(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 601 10 493 T2** 2006.01.05

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 154 575 B1**
(21) Deutsches Aktenzeichen: **601 10 493.5**
(96) Europäisches Aktenzeichen: **01 102 700.0**
(96) Europäischer Anmeldetag: **07.02.2001**
(97) Erstveröffentlichung durch das EPA: **14.11.2001**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **04.05.2005**
(47) Veröffentlichungstag im Patentblatt: **05.01.2006**

(51) Int Cl.[8]: ***H03M 7/40*** (2006.01)

(30) Unionspriorität:
565015 **04.05.2000** **US**

(73) Patentinhaber:
Hewlett-Packard Development Co., L.P., Houston, Tex., US

(74) Vertreter:
Schoppe, Zimmermann, Stöckeler & Zinkler, 82049 Pullach

(84) Benannte Vertragsstaaten:
DE, FR, GB

(72) Erfinder:
Crane, Randy T., Fort Collins, US

(54) Bezeichnung: **Entropie Kodierer/Dekodierer zur schnellen Datenkompression und -dekompression**

Anmerkung: Innerhalb von neun Monaten nach der Bekanntmachung des Hinweises auf die Erteilung des europäischen Patents kann jedermann beim Europäischen Patentamt gegen das erteilte europäische Patent Einspruch einlegen. Der Einspruch ist schriftlich einzureichen und zu begründen. Er gilt erst als eingelegt, wenn die Einspruchsgebühr entrichtet worden ist (Art. 99 (1) Europäisches Patentübereinkommen).

Die Übersetzung ist gemäß Artikel II § 3 Abs. 1 IntPatÜG 1991 vom Patentinhaber eingereicht worden. Sie wurde vom Deutschen Patent- und Markenamt inhaltlich nicht geprüft.

Beschreibung

Gebiet der Erfindung

[0001] Die vorliegende Erfindung bezieht sich allgemein auf einen Entropiecodierer/decodierer für eine schnelle Datenkomprimierung und -dekomprimierung. Genauer gesagt bezieht sich die vorliegende Erfindung auf einen Entropiecodierer/decodierer, der mit jedem Komprimierungsschema verwendet werden kann, das einen Entropiecodierungsschritt umfasst.

Hintergrund der Erfindung

[0002] Bei Datenübertragungs- und Verarbeitungsanwendungen ist es üblich, dass Daten vor oder während der Verarbeitung oder Übertragung von Daten gemäß verschiedenen Komprimierungsalgorithmen komprimiert werden. Außerdem ist es üblich, dass komprimierte (codierte) Daten während oder nach der Verarbeitung oder Übertragung decodiert werden, um die codierten Daten zurück zur ursprünglichen Form umzuwandeln.

[0003] Einige übliche Komprimierungsschemata (oder Algorithmen) umfassen, was als Entropiecodierungsschritt bekannt ist. Beispiele dieser übliche Algorithmen, die einen Entropiecodierungsschritt umfassen, umfassen LZW, verlustfreie JPEG, G3, G4 usw. Komprimierungsschemata, die einen Entropiecodierungsschritt umfassen, erzeugen typischerweise einen Ausgabebitstrom, der eine variable Länge hat. Aufgrund der variablen Länge der codierten Ausgabe erfordert das Verarbeiten dieser Daten eine große Menge an Rechnungsaufwand auf Seiten der Verarbeitungshardware und des zentralen Prozessors oder der zentralen Steuerung.

[0004] Typische Komprimierungshardware ist typischerweise zweckgebunden zum Verarbeiten/Codieren gemäß nur einem vordefinierten Komprimierungsalgorithmus. Falls Daten, die unter Verwendung verschiedener Komprimierungsalgorithmen codiert wurden, verarbeitet oder übertragen werden sollen, ist es aufgrund dieser Beschränkung notwendig, dass mehrere Hardwareimplementierungen vorgesehen sind, um jedes der verfügbaren Komprimierungsalgorithmen/-formate unterzubringen. Dies erhöht die Kosten in Zusammenhang mit der Verarbeitung oder Übertragung von Daten, die gemäß mehreren Komprimierungsalgorithmen codiert sind.

[0005] Beispiele bekannter Systeme und Techniken zum Komprimieren von Daten wurden in dem US Patent 5,499,382 an Nusinov u.a. erörtert, für eine Schaltung und ein Verfahren zum Bit-Verdichten und Bit-Entpacken unter Verwendung eines Barrel-Schiebers, und in dem US-Patent 4,360,840 an Wolrum für ein Echtzeitdaten-Komprimierungs-/Dekomprimierungsschema für Facsimile-Übertragungssysteme.

[0006] Es ist die Aufgabe der vorliegenden Erfindung, einen Codierer/Decodierer zu schaffen, der mit jedem Komprimierungsschema verwendet werden kann, das einen Entropiecodierungsschritt umfasst.

[0007] Diese Aufgabe wird durch einen Codierer/Decodierer gemäß Anspruch 1 gelöst.

[0008] Die vorliegende Erfindung liefert ein System zum Codieren und Decodieren von Informationen.

[0009] Kurz gesagt kann das System bezüglich der Architektur wie folgt implementiert werden. Es ist ein Codierer zum Codieren von Daten vorgesehen, der ein Datenregister zum Empfangen und Speichern eines Codeworts variabler Länge, ein Bitstromregister zum Empfangen von Daten, einen Multiplexer zum Laden gültiger Bits von dem Steuerregister in die höchstwertigsten Bits, die in dem Bitstromregister verfügbar sind, ein Zuerst-Hinein-Zuerst-Hinaus-(FIFO-)Register zum Empfangen der Inhalte des Bitstromregisters, wenn alle verfügbaren Bits des Bitstromregisters mit gültigen Datenbits geladen sind, und eine Unterbrechungssteuerung zum Erzeugen eines Unterbrechungssignals umfasst, um eine Auslesung von Daten von dem FIFO-Register einzuleiten.

[0010] Bei einem weiteren Ausführungsbeispiel der vorliegenden Erfindung ist ein Decodierer zum Decodieren von Daten vorgesehen. Dieser Decodierer umfasst ein erstes Register zum Empfangen codierter Datenwortdaten fester Länge, ein Bitstromregister zum Empfangen des codierten Datenworts fester Länge, einen Multiplexer zum Laden Codewortdaten variabler Länge von dem Bitstrompuffer in ein Datenregister, und eine Unterbrechungssteuerung zum Erzeugen eines Unterbrechungssignals zum Einleiten des Schreibens codierter Daten fester Länge in das erste Register.

[0011] Die vorliegende Erfindung kann auch so gesehen werden, dass sie ein Verfahren zum Codieren liefert. Diesbezüglich kann das Verfahren durch die folgenden Schritte grob zusammengefasst werden: Empfangen von Codewortdaten variabler Länge, Bestimmen der Anzahl gültiger Bits der Codewortdaten, Laden der Codewortdaten in einen Bitstrompuffer, falls alle gültigen Bits passen. Falls nicht alle gültigen Bits in den Bitstrompuffer passen, Laden eines ersten Segments der gültigen Bits in den Bitstrompuffer und dann Laden der Inhalte des Bitstromregisters in ein FIFO-Register, und Laden eines zweiten Segments der teilweise gültigen Bits in den Bitstrompuffer.

[0012] Ein weiteres Verfahren zum Decodieren von Daten ist vorgesehen, das durch die folgenden Schritte grob zusammengefasst werden kann: Emp-

fangen eines Datenworts, Laden des Datenworts in einen Puffer, Auslesen eines Codeworts variabler Länge von dem Datenwort; und Laden des Codeworts variabler Länge in ein Register fester Länge.

[0013] Andere Systeme, Verfahren, Merkmale und Vorteile der vorliegenden Erfindung werden für einen Fachmann auf diesem Gebiet bei der Untersuchung der folgenden Zeichnungen und der detaillierten Beschreibung offensichtlich. Es ist beabsichtigt, dass alle zusätzlichen Systeme, Verfahren, Merkmale und Vorteile, die in dieser Beschreibung enthalten sind, innerhalb des Schutzbereichs der vorliegenden Erfindung liegen und durch die angehängten Ansprüche geschützt sind.

