Fall 1-31-2000

Editing MPEG2 video in compressed domain

Ren Egawa
New Jersey Institute of Technology

ABSTRACT

EDITING MPEG2 VIDEO IN COMPRESSED DOMAIN

by
Ren Egawa

As MPEG2 becomes the heart of most digital video applications, the need to edit an MPEG2 bitstream video in compressed domain has increased. To realize this need, two major technical issues must be resolved: decoder buffer and segment extraction.

In the decoder buffer issue, the edited stream must ensure that the buffer would not overflow or underflow as the result of the editing. This thesis invented a mathematical model that could describe the editing task and predict the editing effect on the buffer. Based on this model, the thesis then built a method to edit a stream safely. This method could produce an edited stream that maintained an MPEG2 decoder's buffer behavior as if the decoder were receiving a non-edited stream.

In the segment extraction issue, the proper method to extract a segment from a source bitstream was discussed. The segment must be independent, that is, no coded picture in the segment would be associated with any other picture data outside of the segment. A proper extraction should also maintain the decoding order and display order of the picture sequence. This thesis visited several past studies on this issue, and proposed a practical method to perform segment extraction.

Simulation results on the discussed methods were also presented.

EDITING MPEG2 VIDEO IN COMPRESSED DOMAIN

by
Ren Egawa

A Master Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirement for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

January 2000

APPROVAL PAGE

EDITING MPEG2 VIDEO IN COMPRESSED DOMAIN

Ren Egawa

[illegible]

Dr. Ali N. Akansu, Thesis Advisor Date
Professor and Co-Director of Electrical and Computer Engineering, NJIT

[illegible]

Dr. Yun-Qing Shi, Committee Member Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Necdet Uzun, Committee Member Date
Assistant Professor of Electrical and Computer Engineering, NJIT

[illegible]

Dr. A. Aydin Alatan, Committee Member Date
Research Associate, NJ Center for Multimedia Research, NJIT

BIOGRAPHICAL SKETCH

Author:	Ren Egawa
Degree:	Master of Science in Electrical Engineering
Data:	January 2000

Undergraduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2000

- Bachelor of Science in Electrical Engineering,
  University of Arizona, Tucson, AZ, 1990

Major: Electrical Engineering

To my beloved wife, Rujing.

ACKNOWLEDGMENT

I would like to express my deepest appreciation to my advisor Dr. Ali N. Akansu and Dr. A. Aydin Alatan of Research Associate, NJ Center for Multimedia Research, NJIT. Dr. Alatan not only supervised my thesis, but also provided valuable and countless comments. Special thanks are given to Dr. Yun-Qing Shi and Dr. Necdet Uzun for actively participating in my committee.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

Compression has been a truly enabling technology for digital video applications. For example, without compression HDTV cannot be delivered to the home over a 6 MHz channel. The dominant compression standard has been the MPEG2 Standard [1]. It covers two major applications below:

1. Digital Broadcasting:
    - BSS (Broadcasting Satellite Service)
    - CATV (Cable TV distribution on optical networks, copper, etc.)
    - DTTB (Digital Terrestrial Television Broadcasting)
    - WB (Web Broadcasting)
2. Storage Media:
    - DVD (Digital Video Disk)
    - DVC (Digital VCR)
    - ISM (Interactive Storage Media)

The MPEG2 Standard inherited from its processor, the MPEG1 Standard [2]. In contrast to MPEG1 that targets Storage Media applications, MPEG2 addresses wider applications such as Digital Broadcasting. The BBS industry, the DVB digital transmission format in Europe [3], and the ATSC format for transmitting digital TV in the U.S. [4], have all standardized on the MPEG2 compression format. Similarly, network television has embraced the use of compression for delivering digital video contents from remote sites to network headquarters.

As with any new technology, the Digital Broadcasting system has experienced growing pains when actually implementing the MPEG2 Standard. By far, the largest technical issue is editing of MPEG2 bitstreams. This issue needs to be resolved before television production can take place in the compressed domain. A television production needs an editing system regardless of the change or advance in technology. The editing may be a commercial insertion in the middle of a movie source, or appending a program to a pre-transmitted program. In the uncompressed domain, this has been a simple procedure [5]. However, this requires decoding the bitstream back to uncompressed domain, editing the decoded video, and finally compressing the video data again. This cascading decode/encode process has the following disadvantages:

- The picture quality can degrade very rapidly as the number of cascade coding stages increases.
- The encoding and decoding process require substantial processing power, memory, and delays.

It is therefore desirable to remain in the compressed domain to the fullest extent possible. Yet, when we try to edit an MPEG2 bitstream, we quickly find that what had been simple switching between two video sources has suddenly become a complex task due to the following two major technical challenges in compressed domain:

1. The number of bits to represent a coded picture varies from picture to picture.
2. The coded pictures may be encoded together with its neighboring pictures, and the decoded order and the displayed order of the picture sequence may be different.

The first challenge involves the state of a decoder's input buffer. An MPEG2 bitstream requires careful management of buffer fullness. When a decoder's input buffer receives an MPEG2 bitstream, there is an inherent buffer occupancy at every point in time. The buffer fullness corresponds to a delay, or the length of time that a byte can spend in the input buffer before the byte is removed from the buffer for decoding. An MPEG2 bitstream is carefully created so that this buffer will not overflow or underflow the buffer. Suppose now that we have two such bitstreams that were encoded independently. If we concatenate these two bitstreams without careful consideration, the resulting new stream is most likely going to overflow or underflow the decoder's input buffer sooner or later. Several papers have mentioned that redundant data stuffing may be used to solve this problem [6], [7]. However, this thesis is believed to be the first paper to provide a mathematical model to analyze this problem, and shows a method to determine the exact number of bits of such redundant data stuffing.

The second challenge involves extracting a segment from a pre-encoded bitstream, such that the segment can be concatenated with another segment subsequently. Segment extraction is an important task, because an improperly extracted segment can potentially create a decoded picture that would be totally different from the encoder's intent. This phenomenon could occur because the current picture may have been coded together with a previously transmitted picture. If this previous picture is absent from the segment due to a bad extraction, then the current picture can not be decoded correctly. Even worse, the transmitted order of the coded picture in the bitstream may be different from the display order of the decoded picture. Therefore, even if the current picture is decoded correctly, it may be displayed to a viewer at a wrong time and in a wrong order with respect to its neighboring pictures. To avoid this kind of improper segment extraction, SMPTE 312M has standardized a method based on a concept called *splice point* [8], where two kinds exist as follows:

- *In points*, places in the bitstream that can be the start of the segment
- *End points*, places in the bitstream that can be the end of the segment

In essence, SMPTE 312M requires that the original source encoder provide marked points in the bitstreams, so that a stream editor could safely choose a start and an end point of a segment. However even if the encoders neglect to encode *splice points* in the streams, which is what most encoders do as of the writing of this thesis, the stream editor still needs to perform proper segment extraction. The thesis will show how to identify the safe points without relying on *splice point*. When a *splice point* cannot be found at a needed location in a bitstream, some studies have used a picture-type conversion technique [9], which effectively force a non-*splice point* to a *splice point*. This thesis will show that not all picture-type conversions are necessary, and will identify only one kind of conversion that is worth the effort.

CHAPTER 2

OVERVIEW OF MPEG2 ALGORITHM

2.1 MPEG2 Video Compression Algorithm

MPEG2 video compression algorithm employs a combination of complex processing techniques. The techniques involve block-by-block based DCT, DPCM, non-linear quantization scale, Huffman-like variable length coding, among others. For detail pixel-level algorithm, one is suggested to consult with the MPEG2 Standard itself [1]. This thesis is not interested in pixel-level algorithm. It discusses compressed data in unit of pictures, and its focus is on the amount of each compressed picture data bits. In particular, this thesis intends to solve the technical challenges facing an MPEG2 bitstream editor, due to the following characteristics of the MPEG2 bitstream:

1. The compressed picture's data amount varies from one picture to the next.
2. Depending on the coded picture type, a picture may be coded entirely independently, while another picture may be coded together with its neighboring pictures.
3. The decoded picture may have to be presented to a viewer later than a future picture that has not being decoded yet.

The information describing all these three characteristics is coded in the bitstream. To retrieve this information, one only needs to understand the most elementary data unit in the MPEG2 Standard, called "Data Element", and learn the rules or "Semantics" that these Data Elements are delivered to a MPEG2 Decoder in correct order.

2.2 Definitions

This section covers all the notations used in this thesis. Many of the notations are bitstream Data Elements as defined in the MPEG2 Standard. How these Data Elements appear in a bitstream will explained in section 2.3.

- **SC**: Start Code, a Data Element of a unique 24-bit code in an MPEG2 bitstream, that is 23 0s followed by a 1. This pattern cannot appear in the bitstream for any purpose other than being identified as a Start Code. A Start Code followed by the following hexadecimal number has a special meaning and will be explained in section 2.3:

  1. **Picture Start Code: SC + hex 00**
  2. **Group Start Code: SC + hex B8**
  3. **Sequence Header Code: SC + hex B3**
  4. **Sequence End Code: SC + hex B7**

  Other tailing hexadecimal numbers can be found in the MPEG2 Standard; however, only the above four types of SC's are of this thesis's interest.

- **PSC**: Picture Start Code, a SC that signals the start of a coded picture data.
- **GSC**: Group Start Code, a SC that signals the start of a group of pictures.
- **SHC**: Sequence Header Code, a SC that indicates the start of a sequence.
- **SEC**: Sequence End Code. This is a SC that signals the end of the entire bitstream.
- **Coded Data**: All the data bits between any two adjacent SC's: SHC, GSC, PSC, or SEC. Therefore Coded Data can be categorized as follows.

  1. **Picture Coded Data:** all the data bits between a PSC and the next SHC, GSC, PSC, or SEC.
  2. **GOP Coded Data:** all the data bits between a GSC and the next PSC.

3. **SH Coded Data:** all the data bits between a SHC and the next GSC or PSC.

- **ZSB**: Zero Stuffing Bit, a Data Element of one bit with value of 0.
- **n**: Index of a Coded Picture, which counts from 0 and increments in coded order.
- **m**: Index of a decoded picture, which counts from 0 and increments in display order.
- **B(n)**: All the bits of the n'th PSC and its Coded Data, plus GSC and/or SHC and their Coded Data, if any, immediately preceding the PSC. If SEC follows the Coded Picture, then the SEC is included in B(n).
- **b(n):** All the bits in B(n) excluding its Picture Coded Data and SEC, if exists.
- **D(n):** B(n) – b(n) + b(n+1).

To help visualize the notations of B(n), b(n), and D(n) in a bitstream structure, Figure 1 illustrates their relations with various types of Coded Data.

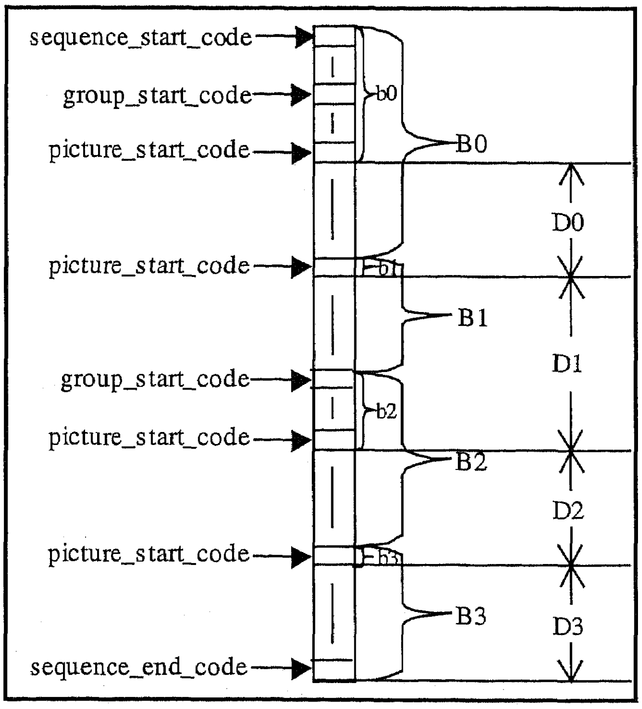


Figure 1 Use of B(n), b(n), and D(n) to Express an MPEG2 video bitstream

- **Segment**: A segment is a portion of an MPEG2 bitstream. A segment must start and end with a B(n) boundary. For example, two segments can be cut from Stream1 and Stream2, respectively. It can be assumed that Stream1 and Stream2 are generated by two independent encoders.
- **Concatenated Stream**: An MPEG2 video bitstream that is consist of Stream1 as the front portion and Steam2 as the end portion.
- **Stream Editor**: In this thesis, the Stream Editor is used to describe a hypothetical module that extracts Segment1 and Segment2 from Stream1 and Stream2, respectively. Next, the Stream Editor concatenates Segment1 and Segment2 to form a Concatenated Stream.
- **VBV**: Video Buffer Verifier, a hypothetical buffer model defined in the MPEG2 Standard. This video buffer is conceptually connected to the output of an MPEG2 Encoder, and is also connected to the input of an MPEG2 Decoders. All the bits of an MPEG2 video bitstream must enter VBV at a certain Bit-Rate. At the time of decoding picture data B(n), all the bits of B(n) shall be removed from VBV instantaneously. This is because the MPEG2 Standard defines VBV as a no-decoding delay model. The Encoder and Decoder treat their own VBV in the identical manner.
- **vbv_delay(n)**: a bitstream Data Element that indicates the length of time that B(n) shall remain in VBV. This time length counts from the time when D(n)'s first bit enters VBV until the time when B(n)'s bits are removed from VBV for decoding.
- **t(n)**: the time that B(n) should be removed from VBV for decoding. This time is determined by registering the time that B(n)'s first bit enters VBV and the value of vbv_delay(n).

- **VBV Underflow**: a condition that all the data bits of B(n) are not in VBV at time t(n). When VBV Underflow occurs, the MPEG2 Standard specifies that the Decoder shall wait until t(n+1) or the next earliest t(n+c), where 'c' is a positive integer, such that all data bits of B(n) are available in VBV, and then decode B(n).
- **VBV Overflow**: a condition that VBV has no room to receive a new bit. This means that the new bit is entering VBV when VBV is already filled up with all the new bits, portion of which can only be removed at the next t(n). When VBV Overflow occurs, data bits will be lost, since this new bit is most likely going to overwrite another bit in the VBV. The MPEG2 Standard prohibits VBV Overflow under all circumstances.
- **I-Picture or Intra Picture**: a picture coded using information only from itself.
- **P-picture or Predictive-coded Picture**: A picture that is coded using motion compensated prediction from past reference pictures.
- **B-picture or Bidirectionally predictive-coded picture**: A picture that is coded using motion compensated prediction from past and/or future reference pictures.
- **Low Delay Mode**: An MPEG2 video bitstream of low delay mode contains no B-picture. A Low-Delay Mode bitstream is allowed to cause VBV Underflow.

2.3 Basic MPEG2 Video Bitstream Semantics

This section reviews some basic MPEG2 semantics to cover this thesis's interest. In a most general classification, a MPEG2 bitstream is made of the following three classes:

1. Start Code (SC), i.e., PSC, GSC, SHC, SEC
2. Coded Data, i.e., Picture Coded Data, GOP Coded Data, SH Coded Data.
3. Zero Stuffing Bit (ZSB)

A legal MPEG2 bitstream must start with a SHC, followed by at least one PSC, and ended with a SEC. The SHC may appear more than once in a bitstream, but each time it appears it must be followed by at least one PSC before another SHC appears again. The GSC is optional, but if exists, the next Picture Coded Data must be an I-Picture data. The last picture of a GOP is the last picture before a new SHC appears. ZSB can appear immediately before any SC. There can be as many ZSB's as needed to perform rate control. A typical MPEG2 bitstream is shown as follows:

Table 1 Typical MPEG2 Video Bitstream Data Contents

{SHC + SH Coded Data}
+
{GSC + GOP Coded Data}
+
{Zero Stuffing Bits}
+
{PSC + I-Picture Coded Data}
+
:
+
{Zero Stuffing Bits}
+
{SHC + SH Coded Data}
+
{PSC + I/P/B-Picture Coded Data}
+
:
+
{Zero Stuffing Bits}
+
{GSC + GOP Coded Data}
+
{PSC + I-Picture Coded Data}
+
:
+
{PSC + I/P/B-Picture Coded Data}
+
{SEC}

respectively. Then both segments would be "VBV-Compliant" as well. The Stream Editor's next task is to concatenate Segment1 and Segment2 properly to produce a Concatenated Stream that would still be "VBV-Compliant". If the Stream Editor failed to concatenate both segments properly, then there is no assurance that the Concatenated Stream would still be "VBV-Compliant". In fact in most cases, it would be VBV-incompliant. A VBV-incompliant bitstream could potentially cause buffer overflow, and force the Decoder to loose bitstream data. As Figure 2 illustrates, this may occur very quickly. In the figure, the vertical lines show that picture data are removed from VBV for decoding. The diagonal lines show that fresh data are entering VBV.

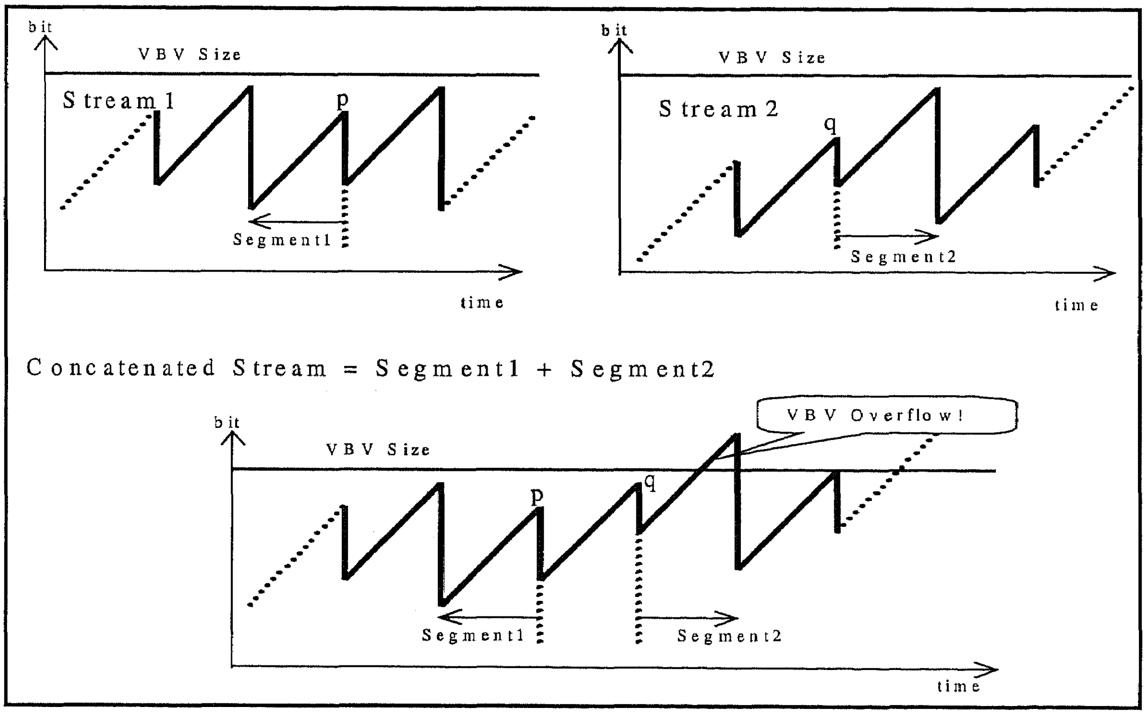


Figure 2 Example of a Concatenated Stream not VBV-Compliant

In this example, the Stream Editor simply connected the end of Segment1 (point p) to the beginning of Segment2 (point q). The result is a Concatenated Stream that mismatched Segment2's original VBV status path, and overflowed VBV.

