Accurate x/1001 Rate ST12 Time-of-day Calculation

Brooks Harris V5 2024-04-08

1. Introduction

The Society of Motion Picture and Television Engineers (SMPTE) has developed a synchronization protocol for video media called ST2059 based on IEEE 1588 Precision Time Protocol (PTP). PTP operates at seconds and nanosecond resolution. The calculation of these seconds and decimal fractions with respect to date and time-of-day is well known and straight forward.

Video media can be produced, distributed and broadcast at various frame rates. Some, such as 24/1 (film) and 25/1 (European video), are sometimes called “integer rates” because they have a one-to-one relation to true time-of-day. It is more or less straight forward to calculate frames and video labels for these rates.

But the rates used in the USA and other countries have peculiar rates such as 30000/1001 and 60000/1001, sometimes called “non-integer rates”. These x/1001 rates resolve to *repeating decimal fraction* frame rates, for example 30000/1001 = 29.9700… frames per second. This requires careful attention to achieve accurate implementation. This is well known in the industry for a range up to 24-hours.

SMPTE timecode is encoded in an hours:minutes:seconds:frames form (hmsf). At the “integer rates” this is straight forward to calculate because the labels correspond to true-time hours:minutes:seconds. But at the x/1001 rates the labeling does not correspond to true time so a special “drop-frame” labeling scheme is applied where two frame labels, xx:xx:xx:00 and xx:xx:xx:01, are omitted (“dropped”) each second except on each tenth minute. This results in a label that very nearly corresponds to the true hours:minutes:seconds.

The x/1001 rates present difficulties with respect to date and time-of-day because they do not fit exactly into the 24-hour day. For example, the 30000/1001 frame rate using drop-frame labeling the day rolls over -2.589410... frames earlier than 24 hours (86400 seconds). Traditionally a so-called “Daily Jam” procedure is used to 'paper over' these slight inaccuracies together with leap-second and Daylight Saving discontinuities by resetting (“jamming”) the timecode label to match the true time-of-day each day. This produces an acceptable, if slightly inaccurate, ST12 time-of-day timecode labeling scheme.

The use of ST2059 and IEEE 1588 demands accurate alignment of video signals with respect to local calendar date and time-of-day. This poses a challenge for accurate calculation of the video labeling at the x/1001 non-integer rates.

The author and his colleague Stephen Scott, of Skotel Corporation, worked for years to find a solution to this problem, arriving at key discoveries in 2016. This work has found its way into a new project at SMPTE, ST12-4 UTC Aligned Timecode.

The ST12-4 UTC Aligned Timecode project proposes a specification that can overcome the inaccuracies inherent in conventional ST12 timecode labeling schemes as applied to x/1001 frame rates with respect to date and time-of-day.

The ST12-4 document is very detailed and this may obscure the central idea. It may be helpful to explain the underlying concept in more detail. Here we describe the UTC compensated counting method proposed in ST12-4 that can compensate for these imperfections. This algorithm actually describes the *natural* occurrence of the cadence of x/1001 rate video frames with respect to UTC date and time-of-day. The method makes it possible calculate past and future time points, frame counts and ST12 timecode labeling that correspond with the natural behavior of the video with respect to calendar date and time-of-day.

This is applicable to any x/1001 rate. We concentrate the description on the example of the classic 30000/1001 frame rate.

2. Short and Long Days

30000/1001 drop frame will rollover at 23:59:59;29 or 2589408 frames. Frame boundaries for any x/1001 rate will coincide with date midnights every 1001 days. For the 30000/1001 rate 1001 86400 second days is exactly 2592000000 frames.

1001 days * 2589408 frames = 2591997408 frames, 2592 fewer than the required 2592000000 frames. Thus, we need 2592 additional frame numbers or labels to reach the total 2592000000 frames of 1001 days. To compensate we must extend each drop-frame day by some integral number of frames.

Define a "short day" (short) as 2589408 + 2 = 2589410 frames.
Define a "long day" (long) as 2589408 + 4 = 2589412 frames.

2589410 short * 1001 days = 2591999410 frames.
2592000000 - 2591999410 = 590 frames *less* than the required 2592000000 frames.

