# Design of an MPEG-2 transport demultiplexor core

by R. E. Anderson
E. M. Foster

**A new high-performance MPEG-2 transport demultiplexor hardware architecture is presented which minimizes the support required from a host processor for common tasks such as clock recovery and table section filtering, yet provides an interface for a processor to modify the handling of the incoming transport stream. A common on-chip SRAM is used for temporary storage of transport packets as they are processed, as well as for control and status registers, allowing space for efficient storage and reducing silicon size. Memory storage is further reduced by delivering data directly to the audio and video decoder rate buffers and to system memory as part of a total, integrated MPEG subsystem design. The architecture is implemented as part of the IBM Blue Logic™ core library.**

## Overview

Digital television offers viewers high-quality audio and video. For broadcasters, the compression of data allows several programs to be delivered over the same analog bandwidth. Digital television can be distributed using adapted versions of current analog systems, including satellite, terrestrial, and cable. A terrestrial system is illustrated in **Figure 1**. The audio and video components of a program are compressed at the source and time-multiplexed with other programs and system information needed to recreate the original program. The digital multiplex is modulated and transmitted to the subscriber's home. The set-top box (STB) demodulates the signal to recover the multiplexed digital streams, extracts the program of interest, and decodes the compressed audio and video for presentation on the television.

There are several possible variations of this system. The digital stream can also be modulated and broadcast over a coaxial cable system currently installed or delivered over satellite systems. In addition, the receiver may be a stand-alone STB, as described above, or it may in time be integrated directly into the television.

There are a variety of possible methods of encoding and delivering digital television programs. To promote the development of interoperable components from different manufacturers, the MPEG-2[1] international standard [1–3] was developed. The standard does not specify the techniques for encoding, multiplexing, and decoding the bit streams, but only the format of the data. This leaves an opportunity for manufacturers to differentiate their products through the way in which they use resources such as silicon, processor power, and memory, and through their ability to conceal or recover from errors. The

[1] MPEG—Moving Picture Experts Group.



0018-8646/99/$5.00 © 1999 IBM

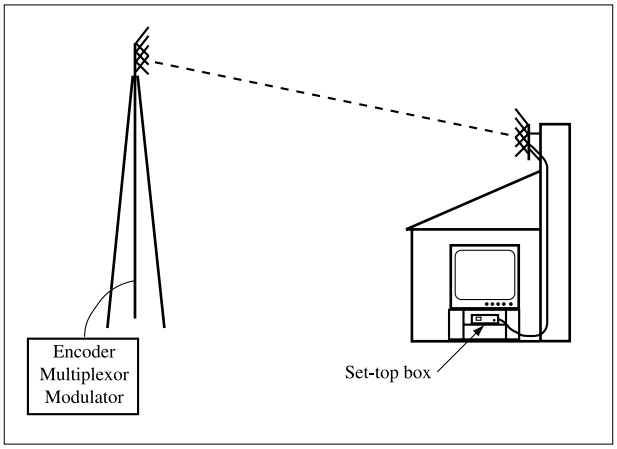


**Figure 1**

Example of digital television terrestrial distribution.

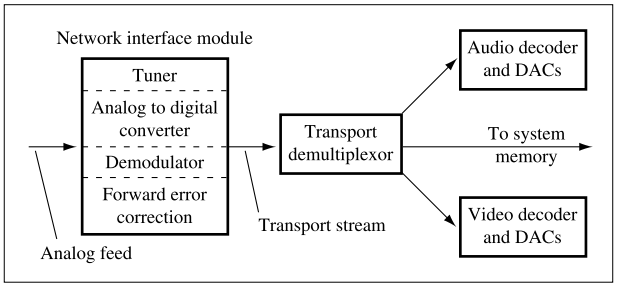


**Figure 2**

STB data flow.

standard is composed of three primary parts covering systems, video, and audio. The video and audio parts specify the format of the compressed video and audio data, while the systems part specifies the formats for multiplexing the audio and video data for one or more programs as well as information necessary for recovery of the programs.

The MPEG-2 systems standard [1] specifies two stream formats: one for error-free environments such as a digital storage medium, and one for error-prone environments such as satellite, cable, ATM, and other networks. The latter format, often referred to as the transport stream, is used for broadcast applications such as the one shown in Figure 1.

An integral part of the broadcast distribution is the reception and subsequent processing of the data in the user's home. **Figure 2** is simplified block diagram of an STB from Figure 1 showing the flow of program information. A tuner extracts the analog signal, which is then digitized. The demodulator recovers the symbols, which represent the incoming bit stream. The bit stream contains additional data for forward error recovery. This group of four functions is often referred to as the network interface module (NIM). The final output of the NIM is an MPEG-2 transport stream. The transport demultiplexor extracts the audio and video portions of the program to be sent to the audio and video decoders. The data which specifies the stream content is delivered to memory to enable the processor to configure the STB to deliver a particular program.

The remainder of this paper presents the following topics: The transport stream syntax is introduced, along with the minimum set of processing requirements. The set of requirements is expanded to include those for an MPEG-2 transport demultiplexor implemented as part of the IBM Blue Logic core library [6] for ASICs. The IBM transport architecture is described, along with discussions of the important design and implementation decisions required to meet the requirements.

## Transport-stream syntax

The transport-stream specification for error-prone environments uses short, 188-byte packets. The small packet size minimizes the amount of data which might be lost or corrupted by the delivery network. The packets for a particular stream within the multiplex are intermixed, so that burst errors have reduced impact on individual streams. An important feature of an STB is its ability to manage and conceal or at least minimize errors as seen by the user. The MPEG-2 specification does not specify error handling, so each manufacturer decides how errors are managed.

