

Frame Rate Performance Modeling of Software MPEG Decoder

Victor Ramamoorthy

Adaptive Media Technologies
2, North First Street,
San Jose, CA 95113

ABSTRACT

A software MPEG decoder, though attractive in terms of performance and cost, opens up new technical challenges. The most critical question is: When does a software decoder drop a frame? How to predict its timing performance well ahead of its implementation? It is not easy to answer these questions without introducing a stochastic model of the decoding time. With a double buffering scheme, fluctuations in decoding time can be smoothed out to a large extent. However, dropping of frames can not be totally eliminated. New ideas of *Slip* and *Asymptotic Synchronous Locking* are shown to answer critical design questions of a software decoder. Beneath the troubled world of frame droppings lies the beauty and harmony of our stochastic formulation.

Keywords: MPEG decoder, software decoder, frame dropping, frame rate performance, double buffering, stochastic stability, asymptotic locking performance

1. INTRODUCTION

As the processor technology advances in leaps and bounds, many systems hitherto designed with dedicated hardware can now readily be implemented as software applications. One striking example of this trend is seen in multimedia computing environments where MPEG audio and video solutions are designed to take advantage of the raw CPU compute capacity either alone or with additional acceleration units. The benefits are low cost and high utilization. The performance of such hybrid or purely software solutions are hard to quantify or predict in advance because one has to wrestle with the complex issues of system and processor architecture, operating system, program compilation, process scheduling, data paths, I/O bottlenecks, memory architecture and the display system.

This paper is concerned with a difficult design issue of MPEG software decoder in a PC system. Though MPEG system model assumes an instantaneous decoder, a practical software or hardware system takes a finite amount of time to decode an input frame.

Since MPEG encoding is a variable rate scheme, the time to decode a given frame may vary widely depending on (1) information content in the frame, (2) instantaneous bit rate (3) type of frame (i.e., an I-frame, a P-frame or a B-frame), (4) the state of system resources. Though care is taken in the design process to complete decoding within the target frame period T , there is *no guarantee* that the software decoder, once included in a PC system, will always complete the decoding process within the period T . The decoding time in a software decoder fluctuates because of the influence of the PC system dynamics on the decoder.

There are quite a few *open* questions: Suppose we have a software decoder with algorithmic correctness but widely varying decoding time. How bad is this design compared to a dedicated hardware solution? What are the consequences of a decoder with fluctuating decoding time? When does it drop a frame? How does double buffering help? How do we measure the frame rate performance? How does a designer know which decoder is better? This paper, though far from complete, attempts to provide some clear answers and acceptable solutions. It is admittedly just a step in the right direction.

2. DECODER SYSTEM MODEL

Fig.1 shows the software architectural model of a MPEG decoder. The MPEG decoder consists of a variable length decoder for parsing and detecting the coded data, an inverse quantizer to reconstruct the amplitude values of DCT coefficients, an inverse DCT to find the pixel domain values and a DPCM decoder system to perform motion compensated reconstruction of the video frames. MPEG standards documents, [1], describe precisely how the decoder should be constructed. Fig.1 also shows the necessary data buffers to facilitate the required synchronous decoding behavior.

The input buffer in fig. 1 stores the input coded bit stream which complies with the MPEG bitstream syntax. The first task therefore is to separate all coded data relevant to the current picture. This is accomplished by exercising a variable length decoder module along with a couple of variable length decoder buffers. In order to achieve effective compression, different pictures are coded differently as I-frames, P-frames and B-frames. This means that the decoder must keep track of previously decoded pictures which are stored in reference buffers 1 and 2. Finally the decoded pictures are double buffered with buffers A and B in a “ping-pong” fashion and are sent to the video display system.

There are two different ways of operating the MPEG decoder: (1) In a “Pull” model, the decoder system “pulls” the coded data - typically from a CD-ROM - so that the decoding of the bitstream can be done without any interruptions. In this model, once the frame rate information is decoded, the video display system is set to run at this rate; all other decoder functions are driven by the display system clock. The control system that manages the entire decoder makes sure that the input buffer is suitably filled all the time and that there is enough data in the input buffer to perform decoding without any decoder stalls. The “Pull” model is less stringent with respect to decoding time. Current PC implementations belong to this category.

(2) In the “Push” model, the incoming bitstream carries the timing information in the form of time-stamps known as PCR. The decoder synchronizes its clock to that in the encoder system. As a result, the Read Clocks and Write Clocks (to be described later) coincide with each other. The “Push” model is restrictive and demands a well-designed decoder ; it is appropriate for real-time transmission where timing synchrony is critical.

Our approach is applicable to both these models. In some sense, it combines both these methods to create a general solution space.

