US005675654A

United States Patent [19]

Ryan

[11] **Patent Number: 5,675,654**

[45] **Date of Patent: Oct. 7, 1997**

[54] **SYSTEM AND METHOD FOR INTERFACING A TRANSPORT DECODER TO A NATIONAL RENEWABLE SECURITY SYSTEMS (NRSS) SMART CARD**

[75] Inventor: **Robert T. Ryan**, Langhorne, Pa.

[73] Assignee: **Matsushita Electric Corporation of America**, Secaucus, N.J.

[21] Appl. No.: **626,176**

[22] Filed: **Mar. 29, 1996**

[51] **Int. Cl.**[6] **H04N 7/167**

[52] **U.S. Cl.** **380/48**; 380/9; 380/20

[58] **Field of Search** 380/48, 9, 20

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,941,173	7/1990	Boule et al.	380/48
5,361,062	11/1994	Weiss et al.	380/48
5,581,310	12/1996	Vinekar et al.	348/718
5,588,025	12/1996	Strolle et al.	375/316
5,598,222	1/1997	Lane	348/568
5,613,001	3/1997	Bakhoum	380/48

OTHER PUBLICATIONS

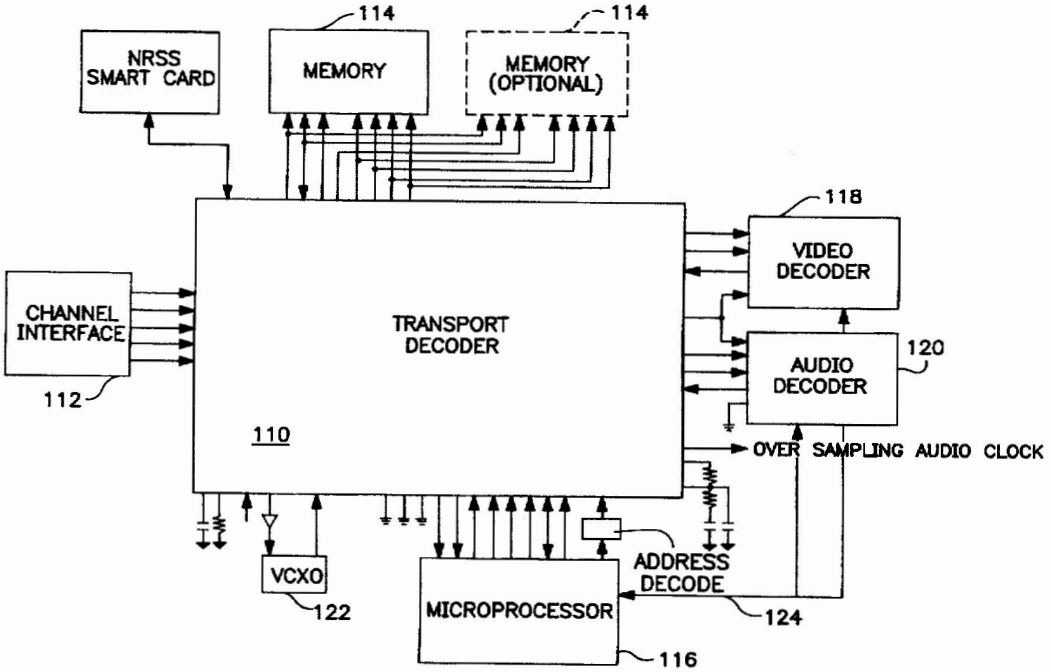
National Renewable Security Standards Committee (NRSS), "EIA Standard for Conditional Access", Version 2.6 Draft, Apr. 21, 1995, pp. 1–20.

Primary Examiner—David C. Cain
Attorney, Agent, or Firm—Ratner and Prestia

[57] **ABSTRACT**

A transport decoder **110**, for receiving and processing a transport data stream using MPEG-2 formats, includes connections to a physical layer channel interface (channel interface) **112**, a buffer memory **114**, a host microprocessor **116**, audio and video decoders **118/120**, and clock signal circuitry **122**. The transport decoder also includes an interface to a decoder which does not include a "data valid" input terminal. Upon receipt of encoded data packets, the transport decoder recognizes a frame synchronization byte and transfers an encoded data packet to an external decoder via the interface. The transport decoder sets a count value for a predetermined number of bytes in the encoded data packet and sends the packet data to the external decoder. When the specified number of bytes have been sent to the external decoder, the transport decoder determines if another synchronization byte has been encountered. If another synchronization byte has been found, the transport decoder continues to send encoded data to the decoder. Otherwise, the transport decoder sends zero-valued data to the external decoder.

2 Claims, 7 Drawing Sheets



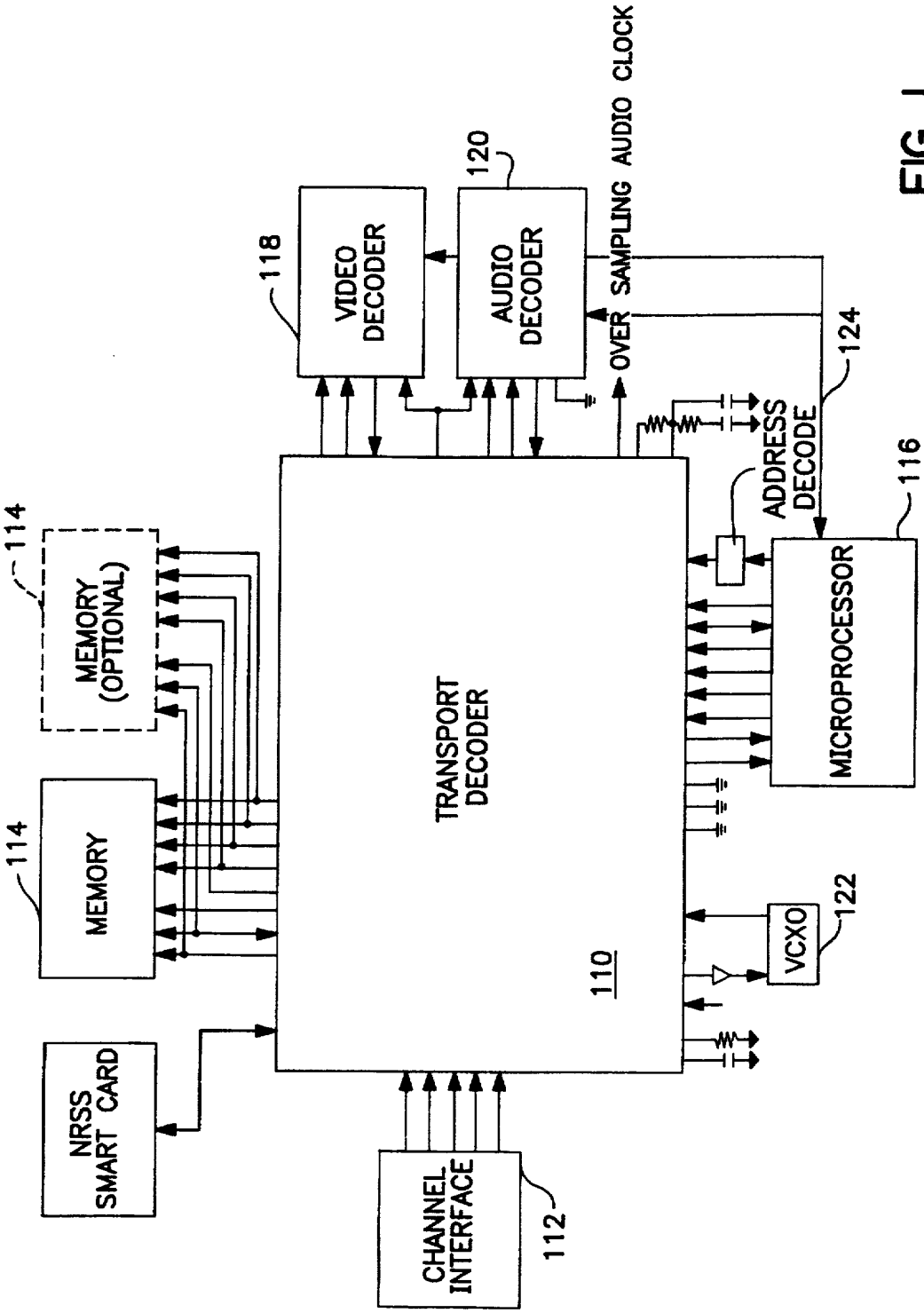
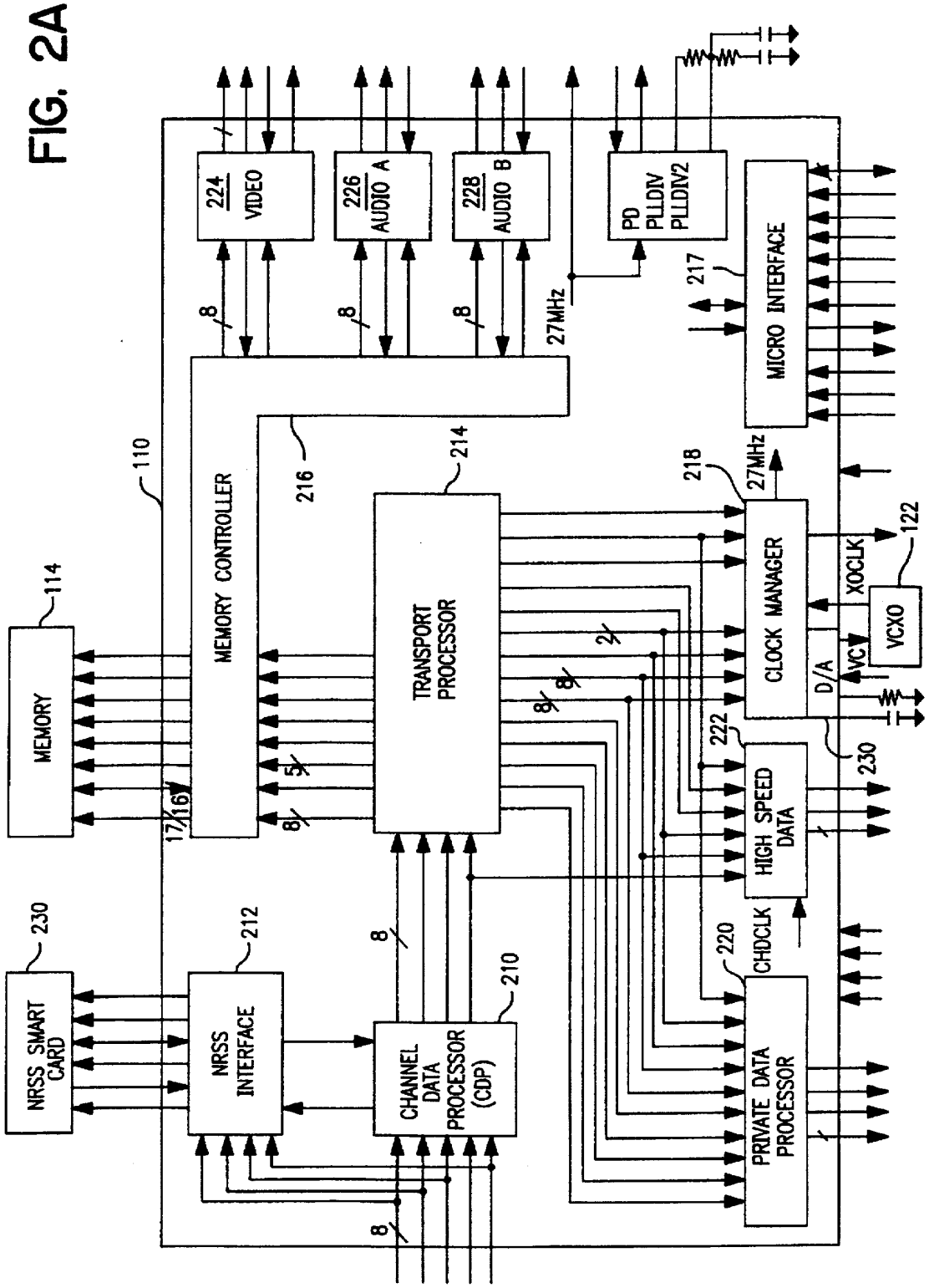

