Building BusyBox/Linux with Navigator™ ICS

This application note describes how to configure your system and import a BusyBox Linux project into Navigator ICS.
.

Document Number: MD01031
Revision 1.00
November 5, 2013

1. Introduction

This document describes how to configure your system and import a BusyBox Linux project into Navigator ICS. Though some of the examples in this document assume you are running on a Malta board, the information is general enough to be useful for most Linux targets. By default, the ROM Monitor installed on Malta boards is YAMON™.

This document assumes that both Navigator ICS and Navigator Console have been installed and that a Navigator probe is already attached via USB.

The overall concept which this document describes is to build BusyBox and the Linux kernel, download the kernel, set a breakpoint and then use a serial console window to issue the standard YAMON **go** command to start execution of Linux.

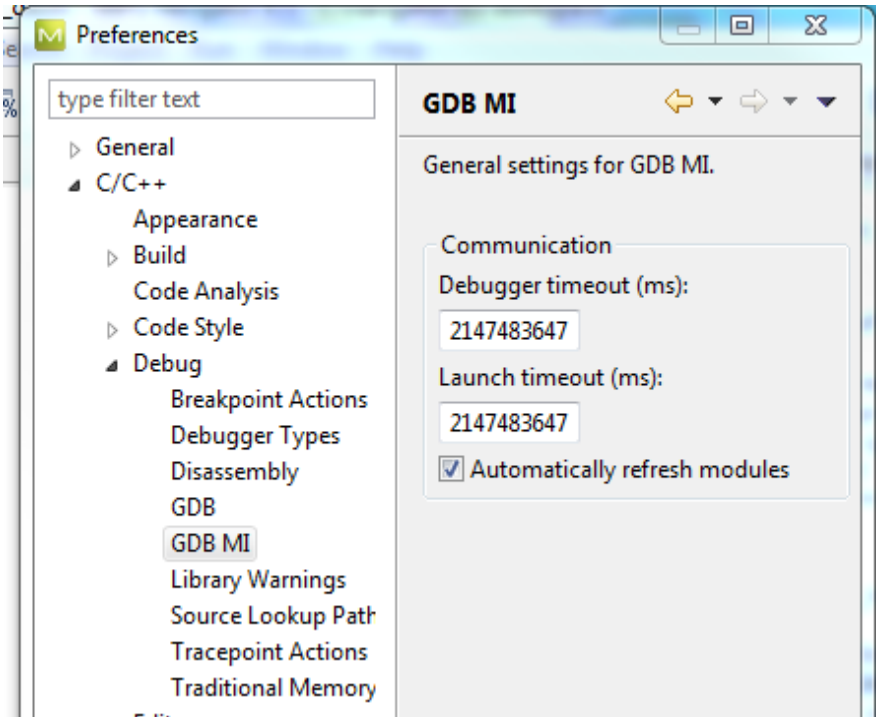
You will need a Linux host system with Navigator ICS installed and a System navigator Probe. You might need to install additional Linux applications on your system. For example, you will need the ncurses and make packages.

2. Initial Software Configuration

At this point it is assumed that you have installed Navigator ICS. If not you can get the latest release at: http://www.mips.com/
For Linux hosts, by default Navigator ICS is installed in your home directory.

Navigator ICS has some default preferences set that are geared more towards bare iron debugging as opposed to Linux debugging. So before creating a Linux project, let's change a few preferences to be better suited for Linux debugging.

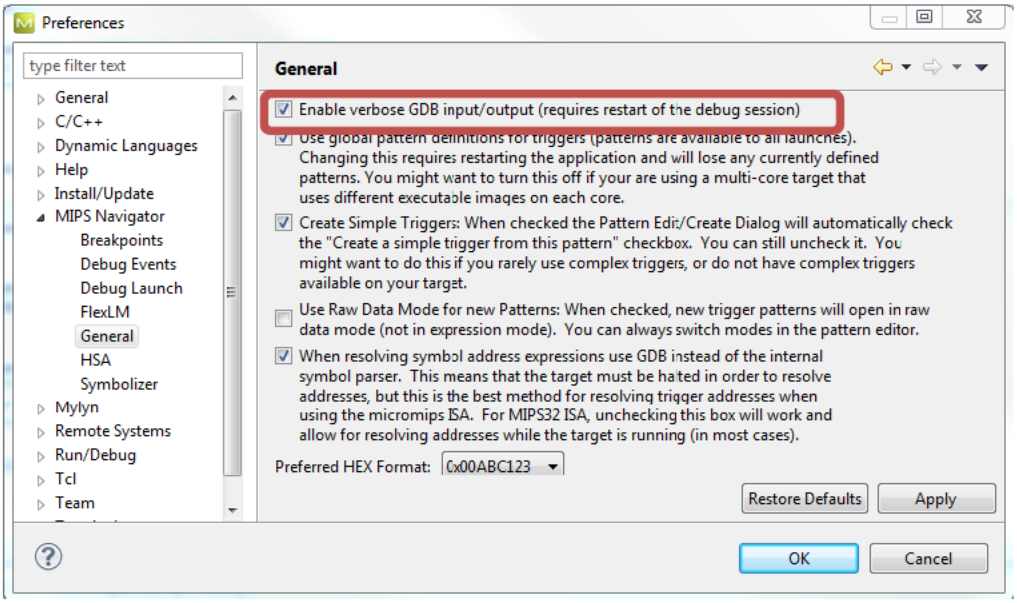
1) On the main menu bar select **Window | Preferences…**
   a. In the tree view on the left side of the **Preferences** dialog box, expand the **C/C++ | Debug** node and select the **GDB MI** node.
   b. On the right side of the **Preferences** dialog box, increase the **Debugger** and **Launch** timeouts to a value that will be high enough to account for the kernel download time (or set them to the maximum, which is 2147483647):



      i. This change is needed because some GDB commands can take a long time to complete (i.e., downloading the kernel), and we do not want Navigator ICS to time out before the GDB command has completed.
      ii. **NOTE:** This may not be needed, or the time can be greatly reduced, if fast data can be used on your target, and the additional steps listed in the Configuring a Debug Launch for Linux section are followed.

c. In the tree view on the left side of the **Preferences** dialog box, expand the **MIPS Navigator ICS** node and select **General.**

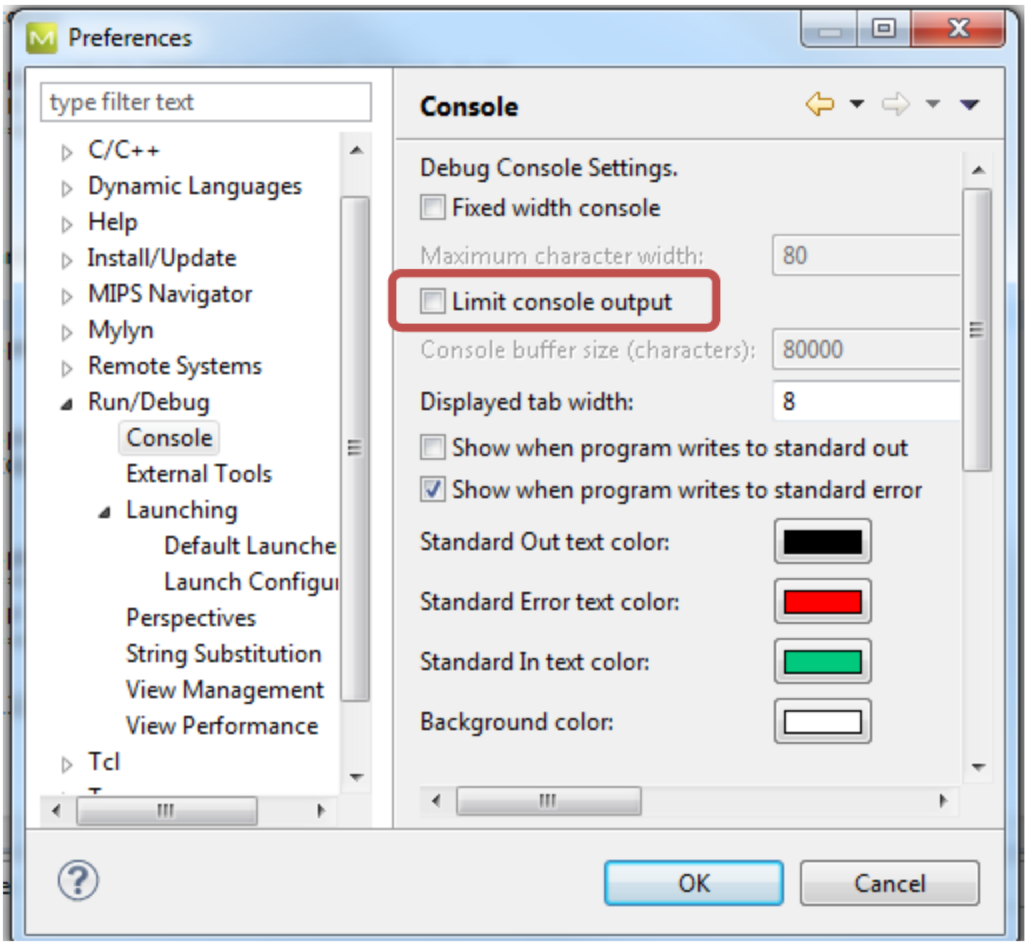
d. On the right side of the **Preferences** dialog box, check the **Enable verbose GDB input/output** checkbox:



i. This will display all the communication between Navigator ICS and GDB in the gdb console window. **NOTE:** By default, there are several console windows on top of each other, and you will have to select the gdb window in order to see this particular communication, using the action buttons at the top of the console view.

e. In the tree view on the left side of the **Preferences** dialog box, expand the **Run/Debug** node and select the **Console** node.

