ESP32 setup on Eclipse IDE running on Ubuntu

Overview

Following is the procedure to setup ESP32 on Eclipse running on Ubuntu

```
step 1: Prerequistes for toolchain and ESP-IDF
step 2: Setup toolchain ( c/c++ compiler)
step 3: Setup ESP_IDF (containing software libraries and source code)
step 4: Setup Eclipse for ESP32 build environment
step 5: Setup debugger in Eclipse
```

!!! Note: Setup version during the time of documentation

```
Ubuntu: 18.04LTS
Toolchain version: xtensa-esp32-elf-gcc 8.2.0 for esp32-2019 release-1
ESP_IDF Version:Release V2.0
Eclipse IDE Version:Eclipse IDE 06-2019
Hardware: ESP32-WROOM
Debugger: JLink segger edu
```

Step 1: Prerequistes for toolchain and ESP-IDF

Install dependencies for ESP-IDF by running below command

> sudo apt-get install git wget libncurses-dev flex bison gperf python python-pip python-setuptools python-serial python-click python-cryptography python-future python-pyparsing python-pyelftools cmake ninja-build ccache

Install dependencies for toolchain by running below command

Note:

!!! Run any one of below mentioned command depending on the ubuntu version

For ubuntu 16.04 and newer

> sudo apt-get install gawk gperf grep gettext python python-dev automake bison flex texinfo help2man libtool libtool-bin make

For ubuntu version previous of 16.04

> sudo apt-get install gawk gperf grep gettext libncurses-dev python python-dev automake bison flex texinfo help2man libtool make

Step 2: Setup toolchain

There are 2 ways of building toolchain

```
1: Building toolchain from source (Hard way of doing)
2: Downloading prebuilt toolchain from ESP website (Easy way of doing)
```

Suggestion:

Choose anyone way of setting up of toolchain. I personally recommend to follow 2nd way i.e, downloading pre-built toolchain fron ESP website

Method 1: Building toolchain from source

Create the working directory and go into it

```
mkdir -p ~/esp
```

```
cd ~/esp
```

Download crosstool-NG and build it

```
git clone https://github.com/espressif/crosstool-NG.git
```

```
cd crosstool-NG
```

```
git checkout esp32-2019r1
```

```
./bootstrap && ./configure --enable-local && make install
```

Build toolchain

```
./ct-ng xtensa-esp32-elf
```

```
./ct-ng build
```

```
chmod -R u+w builds/xtensa-esp32-elf
```

Note: Toolchain will be built in ~/esp/crosstool-NG/builds/xtensa-esp32-elf

Method 2: Downloading prebuilt toolchain from ESP website

Download prebuilt toolchain from below link

```
https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_2_0-esp32-2019r1-linux-amd64.tar.gz
```

Download this file, then extract it in ~/esp directory

```
mkdir -p ~/esp
```

```
cd ~/esp
```

```
tar -xzf ~/Downloads/xtensa-esp32-elf-gcc8_2_0-esp32-2019r1-linux-amd64.tar.gz
```

Now the toolchain will be extracted into ~/esp/xtensa-esp32-elf/ directory

Setting xtensa toolchain in PATH environment variable.

To make xtensa-esp32-elf available for all terminal sessions. PATH variable need to be modified in .profile file

```
vim ~/.profile
```

In the file, at the end add below line

```
export PATH ="$HOME/esp/xtensa-esp32-elf/bin:$PATH"
```

Verify PATH variable update

Log off and log in back to make the .profile changes effective. Run the following command to verify if PATH is correctly set.

Now check PATH environment variable is updated or not. By typing below command

```
printenv PATH
```

Now you should be able to see path to /home/your-user-name/xtensa-esp32-elf/bin folder

Step 3: Setup ESP_IDF

Open Terminal, and run the following commands

```
cd ~/esp
```

```
git clone --recursive https://github.com/espressif/esp-idf.git
```

```
cd esp-idf
```

```
git submodule update --init
```

Set up IDF_PATH by adding the following line to ~/.profile file

```
vim ~/.profile
```

In the file, at the end add below line

```
export IDF_PATH=~/esp/esp-idf
```

Log off and log in back to make this change effective.

