(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4226660号
(P4226660)

(45)発行日　平成21年2月18日(2009.2.18)　(24)登録日　平成20年12月5日(2008.12.5)

(51)Int.Cl.			F I		
H04N	5/92	(2006.01)	H04N	5/92	H
H04N	5/91	(2006.01)	H04N	5/91	N
H04N	7/32	(2006.01)	H04N	7/137	Z

請求項の数 30　(全 47 頁)

(21)出願番号	特願平10-542840
(86)(22)出願日	平成10年3月30日(1998.3.30)
(65)公表番号	特表2001-519118(P2001-519118A)
(43)公表日	平成13年10月16日(2001.10.16)
(86)国際出願番号	PCT/US1998/006246
(87)国際公開番号	WO1998/046023
(87)国際公開日	平成10年10月15日(1998.10.15)
審査請求日	平成15年10月22日(2003.10.22)
(31)優先権主張番号	08/832,987
(32)優先日	平成9年4月4日(1997.4.4)
(33)優先権主張国	米国(US)

(73)特許権者
アヴィッド・テクノロジー・インコーポレーテッド
アメリカ合衆国マサチューセッツ州０１８７６，テュークスバリー，ワン・パーク・ウエスト，メトロポリタン・テクノロジー・パーク
(74)代理人
弁理士　社本　一夫
(74)代理人
弁理士　今井　庄亮
(74)代理人
弁理士　増井　忠弐
(74)代理人
弁理士　栗田　忠彦

最終頁に続く

(54)【発明の名称】インターフレームおよびイントラフレーム技術を用いて圧縮されたモーション・ビデオを捕捉、編集および再生するコンピュータ・システムおよびプロセス

(57)【特許請求の範囲】

【請求項１】

圧縮データのビットストリームにおける時間メディア・データのサンプルに対しランダム・アクセスを可能にするインデックスを作成する方法であって、前記時間メディア・データは、インターフレームおよびイントラフレーム圧縮ビデオ・データを含み、前記時間メディア・データの前記サンプルは、ビットストリーム順序とは異なる時間サンプル順序を有し、前記方法が、

各サンプルに関し前記インデックス内にエントリを作成するステップであって、前記インデックス内の前記エントリは、前記ビットストリーム内の前記サンプルに関し、前記圧縮データの前記順序で順序付けされ、各エントリは前記インデックス内においてエントリ位置を有する、ステップと、

各エントリに関し、あるサンプルに対する圧縮データに対するバイト・オフセットを前記ビットストリームに格納するステップであって、前記エントリは、前記エントリの前記エントリ位置に対応する前記ビットストリーム順序内の位置を有する前記サンプルに対する前記バイト・オフセットを格納する、ステップと、

各エントリに関し、前記時間サンプル順序におけるあるサンプルの位置と、前記ビットストリーム順序における前記サンプルの位置との間の時間オフセットを格納するステップであって、前記エントリは、前記エントリの前記エントリ位置に対応する前記時間サンプル順序内の位置を有する前記サンプルに対する前記時間オフセットを格納する、ステップと、

を含む方法。
【請求項２】
請求項１記載の方法であって、さらに、
各エントリに関し、別のエントリに対するランダムアクセス・オフセットを格納し、前記別のエントリは、前記サンプルへの前記バイト・オフセットを格納し、そのサンプルから、前記エントリに対応する前記サンプルを再構成するため前記ビットストリームの圧縮解除を開始する、ステップ、
を含む方法。
【請求項３】
請求項１記載の方法であって、さらに、
各エントリに関し、前記ビットストリームの圧縮解除を開始するためのランダムアクセスが前記エントリに対応する前記サンプルを使用して可能かどうかの指示を格納するステップ、
を含む方法。
【請求項４】
請求項１記載の方法であって、さらに、
各エントリに関し、前記エントリに対応する前記サンプルのタイプの指示を格納するステップ、
を含む、方法。
【請求項５】
請求項１記載の方法であって、さらに、
各エントリに関し、前記エントリに対応する前記サンプルに関しシーケンス・ヘッダ・ビットの指示を格納するステップ、
を含む、方法。
【請求項６】
請求項１記載の方法において、さらに、
前記ビットストリームを処理して、復号および表示のため使用する状態情報を識別するステップと、
各々のイントラフレーム圧縮されたイメージおよび各々のインターフレーム圧縮されたイメージに対する前記ビットストリームに、前記状態情報を挿入することにより、任意のイントラフレーム圧縮イメージに対するランダム・アクセスを可能にするステップと、
を含む方法。
【請求項７】
圧縮データのビットストリームにおける時間メディア・データのサンプルに対しランダム・アクセスを可能にするインデックスを作成する装置であって、前記時間メディア・データは、インターフレームおよびイントラフレーム圧縮ビデオ・データを含み、前記時間メディア・データの前記サンプルは、ビットストリーム順序とは異なる時間サンプル順序を有し、前記装置が、
各サンプルに関し前記インデックス内にエントリを作成する手段であって、前記インデックス内の前記エントリは、前記ビットストリーム内の前記サンプルに関し、前記圧縮データの前記順序で順序付けされ、各エントリは前記インデックス内においてエントリ位置を有する、手段と、
各エントリに関し、あるサンプルに対する圧縮データに対するバイト・オフセットを判定して前記ビットストリームに格納する手段であって、前記エントリは、前記エントリの前記エントリ位置に対応する前記ビットストリーム順序内の位置を有する前記サンプルに対する前記バイト・オフセットを格納する、手段と、
各エントリに関し、前記時間サンプル順序におけるあるサンプルの位置と、前記ビットストリーム順序における前記サンプルの位置との間の時間オフセットを判定し格納する手段であって、前記エントリは、前記エントリの前記エントリ位置に対応する前記時間サンプル順序内の位置を有する前記サンプルに対する前記時間オフセットを格納する、手段と、

を含む装置。
【請求項８】
請求項７記載の装置であって、さらに、
各エントリに関し、別のエントリに対するランダムアクセス・オフセットを判定し格納する手段であって、前記別のエントリは、前記サンプルへの前記バイト・オフセットを格納し、そのサンプルから、前記エントリに対応する前記サンプルを再構成するため前記ビットストリームの圧縮解除を開始する、手段、
を含む装置。
【請求項９】
請求項７記載の装置であって、さらに、
各エントリに関し、前記ビットストリームの圧縮解除を開始するためのランダムアクセスが前記エントリに対応する前記サンプルを使用して可能かどうかの指示を判定し格納する手段、
を含む装置。
【請求項１０】
請求項７記載の装置であって、さらに、
各エントリに関し、前記エントリに対応する前記サンプルのタイプの指示を判定し格納する手段、
を含む、装置。
【請求項１１】
請求項７記載の装置であって、さらに、
各エントリに関し、前記エントリに対応する前記サンプルに関しシーケンス・ヘッダ・ビットの指示を判定し格納する手段、
を含む、装置。
【請求項１２】
請求項７記載の装置において、さらに、
前記ビットストリームを処理して、復号および表示のため使用する状態情報を識別する手段と、
各々のイントラフレーム圧縮されたイメージおよび各々のインターフレーム圧縮されたイメージに対する前記ビットストリームに、前記状態情報を挿入する手段であって、これにより、任意のイントラフレーム圧縮イメージに対するランダム・アクセスを可能にする、手段と、
を含む装置。
【請求項１３】
コンピュータ・プログラム命令を格納したコンピュータ読み取り可能記録媒体であって、
前記コンピュータ・プログラム命令は、プロセッサにより実行されたとき、圧縮データのビットストリームにおける時間メディア・データのサンプルに対しランダム・アクセスを可能にするインデックスを作成する方法を前記プロセッサに実行させ、前記時間メディア・データは、インターフレームおよびイントラフレーム圧縮ビデオ・データを含み、前記時間メディア・データの前記サンプルは、ビットストリーム順序とは異なる時間サンプル順序を有し、前記方法が、
各サンプルに関し前記インデックス内にエントリを作成するステップであって、前記インデックス内の前記エントリは、前記ビットストリーム内の前記サンプルに関し、前記圧縮データの前記順序で順序付けされ、各エントリは前記インデックス内においてエントリ位置を有する、ステップと、
各エントリに関し、あるサンプルに対する圧縮データに対するバイト・オフセットを前記ビットストリームに格納するステップであって、前記エントリは、前記エントリの前記エントリ位置に対応する前記ビットストリーム順序内の位置を有する前記サンプルに対する前記バイト・オフセットを格納する、ステップと、
各エントリに関し、前記時間サンプル順序におけるあるサンプルの位置と、前記ビットス

トリーム順序における前記サンプルの位置との間の時間オフセットを格納するステップであって、前記エントリは、前記エントリの前記エントリ位置に対応する前記時間サンプル順序内の位置を有する前記サンプルに対する前記時間オフセットを格納する、ステップと、
を含む、コンピュータ読み取り可能記録媒体。
【請求項１４】
請求項１３記載のコンピュータ読み取り可能記録媒体であって、前記方法がさらに、
各エントリに関し、別のエントリに対するランダムアクセス・オフセットを格納し、前記別のエントリは、前記サンプルへの前記バイト・オフセットを格納し、そのサンプルから、前記エントリに対応する前記サンプルを再構成するため前記ビットストリームの圧縮解除を開始する、ステップ、
を含むコンピュータ読み取り可能記録媒体。
【請求項１５】
請求項１３記載のコンピュータ読み取り可能記録媒体であって、前記方法がさらに、
各エントリに関し、前記ビットストリームの圧縮解除を開始するためのランダムアクセスが前記エントリに対応する前記サンプルを使用して可能かどうかの指示を格納するステップ、
を含むコンピュータ読み取り可能記録媒体。
【請求項１６】
請求項１３記載のコンピュータ読み取り可能記録媒体であって、前記方法がさらに、
各エントリに関し、前記エントリに対応する前記サンプルのタイプの指示を格納するステップ、
を含む、コンピュータ読み取り可能記録媒体。
【請求項１７】
請求項１３記載のコンピュータ読み取り可能記録媒体であって、前記方法がさらに、
各エントリに関し、前記エントリに対応する前記サンプルに関しシーケンス・ヘッダ・ビットの指示を格納するステップ、
を含む、コンピュータ読み取り可能記録媒体。
【請求項１８】
請求項１３記載のコンピュータ読み取り可能記録媒体において、前記方法がさらに、
前記ビットストリームを処理して、復号および表示のため使用する状態情報を識別するステップと、
各々のイントラフレーム圧縮されたイメージおよび各々のインターフレーム圧縮されたイメージに対する前記ビットストリームに、前記状態情報を挿入することにより、任意のイントラフレーム圧縮イメージに対するランダム・アクセスを可能にするステップと、
を含むコンピュータ読み取り可能記録媒体。
【請求項１９】
圧縮データのビットストリームにおける時間メディア・データのサンプルに対しランダムにアクセスするためインデックスを使用する方法であって、前記時間メディア・データは、インターフレームおよびイントラフレーム圧縮ビデオ・データを含み、前記時間メディア・データの前記サンプルは、ビットストリーム順序とは異なる時間サンプル順序を有し、前記方法が、
各サンプルに関しエントリのインデックスにアクセスするステップであって、前記インデックス内の前記エントリは、前記ビットストリーム内の前記サンプルに関し、前記圧縮データの前記順序で順序付けされ、各エントリは前記インデックス内においてエントリ位置を有し、各エントリは、あるサンプルに対する圧縮データに対するバイト・オフセットを前記ビットストリームに格納し、前記エントリは、前記エントリの前記エントリ位置に対応する前記ビットストリーム順序内の位置を有する前記サンプルに対する前記バイト・オフセットを格納し、かつ前記エントリは、前記時間サンプル順序におけるあるサンプルの位置と、前記ビットストリーム順序における前記サンプルの位置との間の時間オフセット

を格納し、前記エントリは、前記エントリの前記エントリ位置に対応する前記時間サンプル順序内の位置を有する前記サンプルに対する前記時間オフセットを格納する、ステップと、
前記時間サンプル順序におけるあるサンプルの位置の指示を使用して、前記サンプルの位置に対応するエントリ位置を有する前記インデックス内のエントリにアクセスし、前記サンプルに関する前記時間オフセットを検索するステップと、
前記時間サンプル順序における前記サンプルの前記位置に対し前記時間オフセットを加えて、前記ビットストリーム順序における前記サンプルの位置を得るステップと、
前記ビットストリーム順序内の前記サンプルの前記位置に対応するエントリ位置を有する前記インデックス内の前記エントリにアクセスして、前記サンプルに関する前記バイト・オフセットを検索するステップと、
を含む方法。
【請求項２０】
請求項１９記載の方法において、各エントリは、さらに、別のエントリに対するランダムアクセス・オフセットを格納し、前記別のエントリは、前記サンプルへの前記バイト・オフセットを格納し、そのサンプルから、前記エントリに対応する前記サンプルを再構成するため前記ビットストリームの圧縮解除を開始し、前記サンプルに関する前記バイト・オフセットを検索するため前記エントリにアクセスすることは、さらに前記サンプルに関する前記ランダムアクセス・オフセットを検索するために実行する、方法。
【請求項２１】
請求項１９記載の方法であって、さらに、
前記ビットストリーム順序内の前記サンプルの位置に対応するエントリ位置を有する前記エントリより前の、前記インデックスの各エントリを走査して、前記ビットストリームへのランダムアクセスを開始できる別のサンプルに関するエントリを識別するステップ、
を含む、方法。
【請求項２２】
請求項１９記載の方法において、前記時間サンプル順序内のあるサンプルの位置の前記指示を、グラフィカル・ユーザ・インターフェースから受け、該グラフィカル・ユーザ・インターフェースは、前記時間メディア・データを使用する合成物に関係した時間ライン表示において位置バーを含む、方法。
【請求項２３】
圧縮データのビットストリームにおける時間メディア・データのサンプルに対しランダムにアクセスするためインデックスを使用する装置であって、前記時間メディア・データは、インターフレームおよびイントラフレーム圧縮ビデオ・データを含み、前記時間メディア・データの前記サンプルは、ビットストリーム順序とは異なる時間サンプル順序を有し、前記装置が、
各サンプルに関しエントリのインデックスにアクセスする手段であって、前記インデックス内の前記エントリは、前記ビットストリーム内の前記サンプルに関し、前記圧縮データの前記順序で順序付けされ、各エントリは前記インデックス内においてエントリ位置を有し、各エントリは、あるサンプルに対する圧縮データに対するバイト・オフセットを前記ビットストリームに格納し、前記エントリは、前記エントリの前記エントリ位置に対応する前記ビットストリーム順序内の位置を有する前記サンプルに対する前記バイト・オフセットを格納し、かつ前記エントリは、前記時間サンプル順序におけるあるサンプルの位置と、前記ビットストリーム順序における前記サンプルの位置との間の時間オフセットを格納し、前記エントリは、前記エントリの前記エントリ位置に対応する前記時間サンプル順序内の位置を有する前記サンプルに対する前記時間オフセットを格納する、手段と、
前記時間サンプル順序におけるあるサンプルの位置の指示を使用して、前記サンプルの位置に対応するエントリ位置を有する前記インデックス内のエントリにアクセスし、前記サンプルに関する前記時間オフセットを検索する手段と、
前記時間サンプル順序における前記サンプルの前記位置に対し前記時間オフセットを加え