3. ABSTRACT TIMING MODEL

For the sake of clarity, we define an abstract timing model of our prototype decoder as shown in fig. 2. What is shown is the timing diagram of two different clocks: one is the Read Clock which initiates the reading the input data contained in the input buffer. The other is the Write Clock that performs transferring of decoded picture into the display frame buffer. If the total decoding time is less than the frame period, as it is prescribed in the MPEG specifications, then these two clocks coincide with each other. In this case, the decoder will decode without any frame dropping. However, if there are fluctuations in the decoding time - sometimes the decoder decoding within the frame period and sometimes exceeding this limit - then, there is a good chance that *occasional* frame droppings may happen. Controlling the decoding time in a software environment is difficult. The “ping-pong” double buffers are provided to absorb the variations in the decoding time. The aim of this paper is to highlight the efficacy of the double buffering at the output and to provide necessary design guide lines for the developers.

Let us assume that initially the Read and Write clocks are identical. Referring to fig. 2, assume that Read Clock at the beginning of frame period 0 starts the decoding of the frame 0. Let us further assume that this frame is decoded within the target frame period and the decoded frame is sent to the buffer A before the arrival of the Read Clock for frame 1. Because of the double buffering arrangement, the decoded frame 0 will be written into the display buffer only when the Read Clock for frame 2 arrives. The decoded picture will be displayed for one frame period between Read Clock 2 and Read Clock 3. As shown in fig. 2, Write Clock 0 coincides with the Read Clock 2.

When Read Clock 1 arrives, the decoding of frame 1 can start - because the decoder is free to accept a new input as it finished ahead of its schedule for frame 0 - and let us assume that for frame 1, the decoder takes more time to finish decoding than the targeted limit. As a result, the decoded frame 1 is written into buffer B at a time instant between Read Clock 2 and 3. The contents of buffer B is written into the display buffer at Write Clock 1 - which is the same as the Read Clock 3.

Because the decoding of frame 1 took a longer duration than the target period, when Read Clock 2 arrives, the data in the input buffer for frame 2 can not be immediately used. The reason is the decoder is not yet free to start processing new input and it is still

busy with frame 1. As a result, the Read Clock 2 has to be delayed until the decoder is ready to accept a new input. The delaying of Read Clock 2 has further consequences: As shown in fig. 2, Read Clock 3 also gets delayed to a later time instant. The effect of delaying any of Read Clock can be cumulative and the system may never recover from this disturbance.

Whenever the delays accumulate, the only remedy is to drop a frame and hope that the situation may be better in the future! This phenomenon of “Lock and Slip” occurs whenever the decoding time fluctuates above the desired frame period. We show that this system has only a *stochastic stability* and that the *statistics* of decoding fluctuations control the frame dropping behavior.

4. STOCHASTIC MODELING OF DECODING TIME

Consider a random uncorrelated time series $\{\Gamma_n\}$. The realizations of this stochastic sequence be $\{g_n\}$. Let us assume that Γ_n has an arbitrary distribution over the interval $(0, \Delta + \delta)$ and the probability that $\Gamma_n = g$ is given by the density function $P_\Gamma(g)$. From this definition, it is clear that $P_\Gamma(g) \geq 0$ and mean value of the series, $m_\Gamma = \int_0^{\Delta+\delta} g P_\Gamma(g) dg$ and variance $\sigma_\Gamma^2 = \int_0^{\Delta+\delta} (g - m_\Gamma)^2 P_\Gamma(g) dg$. With the above construction, we model the decoding time $t_d(n)$ as a sum of fixed decoding time $(T - \Delta)$ and a variable time given by the random variable Γ_n . That is, the decoding time for the n th frame is $t_d(n) = (T - \Delta) + g_n$ with a support from $(T - \Delta)$ and $(T + \delta)$. We implicitly assume that $(T + \delta) \leq 2T$. This model is illustrated in fig. 3. By performing appropriate measurements of a decoder design, the parameters of this model can be estimated. It should be noted that the decoding time is a function of bit rate, frame rate, picture size, and other system parameters. There may be also a memory structure because the dependencies between I, P, and B frames. In many cases, it may be difficult to evaluate explicitly the actual hidden stochastic process that drives the system behavior.

In what follows, we develop a treatment for a general case with arbitrary density functions. We will show that it is possible to get a handle on the situation without the explicit knowledge of density functions. To validate our development, we employ computer simulations with a simple random model with a uniform density of

$$P_\Gamma(g) = \frac{1}{(\Delta + \delta)}$$

This simple model, as we show later, confirms our intuition on the actual behavior of the system and gives valuable insights. In the absence of any prior knowledge, a simple random model is the best choice. The conclusions derived for this model can not be very different from the one using exactly measured statistics. This

model is shown in fig. 4. This model has only two parameters Δ and δ , which are easier to measure experimentally.

5. LOCK AND SLIP

A simplified version of the timing model is shown in fig. 5. Let $R0$ be the time instant at which the decoder received its input frame numbered 0. $R0$ is actually the Read Clock 0 shown in fig. 2. Then, the decoder took a fixed time of $(T - \Delta)$ and a random processing duration of $\Gamma_0 = g0$ seconds to complete the decoding. The decoder then finished writing its output to one of two double buffers - buffer A or B - at time $W0 = R0 + (T - \Delta) + g0$. If the time instant $W0$ occurred prior to the arrival of the next input frame at time $R1$, then the decoder has finished its processing well ahead of its deadline.