VIDEO DECODER
118
AUDIO DECODER
120
OVER SAMPLING AUDIO CLOCK
124
ADDRESS DECODE
116
MICROPROCESSOR
MEMORY (OPTIONAL)
114
MEMORY
114
TRANSPORT DECODER
110
VCXO
122
NRSS SMART CARD
CHANNEL INTERFACE
112

FIG. 1

FIG. 2A


224
VIDEO
226
AUDIO A
228
AUDIO B
PD
PLLDIV
PLLDIV2
MICRO INTERFACE
217
27MHz
8
110
MEMORY CONTROLLER
216
214
TRANSPORT PROCESSOR
218
CLOCK MANAGER
XOCLK
122
VCXO
VC
D/A
230
222
HIGH SPEED DATA
220
CHDCLK
PRIVATE DATA PROCESSOR
114
MEMORY
17/16
5
2
NRSS SMART CARD
212
NRSS INTERFACE
210
CHANNEL DATA PROCESSOR (CDP)

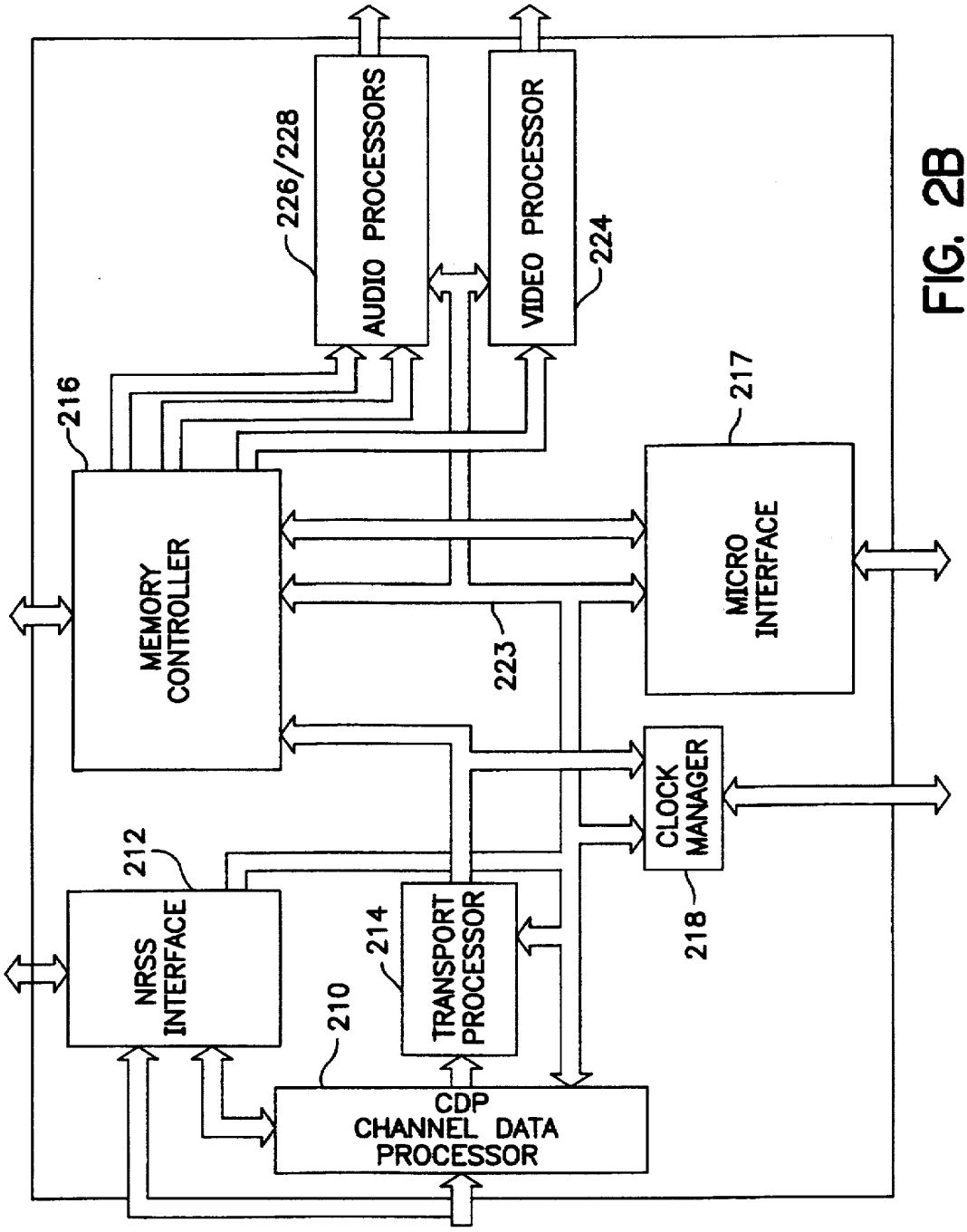

226/228
AUDIO PROCESSORS
VIDEO PROCESSOR
224
216
MEMORY CONTROLLER
217
MICRO INTERFACE
223
CLOCK MANAGER
218
212
NRSS INTERFACE
214
TRANSPORT PROCESSOR
210
CDP
CHANNEL DATA PROCESSOR

