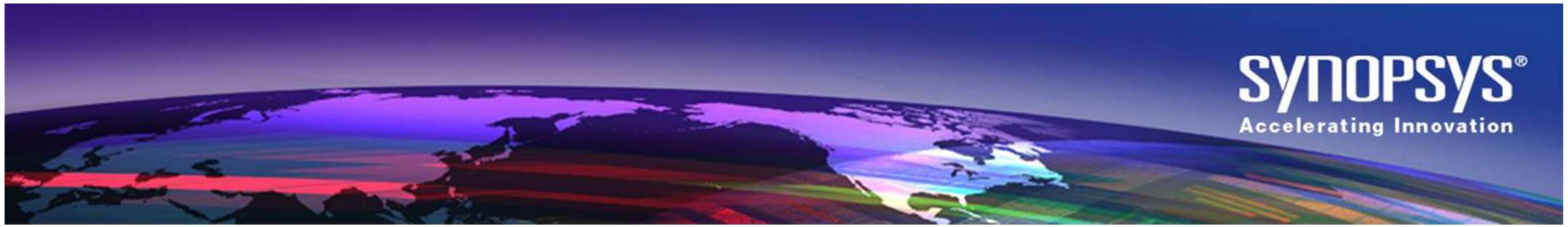


Using GStreamer for Seamless Off-Loading Audio Processing to a DSP

ELC 2013, San Francisco

Ruud Derwig

Abstract

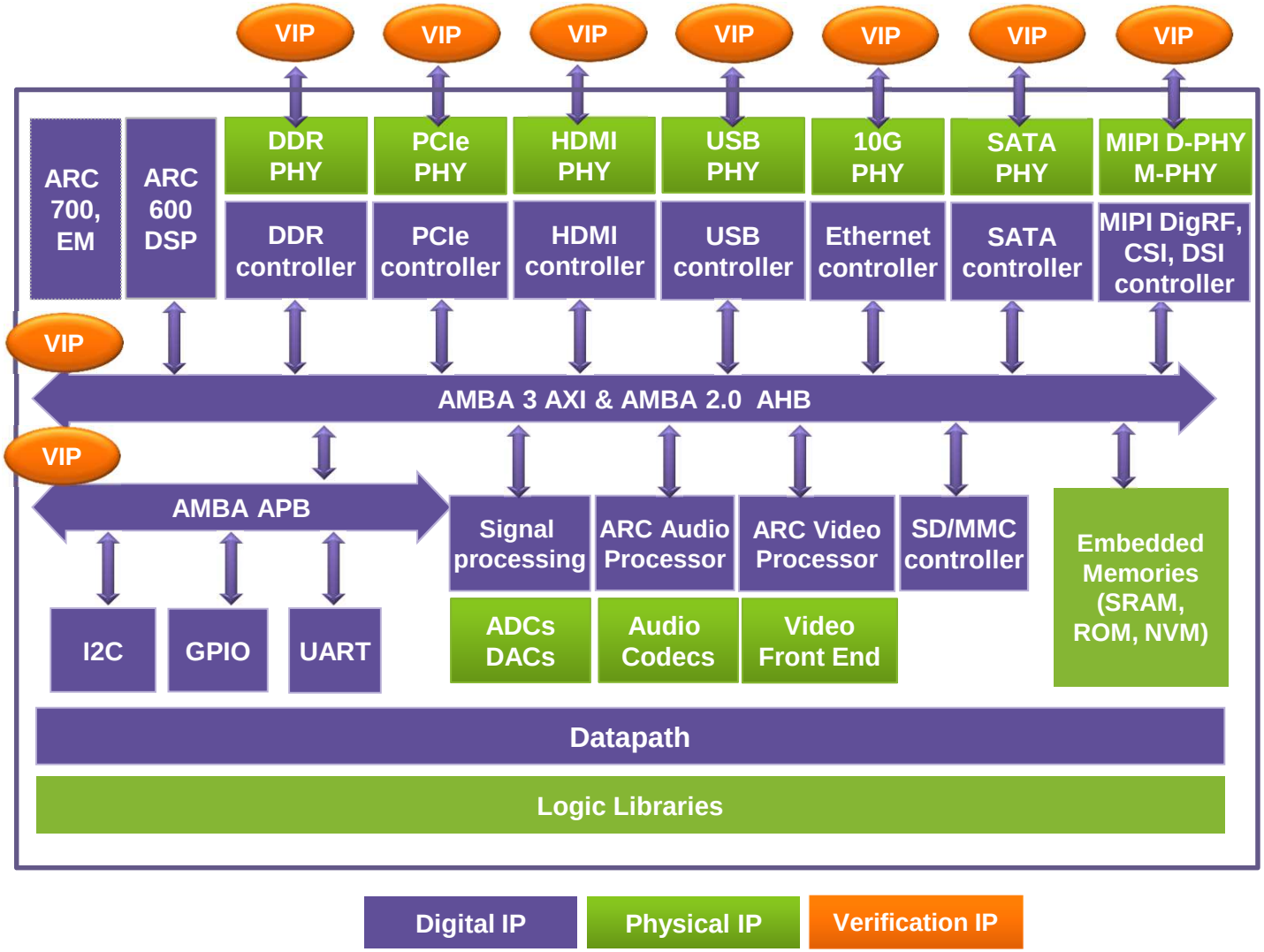
This presentation explains how off-loading of audio processing from an application processor to an audio DSP can be made easy using GStreamer. Despite the ridiculously high compute power of modern multi-core application processors, the SoC design trend remains towards heterogeneous architectures with specialized subsystems. For power efficiency and hardware cost such heterogeneous architectures are optimal, for software developers they are a pain. Whereas in the homogenous, SMP-Linux case most complexities are hidden, in the heterogeneous case developers must deal with different tools, shared memory (including cache coherency), multiple OSes, optimization of DSP code, and more.

Solutions like remoteproc are a good first step in simplifying the use of the different cores found on a modern SoC. In this presentation the basic management and control is taken a step further by leveraging the domain specifics of audio processing. Most of the complexities of audio off-loading can be hidden inside GStreamer elements, while retaining the flexible, plug-and-play processing graph creation of GStreamer.

Agenda

- Introduction
  - Synopsys, Audio Processing in CE devices, GStreamer
- Problem statement & Solution Sketch
  - Heterogeneous multi-core hardware for efficiency … but how about the SW?
- The Details!
  - GStreamer
  - Plug-in with elements that off-load processing to DSP
- Demo
- Conclusions, Q&A

DesignWare IP Portfolio



Established Provider

~$236M in Revenue
Second Largest IP Vendor*

Committed to Your Success

~1400 IP Engineers Worldwide

Trusted IP Supplier

#1 in Interface, Analog, Embedded Memories

Digital IP | Physical IP | Verification IP

*Source: Gartner, March 2012

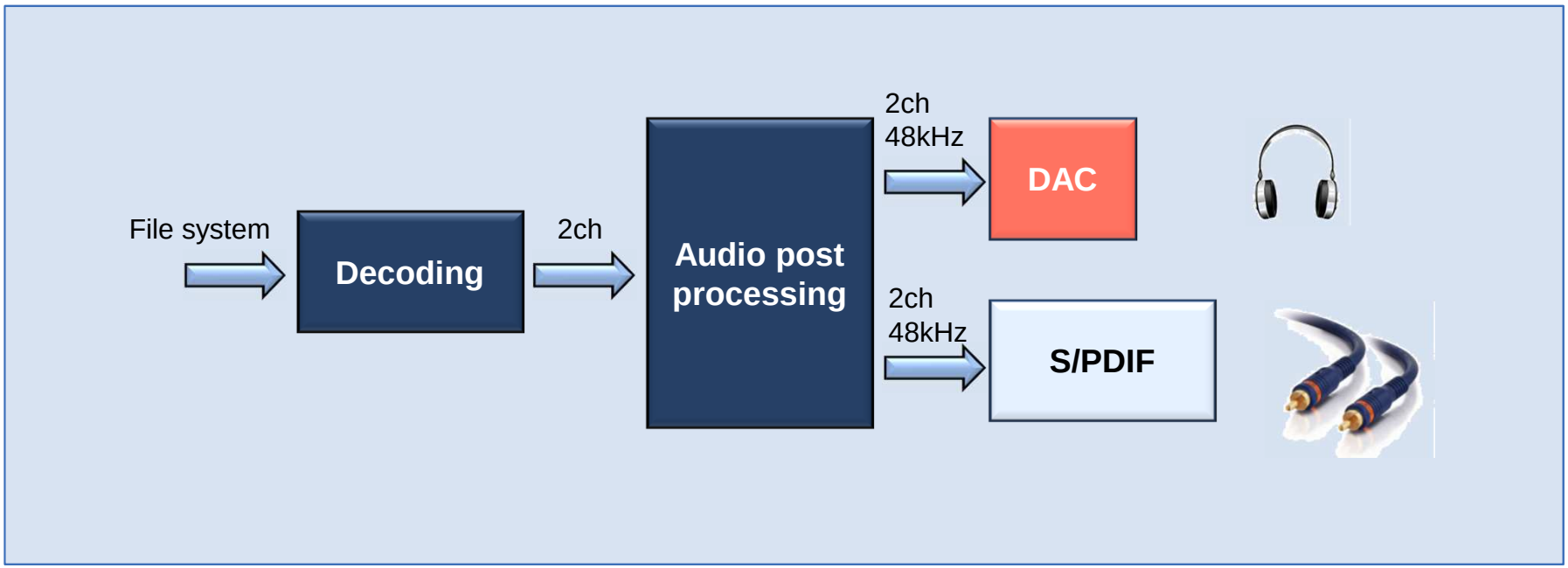
Growing Complexity in Audio

 **Internet Enabled Devices**	• Wider range of audio formats
 **Multi-channel Audio Content**	• From 2.0 up to 7.1 audio channels
Higher Sampling Rates	• 24-bits precision, 192 KHz + Meta data
 **Sound Processing**	• Virtual Surround, Adaptive Volume

