Developing low latency video telephony solutions using Gstreamer

Date: 2023-09-26

Devarsh Thakkar

Gstconf 2023: A Coruna, Spain

About us: TI Processors and Open source

THE LINUX FOUNDATION

Real-Time LINUX

yocto PROJECT

Decades of contribution and collaboration

Ingrained culture to give back to the community

Upstream FIRST!

Focus on long term, sustainable and quality products

Upstream and opensource ecosystem in device architecture

Upstream FIRST mentality!

Introduction to the Speakers

Devarsh Thakkar, Software Engineer *at Texas Instruments, Bangalore.*

I work as a Software Engineer at Texas Instruments. My experience has been on linux bootloaders, device drivers and multimedia frameworks such as Gstreamer and I have made contributions to Linux kernel, Gstreamer and Yocto projects.

https://www.linkedin.com/in/devarsh-thakkar-541a6a49/

Overview

- Introduction :
  - What is Video Telephony use-case
  - Design considerations
- Getting the (relevant) basics right :
  - Audio
  - Video
  - Both together: How to maintain AV Sync ??
  - How to Stream them
  - Error recovery
  - Gstreamer tracer
- Getting into action : let’s prototype it :
  - Audio capture
  - Video capture and Streaming
  - Audio telephony
  - Video telephony

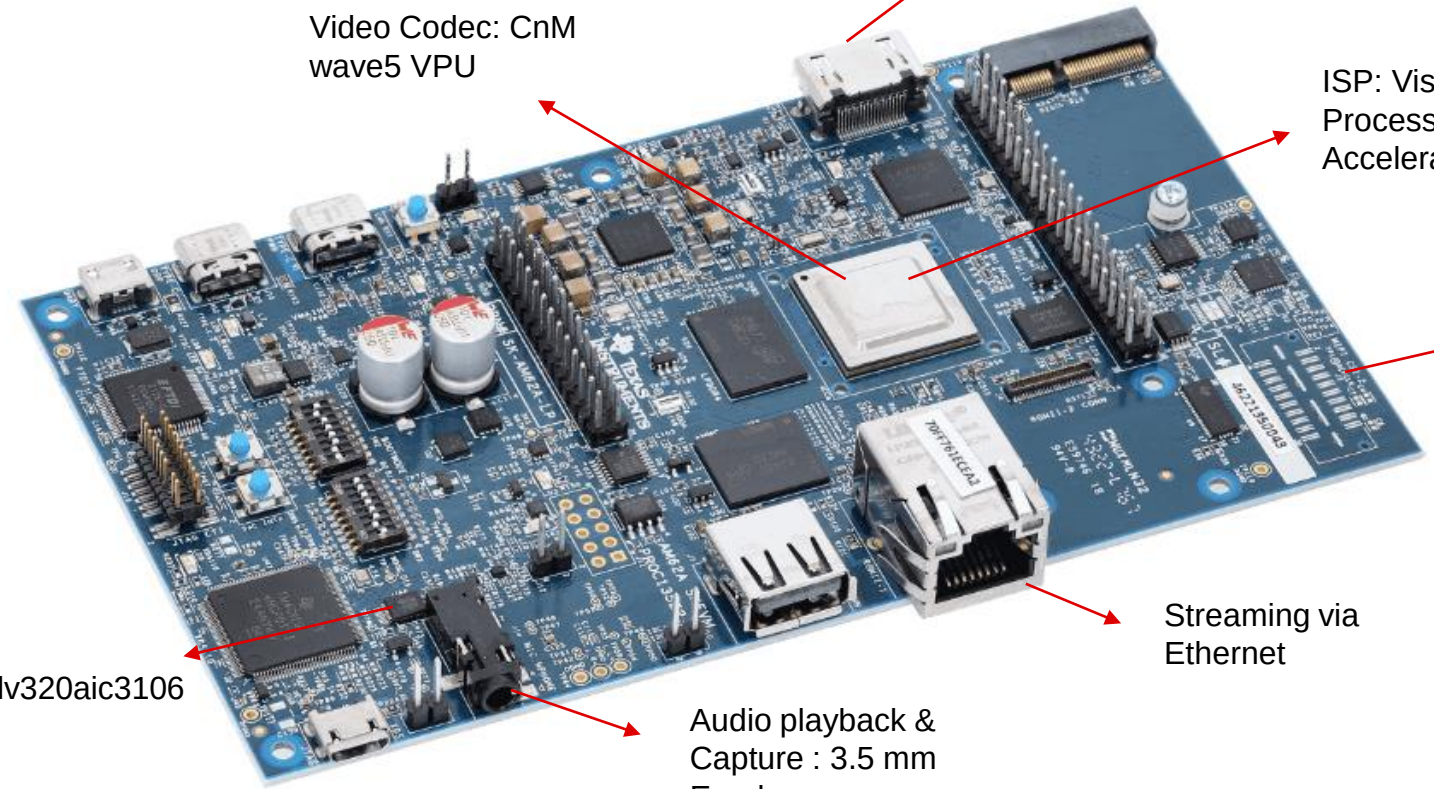
Introduction

Introduction

- What is Video Telephony use-case?
  - Videotelephony is the two-way or multipoint reception and transmission of audio and video signals by people in different locations for real-time communication e.g similar to Skype, WhatsApp Video call.
- What H/W do you need to support this ?
  - Video Capture Interfaces
    - HDMI Camera : High resolution, higher cost, stream live feed on monitor (same end)
    - USB Camera : Lower Cost, Medium resolutions
    - MIPI CSI based Camera : High resolution, Higher speed, Cost effective, Space effective
    - MIPI CSI Camera Sensor types
      - Raw Sensor e.g. imx219 (e.g https://in.element14.com/raspberry-pi/rpi-8mp-camera-board/raspberry-pi-camera-board-v2/dp/3677845)
      - Smart Sensor e.g ov5640 (e.g https://www.technexion.com/products/embedded-vision/mipi-csi2/tevi-ov5640/)

- Video ISP (If using Raw Sensor)
  - Memory-to-Memory transform from bayer->RGB/NV12..mostly proprietary DSP's
- Audio Capture/Record Interfaces
  - USB Camera Mic (Mic H/W does Analog to Digital)
  - PMOD I2S (Mic H/W does Analog to Digital)
  - Earphone (3.5mm Jack) (No Analog to Digital -> Need audio codec)
- Display
  - EDP, HDMI, DSI , OLDI
- Audio Playback
  - HDMI, Earphone, USB
- Video Codec
  - H264, H265, H266, VP9 e.t.c
- Audio Codec (optional)
  - AAC (DSP)
- Network Interface
  - Wifi, Ethernet

