Ubuntu Tutorial:

MSPGCC and Eclipse

Alvaro Aguilar

August 17, 2010

Abstract

I put together this guide from various sources I found on the internet. It is meant to help a beginner set-up the Eclipse IDE to work with MSPGCC and be able to use it to debug and program MSP430 devices. This guide is just a compilation, so you may find further insight into some of the steps by visiting the following sites:

1. MSPGCC Wiki
2. MSPGCC4 and MSPDEBUG
3. MSP430 Eclipse MSPGCC Ubuntu Tutorial
4. Eclipse IDE with MSPGCC

Before beginning with the installation, you have to make sure you have the following packages:

- subversion
- texinfo
- patch
- libncurses5-dev
- zlibc
- zlib1g-dev
- libx11-dev
- libusb-dev
- libreadline6-dev

Now that the basics are covered, we will first install and set up mspgcc. We will later tell Eclipse to use these tools in order to compile our C programs and output a code that the MSP430s can understand. Keep in mind during the process the Eclipse is a general working environment, and we must tell it what to do with the files: what compiler to feed them to, and with what parameters.

MSPGCC

There are two option for this package. The older version is more robust, but needs additional installation of certain packages. Basically, you will need to install an older version of `gcc` because Ubuntu ships with v.4.x but you need v.3.x for the original MSPGCC. Additional information can be found at MPGCC Wiki The newer version, mspgcc4, is available for download as a pre-built package.I installed mspgcc version 4 by visiting the following website:

```
  http://sourceforge.net/projects/mspgcc4/files/
```

Extract its contents onto `/opt/mspgcc` and change its permissions using the following command:

```
sudo chown -R $USER.$USER /opt/mspgcc
```

In order to debug your device, you will additionally need "`msp430-gdbproxy`". Run the following set of commands to get it along with its libraries:

```
cd /opt/mspgcc/bin
wget http://www.soft-switch.org/downloads/mspgcc/msp430-gdbproxy
chmod 777 msp430-gdbproxy
cd /usr/lib
sudo wget http://www.soft-switch.org/downloads/mspgcc/libHIL.so
sudo wget http://www.soft-switch.org/downloads/mspgcc/libMSP430.so
```

Now that we have mspgcc installed we must add the `/opt/mspgcc/bin` path to our environment so that we may call these functions from the command line. Edit the file `/etc/environment` and add the path at the end of the line. A reboot is needed in order for the change to take effect.

Finally, there is a small tweak needed in order to recognize the newer MSP430 devices. Running the command `dmesg | tail` after plugging in your device should reveal what is needed. If you see an error saying `ti_download_ firmware - firmware not found`, then you must perform the following step. You will need the firmware called `ti_3410.fw`, which can be found under `/lib/firmware/ti_3410.fw`. Run the following command in order to make it accessible:

```
ln -s /lib/firmware/ti_3410.fw /lib/firmware/ti_usb-3410.bin}
```

Reconnecting the device and running `dmesg | tail` again should generate an output in which you see a line saying:

```
TI USB 3410 1 port adapter converter now attached to ttyUSB0
```

If you don't see this, then you should change the configuration value of the USB device. **Attention:** The USB device that you change will depend on your `dmesg | tail` output, and the device that is recognized (in my example it was 5-1). Become root before attempting to do this.

```
echo 2 > /sys/bus/usb/devices/5-1/bConfigurationValue
```

Since `/dev/ttyUSB0` is owned by root.uucp, you should add your $USER to the uucp group. Replace $USER in the following command in order to add a different user to the group. Also, be sure to use the -a flag in order to *append* the file to the group rather than overwriting its contents.

```
usermod -a -G uucp $USER
```

You may now start msp430-gdbproxy by trying the following command:

```
msp430-gdbproxy msp430 /dev/ttyUSB0
```

If all goes well the proxy should be initialized and ready to receive commands through port 2000.

By the way, this is not the only way to communicate with your device. You may also download a tool called mspdebug. This one supports newer devices that might not be supported by the usual msp430-gdb (like the eZRF-2500). Check out this tutorial for info on how to set it up.

Now that MSPGCC is set-up, we can set up Eclipse in order to work with it.