More Audio Data Processing:
Host Processor ➔ Dedicated Audio Subsystems

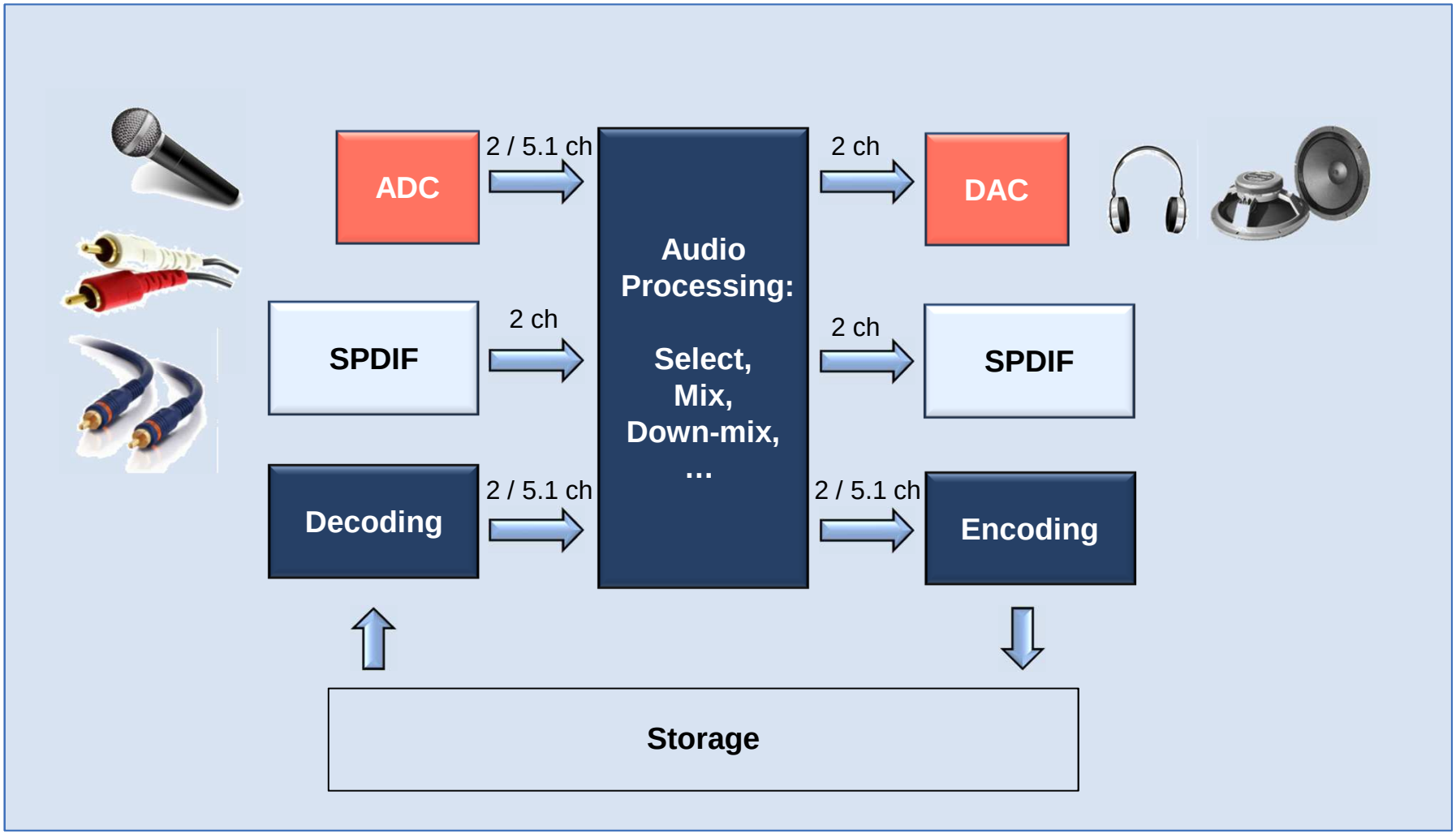
Example Use Case (simple)

