Cross-compiling PJSIP for the Beaglebone Green

by:
Sansеerat Virk
Ryan Tio
Mikhail Egorov
Nhi Mai-Do

Last update: Apr 12, 2023

This guide walks the user through
1. Cross-compiling the PJSIP library
2. Testing the compilation
3. Building a sample application

This guide will focus on cross-compiling PJSIP, a very powerful open source multimedia communication library written in C. This library has many uses, but for the purposes of this guide, the focus will be implementing a sip peer to peer connection between the host and target. To make sure everything compiled successfully, a sample makefile will be provided which allows you to use the PJSIP API in your C program.

Table of Contents

0. Pre-requisites

Software requirements
This guide was written for **PJSIP version 2.13**
The host machine runs **Debian 11** inside a **Vmware workstation 17** configured using Brian's guide

Prerequisite knowledge
This guide is assuming that you have followed Dr Brian's initial guide on building cross-compiled applications.

It is also assuming you are referencing the following resources
https://docs.pjsip.org/en/latest/overview/features.html
https://docs.pjsip.org/en/latest/get-started/posix/build_instructions.html

The following is another guide using a different approach, also useful for understanding.
The second half explains setup for host to target peer to peer.
https://www.hackster.io/leograba/setting-a-voip-sip-user-agent-with-embedded-linux-827a7

Dr Brian Fraser will be referred to as Brian in this guide.

Formatting
1. Commands to be run on the host machine are prefixed with `(host)$`
2. Commands to be run on the beaglebone are prefixed with `(bbg)$`

1. Compile the PJSIP library

1.1 Obtain the PJSIP library

Option 1: Clone the latest release from their github:
https://github.com/pjsip/pjproject

Option 2: Download the tar or the zip version from their website:
https://www.pjsip.org/download.htm

At the time of writing this guide, the zip file should yield a folder named `pjproject-2.13.` Place this in your home directory.

1.2 Link the asound library

Travel inside this directory.

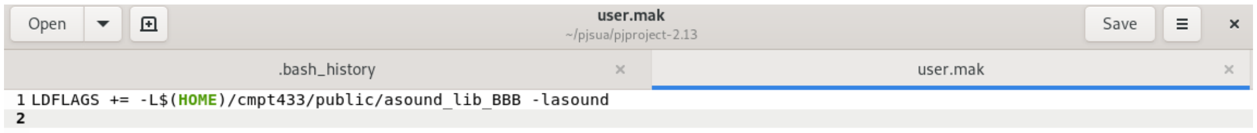
```
(host)$ cd pjproject-2.13
```

Create a `user.mak` file in the `pjproject-2.13` directory

```
(host)$ sudo nano user.mak
```

In addition to the tools installed by following Brian's audio guide, you need the path to the library `libasound.so`. This is compiled on the target and placed in the public nfs directory of the folder shared between the target and host.

Inside the `user.mak` file you created, place the following content:

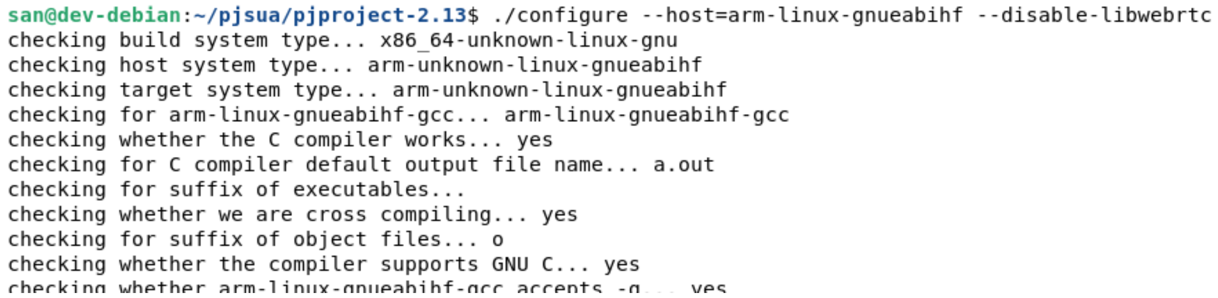


```
LDFLAGS += -L$(HOME)/cmpt433/public/asound_lib_BBB -lasound
```

