

# Reference design

Figure 1 ..... 1

Figure 2 ..... 1

Figure 3 ..... 2

Figure 4 ..... 3

Figure 5 ..... 3

Figure 6 ..... 4

Figure 7 ..... 5

Figure 8 ..... 6

Figure 9 ..... 6

## Example

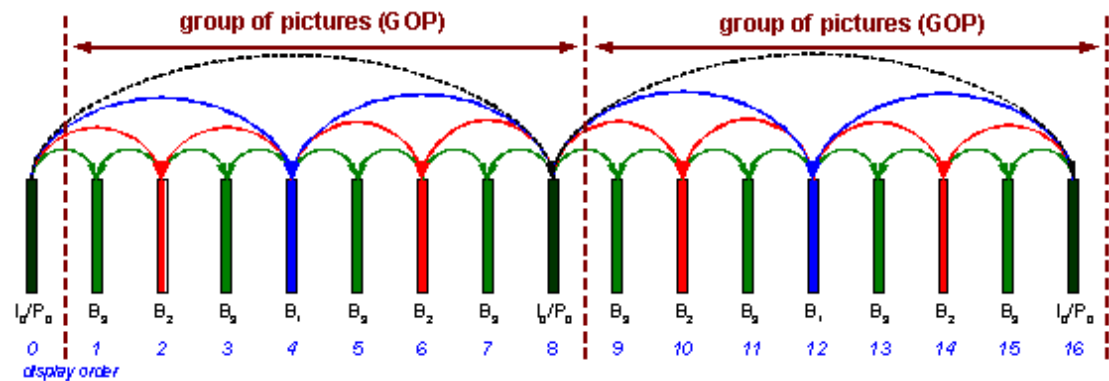


Figure 1

PTS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DTS	0	4	3	5	2	7	6	8	1	12	11	13	10	15	14	16	9

sequence	0	8	4	2	1	3	6	5	7	16	12	10	9	11	14	13	15
----------	---	---	---	---	---	---	---	---	---	----	----	----	---	----	----	----	----

Figure 2

Fig 1 is the example we explore for the proposed method.

Fig 2 lists the corresponding PTS & DTS of each frame.

The table in red lists the actual PTS in sequence when demuxing. It is, we should receive 0 firstly. Then comes by 8 (the start of next GOP), 4 (1<sup>st</sup> level B frame) 2 (2<sup>nd</sup>

level B frame) 1 (3<sup>rd</sup> level B frame) ...