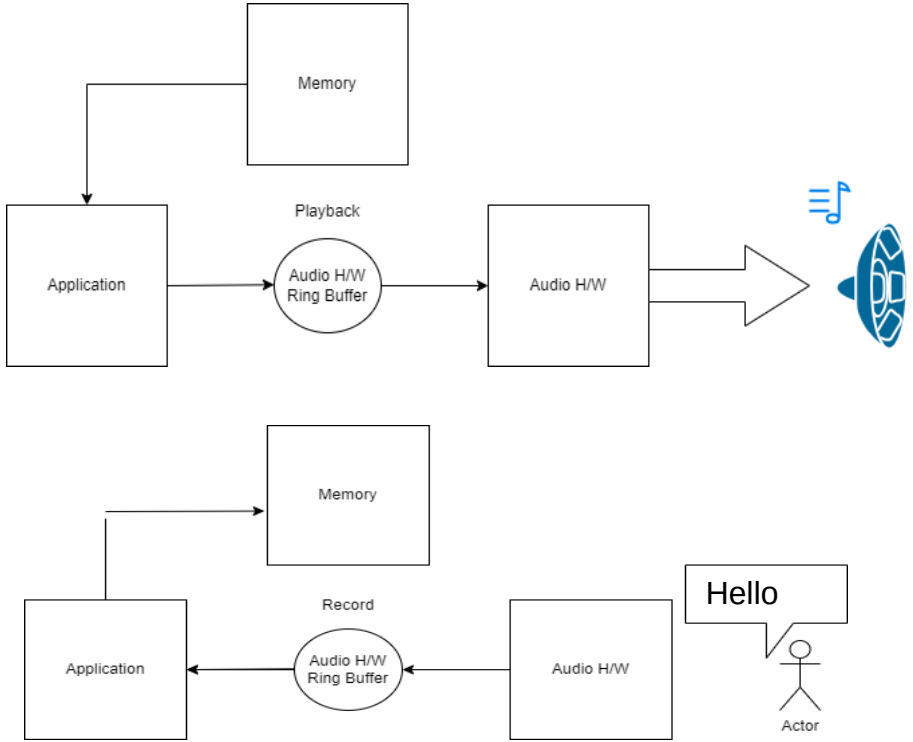
Prototyping with AM62A SK



Design Considerations

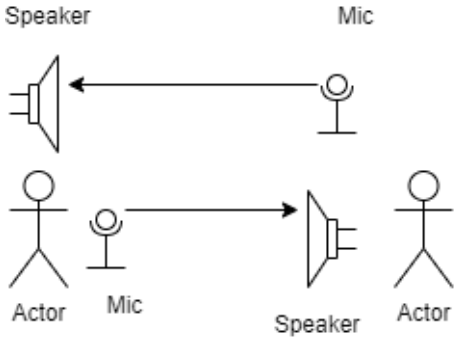
- Audio/Video Sync
- Low Latency
  - Lag/gap should not be recognizable
  - Initial checkout with < 500ms, then reduce it further towards < 100ms
- Noise Reduction
- Echo cancellation
- Error recovery
- Clock Skew
  - Reducing/Avoiding Video Frame Skips/duplications

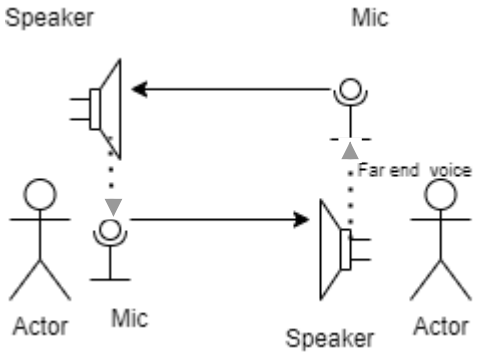
Audio underrun and overrun issue



- Size of audio h/w ring buffer is called **buffer size** and it is quite large so no need to fill it fully before transferring to avoid high latencies
- Audio is transferred b/w application and hardware in chunks of audio frames called "**periods"**
- For playback, If Application is writing faster than audio h/w is consuming, data will get overridden in ring buffer thus causing loss of data and this is called **overrun** issue**.**
- Similarly, if application is writing slower than the rate at which audio h/w is consuming, audio h/w reads all data from ring buffer and this is called **underrun** issue.
- Similar issues can occur on capture side too.
- Gstreamer provides **alsasrc** and **alsasink** plugins for capture and playback which support **latency-time and buffer-time** properties to tweak this params

Audio Echo cancellation and noise reduction





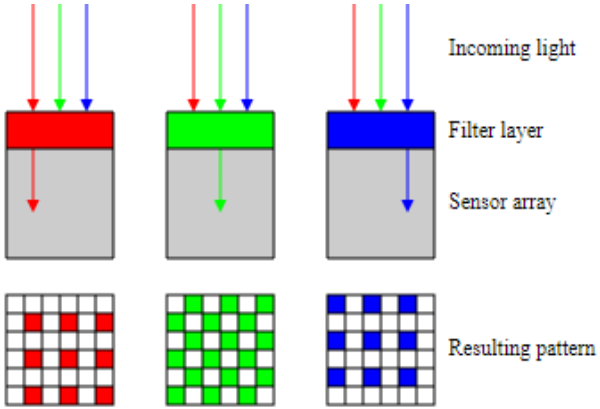
- Echo means speaker hearing their own voice while speaking
- This happens because far end voice (i.e. incoming voice from other side) is caught by receiver mic and transferred back to other side
- Need some intelligence to detect this far end voice and subtract it.
- Gstreamer provides element webrtcdsp for noise reduction and webrtcechoprobe for echo cancellation which use libwebrtcdsp-audio-processing library
- `gst-launch-1.0 far-end-src ! audio/x-raw,rate=48000 ! webrtcechoprobe ! pulsesink \ pulsesrc ! audio/x-raw,rate=48000 ! webrtcdsp ! far-end-sink`

Let’s start the bringup/prototyping!

