Freescale Semiconductor
User Guide

Document Number:IMX6YMMUG
Rev L3.10.17_1.0.0-ga, 05/2014

i.MX 6 Series Yocto Project Multimedia User's Guide

Contents

1 About This Book

This document describes how to build Freescale Multimedia components (Gstreamer plugins, decoder/encoder, and demuxer) with Yocto Project, and how to run various multimedia usage cases by Gstreamer command lines. Users can refer to these command lines to create their multimedia products.

1.1 Audience

This document is intended for software, hardware, and system engineers who are planning to use multimedia codecs with GStreamer architecture and for anyone who wants to understand more about multimedia codecs. The document assumes that the user has a basic understanding of GStreamer and Linux architecture.

1.2 Conventions

This document uses the following conversions:

- `Courier New` font: This font is used to identify commands, explicit command parameters, code examples, expressions, data types, and directives.
- $ Sign: It is used to specify replaceable command parameters.

1.3 References

- i.MX 6 SABRE-SD Linux User's Guide
- i.MX 6 SABRE-AI Linux User's Guide
- i.MX 6SoloLite EVK Linux User's Guide
- Freescale Yocto Project User Guide
- i.MX 6 Series Yocto Project Multimedia Release Notes

2 Building Multimedia Packages

This chapter describes how to set up the Yocto Project build environment and how to build multimedia packages into a Yocto Project image.

2.1 Building Yocto Project

See the *Freescale Yocto Project User's Guide* for how to set up the Yocto Project environment and how to build a Yocto Project image.

2.2 Building Freescale Multimedia Components

This section describes the Freescale multimedia components and how to build them.

2.2.1 Freescale multimedia packages

Due to the license limitation, Freescale multimedia packages consist of three parts:

- Standard package: no license limitation packages
- Special package: license limitation packages
- Excluded package: license limitation packages

For each package details, see the *i.MX 6 Series Yocto Multimedia Release Notes*.

2.2.2 Building standard packages

Standard multimedia packages are built into Yocto images by default.

If you want to update a package and build it, you can put it under the `downloads` directory and perform the following commands:

```
$ bitbake -c cleanall $packagename
```

```
$ bitbake $packagename
```

The package name should be identical to the recipe name (under `sources/meta-fsl-arm/recipes-multimedia/$component/$packagename_$version.bb`).

For example,

```
$ bitbake gst-fsl-plugin
```

2.2.3 Building special and excluded packages

Place the special or excluded packages in the `downloads` directory and read the `readme` file in each package.

For example, README-microsoft in the package `fslcodec-microsoft-$version.tar.gz`.

3 Multimedia User Cases

3.1 Playbacks

Playbacks include the following:

- Audio only playback
- Video only playback
- Audio/Video file playback
- Other methods for playback

3.1.1 Audio only playback

```
gst-launch filesrc location=$clip_name [typefind=true] ! $audio_parser_plugins !
$audio_decoder_plugin ! alsasink
```

MP3 playback example:

```
gst-launch filesrc location=test.mp3 [typefind=true] ! mpegaudioparse !
beepdec ! alsasink
```

3.1.2 Video only playback

```
gst-launch filesrc location=test.video typefind=true ! $demuxer_plugin ! queue
max-size-time=0 ! $video_decoder_plugin ! $video_sink_plugin
```

AVI file video only playback example:

```
gst-launch filesrc location=test.avi typefind=true ! aiurdemux ! queue max-
```

```
size-time=0 ! vpudec ! mfw_v4lsink
```

3.1.3 Audio/Video file playback

```
gst-launch filesrc location=test_file typefind=true ! $demuxer_plugin
name=demux demux. ! queue max-size-buffers=0 max-size-time=0 !
$video_decoder_plugin ! $video_sink_plugin demux. ! queue max-size-buffers=0
max-size-time=0 ! $audio_decoder_plugin ! alsasink
```

AVI file playback example:

```
gst-launch filesrc location=test.avi typefind=true ! aiurdemux name=demux
demux. ! queue max-size-buffers=0 max-size-time=0 ! vpudec ! mfw_v4lsink
demux. ! queue max-size-buffers=0 max-size-time=0 ! beepdec ! alsasink
```

