

Clock Synchronization in Software MPEG-2 Decoder

Victor Ramamoorthy

Adaptive Media Technologies
2, North First Street,
San Jose, CA 95113
victor@adaptmedia.com

ABSTRACT

A novel design approach to the problem of clock synchronization in software MPEG-2 decoders is presented. A software MPEG decoder is attractive in terms of cost and performance. However a software decoder is prone to timing uncertainty and delay jitters. By a clever use of adaptive filtering and sub sampling of time stamps, a frequency locked loop can be designed to deliver almost instantaneous capture of the unknown encoder clock frequency and with high tolerance to delay jitter. The exact analysis of the system is complex. By invoking suitable approximations, a complete design methodology is derived. Computer simulations verify the design approach illustrated.

Keywords: Clock Synchronization, MPEG decoder, Software decoder, Frequency-Locked-Loop, Real-Time system, Scheduling, Instantaneous Clock Acquisition

1. INTRODUCTION

Clock synchronization is an important part of a MPEG system, especially in designs where the compressed input to the decoder arrives from an uncontrollable real-time channel. Even in designs where the decoder obtains its input from a controllable digital medium such as a CD-ROM, it is desirable to extract the timing information though it is not always necessary.

Fig.1 shows a simplified diagram of a MPEG system. The origins of clock synchronization stem from the fact that a MPEG encoder is a variable bit rate device. To transport a variable bit rate stream through a constant rate channel or medium, buffers are required both at the encoder and the decoder. MPEG encoder may consist of several sets of separate video and audio encoders. Fig.1 shows only a generic arrangement of one video and two audio streams to illustrate the issues. Since buffers are placed at the encoder and the decoder sides, these buffers have to be managed so that they neither overflow and underflow. Buffer management in turn requires clock synchronization between the encoder system clock and the decoder system clock.

Notice that audio and video have separate frame size definitions and sampling rates and have entirely separate encoding strategies. To synchronize audio and video streams, it is convenient to have a common clock between the encoder system and the decoder system. This common clock between the encoder and the decoder systems can easily provide the time reference needed for inter stream synchronization. This common clock is achieved by the process of clock synchronization.

The third reason for having a synchronization system is that the encoder may transmit the bit stream through a packet network. A packet network may introduce variable delays for different packets. Some packets may arrive early and some may arrive late depending on the instantaneous network congestion. Some packets may also arrive out of order. By sending a sequence of timing references, the encoder system

provides the facility to resynchronize the component streams at the decoder. The timing references, commonly known as *time stamps* are embedded in the multiplexed bit stream by the encoder.

MPEG system clock is derived from a stable master system clock running at 27 MHz +/- 30 parts per million. In view of available off-the-shelf components, we assume a master clock running at 27 MHz +/- 50 parts per million. From this master clock, actual MPEG system clock of 90000 Hz is generated by the use of a divide-by-300 counter. The choice of 90000 Hz clock is based on the requirement that MPEG system clock must provide the ability to generate a set of component clocks necessary for driving the component encoders running at different frame rates. MPEG system specifies a 33 bit counter both at the encoder and the decoder for accomplishing phase unwrapping. The 33 bit counter upon receiving the nominal 90000 Hz clock will take 95443.72 seconds (26.51 hours) to wrap around to zero value. As we shall see, this long counter simplifies the task of clock synchronization.

The encoder system samples the 33 bit counter occasionally to create the required timing reference. The samples of the 33 bit counter values are denoted as *PCR* (Program Clock Reference). In MPEG-2, there is also a facility to sample the "divide by 300" counter to create the *PCR_ext*. This two part representation of *PCR_base* (33 bit sample) and *PCR_ext* (9 bit sample) is multiplexed along with the audio and video streams in the encoder system multiplexor. MPEG standards specify also two other types of time stamps: *DTS* (Decoder time stamp) and *PTS* (Presentation Time Stamp). *DTS* tells the decoder when to start decoding so that buffers are managed properly. *PTS*, on the other hand, instructs the decoder when to present a presentation unit (audio or video) at the output. We will be only dealing with *PCRs* in this paper which guide the decoder to lock onto the encoder clock frequency.

The multiplexed bit stream from the encoder is made available to the decoder system through a channel or storage medium. The decoder system is driven a separate decoder clock. Since the encoder clock can be anywhere between 89995 Hz and 90005 Hz, the decoder may not know the exact frequency of the encoder clock, though it knows the operating range of the encoder clock. The function of the clock recovery system at the decoder is to extract the transmitted *PCRs* in the stream and adjust the decoder clock to be the same as that at the encoder. For indefinitely long error-free operation, it is necessary that the decoder clock must be the same as the encoder clock. MPEG specifications indicate a frequency-locked-loop to synchronize the two clocks.

In a purely hardware based decoder system, building a synchronization subsystem is not difficult. A large body of literature is available on the subject of frequency and phase locked loops. See for instance, [5] Or [7]. A recent study, [6], addresses the MPEG clock recovery problem with network delay jitter in an ATM network. However, in a *software decoder*, it is not clear how clock synchronization can be done or what the resulting performance would be. Our analysis method is quite different. We do not place any restriction on the generation of *PCRs*. They can have a random distribution. We assume that MPEG system clock has very low frequency drift with time. MPEG specifies drift of 75 milliHertz/sec from the center frequency of 27 MHz. Unlike the conventional case of delay jitter being introduced by network transport, in a software decoder, delay jitter is an integral part of software processing. There exists opportunities to include novel twists in the design problem.

2.SOFTWARE DECODER ENVIRONMENT

Before we describe the actual synchronization issue, it is necessary to describe the model of the software environment which is depicted in fig.2. In order to handle real-time transmission, it is necessary to consider both the hardware and software components of the system. The necessary hardware component is shown in the dotted box in fig.2. The hardware merely consists of two clocks and two timers. One is the master system clock that runs the entire host CPU or accelerator CPU. The other clock is a variable frequency clock nominally running at 27 MHz. By writing to an input register of this controllable clock, its frequency can be altered. Timer 1 is driven by the master clock. Timer 1 generates an interrupt every T seconds. Timer 1 also initiates a new software iteration cycle. Significance of this timer will be made clear in the next section. Timer 2 on the other hand, is driven by the variable frequency clock. Timer 2 also generates an interrupt N times a second.

In fig.2., an example configuration is shown where Timer 2 counts up to 9000000. When it reaches the value 9000000, it generates an interrupt; thus it generates approximately 3 interrupts every second since it is driven by the nominally 27 MHz clock. Timer 2 is structured so that it is possible to read its value at any time by the software system. A different configuration of Timer 2 with a different count limit might equally work well. The main purpose of these two timers is to reduce the frequency of interrupt generation. Timer 2 helps in computing a *Local PCR* without excessive process overload. The hardware system shown in fig.2 in the dotted box can be constructed with existing host CPU /accelerator CPU systems. Clocks and timers can be salvaged from an existing system or simply added at a modest cost. The actual synchronization task is done by the software system sketched in fig.2. Whenever the interrupt from Timer 2 occurs, the 33 bit (“software”) decoder counter is incremented by 30000. Since Timer 2 interrupts occur approximately 3 times a second, a fixed increment of 30000 would make a total of 90000 counts a second which is expected from the nominal MPEG system clock.

