PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04H 5/00 | A1

(11) International Publication Number: WO 99/01953

(43) International Publication Date: 14 January 1999 (14.01.99)

(21) International Application Number: PCT/US98/13844

(22) International Filing Date: 2 July 1998 (02.07.98)

(30) Priority Data:
08/887,362 2 July 1997 (02.07.97) US

(63) Related by Continuation (CON) or Continuation-in-Part (CIP) to Earlier Application
US 08/887,362 (CON)
Filed on 2 July 1997 (02.07.97)

(71) Applicant *(for all designated States except US)*: CREATIVE TECHNOLOGY, LTD. [SG/SG]; 67 Ayer Rajah Crescent # 103–18, Singapore 139950 (SG).

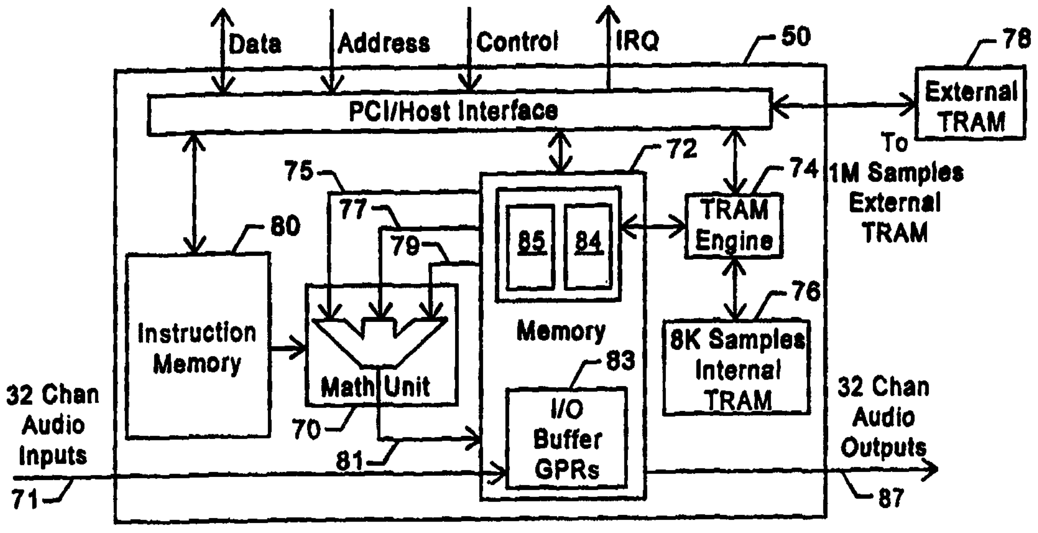
(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: HOGE, Stephen [US/US]; 150 Baldwin Street, Santa Cruz, CA 95060 (US).

(74) Agents: LANG, Dan, H. et al.; Townsend and Townsend and Crew LLP, 8th floor, Two Embarcadero Center, San Francisco, CA 94111–3834 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: AUDIO EFFECTS PROCESSOR HAVING DECOUPLED INSTRUCTION EXECUTION AND AUDIO DATA SEQUENCING



(57) Abstract

A sound effects processor (30) including a first memory (72), an instruction execution unit (70) and a sound memory engine (74) integrated onto a single chip. The first memory (72) includes a first address space that is addressable by both the instruction execution unit (70) and the sound effects engine (50). The instruction execution unit (70) executes sound processing instructions, including instructions to read from and write to said first address space and the sound memory engine (74) operates independent of the instruction execution unit (70) to read and write to the first address space and perform address arithmetic and memory accesses to a second memory as necessary to implement audio delay lines.

AUDIO EFFECTS PROCESSOR HAVING DECOUPLED INSTRUCTION EXECUTION AND AUDIO DATA SEQUENCING

BACKGROUND OF THE INVENTION

The present invention relates to an audio signal processor. More specifically, the present invention relates to an audio effects processor integrated on a single chip and including memory and architecture capable of implementing audio delay lines independent of the operation of an instruction execution unit of the effects processor.

