1 Raspbian cross-development notes

This document is a set of notes for setting up a PC Linux system and Eclipse for cross-developing programs for Raspberry Pi running Raspbian or RaspiOS operating system.

The notes include setting up both systems from the installation media.

2 Preliminary setup

This part covers the initial setup of tools needed outside of Eclipse.

2.1 General

2.1.1 Development tools

The test setup was made with a Kubuntu 20.04 LTS Linux on a PC, and a fresh Raspbian / Raspberry Pi OS version 10 (Buster). The description begins from the initial installation from the distribution media, but it is not necessary to destroy a good working installation to follow the recipes.

The recipes assume a Debian-compatible system on the PC. The Debian installer just ignores the requests to install packages that are already in the system.

2.1.2 Special hardware required

To write the SD card for Raspberry Pi mass memory, a connection for the card to the host PC is needed, either a card slot on the PC or a suitable adapter.

2.1.3 Network connections needed

For running the remote debugging and installing software, it is necessary to have a network connection on both the PC and the Raspberry. Both have a need to reach each other and to have a connection to the Internet. There is a need to know the IP address or DNS name of the Raspberry. The details how to set up the networking depend on the available network.

2.2 Host Linux setup

2.2.1 Set up from distribution image

Get the installation image from a distribution site, put it into suitable installation medium, depending on the PC properties. The test installation used the **kubuntu-20.04.3-desktop-amd64.iso** DVD image. Please note that Eclipse needs a 64 bit system to run.

The installation asks for an user name and password. Both are needed later.

2.2.2 Set up keyboard, time zone and locale

Depending on your keyboard hardware and location, the keyboard, time zone and locale probably need to be set up properly.

2.2.3 Update

The installation images are not updated frequently, so an update is probably needed. Log in and open a text terminal (Konsole in KDE).

```
sudo apt update
```

Respond with your password to get superuser rights.

```
sudo apt full-upgrade -y
```

Restart

```
sudo shutdown -r now
```

2.2.4 Clean up

Log in again and open the text terminal for clean-up

```
sudo apt autoremove -y
sudo apt autoclean
```

The initial PC setup is completed.

2.3 Raspberry Pi setup

2.3.1 Set up from distribution image

Get the Raspberry Pi OS from https://www.raspberrypi.com/software/operating-systems/ and follow the instructions there. Please just copy the image onto the SD card, do not use the NOOBS installer. NOOBS occasionally sets irreversible write-protection on the card, spoiling it.

2.3.2 Ensure SSH access

After writing the SD card, its BOOT partition can be mounted on the host PC. The Kubuntu file manager asks if the partition should be accessed. Accept the boot partiton. The easiest way to add SSH access is to create a zero length file with the name **ssh** into the BOOT partition. Kubuntu mounts the partition to

```
/media/yourusername/boot/
```

Substitute **yourusername** with your username on the host Linux.

Open text terminal and create the file

```
touch /media/yourusername/boot/ssh
```

Use the file manager to unmount the SD card. Move the card to the Raspberry.

2.3.3 Update

Start up Raspberry.

If you have keyboard and display on the Pi, use them, otherwise use the host PC SSH to log in as user **pi** and password **Raspberry**.

Set up the keyboard layout, time zone and locale if needed.

Do the same update operations as with the host PC

```
sudo apt update
sudo apt full-upgrade -y
sudo shutdown -r now
```

2.3.4 Clean up

After restart, log in again and clean up

```
sudo apt autoremove -y
sudo apt autoclean
```

2.3.5 Install gdbserver

Install GDB server on Raspberry

```
sudo apt install gdbserver
```

2.3.6 Create an user to match the username on the host

To simplify the remote handling, create an user with the same name as on the host PC and get the right to change to superuser if needed

```
sudo adduser yourusername
sudo adduser yourusername sudo
```

Replace **yourusername** with your user name on the host PC.

The initial setup of Raspberry is completed.

2.4 Host Linux setup, continued

2.4.1 Install required packages

Open text console on host Linux and install

```
sudo apt install build-essential git gdb-multiarch openjdk-11-jdk
```

2.4.2 Install Raspbian cross-tools

Change to superuser, set up installation directory for Raspbian tools and install

```
sudo bash
cd /opt
mkdir raspbian
cd raspbian
git clone https://github.com/raspberrypi/tools
```

The cloning takes some time. After it is done, continue

```
ln -s tools/arm-bcm2708/arm-linux-gnueabihf toolbase
ln -s toolbase/bin  bin
ln -s toolbase/arm-linux-gnueabihf/sysroot sysroot
exit
```

The symbolic links are set to simplify the setup of toolset paths later.

