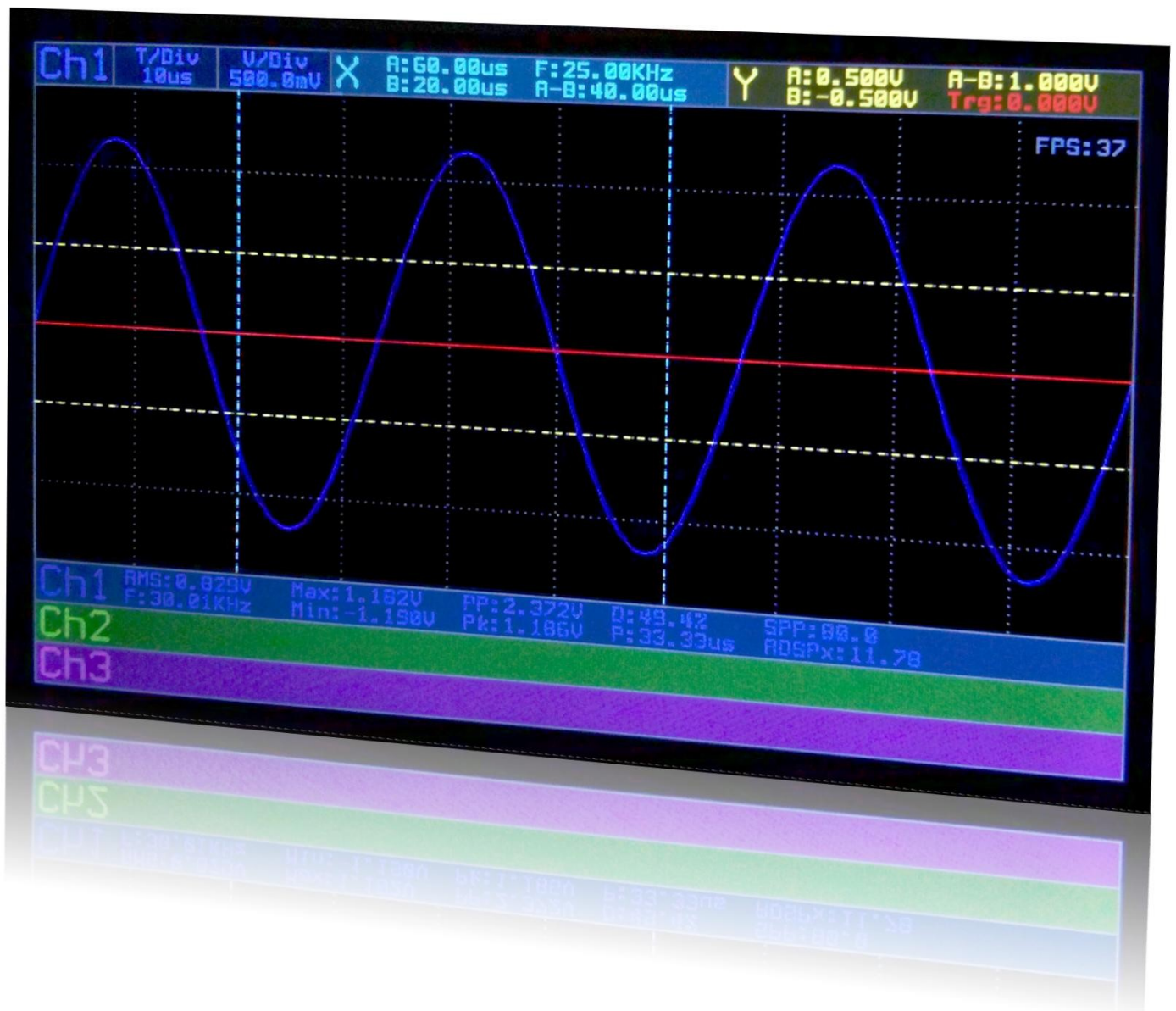


DSO PROJECT



By Manos Tsachalidis

Tutorial for setting up STM32F4 discovery board on linux with Eclipse, GNU ARM toolchain, OpenOCD and RTOS ChibiOS + uGFX and eventually experiment with my DSO project