Sound effects audio signal processors (ASPs) are commonly used to generate sound effects in a multitude of audio components, some examples being programmable musical instruments, video games, cinema sound systems, virtual reality systems, and computer audio systems. Such sound effects include, but are not limited to, reverberation effects, 3-D audio, and distortion effects. ASPs create sound effects by executing programs containing a series of instructions. Each instruction directs the ASP to perform a specific logical and/or arithmetic operation on the received signal, resulting in the program's creation of a particular sound effect.

It is common for ASPs to implement sound effects through the use of delay lines, which often take the form of a large block of relatively slow "tank" RAM (TRAM) memory. Some known ASPs perform operations on the TRAM data through a TRAM engine under the control of an instruction execution unit of the ASP. That is, when the instruction execution unit wants to operate on the signal output from a delay line, it schedules a TRAM read operation to begin at a specific instruction cycle, waits for a specific number of cycles for the read operation to occur and then performs an operation to retrieve the data.

In processors of this type, the instruction execution unit must be tightly coupled to the TRAM engine because any mis-synchronization in the scheduling of operations could result in the loss of the TRAM data. Also, valuable processor time of the instruction execution unit is used in coordinating and implementing the sound effects. Accordingly, improvements to the architecture of such ASPs are desirable.

SUMMARY OF THE INVENTION

The present invention provides an audio effects processor in which the TRAM engine (hereinafter referred to as the "sound memory engine") is decoupled from the instruction execution unit of the processor. In the present invention, the sound memory engine and instruction execution unit operate independent of each other, passing TRAM data through a shared area of a memory. The sound memory engine reads and writes TRAM data according to its own schedule, using the shared memory area as a source for TRAM writes and a destination for TRAM reads. When the executing program requires TRAM data, the instruction execution unit reads or writes to these same shared register file locations.

A sound effects processor according to the present invention includes a first memory, an instruction execution unit and a sound memory engine integrated onto a single chip. The first memory includes a first address space that is addressable by both the instruction execution unit and the sound memory engine. The instruction execution unit executes sound processing instructions, including instructions to read from and write to the first address space, and the sound memory engine operates independent of the program counter (instruction execution rate) of the instruction execution unit to read from and write to the first address space and perform address arithmetic and memory accesses to a second memory as necessary to implement audio data storage (e.g., delay lines whose delay length can be varied dynamically -- a capability required for many audio effects like chorusing, flanging and pitch shifting). The second memory can be internal or external to the effects processor.

For a further description of the nature and advantages of the present invention, reference should be made to the following description taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1A depicts a representative multimedia personal computer system in which the audio effects processor of the present invention may be employed;

Fig. 1B depicts a simplified representation of the internal architecture of the multimedia personal computer system depicted in Fig. 1A;

Fig. 2 is a simplified block diagram of multimedia board 28 shown in Fig. 1B, a board onto which the audio signal processor of the present invention could be incorporated;

Fig. 3 is a simplified block diagram of one embodiment of the audio signal processor shown in Fig. 2;

Fig. 4 is a simplified block diagram of audio effects processor 50 in accordance with the present invention shown in Fig. 3;

Fig. 5 is a block diagram of one embodiment of sound memory engine 74 shown in Fig. 4;

Fig. 6 is a block diagram showing the allocation of memory space within one embodiment of memory 72 shown in Fig. 4;

Figs. 7A-7D are diagrams that illustrate the circular addressing scheme used for generating delay lines according to the present invention; and

Figs. 8A-8D are figures illustrating how sound memory engine 74 prevents TRAM read and write hazards that would otherwise be possible because memory 72 is shared with instruction execution unit 70.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention is applicable to digital sound generation systems of all kinds. Advanced audio effects can be provided to video games, multimedia computer systems, virtual reality environments, cinema sound systems, home theater, and home digital audio systems, for example. Fig. 1A depicts a representative multimedia personal computer 10 with a monitor 12 and left and right speakers 14 and 16, an exemplary system that can be enhanced in accordance with the invention with sound effects such as three-dimensional audio.