Video Capture device

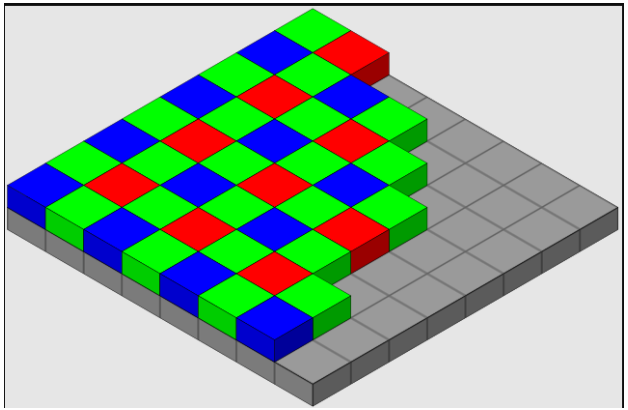
- Video capture device :
  - Captures video data and does a device-to-memory transfer of captured data
  - Zero-copy:
    - The captured data should be transferred to next element in pipe (ISP, Video Codec) using dmabuf (for e.g. v4l2src io-mode=dmabuf or io-mode=dmabuf-import) to maintain performance and reduce latency
  - Driver should support dmabuf import or export with DMA (contiguous) buffer or scatter-gather DMA if MMU is supported
  - Sample pipeline:
    - Capture to a file :
      - gst-launch-1.0 v4l2src io-mode=dmabuf ! video/x-raw, width=1920, height=1080, framerate=30/1, format=NV12 ! filesink location="/run/op_1080p.yuv"
    - Check the performance of your pipeline :
      - gst-launch-1.0 v4l2src io-mode=dmabuf ! video/x-raw, width=1920, height=1080, framerate=30/1, format=NV12 ! fpsdisplaysink name=fpssink video-sink="fakevideosink" text-overlay=false –v
    - For TI K3 platform (capture bayer pattern):
      - yavta -s 1640x1232 -f BGGR /dev/video-rpi-cam0 -c100
      - gst-launch-1.0 -v v4l2src device=/dev/video-rpi-cam0 io-mode=5 ! video/x-bayer,width=1920,height=1080,format=bggr ! filesink location="/run/op.bggr"

Video ISP engine



By Cburnett - Own work, CC BY-SA 3.0,

https://commons.wikimedia.org/w/index.php?curid=1496872



By en:User:Cburnett - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=1496858

bggr

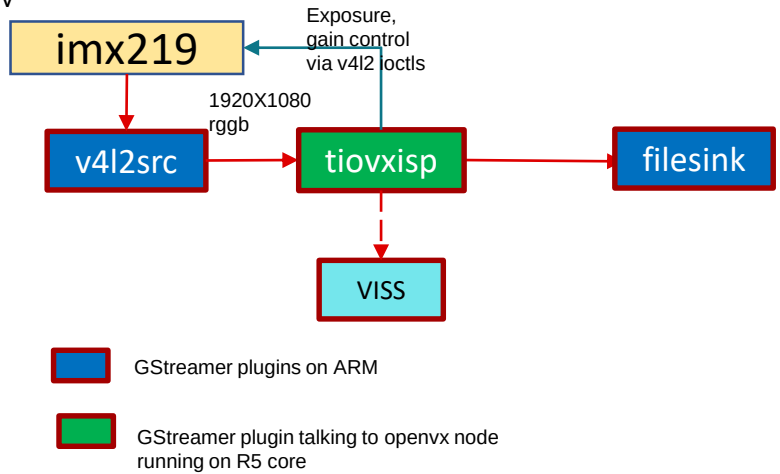
start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 20:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}

NV12

- Raw sensors capture data in bayer format which has twice the green for every red and blue since human eyes are more sensitive to green
- We need an ISP to convert it back to RGB/YUV format since displays/video codecs don't understand bayer format

Video ISP engine

- If you have a V4L2 based mem2mem driver for your ISP then you can directly use Gstreamer's v4l2convert element :
  - gst-launch-1.0 v4l2src num-buffers=5 device=/dev/video0 io-mode=dmabuf ! video/x-bayer,width=1920,height=1080,format=bggr ! v4l2convert output-io-mode=dmabuf-import capture-io-mode=dmabuf ! video/x-raw,format=NV12 ! filesink location="/run/1080p.yuv"
  - For TI K3 platform (capture bayer pattern):
    - gst-launch-1.0 -v v4l2src num-buffers=5 device=/dev/video-rpi-cam0 io-mode=5 ! video/x-bayer,width=1920,height=1080,format=bggr ! tiovxisp sensor-name=SENSOR_SONY_IMX219_RPI dcc-isp-file=/opt/imaging/imx219/linear/dcc_viss_1920x1080.bin sink_0::dcc-2a-file=/opt/imaging/imx219/linear/dcc_2a_1920x1080.bin sink_0::device=/dev/v4l-rpi-subdev0 ! video/x-raw,format=NV12 ! Filesink location="/run/1080p.yuv"



Video Capture device – (Glass-to-Glass latency)

- At this point, we can measure glass-to-glass latency of camera
  - gst-launch-1.0 ! v4l2src io-mode=dmabuf ! video/x-raw, width=1920, height=1080, framerate=30/1 ! kmssink
- For TI K3 platform :
  - gst-launch-1.0 -v v4l2src num-buffers=5 device=/dev/video-rpi-cam0 io-mode=5 ! video/x-bayer,width=1920,height=1080,format=bggr ! tiovxisp sensor-name=SENSOR_SONY_IMX219_RPI dcc-isp-file=/opt/imaging/imx219/linear/dcc_viss_1920x1080.bin sink_0::dcc-2a-file=/opt/imaging/imx219/linear/dcc_2a_1920x1080.bin sink_0::device=/dev/v4l-rpi-subdev0 ! video/x-raw,format=NV12 ! kmssink driver-name=tidss
- Add any sequence identifier (frame-number, timestamp) on an existing encoded video file (mp4, h264, h265) using ffmpeg or gstreamer
  - gst-launch-1.0 videotestsrc ! video/x-raw, width=1920, height=1080, format=NV12 ! timeoverlay ! filesink location="timestamped_1080p_NV12.yuv"
  - Encode back the file using ffmpeg or gstreamer :
    - gst-launch-1.0 filesrc location='timestamped_1080p_NV12.yuv"' ! rawvideoparse width=1920, height=1080, format=NV12 ! x264enc ! filesink location="timestamped_1080p_NV12.h264"
  - Otherwise, if you already have an mp4 which you want to update with frame numbers then use ffmpeg:
    - ffmpeg -i bbb_sunflower_1080p_30fps_normal.mp4 -vf "drawtext=fontfile=Arial.ttf: text='%{frame_num}': start_number=1: x=(w-tw)/2: y=h-(2*lh): fontcolor=black: fontsize=20: box=1: boxcolor=white: boxborderw=5" -c:a copy bbb_sunflower_1080p_30fps_normal_ts.h264
  - Play the time-stamped/frame-sequenced video file on your host machine :
    - ffplay bbb_sunflower_1080p_30fps_normal_ts.h264
- Point your camera towards video being played in your host machine and keep the display i.e. connected to your board besides it so that both screens are side-by-side
- Take a picture now !