1.3 Run the configuration script

Run the following command:

```
(host)$ ./configure --host=arm-linux-gnueabihf --disable-libwebrtc
```



```
san@dev-debian:~/pjsua/pjproject-2.13$ ./configure --host=arm-linux-gnueabihf --disable-libwebrtc
checking build system type... x86_64-unknown-linux-gnu
checking host system type... arm-unknown-linux-gnueabihf
checking target system type... arm-unknown-linux-gnueabihf
checking for arm-linux-gnueabihf-gcc... arm-linux-gnueabihf-gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... yes
checking for suffix of object files... o
checking whether the compiler supports GNU C... yes
checking whether arm-linux-gnueabihf-gcc accepts -g... yes
```

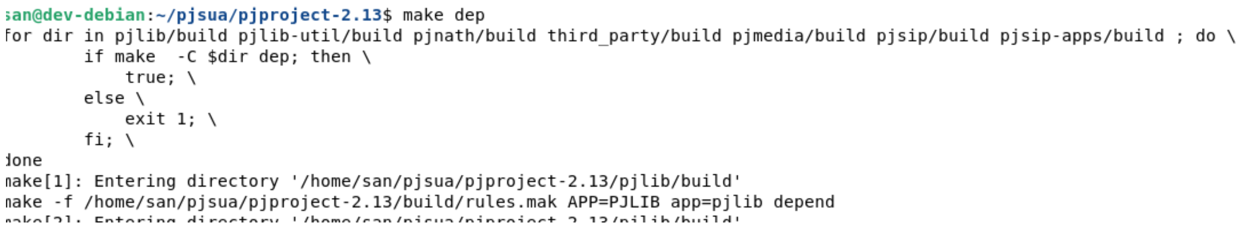
Figure 1: Expected output

The `./configure` parameters will need to be adjusted based on your needs. The --`host` parameter specifies the cross compiler being used for the Beaglebone. This is a prerequisite that needs to be met through Brian's guide. We found that we needed to disable the **`libwebrtc`** in order for cross-compilation to work.

1.4 Run the makefile

Now run:

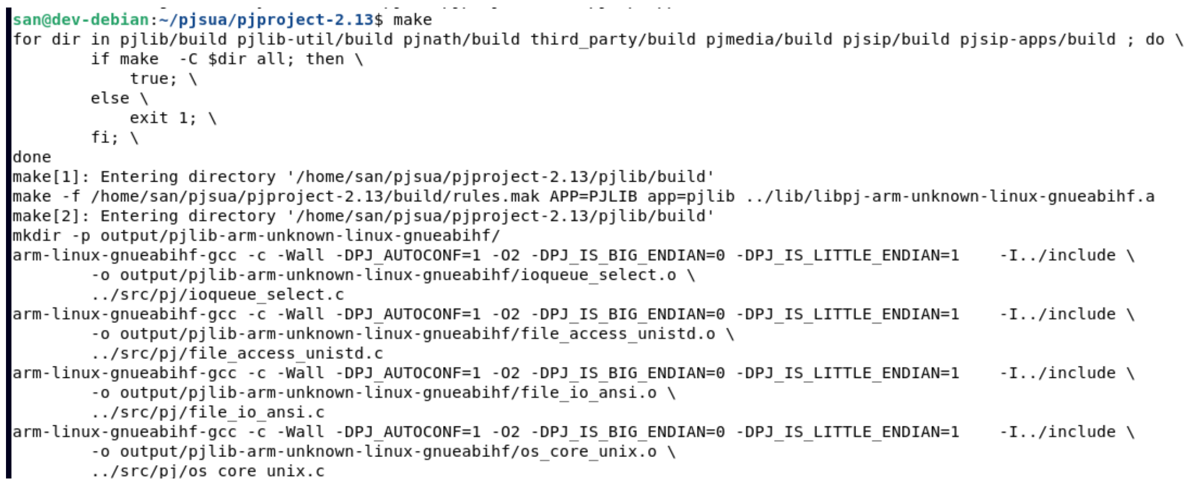
```
(host)$ make dep
```



```
san@dev-debian:~/pjsua/pjproject-2.13$ make dep
for dir in pjlib/build pjlib-util/build pjnath/build third_party/build pjmedia/build pjsip/build pjsip-apps/build ; do \
        if make  -C $dir dep; then \
            true; \
        else \
            exit 1; \
        fi; \
done
make[1]: Entering directory '/home/san/pjsua/pjproject-2.13/pjlib/build'
make -f /home/san/pjsua/pjproject-2.13/build/rules.mak APP=PJLIB app=pjlib depend
make[2]: Entering directory '/home/san/pjsua/pjproject-2.13/pjlib/build'
```