This is a tutorial for setting up the STM32F4 discovery board on linux. The distro i am using at this tuto is Kubuntu 13.10 64bit. First thing to do is download the apps. The following links are related to architecture specific releases and may not work fo your distro. You need to be aware of your distro's architecture (32 or 64bit). You may of course experiment with different versions.

Eclipse Indigo (64bit):

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR2/eclipse-cpp-indigo-SR2-incubation-win32-x86_64.zip

Eclipse Kepler (64bit) - Recommended:

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/kepler/SR2/eclipse-cpp-kepler-SR2-linux-gtk-x86_64.tar.gz

GNU ARM toolchain (32 & 64bit):

https://launchpad.net/gcc-arm-embedded/4.8/4.8-2013-q4-major/+download/gcc-arm-none-eabi-4_8-2013q4-20131204-linux.tar.bz2

GNU ARM toolchain Readme file:

<https://launchpadlibrarian.net/160487135/readme.txt>

OpenOCD 0.6.1 (Debugger)

<http://sourceforge.net/projects/openocd/files/openocd/0.6.1/openocd-0.6.1.zip/download>

ChibiOS 2.6.0

http://sourceforge.net/projects/chibios/files/ChibiOS_RT%20stable/Version%202.6.0/ChibiOS_2.6.0.zip/download

uGFX 1.80

<http://ugfx.org/images/resources/downloads/releases/ugfx-release-1.8.zip>

In this tutorial i will mkdir a directory named .devel in my ~/ .After downloading eclipse, unzip it anywhere you like and move it in ~/.devel . Same applies for the GNU arm toolchain. After unzipping the toolchain you can move the dir in ~/.devel. Do the same for openocd. After unzipping, move the dir to ~/.devel . ChibiOS and uGFX should be downloaded in a common path if a network drive is used. For users that will work from exclusively one computer, they can use the path ~/.devel .

After you download ChibiOS 2.6.0, unzip it in ~/.devel and rename the folder to ChibiOS_2.6.0. After you download uGFX, unzip it and rename the folder to ugfx_1.8. Next thing to do is to move the ugfx_1.8 directory inside ChibiOS_2.6.0/ext .

OpenOCD

This may be tricky for some people but i hope you will get over it. Besides, linux users should know the basics of building an application.

Install the libftdi (or libftdi1) and libusb (plus dependencies) with the following command:

`sudo apt-get install libftdi1 libusb-1.0`

The openocd version i built is 0.6.1 and worked fine but 0.7.0 should work as well.

To build the executables go to: **`~/devel/openocd-0.6.1`**

Run this: **`./configure --enable-ft2232_libftdi1 --enable-stlink`** to enable support for the stm32f4 stlink.

Next run: **`make`** to compile the source files.

Last thing to do is install the executables by doing: **`sudo make install`**

You are done with OpenOCD. This has worked for me fine in a fresh Kubuntu 13.10 installation.

GNU ARM toolchain

You have to update your environment variable \$PATH with the path of the executables. For Kubuntu, you need to make a new file in ~/.kde/env/ called whatever you like (i named mine envar.sh) and enter the following text:

```
export PATH=$HOME/.devel/gcc-arm-none-eabi-4_8-2013q4/bin:$PATH
```

In Ubuntu, you need to edit your ~/.profile and add the following text:

```
PATH=$HOME/.devel/gcc-arm-none-eabi-4_8-2013q4/bin:$PATH
```

Both files above are used to update your environment variable \$PATH while entering your personal session in Ubuntu or Kubuntu. To confirm that your \$PATH contains the path to the arm toolchain, you must log out and log back in or try rebooting. Then open a terminal window and see if the environment variable contains the path to the arm toolchain executables with the following command: **echo \$PATH**