Install python packages

The python packages required by ESP-IDF are located in IDF_PATH/requirements.txt. To install them run following command First check python version

```
python --version
```

Note: !!! Depending on python version run below command replacing 'python-version-you-have'

```
python-version-you-have -m pip install --user -r
$IDF_PATH/requirements.txt
```

Test to check Xtensa toolchain and ESP-IDF are setup as intended and to configure new ESP32 device

Copy get-started/hello_world to the ~/esp directory

```
cd ~/esp
```

```
cp -r $IDF_PATH/examples/get-started/hello_world .
```

Connect Your Device and check which serial port ESP32 uses

Plug ESP32 and run below command

```
dmesg | grep tty
```

This should show output 'some-serial-port-name' attached in last line. Example in my case log shows: ttyUSB0 attached

Now, Unplug ESP32 and run below command

```
dmesg | grep tty
```

This should show output 'some-serial-port-name' detached in last line. Example in my case log shows: ttyUSB0 detached

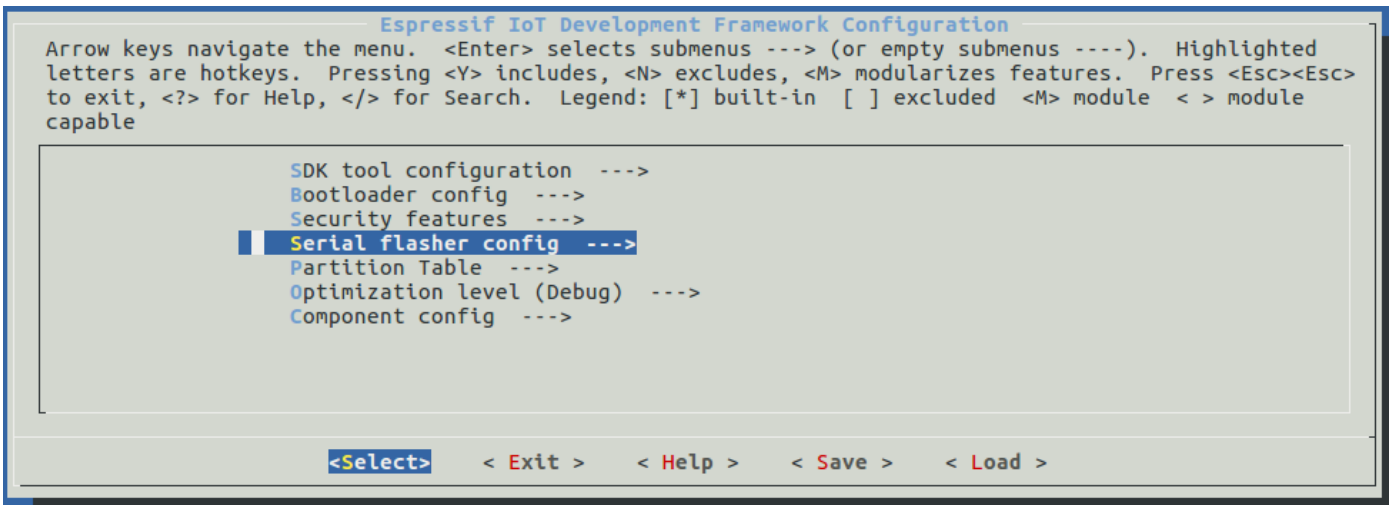
Then again plug-in the ESP32 module.

Run menuconfig to configure ESP32

Go to copied hello_world folder and run menuconfig

```
cd ~/esp/hello_world
```

```
make menuconfig
```



In the menu, navigate to

```
Serial flasher config ---> Default serial port
```

Update with serial-port name identified earlier using dmesg command.

Confirm selection by pressing enter, save configuration by selecting

```
< Save >
```

and then exit menuconfig by selecting

```
< Exit >
```

Build source and flash ESP32 with binaries

To build source in the project. Run below command

```
make all
```

Then to flash the code into ESP32. Run below command

```
make flash
```

USB error encountered during make flash

- Errors due to permission constraint as shown below:

```
  File "/home/biba/.local/lib/python2.7/site-packages/serial/serialposix.py", line 268, in open
    raise SerialException(msg.errno, "could not open port {}: {}".format(self._port, msg))
serial.serialutil.SerialException: [Errno 13] could not open port /dev/ttyUSB0: [Errno 13] Permission denied: '/dev/ttyUSB0'
/home/biba/BIBA/esp/ComponentRepo/esp-idf/components/esptool_py/Makefile.projbuild:83: recipe for target 'flash' failed
make: *** [flash] Error 1
biba@biba:~/BIBA/esp/hello_world$
```

Solution: Run the below command to give permission to the port (its making the port readable, writable and executable by everyone) where esp is attached.

```
chmod 777 -R /dev/ttyUSB0
```

Alternatively, the below mention command can also be used to read and write the serial port without root:

```
sudo usermod -a -G dialout $USER
```

Finally run the command make flash.