て、前記ビットストリーム順序における前記サンプルの位置を得る手段と、
前記ビットストリーム順序内の前記サンプルの前記位置に対応するエントリ位置を有する前記インデックス内の前記エントリにアクセスして、前記サンプルに関する前記バイト・オフセットを検索する手段と、
を含む装置。
【請求項２４】
請求項２３記載の装置において、各エントリは、さらに、別のエントリに対するランダムアクセス・オフセットを格納し、前記別のエントリは、前記サンプルへの前記バイト・オフセットを格納し、そのサンプルから、前記エントリに対応する前記サンプルを再構成するため前記ビットストリームの圧縮解除を開始し、前記サンプルに関する前記バイト・オフセットを検索するため前記エントリにアクセスする前記手段は、さらに前記サンプルに関する前記ランダムアクセス・オフセットを検索する手段を含む、装置。
【請求項２５】
請求項２３記載の装置であって、さらに、
前記ビットストリーム順序内の前記サンプルの位置に対応するエントリ位置を有する前記エントリより前の、前記インデックスの各エントリを走査して、前記ビットストリームへのランダムアクセスを開始できる別のサンプルに関するエントリを識別する手段、
を含む、装置。
【請求項２６】
請求項２３記載の装置において、さらに、前記時間サンプル順序内のあるサンプルの位置の前記指示を、グラフィカル・ユーザ・インターフェースから受ける手段を含み、前記グラフィカル・ユーザ・インターフェースは、前記時間メディア・データを使用する合成物に関係した時間ライン表示において位置バーを含む、装置。
【請求項２７】
コンピュータ・プログラム命令を格納したコンピュータ読み取り可能記録媒体であって、
前記コンピュータ・プログラム命令は、プロセッサにより実行されたとき、圧縮データのビットストリームにおける時間メディア・データのサンプルに対しランダムにアクセスするためインデックスを使用する方法を前記プロセッサに実行させ、前記時間メディア・データは、インターフレームおよびイントラフレーム圧縮ビデオ・データを含み、前記時間メディア・データの前記サンプルは、ビットストリーム順序とは異なる時間サンプル順序を有し、前記方法が、
各サンプルに関しエントリのインデックスにアクセスするステップであって、前記インデックス内の前記エントリは、前記ビットストリーム内の前記サンプルに関し、前記圧縮データの前記順序で順序付けされ、各エントリは前記インデックス内においてエントリ位置を有し、各エントリは、あるサンプルに対する圧縮データに対するバイト・オフセットを前記ビットストリームに格納し、前記エントリは、前記エントリの前記エントリ位置に対応する前記ビットストリーム順序内の位置を有する前記サンプルに対する前記バイト・オフセットを格納し、かつ前記エントリは、前記時間サンプル順序におけるあるサンプルの位置と、前記ビットストリーム順序における前記サンプルの位置との間の時間オフセットを格納し、前記エントリは、前記エントリの前記エントリ位置に対応する前記時間サンプル順序内の位置を有する前記サンプルに対する前記時間オフセットを格納する、ステップと、
前記時間サンプル順序におけるあるサンプルの位置の指示を使用して、前記サンプルの位置に対応するエントリ位置を有する前記インデックス内のエントリにアクセスし、前記サンプルに関する前記時間オフセットを検索するステップと、
前記時間サンプル順序における前記サンプルの前記位置に対し前記時間オフセットを加えて、前記ビットストリーム順序のおける前記サンプルの位置を得るステップと、
前記ビットストリーム順序内の前記サンプルの前記位置に対応するエントリ位置を有する前記インデックス内のエントリにアクセスして、前記サンプルに関するバイト・オフセットを検索するステップと、

を含む、コンピュータ読み取り可能記録媒体。
【請求項２８】
請求項２７記載のコンピュータ読み取り可能記録媒体において、各エントリは、さらに、別のエントリに対するランダムアクセス・オフセットを格納し、前記別のエントリは、前記サンプルへの前記バイト・オフセットを格納し、そのサンプルから、前記エントリに対応する前記サンプルを再構成するため前記ビットストリームの圧縮解除を開始し、前記サンプルに関する前記バイト・オフセットを検索するため前記エントリにアクセスすることは、さらに前記サンプルに関する前記ランダムアクセス・オフセットを検索するために実行する、コンピュータ読み取り可能記録媒体。
【請求項２９】
請求項２７記載のコンピュータ読み取り可能記録媒体であって、前記方法がさらに、
前記ビットストリーム順序内の前記サンプルの位置に対応するエントリ位置を有する前記エントリより前の、前記インデックスの各エントリを走査して、前記ビットストリームへのランダムアクセスを開始できる別のサンプルに関するエントリを識別するステップ、
を含む、コンピュータ読み取り可能記録媒体。
【請求項３０】
請求項２７記載のコンピュータ読み取り可能記録媒体において、前記時間サンプル順序内のあるサンプルの位置の前記指示を、グラフィカル・ユーザ・インターフェースから受け、該グラフィカル・ユーザ・インターフェースは、前記時間メディア・データを使用する合成物に関係した時間ライン表示において位置バーを含む、コンピュータ読み取り可能記録媒体。
【発明の詳細な説明】
発明の分野
本発明は、ディジタル形式のモーション・ビデオおよび関連したオーディオの捕捉、編集および再生であって、モーション・ビデオ・データがインターフレームおよびイントラフレーム技術を使用して圧縮されたものに関している。
発明の背景
モーション・ビデオおよび関連したオーディオの捕捉、編集および再生のため、幾つかのシステムが現在入手可能である。そのようなシステムの特定の範疇には、ディジタル非線形ビデオ・エディタが含まれている。そのようなシステムは、ランダム・アクセス・コンピュータ読取り可能媒体上のコンピュータ・データ・ファイルに、ディジタル静止イメージのシーケンスを表すモーション・ビデオ・データを、ディジタル・データとして記憶する。静止イメージは、モーション・ビデオ・データの単一のフレーム、すなわち、２個のフィールド、または単一のフィールドを表すことができる。そのようなシステムは、一般的に、静止イメージのシーケンスの中の任意の特定のイメージに対し、編集および再生のためランダムにアクセスするのを可能にする。ディジタル非線形ビデオ・エディタは、ビデオ情報に対して線形的なアクセスのみを提供する先のビデオ・テープベースのシステムを超えた幾つかの利点を有している。
モーション・ビデオを表すディジタル・データは、特にフル・モーション放送品質ビデオ（例えば、ＮＴＳＣにとって６０フィールド／秒およびＰＡＬにとって５０フィールド／秒）に対して、大量のコンピュータ・メモリを消費することがあるので、ディジタル・データは、通常、記憶要求を低減するため圧縮される。モーション・ビデオ情報に対しては、幾つかの種類の圧縮がある。１つの種類の圧縮は「イントラフレーム（intraframe）」圧縮と呼ばれ、それは、各静止イメージを表すデータを、他の静止イメージとは独立に圧縮することが含まれる。一般に用いられているイントラフレーム圧縮技術は、空間領域から周波数領域への変換を採用し、例えば離散コサイン変換を使用することにより行う。その結果生じた値は、通常、量子化されかつ符号化される。イントラフレーム圧縮を用いた一般に使用のモーション・ビデオ圧縮スキームには、「モーションＪＰＥＧ」および「Ｉフレームのみ」ＭＰＥＧ（“I-frame only”ＭＰＥＧ）が含まれる。イントラフレーム圧縮は特定のイメージ内のデータの冗長性を低減する一方、それはモーション・ビデオ・シ

ーケンス中の隣接イメージ間のデータの著しい冗長性を低減するものではない。しかしながら、イントラフレーム圧縮されたイメージ・シーケンスに関して、そのシーケンス中の各イメージは、個々にアクセスでき、かつ他のイメージを参照することなく圧縮解除することができる。したがって、イントラフレーム圧縮は、シーケンス中の任意のイメージに対しても純粋に非線形にアクセスすることを可能にする。

一般に、「インターフレーム」圧縮と呼ばれるものを使用することにより、更なる圧縮をモーション・ビデオ・シーケンスに対して得ることができる。インターフレーム圧縮には、１つのイメージを使用して別のイメージを予測することが含まれている。この種の圧縮は、イントラフレーム圧縮と組み合わせて使用することがよくある。例えば、第１のイメージは、イントラフレーム圧縮を使用して圧縮でき、これは通常、キー・フレームと呼ぶ。後続のイメージは、他のイメージ・データと組み合わされたとき所望のイメージをもたらすような予測情報を発生することにより、圧縮できる。イントラフレーム圧縮イメージは、シーケンス全体にわたり非常によく生じることがある。幾つかの標準は、ＭＰＥＧ－１（ISO/IEC 11172-1～5）、ＭＰＥＧ－２（ISO/IEC 13818-1～9）およびＨ．２６１、国際電気通信連合（ＩＴＵ）標準のようなインターフレーム圧縮技術を使用している。ＭＰＥＧ－２は、例えば、イントラフレーム圧縮を使用して一部のイメージを圧縮し（Ｉフレームまたはキー・フレームと呼ぶ）、そして、インターフレーム圧縮技術を使用して、例えばイメージ間の予測エラーを計算することにより他のイメージを圧縮する。予測エラーは、前方予測（Ｐフレームと呼ばれる）または双方向予測（Ｂフレームと呼ばれる）に対し計算することができる。ＭＰＥＧ－２は、放送品質のフル・モーション・ビデオを提供するよう設計されている。

インターフレーム圧縮されたイメージ・シーケンスに関して、そのシーケンス中のインターフレーム圧縮イメージは、シーケンス中の他のイメージを参照することだけでアクセスし圧縮解除することができる。したがって、イメージがシーケンス中の先のイメージまたは次のイメージのいずれかに依存するので、インターフレーム圧縮は、シーケンス中のどのイメージに対しても、純粋に非線形なアクセスを可能としない。概して言えば、シーケンス中のイントラフレーム・イメージのみが非線形にアクセスできる。しかしながら、ＭＰＥＧ－２のような一部の圧縮フォーマットにおいて、量子化テーブルのような、イントラフレーム圧縮イメージを復号または表示するため必要なある種の状態情報はまた、圧縮されたビットストリーム中のどこか他のところで生じる場合があり、これは、イントラフレーム圧縮イメージにさえ非線形にアクセスする能力を排除する。

任意のシーケンス中の直列従属セグメントの再生を処理する１つのアプローチは、米国特許4,729,044（Keisel）に開示されている。このシステムにおいて、セグメント中のイメージ間の従属性は、記憶媒体、すなわちビデオ・テープの線形性質に起因している。同じ素材（material）を含む５～６本のテープを用いる。再生すべき任意の所与のセグメントに関して、素材をアクセスすべきテープのうちの１つを選択するためアルゴリズムを用いる。同時に、後続のセグメントに対するテープを識別し、次のセグメントのスタートに対してキューを与える。その結果、幾つかの同一のソースは、最終プログラムを生成するために平行して処理する。

非線形システムにおいては、任意のシーケンスのセグメントを生成するためのビデオ・ソースの多重コピーの必要性は、媒体のランダム・アクセスの性質により回避されている。多重データ・ファイルからの任意のシーケンスのセグメントは、モーション・ビデオ・データに対する非線形アクセスをパイプライン化しかつバッファリングすることにより与える。すなわち、米国特許5,045,940（Peters他）に示されるように、あるデータが圧縮解除され再生されている間、他のデータはデータ・ファイルから検索されている。

そのようなシステムにおいては、ビデオ・セグメントは、２つのセグメント間のディゾルブおよびフェードのようなある一定の特別の効果を生成するため並列に処理する必要が依然としてある。そのような効果を実行する１つのシステムは、ＰＣＴ公開ＷＯ94/24815（Kurtze他）に記載されている。このシステムにおいては、２つのビデオ・ストリームを関数αＡ＋（１－α）Ｂで混合する。ここで、ＡおよびＢは、２つのビデオ・ストリームの

対応するイメージ中の対応するピクセルである。このシステムの通常の使用は、セグメントＡを再生し、そして数個のイメージにわたってセグメントＢへの遷移を起こすようにすることである。セグメントＢにとって必要なデータは、バッファにロードされかつ圧縮解除される一方、Ａは再生されつつあり、そのためセグメントＢに対する復号されたピクセルは、遷移が起こるべき時間に利用可能とする。また、これと類似のシステムも、米国特許5,495,291（Adams）および5,559,562（Ferster）に示されている。インターフレーム圧縮を使用するとき、第２のセグメントがインターフレーム・イメージから開始する場合、この第２セグメントの処理は、第２セグメントの所望のイメージを利用可能とすることができるようにするため、先の第１セグメントの処理の間に一層早期に始めねばならない。理想的には、第２セグメントは、先のイントラフレーム圧縮イメージから処理すべきである。しかしながら、これらの先行するイメージは、出力において用いられるものではない。

インターフレームおよびイントラフレームの圧縮されたビデオの第３のセグメントを再生するときに、問題が生じる。特に、第２セグメントは、第３セグメントの第１のイメージを先のイントラフレーム圧縮イメージから完全に処理できるようにするのに十分な程長くなければならない。復号器の２つのチャネルのみが利用可能である場合、第３シーケンスに対するこの処理は、第１セグメントを処理するため使用したのと同じ復号器を使用して、第１シーケンスを処理した後に実行することになる。あるケースにおいては、第１の復号器もまた、最後の所望のイメージが出力された後に幾つかのイメージを出力することがある。任意の第２セグメントの最小サイズは、カット密度（cut density）と呼ぶ。このカット密度は、原理的には、イントラフレーム圧縮のみを使用することにより単一のフィールドまで低減することができる一方、インターフレーム圧縮では、より一層良い圧縮を提供する。したがって、インターフレーム圧縮を使用してカット密度を最小にすることが望ましい。

