WHITE PAPER

GStreamer Optimized Multimedia Processing for Audio and Video

abaco SYSTEMS

GStreamer Optimized Multimedia Processing for Audio and Video

GStreamer is a platform-independent (Microsoft® Windows®, Linux®, Android™, OS X®, BSD, OpenSolaris) multimedia framework for constructing modular processing pipelines using an extensible plug-in architecture. Applications range from simple Ogg Vorbis playback and audio/video streaming to complex audio (mixing), video (non-linear editing) processing, metadata, subtitles and much more.

Applications built on the GStreamer framework can take advantage of advances in codec and filter technology transparently. Developers and hardware vendors can add new codecs and filters by writing a simple plug-in with a clean, generic interface. GStreamer is released under the GNU LGPL.

Background

The GStreamer project started life in 1999 and since that time has gained significant commercial recognition. Designed for used in small embedded devices, it has been adopted by the mobile phone industry to manage compression and decompression of video and today is widely used in Android tablets and cell phones.

GStreamer is a powerful multimedia framework that is both a successful open source project and commercial middleware due to both its extensibility and rich set of readily available plug-ins. It has been adopted in a broad range of products from the like of Intel, NVIDIA, Nokia, Motorola, Texas Instruments and Freescale.

The API and libraries are written in C making them very portable. Internally, the code uses an object system called GObject to maximize code reuse and modularity while maintaining cross-platform support. The project aims to provide developers with a clean and powerful interface, an object-oriented design approach, extensible framework with dynamically loaded plug-ins, binary only plug-in deployment and a high performance plug-in architecture with minimal overhead and allowing for hardware acceleration.

Today, GStreamer is used in many open source applications including Totem, Rhythmbox and Songbird as well as being heavily utilized by the GNOME desktop environment under Linux.

Roll your own Plug-in

Increasingly in bespoke solutions, we see the need to create something new to handle media in a specific way. This can be for many reasons such as to allow interoperability with a new device (e.g. camera, capture card) or to handle a specific media format. Using the GStreamer plug-in template, it is possible to manipulate the media and convert it into something more usable.

It does not take long to import video into a new GStreamer source element (using the provided templates) from a new video source and manipulate the video buffer into a standard format so that it can be processed. Below, GStreamer is