3.1.4 Other methods for playback

You can use the playbin2 plugin or FSL gplay command line player for media file playback.

```
gst-launch playbin2 uri=file:///mnt/sdcard/test.avi
gplay /mnt/sdcard/test.avi
```

3.2 Audio Equalizer

```
gst-launch filesrc location=test.mp3 typefind=true ! beepdec ! mfw_audio_pp
enable=1 eqmode=2 ! alsasink
gst-launch playbin2 uri=file:///test.mp3 audio-sink="mfw_audio_pp enable=true
eqmode=2 ! alsasink"
```

NOTE

The `eqmode` value **2** indicates the “bass booster” scene.

3.3 HTTP Streaming

The HTTP streaming includes the following:

- Manually pipeline

```
gst-launch souphttpsrc location= http://SERVER/test.avi ! typefind !
aiurdemux name=demux demux. ! queue max-size-buffers=0 max-size-time=0 !
vpudec ! mfw_v4lsink demux. ! queue max-size-buffers=0 max-size-time=0 !
beepdec ! alsasink
```

- playbin2

```
gst-launch playbin2 uri=http://SERVER/test.avi
```

- gplay

```
gplay http://SERVER/test.avi
```

3.4 Video playback to multiple displays

Video playback to multiple displays can be supported by both mfw_v4lsink and mfw_isink.

This usage case requires system boots in multiple-display mode (dual/triple/four, the number of displays supported is determined by SOC and BSP). For how to configure system boot in such mode, see the *i.MX_6_BSP_Porting_Guide*.

To use mfw_isink for multiple displays, you need to configure vssconfig for multiple-display mode. Refer to Appendix 1: mfw_isink usage.

3.4.1 Different videos to different displays

- mfw_v4lsink

```
gst-launch playbin2 uri=file:///$file1 video-sink="mfw_v4lsink device=$VIDEO_DEVICE1
disp-width=$width disp-height=$height" &
gst-launch playbin2 uri=file:///$file2 video-sink="mfw_v4lsink device=$VIDEO_DEVICE2
disp-width=$width disp-height=$height"
```

Example on i.MX6DQ SD board, LVDS (primary) + HDMI:

```
gst-launch playbin2 uri=file:///$file1 video-sink="mfw_v4lsink device=/dev/video17" &
gst-launch playbin2 uri=file:///$file2 video-sink="mfw_v4lsink device=/dev/video19 disp-
width=1920 disp-height=1080"
```

- mfw_isink

Example on i.MX6DQ SD board, LVDS (primary) + HDMI:

```
export VSALPHA=1
gst-launch playbin2 uri=file:///$file1 video-sink="mfw_isink display=LVDS" playbin2
uri=file:///$file2 video-sink="mfw_isink display=HDMI"
```

Note: LVDS and HDMI in the command are the display names defined in the vssconfig file.

3.4.2 Same video to different displays

- mfw_v4lsink

```
                    gst-launch playbin2 uri=file:///$filename video-sink="tee
name=tee !
                    queue ! mfw_v4lsink device=$VIDEO_DEVICE1 disp-width=$width disp-
height=$height tee. !
                    queue ! mfw_v4lsink device=$VIDEO_DEVICE2 disp-width=$width disp-
height=$height"
```

- mfw_isink

```
                    export VSALPHA=1
                    gst-launch playbin2 uri=file:///$file video-sink="mfw_isink
display=LVDS display-1=HDMI"
```

3.5 Multiple video overlay

The mfw_isink plugin supports compositing multiple videos together and rendering them to the same display.

For example, if you want to play three videos to different windows of one display, 320x240 at (0,0), 640x480 at (400,0), and 320x240 at (400, 400), you can do it with the following commands:

```
            export VSALPHA=1
            gst-launch playbin2 uri=file:///$file1 video-sink="mfw_isink axis-left=0 axis-
top=0 disp-width=320 disp-height=240” &
            gst-launch playbin2 uri=file:///$file2 video-sink="mfw_isink axis-left=400 axis-
top=0 disp-width=640 disp-height=480” &
            gst-launch playbin2 uri=file:///$file3 video-sink="mfw_isink axis-left=400 axis-
top=500 disp-width=320 disp-height=240”
```

3.6 Encoding

Encoding includes audio encoding and video encoding.