FIG. 2B

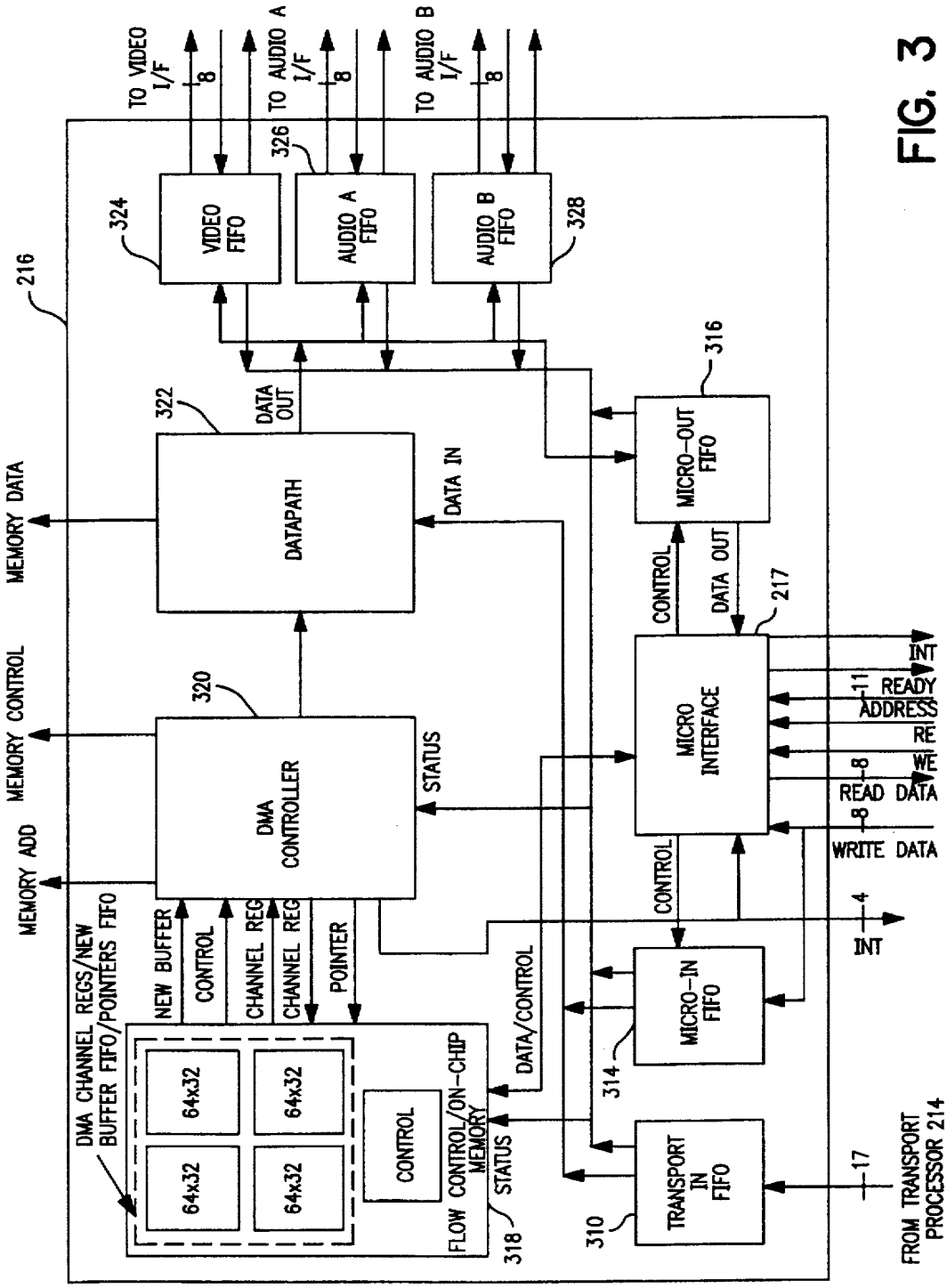

TO VIDEO I/F
TO AUDIO A I/F
TO AUDIO B I/F
8
8
8
326
324
328
VIDEO FIFO
AUDIO A FIFO
AUDIO B FIFO
216
322
DATA OUT
DATA IN
DATAPATH
MEMORY DATA
MEMORY CONTROL
MEMORY ADD
320
DMA CONTROLLER
STATUS
316
MICRO-OUT FIFO
CONTROL
DATA OUT
217
MICRO INTERFACE
INT
11
READY
ADDRESS
RE
8
WE
READ DATA
8
WRITE DATA
CONTROL
4
INT
MICRO-IN FIFO
314
DMA CHANNEL REGS/NEW BUFFER FIFO/POINTERS FIFO
NEW BUFFER
CONTROL
CHANNEL REG
CHANNEL REG
POINTER
64x32
64x32
64x32
64x32
CONTROL
FLOW CONTROL/ON-CHIP MEMORY
STATUS
318
DATA/CONTROL
TRANSPORT IN FIFO
310
17
FROM TRANSPORT PROCESSOR 214

FIG. 3

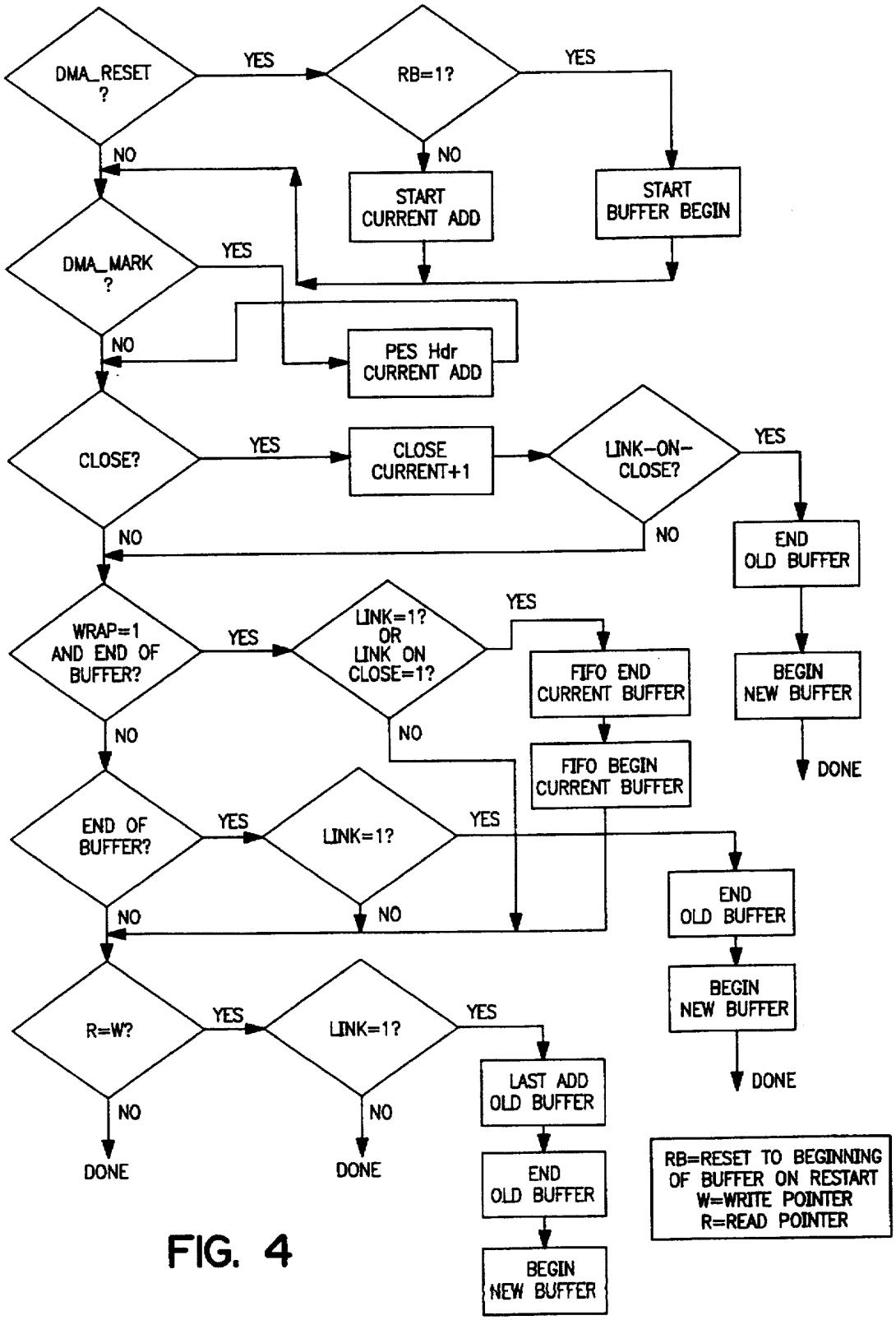

DMA_RESET ?
YES
RB=1?
YES
NO
NO
START CURRENT ADD
START BUFFER BEGIN
DMA_MARK ?
YES
NO
PES Hdr CURRENT ADD
CLOSE?
YES
CLOSE CURRENT+1
LINK-ON-CLOSE?
YES
NO
NO
END OLD BUFFER
WRAP=1 AND END OF BUFFER?
YES
LINK=1? OR LINK ON CLOSE=1?
YES
FIFO END CURRENT BUFFER
BEGIN NEW BUFFER
NO
NO
FIFO BEGIN CURRENT BUFFER
DONE
END OF BUFFER?
YES
LINK=1?
YES
END OLD BUFFER
NO
NO
BEGIN NEW BUFFER
R=W?
YES
LINK=1?
YES
DONE
LAST ADD OLD BUFFER
NO
NO
END OLD BUFFER
RB=RESET TO BEGINNING OF BUFFER ON RESTART
W=WRITE POINTER
R=READ POINTER
DONE
DONE
BEGIN NEW BUFFER