As Figure 2 has demonstrated, a proper concatenation is a critical task for editing. A trial-and-error approach to achieve this task is not an acceptable practice either, because the overflow point may occur much later in time. What the Stream Editor needs is a mathematical model that can describe the VBV activities at concatenation, and a concatenation method that can is built on this model. This chapter will discuss such model and method. However, one must first understand the VBV model as specified by the MPEG2 Standard, which is discussed in section 3.2, before proceeding to the concatenation model and method, which are discussed in section 3.3.

3.2 Mathematical Representation of VBV Model

This section contains no new concept, but to assist readers understand VBV models as specified in the MPEG2 Standard. There are two kinds of VBV models in the standard:

1. Constant Bit-Rate VBV Model
2. Variable Bit-Rate VBV Model

Each VBV model is discussed below. Note that all notations are defined in Section 2.2.

3.2.1 Constant Bit-Rate VBV (CBR VBV) Model

In Constant Bit-Rate VBV model, data bits enter VBV at the rate R(n):

$$R(n) = D(n) / T(n)$$

where

- R(n) is the Bit-Rate, in bits/second, that D(n) enters VBV.
- $T(n) = vbv\_delay(n) - vbv\_delay(n+1) + t(n+1) - t(n)$

For all n, R(n) must always be lower than or equal to Rmax:

$$R(n) \leq Rmax$$

where Rmax represents the upper bound bit rate of the MPEG2 Profile & Level for which the bitstream is coded. For example, if the bitstream is a Main Profile & Main Level bitstream, then Rmax is 15 Mbits/sec. Table 1 shows other Rmax values for different levels in Main Profile of the MPEG2 Standard.

Table 2 Bit-Rate Upper bound of MPEG2 Main Profile Bitstreams

Levels	Upper bounds for bit rates (Mbits/sec)
High	80
High-1440	60
Main	15
Low	4

Since R(n) can change from one 'n' to the next, the word "constant Bit-Rate" used by the MPEG2 Standard is actually misleading. One should keep in mind that MPEG2's "constant Bit-Rate" is piece-wise. That is, the Bit-Rate is constant only during T(n). However, there are two periods that the Bit-Rate must be always be at Rmax. They are the period that b(0) enters VBV, and the period that last D(n) enters VBV. One last note regarding R(n) is that the standard recommends that R(n) be constant throughout the entire bitstream, and that its value be consistent with the value actually coded as *bit_rate* in the bitstream Data Element, even though technically R(n) is allowed to vary.

The next figure summarizes the CBR VBV model. R(n) is assumed to be constant always in this example. Note that b0 and D7 enter VBV at Rmax rate, but other D(n) enter VBV at the rate dictated by its T(n) length. Therefore, if the Encoder intends to maintain a constant R(n) at any 'n', then the Encoder must carefully choose the value of *vbv_delay(n)* and *vbv_delay(n+1)* to be encoded in the bitstream. This is because these two Data Elements directly affect the value of T(n) as described before. After the last bit of D7 enters VBV, no more data bit enters VBV. Also note that the last bit of D(n) and b(n) always coincides, with the exception of D7 that coincides with B(7).

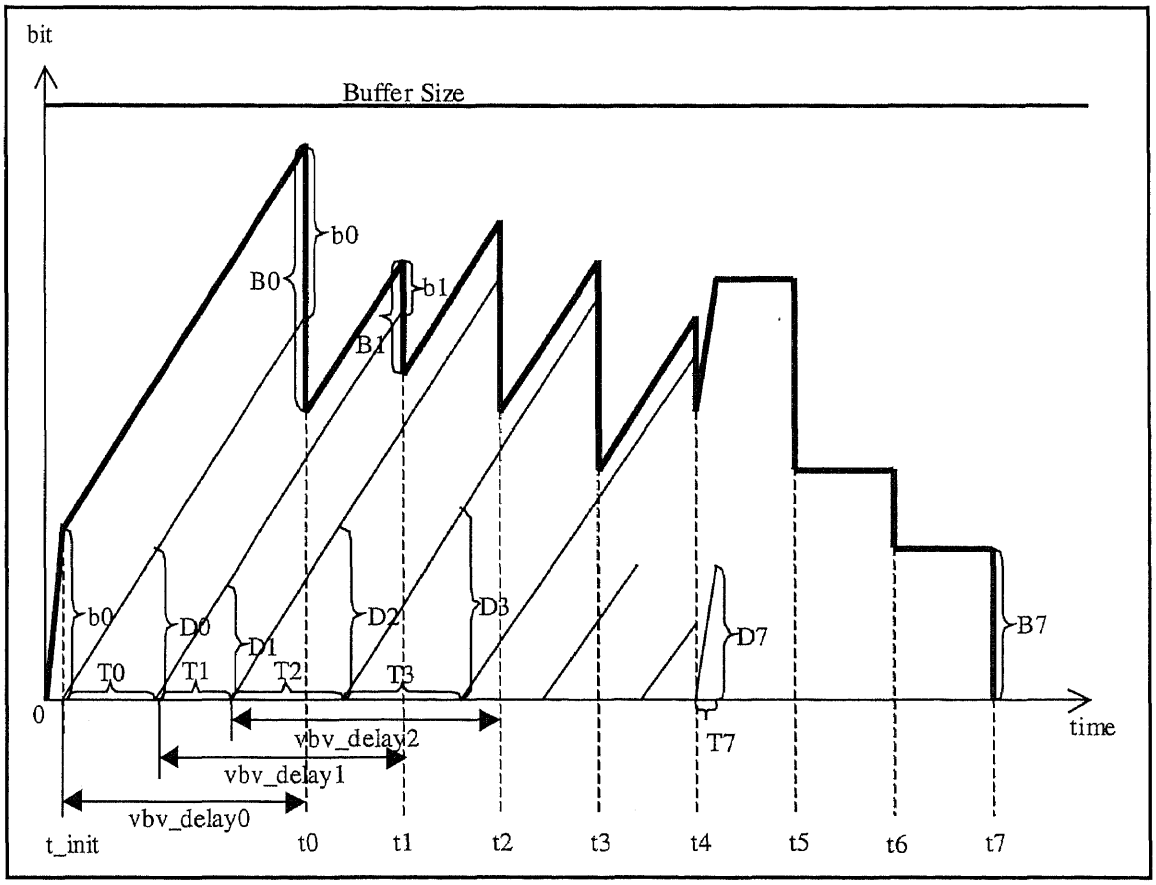


Figure 3 CBR VBV Model

3.2.2 Variable Bit-Rate VBV (VBR VBV) Model

In Variable Bit-Rate VBV model, the data bits always enter VBV at Rmax rate, when VBV is not full. When VBV is full, no data enters VBV until the next t(n) and after B(n) is removed from VBV. In VBR VBV, b(n) and D(n) are irrelevant, since the Bit-Rate is always either Rmax or 0. Similarly there is no concept of vbv_delay in VBR VBV model.

Buffer synchronization between the Encoder and Decoder is achieved through the following steps. First the bitstream enters VBV at Rmax. When the VBV reaches its buffer fullness for the first time, both Encoder and Decoder will mark that time as t(0). After t(0), both Encoder and Decoder will follow a set of MPEG2 rules to determine each subsequent t(n). These rules are affected by other Data Elements such as *frame_rate*. For example, if the *frame_rate* is coded as 29.97, then each subsequent t(n) will be separated by the time duration that is the inverse of the *frame_rate*. The next figure summarizes the VBR VBV model.

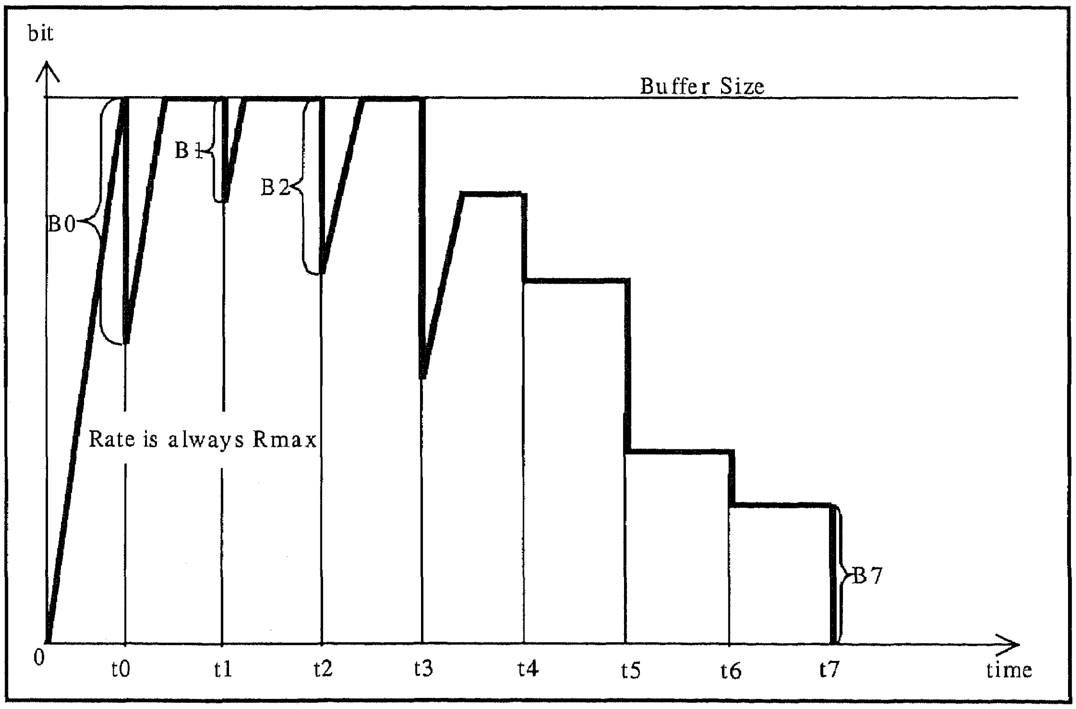


Figure 4 VBR VBV Model

3.3 VBV and Segment Concatenation

The key to concatenate Segment1 and Segment2 properly is to match Concatenated Stream's VBV to Segment2's VBV. This thesis proposes two methods to accomplish this task.

- Method 1: Insert a finite number of ZSB's (Nzsb) between the two segments.
- Method 2: Insert a wait period (Twait) after Segment1 entered VBV completely and before Segment2 starts entering VBV.

The choice of the method depends on the case of the segment concatenation:

- Case 1: Segment1(CBR) with no SEC + Segment2(CBR)
- Case 2: Segment1(CBR) with SEC + Segment2(CBR)
- Case 3: Segment1(VBR) with no SEC + Segment2(VBR)
- Caes 4: Segment1(VBR) with SEC + Segment2(VBR)
- Case 5: Segment1(CBR) with SEC + Segment2(VBR)
- Case 6: Segment1(VBR) with SEC + Segment2(CBR)

Note that CBR+VBR and VBR+CBR cases do not have "no SEC" option. This is because MPEG2 Standard requires that a SEC be inserted before changing the Bit-Rate mode. Each case will be discussed in the next sections. For simplicity the following notations are used in all sections:

- p = the index of Segment1's last picture.
- p+1 = the index of Stream1's picture immediately following Segment1. If Segment1 and Stream1 both end at the same picture, this index is undefined.
- q = the index of Segment2's first picture.
- F(x) = VBV fullness at time t(x) before data B(x) exits VBV
- BS(x) = VBV size specified by the Encoder that generated Segment(x)

3.3.1 Case 1: Segment1(CBR) with No SEC + Segment2(CBR)

In this case, Segment1 and Segment2 are both CBR, and SEC is not present at the end of Segment1. The method 1 (ZSB insertion) will be used when vbv_delay[p+1] is shorter than vbv_delay[q], and when Bit-Rate has to remain constant. There is no work required when vbv_delay[p+1] is longer than or equal to vbv_delay[q], and when Bit-Rate does not need to remain constant.

3.3.1.1 When (vbv_delay[p+1] ≥ vbv_delay[q]) and CBR is Not Required: In this case, Stream1's picture B(p+1) provides a vbv_delay value that is long enough to support vbv_delay value of Segment2's first picture, B(q). Thus the Stream Editor can connect Segment1 and Segment2 with no additional intervention. The Concatenated Stream will naturally match Segment2's original VBV model after t(p+1). The next Figure 5 demonstrates an example where p=1, p+1=2, and q=1. Note however that during the concatenation point, the Bit-Rate dropped in the Concatenated Stream's VBV path, i.e.,

R1' < R1, and T1 < T1'

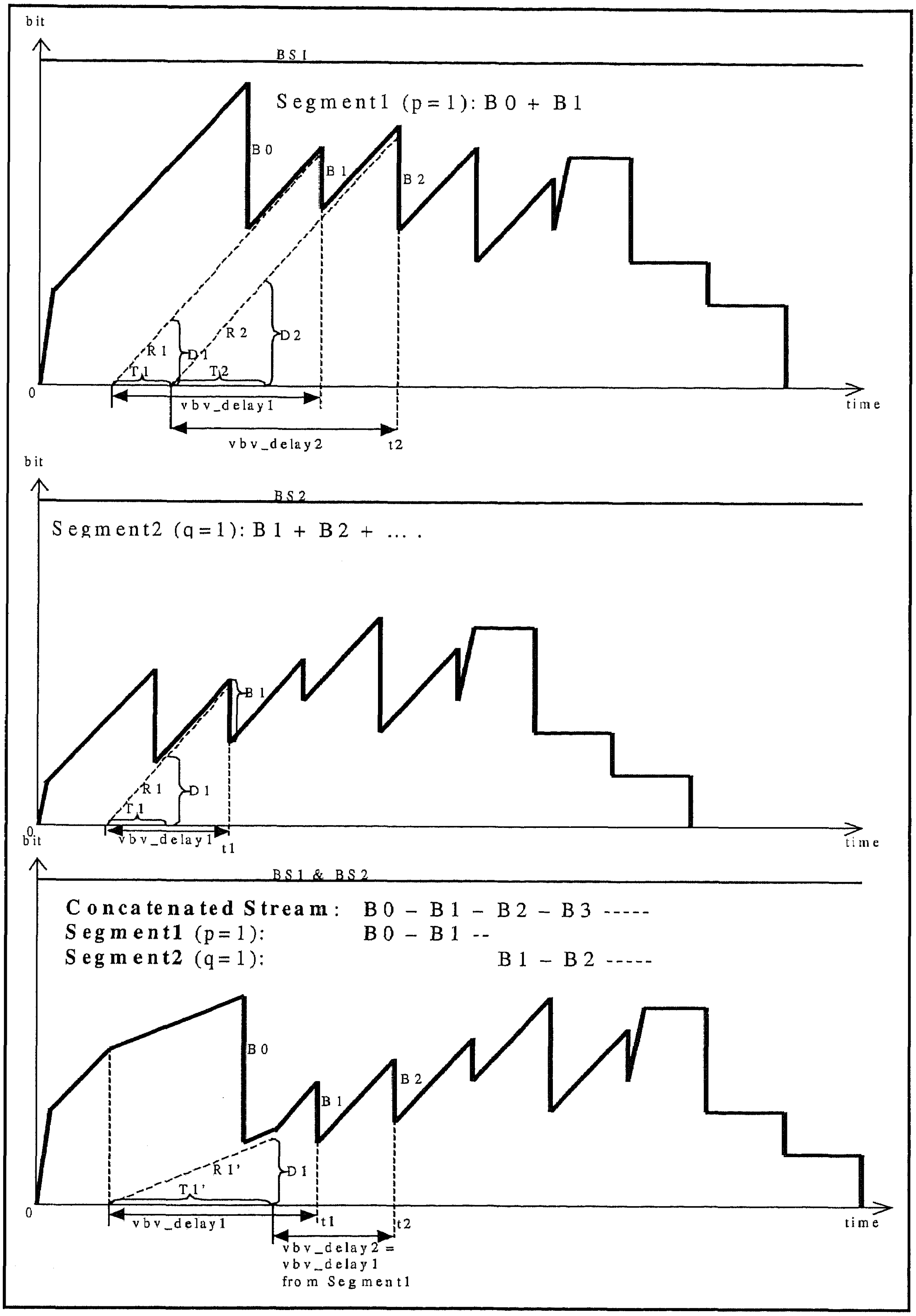


Figure 5 CBR to CBR Direct Concatenation when vbv_delay[p+1] ≥ vbv_delay[q]

The Bit-Rate drop effect would always exist, when the Stream Editor simply hooked Segment1's tail to Segment2's head, and when vbv_delay[p+1] > vbv_delay[q]. As mentioned earlier, even though MPEG2 allows R(n) of CBR be any rate as long as it is lower than Rmax, MPEG2 also recommends that R(n) be constant throughout the entire bitstream, and that R(n) match the actual value of *bit_rate* coded in the bitstream. The coded value of *bit_rate* in the bitstream is referring to R1 but not R1'. Therefore, if the Stream Editor is required to maintain the same Bit-Rate, R(n), throughout the entire bitstream, and is also required to keep *bit_rate* field accurately reflecting the actual Bit-Rate in the VBV, then the Stream Editor cannot simply hook the two segments together back-to-back. Instead, the Stream Editor must employ the technique discussed in the next section to achieve this requirement.