Video Capture device – (Glass-to-Glass latency)

For e.g. from this picture,
the latency looks like
2:20:570 – 2:20:620 = ~50ms

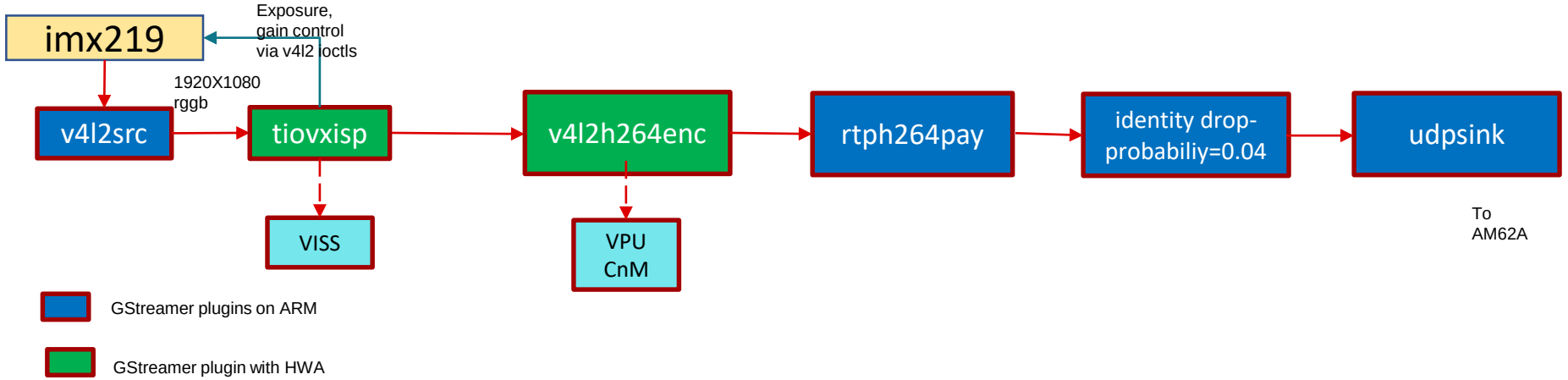
Video Codec

- Video codec device :
  - Compresses the raw data as it need to be streamed via network
  - This should also support zero-copy or dma-buf/dma-fences for optimum performance and latency
- What to take care :
  - Prefer to use hardware encoder as it is needed especially for higher resolutions for real-time encoding
  - Tweak properties :
    - Video encoder (H264/H265) generally has properties like target-bitrate, bitrate-mode, gop-mode, IDR periodicity which can be tweaked as per use-case/bandwidth requirements
      - No b-frames, IDR,P,P,P…IDR,P,P,P with constant bitrate & baseline/constrained baseline profile is recommended
    - If your video encoder supports “video/x-raw, alignment=nal” then use it to optimize latency
    - You can use gdr-mode property too if your codec supports it as helps maintaining consistent packet size for encoded frames
      - In gdr-mode: the I frame is spread across rows of macroblocks of group of pictures

Video Codec - Error recovery

- Need to make sure that decoder hardware/software is robust enough to recover from packet drops happening over the network
  - You can test the robustness of your decoder in real-time using identity element :
- Server :
  - gst-launch-1.0 -v v4l2src device=/dev/video-rpi-cam0 io-mode=5 ! video/x-bayer,width=1920,height=1080,format=bggr ! tiovxisp sensor-name=SENSOR_SONY_IMX219_RPI dcc-isp-file=/opt/imaging/imx219/linear/dcc_viss_1920x1080.bin sink_0::dcc-2a-file=/opt/imaging/imx219/linear/dcc_2a_1920x1080.bin sink_0::device=/dev/v4l-rpi-subdev0 ! video/x-raw,format=NV12 ! v4l2h264enc output-io-mode=dmabuf-import extra-controls="controls,h264_i_frame_period=60,video_gop_size=60" ! queue ! rtph264pay ! **identity drop-probability=0.05** ! udpsink port=5004 buffer-size=60000000 host=<ip_addr_of_receiver>
- Client :
  - gst-launch-1.0 -v udpsrc port=5004 caps = "application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96" ! rtpbin.recv_rtp_sink_1 rtpbin. ! queue ! rtph264depay ! h264parse ! v4l2h264dec capture-io-mode=dmabuf ! queue ! fpsdisplaysink text-overlay=false name=fpssink video-sink="kmssink driver-name=tidss sync=true show-preroll-frame=false force-modesetting=true"

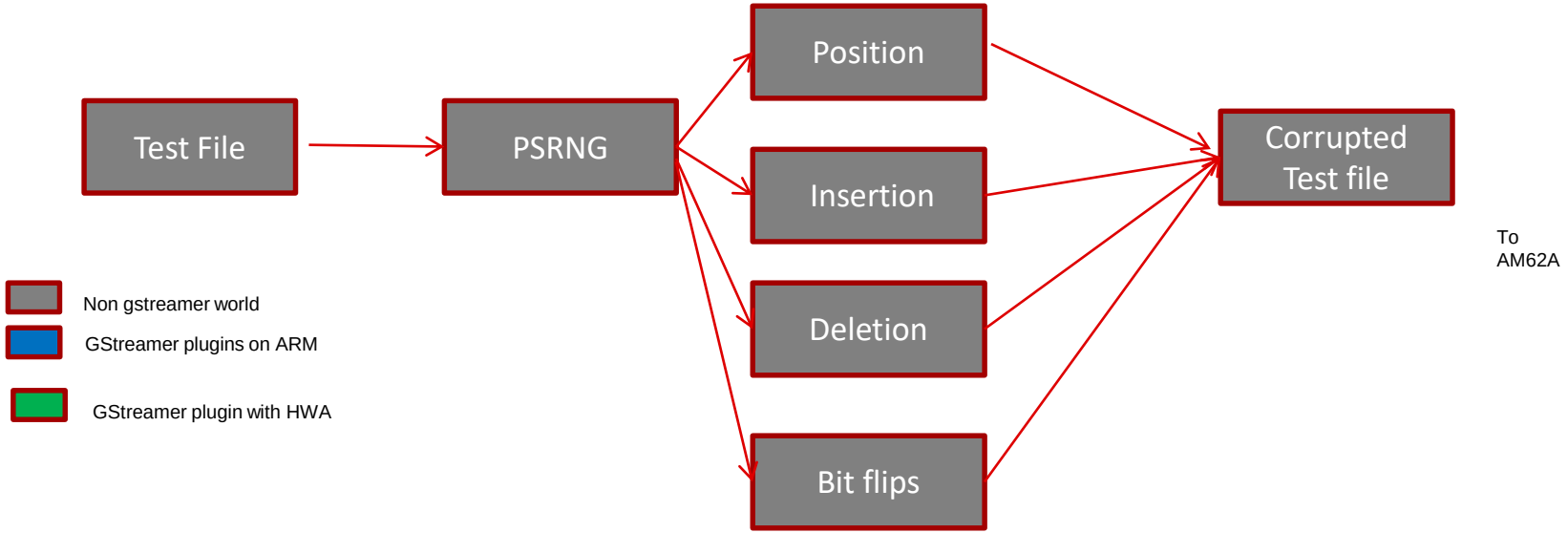
Real time packet drop simulation



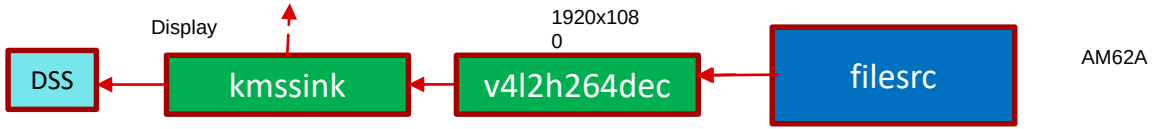
Remote core or HWA

- DSS (Display Subsystem)
- VISS (Vision Imaging Subsystem)