FIG. 4

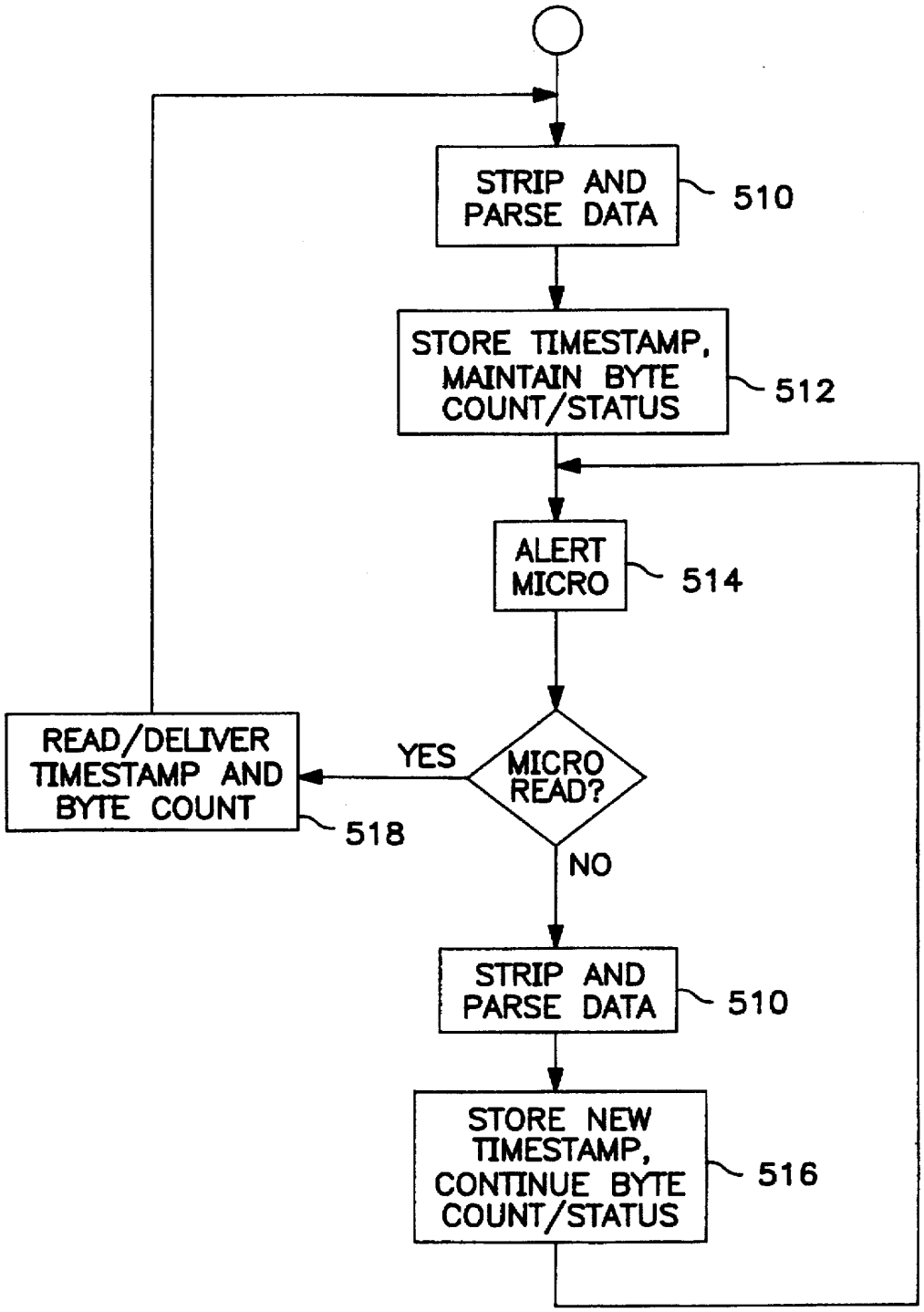

STRIP AND PARSE DATA
510
STORE TIMESTAMP, MAINTAIN BYTE COUNT/STATUS
512
ALERT MICRO
514
MICRO READ?
YES
NO
READ/DELIVER TIMESTAMP AND BYTE COUNT
518
STRIP AND PARSE DATA
510
STORE NEW TIMESTAMP, CONTINUE BYTE COUNT/STATUS
516

FIG. 5A

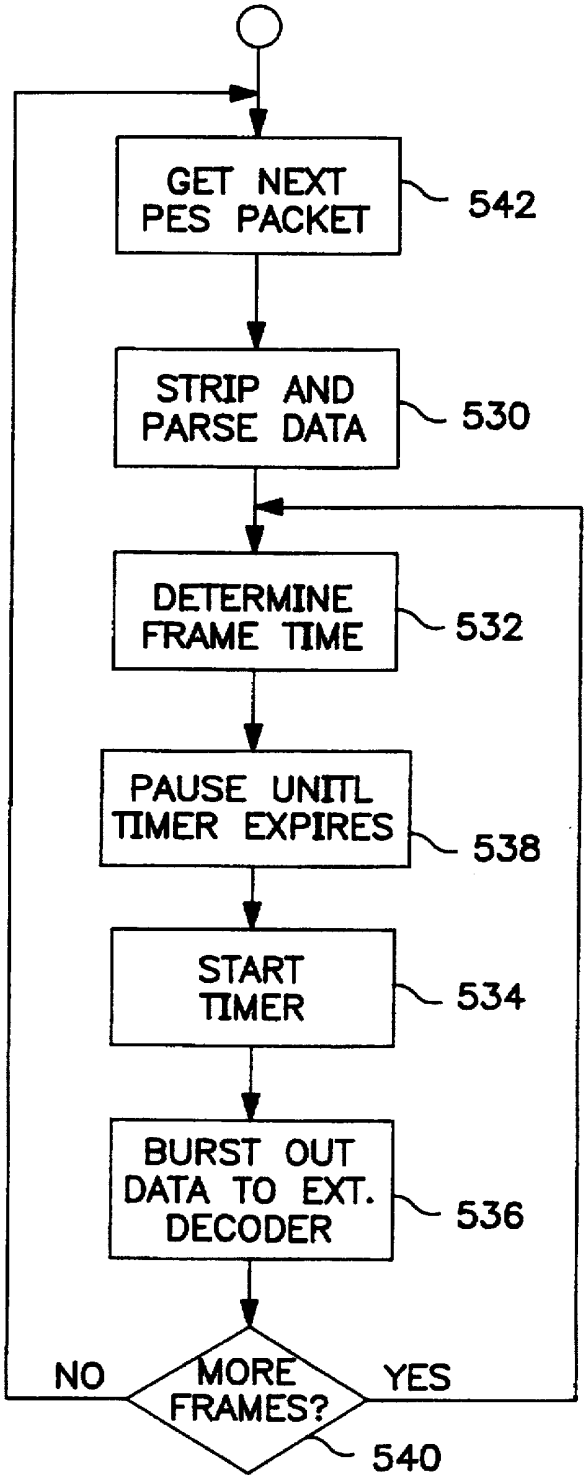

GET NEXT PES PACKET
542
STRIP AND PARSE DATA
530
DETERMINE FRAME TIME
532
PAUSE UNITL TIMER EXPIRES
538
START TIMER
534
BURST OUT DATA TO EXT. DECODER
536
MORE FRAMES?
540
NO
YES

FIG. 5B

SYSTEM AND METHOD FOR INTERFACING A TRANSPORT DECODER TO A NATIONAL RENEWABLE SECURITY SYSTEMS (NRSS) SMART CARD

FIELD OF THE INVENTION

The present invention relates generally to data reception, processing and transmission according to the MPEG-2 standards and, more particularly, the present invention relates to the operation and interfacing of a transport decoder which handles an encoded MPEG-2 format datastream using an NRSS smart-card decoder.

BACKGROUND OF THE INVENTION

High Definition Television (HDTV) continues to make progress in its attempts to replace conventional television. Paving the way for this progress are various companies and associations working on standards to provide for a global market for HDTV.

One such group of companies is known as the "Digital HDTV Grand Alliance" including members such as AT&T, David Sarnoff Research Center, Massachusetts Institute of Technology and others. A comprehensive overview of the strides made by this group are presented in an article by Robert Hopkins entitled "Digital Terrestrial HDTV for North America: The Grand Alliance HDTV System" published in the IEEE Transactions on Consumer Electronics (Summer 1994). This article is herein incorporated by reference for all of its teachings regarding the background and basics of HDTV systems including the use of Program and Transport Packet Streams.