For the next frame, the variable processing time $g1$ happened to be longer and the decoder could only finish decoding at time $(W1 = R1 + (T - \Delta) + g1) > R2$, which means that it missed its target deadline. The **Slip** in the synchrony of processing is measured by the quantity $\varphi_1 = W1 - R2$. Note that the slip for frame 1 is a positive quantity whereas it is a negative number for frame 0. Since there is a positive slip currently for frame 1, the best strategy is to proceed immediately with the next input which has been waiting for the same duration as the slip. This is made possible by delaying the Read Clock by the same amount as the current slip. This is also the policy of the “Pull” model decoder implementations.

Now we have set the stage for incisive observation: Let $\{R_n\}$ be the instants when the decoder should *ideally* start its decoding process by consuming the data in the input buffer. Because of the software implementation and the lack of precise real-time control, we will always have $R_n = nT + S_n$, where $\{S_n\}$ is an uncontrollable zero mean noise sequence that affects software operations. The actual realizations are denoted by $\{r_n\}$ and $\{s_n\}$ respectively for decoder start instances and the noise respectively. Let $\{W_n\}$ be the actual time instant at which the decoder empties its output to one of the two double buffers. Let us assume that the system is started at time 0. Then for the frame 0, the following apply:

$$r_0 = s_0;$$

$$w_0 = r_0 + (T - \Delta) + g_0;$$

$$\varphi_0 = w_0 - r_1 = w_0 - T - s_1 = (s_0 - s_1) + g_0 - \Delta;$$

Similarly for frame 1:

$$W_1 = \begin{cases} r_1 + (T - \Delta) + g_1 = T + s_1 + (T - \Delta) + g_1 & \text{if } \varphi_0 \leq 0 \\ W_0 + (T - \Delta) + g_1 = s_0 + 2(T - \Delta) + g_0 + g_1 & \text{if } \varphi_0 > 0 \end{cases}$$

$$\varphi_1 = (W_1 - r_2) = \begin{cases} T + s_1 + (T - \Delta) + g_1 - 2T - s_2 = s_1 - s_2 + g_1 - \Delta & \text{if } \varphi_0 \leq 0 \\ (s_0 - s_2) + g_0 + g_1 - 2\Delta = \varphi_0 + (s_1 - s_2) + (g_1 - \Delta) & \text{if } \varphi_0 > 0 \end{cases}$$

Continuing this way, it is possible to find a general expression for slip at frame n as:

$$\varphi_n = \begin{cases} (s_n - s_{n+1}) + (g_n - \Delta) & \text{if } \varphi_{n-1} \leq 0 \\ \varphi_{n-1} + (s_n - s_{n+1}) + (g_n - \Delta) & \text{if } \varphi_{n-1} > 0 \end{cases}$$

The above recursive relationship is an interesting one. For all positive values, slip behaves as a First Order Auto regressive process with unity correlation coefficient. If a sequence of $\varphi_0, \varphi_1, \varphi_2, \varphi_3, \dots, \varphi_n$ are all non-zero positive numbers, then this situation corresponds to the catastrophic accumulation of slip values such that the final n th frame is bound to be dropped by the decoder. This can happen only if the following is true:

$$\varphi_n = (s_0 - s_n) + \sum_{i=0}^n g_i - (n+1)\Delta = (s_0 - s_n) + (n+1) \left[\frac{1}{(n+1)} \sum_{i=0}^n g_i - \Delta \right] > T$$

Considering only the left hand terms in the above equation, we can arrive at the expected value of slip (assuming stochastic convergence) at any instant n as:

$$E(\varphi_n) = (n+1)[m_r - \Delta]$$

If $m_r < \Delta$, the expected value of slip becomes negative, which in turn means that the system is exhibiting *Asymptotic Synchronous Locking*. This means that even if the system has tendency to have slips, they do not jeopardize the stability of the system; the system has long term stability and slips do not result in frame drops. In other words, the statistics of the decoding fluctuations keep the system under control! Such *Stochastic Stability* is enforced by the manifestation of Tchebycheff Inequality [2-4].

Referring back to fig. 3, the above reasoning results in the condition that *the mean value of the decoding time must be always less than the frame period T* to achieve the *almost non-frame-dropping* operating point. Hence the essential constraint for *almost drop-free* software decoding is that the mean value of the decoding time should be less than the frame period - and not the conventional wisdom of fixing the maximum decoding time to

be always less than the frame period, which is of course difficult to attain in software decoders. This actually frees the design constraints somewhat and gives the implementers the freedom to innovate.

A slightly weaker stability exists for the condition, $m_r = \Delta$. The slip value has a tendency to increase and it can be reset only after a frame drop. Though frame drops are infrequent, the system is only quasi-stable.