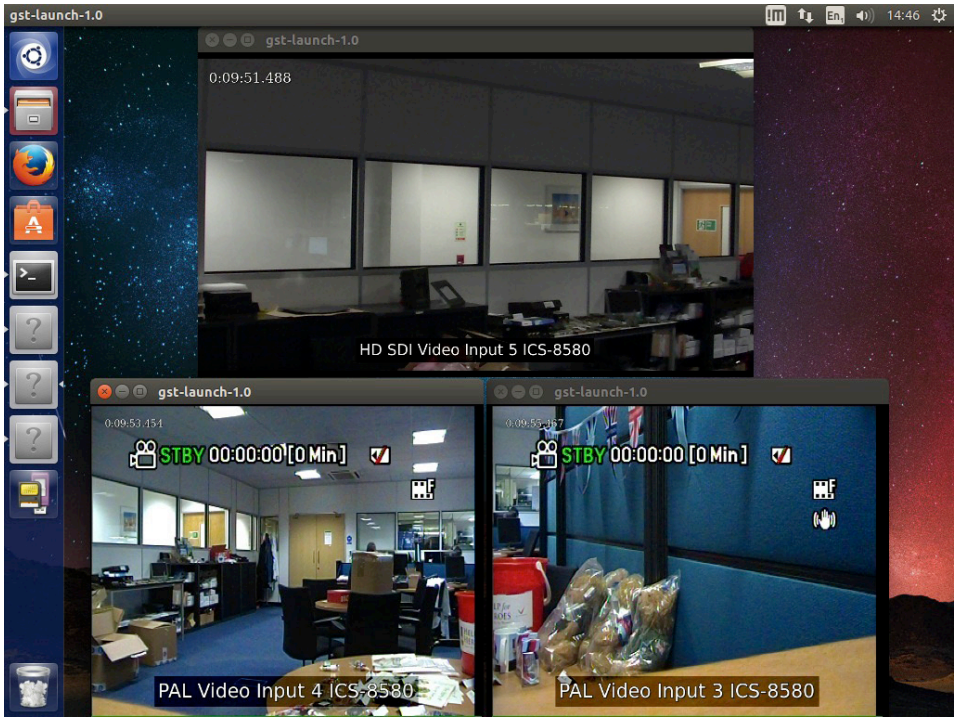


Figure 1 - ICS8580 / DAQMAG2A – GStreamer for video input

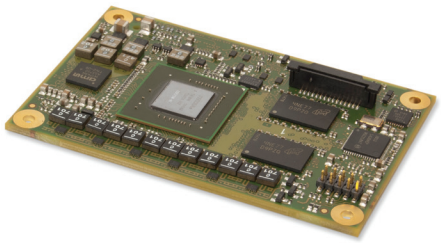
being used to acquire video data from the Abaco Systems ICS-8580 video capture and compression XMC. Three separate video inputs (two PAL and one HD SDI) are being processed using a single GStreamer plug-in; the pipeline then applies some text overlay before being displayed on the screen. As it was only necessary to deal with the image acquisition (the GStreamer source element) it was then possible to very quickly display, manipulate, compress and finally stream the incoming video, leveraging the power of the existing GStreamer plug-ins.

Extensibility and future proofing

The future will bring 4K, Ultra HD, H.265, HDR (High Dynamic Range) and other emerging formats and these can easily be accommodated using the GStreamer framework with the addition of more efficient encoders/decoders adding to the already rich set of open source plug-ins.

These innovations will require increased processing capability and it will be the hardware vendors who will step up and handle the burden of media processing, freeing the CPU to perform other tasks. Today, products such as the Abaco Systems mCOM10-K1 NVIDIA Tegra ARM-based system-on-module (SoM) provide hardware acceleration for H.264 streaming (using OpenMAX1-enabled GStreamer plug-ins) while delivering up to 97% compression of video streams.

Typical applications for this include network-enabled video recording/ playback and streaming video where bandwidth is limited (e.g. wireless RF) from remote locations.



Uses of the mCOM10-K1 are broad ranging and, when combined with the 192 GPGPU (general purpose computing on graphics processing unit) cores, it is possible to undertake detailed analysis of video data to enable increased autonomy and reduce the operator load inside ground vehicles. In both commercial and military applications, smaller embedded processors are paving the way towards greater autonomy and increasing safety.

Abaco Systems utilizes the GStreamer framework inside its network-enabled recording solution. This has enabled a key customer to capture, analyze, manipulate and store over 6 Terabytes of intelligence data, including a vast array of still- and moving images, and to share this intelligence in real, or very near real, time, depending on requirement.

Multimedia processing examples

Below is shown one of the simplest pipelines that is possible. It takes the YUYV-encoded raw video directly from the source (in this case the ICS-8580) shown in red and passes it unmodified (via the capabilities filter shown in green) to the sink shown in purple, to be displayed on the screen. Figure 2 is a pipeline (graph) showing this processing.

The following examples use the macros defined below (PAL resolution at 24fps) and can be modified as required:

```
export MULTICAST=239.192.1.114
$ Multicast IP Address
export PORT=5004
$ IP Port number
export WIDTH=720
export HEIGHT=576
export RATE=24
export BITRATE=8000000
```

All the available plug-ins installed on a system can be seen by typing:

```
$ gst-inspect-1.0
```

To quickly test a pipeline, the gst-launch-1.0 command from the command line can be used. To create a synthetic video source for testing purposes, the videotestsrc source element can be used and the result displayed on the screen using the xvimagesink sink element:

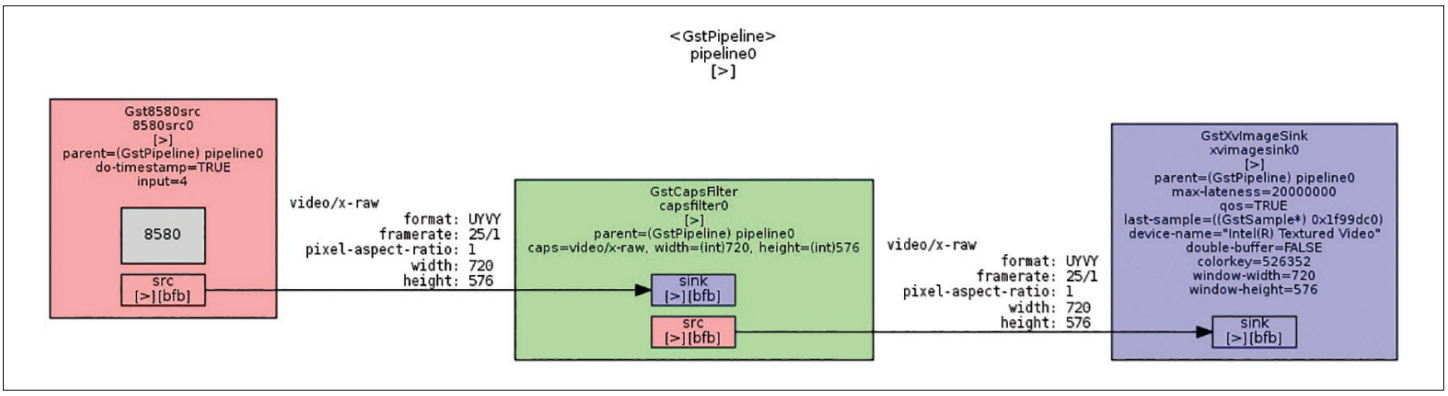


Figure 2 GStreamer example pipeline (PAL video import)

```
$ gst-launch-1.0 -v
videotestsrc pattern=snow !
"video/x-raw, width=${WIDTH},
height=${HEIGHT}," ! xvimagesink
```

To use a different source (e.g. ICS-8580 as a video capture card) the source element can be changed as follows:

```
$ gst-launch-1.0 -v 8580src
input=4 type=1 res=2 channel=0
! "video/x-raw, width=${WIDTH},
height=${HEIGHT}, " !
xvimagesink
```

In the remaining examples, the video test source element will be used to illustrate the examples.

This example shows how raw video can be streamed (multicast) over Ethernet using RTP (Real Time Protocol). The 'tee' plug-in is used to fork the video to the screen and the UDP sink.

```
$ gst-launch-1.0 -q
${SOURCE} ! "video/x-raw,
width=${WIDTH}, height=${HEIGHT},
framerate=${RATE}/1,
format=(string)UYVY" ! tee
name=t ! queue ! rtpvrawpay
! udpsink host=${MULTICAST}
port=${PORT} t. ! xvimagesink
```

This pipeline can receive the incoming video and overlay some text:

```
$ gst-launch-1.0 udpsrc
port=${PORT} multicast-
group=${MULTICAST}
caps="application/x-
rtp, media=(string)video,
encoding-name=(string)
RAW, sampling=(string)
YCbCr-4:2:2, depth=(string)8,
width=(string)${WIDTH},
height=(string)${HEIGHT},
payload=(int)96,
framerate=${RATE}/1" ! queue
! rtpvrawdepay ! textoverlay
halignment=2 shaded-
background=true text="Playing
RAW..." ! xvimagesink
```

To optimize bandwidth efficiency, the H.264 media encoder can be used to compress/decompress the video data. In this last example, the OpenMax (omx) media encoders provided by NVIDIA (supported on the mCOM10-K1) can be used to offload the video compression:

```
$ gst-launch-1.0 -v
videotestsrc ! "video/x-raw,
width=${WIDTH}, height=${HEIGHT},
framerate=${RATE}/1,
format=(string)UYVY" ! tee
name=t ! queue ! videoconvert
! omxh264enc control-rate=2
target-bitrate=${BITRATE}
! rtph264pay! udpsink
host=${MULTICAST} port=${PORT}
t. ! xvimagesink
```

The following pipeline can receive the incoming compressed video stream, decompress it, overlay some text and display the result:

```
$ gst-launch-1.0 -v udpsrc
port=${PORT} multicast-
group=${MULTICAST}
caps="application/x-
rtp, media=(string)video,
encoding-name=(string)
RAW, sampling=(string)
YCbCr-4:2:2, depth=(string)8,
width=(string)${WIDTH},
height=(string)${HEIGHT},
payload=(int)96,
framerate=${RATE}/1" ! queue
! rtph264depay ! omxh264dec
! textoverlay halignment=2
shaded-background=true
text="Playing H.264..." !
xvimagesink
```

If a platform does not have omx-enabled plug-ins, software encoders can be used as a fall back (x264enc is a software only encoder for H.264).

In this final example, a raw video stream is captured, compressed and stored in a file for later playback.

```
$ gst-launch-1.0 udpsrc
port=${PORT} multicast-
group=${MULTICAST}
caps="application/x-
rtp, media=(string)video,
encoding-name=(string)
RAW, sampling=(string)
YCbCr-4:2:2, depth=(string)8,
width=(string)${WIDTH},
height=(string)${HEIGHT},
payload=(int)96,
framerate=${RATE}/1" !
rtpvrawdepay ! nvvidconv !
capsfilter caps="video/x-
nvrm-yuv,format=(fourcc)
I420" ! nv _ omx _ h264enc
bitrate=${BITRATE} ! queue
! matroskamux ! filesink
location=MyCompressedVideo.mkv
```

Video files can then be played back offline using a standard media player such as VLC Media Player2.