Fig. 1B depicts a greatly simplified representation of the internal architecture of personal computer 10. Personal computer 10 includes a CPU 18, memory 20, a floppy drive 22, a CD-ROM drive 24, a hard drive 26, and a multimedia card 28. Each of the components of computer 10 shown in Fig. 1B communicate with each other over a bus system 29. Of course, many possible computer configurations could be used with the invention. In fact, the present invention is not limited to the context of personal computers and finds application in video games, cinema sound systems and many other systems.

Fig. 2 illustrates a typical multimedia card 28 on which an integrated circuit according to the present invention may be mounted. Multimedia card 28 includes a sound processor chip 30 mounted on a circuit board 32. As shown in Fig. 2, a CDROM connector 34, an AC97 CODEC 36, an optional AC3 decompressor/decoder 38 and a mixer 39 are all connected to sound processor chip 30 through appropriate interfaces.

Also shown in Fig. 2 are various other connections to sound processor 30, including a joystick connector 42, a phone line connection 44 for a modem (not shown), a line-in connector 46, a microphone connector 47, and a speaker output 48. In addition, a connection to a PCI bus 49, which is part of bus system 29, is shown. Bus 49 connects to the host microprocessor 18 and to main memory 20.

A simplified block diagram of sound processor 30 is shown in Fig. 3. Sound processor 30 includes three primary functional units: a sound processing engine 40, a sound effects engine 50 and a host interface unit 60. Sound processing engine 40 is a 64-voice wavetable synthesizer that employs an eight-point interpolation algorithm for professional quality audio playback as described in U.S. Patent No. 5,342,990 entitled "Digital Sampling Instrument Employing Cache Memory" and 16 summing effects send buses. Each of the 64 voice channels can be routed, at its own programmable amplitude, to an arbitrary selection of four of these buses. Host interface unit 60 interfaces sound processor 30 with host CPU 18 using the PCI protocol. Sound effects engine 50 receives input from sound processing engine 40 and from additional audio inputs such as CD Audio, ZVideo, a microphone jack, a stereo input and an auxiliary S/PDIF input among others. Further details of sound effects engine 50 and host interface unit 60 are described below with respect to Fig. 4. Other details of host interface unit 60 along with some details of sound processing engine 40 and other portions of sound processor 30 are set forth in U.S. Serial No.___/____________ (attorney docket 17002-86) entitled "AUDIO EFFECTS PROCESSOR WITH MULTIPLE ASYNCHRONOUS AUDIO STREAMS," having David P. Rossum and Scott Fuller as inventors and assigned to Creative Technologies, Ltd., the assignee of the present invention. The 5,342,990 patent and the ___/____________ application (attorney docket 17002-86) are both hereby incorporated by reference in their entirety.

<u>Audio Effects Processor 50</u>

According to the present invention, sound effects engine 50 (also referred to as "effects processor 50") includes separate functional units to 1) execute audio signal processing instructions and 2) implement audio data storage (e.g., audio signal delay lines and table look-ups). These functional units operate independent of each other and exchange data through a shared memory address space. One particular implementation of an audio effects processor of the present invention is illustrated in Fig. 4 and discussed below. In no respect is the present invention limited to this specific implementation, however. After reading the following description, a person of ordinary skill in the art will understand that other implementations of effects processor 50 are possible without departing from the concept of the present invention.

Fig. 4. is a simplified block diagram of one embodiment of effects processor 50 shown in Fig. 3 above. The basic architecture of effects processor 50 combines an instruction execution unit 70, a high-speed internal memory 72, and a sound memory engine 74 (also referred to as "TRAM engine 74"), which interfaces to internal and external slower-speed, high capacity TRAM memories 76 and 78. Audio signals are received by effects processor 50 at a processor input 71, where they are directed to a dual buffered portion 83 of memory 72. In this embodiment, the audio signals received at input 71 include 16 voices output from sound processing engine 40, 2 lines for a stereo input, 2 lines for a CD audio input, 2 lines for a zvideo input, a microphone input, 2 lines for an auxiliary S/PDIF input and a 6-line I^2S input.

