Virtual ALSA Device Simulator (VADS)

# Introduction

Virtual ALSA Device Simulator (VADS) can create audio devices, which exist in user space. VADS relies on SmartX, which instantiates the devices. Device configuration is stored in an XML file.

# Architecture

The generic architecture is shown below:



There are three main components in VADS:

* Devs - the Dev Manager is a container for all available devices. It handles parsing XML file, which contains device descriptions. Dev Manager stores devices in two groups: sources and sinks.
* Buffer managers - contain a buffer reader and writer. Reader empties the device buffer, whilst Writer delivers samples to the buffer.
* Files - the file’s location is configurable in the XML configuration. Reader saves the samples in the Reader sink and Writer gets samples from the Writer source. It’s the user responsibility to prepare the contents of the Writer source.

# XML configuration

The xml structure is shown below:

<?xml version="1.0"?>

<vads>

<version>

<oemId>INTEL</oemId>

<oemRevision>1</oemRevision>

</version>

<devices>

<dev>

<id>0</id>

<type>playback</type>

<name>vsink0</name>

<format>32</format>

<frequency>48000</frequency>

<channelNo>2</channelNo>

<clock>slave</clock>

<periodNo>4</periodNo>

<periodSize>192</periodSize>

<mode>blocking</mode>

<file>/tmp/vsink0.wav</file>

<timing>

<tick prefix="us">4000</tick>

<sequence>

<ticksNumber drift="10">500</ticksNumber><!--step 1-->

<ticksNumber drift="-15">80</ticksNumber><!--step 2-->

<ticksEmpty>1000</ticksEmpty><!-- step 3 -->

<ticksNumber drift="0">500</ticksNumber><!--step 4-->

</sequence>

</timing>

</dev>

<dev>

><!—- another device -->

</dev>

</devices>

</vads>

Every device contains generic audio information and a timing section. The timing section defines a main tick in microseconds (possible improvement here: milliseconds option, see [TODO](#_TODO) section) and a sequence of steps to execute. There are 3 possible options to define in the sequence:

* Drifted tick, where there is a time distortion introduced to the clear tick
* Clear (non-drifted) tick, where drift is zero
* Empty tick, where no data is delivered/fetch to/from a buffer

The sequence is will be executed in the order defined by user. In the example above the main tick will be triggered every 4000 µs (4 ms). The corresponding sequence will be executed:

* Step 1: 500 ticks with a drift 10 µs. That means there will be 500 ticks every 4010 µs executed.
* Step 2: 80 ticks with a drift -15 µs. Means there will be 80 ticks every 3985 µs.
* Step 3: 1000 empty ticks. That means for 1000 ticks there will be no buffer read/write.
* Last step: 500 clear ticks. That means 500 ticks will be executed with no drift.
* After last step program will start executing Step 1 again (possible improvement here: add number of iterations for the sequence, see [TODO](#_TODO) section)

# Building

VADS is dependent on SmartX. To build the application one needs to build SmartX. The best known method to build and run SmartX is the following:

* Setup Yocto
* Sync Yocto with particular release that needs to be used on the target:

repo sync --force-sync -m <path\_to\_collaterals/\*.xml>

* Populate SDK based on the synced Yocto
* Build SmartX using that SDK:

<path\_to>/ias\_set\_build\_dir.bash --sdk=<path\_to>/ias-kc-ref-image\_gr-mrb-64.cmake> <build\_dir>

* Flash the target with the chosen release + SmartX binaries

This method guarantees no library dependencies problem for SmartX. To build VADS:

* Checkout the latest code ([ssh:// kavgit001.ka.intel.com:29418/target/audio/alsa\_simulator](ssh://%20kavgit001.ka.intel.com:29418/target/audio/alsa_simulator))
* Edit setup.sh script and set path to the SDK used to build SmartX- this is necessary as VADS is dependent on SmartX
* Edit Makefile and set path to the SDK and SmartX libraries.
* Save the file and do source ./setup.sh
* Do make

# Usage:

* Upload VADS binary to the target.
* Prepare your XML device configuration. Alternatively you can use the default.xml, which contains two sink and one source.
* Start the executable and pass a path to the XML: ./vads –c <path\_to\_xml>
* After parsing the XML device configuration, VADS prints usage, for example:

Sink usage: aplay -Dplug:smartx:vsink1 <file>, output: /tmp/vsink1.wav

* Open another terminal and do:

aplay -Dplug:smartx:vsink0 -c2 -fS32\_LE /opt/audio/gpmrb/alsa\_simulator/wav/r48\_c2\_f32.wav -vvv

Samples should start flowing from aplay source to the VADS sink.

Note: do not confuse the file definition in XML with the file name passed to aplay/arecord. For a virtual sink the XML file acts as an output for the buffer reader while the file passed to aplay is the source of the samples. Conversely for a virtual source- the XML definition is an input for the buffer writer while the file passed to arecord is the sink for the samples.

# TODO

Some future improvements below:

* Optional XML attributes - at the moment when user doesn’t define any of the XML tree there will be an exception thrown. It would be better to have some flexibility here and more forgiving parsing.
* Introduce milliseconds both in XML and parser. At the moment user can only define micro-seconds.
* Define in sequence number of iterations. Currently every sequence is executed in an infinite loop - it might be a good idea to let user define number of iterations.
* Create a logger instead of printing (for someone who doesn’t want to connect DLT)