Project directory

This will need your full attention as everything needs to be in perfect order for everything to work. Make a new directory called “projects” into ~/.devel/. Navigate to ~/.devel/ChibiOS_2.6.0/demos/, copy dir “ARMCM4-STM32F407-DISCOVERY” and paste it to ~/.devel/projects. Navigate to the path ./devel/projects/ARMCM4-STM32F407-DISCOVERY/. You can delete the dirs “iar” & “keil” as you are using eclipse. Create a file named “OpenOCD.launch” and paste the xml code that follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<launchConfiguration type="org.eclipse.ui.externaltools.ProgramLaunchConfigurationType">
<listAttribute key="org.eclipse.debug.ui.favoriteGroups">
<listEntry value="org.eclipse.ui.externaltools.launchGroup"/>
</listAttribute>
<stringAttribute key="org.eclipse.ui.externaltools.ATTR_LAUNCH_CONFIGURATION_BUILD_SCOPE" value="{project}"/>
<stringAttribute key="org.eclipse.ui.externaltools.ATTR_LOCATION" value="/usr/local/bin/openocd"/>
<stringAttribute key="org.eclipse.ui.externaltools.ATTR_TOOL_ARGUMENTS" value="-f openocd/board/stm32f4discovery.cfg"/>
<stringAttribute key="org.eclipse.ui.externaltools.ATTR_WORKING_DIRECTORY" value="{workspace_loc:/NAME_OF_PROJECT}"/>
</launchConfiguration>
```

NAME_OF_PROJECT: Replace with the dir name of your project which in our case is:
ARMCM4-STM32F407-DISCOVERY

Download the zip file dso.naftilos76.com/dl/openocd.zip that contains the openocd scripts that will make it work with your stm32f4 discovery board and unzip it in your project:

~/.devel/projects/ARMCM4-STM32F407-DISCOVERY/. Let's consider that referring to your stm32f4 project will from now on mean dir: ./devel/projects/ARMCM4-STM32F407-DISCOVERY/.

Download the zip file dso.naftilos76.com/dl/Makefile.zip and copy / overwrite the file Makefile in your project. This file will be used by eclipse to compile, Link, include the proper header files etc.

IMPORTANT: Edit the file Makefile after you copy. Go to line 67 and substitute with your own home path. Save and close the file.

Eclipse first time run

Navigate to ~/.devel/eclipse and execute (click or doubleclick on) the executable file eclipse. A splash window shows up and later a dialog asking you to declare a workspace path. The path ~/.devel/projects/ will do just fine. Do not click the option “Use this as the default...” just in case you need to enter an other path later. After eclipse opens up, navigate to menu Project and uncheck menu item 'Build automatically'. Go to menu Help and click on menu item 'Install new software'. Go to the text field / combo box at the top (where it says: 'work with') with your mouse and select the '--All Available Sites--'. You will notice that a list of various s/w sources will appear in a list below. Go to the text field just above the list and type the string “gdb”. Once you do that the list below will change and will, among others, contain “C/C++ GDB Hardware Debugging.....”. Check the check box on the left and click on the button 'Next' twice, then accept the terms of the licence and click on 'Finish'. This is the debugger that you will be using for C/C++ projects. Strangely, this will take quite a few minutes.

STM32F4 discovery board programmer

The on-board programmer st-link v2 (or a standalone external one) on the discovery board can work if you enable it in the udev rules. This means that you have to create a new file called 33-openocd.rules (any other number instead of 33 will do) in /etc/udev/rules.d/. Use an editor with sudo to enter the following text in it:

```
#FT232
ATTRS{idProduct}=="6014", ATTRS{idVendor}=="0403", MODE="666", GROUP="plugdev"

#FT2232
ATTRS{idProduct}=="6010", ATTRS{idVendor}=="0403", MODE="666", GROUP="plugdev"

