mipi-gstreamer-1

February 10, 2021

1 Using MIPI Sensors with GStreamer

1.1 Part 1

Import the GStreamer module:

[ ]:
```
import gi
gi.require_version("Gst", "1.0")
from gi.repository import Gst

Gst.init(())
```

GStreamer works with graphs of media-handling components, called "Pipelines". Components could be:

- *source*-elements that create media samples
- *intermediate*-elements that process, transform, analyze or route the samples
- *sink*-elements that finally consumes the samples

The *source* and *sink* elements are mandatory.

The source element for camera modules connected via the CSI-2 port ("MIPI modules") is called **nvarguscamerasrc**. Let' s create a simple pipeline that just produces and consumes data from the camera:

[ ]:
```
pipeline = Gst.parse_launch("nvarguscamerasrc name=src ! appsink max-buffers=1␣
↪name=sink")
```

Tip:

The max-buffers property should always be set on the appsink because it would otherwise store an unlimited amount of buffers if they are not consumed by the application. This will cause the system to go out-of-memory sooner or later.

Now we could start the pipeline. This will set up the camera and the source element will start producing media samples (ie.: images).

[ ]:
```
pipeline.set_state(Gst.State.PLAYING)
```

`set_state` returned `GST_STATE_CHANGE_ASYNC` which means that the whole process of setting up the camera and starting the video stream is now running in the background while our python code continues. Let's make sure, everything is ready before we continue:

[ ]:
```
pipeline.get_state(Gst.CLOCK_TIME_NONE)[0] == Gst.StateChangeReturn.SUCCESS and␣
↪"OK" or "FAILED!"
```

From our simple pipeline, we can extract the image data from the *appsink* element. This can be done by activating the "try-pull-sample" action. It gets a timeout value as a parameter.

[ ]:
```
sink = pipeline.get_by_name("sink")

sample = sink.emit("try-pull-sample", 1 * Gst.SECOND)
print("Caps:", sample.get_caps().to_string())
```

The appsink element returns a `GstSample` object. This object contains metadata like the `GstCaps` object that we just printed. It also contains the actual buffer which holds the image data.

[ ]:
```
buffer = sample.get_buffer()
print("Got a buffer of size: %d Bytes" % (buffer.get_size()))
```

Note that the buffer size is only a few bytes. Having a look at the `GstCaps` again it says that the video format is:

video/x-raw(**memory:NVMM**), width=(int)1920, height=(int)1080

NVMM memory is accessible from the GPU subsystem of the Tegra SoC. This allows for fast processing of the data using the GPU, ISP or video encoder hardware on the SoC. On the downside, this memory is not directly accessible from Python. So we could pass this buffer on to other GStreamer components that can work with NVMM memory but the Python script itself can not access the data.

For now, we stop the pipeline to release the camera:

[ ]:
```
pipeline.set_state(Gst.State.NULL)
```

[ ]:
```

```