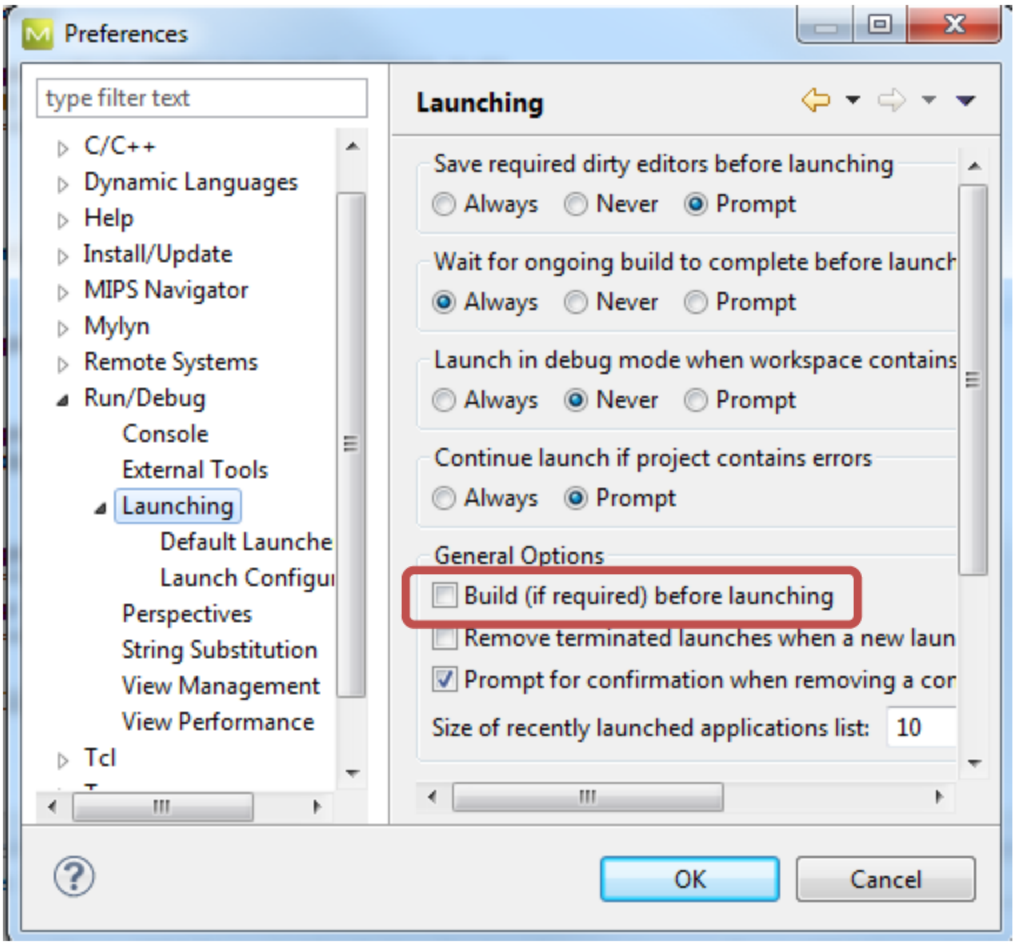
f. On the right side of the **Preferences** dialog box, uncheck the **Limit console output** checkbox:



i. This will allow you to see the entire history of the communication between Navigator ICS and GDB, which can be useful for seeing the address at which the elf file was loaded.

g. With the **Run/Debug** node still expanded, select the **Run/Debug | Launching** node (do not expand the **Launching** node, just select it).

h. On the right side of the **Preferences** dialog box, uncheck the **Build (if required) before launching** checkbox:



i. This will prevent Eclipse from automatically re-building the Linux project each time you launch a debug session. Just remember that the project will have to be manually built before launching a debug session.
ii. Click OK.

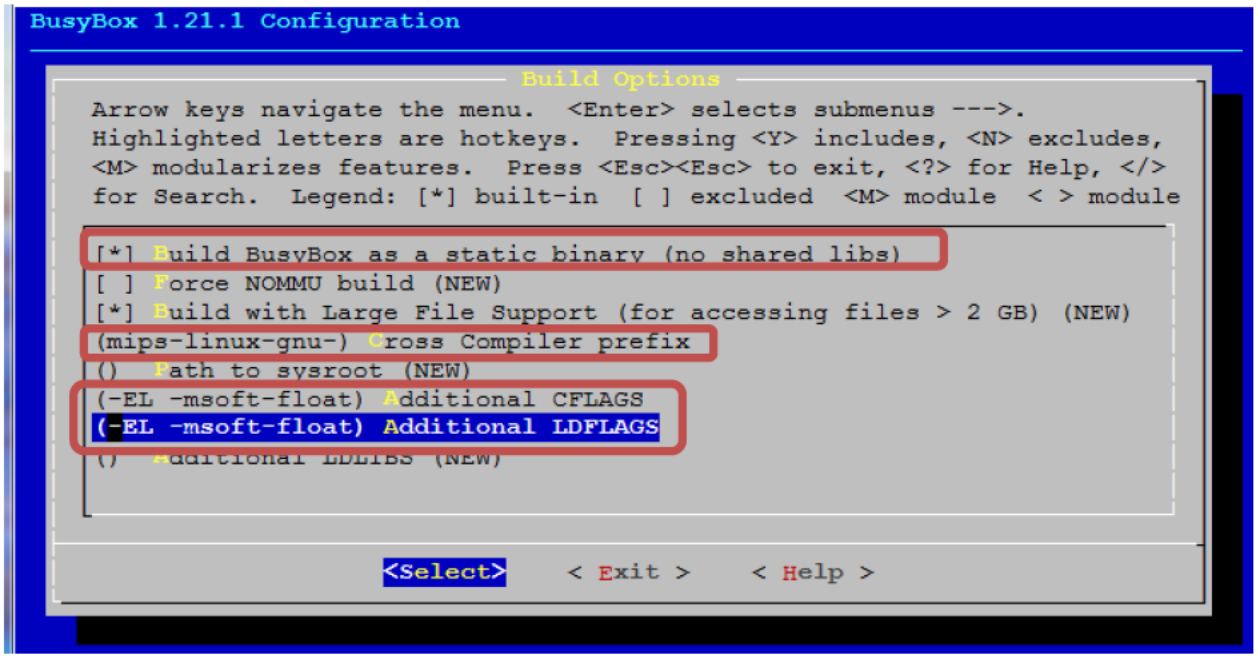
3. BusyBox – The File System

This example uses BusyBox as a root file system. It will build the file system into the Kernel binary. The BusyBox sources can be found at www.busybox.net. It is usually best to select the latest stable release. Download it and do the following:

1. Untar the download. For example: tar -xjf busybox-x.xx.x.tar.bz2 (substitute the name of your download file).
2. Change to the busybox directory: cd busybox-x.xx.x
3. Run: make menuconfig
   - Busybox Settings -> Build Options
   - Select [*] Build BusyBox as a static binary (no shared libs) - Y
   - Select () Cross Compiler prefix and enter mips-linux-gnu-

- Select () Additional CFLAGS (NEW) and enter -EL -msoft-float
- Select () Additional LDFLAGS (NEW) and enter -EL -msoft-float

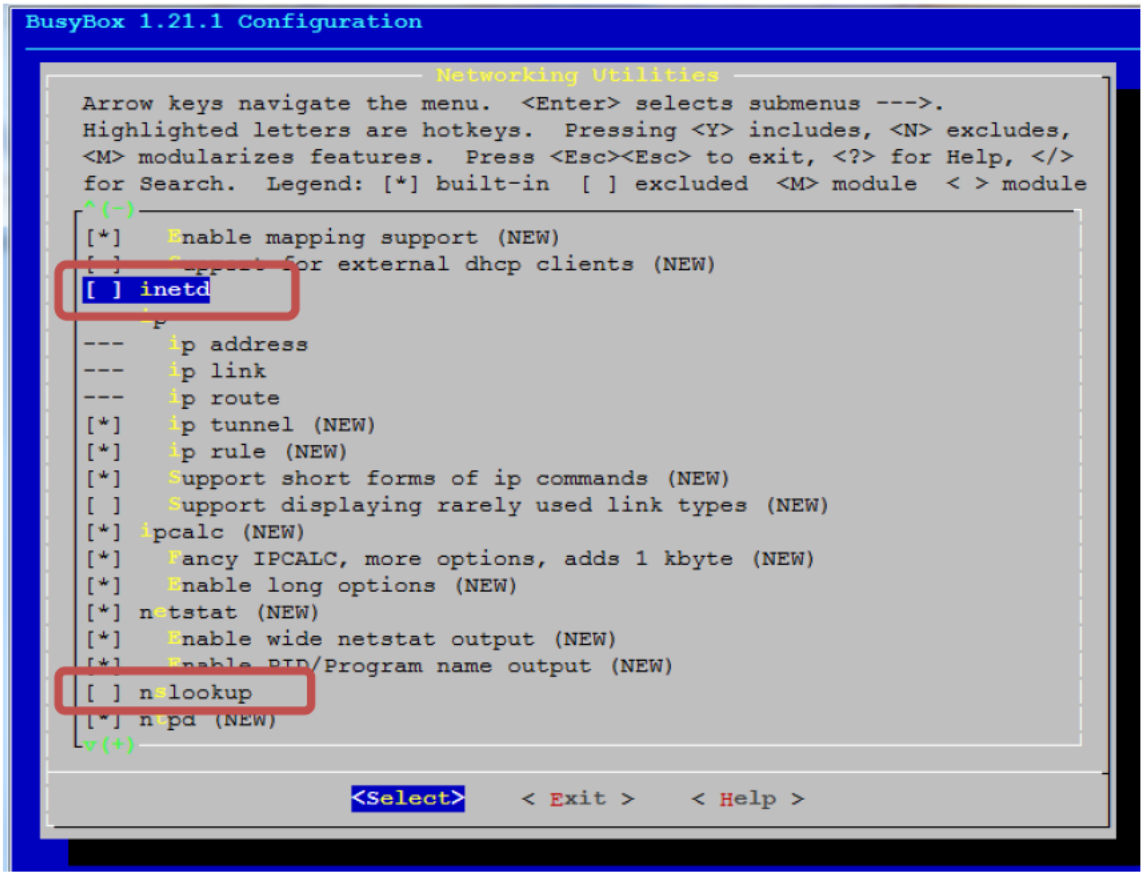
The window should look like this:



4. Exit - Exit (at the bottom of the screen).

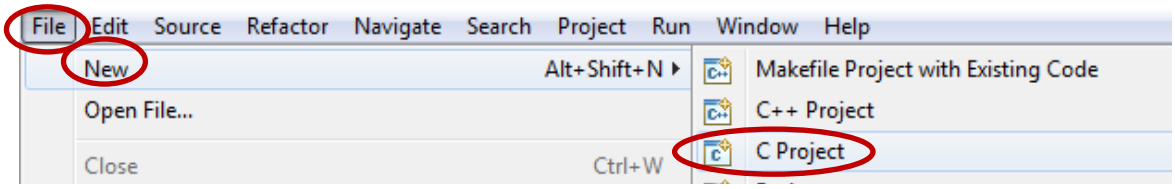
At the the time of this writing, there were problems with some of the network utilities, so these were unselected from the configuration, as detailed here:

5. Select Networking Utilities and then unselect (N) nslookup and inetd. The window should look like this:

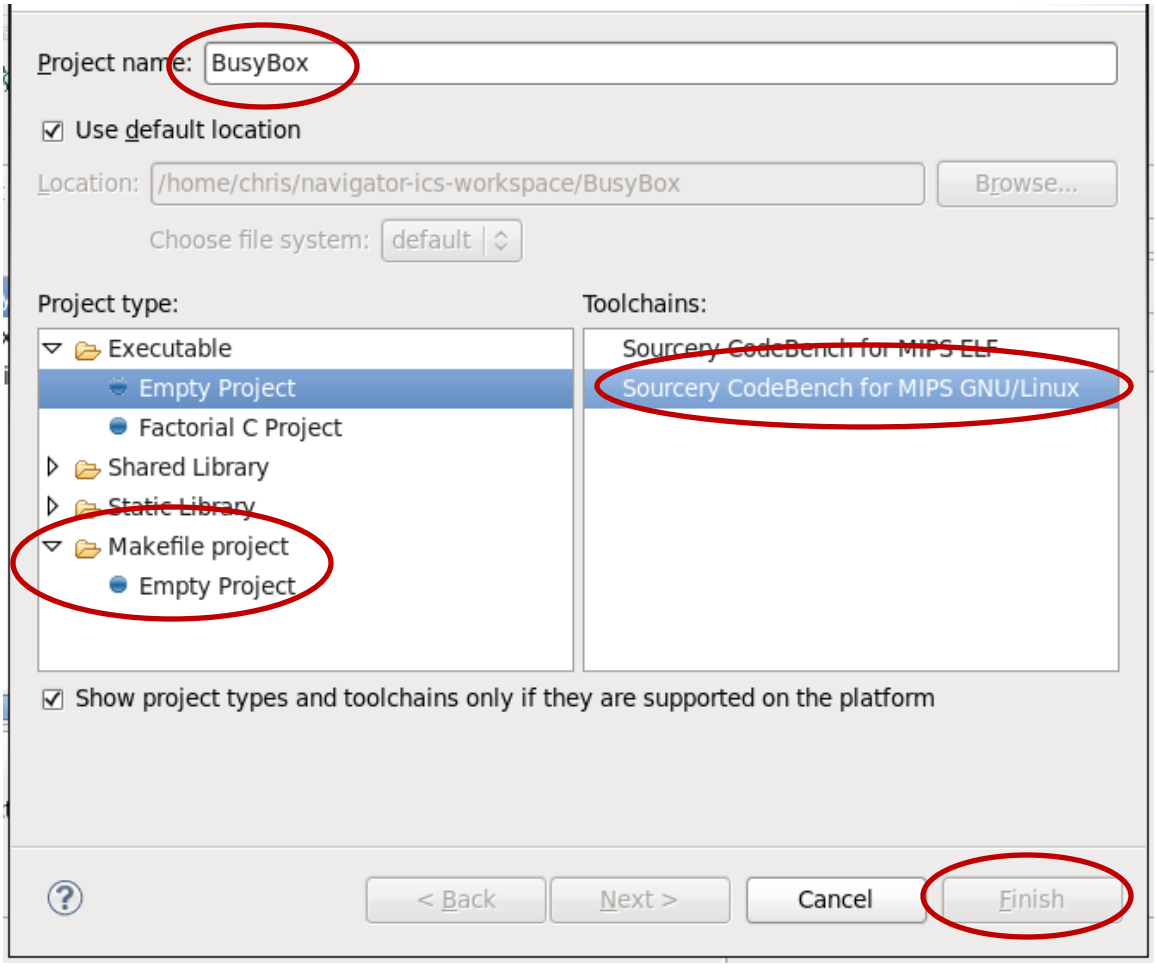


6. Exit- Exit – Yes.

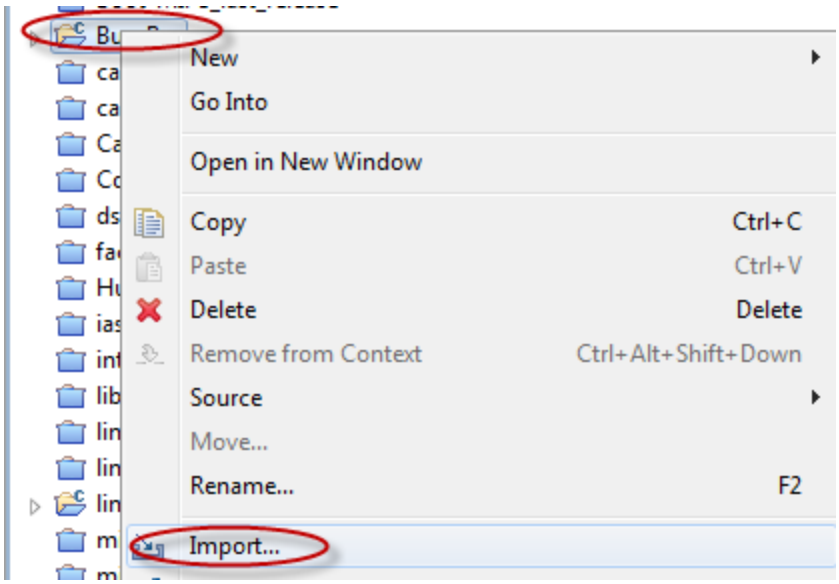
7. Create a Navigator ICS BusyBox Project by selecting File-> New-> C Project.



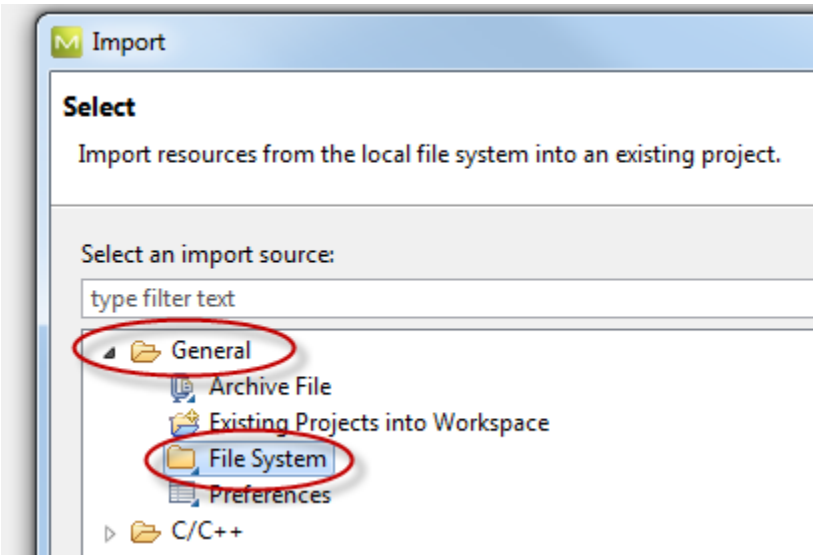
8. Fill in the "Project name", and select "Makefile project" and "Empty Project". Also select "Sourcery CodeBench for MIPS GNU/Linux":



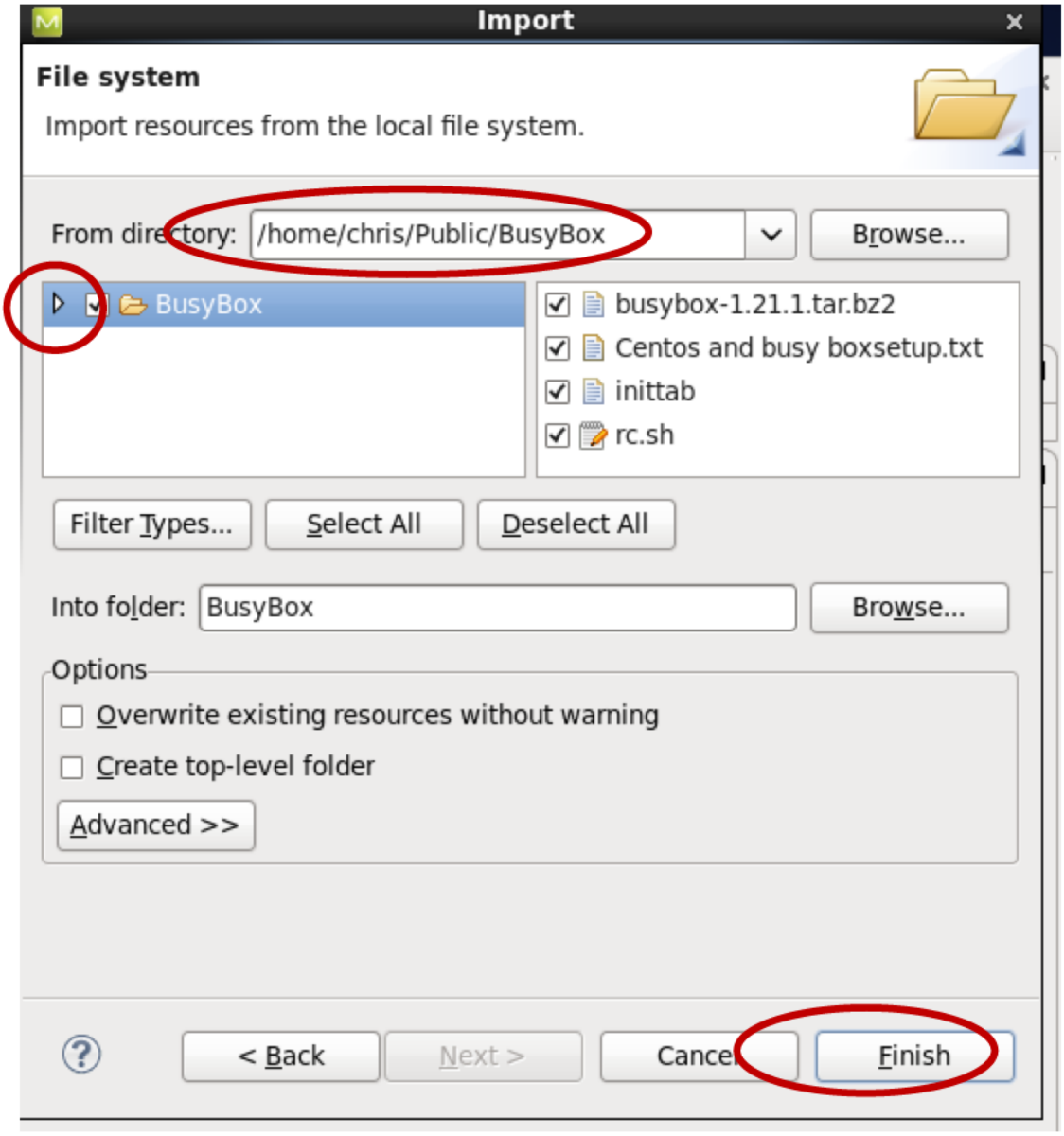
9. In the "Project Explorer", right click on the project name (BusyBox) and select "Import":