The bit stream containing *PCRs* forms the input to the software decoder. It is fed to a software parser that extracts the embedded *PCRs* and outputs them to a buffer. The parser keeps writing the incoming *PCR* to the buffer. When the Timer 1 generates an interrupt, the latest *PCR* in the buffer will be directed to the synchronization system described below. A new software update cycle is also started. At the same instant, the contents of the Timer 2 will be read, scaled by an appropriate number (i.e., 1/300 in the example given in fig.2.) and added to the 33 bit decoder counter. The net effect of all these operations result in a *Local PCR (LPCR)* generated by the 33 bit decoder counter which is driven by the variable clock. The basic idea of using Timer 2 is to reduce the frequency of interrupts generated in the system so that the CPU performing the software decoding is not overloaded. Notice that the 33 bit counters used in the encoder and the decoder is large enough to avoid “wrap around to zero” in short intervals. Hence by comparing the incoming *PCR* with the locally generated *LPCR*, the frequency error between the encoder MPEG system clock and the decoder MPEG system clock can be found. This is shown in fig.2 as the difference signal $e(j) = PCR(j) - LPCR(j)$. In order to make the decoder clock follow the encoder clock, it is necessary to filter the instantaneous error between the two *PCRs*. We suggest a simple single pole infinite impulse response filter to operate on the instantaneous error. This filter can also be thought of a stochastic approximation mean value estimator with an exponential window. The properties of this filter can be controlled by varying the filter coefficient ρ . The output of this filter is multiplied with a gain constant G and added with the offset $F = 90000$ to create the correction value to be written into the control register of the variable frequency clock.

The entire system can be thought of as a conventional discrete-time frequency-locked-loop except for two key differences: (1) the update mechanism is controlled by software which may not have precise idea of “time” (2) the occurrence of *PCRs* in the bit stream is unpredictable; the inter-arrival duration between two successive *PCRs* at any time t is assumed to be a random variable $\eta(t)$ distributed uniformly between 0 and H seconds; MPEG-2 specifies H to be 0.7 second for *program streams* and 0.1 second for *transport streams*.

Let us amplify the first difference: In a software decoder, there are many tasks such as video decoding, audio decoding, system stream parsing, interrupt handling etc. These tasks in turn can be broken down to many levels of sub-tasks. There could be interrupts from user interaction from the host machine, interrupts from real-time constraints, interrupts from input/output devices and interrupts warning of a special condition such as overflow in a computation. This classification of interrupts is by no means complete. Since the same CPU is handling all these tasks, it has to continuously switch contexts and create the illusion of multiple simultaneous actions. Since the processor has only finite amount of resources, any attempt to optimize resource allocation invariably introduces delays. The net result of all these turns out to be a *timing or delay jitter* in completing a real-time task. This has immediate implications in our design of software clock synchronizer. Even if a *Real-Time Operating system* is used, this delay jitter problem does not go away: A Real-Time Operating system attempts to mitigate the timing precision problem by classifying tasks such as: (1) tasks with *hard* deadlines (2) tasks with *soft* deadline (3) *periodic* tasks and (4) *aperiodic* tasks. Then by using a scheduling algorithm such as the *Rate-Monotonic Scheduling*, the

tasks are scheduled to meet the timing requirements. It is to be noted that scheduling problem has many open issues and only rudimentary results are available even to answer the simple question of whether a set of tasks are schedulable. Very simple task models are used in practice: their applicability to MPEG decoder tasks where process loads vary with time is not known. See, for example, [2]-[4].

Hence in spite of the existence of a Real-Time Operating System to guard against the timing imprecision, it is safer to assume that there will be timing uncertainties with respect to extraction of PCR s and calculating their difference. We make the assumption that a delay jitter uniformly distributed in the range from 0 to Δ seconds is present in the software system. The value of Δ can be as high as a few milliseconds. Our initial experimentation shows that a value of 1 millisecond may be appropriate in a well balanced design. It could be even larger for designs that are heavily loaded.

3. IDEA OF SUB SAMPLING

Referring to fig.2, let us assume that the i th PCR , namely $PCR(i)$, parsed by the decoder parser be written into its buffer and taken in by the synchronization system at time $t(i)$. The decoder, because it is a software system, takes $\delta(i)$ time to compute the local PCR denoted by $LPCR(i)$. The delay $\delta(i)$ is the summation of all processing delays in the software implementation: it is difficult to pinpoint the causes of the software delay. It may be caused by scheduling conflicts, sudden overload to process excessively long stream of audio/video coded data, additional task burden placed on the CPU by another application, delays in the input/output data pipes and delays due to context switching. Because of the uncertainty in timing, the value of difference signal $e(i)$ can be decomposed as:

$$e(i) = \underset{\text{time} = t(i)}{PCR(i)} - \underset{\text{time} = t(i) + \delta(i)}{LPCR(i)} = \underset{\text{time} = t(i)}{PCR(i)} - \underset{\text{time} = t(i)}{LPCR(i)} - \delta(i) \cdot f_{decoder}[t(i)] \quad (1)$$

Though what is sought after is the true difference between two PCR s, the measured signal $e(i)$ is contaminated by a noise term $\varepsilon(i) = \delta(i) \cdot f_{decoder}[t(i)]$. Hence the signal-to-noise ratio in the measurement is given by:

$$SNR = \frac{E(i)}{\varepsilon(i)} = \frac{PCR(i) - LPCR(i)}{\delta(i) \cdot f_{decoder}[t(i)]} \quad (2)$$

where the true difference between the two PCR s is denoted by $E(i)$ - which is not available. Notice that because of the very nature of software system, the noise term in (2) can not be eliminated. Though $\delta(i)$ is in the range of 0.001, the product of $\delta(i)$ and $f_{decoder}[t(i)]$ can be in the same range as the true difference term $E(i)$. This means that the synchronization loop driven by $e(i)$ will be very noisy and will be practically useless. We assume that $\varepsilon(i)$ is a random variable uniformly distributed in the range 0 and E .

How can we improve the signal-to-noise ratio of the loop error signal? Suppose that we accept only every M th PCR instead of using every incoming PCR . This would make a quantum difference in the processor load. The process load is reduced because we are now processing only every M th PCR instead of every incoming PCR . This is beneficial in simplifying the software design. What is more significant is that it will also raise the signal-to-noise ratio:

$$SNR = \frac{\sum_{i=1}^M E(i)}{\varepsilon(M)} = \frac{\sum_{i=1}^M [PCR(i) - LPCR(i)]}{\delta(M) \cdot f_{decoder}[t(M)]} \quad (3)$$

Hence the use of sub sampling is doubly beneficial. The disadvantage of sub sampling is that the synchronization system is oblivious to the state of the encoder clock for intervals in between the sampling instants. MPEG standards specify a drift of 75 milliHertz/ second for 27 MHz clock. Since the normal drift in the encoder clock is very small, the idea of sub sampling seems to be ideal for software decoders.

There are two ways of sub sampling: (1) *sequence-based* sub sampling and (2) *time-based* sub sampling. Sequence-based sub sampling amounts to non-uniform sampling whereas time-based sampling is uniform. We start with sequence-based sub sampling and show that time-based sub sampling is more useful in our context.