ＭＰＥＧ－２のような一部の標準と互換性のあるシステムを設計する点での別の問題は、コード化されたビットストリームに存在したりまたは存在しなかったりする多くのオプションがあることである。例えば、ＭＰＥＧ－２にフォーマット化されたビットストリームは、Ｉフレームのみ、またはＩとＰのフレーム、あるいはＩ、ＢおよびＰのフレームを含むことある。これらのフレームが表示される順序もまた、それらが記憶された順序とは異なることがある。各圧縮イメージはまた、ほぼゼロから６個のフィールドの出力を生じることがある。Ｉフレームを含む任意の特定のイメージを復号するため必要とされる状態情報はまた、ビットストリームの任意の点で生起することがある。その結果、任意のＭＰＥＧ－２準拠（compliant）のビットストリーム中の特定のフィールドにランダムにアクセスできる能力は、このビットストリームの実際のフォーマットにより決まることがある。したがって、本発明の一般的な目的は、最小カット密度をもつインターフレームおよびイントラフレーム圧縮のモーション・ビデオを、非線形編集できるシステムを提供することにある。本発明の１つの実施形態における別の一般的な目的は、異なった圧縮フォーマットを有するインターフレームおよびイントラフレーム圧縮のデータ・ストリームの混合編集を可能にすることである。

発明の概要

インターフレームおよびイントラフレーム双方の技術を使用して圧縮されたビデオ・セグメント、の任意のフィールドに対するランダム・アクセスは、前記圧縮ビットストリーム中の適当な点に、復号し表示するため、状態情報を含めて、各イントラフレーム圧縮のイメージに対するランダム・アクセスを可能にすることにより、各イントラフレーム圧縮のイメージがランダムにアクセスされるのを可能にすることにより、強化する。さらに、フィールド・インデックスを発生し、これは、各時間フィールドを、そのフィールドを復号するのに使用するデータの前記圧縮ビットストリーム内におけるオフセットに対してマップする。さらなる利点は、２つ以上の交互に使用する復号器を使ってセグメントを再生することにより与えられる。カット密度の改善は、各復号器に印加されるビットストリームから、双方向に圧縮されたイメージに対応する任意のデータであってさもなければ復号器

が使用して所望のフィールドの前にフィールドを発生することになるそのようなデータを排除することにより行う。
したがって、本発明の１つの局面は、インターフレームおよびイントラフレーム技術を使用して圧縮されたモーション・ビデオ、を編集するためのコンピュータ・システムである。このコンピュータ・システムは、編集されるべき各モーション・ビデオ・ソースのため圧縮されたビットストリームを記憶する。各圧縮ビットストリームを処理することにより、圧縮データを復号および／または表示するのに使用する状態情報を検出する。この検出した状態情報は、各イントラフレーム圧縮イメージに対するビットストリーム中の適切な点に加える。この状態情報はまた、圧縮中に適切に挿入するようにしてもよい。本コンピュータ・システムはまた、圧縮ビットストリームを処理することによってインデックスを発生し、そしてこのインデックスにより、対応する圧縮解除された出力イメージ・シーケンスの各時間フィールドを、時間フィールドの圧縮解除を開始するのに使用する第１の圧縮イメージと、この第１圧縮イメージのためのデータのビットストリーム内におけるオフセットと、にマップする。このインデックスの作成は、モーション・ビデオを捕捉しまたはインポート（import）する間に、またはポスト処理アプローチを使用することにより行うことができる。本コンピュータ・システムは、編集システムを提供し、そしてこの編集システムは、ユーザがモーション・ビデオ・セグメントの合成物を指定できるようにし、そしてこれにおいて、各セグメントは、モーション・ビデオ・ソース内の時間フィールドに関して指定された範囲により定める。フィールド・インデックスを使用することにより、各モーション・ビデオ・セグメントを発生するのに使用する圧縮ビットストリームの部分を、セグメントを定める範囲を使用して識別する。２つ以上の復号器を使用することにより、各モーション・ビデオ・セグメントに対する圧縮ビットストリームの前記識別された部分を交互に処理する。
本発明の別の局面は、インターフレームおよびイントラフレーム技術を使用して圧縮されたモーション・ビデオ・データ、の圧縮されたビットストリーム中の各イントラフレーム・イメージに対し、ランダムにアクセスするのを可能にする方法である。圧縮ビットストリームを処理することによって、状態情報を検出する。この検出した状態情報は、各イントラフレーム圧縮イメージに対するビットストリームに加えることにより、任意のイントラフレーム圧縮イメージに対するランダム・アクセスを可能にする。
本発明の別の局面は、イントラフレームおよびインターフレーム技術を使用して圧縮されたモーション・ビデオ・データ、の圧縮されたビットストリームに対するフィールド・インデックスを発生する方法である。この方法においては、各圧縮イメージが表すビデオ・フィールドの数を判定する。次いで、ビットストリームの圧縮解除を開始して時間フィールドを得るのに使用する圧縮イメージを、識別する。次いで、各時間フィールドに対するフィールド・インデックス・エントリを発生し、そしてこれは、時間フィールドを、圧縮モーション・ビデオ・データのビットストリーム内におけるオフセットにマップし、そしてこれは、ビットストリームの圧縮解除を開始して時間フィールドを生成するのに使用する。このインデックスには、所望の時間フィールドの指示を入力として使用してアクセスすることができる。
本発明の別の局面は、インターフレームおよびイントラフレーム技術を使用して圧縮された複数のモーション・ビデオ・データ、を復号する回路である。この回路は、圧縮ビデオ・データを復号する複数の復号器を含む。インターフェースは、圧縮ビデオ・データを受け取り、そしてその圧縮ビデオ・データを復号器に与える。このインターフェースは、各復号器に印加されるビットストリームから、双方向に圧縮されたイメージに対応するデータであってさもなければ復号器が使用して所望のフィールドの前にフィールドを発生することになってしまうそのようなデータを、排除する。これら復号器の出力に接続したスイッチは、モーション・ビデオのどのフィールドを復号器から出力するかを制御して、これにより指定された時間フィールドの範囲内のそれらのフィールドのみが出力されるようにする。
本発明の他の局面は、本発明の前述の局面に対応するプロセス、システムまたは回路、お

よびそれらの種々の組合わせを含むものである。

【図面の簡単な説明】

図面において、

図１は、ビデオ編集システムのブロック図である。

図２は、図１の要素の１つ以上を実現するのに使用できるコンピュータ・システムのブロック図である。

図３は、本発明の一実施形態においてＭＰＥＧ－２のビットストリームを再フォーマット化する方法を示すフローチャートである。

図４は、フィールド・インデックスの一実施形態を図示する。

図５は、圧縮されたデータのビットストリーム順序と、時間フィールドおよびフィールド・インデックスに対する関係を示す図。

図６は、フィールド・インデックスを使用して、時間イメージ・フィールドに対応する圧縮イメージ・データを識別する方法ををを示すフローチャートである。

図７は、異なるビデオ・ソースからの複数のセグメントで構成される編集済みのビデオ・シーケンスの１つの表現を、例として示す図である。

図８は、本発明の一実施形態に従った回路のブロック図である。

図９は、図８のインターフェース回路のブロック図である。

図１０は、図８におけるピクセル・スイッチのブロック図である。

図１１は、図７に示したようなビデオ・プログラム表現を、図８から図１０の回路で実行するコマンドへと変換する方法を示すフローチャートである。

発明の詳細な説明

本発明は、添付図面と関連して読まれるべき以下の詳細な説明にからより完全に理解されるであろう。なお、図面において類似の参照番号は類似の構造を示す。本明細書において引用された全ての参考文献は、本文に援用する。

まず図１を参照すると、典型的な非線形ビデオ編集システム３０の主要構成要素が示されている。この編集システムは、捕捉システム３２を含み、これは、ビデオおよび／またはオーディオの情報をアナログまたはディジタルのソースから受け取り、その情報を所望のフォーマットに変換し、そしてこの情報を記憶システム３４に記憶する。この捕捉システム３２は、非圧縮のモーション・ビデオ情報を受け取り、イントラフレームおよび／またはインターフレーム技術を使用してその情報を圧縮するようにできる。代替例として、捕捉システム３２は、既に圧縮されたデータを受け取るようにしてもよい。この圧縮ビデオ・データは、以下に記載する要領で処理して、各イントラフレーム圧縮されたイメージに対するランダム・アクセスを可能にする。記憶システムは、通常、オペレーティング・システムのファイル・システムを介して他のアプリケーション・プログラムがアクセス可能なデータ・ファイル中に、データを記憶する。例えば、捕捉システム３２は、アプリケーション・プログラムまたはアプリケーション・プログラムの一部としてもよく、そしてそれは、ファイル・システム中のファイルにアクセスするオペレーティング・システムのコマンドを使用して、到来データをデータ・ファイル中に書き込む。記憶システム３４は、通常、コンピュータにより読取り可能かつ書込み可能な１つ以上のディスクである。編集システム３０はまたエディタ３６を含む。このエディタは、通常、モーション・ビデオ・プログラムの表現を操作し、そしてこれは、記憶システム３４に記憶されているファイルと、編集されたモーション・ビデオ・プログラムに含まれるべきマルチメディア・コンテンツのためのそれらのファイル内の範囲とへの参照を含む。再生システム３８もまた、編集システム３０の一部であり、編集されたモーション・ビデオ・プログラムを再生するため、並びに編集処理中において記憶システム３４からの情報を表示するために使用する。したがって、エディタ３６は、再生システム３８も含むようにしてもよい。

図１に示したシステムは、１つのコンピュータまたは幾つかのコンピュータで実現することができる。例えば、単一のスタンドアロンのコンピュータであって、捕捉システム３２、エディタ３６および再生システム３８の機能を定めるアプリケーション・プログラムを有ししかも適切な記憶システム３４を有するものを設けることができる。さらに、捕捉シ

ステム３２、エディタ３６、再生システム３８および記憶システム３４は、例えば、ネットワーク３９を介したクライアント／サーバ・プロトコルを使用して相互作用する別個のマシンとすることもできる。

次に、図２を参照して、図１の構成要素のいずれかまたは全てを実現するため使用できる典型的なコンピュータ・システム４０について、次に説明する。このコンピュータ・システム４０は、通常、情報をユーザに表示する出力装置４２を備えている。このコンピュータ・システムは、主装置４１を備え、これは、出力装置４２と、キーボードのような入力装置４４とに接続されている。主装置４１は、一般的に、メモリ・システム４８に相互接続機構を介して接続されたプロセッサ４６を備える。入力装置４４はまた、プロセッサ４６およびメモリ・システム４８に相互接続機構５０を介して接続しており、そして出力装置４２もこれと同様である。

ここで、１つ以上の出力装置をコンピュータ・システムに接続するようにできることは理解されるべきである。例示の出力装置としては、陰極線管（ＣＲＴ）ディスプレイ、液晶ディスプレイ（ＬＣＤ）、プリンタ、モデムのような通信装置、およびオーディオ出力装置を含み、再生システムは、ディスプレイへの出力のため圧縮イメージを復号する出力装置にアクセスするようにもできる。また、１つ以上の入力装置をコンピュータ・システムに接続するようにできることも理解されるべきである。例示の入力装置としては、キーボード、キーパッド、トラック・ボール、マウス、ペンおよびタブレット、通信装置、捕捉用ビデオおよびオーディオ入力装置、およびスキャナを含む。本発明は、コンピュータ・システムと組み合わせて使用する特定の入力装置または出力装置に、またはここで記述する入力装置または出力装置に限定されるものではないことは、理解されるべきである。

コンピュータ・システム４０は、ＡＣまたはＡＰａｓｃａｌ@のような高水準コンピュータ・プログラミング言語を使用してプログラム可能な汎用コンピュータ・システムとすることもできる。コンピュータ・システムはまた、特別にプログラムされた、特別目的のハードウエアとすることもできる。汎用コンピュータ・システムにおいては、プロセッサは通常商業的に入手可能なプロセッサであり、この場合インテルから入手可能なシリーズｘ８６プロセッサ、およびモトローラから入手可能な６８０Ｘ０シリーズ・プロセッサはその例である。他の多くのプロセッサも利用可能である。そのようなマイクロプロセッサは、オペレーティング・システムと呼ぶプログラムを実行し、そのオペレーティング・システムのうちのＵＮＩＸ、ＤＯＳおよびＶＭＳはその例である。このオペレーティング・システムは他のコンピュータ・プログラムの実行を制御し、スケジューリング、デバッギング、入力／出力制御、アカウンティング、コンパイル、記憶割り当て、データ管理およびメモリ管理、通信制御、および関連のサービスの実行を制御する。プロセッサおよびオペレーティング・システムは、コンピュータ・プラットフォームを定め、これに対し高水準プログラミング言語でのアプリケーション・プログラムが書かれる。

メモリ・システムは、通常、コンピュータにより読取り可能でかつ書込み可能な不揮発性記録媒体を備え、そのうちの磁気ディスク、フラッシュ・メモリおよびテープはその例である。ディスクは、フロッピー・ディスクとして知られた着脱可能のものとしてよく、またハード・ドライブとして知られた恒久的に装着したものとしてもよい。ディスクは多数のトラックを有し、そのトラック内に信号が、通常２進形式で、すなわち、１と０のシーケンスとして解釈される形式で記憶される。そのような信号は、マイクロプロセッサが実行すべきアプリケーション・プログラムを、またはアプリケーション・プログラムが処理すべきディスク上に記憶された情報を定めることができる。典型的には、動作において、プロセッサは、データを不揮発性記録媒体から集積回路メモリ要素に読み出すようにし、ここでその集積回路メモリ要素は、通常、ダイナミック・ランダム・アクセス・メモリ（ＤＲＡＭ）またはスタティック・メモリ（ＳＲＡＭ）のような揮発性のランダム・アクセス・メモリである。集積回路メモリ要素は、プロセッサによる情報に対するアクセスを、ディスクにおける場合より高速で行うことを可能にする。プロセッサは、一般的に、集積回路メモリ内のデータを操作して、次いで処理が完了したときそのデータをディスクにコピーする。ディスクと集積回路メモリ要素との間のデータの移動を管理するための種々の