Playback From File



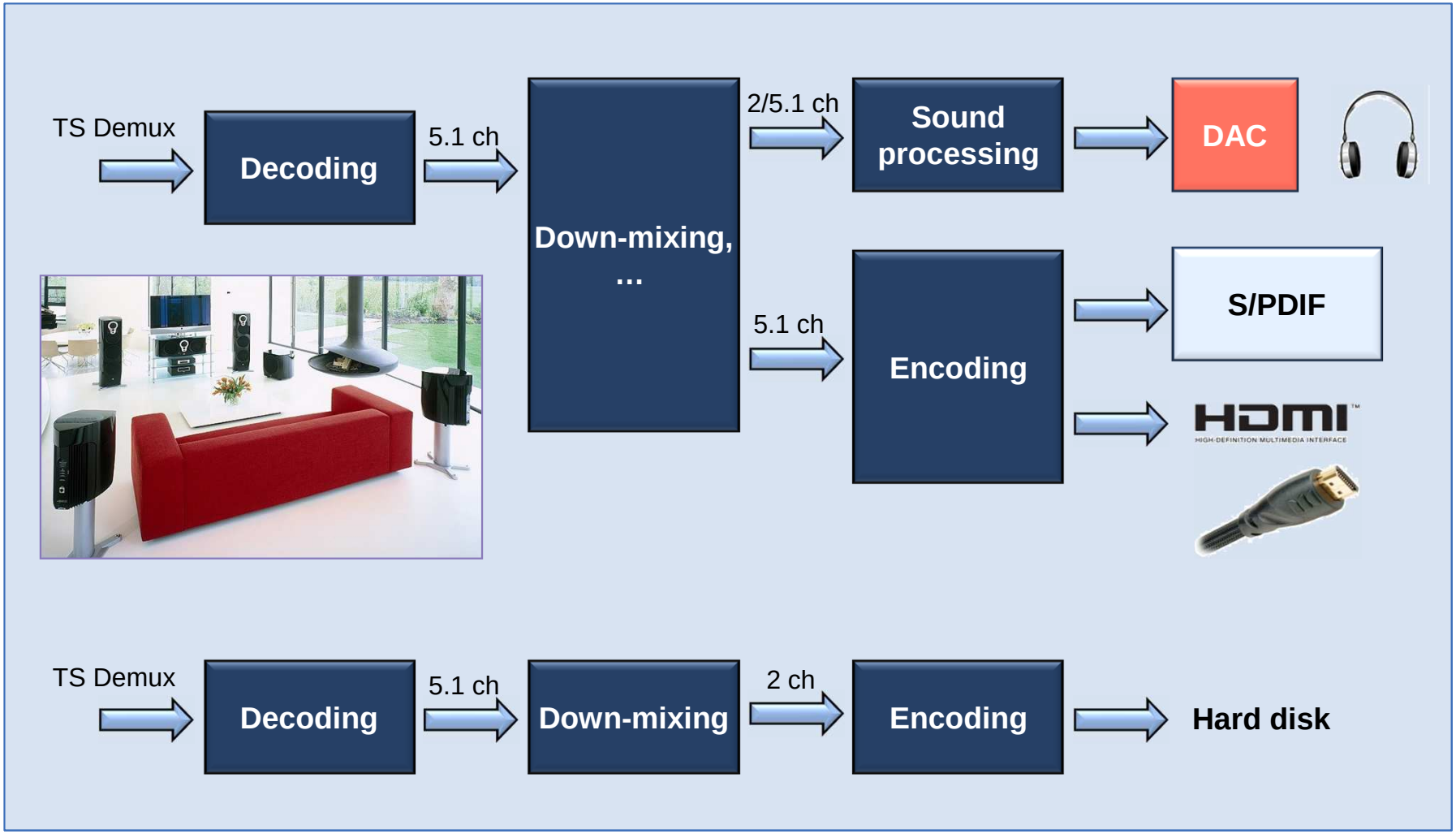
Example Use Case

Recorder / Player



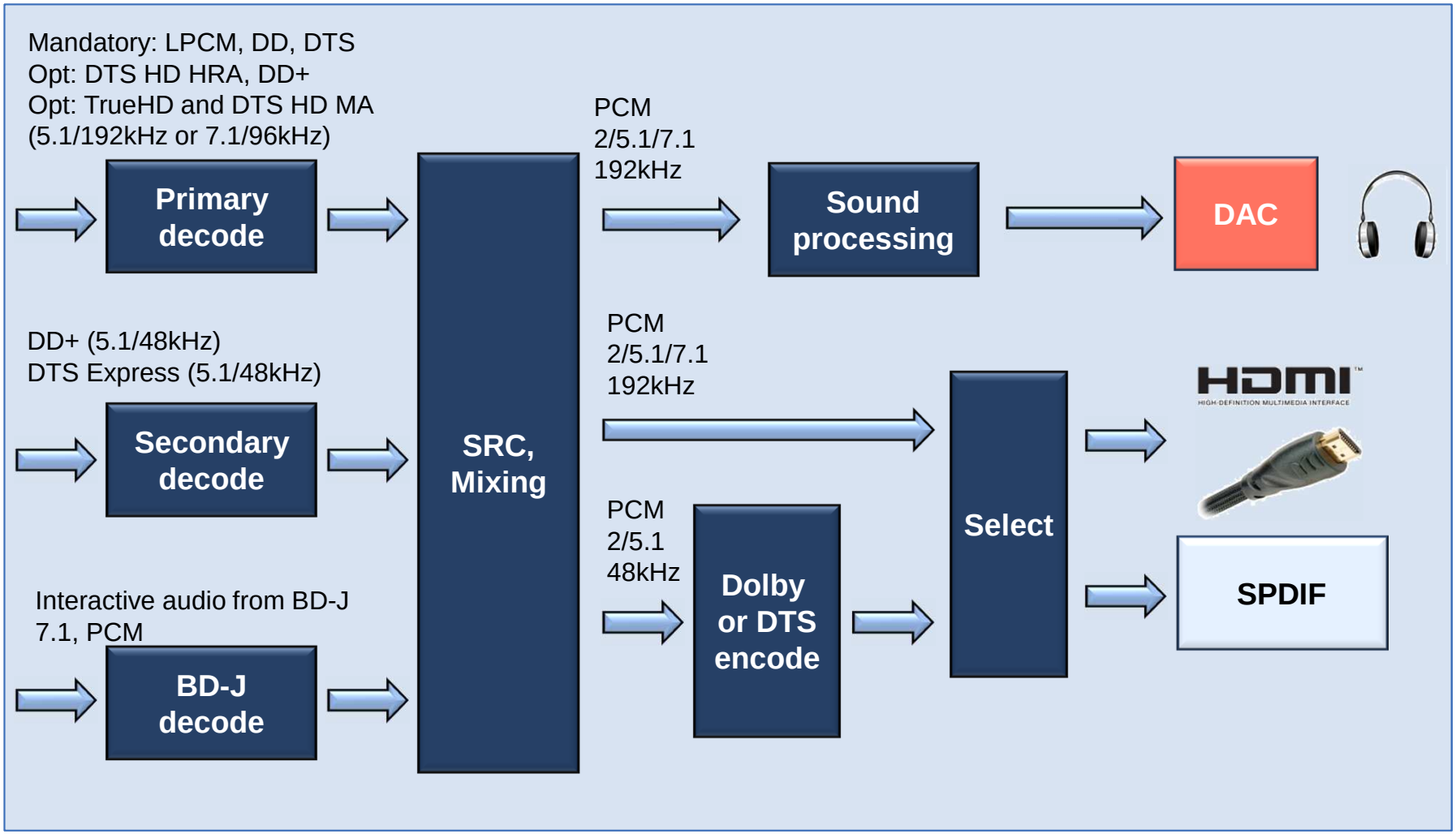
Example Use Case

DTV: Watch and Record Different Channels



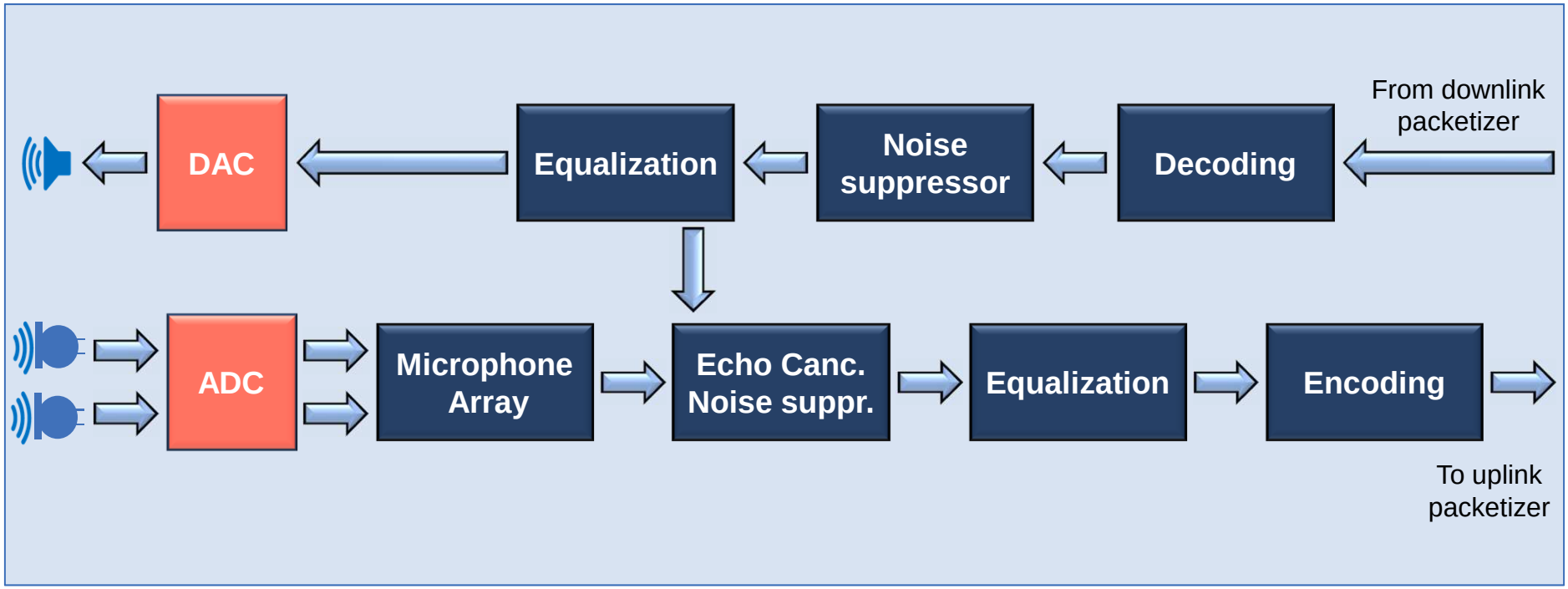
Example Use Case

Blu-ray Disc: Multi-channel, HD Audio



Example Use Case

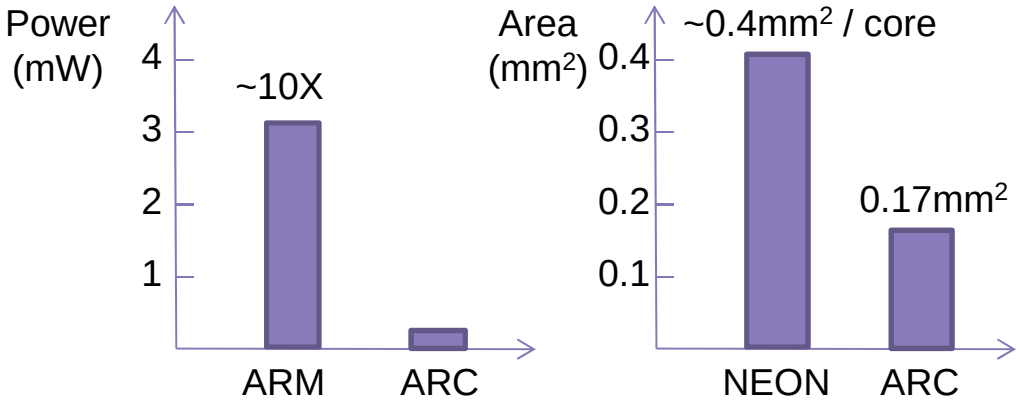
Mobile Phone: LTE voice call

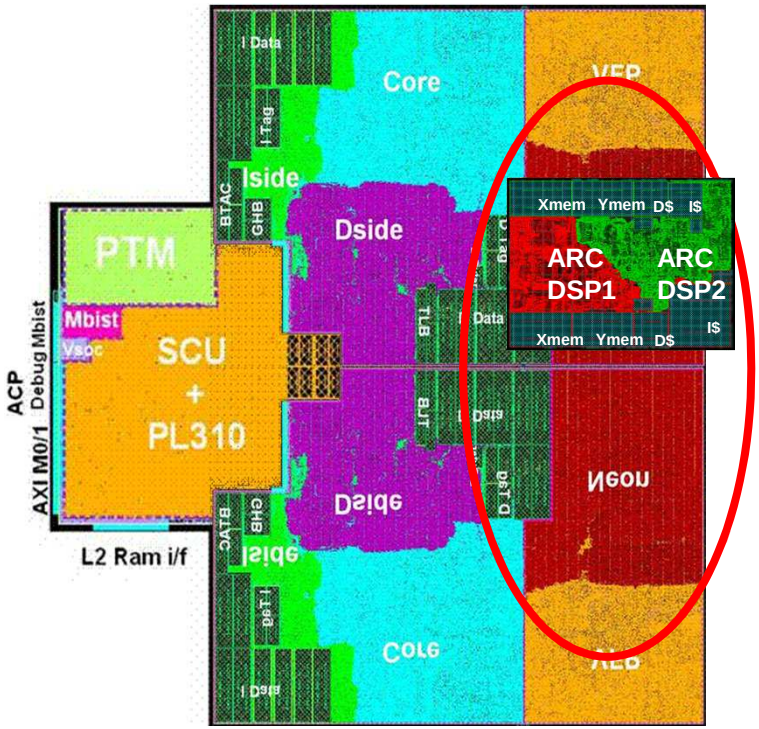


The case for off-loading to an ARC processor

Off-loading of audio processing gives area and power benefits

- ARM Cortex-A9 dual core
  - 4.6mm^2 with 32K I$ / 32K D$ and NEON in TSMC 40G *
  - Total power 0.5W @ 800MHz *
- Power consumption MP3 decode
  - MP3 decode on ARM with NEON: 10MHz**
  - ARM: 500/800*10 / 2 = 3.125mW / core
  - ARC AS211SFX in 40nm: 0.27mW





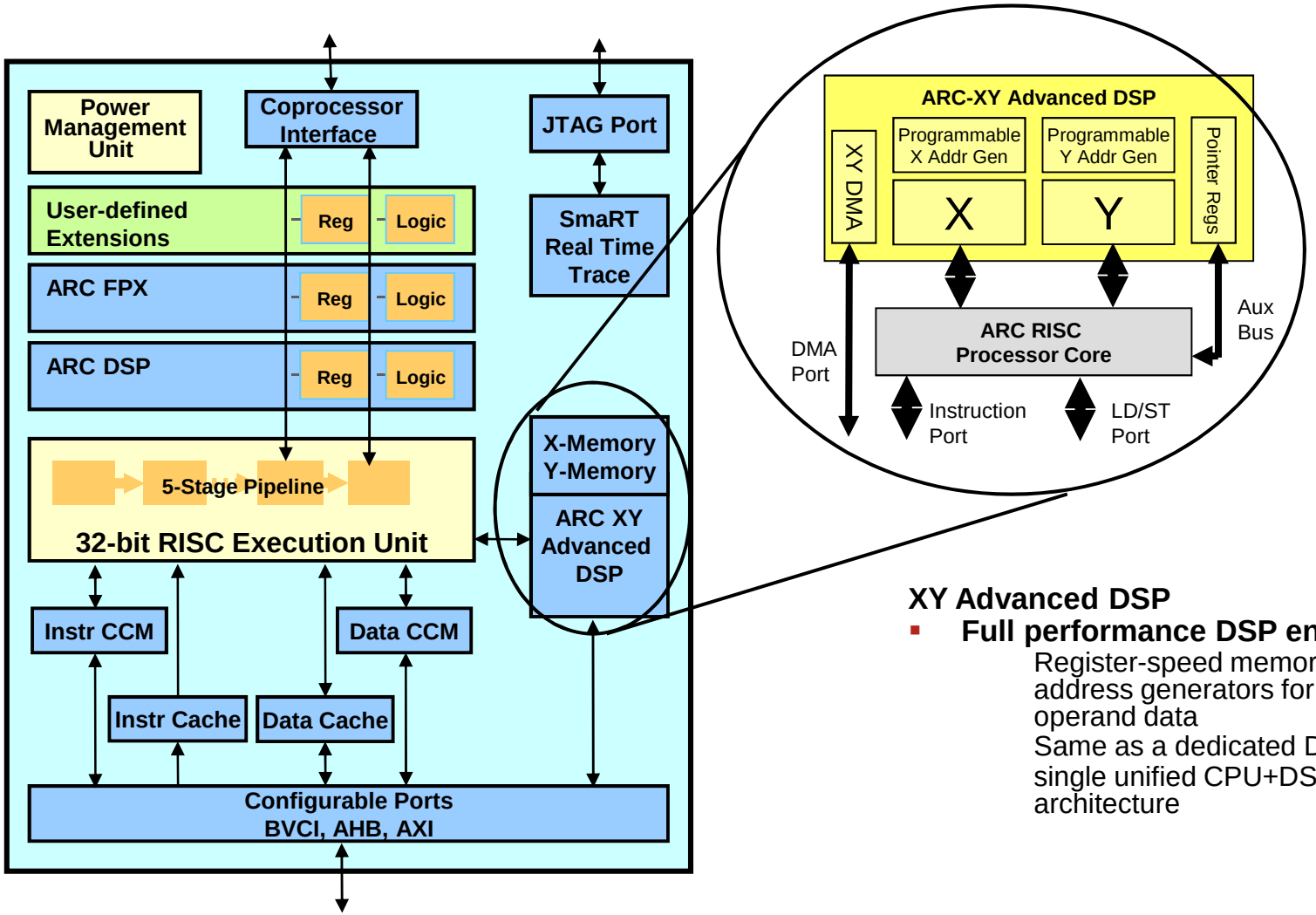
ARM Cortex-A9 dual core
Power optimized hard macro