Step 4: Setup Eclipse for ESP32 build environment

If you don't have Eclipse IDE installed. Then Download and Install Eclipse IDE 2019-06 from below link

```
https://www.eclipse.org/downloads/packages/release/neon/3rc3/eclipse-ide-cc-developers
```

After installation follow below steps

Step 1: Launch Eclipse

Step 2: Create new folder and setup it as new workspace

Go to terminal and create new folder

```
mkdir ~/esp32_workspace
```

In Eclipse choose 'esp32_workspace' as new workspace.

Step 3: Import Project

Go to File ---> Import....

In the dialog that pops up, choose “C/C++” -> “Existing Code as Makefile Project” and click Next.

On the next page, enter “Existing Code Location” to be the directory of your IDF project. Don’t specify the path to the ESP-IDF directory itself (that comes later). The directory you specify should contain a file named “Makefile” (the project Makefile).

On the same page, under “Toolchain for Indexer Settings” choose “Cross GCC”. Then click Finish.

Step 4: Project properties

Left Click on top most folder in Project explorer and select properties

Click on the “Environment” properties under “C/C++ Build”. Click “Add” and enter name BATCH_BUILD and value 1.

Click “Add…” again, and enter name IDF_PATH. The value should be the full path where ESP-IDF is installed.

Edit the PATH environment variable. Keep the current value, and append the path to the Xtensa toolchain installed as part of IDF setup, if this is not already listed on the PATH. A typical path to the toolchain looks like /home/user-name/esp/xtensa-esp32-elf/bin. Note that you need to add a colon : before the appended path. Windows users will need to prepend

Navigate to “C/C++ General” -> “Preprocessor Include Paths” property page

Click the “Providers” tab

In the list of providers, click “CDT Cross GCC Built-in Compiler Settings”.

Change “Command to get compiler specs”

```
xtensa-esp32-elf-gcc ${FLAGS} -std=c++11 -E -P -v -dD "${INPUTS}" .
```

In the list of providers, click “CDT GCC Build Output Parser” and

change the “Compiler command pattern” to

```
xtensa-esp32-elf-(gcc|g\+\+|c\+\+|cc|cpp|clang)
```

Navigate to “C/C++ General” -> “Indexer” property page

Check “Enable project specific settings” to enable the rest of the settings on this page. Uncheck “Allow heuristic resolution of includes”. When this option is enabled Eclipse sometimes fails to find correct header directories.

Navigate to “C/C++ Build” -> “Behavior” property page:

Check “Enable parallel build” to enable multiple build jobs in parallel.

Step 5: Building in Eclipse

Go to Project -> Clean then choosing Project -> Build All

Step 6: Adding Targets to flash binaries using Eclipse UI

You can integrate the “make flash” target into your Eclipse project to flash using esptool.py from the Eclipse UI:

Right-click your project in Project Explorer (important to make sure you select the project, not a directory in the project, or Eclipse may find the wrong Makefile.)

Select Build Targets -> Create… from the context menu.

Type “flash” as the target name. Leave the other options as their defaults.

Step 5: Setup Debugger

Set up OpenOCD

Download from link below openOCD binary download (https://github.com/espressif/openocd-esp32/releases) additionally you can also find it [here](

Extract the downloaded file in ~/esp/ directory:

```
cd ~/esp  tar -xzf ~/Downloads/openocd-esp32-linux64-<version>.tar.gz
```

Dependency

Following are needed before installing OpenOCD binary

- make

```
sudo apt-get install make
```

- libtool

```
sudo apt-get install libtool
```

- pkg-config >= 0.23 (or compatible)

```
sudo apt-get install pkg-config
```

- autoconf >= 2.64

```
sudo apt-get install autoconf
```

- automake >= 1.14

```
sudo apt-get install automake
```

- texinfo

```
sudo apt-get install texinfo-doc-nonfree
```

Running OpenOCD for Jlink segger

Open terminal, go to directory where OpenOCD is installed and start it up

```
cd ~/esp/openocd-esp32
```

```
bin/openocd -s share/openocd/scripts -f interface/jlink.cfg -f
board/esp-wroom-32.cfg
```

Setting up Eclipse debugger

In Eclipse go to Run > Debug Configuration. A new window will open.