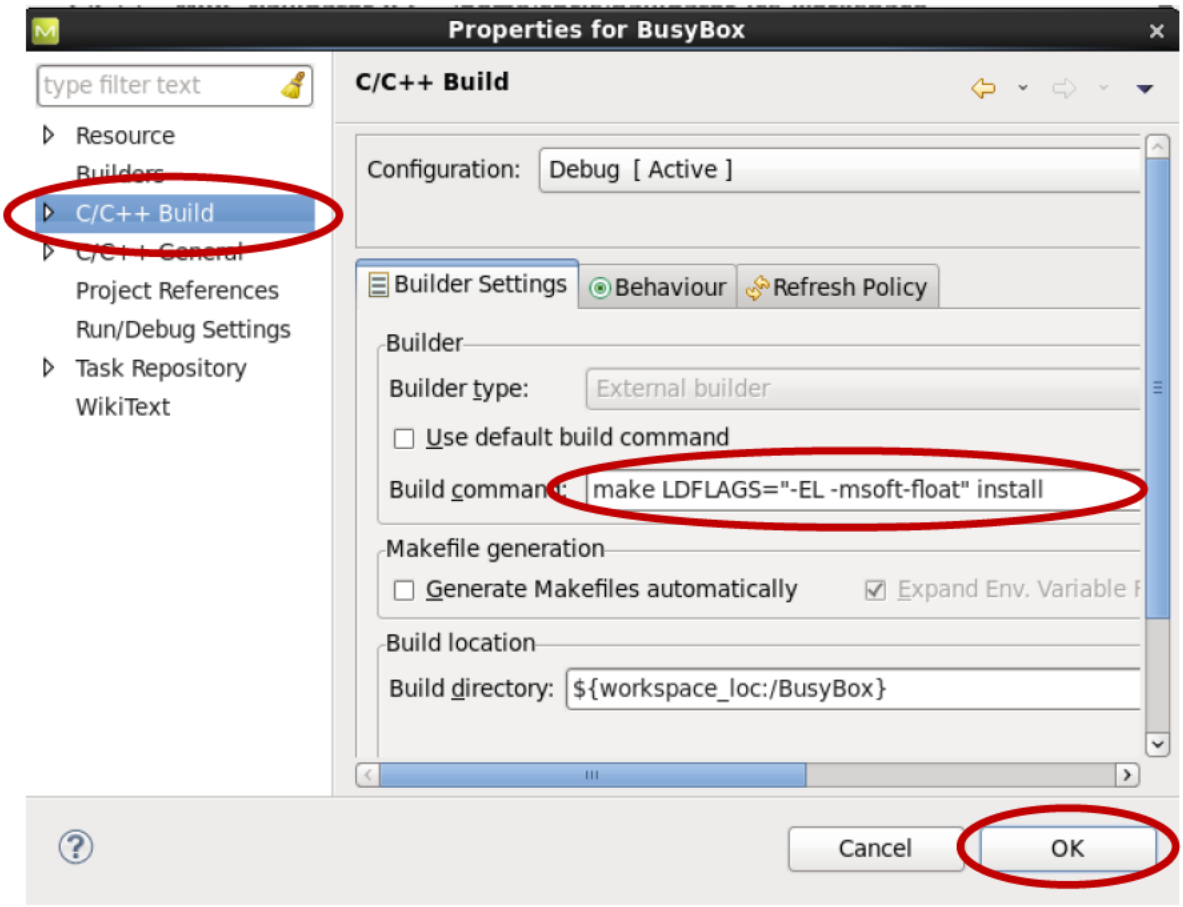
10. Expand the “General” item and select “File System”:



11. Click “Next”. Enter a file name in the “From directory” by browsing to the BusyBox source directory. Then select BusyBox in the Left pane and click on “Finish”:



12. In the "Project Explorer", right click on BusyBox and select "Properties", then select "C/C++ Build". Unselect the "Use default build command", and for the "Build command", enter "make LDFLAGS="-EL –msoft-float" install". Then click "OK":



13. By default, this will install BusyBox to BusyBox source directory _install.
14. Outside of Navigator ICS in a Terminal window: cd _install
15. ln -s bin/busybox init
16. mkdir dev
17. cd dev
18. (as root user) mknod -m 600 console c 5 1
19. cd ..
20. mkdir etc
21. mkdir sys
22. mkdir proc
23. cd etc
24. Cut and past the following to the file inittab:

```
# This is run first except when booting in single-user mode
#
::sysinit:/etc/rc.sh

# /bin/sh invocations on selected ttys
#
# Start an "askfirst" shell on the console (whatever that may be)
::askfirst:-/bin/sh
# Start an "askfirst" shell on /dev/tty2-4
```

```
#tty2::askfirst:-/bin/sh
#tty3::askfirst:-/bin/sh
#tty4::askfirst:-/bin/sh

# Stuff to do when restarting the init process
::restart:/sbin/init

# Stuff to do before rebooting
::ctrlaltdel:/sbin/reboot
::shutdown:/etc/shutdown
```

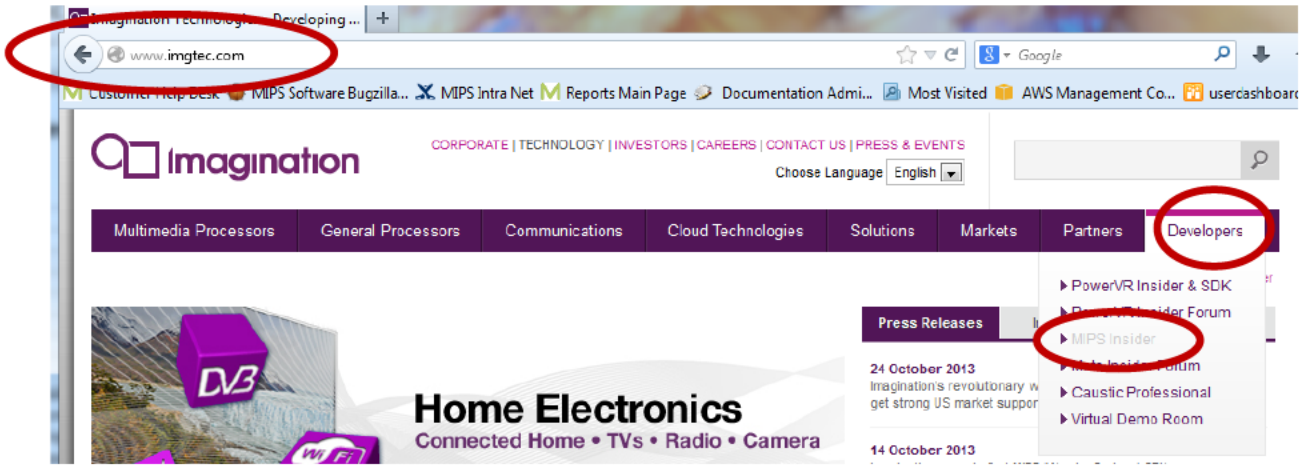
25. Cut and paste the following to a file named rc.sh:

```
#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mknod /dev/tty2 c 4 2
mknod /dev/tty3 c 4 3
mknod /dev/tty4 c 4 4
```

26. Copy rc.sh and inittab to _install/etc/
27. chmod 755 _install/etc/rc.sh

4. Getting the Linux Source

For the latest kernel sources, go to www.imgtec.com. In the Developers drop-down menu, select MIPS Insider:



On the MIPS Insider page, select MIPS Linux:

MIPS Linux

MIPS Linux is a port of Linux to the MIPS architecture.

Follow the directions to download the Kernel source:

— You must first clone the git repository with the following command:

```
git clone git://git.linux-mips.org/pub/scm/linux-mti.git
```

After cloning the git repository, you can then locally check out the branch you are going to work from. For example, to start developing with the 3.8 branch, do the following:

```
cd linux-mti
git checkout linux-mti-3.8
```

This creates a local copy of the 3.8 branch that tracks the remote branch in the repository that you cloned from. To keep your local git repository up to date, you do a pull from the remote repository with:

```
git pull
```