3.6.1 Audio encoding

- MP3 encoding

```
                    gst-launch filesrc location=test.wav ! wavparse ! mfw_mp3encoder !
  filesink location=output.mp3
```

- WMA encoding

```
                    gst-launch filesrc location=test.wav ! wavparse ! mfw_wma8encoder !
  filesink location=output.wma
```

3.6.2 Video encoding

```
            gst-launch filesrc location=test.yuv blocksize = $BLOCK_SIZE ! 'video/x-raw-yuv,
format=(fourcc)I420, width=$WIDTH, height=$HEIGHT,
            framerate=(fraction)30/1' ! vpuenc codec=0 ! matroskamux ! filesink
location=output.mkv sync=false
```

NOTE

The blocksize property of the filesrc plug-in depends on the resolution of the input image. For I420 YUV files, Blocksize = inputwidth * inputheight * 1.5

The codec type property of the $video_encoder_plugin plug-in controls the target encode codec type. It could be 0 (MPEG4), 5(H263), 6(H264), or 12(MJPG).

3.7 Transcoding

The command line example is as following:

```
            $ gst-launch filesrc location=$filename typefind=true ! aiurdemux ! vpudec !
mfw_ipucsc !
            'video/x-raw-yuv, format=(fourcc)NV12, width=1280, height=720' ! vpuenc !
matroskamux ! filesink location=720p.mkv
```

3.8 Recording

Recording includes the following types:

- Audio recording
- Video recording
- Audie/Video recording
- TV-in Source

3.8.1 Audio recording

Audio recording includes the following types:

- MP3 recording

```
                    gst-launch alsasrc num-buffers=$NUMBER blocksize=$SIZE !
mfw_mp3encoder ! filesink location=output.mp3
```

- WMA recording

```
                    gst-launch alsasrc num-buffers=$NUMBER blocksize=$SIZE !
mfw_wma8encoder ! filesink location=output.wma
```

NOTE

The recorded duration calculated as $NUMBER*$SIZE*8/(samplerate*chanel*bitwidth).

For example, to record 10 seconds of stereo channel sample with 44.1K sample rate and a16bit width, use the following command:

```
                gst-launch alsasrc num-buffers=430
blocksize=4096 ! mfw_mp3encoder ! filesink location=output.mp3
```

3.8.2 Video recording

Different cameras need to be set with different capture modes for special resolutions (see the BSP document for camera). One example of recording is as follows:

```
            gst-launch mfw_v4lsrc fps-n=15 capture-mode=X !  queue ! $video_encoder_plugin
codec=0 ! matroskamux ! filesink location=output.mkv sync=false
```

NOTE

The fps-n property of the mfw_v4lsrc plug-in controls the camera capture frame rate.

The codec property of the $video_encoder_plugin plug-in controls the target encode codec type. Use the `gst-inspect` command to get more details about the codec property.

3.8.3 Audio/Video recording

```
            gst-launch -e mfw_v4lsrc capture-mode=X fps-n=30 ! $video_encoder_plugin
            codec=0 ! queue ! mux. alsasrc ! 'audio/x-raw-int,rate=44100,channels=2' !
            mfw_mp3encoder ! queue ! mux. $MUXER name=mux ! filesink location= output.
$EXTENSION sync=false
```

NOTE

- `-e` indicates to send EOS when the user presses **Ctrl+C** to avoid output corruption.
- `$MUXER` can be matroskamux, mp4mux, avimux, flvmux, qtmux, or mpegtsmux.
- If multiplexing the MPEG4 video to mpegtsmux, vpuenc needs to set property seqheader-method=2 and FSL MPG parser cannot support the MPEG4 format.
- `$EXTENSION` is the filename extension according to the multiplexer type.