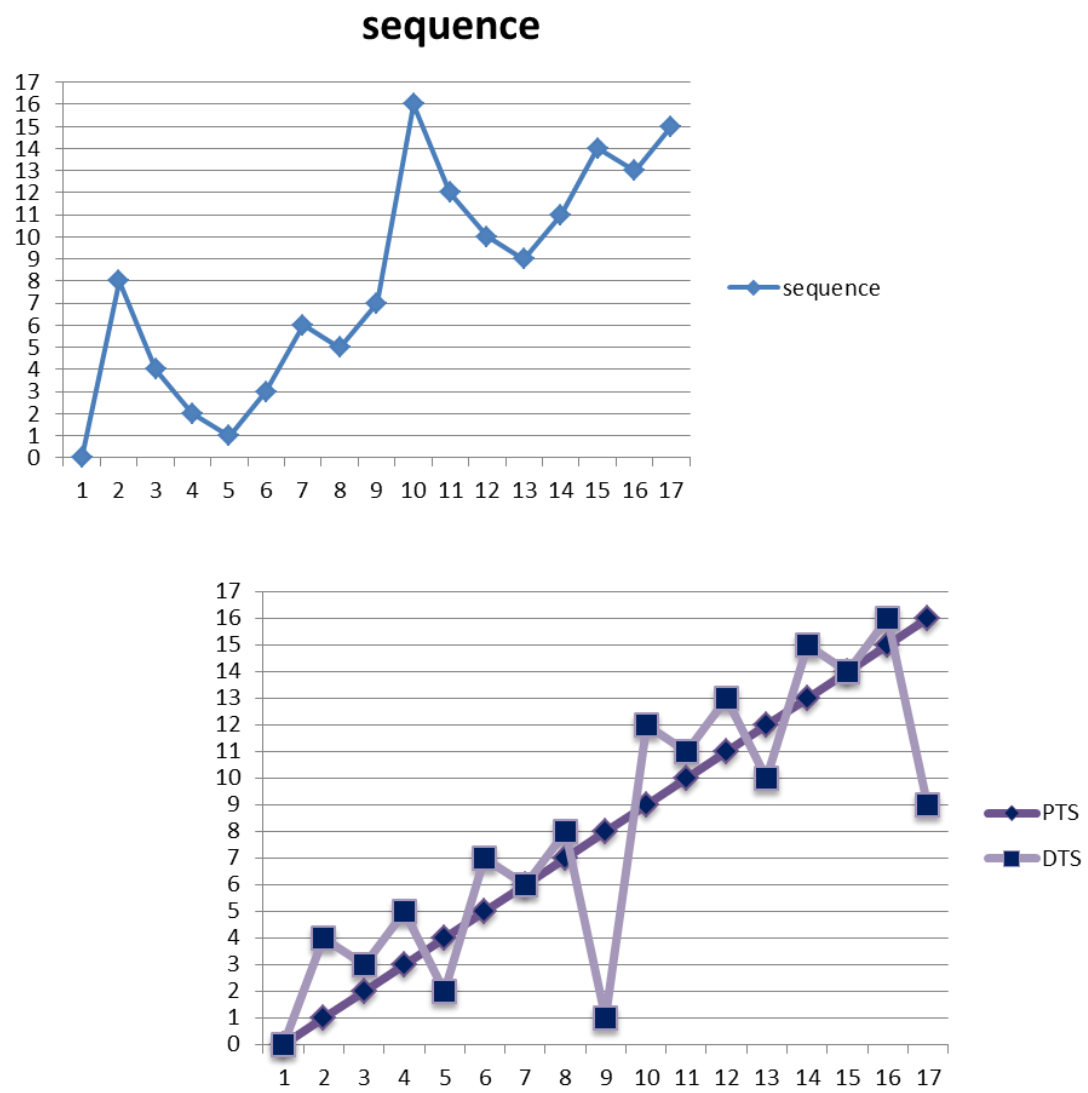


Figure 3

# Issued case

		1	2	3	4	5	6	7	8	9	10	missed
0.5	→	0	8									None
1.5	→	0	8									1
2.5	→	0	8									1, 2
3.5	→	0	8									1, 2, 3
4.5	→	0	8									1, 2, 3, 4
5.5	→	0	8									1, 2, 3, 4, 5
6.5	→	0	8									1, 2, 3, 4, 5, 6
7.5	→	0	8									1, 2, 3, 4, 5, 6, 7
8.5	→	0	8	4	2	1	3	6	5	7	16	None

Red: dropped at the end of flow

Figure 4

Fig 4 indicates the issue we have now. One could check for this example **we may MISS at most 7 frames if stop is set to 7.5**. Fig 5 lists the original flow.

```

/* do timestamps, we do this first so that we can know when we
 * stepped over the segment stop position. */
timestamp = gst_ffmpeg_time_ff_to_gst(pkt.pts, avstream->time_base);
if (GST_CLOCK_TIME_IS_VALID (timestamp)) {
    stream->last_ts = timestamp;
}
duration = gst_ffmpeg_time_ff_to_gst(pkt.duration, avstream->time_base);
if (G_UNLIKELY (!duration)) {
    GST_WARNING_OBJECT (demux, "invalid buffer duration, setting to NONE");
    duration = GST_CLOCK_TIME_NONE;
}
.....

/* check if we ran outside of the segment */
if (demux->segment_stop != -1 && timestamp > demux->segment_stop)
    goto drop;

```

Figure 5

# 1<sup>st</sup> proposal

We list the two judgments of 1<sup>st</sup> proposal here firstly.