One standard that has been adopted by the Grand Alliance is the MPEG-2 standard for encoding video and audio information, developed by the Moving Pictures Expert Group (MPEG), a committee within the International Standards Organization (ISO). Accepted standards are periodically published such as, for example, the Video Section of Information Technology—Generic Coding of Moving Pictures and Associated Audio ISO/IEC 13818-2 (1995) (hereinafter "Video Section") and the Systems Section of Information Technology—Generic Coding of Moving Pictures and Associated Audio ISO/IEC 13818-1 (November 1994) (hereinafter "Systems Section") both of which are incorporated herein by reference for their teachings regarding established standards and formats.

The syntax for the MPEG-2 standard defines several layers of data records which are used to convey both audio and video data. To transmit information, a digital data stream, representing, for example, multiple video sequences, is divided into several smaller units and each of these units is encapsulated into a respective packetized elementary stream (PES) packet. For transmission, each PES packet is divided, in turn, among a plurality of fixed-length transport packets. Each transport packet contains data relating to only one PES packet. The transport packet also includes a header which holds control information, sometimes including an adaptation field, to be used in decoding the transport packet. However, PES data may include multiple elementary streams.

This datastream, in the form of transport packets, may for security purposes be encrypted at a transport encoder and decrypted at a transport decoder. For example, a National Renewable Security Standard (NRSS) Smart Card may be used in the encryption/decryption process. It is noted that generally an NRSS Smart Card includes a resident, programmable microprocessor and decryption engine. The EIA

Standard for Conditional Access, Version 2.6, NRSS Committee (April 1995) is hereby incorporated by reference for its teachings relating to the NRSS Smart Card.

However, when decryption is necessary, valid data, beginning with a predetermined synchronization byte, is sent to the NRSS Smart Card. The Smart Card recognizes this predetermined synchronization byte (i.e., distal pattern), synchronizes to it and begin decrypting, for example, the next 188 bytes.

A problem, however, could exist if, in some random datastream flowing between valid packets of data, a pattern matching the predetermined sync byte was allowed to pass to the Smart Card. If this were to occur, the Smart Card would synchronize to essentially a false sync byte, begin decrypting and potentially miss a true sync byte, thus, failing to properly decrypt the valid encrypted data.

The present invention addresses this problem.

SUMMARY OF THE INVENTION

According to the present invention, a transport decoder system, designed for decoding a transport datastream which includes transport packets each having a predetermined synchronization pattern and a predetermined number of bytes, is coupled to an NRSS Smart Card for decryption of at least part of the transport datastream. The NRSS Smart Card uses a received synchronization pattern to signify when, in the data stream, decryption operations should start. The present invention implements an interface between the transport decoder and the NRSS Smart Card by receiving a transport datastream; detecting a synchronization pattern and setting a counter based on a predetermined number of bytes which should exist between synchronization patterns; forwarding the synchronization pattern and predetermined number of bytes to the NRSS Smart Card; and if, at the end of the predetermined number of bytes forwarded to the NRSS Smart Card a second synchronization pattern is not detected, forwarding a stream of zeroes to the NRSS Smart Card until the next synchronization pattern is detected.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is best understood from the following detailed description when read in connection with the accompanying drawing, in which:

FIG. 1 shows a high-level functional block diagram of an exemplary digital transport decoder and its various interfaces.

FIG. 2A shows a high-level functional block diagram of an exemplary implementation of a the transport decoder shown in FIG. 1A.

FIG. 2B shows a high-level data/control flow block diagram of an exemplary implementation of a the transport decoder shown in FIG. 1A.

FIG. 3 shows a functional block diagram of the memory controller used in the transport decoder shown in FIG. 2.

FIG. 4 shows an exemplary flowchart illustrating how the memory controller handles Pointers FIFO Entries.

FIG. 5A shows an exemplary flowchart illustrating steps executed during the video decoder interface aspect of the present invention.

FIG. 5B shows an exemplary flowchart illustrating steps executed during the audio decoder interface aspect of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

Turning to the figures, FIG. 1A shows a high-level functional block diagram of an exemplary digital transport

decoder **110** and its various interfaces. As shown in FIG. 1A, transport decoder **110** includes connections to a physical layer channel interface (channel interface) **112**, buffer memory **114**, a host microprocessor **116**, external video and audio decoders **118** and **120**, and clock signal circuitry (e.g., VCXO) **122**. In the exemplary embodiment of the present invention, the transport decoder **110** supports one video interface and up to two audio interfaces (e.g., audio A and audio B).

Host microprocessor **116** is also coupled, external to transport decoder **110**, to the video and audio decoders **118** and **120** by way of microbus **124**. This connection may use parallel or serial data paths and depends on the requirements of the individual external decoders selected for use. In any event, as will be appreciated by those skilled in the art, this method of coupling the host microprocessor **116** to external decoders is well known.

Generally, the transport decoder **110** and each of the external video and audio decoders **118** and **120** include a System Time Constant (STC) value (not shown) maintained in a register or counter for purposes of synchronizing the decoding and display of the received data stream.

In the present invention, microprocessor **116** has an important function in controlling the synchronization of the datastream and the operation of the video and audio decoders **118** and **120**. In the present invention, microprocessor **116** has direct, memory-mapped access to various on-chip registers within the transport decoder **110**. Microprocessor **116** also has access to buffer areas in the external memory **114** (through read, write and watermark pointers) and to on-chip 32-byte Read and Write FIFOs (shown in FIG. 3). Microprocessor **116** is coupled to several internal functional blocks by an internal microbus **223** (shown in FIG. 2B) and allows for various event and error conditions to be signaled via maskable interrupts.

FIGS. 2A and 2B show high-level functional block diagrams of an exemplary implementation of the transport decoder **110** shown in FIG. 1. With reference to FIGS. 2A and 2B, the overall data flow in the transport decoder **110** is described.

As shown in the figures, the channel interface **112** of FIG. 1A leads to a channel data processor (CDP) **210**. Channel interface **112** also provides data to an NRSS interface **212** for purposes of decryption by the NRSS Smart Card **230**. The interconnections and operation of the NRSS interface **212** are described below in detail.

The CDP **210** is provided to preprocess (e.g., detect, synchronize and validate) the packets in the received datastream. The CDP **210**, after performing its processing, forwards its output data to transport processor (TPROC) **214** for further processing. TPROC **214** performs various processing, such as parsing, in accordance with a PID Table described below in detail.

The TPROC **214** interacts with several of the functional blocks within transport decoder **110** including a Memory Controller **216**, a Clock Manager **218**, a Private Data Processor **220**, and a High Speed Data Port **222**. The microprocessor **116** has direct access to various on-chip registers (not shown). This access is gained by way of a Micro interface **217** and the internal microbus **223** (shown in FIG. 2B). Memory Controller **216**, in addition to being coupled to TPROC **214**, is also coupled to external memory **114** and video and audio decoders **224**, **226** and **228**.

The details of the processing of the digital datastream, with respect to the individual functional blocks, is now described.

Channel Data Processor (CDP) 210

It is noted that the datastream from the channel interface **112** is passed to the CDP **210**, after possible buffering (not shown).

The CDP **210** performs various preprocessing steps on the received datastream. In particular, CDP **210** determines the byte and frame boundaries in the datastream and converts the received datastream into an 8-bit parallel format. In the exemplary embodiment of the present invention, operations occurring in the transport decoder **110** subsequent to the CDP **210** are performed using a byte format.

To determine byte and frame boundaries, the CDP **210** synchronizes to a framing pattern contained in the header of a Transport Packet. In particular, this synchronization is achieved by searching for the MPEG-2 Transport Stream sync_byte (e.g., 47 hex) and verifying its occurrence at the start of a programmable number of consecutive transport packets.

In doing so, the CDP **210** checks for the sync_byte, "locks" on to the datastream and continues validating packets. When the sync_byte search is in progress, a Search_State bit is set in a CDP Status Register (not shown). When the sync_byte has been found and verification of successive occurrences is in progress, a Sync_State bit is set. Finally, a Lock_State bit is set when a predetermined search criterion is met. In the exemplary embodiment of the present invention, the search criterion is three consecutive validated packets. Also, a Sync_Hysterisis value, indicating the number of packets with corrupted sync_bytes that the CDP **210** is programmed to allow to pass through before declaring loss of Lock_State, is established. When this hysteresis value is exceeded, a loss of lock is declared. The CDP **210** then issues a Lock_Lost interrupt and transitions from the Lock_State back to the Search_State.

Additionally, if the channel interface **112** (shown in FIG. 1A) provides a CHPSTRT signal, the transport decoder **110** may be configured to use this signal as a means to synchronize to the datastream by setting a Framing_Mode bit in a CDP Framing and Sync Control Register to logical "1". In this case, synchronization is achieved within one transport packet time after the CHPSTRT is asserted.

Alternatively, if an Ignore_Sync bit, in the CDP Framing and Sync Control Register (not shown), is set to logical "0", the CDP **210** verifies sync_byte integrity at the expected (first) byte location relative to the CHPSTRT signal transition and declares a lock condition if verification fails. If Ignore_Sync is set to 1, the CDP **210** does not verify sync integrity.

