HOCHSCHULE FURTWANGEN UNIVERSITY HFU

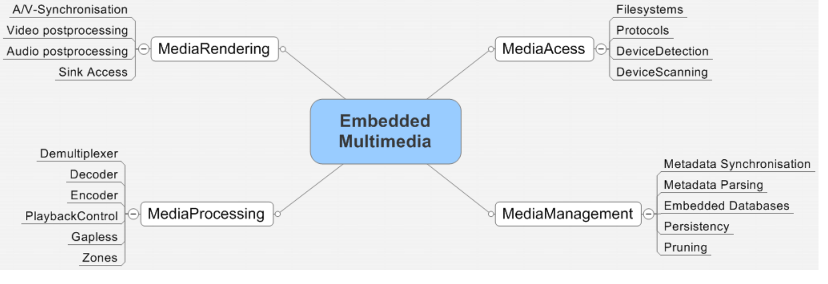
Lab „Platforms for Embedded Systems“
Chapter 03: Audio

Prof. Dr. Elmar Cochlovius

- **Goals of this Chapter 03:**
  - Example of an application domain for embedded plattforms: „Multimedia“
  - Core functionalities of Mobile Multimedia Systems (MMS)
  - The basics: sampling rate and bit-width
  - The mathematics: creating our own sound waves
  - Multimedia UseCases and Filtergraph Architectures
  - Example: GStreamer

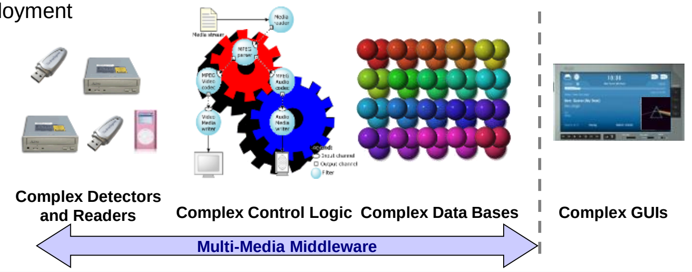
- **Overview**
  - Basic Aspects of Mobile Multimedia Systems (MMS)
  - How to calculate and create your own WAV-files
  - Cross-plattform Playback
  - Software-Architecture: Some experiments using filtergraphcs and middleware on host AND target

- **Core Functions of Mobile Multimedia-Systems:**



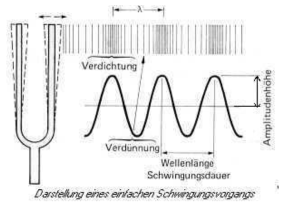
Requirements of Multimedia-Systems:

- Plattform-independent Software-Architecture:
  - Scalability across functions (e.g. audio-only, audio/video, premium)
  - Scalability across plattforms (e.g. x86, ARM, MIPS, DSP, SOC)
- Product-proven approach: multimedia middleware
  - Component-based multimedia solution to reduce recurring development efforts
  - Provides MM-applications and –systems based on various plattforms
- Core topics include: Media-access, -management, -processing and -deployment



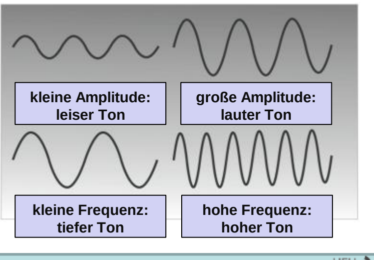
Digital Audio: Basics

- Physical domain: Audio consists of sound-waves, i.e. periodic oscillations of air pressure
- Created by sound transducers, i.e. Speakers („audio sink“)



Darstellung eines einfachen Schwingungsvorgangs

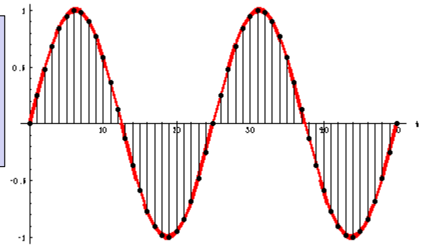
- We notice different audio oscillations as tones or noise



- **Digital Audio: Basics**
  - Electric domain: periodic oscillations of electical current at the speaker
  - These oscillations can be describes (math.) as (overlay of) various sine waves
  - But how do we enter the digital domain?
  - Answer: We need to digitize the analoge functions by means of **sampling:**