4. SYSTEM DYNAMICS

In view of the advantages of sub sampling, a simplified clock recovery model is shown in fig.3. We will use the idea of sequence-based sub sampling in the following. In this model, we describe the operation of the entire system as a function of the index of incoming *PCRs*. Let $i = 0,1,2,3,\dots$ be the index of the incoming *PCRs* starting from the time the decoder is switched on. Let $j = 0,1,2,3,\dots$ be the index of the

PCRs after sub sampling by a factor of M and $j = \frac{i}{M}$. Let the sequence of *PCRs* after sub sampling be denoted by $\overline{PCR}(j)$. Let the encoder clock frequency be $f_{encoder}$ which is unknown to the decoder. The objective here is to find how the decoder clock frequency set to an arbitrary value of $f_{decoder}(0)$ at $j = 0$, approaches the unknown encoder clock frequency $f_{encoder}$ for increasing values of j and what parameters affect this dynamic behavior.

In fig. 3, the components of fig.2 are shown with greater elaboration. The aggregate software processing delay jitter for any j is $\epsilon(j)$. The hardware variable frequency clock is shown in heavy box. The sequence-based sub sampling is shown as a heavy circle with an arrow.

At $j=0$, the following is true: $LPCR(0) = PCR(0)$; that is, the first incoming *PCR* is loaded into the decoder 33 bit counter. Let $s(0)$ be the output of the filter at $j=0$. Let F is the center frequency of the decoder variable frequency clock. Then $f_{decoder}(0) = F + G \cdot s(0)$, where G is the gain of the filter.

Let the inter-arrival period between two successive *PCRs* be denoted by $\eta(i) = t(i+1) - t(i)$, where $t(i+1)$ is the arrival time of $PCR(i+1)$ and $t(i)$ is the arrival time of $PCR(i)$. For any value of i , $\eta(i)$ is assumed to be a random variable uniformly distributed in $(0,H)$. The exact value of $\eta(i)$ is decided by the encoder when it decides to include a sample of the 33 bit counter as a *PCR* in the coded bit stream. The decoder has no way of knowing the value of $\eta(i)$. MPEG standards require that $\eta(i)$ is upper bounded by $H=0.1$ second for transport stream and by $H=0.7$ second for program stream.

Then at $j = 1, i = M$ and the following apply from fig.3:

$$\overline{PCR}(0) = PCR(M) = PCR(0) + f_{encoder} \cdot \sum_{k=0}^{M-1} \eta(k) \quad (4)$$

$$LPCR(1) = PCR(0) + f_{decoder} \cdot \sum_{k=0}^{M-1} \eta(k) \quad (5)$$

$$E(1) = [f_{encoder} - f_{decoder}(0)] \cdot \sum_{k=0}^{M-1} \eta(k) \quad (6)$$

$$e(1) = E(1) - \varepsilon(1) = E(1) - \delta(M) \cdot f_{decoder}(0) \quad (7)$$

$$s(1) = (1 - \rho) \cdot e(1) + \rho \cdot s(0) \quad (8)$$

$$f_{decoder}(1) = F + G \cdot (1 - \rho) \cdot [f_{encoder} - f_{decoder}(0)] \cdot \left[\sum_{k=0}^{M-1} \eta(k) \right] - G \cdot (1 - \rho) \cdot \varepsilon(1) + G \cdot \rho \cdot s(0) \quad (9)$$

The equation (9) shows how the decoder clock frequency is updated at $j=1$. Continuing this way, it is easy to find a general expression for any j :

$$f_{decoder}(j) = f_{decoder}(j-1) + \{ [f_{encoder} - f_{decoder}(j-1)] \cdot Q(j) \} - N(j) \quad (10)$$

where

$$Q(j) = \left\{ G \cdot (1 - \rho) \cdot \left[\sum_{k=(j-1)M}^{Mj-1} \eta(k) \right] \right\} \quad (11)$$

and

$$N(j) = G \cdot (1 - \rho) \cdot [\varepsilon(j) + s(j-1)] \quad (12)$$

Equation (10) shows the update relationship between $f_{decoder}(j)$ and its previous value $f_{decoder}(j-1)$. The last two terms in (10) are of special interest. The middle term in (10) is the correction term guiding the decoder clock frequency in its approach towards the encoder clock frequency. The last term can be considered as a “noise” term controlled both by the filter and delay jitter. Notice that $Q(j)$ is always a positive number as $G > 0$, $0 < \rho < 1$, for the system to work. Then if indeed $f_{encoder}$ happens to be greater than $f_{decoder}(j-1)$, then the second term in (10) will be a positive correction term to improve upon the previous estimate of $f_{decoder}(j-1)$. On the other hand, if $f_{encoder}$ is less than $f_{decoder}(j-1)$, then the correction will be a negative number to pull the estimate $f_{decoder}(j-1)$ in the right direction. As seen in (11), $Q(j)$ is a function of stochastic variables $\eta(i)$ and hence is time varying. The constants G, ρ , have to be chosen to maintain the loop stability and a fast response.

5. INSTANTANEOUS ACQUISITION, CONVERGENCE AND LOOP BANDWIDTH

It is of interest to note that if $Q(j) = 1$ in equation (10), then this equation reduces to:

$$f_{decoder}(j) = f_{encoder} - N(j) = f_{encoder} - \left[\frac{\varepsilon(j) + s(j-1)}{\sum_{k=M(j-1)}^{Mj-1} \eta(k)} \right] \quad (13)$$

This means that the loop is able to perform *Instantaneous Acquisition* of the unknown clock frequency! After acquiring the unknown encoder clock frequency, the decoder clock frequency at any time j will be perturbed by the “noise” term $N(j)$. We call the condition $Q(j) = 1$ as “*Critically Damped*” condition. Notice that for critical damping, the following must be true:

$$Q(j) = 1 \Leftrightarrow G = \frac{1}{(1-\rho) \cdot \sum_{k=M(j-1)}^{Mj-1} \eta(k)} \quad (14)$$

Since the variables $\eta(i)$ are unknown and are stochastic, it may appear that the “critically damped” condition is difficult to attain in practice. But that is not true! As we shall see later, by simply employing time-based sub sampling, this condition can be achieved without any difficulty. It is also possible to devise tracking algorithms that enforce the condition in (14). This is exactly achieved by the Timer 1 fig. 2. Another simple solution is to set G to:

$$G = \frac{\alpha}{(1-\rho) \cdot M \cdot \eta^*} \quad (15)$$

where η^* is either the mean $\bar{\eta}$ or maximum value H of variables $\eta(i)$. α is a gain factor (close to unity) which can be used to trim the gain to the desired value. If the maximum is used for η^* in (15), then $Q(j) < 1$. Next, if we rewrite equation (10) in terms of the difference between the encoder clock frequency and the decoder clock frequency, then the following is true:

$$\Delta f(j) \equiv f_{encoder} - f_{decoder}(j) = \Delta f(j-1) \cdot [1 - Q(j)] + N(j) \quad (16)$$

Equation (16) is complex in nature. An explicit closed form solution is difficult to derive. We will however, try to find approximate solutions that are valid under special conditions. First, if the difference $\Delta f(j)$ tends towards a stationary value Δf^* for large values of j , then:

$$\Delta f(j) \rightarrow \Delta f^* \Rightarrow \Delta f^* \rightarrow \frac{N(j)}{Q(j)} = \frac{[s(j-1) + \varepsilon(j)]}{\sum_{k=(j-1)M}^{jM-1} \eta(k)} \approx \frac{\varepsilon(j) + \Delta f^* \cdot \bar{\eta} \cdot M - \bar{\varepsilon}(j)}{\sum_{k=(j-1)M}^{jM-1} \eta(k)} \quad (17)$$

Notice that the filter used is an exponential window mean value estimator. Hence the numerator in (17) converges to the sum of prediction error of the delay jitter and $\Delta f^* \cdot \bar{\eta}$. For sufficiently large value of M , under fairly general conditions, Δf^* can be shown to converge to zero. Once again, from (17), the condition that $Q(j) = 1$ is implied for convergence.