3.3.1.2 When (vbv_delay[p+1] < vbv_delay[q]) or CBR Is Required: When Stream1's picture B(p+1) provides a vbv_delay value that is too short to support vbv_delay value of Segment2's first picture, B(q), the Stream Editor cannot connect Segment1 and Segment2 back-to-back. Doing so will result in a Concatenated Stream that differs from Segment2's original VBV status path. Below, the thesis provides a mathematical model for analysis, and proposes a method to perform this task properly.

The method uses ZSB to adjust VBV fullness and timing to match Segment2's VBV. The Stream Editor will first determine the number of ZSB needed, and then insert these ZSB before B(q). The number of ZSB (or "Nzsb" for short) is determined as follows.

$$\begin{aligned} Nzsb &= (Tnext - Treq) * R(p), && \text{if } Tnext \geq Treq \\ &= (Tnext + \Delta t*k - Treq) * R(p), && \text{if } Tnext < Treq \end{aligned}$$

where:

Tnext = vbv_delay(p+1) + b(p+1)/R(p), see Figure 6.

Treq = vbv_delay(q) + b(q) / R(p)

Δt = Frame Rate of Segment1

k = a smallest integer that satisfies: $(Tnext + \Delta t*k - Treq) \geq 0$

Note that the smallest possible "k" value is 1, because k=0 automatically implies that $Tnext \geq Treq$, in which case the Stream Editor can simply use the first equation. To help visualize the temporal implications of these equations, Figure 6 demonstrated an example using k of 1. Note that the time to remove picture(q)'s data has been delayed to:

$$t(q) = t(p) + \Delta t + \Delta t*k = t(p+2)$$

If no ZSB was inserted, then t(q) would have been:

$$t(q) = t(p+1)$$

When the second equation must be used, i.e., when Tnext < Treq, then VBV underflow will occur. MPEG2 allows VBV underflow only in Low-Delay Mode. Therefore, if the segments are not Low-Delay Mode segments, then the Stream Editor must find another point p or point q that satisfies $Tnext \geq Treq$. Alternatively the Stream Editor may insert a

SEC at the end of Segment1 and avoid this underflow restriction. The MPEG2 Standard does not prohibit underflow after t(p) if B(p) ends with the SEC. The next section will discuss the concatenation method when B(p) ends with a SEC.

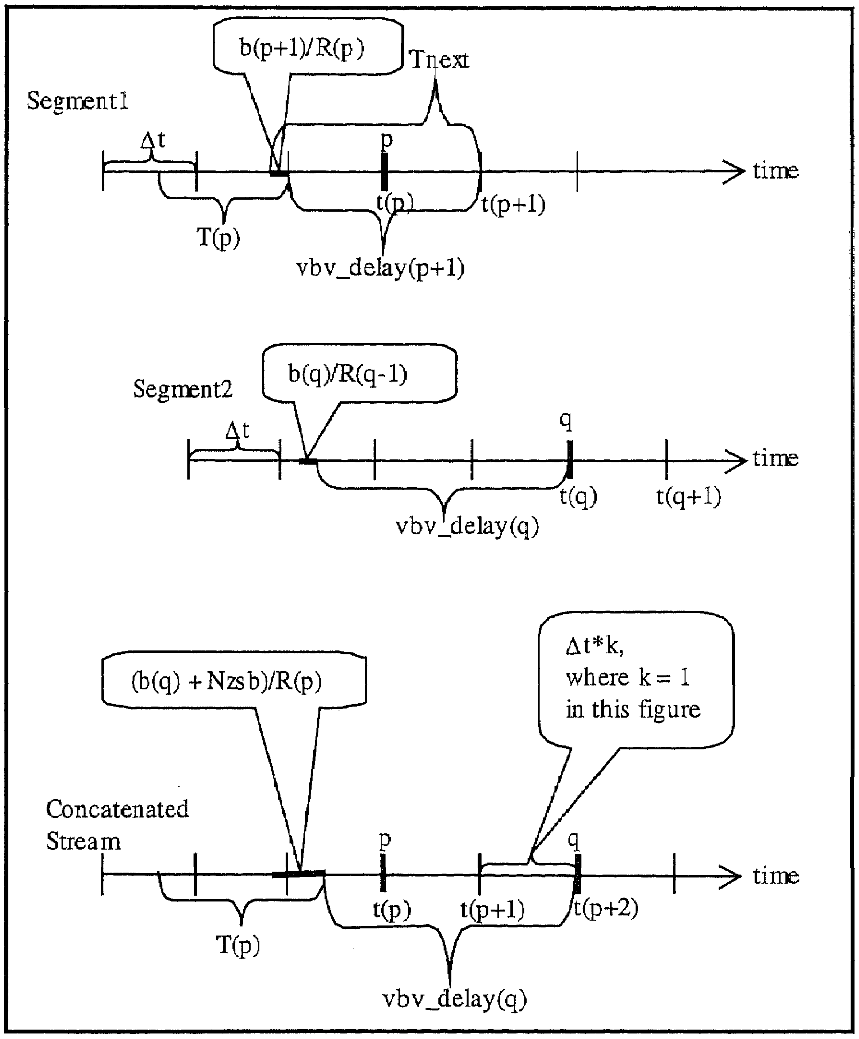


Figure 6 CBR to CBR Concatenation with no SEC when Treq > Tnext

Figure 7 summarizes this section with an example using p=3 and q=4. The Stream Editor appends the ZSB's to the end of Segment1's B3, which increases the size of B3 in the Concatenated Stream. As a result, VBV fullness F(4) in Segment2 and in Concatenated Stream matched at time t(4). Thereafter, both VBV continues to match to the end.

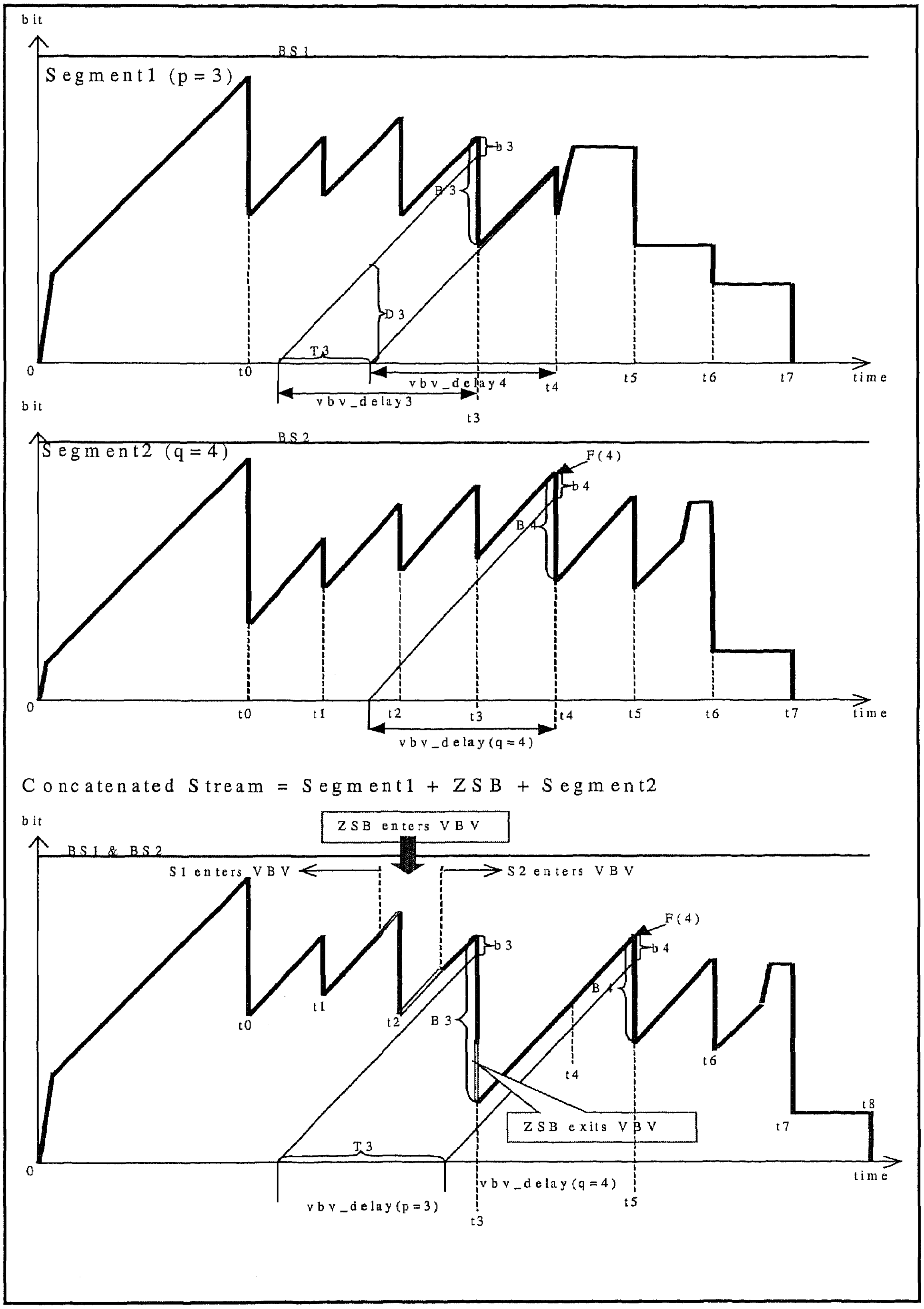


Figure 7 CBR to CBR Concatenation when vbv_delay[p+1] < vbv_delay[q]

3.3.2 Case 2: Segment1(CBR) with SEC + Segment2(CBR)

In this case, Segment1 and Segment2 are both CBR, but SEC is present at the end of Segment1. The method used here employs Twait instead of ZSB. During a Twait period, the Bit-Rate is zero, which is similar to VBR case when VBV fullness was reached. The use of Twait offers an advantage over that of inserting Nzsb in the case that SEC is present. Because of the SEC, the risk of buffer overflow exists during the concatenation point, if the Stream Editor inserts ZSB. This risk exists because Segment1's last picture B(p) enters VBV at Rmax1 rate, and Segment2 first picture's headers b(q) also enters VBV at R2max rate. Ironically SEC also makes use of Twait possible without violating MPEG2. This is because MPEG2 Standard has completely left VBV unspecified during the period that Segment1's SEC enters the VBV and Segment2's first bit enters the VBV.

The Stream Editor still must first find Nzsb, and then uses Nzsb to determine Twait. In ZSB method, ZSB are actually transmitted, while in Twait method, no ZSB was transmitted. The equation to find Nzsb is similar to that in the previous section. The difference is that T(p) here is a function of Rmax1, and Nzsb is a function of R2max:

$$\begin{aligned} \text{Nzsb} &= (\text{Tnext} - \text{Treq}) * \text{R2max}, && \text{if Tnext} \geq \text{Treq} \\ &= (\text{Tnext} + \Delta t * k - \text{Treq}) * \text{R2max}, && \text{if Tnext} < \text{Treq} \end{aligned}$$

where:

$\text{Tnext} = \text{vbv\_delay(p)} + \Delta t - \text{T(p)}$

$\text{Treq} = \text{vbv\_delay(q)} + \text{b(q)} / \text{R2max}$

Δt = any value; recommended value is Frame Rate of Segment1

k = a smallest integer that satisfies: $(\text{Tnext} + \Delta t * k - \text{Treq}) \geq 0$

Note that the smallest possible "k" value is 1, because k=0 automatically implies that Tnext $\geq$ Treq, in which case the Stream Editor can simply use the first equation.

Once Nzsb is found, then the Stream Editor can easily derive Twait from the equation below.

$$\text{Twait} = \text{Nzsb} / \text{R2max}$$

The time to remove B(q) in the Concatenated Stream is therefore stretched to:

$$t(p+1) = t(p) + \Delta t + \Delta t * k$$

Figure 8 summarizes this section with an example. Note that VBV fullness F(q) in Segment2 and F(p+1) in the Concatenated Stream matched. Naturally, VBV continued to match even after time t(p+1).

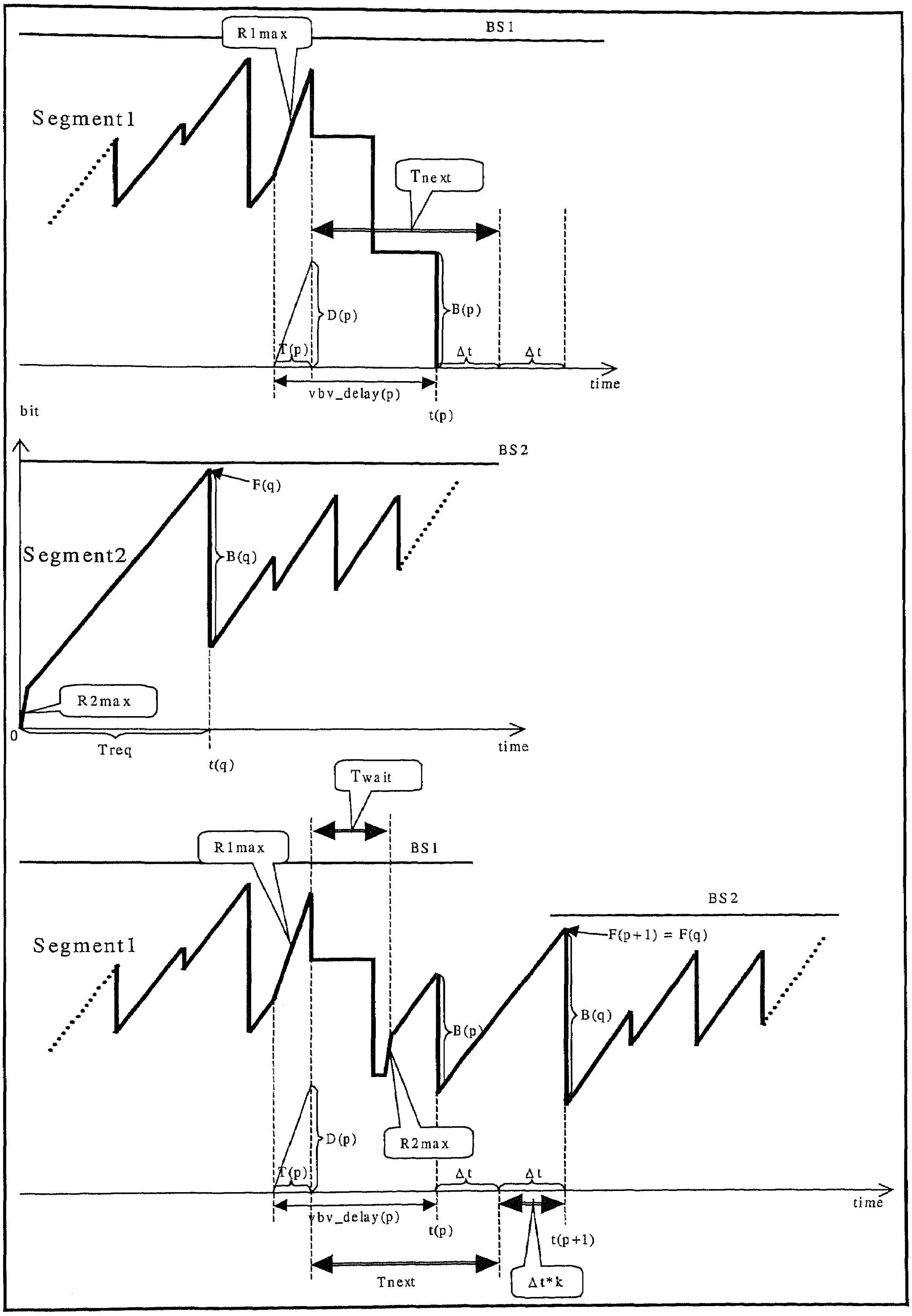


Figure 8 CBR to CBR Concatenation with SEC

3.3.3 Case 3: Segment1(VBR) with No SEC + Segment2(VBR)

In this case, both Segment1 and Segment2 are VBR, and SEC is not present at the end of Segment1. The method 1 (ZSB insertion) will be used when F(p+1) ≠ F(q). There is no ZSB-related work required when F(p+1) = F(q).

3.3.3.1 When F(p+1) = F(q): In this case, VBV fullness is the consistent at t(p+1) and t(q) before the data B(p+1) and B(q) are removed from VBV. Therefore the Stream Editor can safely concatenate Segment1 and Segment2 with no additional work. The Concatenated Stream will naturally follow Segment2's original VBV status path. Figure 9 illustrates an example with p=2 and q=3. Note that the buffer fullness when B(p+1) enters VBV and that when B(q) enters VBV are different. However, this does not cause problem as long as F(p+1) is equivalent to F(q).

