Test results report
2016-2017

Master's thesis
Exploration of the TI AM5728 Audio/Video Subsystem

Dennis Joosens

Universiteit Antwerpen

1 Abstract

In this report all the measurements that are conducted during the master thesis are briefly summarized. This includes setting up the test environment to various schedules and test results of audio latency and network latency.

Contents

2 Hardware

Hereby a concise overview of the hardware that has been used during the execution of this master thesis.

- 2 x Texas Instrument AM5728 Evaluation Module
- 3.5 mm jack male to 3.5 mm jack male stereo cable
- Agilent MSOX3024A Mixed Signal Oscilloscope, 200 MHz 4Gs/s
- NTi Audio Minirator MR-PRO
- Agilent 33220A 20 MHz Function / Arbitrary Waveform Generator
- Tektronix TBS1104 Four Channel Digital Storage Oscilloscope, 100 MHz 1Gs/s
- 3 x Ethernet Cat5e cables
- Gigabit Ethernet switch
- Virtual machine with the latest version of Linux Ubuntu installed on it

3 Setup virtual machine

For this project we have setup a Virtual Machine that runs Linux Ubuntu 16.04 LTS. The reason for this is that the EVM board also runs Linux which facilitates a lot of stuff. Another main reason is that the communication with the EVM board happens using a USB TTL serial cable which is natively supported by Linux.

3.1 Communicating to the EVM board

The communication with the EVM board happens using a serial connection. To communicate over a serial link we use Minicom. Minicom is a serial port communications program. The program is natively not installed on Linux. By executing the following commands we can start communication to the EVM board.

```
# sudo apt-get install minicom
```

To find the name of our port:

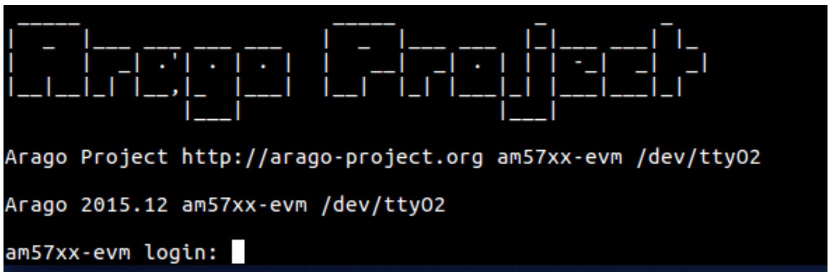
```
# dmesg | grep tty
```

In our case the name is ttyUSB0. Now we need to change the setup configuration of Minicom.

```
# minicom -s
```

From here go to Serial port setup and make sure the **Serial Device name** is /dev/ttyUSB0. Eventually save this setup as default.

Now select **exit** and the serial connection to the board starts. If all goes well, we get a terminal login.



```
Arago Project http://arago-project.org am57xx-evm /dev/ttyO2

Arago 2015.12 am57xx-evm /dev/ttyO2

am57xx-evm login:
```

Figure 1: Login screen when connected to the EVM board

3.2 Setup static IP on the EVM boards and the VM

The communication with the EVM board happens using a serial connection at first. We have setup Minicom so we can now set a static IP on the EVM boards and eventually connect the VM and the EVM boards to a network switch.

Note: The IP range can be chosen freely as long as you stay in the same range for all the configurations.

We have chosen the following IP's for the setup:

```
EVM board 1: 192.168.10.10
EVM board 2: 192.168.10.11
         VM: 192.168.10.20
```

We can now configure a static IP by changing the following file:

```
# nano /etc/network/interfaces
```

Add the following to the file:

```
auto lo eth0
iface eth0 inet static
        address 192.168.10.10
        netmask 255.255.255.0
        gateway 192.168.10.1
```

Note that the device name **eth0** can deviate depending on the machine. A reboot might be needed.

```
# reboot
```

3.3 Cross compiling

GCC is by default installed on most Linux machines. However to compile code for the ARM architecture we need to download the SDK and install it. The **Processor SDK** is available from the website of Texas Instruments.

```
# opt/ti-processor-sdk-linux-am57xx-evm-03.02.00.05/
linux-devkit/sysroots/x86_64-arago-linux/usr/bin/
arm-linux-gnueabihf-gcc <file.c> -o <file>
```

You can add the full path to the **$PATH** environmental variable.

```
# nano /etc/environment
```

To check if it is successfully added.

```
# echo $PATH
```

3.4 Setup SCP

In this project a lot of C code has been written and needs to be transferred to the EVM board. We could use a USB dongle to do this. However, from our experience we know that this process is kind of slow since we have to mount our USB dongle every time and copy the files. Another approach is to send it over the USB TTL serial cable, but we would need to encode and decode the data every time. The best and much faster approach is to use a point-to-point Ethernet connection to the EVM board and use SCP. SCP is standard installed on our Linux machine, we just needed to setup a static IP's on the hosts and we are ready to send a file directly to the EVM board. The following command has been used:

```
$ scp <my_file> root@192.168.10.10:/home/root
```

The compiled C file will be placed at the /home/root directory on the EVM board.

4 Audio latency measurements

4.1 Measurement 1: Gstreamer sine wave

To conduct this measurement we use Gstreamer. Gstreamer is installed by default on the TI EVM board. Note that the signal we send into the device should not have an amplitude bigger than 3.3 V. Using a smaller amplitude signal is advised. To connect the capture device to the playback device in software, we use the following command:

```
# gst-launch-1.0 alsasrc ! alsasink
```