Instruction Execution Within Effects Processor 50

Instruction execution unit 70 executes instructions stored in an internal microprogram memory 80, separate from memory 72 to perform particular operations on one or more of the audio signals stored in memory address space 83. Each instruction has a 4-bit opcode field, three 10-bit input operand address fields and a 10-bit output operand address field. The opcode field holds the instruction's operation code (opcode) that specifies the instruction's operation to the processor. The operand address fields hold addresses to spaces in memory 72. Because the operands are 10-bits long in this embodiment, the maximum addressable space of memory 72 available to DSP instructions is 1024 locations. All 1024 locations of memory 72 are addressable.

Instruction execution unit 70 retrieves data stored in the locations in memory 72 from the locations specified in the instruction's three input operands over input data lines 75, 77 and 79. Instruction execution unit 70 then performs the requested operation specified by the instruction's opcode and writes the resultant data over output data line 81 to the memory address specified in result operand of the received instruction. The resultant data may be directed to an audio output port 87 by writing to one of the 32 output buffers in memory space 83, stored in another portion of memory 72, or reused in subsequent operations by instruction execution unit 70.

Instruction execution within instruction execution unit 70 is sample-locked and memory-based, rather than register-based. Sample-locked execution means that instruction unit 70 executes a single pass through microprogram memory 80 every sample period. Thus, instruction execution is inherently tied to the system's audio sample rate. In this embodiment, the sample rate is 48 KHz as derived from AC97 Codec 36 shown in Fig. 2. In other embodiments this sample rate may be a different set frequency or may be a variable frequency. For example, in another embodiment, the sample rate is in the range of about 40-50 KHz as derived from a system clock on board 32 (not shown).

Memory-based instruction execution means that operands are fetched directly from memory 72 without the bottleneck of an input register set at the input to the instruction execution unit. It also means that there are no volatile output registers that need to be flushed after each instruction in order to save results.

Instruction execution unit 70 executes all the instructions in microprogram memory 80 "in-line" without branches, loops or subroutine calls. Such in-line execution ensures that all processing within the basic real-time deadline (one sample period) is achieved by design. It also simplifies the design of the execution pipeline, since instruction sequencing is completely deterministic, i.e., there are no branches in the flow of control.

Although instruction execution always proceeds in-line from the start of microprogram memory 80 to the end of the memory, there are provisions for the conditional execution of instructions. Details of such provisional instruction execution along with specifics of the instruction set executed by instruction execution unit 70 are set forth in U.S. Serial No. ___/___________ (attorney docket 17002-84) entitled, "Processor With Instruction Set for Audio Effects," having Stephen Hoge as the inventor and assigned to Creative Technologies, LTD., the assignee of the present invention. The ___/___________ application (attorney docket 17002-84) is hereby incorporated by reference for all purposes.

Each instruction executes in a single instruction cycle whose period is the sample rate divided by the number of instructions in microprogram memory 80. Thus, in an embodiment where microprogram memory 80 is 512 instructions long and the sample rate is 48 KHz, the instruction cycle is 40.7 nanoseconds long. The results of each instruction are available for use in the following instruction cycle.

<u>Host Interface Unit 60</u>

Also shown in Fig. 4 is host interface unit 60, which allows the host processor (CPU 18) to control the operation of audio effects processor 50. Such control is achieved by allowing the host to initialize and read and write data and executable instructions to memory 72 and/or microprogram memory 80. Such memory read and write operations can be executed transparent to the execution of digital signal processing (DSP) programs allowing audio effects processor 50 to support simultaneous startup, execution and shutdown of multiple independent and separately loaded programs. In this embodiment, communication between the host processor and audio effects processor 50 through interface unit 60 occurs over a PCI bus using a PCI protocol, but other protocols can be implemented in other embodiments. Also in still other embodiments, host interface unit 60 is a direct, memory-mapped microprocessor interface suitable for use with either big-endian or little-endian processors.

Sound Memory Engine 74

Sound memory engine 74 is the interface between memory 72 and the large capacity TRAM memory used for long-term audio data storage. Fig. 5 is a block diagram of one embodiment of sound memory engine 74 configured to implement audio data storage in the form of audio delay lines. As shown in Fig. 5, the primary components of sound memory engine 74 are control bits 83, the TRAM data and address buffers 84 and 85 (part of memory 72), a delay base address counter 86, an align control circuit 87, an address generator 88 and a data formatter 90. The address generator is responsible for circular addressing for generating delay lines. The data formatter is used to compress 32-bit operands in memory 72 into the narrower word widths implemented by the physical TRAM during TRAM write operations and then to expand them back during read operations.