機構が知られており、したがって本発明はそれらに限定されるものではない。また、本発明は、特定のメモリ・システムに限定されないことも理解されるべきである。
本発明は、特定のコンピュータ・プラットフォーム、特定のプロセッサ、または特定の高水準プログラミング言語に限定されないことは理解されるべきである。さらに、コンピュータ・システム４０は、マルチプロセッサ・コンピュータ・システムとしてもよく、またコンピュータ・ネットワークを介して接続された複数のコンピュータを含むようにしてもよい。
次に、捕捉システム３２の一実施形態の実現例について説明する。捕捉システムは、一般的に、到来するオーディオまたはビデオのデータを処理し、そしてそれを前述した記憶システム３４上の記憶ファイル中へと処理する。この一般的プロセスは周知である。受信したビデオ・データは、インターフレームおよび／またはイントラフレーム技術を使用して捕捉システムにより圧縮するようにしたり、または捕捉システムが、インターフレームおよびイントラフレーム技術を使用して圧縮された先に圧縮のビットストリームを受け取るようにしたりできる。圧縮ビットストリーム中の各イントラフレーム圧縮のイメージに対してランダム・アクセスを可能にするため、ビットストリームを再フォーマット化する。特に、圧縮イメージ・データを復号および／または表示するため使用する任意の状態情報は、ビットストリーム内の適切な点にコピーしそして挿入する。さらに、フィールド・インデックスを発生し、これは、圧縮モーション・ビデオ中の各時間フィールドを、フィールドを復号するため用いるデータの圧縮ビットストリームにおけるオフセットにマップする。
次に、図３で、圧縮ビットストリームを再フォーマット化するプロセスを説明する。以下の説明は、イントラフレームおよびインターフレーム双方の圧縮を与える例示的圧縮フォーマットとしてＭＰＥＧ－２を使用している。本発明は、インターフレームおよびイントラフレーム技術を使用する他の種類の圧縮に適用可能であり、したがって本発明のこの記載は例示としてのみ与えるものであることは理解されるべきである。
任意のイントラフレーム圧縮イメージに対してランダム・アクセスを可能にするため圧縮ビットストリームを再フォーマット化するプロセスは、捕捉プロセス中で、ビデオ・ストリームが符号化されつつある間、またはポスト処理またはインポートのステップが先に圧縮のデータに実行されている間において、実行するようにできる。このプロセスが実行できるのは、ＭＰＥＧ－２ビットストリームにおける多くのパラメータを一度指定し、次いで後続の全てのイメージに対して適用することができるからである。それらのパラメータは、ヘッダにおいて指定し、シーケンス・ヘッダ、シーケンス拡張子、シーケンス表示拡張子、シーケンス・スケーラブル拡張子、量子化マトリックス拡張子、および画像表示拡張子のような値を指定し得る。種々のヘッダは、ＭＰＥＧ－２仕様においてより詳細に記述されている。関心のあるパラメータは、コピーライト（copyright）・ヘッダまたは「ＧＯＰ」ヘッダのような単なる情報を与えるヘッダではなく、むしろ復号及び表示に影響を及ぼすヘッダである。任意のヘッダが、圧縮ビットストリーム中の第１の画像の後に生じる場合、しかもそれらが後続のイメージの復号および表示に適用するいずれの状態を実際に変える場合、ビットストリームを再フォーマット化して第１のそのような変化に続く各後続のＩフレーム前にそのヘッダを挿入する。
このプロセスの第１のステップ５０は、ＭＰＥＧ－２システム層ストリームを、別個のオーディオおよびビデオのパケット化されたエレメンタリ・ストリーム（Packetized Elementary Streams）（ＰＥＳ）またはエレメンタリ・ストリーム（Elementary Streams）（ＥＳ）に多重化解除（demultiplex）することである。次に、ステップ５２において、プログラム情報フィールドを、ビットストリームから捜し出して抽出するようにできる。これらのフィールドの例は、トランスポート（transport）ストリームにおけるプログラム・マップ・テーブル、またはプログラム・ストリームにおけるプログラム・ストリーム・マップを含む。プログラム情報は、オーディオおよびビデオのビットストリームの連関をプログラムとして定める。次いで、オーディオおよびビデオのビットストリームのサブセットを、ステップ５４において、システム・ストリームからのインポートのため選択する

。オーディオは、ステップ５６において圧縮解除し（ＭＰＥＧオーディオまたはＡＣ－３オーディオのいずれか）、そしてＰＣＭ（ＡＩＦＣ）データとして、例えば別個のデータ・ファイルに記憶する。非圧縮のオーディオの編集が通常なされる。代替として、圧縮オーディオ・データは、記憶しそして編集するようにできる。そのような圧縮オーディオ・データをランダム・アクセスの要領で編集することにはまた、圧縮により生じた従属性の故に、圧縮ビデオを編集するのに使用する技術と類似の技術が関係することがある。次いで、この圧縮ビデオは、ステップ５８において、適切なＭＰＥＧ－２ヘッダを挿入することにより、任意のＩフレームでアクセスできる形式に変換する。このインポート・プロセスは、その圧縮データ・ファイルがビデオ・データのみを含む場合、このステップ５８から始まる。特に、前述したように、ＭＰＥＧ－２ビットストリームは、状態情報を含む線形媒体であり、その状態情報はビットストリーム中のある一定の点で指定することができ、そしてこれは、続く全ての圧縮ビデオ画像に対し、あるいはリセット条件がビットストリーム中で生じるまで続く全ての圧縮ビデオ画像に対して効力を生じる。したがって、いずれの任意にかつランダムにアクセスされるＩフレームも、ビットストリームの復号を開始することができるようにするため、ある種の状態情報は、全ての後続のＩフレームの前に反復する必要があるようにして、復号器を、その開始からビットストリームを線形に復号した場合にそうであったであろう状態にセットするようにできる。特定の例について、次の３つのステップにおいて与える。これらは、主プロフィール、単純プロフィールおよび４：２：２プロフィールと呼ばれる状態情報のケースをカバーしている。ＳＮＲプロフィール、スケーラブル・プロフィールおよび高プロフィールに対して、追加のヘッダは、類似の要領で挿入しなければならないことになる。

特に、任意の量子化テーブルが第１のシーケンス・ヘッダの後の任意のシーケンス・ヘッダに存在する場合、最も最近発生した組みの量子化テーブルを有するシーケンス・ヘッダを、ステップ６０において、このビットストリームの残部に対する各コード化されたＩフレームの直前に、挿入する。ＭＰＥＧ－２のケースにおいては、シーケンス拡張子もまた、シーケンス・ヘッダを挿入する度毎に挿入する。ＭＰＥＧ－２のケースにおいてもまた、第１のシーケンス・ヘッダに続いてシーケンス表示拡張子が生じた場合、シーケンス表示拡張子は、シーケンス・ヘッダおよびシーケンス拡張子を挿入する度毎にシーケンス拡張子の後に挿入する。

同様に、任意のコード化された画像の画像コード化拡張子に続いて量子化マトリックス拡張子が生じた場合、量子化マトリックス拡張子を、ステップ６２において、全ての後続の画像の画像コード化拡張子に続いて挿入し、その画像コード化拡張子に対して、量子化マトリックス拡張子におけるマトリックスは、別の量子化拡張子が生じるかまたは次のシーケンス・ヘッダが生じるかのいずれかのときまで適用する。

次に、ステップ６４において、任意の画像コード化拡張子に続いて画像表示拡張子が生じた場合、最も最近に復号されたフレームの中心オフセットを有する画像表示拡張子を、別の画像表示拡張子が生じるかまたは次にシーケンス・ヘッダが生じるかのいずれかときまで、全ての後続の画像コード化拡張子に続いて挿入する。

インポート・プロセスは、モーション・ビデオをディジタル化して圧縮することにより避けることができ、これによって状態情報は、任意のイントラフレーム圧縮イメージに対するランダム・アクセスおよびそれからの再生を可能にするように、ビットストリーム中に既に存在するようにすることができる。特に、符号器は次の制約を実現するようにすべきである。第１に、シーケンス・ヘッダを適正に挿入するため、符号器は、次の３つの条件の１つが真であるように、ビットストリームを符号化するようセットアップすべきである。１）ビットストリームの始めにシーケンス・ヘッダがあり、かつこのビットストリーム中に他のシーケンス・ヘッダがない。または、２）全てのイントラフレームの前にシーケンス・ヘッダがある。または、３）ビットストリームの始めにかつ量子化テーブルを含む第１の反復シーケンス・ヘッダに続く全てのイントラフレームの前にシーケンス・ヘッダがある。なお、その量子化テーブルは、第１のシーケンス・ヘッダにおいて指定されたものがあった場合における第１のシーケンス・ヘッダ中のそれとは異なるか、または第１シ

ーケンス・ヘッダにおいてテーブルが指定されなかった場合におけるデフォルト量子化テーブルとは異なる。
量子化マトリックス拡張子（クオント・マトリックス拡張子（Quant Matrix Extension）すなわちＱＭＥ）を適正に取り扱うため、符号器は、ビットストリームを符号化するようセットアップし、これにより１）ＱＭＥがイントラピクチャ（intra-picture）内に現れる場合、ＱＭＥが、次のシーケンス・ヘッダが挿入されるまで、全てのイントラピクチャ内に現れねばならないようにし、また、２）クオント・マトリックス拡張子（ＱＭＥ）がインターピクチャ（inter-picture）内に現れる場合、ＱＭＥが、次のシーケンス・ヘッダが挿入されるまで、全てのインターピクチャ内に現れねばならないようにする。
画像表示拡張子（ＰＤＥ）を適正に取り扱うため、符号器は、ビットストリームを符号化するようセットアップし、これによりＰＤＥが任意の圧縮画像内に現れる場合、ＰＤＥは、次のシーケンス・ヘッダが挿入されるまで、全ての圧縮画像内に現れねばならないようにする。
ＭＰＥＧストリームが再フォーマット化された、または適正にフォーマット化されたストリームが捕捉された後に、フィールド・インデックスをステップ６６において生成する。このフィールド・インデックスを使用して、特定のビデオ・フィールドに対応する圧縮ビデオ・データを見つけ、そしてどの圧縮ビデオ・データを特定のビデオ・フィールドを再生するためにＭＰＥＧ復号器に供給すべきかを判定する。
次に、図４で、インデックスの一実施形態のフォーマットを説明する。各ＭＰＥＧファイルに対して、インポート・プロセスまたはディジタル化プロセスのいずれかは、フィールドのような各イメージに対し１つのエントリ７２をもつインデックス７０を生成する。インデックス中のエントリ７２は、圧縮イメージがビットストリーム中に生じる順序で、すなわち、コード化された順序であって表示順序ではなく記憶されていることに注意されたい。
各エントリ７２は、６４ビットの長さであり、かつオフセット７４を含み、このオフセット７４は、４８ビット、例えばビット０：４７により表すようにできる。これらのビットは、ＭＰＥＧヘッダのビットストリーム（ＯＭＦＩファイルではない）の中へのバイト・オフセットであり、そのＭＰＥＧヘッダはこのイメージを表す圧縮画像に先行する。この画像に対し、介在する画像なしでシーケンス・ヘッダが先行する場合、インデックスは、シーケンス・ヘッダに対するバイト・オフセットである。また、その画像に対し、介在する画像なしで画像ヘッダのグループが先行する場合、インデックスは、画像ヘッダのグループに対するバイト・オフセットである。そうでない場合は、インデックスは、その画像に先行する画像ヘッダのバイト・オフセットである。
各エントリ７２はまた、画像タイプ７６の指示を含み、この画像タイプ７６は２ビット、例えばビット４８－４９により表すようにできる。例としての値は、０１＝Ｉフレーム、１０＝Ｐフレーム、１１＝Ｂフレームである。値００は予約されている。これは、ビットストリーム中の指示されたオフセット７４で見つけられる圧縮ＭＰＥＧ画像の画像タイプである。
ランダム・アクセス・ビット７８もまた記憶されている。これは、単一のビットとすることができ（例えば、ビット５０）、これは、このフィールド・インデックス・エントリ７２により与えられるオフセット７４でビットストリームに対しランダム・アクセスが可能であるか否かを指示する。シーケンス・ヘッダ・ビットもまた、このフィールド・インデックス・エントリ７２がシーケンス・ヘッダを参照するか否かを指示するため記憶できる。これは単一のビット（例えば、ビット５１）により表すようにできる。例えば、このフィールド・インデックス・エントリ７２が画像ヘッダまたはＧＯＰヘッダを指す場合、ビット５１はゼロである。このフィールド・インデックス・エントリがシーケンス・ヘッダを指す場合、ビット５１は１である。
エントリ７２における最後の値は、時間オフセット８２である。この値は、あるビデオ・フィールドの時間フィールド番号と、そのビデオ・フィールドを含む圧縮ＭＰＥＧ画像のオフセット値を含むフィールド・インデックス７０中のエントリ番号との間のオフセット

を意味する。ビデオ・フィールドN（但し、Nは関心のあるビデオ・フィールドの時間番号である）にアクセスするため、フィールド・インデックス・エントリNを読み出し、そしてこれが含む時間オフセット８２の値にNを加える。この和を使用して、フィールド・インデックス７０中に再びインデックスして、関心のあるフィールドを含む圧縮画像のオフセット７４を含むフィールド・インデックス・エントリ７２を検索する。

このインデックスの発生は、ポスト処理タスクとして行うようにしたり、またはモーション・ビデオが圧縮されつつある間に実行することができる。イントラフレームのみのシーケンスをインデックスするためのプロセスは、米国特許5,577,190（Peters）に開示されており、それは本明細書に援用する。そのプロセスにおいては、割込みが、符号器により出力された各圧縮イメージの終わりで発生する。データ・バッファをモニタすることにより、そのイメージのために使用する圧縮データの量を判定する。インターフレームおよびイントラフレームの圧縮イメージのシーケンスをインデックスするため、類似の技術をが用いるが、しかし、追加の情報を、割込みが発生された時間に各イメージに対して利用可能にするべきである。特に、各圧縮画像の画像タイプ、および各圧縮画像により表されるビデオ・フィールド数が必要となる。この情報は、符号器のセッティングにより予め既知とすることができる。例えば、符号器をセットして、ディスエーブルされた反転テレシネ（inverse telecine）（反転３：２プルダウン）を有する画像の正規のグループを使用することができる。代替例として、符号器は、この符号器からの出力によるか、または読み出すことができるレジスタによるかのいずれかにより、別個のデータ・パスを与えて、各圧縮画像のため、画像タイプ、バイト単位での圧縮サイズ、および圧縮画像により表されるフィールド数を出力するようにできる。