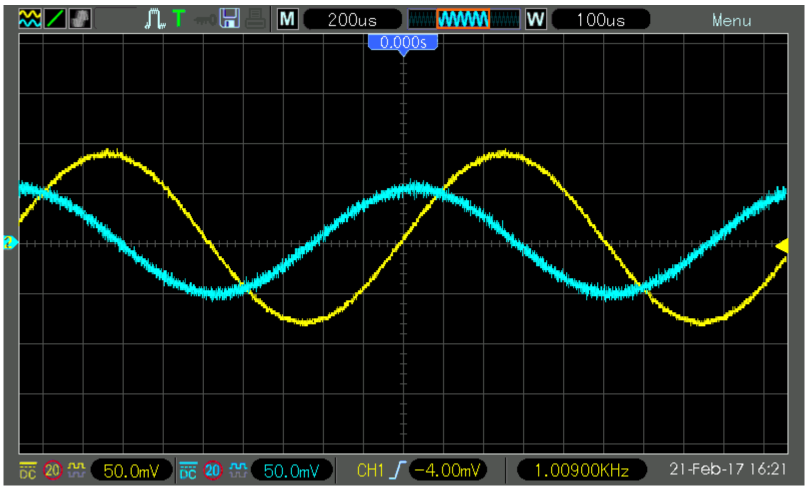


Figure 2: Measurement sine wave as input

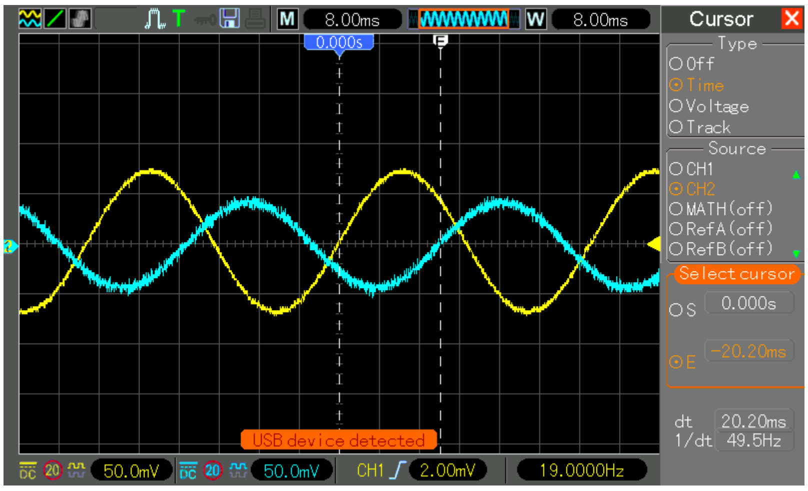


Figure 3: Measurement sine wave as input

The latency we get is dependable on the frequency of the signal that we have sent into the device. We get 20 ms at 20 Hz and 755 μs at 20 kHz. The higher the frequency, the lower the latency. However the way this measurement is executed is wrong. A delay should not be frequency dependent but stay the same value. The main reason is that we make use of sinusoidal signals. These signals are repetitive signals and thereby give a wrong impression. The delay could be smaller or even larger. Another effect is that the output signal is attenuated. When we send in a frequency above 20 kHz we see that the signal attenuates. The output signal totally vanishes when we reach 21 kHz.

4.2 Measurement 2: GStreamer microphone and headphones

An alternative measurement to get an indication of the latency is by executing the command of measurement 1. From here on, we connect a microphone to the capture device and a headphone or speaker to the playback device. When talking into the microphone, we notice that the delay is audible, there is an echo. We can conclude that the delay exceeds 100 ms. However, from this measurement we cannot determine the latency yet.

4.3 Measurement 3: GStreamer impulse measurement

A better way to measure the latency is by using an impulse signal as input signal or a repetitive burst signal with changing amplitude. We execute the terminal command of measurement 1 again and get the result that is shown in figure 4.

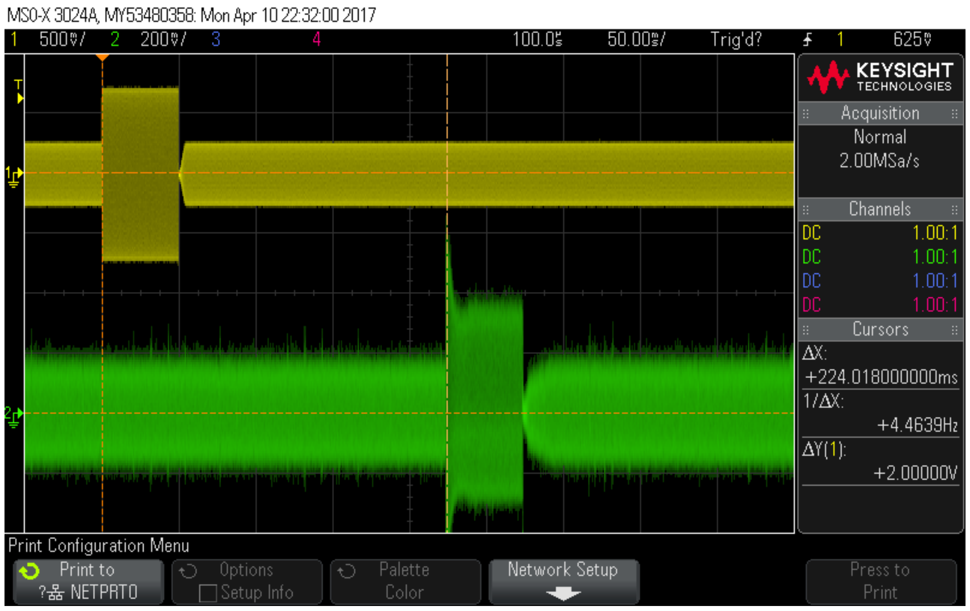


Figure 4: GStreamer with impulse as input signal

The figure shows a latency of about 224-225 ms which can be correct.

4.4 Measurement 4: latency file

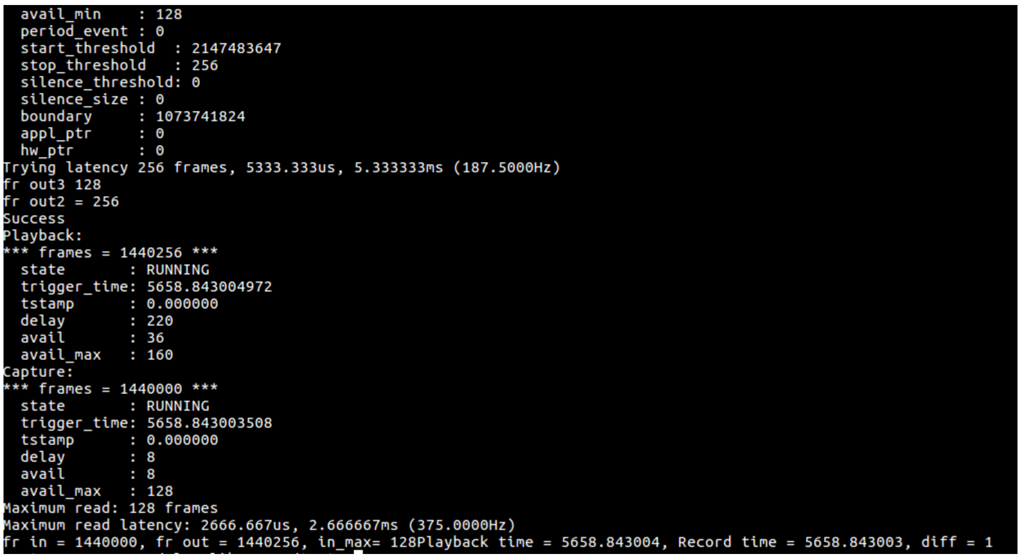
When we use the latency file we get a calculated latency however when comparing to the hardware latency we see that these values slightly differ. We used the following commands.

```
# arm-linux-gnueabihf-gcc latency.edited.c
-o latency.edited -lasound -lm

# scp latency.edited root@192.168.10.10:/home/root

# ./latency.edited -m 256 -p
```

Note that the frame size (256) varies in this test.



```
  avail_min    : 128
  period_event : 0
  start_threshold  : 2147483647
  stop_threshold   : 256
  silence_threshold: 0
  silence_size : 0
  boundary     : 1073741824
  appl_ptr     : 0
  hw_ptr       : 0
Trying latency 256 frames, 5333.333us, 5.333333ms (187.5000Hz)
fr out3 128
fr out2 = 256
Success
Playback:
*** frames = 1440256 ***
  state       : RUNNING
  trigger_time: 5658.843004972
  tstamp      : 0.000000
  delay       : 220
  avail       : 36
  avail_max   : 160
Capture:
*** frames = 1440000 ***
  state       : RUNNING
  trigger_time: 5658.843003508
  tstamp      : 0.000000
  delay       : 8
  avail       : 8
  avail_max   : 128
Maximum read: 128 frames
Maximum read latency: 2666.667us, 2.666667ms (375.0000Hz)
fr in = 1440000, fr out = 1440256, in_max= 128Playback time = 5658.843004, Record time = 5658.843003, diff = 1
```