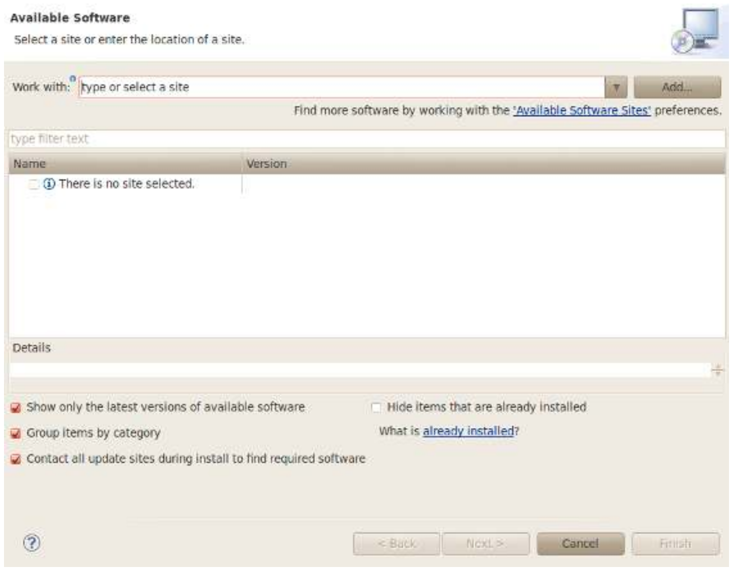
Eclipse

You will need to install Eclipse and a few add-ons. Begin by installing eclipse from the command line:

```
sudo apt-get install eclipse
```

After the installation is done, fire up eclipse. Go to Help : Install New Software...

You will want to install two software packages from the following locations:


Available Software

Select a site or enter the location of a site.

Work with: type or select a site

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

type filter text

Name	Version
There is no site selected.	

Details

- [✓] Show only the latest versions of available software
- [✓] Group items by category
- [✓] Contact all update sites during install to find required software
- [ ] Hide items that are already installed

What is already installed?

Figure 1: Eclipse Install New Software

1. CDT add-on (Choose the one for your version of Eclipse):
   http://www.eclipse.org/cdt/downloads.php

2. Zylin Embedded CDT:
   http://opensource.zylin.com/embeddedcdt.html

On those sites, you will find the correct link to plug into the "Work with:" box in Eclipse Install New Software. When the installation is done, Eclipse will need to reboot for the changes to take effect.

After the restart, you will have to make a new C project. I created a regular project like this:

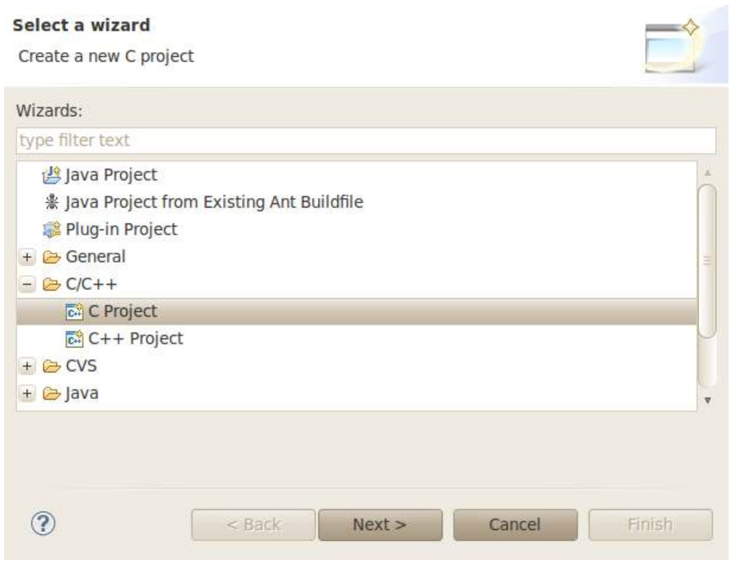
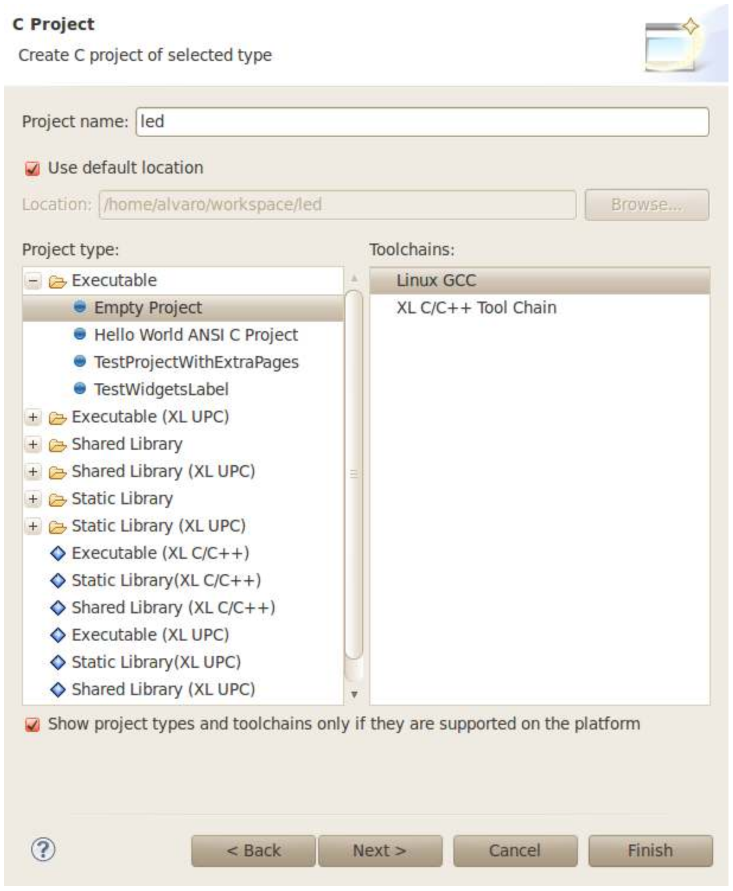


Figure 2: Eclipse New Project

Click next and create a new project (I called mine led) as an Empty Executable:



C Project

Create C project of selected type

Project name: led

[✓] Use default location

Location: /home/alvaro/workspace/led

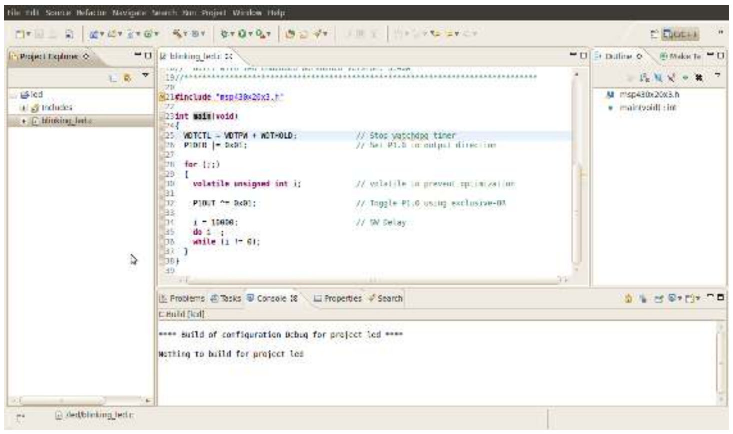
Project type:
- Executable
  - Empty Project
  - Hello World ANSI C Project
  - TestProjectWithExtraPages
  - TestWidgetsLabel
- Executable (XL UPC)
- Shared Library
- Shared Library (XL UPC)
- Static Library
- Static Library (XL UPC)
- Executable (XL C/C++)
- Static Library(XL C/C++)
- Shared Library (XL C/C++)
- Executable (XL UPC)
- Static Library(XL UPC)
- Shared Library (XL UPC)