* Source: www.arm.com

** Source: “Employing ARM NEON in embedded system’s audio processing”, Freescale Semiconductor Inc., EE Times Asia
ARC area post-layout, stdcell + memory, 40nm LP, power consumption dynamic + leakage

AS211SFX Audio Processor

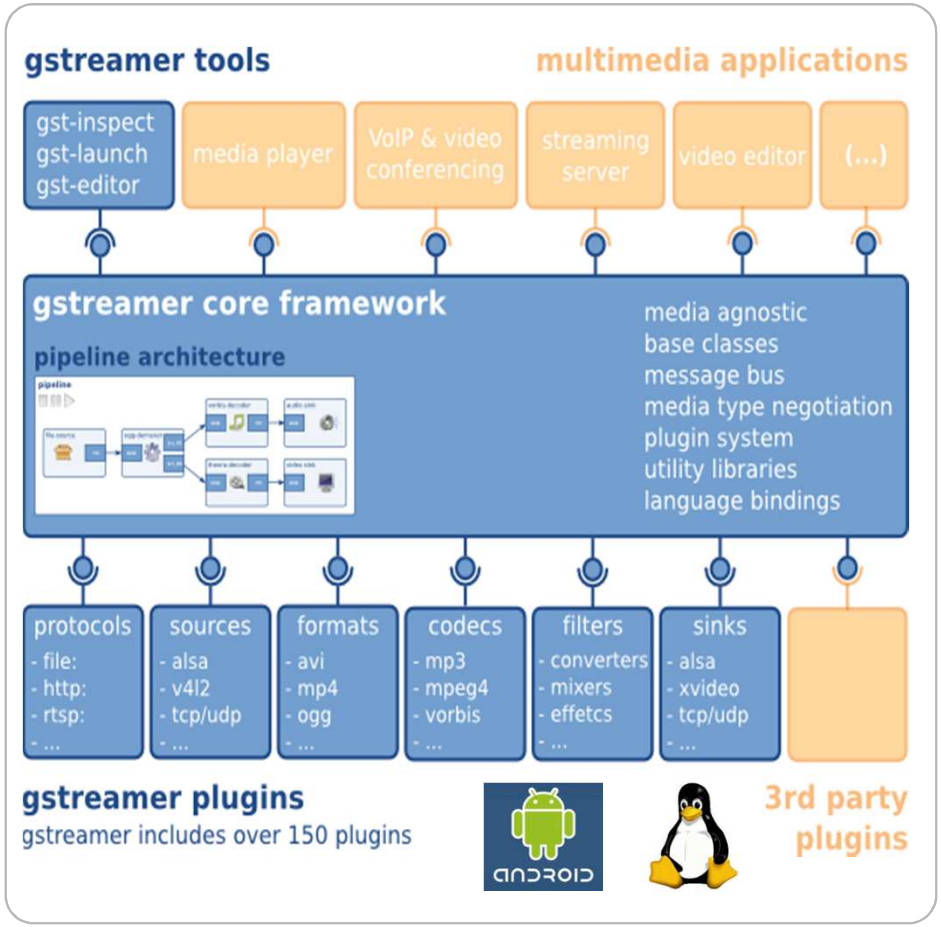
Audio Optimized RISC/DSP



XY Advanced DSP

- **Full performance DSP engine**
  - Register-speed memory and address generators for DSP operand data
  - Same as a dedicated DSP single unified CPU+DSP architecture

Framework for Creating Streaming Media Applications



source: GStreamer Application Developers Manual

- GStreamer provides
  - API for multimedia applications
  - Plug-in architecture
  - Pipeline architecture
  - Mechanism for media type handling/negotiation
  - Over 150 plug-ins
  - Set of tools
- GStreamer plug-ins
  - Protocols handling
  - Sources: for audio and video
  - Sinks: for audio and video
  - Formats: parsers, formaters, muxers, demuxers, metadata, subtitles
  - Codecs: coders and decoders
  - Filters: converters, mixers, effects, ...

File Player Code Snippet

GStreamer Example

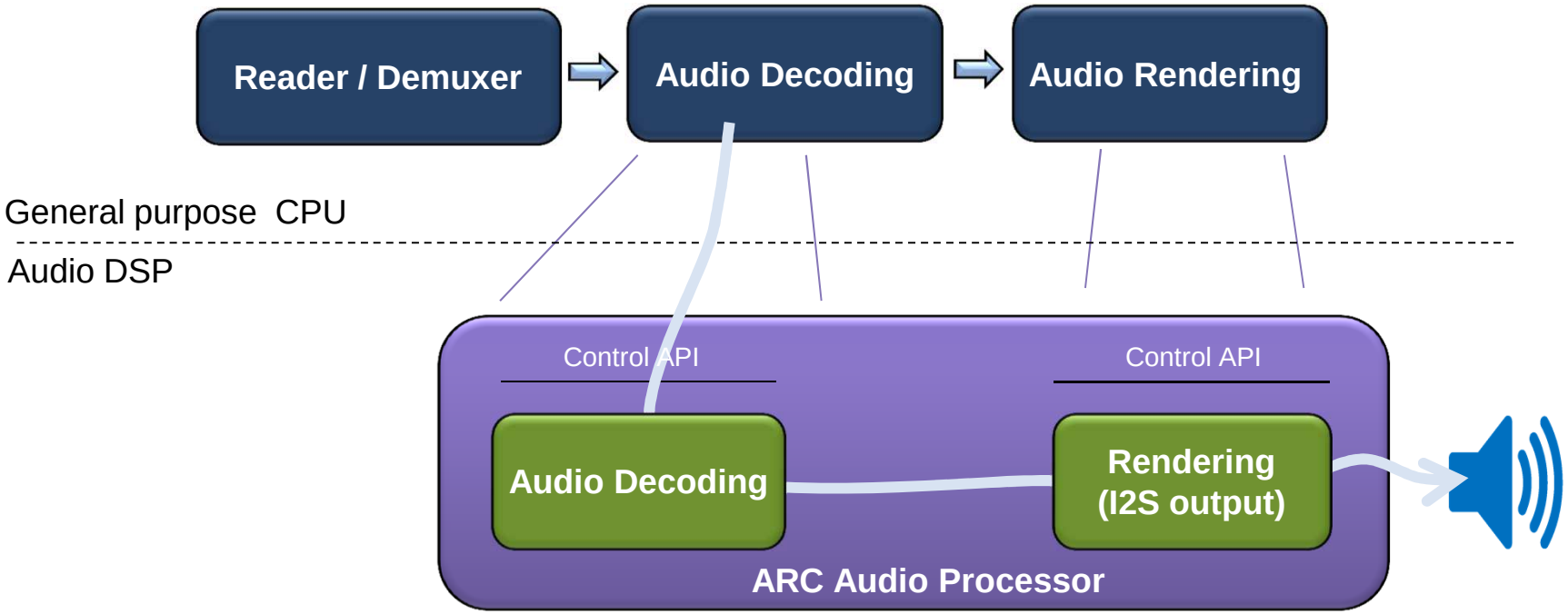

```
pipeline = gst_pipeline_new ("my-pipeline");

source = gst_element_factory_make ("filereader", "filereader");
g_object_set (G_OBJECT (source), "location", filename, NULL);
g_object_set (G_OBJECT (source), "track", track, NULL);
g_object_get (G_OBJECT (source), "decodertype", &decodertype, NULL);

decoder = gst_element_factory_make ("decoder", "decoder");
g_object_set (G_OBJECT (decoder), "decodertype", decodertype, NULL);

sink = gst_element_factory_make (“sink", “renderer I2S stereo");
g_object_set (G_OBJECT (sink), “sinktype", I2S-STEREO, NULL);

gst_bin_add_many (GST_BIN (pipeline), source, decoder, sink, NULL);
gst_element_link_many (source, decoder, sink, NULL);

gst_element_set_state (pipeline, GST_STATE_PLAYING);
```

Agenda

- Introduction
- Problem statement & Solution Sketch
  - Heterogeneous multi-core hardware for efficiency … but how about the SW?
- The Details!
- Demo
- Conclusions, Q&A