Figure 5: latency.edited -m 256 -p, @48kHz

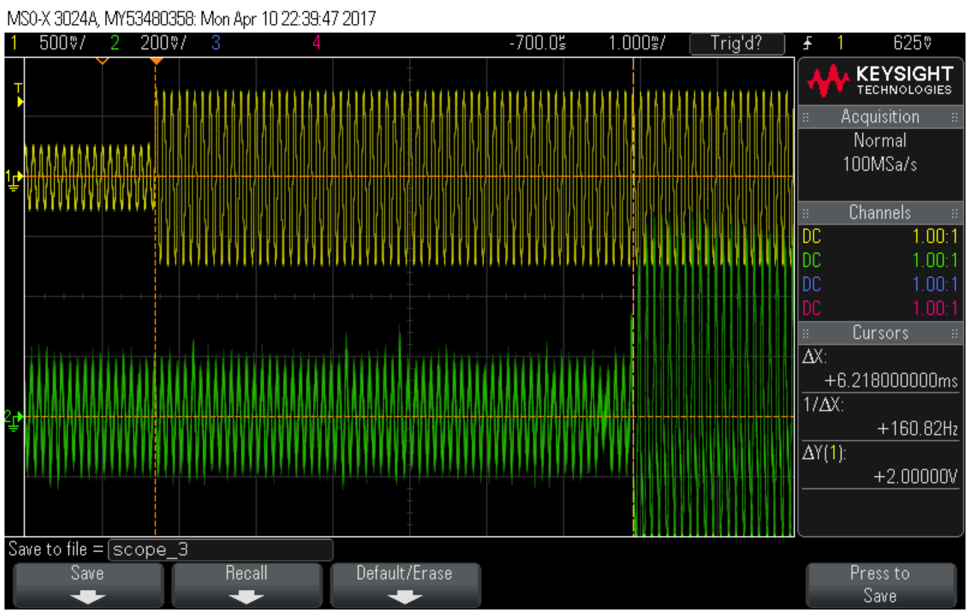
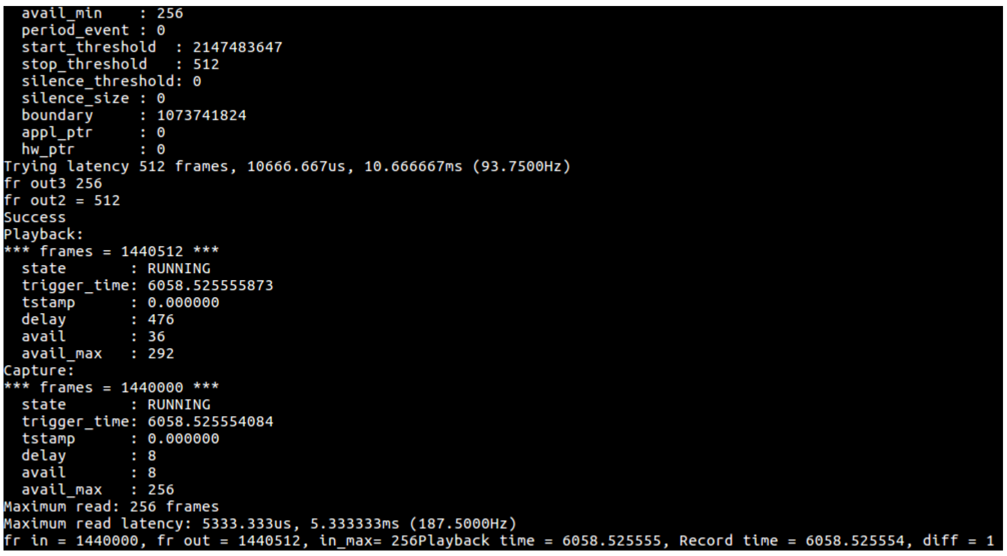


Figure 6: latency.edited -m 256 -p, @48kHz



```
  avail_min    : 256
  period_event : 0
  start_threshold  : 2147483647
  stop_threshold   : 512
  silence_threshold: 0
  silence_size : 0
  boundary     : 1073741824
  appl_ptr     : 0
  hw_ptr       : 0
Trying latency 512 frames, 10666.667us, 10.666667ms (93.7500Hz)
fr out3 256
fr out2 = 512
Success
Playback:
*** frames = 1440512 ***
  state       : RUNNING
  trigger_time: 6058.525555873
  tstamp      : 0.000000
  delay       : 476
  avail       : 36
  avail_max   : 292
Capture:
*** frames = 1440000 ***
  state       : RUNNING
  trigger_time: 6058.525554084
  tstamp      : 0.000000
  delay       : 8
  avail       : 8
  avail_max   : 256
Maximum read: 256 frames
Maximum read latency: 5333.333us, 5.333333ms (187.5000Hz)
fr in = 1440000, fr out = 1440512, in_max= 256Playback time = 6058.525555, Record time = 6058.525554, diff = 1
```

Figure 7: latency.edited -m 512 -p, @48kHz

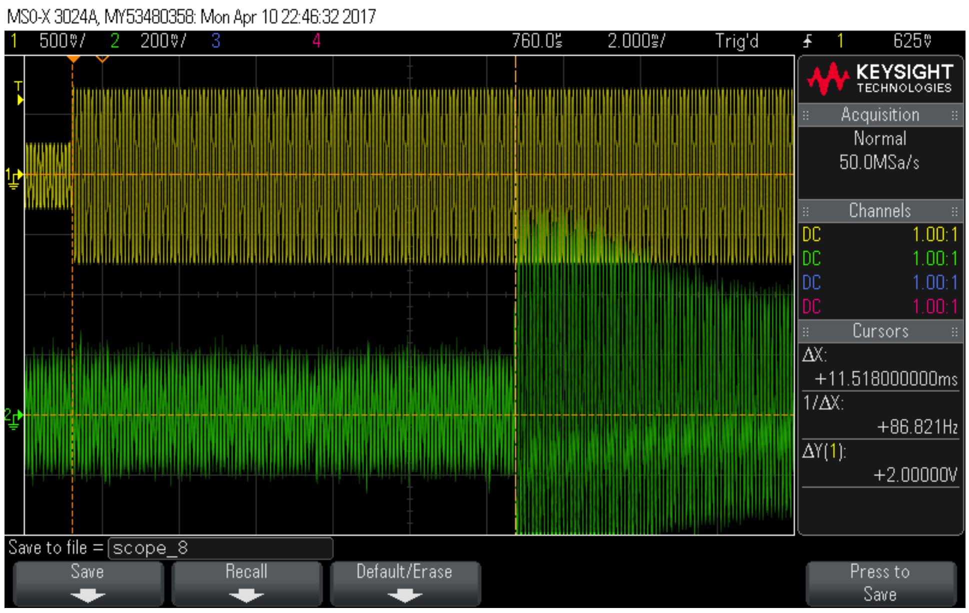
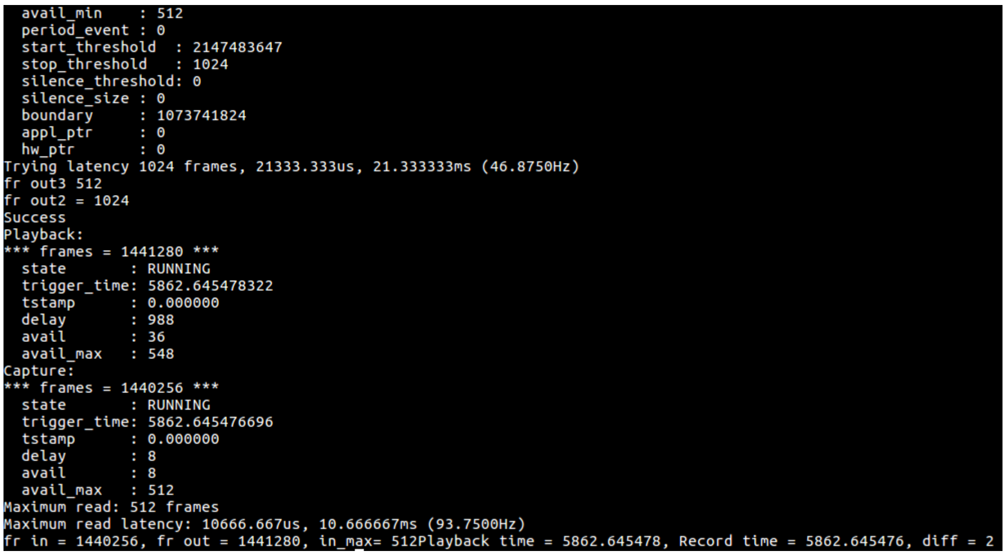