Toolchains:
- Linux GCC
- XL C/C++ Tool Chain

[✓] Show project types and toolchains only if they are supported on the platform

Figure 3: Eclipse LED C Executable Project

Click finish and you will be greeted by the default C/C++ perspective under Eclipse. On the left side you will see the Project Explorer, where you will be able to visualize everything under your source code. For now, you can drag and drop the flash LED example in there. I got mine from the IAR Embedded Workbench installation on a Windows Machine, but you can download sample code from the TI website. This is what the screen should look like after opening up the source file:



```
//******************************************************************************

#include "msp430x20x3.h"

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
  P1DIR |= 0x01;                        // Set P1.0 to output direction

  for (;;)
  {
    volatile unsigned int i;            // volatile to prevent optimization

    P1OUT ^= 0x01;                      // Toggle P1.0 using exclusive-OR

    i = 10000;                          // SW Delay
    do i--;
    while (i != 0);
  }
}
```

```
**** Build of configuration Debug for project led ****

Nothing to build for project led
```

Figure 4: Eclipse C/C++ Perspective

Now that we have a project made, we need to specify the settings so that Eclipse may build this project using the MSPGCC compiler. Right-click on the project name and select **Properties** from the menu. Go to **C/C++ Build** and expand its contents to select **Settings**. Under the **Tool Settings** tab, you will find settings for the **Compiler, Linker** and **Assembler**. First of all, open up a terminal and type the following command:

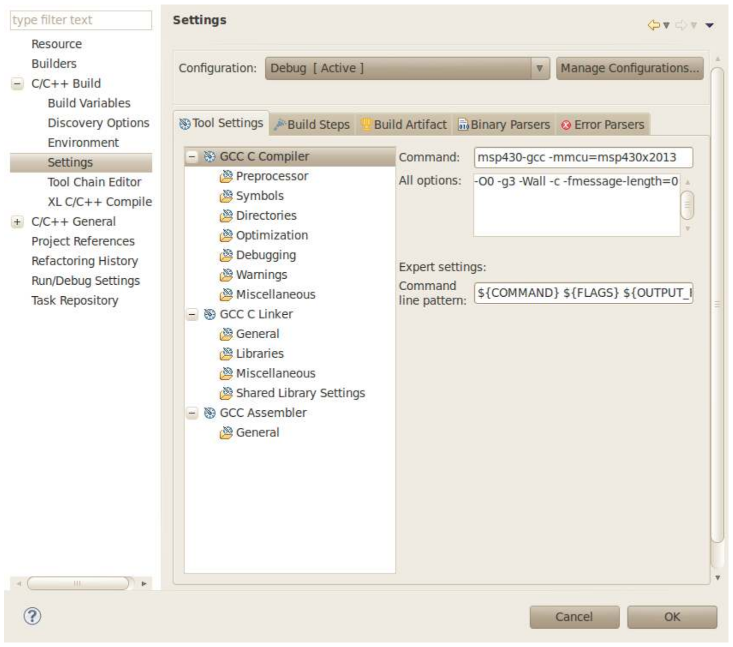
```
msp430-gcc --target-help
```

The output will show all the supported MCU names; keep the correct one for your device in mind. For example, my device is the **MSP430F2013**, and the correct MCU name was **msp430x2013**.

Back on the **GCC C Compiler** option inside eclipse, specify the following command:

```
msp430-gcc -mmcu=msp430x2013
```

Of course, change to your specific MCU name from before.



Configuration: Debug [Active]