Audio DSP Software Made Easy

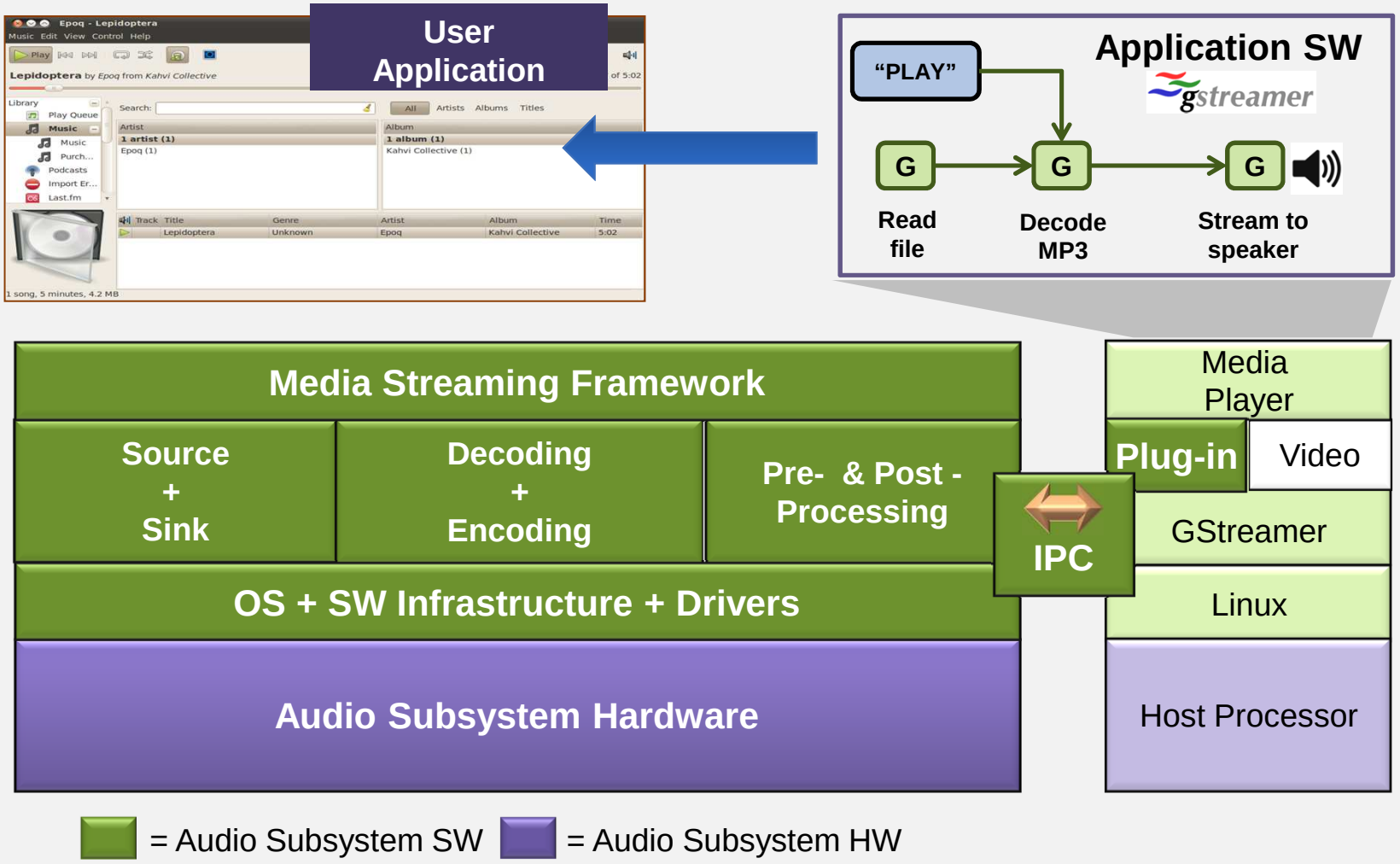
Transparently off-load audio processing to DSP



Key Software Architecture Value Drivers

- Ease of use : transparent heterogeneous multi-core, standard & high-level API
- Efficiency : optimized ARC audio processors, keep data/control local

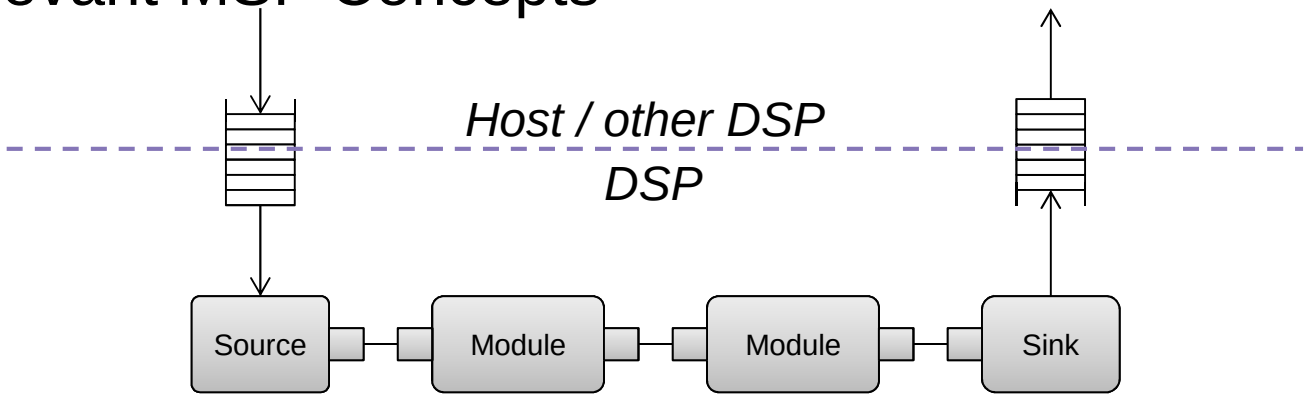
What Solution Components Do We Need?



DSP Streaming Framework Candidates

- GStreamer -> large, GObject & other dependencies
- OpenMAX-IL (OMX) -> ‘standard’, but …
  - (deep) tunneling often not supported, ok single codec offload
- Proprietary from DSP vendor
  - Synopsys : MSF / MM-MQX

- Relevant MSF Concepts



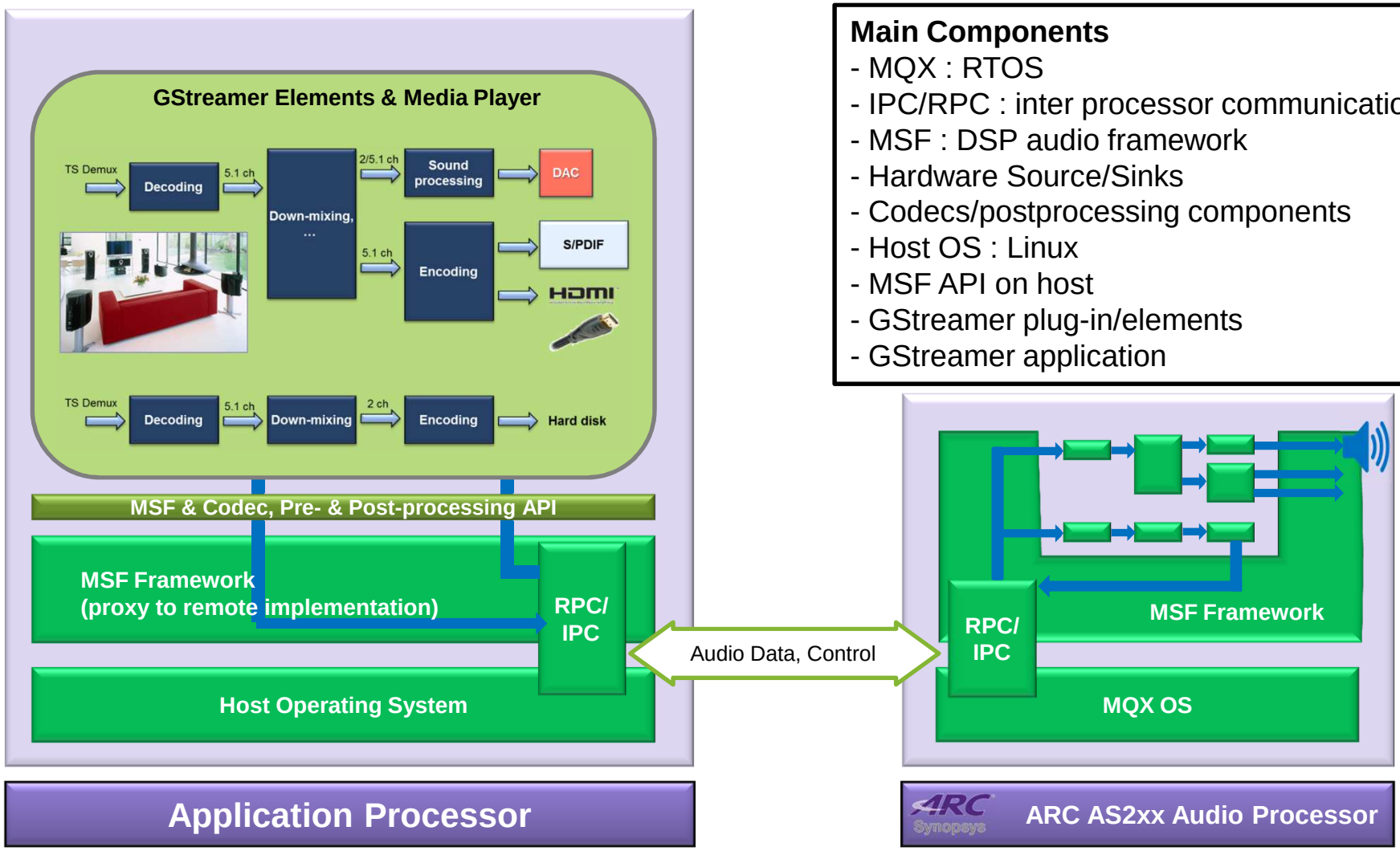
IPC Candidates

- RemoteProc
- Standard?
  - MCAPI, MPI, OpenCL, …
- Proprietary from DSP Vendor
  - Synopsys : mciCOM

- Required Features
  - Start/stop/reset DSP
  - Download firmware
  - Shared memory management
  - Message communication
  - Nice-to-have : remote procedure calls

Integration Example

GStreamer Broadcast watch & record



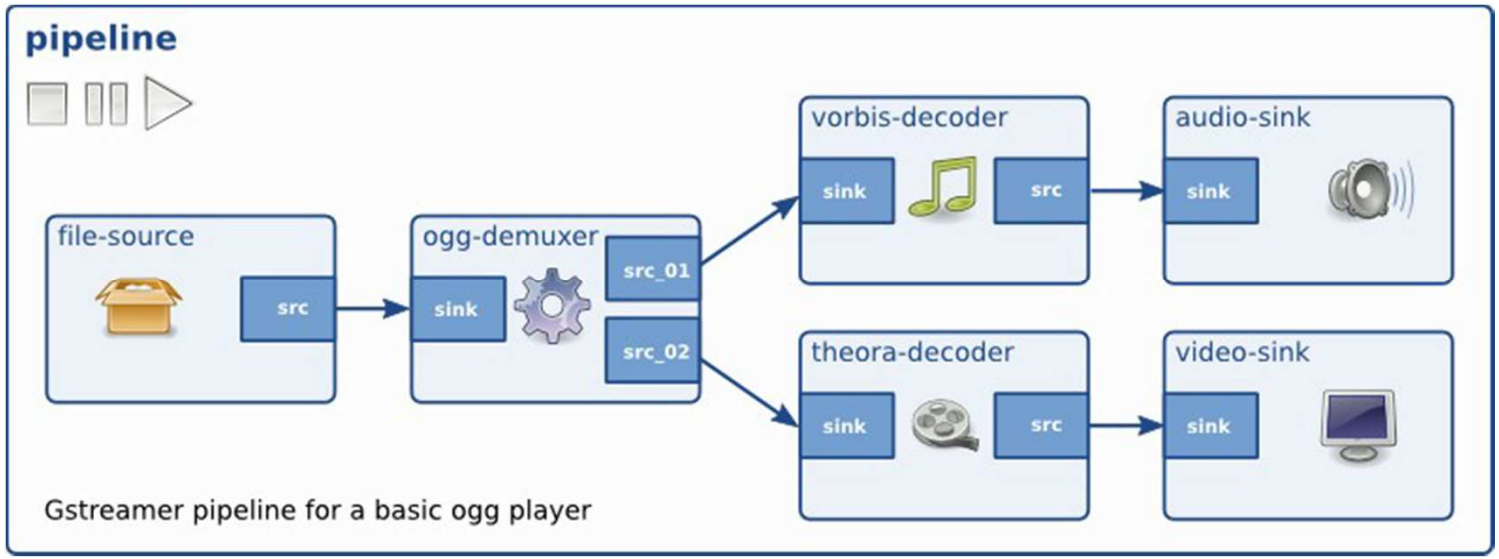
Main Components

- MQX : RTOS
- IPC/RPC : inter processor communication
- MSF : DSP audio framework
- Hardware Source/Sinks
- Codecs/postprocessing components
- Host OS : Linux
- MSF API on host
- GStreamer plug-in/elements
- GStreamer application