Kurze Beschreibung der Zeichnungen

[0014] Die Erfindung ist besser verständlich mit Bezugnahme auf die folgenden Zeichnungen. Die Komponenten in den Zeichnungen sind nicht notwendigerweise maßstabsgerecht, stattdessen wurde der Schwerpunkt darauf gelegt, die Prinzipien der vorliegenden Erfindung deutlich darzustellen. Darüber hinaus bezeichnen in den Zeichnungen gleiche Bezugszeichen entsprechende Teile in den mehreren Ansichten.

[0015] Fig. 1 ist ein Diagramm, das ein System darstellt, das den CODEC der vorliegenden Erfindung umfasst;

[0016] Fig. 2 ist ein Blockdiagramm, das den CODEC der vorliegenden Erfindung näher darstellt;

[0017] Fig. 3 ist ein Diagramm, das ein Steuerregister darstellt;

[0018] Fig. 4 ist ein Diagramm, das ein FIFO-Pegelregister darstellt;

[0019] Fig. 5 ist ein Flussdiagramm, das das Codierverfahren der vorliegenden Erfindung darstellt;

[0020] Fig. 6 ist ein Diagramm, das das Decodierverfahren der vorliegenden Erfindung darstellt; und

[0021] Fig. 7 ist ein Diagramm, das den Prozess des Ladens/Entladens von Datenwörtern variabler Länge in den Bitstrompuffer **150** darstellt.

Detaillierte Beschreibung des bevorzugten Ausführungsbeispiels

[0022] Die vorliegende Erfindung bezieht sich auf einen Codierer und Decodierer (CODEC), der mit im Wesentlichen jedem Komprimierungsschema verwendet werden kann, das einen Entropiecodierungsschritt umfasst. Ferner liefert die vorliegende Erfindung Daten-FIFO und Barrel-Schieber, die verwendet werden können, um Daten entweder während der Codier- oder Decodieroperation zu verarbeiten.

[0023] Fig. 1 stellt ein System dar, dass den Entropie-CODEC der vorliegenden Erfindung umfasst. Eine zentrale Verarbeitungseinheit (CPU) **10** ist vorgesehen, die über eine lokale Schnittstelle **102** eine Schnittstelle mit dem CODEC **1** bildet.

[0024] Mit Bezugnahme auf Fig. 2 ist ein Ausführungsbeispiel des Entropie-CODEC **1** der vorliegenden Erfindung dargestellt. Der CODEC **1** arbeitet auf zwei Weisen: Codiermodus und Decodiermodus. In Fig. 2 ist eine Registerschnittstelle **100**, ein Registerblock **110** und eine lokale Schnittstelle **120** gezeigt. Die Registerschnittstelle **100** bildet eine Schnittstelle mit der CPU **10** zum Steuern des Eingangs und Lesen von Daten in/von den Registern des Registerblocks **110**. Ein Zähler **170** ist vorgesehen zum Zählen des Pegels/der Anzahl von Datenwörtern, die zu einem Zeitpunkt in dem FIFO **160** gespeichert sind. Der Zähler **170** liefert eine Eingabe zu der Unterbrechungssteuerung **180**, wenn der Pegel/die Anzahl von Datenwörtern, die in dem FIFO **160** gespeichert sind, einen Wert erreicht, der dem Zähler **170** entspricht.

[0025] Der Registerblock **110** umfasst ein Bits-Legen-Längen-Register **110A**, ein Bits-Legen-Coderegister **110B**, ein FIFO-Pegelregister **110C**, ein Rücksetzregister **110D**, ein Steuerregister **110E**, ein Flush-Zuerst-Hinein-Zuerst-Hinaus (FLUSH-FIFO) Register **110F**, ein Bits-Holen-Register **110G**, ein Bitzeigerregister **110H** und ein Gepackte-Ausgabewörter-Register **110I**. Außerdem sind eine Steuerung **130**, die den Betrieb eines Multiplexers **140** steuert, ein Bitstrompuffer **150** und ein Zuerst-Hinein-Zuerst-Hinaus-Registerblock (FIFO) **160** gezeigt. Die Steuerung **130** arbeitet gemäß Befehlen von der CPU **10**. Der FIFO **160** ist beispielsweise ein 32 Bit × 16 Wort FIFO-Registerblock. Der Multiplexer **140** ist mit der lokalen Schnittstelle **120** verbunden, über die Daten zu und von dem Registerblock **110** übertragen werden.

[0026] Das Rücksetzregister **110D** ist ein Nur-Schreibe-Register, das verwendet wird, um die Hardware, einschließlich allen Zeigern und des FIFO **160**, der vorliegenden Erfindung 1 zurückzusetzen. Das Steuerregister **110E** speichert Daten, die die Steuerbits darstellen, wie es in Fig. 3 dargestellt ist. Mit Bezugnahme auf Fig. 3 ist ersichtlich, dass das Steuerregister **110E** so konfiguriert werden kann, dass beispielsweise die Bits 0–4 einen vordefinierten Wert oder einen FIFO-Unterbrechungspegel darstellen, der einen maximalen oder minimalen Pegel/Anzahl von Datenwörtern darstellt, die in dem FIFO **160** gespeichert werden dürfen, abhängig von der Funktionsweise. Das Bit 5 des Steuerregisters **110E** ist das Codier/Decodierbit (EN/DEC). Der Wert des

EN/DEC-Bits zeigt an, ob der CODEC **1** Daten von dem FIFO **160** decodiert oder Daten in den FIFO **160** codiert. Wenn das EN/DEC-Bit beispielsweise 0 ist, codiert der CODEC **1**. Dies wird auch als Codiermodus bezeichnet. Wenn das EN/DEC-Bit beispielsweise 1 ist, decodiert der CODEC **1**. Dies wird auch als Decodiermodus bezeichnet. Das Bit 6 des Steuerregisters **110E** stellt das Unterbrechungsfreigabebit dar. Das Unterbrechungsfreigabe-(IE)-Bit kann beispielsweise eine 1 sein (hoch), was es dem CODEC **1** ermöglicht, die CPU **10** zu unterbrechen. Die verbleibenden Bits des Steuerregisters **110E** werden verwendet, um Codewortdaten variabler Länge zu sammeln. Diese Codewortdaten werden nachfolgend von dem Steuerregister **110E** ausgelesen und über den Multiplexer **140** zu dem Bitstrompuffer **150** geleitet. Alle Bits in dem Steuerregister **110** sind bei einer Zurücksetzung auf 0 voreingestellt.

[0027] Ein Bitzeigerregister **110H** ist vorgesehen. Das Bitzeigerregister **110H** ist vorzugsweise ein Nur-Lese-Register, das während dem Codiermodus Daten speichert, die einen Zeigerwert reflektieren, der das nächst verfügbare höchstwertigste Bit (MSB) in dem Bitstromregister **150** anzeigt, das mit Daten geladen werden kann. Wo der Bitstrompuffer **150** beispielsweise 32 Bits lang ist, kann das Bitstromzeigerregister zu einem von 32 Bits, Bit 0 bis Bit 31, zeigen. Wenn der Bitstrompuffer **150** voll ist, zeigt der Bitstromzeiger zu dem Bit 0, was anzeigt, dass der Bitstrompuffer **150** voll ist. Falls das Bitstromzeigerregister einen Wert zwischen 1 und 31 anzeigt, hat der Bitstrompuffer **150** Bits verfügbar, um eine Dateneingabe anzunehmen. Falls das Bitzeigerregister **110H** beispielsweise einen Wert von 28 anzeigt, dann ist es möglich, zusätzliche Daten in den Bitstrompuffer **150** zu laden, beginnend mit dem nächsten MSB 29 des Bitstrompuffers **150**. Kurz gesagt, der Wert in dem Bitzeiger **110H** spezifiziert das MSB-Bit in dem Bitstrompuffer **150**, in das Daten geladen werden können.