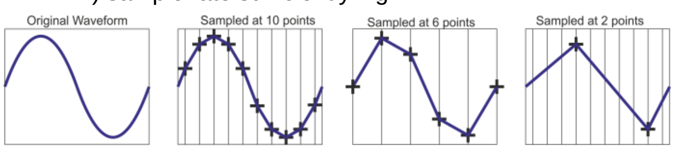
Bit-depth: Resolution of each sample

→ may lead to quantization errors

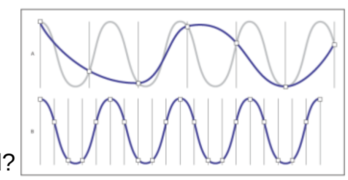


Sample rate: number of samples per sec.

- **Digital Audio: Basics**
  - Two preconditions for good audio quality (I/II):
    - 1) sample rate sufficiently high

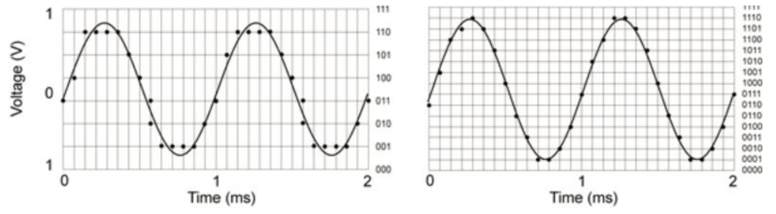


    - **Sampling Theorem**:
      - The sample rate has to be at least 2 x of the max. frequency of the signal to be sampled.
      - Otherwise, reconstruction of the original signal will lead to errors!
      - e.g. original signal (grey) can NOT be reconstructed
      - What sample rate is required?



03: Multimedia - Audio

- **Digitales Audio: Basics**
  - Two preconditions for good audio quality (I/II):
    - 2) bit depth sufficiently high



    - Which bit depth is required?

03: Multimedia - Audio

- **Overview**
  - Basic Aspects of Mobile Multimedia Systems (MMS)
  - How to compute and create your own WAV-files
  - Cross-plattform Playback
  - Software-Architecture: Some experiments using filtergraphcs and middleware on host AND target

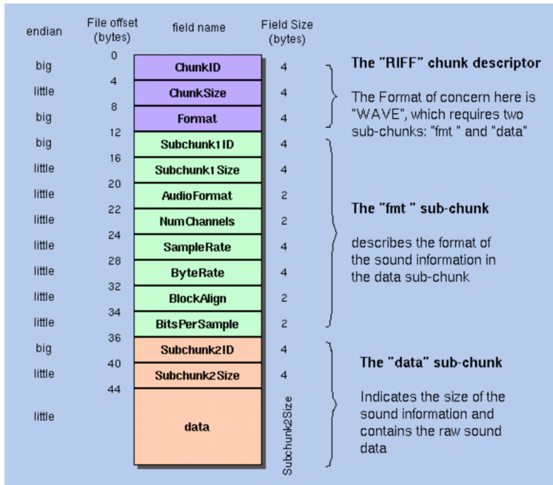
Digital Audio: The Major Steps

- **Digitizing:** Transforming continous signals into discrete values by sampling and quantization
  - Result: sequence of samples (PCM, LPCM, WAV-Format)
- **Coding:** Compressing the uncompressed samples by:
  - Eliminating redundancy
  - Exploiting psych-accoustic effects (e.g. hiding, shadowing)
  - Using an encoder
  - Result: file or stream of encoded data, e.g. in mp3, wma, aac, ogg format
- **Decoding:** „unwrapping" the compressed data by means of a decoder („codec")
  - Result: digitized signal (PCM, LPM, WAV), i.e. approximation of original sequence of samples.