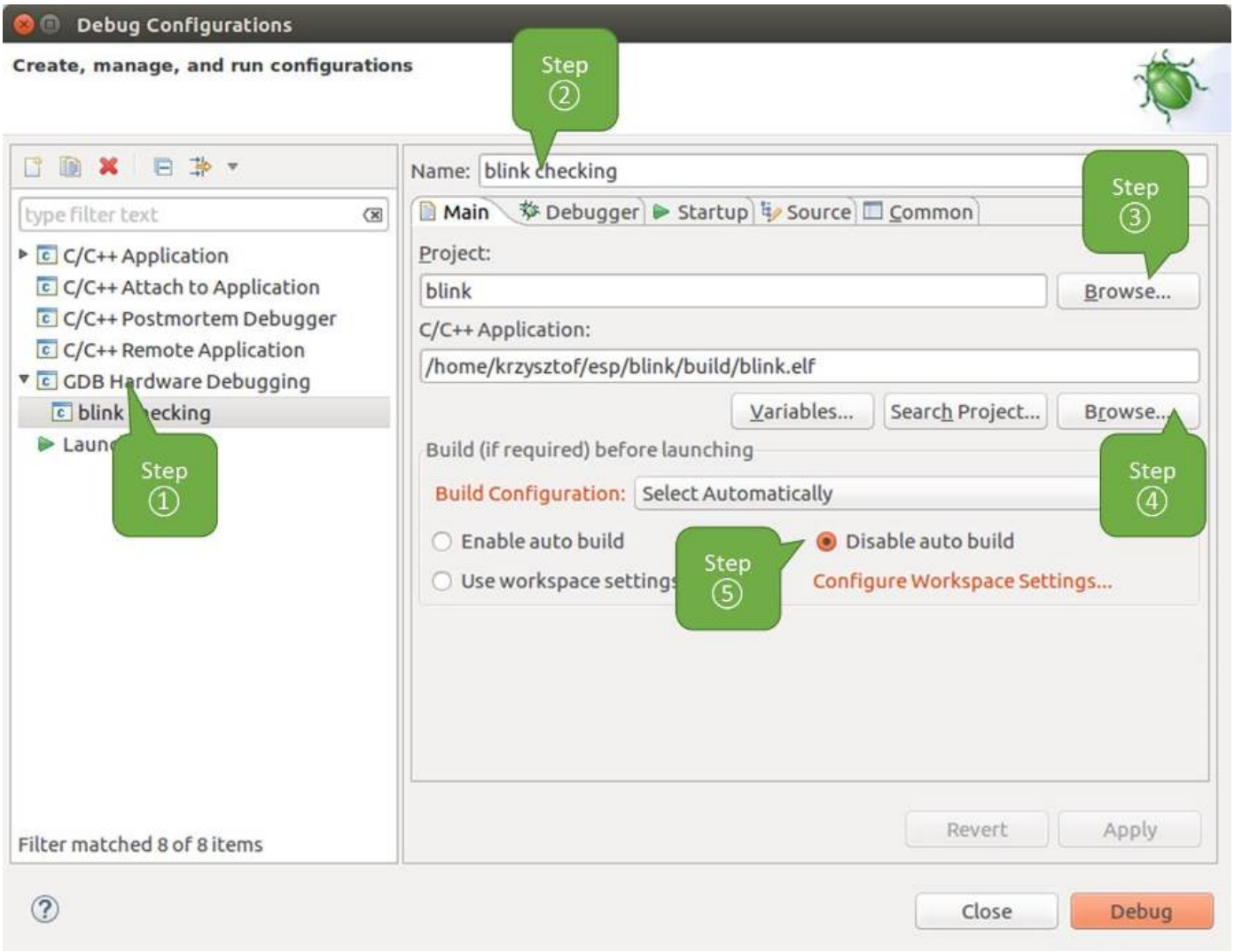
In the window's left pane double click "GDB Hardware Debugging" (or select "GDB Hardware Debugging" and press the "New" button) to create a new configuration.

