Cross-Compiling GTK Application for BeagleBone Black

by Song Tung Nguyen
Last Updated: April 10th, 2022

Note:

- The following guide is based on the same guide for the Raspberry Pi with modification and additional troubleshooting steps to make it work for the BeagleBone Black.
- This guide should work for Qt as well, but it has not yet been tested.

This document guides the user through:
1. Setting up the host for cross-compilation of a GUI application for the BeagleBone Black.
2. How to write and cross-compile a GTK application written in C/C++.
3. Briefly mention how to handle touches on a touch screen.

Table of Contents

Formatting

1. Commands for the host Linux's console are shown as:
```
(host)$ echo "Hello world!"
```
2. Commands for the target (BeagleBone) Linux's console are shown as:
```
(target)$ echo "Hello World!"
```
3. All commands are case-sensitive.

1. Setting up GTK library for the cross compiler

GTK is a free and open-source cross-platform toolkit for developing graphical interface application. It is especially popular in the Linux world as most Linux distributions like Ubuntu, Fedora, and Debian make use of it for its application. However, the only officially supported way for compiling a GTK application for an ARM based device is to compile it natively, which could slow down the development process. This guide will enable you to cross-compile your C/C++ GTK application straight from the host PC.

It must be noted that this solution is just a “work-around”, so if there is an official way to cross-compiling GTK in the future then you should follow it.

1.1. Getting the cross-compiling toolchain

I strongly encourage you to follow this step even if you have already had the cross-compiler for the BeagleBone from the **Quick-Start Guide**. I have tried to link the GTK library to the cross-compiler from the **Quick-Start Guide** and it does not seem to work.

1. Create a directory to store the toolchain and the library files. I would suggest putting it in **$(HOME)/BBB/tools**:
```
(host)$ mkdir ~/BBB/tools
(host)$ cd BBB/tools
```

2. Get the pre-built toolchain and extracts it (you can try newer version if you like):
```
(host)$ wget -c
https://releases.linaro.org/components/toolchain/binaries/6.3-
2017.05/arm-linux-gnueabihf/gcc-linaro-6.3.1-2017.05-x86_64_arm-
linux-gnueabihf.tar.xz
(host)$ tar xf gcc-linaro-6.3.1-2017.05-x86_64_arm-linux-
gnueabihf.tar.xz
```

3. Update the PATH variable to allow access to the toolchain directly from the command line. Open the ***~/.bashrc*** file:
```
(host)$ nano ~/.bashrc
```

4. Put the following command at the bottom of the ***~/.bashrc*** file:
```
(host)$ export PATH=$PATH:$HOME/BBB/tools/gcc-linaro-6.3.1-2017.05-
x86_64_arm-linux-gnueabihf/bin
```
5. You can either use the following command or open a new terminal session to apply the change:
```
(host)$ source ~/.bashrc
```

6. Test to see if the change has taken effect. If the changes were applied successfully, you will see the same output as below:
```
(host)$ arm-linux-gnueabihf-gcc --version
```

```
arm-linux-gnueabihf-gcc (Linaro GCC 6.3-2017.05) 6.3.1 20170404
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There
is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

1.2. Getting the GTK library files for the cross-compiler

1. Install GTK for the BeagleBone (we will be using GTK-3.0 for this guide):
   ```
   (target)$ sudo apt-get install libgtk-3-dev
   ```

2. Copy ***/usr/*** and ***/lib*** from the target into **$HOME/BBB/**. You can either copy it to the NFS mounted directory and copy it back to **$HOME/BBB** or you can scp straight into the folder. I will be using scp for this part as we will be able to avoid copying large directories twice:
   NOTE: Copying these directories may take up to 1 hour!
   ```
   (host)$ scp -r debian@192.168.7.2:/lib ~/BBB/
   (host)$ scp -r debian@192.168.7.2:/usr ~/BBB/
   ```

3. On the host, create 2 symlinks to the directories that we got from the previous step:
   ```
   (host)$ sudo ln -s  $HOME/BBB/usr/lib/arm-linux-gnueabihf/
   /usr/lib/arm-linux-gnueabihf

   (host)$ sudo ln -s  $HOME/BBB/lib/arm-linux-gnueabihf/ /lib/arm-
   linux-gnueabihf
   ```

4. Verify that the symlinks has been created. If the symlinks have been created, it should give an output like the following:
   ```
   (host)$ ls -ld /lib/arm-linux-gnueabihf
   drwxr-xr-x 2 root root 4096 Apr  7 12:56 /lib/arm-linux-gnueabihf