Digital Audio: WAVE-Files

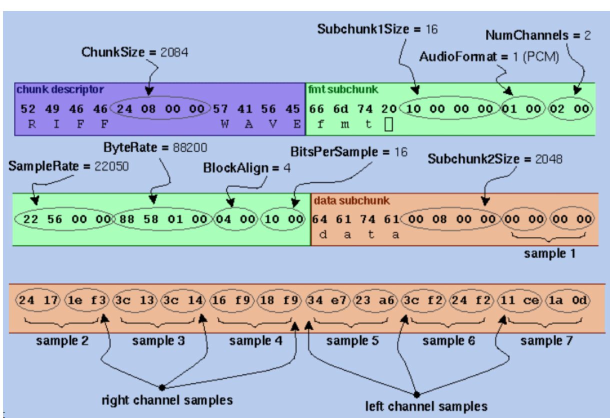
- Next step: creating our own WAV-file to represent a tone
- Problem: raw sequence of digital values is NOT very helpful → we need add. metadata information for correct „interpretation" of the values by our player
- Metadata might include:
  - Number of channels
  - Sample Rate
  - Byte Rate (= SampleRate * NumChannels * Bit depth/8)
  - BitsPerSample (bit depth)
- The WAV-format provides various segments („chunks") to keep this metainformation.

- **Digital Audio: WAVE-Files**
  - Generic WAVE format (canonical WAVE)



[Ref.: https://ccrma.stanford.edu/courses/422/projects/WaveFormat]

- **Digital Audio: Example of WAVE-File**



[Ref.: https://ccrma.stanford.edu/courses/422/projects/WaveFormat]

- **Digital Audio: WAVE-Files**
  - What else is required:
  - A library to simplify the creation and formatting of the WAV file and to take care of the header information:
    - Make sure to install `libsndfile1` and its developer library plus header files `libsndfile1-dev`, as well as some utilitiy programs `sndfile-programs` on **host and target.**
  - Install an Hex-editor to control the result, e.g. `hexer`
  - An audio player to render the digital samples into an audio sink (e.g. `/dev/snd`) → `aplay` or `sndfile-play`
  - A code template to get started with the basics
    - In `~coe/LabPMS/Res/QuickStart/07`, copy the file wav_writer.cc and create a new project in Eclipse.

- **Digital Audio: WAVE-Files**
  - Now, we are ready for:
    - **Exercise 07 –**
      Digital Audio using WAVE-Files

Digital Audio: Target Libraries on the HOST (1/3)

- **Current Problem:**
  - Cross-compilation of `sndfile` programs on the host is NOT possible
  - Reason: on the host we have required libs and headers of `sndfile`, but NOT in ARM format → Cross Compiler will fail because of missing headers, Cross Linker will fail because of missing library
- **Naive Solution:**
  - Install all libraries on the host in 2 flavors (1st instance native, 2nd instance for the target)
  - Note: for professional work environments, this is NOT a viable solution. Why not?

Digital Audio: Target Libraries on the HOST (2/3)

- **Alternative Solution (preferred):**
  1. We provide the complete root filesystem of the target to the host
  2. We set the `#include` paths in Eclipse
  3. We access the target libraries for linking the code
- **How can we get this accomplished (1/2)?**
  - On the target, we have to export the root directory „/“ to the host using `exportfs`
  - For this, make sure that the packet `nfs-kernel-server` is installed on the target and is running
  - On the host, mount this directory on `/mnt/rootfs` using NFS
  - In Eclipse: …

Digital Audio: Target Libraries on the HOST (3/3)

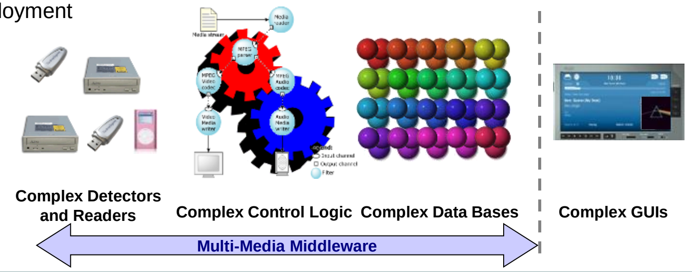
- **Preparations inside Eclipse (2/2):**
  1. In Project Properties → C/C++ Build→ Settings add the Standard include path to Cross GCC → Includes
  2. In Cross G++ Linker → Misc under "Linker Flag": set sysroot-Option to the importet root directory using: `--sysroot=/mnt/rootfs`
  3. As usual: list the required lib (here: `sndfile`) in the Libraries field

Digital Audio: WAVE Files

- Now we should work on:
  - **Exercise 08 – „Instead of copying: cross-compiling!"**