Error injection using PSRNG - Non real time



- DSS (Display Subsystem)
- VISS (Vision Imaging Subsystem)

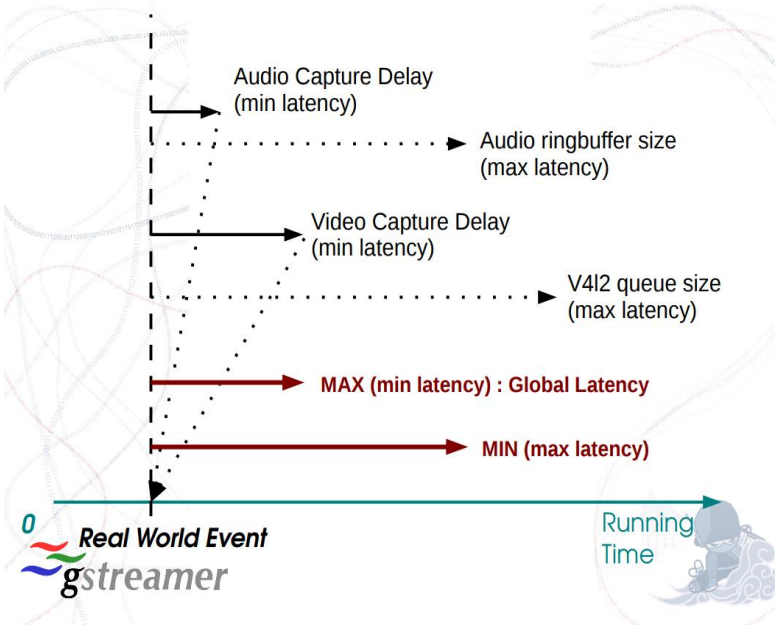


Display

- If using kmssink
  - Check if correct mode is set : use modetest –M <driver_name>
    - Modetest_primary_and_overlay_zpos_1 (github.com)
      - CRTCs:

id	fb	pos	size
38	53	(0,0)	(1920x1080)
  - Check pixel-clock accuracy
    - modetest –s <connector_id>:1920x1080-30@BG24 –v
      - 30 or 29.95 ?
    - modetest –s <connector_id>:1920x1080-29.97@BG24 –v
    - Could cause drift b/w pixel clock and gstreamer clock
  - HACK : Schedule next buffer early/late to compensate for clock skew :
    - kmssink: Add do-timestamp property boilerplate code · devarsht/gst-plugins-bad@572ca71 (github.com)

AV synchronization & Pipeline latency



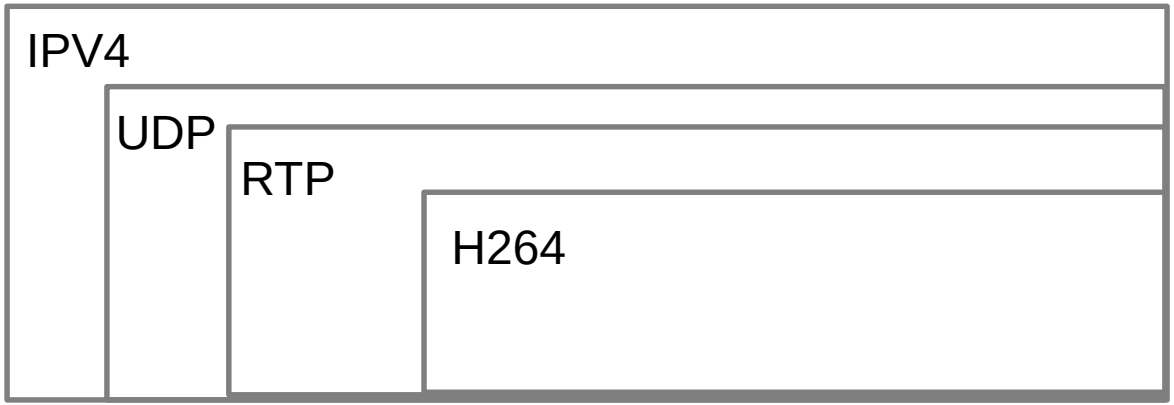
CC: Edward Hervey (gstconf-2013-time-edward-hervey.pdf)

Run AV sync test:
https://www.youtube.com/results?search_query=av+sync+test

- Base time :
  - Time at which pipe starts playing
- Running time
  - Time spent in playing state
- Minimum latency
  - Minimum time needed by element to produce one output unit after receiving input
- Maximum latency
  - Maximum time/buffering element can provide before overflowing
- Reported latency
  - Latency estimate reported by each element beyond which pipe start dropping frames
  - Pipeline latency : GST_DEBUG=*basesink*:6 gst-launch-1.0 <pipe>
  - Individual elements latency : GST_DEBUG=*v4l2*:6,*alsa*:6, gst-launch-1.0 <pipe>

Video Streaming

- RTP : Real time protocol
- RTCP : Real time control protocol
- UDP : Loss-less but fast