Let us assume for the moment that the term $N(j)$ is negligible. Then, the necessary condition for the convergence is that $|1 - Q(j)| < 1$. This means that $0 < Q(j) < 2$. If $0 < Q(j) < 1$, then the convergence will be *monotonic without any oscillations*. This condition is also equivalent to an *over-damped* system. *Oscillatory* behavior is exhibited for the condition $1 < Q(j) < 2$. If $Q(j) > 2$, then no convergence is possible. The behavior of the synchronization system can also be explained in terms of “*Loop Bandwidth*”. Under the assumption that $\sum_{k=(j-1)M}^{jM-1} \eta(k) \approx M \cdot \bar{\eta}$, we can show that the loop bandwidth B , for monotonic convergence is given by:

$$B \approx -\frac{\log[1 - G \cdot (1 - \rho) \cdot M \cdot \bar{\eta}]}{M \cdot \bar{\eta}} \quad \text{Hz} \quad (18)$$

A similar expression is available for oscillatory convergence. Returning to fig.3., we can consider the MPEG encoder as a “modulator” for sending the encoder clock frequency information in terms of *PCRs*. The decoder clock recovery system then can be thought of as a “demodulator” for recovering the clock frequency hidden in the *PCRs*. If we modulate the encoder clock frequency to be a square wave between 89995 Hz and 90005 Hz, then a good recovery system should produce a “good” version of this square wave at the decoder. In fact, what is attempted is a measurement of total system quality for square wave inputs. The quality of the entire clock recovery system is measured by:

$$SNR(dB) = 10 \cdot \log \left[\frac{\sum_{j=1}^N (f_{encoder}(j) - F)^2}{\sum_{j=1}^N (f_{encoder}(j) - f_{decoder}(j))^2} \right] = 10 \cdot \log \left[\frac{25 \cdot N}{\sum_{j=1}^N (f_{encoder}(j) - f_{decoder}(j))^2} \right] \quad (19)$$

In order to make the comparison easier, we also assume that the encoder square wave modulation is performed to synchronize with the decoder iteration cycle.

The basic ideas described above are illustrated in the computer simulation study shown in figs. 4 - 8 . Our focus is clearly on Program Streams. Fig. 4a shows the convergence aspects of the loop when there is no delay jitter in software processing. The sub sampling factor M is set to 10 and $H = 0.7$. Different parameter set ups result in different loop bandwidths. The existence of instantaneous acquisition is clearly shown in this figure for $\rho = 0.1$ and $\alpha = 0.9$. Monotonic and oscillatory convergences are exhibited.

Fig. 4b shows the situation where the delay jitter is present. Here a sub sampling factor $M = 10$ is chosen for the maximum delay jitter value of $E = 10$. For the instantaneous acquisition, though the response is rapid, the presence of noise is evident. By making the loop acquire the unknown frequency fast, we have failed to constrain the bandwidth to avoid noise. While the over damped system suppresses noise, its response is slow. Different convergence behavior is clear in this figure.

Fig.5 shows the condition when the sub sampling factor is chosen to be $M = 10$ and the delay jitter is allowed to take two different values of $E = 10$ and $E = 100$. When the sub sampling factor is mismatched to the delay jitter, then as shown in fig. 5., the loop behavior becomes unstable. Figs 6 and 7 show the loop behavior when $M = 100$. A larger value of M suppresses the delay jitter and create an acceptable locking behavior. The value of M plays a critical role in controlling the noise. Note that the condition of $Q(j) = 1$ indeed produces instantaneous frequency locking in these figures.

Fig.8 shows the performance of the system as measured by (19) for a range of values of M and E . For the case when there is no delay jitter, little can be gained by increasing the sub sampling factor. As we increase the delay jitter, the sub sampling factor helps in improving the signal-to-noise ratio. However, there seems to be ceiling at SNR of 21 dB. The system’s performance can not be improved beyond this limit even with large increases in the sub sampling factor. This has to do with the single pole filter used in the system.

6. ADAPTIVE SYNCHRONIZATION

One of the disadvantages of instantaneous acquisition setting is that the filter coefficient ρ needs to be set to small values like 0.1 to avoid undue overshoots and undershoots of the decoder clock frequency. This is of course coupled to the gain value set to α . If ρ is increased then α must be decreased to maintain the stability of the system. Any decreases in α then cuts down the loop bandwidth and the system becomes

sluggish. As a result, the filtering benefits of higher values of ρ is lost. Clearly what is desirable is both fast response *and* elimination of noise. This means that the filter has to be changed dynamically to meet our requirements.

Fig. 9. Shows an adaptive strategy of two mode operation: a “hunt” mode and a “filter” mode. During the “hunt” mode, instantaneous acquisition is needed, and hence ρ is set to low values so that quick frequency lock is obtained. When the value of ρ is small, the system acts as a broadband system and lets the noise in. Then, after the unknown frequency is locked onto, the value of ρ is slowly increased. When ρ is increased, the length of the exponential window associated with the filter is also increased - making this as a low pass filter. Since the lock is achieved, the low pass behavior is only good as this filters out the noise. This is the “filter” mode. This kind of adaptation of ρ will filter out the excess noise allowed by the instantaneous acquisition. Care must be exercised so that ρ is not unduly increased. The algorithm for the adaptation of ρ is clearly illustrated in fig. 9.

Next is the question of when to be in the “hunt” mode and when in the “filter” mode. The decision to go into a “hunt” mode is made by computing the successive absolute differences of the input and comparing it to a threshold. When the threshold is exceeded, then it is an indication that some change has occurred in the encoder frequency. At this point, the system is kept in the “hunt” mode for L iterations to make sure that the changes have indeed occurred. After L iterations, the system returns to the “filter” mode by slowly increasing the value of ρ . As mentioned earlier, a proper window length is desirable in the “filter” mode such that slow drifts in the encoder frequency can be tracked.

As shown in fig. 10, this adaptive strategy clearly improves the performance of the entire system. The desired instantaneous acquisition and noise removal are simultaneously attained. The synchronization performance of this adaptive scheme is shown in fig. 11 as a function of the sub sampling factor. It can be seen that overall performance is clearly improved by adaptation.

7. ACKNOWLEDGEMENTS

This work was carried out while the author was employed at Philips Trimedia Product Group in Sunnyvale, CA. Thanks to Trimedia friends for encouragement and support.

8. REFERENCES

- [1] MPEG-2 Standards, ISO/IEC 13818-1, Recommendation H.222.0, Annex D
- [2] Daniel I. Katcher, “Engineering and Analysis of Real-Time Operating System”, Ph.D Thesis, Carnegie Mellon University, August 1994.
- [3] Lui Sha, Rangunathan Rajkumar and Shirish S. Sathaye, “ Generalized Rate-Monotonic Scheduling Theory: A Frame work for developing Real-Time Systems”, Invited paper, Proc of the IEEE, vol.82., No.1, January 1994, pp 68-82.
- [4] John A. Stankovic, Marco Spuri, Marco Di Natale and Giorgio C. Buttazzo, “Implications of Classical Scheduling Results for Real-Time Systems”, IEEE Computer Magazine, June 1995, vol.28., No.6., pp 16-25.
- [5] Edward A. Lee and David G. Messerschmitt, “ *Digital Communication*”, Kluwar Academic Publishers, 1988, Chapter 13.
- [6] Giovanni Fausto Andreotti, Giampaola Michieletto, Luigi Mori, and Alberta Profumo, “Clock Recovery and Reconstruction of PAL Pictures for MPEG coded Streams Transported Over ATM

Networks”, IEEE Trans on Circuits and Systems for Video Technology, Vol. 5., No. 6., December 1995, pp 508-514.