Criterion 1: If we have met two consecutive frames F1 & F2 and both of their PTS are greater than segment's stop.

Criterion 2: F2's PTS is greater than F1's PTS.

**If bothe criterion 1 & 2 are satisfied, after F2 there is NOT a frame F whose PTS is less than F1.**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.5	→	0	8	4	2	1	3	6							
1.5	→	0	8	4	2	1	3	6							
2.5	→	0	8	4	2	1	3	6							
3.5	→	0	8	4	2	1	3	6	5	7					
4.5	→	0	8	4	2	1	3	6	5	7					
5.5	→	0	8	4	2	1	3	6	5	7	16				
6.5	→	0	8	4	2	1	3	6	5	7	16				
7.5	→	0	8	4	2	1	3	6	5	7	16	12	10	9	11
8.5	→	0	8	4	2	1	3	6	5	7	16	12	10	9	11
8	→	0	8	4	2	1	3	6	5	7	16	12	10	9	11

**Green:** dropped at renderer

**Red:** dropped at the end of flow

Figure 6

Take segment.stop = 4.5 for example; originally when meeting 2<sup>nd</sup> frame with PTS '8' we will go to drop. Therefore the frames with PTS = 1, 2, 3, 4 are all missed.

With the proposed method, only when reading the frames with PTS = 5 & 7 both the criterions are satisfied. Notice that for this case we passed down the frames with PTS = 0, 1, 2, 3, 4, 6, 8. **Among them 0, 1, 2, 3, 4, 8 are necessary for decoding but 6 is redundant. Since 6 & 8 will be dropped at renderer when checking segment's stop where, we don't need to care about even we pass them down.**

Please verify the correctness of 1<sup>st</sup> proposal by inspecting the given example.

The flowchart is given in fig 7 below.

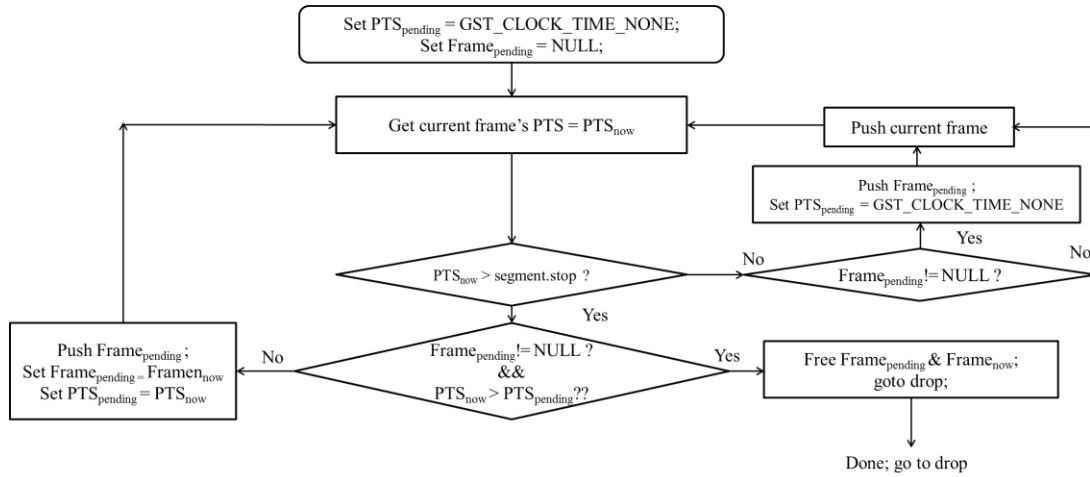


Figure 7

## 2<sup>nd</sup> proposal

We list the lemma on which 2<sup>nd</sup> proposal is based here firstly.