The condition that $m_r \leq \Delta$ does not imply the system will be free from any frame drop. There will be occasional frame drops depending on the statistics of the random sequence $\{\Gamma_n\}$. The catastrophic condition for dropping frames occurs only if

$\frac{1}{(n+1)} \sum_{i=0}^n g_i > \left[\Delta + \frac{T - (S_0 - S_n)}{(n+1)} \right]$. The right hand side of this equation, neglecting

implementation noise, can be approximated as $[\Delta + \frac{T}{(n+1)}]$. The left hand side is the sample mean of uncorrelated random time series $\{\Gamma_n\}$ which, by the force of the central limit theorem, has the density function $f(x) \approx \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\eta)^2}{2\sigma^2})$ where $\eta = m_r$

and $\sigma^2 = \frac{\sigma_r^2}{(n+1)}$. Hence the probability of dropping a frame at n th frame is approximately given by:

$$\int_{[\Delta + \frac{T}{(n+1)}]}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x-\eta)^2}{2\sigma^2}\right\} dx = \frac{\int_{[\Delta + \frac{T}{(n+1)] - m_r}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{\beta^2}{2}\right\} d\beta}{\sqrt{\frac{\sigma_r^2}{(n+1)}}}$$

$$= Q\left[\frac{[\Delta + \frac{T}{(n+1)}] - m_r}{\sqrt{\frac{\sigma_r^2}{(n+1)}}}\right]$$

The above expressions are true for any arbitrary density functions. The appearance of the $Q(\cdot)$ function - which is well known in communication systems, see [3] - is more than a coincidence.

What is accomplished above is useful in the design process. By the invocation of central limit theorem, we have managed to transform all the unknown statistics of $\{\Gamma_n\}$ into a

known Gaussian random process. Hence there is actually *no need* to know the exact density functions of $\{\Gamma_n\}$. The probability of dropping nth frame with a given set of Δ, T, m_r, σ_r is given by the above $Q(\cdot)$ function. It is enough to know only mean and variance characterization; different density functions with the same mean and variance would perform identically. Notice that if $\Delta=0$, then the probability of frame drop quickly approaches 0.5. Hence it is critical that Δ is a non-zero number, preferably much greater than δ to avoid dropping of frames.

6. SIMULATION RESULTS

As explained before, we assumed a white, uniformly distributed statistics for the stochastic component of the decoding time. Fig. 4 shows this model. The beauty of this model is that it is totally specified by two numbers Δ and δ . A designer can estimate these two quantities easily. We simulated the entire system model and collected data to validate our analysis.

Fig. 6 shows the plot of slip for a sample of 2000 frames. Here we set $\Delta = 0.1T$ and $\delta = 0.05T$. For this setting, the decoding time varies between $0.9T$ and $1.05T$. This is very close to the conventional tight design of having the decoding time always less than T . The variable component of the decoding time is distributed uniformly between 0 and $0.15T$ with mean value $m_r = 0.075T$. Since $\Delta = 0.1T > m_r$, the expected slip behavior is that it is small with rare excursions to large values. This is exactly shown in fig. 6.

Next $\delta=\Delta=0.05T$ is set. For this case, the mean value m_r is equal to Δ . This means that the decoding time can be anywhere in between $0.95T$ and $1.05T$. As shown in fig. 7, the stability of the system is weakened. There are occasional large swings and the system drops frames occasionally.

When $\Delta=0$, the system drops the frames almost periodically. In this case, shown in fig. 8, the decoding time varies between T and $1.05T$. There are no stochastic restraint to limit the slip. A close-up view of all the above three conditions are again shown in fig. 9. This figure clearly substantiates our analysis.