Now the tools are in **/opt/raspbian/***.

2.4.3 Cross-debugger

For proper debugging, a special version of the GDB debugger is needed. It has to run on PC hardware, but its target code has to be ARM code of Raspberry. There are two options:

- gdb-multiarch
- arm-linux-gnueabihf-gdb

The multiple-architecture GDB is in recent Linux distributions. The arm-linux-gnueabihf-gdb is in the Raspbian toolkit. Its access path is the same as for the compiling tools.

2.4.4 Generate SSH keys

To simplify SSH access to Raspberry, create user identification SSH keys for login without a password.

The keys may be a bit tricky. By default, OpenSSH creates private keys in a format which cannot be used by the encryption libraries used by Eclipse. OpenSSH can also use the PEM keys which are good for Eclipse.

In a system with SSH already set up, there may be a key pair already. Check if the file **~/.ssh/id_rsa** exists. If it is there and there is no file **~/.ssh/id_rsa.pem** and the first text line of the key file (**id_rsa**) is

```
-----BEGIN OPENSSH PRIVATE KEY-----
```

it needs to be converted

```
cd ~/.ssh
cp id_rsa id_rsa.pem
ssh-keygen -p -m PEM -P “” -N “” -f ~/.ssh/id_rsa.pem
```

In a fresh installation, there are no SSH keys yet, generate them

```
ssh-keygen -t rsa -b 4096 -m PEM -N “” -f ~/.ssh/id_rsa
```

The first line of the private key in the required PEM format is

```
-----BEGIN RSA PRIVATE KEY-----
```

2.4.5 Install public SSH key to Raspberry

Check that Raspberry is running and accessible from net with SSH, install the key

```
ssh-copy-id yourusername@raspberryip
```

Where **yourusername** is the username set up previously and **raspberryip** is the IP address or DNS name of Raspberry.

Respond with your password on Raspberry. Check that you can login without password

```
ssh raspberryip
```

If the login is successful, log out

```
exit
```

The setup is completed.

2.5 Cross-compilation test

2.5.1 Create test source, compile it

Create a working directory, change to it and create **hello.c**

```
mkdir -p ~/tmpdir
cd ~/tmpdir
```

Use your favourite editor to create **hello.c**

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
      {
      printf("Hello World!\n");
      return EXIT_SUCCESS;
      }
```

Compile **hello.c** to debug binary **hello**

```
export PATH=/opt/raspbian/bin:$PATH
export SYSROOT=/opt/raspbian/sysroot
arm-linux-gnueabihf-gcc -g -Og --sysroot=$SYSROOT -o hello hello.c
```

2.5.2 Start gdbserver on Raspberry

On Raspberry console, ensure that **~/Downloads** exists, change to it

```
mkdir -p ~/Downloads
cd ~/Downloads
gdbserver --multi :2345
```

2.5.3 Run GDB on host, using Raspberry remote target

Use GDB for multiple target architectures, tell it where is the system root used for developing the target code, set up remote target, copy the code to Raspberry, and run it.

On host console

```
gdb-multiarch
set sysroot /opt/raspbian/sysroot
set solib-search-path /opt/raspbian/sysroot/lib
file hello
target extended-remote raspberryip:2345
remote put hello hello
set remote exec-file hello
run
monitor exit
disconnect
quit
```

The Raspberry console shows the Hello World message if everything is working.

2.5.4 Optional cleanup on Raspberry tools

If it is desired to save host disk space, much of the cloned GIT contents may be deleted

```
sudo bash
cd /opt/raspbian/tools
rm -rf .git*
cd arm-bcm2708
rm -rf arm-bcm*
rm -rf gcc-linaro*
exit
```

2.6 Eclipse setup

This part covers the setup of Eclipse.

2.6.1 Download, unzip installer and start it

Get the Eclipse installer from <https://www.eclipse.org/downloads/packages/installer>.

The correct one for PC Linux is **Linux x86_64**. Select the ‘save’ option for the downloaded file handling. The default browser, Firefox, saves it into **~/Downloads**.