In a form that will show up on the right, enter the "Name:" of this configuration, e.g. "your_project".

On the “Main” tab below, under “Project:”, press “Browse” button and select the “your_project” project.

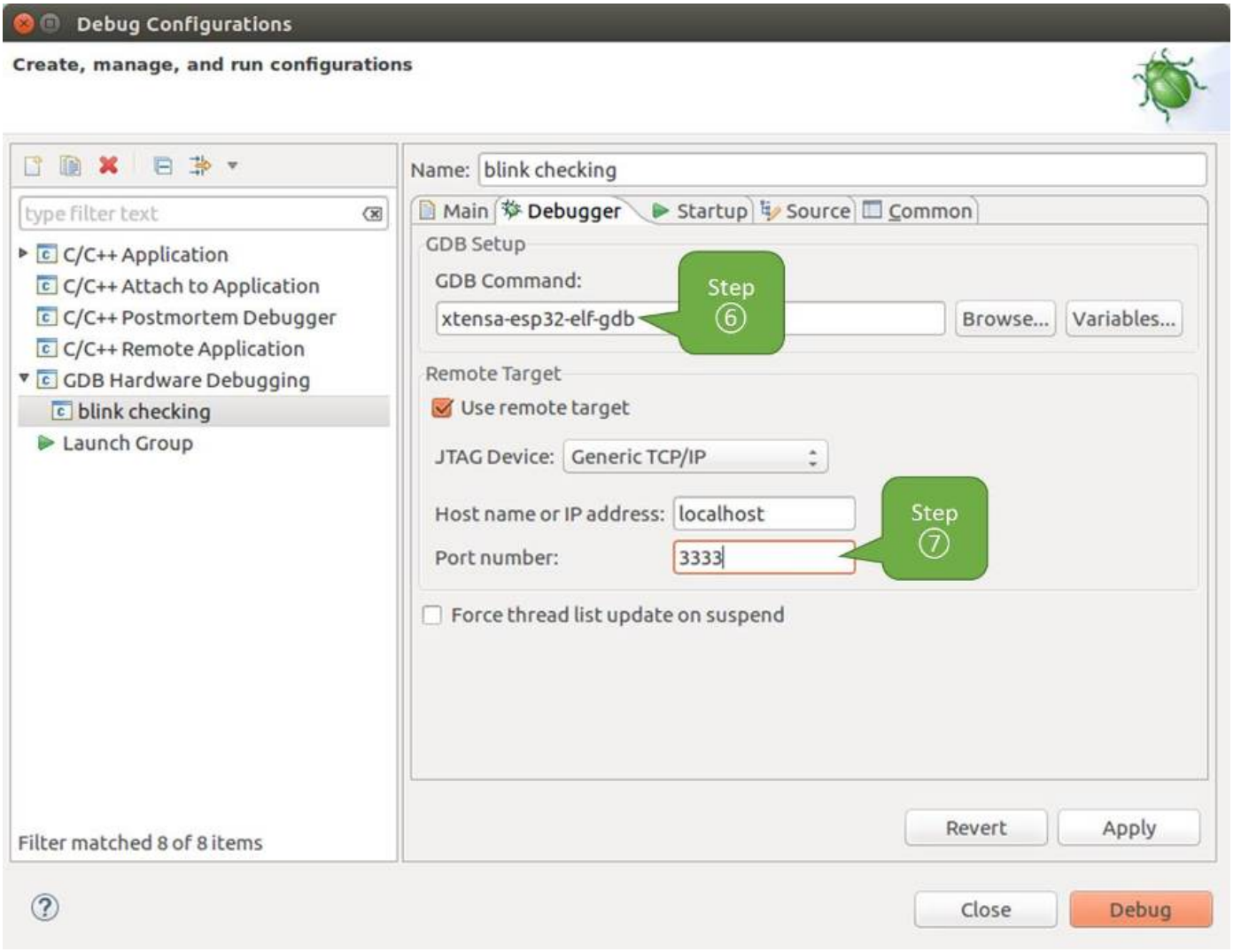
In next line “C/C++ Application:” press “Browse” button and select “your_project.elf” file. If “your_project.elf” is not there, then likely this project has not been build yet.

Finally, under “Build (if required) before launching” click “Disable auto build”



Click “Debugger” tab. In field “GDB Command” enter xtensa-esp32-elf-gdb to invoke debugger.