A transport stream is composed of time-multiplexed packets from different streams. Each transport packet contains a header, an optional adaptation field, and a payload, as shown in **Figure 3**. Several of the important header fields are shown in **Figure 4**. The Sync byte can be used to establish the packet boundaries to allow random access into the transport stream. The packet identifier (PID) identifies all of the packets belonging to the same data stream, making it possible to reconstruct the stream within the STB. Packets with the same PID are considered a PID stream. As an example, a program typically requires at least one audio and one video stream. Within the transport stream, the packets containing the audio stream will have the same PID. Similarly, the packets comprising the video stream are identified by the same PID value, which is unique to the video stream.

In addition to the Sync byte and PID fields, each transport packet header also contains a continuity counter (CC) field, which can be used to detect missing packets in

a PID stream (Figure 4). The CC represents a modulo-16 counter which increments by one with each packet of the same PID which contains payload. If the next packet of a PID stream has an incorrect CC value, it is very likely that at least one packet of the PID stream has been lost. There are a few additional rules concerning the CC values which, along with other syntax details, can be found in the MPEG-2 systems standard [1].

An adaptation field may follow the packet header and may contain a program clock reference (PCR), which is used to synchronize the decoding and playback of a program and is described in the section on audio and video synchronization. The payload may follow immediately after the header or after an adaptation field. It may exclusively contain audio data, video data, private data defined by the service provider, or table sections. (Table sections are discussed in the next section.)

**Figure 5** illustrates a short sequence of transport packets from a possible transport stream. Further detail of three of the packets is shown. One of the packets is a video packet with a PID field of 34; this video packet contains not only video data in the payload field, but also a PCR in the adaptation field. The PID field of the other two packets is 21, indicating that they are from the same PID stream. This PID stream contains tables, which are described in the next section.

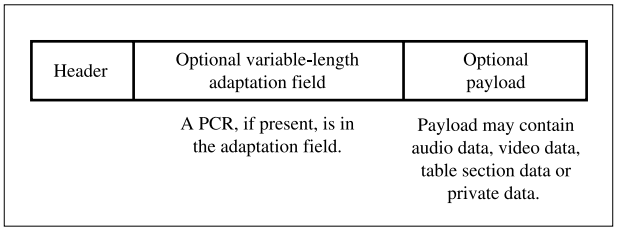


**Figure 3**

Transport packet.

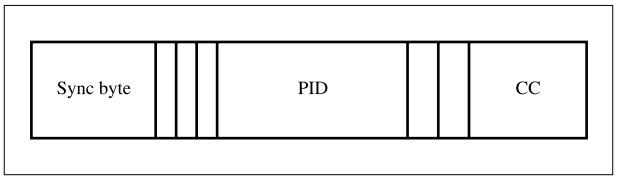


**Figure 4**

Four-byte transport packet header.

- *Table sections*

The transport-stream syntax also defines a table section data structure. The MPEG-2 specification includes several mandatory types of tables which describe the content of the stream. The required tables specify programs in the stream, as well as necessary information such as the audio and video PIDs for each program. The same data structure can be used to send private data, such as a software update for the STB.

A table is transmitted as one or more table sections. The first field in the table section is the table ID, which allows the STB to identify all of the table sections for a table so that the STB can reconstruct the complete table data structure. The table ID allows multiple tables to be transmitted in a single PID stream, as shown for PID 21 in Figure 5, which contains table sections for tables 49, 12, and 71.

There are several important table characteristics which place processing demands on the STB. First, table sections can span multiple transport packets that may be spaced some distance apart in the transport stream. The STB must be able to reconstruct a table section as a continuous structure in memory. Second, the last four bytes of a table section may contain a cyclic redundancy check (CRC), which the STB can use to determine whether the table section was delivered intact. The CRC provides an additional level of data checking beyond the CC in the transport packet header. Third, multiple table sections, or portions of them, may be contained in a single packet. Fourth, table sections are repeated in a stream to permit random access by the STB. Thus, once a particular table section is acquired, it is not necessary to acquire it again until it has changed.

The transport stream arrives in real time, so the STB must be able to process sections as they arrive without risk of data loss or impact on the operation of the rest of the system. The STB must also be able to handle multiple consecutive packets containing several table sections.

- *Audio and video program synchronization*

The MPEG-2 systems standard identifies a constant-delay model between the transport-stream source and the STB. PCRs are inserted into the transport stream as it is created, as shown in the second packet of Figure 5. The PCRs reflect the current value of a counter at the source, incrementing at 27 MHz. Since the transport stream is delivered with constant delay, the PCRs can be extracted to control the frequency of the STB's system time clock (STC). Matching the frequency of the STC to the arriving PCRs ensures that the decoder's audio and video buffers do not overflow or underflow. If either of these conditions occurs, the viewer may see or hear brief disruptions in the program.

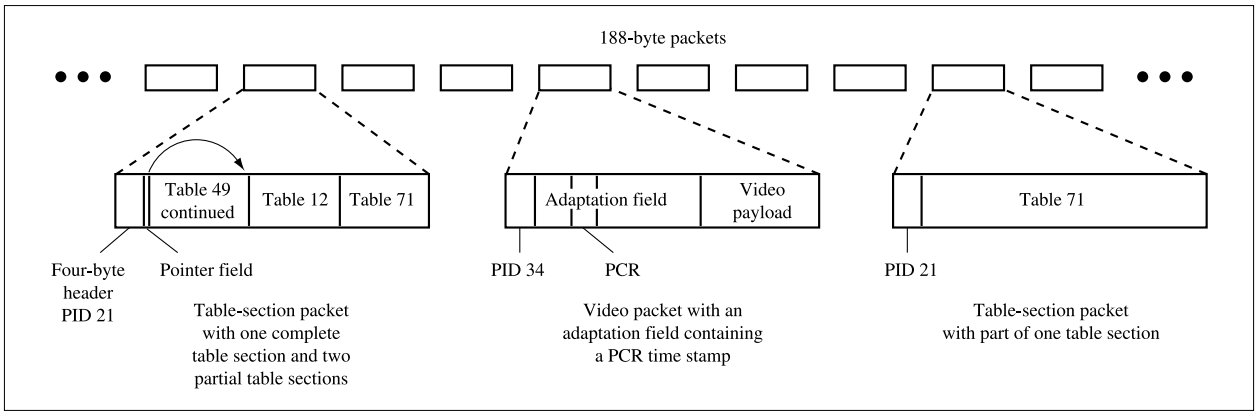


**Figure 5**

Transport-stream example.