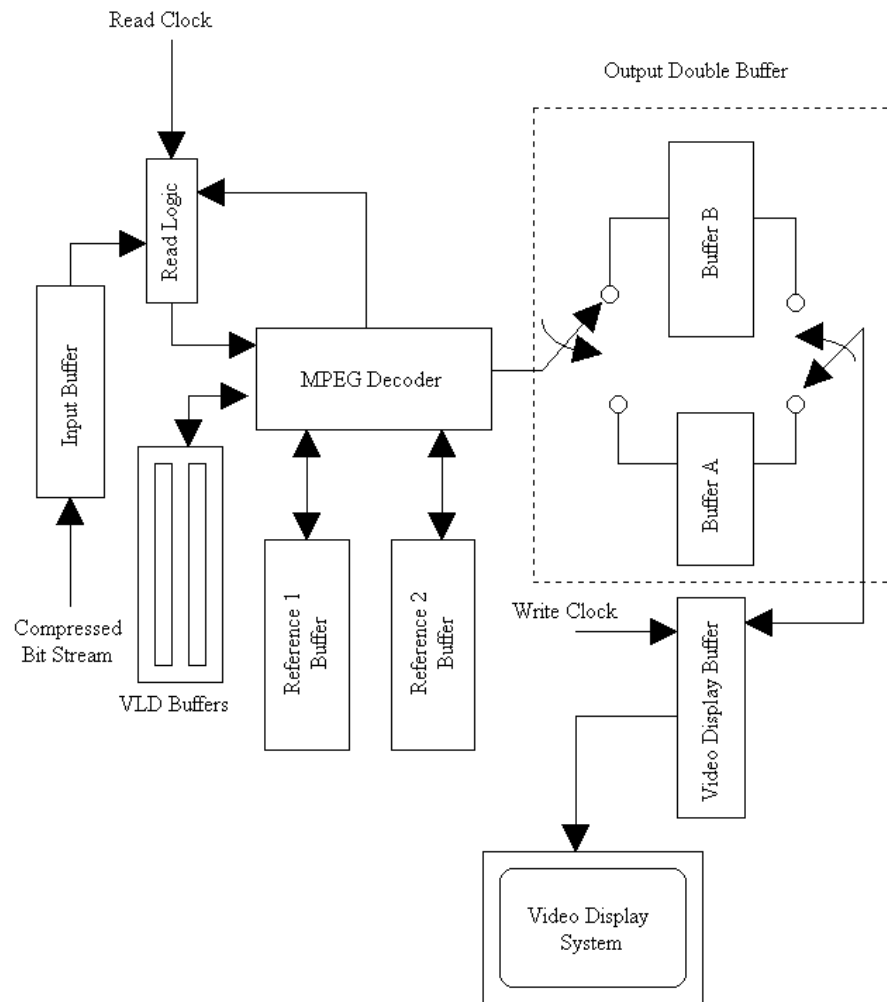
We then fixed a value for δ and then varied Δ and measured the probability of frame drop. This gives us the precise answer for questions like “what should be the design for guaranteed performance of probability of frame drop less than 10^{-4} ?”. From fig. 10, we can see that if $\delta=0.25T$, then Δ must be at least $0.45T$ to achieve a probability of frame drop less than or equal to 10^{-3} . In other words, a decoder having a decoding time varying between $0.55T$ and $1.25T$ will have a probability of frame drop less than or equal to 10^{-3} . Though the underlying assumptions in the analysis are idealistic, the results are general and are applicable to a wide variety of situations. The value of our work lies in creating this design aid for the developer.

7. ACKNOWLEDGEMENTS

Bulk of the work was done when the author was at the Philips Trimedia Product Group in Sunnyvale, CA. He gratefully acknowledges the discussions, friendship, help and support at Philips. The main design problem was posed while he was at S3 Inc. Thanks again to friends at S3.

8. REFERENCES

- [1] MPEG-1 and MPEG-2 Standards Documents
- [2] *Principles and Practice of Information Theory*, Richard E. Blahut, Addison-Wesley Publishing Company, 1987.
- [3] *Principles of Communication Engineering*, John M. Wozencraft and Irwin Mark Jacobs, John Wiley & Sons, 1965.
- [4] *Probability, Random Variables, and Stochastic Processes*, Athanasios Papoulis, McGraw-Hill Kogakusha Ltd, 1965



For MPEG-2 Decoding, the following buffer requirements apply:
 5 buffers to hold 720 X 486 pixels of YUV data = $5 \times 720 \times 486 \times 1.5 = 2.62$ Megabytes;
 Input buffer to hold 128 Kilo bytes of data and VLD buffers to hold 100 Kilobytes;

Fig.1 Our prototype system configuration. The video display system serves as the master clock to drive the entire decoder.