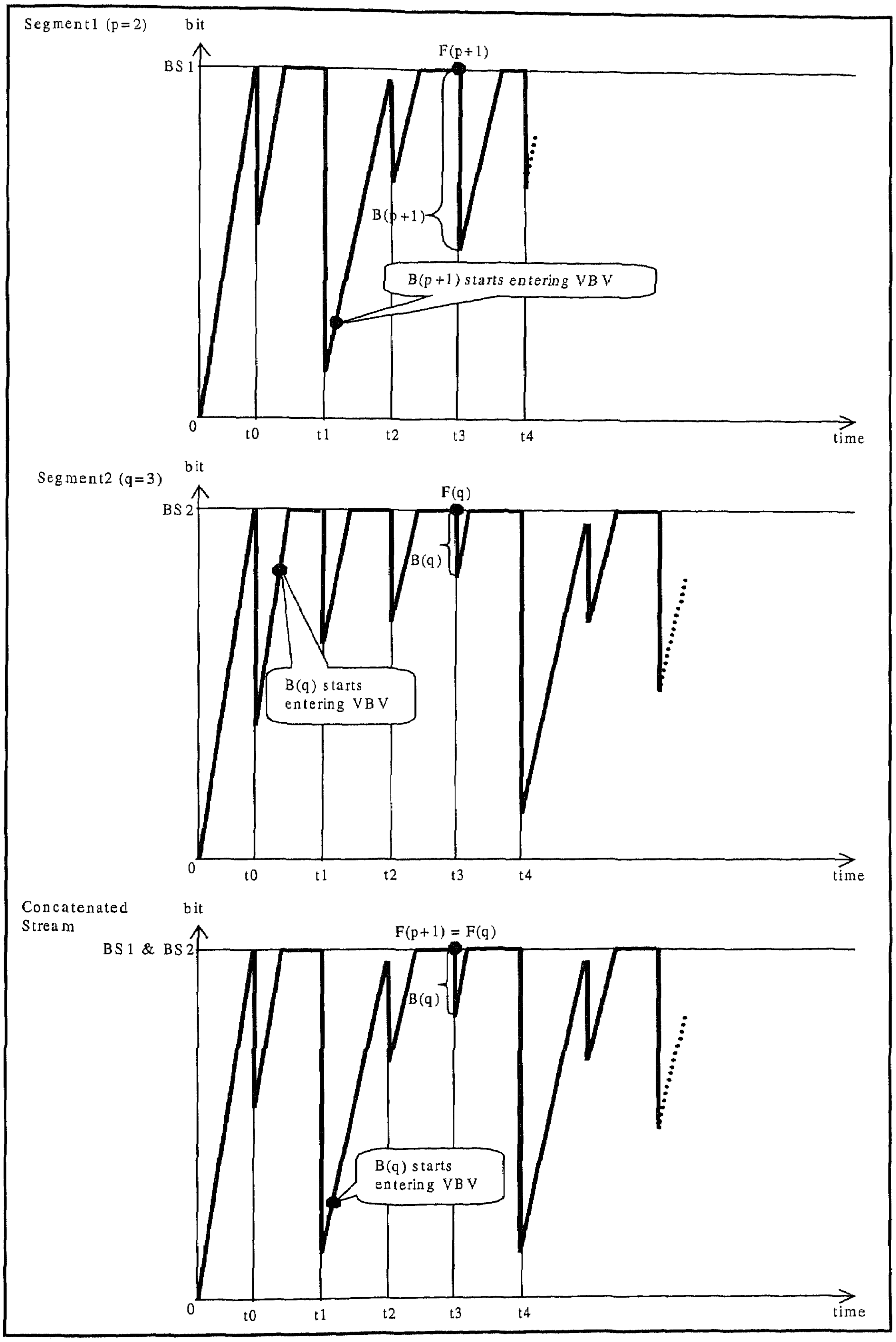


Figure 9 VBR to VBR Concatenation with No SEC when F(p+1) = F(q)

3.3.3.2 When F(p+1) > F(q): In this case, VBV fullness is more full at t(p+1) than at t(q). Therefore the Stream Editor cannot simply concatenate Segment1 and Segment2. Doing so will result in a Concatenated Stream that mismatches with Segment2's VBV.

The Stream Editor uses ZSB to adjust VBV fullness and timing to match the VBV status path between Segment2 and the Concatenated Stream. The number of ZSB, Nzsb, is determined as follows.

Nzsb = Nnext – Nnew

where:

- Nnext = Amount of Stream1's data entering VBV during the period from when B(p+1) starts entering VBV to the time of t(p+1).
- Nnew = Amount of Segment2's data entering VBV during the period from when Segment2 starts entering VBV to the time of t(q).

The effect of ZSB insertion can be easily seen on the next Figure 10. Let's introduce another variable f(p) which represents Stream1's VBV fullness at t(p) after removing B(p). The figure shows that ZSB has lowered the original f(p) to a new low point f'(p), thereby matching F(q) in Segment2 and Concatenated Stream. Note that the insertion point is immediately before the SEC, thus B(p) is increased as shown in the Concatenated Sequence VBV.

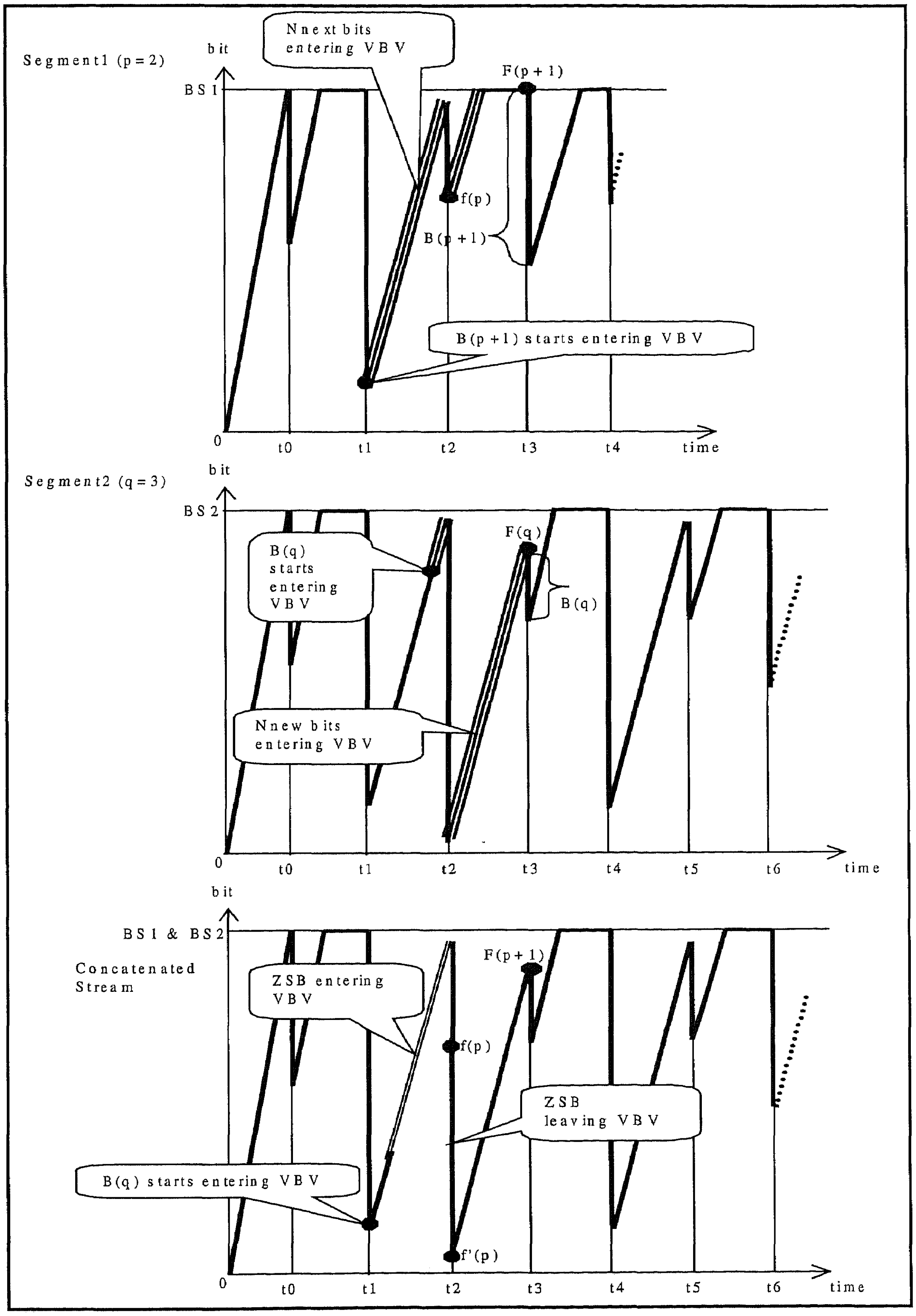


Figure 10 VBR to VBR Concatenation when $F(p+1) \geq F(q)$

3.3.3.3 When F(p+1) < F(q): In this case, VBV fullness is less full at t(p+1) than at t(q). Therefore the Stream Editor cannot simply hook Segment1 with Segment2. Doing so will result in a Concatenated Stream that mismatches Segment2's VBV.

The Stream Editor uses ZSB to adjust VBV fullness and timing to match the VBV between Segment2 and Concatenated Stream. The Stream Editor will need to determine the number of necessary ZSB, and then inserts these ZSB immediately before the SEC. However since F(p+1) < F(q), the Stream Editor must first determine a new position, $t(p+1+\Delta t*k)$ such that the inequality below can be satisfied.

$$\text{Nextra} - \text{Nnew} \geq 0$$

where:

- Nextra = Amount of data entering VBV during the period from when B(p+1) starts entering VBV to $t(p+1+\Delta t*k)$.
- Nnew = Amount of Segment2's data entering VBV during the period from when Segment2 starts entering VBV to t(q).
- Δt = any value; recommended value is Frame Rate of Segment1

Obviously k has be > 1, otherwise F(p+1) would have been ≥ F(q), in which case Nzsb could be determined from the previous cause 3.3.3.2. Finally Nzsb is determined by:

$$\text{Nzsb} = \text{Nextra} - \text{Nnew}$$

Note that VBV underflow will occur in this case. As mentioned before, the MPEG2 Standard allows VBV underflow only in Low-Delay Mode. Therefore if the segments are not Low-Delay Mode, then the Stream Editor must find another 'p' point or 'q' so that $F(p+1) \geq F(q)$, i.e., k=0. Alternatively the Stream Editor may insert a SEC at the end of Segment1 to avoid this underflow restriction. The MPEG2 Standard does not prohibit underflow after t(p) if B(p) ends with a SEC. The next section describes the concatenation method when Segment1 ends with a SEC.

3.3.4 Case 4: Segment1(VBR) with SEC + Segment2(VBR)

In this case, both Segment1 and Segment2 are VBR, but SEC is present at the end of Segment1. The number of zero stuffing bits can be derived using the nearly identical algorithm in the previous section, i.e., Nzsb = Nextra – Nnew. The only difference is the definition of Nextra:

Nextra = Amount of VBR data that can enter VBV at R2max rate during the period from when all B(p) data entered VBV and until the time: $t(p) + \Delta t + k*\Delta t$.

where

- Δt = any value; the recommended value is Frame.
- k is a minimum integer which must satisfy the inequality:

$$\text{Nzsb} = \text{Nextra} - \text{Nnew} \geq 0$$

The Bit-Rate during the period when the ZSBs are entering the VBV is also assumed to be entering at R2max, although the MPEG2 Standard has left this number unspecified.

3.3.5 Case 5: Segment1(CBR) with SEC + Segment2(VBR)

In this case, there is always a SEC at the end of Segment1. The Stream Editor must determine the time to remove B(q) in the Concatenated Stream which will be stretched to:

$t(p+1) = t(p) + \Delta t*k.$

- Δt = any value; the recommended value is Frame Rate of Segment 1.
- k is a minimum integer to satisfy: $Nzsb = Nnext - F(q) \geq 0$

Nnext is the number of bits that may enter at R2max rate during the period between when all B(p) data entered VBV and until t(p). Determining Nnext is not a straightforward process. We will rely on the next Figure 11 to explain this process. From this figure, Nnext starts counting slightly before t(p-2), and the value of Nnext is the sum of the four terms below:

$Nnext = Np\text{-}2 + Np\text{-}1 + Np + N_{p+k*\Delta t}$

where:

- $Np\text{-}2 = \max[0, \min\{BS2\text{-}F(p\text{-}2), Tp\text{-}2*R2max\}]$
- $Np\text{-}1 = \max[0, \min\{BS2\text{-}(N_{p-2} + F(p\text{-}1)\}, Tp\text{-}1 * R2max\}]$
- $Np = \max[0, \min\{BS2\text{-}(Np\text{-}1 + F(p)\}, Tp * R2max\}]$
- $N_{p+k*\Delta t} = \max[0, \min\{BS2\text{-}(Np + 0\}, t(p+k*\Delta t) * R2max\}]$

The zero entry in the max[0,x] function is to handle the case when x becomes a negative number. This can happen when BS2 < BS1. The min{a,b} function is to stop Segment2 entering VBV when BS2 is full. Obviously the larger k's value is, the larger Nnext is. The Stream Editor needs to find a smallest 'k' that produces large enough Nnext to exceed the value of F(q). The ZSB's are inserted at the end of B(q), so that f(q) will be consistent before and after the concatenation.

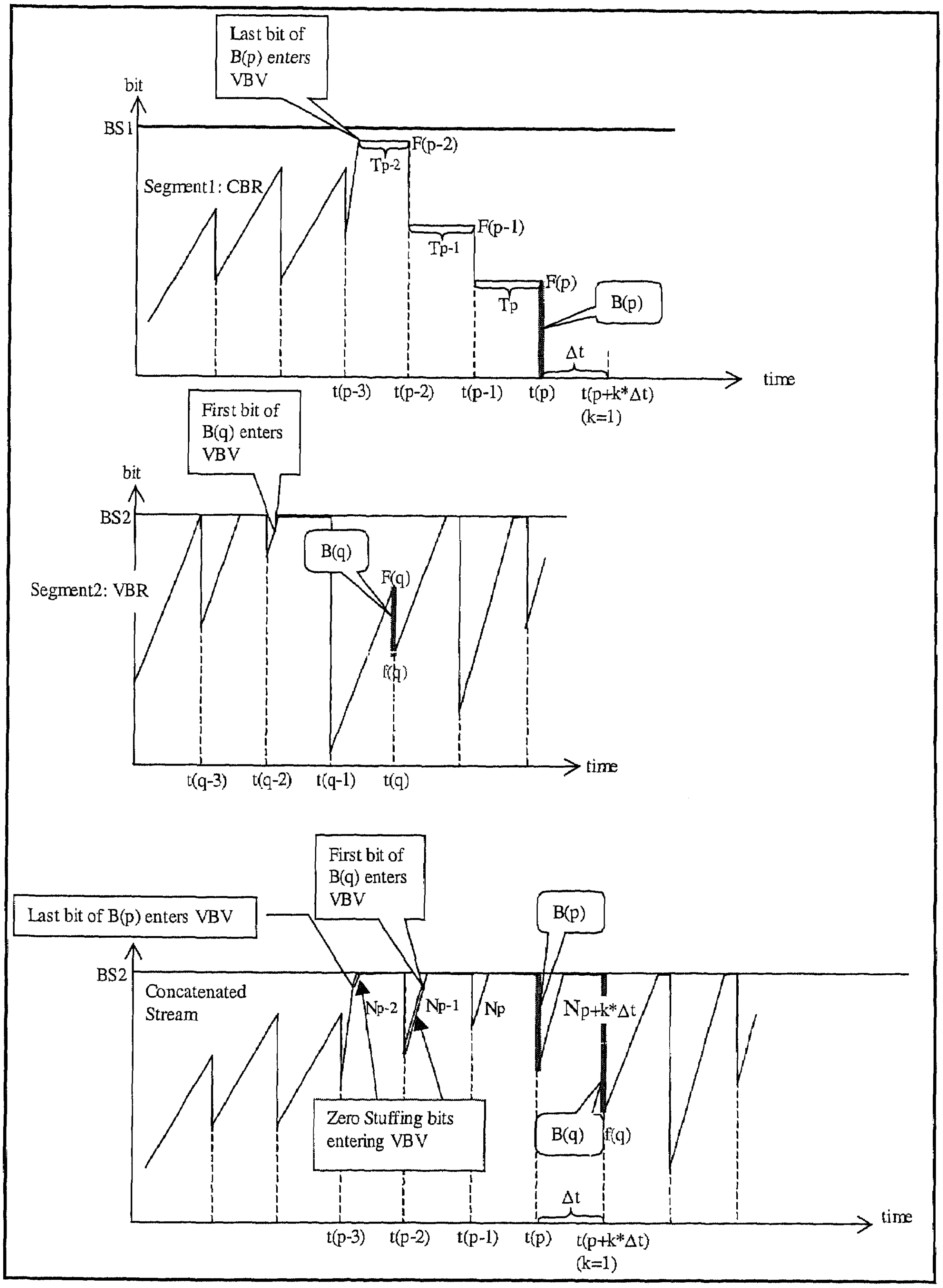


Figure 11 CBR to VBR Concatenation

3.3.6 Case 6: Segment1(VBR) with SEC + Segment2(CBR)

In this case, there is always a SEC at the end of Segment1. Since Segment2 is CBR, once Segment2 starts entering VBV, it cannot be stopped when VBV fullness is reached. Therefore the Stream Editor must deal with the risk of VBV overflow in the beginning stage of sending Segment2 after Segment1's SEC. For this reason, the Stream Editor employs Twait method.

As with other cases where Segment1 ends with a SEC, the time difference between t(p+1) and t(p) can be expressed as follows:

$$t(p+1) - t(p) = \Delta t + \Delta t * k$$

where

- Δt = any value; the recommended value is Frame.
- k is a minimum integer which must satisfy the inequality:

$$\{Tnext - vbv\_delay(q)\} * R2max \geq b(q)$$

where Tnext is the period from the last bit of B(p) entered VBV to t(p+1).

The Twait period can then be determined as follows:

$$Twait = Tnext - (vbv\_dealy(q) + b(q)/R2max)$$

The following figure illustrates an example where k=0. Note that Segment2's F(q) matches Concatenated Stream's F(p+1), which is the beginning of VBV matching between the two.