- *Transport-stream errors*

The discussion so far assumes that the transport stream is delivered without errors. The short transport-stream packets are intended to minimize the disruption of the system when errors in the delivery system occur. The MPEG-2 systems specification provides a means of detecting errors, but does not define the appropriate STB action when an error does occur.

Errors may manifest themselves as missing or invalid packets. The CC field in the transport packet header allows the STB to detect these events. The possible effect on the system depends on the type of missing data, such as audio, video, table sections, and so on. The audio and video portions of the STB must be designed to continue in such an event by either processing the current partial data and minimizing any temporary artifacts in the output, or discarding the partial data and repeating previous output until good data is available. For tables, all or part of a table section may be lost for each missing packet. Since table sections are generally repeated at regular intervals, the loss of a packet containing a table section does not have a noticeable impact.

- *Transport requirements*

The MPEG-2 systems standard places a set of requirements on all transport demultiplexor implementations. Some practical requirements exist, such as the ability to receive a digital stream from a NIM. The stream interface between the NIM and the transport demultiplexor can be either serial (1 bit) data or parallel (byte-wide) data. If the NIM has not already established the packet boundaries, the demultiplexor must do so using the sync byte. Once boundaries are determined, the MPEG-2 transport syntax defines the bit fields within the packet so that data can be parsed and processed accordingly. Typically, as controlled by the host processor in the STB, the demultiplexor will

- Filter the arriving packets based on a set of PIDs.
- Extract the PCRs and provide a means to ensure synchronization of the audio and video presentation.
- Forward the payload of audio and video packets to the respective decoder.
- Forward program-specific information (PSI) and private tables to system memory for further processing.

In addition, the focus on data broadcasting (non-audio/video streams) and on improved user perception has introduced requirements for flexible filtering of table data to efficiently provide only the necessary information for host processing, and efficient error detection and concealment to prevent noticeable artifacts in the audio or video output.

## Transport design goals

On the basis of the predominance and established definition of the MPEG-2 transport syntax, we have pursued a hardware-based design that yields the greatest performance in the least silicon area. The design goals summarized in the following sections influenced the architectural decisions of the IBM transport.

- *Integrate with other cores in the IBM Blue Logic* core library*

The transport core is integrated with other cores from the IBM Blue Logic core library to create a chip. This creates an opportunity to optimize communication with other

cores, such as the host processor, and with the audio and video decoders. In addition, integration into a larger design allows the use of common memory. The goal is to exploit as many of these advantages as possible while still maintaining compatibility with more typical interfaces.

- *Limit use of host processor and memory resources*

The transport design must be self-contained for typical usage and provide sufficient performance to avoid placing an undue burden on the host processor, leaving it free for other functions. For example, the PCRs used to control the STC in the STB are transmitted at least every 100 ms and usually much more frequently. The arrival of each PCR could generate an interrupt to signal the processor to examine the difference between the PCR and the current STC count and adjust the STC frequency as required. Reducing the number of clock recovery interrupts reduces the load on the processor.

As a further example, processing table sections can consume considerable processor and memory resources. Only a few table sections transmitted in a PID stream are actually needed by the STB at any time. If unnecessary table sections are received, they are discarded. To understand the impact of filtering table sections using software running on a processor, it is helpful to look at two such approaches. The first is to deliver all table sections to memory and then filter the sections to determine which ones are needed. This typically requires a relatively large working buffer to store the table sections while they are filtered and another memory area to store the filtered table sections. The size of the working buffer depends on the latency for software filtering. The second approach is similar to the first and uses a minimum working buffer, typically the size of one transport packet. However, the processor must filter the table sections in real time. While this second software approach minimizes the memory needed, it places additional performance and latency requirements on the processor. The goal in this design was to provide an extremely flexible set of table-filtering capabilities in hardware that allow filtering to be performed before the data is delivered to memory, saving both processor and memory resources.

- *Minimize silicon area while maintaining flexibility*

Minimizing silicon area is a key requirement for all cores, especially those used in larger integrated designs. A hardware-based design provides the greatest efficiency for a given set of functions, and the capability of a hardware design can be extended through the use of highly configurable resources. There is further benefit in the ability to adapt the design to new requirements as they arise. The goal is to reach a balance between functions which are implemented in hardware and those which require more flexibility and can be implemented as software running on the host processor.

## Design architecture

The transport architecture is divided into three major sections: a front end, an interim buffer, and a back end, as shown in **Figure 6**. The front end consists of the packet synchronization, PID filter, parsing, and descrambling functions; it processes fields related to the header and adaptation fields. The buffer stores arriving packets for processing by the back end, as well as control and status information. The back end, which processes packet payloads, consists of three separate unloaders: one for audio, one for video, and one for system data. Both the front end and the back end access the packet buffer through a common cycle-by-cycle arbiter. The front end passes information about each packet to the back end through an information word that is appended to the beginning of the packet.