[0028] Das Flush-FIFO-Register **110F** kann adressiert werden, um zu bewirken, dass Daten, die in den Bitstrompuffer **150** gespeichert sind, in den FIFO **160** geschrieben werden. Vorzugsweise sollten die Inhalte des Bitzeigerregisters **110H** ausgelesen werden, bevor in das FLUSH-FIFO-Register **110F** geschrieben wird.

[0029] Ein Beispiel eines FIFO-Pegelregisters **110C** ist in Fig. 4 dargestellt. Hier ist ersichtlich, dass die Bits 0–4 verwendet werden, um den FIFO-Pegel zu speichern. Der FIFO-Pegel ist ein Wert, der die Anzahl von Datenwörtern darstellt, die in den FIFO **160** geladen werden können, bevor die Inhalte des FIFO **160** ausgelesen werden, um Platz für zusätzliche Datenwörter zu machen. Der FIFO-Pegelregister **110C** kann sowohl während dem Codier- als auch dem Decodiermodus gelesen werden. Ein Bits-Legen-Coderegister **110B** ist zum Speichern von Daten vorgesehen, die das Codewort der nächsten variablen Bitlänge darstellen, das ausgegeben werden soll oder zu der Ausgabedatenzeichenfolge geschrieben werden soll. Daten, die in das Bits-Legen-Register **110B** geschrieben sind, sind vorzugsweise rechts ausgerichtet. Es ist auch ein Bits-Legen-Längenregister **110A** vorgesehen, das Daten speichert, die die Anzahl von Bits des Bits-Legen-Coderegisters **110B** anzeigt, die in die Ausgabedatenzeichenfolge geschrieben werden soll. Daten, die in das Bits-Legen-Längenregister **110A** geschrieben werden, bewirken, dass der CODEC **1** tatsächlich die Codedaten schreibt, die in dem Bits-Legen-Coderegister **110B** gespeichert sind. Bei einem bevorzugten Ausführungsbeispiel werden Daten zuerst in das Bits-Legen-Coderegister **110B** geschrieben, gefolgt vom Schreiben von Daten in das Bits-Legen-Längenregister **110A**.

[0030] Das Bits-Holen-Register **110G** wird während dem Decodiermodus verwendet, um Codes variabler Bitlänge von dem Eingangsdatenstrom zu extrahieren. Wo es beispielsweise gewünscht wird, dass die nächsten fünf Bits von dem Dateneingangsstrom gelesen werden sollen, wird ein Wert von 5 in das Bits-Holen-Register **110G** geschrieben. Die fünf niedrigwertigsten Bits des Bits-Holen-Registers **110G** enthalten dann das Codewort.

[0031] Ein Gepacktes-Ausgabewort-Register **110I** ist vorgesehen. Das Gepacktes-Ausgabewort-Register **110I** kann beispielsweise über einen Blocklesebefehl einer Steuerung oder eines Zentralprozessors (CPU) zugegriffen/adressiert werden. Durch Lesen der Daten, die in den adressierten Räumen des Gepacktes-Ausgabewort-Registers **110I** enthalten sind, wird das nächste komprimierte Datenwort von dem FIFO **160** zu der CPU **10** ausgegeben.

[0032] Es wird angemerkt, dass während dem Codiermodus der Datenfluss im Allgemeinen von dem Multiplexer **140** zu dem Bitstrompuffer **150** zu dem FIFO **160** fließt. In dem Decodiermodus fließt der Datenfluss im Allgemeinen von dem FIFO **160** zu dem Bitstrompuffer **150** zu dem Multiplexer **140**.

CODIERMODUS

[0033] Im Codiermodus werden Daten in das Steuerregister **110E** geschrieben, um den FIFO-Unterbrechungspegel und das Unterbrechungs-Freigabebit zu setzen. Das Codier/Decodierbit wird auf Codieren gesetzt. Daten, die zu codieren sind, werden in das Bits-Legen-Coderegister **110B** geladen. Diese Daten bestehen aus Datenwörtern variabler Länge. Die Steuerung **130** bewirkt, dass der Multiplexer **140** die gültigen Bits von dem Bits-Legen-Coderegister **110B** auswählt, zum Lesen der gültigen Bits von dem Datenwort fester Länge in den Bitstrompuffer **150**.

[0034] Sobald alle Bits des Bitstrompuffers **150** geladen sind, werden die Inhalte derselben in das Zuerst-Hinein-Zuerst-Hinaus-(FIFO)Register **160** verschoben. Der FIFO-Registerblock **160** ist einem Zähler **170** zugeordnet, der die Anzahl von 32 Bitwörtern zählt, die zu jedem bestimmten Zeitpunkt in dem FIFO-Register **160** enthalten sind. Der Zählwert des Zählers **170** wird in dem FIFO-Pegelregister **110C** als FIFO-Pegeldaten gespeichert. Diese Daten können durch die CPU **10** verwendet werden, um zu bestimmen, ob die Daten in dem FIFO **160** gelesen oder geschrieben werden oder nicht. Beispielsweise könnte die CPU **10** das FIFO-Pegelregister abfragen, und wenn die FIFO-Pegeldaten, die in dem FIFO-Pegelregister **160** gespeichert sind, den FIFO-Unterbrechungspegeldaten entsprechen, die in dem Steuerregister **110E** gespeichert sind, abhängig von der Funktionsweise, könnte die CPU bewirken, dass Daten von dem FIFO **160** gelesen werden oder in denselben geschrieben werden. Daten können auch von dem FIFO **160** gelesen werden oder in denselben geschrieben werden, wo der Wert des Zählers **170** den Inhalten der FIFO-Unterbrechungspegeldaten entspricht, die in dem Steuerregister **110E** gespeichert sind. In diesem Fall bewirkt die Unterbrechungssteuerung **180**, dass ein Unterbrechungssignal an die CPU **10** gerichtet wird, um anzuzeigen, dass es für die CPU Zeit ist, die Inhalte von/in dem FIFO-Register **160** zu lesen (während dem Codiermodus) oder zu schreiben (während dem Decodiermodus).

[0035] Fig. 5 zeigt ein Flussdiagramm, das das Codierverfahren der vorliegenden Erfindung darstellt. Mit Bezugnahme auf Fig. 2 und Fig. 5 wird angemerkt, dass das Bits-Legen-Längenregister **110A** adressiert wird, um zu bewirken, dass ein Codewort beispielsweise von dem Bits-Legen-Coderegister **110B** geladen wird (**500**). Das Codewort hat eine variable Länge und kann das Ergebnis eines Entropiecodierungsprozesses sein. Das Bits-Legen-Coderegister **110B** hat eine feste Anzahl von Bits (Länge). Die Länge des Codeworts variabler Länge ist in dem Bits-Legen-Längenregister **110A** gespeichert. Die Daten des Codeworts variabler Länge, die in das Bits-Legen-Coderegister **110B** geladen sind, erfordern eventuell nicht alle verfügbaren Speicherbits des Bits-Legen-Coderegisters **110B**. Sobald die Daten des Codeworts variabler Länge in das Bits-Legen-Coderegister **110B** geladen sind, können dieselben alle verfügbaren Bits des Bits-Legen-Längenregisters **110A** besetzen oder nicht. Das Bits-Legen-Coderegister **110B** ist beispielsweise 16 Bits lang und ein Codewort variabler Länge von 4 Bits wird in das Bits-Legen-Coderegister **110B** geladen. Die 4 Bits des Bits-Legen-Coderegisters **110B**, die die Daten des Codeworts variabler Länge tatsächlich speichern, werden als gültige Bits bezeichnet. Die Anzahl von gültigen Bits von dem Bits-Legen-Coderegister **110B** wird dann bestimmt (**501**) durch Bezugnahme auf die Datenwerte, die vorher in dem Bits-Holen-Register **110G** gespeichert waren. Diese Bestimmung wird über die Steuerung **130** durchgeführt. Es ist jedoch möglich, dass eine solche Bestimmung durch die CPU **10** ausgeführt wird. Es wird dann bestimmt, ob alle gültigen Bits in verbleibende offene Bits des Bitstrompuffers **150** passen (**502**). Falls dies der Fall ist, werden diese gültigen Bits in verbleibende offene Bits des Bitstrompuffers **150** geladen (**503**) und nachfolgend als Datenwörter fester Länge ausgelesen (**511**).