Figure 8: latency.edited -m 512 -p, @48kHz



```
  avail_min    : 512
  period_event : 0
  start_threshold  : 2147483647
  stop_threshold   : 1024
  silence_threshold: 0
  silence_size : 0
  boundary     : 1073741824
  appl_ptr     : 0
  hw_ptr       : 0
Trying latency 1024 frames, 21333.333us, 21.333333ms (46.8750Hz)
fr out3 512
fr out2 = 1024
Success
Playback:
*** frames = 1441280 ***
  state       : RUNNING
  trigger_time: 5862.645478322
  tstamp      : 0.000000
  delay       : 988
  avail       : 36
  avail_max   : 548
Capture:
*** frames = 1440256 ***
  state       : RUNNING
  trigger_time: 5862.645476696
  tstamp      : 0.000000
  delay       : 8
  avail       : 8
  avail_max   : 512
Maximum read: 512 frames
Maximum read latency: 10666.667us, 10.666667ms (93.7500Hz)
fr in = 1440256, fr out = 1441280, in_max= 512Playback time = 5862.645478, Record time = 5862.645476, diff = 2
```

Figure 9: latency.edited -m 1024 -p, @48kHz

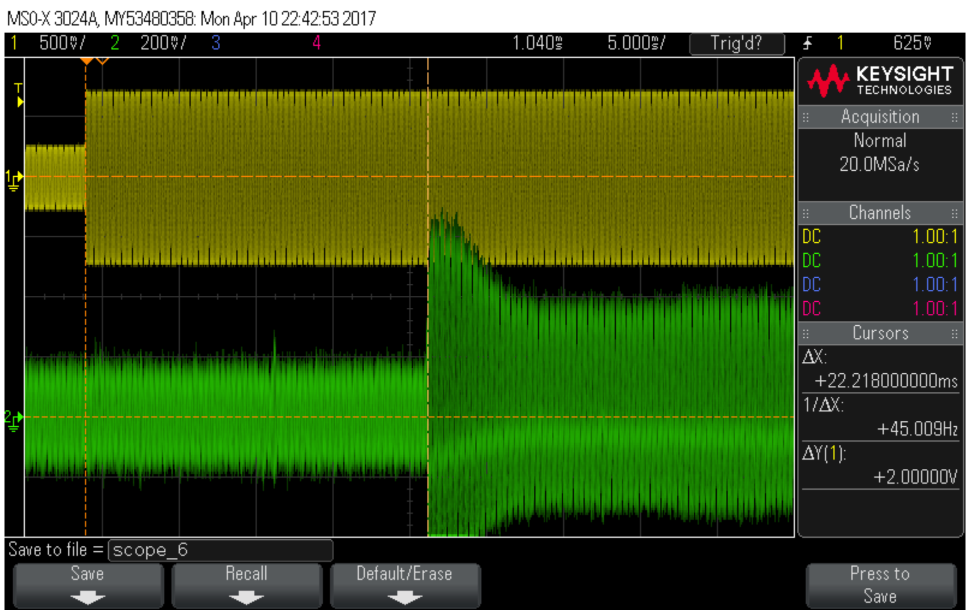
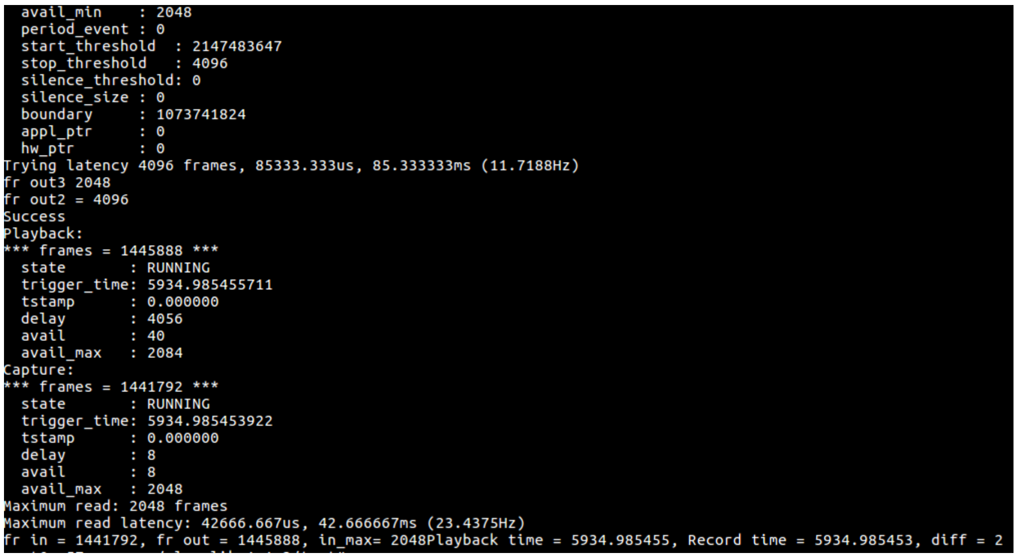


Figure 10: latency.edited -m 1024 -p, @48kHz



```
  avail_min    : 2048
  period_event : 0
  start_threshold  : 2147483647
  stop_threshold   : 4096
  silence_threshold: 0
  silence_size : 0
  boundary     : 1073741824
  appl_ptr     : 0
  hw_ptr       : 0
Trying latency 4096 frames, 85333.333us, 85.333333ms (11.7188Hz)
fr out3 2048
fr out2 = 4096
Success
Playback:
*** frames = 1445888 ***
  state       : RUNNING
  trigger_time: 5934.985455711
  tstamp      : 0.000000
  delay       : 4056
  avail       : 40
  avail_max   : 2084
Capture:
*** frames = 1441792 ***
  state       : RUNNING
  trigger_time: 5934.985453922
  tstamp      : 0.000000
  delay       : 8
  avail       : 8
  avail_max   : 2048
Maximum read: 2048 frames
Maximum read latency: 42666.667us, 42.666667ms (23.4375Hz)
fr in = 1441792, fr out = 1445888, in_max= 2048Playback time = 5934.985455, Record time = 5934.985453, diff = 2
```