Figure 2: Expected output

```
(host)$ make
```



```
san@dev-debian:~/pjsua/pjproject-2.13$ make
for dir in pjlib/build pjlib-util/build pjnath/build third_party/build pjmedia/build pjsip/build pjsip-apps/build ; do \
        if make  -C $dir all; then \
            true; \
        else \
            exit 1; \
        fi; \
done
make[1]: Entering directory '/home/san/pjsua/pjproject-2.13/pjlib/build'
make -f /home/san/pjsua/pjproject-2.13/build/rules.mak APP=PJLIB app=pjlib ../lib/libpj-arm-unknown-linux-gnueabihf.a
make[2]: Entering directory '/home/san/pjsua/pjproject-2.13/pjlib/build'
mkdir -p output/pjlib-arm-unknown-linux-gnueabihf/
arm-linux-gnueabihf-gcc -c -Wall -DPJ_AUTOCONF=1 -O2 -DPJ_IS_BIG_ENDIAN=0 -DPJ_IS_LITTLE_ENDIAN=1    -I../include \
        -o output/pjlib-arm-unknown-linux-gnueabihf/ioqueue_select.o \
        ../src/pj/ioqueue_select.c
arm-linux-gnueabihf-gcc -c -Wall -DPJ_AUTOCONF=1 -O2 -DPJ_IS_BIG_ENDIAN=0 -DPJ_IS_LITTLE_ENDIAN=1    -I../include \
        -o output/pjlib-arm-unknown-linux-gnueabihf/file_access_unistd.o \
        ../src/pj/file_access_unistd.c
arm-linux-gnueabihf-gcc -c -Wall -DPJ_AUTOCONF=1 -O2 -DPJ_IS_BIG_ENDIAN=0 -DPJ_IS_LITTLE_ENDIAN=1    -I../include \
        -o output/pjlib-arm-unknown-linux-gnueabihf/file_io_ansi.o \
        ../src/pj/file_io_ansi.c
arm-linux-gnueabihf-gcc -c -Wall -DPJ_AUTOCONF=1 -O2 -DPJ_IS_BIG_ENDIAN=0 -DPJ_IS_LITTLE_ENDIAN=1    -I../include \
        -o output/pjlib-arm-unknown-linux-gnueabihf/os_core_unix.o \
        ../src/pj/os_core_unix.c
```

Figure 3: Expected output

You should now have the executable of a sample app made to run on your beagle bone which composes most of the feature offered pjsip library