[7] Harry L. Van Trees, “*Detection, Estimation, and Modulation Theory - Part II : Nonlinear Modulation Theory*”, John Wiley and Sons, Inc.,1971, Chapter 3.

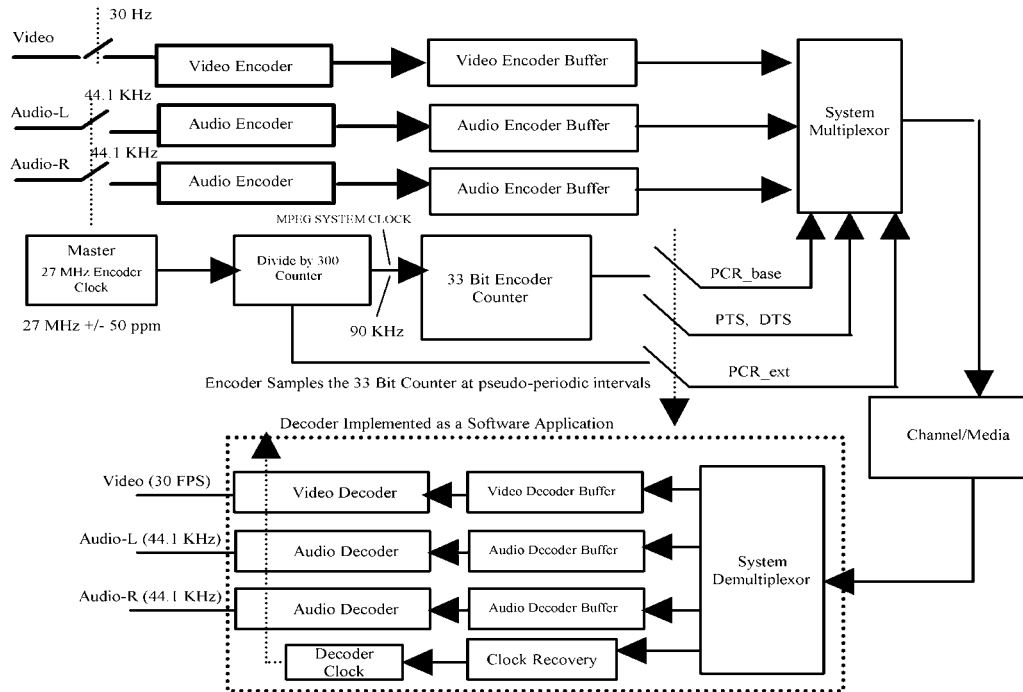


Fig.1 MPEG System Model. Clock Synchronization plays a key role in making the system work for real-time transmissions.

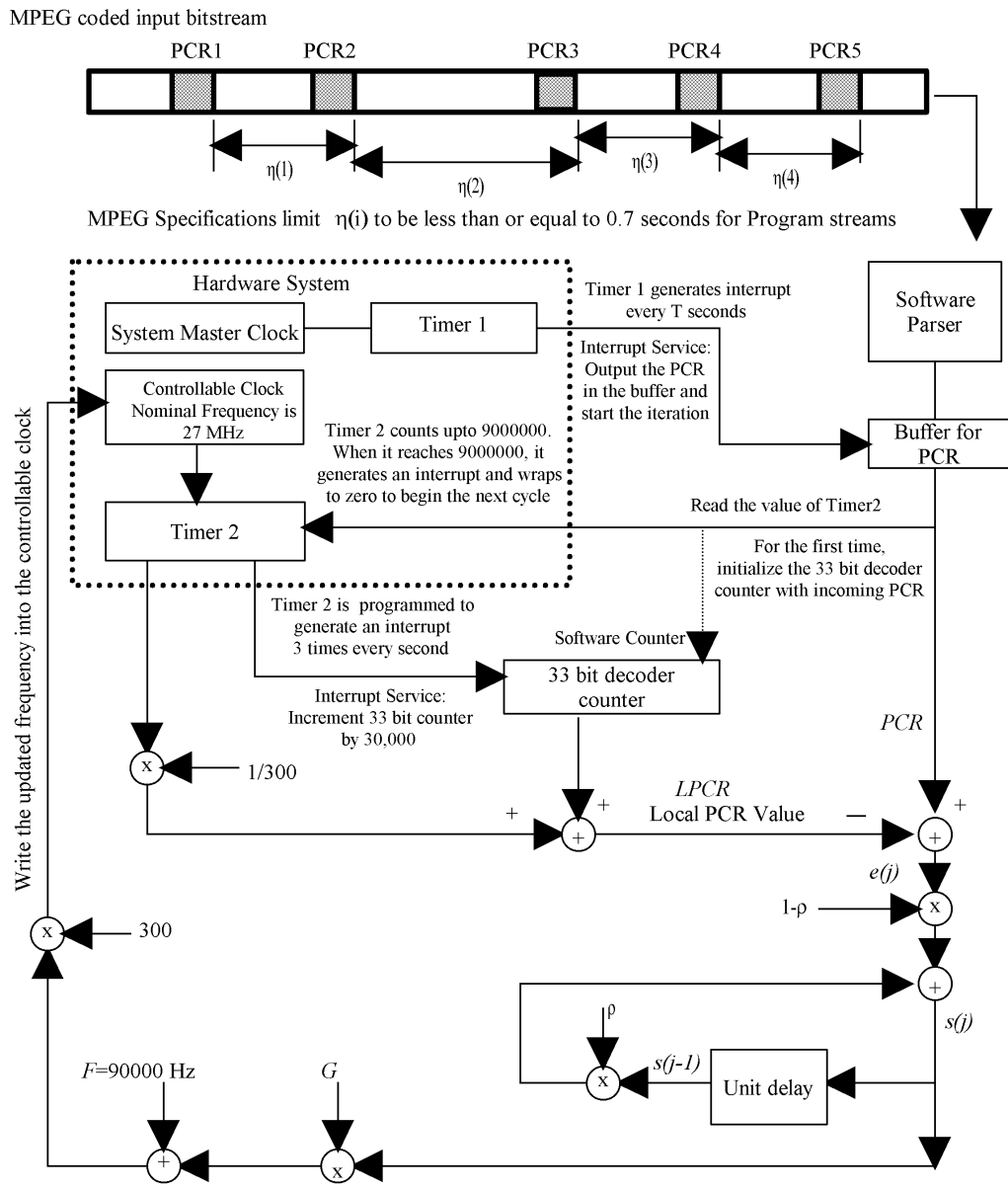


Fig.2 Software Clock Synchronization System. The dotted box shows the hardware required for synchronization. Software synchronizer accepts interrupts from the hardware and writes the updated frequency value into the control register of 27 MHz controllable clock at each iteration. Timer 1 triggers each iteration cycle.