Figure 11: latency.edited -m 4096 -p, @48kHz

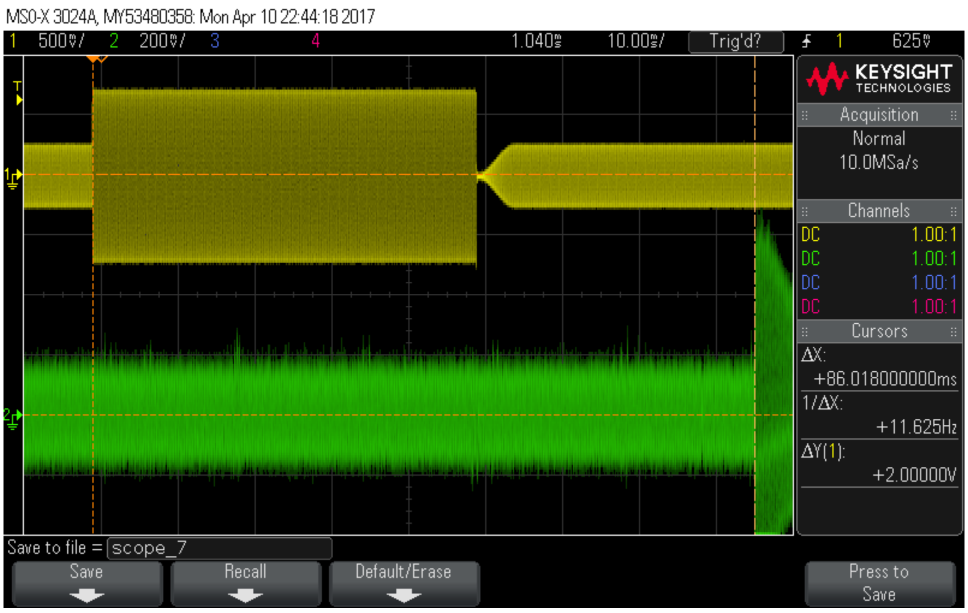
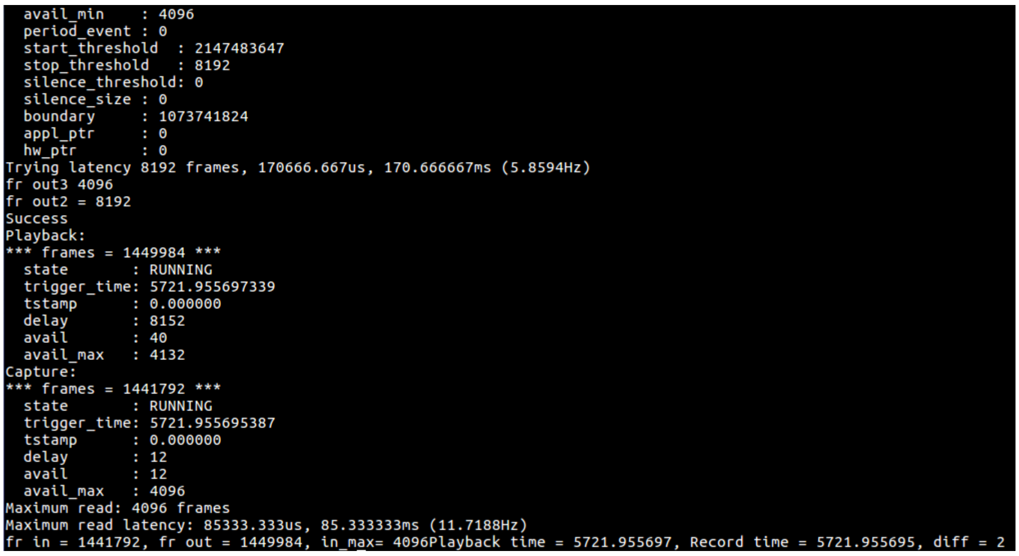


Figure 12: latency.edited -m 4096 -p, @48kHz



```
  avail_min    : 4096
  period_event : 0
  start_threshold  : 2147483647
  stop_threshold   : 8192
  silence_threshold: 0
  silence_size : 0
  boundary     : 1073741824
  appl_ptr     : 0
  hw_ptr       : 0
Trying latency 8192 frames, 170666.667us, 170.666667ms (5.8594Hz)
fr out3 4096
fr out2 = 8192
Success
Playback:
*** frames = 1449984 ***
  state       : RUNNING
  trigger_time: 5721.955697339
  tstamp      : 0.000000
  delay       : 8152
  avail       : 40
  avail_max   : 4132
Capture:
*** frames = 1441792 ***
  state       : RUNNING
  trigger_time: 5721.955695387
  tstamp      : 0.000000
  delay       : 12
  avail       : 12
  avail_max   : 4096
Maximum read: 4096 frames
Maximum read latency: 85333.333us, 85.333333ms (11.7188Hz)
fr in = 1441792, fr out = 1449984, in_max= 4096Playback time = 5721.955697, Record time = 5721.955695, diff = 2
```

Figure 13: latency.edited -m 8192 -p, @48kHz

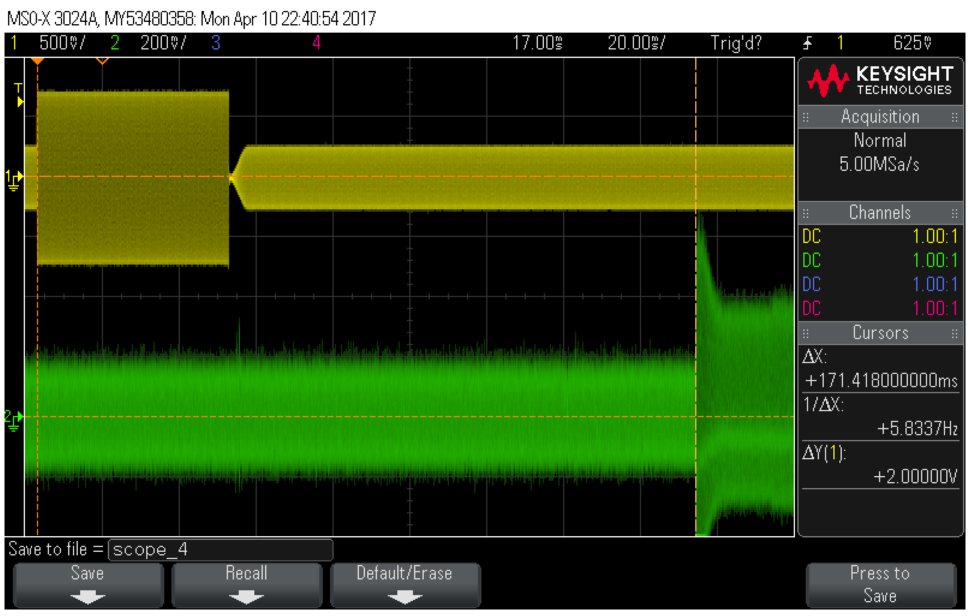


Figure 14: latency.edited -m 8192 -p, @48kHz

4.5 Pink noise

To execute the following command you need the code of the following directory which can be found in the portfolio: pink_noise/. The cross compiled code can also be found here. Make sure that the ALSA library is available on the VM and execute the file in the following folder: alsa-lib-1.1.3/test/.

```
# arm-linux-gnueabihf-gcc pcm_min.c
-o pcm_min -lasound
```