This modular approach allows parts of the design to be changed with minimum impact on the remaining elements. In addition, the use of a single SRAM memory for the packet buffer and an information word allows other memory elements such as registers to be stored in a single array.

Figure 6 represents the dataflow of a transport stream through the design. The *sync* block finds the start of the transport packet by searching for the sync byte at the beginning of a transport packet. The *packet parser* extracts data from the transport packet header and adaptation field. For example, the PID in the stream is compared to active PIDs in the *PID filter*. Concurrently, the packet parser sends PCRs from matching PCR packets to the *clock recovery* unit to reconstruct the STC. The optional *descrambler* may be configured to descramble the payloads of selected packets.

Status indicators representing parsed information are sent along with the complete transport packet to the *packet loader* to be stored in the *packet buffer*. The status indicators reside in a 32-bit information word, making the total storage requirements for a packet 48 32-bit words. The packet buffer holds up to ten transport packets while they are delivered to the decoders and memory queues, and efficiently absorbs any latency of these data targets.

The three unloaders retrieve data from the packet buffer. The *audio unloader* and *video unloader* send data to the respective decoders as it is requested. The *memory unloader* sends data to the *bus controller* for subsequent transfer to system memory. It can optionally be configured to filter table sections and perform CRC checking. The *host port* is used by the host processor to configure, monitor, and control the operation of the transport core. The *transport assist port* can be used to further filter, parse, and direct data before it is delivered.

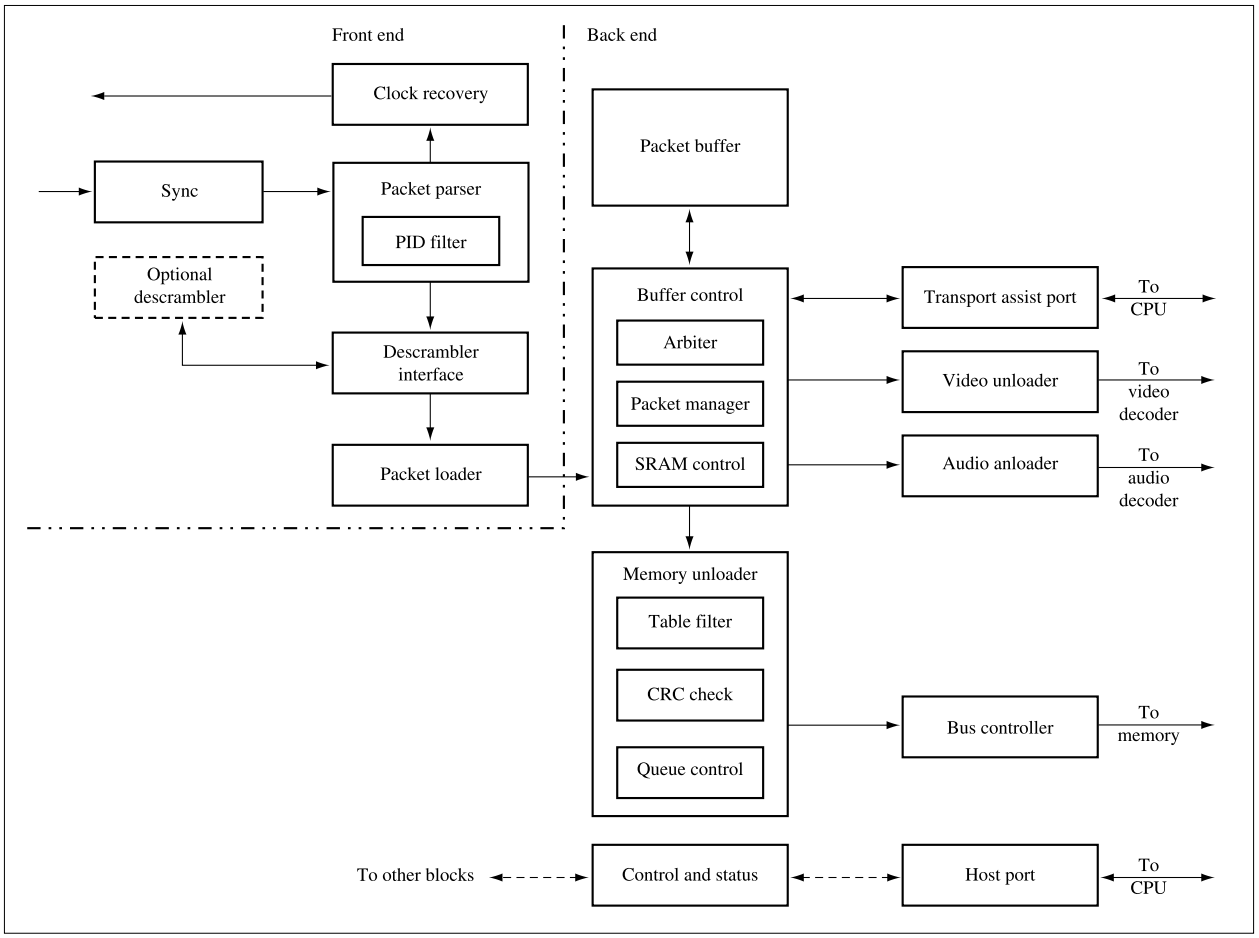


**Figure 6**

Transport hardware architecture.

## Features of the design

Using the described architecture, we tailored many areas to achieve our stated design goals. Unique work was done to reduce host processing requirements, to provide flexible management mechanisms in order to handle foreseeable use of the hardware, to maximize storage efficiency, to allow an option for modification of the operation of the design, and to increase communication and dataflow performance between cores. Key areas of work are described in the following sections.