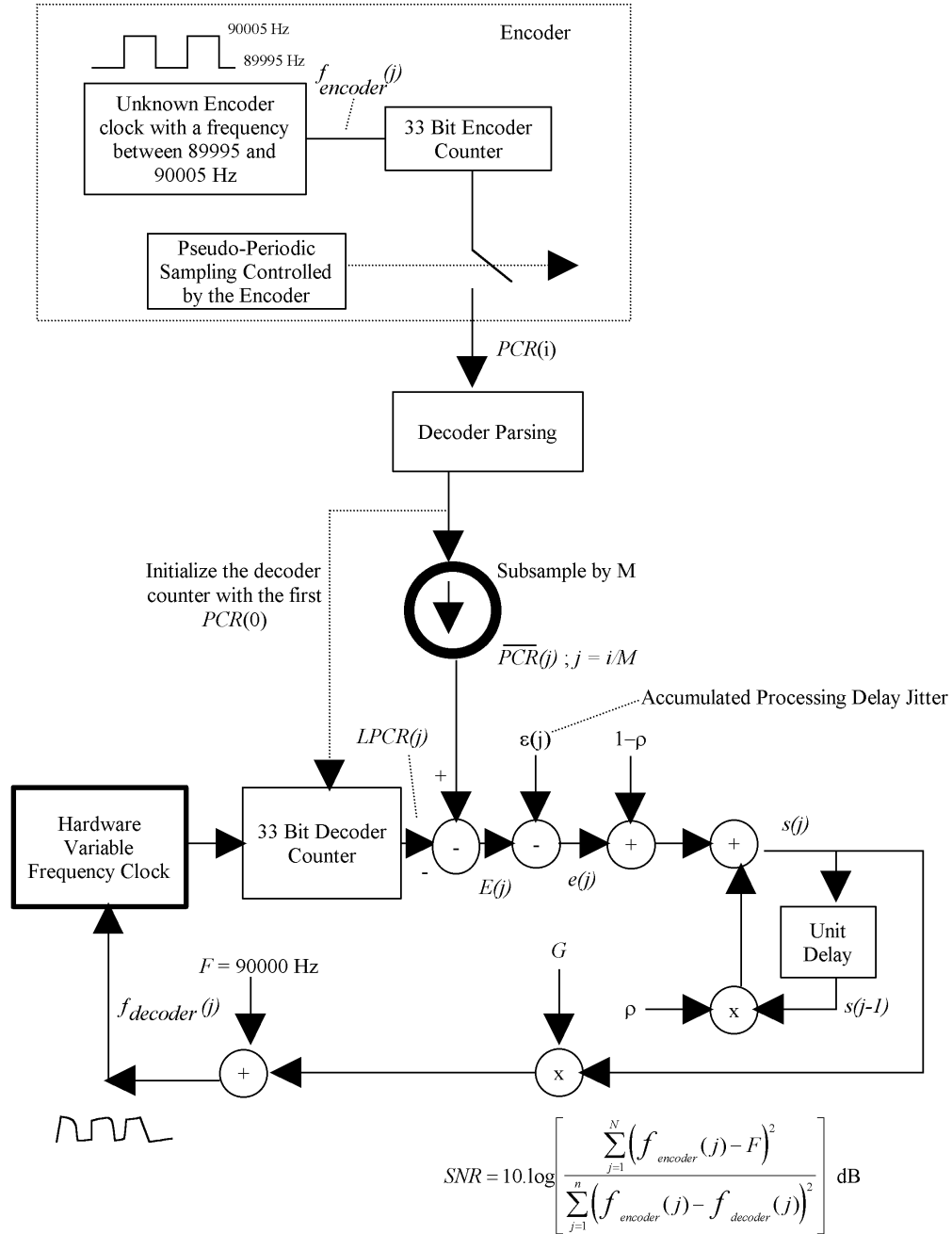


Fig.3 Synchronization System Model used in the analysis

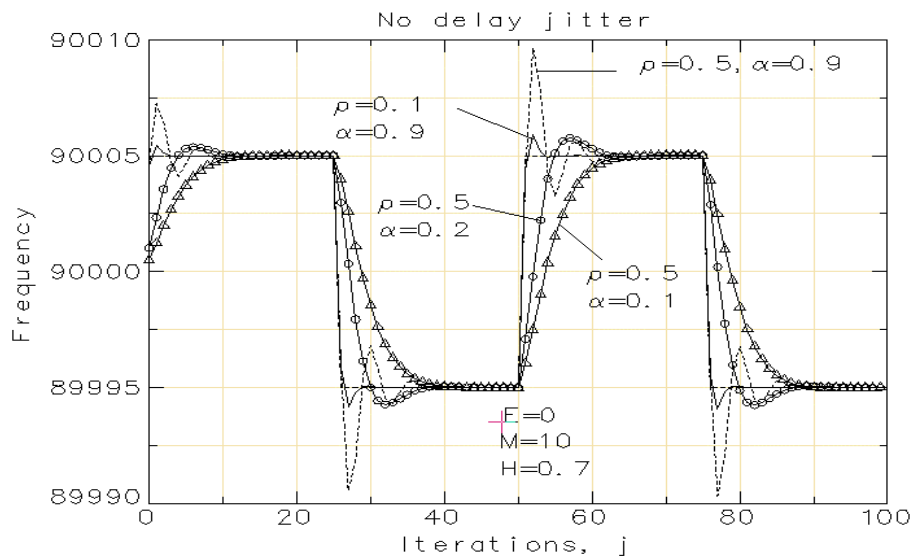


Fig. 4a. The response of the synchronization system without any delay jitter.

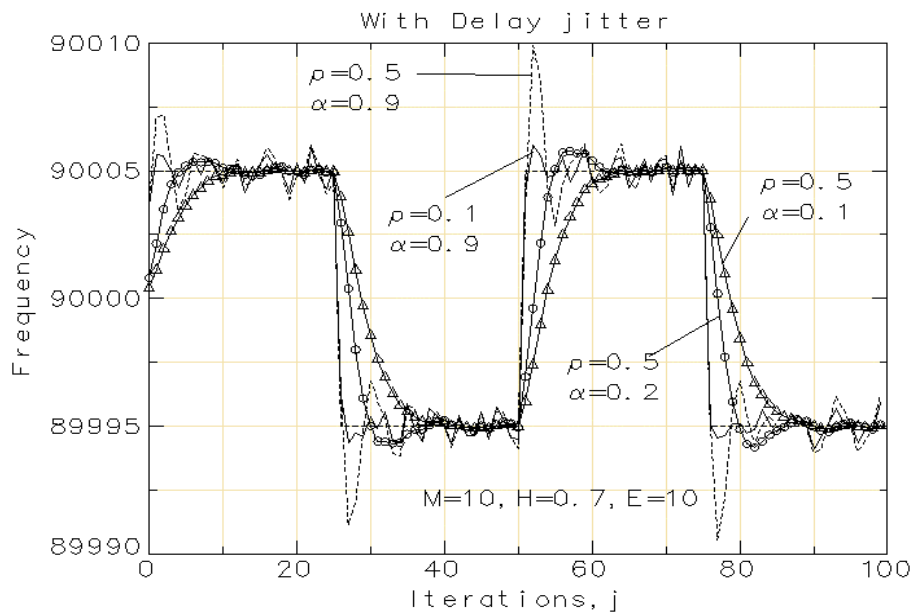


Fig. 4b. The behavior of the system with delay jitter ($E=10$).