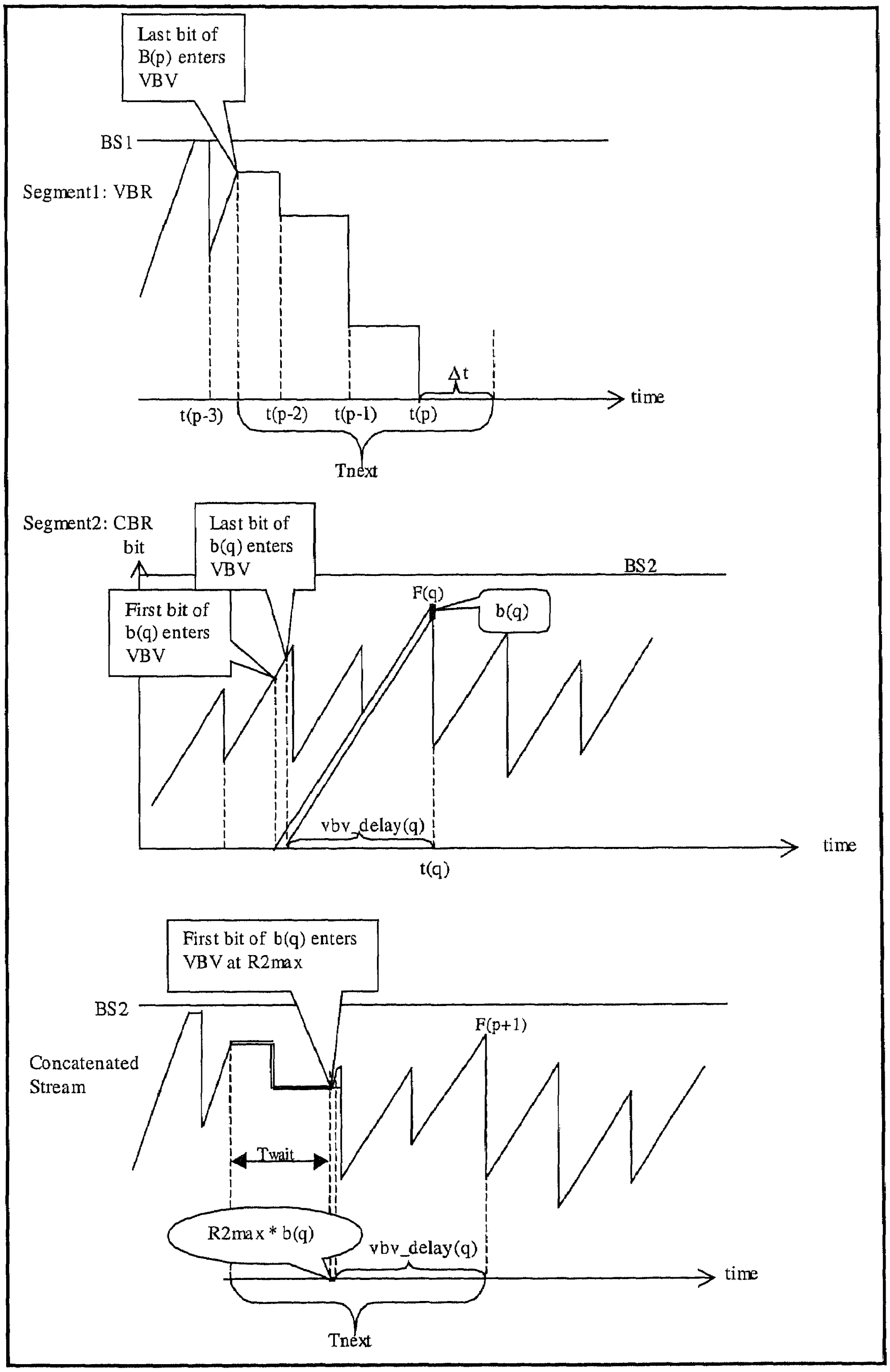


Figure 12 VBR to CBR Concatenation

3.4 Simulation Result

This simulation is performed to validate some theoretical aspects of the solutions described in this chapter. Particularly the case where a CBR to CBR concatenation with Tnext < Tnext is performed. This case is chosen because it demonstrates the iterative process of finding the value for 'k' (i.e., the smallest integer that satisfies $\text{Tnext} + \Delta t^*K - \text{Treq} \geq 0$). This iterative process is essentially same for finding the 'k' value all the other cases, including the VBR concatenation cases.

The simulation implements each step as described in Chapter 3, and finds the number of zero stuffing bits needed to insert between Segment1 and Segment2. The simulator then constructs the Concatenated Stream with Segment1, zero stuffing bits, and Segment2. Finally the simulation plots the Concatenated Stream's VBV status, and shows that it matches the VBV status of Segment1 and Segment2. For comparison purpose, the simulator also constructs a "bad" Concatenated Stream, in which Segment1 and Segment2 are simply "hooked" without careful consideration. The VBV status of this "hooked" stream is also shown.

In the first step, the simulator receives two CBR streams (Stream1 and Stream2) which are created independently by unknown sources. A preliminary analysis of the bitstreams reveals that both streams share the following characteristics:

- VBV Buffer Size: 1.65 Mbits
- 150-picture long
- Frame rate of 29.97 Hz

The simulator then plots Stream1's and Stream2's VBV status as shown in Figure 13 and Figure 14, respectively.

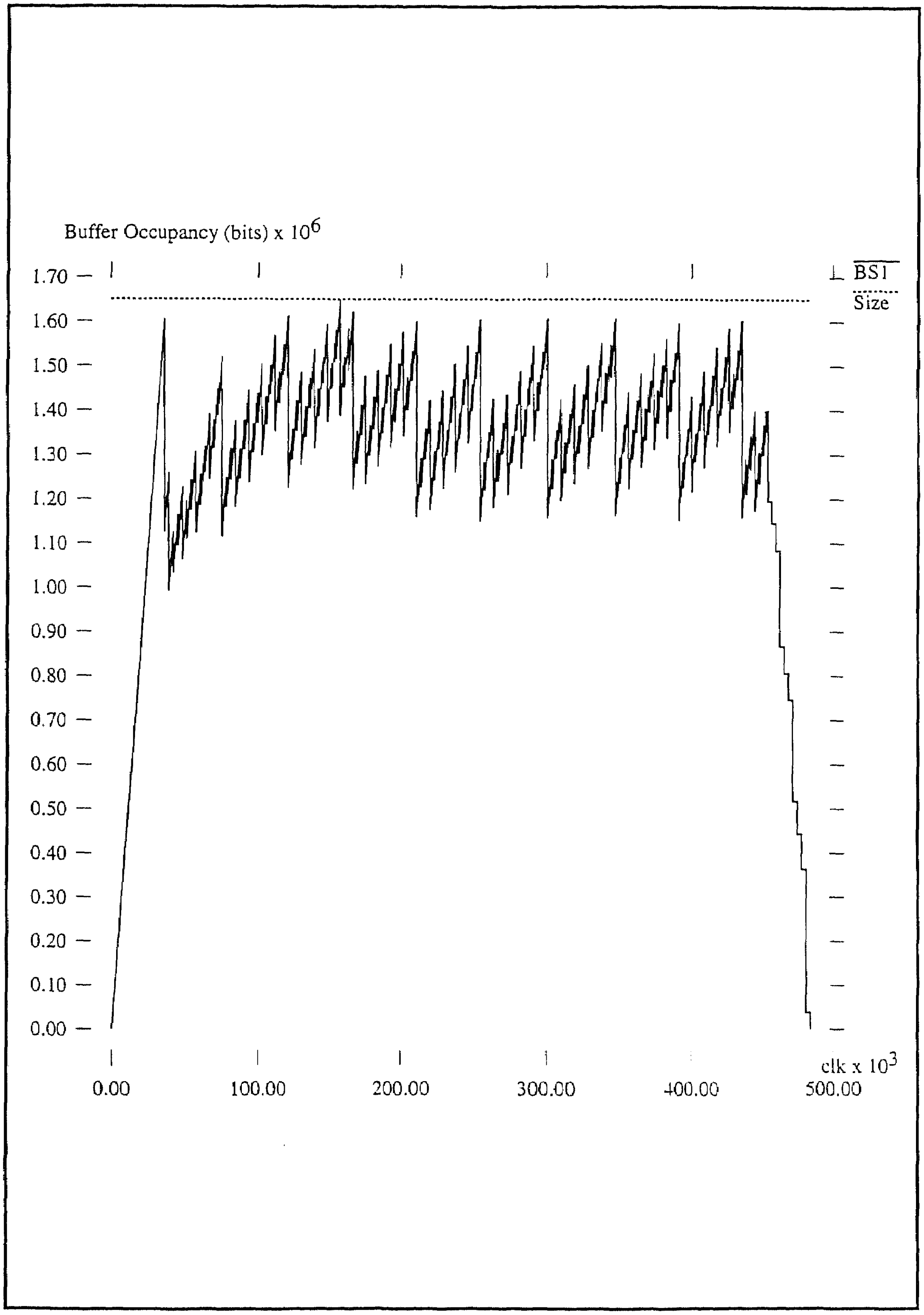


Figure 13 Stream 1 Original VBV Status

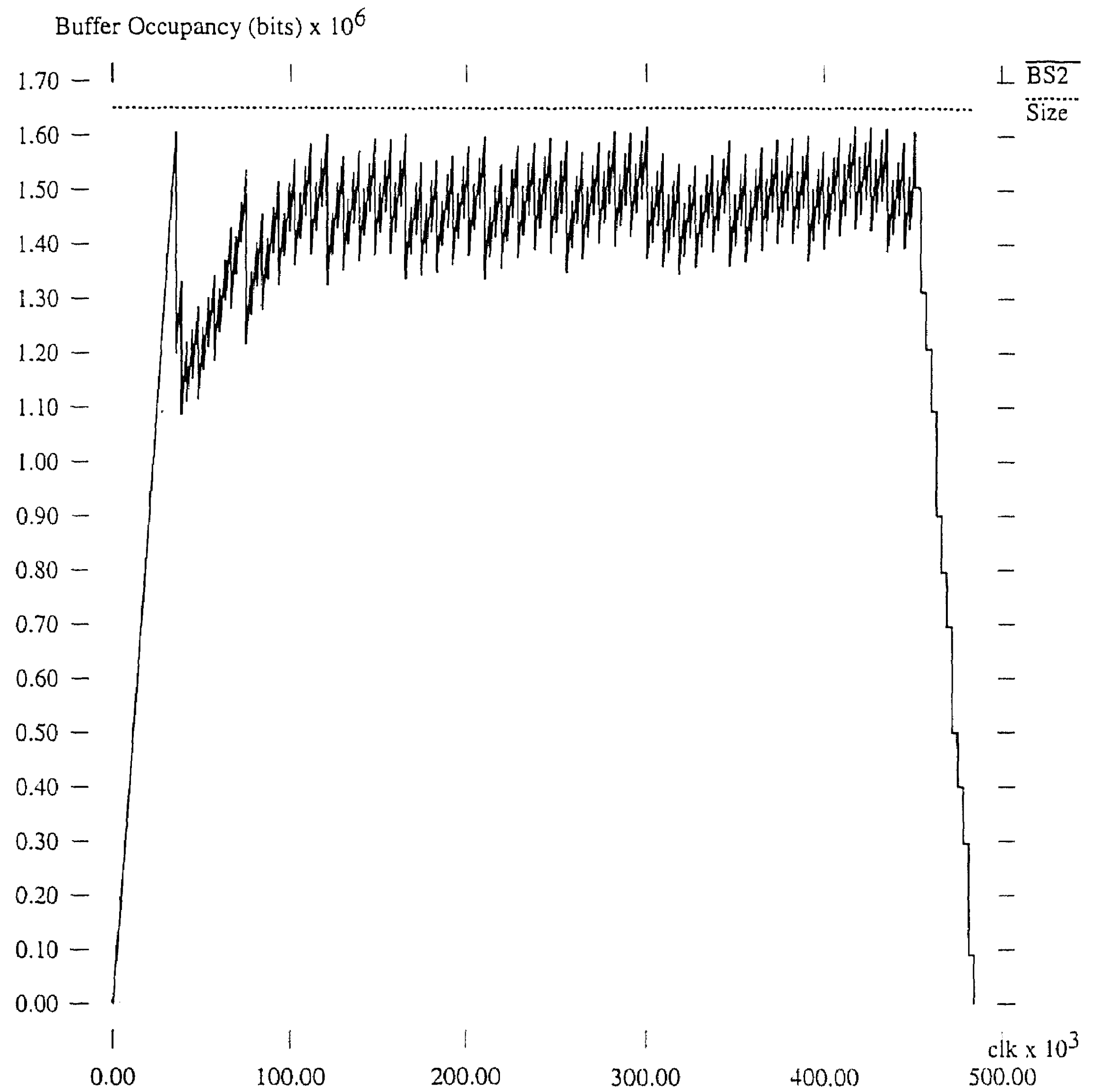


Figure 14 Stream 2 Original VBV Status

Figure 13 and Figure 14 show that no VBV overflow is present. Thus the simulator is assured that both streams are VBV-Compliant. Assume that the simulator is requested to extract Segment 1 from Bitstream 1's picture 0 to 74, and Segment 2 from Bitstream 2's picture 76 to picture 149. Recall from Chapter 3 that the number of zero stuffing bits is determined by the following formula:

$$\begin{aligned} Nzsb &= (Tnext - Treq) * R(p), & \text{if } Tnext \geq Treq \\ &= (Tnext + \Delta t * k - Treq) * R(p), & \text{if } Tnext < Treq \end{aligned}$$

where:

$Tnext = vbv\_delay(p+1) + b(p+1)/R(p)$

$Treq = vbv\_delay(q) + b(q) / R(p)$

Δt = Frame Rate of Segment1

k = a smallest integer that satisfies: $(Tnext + \Delta t * k - Treq) \geq 0$

In this simulation, p=74 and q=76. The next step for the simulator is to purse the bitstreams and determine the following information:

From Bitstream 1:

- vbv_delay(p) = vbv_delay(74) = 28845 clocks
- vbv_delay(p+1) = vbv_delay(75) = 30420 clocks
- t(p) = t(74) = 258342.208 clocks
- t(p+1) = t(75)=261345.208 clocks
- D(p) = D(74) = 63496 bits
- b(p+1) = b(75) = 32 bits

From Bitstream 2:

- b(q) = b(76) = 32 bits
- vbv_delay(q) = vbv_delay(76) = 33886 clocks

Note that time unit is 1/90,000 second, due to the fact that it is the unit of the time information encoded in any MPEG2 bitstream. Thus, each clock represented in the previous page is a period of 1/90,000 second. With this information, the simulator can then compute the essential parameters below.

- T(p) = T(74) = vbv_delay(74) – vbv_delay(75) + t(75) – t(74) = 1428 clocks
- R(p) = R(74) = D(74) / T(74) = (63496 bits) / (1428 clocks) = 44.465 bits/clock

The two T values are easily derived as follows.

- Tnext = vbv_delay(75) + b(75)/R(74) = 30420.72 clocks
- Treq = vbv_delay(76) + b(76)/R(74) = 33886.72 clocks

Since (Tnext – Treq) < 0, the simulator needs to find 'k' that satisfied the inequality:

$$(\text{Tnext} + \Delta t * k - \text{Treq}) \geq 0$$

where:

Δt = frame rate of the streams at 1/29.97 second = 3003 clocks (90kHz).

Through an iterative process, with k starting at zero, the simulator then finds that 2 is the smallest integer 'k' to satisfy the inequality:

$$(Tnext + 2 * \Delta t - Treq) = 2540 > 0$$

Finally the number of stuffing bits is determine as follows:

$$Nzsb = (Tnext + \Delta t * k - Treq) * R(p) = 112941 \text{ bits}$$

Note that the rounding is not performed until the very last stage of finding the exact value of Nzsb. This Nzsb amount of zero stuffing bits is then inserted between the two segments to form the Concatenated Stream. Figure 15 shows the VBV status of the Concatenated Stream.

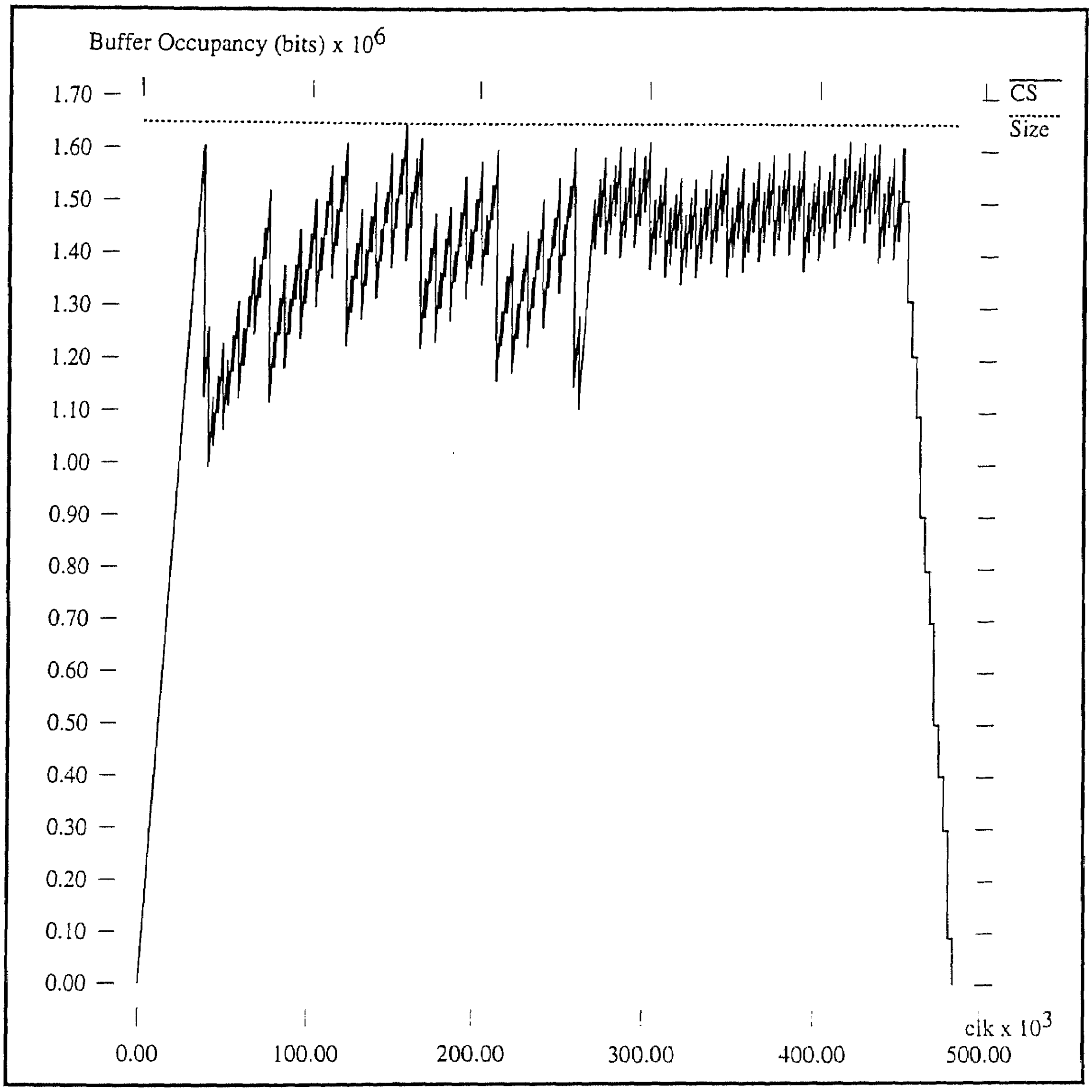


Figure 15 Concatenated Stream's VBV Status

In order to show that this VBV status matches those of Stream 1 and Stream 2, the simulator plots Figure 16. It shows all three VBV status paths on the same graph.