After achieving the Lock_State, the output of the CDP **210** can be passed to and processed by the TPROC **214**, which further processes (e.g., filters, parses, and formats) the data now in byte format.

Transport Processor (TPROC) 214

Signals produced by the TPROC **214** control the flow of data to the Memory Controller **216**, the Micro Interface **217**, the Clock Manager **218**, the Private Data Processor **220** and the High Speed Data Port **222**.

In the exemplary embodiment of the present invention, the TPROC **214** processes the data stream under control of a PID Table. As described in the Systems Section, a PID is a 13-bit field in a Transport Stream Header, indicating the type of data stored in the packet payload. Some PID values are assigned and some are reserved. Additional details of the PID Table are described in Section 3.3 of the Systems Section of the above-referenced MPEG-2 specification.

Regarding the PID Table, the TPROC **214** is capable of concurrently processing up to 32 user-selectable PIDs. The PIDs which the transport decoder **110** is to process are specified in the PID Table. Bit fields in the PID Table specify various processing options for the data.

In operation, the PID is extracted by the TPROC **214** from the header of an incoming transport packet and simultaneously compared to all of the entries in the PID Table. If a match is found, the data in the packet is processed according to the options set in the matching PID Table entry. If no PID Table entry matches, the packet is discarded.

Each location in the PID Table for which the Payload Format bits are not set to Discard (000), is associated with a channel number. In conjunction with the TPROC **214**, the Memory Controller **216** is responsible for transferring data to a specific buffer area (channel) in RAM. Operation of the Memory Controller **216** is discussed in detail below with reference to FIGS. **2**, **3** and **4**.

In the exemplary embodiment of the present invention, the PID Table, maintained in TRPROC **214**, is set up under software control by the microprocessor **116** via the internal microbus **223** during an initialization process.

Regarding PID Table entry format, a PID Table entry includes, among other information, 1) a 13-bit PID and 2) a 1-bit PCR PID field which, when set, indicates that this PID carries the PCR for the program being decoded. An exemplary PID Table entry also includes a 1 bit PES HDR bit which, when set, enables the RAM buffer address corresponding to a "Stream ID" in the PES header to be stored to a Pointers FIFO. An interrupt is issued by the Memory Controller **216** when it writes data to the Pointers FIFO. This interrupt is referred to as a DMA_MARK signal which serves to alert the microprocessor **116** of the delivery of the Stream_ID byte. In the exemplary embodiment of the present invention, Channel Numbers are assigned to PID Table entries in numerical order based on their location in the table, for example, the first entry is assigned DMA Channel **0**, the next one DMA Channel **1** and so on.

The TPROC **214** indicates that transport packets with PCR PID should be processed by the Clock Manager **218** by placing a PCR_PID signal in the PID Table entry. The Clock Manager **218** then monitors the adaptation fields of such packets for a PCR_flag and extracts the PCR.

After separating the packets according to their PIDs, the TPROC **214** removes the headers of the transport packets and, in conjunction with the Memory Controller **216**, stores packet payloads (e.g., portions of PES packets) for the respective PID's in sequential memory locations of the designated channel in the external memory **114**. The data in the memory **114** represents PES packets.

The TPROC **214** also handles write operations to the external memory from the host microprocessor **116**.

MEMORY CONTROLLER 216

The Memory Controller **216** stores the parsed payload of Transport Packets in external memory **114** and provides video, audio, PSI and other data (e.g., private data) upon demand to the Video Processor **224**, up to two Audio Processors **226** and **228** and to the host microprocessor **116** (or microcontroller).

FIG. **3** shows an exemplary functional block diagram of a Memory Controller suitable for use in the transport decoder **110** shown in FIGS. **2A** and **2*b***. As shown in FIG. **3**, the Memory Controller **216** receives data from the TPROC **214** via a Transport In FIFO **310**. Memory Controller **216** receives/transmits data from the microprocessor **116** via Micro Interface **217** including Micro-in FIFO **314** and Micro-out FIFO **316**. It is noted that the Micro Interface **217**, shown external to Memory Controller **216** in FIGS. **2A** and **2B**, is shown in FIG. **3** for the sake of clearly illustrating its interrelationship with FIFOs **314** and **316**.

The Memory Controller **216** also includes a Flow Control Unit **318** containing On Chip Memory, a DMA Controller **320**, Datapath logic **322**, and FIFOs **324**, **326** and **328** for delivering data to the video and audio decoders **224**, **226** and **228**, respectively.

The Memory Controller **216** writes data into external memory from two sources: one is the TPROC **214** (through the Transport In FIFO **310**) and the other is the microprocessor **116** (through the Micro In FIFO **314**).

The Memory Controller **216** reads data from external memory and supplies it to the following four destinations 1) Video Interface (through the Video FIFO **324**), 2) Audio A Interface (through the Audio A FIFO **326**), 3) Audio B Interface (through the Audio B FIFO **328**), and 4) microcontroller **116** (through the Micro Out FIFO **316**).

The external memory space is divided into separate, non-overlapping buffer areas, one for each DMA Channel in use. Up to 32 channels can be used. A channel number is associated with data written into the two source FIFOs or read out of the four destination FIFOs.

Data provided by the TPROC **214** and written into the Transport In FIFO **310** carries a channel number tag. The channel number for each of the other FIFOs is defined by the microcontroller **116** in a DMA Interface Register. A 128-bit DMA Channel Configuration Register (Channel Register) is set up by the microprocessor **116** for each channel in use. The Channel Register contains configuration information such as the buffer space defined by Begin and End Addresses, whether the buffer is configured as a FIFO (read and write pointers wrap around, sometimes referred to as a circular queue) or as a Queue (read and write pointers do not wrap around), etc.

If the Channel Register allows; the DMA Controller **320** can obtain a new buffer when the one it is currently using becomes full. This feature is normally used only for buffers which store data that is destined for the host microprocessor **116**, such as PSI sections. The Begin and End Addresses of these new buffers are stored in a New Buffer FIFO. Only the microprocessor **116** can write new buffers to this FIFO. Used buffer pointers and other types of pointers are written by the DMA Controller **320** to the Pointers FIFO. In the exemplary embodiment of the present invention, only the microprocessor **116** can read the Pointers FIFO. These pointers are used by the microprocessor **116** to manage reading of buffer data and to return used buffers back to the New Buffer FIFO. The Pointers FIFO Entries Flowchart, shown in FIG. **4**, shows the entries to the Pointers FIFO for all possible signal conditions, Channel Register configurations and buffer conditions.

Continuing with FIG. **3**, the DMA Controller **320** and Datapath logic **322** provide all interface signals to the external memory. The Flow Control Unit **318** directs the DMA Controller **320** to service a specific interface FIFO (one of six) by providing the appropriate control signals to it. These include:

1. DMA Controller Start

This signal commands the DMA Controller **320** to start servicing a specific FIFO using the commands such as DMA Controller Operation, DMA Input FIFO Select and DMA Output FIFO Select. At the same time, Flow Control Unit

318 reads out the 128-bit Channel Register and makes it available on the data bus of the Channel Register (internal memory). The DMA Controller 320 reads this 128-bit word into its internal register and the Flow Control Unit 318 is then free to do other read/write operations as required on the internal memory.

2. DMA Controller Operation (1 bit)

This bit signals a Write or Read operation to be performed by the DMA Controller on the interface FIFOs.

3. DMA Input FIFO Select (1 bit)

Signals which of the two input FIFOs 310 or 314 to service.

4. DMA Output FIFO Select (2 bits)

Signals which of the four output FIFOs 316, 324, 326 or 328 to service.

5. Channel Data In Request/Channel Data/Channel Number/DMA Controller Stop

Usually, the DMA Controller 320 continues to service a FIFO until it cannot service the FIFO any longer due to some condition such as a full interface FIFO, or no data in external memory, etc. When this type of condition occurs, the DMA Controller 320 asserts the Channel Data In Request to the Flow Control Unit 318 and provides the Channel Register data and Channel Number data to the Flow Control Unit 318. The Flow Control Unit 318 updates the Channel Register memory with the Channel Data (updated information includes, for example, the Write and Read Pointers) and directs the DMA Controller 320 to service the next FIFO in the loop, unless a specific FIFO needs emergency action (because, for example, it may be approaching a full or empty status).

Occasionally, the Flow Control Unit 318 may need to issue a DMA Controller Stop command to the DMA Controller 320 in response to a need to service another FIFO such as the Transport In FIFO which may have a higher priority. This command directs the DMA Controller 320 to stop servicing the FIFO it is currently servicing and write the Channel data to the Flow Control Unit 318.

The Channel Register is updated by the DMA Controller 320 with a Read Pointer, a Write Pointer and other information at the point at which it stops servicing a FIFO. Since the Flow Control Unit 318 writes this to the internal Channel Register memory, the DMA Controller 320 can resume service to the interrupted FIFO from the point at which it was interrupted.