Command: msp430-gcc -mmcu=msp430x2013

All options: -O0 -g3 -Wall -c -fmessage-length=0

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_

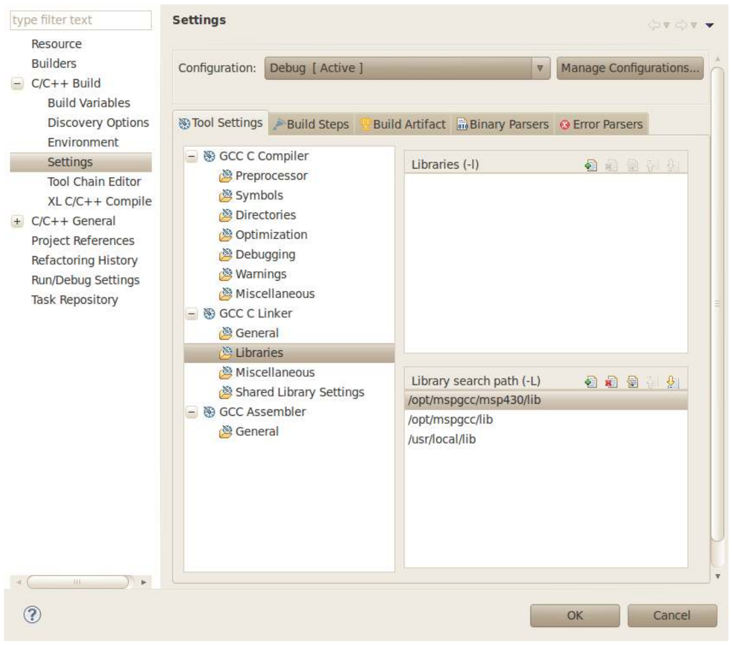
Figure 5: GCC C Compiler options

In order to include header files, the compiler must know where to find them. Go to the **Directories** option under GCC C Compiler and specify the directories where you keep the header files for mspgcc. The default directory is `/opt/mspgcc/msp430/include`. In the end, I included the following directories.

```
/opt/mspgcc/msp430/include
/opt/mspgcc/include
/usr/include
```

Now we need to set up the **GCC C Linker** part in a similar manner. The command needs to be the same, so just copy and paste the same information into that field. However, we must tell the linker where the libraries are, so under **GCC C Linker**, go to **Libraries** and include the following under “Library search path (-L)”:

```
/opt/mspgcc/msp430/lib
/opt/mspgcc/lib
/usr/local/lib
```



Configuration: Debug [Active]

Libraries (-l)

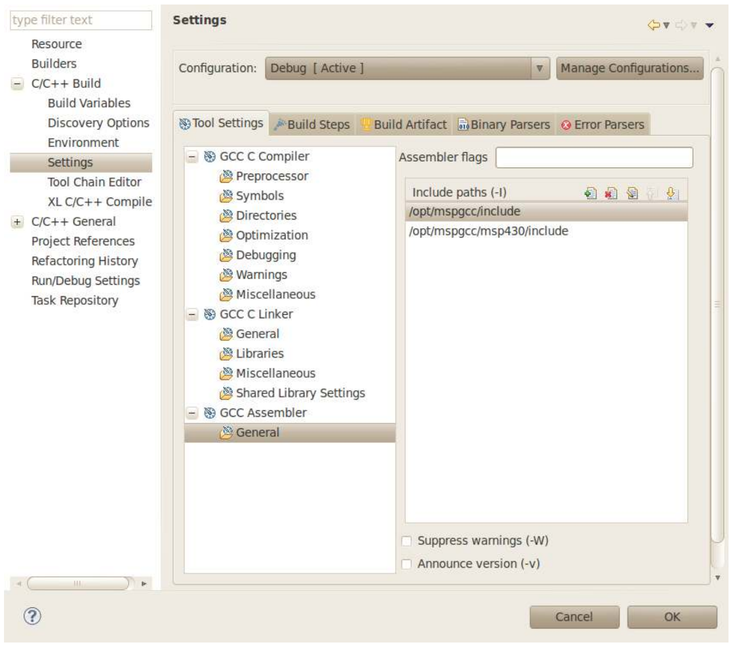
Library search path (-L)

/opt/mspgcc/msp430/lib
/opt/mspgcc/lib
/usr/local/lib

Figure 6: GCC C Linker Libraries

Finally, the Assembler must be set-up correctly, so navigate down to it. This time, the command is different, so in the "Command" field, enter the following: `msp430-as`. As usual, we must also tell this assembler where to find the include files. Click on **General** and add the following paths to the "Include paths (-I)" box:

```
/opt/mspgcc/msp430/include
/opt/mspgcc/include
```



Configuration: Debug [Active]

Assembler flags

Include paths (-I)