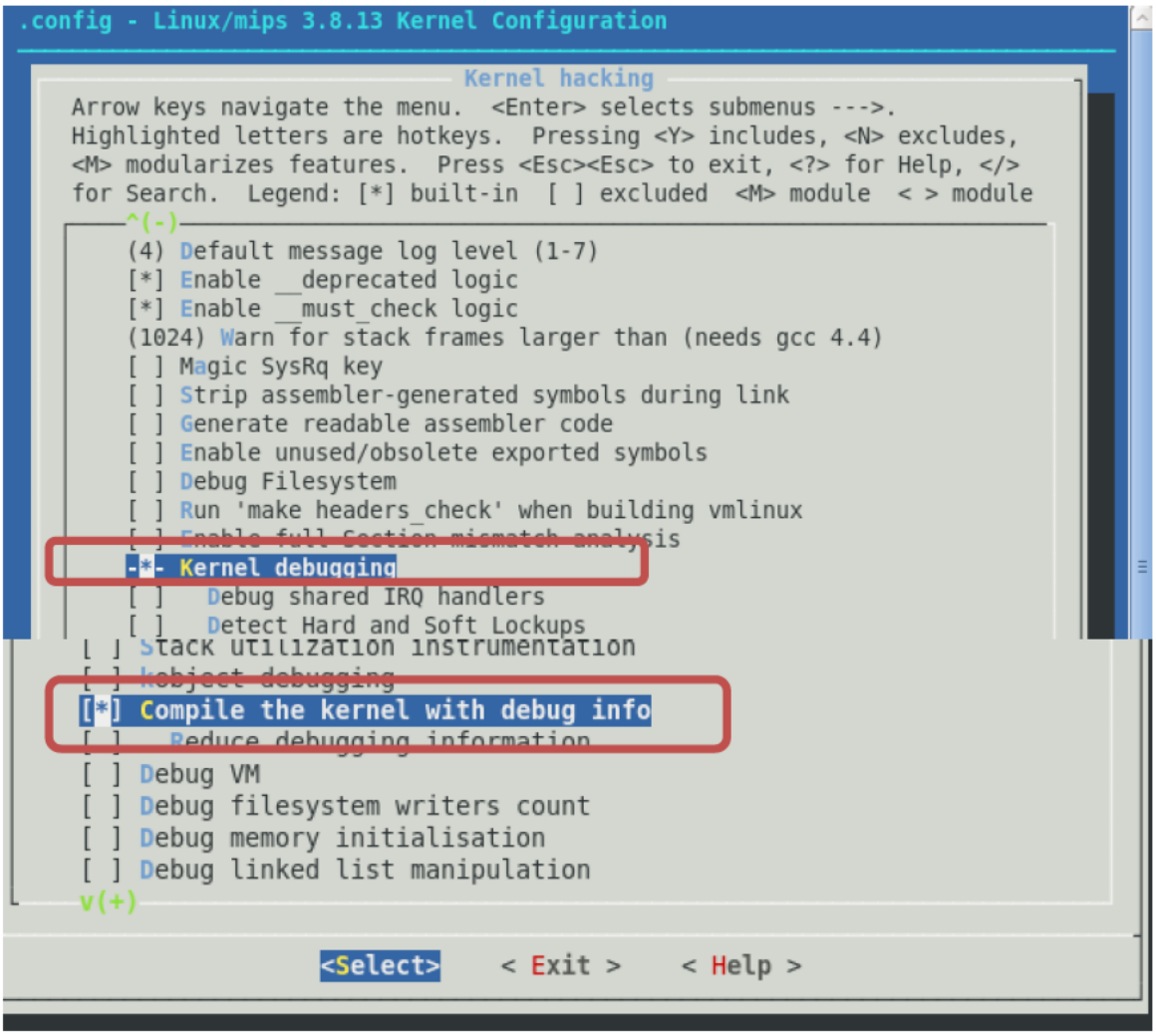
Git will notify you as part of the pull operation if the remote branch was updated or not.

The initial configuration of the kernel .config file is the only portion of building the Linux kernel that needs to be run outside of Navigator ICS. The .config file is not modified directly, but instead is modified using a simple GUI. Invoke the GUI by typing **make menuconfig**.

1) In a console window, navigate to the directory of your kernel source.
2) To make sure the kernel source is clean, at this point you can type **make distclean**. This only needs to be run once. **NOTE:** This command deletes any config file, so make sure it is run before creating the .config file or running **make menuconfig.**
3) There are several preconfigured config files located in the **../arch/mips/configs** directory. To use one of these configurations as a starting point, type **cp arch/mips/configs/maltasmvp.defconfig .config** in the console window **from the root kernel directory**. **NOTE:** After the copy, this file should be in the root kernel directory (not in the arch/mips/configs directory).
   For example: cp arch/mips/configs/maltasmvp_defconfig .config
4) To edit the config (this example is for little endian), type **make menuconfig**.
5) The Kernel should be built with debugging symbols enabled.
   - To enable debugging symbols, select **Kernel hacking**.
   - Scroll down to **Kernel debugging** and press "**Y**".

```
File  Edit  View  Search  Terminal  Help
.config - Linux/mips 3.8.13 Kernel Configuration

                 Linux/mips 3.8.13 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module

        Machine selection  --->
        Endianness selection (Little endian)  --->
        CPU selection  --->
        Kernel type  --->
        General setup  --->
    [*] Enable loadable module support  --->
    [*] Enable the block layer  --->
        Bus options (PCI, PCMCIA, EISA, ISA, TC)  --->
        Executable file formats  --->
        Power management options  --->
    [*] Networking support  --->
        Device Drivers  --->
        Firmware Drivers  --->
        File systems  --->
        Kernel hacking  --->
        Security options  --->
    -*- Cryptographic API  --->
        Library routines  --->
    ---
        Load an Alternate Configuration File
        Save an Alternate Configuration File

                <Select>    < Exit >    < Help >
```

- Scroll down to **Compile the kernel with debug info** and press "**Y**".
  The window should look like this:



- Select **Exit**.

6) To Build in BusyBox from the top menu, select **General setup**:

```
File  Edit  View  Search  Terminal  Help
.config - Linux/mips 3.8.13 Kernel Configuration
                Linux/mips 3.8.13 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module

        Machine selection  --->
        Endianness selection (Little endian)  --->
        CPU selection  --->
        Kernel type  --->
        General setup  --->
    [*] Enable loadable module support  --->
    [*] Enable the block layer  --->
        Bus options (PCI, PCMCIA, EISA, ISA, TC)  --->
```

7) For the Cross-compiler tool prefix, enter **mips-linux-gnu-**.

```
    [*] Prompt for development and/or incomplete code/drivers
    (mips-linux-gnu-) Cross-compiler tool prefix
    (cmp) Local version - append to kernel release
    [*] Automatically append version information to the version string
        Kernel compression mode (Gzip)  --->
```

8) Scroll down to **Initial RAM filesystem and RAM disk (initramfs/initrd) support** and enter **Y**.

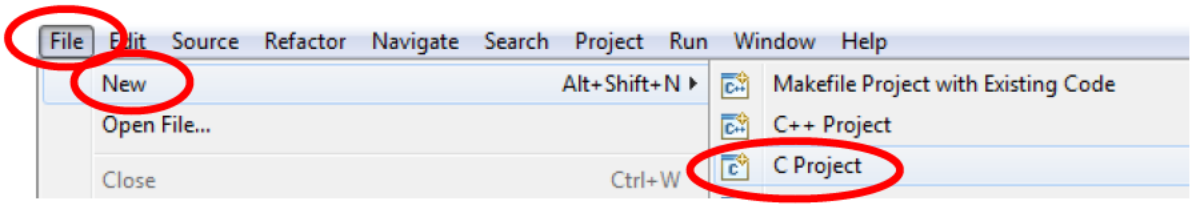
9) Scroll down and select **Initramfs source files** and enter the BusyBox install path:

```
    [ ] Kernel->user space relay support (formerly relayfs)
    [*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
    (/home/chris/navigator-ics-workspace/BusyBox/_install) Initramfs source file(s)
    (0)     User ID to map to 0 (user root)
```

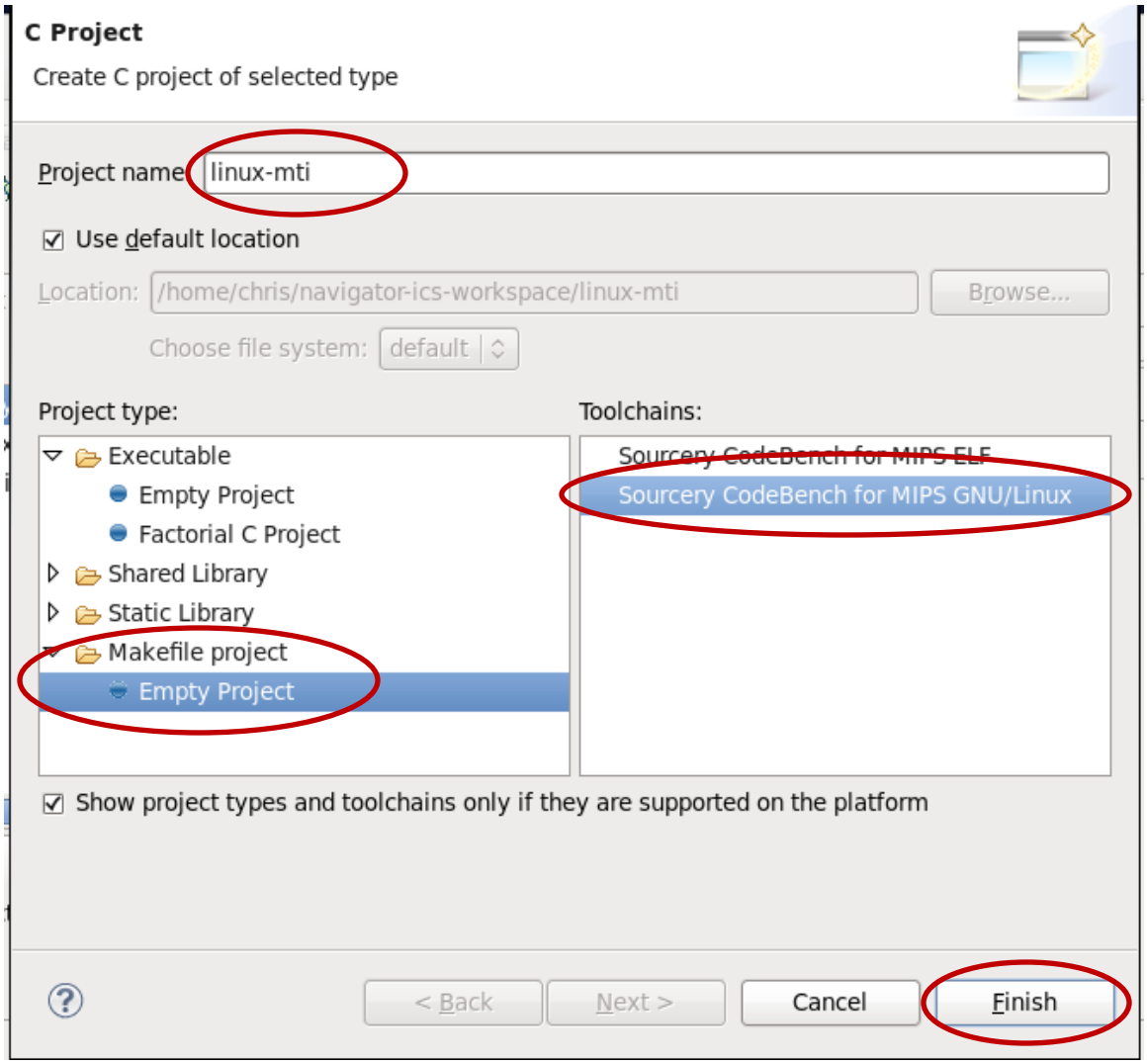
10) **Exit Yes** to save the configuration.