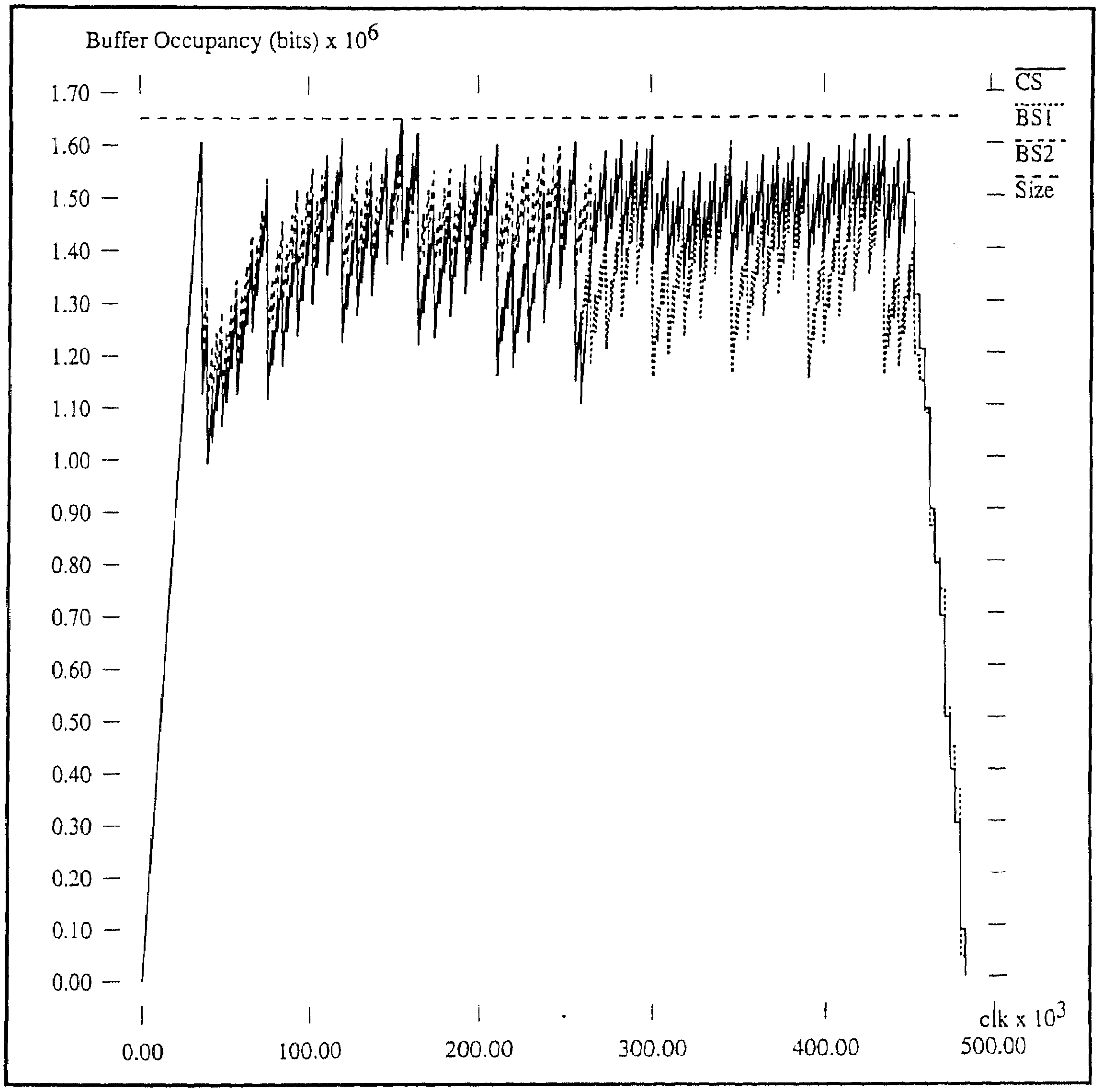


Figure 16 Stream 1, 2, Concatenated Stream VBV Status

In this figure, one should be able to see that the Concatenated Stream's VBV follows that of Stream 1 before the concatenation point, and then matches that of Stream 2 after the concatenation point. To help verifying this path, the Figure 17 on the following page zooms in the concatenation region in Figure 16. Note in Figure 17 that there are two frame-rate periods where picture data is not removed from VBV. This period is created purposely to delay Segment 2's data entry to VBV. This is accomplished by inserting the zero stuffing bits that enter the VBV in advance. As a result, Segment 2 first picture's vbv_delay value starts counting later, which ultimately pushes the time of removing the first picture of Segment 2 late also. These two frame-rate periods are the minimum time period necessary to match Segment 2's VBV with the Concatenated Stream's VBV at the earliest frame-rate point.

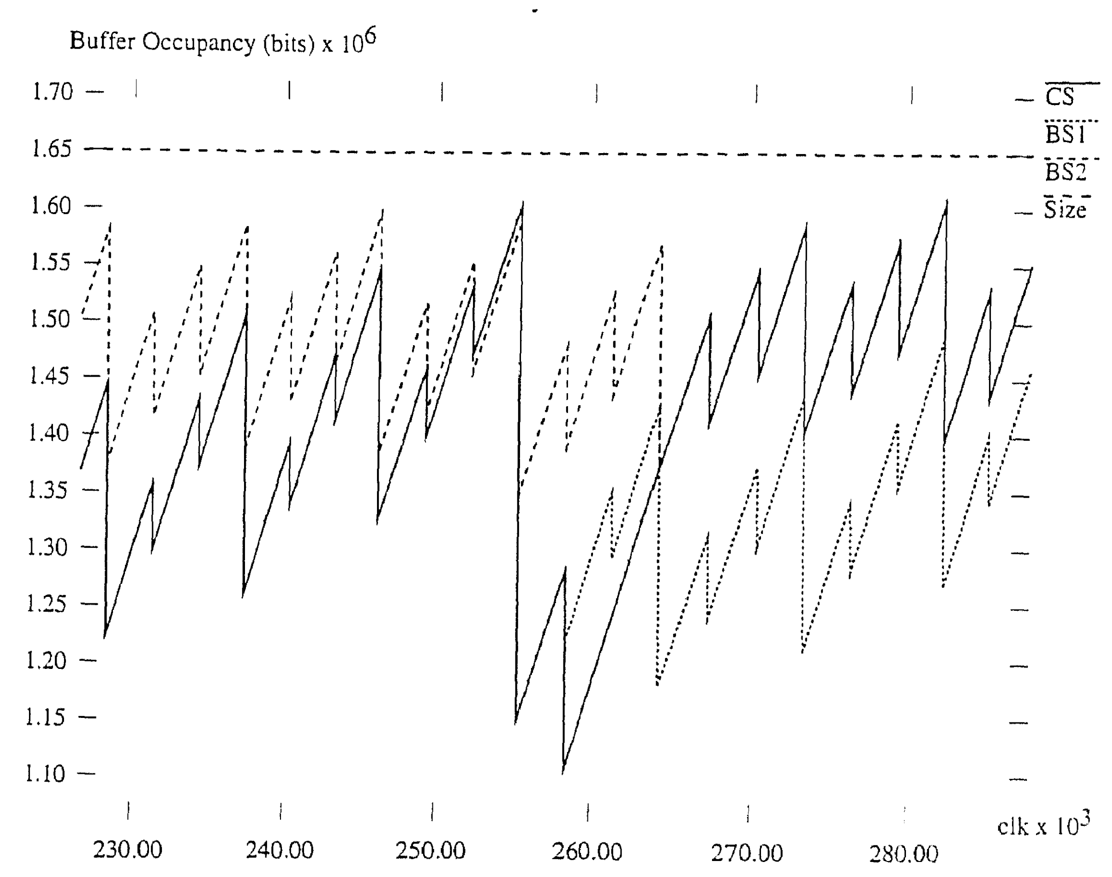


Figure 17 Stream 1, 2, Concatenated Stream VBV Status (zoomed)

For the purpose of demonstrating a failed example, the simulator also performs a concatenation process in which Segment 1 and Segment 2 are simply "hooked" together without any stuffing bit. The Concatenated Stream created by such process will violate VBV compliance, and will mostly likely overflow VBV at one time or another. Figure 18 below depicts the VBV status of this "hooked" stream. VBV overflow regions are clearly visible on this figure.

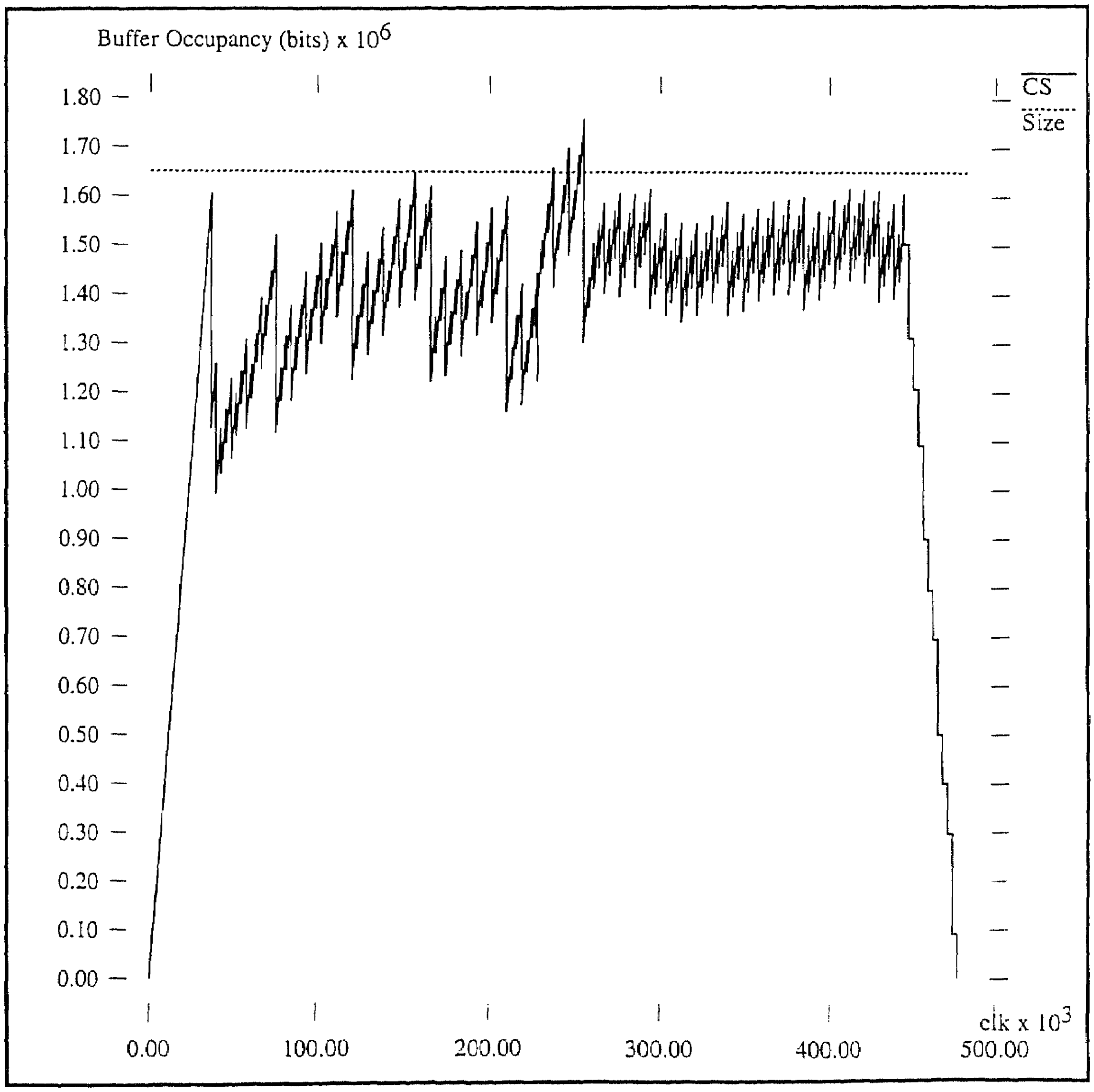


Figure 18 "Hooked" Stream VBV Status

Figure 19 below depicts together all three VBV status paths from Stream 1, Stream 2, and the "hooked" stream. One can see not only that VBV overflow occurs to the "hooked" stream, but also that Stream 2's VBV and the "hooked" stream's VBV take two different paths.

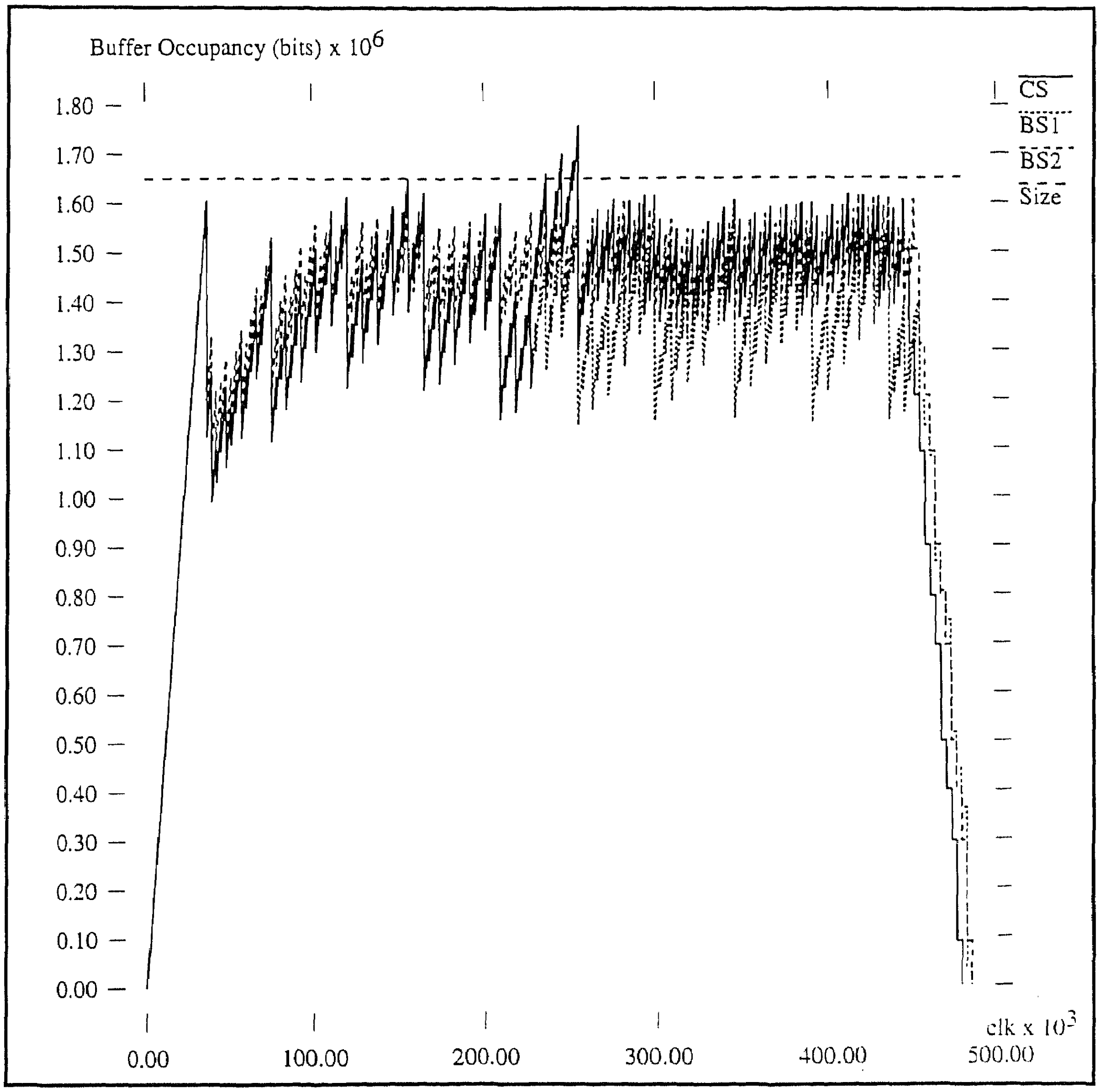


Figure 19 Stream 1, 2, "Hooked" Stream VBV Status

Figure 20 on the next page zooms in the concatenation region in Figure 19. One can see a strange negative or decreasing path of the VBV status for the "hooked" stream. This is due to the fact that the "hooked" stream is VBV-incompliant, and thus the MPEG2 Standard's equation to determine T(q) becomes inapplicable:

$$T(q) = vbv\_delay(p) - vbv\_delay(q) + t(p+1) - t(p) < 0$$

The negative T(q) value is caused by the fact that vbv_delay(p) << vbv_delay(q). This is a typical phenomenon of a VBV-incompliant stream.

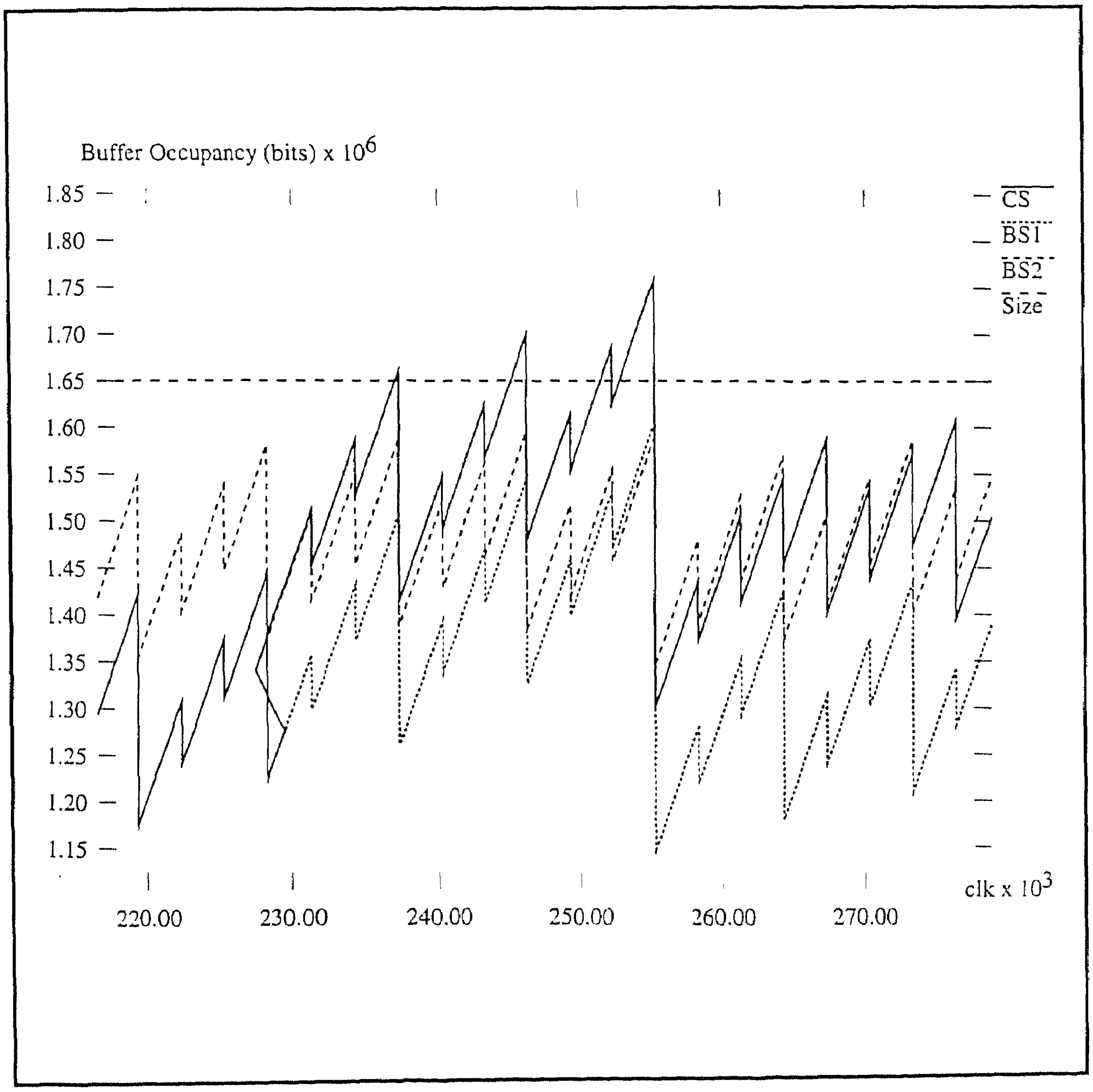


Figure 20 Stream 1, 2, "Hooked" Stream VBV Status (zoomed)

CHAPTER 4

II – SEGMENT EXTRACTION AND TEMPORAL DEPENDENCY

When extracting a segment from a pre-coded bitstream, the major issue to be concerned is the temporal dependency among coded pictures in the bitstream. Since the dependency could be on a past picture and a future picture in display order, the Stream Editor must satisfy the two requirements below.