#FT230X
ATTRS{idProduct}=="6015", ATTRS{idVendor}=="0403", MODE="666", GROUP="plugdev"

#STLINK V1
ATTRS{idProduct}=="3744", ATTRS{idVendor}=="0483", MODE="666", GROUP="plugdev"

#STLINK V2
ATTRS{idProduct}=="3748", ATTRS{idVendor}=="0483", MODE="666", GROUP="plugdev"
```

Copy the text in the empty file and save. Now you need to reload the udev rules with **sudo /etc/init.d/udev restart**

in order to allow the discovery board to operate in linux. Actually the last line of the above text is the one that you need if you have a st-link v2 programmer.

Importing the STM32F4 project in Eclipse

Start Eclipse and navigate to menu File and select menu item 'Import'. A window will show up containing a list of import sources. Double click on the first called 'General' to expand and select 'Existing projects into Workspace'. Click on 'Next', click on 'Browse' (select root directory) to point to the project. A new window shows up. Use it to navigate to your project. Reach the point where you have double clicked on your project directory. Click ok to accept. You will now see that the list in the 'Import' window contains the project you wish to import. Without clicking on anywhere else, click on 'Finish'. The project will now appear in Eclipse.

Building the project for the first time

After importing your project in Eclipse, double click on it so that it expands. You can now see the files in your project like main.c and other files. Doubleclick on main.c so that it opens. What you see now is a chaos of errors and warnings indicated by red waving lines, small rectangles etc. Do not panic! This will all disappear in a second. You have to build the project again and again for a couple of times for all the errors and warnings to disappear. You also need to go to the Project explorer on the left, right click on the project name "ARM....DISCOVERY" at the top and select at least once the menu item 'Index' and submenu item 'Rebuild'. Both building the project and rebuilding the index is required to get rid of the errors and warnings. However, some errors may persist despite the fact that the project compiles fine. If that is the case, you can try shutting down Eclipse and entering back in. That may help get rid of the last error indications in main.c or other source files. To confirm that you have successfully built your project, you can check the tab "Console" at the bottom where source files being compiled as well as other stuff are listed. At the end of the "Console" tab contents you may see something like (example):

22:49:17 Build Finished (took 530ms)

This means the built was successful despite any error messages in the tab "Problems" at the bottom. One more thing to do is to add the path of the GNU ARM toolchain header files. To do this, go to menu "Project" and select menu item "Properties". A window pops up. Go to the left top text field and write the text "symbols". You will notice that the list below the text field now contains only "Paths and Symbols". Click on it and go to the right side of the window and click on button "Add" and enter the path:

`/home/YOUR_HOME_DIR/.devel/gcc-arm-none-eabi-4_8-2013q4/arm-none-eabi/include`. Do this for all 3 languages (Assembly, C and C++).

Flashing and debugging

In order to confirm that debugging and flashing works, we will test everything with a small piece of code containing a loop which will toggle the 4 leds on board the stm32f4 discovery board.

Copy and paste the following code in your main.c

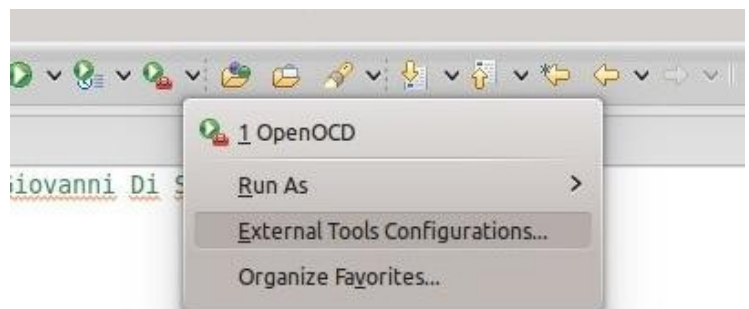
```
#include "ch.h"
#include "hal.h"