Execute the file on the EVM board.

```
# ./pcm_min
```

To plot 100 samples we need to have gnuplot installed.

```
# apt-get install gnuplot-x11
```

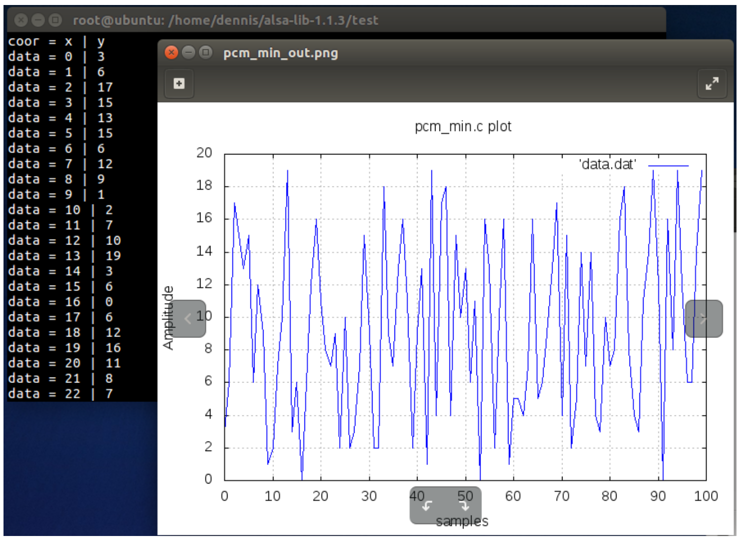
To execute the following command you need the code of the following directory which can be found in the portfolio: pink_noise/.

```
# gcc pcm_min_data_plot.c -o pcm_min_data_plot
```

Execute the file on the VM.

```
# ./pcm_min_data_plot
```

A .PNG file is created in the directory where the file is executed. The result is pink noise which is also used in the latency.c file.



```
coor = x | y
data = 0 | 3
data = 1 | 6
data = 2 | 17
data = 3 | 15
data = 4 | 13
data = 5 | 15
data = 6 | 6
data = 7 | 12
data = 8 | 9
data = 9 | 1
data = 10 | 2
data = 11 | 7
data = 12 | 10
data = 13 | 19
data = 14 | 3
data = 15 | 6
data = 16 | 0
data = 17 | 6
data = 18 | 12
data = 19 | 16
data = 20 | 11
data = 21 | 8
data = 22 | 7
```

Figure 15: Pink noise

5 Stress testing

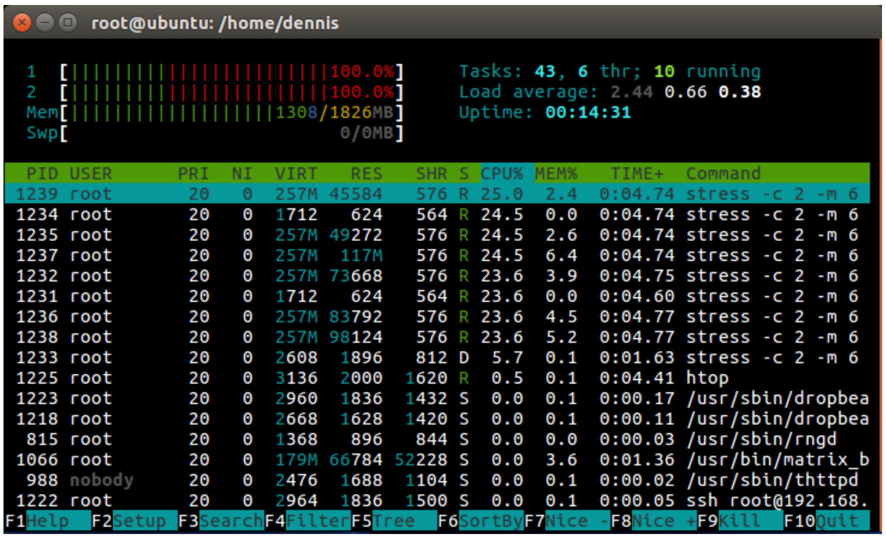
Another test was to investigate how the audio latency responds and performs under stress. By conducting this measurement we want to see if the latency increases.

There are many tools freely available to execute this measurement. However we have used two tools which where already installed on the EVM board.
We can also use the following command to generate CPU load.

```
# md5sum /dev/zero &
```

However, this only sets one CPU core to 100%. Notice that the ampersand sign adds the process to the background while releasing the terminal. We can call the process by typing **fg**.

To do some advanced stress testing we use the ”stress tool” and the ”cpuburn tool”. With the cpuburn tool we can only stress the CPU’s and no other hardware components. This is why we only display results of the stress test tool. Another tool we use is htop to monitor the system closely. In figure 16 we see the resource usage when maximum stress is applied.



```
root@ubuntu: /home/dennis

  1  [|||||||||||||||||||||||||100.0%]     Tasks: 43, 6 thr; 10 running
  2  [|||||||||||||||||||||||||100.0%]     Load average: 2.44 0.66 0.38
  Mem[||||||||||||||||||||1308/1826MB]     Uptime: 00:14:31
  Swp[                          0/0MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1239 root       20   0  257M 45584   576 R 25.0  2.4  0:04.74 stress -c 2 -m 6
 1234 root       20   0  1712   624   564 R 24.5  0.0  0:04.74 stress -c 2 -m 6
 1235 root       20   0  257M 49272   576 R 24.5  2.6  0:04.74 stress -c 2 -m 6
 1237 root       20   0  257M  117M   576 R 24.5  6.4  0:04.74 stress -c 2 -m 6
 1232 root       20   0  257M 73668   576 R 23.6  3.9  0:04.75 stress -c 2 -m 6
 1231 root       20   0  1712   624   564 R 23.6  0.0  0:04.60 stress -c 2 -m 6
 1236 root       20   0  257M 83792   576 R 23.6  4.5  0:04.77 stress -c 2 -m 6
 1238 root       20   0  257M 98124   576 R 23.6  5.2  0:04.77 stress -c 2 -m 6
 1233 root       20   0  2608  1896   812 D  5.7  0.1  0:01.63 stress -c 2 -m 6
 1225 root       20   0  3136  2000  1620 R  0.5  0.1  0:04.41 htop
 1223 root       20   0  2960  1836  1432 S  0.0  0.1  0:00.17 /usr/sbin/dropbea
 1218 root       20   0  2668  1628  1420 S  0.0  0.1  0:00.11 /usr/sbin/dropbea
  815 root       20   0  1368   896   844 S  0.0  0.0  0:00.03 /usr/sbin/rngd
 1066 root       20   0  179M 66784 52228 S  0.0  3.6  0:01.36 /usr/bin/matrix_b
  988 nobody     20   0  2476  1688  1104 S  0.0  0.1  0:00.02 /usr/sbin/thttpd
 1222 root       20   0  2964  1836  1500 S  0.0  0.1  0:00.05 ssh root@192.168.
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 16: Monitoring the system using htop

The commands we have used.