- **Overview**
  - Basic Aspects of Mobile Multimedia Systems (MMS)
  - How to calculate and create your own WAV-files
  - Cross-plattform Playback
  - Software-Architecture: Some experiments using filtergraphs and middleware on host AND target

- **Recap: „Requirements of Multimedia-Systems“**
  - Plattform-independent Software-Architecture:
    - Scalability across functions (e.g. audio-only, audio/video, premium)
    - Scalability across plattforms (e.g. x86, ARM, MIPS, DSP, SOC)
  - Product-proven approach: multimedia middleware
    - Component-based multimedia solution to reduce recurring development efforts
    - Provides MM-applications and –systems based on various plattforms
  - Core topics include: Media-access, -management, -processing and -deployment



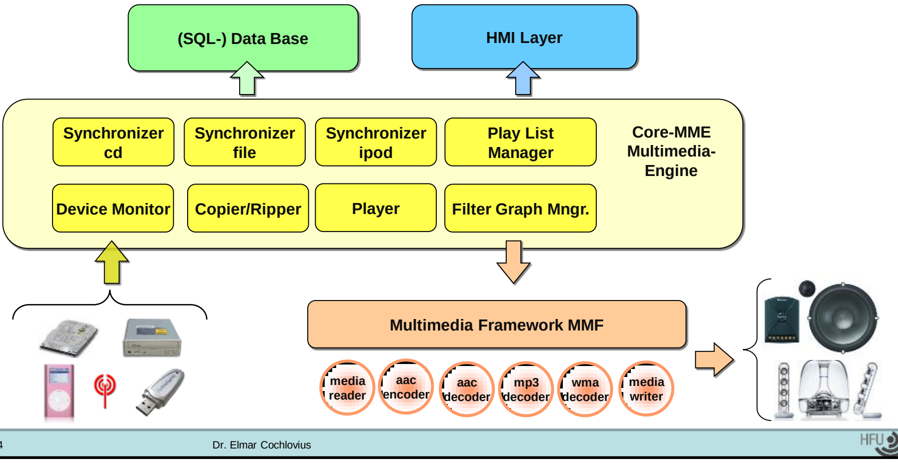
GStreamer: Example of a Multimedia Architecture

- Benefits:
  - Based on flexible software concept of filter graphs
  - OSSW and available on many platforms, incl. X86 and ARM
- Drawbacks:
  - Some features required for commercial/embedded applications are overly simplistic or not possible at all
  - Some specific functionalities are still instable
- Overview:

Applications	AudioPlayer	StreamingServer	VideoPlayer	Ripper	Synchronizer
Framework	GStreamer				
Plugins	MediaReader	Streamer	Codecs	Postprocessing	MediaWriter

Commercial Alternative: MultiMedia Engine MME

- Small footprint, highly efficient, very customizable, stable
- Used e.g. in many automotive projects (e.g. Audi, BMW, Daimler)

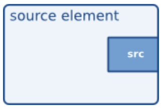


Multimedia Architecture: Filter Graphs

- Which Types of filters are required?
- **Source / Sink:**
  - reader, streamer, writer (renderer)
- **Analyze and Control:**
  - Parser
  - Demultiplexer
  - Navigator
- **Processing:**
  - Decoder
  - Encoder
  - Mixer
  - A/V Synchronizer
  - Queue
  - Sample Rate Converter

Multimedia Architecture: Filter Graphs (GStreamer)

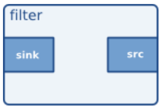
- Source Element (used for data providers):



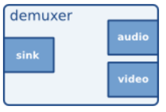
- Sink Element (used for data consumers):

Multimedia Architecture: Filter Graphs (GStreamer)

- Generic Filter element:
  - 1 Input-, 1-Output connection („Pad“)
  - processes / transforms data, which are received on the input pad
  - Generates output data

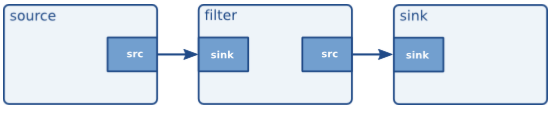


- Also possible: >1 Output Pad („Demultiplexer“)