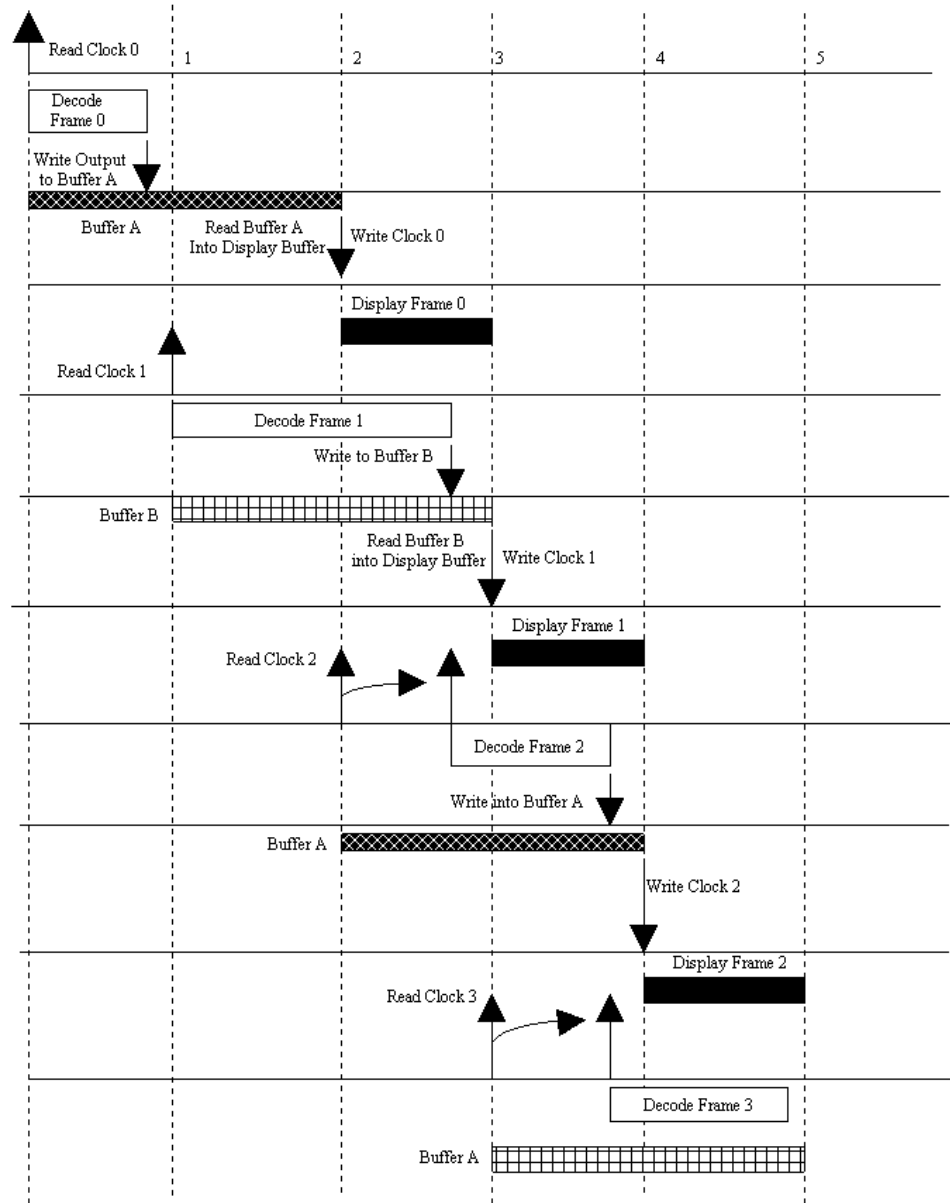
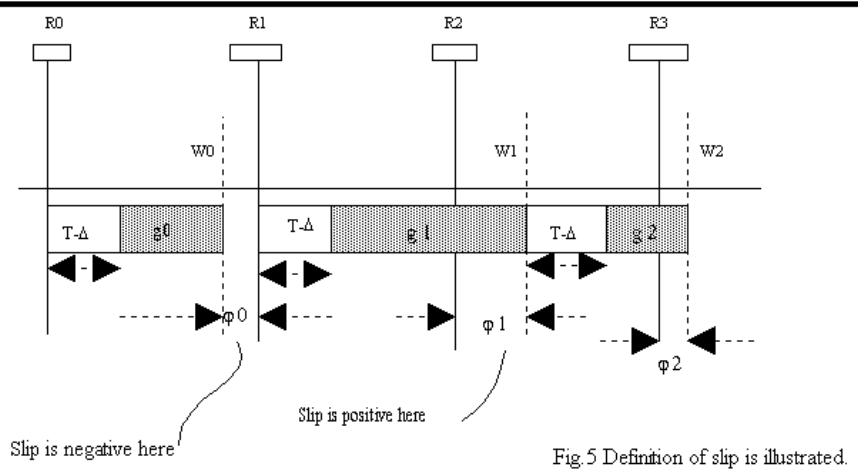
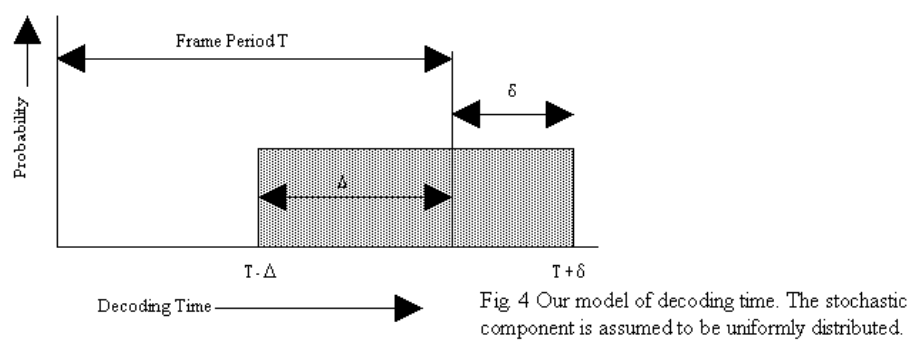
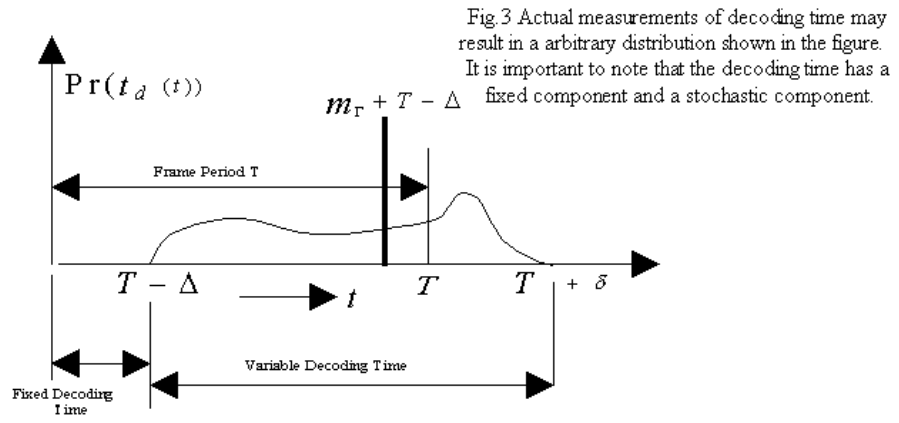