Rtpbin

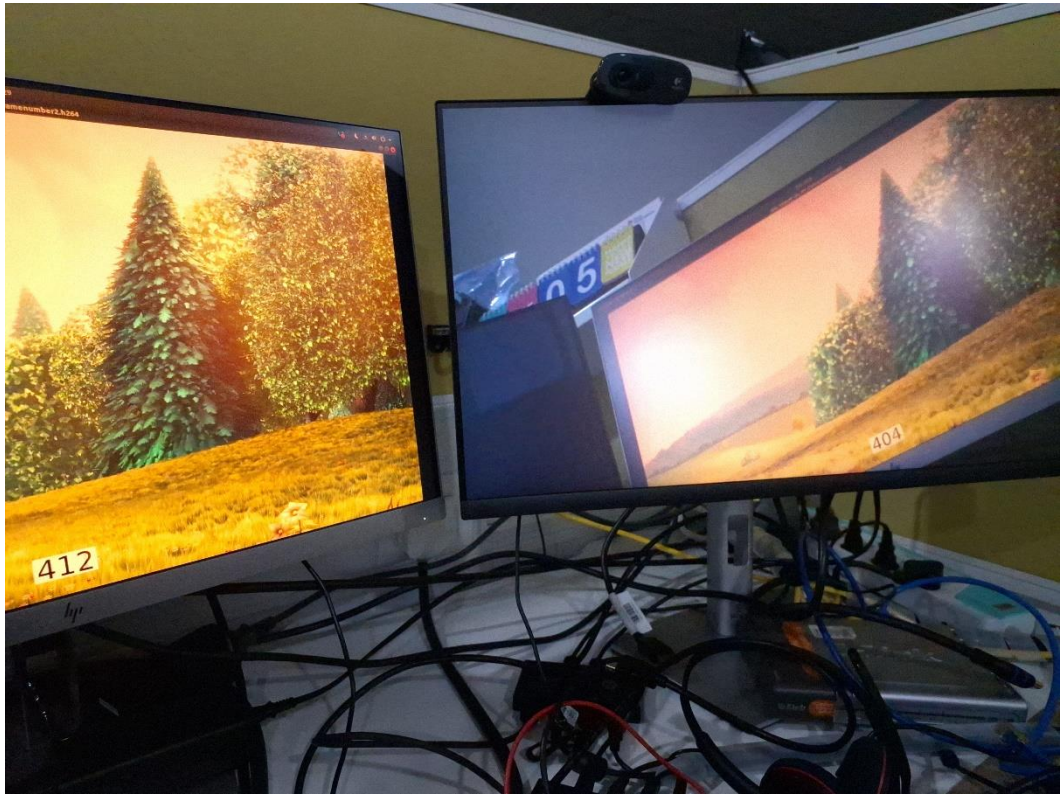
- Rtpsession :
  - Audio/video Server and Client participating on two-way communication will have be modelled as participants of same rtpsession with same SSRC id
    - Packet ordering validation, per participant statistics using RTCP, Sender Report/Receiver Report RTCP packets
- Rtpssrcdemux
  - Demux RTP stream based on SSRC and pass to downstream
- Rtpptdemux
  - Detect payload type and pass corresponding RTP packets to downstream
- Rtpjitterbuffer
  - Re-order and remove duplicate RTP packets as received from network source, configure with “latency” property

Video Streaming

gst-launch-1.0 -v v4l2src device=/dev/video-rpi-cam0 io-mode=5 ! video/x-bayer,width=1920,height=1080,format=bggr ! tiovxisp sensor-name=SENSOR_SONY_IMX219_RPI dcc-isp-file=/opt/imaging/imx219/linear/dcc_viss_1920x1080.bin sink_0::dcc-2a-file=/opt/imaging/imx219/linear/dcc_2a_1920x1080.bin sink_0::device=/dev/v4l-rpi-subdev0 ! video/x-raw,format=NV12 ! **v4l2h264enc output-io-mode=dmabuf-import extra-controls="controls,h264_i_frame_period=60,video_gop_size=60,video_bitrate=15000000, prepend_sps_and_pps_to_idr=1**" ! queue ! rtph264pay config-interval=60 ! rtpbin.send_rtp_sink_0 rtpbin.send_rtp_src_0 ! udpsink port=5004 host=$1 async=false rtpbin.send_rtcp_src_0 ! udpsink host=$1 port=5005 sync=false async=false udpsrc port=5005 ! rtpbin.recv_rtcp_sink_0

gst-launch-1.0 -v rtpbin latency=20 udpsrc port=5004 caps = "application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96" ! rtpbin.recv_rtp_sink_3 rtpbin. ! queue ! rtph264depay ! h264parse ! v4l2h264dec capture-io-mode=dmabuf ! queue ! fpsdisplaysink text-overlay=false name=fpssink video-sink="kmssink driver-name=tidss sync=true show-preroll-frame=false force-modesetting=true" sync=true udpsrc port=5005 ! rtpbin.recv_rtcp_sink_3 rtpbin.send_rtcp_src_3 ! udpsink port=5005 host=$1 sync=false async=false

Video Streaming



Gstreamer: Latency tracer

GST_TRACERS=latency(pipeline+element) GST_DEBUG=GST_TRACERS:7 <pipeline>

tail /run/latency_1.txt

0:02:55.266062147 3853 0x31a8d2a0 TRACE GST_TRACER :0:: latency, src-element-id=(string)0x31b28a30, src-element=(string)udpsrc2, src=(string)src, sink-element-id=(string)0x31b441c0, sink-element

0:00:02.011199484 ^[[32m 3853^[[00m 0x31a8d300 ^[[37mTRACE ^[[00m ^[[00;34m GST_TRACER :0::^[[00m element-latency, element-id=(string)0x31b00500, element=(string)**rtph264depay0**, src=(string)src, time=(guint64)272861, ts=(guint64)2011113553;
0:00:02.011956792 ^[[32m 3853^[[00m 0xffff50001d80 ^[[37mTRACE ^[[00m ^[[00;34m GST_TRACER :0::^[[00m element-latency, element-id=(string)0x31b3aad0, element=(string)**v4l2h264dec0,** src=(string)src, time=(guint64)**133397336**, ts=(guint64)2011830882;
0:00:02.012013453 ^[[32m 3853^[[00m 0x31a8d300 ^[[37mTRACE ^[[00m ^[[00;34m GST_TRACER :0::^[[00m element-latency, element-id=(string)0x31b36630, element=(string)**h264parse0**, src=(string)src, time=(guint64)824439, ts=(guint64)2011937992;
0:00:02.013922387 ^[[32m 3853^[[00m 0x31a909e0 ^[[37mTRACE ^[[00m ^[[00;34m GST_TRACER :0::^[[00m element-latency, element-id=(string)0x31b6a610, element=(string)**rtpsession2**, src=(string)recv_rtp_src, time=(guint64)324556, ts=(guint64)2013802026;
0:00:02.014097693 ^[[32m 3853^[[00m 0x31a909e0 ^[[37mTRACE ^[[00m ^[[00;34m GST_TRACER :0::^[[00m element-latency, element-id=(string)0x31b02d80, element=(string)**rtpstorage2**, src=(string)src, time=(guint64)188281, ts=(guint64)2013990307;
0:00:02.014225828 ^[[32m 3853^[[00m 0x31a909e0 ^[[37mTRACE ^[[00m ^[[00;34m GST_TRACER :0::^[[00m element-latency, element-id=(string)0x31b73320, element=(string)**rtpssrcdemux2**, src=(string)src_3480349229, time=(guint64)153446, ts=(guint64)2014143753;
0:00:02.014466994 ^[[32m 3853^[[00m 0xffff70017760 ^[[37mTRACE ^[[00m ^[[00;34m GST_TRACER :0::^[[00m element-latency, element-id=(string)0xffff9401f5e0, element=(string)**rtpjitterbuffer0**, src=(string)src, time=(guint64)203181, ts=(guint64)2014346934;
0:00:02.014631405 ^[[32m 3853^[[00m 0xffff70017760 ^[[37mTRACE ^[[00m ^[[00;34m GST_TRACER :0::^[[00m element-latency, element-id=(string)0x31b736e0, element=(string)**rtpptdemux0**, src=(string)src_96, time=(guint64)183176, ts=(guint64)2014530110;
0:00:02.014697145 ^[[32m 3853^[[00m 0x31a8d2a0 ^[[37mTRACE ^[[00m ^[[00;34m GST_TRACER :0::^[[00m latency, src-element-id=(string)0x31b28a30, src-element=(string)**udpsrc2**, src=(string)src, sink-element-id=(string)0x31b441c0, sink-element=(string)fpssink, sink=(string)sink, time=(guint64)**150416292**, ts=(guint64)2014585110;