```
# latency.edited -m 256 -s 120 -p
# stress -c 2 -m 6 -d 1
```

The following figures are some measurements that we have conducted.

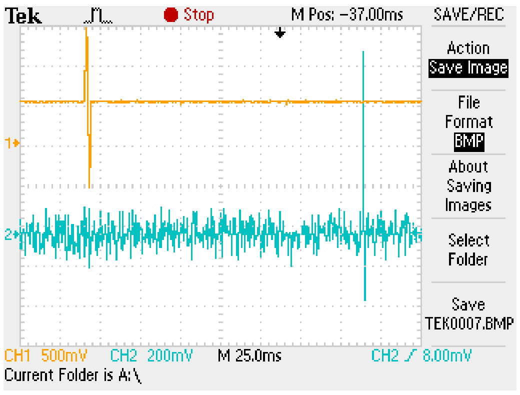


Figure 17: Commands: ./latency.edited -m 8192 -s 120 stress -c 2. We get a hardware latency of 174 ms and in software 170.66 ms.

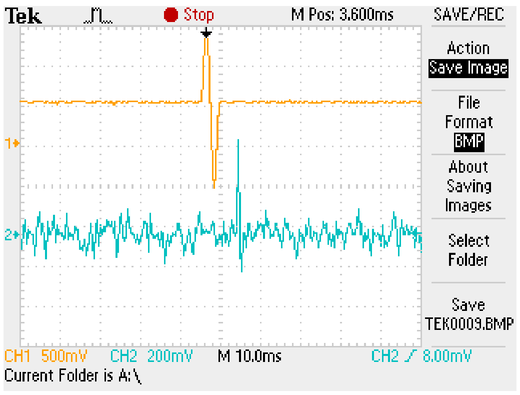


Figure 18: Commands: ./latency.edited -m 256 -s 120, stress -c 2. We get a hardware latency of 8.8 ms and in software 6.21 ms.

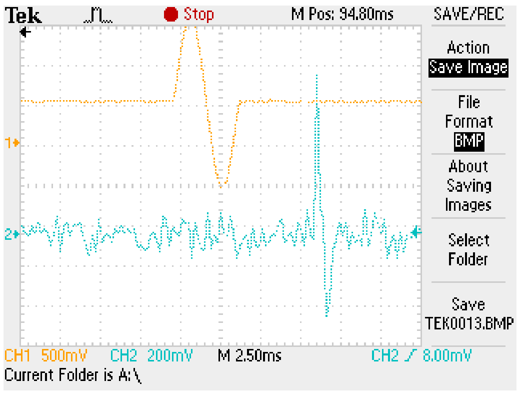


Figure 19: commands: ./latency.edited -m 256 -s 120 stress -c 2 -m 6. We get a hardware latency of 8.8 ms and in software 6.21 ms.

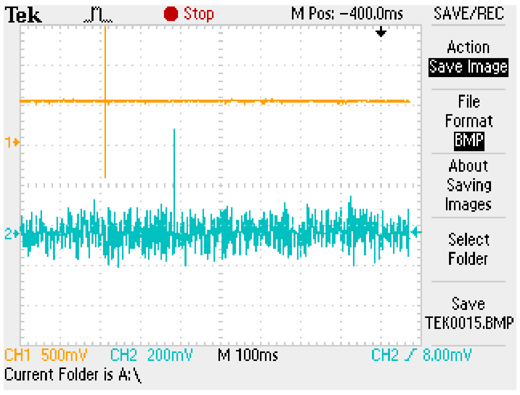


Figure 20: Commands: ./latency.edited -m 8192 -s 120, stress -c 2 -m 6 -d 1. We get a hardware latency of 174 ms and in software 171 ms.

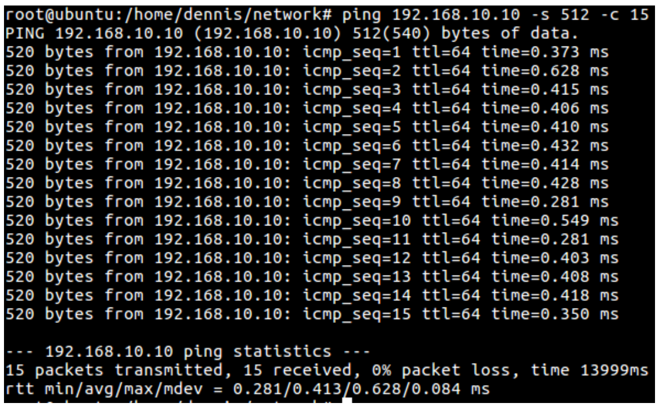
We conclude that the stress testing does not have a lot of influence on the audio latency which is a good sign.

6 Network latency measurements

6.1 Measurement 1: Execution a ping

For the first measurement we have executed a simple ping command from the VM board to the EVM board. The ping command makes use of the ICMP protocol. ICMP is a network layer protocol that is used for simple queries and error reporting. The ICMP header and ICMP payload is embedded into an IP packet. The ICMP header has a default size of 8 bytes. In our test we have send 15 ECHO_REQUESTS if we get a ECHO_REPLY, we can see the Round-Trip-Time. We send packets with a payload of 512 bytes. These are shown as 520 bytes with the ICMP header included. The experiment gave us an average Round Trip Time of 0.413 ms.We have used the following command:

```
$ ping 192.168.10.10 -s 512 -c 15
```



```
root@ubuntu:/home/dennis/network# ping 192.168.10.10 -s 512 -c 15
PING 192.168.10.10 (192.168.10.10) 512(540) bytes of data.
520 bytes from 192.168.10.10: icmp_seq=1 ttl=64 time=0.373 ms
520 bytes from 192.168.10.10: icmp_seq=2 ttl=64 time=0.628 ms
520 bytes from 192.168.10.10: icmp_seq=3 ttl=64 time=0.415 ms
520 bytes from 192.168.10.10: icmp_seq=4 ttl=64 time=0.406 ms
520 bytes from 192.168.10.10: icmp_seq=5 ttl=64 time=0.410 ms
520 bytes from 192.168.10.10: icmp_seq=6 ttl=64 time=0.432 ms
520 bytes from 192.168.10.10: icmp_seq=7 ttl=64 time=0.414 ms
520 bytes from 192.168.10.10: icmp_seq=8 ttl=64 time=0.428 ms
520 bytes from 192.168.10.10: icmp_seq=9 ttl=64 time=0.281 ms
520 bytes from 192.168.10.10: icmp_seq=10 ttl=64 time=0.549 ms
520 bytes from 192.168.10.10: icmp_seq=11 ttl=64 time=0.281 ms
520 bytes from 192.168.10.10: icmp_seq=12 ttl=64 time=0.403 ms
520 bytes from 192.168.10.10: icmp_seq=13 ttl=64 time=0.408 ms
520 bytes from 192.168.10.10: icmp_seq=14 ttl=64 time=0.418 ms
520 bytes from 192.168.10.10: icmp_seq=15 ttl=64 time=0.350 ms