Fig.2 Detailed description of the timing mechanism in the decoder. The Read clock triggers the transfer of data from the input buffer to the decoder. The Write clock displays the decoded picture.



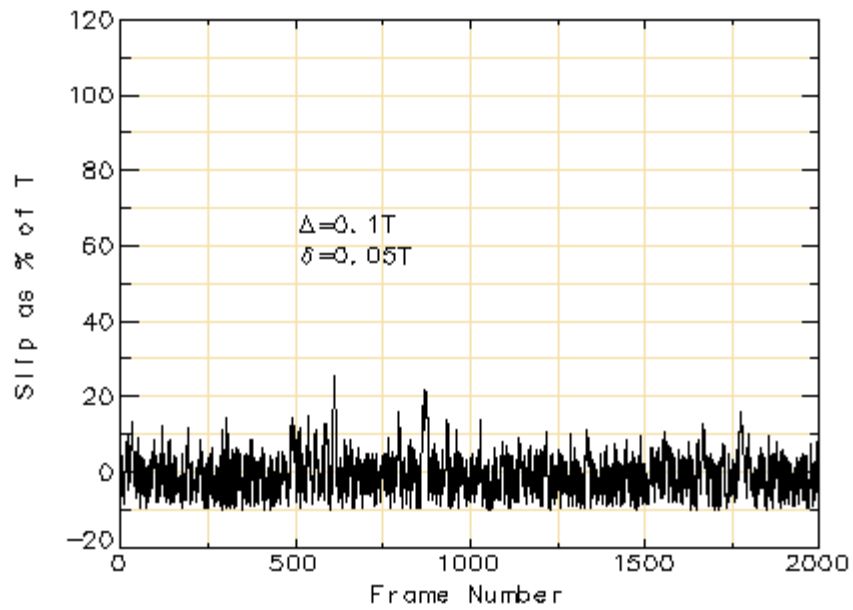


Fig. 6. The slip values are shown as a percentage of the frame period

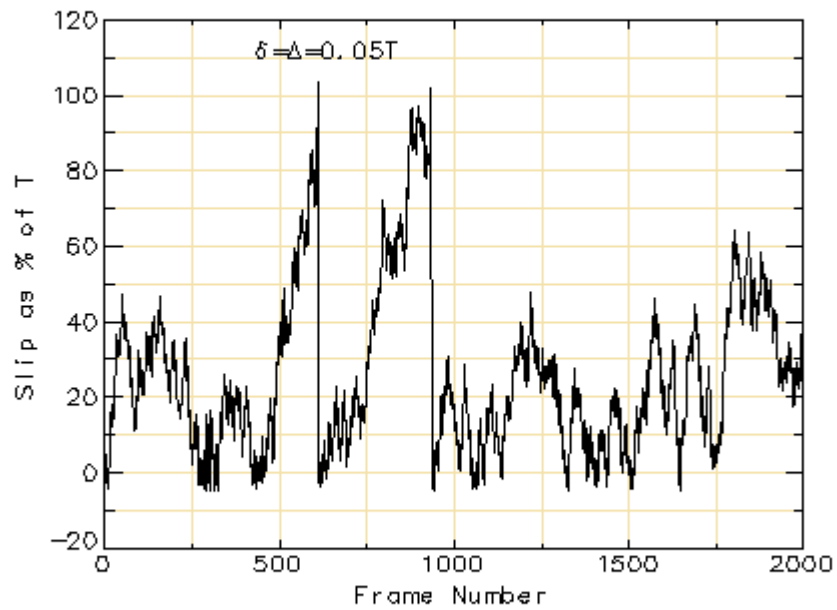


Fig. 7. When the slip values exceed the frame period, frame dropping occurs.