5. Importing a Linux Project

1) If not already in the C/C++ perspective, switch to that perspective now by clicking **Window -> Open Perspective -> Other…** and selecting **C/C++** from the dialog box that opens.
2) On the main menu bar, click **File -> New -> C project.**

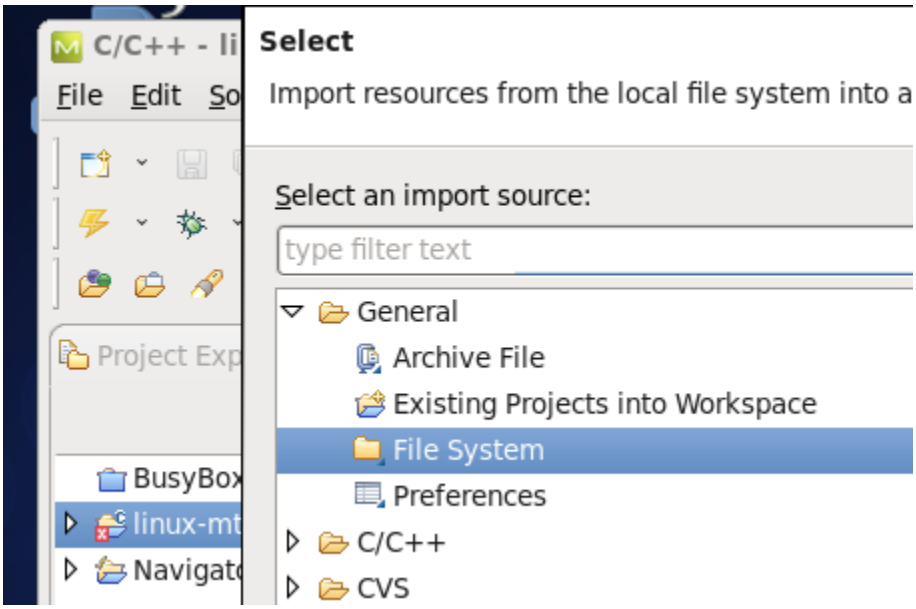


3) In the C Project dialog box:
   a. Type the desired project name in the **Project name** text box.
   b. In the **Project type** list box, select **Makefile project** and **Empty Project**.
   c. In the **Toolchain** list box, select the **Sourcery CodeBench for MIPS GNU/Linux**. It should appear similar to this:



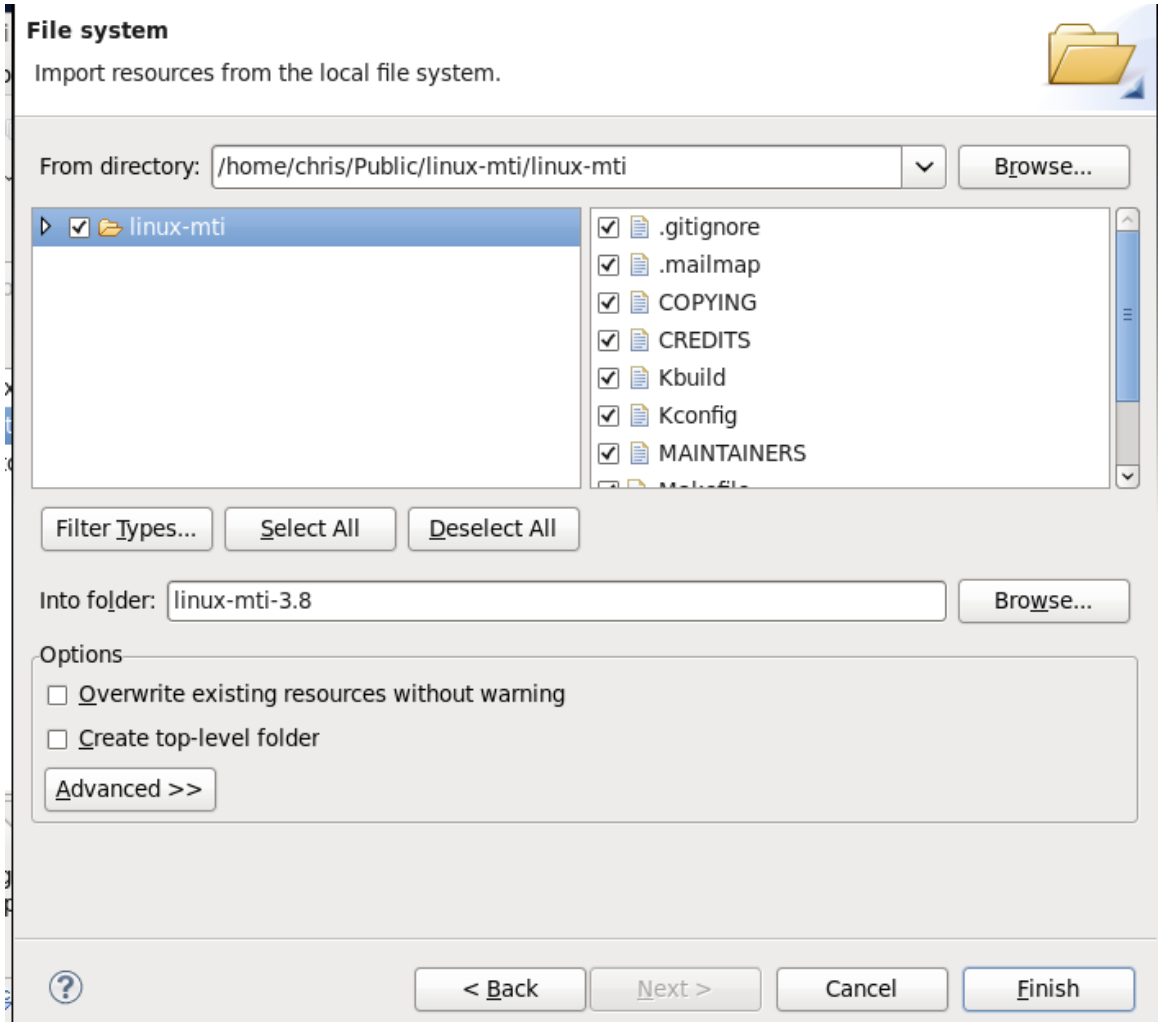
4) Click the **Finish** button.

5) To import the Linux tree, right-click on the project name (the name you entered above) in the “Project Explorer” and select “Import”. In the select screen, expand the **General** item and select **File System**.



6) Click **Next**.

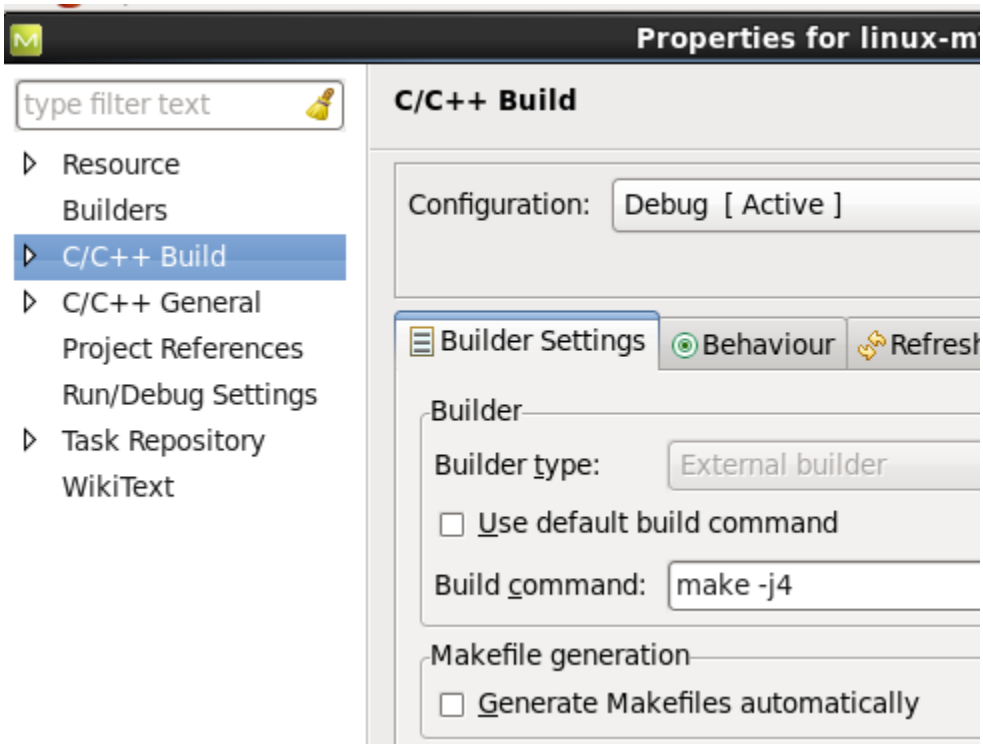
7) Select the **From directory** by browsing to the top of your Linux source tree.
8) Then select the **linux-mti** box to select all of the files in the directory and click **Finish**.



6. Configuring the Build

The build command should be similar to **make –j4 .** This command will use up to 4 processors for the build, which should make it faster. Of course, your build speed will depend on your Linux system.

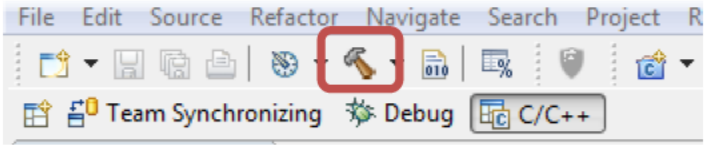
1) With the **linux-mti** project selected in the "Project Explorer", right click and select **Properties**. This will bring up the properties dialog.
2) In the treeview on the left side, select the **C/C++ Build**.
3) On the right side under the **Builder Settings** tab, uncheck the check boxes **Use default build command** and **Generate Makefiles automatically**.
4) Also, on the right side under the builder settings tab, enter **make –j4** in the **Build Command**.
   The screen should look like this:



5) Click **OK**.