[0036] Falls die gültigen Bits nicht in den Bitstrompuffer (**150**) passen, wird alternativ bestimmt, ob eines der gültigen Bits in den Bitstrompuffer **150** passt (**504**). Wo einige der gültigen Bits in den Bitstrompuffer **150** passen, werden dieselben (erster Satz von teilweise gültigen Bits) in den Bitstrompuffer **150** geladen (**505**), um alle verfügbaren Bits des Bitstrompuffers **150** zu füllen. Die Bitstrompufferinhalte werden dann durch ein Zuerst-Hinein-Zuerst-Hinaus-(FIFO)Register **160** (**506**) ausgelesen. Der Bitstrompuffer **150** ist dann frei von jeglichen Daten und der zweite Satz von teilgültigen Bits wird in den Bitstrompuffer **150** geladen (**507**). Dieser Prozess ist in Fig. 7 dargestellt, der nachfolgend näher erörtert wird.

[0037] Wo keines der gültigen Bits in den Bitstrompuffer passt, weil derselbe bereits voll mit Daten ist, werden die Inhalte des Bitstrompuffers **150** in den FIFO **160** geladen (**508**). Falls der FIFO **160** voll ist (**509**), werden die Inhalte desselben geleert (**510**). Der FIFO **160** ist einem Zähler **170** zugeordnet. Der Zähler **170** behält einen Zählwert von beispielsweise der Anzahl von 32 Bitwörtern bei, die tatsächlich in dem FIFO **160** geladen sind. Wenn der Zählwert des Zählers **170** dem FIFO-Unterbrechungspegelwert entspricht, der in dem Steuerregister **110C** gespeichert ist, bewirkt die Unterbrechungssteuerung **180**, dass eine Unterbrechung erzeugt wird und an die CPU **10** gerichtet wird. Die CPU **10** wird wiederum eine vorbestimmte Anzahl von Wörtern von dem FIFO **160** auslesen. Dies macht Raum in dem FIFO **160** verfügbar, zum Aufnehmen zusätzlicher Codewortdaten von dem Bitstrompuffer **150**.

DECODIERMODUS

[0038] Fig. 6 zeigt ein Flussdiagramm, das das Decodierverfahren der vorliegenden Erfindung darstellt. In dem Decodiermodus werden Daten in das Steuerregister **110E** geschrieben, um den FIFO-Unterbrechungspegel und das Unterbrechungsfreigabebit zu setzen. Das Codier/Decodierbit wird auf Decodieren gesetzt. Das Bits-Holen-Register **110G** wird adressiert, um zu bewirken, dass ein Datenwort fester Länge beispielsweise in den FIFO **160** geladen wird. Der Bitstrompuffer **150** empfängt ein Datenwort fester Länge von dem FIFO **160**. Das Datenwort fester Länge besteht aus mehreren Codewörtern variabler Län-

ge. Ein Codewort variabler Länge wird von dem Bitstrompuffer **150** ausgelesen, gemäß Datenlängeninformationen, die in dem Bits-Holen-Register **110G** gespeichert sind. Der Multiplexer **140** leitet dann das Codewort variabler Länge zu dem Bits-Holen-Register **110G**, von wo aus es dann als ein Codewort fester Länge ausgelesen wird. Der Betrieb des Multiplexers **140**, des Bitstrompuffers **150** und des FIFO **160** werden durch die Steuerung **130** gemäß Daten gesteuert, die in dem Registerblock **110** gespeichert sind. Sobald alle Codewörter variabler Länge von dem Codewort fester Länge, das in den Bitstrompuffer **150** geladen ist, ausgelesen wurden, wird ein weiteres Codewort fester Länge von dem FIFO **160** wiedergewonnen und in den Bitstrompuffer **150** geladen.

[0039] Der FIFO **160** ist beispielsweise ein 32 Bit mal 16 Wort FIFO-Register. Der Zähler **170** behält einen Zählwert der Anzahl von Datenwörtern bei, die zu einem bestimmten Zeitpunkt in dem FIFO **160** gespeichert sind, und speichert diesen Wert in dem FIFO-Pegelregister **110C**. Wenn der Wert, der in dem FIFO-Pegelregister **110C** gespeichert ist, einem FIFO-Unterbrechungspegelwert entspricht, der in dem Steuerregister **110** gespeichert ist, erzeugt die Unterbrechungssteuerung **180** ein Unterbrechungssignal. Dieses Unterbrechungssignal wird an die CPU **10** gerichtet, diese antwortet durch Bewirken, dass zusätzliche Daten fester Länge in den FIFO **160** geladen werden.

[0040] Fig. 7 stellt dar, wie gültige Bits von Codewörtern variabler Länge während den Codiermodusoperationen der vorliegenden Erfindung aus dem Bits-Legen-Coderegister **110B** ausgelesen und in den Bitstrompuffer **150** geladen werden. Fig. 7 zeigt auch, wie Codewörter fester Länge aus dem Bitstrompuffer **150** ausgelesen werden und in das Bits-Legen-Coderegister **110B** geladen werden, während den Codiermodusoperationen der vorliegenden Erfindung. Der Bitstrompuffer **150A** zeigt den Status des Bitstrompuffers **150**, nachdem gültige Bits der Codewörter variabler Länge, die einen ersten Teildatensatz umfassen, die verfügbaren Bits des Bitstromregisters **150** vollständig gefüllt haben. Der Bitstrom **150B** zeigt den Status des Bitstrompuffers **150**, nachdem ein zweiter Teilsatz der Codewörter variabler Länge in den Bitstrompuffer **150** geladen wurde.

[0041] Der Anfangszustand des Bitstrompuffers **150** ist frei und alle Bits sind verfügbar, um Daten aufzunehmen. Es wird angemerkt, dass die Darstellung in Fig. 7 den Bitstrompuffer **150** als ein 16-Bit-Register zeigt. Bei einem bevorzugten Ausführungsbeispiel ist der Bitstrompuffer **150** jedoch 32 Bits lang. Es ist klar, dass der Bitstrompuffer **150** jede Länge aufweisen kann.

[0042] Bezüglich der Codiermodusoperationen ist ersichtlich, dass die gültigen Bits **701**–**704** eines ersten Codeworts **700** in die höchstwertigsten vier (4) Bits des Bitstrompuffers **150** geladen werden, die verfügbar sind. Nachfolgend werden die gültigen Bits **801**–**806** des Codeworts **800** in die nächsten höchstwertigsten Bits des Bitstrompuffers **150A** geladen, die verfügbar sind. Als nächstes werden die gültigen Bits **901**–**903** des Codeworts **900** in die nächsten höchstwertigsten Bits des Bitstrompuffers **150A** geladen, die verfügbar sind. Im Fall des Codeworts **1000** gibt es sechs gültige Datenbits **1001**–**1007**. Der Bitstrompuffer **150A** hat jedoch nur drei (3) verfügbare Bits, die zum Aufnehmen von Daten verbleiben. In diesem Fall werden die Bits **1001**–**1003** in die verbleibenden verfügbaren Bits des Bitstrompuffers **150** geladen. Nachfolgend werden die Inhalte des Bitstrompuffers **150A** ausgelesen und in dem FIFO **160** gespeichert. Der Bitstrompuffer **150** ist dann frei (**150B**) und offen, um zusätzliche Datenbits aufzunehmen. Die verbleibenden Bits **1004**–**1006** werden dann in die höchstwertigsten Bits des Bitstrompuffers **150B** geschrieben. Dies wird fortgesetzt, bis alle verfügbaren Bits des Bitstrompuffers **150** mit gültigen Codewortdaten geladen sind, oder es keine weiteren Codewortdaten zum Laden gibt.