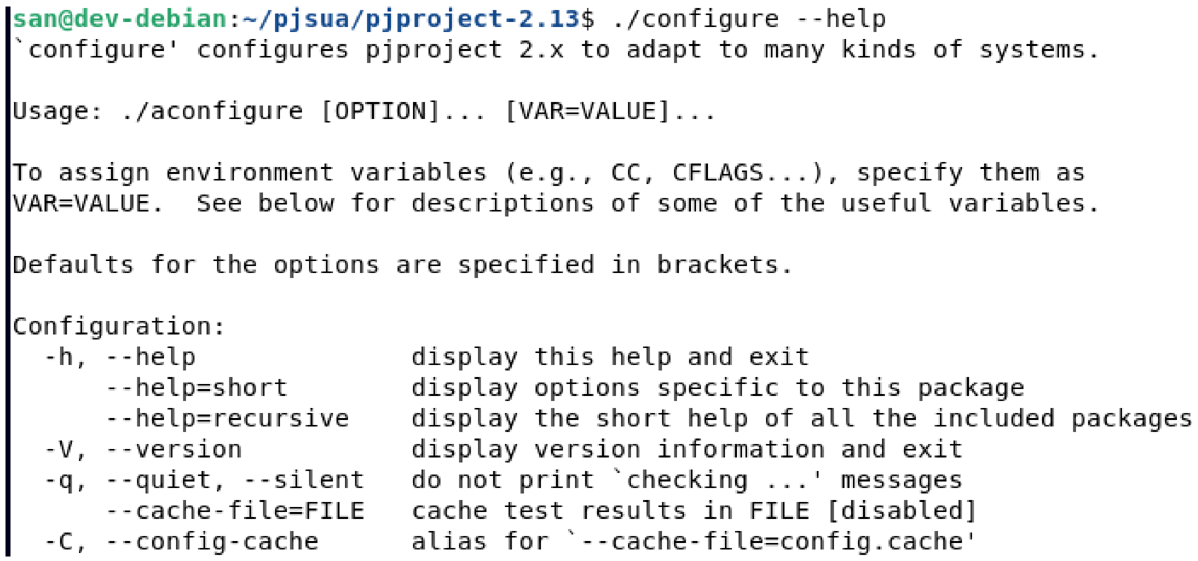
Troubleshooting:
At all these stages you will encounter many errors. These are mainly due the scripts trying to find the resources on your host in order to compile the libraries. In some cases, there may be no support for the entire set of features provided by the library which are supported.

In order to work around these errors you have two options:

1. Disable the features that require the missing resources

    ```
    (host)$ ./configure --help
    ```

    This bring up all options that can be passed to this configure file



```
san@dev-debian:~/pjsua/pjproject-2.13$ ./configure --help
`configure' configures pjproject 2.x to adapt to many kinds of systems.

Usage: ./aconfigure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE.  See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this package
      --help=recursive    display the short help of all the included packages
  -V, --version           display version information and exit
  -q, --quiet, --silent   do not print `checking ...' messages
      --cache-file=FILE   cache test results in FILE [disabled]
  -C, --config-cache      alias for `--cache-file=config.cache'
```

Figure 4: Expected output

2. Install the missing resources

1.5 Copy the executable to the beaglebone

Move inside the folder containing the sample app binaries:

```
(host)$ cd /pjproject-2.13/pjsip-apps/bin
```

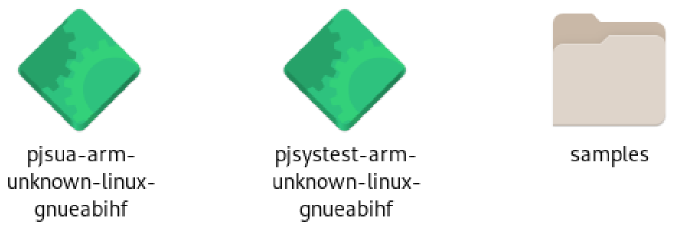

pjsua-arm-unknown-linux-gnueabihf
pjsystest-arm-unknown-linux-gnueabihf
samples

Figure 5: Folder contents

Copy the executable `pjsua-arm-unknown-linux-gnueabihf` to the nfs folder `myApps:`