- *Extended clock recovery*

A transport extracts the PCRs from the transport stream in real time to ensure that the constant-delay model is maintained. At the same time it captures the value of the STC, which is the local clock in the STB. The difference in rate and the difference between the PCR and the STC values can be used in a feedback loop to control the oscillator driving the STC as well as the audio and video decoders.

The transport core provides extended support in hardware for recovering the clock from the arriving PCRs, as shown in **Figure 7**. In addition to extracting the PCR from the stream and capturing the matching STC value, it calculates the difference between the two values and compares it to a programmable threshold to determine whether an interrupt should be issued. Further, a simple internal algorithm is provided which can make minor corrections to the frequency of the voltage-controlled crystal oscillator (VCXO), so that the STC tracks the PCR without support from the host processor. The VCXO frequency is controlled by a pulse generator whose output is filtered by a low-pass RC filter network to create a steady voltage input.

A software clock recovery algorithm running on the processor completes the solution. The software algorithm was developed to take advantage of the features provided in hardware, including the hardware algorithm and the programmable threshold. The software algorithm is used initially to match both the rate and value of the STC to the arriving PCRs. This is accomplished by setting the threshold value for generating an interrupt to 0, so that the algorithm is called each time a PCR is received by the STC. Once the software algorithm has closely matched the STC to the PCRs, full control returns to the hardware algorithm. This is accomplished by setting the threshold to a value greater than 0; in the implemented software algorithm, a value of 256 was used. The software algorithm is not called until the difference between an arriving PCR and the STC is greater than 256. Meanwhile, the hardware algorithm maintains the feedback to the VCXO.

The software and hardware solution for clock recovery was originally tested on an emulation system containing the transport design in FPGAs [7], and then later in hardware. When the local clock frequency differed by 100 ppm, the software algorithm requires 60–70 PCRs to initially lock and switch the feedback control solely to the hardware algorithm. The hardware algorithm maintains the difference between the STC and the PCR to less than 256 for 6–8 minutes. Once the magnitude exceeds the threshold value, the software algorithm can relock the clock on the basis of 20–30 PCRs. We have measured a 98% reduction in the number of clock recovery interrupts to the host.

- *Flexible and extensive table section filtering*

Like PCRs, table sections require constant handling by the STB. Several types of table sections required by the MPEG-2 systems standard must be repeated at least ten times a second to allow random access by all STBs which may receive the transport stream. The STB must monitor the table sections to determine whether they have been updated. Thus, table sections can place a burden on the STB, even when the data they contain remains the same.

Like clock recovery, the IBM transport demultiplexor reduces the load on the host processor by performing the repetitive tasks and limiting delivery of table sections transmitted within the PID streams to only those needed by the STB. The processor resources are used only when table sections of interest have been acquired by the transport hardware and delivered to memory.

The hardware table section filtering must be compatible with existing software application programming interfaces (APIs). These APIs can assume bit-level masking of the filtered fields. They can also permit variable-length filters, which may be accomplished by using a combination of hardware and software filtering.

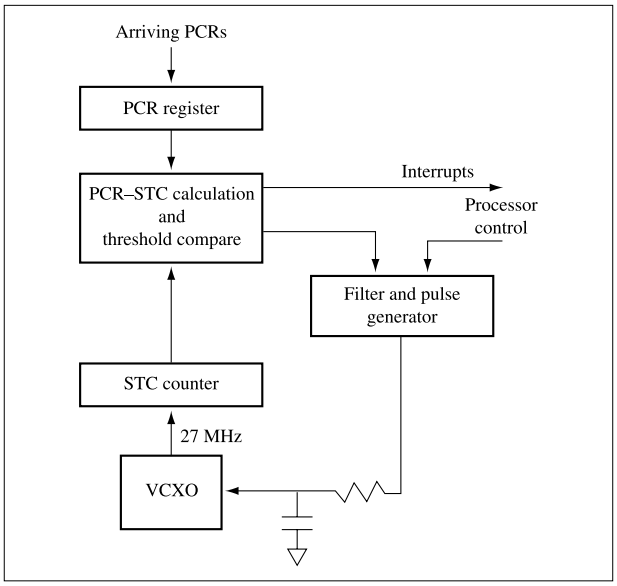


**Figure 7**

Hardware clock recovery.

The transport architecture provides a pool of four-byte filter blocks. Each filter block includes four bytes of compare value, four bytes of compare masking, and a 32-bit control word. The current design includes 64 filter blocks, corresponding to 192 32-bit words of storage. To minimize silicon area, the filter blocks are stored in the common SRAM along with the received packets, and are retrieved from the SRAM as needed while a table section is processed.

A filter block is applied to a table section on a bitwise basis. Each bit of the filter value and mask is applied to the corresponding bit in an arriving table section. A bit always matches if the mask bit is set; otherwise, a bit matches if the filter bit and the table section bit match. A match results for the filter block if for all 32 bits the result is a match.

*Constructing section filters from filter blocks*

The filter block architecture provides two important features for flexibility. The first is the ability to allocate the filter blocks in any manner among the received PIDs. The second is the ability to create section filters using any number of filter blocks from one to 64.