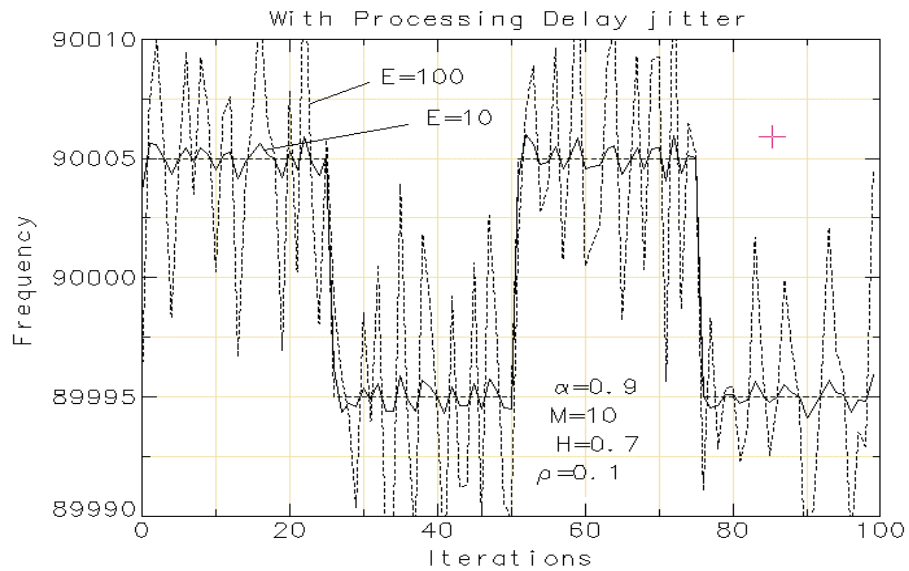


Fig. 5 The effect of a large delay jitter ($E=100$) clearly produces noisy synchronization behavior. The sub sampling factor ($M=10$) is mismatched to this condition.

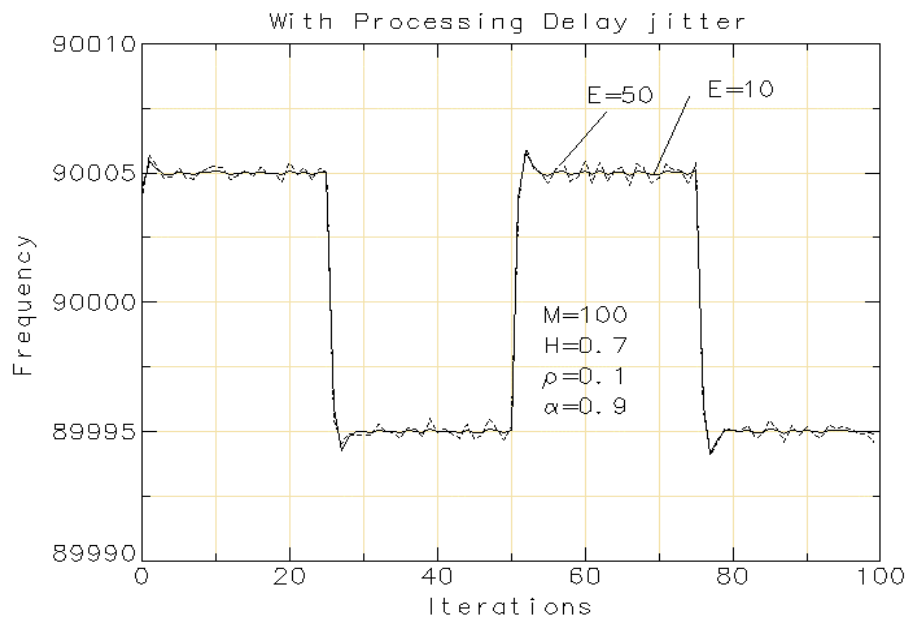


Fig. 6. By sufficiently increasing the sub sampling factor, the disturbance due delay jitter can be reduced.

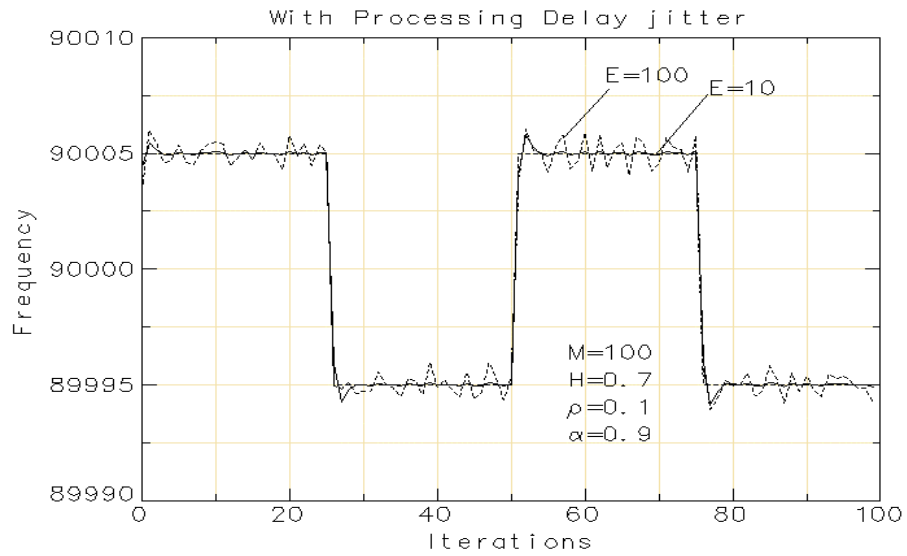


Fig. 7. Another example of the importance of sub sampling factor M . With a suitable high value, noise due to delay jitter can be reduced.

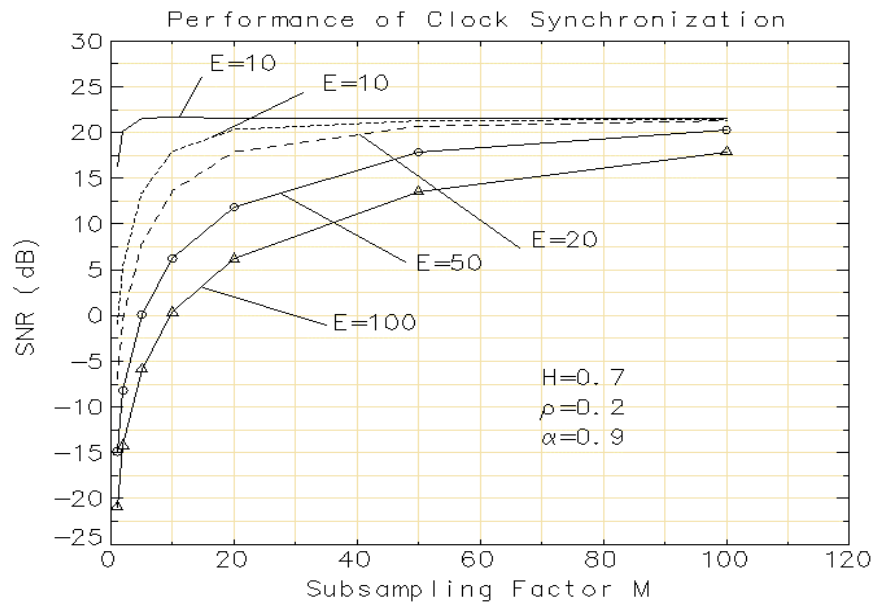


Fig. 8 The overall synchronization performance of the system as a function of the sub sampling factor M .