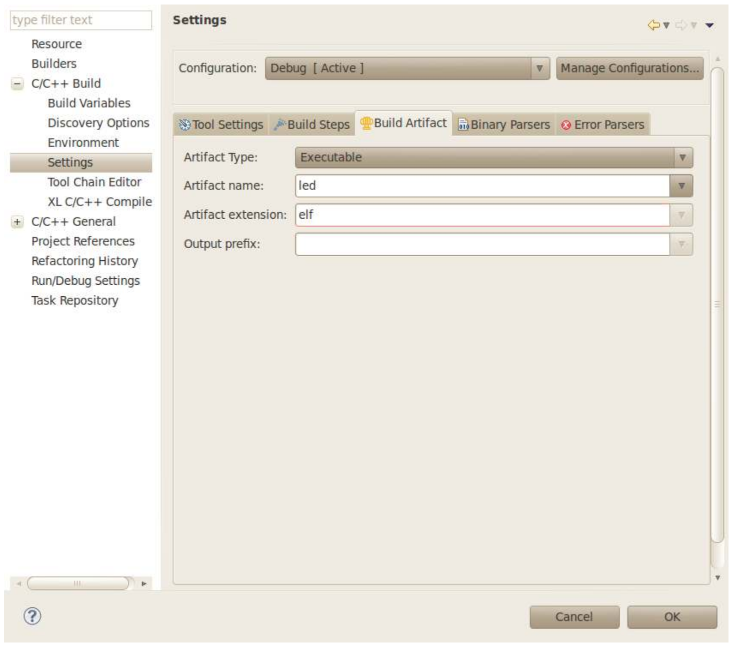
```
/opt/mspgcc/include
/opt/mspgcc/msp430/include
```

Suppress warnings (-W)

Announce version (-v)

Figure 7: Assembler include paths

Now that our compiling tools are ready to be used, we need to tell Eclipse what type of file to create. Head over to the "**Build Artifact**" tab and type in "`elf`" under "**Artifact Extension**". On the next tab over "**Binary Parsers**", make sure that the Elf Parser is selected.



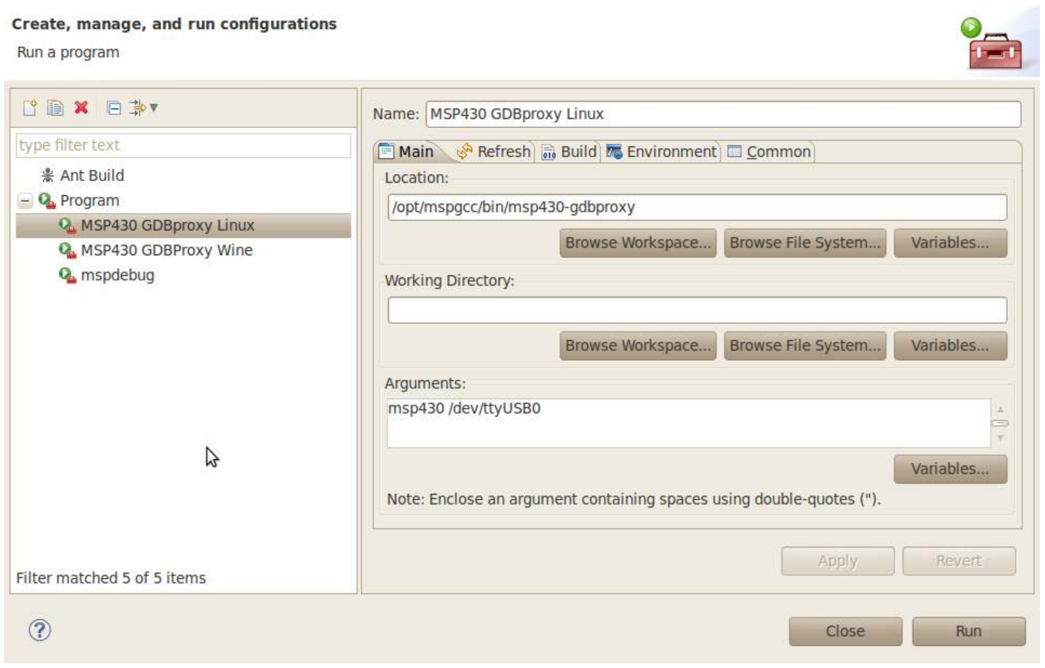
Configuration: Debug [Active]

Artifact Type: Executable
Artifact name: led
Artifact extension: elf
Output prefix:

Figure 8: Choose the elf extension under the Build Artifact tab

Compiling set-up is now finally done, so clicking OK should bring you to the main page. Try building the project by selecting **Project** : **Build Project**. If everything went well, you should have a succesful build. Otherwise, check the console and correct the issues presented.