Change default configuration of “Remote host” by entering 3333 under the “Port number”.



The last tab to that requires changing of default configuration is “Startup”.

Under “Initialization Commands” uncheck “Reset and Delay (seconds)” and “Halt”“.

Then, in entry field below, enter the following lines:

mon reset halt flushregs set remote hardware-watchpoint-limit 2

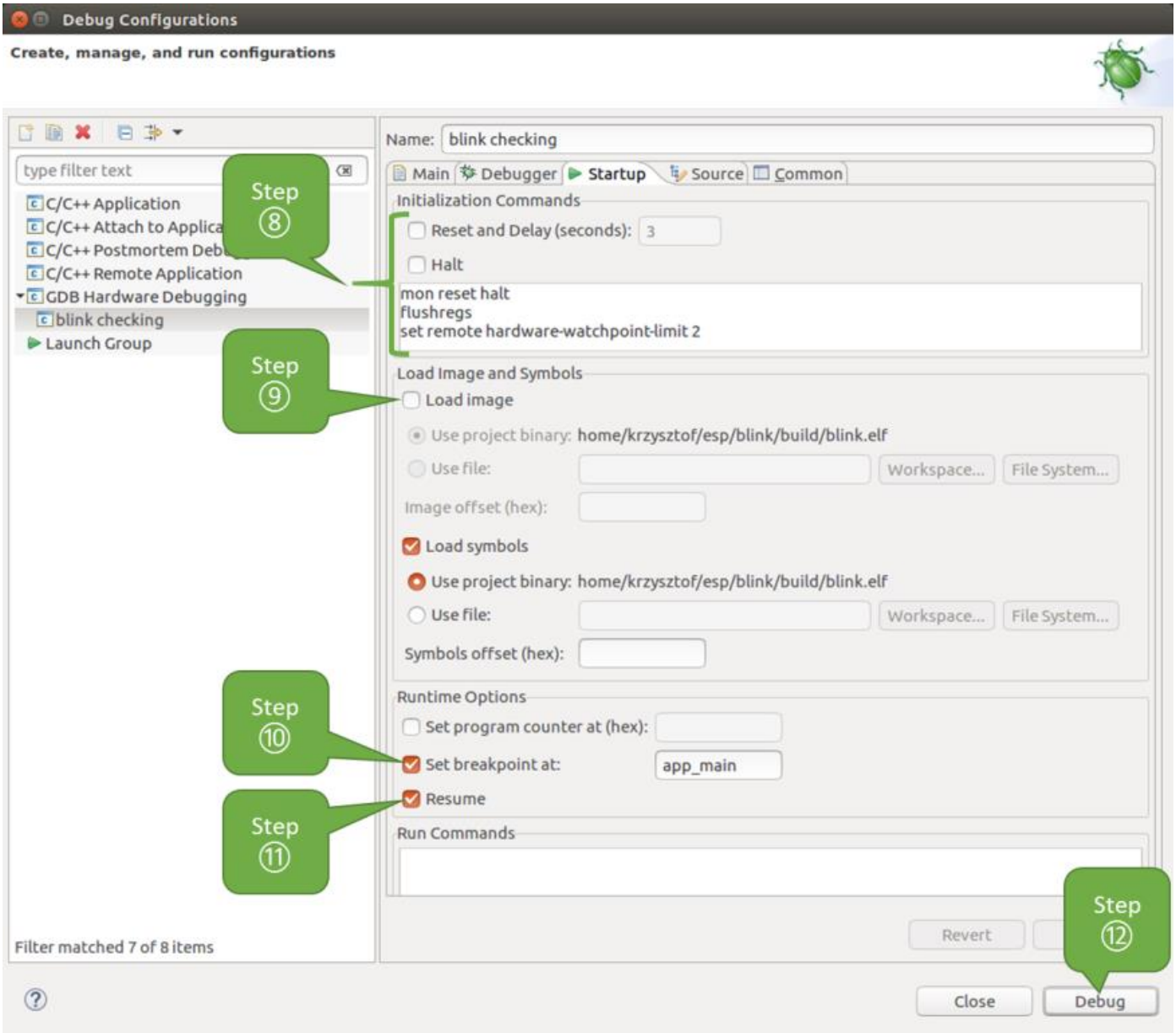
Under “Load Image and Symbols” uncheck “Load image” option.