Agenda

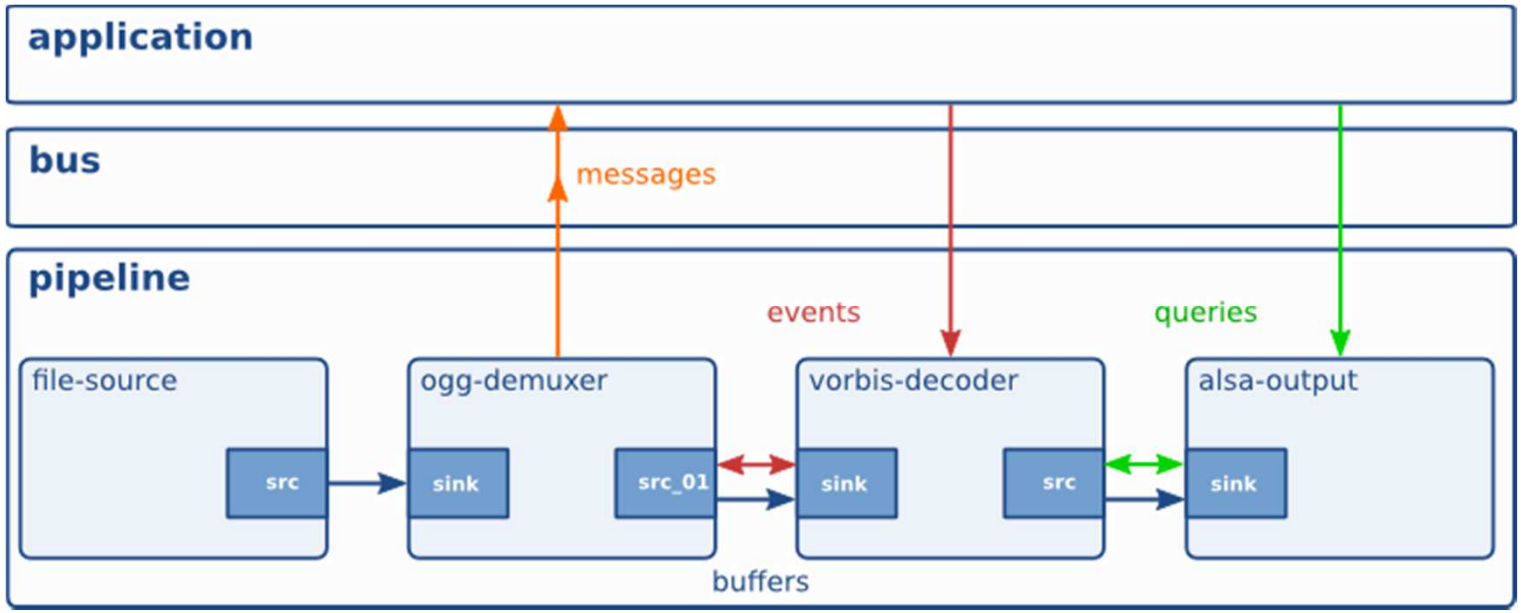
- Introduction
- Problem statement & Solution Sketch
- The Details!
  - GStreamer
  - Plug-in
- Demo
- Conclusions, Q&A

GStreamer Concepts (1/3)



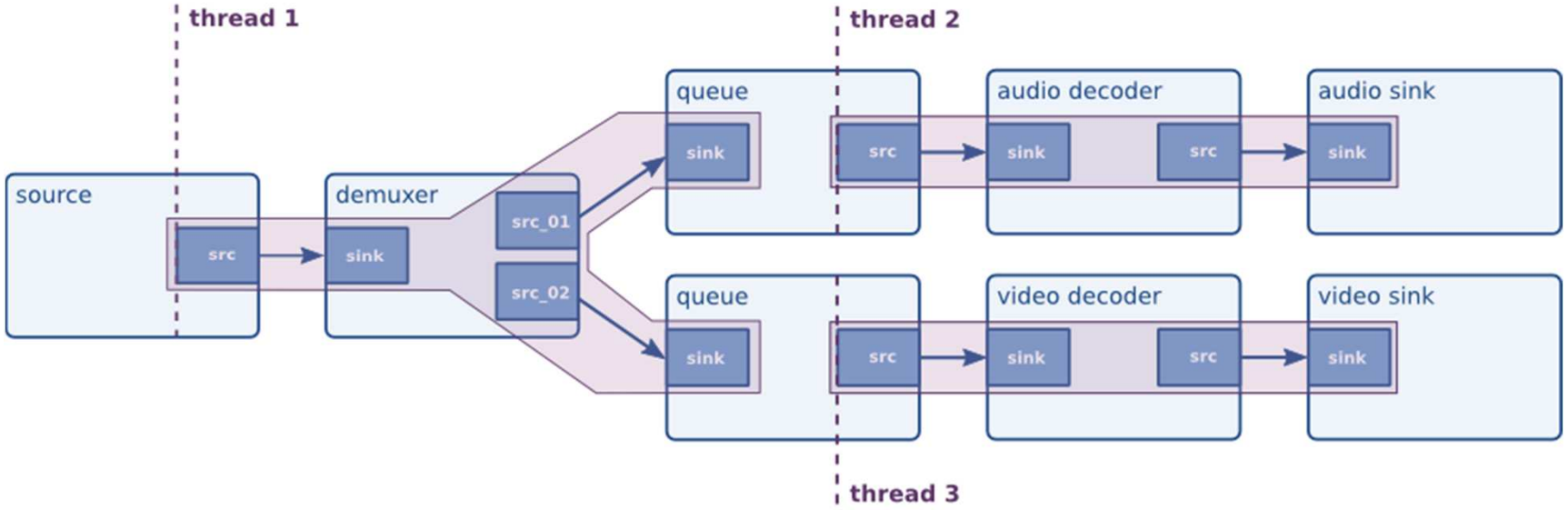
- Elements
- Source & Sink Pads
  - Capabilities
- Pipeline

GStreamer Concepts (2/3)



- Buffers
- Messages & Message Bus
- Events & Queries

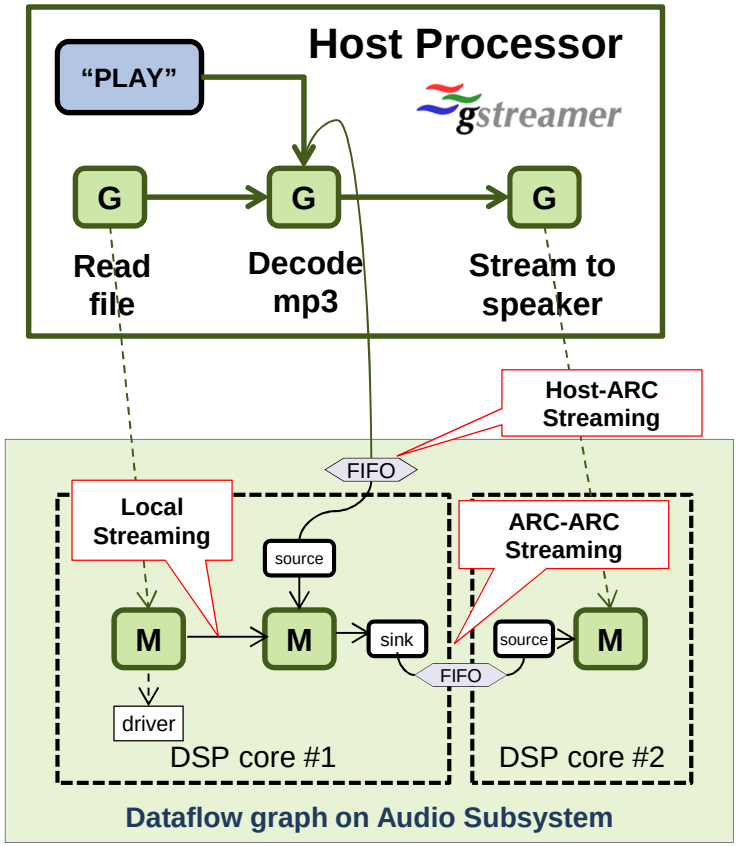
GStreamer Concepts (3/3)