[0043] In dem Fall von Decodieroperationen werden Codewörter variabler Länge **701**–**704** von dem Bitstrompuffer **150A** ausgelesen und in das Bits-Legen-Coderegister **110B** geladen. Das Bits-Legen-Coderegister **110B** wird dann ausgelesen und gelöscht. Die Codewörter variabler Länge **801**–**806** werden dann von dem Bitstrompuffer **150A** ausgelesen und in das Bits-Legen-Coderegister **110B** geladen. Dies wird fortgesetzt, bis alle Codewörter variabler Länge aus dem Bitstrompuffer **150A** ausgelesen sind. Wenn die Codewörter variabler Länge **1001**–**1003** ausgelesen sind und in das Bits-Legen-Coderegister **110B** geladen sind, wird erkannt, dass diese Bits nur ein Teilsegment des vollen Codeworts sind. In diesem Fall wird das Bits-Legen-Coderegister **110B** nicht unmittelbar ausgelesen und gelöscht. Es wird jedoch ein anderes Wort variabler Länge in den Bitstrompuffer **150B** geschrieben. Das zweite Segment des Codeworts variabler Länge, die Bits **1004**–**1006**, werden dann aus dem Bitstrompuffer **150B** ausgelesen und in das Bits-Legen-Register **110B** geladen, das dann ausgelesen wird und von Daten gelöscht wird.

[0044] Obwohl das Bitstromregister **150** hierin als 16-Bit-Register dargestellt ist, wird angemerkt, dass es als Register jeder Größe implementiert werden kann, wie es für die spezifischen Zwecke geeignet ist, einschließlich, aber nicht beschränkt auf, beispielsweise ein 8-Bit-Register, ein 32-Bit-Register oder ein 64-Bit-Register. Gleichartig dazu können der Multiplexer **140** und der FIFO **160** in jeder Bitlänge implementiert werden, die gewünscht wird, oder als am geeignetsten für die entsprechende Anwendung bestimmt wird.

[0045] Das Verfahren der vorliegenden Erfindung kann in Hardware, Software, Firmware oder einer Kombination derselben implementiert werden. Bei den bevorzugten Ausführungsbeispielen ist das Verfahren in Software oder in Firmware implementiert, die in einem Speicher gespeichert ist, und die durch ein geeignetes Befehlsausführungssystem ausgeführt wird. Falls dasselbe in Hardware implementiert ist, wie bei einem alternativen Ausführungsbeispiel, kann das Verfahren mit jeder oder einer Kombination der folgenden Technologien implementiert sein, die alle in der Technik gut bekannt sind: eine diskrete Logikschaltung mit Logikdaten zum Implementieren von Logikfunktionen auf Datensignale hin, eine anwendungsspezifische integrierte Schaltung (ASIC) mit entsprechenden Kombinationslogikdaten, ein programmierbares Gatterarray (PGA), ein feldprogrammierbares Gatterarray (FPGA), usw.

[0046] Die Flussdiagramme von Fig. 5 und Fig. 6 zeigen die Architektur, Funktionalität und den Betrieb einer möglichen Implementierung des Codierungs- und Decodierungsverfahrens der vorliegenden Erfindung. Diesbezüglich stellt jeder Block ein Modul, ein Segment oder einen Abschnitt des Codes dar, der einen oder mehrere ausführbare Befehle zum Implementieren der spezifizierten logischen Funktion(en) umfasst. Es sollte auch angemerkt werden, dass bei einigen alternativen Ausführungsbeispielen die Funktionen, die in den Blöcken angemerkt werden, außerhalb der Reihenfolge auftreten können, die in Fig. 5 oder Fig. 6 angemerkt ist. Beispielsweise können zwei Blöcke, die in Fig. 5 oder Fig. 6 aufeinanderfolgend gezeigt sind, in der Tat im Wesentlichen gleichzeitig ausgeführt werden, oder die Blöcke können manchmal in umgekehrter Reihenfolge ausgeführt werden, abhängig von der betreffenden Funktionalität, wie es oben erörtert ist.

[0047] Es sollte betont werden, dass die oben beschriebenen Ausführungsbeispiele der vorliegenden Erfindung insbesondere jegliche „bevorzugte" Ausführungsbeispiele lediglich mögliche Beispiele von Implementierungen sind und lediglich für ein klares Verständnis der Prinzipien der Erfindung beschrieben sind.

Patentansprüche

1. Ein Codierer/Decodierer, der konfiguriert ist, um abwechselnd in einem ersten Betriebsmodus und einem zweiten Betriebsmodus zu arbeiten, wobei der Codierer/Decodierer folgende Merkmale aufweist:
ein Datenregister (**110**), das eine vorbestimmte Anzahl von Bits umfasst;
ein Bitstromregister (**150**), das eine vorbestimmte Anzahl von Bits umfasst;
einen Multiplexer (**140**);
ein Zuerst-Hinein-Zuerst-Hinaus-(FIFO-)Register (**160**), das eine Mehrzahl von verfügbaren Mehrbitregistern umfasst;
eine Unterbrechungssteuerung (**180**) zum Erzeugen eines Unterbrechungssignals;
wobei das Datenregister (**110**) konfiguriert ist, um während dem ersten Betriebsmodus ein Codewort variabler Länge zu empfangen und zu speichern, das gültige Datenbits umfasst, die durch einen Entropiecodierungsalgorithmus komprimiert werden, und um während dem zweiten Betriebsmodus Daten, die ein Codewort variabler Länge umfassen, das durch einen Entropiedecodierungsalgorithmus dekomprimiert werden soll, zu empfangen und zu speichern;
wobei das Bitstromregister (**150**) konfiguriert ist, um während dem ersten Betriebsmodus Daten zu empfangen und während dem zweiten Betriebsmodus ein codiertes Datenwort fester Länge von dem FIFO-Register (**160**) zu empfangen;
wobei der Multiplexer konfiguriert ist, um während dem ersten Betriebsmodus zumindest einen Teil der gültigen Bits von dem Datenregister (**110**) in die höchstwertigsten nichtverwendeten Bits in dem Bitstromregister (**150**) zu laden, und während dem zweiten Betriebsmodus ein Codewort variabler Länge gemäß einem ausgewählten Wert, der in den Codierer/Decodierer geschrieben ist, von dem Bitstromregister (**150**) in das Datenregister (**110**) zu laden;
wobei das FIFO-Register (**160**) während dem ersten Betriebsmodus konfiguriert ist, um den Inhalt des Bitstromregisters (**150**) zu empfangen, wenn die vorbestimmte Anzahl von Bits des Bitstromregisters (**150**) mit einem Teil der gültigen Bits von Daten geladen sind und der Rest der gültigen Bits geladen werden muss;
wobei das FIFO-Register (**160**) während dem zweiten Betriebsmodus konfiguriert ist, um Daten zu empfangen, die aus dem codierten Datenwort fester Länge bestehen, wobei das Datenwort das Codewort variabler Länge umfasst;
wobei die Unterbrechungssteuerung (**180**) konfiguriert ist, um während dem ersten Betriebsmodus ein Auslesen von Daten von dem FIFO-Register (**160**) auszulösen; und
wobei die Unterbrechungssteuerung (**180**) ferner konfiguriert ist, um während dem zweiten Betriebsmodus das Schreiben des codierten Datenworts fester Länge in das FIFO-Register (**160**) auszulösen.