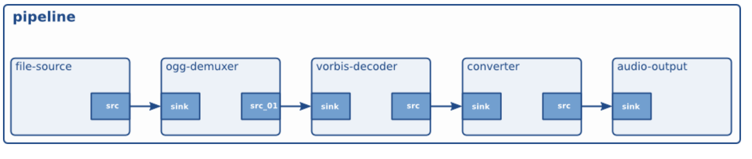


Multimedia Architecture: Filter Graphs (GStreamer)

- Next: simple Filter Graph
  - implements single, specific use case
  - Constructed dynamically at run time by instantiating and connecting required filter elements („nodes“) → „filter pipeline“

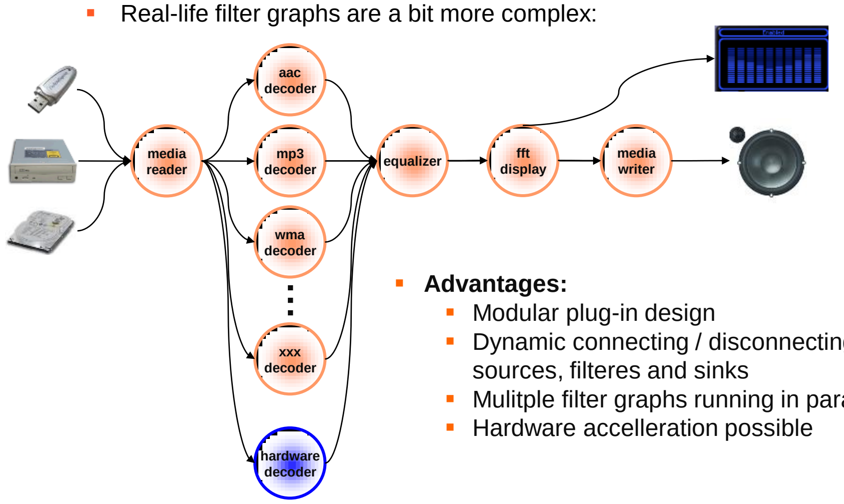


  - „Hello World“ filter graph for a simple Ogg/Vorbis player:



Multimedia Architecture: Filter Graphs (MME)

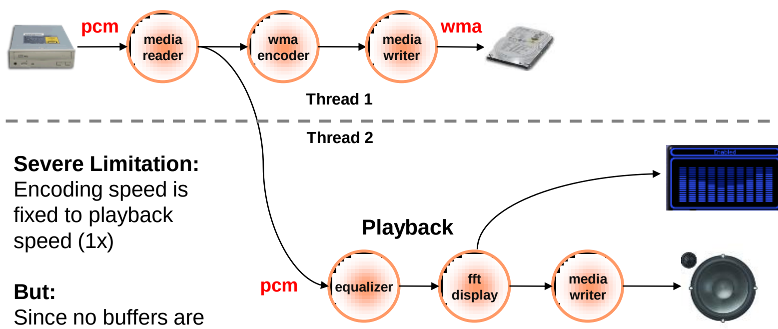
- Real-life filter graphs are a bit more complex:



- **Advantages:**
  - Modular plug-in design
  - Dynamic connecting / disconnecting sources, filteres and sinks
  - Mulitple filter graphs running in parallel
  - Hardware accelleration possible

Multimedia Architecture: Realtime Encoding

- What are the filters required for Encoding / Transcoding / Ripping of an audio source?

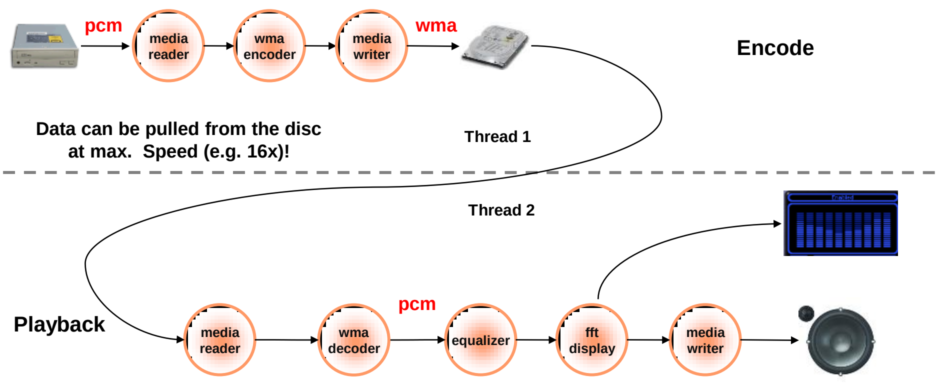


Severe Limitation:
Encoding speed is fixed to playback speed (1x)