--- 192.168.10.10 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 13999ms
rtt min/avg/max/mdev = 0.281/0.413/0.628/0.084 ms
```

Figure 21: Ping command

6.2 Measurement 2: Random data over TCP, UDP and RTP

To send data over UDP we first need to know how we need to send data over UDP and TCP. By comparing these, we know what protocol is best used for our application. When we compare the RTT of TCP, UDP and RTP we get the result that are shown in table 1.

Table 1: RTT over UDP and TCP while sending random data

	send 5 bytes			send 512 bytes			send 1024 bytes		
#	**TCP (ms)**	**UDP (ms)**	**RTP (ms)**	**TCP (ms)**	**UDP (ms)**	**RTP (ms)**	**TCP (ms)**	**UDP (ms)**	**RTP (ms)**
1	1.201	0.148	0.233	1.150	0.317	0.219	0.637	0.275	0.221
2	0.633	0.142	0.142	0.703	0.174	0.214	0.817	0.353	0.287
3	0.697	0.235	0.218	0.565	0.204	0.294	0.570	0.395	0.288
4	0.889	0.141	0.236	0.677	0.180	0.377	0.785	0.226	0.288
5	0.744	0.216	0.225	0.791	0.910	0.241	0.598	0.264	0.298
6	0.697	0.233	0.254	0.663	0.241	0.269	0.737	0.265	0.269
7	0.721	0.215	0.237	0.770	0.283	0.242	0.517	0.306	0.361
8	0.421	0.141	0.170	0.764	0.389	0.281	1.483	0.281	0.286
9	0.704	0.218	0.239	0.691	0.166	0.318	0.783	0.425	0.283
10	0.505	0.260	0.261	0.779	0.213	0.342	0.456	0.258	0.268
11	0.377	0.237	0.284	0.743	0.245	0.309	0.859	0.191	0.267
12	0.690	0.219	0.283	0.771	0.239	0.257	0.366	0.261	0.191
13	0.531	0.265	0.280	0.755	0.258	0.300	0.836	0.187	0.327
14	0.732	0.166	0.222	0.738	0.159	0.270	0.924	0.324	0.310
15	0.681	0.216	0.232	0.750	0.159	0.257	1.156	0.266	0.355
min.	0.377	0.141	0.142	0.565	0.159	0.214	0.366	0.187	0.191
max.	1.201	0.265	0.284	1.150	0.910	0.377	1.483	0.425	0.361
avg.	0.681	0.203	0.234	0.754	0.276	0.279	0.768	0.285	0.287
std. dev.	0.189	0.042	0.038	0.120	0.180	0.044	0.272	0.065	0.042

From table 1 we get an indication that UDP and RTP are better than TCP if we want to maintain the smallest latency.

6.2.1 TCP Round-Trip-Time

Note: To execute the following commands you need the code of the following directory which can be found in the portfolio: network/TCP/RTT/. The cross compiled code can also be found here. If you need to cross compile it again we refer to section 3.3. No extra arguments are needed.

Copy the file to EVM board 1:

```
# scp tcpserver root@192.168.10.10:/home/root
```

Connect to EVM board 1:

```
# ssh root@192.168.10.10
```

Make sure you run the server before running the client.

```
# ./tcpserver
```

Copy the file to EVM board 2:

```
# scp tcpclient root@192.168.10.11:/home/root
```

Now connect to EVM board 2 from the VM:

```
# ssh root@192.168.10.11
```

Execute the following command:

```
# ./tcpclient
```

Note: In the tcpclient.c is a static IP defined (server's IP). If the IP range is changed, you also need to change it here.

6.2.2 UDP Round-Trip-Time

Note: To execute the following commands, you need the code of the following directory which can be found in the portfolio: network/UDP/RTT/. The cross compiled code can also be found here. If you need to cross compile it again we refer to section 3.3. No extra arguments are needed.

Copy the file to EVM board 1:

```
# scp udpserver root@192.168.10.10:/home/root
```

Connect to EVM board 1:

```
# ssh root@192.168.10.10
```

Make sure you run the server before running the client.

```
# ./udpserver
```

Copy the file to EVM board 2:

```
# scp udpclient root@192.168.10.11:/home/root
```

Now connect to EVM board 2 from the VM:

```
# ssh root@192.168.10.11
```

Execute the following command:

```
# ./udpclient
```

Note: In the udpclient.c is a static IP defined (server's IP). If the IP range is changed, you also need to change it here.

6.2.3 RTP Round-Trip-Time

Note: To execute the following commands, you need the code of the following directory which can be found in the portfolio: network/RTP/RTT/. The cross compiled code can also be found here. If you need to cross compile it again we refer to section 3.3. No extra arguments are needed.

Copy the file to EVM board 1:

```
# scp rtpserver root@192.168.10.10:/home/root
```

Connect to EVM board 1:

```
# ssh root@192.168.10.10
```

Make sure you run the server before running the client.

```
# ./rtpserver
```

Copy the file to EVM board 2:

```
# scp rtpclient root@192.168.10.11:/home/root
```

Now connect to EVM board 2 from the VM:

```
# ssh root@192.168.10.11
```

Execute the following command:

```
# ./rtpclient
```

Note: In the rtpclient.c is a static IP defined (server's IP). If the IP range is changed, you also need to change it here.

6.3 Audio transfer with latency file over UDP

Note: To execute the following commands you need the code of the following directory which can be found in the portfolio: network/UDP_with_latency/. The cross compiled code can also be found here. If you need to cross compile it again we refer to section 3.3. Note that we need to add the "-lasound" parameter.

Copy the file to EVM board 1:

```
# scp latency_udp_receiver root@192.168.10.11:/home/root
```

Connect to EVM board 1:

```
# ssh root@192.168.10.11
```

Make sure you run the server before running the client.

```
# ./latency_udp_receiver -m 256 -p
```

Copy the file to EVM board 2:

```
# scp latency_udp_sender root@192.168.10.10:/home/root
```

Now connect to EVM board 2 from the VM:

```
# ssh root@192.168.10.10
```

Execute the following command.

```
# ./latency_udp_sender -m 256 -p
```

Note: In the latency_udp_sender.c is a static IP defined (server's IP). If the IP range is changed, you also need to change it here. You also need to connect the two boards with an Ethernet switch and add a microphone to the sender and a speaker to receiver board.

6.4 Audio transfer with latency file over RTP

Note: To execute the following commands you need the code of the following directory which can be found in the portfolio: network/RTP_with_latency/. The cross compiled code can also be found here. If you need to cross compile it again we refer to Section 3.3. Note that we need to add the "lasound" parameter.

Copy the file to EVM board 1:

```
# scp latency_rtp_receiver root@192.168.10.11:/home/root
```

Connect to EVM board 1:

```
# ssh root@192.168.10.11
```

Make sure you run the server before running the client.

```
# ./latency_rtp_receiver -m 256 -p
```

Copy the file to EVM board 2:

```
# scp latency_rtp_sender root@192.168.10.10:/home/root
```

Now connect to EVM board 2 from the VM:

```
# ssh root@192.168.10.10
```

Execute the following command.

```
# ./latency_rtp_sender -m 256 -p
```

Note: In the latency_rtp_sender.c is a static IP defined (server's IP). If the IP range is changed, you also need to change it here. You also need to connect the two boards with an Ethernet switch and add a microphone to the sender and a speaker to receiver board.

7 List of references

- `http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_Getting_Started_Guid`
- `http://www.alsa-project.org/alsa-doc/alsa-lib/_2test_2latency_8c-example.html`

- https://www.ietf.org/rfc/rfc768.txt
- https://tools.ietf.org/html/rfc793
- https://www.ietf.org/rfc/rfc3550.txt
- https://www.ietf.org/rfc/rfc3551.txt
- https://www.cs.cmu.edu/afs/cs/academic/class/15213-f99/www/class26/
- https://www.stresslinux.org/sl/
- http://www.hecticgeek.com/2012/11/stress-test-your-ubuntu-computer-with-stre