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

- ARC 700, EM
- DDR PHY
- DDR controller
- AMBA 3 AXI & AMBA 2.0 AHB
- Signal processing
- ADCs
- DACs
- I2C
- GPIO
- UART
- Logic Libraries
- Datapath
- Digital IP
- Physical IP
- Verification IP

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

*Source: Gartner, March 2012
## Growing Complexity in Audio

<table>
<thead>
<tr>
<th>Internet Enabled Devices</th>
<th>• Wider range of audio formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-channel Audio Content</td>
<td>• From 2.0 up to 7.1 audio channels</td>
</tr>
<tr>
<td>Higher Sampling Rates</td>
<td>• 24-bits precision, 192 KHz + Meta data</td>
</tr>
<tr>
<td>Sound Processing</td>
<td>• Virtual Surround, Adaptive Volume</td>
</tr>
</tbody>
</table>

---

**More Audio Data Processing:**
- Host Processor ➔ Dedicated Audio Subsystems
Example Use Case (simple)

Playback From File

File system → Decoding → Audio post processing

2ch 48kHz → DAC

2ch 48kHz → S/PDIF

DAC

S/PDIF
Example Use Case

Recorder / Player

Audio Processing:
Select, Mix, Down-mix, ...

ADC 2 / 5.1 ch → 2 ch → DAC

SPDIF 2 ch → 2 ch → SPDIF

Decoding 2 / 5.1 ch → 2 / 5.1 ch → Encoding

Storage
Example Use Case

DTV: Watch and Record Different Channels

1. TS Demux → Decoding (5.1 ch) → Down-mixing, … → Encoding (2 ch) → Hard disk
2. TS Demux → Decoding (5.1 ch) → Down-mixing, … → Encoding (5.1 ch) → Sound processing → DAC → S/PDIF → HDMI
Example Use Case

*Blu-ray Disc: Multi-channel, HD Audio*

Mandatory: LPCM, DD, DTS
Opt: DTS HD HRA, DD+
Opt: TrueHD and DTS HD MA
(5.1/192kHz or 7.1/96kHz)

**Primary decode**

**Secondary decode**

DD+ (5.1/48kHz)
DTS Express (5.1/48kHz)

**SRC, Mixing**

PCM 2/5.1/7.1 192kHz

**Sound processing**

PCM 2/5.1/7.1 192kHz

**Dolby or DTS encode**

PCM 2/5.1 48kHz

**Select**

**SPDIF**

**DAC**

Interactive audio from BD-J
7.1, PCM

**BD-J decode**

**Sound processing**
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

* 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

© Synopsys 2013
AS211SFX Audio Processor
Audio Optimized RISC/DSP

ARC-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

XY DMA
- X
- Y

ARC RISC Processor Core

Aux Bus
- DMA Port
- Instruction Port
- LD/ST Port

Power Management Unit

User-defined Extensions
- Reg
- Logic

ARC FPX
- Reg
- Logic

ARC DSP
- Reg
- Logic

32-bit RISC Execution Unit
- 5-Stage Pipeline
- Instr CCM
- Data CCM
- Instr Cache
- Data Cache

Configurable Ports
- BVCI, AHB, AXI

Intr Cache

Data Cache

Intr Cache

Data Cache

SmaRT Real Time Trace

Coprocessor Interface

JTAG Port

Configure Ports
- BVCI, AHB, AXI

© Synopsys 2013

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

- **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, formatters, muxers, demuxers, metadata, subtitles
  - Codecs: coders and decoders
  - Filters: converters, mixers, effects, ...

source: GStreamer Application Developers Manual
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?

- **Media Streaming Framework**
  - Source + Sink
  - Decoding + Encoding
  - Pre- & Post-Processing

- **OS + SW Infrastructure + Drivers**

- **Audio Subsystem Hardware**
  - Audio Subsystem SW
  - Audio Subsystem HW

- **Application SW**
  - Read file
  - Decode MP3
  - Stream to speaker

- **User Application**

- **GStreamer**
  - Media Player
  - Video
  - Linux
  - Host Processor

- **Plug-in**

- **IPC**

- **What Solution Components Do We Need?**

  - Plug-in
  - Video
  - GStreamer
  - Linux
  - Host Processor

- **“PLAY”**

- **Read file**

- **Decode MP3**

- **Stream to speaker**

- **User Application**

- **Audio Subsystem Hardware**
  - Audio Subsystem SW
  - Audio Subsystem HW
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

Diagram:
- Element
  - SrcPad
- Peer-Element
  - Peer-SinkPad
- Source
  - Pin
- Peer-Module
  - Pin
- GStreamer
- MSF
Implementation on DSP, Data Back to Host

Diagram:
- Element
- Peer-Element
- GStreamer
- MSF
- Module
- Sink
- SrcPad
- Peer-SinkPad
- Pin
- Pin
Both Components on DSP: Deep-Tunneled
Deep Tunneled & DSP Core Crossing

HOST

Peer-Element

GStreamer

MSF

DSP0

Module

Sink

DSP1

Source

Peer-Module

SrcPad

Peer-SinkPad

Pin

Pin

Pin

Pin
Threading

<table>
<thead>
<tr>
<th>Connection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Chain function handles data transfer to the FIFO</td>
</tr>
<tr>
<td>B</td>
<td>Loop function handles data transfer from the FIFO</td>
</tr>
<tr>
<td>C</td>
<td>Only connected, nothing else happens here</td>
</tr>
</tbody>
</table>
### Mapping States

<table>
<thead>
<tr>
<th>GStreamer state transition</th>
<th>MSF actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL $\rightarrow$ READY</td>
<td>Create MSF modules, FIFO’s</td>
</tr>
<tr>
<td>READY $\rightarrow$ PAUSED</td>
<td>Create Sources and Sinks, Connect the modules</td>
</tr>
<tr>
<td>PAUSED $\rightarrow$ PLAYING</td>
<td>Sink Element send RENDER call to driver</td>
</tr>
<tr>
<td>PLAYING $\rightarrow$ PAUSED</td>
<td>Sink Element send PAUSE call to driver</td>
</tr>
<tr>
<td>PAUSED $\rightarrow$ READY</td>
<td>N/A</td>
</tr>
<tr>
<td>READY $\rightarrow$ NULL</td>
<td>Destroys all modules and FIFO’s (automatically disconnected)</td>
</tr>
</tbody>
</table>
GStreamer Deep Tunneling

```c
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", outputfifo_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

- Complete, configurable audio subsystem consisting of hardware, software and prototyping
- Single and dual-core processors support latest audio standards: 2013 and 2014
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*

<table>
<thead>
<tr>
<th>Use Case</th>
<th>File playback</th>
<th>File playback + SRS TruVolume</th>
<th>Rip &amp; Record (Dolby encode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio processing</td>
<td>24.6 MHz</td>
<td>57.7 MHz</td>
<td>62.8 MHz</td>
</tr>
<tr>
<td>Peripheral output</td>
<td>3.8 MHz</td>
<td>3.5 MHz</td>
<td>2.8 MHz</td>
</tr>
<tr>
<td>Host communication</td>
<td>0.8 MHz</td>
<td>0.8 MHz</td>
<td>0.6 MHz</td>
</tr>
<tr>
<td>Total</td>
<td>29.2 MHz</td>
<td>62.0 MHz</td>
<td>66.2 MHz</td>
</tr>
</tbody>
</table>
References

- GStreamer
  - http://gstreamer.freedesktop.org/
  - http://gstreamer.freedesktop.org/documentation/

- OpenMAX
  - http://www.khronos.org/openmax/

- SoundWave & ARC Audio Processor
Thank You