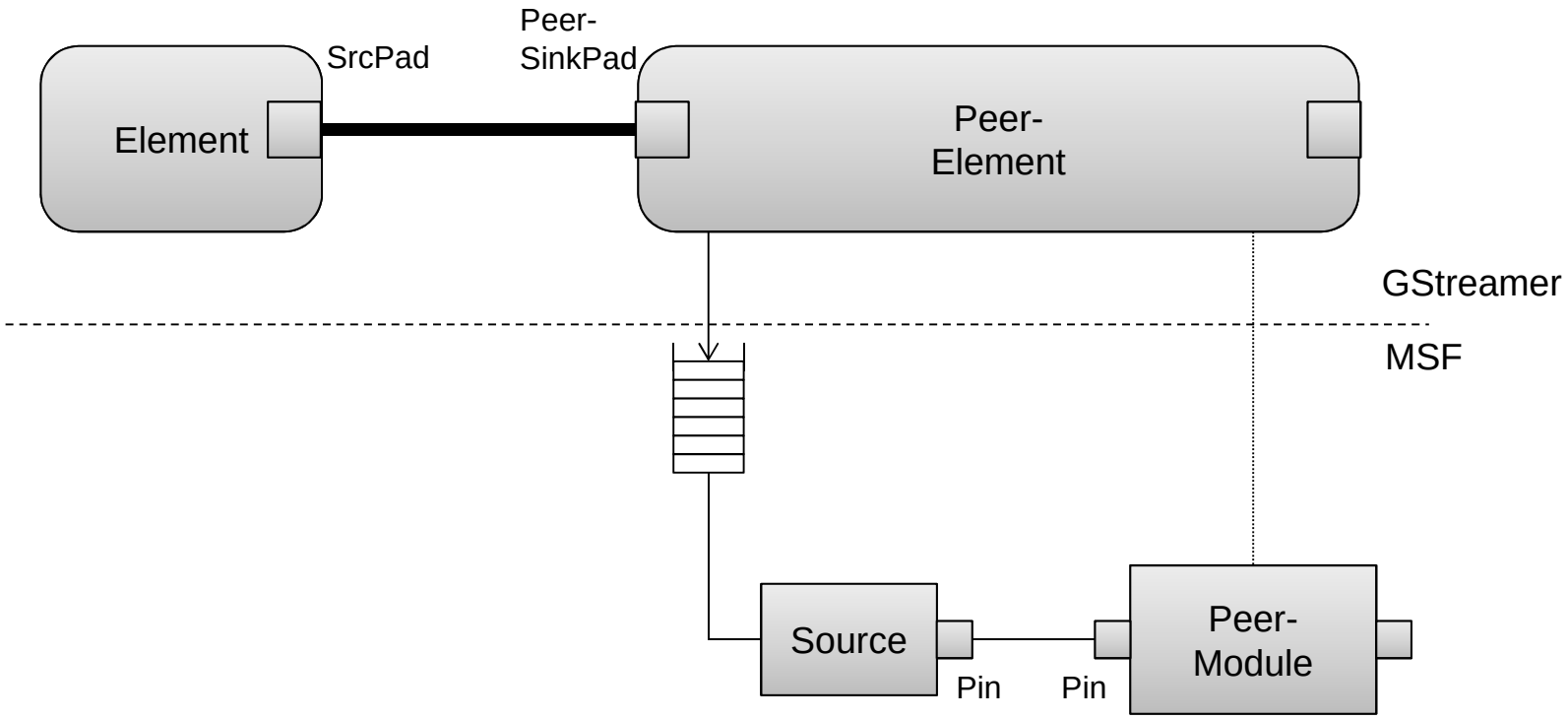
- Queues
- Threads
- Chain & Loop functions

GStreamer DSP Off-loading Overview

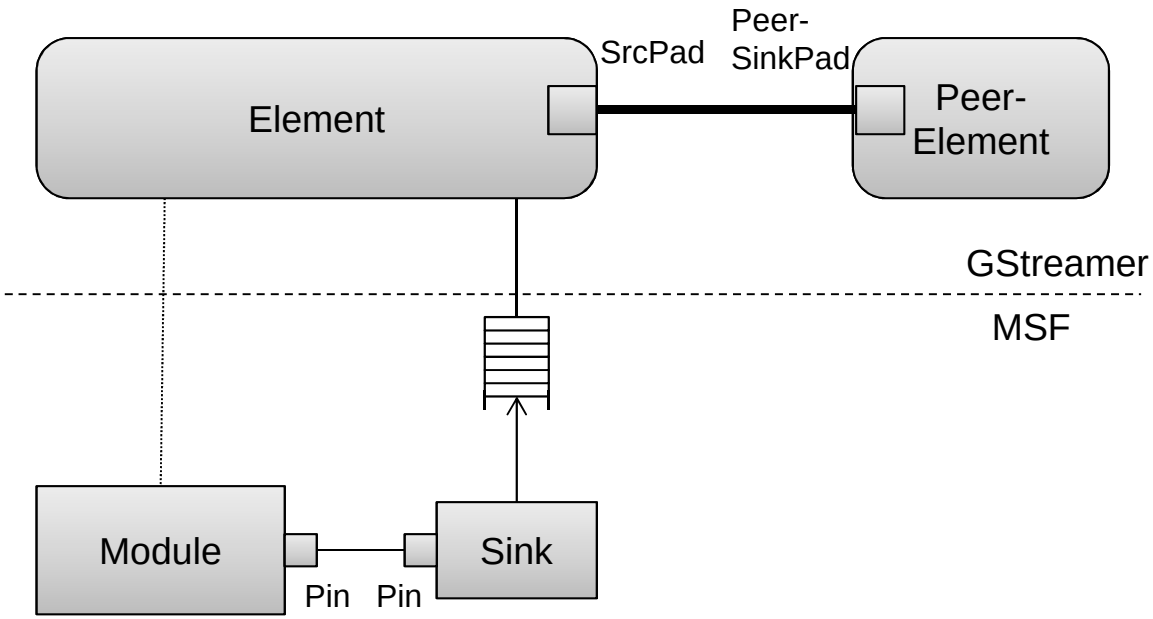
- Instantiation of GStreamer element → instantiation of module on one of the ARC cores
- Creation of link → local connection or core-crossing connection between modules



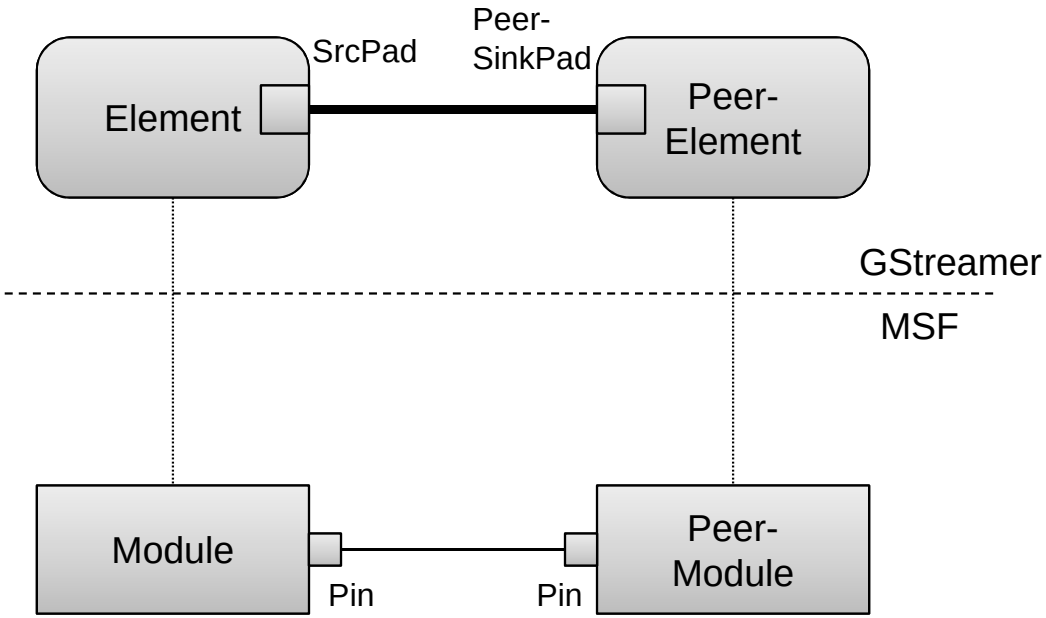
Implementation on DSP, Data From Host



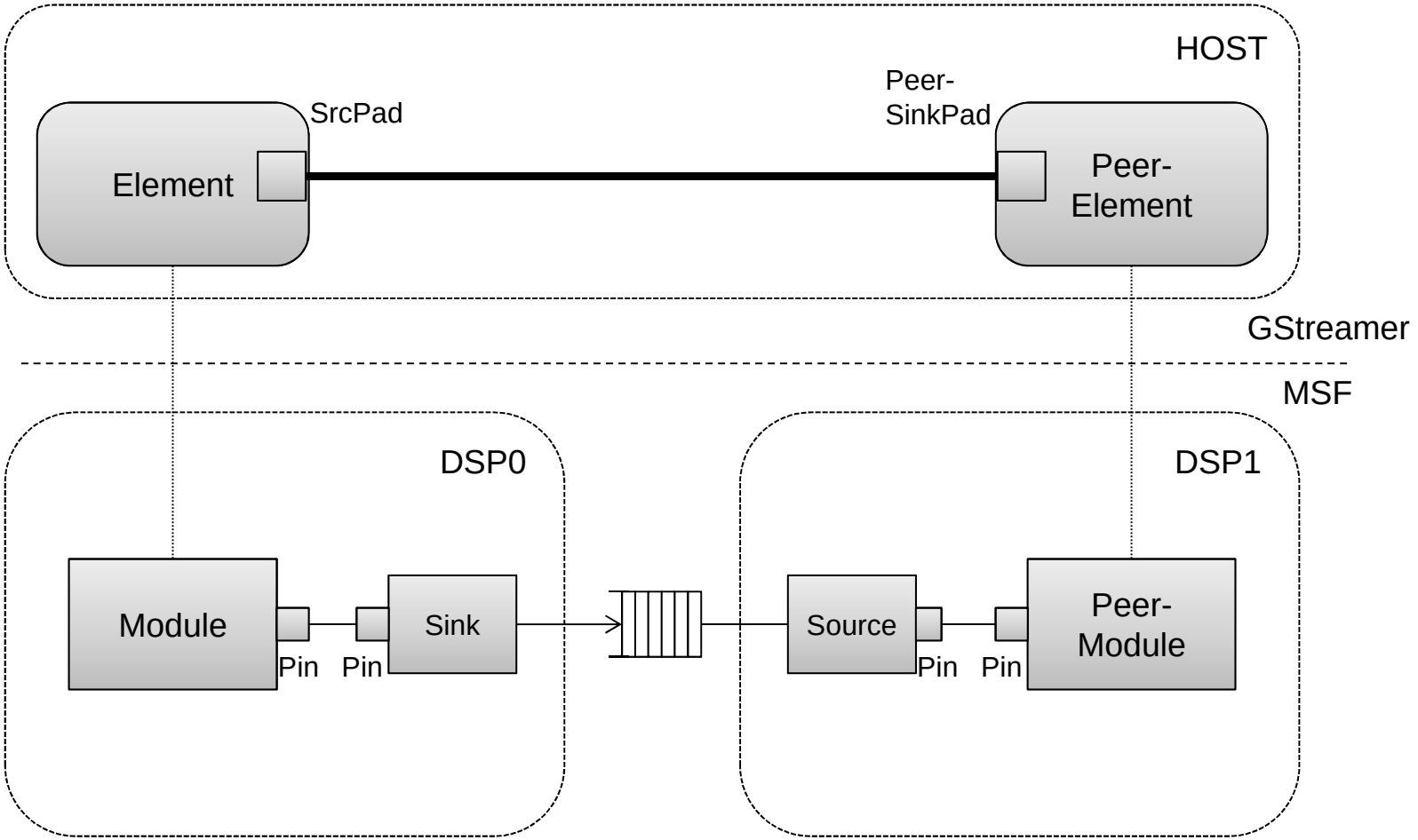
Implementation on DSP, Data Back to Host