Audio Telephony

- GST_TRACERS="latency(flags=pipeline+element)" GST_DEBUG=GST_TRACER:7 GST_DEBUG_FILE="/run/latency_server.txt" gst-launch-1.0 -v rtpbin name=rtpbin **alsasrc latency-time=20000 buffer-time=40000** ! webrtcdsp ! audioconvert ! audio/x-raw, channels=1, rate=16000, format=S16LE ! audioconvert ! queue ! rtpL16pay pt=97 ! rtpbin.send_rtp_sink_1 rtpbin.send_rtp_src_1 ! udpsink port=5008 host=172.24.227.36 async=false rtpbin.send_rtcp_src_1 ! udpsink host=172.24.227.36 port=5009 sync=false async=false udpsrc port=5009 ! rtpbin.recv_rtcp_sink_1 > /run/1_server.txt 2>&1&GST_TRACERS="latency(flags=pipeline+element)"
- GST_DEBUG_FILE="/run/latency_client.txt“GST_DEBUG=GST_TRACER:7 gst-launch-1.0 -v rtpbin name=rtpbin udpsrc caps="application/x-rtp,media=(string)audio,clock-rate=(int)16000,encoding-name=(string)L16,encoding-params=(string)1,octet-align=(string)1" port=5008 ! rtpbin.recv_rtp_sink_1 rtpbin. ! queue ! rtpL16depay ! audioconvert ! webrtcdsp echo-cancel=false ! audioresample ! **alsasink device="hw:0,0" latency-time=20000 buffer-time=40000** " udpsrc port=5009 ! rtpbin.recv_rtcp_sink_1 rtpbin.send_rtcp_src_1 ! udpsink port=5009 host=172.24.227.36 sync=false async=false > /run/1_client.txt 2>&1&
- For removing echo cancellation :
- gst-launch-1.0 -v rtpbin latency=20 name=rtpbin **alsasrc latency-time=20000 buffer-time=40000** device="hw:0,0" ! webrtcdsp noise-suppression-level=high echo-suppression-level=high ! audioconvert ! audio/x-raw, channels=1, rate=16000, format=S16BE ! queue ! rtpL16pay pt=97 ! rtpbin.send_rtp_sink_1 rtpbin.send_rtp_src_1 ! udpsink port=5008 host=$1 async=false rtpbin.send_rtcp_src_1 ! udpsink host=$1 port=5009 sync=false async=false udpsrc name=audio caps="application/x-rtp,media=(string)audio,clock-rate=(int)16000,encoding-name=(string)L16,encoding-params=(string)1,octet-align=(string)1" port=5008 ! rtpbin.recv_rtp_sink_4 rtpbin. ! queue ! rtpL16depay ! audioconvert ! webrtcechoprobe ! **alsasink device="hw:0,0" latency-time=20000 buffer-time=40000** udpsrc port=5009 ! rtpbin.recv_rtcp_sink_4 rtpbin.send_rtcp_src_4 ! udpsink port=5009 host=$1 sync=false async=false

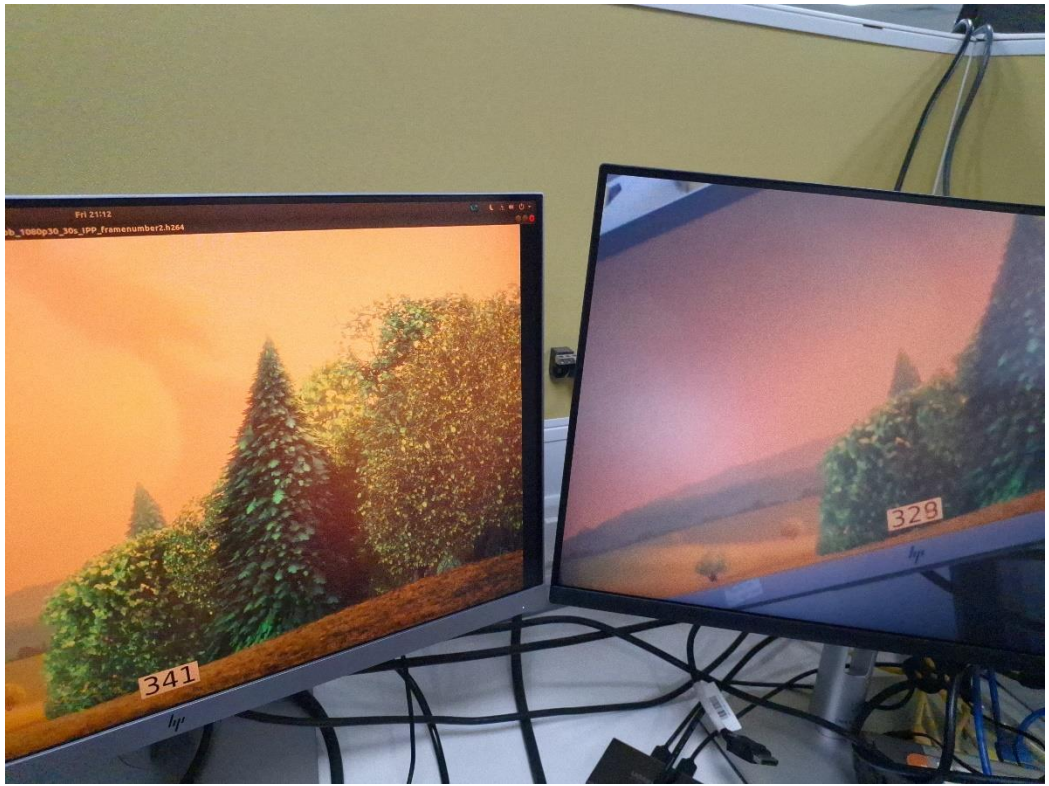
Audio + Video = Video Telephony (with opusenc)