次に、図５で、ＭＰＥＧビットストリームおよびそれに関連したフィールド・インデックスの例を示す。「ＭＰＥＧ画像のビットストリーム順序」と付した第１のセクション９０は、ＭＰＥＧビットストリーム中に見つけられる圧縮画像を表す。「各コード化された画像により表されるビデオ・フィールドの数」と付した第２のセクション９２は、第１のセクションの各圧縮ＭＰＥＧ画像に含まれるビデオ・フィールドの数を示す。第３のセクション９４は、ビットストリーム中のビデオ・フィールドの表示順序を表す。各ビデオ・フィールドは、時間フィールド番号により番号付けされ、垂直線により表している。垂直線の位置は、それが上側フィールドか下側フィールドかを示す。「ＭＰＥＧ画像」と付したライン９６は、ビットストリーム中のどのＭＰＥＧ画像がどの時間ビデオ・フィールドを表すかを示す。ここで、ＭＰＥＧ画像は、ビットストリーム順序ではなくむしろ時間順序で示している。「フィールド・インデックス：エントリ番号」、「フィールド・インデックス：オフセット」および「時間オフセット」とそれぞれ付したライン９７－９９は、前述したフィールド・インデックス７０の各部分を表す。

N番目の時間ビデオ・フィールドに対応するＭＰＥＧ圧縮画像を捜し出すため、図６に示すプロセスに従う。特に、エントリ番号を、ステップ１００において、フィールド・インデックス７０のN番目のエントリ７２にアクセスして、時間オフセット・ロケーション８２に記憶されている値を検索することにより計算する。その時間オフセット値は、このエントリ番号を得るため値Nに加える。所望の画像のオフセットは、ステップ１０２において、フィールド・インデックス７０からステップ１００において計算したエントリ番号に対応するエントリにアクセスすることにより判定する。判定したこのエントリに記憶されたオフセット７４が、所望の画像オフセットである。図５に示した例を使用すると、時間フィールド番号Nが８である場合、エントリ番号は３である。画像オフセットは、フィールド・インデックスのエントリ番号３に記憶されているオフセット値であり、それは第２フィールドＰ３のイメージである。

ＭＰＥＧ符号化されたモーション・ビデオ／オーディオ・データを含む媒体ファイルの内容について、をここまで説明したが、次にエディタによるビデオ・プログラムの発生について、図７で説明する。

図７には、ビデオ・プログラムの１表現例を示している。ビデオ・プログラム１１０は、１１２、１１４、１１６および１１８で示す幾つかのセグメントを含む。ビデオ・プログ

ラム１１０中に幾つかのさらに多くのセグメントがあるようにしてもよいことは、理解されるべきである。ある例では、ビデオ・プログラムを定める２トラックのビデオがあるようにしてもよく、その場合、第１および第２のトラックは、ある方法で混合または組み合わせることにより、例えばピクチャ・イン・ピクチャ（picture in picture）を発生したり、ディゾルブされた遷移、または他の任意の三次元ディジタル・ビデオ効果のような特別の効果を発生させる。各セグメント例えば１１２は、媒体オブジェクトに対する参照を含み、この媒体オブジェクトは、媒体データのソースと、およびこのセグメントを生成するため用いるべきそのソース内の範囲を意味する。ＰＣＴ公開ＷＯ93/21636（Wissner）および米国特許5,267,351（Reber）に示されたもののような、ビデオ・プログラム１１０の構造を表す多くの方法がある。１つのファイル内のその範囲は、通常、時間コードを使用することによるように、ソース内のセグメントの始めおよび終わりでの時間フィールドの何らかの指示を使用して表される。

図７に記載されたようなある編集されたシーケンスを仮定すると、これは、編集プロセスにおけるのと同じように再生したり、または最終ビデオ・プログラムを与える出力を発生するようにできる。このようなプログラムは、次に図８から図１１で説明するように再生することができる。イントラフレーム圧縮フォーマットにおいてのみ定められたセグメントのシーケンスの再生、および遷移を与えること等は、例えば、公開されたＰＣＴ国際出願ＷＯ94/24815並びに米国特許5,405,940および米国特許5,267,351（Reber）に記載されている。本発明においては、効果およびシーケンスの双方を生成するためイントラフレームおよびインターフレームの双方の技術を使用して圧縮されたモーション・ビデオを処理する能力を含むようそのようなシステムを拡張することについて、次に説明する。

インターフレームおよびイントラフレーム圧縮されたビデオのセグメントを取り扱う本発明の一実施形態における回路について、次に図８を参照して説明する。この実施形態は、ＭＰＥＧ－２を例示の圧縮フォーマットとして使用して説明する。

図８は、典型的なコンピュータ・システムの周辺接続インターフェース（ＰＣＩ）バスに接続するよう設計した回路である。多くの他のタイプのバスおよび接続を使用してもよいことは、理解されるべきである。したがって、本ボードは、ＰＣＩインターフェース１２０を備える。ＰＣＩインターフェース１２０は、Digital Equipment社により製造されたＰＣＩ対ＰＣＩブリッジ・チップ２１１５２を使用して実現することもできる。このインターフェースには、直接メモリ・アクセス（ＤＭＡ）コントローラ１２２および１２４を接続し、そしてこれらＤＭＡコントローラ１２２および１２４は、ホスト・コンピュータからのコマンド、特に再生または編集のアプリケーションに応答して、再生すべき記憶装置３４上のデータ・ファイルから転送されたビデオ・データを取り扱う。これらＤＭＡコントローラは、到来データをバッファリングするため、関連したメモリ１２６および１２８をそれぞれ有する。各ＤＭＡコントローラは、１つのＰＣＩロードを表す。ＰＣＩブリッジは、バス１２１上での多数のＤＭＡコントローラの使用を可能にする。次いで、これらＤＭＡコントローラは、データを１３０で示される４つの復号器に与え、そしてこの４つの復号器の各々は、関連のメモリ１３２を有する。コントローラ１２２および１２４を復号器１３０に接続するインターフェースは、１３４および１３６のそれぞれで指示している。復号器１３０は、International Business Machine（ＩＢＭ）から入手可能なＭＰＥＧＭＥ３１チップ・セットのような、例えばＭＰＥＧ－２復号器とすることもできる。ピクセル・スイッチ１３８は、上記復号器の出力に接続し、そして選択した復号器の出力をバッファ１４０に与える。バッファ１４０は、ビデオ情報の１フィールドを保持するのに十分なデータを含むフィールド・バッファまたはフレーム・バッファとすることもできる。これらバッファの出力は、混合器（blender）１４２に与え、この混合器１４２は、関連のメモリ１４６を有するアルファ／アドレッシング回路１４４により、ＰＣＴ公開ＷＯ94/24815に開示された要領で制御する。同様に、ＰＣＴ公開ＷＯ94/24815に開示されているように、この混合器への１つの入力はまた、ディジタル・ビデオ効果ユニット（digital video effect unit）１４８に与えることができ、一方、混合器の出力は、ディジタル・ビデオ効果ボードの別の入力に与えるようにすることもできる。１５０で示すディジ

タル・ビデオ効果ボードの出力は、適当なビデオ符号器へ再生されるようになる前にバッファ１５２に入力する。パラメータ・バス１５４を使用して、この再生回路の種々のレジスタおよびメモリ・ロケーションおよび制御ポートをセットする。
次に、図９で、インターフェース１３４および１３６について説明する。これらのインターフェースは、フィールド・プログラマブル・ゲート・アレイを使用して実現でき、そして復号器１３０内のＤＭＡコントローラ１２２とＤＭＡコントローラ１２４との間のインターフェース層として作用する。これらのインターフェースは、バス・フォールディング（bus folding）、アドレス・デマルチプレックシング（アドレス多重化解除）、マーカー・コード検出、データ・フラッシングおよび一般的インターフェース変換のような、圧縮データ・ストリームに対するデータ経路機能を行う。
これらインターフェースを介して生じる３つのクラスのデータ転送、すなわち、３２ビットＤＭＡ転送、１６ビット・スレーブ転送および３２ビット・スレーブ転送がある。ＤＭＡ転送は、バッファ１２６および１２８からＭＰＥＧ復号器のＦＩＦＯスペースへの書込み転送である。ＭＰＥＧ復号器は１６ビット幅インターフェースを有し、ＤＭＡ転送は３２ビット幅である。このインターフェースは、ＭＰＥＧ復号器アドレス０８（１６進）でＭＰＥＧ復号器のビデオＦＩＦＯレジスタへの２つのバック・ツー・バック（back-to-back）書込みサイクルへと、ＤＭＡ転送を折畳む。ＤＭＡ読出し転送は、これらのインターフェース１３４および１３６によりサポートされる必要がない。ＭＰＥＧ復号器レジスタ・アクセスは、パラメータ・バス１５４上の読出しおよび書込みサイクルとして生じ、そしてインターフェース１３４および１３６によりＭＰＥＧ復号器バス上の読出しおよび書込みサイクルへ変換される。
ＭＰＥＧ復号器１６ビット幅レジスタのこのインターフェースを介するアドレス・マッピングは、パラメータ・バス１５４上の３２ビット幅スペースに対してマップする。データは、パラメータ・バスの２有意バイト（two significant bytes）上に通す。小さいＭＰＥＧ復号器レジスタ・アドレスは左に２つシフトする。０２（１６進）のＭＰＥＧ復号器アドレスは、パラメータ・バス・アドレス０８（１６進）である。インターフェース１３４および１３６の内部レジスタはまた、４バイト・アドレス境界上に整列させ、そして３２ビットの長さとすることができる。
インターフェース１３４および１３６はまた、バイト・フラッシング機能を行い、そしてその機能において、それらインターフェース１３４および１３６は、ビデオ・データ・ストリームへのＭＰＥＧ上のＩ、ＢおよびＰ画像ヘッダ・コードのためのデータ経路を介してのＤＭＡデータ通過を走査する。Ｂ画像ヘッダに出会ったとき、このインターフェースは、次の事象の１つ、すなわち１）Ｂ画像ヘッダ以外のヘッダが検出されること、または２）プリセットＢ画像カウンタがゼロに減分されること、が真になるまでＤＭＡデータ・ストリーム中の全てのバイトを捨てる。所望のビデオ・フィールドの前に生じるビットストリーム内のどの「Ｂ」画像も、所望の出力に対して何も寄与しないので、上記バイト・フラッシング機能を使用する。それら画像を落とすことにより、フィールドのシーケンスを復号する時間はより短くすることができる。
インターフェース１３４および１３６が実行する別の機能は画像開始コード検出であり、これにより、前述したようにＢ画像を検出して捨てることが可能となる。その検出は、一般的に、開始コード・シーケンスのためＤＭＡコントローラからの到来データ・ストリームを解析（parse）するのを可能にする。特に、画像ヘッダおよび全てのそれらのＭＰＥＧヘッダは、「１」の１ビットが続く「０」の２３個のビットの開始コードにより始まる。画像開始コードは、ヘッダ開始コードの直後に続く。画像開始コードの値は「００」である。したがって、画像ヘッダに対し検出することが必要なバイト・シーケンスは「０ｘ０００００１００」である。画像がＢフレームであることを判定するため、論理回路は、画像開始コードの終わりから１０ビット後に生じる３ビットである画像コーディング・タイプ・フィールドを検査する。したがって、走査することになる全バイト・ストリングは、０ｘ０００００１００ｘｘｃｃである。但し、ｃｃは、ビット・ストリングＸＸｐｐｐＸＸＸに等しく、ここでｐｐｐは画像コーディング・タイプに等しい。許される画像コー

ディング・タイプは、ＭＰＥＧ－１においては、Ｉ画像に対して００１であり、Ｐ画像に対して０１０であり、Ｂ画像に対して０１１であり、そしてＤ画像のため用いられる１００である。
ヘッダを復号するため６バイトを処理する。これらのバイトは、それらを使用するか否かを判定するまでポスト検出器バッファに保持する。この画像がＢ画像であり、Ｂ画像のドロップがイネーブルにされ、かつＢ画像カウンタがゼロでない場合、ポスト検出器のバイト記憶装置をフラッシングし、そして全ての到来バイトを、次の画像開始コードを検出するまで落とす。上記のことが真でない場合、全てのバイトを、ＭＰＥＧ－２復号器に通すことになる。
本発明の一実施形態において、インターフェース１３４および１３６は同一であり、各ＤＭＡエンジンに対し１つ設けている。このような変調器の設計により、２つのモーションＪＰＥＧエンジンが、ＭＰＥＧ－２復号器の代わりにＤＭＡエンジンの１つと組み合わせて使用することができるようになる。追加のまたは代替の圧縮エンジンは、媒体タイプの混合を許すため、ドーター・カード（daughter card）を使用して実現するようにする。
図９は、インターフェース１３４および１３６の一実施形態を図示している。この図は、これらのインターフェースの１つを表している。パラメータ・バス１５４は、入力レジスタ１６０および出力レジスタ１６２に接続している。アドレス・データは、アドレス・ラッチ１６４を介してまた入力１６６を通して受け取る。入力レジスタ１６０を介して受け取る入力データは、マルチプレクサ１６８に印加する。画像検出器１７０および１７２は、画像がライン１７４上で利用可能か否か、そしてその画像がＢフレームであるか否かを検出する。画像検出器１７０は第１の復号器に対し使用し、一方、画像検出器１７２は、第２の復号器に対し使用する。これら画像検出器の出力は、ドロップ（drop）ロジック１７６および１７８のそれぞれに印加する。画像カウンタ１８０および１８２は、インターフェースにより検出された画像の数を追跡する。第１のチャネルに対して、データ出力レジスタ１８４は、出力ビデオ・データを与える。データ入力レジスタ１８６は、入力ビデオ・データを符号器バスから受け取る。アドレス／コマンド・レジスタ１８８は、アドレスおよびコマンドの情報を第１復号器に出力する。類似の入力／出力およびコマンド・レジスタ１９０、１９２および１９４は、第２復号器に対し設けている。さらに、これら復号器からのビデオ要求は、要求ロジック要素１９６および１９８により受け取る。これらの要求要素は、それら要求をＤＭＡエンジンに要求２００として渡す。
次に、図１０で、ピクセル・スイッチ１３８について説明する。このピクセル・スイッチは、ピクセルのストリームをＭＰＥＧ復号器から受け取る４つのポート２１０、２１２、２１４および２１６を含む。ピクセル・スイッチはまた、パラメータ・バス・インターフェース２１８を含み、このパラメータ・バス・インターフェース２１８は、時間基準発生器２２０、フィールド・シーケンサ・ロジック２２２、２２４、２２６および２２８およびマルチプレクサ・コントローラ２３８を制御するためパラメータ・バス１５４（図８）から受け取る制御情報を記憶する制御レジスタである。フィールド・シーケンサ・ロジックは、ピクセル・ポート２１０～２１６を制御する。マルチプレクサ２３０および２３２は、それぞれのピクセル・バス２３４および２３６上に出力してピクセル・スイッチの出力を供給するため、出力ビデオ・データを４つの全てのピクセル・ポートから受け取る。これらのマルチプレクサは、以下に説明するように、再生されるべきビデオ・プログラムにしたがってコントローラ２３８により制御される。
このピクセル・スイッチは、ＭＰＥＧ復号器とピクセル処理パイプまたはチャネルとの間のインターフェース層として作用する。ピクセル・スイッチは、４つのＭＰＥＧピクセル出力の１つを、この回路上のいずれかのピクセル・パイプに指向させるのを可能にする。ピクセル・スイッチのこのスイッチングは、以下で説明するように、垂直帰線消去期間に生じ、そしてこれは、フィールド毎に変化させることができる。
ピクセル・スイッチはまた、各ＭＰＥＧ復号器に対して１つ当てで、４つのシーケンサを含む。これらのシーケンサは、復号器をフィールド毎に進める働きをする。この機能を使用することにより、編集されたビデオ・プログラムによりカット点（cut point）として

