sites" preferences.

Name	Version
☐ CDT Main Features	
■ CDT Optional Features	
☐ C/C++ Code Analysis Framework (Codan)	1.0.0.201102110609
☐ C/C++ Debugger Services Framework (DSF) Examples	2.1.0.201102110609
☐ C/C++ Development Platform	7.0.2.201102110609
☐ C/C++ DSF GDB Debugger Integration	2.1.0.201102110609
☐ C/C++ GCC Cross Compiler Support	1.0.0.201102110609
✓ C/C++ GDB Hardware Debugging	7.0.0.201102110609
☐ C/C++ GNU Toolchain Build Support	7.0.0.201102110609
☐ C/C++ GNU Toolchain Debug Support	7.0.1.201102110609
☐ C/C++ Memory View Enhancements	2.1.0.201102110609
☐ C/C++ Remote Launch	6.0.0.201102110609

Now import a stm project to the Eclipse as we did in the previous step and go to the:

Run -> Debug Configurations..

In the left list box select **GDB Hardware Debugging** and click the 'new configuration' icon in the left top corner - the paper sheet with a small plus in the corner. You should get something like this, of course with the project of your name.


Debug Configurations

Create, manage, and run configurations

type filter text

- Android Application
- Android JUnit Test
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- Eclipse Application
- GDB Hardware Debugging
  - my_test
- Iron Python Run
- Iron Python unittest
- Java Applet
- Java Application
- JavaScript
- JUnit

Filter matched 25 of 25 items

Name: my_test

Main | Debugger | Startup | Source | Common

C/C++ Application: /home/jiri/stm_projects/my_test/my_tes

Project: my_test

Build (if required) before launching

Build configuration: Use Active

[✓] Select configuration using 'C/C++ Application'

Enable auto build | Disable auto build

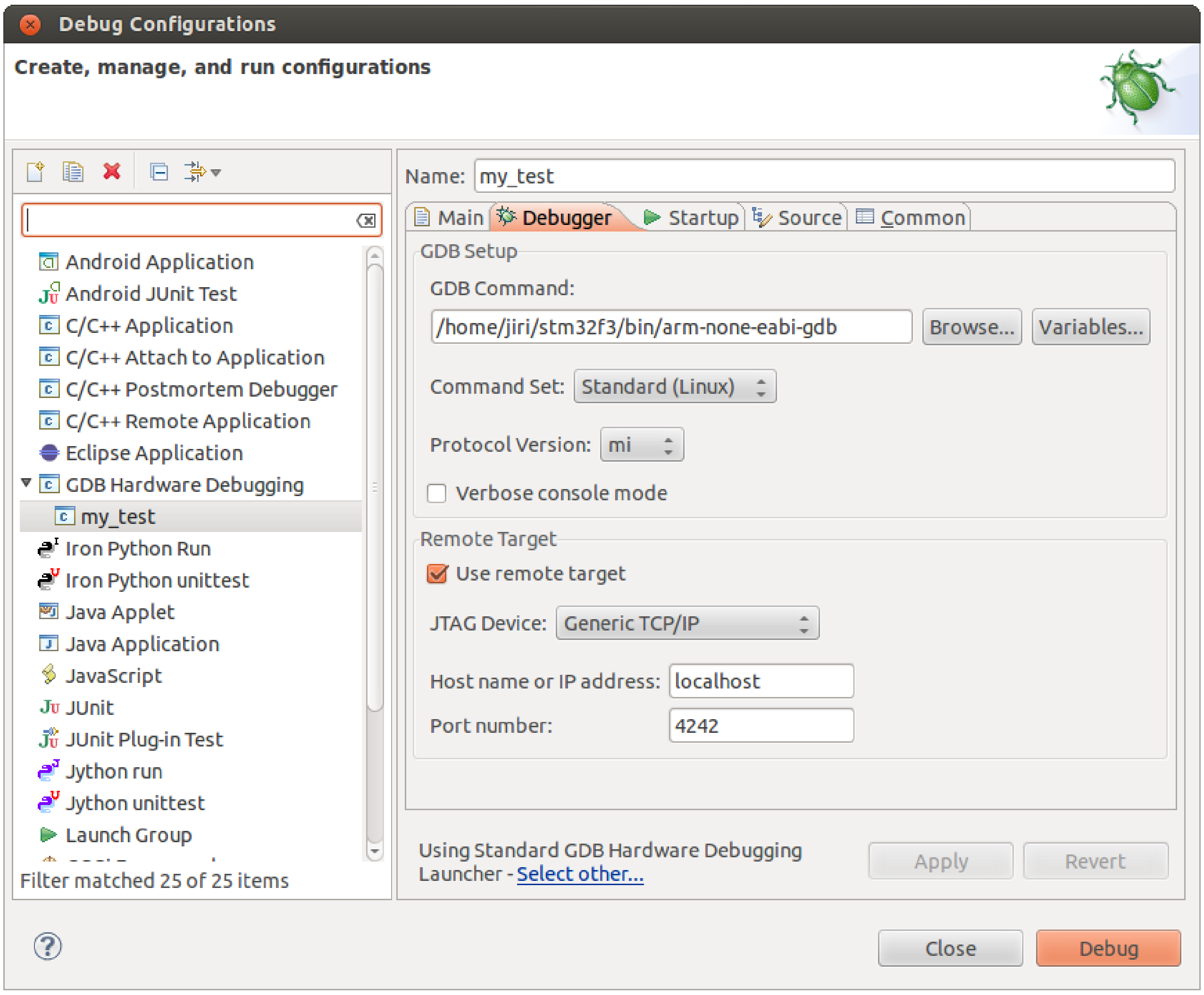
[✓ Use workspace settings] | Configure Workspace Settings...

Using Standard GDB Hardware Debugging Launcher - Select other...

Make sure the you are using the **Standard GDB Hardware Debugging** tool. You can see and change it at the bottom of the dialog next to the Apply button.

Next step is to configure the debugger. Go to the **Debugger** tab and in the **GDB Command** browse to the ~/stm32f3/bin/arm-none-eabi-gdb The path must be absolute.

In the **Remote Target** set the port to: 4242


Debug Configurations

Create, manage, and run configurations

Name: my_test

GDB Setup

GDB Command: /home/jiri/stm32f3/bin/arm-none-eabi-gdb

Command Set: Standard (Linux)

Protocol Version: mi

Verbose console mode

Remote Target

Use remote target

JTAG Device: Generic TCP/IP

Host name or IP address: localhost

Port number: 4242

Using Standard GDB Hardware Debugging Launcher - Select other...

Filter matched 25 of 25 items

Click **Apply** and **Close**

Now run the stlink server.

```
~/stm_projects/common/stlink_server.sh
```

This will connect to your stm32f3 board and starts gdb server that the Eclipse will connect to (in a fact gdb from Eclipse)

Build the project in the Eclipse (Ctrl+B) Next click on the Debug icon in the toolbar and from the drop-down menu select the debug profile of your project. The project code *.elf will get uploaded on the stm board and the debugger starts at the reset entry point.

In you don't like this behaviour go to the **Debug Configurations -> Startup** and select the **Resume** option. In this case the debugger won't stop and you have to either place a breakpoint in your code or suspend it from the Eclipse GUI.

Note while the **stlink_server.sh** is running you cannot use the **make flash** command since the stlink is busy. This is not an issue since while debugging with the Eclipse, your code gets uploaded on the board via the stlink directly so there is not need to run the make flash.

And that's it. Happy discovering....

This entry was posted in ARM, Electronics, Embedded. Bookmark the permalink.