int main(void) {

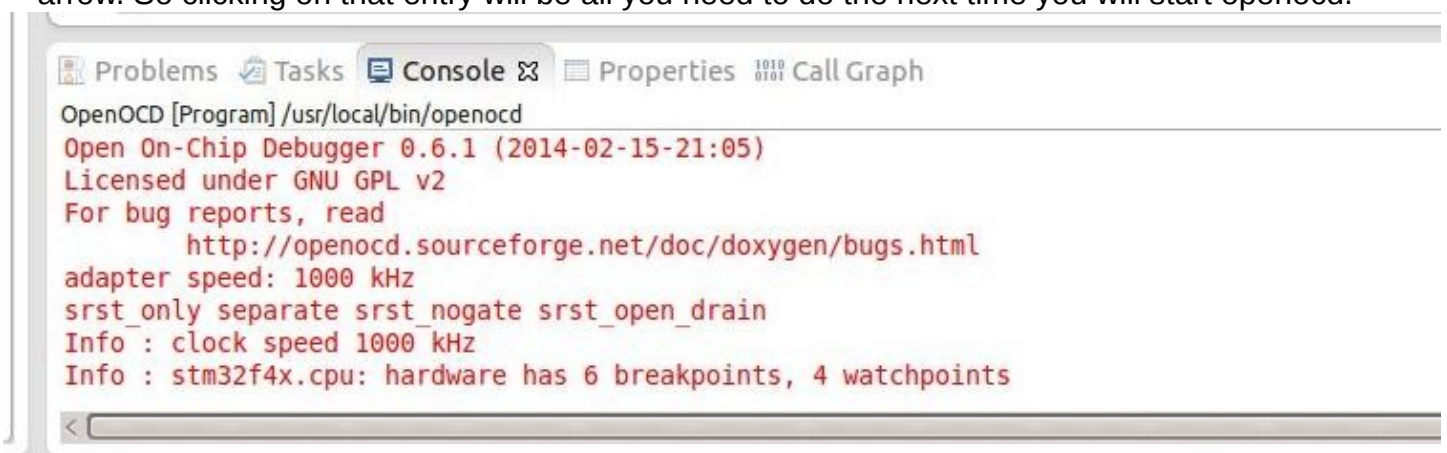
    halInit();
    chSysInit();
    // Set port D and pins 12-15 to push pull output.
    // Push-pull means that whenever we set a pin to 1, 3 Volts will appear at that pin.
    // If we set that pin to 0, then the pin will be connected to GND.
    palSetPadMode(GPIOD, 12, PAL_MODE_OUTPUT_PUSHPULL);
    palSetPadMode(GPIOD, 13, PAL_MODE_OUTPUT_PUSHPULL);
    palSetPadMode(GPIOD, 14, PAL_MODE_OUTPUT_PUSHPULL);
    palSetPadMode(GPIOD, 15, PAL_MODE_OUTPUT_PUSHPULL);