定められた指定のイントラフレーム圧縮イメージの後の任意のフィールドに対して、指定された復号器をシーケンス付けさせる。各シーケンサは、待ち状態かまたは存在するアクティブのフィールドかのいずれかから進めるため、フィールドの数を定めるのに用いる二重バッファ式プログラマブル・レジスタを有するようにすることもできる。各復号器は、リセット後に正しい順番にシーケンス付けして、このシーケンサがこれが第１のフレームの第１のフィールド上にあることが分かるよう保証する。

このリセット手順は、次のとおりである。すなわち、再生アプリケーションは、リセットを、所望の復号器に対しインターフェース１３４および１３６のチャネル・リセット・コマンド・ビットを介して発する。次いで、ピクセル・スイッチ制御レジスタ２１８の中の初期化ビットをセットする。次いで、再生アプリケーションは、シーケンサ２２２からの割込みを待つ。ポート・シーケンサは、復号器のリセット後に、１６．６ミリ秒の３つの垂直同期信号をそれらの通常の周波数で発する。ピクセル・スイッチ中のシーケンサ２２２は、待ち状態に入り、割込みをＰＣＩバスにＤＭＡエンジンを介して通知し、そしてその状態レジスタ内のフラグをセットする。このフラグのセットを検出すると直ぐに、再生アプリケーションは、復号器のマイクロコードおよびレート・バッファ（rate buffer）にロードする。次に、制御レジスタ２１８中の制御ビットをセットして、シーケンサに初期化を完了させる。もう１つの垂直同期信号の後で、ピクセル・スイッチは、３０ミリ秒待ち、次いでもう３つの垂直同期信号を発する。この時点で、復号器は、第１の復号された画像の第１のフィールドを出力しているはずである。

シーケンサが初期化されたとき、シーケンサは、フィールド・コンテンツ・レジスタの内容により進めるべきフィールドの数を知らされる。ある特定の復号器のためのフィールド・スキップ・カウンタがゼロに等しい場合、この復号器は待ち状態に留どめられる。フィールド・カウント・レジスタに非ゼロ値がロードされるか、またはその値がこのピクセル・スイッチによりピクセル・ソースとして選択されると、この待ち状態を出る。フィールド・カウンタ・レジスタは二重にバッファされ、これにより書込まれた値はシャドウ・レジスタ（shadow register）に入り、次いでそれはそのカウンタ中に次の垂直同期信号時にロードされる。ピクセル・スイッチのこの機能は、パラメータ・バス１５４を使用して再生アプリケーションによりロードされる二重バッファ形機能を与える。再生アプリケーションがピクセルのソースを変えた場合、再生アプリケーションは、ピクセル・ポート選択ビットをコントローラ２３８中にロードし、そしてこのコントローラ２３８は、所与のピクセル・ポートのソースを次の同期期間のときに変える。

次に、図１１で、再生アプリケーションが、図８から図１０の回路を使用して、図７に示したビデオ・プログラムにより規定されるように、任意のＥＭＰＥＧ－２の符号化シーケンスを表示する方法について説明する。

合成は、最初に、既知の技術を使用して再生グラフと呼ぶものに変換する。例えば、再生グラフは、この出願と同日にJames Hamiltonにより出願され、発明の名称が「モーション・ビデオ出力デバイスを管理しかつコンテキストおよびバッファ選定をサポートするビデオ・デバイス・マネージャ（VIDEO DEVICE MANAGER FOR MANAGING MOTION VIDEO OUTPUT DEVICE AND SUPPORTING CONTEXTS AND BUFFER ADOPTION）」である米国特許出願に記載されているように、Avid Technology社からの仮想デバイス・マネージャによる使用のための相互接続した仮想デバイスの集合体、またはマイクロソフト社またはMatrox社からのアクティブムービー・ビデオ・デバイス・ドライバ（ActiveMovie video device driver）を使用したフィルタ・グラフとすることができる。そのようなグラフは、再生回路と、ビデオ・データを含むデータ・ファイルに対する読出しオペレーションとに対するコマンドのシーケンスに変換される。

次に、図１１を参照すると、ステップ３００において、所望のフィールドを復号するため必要とされる圧縮された第１のイメージを、フィールド・インデックスを使用して識別する。特に、その指定された時間フィールドのための圧縮ビットストリーム中への、オフセットを含むフィールド・インデックスにおけるエントリを、図６と関連して前述したように判定する。次に、最も近い先行のイントラフレーム圧縮イメージを、フィールド・イン

デックスを第１のＩフレームに対して後方に走査することにより識別する。しかしながら、現在のフレームがＢフレームである場合、少なくとも２つの参照フレーム（Ｉフレーム（複数）またはＰフレーム（複数））が見つけられねばならず、その場合、最後の参照フレームは、それから復号を開始するＩフレームである。フィールド・インデックスを後方に走査するとき、少なくとも２つのフィールドが参照フレームを識別するため必要とされる。したがって、ＰタイプまたはＩタイプの画像の２つの隣接するエントリは、１つのフレームを構成する。

第１の圧縮イメージにより出力された第１フィールドと所望のフィールドとの間のフィールド数は、ステップ３０２において判定する。このステップの実行は、識別したイントラフレーム・イメージで開始してフィールド・インデックスを走査することにより、そしてエントリ（コード化された順序で生じる）を、復号されたフィールドが復号プロセスをエミュレートする方法で出力されることになる順序に論理的に再び順序付けすることにより行う。ＭＰＥＧ符号化された素材の時間範囲のマッピングを実現するソース・コードを、付録Ｉとして添付し、そしてこれは本明細書に援用する。このソースは、図６のプロセスを実現する機能“GetDOrderField”を含む。“GetLeaderInfo”と呼ばれる別の機能は、ステップ３００および３０２において記述するように、指定された時間フィールドの前のフィールド数を識別する。さらに、復号器がクリップの終了後に出力することのあるフィールドの数は、ステップ３０４において同様の方法で判定する。この数は、ゼロ～６のどこかとすることができる。付録における機能の別の対、“GetEndofRange”および“GetTrailingDiscard”は、ステップ３０６を実行するため使用することができる。

次いで、「Ｂ」画像カウンタ１８０および１８２（図９）は、ステップ３０２において判定した値にしたがって、ステップ３０６においてセットする。次いで、復号器は、ステップ３０８においてリセットし、初期化する。次いで、ピクセル・スイッチを、ステップ３１０においてセットすることができる。ステップ３００ないし３１０の初期化がされたすると、データを、ステップ３１２においてデータ・ファイルから読み出し、再生のための回路に送る。さらにデータが必要とされ、そしてシーケンスの再生が進行するにつれ、複数のピクセル・スイッチを異なる方法でセットし、そして追加のデータをＤＭＡコントローラによりデータ・ファイルから読み出して転送するようにでき、これはGetEndofRange機能を使用して定義されたクリップの終わりにより制限される。

前述したように、圧縮ビットストリームを再フォーマット化して、復号および表示に影響を及ぼす状態情報を加えることにより、各イントラフレーム圧縮イメージに対するランダム・アクセスを可能にする。さらに、フィールド・インデックスは、このフィールドを再構成するため使用する圧縮イメージ・データのスタートのビットストリーム内のオフセットに対して、時間フィールドをマップするのを可能にする。このビットストリーム中の情報は、それが双方向に予測されるイメージを表しかつ所望のフィールドの前である場合、復号器に与えられる前にドロップさせることもできる。そのようなデータのドロップにより、フィールドのシーケンスを復号するための時間量を低減させることができ、これによりカット密度を改善することができる。それにより、ランダム・アクセスおよび改善されたカット密度は、インターフレームおよびイントラフレーム技術を使用して圧縮されたモーション・ビデオ・データの任意のセグメントを含むビデオ・プログラムを構成するエディタの能力を改善する。

圧縮されたオーディオは、複数のオーディオ復号器および出力上のサンプル・ドロップ回路でもって、本明細書に記載したような圧縮ビデオとかなり似た方法で編集することができる。

本発明の幾つかの実施形態について記述したが、以上のことは単なる例示であって限定するものではなく、例示としてのみ提示したことが当業者には明らかなはずである。多くの変更および他の実施形態は、当業者の範囲内にあり、添付の請求の範囲により定まる本発明およびその均等の範囲内に入るものである。

付録Ｉ

```
/*
 * /----------------------------------------------------------------\
 * | The following programs are the sole property of Avid Technology, Inc., |
 * | and contain its proprietary and confidential information.
 * |      Copyright 1989-1996 Avid Technology Inc.
 *  |
 * \----------------------------------------------------------------/
 */

/*****************************************************************************
*******
    MPEGMapper.c

    MPEGMapper class and function definitions

*****************************************************************************
*/

#include "masterheader.h"
#include "AMEBase.h"
#include "MPEGMapper.h"
#include "DIDPosition.h"
#include "DIDDescriptor.h"
#include "MPGIDescriptor.h"
#include "MPEGPosition.h"
#include "Exception.h"
#include "memrtns.h"
#include "MPEGDefs.h"

#define MPEGMapperVersion
```

```
#if !PORT_LEXT_INHERITED
#undef inherited
#define inherited AMapper
#endif
OBJECT_STD_C(MPEGMapper)

MPEGMapper::MPEGMapper(void)                    // OBJECT_STD_C requires this, but
don't use it
{
        FtlAssertNotReached();
}

MPEGMapper::MPEGMapper(ameBaseStream *s, DIDDescriptor* desc, AvUnit_t
NumSamples,
                                        long SampleSize, Boolean isfixedsize)
{
        _NFields = desc->GetFrameLayout() == eSEPARATE_FIELDS ? 2 :1;
        IDIDMapper(s, desc, NumSamples * _NFields, SampleSize, isfixedsize,
sizeof(MPEGFrameIndexEntry));
}

void MPEGMapper::GetBOBInfo(AvUnit_t BeginSample, AvUnit_t NumSamples,
                                                AvUnit_t* offset, AvUnit_t* length,
Boolean* needSeqHdr)
{
        if (!_IsFixedSize)
        {
                AvUnit_t dorderSample = GetDOrderField(BeginSample, FALSE);
                AvUnit_t firstIFrame = dorderSample - GetLeaderLen(dorderSample);
                long seqHdrLen = 0;

                // add length of sequence header if needed
```

```
                *needSeqHdr = ! HaveSequenceHdr(firstIFrame);
                if (*needSeqHdr)
                        seqHdrLen = ((MPGIDescriptor*)
_Desc)->GetSequenceHdr(NULL);

                *offset = GetFXOffset(firstIFrame);

                if (NumSamples)
                        *length = GetEndOfRange(BeginSample, NumSamples) - *offset
+ seqHdrLen;
        }
        else
        {
                *offset = 0;
                *length = NumSamples * _SampleSize;
                *needSeqHdr = FALSE;
        }
    }

    APosition*
    MPEGMapper::MapSample(AvUnit_t SampleNum) {
        if (!_IsFixedSize)
        {
                AvUnit_t offset;
                Boolean needSeqHdr;

                GetBOBInfo(SampleNum, 0, &offset, NULL, &needSeqHdr);

        return new MPEGPosition(offset, 0, NullMobID(),
                                        NULL_TRACKLABEL,
SampleNum, 0, FALSE, needSeqHdr,
                                        (MPGIDescriptor*)_Desc, this);
```

```
    }
    else
        return new MPEGPosition(SampleNum * _SampleSize, _SampleSize,
NullMobID(),
                                                    NULL_TRACKLABEL,
SampleNum, 0, FALSE, FALSE,
                                                    (MPGIDescriptor*)_Desc, this);
}

AvUnit_t MPEGMapper::BufferSize(AvUnit_t BeginSample, AvUnit_t NumSamples)
{
    AvUnit_t offset;
    AvUnit_t length;
    Boolean needSeqHdr;

    GetBOBInfo(BeginSample, NumSamples, &offset, &length, &needSeqHdr);

    return length;
}

AvUnit_t MPEGMapper::GetSampleOffset(AvUnit_t SampleNum) {
    AvUnit_t dorderSample = GetDOrderField(SampleNum, FALSE);
    return GetFXOffset(dorderSample - GetLeaderLen(dorderSample));
}

AvUnit_t MPEGMapper::GetFXOffset(AvUnit_t dorderField)
{
    if (!_IsFixedSize)
    {
        MPEGFrameIndexEntry* entryP;

        ValidateSampleNum(dorderField);
```

```
                entryP = (MPEGFrameIndexEntry*) (_FXPtr + 2 * (dorderField -
_rMin));
                return entryP->offsetLow + (entryP->offsetHigh << 32);
            }
            else
                return dorderField * _SampleSize;
        }

        int MPEGMapper::GetPictureType(AvUnit_t dorderField)
        {
            if (!_IsFixedSize)
            {
                MPEGFrameIndexEntry* entryP;

                if (dorderField == _NumSamples)
                    return MPEGIPicture;

                ValidateSampleNum(dorderField);

                entryP = (MPEGFrameIndexEntry*) (_FXPtr + 2 * (dorderField -
_rMin));
                return entryP->flags & MPEGPictureTypeMask;
            }
            else
                return MPEGIPicture;
        }

        int MPEGMapper::GetFieldOffset(AvUnit_t dorderField)
        {
            int result = 0;

            if (!_IsFixedSize)
```

```
        {
            AvUnit_t curFXOffset;
            AvUnit_t ix = dorderField;

            curFXOffset = GetFXOffset(ix);
            ix--;
            while (ix >= 0 && GetFXOffset(ix) == curFXOffset)
            {
                ix--;
                result++;
            }
        }

        return result;
    }

    Boolean MPEGMapper::HaveSequenceHdr(AvUnit_t dorderField)
    {
        if (!_IsFixedSize)
        {
            MPEGFrameIndexEntry* entryP;

            if (dorderField == 0)
                return TRUE;

            ValidateSampleNum(dorderField);

            entryP = (MPEGFrameIndexEntry*) (_FXPtr + 2 * (dorderField -
_rMin));
            return (entryP->flags & MPEGSequenceHdrBit) != 0;
        }
        else
```

```
            return TRUE;
    }

    // GetDOrderField returns the disk order sample index corresponding to the
    // picture which will produce the Nth temporal order frame.  This is determined
    // by a delta stored in the frame index.

    AvUnit_t MPEGMapper::GetDOrderField(AvUnit_t SampleNum, Boolean lastField)
    {
        AvUnit_t result = _NFields * SampleNum;
        MPEGFrameIndexEntry* entryP;

        if (lastField)
            result += _NFields - 1;

        if (!_IsFixedSize)
        {
            ValidateSampleNum(result);

            entryP = (MPEGFrameIndexEntry*) (_FXPtr + 2 * (result - _rMin));

            return min(result + entryP->toDoDelta, _NumSamples-1);
        }
        else
            return result & 1;
    }

    // GetFieldPairing does a localized search to determine whether the given field (in disk
order)
    // is the first or second field of a pair.  This is primarily needed when field-based coding
is
    // involved.  The method returns zero for the first field of a pair, and one for the second.
```