```
cd
tar -xzvf Downloads/eclipse-inst-jre-linux64.tar.gz
eclipse-installer/eclipse-inst&
```

2.6.2 Update installer if needed

There is an icon of three horizontal stripes at the top right part of the installer window. If there is an exclamation point (!) on the icon, click the icon and click ‘Update’ on the next screen. The installer restarts when the update is complete.

2.6.3 Install Eclipse IDE for C/C++ Developers

On the installer screen, select Eclipse IDE for C/C++ Developers. Accept other defaults, but the desktop shortcut may be left out, on certain desktops, it does not function. Click Install. After installation, click LAUNCH. When Eclipse asks for workspace, just click Launch.

The installed Eclipse is in **~/eclipse/cpp-latest-released/eclipse/eclipse.**

2.6.4 Update installation if needed

On the Eclipse menu, click Help → Check for Updates. Accept all updates, if any.

2.6.5 Install Remote System Explorer modules

Click Help → Install New Software.

For Work with, select the current Eclipse release (now: 2022-03).

Open Mobile and Device Development drop-down using the little triangle on the left.

Select from the checkboxes Remote System Explorer End-User Runtime and Remote System Explorer End-User Actions.

Accept licences and finish the installation.

2.7 Cross-compilation test using Eclipse

2.7.1 Create cross-hello using CDT and Raspberry toolset

Open C/C++ perspective on Eclipse.

Click File → New C/C++ Project, C Managed Build → Next

Project name: hello, Executable, Hello World ANSI C Project, Cross GCC → Next

Basic Settings as required → Next

Leave Debug and Release selected → Next

Cross compiler prefix: **arm-linux-gnueabihf-**
Cross compiler path: **/opt/raspbian/bin**

Click Finish

2.7.2 Compile and link Debug target

Select the project in Project Explorer, open the project tree with angle at left

Project → Build Configurations → Set Active → Debug

Project → Build Project.

The project should build without complaints and create a run file in Binaries branch on project tree, with a bug icon.

2.7.3 Run remote debug on created target

Select the binary file icon, from menu select Run → Debug Configurations

Select C/C++ Remote Application, click New Configuration button (leftmost on top row)

On Main tab, check Name, Project and C/C++ Application boxes for proper contents

In Build box, select Disable auto build

In Connection row, select New … and choose SSH from the drop-down list, click OK

Connection name: Raspberry remote

Host information, Host: IP of Raspberry, User: your username on Raspberry,
Public key based authentication

Click Network Connections, SSH2, verify your private PEM key in Private keys,
click Apply and Close

Click Finish in New Connection dialog box

On Main tab, Remote Absolute File Path for C/C++ Application, click Browse …

Accept SSH remote host identification, if asked, fill in the password on Raspberry to the pop-up

Select Downloads from the selection dialog box, click OK

On Debugger tab, Main sub-tab, GDB debugger: gdb-multiarch

Click Debug button on bottom right corner

If requested, accept change to debug perspective

Select debugging options, e.g. breakpoints if needed

Select Run → Resume, or click the green run triangle button

After debugging, return to C/C++ perspective

2.8 Cross-compilation with wiringPi

The wiringPi libraries and header files are included with a RaspiOS or Raspbian installation.

There is the **gpio** utility included in the installation. It can be used to verify that the installation is there, by logging in on Raspberry and typing

```
gpio -v
```

If the Raspberry responds with version information, the installation is present.

The libraries are shared (dynamic) libraries, and copies of them are needed on the development computer to enable proper linking. The upload procedure below should suit for other shared library installations also.

2.8.1 Raspberry Pi setup for wiringPi

2.8.1.1 Give GPIO access to user on Raspberry

Add your user to the **gpio** group to get access to the gpio interface registers

```
sudo adduser myusername gpio
```

substituting your user name instead of **myusername**. Give your password to sudo if requested. Log out and in again to activate the increased credentials.

2.8.1.2 Upload wiringPi files from Raspberry

On Raspberry console, create a tar/gzip archive of the wiringPi files

```
cd /
tar -cvzf ~/wiring.tgz usr/include/wiring* usr/lib/libwiring*
```

2.8.2 Development computer setup for wiringPi

On the development computer console, copy the archive up

```
scp raspberryip:wiring.tgz ~
```

After the copy, there should be the file **wiring.tgz** with a length of about 46000 bytes.

2.8.2.1 Install uploaded files to development computer

On the development computer console, change to the root directory of the Raspberry file tree copy and unpack the archive