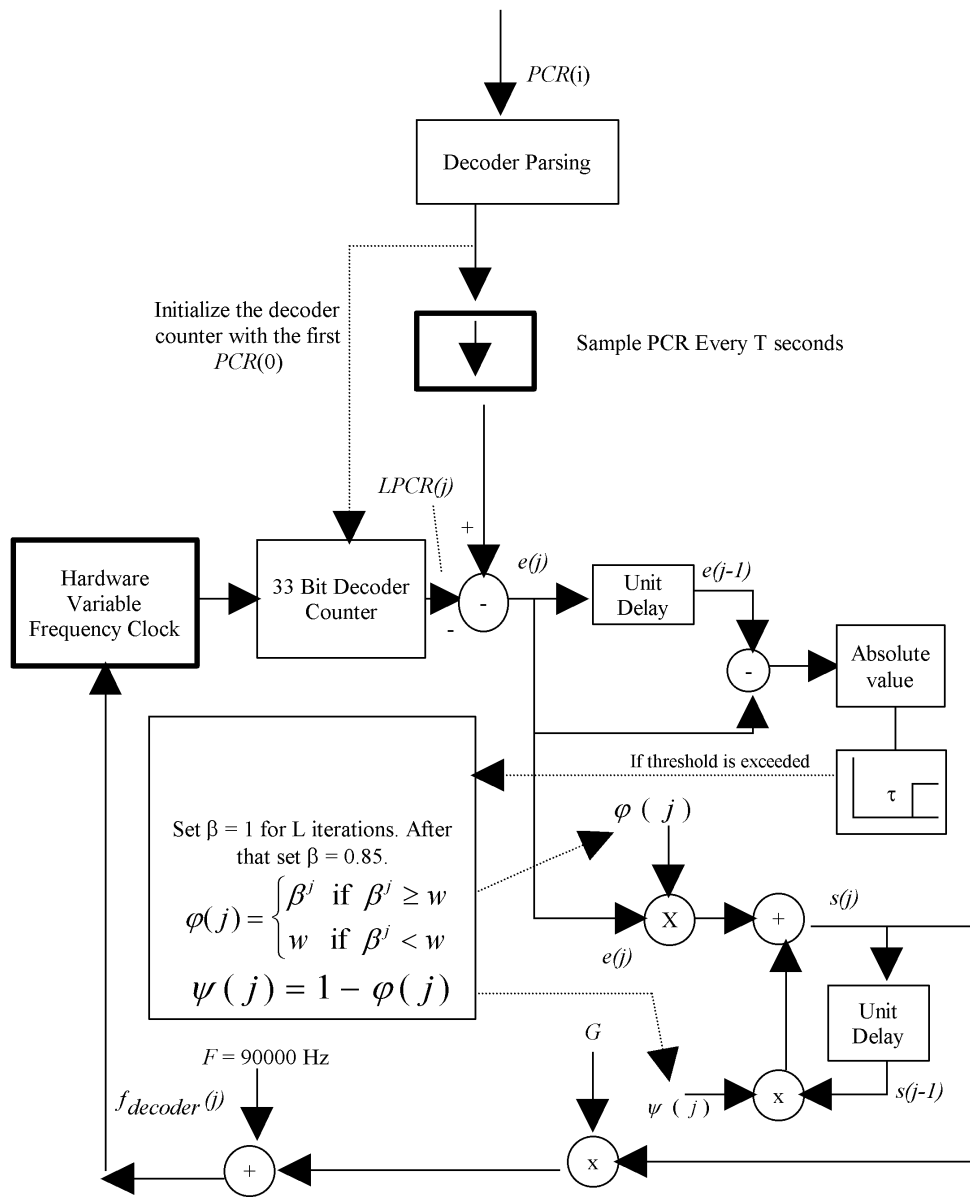


Fig. 9. Synchronization System with adaptive subsampling and filtering.

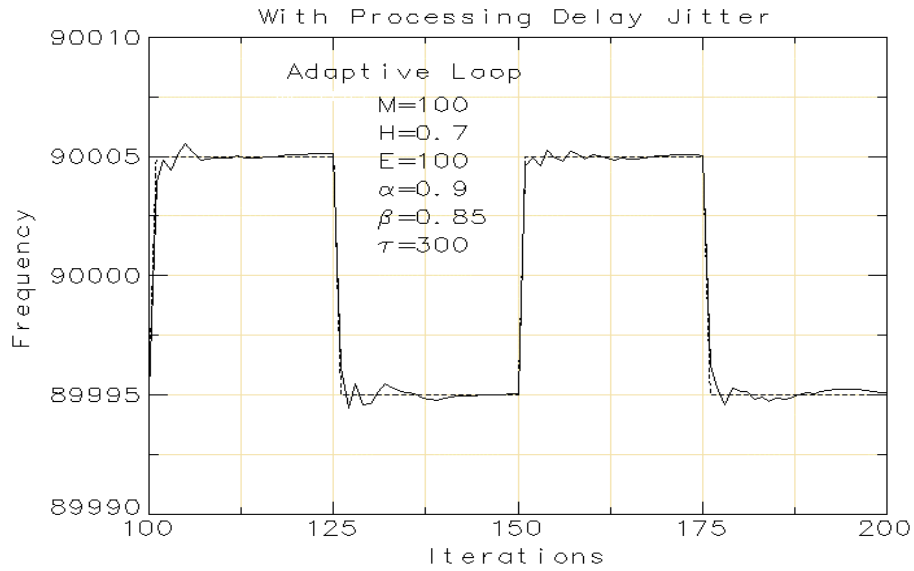


Fig.10. Demonstration of how adaptive filtering clearly improved the performance: it offers instantaneous acquisition as well as noise removal by filtering.

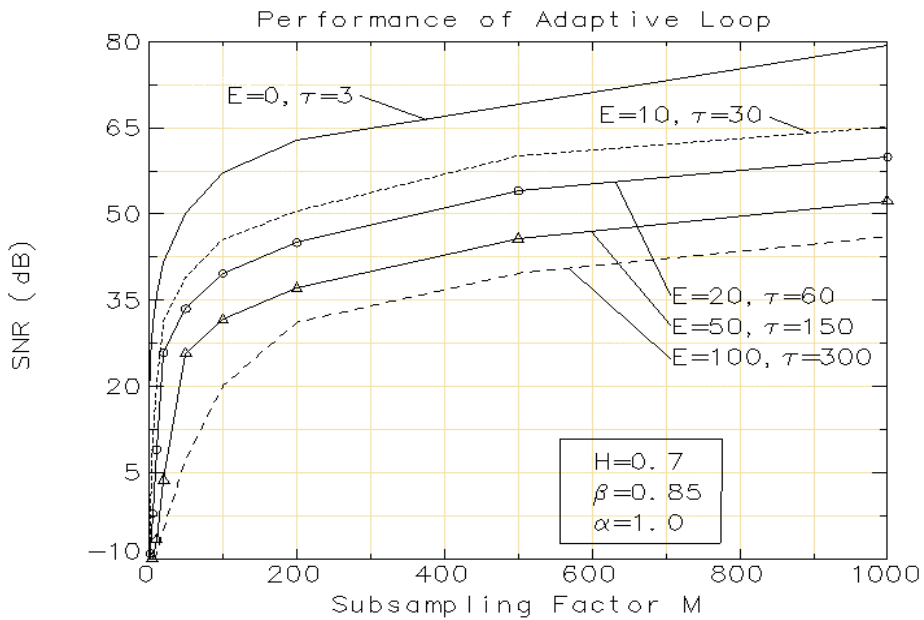


Fig. 11. The overall system performance of the adaptive loop as a function of the sub sampling factor M .