Sound memory engine 74 has access, shared with the instruction execution unit and the host interface, to blocks of RAM mapped into the address space of memory 72 which implement the TRAM address buffer and TRAM data buffer. These twin buffer memories (TRAM data buffer 84 and TRAM address buffer 85) hold data and address pairs which along with control bits 83, including an align bit 83a and read/write bits 83b, completely specify the activity of the TRAM engine during a sample period. In the way that microprogram memory 80 holds the program executed by the instruction execution unit every sample period, these buffers represent the "program" executed by the TRAM engine every sample period. Whenever a program compiled for sound effects engine 50 reads or writes to a delay line, it is actually locations in TRAM data buffer 84 that are used as the operands. Ordinarily, address offsets stored in TRAM address buffer 85 are constants initialized by the host processor, but any instruction executed by instruction execution unit 70 can compute a new delay line address by storing its results in the appropriate TRAM address buffer of memory 72.

During each sample period, the TRAM engine runs sequentially through each of the buffers' address/data pairs, so that the buffer contents represent an ordered list of TRAM accesses throughout the entire sample period. During every TRAM memory cycle within the sample period, a TRAM address offset is fetched by the sound memory engine from the TRAM address buffer GPRs, an absolute TRAM memory address is computed from the offset, and a signal value is either fetched from or written to the address paired TRAM data buffer location.

The TRAM data buffers are the source and sink for audio data flowing from TRAM, and each data buffer location is paired one-to-one with an address buffer location; the address buffer holds the address in the TRAM that corresponds to the data. Control bits 83 are a separate array (only writable from the host and not mapped into memory space 72) having a field associated with each data/address pair. Certain of these control bits (control bits 83b -- a two-bit field in this embodiment) specify what type of TRAM operation, e.g., read or write, should take place using the corresponding data and address pair.

By sharing access to the Address and Data buffers with the instruction execution unit, DSP computations and TRAM accesses are decoupled from each other in time and in memory 72 address space. In other words, there is no fixed temporal relationship between when the TRAM engine and instruction execution unit access memory 72. This decoupling is valuable because of the inherent difference in operational speed between the fast instruction execution unit and the relatively slow physical TRAM memory. Decoupling DSP and memory accesses through the shared buffers allows the instruction execution unit to proceed at full speed without regard to the access latencies and pipeline consequences that burden the sound memory engine and its physical memory interface.

Mapping the TRAM buffers into the address space of memory 72 allows the buffer locations to be accessed as operands from instructions for execution unit 70. In this way, data stored in the TRAM can be funneled through the TRAM data buffers and used as input or output operands in DSP programs. This transparent memory-mapping means that no special instructions are required for TRAM data access and no extra overhead is incurred waiting for slow TRAM cycles during an access to this data.

Memory 72

In this embodiment, memory 72 is actually made up of four separate physical memory sections. A more detailed diagram of memory 72 is shown in Fig. 6. As shown in Fig. 6, memory 72 includes 1024 locations, not all of which are used. All 1024 locations in memory 72 are addressable by instruction execution unit 70, however, and thus the entirety of memory 72 can be said to be addressable by a uniform addressing scheme.

There are four different function areas within memory 72: input/output space 83 (memory locations 0-40), general purpose memory space 92 (memory locations 100-200), TRAM data and address buffers 84 and 85 (memory locations 200-280: internal TRAM data; locations 280-2AO: external TRAM data; locations 300-380: internal TRAM addresses; and 380-3A0: external TRAM addresses) and hardware registers and constants space 94 (locations 40-60). These functional units do not necessarily correspond with the four physical memories that make up memory 72.

General purpose memory space 92 includes 256 locations that are 32-bits wide for storing general purpose operands for DSP programs run by instruction execution unit 70. This memory space is implemented by a single port memory that operates at four times the instruction rate thereby allowing each instruction to access three input operands and one output operand per cycle. Memory space 94 includes hard-wired frequently used constants along with space for various registers.