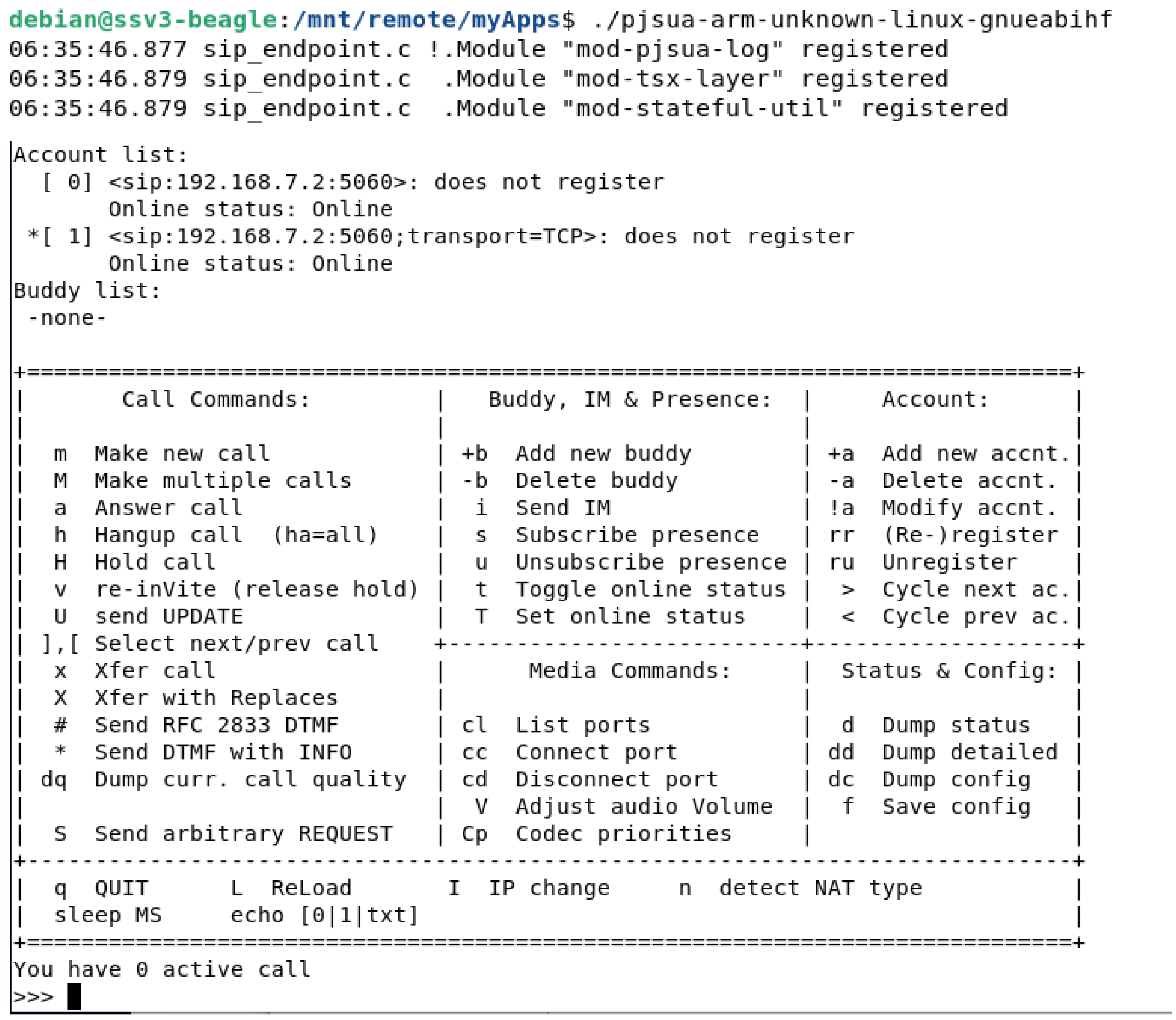
```
(host)$ cp pjsua-arm-unknown-linux-gnueabihf ~/cmpt433/public/myApps
```