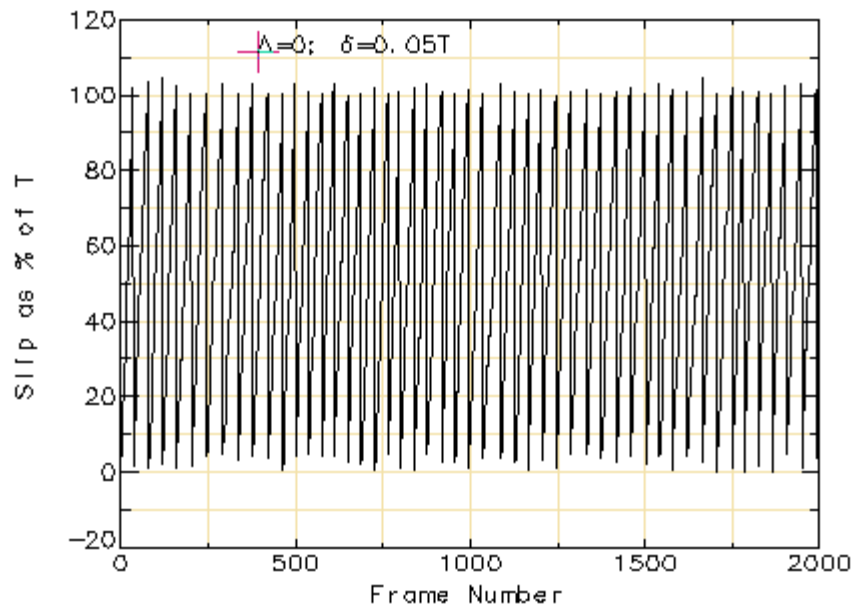


Fig. 8. Illustration of almost periodic frame dropping for $\Delta=0$ and $\delta=0.05T$.

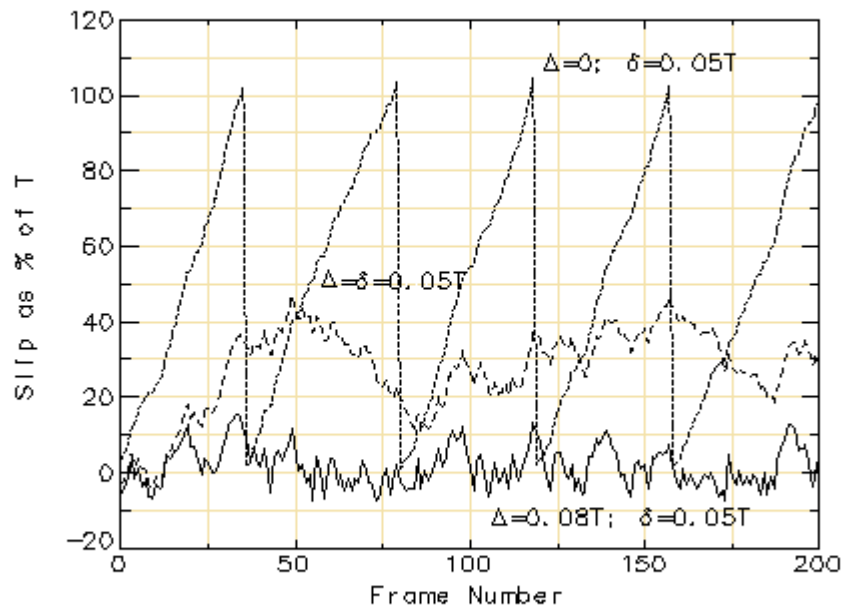


Fig. 9 A detailed view of the behavior of slip for three different parameter settings.

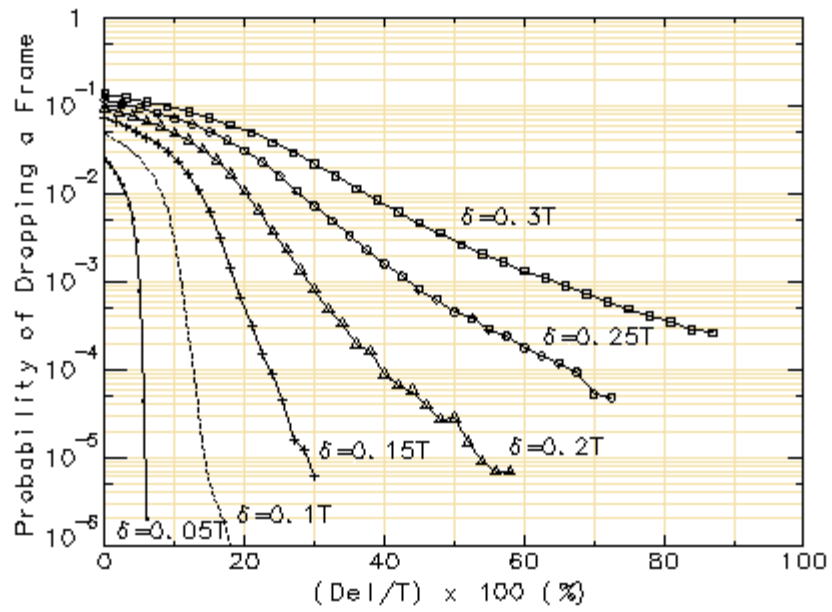


Fig. 10. Probability of dropping a frame as a function of Δ/T for different values of δ .