Now that we have a working executable, we want to debug it. I used `msp430-gdbproxy` along with `msp430-gdb` in order to achieve this. You can also use `mspdebug` by itself to do this step, and the set-up is similar so you may use this guide to set it up under Eclipse as well. The proxy part is just a proxy that forwards the commands from `msp430-gdb` to the actual MSP430 device connected to your computer. Therefore the first thing we will do is set-up Eclipse to run the proxy before anything else. Head over to **Run** : **External Tools** : **External Tools Configurations...**. Create a new configuration and set it up like this. You should take a quick read at the man page of msp430-gdbproxy before configuring the parameters in order to ensure you got the right device. The arguments I passed where msp430 (type of device) and then the port at which it was attached (this should be known from the previously issued command `dmesg | tail`).



Name: MSP430 GDBproxy Linux

Location: /opt/mspgcc/bin/msp430-gdbproxy

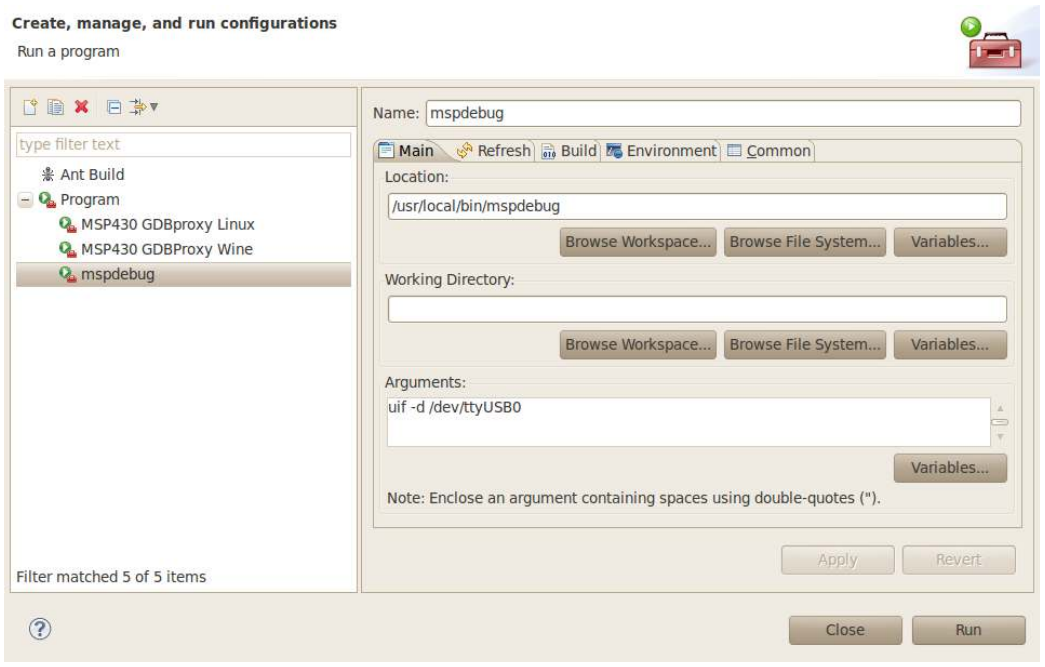
Working Directory:

Arguments: msp430 /dev/ttyUSB0

Note: Enclose an argument containing spaces using double-quotes (").

Figure 9: Setting up external tools in Eclipse

As you can see, I also other tools available. Namely, `msp430-gdbproxy` running under wine. Find a great tutorial for this here. Additionally, I also have a tool for running mspdebug. The following is a screenshot with the parameters that worked for me. However, you should read the man page for that command in order to make sure everything is set-up for your device.