2. Ein Codierer/Decodierer gemäß Anspruch 1, bei dem der Multiplexer (**140**) ein Barrel-Schieberegister umfasst.

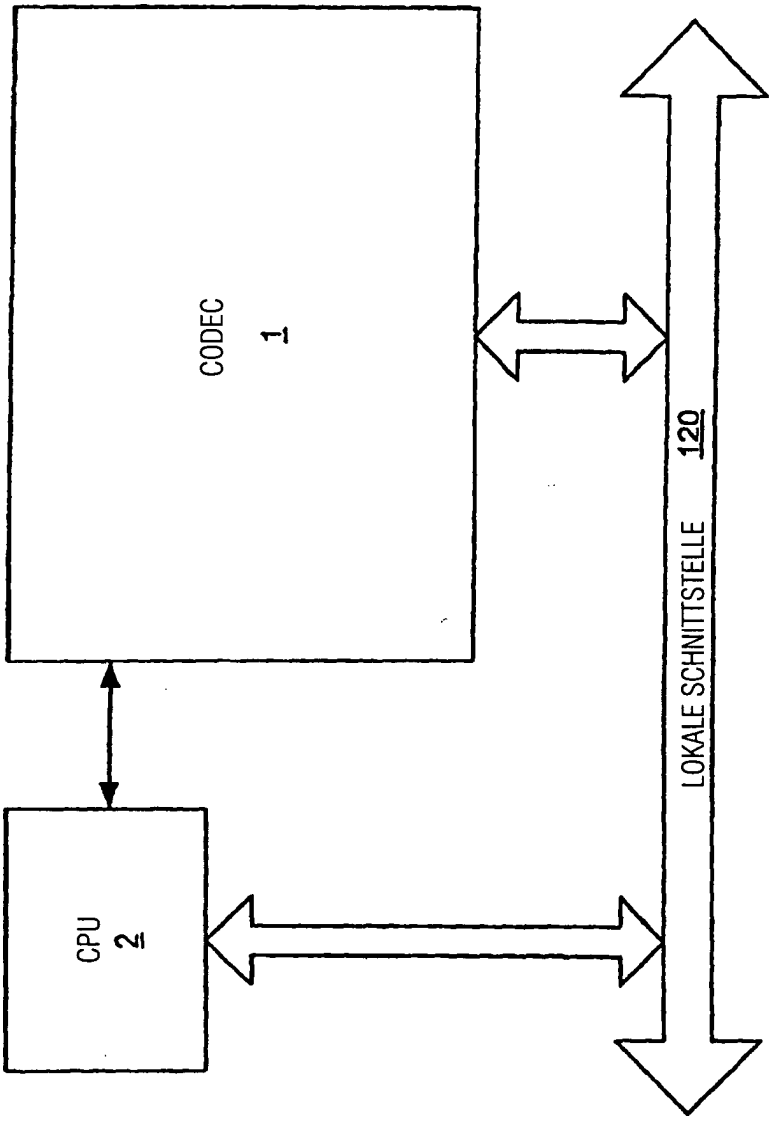
3. Ein Codierer/Decodierer gemäß Anspruch 1, der ferner einen Zähler (**170**) zum Zählen der Anzahl von Mehrbitwörtern umfasst, die in dem FIFO-Register (**160**) gespeichert sind.

4. Ein Codierer/Decodierer gemäß Anspruch 3, bei dem die Unterbrechungssteuerung (**180**) das Unterbrechungssignal erzeugt, wenn der Zähler (**170**)

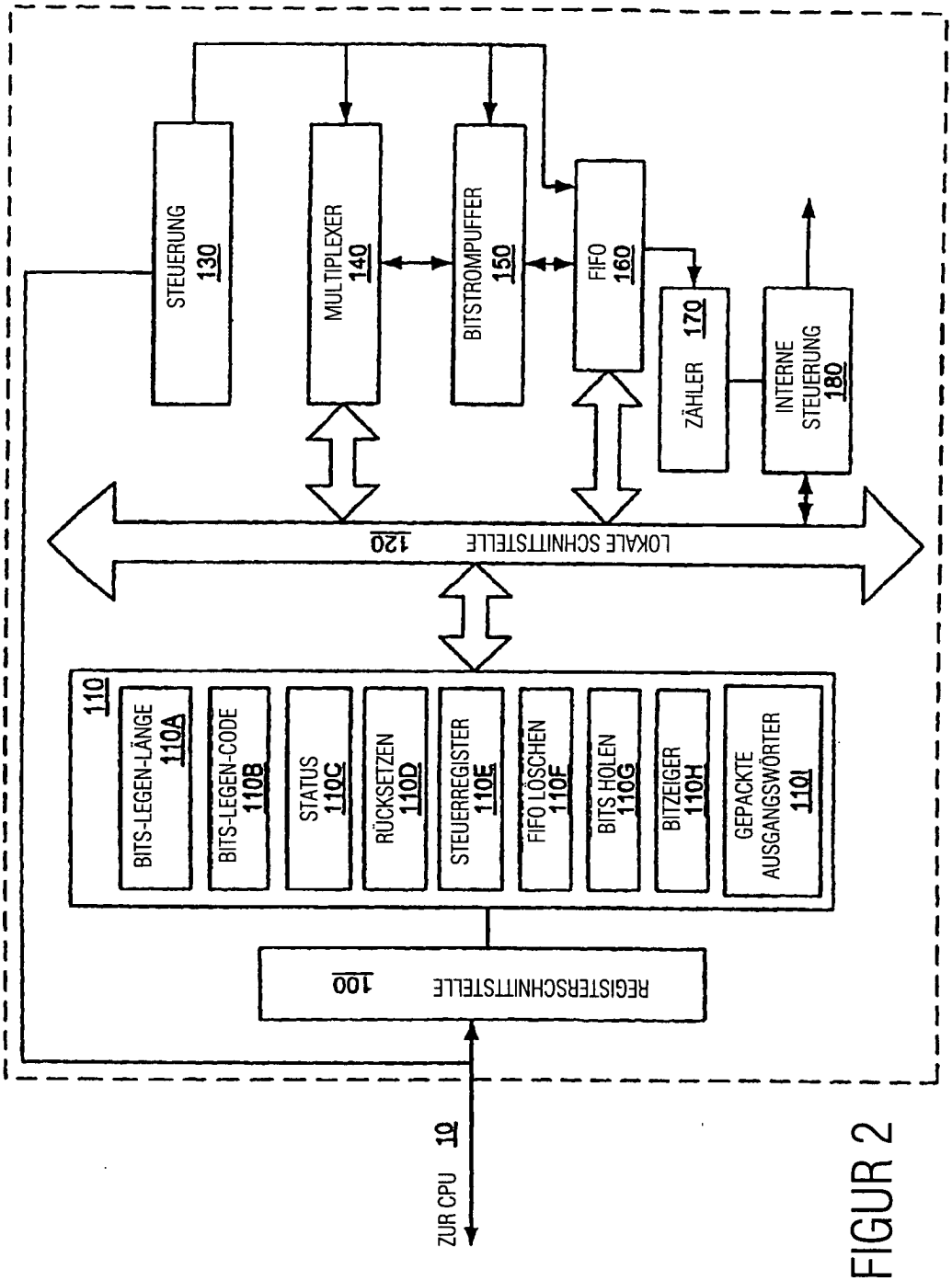
einen vorbestimmten Unterbrechungswert erreicht.