- gst-launch-1.0 -v **rtpbin latency=100 name=rtpbin v4l2src device=/dev/video-rpi-cam0 io-mode=dma-buf** ! video/x-bayer,width=1920,height=1080,format=bggr ! tiovxisp sensor-name=SENSOR_SONY_IMX219_RPI dcc-isp-file=/opt/imaging/imx219/linear/dcc_viss_1920x1080.bin sink_0::dcc-2a-file=/opt/imaging/imx219/linear/dcc_2a_1920x1080.bin sink_0::device=/dev/v4l-rpi-subdev0 ! video/x-raw,format=NV12 ! **v4l2h264enc output-io-mode=dmabuf-import extra-controls="controls,h264_i_frame_period=60,video_gop_size=60,video_bitrate=15000000, prepend_sps_and_pps_to_idr=1**" ! queue ! rtph264pay config-interval=60 ! rtpbin.send_rtp_sink_0 rtpbin.send_rtp_src_0 ! udpsink port=5004 host=$1 async=false rtpbin.send_rtcp_src_0 ! udpsink host=$1 port=5005 sync=false async=false udpsrc port=5005 ! rtpbin.recv_rtcp_sink_0 **alsasrc latency-time=20000 buffer-time=40000** device="hw:0,0" ! audio/x-raw, channels=2, rate=48000, format=S16LE ! webrtcdsp noise-suppression-level=high echo-suppression-level=high ! **opusenc** ! rtpopuspay ! rtpbin.send_rtp_sink_1 rtpbin.send_rtp_src_1 ! udpsink port=5008 host=$1 async=false rtpbin.send_rtcp_src_1 ! udpsink host=$1 port=5009 sync=false async=false udpsrc port=5009 ! rtpbin.recv_rtcp_sink_1 udpsrc port=5004 caps = "application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96" ! rtpbin.recv_rtp_sink_3 rtpbin. ! queue ! rtph264depay ! h264parse ! v4l2h264dec capture-io-mode=dmabuf ! queue ! fpsdisplaysink text-overlay=false name=fpssink video-sink="**kmssink driver-name=tidss sync=true show-preroll-frame=false force-modesetting=true**" sync=true udpsrc port=5005 ! rtpbin.recv_rtcp_sink_3 rtpbin.send_rtcp_src_3 ! udpsink port=5005 host=$1 sync=false async=false udpsrc name=audio caps="application/x-rtp,media=(string)audio,clock-rate=(int)48000,encoding-name=(string)OPUS,encoding-params=(string)1,octet-align=(string)1" port=5008 ! rtpbin.recv_rtp_sink_4 rtpbin. ! queue ! rtpopusdepay ! opusdec ! webrtcechoprobe ! **alsasink device="hw:0,0" latency-time=20000 buffer-time=40000** udpsrc port=5009 ! rtpbin.recv_rtcp_sink_4 rtpbin.send_rtcp_src_4 ! udpsink port=5009 host=$1 sync=false async=false

Audio + Video = Video Telephony (with raw audio)

- gst-launch-1.0 -v rtpbin latency=20 name=rtpbin v4l2src device=/dev/video-rpi-cam0 io-mode=5 ! video/x-bayer,width=1920,height=1080,format=bggr ! tiovxisp sensor-name=SENSOR_SONY_IMX219_RPI dcc-isp-file=/opt/imaging/imx219/linear/dcc_viss_1920x1080.bin sink_0::dcc-2a-file=/opt/imaging/imx219/linear/dcc_2a_1920x1080.bin sink_0::device=/dev/v4l-rpi-subdev0 ! video/x-raw,format=NV12 ! v4l2h264enc output-io-mode=dmabuf-import **extra-controls="controls,h264_i_frame_period=60,video_gop_size=60,video_bitrate=15000000, prepend_sps_and_pps_to_idr=1**" ! queue ! rtph264pay config-interval=60 ! rtpbin.send_rtp_sink_0 rtpbin.send_rtp_src_0 ! udpsink port=5004 host=$1 async=false rtpbin.send_rtcp_src_0 ! udpsink host=$1 port=5005 sync=false async=false udpsrc port=5005 ! rtpbin.recv_rtcp_sink_ **alsasrc latency-time=20000 buffer-time=40000** device="hw:0,0" ! webrtcdsp noise-suppression-level=high echo-suppression-level=high ! audioconvert ! audio/x-raw, channels=1, rate=16000, format=S16BE ! queue ! rtpL16pay pt=97 ! rtpbin.send_rtp_sink_1 rtpbin.send_rtp_src_1 ! udpsink port=5008 host=$1 async=false rtpbin.send_rtcp_src_1 ! udpsink host=$1 port=5009 sync=false async=false udpsrc port=5009 ! rtpbin.recv_rtcp_sink_1 udpsrc port=5004 caps = "application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96" ! rtpbin.recv_rtp_sink_3 rtpbin. ! queue ! rtph264depay ! h264parse ! v4l2h264dec capture-io-mode=dmabuf ! queue ! fpsdisplaysink text-overlay=false name=fpssink video-sink ="**kmssink driver-name=tidss sync=true show-preroll-frame=false force-modesetting=true**" sync=true udpsrc port=5005 ! rtpbin.recv_rtcp_sink_3 rtpbin.send_rtcp_src_3 ! udpsink port=5005 host=$1 sync=false async=false udpsrc name=audio caps="application/x-rtp,media=(string)audio,clock-rate=(int)16000,encoding-name=(string)L16,encoding-params=(string)1,octet-align=(string)1" port=5008 ! rtpbin.recv_rtp_sink_4 rtpbin. ! queue ! rtpL16depay ! audioconvert ! webrtcechoprobe ! **alsasink device="hw:0,0"** " **latency-time=20000 buffer-time=40000** udpsrc port=5009 ! rtpbin.recv_rtcp_sink_4 rtpbin.send_rtcp_src_4 ! udpsink port=5009 host=$1 sync=false async=false

Audio + Video = Video Telephony



What next :

- Optimize video codec params
- Optimize debayering pipe
- Leaky queue ?
- decodebin ?
- More aggressive audio latency time/buffer time

Credits and Acknowledgement

- Texas Instruments Inc.
- Gstreamer conference committee

References

- https://gstreamer.freedesktop.org/documentation/rtpmanager/rtpbin.html?gi-language=c
- https://gstreamer.freedesktop.org/documentation/rtpmanager/rtpsession.html?gi-language=c
- https://www.collabora.com/news-and-blog/blog/2020/08/20/paving-the-way-high-bitrate-video-streaming-gstreamer-rtp-elements/
- GStreamer Echo Canceller (collabora.com)
- people.freedesktop.org/~bilboed/gstconf-2013-time-edward-hervey.pdf
- GStreamer Conference 2023 (24-26 September 2023): RidgeRun / Texas Instruments: Edge AI GStreamer Plugins · Indico

Q&A

- Contact Information:
  - Devarsh Thakkar <devarsht@ti.com>
- Also on IRC @ libera.chat #linux-ti
  - devarsht

Learn more about TI products

- https://www.ti.com/linux
- http://opensource.ti.com/
- https://www.ti.com/processors
- https://www.ti.com/edgeai

Why choose TI MCUs and processors?

Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.