// As a special case, if the given field is part of a multi-field picture, the field offset is returned.

```
int MPEGMapper::GetFieldPairing(AvUnit_t SampleNum)
{
    const long searchLimit = 100;
    AvUnit_t ix = SampleNum;
    AvUnit_t fxOffset = GetFXOffset(ix);
    AvUnit_t origFXOffset = fxOffset;
    int pType = GetPictureType(ix);
    int nextPType;
    AvUnit_t nextOffset;

    if (SampleNum > _NumSamples-SampleNum)          // search backwards
    {
        while (SampleNum - ix < searchLimit && ix > 0)
        {
            ix--;
            nextPType = GetPictureType(ix);

            // if the ptypes are different then we know that ix is the second field of a pair,
            // unless the types are IP, which is ambiguous, so we continue (yes, I know this is suboptimal).
            if (pType != nextPType && (pType != MPEGPPicture || nextPType != MPEGIPicture))
                return (SampleNum - ix + 1) & 1;

            nextOffset = GetFXOffset(ix);

            // if there is ever a multi-field picture, then we know that the field we're on is even
```

```
                    if (nextOffset == fxOffset)
                    {
                        if (fxOffset == origFXOffset)                    // special case
                            return GetFieldOffset(SampleNum);
                        return (SampleNum - ix) & 1;
                    }

                    fxOffset = nextOffset;
                    pType = nextPType;
                }
            }
            else                                                         //
search forwards
            {
                while (ix - SampleNum < searchLimit)
                {
                    ix++;
                    nextPType = GetPictureType(ix);

                    if (pType != nextPType && (pType != MPEGIPicture || nextPType
!= MPEGPPicture))
                        return (ix - SampleNum) & 1;

                    nextOffset = GetFXOffset(ix);

                    if (nextOffset == fxOffset)
                    {
                        if (fxOffset == origFXOffset)                    // special case
                            return GetFieldOffset(SampleNum);
                        return (ix - 1 - SampleNum) & 1;
                    }
```

```
                fxOffset = nextOffset;
                pType = nextPType;
            }
        }

        return 0;           // unknown - guess and hope for the best
    }

    long MPEGMapper::GetLeaderLen(AvUnit_t dorderField)
    {
        AvUnit_t ix = dorderField;

        if (_NFields == 1)          // One field case is simpler, and two-field code may
not work for progressive sequence
        {
            u_char desiredPType = GetPictureType(ix);
            u_char pType = desiredPType;
            int nPPics = 0;

            while (ix > 0 && (pType != MPEGIPicture || (desiredPType ==
MPEGBPicture && nPPics == 0)))
            {
                ix--;
                pType = GetPictureType(ix);
                if (pType == MPEGPPicture)
                    nPPics++;
            }

            // continue to first field of the I-picture we just found
            ix -= GetFieldOffset(ix);
        }
```

```
else    // two-field case -- we need a reference field of each parity
{
        u_char fieldParity = 0;                 // initial setting is arbitrary since we
need one or two of each
        u_char nRefFields[2] = { 0, 0 };
        u_char nIFields[2] = { 0, 0 };
        u_char lastPType = GetPictureType(ix);
        int BCount = 0;
        int prevBCount = 0;
        int fieldPairing = GetFieldPairing(ix);

        if (lastPType != MPEGBPicture)
        {
                nRefFields[0] = nRefFields[1] = 2;              // don't bother
counting ref fields - only I's
                if (lastPType == MPEGIPicture)
                {
                        nIFields[0] = 1;
                        if (GetPictureType(ix+1) == MPEGIPicture)
                                nIFields[1] = 1;
                }
                // if we are going to scan, we need to know the parity of this field
relative to the preceding
                // which means we have to count B fields follwing this frame
                if (nIFields[1] == 0)
                {
                        AvUnit_t ix2 = ix + 1;

                        while (ix2 < _NumSamples && GetPictureType(ix2) ==
MPEGBPicture)
                                ix2++;
```

```
                prevBCount = ix2 - ix - 1;
            }
        }

        while (ix > 0 && (fieldPairing > 0 ||
            nIFields[0] == 0 || nIFields[1] == 0 || nRefFields[0] < 2 ||
nRefFields[1] < 2))
        {
            int pType;

            ix--;
            pType = GetPictureType(ix);

            if (pType == MPEGBPicture)
                BCount++;

            else    // I or P
            {
                if (lastPType == MPEGBPicture || fieldPairing < 0)
                {
                    fieldPairing = min(1, GetFieldOffset(ix)-1);
                    fieldParity = (fieldParity + prevBCount + 1) & 1;
                    prevBCount = BCount;
                    BCount = 0;
                }
                else
                {
                    fieldParity = (fieldParity + 1) & 1;
                    fieldPairing--;
                }

                nRefFields[fieldParity] ++;
```

```
                    if (pType == MPEGIPicture)
                        nIFields[fieldParity] ++;
                }
            }
        }

        return dorderField - ix;
    }

    // GetLeaderInfo returns all required information about the "leader", which is the
    // sequence of pictures that must be input to the decoder in order to get out a given
    // frame. The SampleNum input is the index of the desired frame. If the given
SampleNum
    // is not a B-picture, then there may be B-pictures following it that will come out first
    // and need to be discarded as well. The MPEGLeaderInfo_t contains this information as
well.

    // The algorithm is: if the given frame is an I-picture, the leader length is zero.
    // If the given frame is a P-picture, the leader extends to the preceding I-picture.
    // If the given frame is a B-picture, the leader extends to either the preceding I-picture
    // if there is a P-picture intervening, or the second preceding I-picture if there is no
    // P-picture intervening.

    void MPEGMapper::GetLeaderInfo(AvUnit_t SampleNum, AvUnit_t NumSamples,
                                   MPEGLeaderInfo_t* leaderInfo)
    {
        int i;
        AvUnit_t dorderFirstField = GetDOrderField(SampleNum, FALSE);
        int firstFieldOffset = GetFieldOffset(dorderFirstField);
        int leadingFields = GetLeaderLen(dorderFirstField) - firstFieldOffset;
        AvUnit_t startOfLeader = dorderFirstField - leadingFields;
        AvUnit_t ix;
```

```
AvUnit_t prevFXOffset;
AvUnit_t newFXOffset;
int pendingIPDiscards;
u_char pType;
int leadingDiscard = ((MPGIDescriptor*) _Desc)->GetLeadingDiscard() ? 1 : 0;
int dorderZero = -1;

// if we're playing more than one frame, then we read and discard any B-pictures
following
// an initial I or P
if (GetPictureType(dorderFirstField) != MPEGBPicture && NumSamples >
_NFields)
{
    AvUnit_t nextPic = FindNextPicture(dorderFirstField);

    // Scan for following B-pictures, if we need any to play the desired range
    if (nextPic - dorderFirstField < NumSamples * _NFields)
    {
        AvUnit_t ix2 = nextPic;

        while (ix2 < _NumSamples && GetPictureType(ix2) ==
MPEGBPicture)
            ix2++;

        if (ix2 > nextPic)
            leadingFields = ix2 - startOfLeader;  // includes actual first
picture in this case
    }
}

// discard any initial fields output from the first picture that we don't need
// we count the rest of the discards below
```

```
leaderInfo->leadingDiscardFields = firstFieldOffset;

// add in an extra field if we are playing from start of clip and clip starts with
bottom field
if (SampleNum == 0)
        leaderInfo->leadingDiscardFields += leadingDiscard;
else if (startOfLeader <= 3 && leadingDiscard)
        dorderZero = GetDOrderField(0, FALSE);

pendingIPDiscards = 0;

// now build the frameIndexInfo list

i = 0;
ix = startOfLeader;
pType = MPEGIPicture;
leaderInfo->frameIndexInfo[0].nFields = 0;
prevFXOffset = newFXOffset = GetFXOffset(startOfLeader);

while (TRUE)
{
        if (newFXOffset == prevFXOffset)
        {
                leaderInfo->frameIndexInfo[i].nFields++;
        }
        else
        {
                leaderInfo->frameIndexInfo[i].pictureType = pType;
                leaderInfo->frameIndexInfo[i].pictureLength = newFXOffset -
prevFXOffset;

                if (pType == MPEGBPicture)
```

```
                                leaderInfo->leadingDiscardFields +=
leaderInfo->frameIndexInfo[i].nFields;
                            else
                                pendingIPDiscards =
leaderInfo->frameIndexInfo[i].nFields;

                            pType = GetPictureType(ix);
                            if (pType != MPEGBPicture)
                                leaderInfo->leadingDiscardFields += pendingIPDiscards;

                            i++;
                            leaderInfo->frameIndexInfo[i].nFields = 1;
                        }

                        if (ix >= startOfLeader+leadingFields)
                            break;

                        if (ix == dorderZero)
                            leaderInfo->frameIndexInfo[i].nFields += leadingDiscard;

                        ix++;
                        prevFXOffset = newFXOffset;
                        newFXOffset = GetFXOffset(ix);
                    }

                leaderInfo->leaderLength = i;
            }

            // FindNextPicture: given a disk-order FX position, return the FX position of the next
disk-order
            // picture in the index
```

```
AvUnit_t MPEGMapper::FindNextPicture(AvUnit_t ix)
{
    AvUnit_t fxOffset = GetFXOffset(ix);

    while (++ix < _NumSamples && GetFXOffset(ix) == fxOffset) {}

    return ix;
}

// GetEndOfRange returns the offset of the first picture following the range that does
// not need to be read.from the file in order to contain all of the frames in the given range.
// There are some tricky parts:
// (1) if the last temporal picture is I or P then some number of B pictures
// following it may be included in the range (either all or none, actually). And
// (2) the frame may cross picture boundaries, as indicated by field offsets, and
// (3) the next disk order frame may be part of the same picture, so that we have to
// look further to find the frame index entry corresponding to the next disk-order picture

AvUnit_t MPEGMapper::GetEndOfRange(AvUnit_t SampleNum, AvUnit_t
NumSamples)
{
    AvUnit_t dorderLastSample = GetDOrderField(SampleNum + NumSamples - 1,
TRUE);
    int pType = GetPictureType(dorderLastSample);
    AvUnit_t nextPict = FindNextPicture(dorderLastSample);

    if (pType != MPEGBPicture && NumSamples * _NFields > nextPict -
dorderLastSample)
    {
        while (nextPict < _NumSamples && GetPictureType(nextPict) ==
MPEGBPicture)
            nextPict++;
```

```
        }

        return GetFXOffset(nextPict);
    }


    // GetTrailingDiscards returns the number of fields that will be output from a decoder following
    // play of the frame at SampleNum. This includes two components: (1) if the last field to be played
    // comes from a B-picture, then the preceding I or P picture will come out with as many fields as it
    // is supposed to produce, and (2) the picture the produces the last field may produce more fields than
    // desired to be played.

    int MPEGMapper::GetTrailingDiscards(AvUnit_t SampleNum)
    {
        AvUnit_t dorderLastSample = GetDOrderField(SampleNum, TRUE);
        int pType = GetPictureType(dorderLastSample);
        int result = 0;
        AvUnit_t ix;
        AvUnit_t lastDOrderField;

        if (pType == MPEGBPicture)
        {
            // find the preceding I or P
            ix = dorderLastSample - 1;
            while (ix > 0 && GetPictureType(ix) == MPEGBPicture)
                ix--;

            // now count its fields (there will always be at least two, by the pairing
```

```
rule)
                result += 1 + min(1, GetFieldOffset(ix));

                lastDOrderField = ix;
        }
        else
                lastDOrderField = FindNextPicture(dorderLastSample) - 1;

        // now count any extra fields in the last picture
        result += lastDOrderField - dorderLastSample;

        // if last picture is also last in clip, there may be one more
        // the reason for the extra funny test is to avoid moving the FX cache to the end if
we are nowhere
        // near the end
        if (((MPGIDescriptor*) _Desc)->GetTrailingDiscard() &&
(_NumSamples-lastDOrderField < 256) &&
                lastDOrderField == GetDOrderField(_NumSamples/_NFields-1, TRUE))
                result++;

        return result;
    }

    void MPEGMapper::SetSampleOffset(AvUnit_t SampleNum, AvUnit_t Offset) {
        DoesNotImplement();
    }

    void MPEGMapper::WriteFrameIndex(void)
    {
        DoesNotImplement();
    }
void
MPEGMapper::SetSampleSize(AvUnit_t NumSamples,long SampleSize)
{
    DoesNotImplement();
}
```

```
/*
 * /----------------------------------------------------------------\
 * | The following programs are the sole property of Avid Technology, Inc., |
 * | and contain its proprietary and confidential information.                |
 * |      Copyright 1989-1996 Avid Technology Inc.
 *  |
 * \----------------------------------------------------------------/
 */

#ifndef _MPEG_MAPPER_H
#define _MPEG_MAPPER_H

/*****************************************************************************
*******
 MPEGMapper.h
  -
 MPEGMapper class and function definitions


*****************************************************************************
*/

#include "DIDMapper.h"
#include "MPEGDefs.h"

class MPGIDescriptor;

typedef struct {
        char        toDoDelta;          // temporal order to disk order delta (signed)
        u_char      flags;
        u_short             offsetHigh;
        u_long              offsetLow;
```

```
} MPEGFrameIndexEntry;

// Content of flags:
#define MPEGPictureTypeMask 0x0003
#define MPEGRandomAccessBit 0x0004
#define MPEGSequenceHdrBit  0x0008

class MPEGMapper: public DIDMapper
{
  OBJECT_STD_H(MPEGMapper)


 public:
      MPEGMapper(void);          // OBJECT_STD_C requires this, but don't use it
  MPEGMapper(ameBaseStream *s, DIDDescriptor* desc, AvUnit_t NumSamples,
                        long SampleSize, Boolean isfixedsize);

  virtual APosition* MapSample(AvUnit_t SampleNum);
  virtual AvUnit_t BufferSize(AvUnit_t BeginSample, AvUnit_t NumSamples);

  virtual void SetSampleOffset(long SampleNum, long Offset);
  virtual long GetSampleOffset(long SampleNum);

      virtual void WriteFrameIndex(void);
      virtual void SetSampleSize(AvUnit_t NumSamples, long SampleSize);

      // the following are "private" methods used either internally, or only by the
MPEGReader
                        void   GetBOBInfo(AvUnit_t BeginSample, AvUnit_t
NumSamples,
                                                AvUnit_t* offset, AvUnit_t*
length, Boolean* needSeqHdr);
```

```
                    void    GetLeaderInfo(AvUnit_t SampleNum, AvUnit_t
NumSamples,
                                                    MPEGLeaderInfo_t*
leaderInfo);
                    AvUnit_t GetEndOfRange(AvUnit_t SampleNum, AvUnit_t
NumSamples);
                    int             GetTrailingDiscards(AvUnit_t SampleNum);
                    AvUnit_t GetDOrderField(AvUnit_t SampleNum, Boolean
lastField);

                    // the following all operate on field position (normally temporal)
not frame (sample) number
                    int             GetPictureType(AvUnit_t dorderField);
                    Boolean         HaveSequenceHdr(AvUnit_t dorderField);

        protected:  // these really are private
                    int             GetFieldOffset(AvUnit_t dorderField);
                    long    GetLeaderLen(AvUnit_t dorderField);          //
SampleNum is disk-order
                    AvUnit_t GetFXOffset(AvUnit_t dorderField);
                    AvUnit_t FindNextPicture(AvUnit_t ix);
                    int             GetFieldPairing(AvUnit_t SampleNum);

            int             _NFields;
    };

    #endif // _MPEG_MAPPER_H
```