2589412 long * 1001 days = 2592001412 frames.
2592000000 - 2592001412 = 1412 frames *greater* than the required 2592000000 frames.

It is convenient for several reasons to treat the frame count in pairs. Dividing these less-than and greater-than values by two yields:

1412 frames greater-than / 2 = 706.
590 frames less-than / 2 = 295.

(Notice that 706 + 295 = 1001.)

706 short days * 2589410 short day frames = 1828123460.
295 short days * 2589412 long day frames = 763876540.
1828123460 short day frames + 763876540 long day frames = 2592000000.

Thus, there must be 706 short days and 295 long days in each 1001 day cycle for the 30000/1001 rate. These values are constants for the 30000/1001 rate in the following calculations.

This is the first key to the x/1001 vs. calendar date solution. The second step is how to evenly distribute the long and short days within the 1001 day cycle.

3. Long and Short Day Distribution by "DayMod1001 calculation"

How should the long days be arranged in the 1001 day sequence to most evenly distribute the short and long day compensation? The answer lies what we'll call the "DayMod1001 calculation".

First, it is convenient to number dates with a "day-number". The exact determination of a day-number with leap-second compensation is a somewhat complex, so to illustrate the process, simplify the problem by ignoring the leap-seconds and treat all days since 1970-01-01 00:00:00 (UTC) (Posix “the Epoch”, called UTC1970) as 86400-second days. With this we can get a zero-based day number count since UTC1970 from PTP time as simply:

```
DayNumber = (PTPSeconds - 10) / 86400;
```

With the DayNumber available the DayMod1001 formula provides the critical method that distributes the short and long days over the 1001 day sequence. Note the equation uses the 706 "greater-than" value from above:

```
DayMod1001 = (DayNumber x 706) MOD 1001;
```

From DayMod1001 simple logic using the 295 "less-than" value from above will determine if a day is short or long:

```
IF(DayMod1001 < 295)
   IsLongDay = true;
```

This will distribute the long days amongst the short days in an uninterrupted 1001 day sequence. The calculation is remarkably simple and computationally efficient. Here the steps are shown combined into one line of pseudocode:

```
IF((((PTPSeconds - 10) / 86400) x 706) MOD 1001 < 295)
    IsLongDay = true;
```

To illustrate the results of this simplified example (no leap-seconds) we can number the uninterrupted 1001 day sequence as zero-based 0 to 1000 and calculate DayMod1001 and IsLongDay for each day. The table shows the results of these calculations. You will see the long days occur interspersed amongst the short days every 3 or 4 days, evenly distributing the compensation drift across the 1001 day sequence.

Table - Abridged 1001 sequence with no leap-seconds, DayMod1001, and IsLongDay

1001 Sequence	Day Mod 1001	Is Long Day
0	0	1
1	706	
2	411	
3	116	1
4	822	
5	527	
6	232	1
7	938	
8	643	

9	348	
10	53	1
11	759	
12	464	
13	169	1
14	875	
....	Abridged	....
986	421	
987	126	1
988	832	
989	537	
990	242	1
991	948	
992	653	
993	358	
994	63	1
995	769	
996	474	
997	179	1
998	885	
999	590	
1000	295	

This illustration shows the central idea of the compensated counting scheme. The keys to understanding the scheme are the "greater-than" (706) and "less-than" (295) numbers shown in the first section and the DayMod1001 calculation shown here.

In reality 1001 sequences are often interrupted by a leap-second, so we must account for this. It turns out the DayMod1001 calculation will naturally account for the leap-seconds.

4. Leap-seconds

Leap-seconds complicate timekeeping in general and must be incorporated into the media calculations. We will gloss over the details of obtaining the leap-second information in interest of focusing on the central formulea for calculating accurate x/1001 frame numbering with respect to UTC date and time.