With reference to the DMA Controller's participation in the overall operation of the discontinuity state aspect of the present invention, when a next PES packet containing a Stream ID byte is written to the Memory Controller 216 by TPROC 214, the Flow Control Unit 318 causes the DMA_MARK signal to go high. The transition of the DMA_MARK signal triggers an interrupt at microprocessor 116 and the address of the PES packet header is stored in an input pointer FIFO to which the microprocessor has access. Since this PES header may have a time-stamp, the microprocessor 116, responsive to the flag raised by the discontinuity indicator, searches the stored PES headers for a time stamp and, when it finds one, retrieves the time stamp value. Based on the retrieved timestamp value, microprocessor 116 sets a timer interrupt.

Referring back to FIG. 3, in the exemplary embodiment of the present invention, internal memory of the Flow Control Unit 318 includes four blocks of 64×32 bits, each of which are assigned for Channel Registers, New Buffer FIFO and Pointers FIFO.

Each Channel Register is 128 bits and there may be a maximum of 32 channels, so up to 128×32 bits can be assigned to Channel Registers, in multiples of 128 bits.

Words of 32-bits are written to and read from the New Buffer and Pointers FIFOs. At least 128×32 bits of memory (i.e., 128 words) are available for the New Buffer and Pointer FIFOs. Additionally, if less than 32 channels are in use, unused capacity from the Channel Register memory can be allocated to the New Buffer and Pointer FIFOs.

Following are examples of memory allocation for 32 DMA channels and 16 DMA channels.

1) Memory allocation for 32 used channels
   New Buffer FIFO=64×32 bits
   Pointer FIFO=64×32 bits
   Channel Registers=128×32 bits

2) Memory allocation for 16 used channels
   New Buffer FIFO=64×32 bits
   Pointer FIFO=128×32 bits
   Channel Registers=128×16 bits

CLOCK MANAGER 218

The Clock Manager 218, shown in FIGS. 2A and 2B, of the transport decoder 110 synchronizes a local 27 MHz signal (generated by clock signal circuitry 122) using successive PCR values. The value of the counter in the Clock Manager 218 is referred to as the System Time Constant (STC) value. The microprocessor 116, in the exemplary embodiment of the present invention, has access to the STC value in the counter via the internal microbus 223.

The counter in the Clock Manager 218 is periodically synchronized to the PCRs carried in the transport data stream. When the first PCR in a data stream is encountered, the PCR is "jam loaded" in the counter to become the initial STC value. As subsequent PCR values are received, they are compared to the STC value to adjust the frequency of the clock signal circuitry 122 (e.g., VCXO). In the present invention, this is accomplished using a Digital-to-Analog converter (DAC) described in detail below. Once phase error adjustments are made, generally, the new PCR value is loaded as the STC value.

Normally PCR values occur once per frame. Under the MPEG-2 standard, they must occur at least in every third frame (one-tenth of a second).

The external video and audio decoders 118 and 120 are typically synchronized to the transport decoder 110 by updating the STC value in the external decoders based on the STC value maintained in the Clock Manager 218. The video and audio decoders 118 and 120 use the stored value to determine when data is to be decoded/displayed by comparing it to timestamps in the received PES datastream. It is noted that, in the exemplary embodiment of the present invention, the transport decoder 110, video decoder 118 and audio decoder 120 all receive the same 27 MHz clock signal generated by the Clock Manager 218.

The transport decoder 110 contains hardware support for locally generating a clock signal that is locked to the system clock signal for a selected program. As mentioned above, PCR values received on the specified PCR_PID are made available on a serial interface for use by other devices.

The Clock Manager 218 includes a System Time Counter (STC) (not shown), a System Time Clock Register (not shown), a Program Clock Reference Register (not shown), a Current STC Register (not shown), an Alarm Clock Interrupt Register (not shown) and logic (not shown) to latch PCRs extracted from the headers of packets that are associated

with a selected PID. It also contains the serial PCR interface, and a DAC Register for control of an external system clock signal control loop.

In the exemplary embodiment of the present invention, the STC is a 42-bit counter having a 9-bit base and 33-bit extension.

The STC base divides the nominal 27 MHz System Clock input signal by 300. The resulting 90 KHz signal drives the STC extension. The STC can be read or written by software executing on the microprocessor **116** at any time. Optionally, the STC can be automatically loaded with a received PCR after a discontinuity indicator has been encountered.

As mentioned above, PCRs may be automatically extracted from the incoming bit stream by setting the PCR_PID bit in a PID Table entry. Handling of the received PCRs depends upon options set by software and whether a discontinuity state exists for the current transport packet when the PCR is received.

A discontinuity state is recognized when a transport packet which contains the PCR_PID is received with a logic 1 in the discontinuity_indicator bit in the Transport header. The discontinuity state exists until either a transport packet containing the PCR_PID is received with a logic 0 in the discontinuity_indicator, or until the next PCR is found in a PCR_PID packet.

If enabled, the Clock Manager **218** automatically loads the received PCR into the STC when a PCR is received while a discontinuity state exists. At this point, an interrupt is issued to the host microprocessor **116**.

The Clock Manager **218** contains an Initialize bit which, when set to logic 1 by software, causes the next PCR received to be loaded into the STC regardless of whether a discontinuity state exists or not. The Initialize bit is subsequently cleared automatically by the Clock Manager **218**. The Initialize bit is set only when the STC does not contain a valid PCR (i.e., when a new PID has been selected.)

If no discontinuity state exists or if the automatic STC loading function is disabled, a received PCR is stored in the PCR register and (optionally) an interrupt is issued to the host microprocessor **116**. At the same time that the PCR is captured, the current state of the STC is stored in the STC Register. The stored PCR and STC values are used to compute an error signal to control the external system clock signal (XOCLK).

When a Transport Packet is received which has a PID equal to the PCR_PID and which contains a PCR, the PCR is latched in the PCR Register. The current contents of the STC Counter are also latched in the STC Register at that time, and an interrupt is issued to the host microprocessor **116**. Latching both values eliminates any uncertainty due to varying latency in the interrupt service routine. In the exemplary embodiment of the present invention, both registers consist of a 9 bit base and 33-bit extension. The base part is modulo **300**, whereas the extension is binary. Before the register contents can be used in any calculations, they are converted to a single binary count by multiplying the extension by **300** and adding the base part to the result.

The Current STC Register is a read-only register which allows the microcontroller **116** to obtain the current contents of the STC at any time. The microcontroller **116** can initialize the STC of the Video Decoder or Audio Decoder with this value by way of the external microbus **124**.

As mentioned above, transport decoder **110** contains a 10-bit digital-to-analog converter (DAC) **230** for use in the clock adjustment control loop within the Clock Manager **218**. The DAC **230** is write-only, i.e., software keeps a copy of the last value loaded into the DAC in memory. In operation, the DAC **230** is used to generate the control voltage for the external resonant crystal voltage controlled oscillator, VCXO **122**, which is source of the system clock frequency (27 MHz nominal). By comparing the latched PCR and STC values, software can determine whether the local clock signal requires adjustment. For example, software may take the difference between the PCR and STC and add this error term to the current contents of the input register of DAC **230**.

VIDEO/AUDIO DECODERS 224, 226, 228

Once the transport packets are received, processed and stored, as mentioned above, Memory Controller **216** reads the PES packets out of memory **114** and forwards them to the external video and audio decoders. Before the PES packets reach the external decoders **118** and **120**, however, they are processed by internal video and audio processors **224**, **226** and **228**.

An important role of these internal processors is to ensure that the stream of data leaving the transport decoder is compatible with the external decoders. That is, different external decoders have different data requirements.

For example, some currently available external video decoders can accept a stream of PES packets, including header and payload, essentially in the same form in which it is read out of memory. On the other hand, some external decoders can only accept for processing an elementary stream (i.e., PBS packet payloads but no headers). Additionally, the latter type of decoder may require, in addition to the datastream, associated control information such as timestamps and byte counts. One such video decoder is the CL9100 made by C-Cube, the specification of which—CL9100 Multimode Video Decoder User's Manual (October 1994)—is hereby incorporated by reference. Accordingly, video processor **224** is configurable, by way of microprocessor **116**, to format the outgoing datastream to be compatible with the requirements of the selected external decoder.

With respect to handling the interface to an external video decoder such as a C-Cube CL9100, video processor **224** processes the stream of PES packets in order to strip off the PES header and extract, from the PES header, timestamp information. In addition, video processor **224** maintains a byte count, using a counter (not shown), for the PES packets forwarded to the external decoder. Thus, video processor **224** holds timestamp information, maintains a byte count and forwards the PES packet payloads on to the external video decoder. In order to get the timestamp and byte count information to the external decoder, video processor **224** sends an interrupt to microprocessor **116** which, when it responds, reads the timestamp and byte count information from the video processor **224** via the internal microbus **223** and delivers this control information to the external video decoder via the external microbus **124**.

Sometimes, however, before microprocessor **116** is able to respond to the interrupt from video processor **224**, another PES header with timestamp information arrives. This may occur, for example, if the microprocessor **116** was involved with a higher priority task for an extended period of time. In this case, video processor **224** stores the timestamp information along with a status. In particular, video processor **224** maintains a status register indicating whether microprocessor has "missed" any timestamps. When, in the exemplary embodiment of the invention, microprocessor **116** finally

begins to read the timestamp information, if it has missed one or more timestamps as indicated in by the status, it only retrieves the most recent timestamp.