**Lemma:** If we have following sequence &  $PTS(F2) > PTS(F1)$ , then there is NO any frame F such that  $PTS(F) < PTS(F1)$  within region 3.

\*(region 1), F1, \*(region 2), F2, \*(region 3)

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	Save gain
0.5	→	0	8	4	2	<del>1</del>	3									-2
1.5	→	0	8	4	2	1	3									-1
2.5	→	0	8	4	2	1	3	6								-1
3.5	→	0	8	4	2	1	3	6								-3
4.5	→	0	8	4	2	1	3	6	5	7						-1
5.5	→	0	8	4	2	1	3	6	5	7						-2
6.5	→	0	8	4	2	1	3	6	5	7	16					-1
7.5	→	0	8	4	2	1	3	6	5	7	16					-5
8.5	→	0	8	4	2	1	3	6	5	7	16	12	10	9	11	-1
8	→	0	8	4	2	1	3	6	5	7	16					-5

**Green:** dropped at renderer  
**Red:** dropped at the end of flow

Figure 8

Fig 8 lists the execution result as well as how many steps it saves compared with the 1<sup>st</sup> proposal.

We demonstrate the usage of this lemma by example when stop = 3.5 as well as the flowchart below.

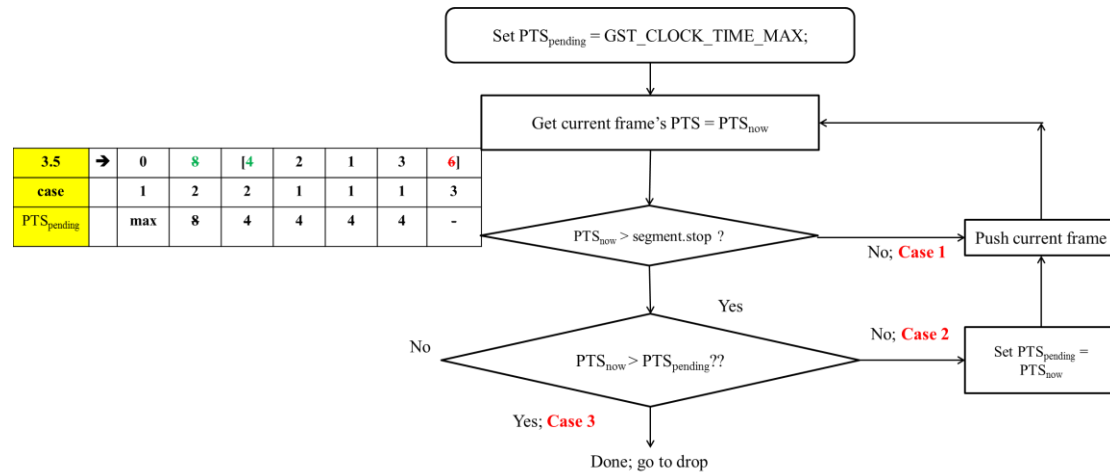


Figure 9

When demuxing 1<sup>st</sup> frame, PTS = 0; since PTS < stop, go to case 1.

When demuxing 2<sup>nd</sup> frame, PTS = 8; since PTS >= stop & PTS<sub>pending</sub> has not been assigned yet, goto case 2 (assign PTS<sub>pending</sub> by 8).

When demuxing 3<sup>rd</sup> frame, PTS = 4; since PTS >= stop & PTS<sub>pending</sub> >= PTS<sub>now</sub>, goto case 2 (replace PTS<sub>pending</sub> by 4)

When demuxing 4<sup>th</sup> frame, PTS = 2; since PTS < stop, goto case 1

When demuxing 5<sup>th</sup> frame, PTS = 1; since PTS < stop, goto case 1

When demuxing 6<sup>th</sup> frame, PTS = 3; since PTS < stop, goto case 1

When demuxing 7<sup>th</sup> frame, PTS = 6; since PTS >= stop, ince PTS >= stop & PTS<sub>pending</sub> < PTS<sub>now</sub>, goto case 3 (complete).