```
cd /opt/raspbian/sysroot
sudo tar -xzvf ~/wiring.tgz
```

I have used the system root location of my installation **/opt/raspbian/sysroot** here.

2.8.3 Cross-compilation test with wiringPi

2.8.3.1 Create test source, compile it

Create a working directory, change to it and create **blink.c**

```
mkdir -p ~/tmpdir
cd ~/tmpdir
```

Use your favourite editor to create **blink.c:**

```
#include <stdlib.h>
#include <wiringPi.h>
int main(void)
      {
      wiringPiSetup();
      pinMode(0, OUTPUT);

      for (;;)
            {
            digitalWrite(0, HIGH);
            delay(500);
            digitalWrite(0, LOW);
            delay(500);
            }

       return EXIT_SUCCESS;
       }
```

Compile **blink.c** to debug binary **blink:**

```
export PATH=/opt/raspbian/bin:$PATH
export SYSROOT=/opt/raspbian/sysroot
arm-linux-gnueabihf-gcc -g -Og --sysroot=$SYSROOT \
-o blink blink.c -lwiringPi
```

2.8.3.2 Start gdbserver on Raspberry

On Raspberry console, ensure that **~/Downloads** exists, change to it, start gdbserver

```
mkdir -p ~/Downloads
cd ~/Downloads
gdbserver --multi :2345
```

2.8.3.3 Run GDB on host, using Raspberry remote target

Use GDB for multiple target architectures, tell it where the target code system root is, set up remote target, copy the code to Raspberry, and run it.

On host console

```
gdb-multiarch
set sysroot /opt/raspbian/sysroot
set solib-search-path /opt/raspbian/sysroot/lib
file blink
target extended-remote raspberryip:2345
remote put blink blink
set remote exec-file blink
```

You can use debugging commands here, e.g.

```
run
```

Terminate the run with control-C, and exit remote debug

```
monitor exit
disconnect
quit
```

2.8.4 WiringPi test using Eclipse

2.8.4.1 Create wiringPi test

Open C/C++ perspective on Eclipse.

Click File → New C/C++ Project, C Managed Build → Next

Project name: blink, Executable, Empty Project, Cross GCC → Next

Basic Settings as required → Next

Leave Debug and Release selected → Next

Cross compiler prefix: **arm-linux-gnueabihf-**
Cross compiler path: **/opt/raspbian/bin**

Click Finish

Click File → Import → General → File System → Next

From directory: Browse to ~/tmpdir, select it

Click selection box left of **blink.c** → Finish

2.8.4.2 Compile and link Debug target

Select the project in Project Explorer, open the file tree with angle at left

Project → Properties → C/C++ Build → Settings → Tool Settings → Cross GCC Linker → Libraries

Libraries (-l): Click leftmost icon to add **wiringPi**

Library search path (-L): Click leftmost icon to add **/opt/raspbian/sysroot/lib**

Click Apply and Close

Project → Build Configurations → Set Active → Debug

Click the hammer icon or Project → Build Project.

The project should build without complaints and create a run file in Binaries branch on project tree, with a bug icon.

2.8.4.3 Run remote debug on created target

Select the binary file icon, from menu select Run → Debug Configurations

Select C/C++ Remote Application, click New Configuration button (leftmost on top row)

On Main tab, check Name, Project and C/C++ Application boxes for proper contents

In Build box, select Disable auto build

In Connection row, click the drop-down button at the right end of the box.

Select **Raspberry remote** if it exists, otherwise create it:

Select New … and choose SSH from the drop-down list, click OK

Connection name: Raspberry remote

Host information, Host: IP of Raspberry, User: your username on Raspberry,
Public key based authentication

Click Network Connections, SSH2, verify your private PEM key in Private keys,
click Apply and Close

Click Finish in New Connection dialog box

On Main tab, Remote Absolute File Path for C/C++ Application, click Browse …

Accept SSH remote host identification, if asked, fill in the password on Raspberry to the pop-up

Select Downloads from the selection dialog box, click OK

On Debugger tab, Main sub-tab, GDB debugger: gdb-multiarch

Click Debug button on bottom right corner

If requested, accept change to debug perspective

Select debugging options, e.g. breakpoints if needed

Select Run → Resume, or click the green run triangle button

After debugging, return to C/C++ perspective