    while (1) {
        // Set the pins to an opposite value. If they are 1, make their output 0 and so on...
        palTogglePad(GPIOD, 12);
        palTogglePad(GPIOD, 13);
        palTogglePad(GPIOD, 14);
        palTogglePad(GPIOD, 15);
        // Create a delay of 500ms
        chThdSleepMilliseconds(500);
    }
    return (0);
}
```

It's now time to run OpenOCD for the first time. As the snapshot on the right indicates, go with your mouse to the debug button which is the round green / white button with a red icon. Click on the little arrow pointing downwards. A menu will appear. Select the menu item "External tools configurations".



A window will appear which contains all the setup information for the openocd. All these settings came from the file named "OpenOCD.launch" which is already in your project directory. All you have to do is double click / expand the list on the left where it says "Program". After it expands, OpenOCD will appear. Click on the OpenOCD entry and the related information will show up in the rest of the window's text fields and tabs. You do not need to configure anything here expect for pressing the button "Run" at the right bottom of the window. This will start OpenOCD. After doing so, what you should see is something like the snapshot below. Remember that after doing this, you will have a menu entry "OpenOCD" as indicated above everytime you click on that arrow. So clicking on that entry will be all you need to do the next time you will start openocd.



You will now configure the GDB debugger that you installed earlier in Eclipse. As previously done with the openocd button, go with your mouse to the button where a bug is illustrated.  Click on the little arrow pointing downwards. A menu will show up. Select the menu item “Debug configurations”. A window will pop up. Go to the left list of entries and double click on the GDB debugger. A sub-entry will be created named as your project. Follow the instructions below to configure the debugger.

Tab: Main

Text field “Name”: You can rename it to “Flash and Debug” or you can leave it as is.

Text field “C/C++ Application”: Enter the text “build/ch.elf” without the quote signs.

Tab: Debugger

Text field “GDB Command”: Enter the debugger executable of the GNU ARM toolchain which is “arm-none-eabi-gdb” without the quote signs.

*As i said i am on a Kubuntu 64bit. When i run this debugger i got an error message that a shared library “libncurses” was not found. This is due to the fact that the ARM tool chain needs 32 bit executables. The solution was to install the 32 bit version of that library as the 64 bit version was already installed but on a different path.