   (host)$ ls -ld /usr/lib/arm-linux-gnueabihf
   drwxr-xr-x 2 root root 4096 Apr  7 12:56 /usr/lib/arm-linux-
   gnueabihf
   ```

5. Troubleshooting:
   - If you have already had the compiler from the **Quick-Start Guide**, it is likely that you will not be able to create a symlink for the libraries that you got from the BeagleBone. If you encounter the following error after entering the commands from step 4 into the terminal:

```
ln: failed to create symbolic link '/usr/lib/arm-linux-gnueabihf/arm-linux-gnueabihf': File exists
```

Then you will have to unlink the existing symlink first. You can start by entering this command:

```
(host)$ unlink /usr/lib/arm-linux-gnueabihf/arm-linux-gnueabihf
```

Proceed to do the same for the other symlink if an error occurs by simply using the **unlink** command and paste the file name from the error next to it. After you have unlinked both files, repeat from step 3.

2. How to compile a GTK application written in C/C++

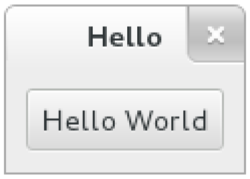
2.1. C Code

Let's get started with a sample program taken from the official GTK tutorial:

```
// demo.c
#include <gtk/gtk.h>

static void
print_hello(GtkButton *self,
            gpointer data)
{
    gtk_button_set_label(self, "Free real estate!");
}

static void
activate(GtkApplication *app,
         gpointer user_data)
{
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *button_box;

    window = gtk_application_window_new(app);
    gtk_window_set_title(GTK_WINDOW(window), "Window");
    gtk_window_set_default_size(GTK_WINDOW(window), 200, 200);

    button_box = gtk_button_box_new(GTK_ORIENTATION_HORIZONTAL);
    gtk_container_add(GTK_CONTAINER(window), button_box);

    button = gtk_button_new_with_label("Hello World");
    g_signal_connect(button, "clicked", G_CALLBACK(print_hello), NULL);
    g_signal_connect(window, "destroy", G_CALLBACK(gtk_widget_destroyed),
&window);
    gtk_container_add(GTK_CONTAINER(button_box), button);

    gtk_widget_show_all(window);
}

int main(int argc,
         char **argv)
{
    GtkApplication *app;
    int status;

    app = gtk_application_new("org.gtk.example", G_APPLICATION_FLAGS_NONE);
    g_signal_connect(app, "activate", G_CALLBACK(activate), NULL);
    status = g_application_run(G_APPLICATION(app), argc, argv);
    g_object_unref(app);

    return status;
}
```

The above program will generate a basic window with 1 button on it like shown below. If you have a touch screen, touching on the button will change the button label just like a left mouse click. With GTK, you can program a touch screen program like any normal desktop program.

I will let you figure out what the label is going to turn into :)



If you want to see how it works on the host, you can use the following command:

```
gcc `pkg-config --cflags gtk+-3.0` demo.c -o demo `pkg-config --libs gtk+-3.0`
```

2.2. Cross-Compiling

We can not use the command from the previous session to compile for our BeagleBone because:

1. **gcc** will compile the code for our host, which is likely not the same as the BeagleBone.

2. Even if we switch out **gcc** for **arm-linux-gnueabihf-gcc**, it will still fail as it will try to link the GTK library for the host instead of the target.

In order to cross-compile a GTK application for the BeagleBone, we will need to manually include all of the library files. You can use the following **Makefile** to compile the application:

```
ARM_PREFIX= arm-linux-gnueabihf-
CC   = $(ARM_PREFIX)gcc
SRC += demo.c
TARGET = demo

LIBRARY += gtk-3
LIBRARY += gdk-3
LIBRARY += atk-1.0
LIBRARY += gio-2.0
LIBRARY += pangocairo-1.0
LIBRARY += gdk_pixbuf-2.0
LIBRARY += cairo-gobject
LIBRARY += pango-1.0
LIBRARY += cairo
LIBRARY += gobject-2.0
LIBRARY += glib-2.0

LIBRARYDIR += $(HOME)/BBB/lib/arm-linux-gnueabihf
LIBRARYDIR += $(HOME)/BBB/usr/lib/arm-linux-gnueabihf
LIBRARYDIR += $(HOME)/BBB/lib
LIBRARYDIR += $(HOME)/BBB/usr/lib

XLINK_LIBDIR += $(HOME)/BBB/lib/arm-linux-gnueabihf
XLINK_LIBDIR += $(HOME)/BBB/usr/lib/arm-linux-gnueabihf