7. Building Linux

To start building the kernel, select the **linux-mti** project in the **Project Explorer** view, and then select the hammer icon from the tool bar at the top of the Navigator ICS screen.



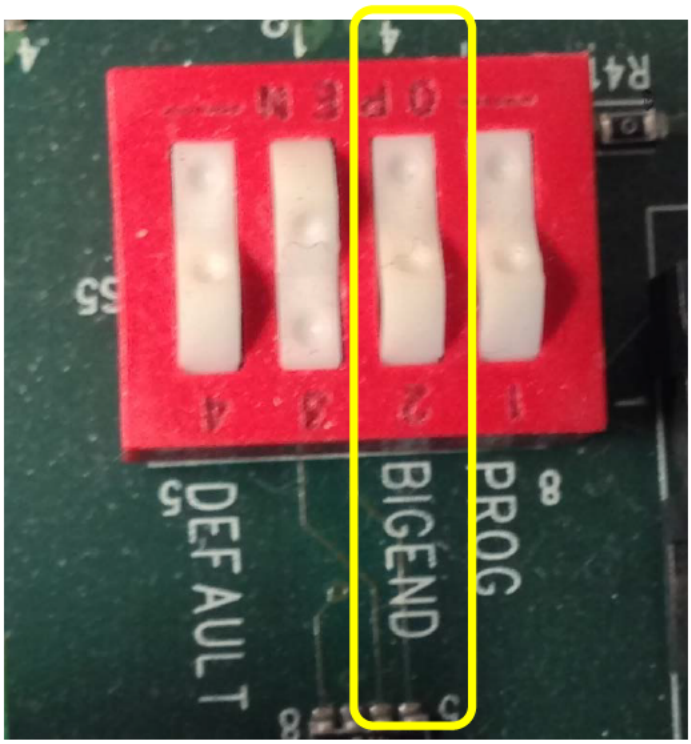
You should see the output of the build in the console window. If not, refer to the trouble shooting section at the end of this document.

NOTE: If you want to use YAMON and TFTP to download the System, then you will need to create a .srec file and save it to your **tftpboot** directory:

```
mips-linux-gnu-objcopy –O srec vmlinux /tftproot/vmlinux.srec
```

8. Malta Configuration

This section assumes your target has a serial port. The baud rate and other settings are correct for a standard Malta board. Also make sure your target endianness is set the same as the cross-compiler toolchain that you are using. Switch 2 of S5 sets endianness on a Malta board. The picture below shows it set for Little Endian.



1) Connect a System Navigator probe to the EJAG connector on the target board.
2) Connect a serial cable to the target board.
3) Then (outside of Eclipse) open a serial console window (minicom) with the following configuration (for a Malta board):
    - **Baud Rate:** 38400
    - **Data bits:** 8
    - **Parity:** None
    - **Stop Bits:** 1
    - **Flow Control:** None

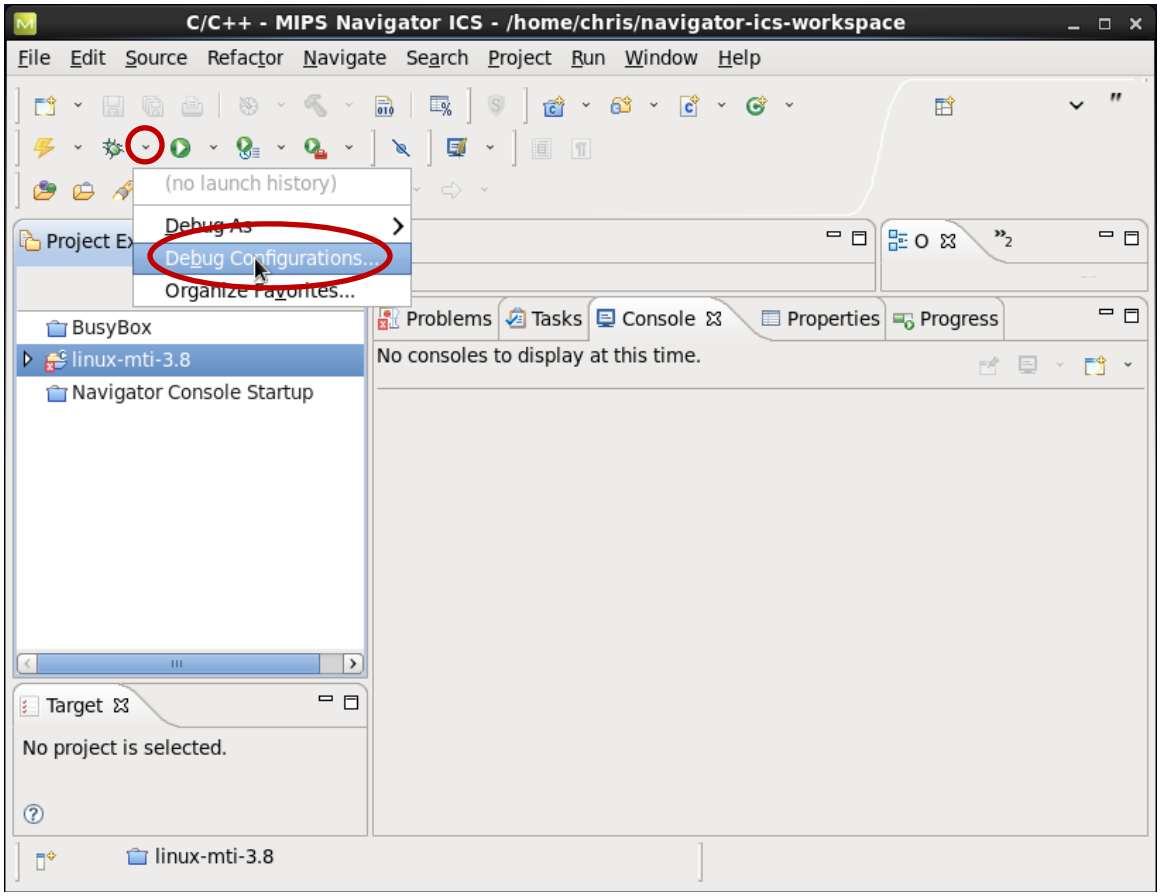
At this point, if you apply power to the Malta board (or press the reset button), you should see text similar to that shown below and be sitting at a YAMON prompt.

```
YAMON ROM Monitor, Revision 02.14.
Copyright (c) 1999-2007 MIPS Technologies, Inc. - All Rights Reserved.

For a list of available commands, type 'help'.

Compilation time =              Dec 17 2007  12:19:49
Board type/revision =           0x02 (Malta) / 0x00
Core board type/revision =      0x09 (CoreFPGA-3) / 0x01
System controller/revision =    MIPS SOC-it 101 OCP / 1.3   SDR-FW-1:1
FPGA revision =                 0x0001
MAC address =                   00.d0.a0.00.02.b8
Board S/N =                     0000000448
PCI bus frequency =             30 MHz
Processor Company ID/options =  0x01 (MIPS Technologies, Inc.) / 0x00
Processor ID/revision =         0x93 (MIPS 24Kc) / 0x80
Endianness =                    Little
CPU/Bus frequency =             32 MHz / 32 MHz
Flash memory size =             4 MByte
SDRAM size =                    64 MByte
First free SDRAM address =      0x800b7de0

YAMON>
```

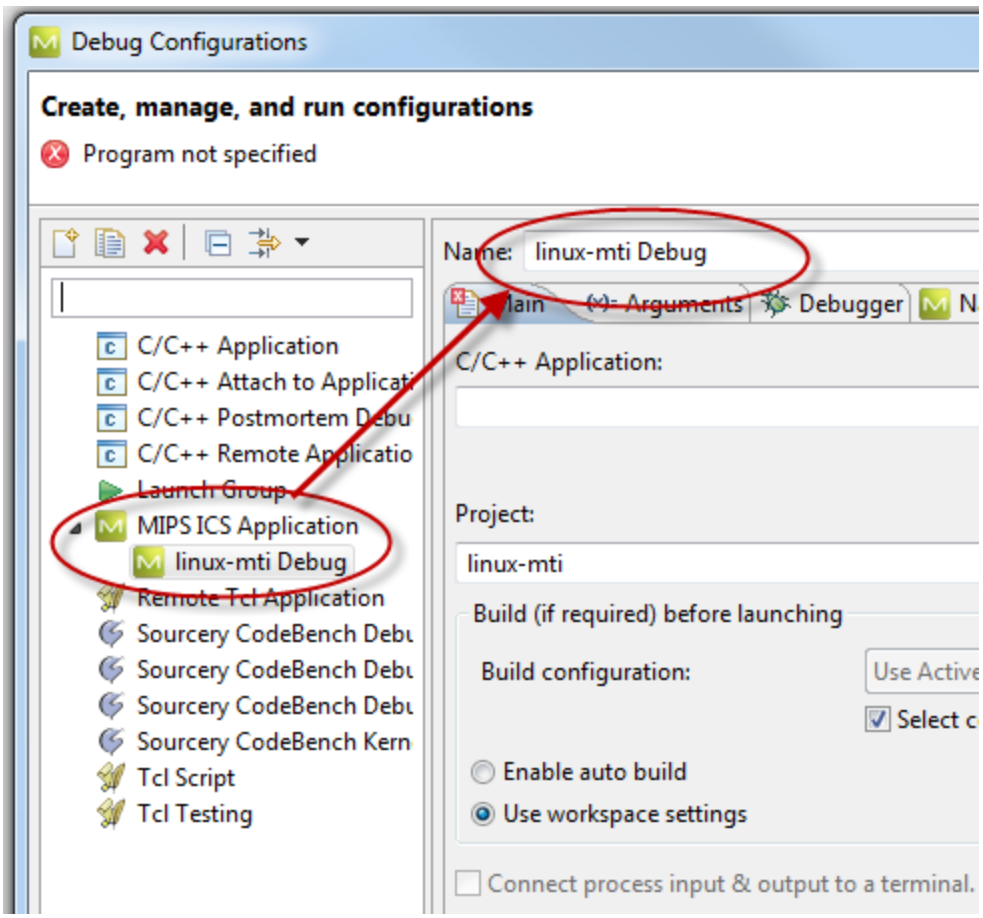
If you do not see this information, refer to the troubleshooting section at the end of this document.