1. The extracted segment is complete by itself. No picture in the segment is coded together with another picture outside of the segment.
2. The display order of the decoded picture is correct. When a coded picture depends on a picture that is future in time in display order, the depended picture would arrive at the decoder's input earlier in time than the coded picture would. This means that the decoder will have to perform decode/display re-ordering. The Stream Editor must make sure that this re-ordering process would not be altered as a result of the editing.

This chapter first reviews the temporal encoding process used by MPEG2, then shows how temporal dependency and decode/display re-ordering are of concern for a Stream Editor. Finally this chapter proposes a solution to avoid or neutralize the temporal dependency during an editing process.

4.1 Overview of Temporal Dependency in an MPEG2 Bitstream

MPEG2 achieves high degree of compression efficiency by exploiting the similarities between successive pictures, i.e., DPCM. There are three coded picture types that indicate the nature of dependency as shown in Table 2. Figure 13 depicts the temporal dependencies among the different picture types in Table 2.

Table 3 MPEG2 Coded Picture Types

Type Name	Dependency
Intra (I-type)	None
Predicted (P-type)	Depends on the previous I or P Picture in display order
Bi-directionally Predicted (B-type)	Depends on both the previous I (or P) Picture and/or the subsequent I (or P) picture in display order

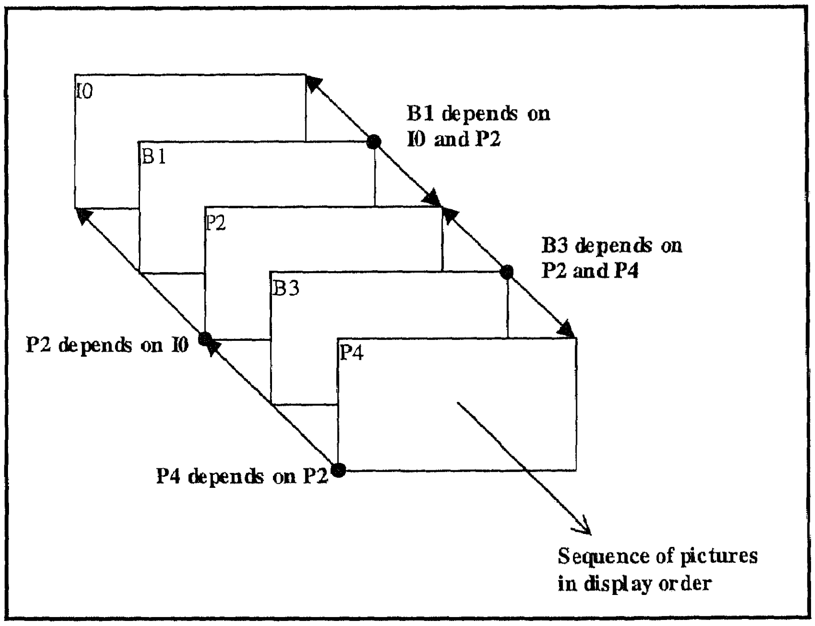


Figure 21 Temporal Dependency Among Coded Pictures

The next example shows how a 5-picture sequence is encoded, transmitted, decoded, and displayed. For convenience, 'S' represents the uncompressed source picture, e.g., a video camera's output. The italic '*S*' represents the decoded picture.

- Camera's output or the Encoder's input picture sequence in time order:

  S0-S1-S2-S3-S4

- Encoder's processing order:

  1. Fetch **<u>S0</u>** -> Compress S0 => **<u>I0</u>**; Decompress I0 => *S0*
  2. Fetch **<u>S2</u>** -> Compute Δ2 = {S2-*S0*} -> Compress Δ2 => **<u>P2</u>**;

     Decompress P2 => *Δ2* -> Compute {*Δ2*-*S0*} => *S2*
  3. Fetch **<u>S1</u>** -> Compute Δ1 = {S1-(*S0*+*S2*)/2} -> Compress Δ1 => **<u>B1</u>**
  4. Fetch **<u>S4</u>** -> Compute Δ4 = (S4-*S2*) -> Compress Δ4 => **<u>P4</u>**;

     Decompress P4 => *Δ4* -> Compute {*Δ4*-*S2*} => *S4*
  5. Fetch **<u>S3</u>** -> Compute Δ3 = {S3-(*S2*+*S4*)/2} -> Compress Δ3 => **<u>B1</u>**.

- Encoder's output / transmission / decoder's input order: I0-P2-B1-P4-B1
- Decoder's processing order:

  1. Receive **<u>I0</u>** -> Decompress I0 => ***<u>S0</u>***
  2. Receive **<u>P2</u>** -> Decompress P2 => *Δ2* -> Compute {*Δ2*+*S0*} => ***<u>S2</u>***
  3. Receive **<u>B1</u>** -> Decompress B1 => *Δ1* -> Compute {*Δ1*+(*S0*+*S2*)/2} => **<u>B1</u>**
  4. Receive **<u>P4</u>** -> Decompress P4 => *Δ4* -> Compute {*Δ4*+*S2*} => ***<u>S4</u>***
  5. Receive **<u>B3</u>** -> Decompress B3 => *Δ3* -> Compute {*Δ3*+(*S2*+*S4*)/2} => **<u>B3</u>**

- Display picture sequence order from an decoder: *S0-S1-S2-S3-S4*

* Note:Process in { } is for illustration of inter-picture relation only. Actual coding involves more complex processes such as motion estimation, DCT, VLD, among others.

4.2 The Challenge to Extract a Segment Properly

The issue relating to the temporal dependency originates in the selection of the first and last picture for a segment from a source bitstream. There are two issues of concern here:

- Display Order
- Reference Picture

Each will be discussed below.

4.2.1 Decode and Display Order Mismatch Problem

Display order issue comes from the fact that the depended (or reference) picture may be transmitted earlier in time than the depending (or predicted) picture. Borrowing the 5-picture sequence example from the previous Figure 13, the next figure shows that P4 is transmitted earlier than B3. However the decoded B3 picture is displayed earlier than the decoded P4 picture. If the Stream Editor selects P4 as the last picture of the segment, then there would be a hole in the display sequence. The next figure shows this operation. The Stream Editor must avoid this kind of display order problem.

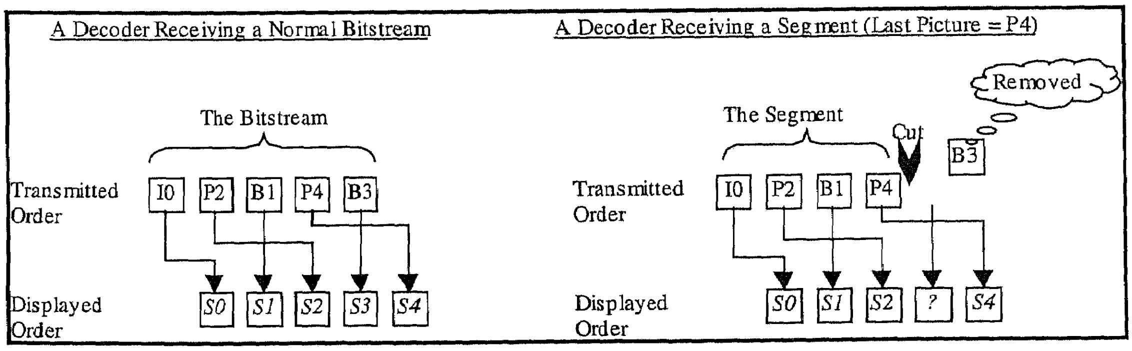


Figure 22 Display Order Affected by Temporal dependency

4.2.2 Cross-Segment Problem

Reference picture issue has to do with maintaining the same reference picture for a depending picture, before and after the editing. The next Figure 15 shows an example that did not maintain the reference picture. Before the editing, Segment2's P6 depended on I5. However after the editing, I5 was no longer available. As a result, P6 wrongly depended on a P4 that belonged to Segment1.

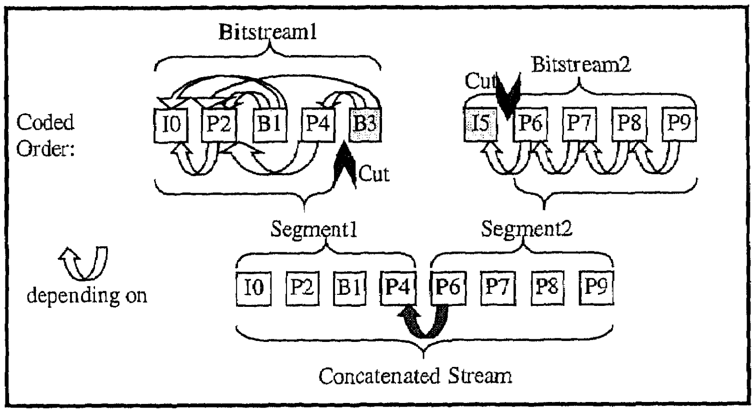


Figure 23 Reference Picture Altered by Concatenation

Note also from Figure 15 that P7, P8, and P9 all depended on P6 as well. Therefore, once P6 was decoded incorrectly, the entire Segment2 becomes invalid.

4.3 Methods to Neutralize Temporal Dependency

This section shows several methods that could be employed to avoid temporal dependency problems discussed in Section 4.2. First, this section introduces a set of Golden Rules that must always be obeyed during a segment extraction process. Then, it discusses techniques to alter the bitstream, only if necessary, to meet the Golden Rules.

4.3.1 Golden Rules

The Stream Editor shall select the boundary pictures based on the following rules:

- Golden Rule 1: The minimum unit of a segment must be B(n).
- Golden Rule 2: The extracted segment's first picture must be an I-type picture.
- Golden Rule 3: The extracted segment's last picture can be any type picture as long as the picture is immediately before an I-type or P-type picture, in coded order, in the source stream.

The next figure shows an example that follows all three rules. Each square represents data in unit of B(n), thus Golden Rule 1 is satisfied. Golden Rule 2 is satisfied, because both segments begin with an I-type picture. Golden Rules 3 is also satisfied, because Segment1's last picture, B1, is immediately before a P-type picture, P4, in Bitstream1. Similarly, Segment2's last picture, P8, is immediately before a I-type picture, I9, in Bitstream 2.

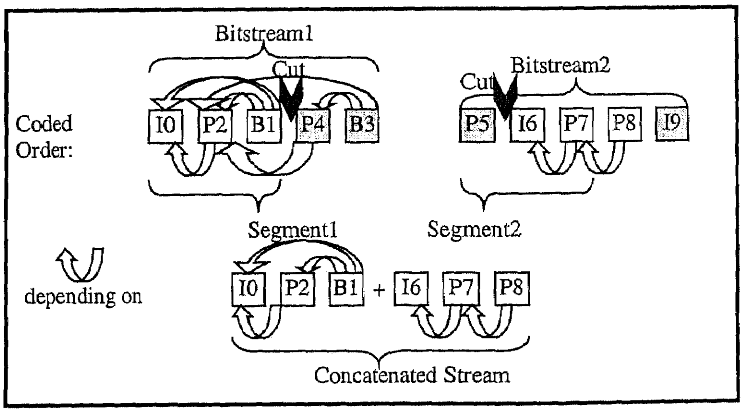


Figure 24 Segment Extraction Based on the Golden Rules

4.3.2 Neutralize B-type Picture Cross-Segment Problem

From Figure 24, we can easily see that if P7 were a B-type picture, then B7 would have depended on I5 as well as P2. The next Figure 17 shows this scenario. This is an example of Cross-Segment problem, where pictures in two completely independent segments became dependent with each other after the editing.

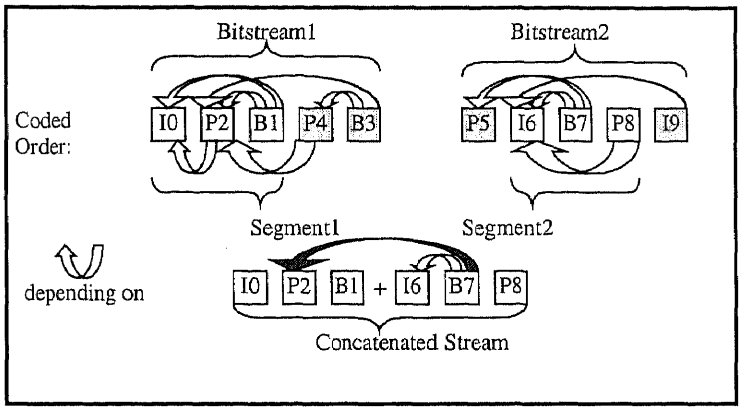


Figure 25 Cross-Segment Problem by B-type Picture

Recently publications [7, 9] have attempted to solve this problem using one of the following picture-type conversion methods:

- Decode B7, and re-encode it as a P-type picture so that it will depend on I6 only [7].
- Decode B7, and re-encode it using backward motion vectors only, effectively creating a special B-type picture which will depend only on I6 [9].

However, the above methods defeat the original objective, that is, performing editing in compressed domain without going back to the uncompressed domain. Also note that decoding B7 actually involves more than just decoding B7 alone; both P5 and I6 must be first decoded before B7 can be decoded.

This thesis suggests an alternate approach that uses an existing syntax tool that is already provided by the MPEG2 Standard. This tool enables a way for the Encoder to warn a Decoder in advance that a B-type picture such as B7 is about to be transmitted. This tool syntax is part of GSC's Coded Data. Therefore this warning will arrive at the Decoder even before the I-type picture (I6). The syntax is *broken_link* and is defined by MPEG2 as follows.

- ***broken_link***: a one-bit flag in GSC's data. When *broken_link* is set to 1, it indicates that the first consecutive B-type pictures (if any) immediately following the first coded I-picture following GSC may not be correctly decoded because the reference picture which these B-type pictures depended on is not available.

The thesis suggests that the Stream Editor simply insert GSC and GSC's Coded Data immediately before the I-type picture (I6), unless such GSC already exists. The method's final step is then simply set *broken_link* to 1. The underlining philosophy to avoid picture-type conversion methods, and instead to use the MPEG2 syntax tool method is two-folds:

1. The benefit of the extra processing power, memory, and delay, for converting a B-type picture to another picture type is slim. Since B-type picture is not depended by any other picture, an incorrectly decoded B-type picture is not going to affect other pictures. Furthermore, the number of consecutive B-type pictures found in a typical stream is very small, which is mostly two. This is because a long sequence of consecutive B-type pictures will severely delay the display of its reference picture,

therefore, the Encoder will seldom create such sequence. This means that even if these incorrectly decoded B-type pictures were actually shown to a viewer, it would be insignificant, and would most likely appear as a short temporal noise.

2. Even if the Stream Editor did perform the picture-type conversion, one of the two consequences is bound to occur:
   - The picture quality of the converted picture is going to be severely degraded. The only reason that an Encoder would use a B-type picture would be that the B-type picture could be coded with far fewer number of bits. In contrast, P- or I-type picture would require more bits. The only way to lower the number of coded bits for a P- or I-type picture is to quantize its DCT coefficients coarsely. This is the reason of the degraded picture quality.
   - Conversely, if the Stream Editor intends to keep the same picture quality, then the number of the coded bits for the converted picture will increase, substantially if the converted picture is of I-type.

For the above reasons, we argue that the better approach is to warn the Decoder, and let the Decoder decide how to handle such kind of B-type picture (B7 in Figure 17).

4.3.3 Special Case - when I-type Picture cannot be Found

From the Golden Rule, we can easily see that the availability of I-type pictures would dictate the availability of choosing Segment2's first picture. This is a problem when the Stream Editor cannot find any I-type picture near the desired editing point. However, note that a typical source stream usually contains an I-type picture within a group of fewer than 15 consecutive pictures. Therefore this problem may be somewhat far-fetched.

On the other hand, in the extreme case, the source stream may contain not a single I-type picture at all, which is in fact legal and allowed by the MPEG2 Standard.

For the sake of handling this extreme case, we suggest that the Stream Editor be equipped with a P-to-I type conversion capability. The workload of converting a P-type picture is far less than converting a B-type picture described in the previous section 4.3.2, yet its advantages are far more convincing:

- Decoding a P-type picture needs only one previously decoded picture at one time.
- Encoding an I-type picture requires substantially smaller workload, because the Encoder does not need to perform temporal prediction and motion estimation.
- P-type pictures are abundant in a typical bitstream. As mentioned above, the MPEG2 Standard allows a bitstream with no I-type picture, but disallows a bitstream consist of only B-type pictures. If an I-type picture were so hard to be found, then many P-type pictures would be easily found in such bitstream.

The next Figure 18 shows an example where P6 is converted to I6.