【図１】

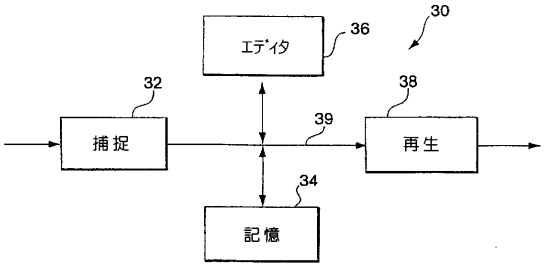

Fig. 1

【図２】

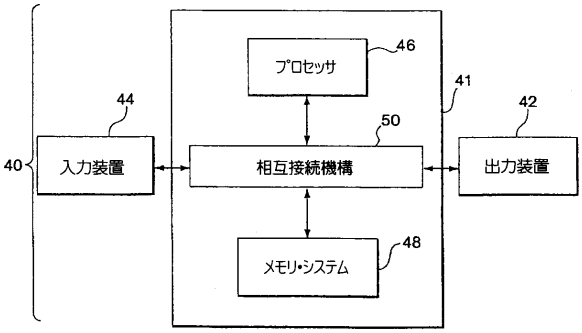

Fig. 2

【図３】

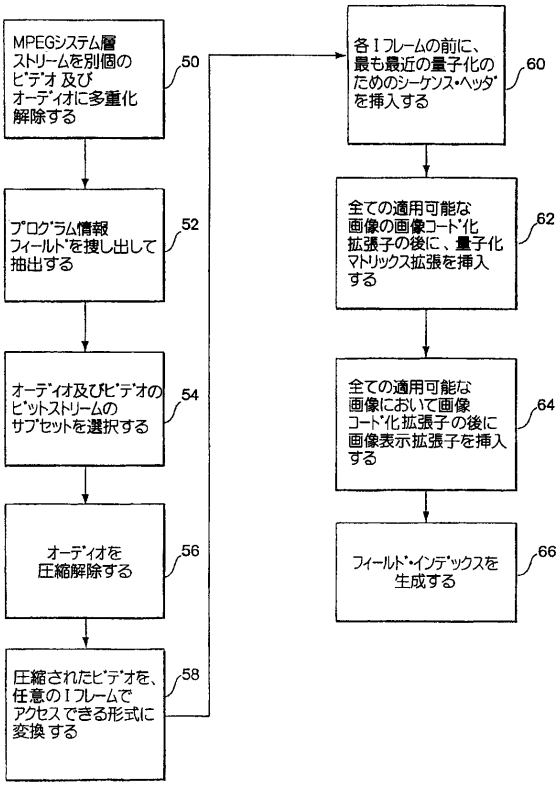

Fig. 3

【図４】

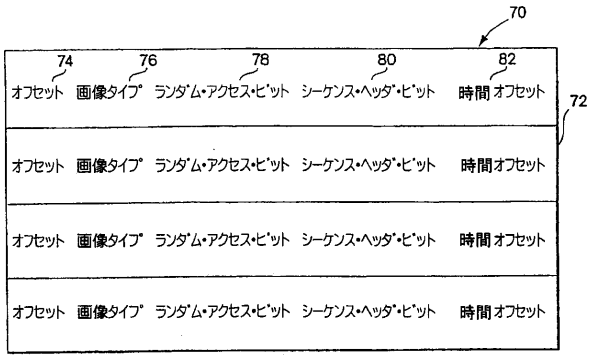

Fig. 4

【図６】

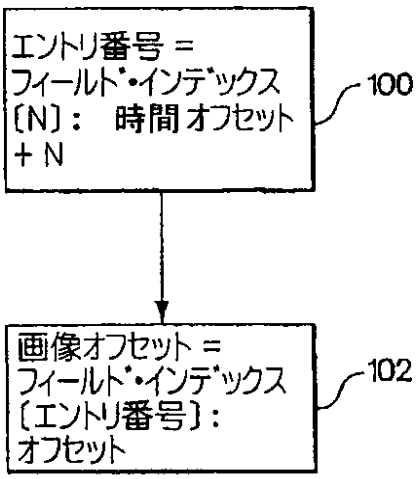

Fig. 6

【図５】

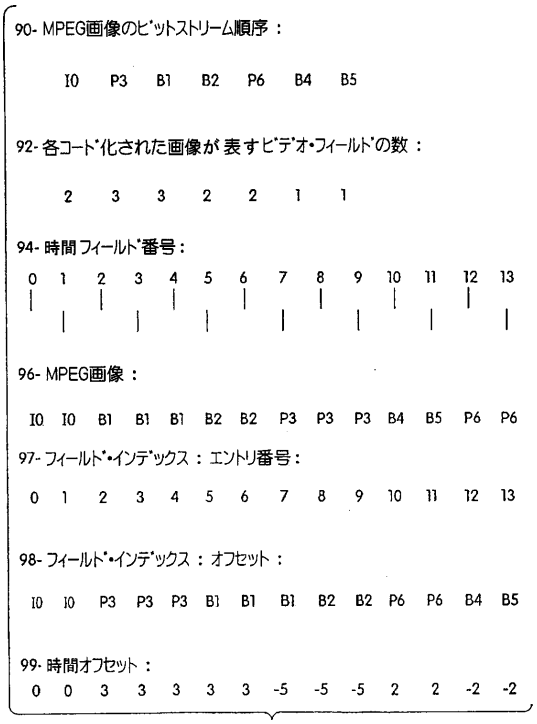

Fig. 5

【図７】

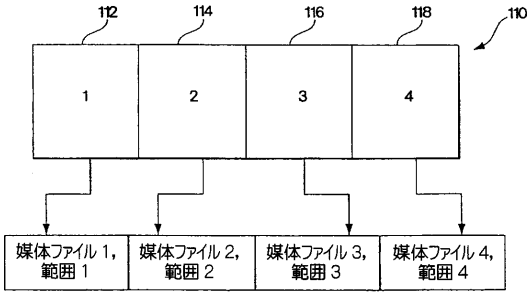

Fig. 7

【図８】

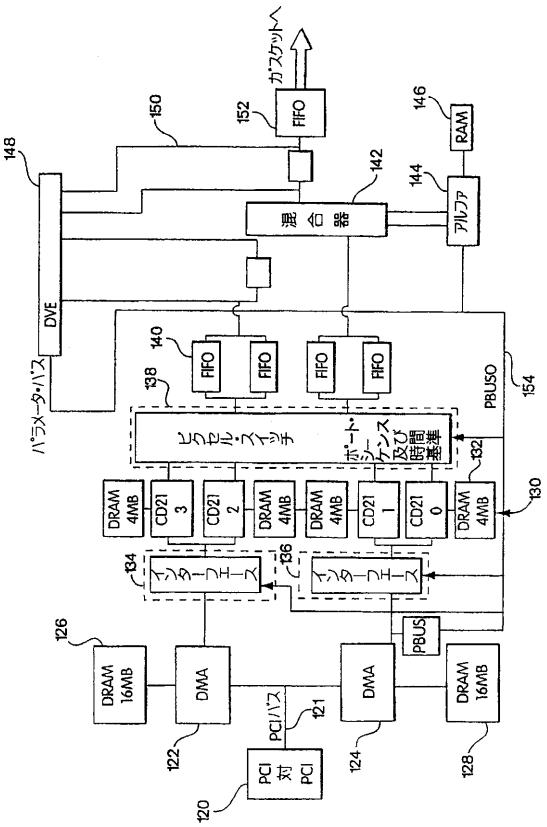

Fig. 8

【図９】

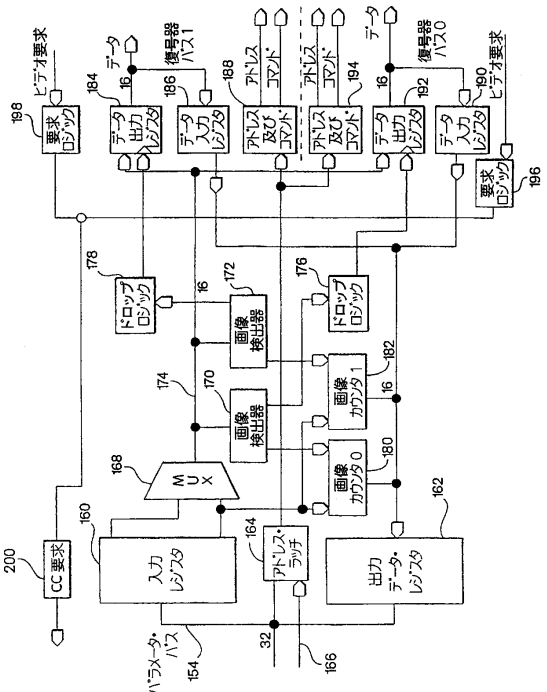

Fig. 9

【図１０】

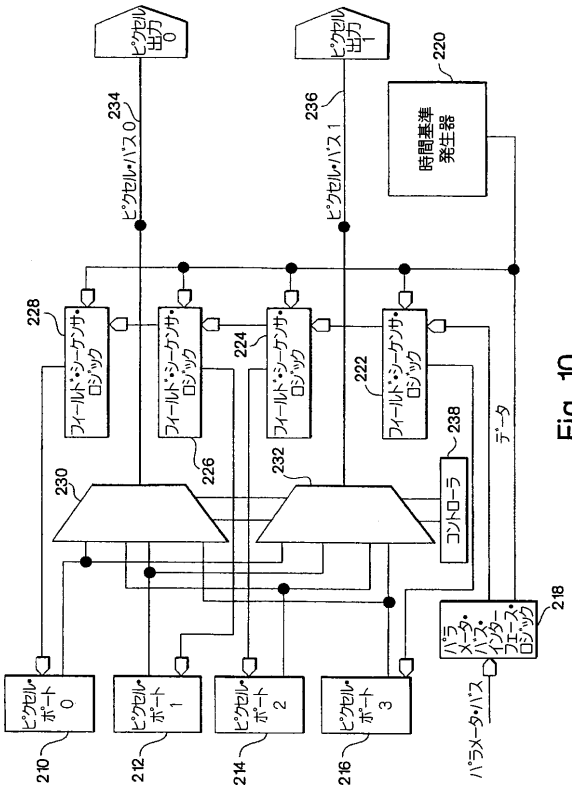

Fig. 10

【図１１】

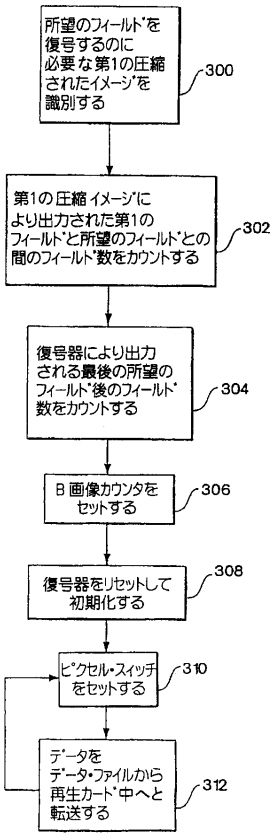

Fig. 11

フロントページの続き

(74)代理人
弁理士　小林　泰
(72)発明者　スポラー，マイケル
アメリカ合衆国マサチューセッツ州０２１８１，ウェルズリー，ロングフェロー・ロード　３１
(72)発明者　コーノグ，カサリーン・エイチ
アメリカ合衆国マサチューセッツ州０１９５０，ニューバリーポート，チェスナット・ストリート　２６
(72)発明者　ザヴォイスキ，ピーター
アメリカ合衆国ニューハンプシャー州０３０５４，メリーマック，パッカード・ドライブ　３２
(72)発明者　ハミルトン，ジェームズ
アメリカ合衆国カリフォルニア州９４０６２，レッドウッド・シティ，ヒルクレスト・ウェイ　６８４

審査官　木方　庸輔

(56)参考文献　特開平０６－０６２３６９（ＪＰ，Ａ）
特開平０７－０４６４６２（ＪＰ，Ａ）
特開平０８－１１１８４３（ＪＰ，Ａ）
特開平０８－２１４２６４（ＪＰ，Ａ）
国際公開第９５／０２３４１１（WO，Ａ１）
特開平０６－１６４５２２（ＪＰ，Ａ）
特開平０６－２６７１９６（ＪＰ，Ａ）
特開平０６－３２５５５３（ＪＰ，Ａ）
特開平０６－０９８３１４（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
H04N　5/76 - 5/956
H04N　7/12 - 7/137