The key to both types of flexibility is a linked list structure used to construct section filters for each PID index. Each section filter may include one or more filter blocks depending on the required filtering depth. The first filter block for a PID index is specified in the configuration of a queue. Additional filter blocks are

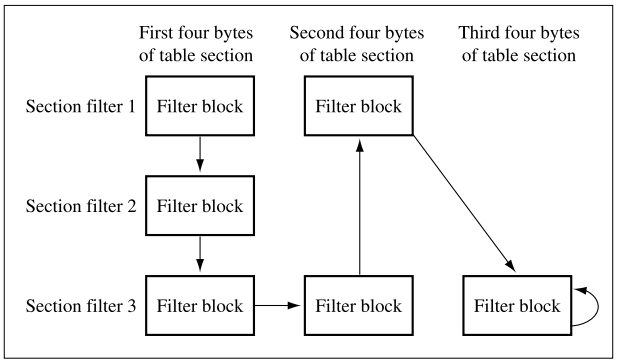


**Figure 8**

Example of filter-block linking.

specified by a pointer to the next filter block in the control word of each filter block. A second field in the control word indicates whether the next filter block should be applied to the next four words of the section filter. This linked list arrangement simplifies the allocation and reallocation of filter blocks to different section filters and PID indexes.

Consider three section filters attached to a PID index. The first section filter has a filter depth of eight bytes, the second section filter four bytes, and the third section filter twelve bytes. Six filter blocks must be allocated for this PID index. It is helpful to think of organizing the filter blocks into rows and columns, as shown in **Figure 8**. Each row implements one of the section filters. The links indicate the order in which the filter blocks are processed by the transport. Filter blocks in the same column are compared against the same byte positions in the table section.

Another field in the filter block control word indicates which filter blocks are part of the same section filter. The section filter ID field is configured with the same value for filter blocks belonging to the same section filter. In Figure 8, three unique section filter IDs are used to specify the filter blocks associated with each of the three section filters. The application can make the section filter IDs unique across all PIDs, or it can use the same section filter IDs for several PIDs, allowing multiple uses of this feature.

*Section filter ID and section filter match word*

The section filter ID field provides a second function to further reduce processing by the host processor. The transport can be configured to write a match word in memory indicating which section filters matched a table section. Each bit position corresponds to the section filter ID field in the filter control of each filter block. The section filter match word is placed before the start of the table section in memory to allow the software to quickly identify the destination(s) of the table section. Without this feature, the software may have to filter the table sections again to determine which section filters matched.

*"Not-match" section filtering*

The table section filtering hardware also supports a "not-match" capability for each byte. Four bits in the control word of each filter block indicate for each byte whether the byte is considered a match only if the value in the filter and the table section are not the same. Not-match is accomplished by inverting the match result for each byte, and is used to detect a change in a byte of a table section.

*Table section filtering performance*

The flexible table section filtering architecture also provides high-performance processing. The hardware can maintain a sustained rate of 60 filter block compares per transport packet, with an input transport-stream rate of 100 Mb/s. For example, a transport packet containing five table sections with four block filters defined for that PID index requires 20 filter compares to complete processing of that packet.

- *Moving system data to memory*

The transport macro supports up to 32 queues in memory. Each of these is associated with a PID index in the PID filter. For example, data from PID index 5 is stored in queue 5. One queue can be allocated to provide special functions which are described later. The transport can be configured to deliver data to memory in a 16MB region, which is also aligned on a 16MB boundary. Queues within the region can be allocated in increments of 4096-byte blocks, with the largest possible queue size encompassing the entire 16 MB. The entire 16MB queue region need not be allocated solely for the queues. The application can place the queues anywhere within the 16MB region and can use the remainder for other functions (**Figure 9**).

*Memory queue management*

Each queue can be configured independently using two registers. The fields in these registers include those to specify the region of memory defined for the queue, a read pointer within the queue which the hardware should not pass, and the type of data to be delivered to the queue. A set of status registers are also provided for each queue. One of the status fields is the write pointer indicating the next memory location to which the hardware will deliver data. Another field indicates the beginning of a table section which is partially delivered because it spans transport packets.

Interrupts are provided to the processor when the hardware has reached the read pointer within the queue,

when a table section or packet has been delivered to memory, or when a programmable number of 256-byte blocks have been delivered to memory. The interrupt, configuration, and status registers allow the host processor to manage each queue independently as a circular buffer.

*Queue data types*

Each queue can be configured to deliver data from packets of a PID stream. Several of the basic options include delivering complete transport packets, the adaptation fields, or just the packet payload. Eight options deliver table sections, with the added flexibility to include section filtering, CRC32 checking, and delivering adaptation fields to a special bucket queue.

A bucket queue allows delivery of adaptation fields from several PID streams to a single queue. This may be desired when either just the packet payload or table sections from the packet payload must be delivered to their corresponding queues, but the adaptation fields are also required from these PID streams. To allow the application to determine which PID is associated with an adaptation field, the transport packet header is also stored in the bucket queue, followed by the adaptation field. The audio and video queues are not used to deliver data to the decoders, so these can be used to deliver adaptation data from the audio and video PIDs to memory. One of these queues can also be used as the adaptation bucket queue.

- *Efficient storage through a common SRAM buffer*

Configuration information for the queues and table section filter requires storage that is easily accessible by the transport demultiplexor. These and other storage requirements are shown in the first column of **Table 1**. One of the critical choices in the design was the implementation of this required state.

There are several storage mechanisms available, including latches, register arrays, SRAMs, and off-chip memory. The latter requires additional pins on the chip, which are not available for chips with a high level of integration. Latches consume the most area per bit, but the value from a latch can be used independently and concurrently with other latch values. Thus, latches were used to implement the PID filters, since concurrent access is required to check all 32 PID filters in one clock cycle. Other registers were also implemented with latches for similar reasons. Registers implemented as latches also require multiplexing logic so that their state can be accessed by a processor. Register arrays and SRAMs do not share this overhead because the multiplexing or address decoding is part of their basic function.

The initial architecture included an SRAM for packet storage, since it was the most area-efficient. The incremental area for storing another bit in the SRAM was approximately 1/30th of a corresponding latch implementation. This created an opportunity for implementing other registers and working data as part of the same SRAM array. Thus, most of the memory storage in the transport is located within a single SRAM array (Table 1).