9. Configuring a Debug Launch for Linux

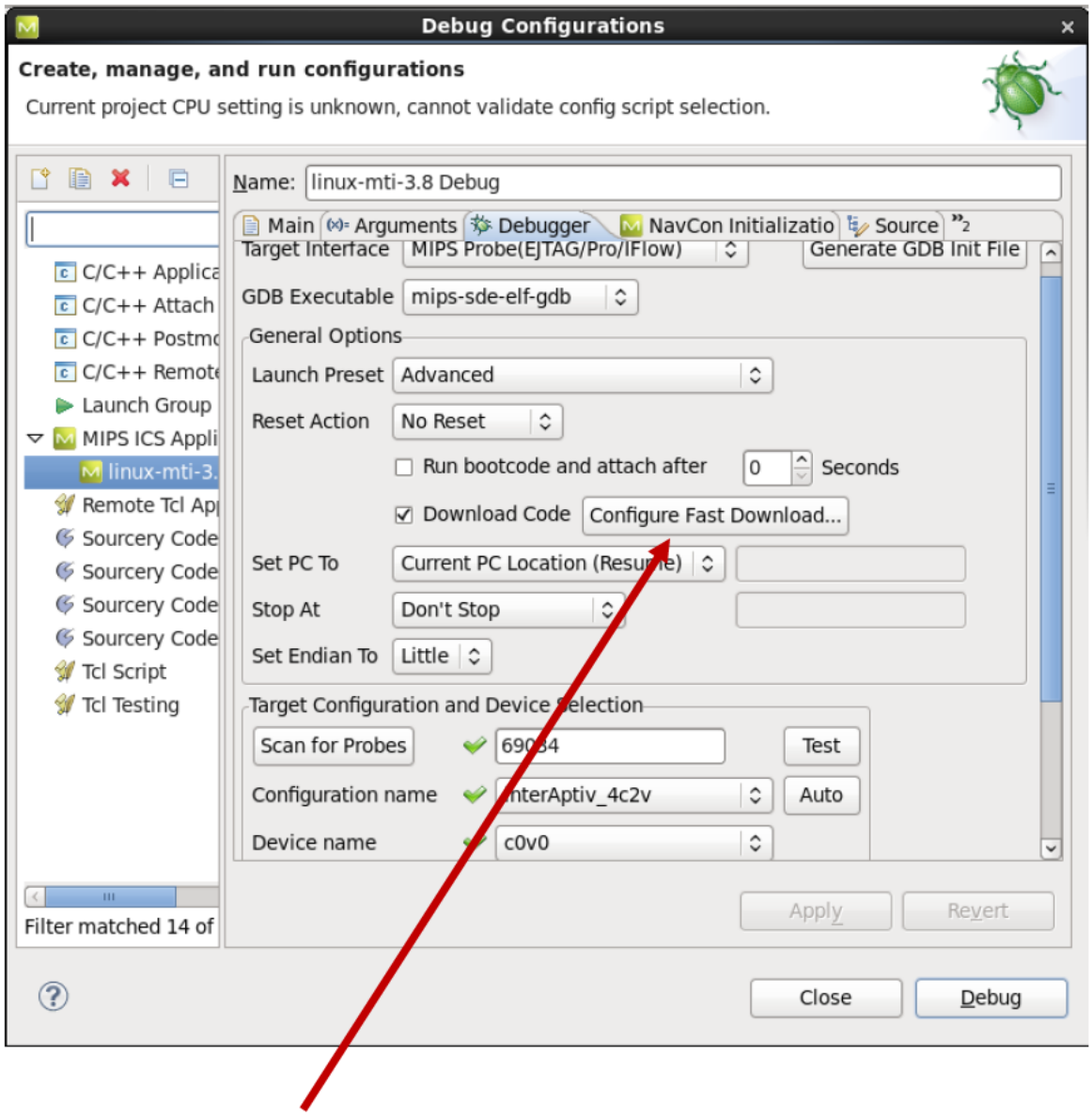
1) Switch to the **C/C++** perspective.
2) Make sure the Linux project is selected in the **Project** pane. Then use the bug icon pull-down menu from the tool bar (at the top of the Navigator ICS screen) to select **Debug Configuration**.



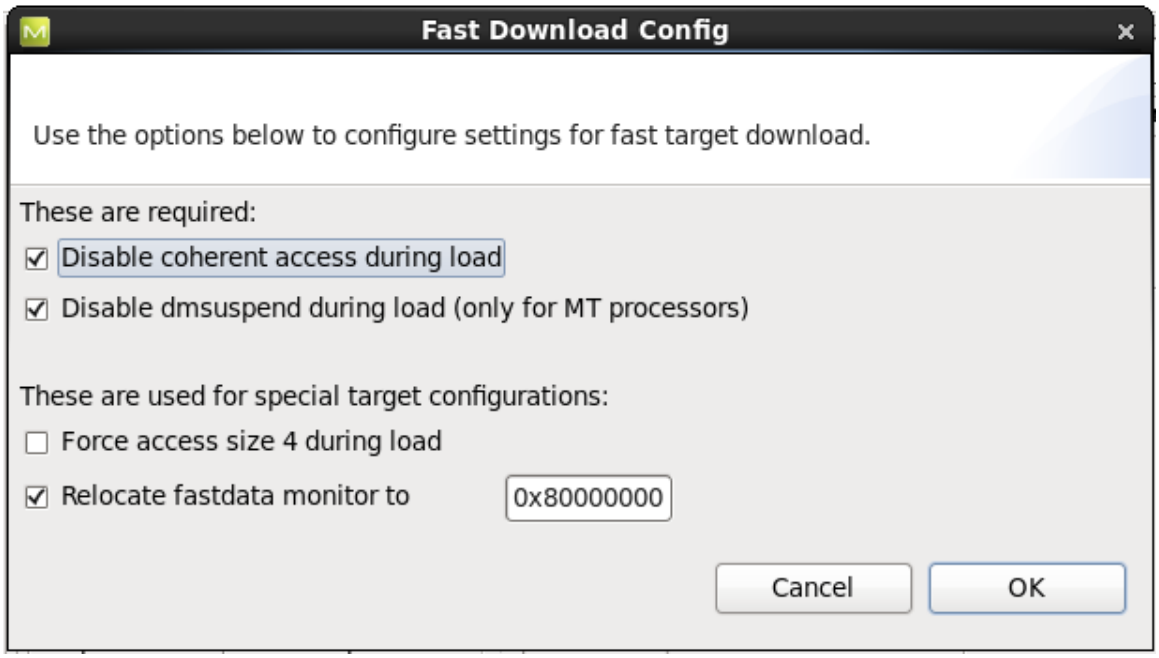
3) In the **Debug Configuration** dialog, double-click on **MIPS ICS Application**. This will create a Debug Configuration using the name of the project. You can change the name to suit your preference.



4) On the **Main** tab of the debug LCD:
   - Under **C/C++ Application**, click the **Search Project** button and select the desired application (i.e., vmlinux which is in the top of the Linux directory tree).
5) On the **Debugger** tab, select the desired settings for your target. Although some options for Linux debugging are specific to your target, other options should generally be set as follows:



6) **Configure Fast Download** will greatly improve download speeds when downloading the kernel through the System Navigator probe. This should be set as follows:



7) Click the **Debug** button on the debug launch control dialog (LCD).
8) After the debug session has launched, the ROM monitor (i.e., YAMON) will be running, but the Linux kernel will not yet be running. At this point, a breakpoint can be placed in Linux. To set a breakpoint from within Eclipse, open the desired source file (i.e., **../init/main.c**) and double-click in the left margin of the source file on the desired line of code (i.e., **start_kernel**).
9) To start the final Linux booting process, enter the standard command that would be entered from the ROM monitor (i.e., YAMON) command prompt in the serial console window (i.e., **go 0x80100400 root=/init**). This command may vary depending on the bootloader/ROM monitor used. It also assumes that BusyBox is being used as the file system.

10. Troubleshooting Q & A

Q: I don't see any output in the serial window when I reset the Malta board.

A: Make sure the serial cable is connected to the correct serial ports on both the Malta board and PC. Then disconnect the System Navigator probe from the Malta board. When reset is pressed, you should now see the text as shown above in this document and something similar to **YAMON** on the display of the Malta board.

Q: I see output in the serial window, but it looks different from that shown above.

A: Make sure the YAMON **start** variable is not defined. To check this, make sure you are at a YAMON prompt (**YAMON>)** and type **setenv,** then press enter. You should see the environment variable displayed on the screen. If the start variable has a value, then type **unsetenv start.**

Q: When I reset the Malta board I see power on in the Malta board display.

A: Disconnect the System Navigator probe to make sure it is not holding the processor in debug mode, and cycle power on the Malta board. If the message remains, then YAMON has probably been erased and will need to be reflashed.

Q: When I clicked the Hammer icon to start the build, I received an error message.

A: First read the error message to see if it gives you a clue as to what is going wrong. Common reasons are that the build command is mistyped, the cross compiler toolchain **bin** directory is not in the path, or the cross compiler toolchains have not been installed. You can always open a **shell** and issue the build from there to make sure everything is setup properly.

Q: I don't know what kernel compilation commands I can type in the a terminal window shell

A: At the shell prompt type **make CROSS_COMPILE=mips-linux-gnu- help**

Q: The code doesn't download or doesn't appear to be downloading properly.

A: Make sure you have built the kernel source with the proper toolchain (especially check the endianness of the toolchain). This will be most visible by checking the build command by right-clicking on the project in the **Project Explorer** and selecting **properties**, then clicking the **C/C++** build node. Also make sure your target is set for the proper endianness. On a Malta board, the endianness is set by switch 2 of S5.

Q: It looks like my debug Launch has just stopped.

A: Most likely your Linux kernel is being downloaded. If you look at the red lights on the System Navigator probe, there should be 2 lights on while the code is being downloaded. One light is the power light and the other is the communication light, which blinks when the probe is communicating with the target. When the code is being downloaded, the communication light is blinking so fast it appears to be solid.