2. Running the sample app

2.1 Running the app

SSH into your beaglebone and run the app:

```
(host)$ ssh debian@192.168.7.2
(bbg)$ cd /mnt/remote/myApps
(bbg)$ ./pjsua-arm-unknown-linux-gnueabihf
```



```
debian@ssv3-beagle:/mnt/remote/myApps$ ./pjsua-arm-unknown-linux-gnueabihf
06:35:46.877 sip_endpoint.c !.Module "mod-pjsua-log" registered
06:35:46.879 sip_endpoint.c  .Module "mod-tsx-layer" registered
06:35:46.879 sip_endpoint.c  .Module "mod-stateful-util" registered
```

```
Account list:
  [ 0] <sip:192.168.7.2:5060>: does not register
       Online status: Online
 *[ 1] <sip:192.168.7.2:5060;transport=TCP>: does not register
       Online status: Online
Buddy list:
 -none-

+=============================================================================+
|       Call Commands:         |   Buddy, IM & Presence:  |     Account:      |
|                              |                          |                   |
|  m  Make new call            | +b  Add new buddy        | +a  Add new accnt.|
|  M  Make multiple calls      | -b  Delete buddy         | -a  Delete accnt. |
|  a  Answer call              |  i  Send IM              | !a  Modify accnt. |
|  h  Hangup call  (ha=all)    |  s  Subscribe presence   | rr  (Re-)register |
|  H  Hold call                |  u  Unsubscribe presence | ru  Unregister    |
|  v  re-inVite (release hold) |  t  Toggle online status |  >  Cycle next ac.|
|  U  send UPDATE              |  T  Set online status    |  <  Cycle prev ac.|
| ],[ Select next/prev call    +--------------------------+-------------------+
|  x  Xfer call                |      Media Commands:     |  Status & Config: |
|  X  Xfer with Replaces       |                          |                   |
|  #  Send RFC 2833 DTMF       | cl  List ports           |  d  Dump status   |
|  *  Send DTMF with INFO      | cc  Connect port         | dd  Dump detailed |
| dq  Dump curr. call quality  | cd  Disconnect port      | dc  Dump config   |
|                              |  V  Adjust audio Volume  |  f  Save config   |
|  S  Send arbitrary REQUEST   | Cp  Codec priorities     |                   |
+-----------------------------------------------------------------------------+
|  q  QUIT      L  ReLoad       I  IP change     n  detect NAT type          |
|  sleep MS     echo [0|1|txt]                                               |
+=============================================================================+
You have 0 active call
>>> 
```

Figure 6: Expected output when running the app

If you have reached this step you have successfully compiled the libraries. Now you can use PJSIP in your own C program for the features you require.

You can learn how to utilize the library by playing around with this sample app and analyzing the source code behind this app. It is open source, and can be found in the pjproject directory pj-apps and on github.

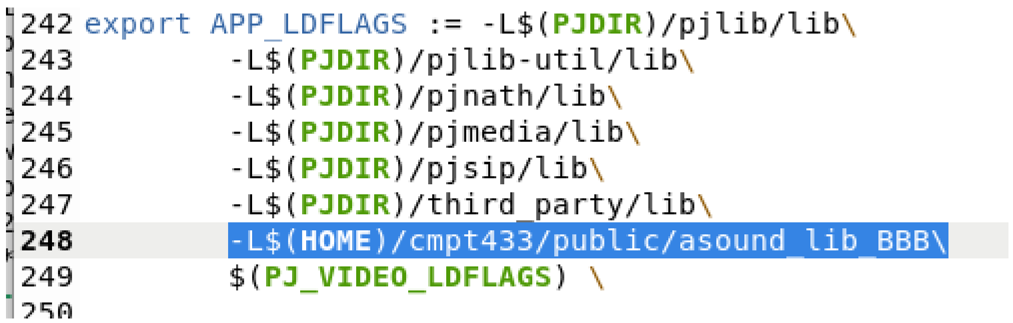
2.2 Interfacing with the app

To interface with this app, you can download a softphone on your host that can call the ip address registered by this app. We recommend using the Linphone app: https://www.linphone.org
You will need to figure out the hardware you are utilizing for audio input and output and pass in appropriate parameters:

```
(bbg)$  ./pjsua-arm-unknown-linux-gnueabihf --help
```