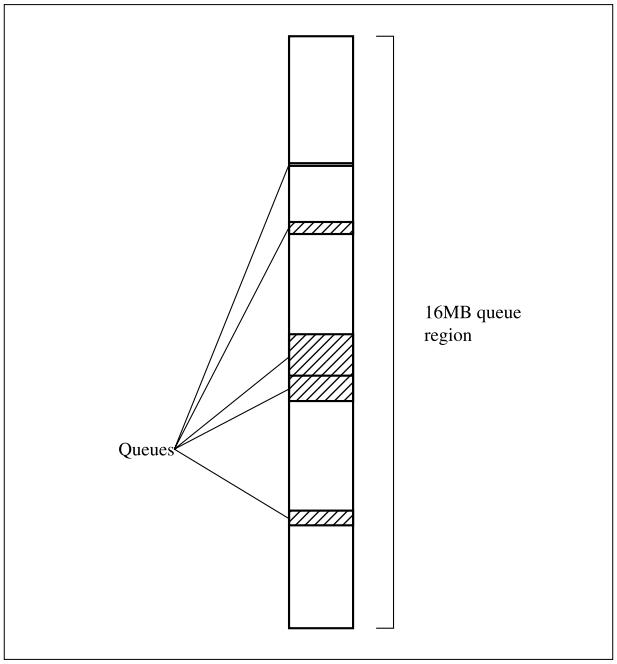


**Figure 9**

Queue memory allocation.

**Table 1** Register implementation.

| *Register type* | *Latches* | *SRAM* |
|---|---|---|
| Configuration | 12 | |
| Clock recovery | 10 | |
| Interrupt and masks | 11 | |
| PID filter | 36 | |
| Ten packets | | 480 |
| Queue configuration | | 64 |
| Section filters | | 192 |
| Queue working data | | 160 |
| Queue interrupts | | 16 |
| Descrambler | | 48 |
| Totals | 69 | 960 |

To use a single SRAM, a simple arbitration scheme was developed to ensure that no data would be lost. The arbitration scheme uses a fixed priority, with descrambler key access and packet data loading having the highest priority. A single access can be one, two, or three 32-bit words for the purpose of arbitration, but no unit is allowed to have two consecutive accesses. The success

**Table 2** SRAM utilization.

| *Access type* | *Maximum accesses per packet* |
|---|---|
| Packet loading | 48 |
| Packet unloading | 48 |
| Read queue configuration | 2 |
| Read queue working data | 4 |
| Update queue working data | 4 |
| Read interrupts | 1 |
| Update interrupts | 1 |
| Sixty filter block lookups | 180 |
| Read descrambler keys | 4 |
| Total | 292 |

of the arbitration scheme has been demonstrated in the emulation system [7], and then later in hardware.

Our goal was to ensure that no more than 50% of the SRAM bandwidth is consumed in order to permit additional SRAM storage in future designs and to handle any unanticipated problems. **Table 2** shows the maximum number of SRAM accesses required to process a transport packet. Loading a packet requires 48 SRAM accesses; unloading a packet can require up to 48 accesses, but may take less. The heaviest usage of the SRAM occurs when the transport is filtering and delivering table sections. In this case, the memory unloader reads the queue configuration and queue working data, packet, and section filters. At the end of a packet, the memory unloader updates the queue working data and reads and updates the interrupts for the queue. Reading the section filters accounts for 62% of the total SRAM accesses. The utilization is determined by the number of system clocks per transport packet. In a typical configuration, the design requires a ratio of 1:4 for the incoming stream byte clock to the system clock. This allows for 752 system clocks per 188-byte packet. Thus, the maximum SRAM utilization for a single packet is 292/752, or 39%.

- *Modifying the transport function through software*

While the transport core has been designed to implement all common functions in hardware, it also has an interface to the host processor that can be used to modify its operation. This provides flexibility and the ability to meet new requirements as they arise. Since the processor interface is used only on an exception basis, the design maintains its goal to minimize its impact on the system resources.

The processor interface, referred to as the transport assist processor (TAP) function, gives the host direct access to the internal packet buffer, including the information word that is passed by the front-end units to the back-end unloaders, and indicates how to process the corresponding packet. The transport can be configured through special registers to hold a given packet type in the buffer under certain conditions such as the start of a table section. The processor can then parse the packet data, determine the proper action, modify the packet data or information word accordingly, and release the packet to the back-end unloaders, which process it as indicated.

The transport core includes a dedicated interrupt and an independent signaling mechanism between the transport assist interface and the application interface so that the two processes can be separated and even implemented with separate processors.

- *Use with other cores in an integrated design*

Since the transport design is intended to be used as a core in the IBM Blue Logic library that will likely be integrated with other related functions, additional signals and ports have been added to allow extending or enhancing the function. For example, there is a set of optional handshake signals between the transport and the audio and video decoders, which provide improved channel changes, time-base changes, and error flagging.

The transport can also take advantage of memory shared with the host controller for passing system data such as filtered table sections. In an integrated design, the transport memory controller is able to access the same address range as the processor, so it can write data directly into buffers that are maintained by the processor without the need to move it a second time. This approach requires less bus bandwidth and reduces the chip pin count. The transport also has the special TAP port to allow a processor to control transport packet processing within the transport core, as described in the preceding section. Using this port, the internal buffer of the transport appears as a data cache to the processor. All of these features are discussed further as part of the integrated MPEG subsystem [8].