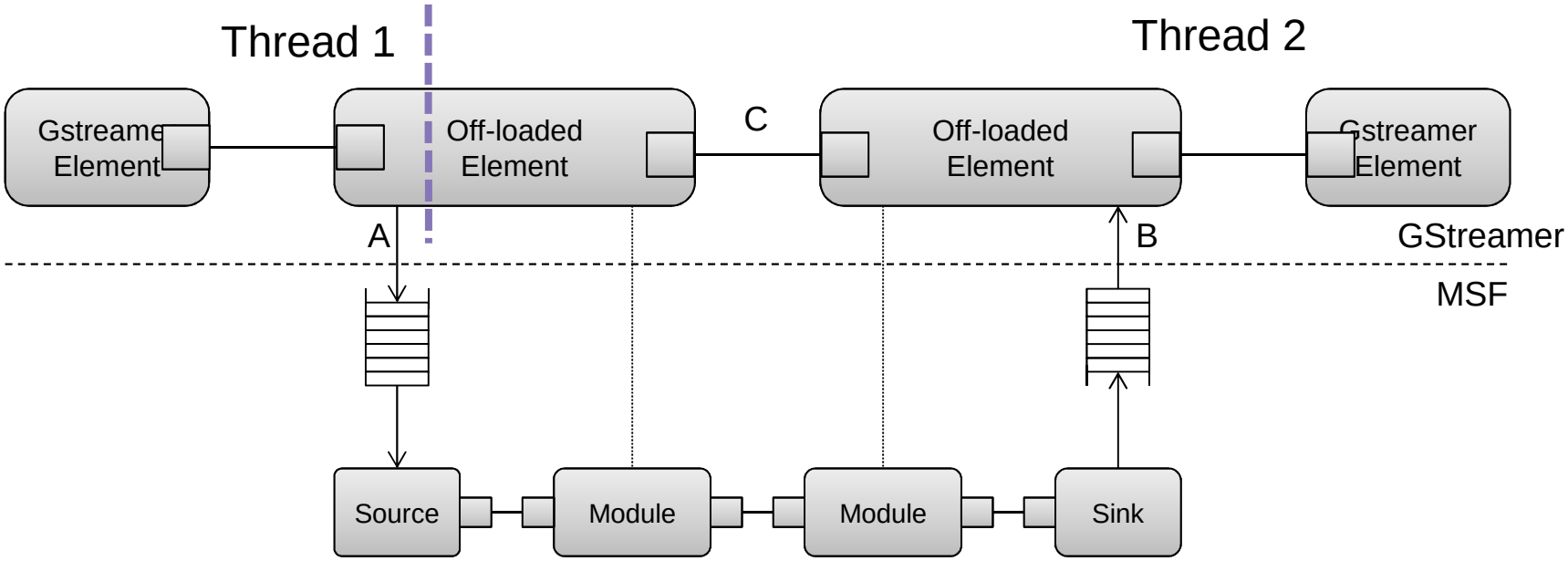
Both Components on DSP: Deep-Tunneled



Deep Tunneled & DSP Core Crossing



Threading



Connection	Description
A	Chain function handles data transfer to the FIFO
B	Loop function handles data transfer from the FIFO
C	Only connected, nothing else happens here

Mapping States

GStreamer state transition	MSF actions
NULL → READY	Create MSF modules, FIFO's
READY → PAUSED	Create Sources and Sinks, Connect the modules
PAUSED → PLAYING	Sink Element send RENDER call to driver
PLAYING → PAUSED	Sink Element send PAUSE call to driver
PAUSED → READY	N/A
READY → NULL	Destroys all modules and FIFO's (automatically disconnected)

GStreamer Deep Tunneling

```
static void connect_msf_outpin (GstPad* pad)
{
    GstPad         *peerpad = gst_pad_get_peer(pad);
    GstElement     *element = gst_pad_get_parent_element( pad );
    GstElement     *peerelement = gst_pad_get_parent_element( peerpad );
    GstAudioModule *filter = GST_AUDIOMODULE(element);
    guint32         result;

    if (!pad_is_deeptunnel(pad))
    {
        /* not a deep tunnel */
        /* create sink module */
        msf_api_sink_module_create(filter->msf_coreid, "Sink module", output_fifo_buffer,
                                   sink_pv_data, sizeof(sink_pv_data), &sink_module_id)));
        msf_api_connect_pins(filter->msf_moduleid, sink_module_id, 0, 0)));
    }
    else
    {
        if (pad_is_corecrossing(pad))
        {
            /* deep tunnel AND core-crossing */
            /* create sink module */
            msf_api_sink_module_create(filter->msf_coreid, "Sink module", filter->msf_sharedfifo,
                                       sink_pv_data, sizeof(sink_pv_data), &sink_module_id)));
            msf_api_connect_pins(filter->msf_moduleid, sink_module_id, 0, 0)))
        }
        else
        {
            /* deep-tunnel AND no core-crossing */
            guint32 peer_module_id;

            /* get the module id of the peer MSF module */
            g_object_get (G_OBJECT (peerelement), "msf_moduleid", &peer_module_id, NULL);

            msf_api_connect_pins(filter->msf_moduleid, peer_module_id, 0, 0)))
        }
    }
}
```

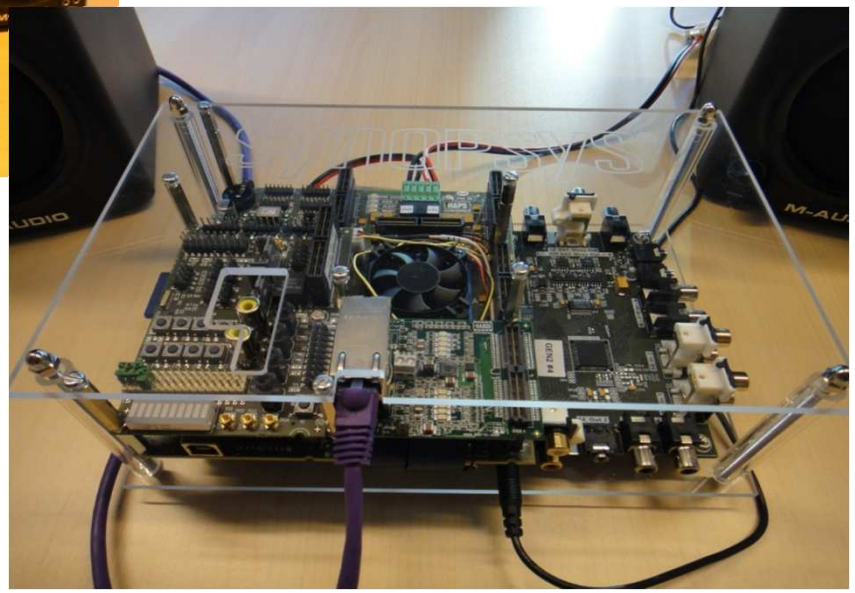
Miscellaneous Topics

- Configuration & control
  - GStreamer / GObject properties
- Events & messages
  - End-of-Stream handling
- Clock & A/V sync
  - HW clocks for audio in/output

Agenda

- Introduction
- Problem statement & Solution Sketch
- The Details!
- Demo
- Conclusions, Q&A

Virtual & FPGA Prototypes



Agenda

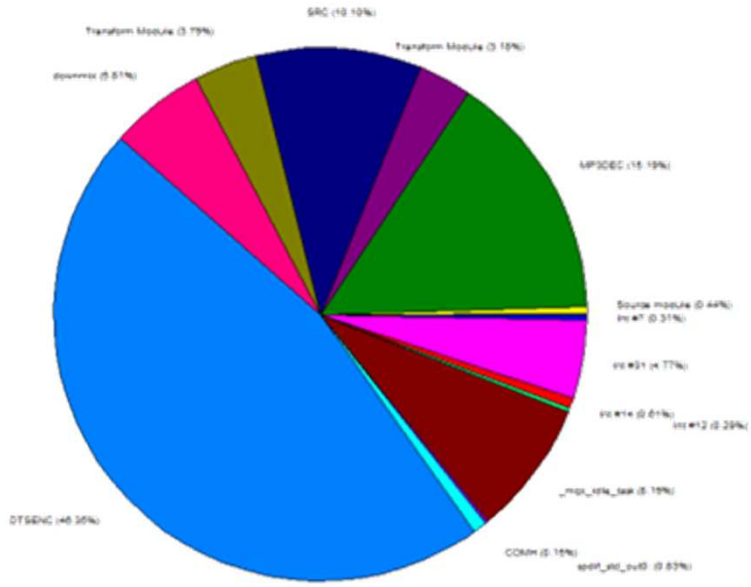
- Introduction
- Problem statement & Solution Sketch
- The Details!
- Demo
- Conclusions, Q&A

Conclusions, Q&A

- A solution for a GStreamer plug-in that off-loads audio processing to an (efficient) DSP
  - Building on RemoteProc or similar IPC solution
  - Utilizing a DSP Media Streaming Framework
- That preserves GStreamer's ease-of-use and flexible graph-creation capabilities
- And is despite of the flexibility still very efficient
  - Not just off-loading a single codec, but complete sub-graphs of a pipeline

Software Performance

Numbers from SoundWave 1.0



Use Case	File playback	File playback + SRS TruVolume	Rip & Record (Dolby encode)
Audio processing	24.6 MHz	57.7 MHz	62.8 MHz
Peripheral output	3.8 MHz	3.5 MHz	2.8 MHz
Host communication	0.8 MHz	0.8 MHz	0.6 MHz
Total	29.2 MHz	62.0 MHz	66.2 MHz

References

- GStreamer
  - http://gstreamer.freedesktop.org/
  - http://gstreamer.freedesktop.org/documentation/
- OpenMAX
  - http://www.khronos.org/openmax/
- SoundWave & ARC Audio Processor
  - http://www.synopsys.com/dw/ipdir.php?ds=arc_audio_processors
  - http://www.synopsys.com/dw/ipdir.php?ds=audio_subsystem


Thank You
SYNOPSYS®
Accelerating Innovation