But:
Since no buffers are available, encoding must(!) be at factor 1x

Multimedia Architecture: Realtime Encoding (2)

- Improving user experience by „Look-Ahead Encoding“ (Rip & Play)



Installation:

- To get started, we need some multimedia libraries. Pls. install on host AND target:

```
▪ libgstreamer0.10
▪ libgstreamer0.10-dev
▪ gstreamer0.10-plugins-good, -bad, -ugly (Codecs
  to play mp3,etc.)
▪ gstreamer0.10-tools
```

- Check your installation e.g. with:

```
gst-inspect-0.10 | grep mp3
```

- and verify, that you have mp3 and ogg codecs.

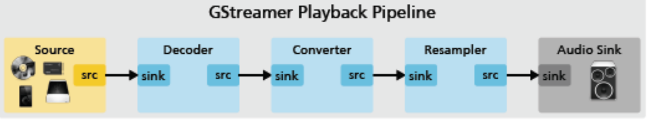
03: Multimedia - Audio

- **Multimedia with GStreamer:**
  - **Next step:** create and exercise your own filter graphs on host AND target using the Gstreamer framework
  - What is required:
  - First audio test using a built-in test source generating a constant sine wave
  - Hint: `gst-launch` is a command line interface (CLI), used to create filter graphs „on the fly“. „!“ is the operator to connect filters together (**all on one line!!**).

```
gst-launch-0.10 audiotestsrc ! audioconvert !
    audioresample ! autoaudiosink
```


03: Multimedia - Audio

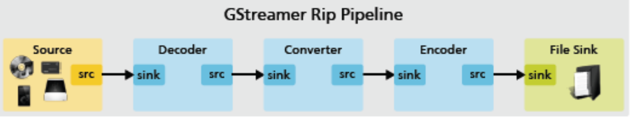
- **Multimedia with GStreamer:**
  - Now we want to play-back a „real“ audio file, e.g. using the public domain OGG/VORBIS format
  - We need the following filter graph:



  - The filter graph is constructed usind `gst-launch` CLI (**note**: one line or use „\“ as line separator)

```
gst-launch-0.10 filesrc location=<my_file.ogg> !
    oggdemux ! vorbisdec ! audioconvert !
    audioresample ! autoaudiosink
```

03: Multimedia - Audio

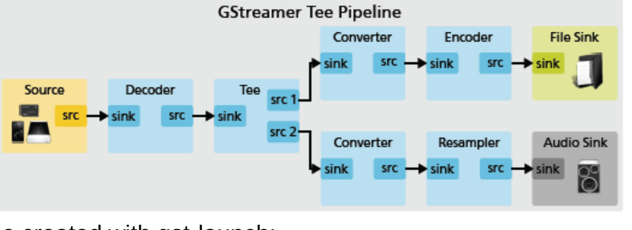
- **Multimedia with GStreamer:**
  - What about ripping of audio content?
  - Here we need the following filter graph:
  - 

  - Again: use gst-launch to create it via CLI:

```
gst-launch-0.10 filesrc location=<my_file.mp3> !
    decodebin ! audioconvert ! vorbisenc !
    oggmux ! filesink location=<my_file.ogg>
```


03: Multimedia - Audio

- **Multimedia with GStreamer:**
  - But what about „Rip & Play“ i.e. ripping and playback IN PARALLEL?
  - Here we need a more complex filter graph using a T-filter

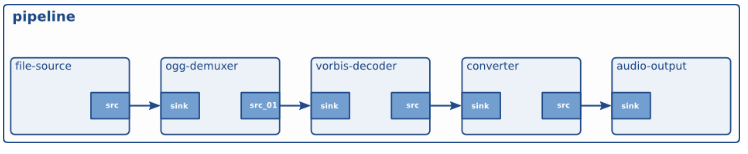


  - To be created with gst-launch:

```
gst-launch-0.10 filesrc location=<my_file.mp3> !
decodebin ! tee name=t ! queue ! audioconvert !
vorbisenc ! oggmux ! filesink
location=<my_file.ogg> t. ! queue !
audioconvert ! audioresample ! autoaudiosink
```