- *Error detection and concealment*

As an example of the communication between the transport and other cores, the transport deals with errors as a two-step process, detection and handling/concealment. This technique is applied to errors which affect individual packets. The transport also detects problems with the incoming stream, including a number of consecutive packets with errors and no delivery of packet data after 1/30th of a second. We refer to these errors as transport-stream errors, since they span a number of packets. Transport-stream errors may affect the data associated with several PIDs.

Errors may also occur in individual packets, which affect the data of only one PID. We refer to these errors as

PID-stream errors. The following types of PID-stream errors are detected:

- Transport error indicator is set.
- Sync byte is missing and the transport is in sync.
- TS error signal is active.
- Continuity counter error is detected.
- Packet buffer overflow has occurred.

Packets with these PID-stream errors are discarded before they are loaded into the packet buffer.

*Audio and video data error handling*

The unloaders report PID-stream errors to the decoders before processing the next valid packet. For the video decoder, the transport sends the error flag to the decoder and, optionally, an error sequence code which is recognized by the video decoder. For the audio decoder, the transport sends only the error flag. Valid data follows the error so that the decoder can begin scanning the stream after the error to determine the next valid decode point.

In the case of transport-stream problems, the video unloader sends an error to the decoder if the packet buffer does not have any more data from that PID index, and the decoder is requesting data. These conditions indicate that valid data has stopped, and the flag allows the decoder to find a valid point to stop decoding data from its rate buffer to perform error masking.

*Error handling for system data*

The memory unloader provides automatic error handling for table sections. For all types of system data, the transport can notify the processor through an interrupt if a PID-stream error occurred on data delivered to memory. Transport-stream errors are not used by the memory unloader, since the real-time delivery requirement is not as important as with the delivery of audio and video data. These errors appear as PID-stream errors at a later time.

*Table section error handling*

The front end discards packets with errors before they are loaded into the packet buffer, since an error may make it impossible to correctly parse the packet. Additional checking is performed by the memory unloader on table sections to ensure that the table section length matches the value in the table section header, and that a CRC32 check on the table section is correct. If any type of error is found during unloading of a table section, the current section is discarded and the queue write pointer is moved back to the end of the previous valid section. This discards the current table section, which has an error.

## Summary

The IBM transport demultiplexor core is implemented as a hardware solution and requires minimum support from a host processor. Most of the requirements for clock recovery and table section filtering are performed in hardware to reduce handling by the processor. Queues defined in the processor address space do not require the processor to copy the data before it is used. The design is optimized to parse the fixed fields in an MPEG-2 transport stream. A single SRAM is used for storing transport packets as well as control and status registers. The transport core is designed with additional interfaces to the decoders and the processor. The decoder interfaces facilitate error handling, channel changes, and time-base changes. The transport assist interface to a processor allows the transport function to be extended as needed if requirements change.

## Acknowledgments

The authors wish to thank the IBM Digital Video Products management team for supporting the development of the MPEG-2 transport core. We also wish to thank all of the members of the set-top-box development project in Raleigh, North Carolina, Endicott, New York, Burlington, Vermont, and Haifa, Israel. They influenced the design of the MPEG-2 transport core and many of the ideas presented here.



## References

1. "Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Systems," *ISO/IEC 13818-1*, 1996.
2. "Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Video," *ISO/IEC 13818-2*, 1996.
3. "Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Audio," *ISO/IEC 13818-3*, 1996.
4. Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg, and Didier J. LeGall, *MPEG Video Compression Standard*, Chapman & Hall, New York, 1997.
5. Barry G. Haskell, Atul Puri, and Arun N. Netravali, *Digital Video: An Introduction to MPEG-2*, Chapman & Hall, New York, 1997.
6. *http://www.chips.ibm.com/bluelogic*.
7. C. A. Reed and D. J. Thygesen, "Pseudorandom Verification and Emulation of an MPEG-2 Transport Demultiplexor," *IBM J. Res. Develop.* **43,** 533–544 (1999, this issue).
8. R. E. Anderson, E. M. Foster, D. E. Franklin, and R. S. Svec, "Integrating the MPEG-2 Subsystem for Digital Television," *IBM J. Res. Develop.* **42,** 795–805 (1998).

*Received November 18, 1998; accepted for publication February 25, 1999*

**Richard E. (Andy) Anderson** *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (andya@us.ibm.com).* Mr. Anderson graduated from MIT with a B.S. in electrical engineering and an M.S. in electrical engineering and computer science in 1987. His previous work included human interface development for point-of-sale terminals at NCR Corporation. While at NCR, he was co-inventor of an issued patent, co-wrote a conference paper, and was an author of a paper. Mr. Anderson joined IBM in 1992 to design and support microprocessors. For the last two years he has worked on the IBM MPEG-2 transport demultiplexor and has published one paper and filed several patents in this field. He recently joined the ASIC cores development team.

**Eric M. Foster** *IBM Microelectronics Division, 1701 North Street, Endicott, New York 13760 (emfoster@us.ibm.com).* Mr. Foster received a B.S. in mechanical engineering from Clarkson University in 1982 and an M.S. in electrical engineering from the State University of New York at Binghamton in 1987. In 1982 he joined the IBM Federal Systems Division, where over time he worked on the mechanical design of large shipboard computers, module packaging design, manufacturing process development, and the electrical analysis of packaging noise effects. In 1989 he moved to the Microelectronics Division, where he worked on the design and characterization of high-performance packaging for optoelectronic datalinks. In 1993 he joined the Digital Video Products group, focused on the development of MPEG products, where he has worked on set-top-box network interface design, PC-based evaluation cards, and MPEG-2 transport architecture. Mr. Foster is currently working on integrated chip architecture and MPEG subsystem design, and has recently moved to the IBM Research Division. He has taught short courses on packaging and optoelectronics, published several internal and external papers, and filed fifteen patents. He is a member of the IEEE Consumer Electronics Society.