Es folgen 7 Blatt Zeichnungen

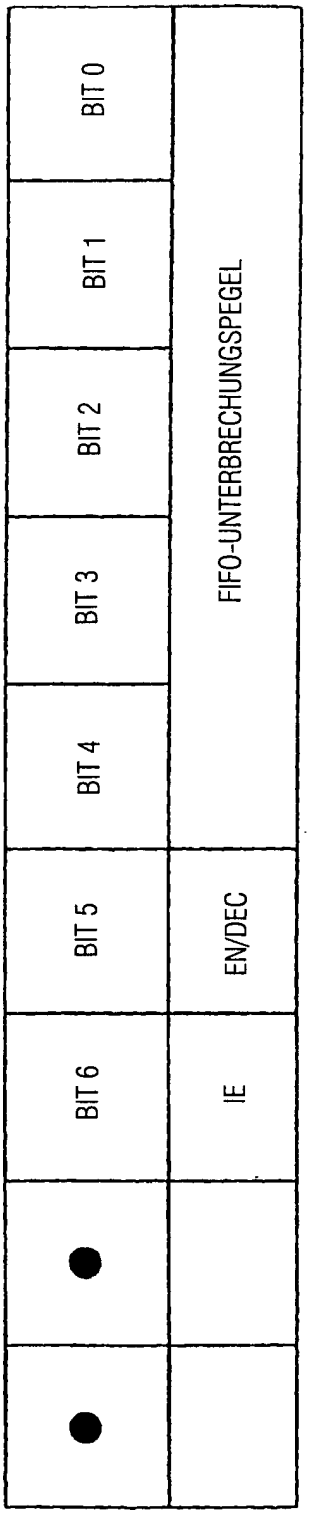
Anhängende Zeichnungen



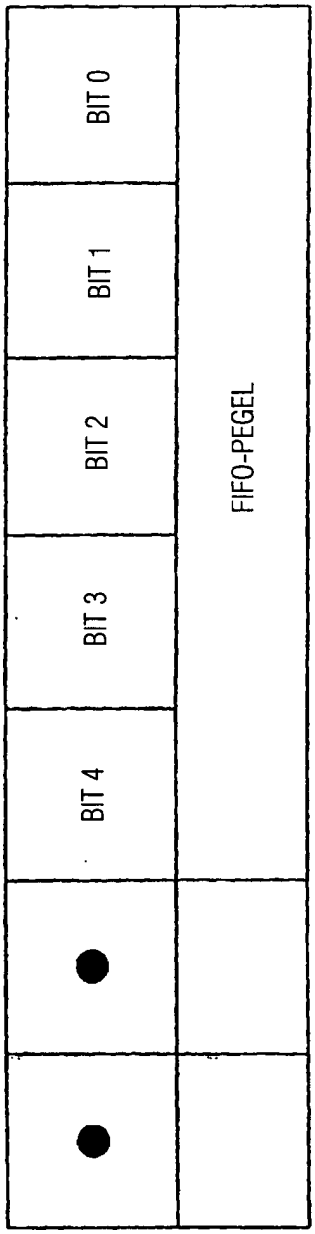
FIGUR 1


STEUERUNG 130
MULTIPLEXER 140
BITSTROMPUFFER 150
FIFO 160
ZÄHLER 170
INTERNE STEUERUNG 180
LOKALE SCHNITTSTELLE 120
110
BITS-LEGEN-LÄNGE 110A
BITS-LEGEN-CODE 110B
STATUS 110C
RÜCKSETZEN 110D
STEUERREGISTER 110E
FIFO LÖSCHEN 110F
BITS HOLEN 110G
BITZEIGER 110H
GEPACKTE AUSGANGSWÖRTER 110I
REGISTERSCHNITTSTELLE 100
ZUR CPU 10