- **Digital Audio: Filter graphs**
  - Now we can work on:
    - **Exercise 09, parts 9.1 – 9.5:**
      Audio UseCases using Gstreamer
      Filter Graphs and gst-launch CLI

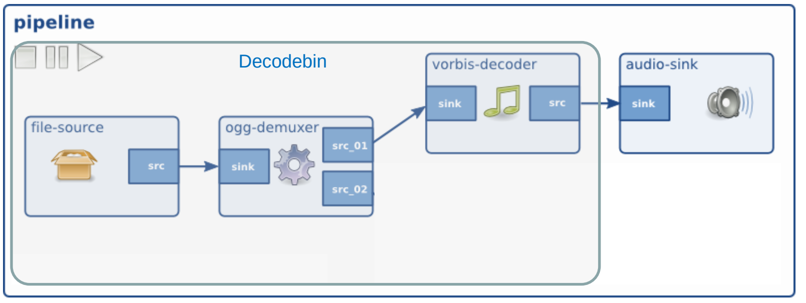
- **Multimedia wih Gstreamer – C-Program:**
  - **CLI using gst-lauch:** is o.k. for quick experiments, but
    NOT suitable for „real-life“ multimedia implementations
  - We need to program our own filter graph for audio playback as a **stand-alone C program.**
  - Simple playback pipeline (a bit too simplistic!):



  - Severe limitations:
    - Only single Format (ogg/vorbis audio files) possible
    - No extensions for playback of video files
  - Alternative approach:
    - Use general-purpose decoder `uridecodebin` instead of vorbis decoder

03: Multimedia - Audio

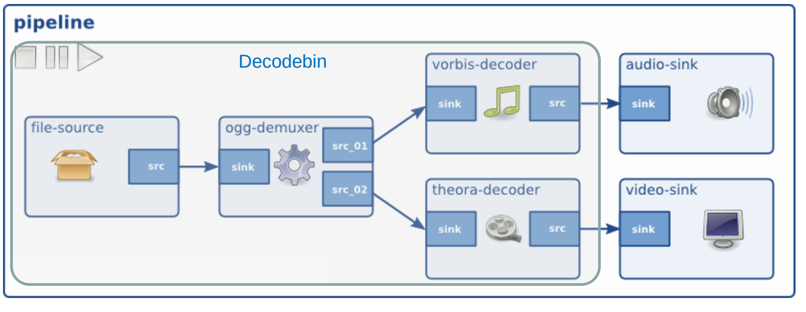
- **Multimedia with Gstreamer – C-Program:**
  - Abstract filter graph of a general-purpose audio player:



  - **Note:**
    - The **`uridecodebin`** filter also contains decoder for wav, mp3, etc.
    - It expects a URI as a parameter, e.g.
      `file:///home/pi/Music/cp.ogg`

03: Multimedia - Audio

- **Multimedia with Gstreamer – C-Program:**
  - **Extension:** „dynamic output pads“, e.g. `src_01, src_2`
    - Depending on currently decoded content, the **`uridecodebin`** requires different successor filters
    - I.e. these can be connected only dynamically during at decode time.
  - **Example: Video Content**



  - **Gstreamer Solution:** use callback mechanism for dynamically connecting filters to the graph.

03: Multimedia - Audio

- **Multimedia with Gstreamer – C-Program:**
  - The following steps are required in our program:
    1. Create data structure to contain all filter elements, i.e. pointer to filters
    2. Create all filter elements required
    3. Connect static filter elements as possible („linking“)
    4. Handover URI as parameter to the `uridecodebin`
    5. Register callback function which gets called automatically, whenever Gstreamer creates a new dynamic pad
    6. Define Callback function including:
    7. Checking the type of the newly created dynamic pad
    8. If type == audio, then link the audio pipeline
    9. Else: ignore the pad
    10. Set the state of the filter graph to PLAYING
    11. Listen to (error) messages and react if required

03: Multimedia - Audio

- **Multimedia with Gstreamer – C-Program:**
  - Walk through of the program (1)
  1. Create data structure to contain all filter elements, i.e. pointer to filters

```
/* Structure to contain all our information, so we can pass it to callbacks */
typedef struct _CustomData {
  GstElement *pipeline;
  GstElement *source;
  GstElement *convert;
  GstElement *sink;
} CustomData;
```