To install that, do this: [sudo apt-get install libncurses5:i386](#)

Text field “Port number”: This is the port where openocd actually listens. The default value is probably 10000. Change that to 3333 and this tab is done.

Tab: Startup

Uncheck the two check boxes “Reset and Delay (seconds)” and “Halt”. Enter the init commands “mon reset init” on the text window below the “Halt” checkbox.


Tab: Source

Nothing to do

Tab: Common

Check the checkbox “Debug”

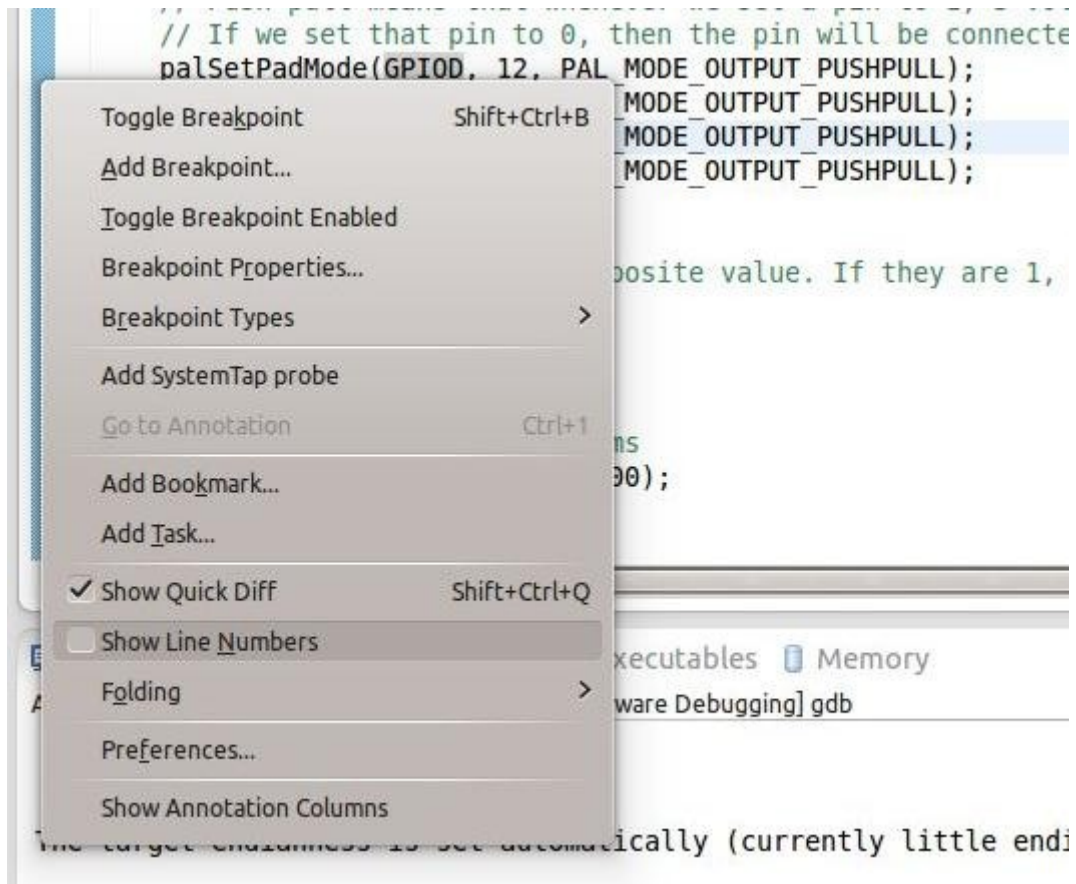
Click on the “Apply” button at the bottom right and you are done.

You can now click on the “Debug” button. This will load the compiled code “ch.elf” in your “build” directory in the microcontroller and start the debugging session. As in openocd window, the name that you used above “Flash and Debug” or any other will appear in the menu of the debugger button  so that you won't have to do all this again and again.

Hopefully you have already flashed your code into the STM32F4 and thinking about what debugging is really all about (if you have not done this before). Debugging is about discovering and finding solutions in error situations and generally see what is going on in our code, see the values of variables as they are updated and every other entity that carries one or more values. If your code is executing you will see the leds of the discovery board to keep blinking once per second. Below is the tab “Debug” of Eclipse tha contains openocd and GDB debugger. You can click on the row of the debugger and then click on the red stop button in case you want to stop the debugger and re-upload the code to the uC. Same applies for the openocd.



Open the main.c and enable the line numbers by right clicking on the left most blueish column and selecting from the menu the menu item “Show line numbers”



Double click on line number 35. You will see a blue round shape to appear on the left of the number. After an instant the arrow, as seen below, should appear as well. The execution stops and the Green led is the only one lit. This means that palTogglePad(GPIOD, 12); has executed but not the next line. In order for the execution to continue, you have to click on the “play” button which is enabled everytime execution is interrupted. After clicking on the “play” button, the execution will continue until the next time it reaches the breakpoint.