For more detailed examples on how to accelerate media processing pipelines using the OpenMAX-accelerated NVIDIA plug-ins on the mCOM10-K1, please refer to the NVIDIA Technical Note 'Jetson TK1/TEGRA Linux Driver Package Multimedia User Guide'.

Creating pipeline diagrams automatically

Pipeline diagrams for documentation or debugging can be created by following the steps below. On Debian3-based Linux systems, install the prerequisites using apt-get (graphviz contains the dot command needed later on):

```
$ sudo apt-get install graphviz
```

Define where the output files are to be sent:

```
$ export GST _ DEBUG _ DUMP _
DOT _ DIR=/tmp/
```

Run the pipeline (using gst-launch-1.0). When done, do an 'ls /tmp' and the following should appear:

```
0.00.00.972540004-gst-launch.
NULL _ READY.dot
0.00.01.051387461-gst-launch.
READY _ PAUSED.dot
0.00.01.074729712-gst-launch.
PAUSED _ PLAYING.dot
0.00.12.187852589-gst-launch.
PLAYING _ PAUSED.dot
0.00.12.201485839-gst-launch.
PAUSED _ READY.dot
psplash _ fifo
```

Generate the diagram from the dot file using the command below:

```
$ dot -Tpng
0.00.24.846778049-gst-launch.
PLAYING _ PAUSED.dot > pipeline.
png
```

Open the image using a favorite viewer

```
$ eog pipeline.png
```

Application Integration

The gst-launch-1.0 command line tool is useful for testing pipelines but ultimately users will want to include these pipelines in their application code and create a machine interface around the functionality. The GStreamer API usage is described in the Applications Developers Guide and allows the pipeline to be manipulated dynamically in a C/C++ application. There are also QtGstreamer4 wrappers available to integrate GStreamer into the Qt5 application development framework to allow platform-agnostic applications that can be run under Linux or Windows.

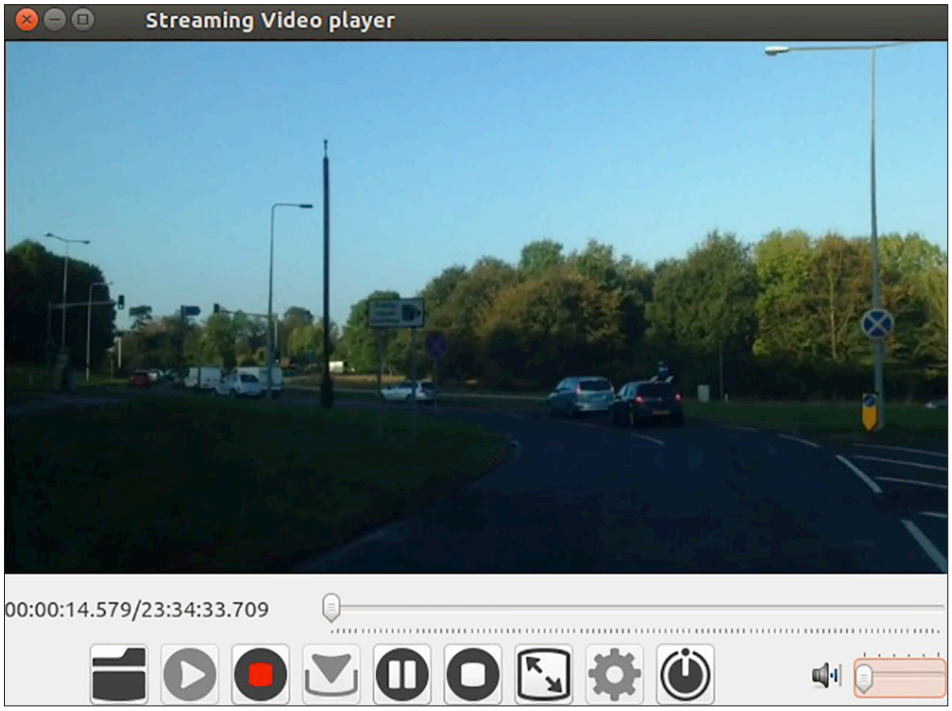


Figure 3 Video Player build on Gstreamer and Qt

To see how Abaco Systems products can help with media processing applications, please contact Abaco's regional sales manager and ask for a demonstration. For more information on GStreamer, please visit the website and project home page. http://GStreamer.freedesktop.org/

- https://www.khronos.org/openmax/
- http://www.videolan.org/vlc
- https://www.debian.org/
- http://GStreamer.freedesktop.org/modules/qt-GStreamer.html
- http://www.qt.io/