3.8.4 TV-in source

The TV-In source plugin gets video frame from the TV decoder. It is based on the V4l2 capture interface. The command line example is as follows:

```
            gst-launch tvsrc ! mfw_v4lsink
            gst-launch tvsrc num-buffers=100 ! vpuenc ! matroskamux ! filesink location=./
output.mkv sync=false
```

NOTE

The TV decoder is ADV7180. It supports NTSC and PAL TV mode. The output video frame is interlaced, so the sink plugin needs to enable deinterlace. The default value of mfw_v4lsink deinterface is True.

3.9 RTSP streaming playback

For the RTSP streaming playback, set the video and audio decoder working in low-latency mode.

For the vpudec playback, if the low-latency property is set to True, it can work in low-latency mode.

For the beepdec playback, if an audio parser is connected before the beepdec input audio is framed, the beepdec playback can work in low-latency mode.

For the H.264 high bit rate playback, if the access-unit property is set to True, the depay plugin outputs H.264 in complete frames to avoid performance deterioration in VPU decoder plugin in low-latency mode.

- Manually pipeline

```
                    gst-launch rtspsrc location=$RTSP_URI name=source ! queue !
                    $video_rtp_depacketize_plugin ! vpudec low-latency=true !
mfw_v4lsink source. ! queue !
                    $audio_rtp_depacketize_plugin ! $audio_parse_plugin ! beepdec !
alsasink
```

For example (H.264 + AAC):

```
                    gst-launch rtspsrc location=rtsp://10.192.241.11:8554/test
name=source !
                    queue ! rtph264depay ! vpudec low-latency=true ! mfw_v4lsink
source. !
                    queue ! rtpmp4gdepay ! aacparse ! beepdec ! alsasink
```

The audio parse plugin is required before the beepdec plugin enables beepdec to work in low-latency mode.

You can run the following command to show the Gstreamer RTP depacketize plugins:

```
                    gst-inspect | grep depay
```

Two properties of RTSPSRC are useful for RTSP streaming:

- Latency: This is the extra added latency of the pipeline, with the default value of 200 ms. If you need low-latency RTSP streaming playback, you can set this property to a smaller value.
- Buffer-mode: This property is used to control the buffering algorithm in use, and it includes four modes:
  - None: Outgoing timestamps are calculated directly from the RTP timestamps, not good for real-time applications.
  - Slave: Calculates the skew between the sender and receiver and produces smoothed adjusted outgoing timestamps, good for low latency communications.
  - Buffer: Buffer packets between low and high watermarks, good for streaming communication.
  - Auto: Chooses the three modes above depending on the stream.

  The default setting is Auto.

- playbin2

  The vpudec low latency needs to be set to True if playing with playbin2.

```
gst-launch playbin2 uri=$RTSP_URI
```

NOTE

If you need to pause or resume the RTSP streaming playback, you need to use slave or none buffer-mode for RTSPSRC, as in buffer buffer-mode. After resuming, the timestamp is forced to start from 0, and this will cause buffers to be dropped after resuming.

3.10 RTP/UDP MPEGTS streaming

- UDP MPEGTS Streaming commands:

```
                    gst-launch udpsrc do-timestamp=false uri=$UDP_URI caps="video/
mpegts" !
```

```
                    aiurdemux streaming_latency=400 name=d d. ! queue ! vpudec low-
latency=true !
                    queue ! mfw_v4lsink sync=true d. ! queue ! beepdec ! alsasink
sync=true
```

For example:

```
                    gst-launch udpsrc do-timestamp=false uri=udp://10.192.241.255:10000
                    caps="video/mpegts" ! aiurdemux streaming_latency=400 name=d d. !
queue !
                    vpudec low-latency=true ! queue ! mfw_v4lsink sync=true d. !
queue ! beepdec ! alsasink sync=true
```

- RTP MPEGTS Streaming commands:

```
                    gst-launch udpsrc do-timestamp=false uri=$RTP_URI caps="application/
x-rtp" !
                    rtpmp2tdepay ! aiurdemux streaming_latency=400 name=d d. ! queue !
vpudec low-latency=true !
                    queue ! mfw_v4lsink sync=true d. ! queue ! beepdec ! alsasink
sync=true
```

For example:

```
                    gst-launch udpsrc do-timestamp=false uri=udp://10.192.241.255:10000
                    caps="application/x-rtp" ! rtpmp2tdepay ! aiurdemux
streaming_latency=400 name=d d. !
                    queue ! vpudec low-latency=true ! queue ! mfw_v4lsink sync=true
d. ! queue ! beepdec ! alsasink sync=true
```

NOTE

The source file that the UDP/RTP server sends must be in TS format.