```
26 // If we set that pin to 0, then the pin will be conr
27 palSetPadMode(GPIOD, 12, PAL_MODE_OUTPUT_PUSHPULL);
28 palSetPadMode(GPIOD, 13, PAL_MODE_OUTPUT_PUSHPULL);
29 palSetPadMode(GPIOD, 14, PAL_MODE_OUTPUT_PUSHPULL);
30 palSetPadMode(GPIOD, 15, PAL_MODE_OUTPUT_PUSHPULL);
31
32 while (1) {
33     // Set the pins to an opposite value. If they are
34     palTogglePad(GPIOD, 12);
35     palTogglePad(GPIOD, 13);
36     palTogglePad(GPIOD, 14);
37     palTogglePad(GPIOD, 15);
38     // Create a delay of 500ms
39     chThdSleepMilliseconds(500);
```



Click on the green arrow
For the execution to continue

The DSO source code requires that you make a lot of enabling / disabling of define macros in various header files. I will save you all that trouble by offering the project dir including all source files of the DSO in a zip file. You can download it at: dso.naftilos76.com/dl/dso.zip . Unfortunately, the ChibiOS and uGFX dir tree should also be downloaded at dso.naftilos76.com/dl/lib.zip . This zip file contains ChibiOS and uGFX as explained in the beginning of this tutorial. That is, uGFX is already copied in ChibiOS, so you do not need to copy uGFX into ChibiOS. If you decide to use the DSO project, you have to delete your ChibiOS dir in ~/.devel and unzip there the lib.zip. After downloading the dso.zip, unzip it and import as a project it as explained earlier.

Using the DSO project makes sense if you have a SSD1963 TFT lcd display 800x480 pix with a touch panel compatible with ADS7843 controller like this one i bought from ebay:

http://www.ebay.com/itm/New-5-0-800-480-TFT-LCD-Module-Display-Touch-Panel-SSD1963-51-AVR-STM32-/180910383286?pt=LH_DefaultDomain_0&hash=item2a1f1960b6

However any SSD1963 lcd will do as long as you initialise it properly. The supplier usually offers code samples that can help you make it work.

A few references are given below for further reading:

<http://ugfx.org/>
<http://ugfx.org/forum/>
<http://www.chibios.org/dokuwiki/doku.php>
<http://chibios.sourceforge.net/html/index.html>

Tutorials with similar content:

- Tutorial for installing software on ubuntu using eclipse
<http://embeddedprogrammer.blogspot.com/2012/09/stm32f4discovery-development-with-gcc.html>
- Tutorial using Eclipse with ST standard peripheral library
<http://www.afflatustech.com/2011/11/eclipse-codesourcery-stm32f4-settings.html>
http://masu.6f.sk/index.php/Stm32f4_eclipse
- Another tutorial for flashing Discovery board
<http://vedder.se/2012/07/get-started-with-stm32f4-on-ubuntu-linux/>
- Linux Client that recognizes ST-Link/V2
<http://code.google.com/p/qstlink2/>
- Programming STM32 F2, F4 ARMs under Linux: A Tutorial from Scratch
<http://www.triplespark.net/elec/pdev/arm/stm32.html>
- Getting Started with the STM32F4 and GCC
http://jeremyherbert.net/get/stm32f4_getting_started
- Starting with the STM32F4 (discovery kit)
<http://www.pittnerovi.com/jiri/hobby/electronics/stm32f4/index.html>
- Setup STM32F4 Environment on Ubuntu With STLINK Debug Function
<http://shareee.netne.net/wordpress/?p=83>

I will set up a website to post this tuto and further update it as well as being able to accept comments and offer help where i can.

For the time being my email is naftilos76@gmail.com

Any questions are welcome.

Pin connections for FSMC mode

LCD Board	Discovery board	FSMC Interface	Pin Function
D0	PD14	FSMC_D0	Databit 0
D1	PD15	FSMC_D1	Databit 1
D2	PD0	FSMC_D2	Databit 2
D3	PD1	FSMC_D3	Databit 3
D4	PE7	FSMC_D4	Databit 4
D5	PE8	FSMC_D5	Databit 5
D6	PE9	FSMC_D6	Databit 6
D7	PE10	FSMC_D7	Databit 7
D8	PE11	FSMC_D8	Databit 8
D9	PE12	FSMC_D9	Databit 9
D10	PE13	FSMC_D10	Databit 10
D11	PE14	FSMC_D11	Databit 11
D12	PE15	FSMC_D12	Databit 12
D13	PD8	FSMC_D13	Databit 13
D14	PD9	FSMC_D14	Databit 14
D15	PD10	FSMC_D15	Databit 15
CS	PD7	FSMC_NE1	chip select
RS	PD11	FSMC_A16	Address bus
RD	PD4	FSMC_NOE	Read enable
WR	PD5	FSMC_NWE	Write enable
RESET	VDD or GPIO	VDD or GPIO	Reset

Pin connections for touch controller (XPT2046 / ADS7843)

Touch controller	STM32F4 discovery board	Pin Function
Interrupt (IRQ)	PB12	Interrupt on touch event
Data out (DO)	PB14	SPI data out
Data in (DIN)	PB15	SPI data in
CS	PB11	Chip Select
CLK	PB13	SPI Clock

My sincere thanks to ChibiOS and μ GFX developers