Using the example of PTP time, we need the number of leap-seconds (`LeapSeconds`) to determine if the PTPSecond is a leap-second (`IsLeapSecond`), adjust the day-number (`DayNumber`) if necessary, and determine if this day is a leap-second day (`IsLeapSecondDay`): Briefly:

```
// Find leap-seconds by Lookup in IERS Bulletin C tables
LeapSeconds = Lookup(PTPSeconds - 10) find (TAI-UTC - 10);

// Determine if current PTPSecond is a leap-second
IF(((PTPSeconds  63072010) - 1) == Lookup leap-second date)
   IsLeapSecond = true;
```

```
// Days since UTC1970 from PTP seconds with leap-second compensation;
DayNumber = ((PTPSeconds - 10) / 86400);
IF(IsLeapSecond)
  DayNumber = DayNumber - 1;

// Determine if current day is a leap-second day
IF(DayNumber == Lookup leap-second date)
   IsLeapSecondDay = true;
```

There are two other factors needed to align PTP time and UTC calendar date and time and treat leap-second appropriately.

We will gloss over the full explanation, but briefly the constant value 135 is an offset that aligns the DayMod1001 value of PTP time to the zeroth day number at UTC1970 for the 30000/1001 rate.

The constant value 15 is used to treat the leap-second as pairs of 30 frames for the 30000/1001 rate.

With these constants, leap-second information and the correct leap-second compensated calculation of DayNumber the DayMod1001 and short and long days can be calculated:

```
// Calculate DayMod1001 for 30000/1001 rate
DayMod1001 = (15 + (LeapSeconds x 15) + (DayNumber x 706) ) MOD 1001;

// Determine if current day is long (or short) for 30000/1001 rate
if( (NOT IsLeapSecondDay AND (1001DayPhase < 295)
     OR (IsLeapSecondDay AND (1001DayPhase < 280) )
   IsLongDay = true;
else
   IsLongDay = false;
```

We see the use of the constant values needed for the 30000/1001 rate:

135 = Offset to align PTP time to UTC1970 zeroth day.
15 = 1/2 30fps, two-frame pairs for the 30000/1001 rate
706 = Factor to determining 1001 day phase for 30000/1001 rate
295, and 280 = Factors to determining long or short days for 30000/1001 rate

This results in exact determination of 30000/1001 date boundaries with respect to date and time at UTC midnights. These points will always be at exact PTP seconds.

The algorithm supplies the midnight PTP time at each date and determines whether that day is a short or long day at the given frame rate. From this the exact PTP time of each frame and the time-of-day of each frame can be determined.

5. PTP Time-of-day Frame Alignment

For x/1001 frame rates there will be a small offset between the PTP local time midnight and the first frame on most dates.

The PTP time and number of frames since the SMPTE Epoch, here called "PTPFrames", can be determined with:

```
FrameRate = 30000/1001 (29.9700...)
FrameDuration = 1/(30000/1001) (0.03336...)
LocalPTPTime = PTPTime + LeapSeconds + TimeZoneOffset + DaylightSaving;
```

The PTP time nearest a frame boundary can be found:

```
PTPTimeAtFrame = floor(LocalPTPTime / FrameDuration) x FrameDuration;
```

The zero-based frame number since the SMPTE/PTP Epoch can be found by:

```
PTPFrames = PTPTimeAtFrame * FrameRate;
```

The PTP time at the UTC local date midnight can be found:

```
LocalDatePTPTime =
   (DayNumber * 86400) + 10 + LeapSeconds +
                                        TimeZoneOffset + DaylightSaving;
```

The frame number nearest the UTC local date midnight can be found by:

```
LocalDatePTPFrames =
   floor(LocalDatePTPTime / FrameDuration) x FrameDuration;
```

LocalDatePTPFrames is the zeroth frame of the day.

The local time-of-day in PTP frames, that is, the integer frame count since midnight, can be found:

```
TODPTPFrames = LocalDatePTPFrames + PTPFrames;
```

With this it is straightforward to produce accurate ST12 time-of-day labeling. Simply convert the TODPTPFrames values to ST12 YMDhmsf by the well know drop-frame algorithm.

6. Conclusion

The WD ST12-4 UTC Aligned Timecode proposal promises to provide frame-accurate ST12 labeling in applications from ST2059 to ST2010 and beyond, perfecting the accuracy of ST12 labeling with respect to date and time-of-day.