It is recommended to start the server one second earlier than the time client starts.

One property of aiurdemux is useful for UDP/RTP TS streaming:

streaming-latency: This is the extra added latency of the pipeline, and the default value is 400 ms. This value is designed for the situation that the client starts first. If the value is too small, the whole pipeline may not run due to lack of audio or video buffers. In that case, you should cancel the current command and restart the pipeline. If the value is too large, you need to wait for a long time to see the video after starting the server.

3.11 RTSP Streaming Server

The RTSP streaming server usage case is based on the open source gst-rtsp-server package. It uses the Freescale aiurdemux plugin to demultiplex the file to audio or video elementary streams and to send them out through RTP. You can start the RTSP streaming server on one board, and play it on another board with the RTSP streaming playback commands.

The gst-rtsp-server package is not installed by default in the Yocto Project release. You can follow these steps to build and install it.

1. Enable the layer–meta-openembedded/meta-multimedia:

   Adding the line `BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-multimedia"` to the configuration file `<yocto_root>/build/conf/bblayers.conf`.

2. Include gst-rtsp-server into the image build:

   Adding the line `IMAGE_INSTALL_append += "gst-rtsp"` to the configuration file `<yocto_root>/build/conf/local.conf`.

3. Run the command `bitbake fsl-image-test/fsl-image-gui/fsl-image-x11/test-internal-x11` to build the image with gst-rstp-server.
4. You can find the test-uri binary in the folder:

   <yocto_root>/build/tmp/work/cortexa9hf-vfp-neon-poky-linux-gnueabi/gst-rtsp/0.10.8-r0/gst-rtsp-0.10.8/examples/.libs/

5. Flash the image.

   Copy test-uri into `/usr/bin` on board and assign the executing permission to it.

More information is as follows:

- Commands:

  ```
  test-url $RTSP_URI
  ```

  For example:

  ```
  test-uri file:///home/root/temp/TestSource/mp4/1.mp4
  ```

- Server address:

  ```
  rtsp://$SERVER_IP/8554/test
  ```

  For example:

  ```
  rtsp://10.192.241.106/8554/test
  ```

- Client operation supported:

  ```
  Play, Stop, Pause/Resume, Seek
  ```

Appendix A mfw_isink Usage

The mfw_isink plugin is based on IPU. It provides two main functions for video rendering:

- Video overlay: composites multiple video playbacks into the same display.
- Multiple displays: shows videos to multiple displays, up to four displays.

isink defines an environment variable VSALPHA to control the video visibility:

- VSALPHA = 1: The video is visible.
- VSALHPA=0 or undefined: The video is invisible.

isink uses a configuration file `vssconfig` to set parameters for each display device. The file is located in the `/usr/share` folder, and the configuration syntax is as follows:

[Display Name]

Specify the display name, used in the mfw_isink property display-x to enable the x display.

type

Currently specified to framebuffer.

format

The framebuffer color format.

fb_num

The frame buffer number for this display to show.

vsmax

The maximum videos can be showed for this display, with default value of 4.

main_fb_num

UI framebufer number. Usually, it is 0.

The following is an example of `vssconfig` for dual-display mode, LVDS(master) + HDMI.

```
# master display
[LVDS]
type = framebuffer
format = RGBP
fb_num = 1
main_fb_num = 0
vsmax=4

# slave display

[HDMI]
type = framebuffer
format = RGBP
fb_num = 2
vs_max = 4
```

We have two examples of `vssconfig` installed into the image, vssconfig.dual.lvds_hdmi, and vssconfig.triple.2lvds_hdmi for dual displays (LVDS + HDMI) and triple displays (LVDS + HDMI + LVDS). You can refer to them respectively.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: IMX6YMMUG
Rev. L3.10.17_1.0.0-ga
05/2014