Additionally, since timestamp information and byte count information are to be provided to the external decoder, when the microprocessor **116** misses a timestamp, video processor **224** maintains a cumulative byte count for however many timestamps may arrive before microprocessor **116** reads the information. Thus, microprocessor **116** reads the most recent timestamp and cumulative byte count and delivers this information to the external video decoder. Subsequently, the status register and byte count may be reset.

As shown in the flowchart of FIG. **5A**, the headers of received data are stripped and parsed (step **510**), timestamp information is stored and a byte count is maintained (step **512**), an interrupt is issued by video processor **224** to microprocessor **116** alerting it that timestamp information is available for reading (step **514**), if additional timestamp information is received before microprocessor reads the previous timestamp information, the new timestamp information is stored, the byte count continues to accumulate (step **516**) and the video processor **224** issues another interrupt to the microprocessor **116**, otherwise microprocessor reads the status register, the timestamp information and the byte count information and delivers it to the external video decoder (step **518**).

Another example of accommodating particular external decoders involves the use of bit rate-constrained audio decoders such as the Zoran ZR38500 AC3 audio decoder, the Specification (October 1994) for which is hereby incorporated by reference. Generally, most audio decoders present a data request indicating they can accept audio data and then the transport decoder **110**, via the audio processors **226/228**, supplies audio data until the external audio processor negates the requests. In other words, there exists a handshake mechanism.

Bit-rate constrained decoders, however, such as the Zoran AC3 audio decoder, require that audio AC3 data be delivered as audio data frames, each frame including a frame time and bit-rate in its header. Delivering data in excess of this predetermined bit rate can cause the internal buffers to overflow and, consequently, cause a loss of data. Furthermore, these bit-rate constrained decoders do not provide a Data-Ready signal to indicate that they are ready to accept additional audio data.

Thus, as with the video decoder **224**, the audio processors **226** and **228**, are configurable to accommodate the selected external audio decoder. It is noted that, in the exemplary embodiment of the present invention, audio processors **226** and **228**, except for addresses which allow access by microprocessor **116**, are essentially identical.

Accordingly, audio processors **226/228**, when programmed to interface with this type of audio decoder, assumes that the decoder is always issuing a data request signal. However, as PES packet audio data is supplied to audio processors **226/228** from Memory Controller **216**, the PES header of the received data is parsed by the audio processors. Contained in the parsed header is a frame size and bit rate for the audio data. From the frame size and bit rate, the audio processors **226/228** can determine the frame boundary (i.e., frame time). Having frame time information, audio processors **226/228** set a frame timer such that, when audio data is available, the timer is set, a frame of data is "burst" out to the external audio decoder then the audio processors **226/228** and then pause until the frame timer expires. When the timer expires, the cycle continues, one audio frame at a time, while additional frames are available for sending to the external audio decoder. Control information, such as timestamps, are retrieved by the microprocessor and delivered via the external microbus.

As shown in the flowchart of FIG. **5B**, PES packets are retrieved from memory (step **542**), the headers of received data are stripped and parsed (step **530**), a frame time is determined (step **532**), the timer begins running (step **534**), audio processors **226/228** burst out a frame of data (step **536**) and audio processors **226/228** pause until the timer expires (step **538**), although, as shown, steps **542**, **530** and **532** can be carried out during the pause. Then, the process can continue.

NRSS INTERFACE 212

Referring back to FIGS. **2A** and **2B**, the NRSS Interface **212** is described in more detail. It is noted that the NRSS Interface **212** is designed to be compliant with the NRSS Smart Card specification.

In particular, NRSS Smart Card **230** has a limitation in that there are only three pins with which to carry out the data transfer. These pins, represented as signal lines on FIG. **2A**, are NRSSDOUT, NRSSDIN and NRSSDCLK. NRSSDOUT is the signal line on which serial data leaves the transport decoder **110** and is supplied to the NRSS Smart Card **230**; NRSSDIN is the signal line on which serial data leaves the NRSS Smart Card **230** and is supplied back to the transport decoder **110**; and, NRSSDCLK is the signal line on which a data clock is provided such that data is transferred, in either direction, on the edge of this clock. The other signal lines include a relatively slow bi-directional communication line NRSSIO, a system clock NRSSCLK and reset NRSSRST. It is noted that generally the NRSS Smart Card **230** includes a microprocessor which has a software encryption engine. And, microprocessor **116** can re-program various parameters of the Smart Card **230** if necessary.

Regarding the dataflow, when decryption is unnecessary, the dataflow is as described above where the CDP **210** receives the datastream, checks for the sync_byte (47 h) and formats the data in byte format.

However, when decryption is necessary, NRSS Interface **212** receives the datastream and performs essentially the same sync_byte detection function as the CDP **210**. NRSS Interface **212** performs this function in order to determine incoming packet boundaries. That is, each time a new packet arrives, as indicated by a sync_byte, the data is sent to the NRSS Smart Card **230** for decryption. In doing so, valid data, beginning with the sync_byte, is sent to the NRSS Smart Card **230**. As mentioned, the first byte of data serially delivered to the Smart Card **230** is the 47 hex. The Smart Card **230** recognizes this predetermined pattern, synchronizes to it and decrypts the next 188 bytes.

However, it is desirable to only send valid data to the NRSS Smart Card **230** because the 47 hex pattern could exist in some random data stream allowed to pass to the Smart Card **230**. If this were to occur, the Smart Card **230** would synchronize to a false sync_byte and potentially miss a true sync_byte, thus, failing to properly decrypt the valid encrypted data.

To avoid such a problem, the NRSS Interface **212** is designed to ensure proper processing. In particular, NRSS Interface **212**, after detecting a proper sync_byte, starts a counter in order to track the predetermined number of bytes included in this packet (e.g., 188 bytes): At the end of the 188 bytes, if a new sync_byte is not detected, the NRSS Interface **212** forces a stream of zeroes to the NRSS Smart

Card **230** to ensure that the Smart Card does not encounter any false sync_byte and begin decrypting. It is noted that NRSS Interface **212** can be programmed to handle a change in the sync_byte pattern or even the elimination of the need for a sync_byte.

Additionally, if the transport packet stream is carrying multiple programs and, for example, only one of the programs requires decryption, the NRSS Interface **212** instructs the NRSS Smart Card **230** which data to decrypt and which data not to decrypt by way of the NRSSIO bi-directional communications signal line. Typically, such a communication is handled by microprocessor **116** via internal microbus **223**, through NRSS Interface **212**, and onto the NRSS Smart Card **230** via the NRSSIO communications line. Also, microprocessor **116** can set various parameters in the NRSS Interface **212** such as baud rate, stop bits, parity, etc., by way of control registers (not shown) Which are also accessed via the NRSSIO communications line.

Finally, the decrypted serial datastream is passed to CDP **210** which formats the data in byte format and forwards it to the TPROC **214**. This decrypted data stream is handled, as described above, in exactly the same way as if an unencrypted data stream had been received from the channel interface **112** (shown in FIG. **1**A).

Although the invention is illustrated and described herein as embodied in a method and apparatus for updating the system time constant following a discontinuity in an MPEG-2 transport data stream, the invention is not intended to be limited to the details shown. Rather, various modifications may be made in the details within the scope and range of equivalents of the claims and without departing from the spirit of the invention.

The invention claimed is:

1. In a transport decoder system designed for decoding a transport datastream which includes transport packets each having a predetermined synchronization pattern and a predetermined number of bytes, said transport decoder coupled to an NRSS Smart Card for decryption of at least part of the transport datastream, said NRSS Smart Card using a received synchronization pattern to signify a start of decryption, a method for handling the interface between the transport decoder and the NRSS Smart Card comprising the steps of:

receiving a transport datastream;

detecting a synchronization pattern and setting a counter based on the predetermined number of bytes;

forwarding the synchronization pattern and predetermined number of bytes to the NRSS Smart Card; and

if, at the end of the predetermined number of bytes forwarded to the NRSS Smart Card no synchronization pattern is detected, forwarding a stream of zeroes to the NRSS Smart Card until the next synchronization pattern is detected.

2. In a transport decoder system designed for decoding a transport datastream which includes transport packets each having a predetermined synchronization pattern and a predetermined number of bytes, said transport decoder coupled to an NRSS Smart Card for decryption of at least part of the transport datastream, said NRSS Smart Card using a received synchronization pattern to signify a start of decryption, a system for handling the interface between the transport decoder and the NRSS Smart Card comprising:

means for receiving a transport datastream;

means for detecting a synchronization pattern and setting a counter based on the predetermined number of bytes;

means for forwarding the synchronization pattern and predetermined number of bytes to the NRSS Smart Card; and

means for forwarding a stream of zeroes to the NRSS Smart Card until a next synchronization pattern is detected.

* * * * *