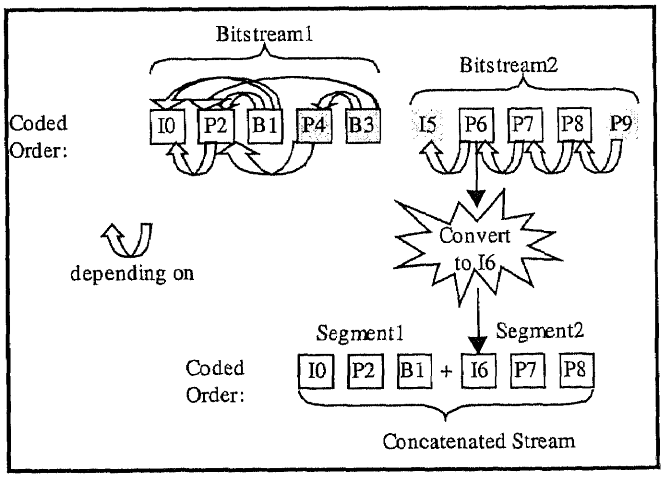


Figure 26 Segment Extraction Problem with P-to-I Conversion

4.4 Simulation Result

This simulation is performed to demonstrate the importance of the Golden Rules explained in Chapter 4. The simulator uses two streams, 'Mobile & Calendar' and 'Football', whose temporal dependency characteristics are shown in the following figure.

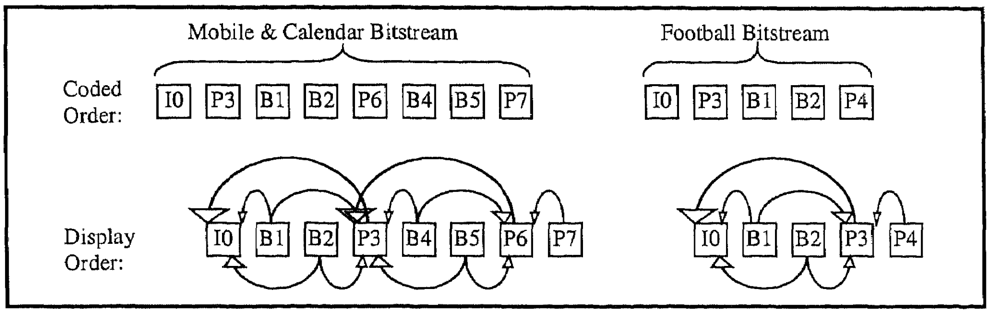


Figure 27 Source Stream's Temporal Dependency Characteristics

The extraction that follows the Golden Rules is depicted in Figure 28 below. Each segment's last picture is a picture before a P-type picture in coded order. And each segment's first picture is an I-type picture.

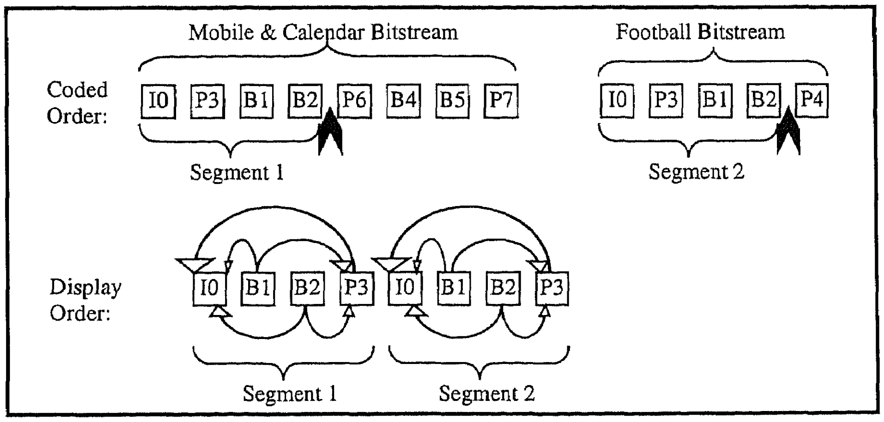


Figure 28 Segment Extraction Based on the Golden Rules

Since the Golden Rules are followed, the Concatenated Stream's display order still maintains the original bitstream's order, and there is no Cross-Segment problem. This can be verified by comparing Figure 27 with Figure 28.

For the purpose of demonstrating the consequences if Golden Rules are not followed, the simulator extracts both segments based on the Figure 29 below.

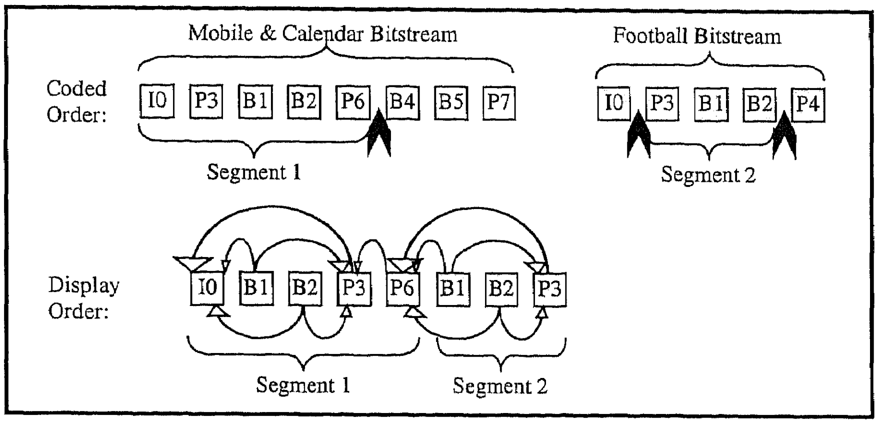


Figure 29 Segment Extraction Against the Golden Rules

This extraction violates the Golden Rules in two ways. First, Segment 1's last picture is not before an I- or P-type picture, but a B-type picture (B4). This violation causes a "jump effect", where P6 is displayed immediately after P3; the original display sequence in Figure 27 shows that there should be two pictures (i.e., B4 and B5) to be displayed before P6 is displayed. The second violation is that Segment 2 does not start with an I-type picture but with a P-type picture (P3). This violation causes "inter-segment" problem, where Segment 2's P3 depends on the wrong picture (P6 of Segment 1) instead of the correct picture (I0 of Segment 2). Consequently all Segment 2's pictures are decoded incorrectly. The actually decoded pictures are all shown in Figure 30. The left-hand sequence is the result from Figure 28, while the right hand sequence is the result from Figure 29. The "jump effect" and the "inter-segment" problem are clearly visible in Figure 30. Finally, Figure 31 enlarges Figure 29's Segment 2's B1 and B2 pictures, so that the severity of the incorrectly decoded pictures can be examined. Notice that no identifiable trace of the football scene exists in either picture.

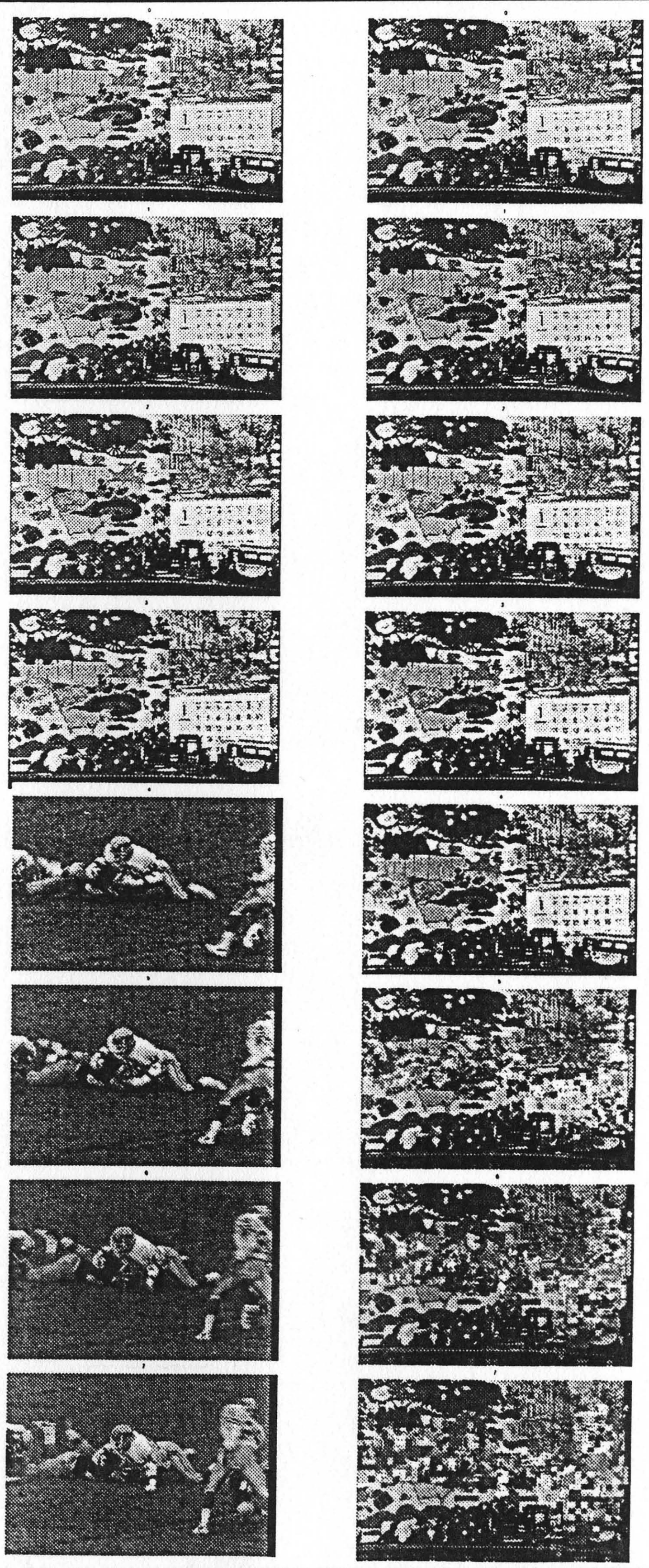
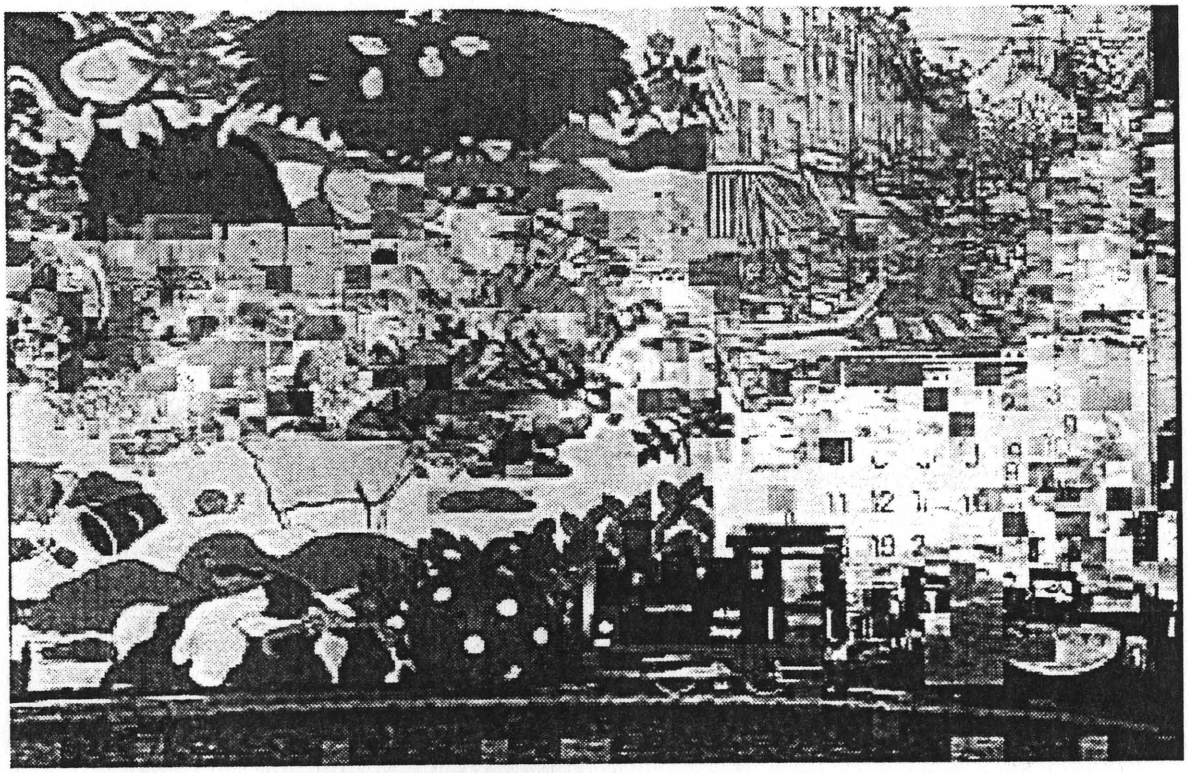


Figure 30 Simulated Pictures based on Golden Rules: For vs. Against

5



6

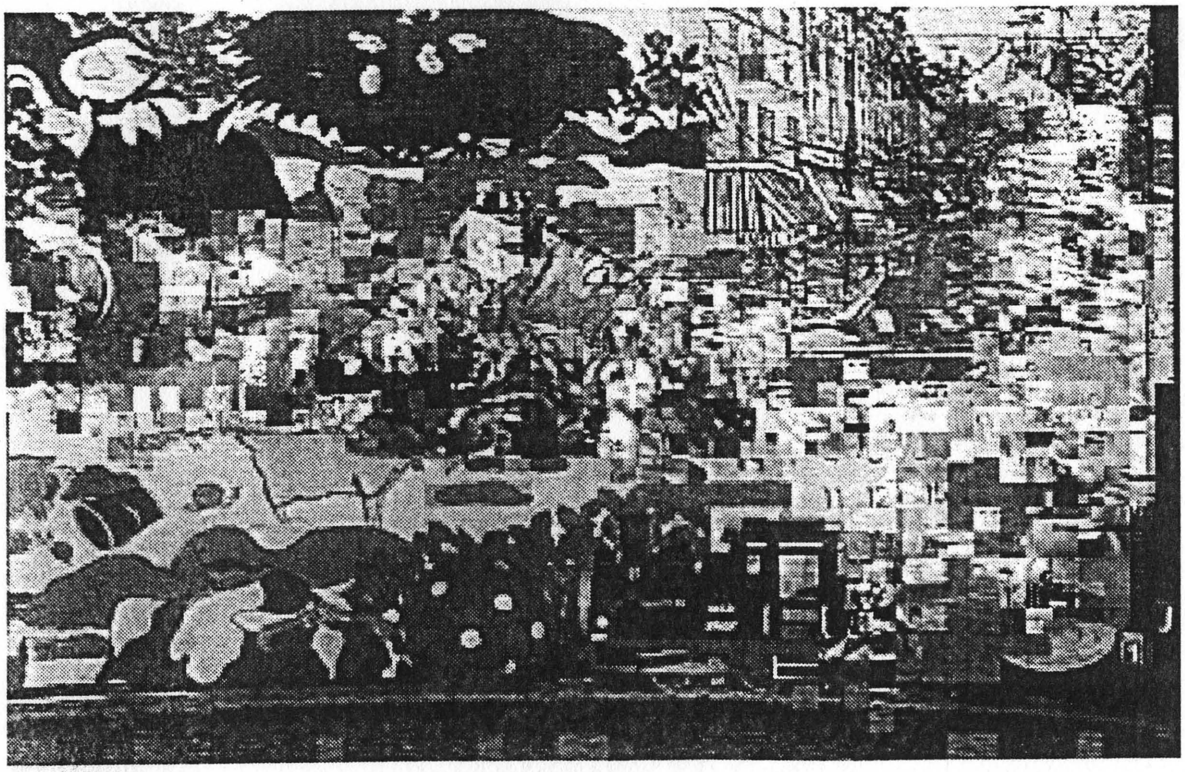


Figure 31 Segment 2's B1 and B2 from Figure 29

CHAPTER 5

CONCLUSION

Two most contentious issues of editing an MPEG2 video bitstream in compressed domain have been VBV compliance and segment extraction. This thesis has analyzed and explained the reasons of the technical difficulties that associated with these two issues, and provided techniques to resolve these issues. The thesis also showed simulation results that verified the techniques.

The significance of this thesis's techniques is that no outside help or constraint is needed to implement the techniques. The only requirement is that the source bitstream should be MPEG2 compliant. This should be obvious, since a Stream Editor cannot "correct" an incompliant stream. However, if the Stream Editor does receive a compliant stream, then this thesis's techniques will enable the Stream Editor to correctly perform the editing job alone, that is, the Stream Editor needs not to depend on some pre-coded information in the source bitstream to correctly perform the editing job.

During an actual editing process, the Stream Editor may first consider how to extract a segment properly from the source stream by referring to the techniques discussed in Chapter 4. Thereafter, the Stream Editor may follow the VBV techniques discussed in Chapter 3, and thereby ensures that the edited stream still meets VBV compliance as if no editing were performed. In a practical application, the two parameters 'k' and 'Δ' which were used in Chapter 3 may be defined differently to meet that particular application's needs. For example, instead of defining 'Δ' as 'frame rate', an interlace-scanning application may wish to re-define 'Δ' as 'field rate'. No matter how these two parameters are defined, all mathematical equations in Chapter 3 will still be valid. Therefore, it is believe that all the techniques discussed in this thesis, along with the flexibility of redefining these two parameters, make this thesis a universal and complete package provided for any Stream Editor to perform any bitstream editing tasks in any application.

REFERENCES

1. ISO/IEC International Standard Committee, *Information Technology – Generic Coding of Moving Pictures and Associated Audio*, ISO/IEC, 1996

2. ISO/IEC International Standard Committee, *Information Technology – Generic Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 M bits/s*, ISO/IEC, 1993

3. ETS 300 744 Standard Committee, *Digital Terrestrial Broadcasting Systems*, ETS, 1997

4. ATSC Digital Television Standard Committee, *Doc. A/53*, ATSC, 1996

5. S. Epstein, "Editing MPEG Bitstreams," *Broadcast Engineering*, Intertec Publishing, Overland Park, Kan., 1997

6. J Meng and S-F Chang, "Tools for Compressed-Domain Video Indexing and Editing," *SPIEC*, Vol.2671, San Jose, CA, 1996.

7. P.J. Brightwell, S.J. Dancer, and M.J. Knee, "Flexible Switching and Editing of MPEG-2 Video Bitstreams," *International Broadcast Convention (IBC97)*, Amsterdam, 1997

8. SMPTE 313M Standard Committee, *Splice Points for MPEG-2 Transport Streams*, SMPTE, 1998

9. J. W. Woods and K. Wang, "MPEG2 Video Editing," Submitted to *IAB Meeting*, 6/3/1999.

10. Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg, and Didier J. LeGall, "Rate Control in MPEG", *MPEG Video Compression Standard*, Chapman & Hall, New York, NY, 1997.