Further down on the same tab, establish an initial breakpoint to halt CPUs after they are reset by debugger.

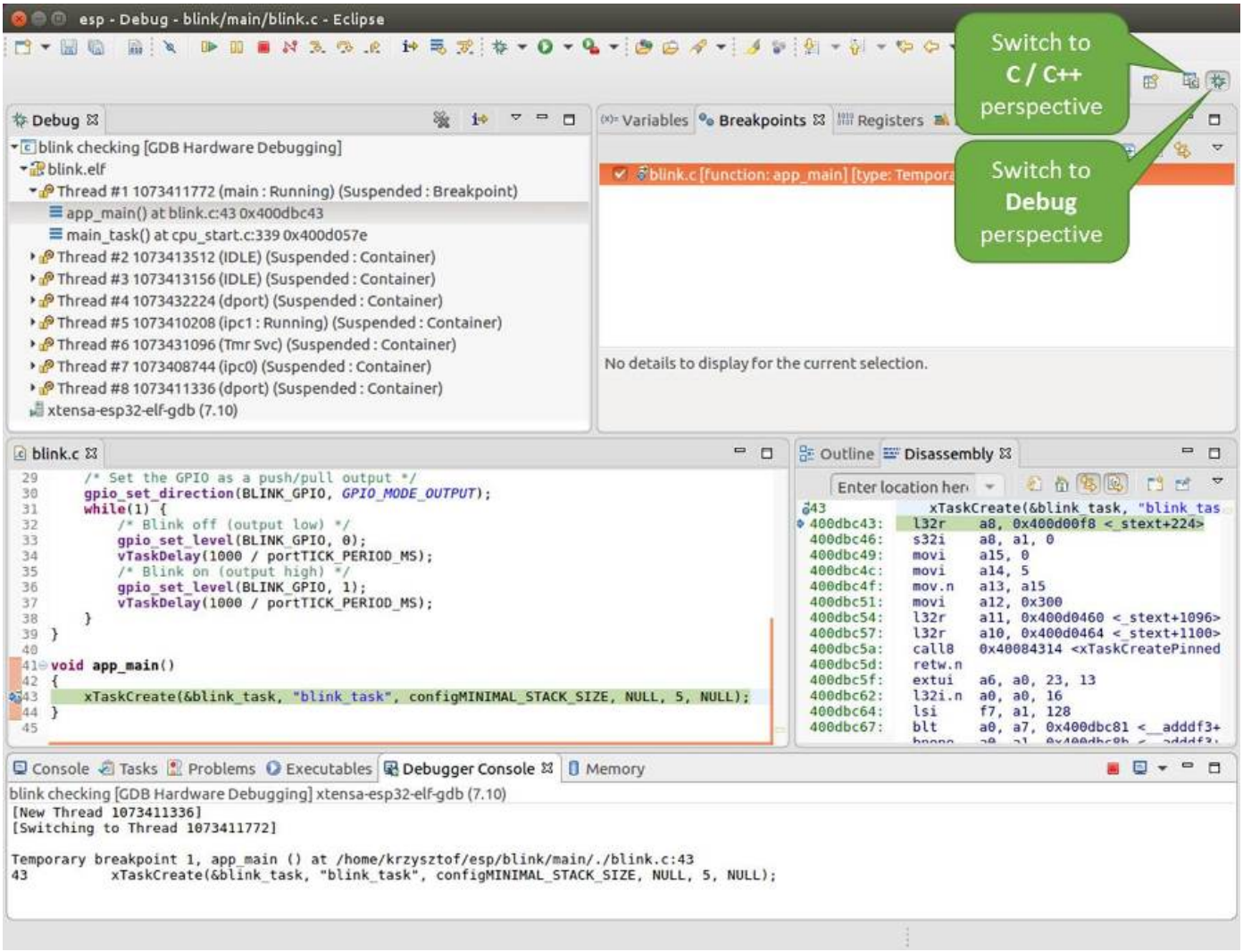
Checkout "Set break point" option and enter app_main in provided field.

Checkout “Resume” option.

This will make the program to resume after mon reset halt is invoked per point 8. The program will then stop at breakpoint inserted at app_main.



Once all 1 - 12 configuration steps are satisfied, the new Eclipse perspective called “Debug” will open as shown on example picture below.



This ends setup procedure for ESP32 for Eclipse.

Have Fun, Happy Coding !!!!!!!