Create, manage, and run configurations

Run a program

Name: mspdebug

Location: /usr/local/bin/mspdebug

Working Directory:

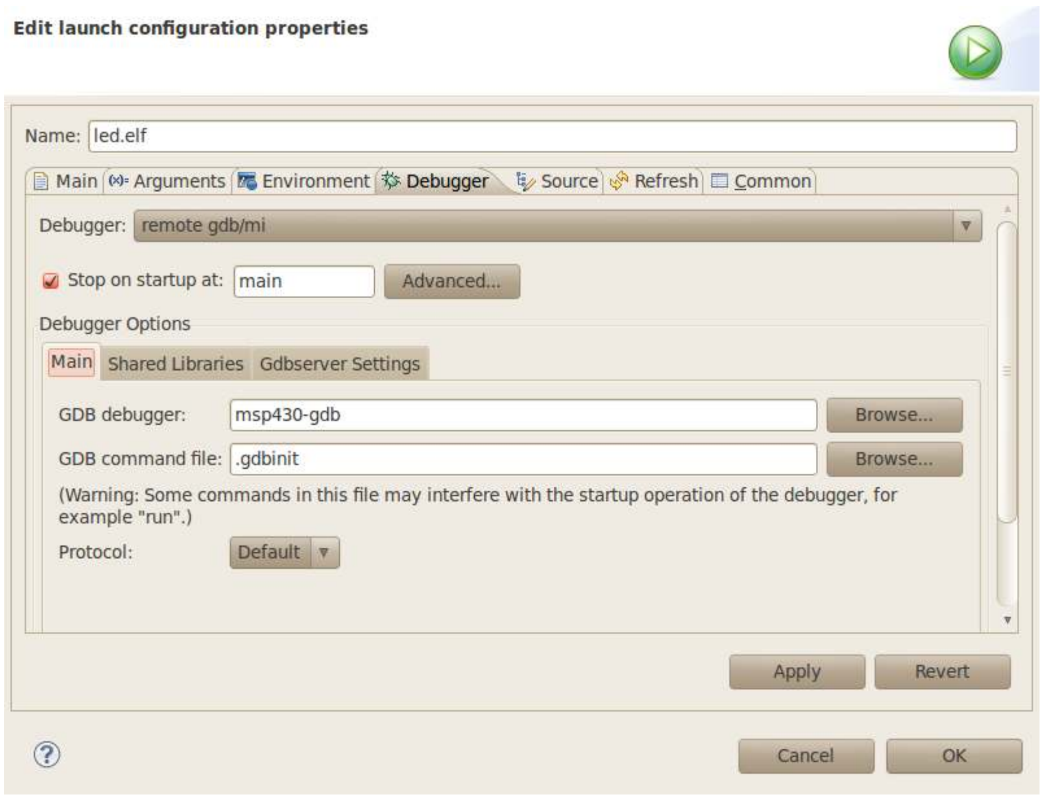
Arguments: uif -d /dev/ttyUSB0

Note: Enclose an argument containing spaces using double-quotes (").

Filter matched 5 of 5 items

Figure 10: Setting up mspdebug in Eclipse

Now that the proxy is set-up, try running it and observe the console output. It should include certain information about your device and end with the line "`Listening on port 2000`" or so. Once that is ready to go, we can set up msp430-gdb to work as the debugger from inside Eclipse. In order to do this, we need to open the Project Properties again. Therefore right-click on the project name, and select Properties from the drop-down menu. Under **Run/Debug Settings**, select **New...** and then click **OK**. Under the "Main" tab ensured that the correct names are given for the Project and for the Executable created during compilation. Now head to the "Debugger" tab and select **remote gdb/mi** from the "Debugger" drop-down list. Type in `msp430-gdb` on the "GDB Debugger" line. Go to the "Gdbserver Settings" tab and select the port that `msp430-gdbproxy` is listening on; the default is **2000** along with **127.0.0.1** for the server name.



Edit launch configuration properties

Name: led.elf

Debugger: remote gdb/mi

Stop on startup at: main

Debugger Options

GDB debugger: msp430-gdb

GDB command file: .gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

Protocol: Default

Figure 11: Setting up msp430-gdb in Eclipse

Finally, create a file in your home directory called `.gdbinit`. Inside of this file, place the following lines:

```
set remoteaddresssize 64
set remotetimeout 999999
target remote localhost:2000
```

Conclusion

Once again, this guide is merely a compilation of different articles found online. If you have questions regarding the procedures, or errors along the way, be sure to check out the following websites:

1. MSPGCC Wiki

2. MSPGCC4 and MSPDEBUG

3. MSP430 Eclipse MSPGCC Ubuntu Tutorial

4. Eclipse IDE with MSPGCC