3. Sample makefile and C program using PJSIP lib

For this makefile to work, go to the `build.mak` file in the `pjproject-2.13` directory and add the path to `-lasound`



```
export APP_LDFLAGS := -L$(PJDIR)/pjlib/lib\
        -L$(PJDIR)/pjlib-util/lib\
        -L$(PJDIR)/pjnath/lib\
        -L$(PJDIR)/pjmedia/lib\
        -L$(PJDIR)/pjsip/lib\
        -L$(PJDIR)/third_party/lib\
        -L$(HOME)/cmpt433/public/asound_lib_BBB\
        $(PJ_VIDEO_LDFLAGS) \
```

Without this you will encounter this error:

```
/usr/lib/gcc-cross/arm-linux-gnueabihf/10/../../../../arm-linux-gnueabihf/bin/ld: cannot find
-lasound
```

Sample makefile:

```
PJDIR = $(HOME)/pjsua/pjproject-2.13
include $(PJDIR)/build.mak
OUTDIR = $(HOME)/cmpt433/public/myApps
OUTFILE = $(OUTDIR)/hello_pjsua

$(OUTFILE): my_app.c
	$(PJ_CC) -o $(OUTFILE) $< $(PJ_CFLAGS) $(PJ_LDFLAGS) $(PJ_LDLIBS)

all: $(OUTFILE)

clean:
	rm -f $(OUTFILE)
```

Sample C program:

```
#include <pjsua-lib/pjsua.h>
#include <pj/log.h>
#include <stdio.h>

#include <pjmedia/sound.h>

int main()
{


  pj_status_t status;

  status = pjsua_create();
  PJ_LOG(3, ("myapp.c", "Hello PJSIP! Bye PJSIP."));

  pjsua_destroy();
  return 0;
}
```

Compile and run the program:

```
(host)$  make
(bbg)$  ./hello_pjsua
```

Sample run demonstrating that you can now use pjsua api in your program:

```
debian@ssv3-beagle:/mnt/remote/myApps$ ./hello_pjsua
07:12:29.245 os_core_unix.c !pjlib 2.13 for POSIX initialized
07:12:29.303 sip_endpoint.c  .Creating endpoint instance...
07:12:29.370          pjlib  .select() I/O Queue created (0x247d2d8)
07:12:29.370 sip_endpoint.c  .Module "mod-msg-print" registered
07:12:29.371 sip_transport.  .Transport manager created.
07:12:29.371   pjsua_core.c  .PJSUA state changed: NULL --> CREATED
07:12:29.371        myapp.c  Hello PJSIP! Bye PJSIP.
07:12:29.371   pjsua_core.c  Shutting down, flags=0...
07:12:29.371   pjsua_core.c  PJSUA state changed: CREATED --> CLOSING
07:12:29.371   pjsua_call.c  .Hangup all calls..
07:12:29.371  pjsua_media.c  .Call 0: deinitializing media..
07:12:29.371  pjsua_media.c  .Call 1: deinitializing media..
07:12:29.371  pjsua_media.c  .Call 2: deinitializing media..
07:12:29.371  pjsua_media.c  .Call 3: deinitializing media..
07:12:29.371   pjsua_pres.c  .Shutting down presence..
07:12:30.379   pjsua_core.c  .Destroying...
07:12:30.380  pjsua_media.c  .Shutting down media..
07:12:30.381 sip_endpoint.c  .Destroying endpoint instance..
07:12:30.383 sip_endpoint.c  .Module "mod-msg-print" unregistered
07:12:30.383 sip_transport.  .Destroying transport manager
07:12:30.384        timer.c  .Dumping timer heap:
07:12:30.385        timer.c  .  Cur size: 0 entries, max: 3070
07:12:30.385 sip_endpoint.c  .Endpoint 0x245697c destroyed
07:12:30.386   pjsua_core.c  .PJSUA state changed: CLOSING --> NULL
07:12:30.387   pjsua_core.c  .PJSUA destroyed...
```