INCLUDEDIR += $(HOME)/BBB/usr/include/gtk-3.0
INCLUDEDIR += $(HOME)/BBB/usr/include/pango-1.0
INCLUDEDIR += $(HOME)/BBB/usr/include/gio-unix-2.0/
INCLUDEDIR += $(HOME)/BBB/usr/include/atk-1.0
INCLUDEDIR += $(HOME)/BBB/usr/include/cairo
INCLUDEDIR += $(HOME)/BBB/usr/include/gdk-pixbuf-2.0
INCLUDEDIR += $(HOME)/BBB/usr/include/freetype2
INCLUDEDIR += $(HOME)/BBB/usr/include/glib-2.0
INCLUDEDIR += $(HOME)/BBB/usr/lib/arm-linux-gnueabihf/glib-2.0/include
INCLUDEDIR += $(HOME)/BBB/usr/include/pixman-1
INCLUDEDIR += $(HOME)/BBB/usr/include/libpng12

OPT = -O0
DEBUG = -g
```

```
WARN= -Wall
PTHREAD= -pthread

INCDIR   = $(patsubst %,-I%,$(INCLUDEDIR))
LIBDIR   = $(patsubst %,-L%,$(LIBRARYDIR))
LIB     = $(patsubst %,-l%,$(LIBRARY))
XLINKDIR = $(patsubst %,-Xlinker -rpath-link=%,$(XLINK_LIBDIR))

all:
    $(CC) $(OPT) $(DEBUG) $(WARN) $(LIBDIR) $(PTHREAD) $(INCDIR) $(XLINKDIR)
$(LIB) $(SRC) -o $(TARGET)

clean:
    rm -rf $(TARGET)
```

Just type the following command to cross-compile it for the BeagleBone:

```
(host)$ make
```

In theory, this should compile just fine but I did receive linker errors when I first compiled the example. Please refer to the troubleshooting section to see how to fix this problem.

If you have successfully compiled it without any linker error, it will produce a binary called **demo**. You can copy that into the BeagleBone and simply launching it like any other binary:

```
(target)$ path/to/binary/demo
```

2.3. Troubleshooting

- If the compilation fails due to not being able to find the library like below:
  ```
  demo.cpp:2:21: fatal error: gtk/gtk.h: No such file or directory
  compilation terminated.
  ```

  Then chances are that you are either using the compiler from the **Quick-Start Guide** or you have not linked the libraries from the BeagleBone correctly. You have to make sure the output of step 6 from **section 1.1** matches the version of the compiler that you downloaded from **section 1.1**.
  If that does not work, then I suggest you remove the compiler from the **Quick-Start Guide**.

- If you receive a linker error like this:
  ```
  /home/user/BBB/tools/gcc-linaro-6.3.1-2017.05-x86_64_arm-linux-
  gnueabihf/bin/../lib/gcc/arm-linux-
  ```

```
gnueabihf/6.3.1/../../../../arm-linux-gnueabihf/bin/ld: cannot
find /lib/libc.so.6 inside /home/user/BBB
/home/user/BBB/tools/gcc-linaro-6.3.1-2017.05-x86_64_arm-linux-
gnueabihf/bin/../lib/gcc/arm-linux-
gnueabihf/6.3.1/../../../../arm-linux-gnueabihf/bin/ld: cannot
find /usr/lib/libc_nonshared.a inside /home/user/BBB
collect2: error: ld returned 1 exit status
make: *** [Makefile:49: all] Error 1
```

Then you will have to manually find the **shared library** file in **$HOME/BBB/usr/lib**. The file will have an extension of **.so**. Just look at the pattern that I highlighted above and find the file that caused the error.

For example, for the error above, you will have to find a file called **libc.so** and edit it using a text editor. The file will look like this:

```
/* GNU ld script
   Use the shared library, but some functions are only in
   the static library, so try that secondarily.  */
OUTPUT_FORMAT(elf32-littlearm)
GROUP ( /lib/libc.so.6 /usr/lib/libc_nonshared.a  AS_NEEDED ( ld-linux-
armhf.so.3 ) )
```

You will need to remove all of the folder path and leave just the file name behind like this:

```
/* GNU ld script
   Use the shared library, but some functions are only in
   the static library, so try that secondarily.  */
OUTPUT_FORMAT(elf32-littlearm)
GROUP ( libc.so.6 libc_nonshared.a  AS_NEEDED ( ld-linux-armhf.so.3 ) )
```

Save the file and now you should be able to compile.