  2. Create all filter elements required

```
/* Create the elements */
data.source = gst_element_factory_make ("uridecodebin", "source");
data.convert = gst_element_factory_make ("audioconvert", "convert");
data.sink = gst_element_factory_make ("autoaudiosink", "sink");
```

Multimedia with Gstreamer – C-Program:

- Walk through of the program (2)

3. Connect static filter elements as possible („linking“)

```
if (!gst_element_link (data.convert, data.sink)) {
  g_printerr ("Elements could not be linked.\n");
  gst_object_unref (data.pipeline);
  return -1;
}
```

4. Handover URI as parameter to the `uridecodebin`

```
/* Set the URI to play */
g_object_set (data.source, "uri", argv[1], NULL);
```

5. Register callback function which gets called automatically, whenever Gstreamer creates a new dynamic pad

```
/* Connect to the pad-added signal */
g_signal_connect (data.source, "pad-added", G_CALLBACK (pad_added_handler),
                                                        &data);
```


Multimedia with Gstreamer – C-Program:

- Walk through of the program (3)

6. Define Callback function:

```
/* Callback: new pad has been dynamically added */
static void pad_added_handler (GstElement *src, GstPad *new_pad,
                               CustomData *data) {
```

7. Checking the type of the newly created dynamic pad

```
/* Check the new pad's type */
new_pad_caps = gst_pad_get_caps (new_pad);
new_pad_struct = gst_caps_get_structure (new_pad_caps, 0);
new_pad_type = gst_structure_get_name (new_pad_struct);
if (!g_str_has_prefix (new_pad_type, "audio/x-raw")) {
  g_print ("  It has type '%s' which is not raw audio. Ignoring...\n",
                                          new_pad_type);
  goto exit;
}
```

03: Multimedia - Audio

- **Multimedia with Gstreamer – C-Program:**
  - Walk through of the program (4)

8. If type == audio, then link the audio pipeline

```
/* Attempt the link */
ret = gst_pad_link (new_pad, sink_pad);
if (GST_PAD_LINK_FAILED (ret)) {
  g_print ("  Type is '%s' but link failed.\n", new_pad_type);
} else {
  g_print ("  Link succeeded (type '%s').\n", new_pad_type);
}
```

9. Else: ignore the pad
10. Set the state of the filter graph to PLAYING

```
 /* Start playing */
  ret = gst_element_set_state (data.pipeline, GST_STATE_PLAYING);
```

11. Listen to (error) messages and react if required…

03: Multimedia - Audio

- **Digital Audio: Gstreamer Program**
  - Now it is time for:
    - **Exercise 09, part 9.6 + 9.7:**
      Audio Player with Gstreamer
      filter graphs as stand-alone C-program

Multimedia with GStreamer: Audio Streaming with RTP

- **On the Host („Audio Input“):**
  - alsasrc (Microphone) -> … -> encoder (Speex) -> … -> udpsink

```
gst-launch-0.10 -v alsasrc ! audioconvert ! audioresample !
   'audio/x-raw-int,rate=8000,width=16,channels=1' ! speexenc
   ! rtpspeexpay ! udpsink host=„IP-Adr“ port=6666
```

- **On the Target („Audio Output“):**
  - udpsrc (using „capabilities“) -> rtpjitterbuffer -> … -> decoder (speex) \
    tee t -> audioconvert -> playback as usual \
    t -> … -> waveenc -> filesink (save as WAV file)

```
gst-launch-0.10 udpsrc port=6666 caps="application/x-rtp,
   media=(string)audio, clock-rate=(int)16000, encoding-
   name=(string)SPEEX, encoding-params=(string)1,
   payload=(int)110" ! gstrtpjitterbuffer ! rtpspeexdepay !
   speexdec ! tee name=t ! queue ! audioconvert !
   audioresample ! autoaudiosink t. ! queue ! audioconvert !
   audioresample ! wavenc ! filesink location=myfile1.wav
```


Summary of Chapter 03:

- Example of an application domain for embedded plattforms: „Multimedia“
- Core functionalities of Mobile Multimedia Systems (MMS)
- The basics: sampling rate and bit-width
- The mathematics: creating our own sound waves
- Multimedia UseCases and Filtergraph Architectures
- Example: GStreamer