FIGUR 2

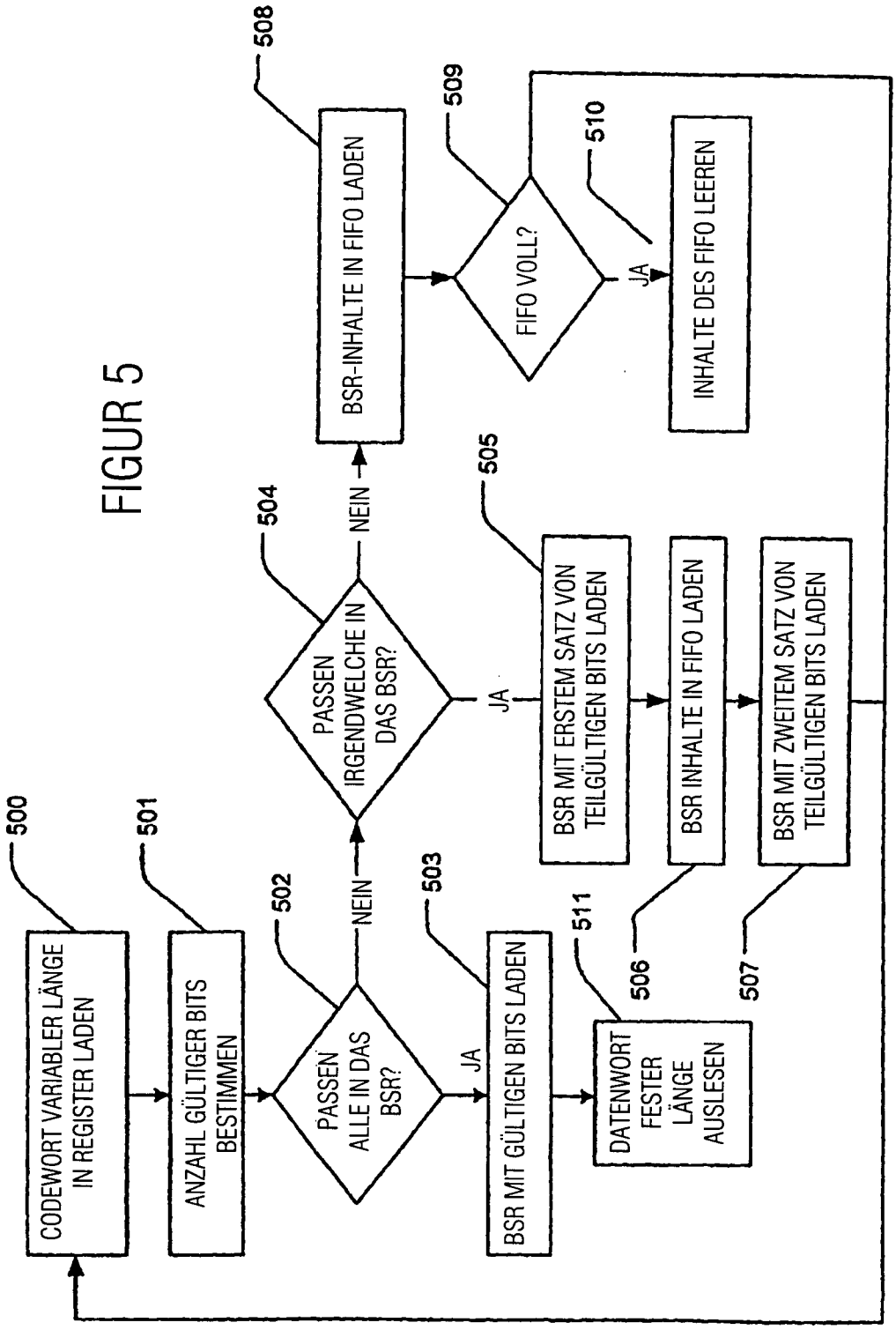

BIT 0
BIT 1
BIT 2
BIT 3
BIT 4
BIT 5
BIT 6
FIFO-UNTERBRECHUNGSPEGEL
EN/DEC
IE

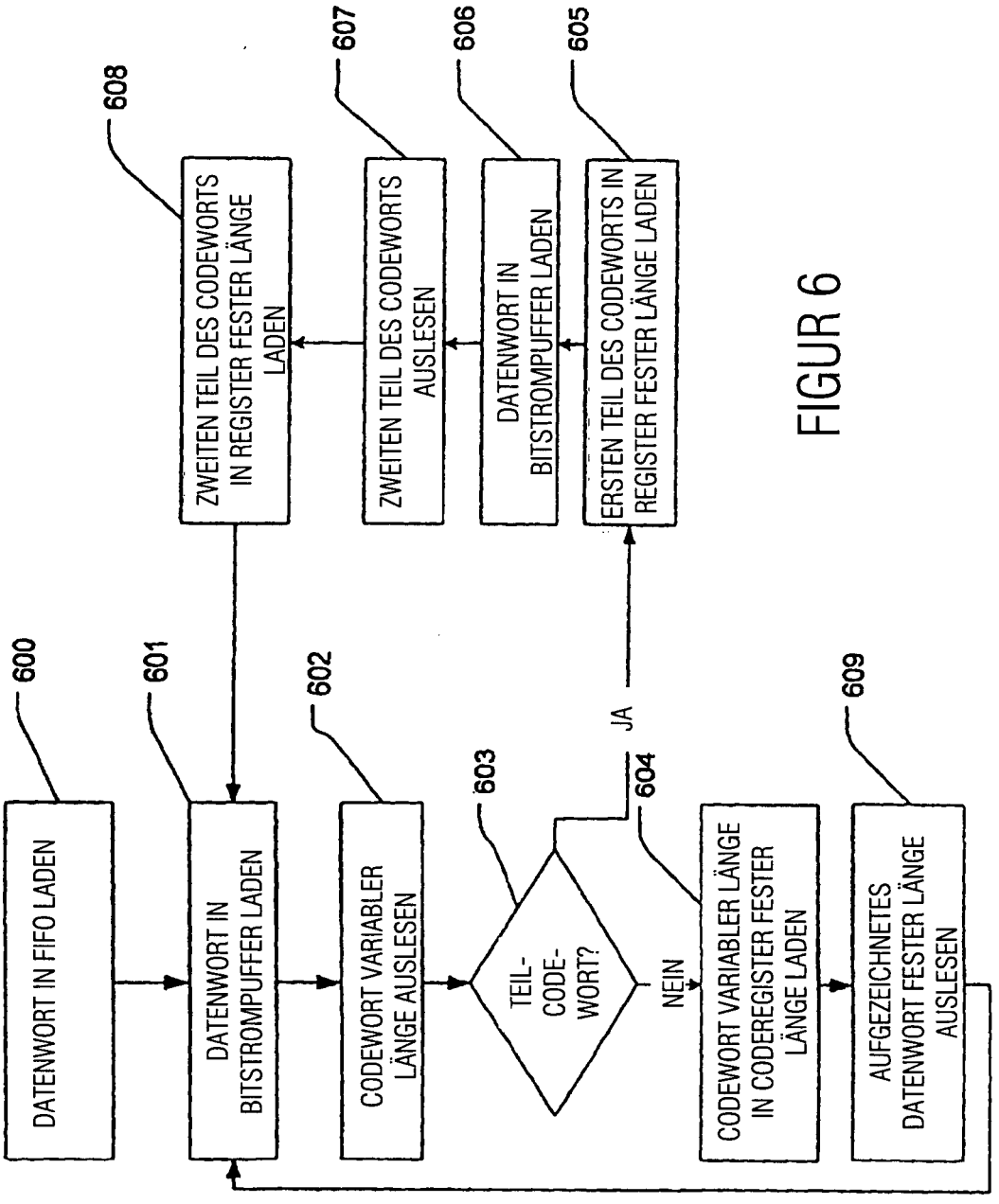
FIGUR 3



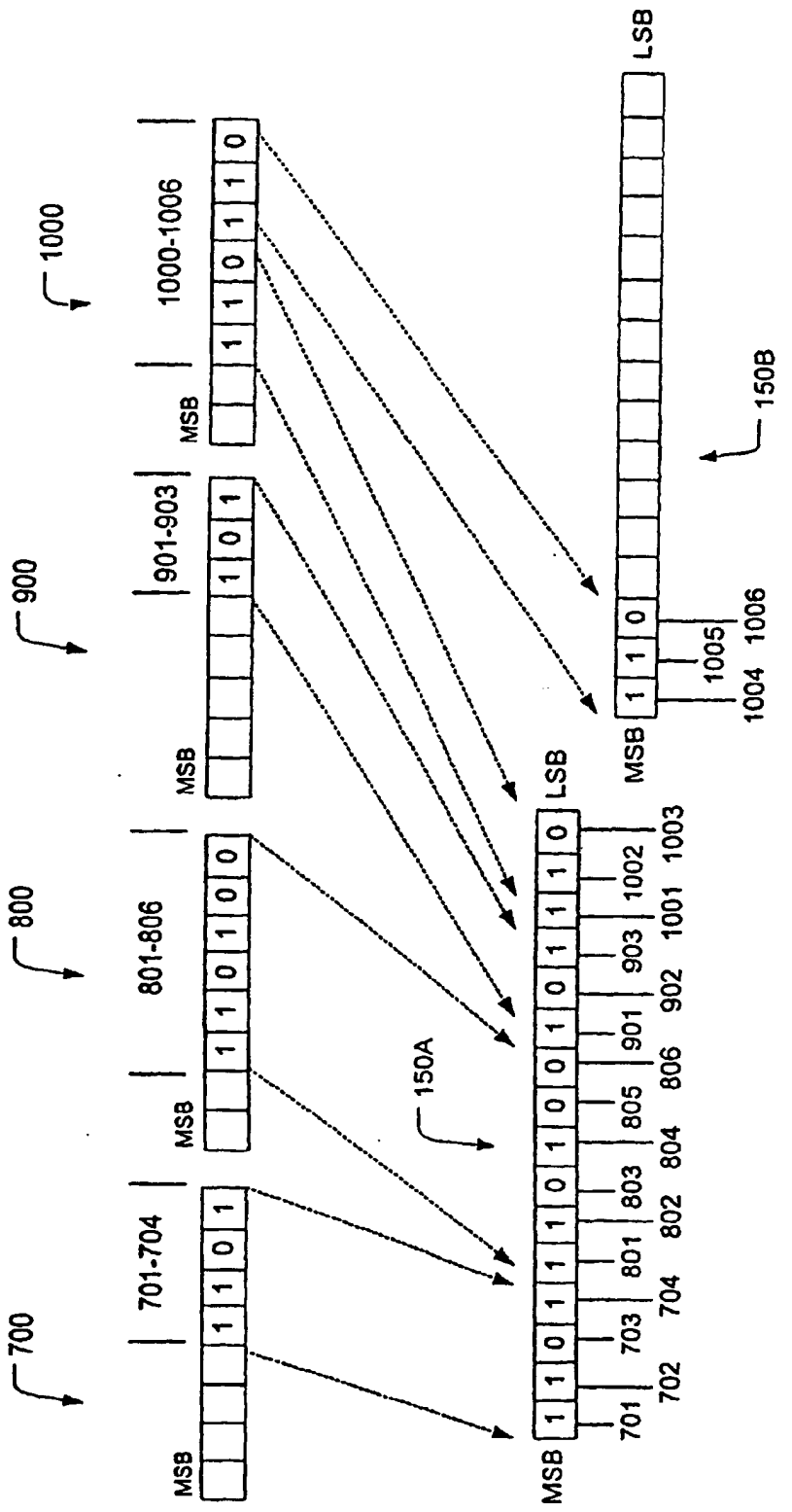
FIGUR 4

FIGUR 5


500
CODEWORT VARIABLER LÄNGE IN REGISTER LADEN
501
ANZAHL GÜLTIGER BITS BESTIMMEN
502
PASSEN ALLE IN DAS BSR?
NEIN
JA
503
BSR MIT GÜLTIGEN BITS LADEN
504
PASSEN IRGENDWELCHE IN DAS BSR?
NEIN
JA
505
BSR MIT ERSTEM SATZ VON TEILGÜLTIGEN BITS LADEN
506
BSR INHALTE IN FIFO LADEN
507
BSR MIT ZWEITEM SATZ VON TEILGÜLTIGEN BITS LADEN
508
BSR-INHALTE IN FIFO LADEN
509
FIFO VOLL?
JA
510
INHALTE DES FIFO LEEREN
511
DATENWORT FESTER LÄNGE AUSLESEN


600
DATENWORT IN FIFO LADEN
601
DATENWORT IN BITSTROMPUFFER LADEN
602
CODEWORT VARIABLER LÄNGE AUSLESEN
603
TEIL-CODE-WORT?
JA
604
NEIN
605
ERSTEN TEIL DES CODEWORTS IN REGISTER FESTER LÄNGE LADEN
606
DATENWORT IN BITSTROMPUFFER LADEN
607
ZWEITEN TEIL DES CODEWORTS AUSLESEN
608
ZWEITEN TEIL DES CODEWORTS IN REGISTER FESTER LÄNGE LADEN
609
CODEWORT VARIABLER LÄNGE IN CODEREGISTER FESTER LÄNGE LADEN
AUFGEZEICHNETES DATENWORT FESTER LÄNGE AUSLESEN

FIGUR 6


700
701-704
MSB
800
801-806
MSB
900
901-903
MSB
1000
1000-1006
MSB
150A
MSB
LSB
701
702
703
704
801
802
803
804
805
806
901
902
903
1001
1002
1003
150B
MSB
LSB
1004
1005
1006

FIGUR 7