As previously stated, TRAM data and address buffers 84 and 85 are ports to and from internal and external TRAMs 76 and 79. Each data buffer location is paired with an address buffer location. The data buffer holds the TRAM data that was read at, or will be written to, the TRAM address contained in the corresponding address buffer location. Like general purpose memory space 92, these buffers and implemented by a register file memory and appear in the DSP program's operand space. Although they appear in separate parts of memory 72's address space, the data and address portions of the TRAM buffers are accessed in parallel by TRAM engine 74.

In this embodiment, the TRAM buffers are not a full 32-bits wide, but instead are limited to 20-bits. These 20 bits accommodate both the 20-bit TRAM addresses and 20-bit decompressed TRAM data in a single, shared internal RAM. The 20 data bits are left-justified into the most significant position (31) of the 32-bit data path. The 20 address bits, however, are justified into the next significant bit (30) of the data path with the MSB read as a 0. This guarantees that addresses always appear as positive quantities to the instruction execution unit.

<u>Implementation of Delay Lines</u>

Circular addressing refers to the ability to implement delay lines using memory pointers which recirculate through a region of memory. A delay line's read pointer follows its write pointer around the circle and the pointers advance at the sample rate; the distance in memory between the read and write pointer is the delay line length. Circular addressing is also known as modulo addressing since all address calculations are performed modulo the region size, which is typically a power of two for simplicity of implementation.

Even though the read and write pointers are referred to as addresses, in effects processor 50 they are not implemented as physical TRAM memory addresses but as address offsets. The initial value of the offsets can be fixed by an assembler and associated driver software as would be understood by a person of ordinary skill in the art. To get the physical memory address for each read and write operation the offsets are summed with the delay base address counter. This base address counter decrements at the sample rate; thus the sums of the read and write offsets and the base address also decrement at the sample rate, and if the write offset is smaller than the read offset, the delay length will be the difference of the two as the read pointer chases the write pointer downward through memory.

This is described below and illustrated in Figs. 7A-7C for a four sample delay line implemented in a hypothetical 32-sample TRAM. The numbers highlighted in the circular memory show which values in the delay line are currently "active," waiting to be retrieved. The write offset of the delay line is 0, and the read offset is 4 -- these values remain constant. Initially, the base address (BA) is at 0 (Fig. 7A), so the physical address of the write operation is also 0, where the value 7 is written.

After four sample periods, the base address counter BA has decremented, modulo the TRAM size of 32, from 0 down to 28 (Fig. 7B). This is where the newest value of -2 will be written to the head of the delay line, where the base address value of 28 is summed with the write offset of 0. The fixed read offset of 4 is also added to the base address, modulo 32, to give a physical read address of 0, where it retrieves the value 7 which was written 4 samples previously.

The read pointer leaves values in its wake which are now inactive and can be discarded. As the base address counter begins to decrement back towards 0 it will begin reusing previously written-to memory (Fig. 7C), as the write pointer overtakes discarded values.

This example of delay line implementation in audio signal processor 10 shows that, unlike many other DSPs, circular delay line addressing is not performed modulo the size of the delay line, but modulo the size of the physical address space (which in this miniaturized and simplified example is 32 locations). This means that multiple delay lines of different lengths can occupy the same physical address space and still use the same modulo arithmetic. An example of this is shown in Fig. 7D. In Fig. 7D, the original delay line of Fig. 7A which writes at offset 0 and reads at offset 4 is joined by 3 other delay lines with the following specifications:

2:	[Write@5, Read@10],
3:	[Write@14, Read@15]
4:	[Write@16, Read@28]

For the sake of complete generality, the example uses a Delay Base Address counter value of 2.

As can be seen in Fig. 7D, the four delay lines, each with distinct read and write offsets relative to the Delay Base Address counter, reside simultaneously in TRAM and simply chase each other around the modulo circle formed by the physical address space. Note also that the 4 delay lines do not fill up all of the available memory space: a memory gap exists unused between the operation of a Read@10 and a Write@14, and similarly, there is unused memory past the last Read@28. These areas are available for additional delay lines, or alternatively, the space leftover can be used to allow the delay lines which Read@10 and Read@28 to have their read pointers modulated, that is, expanded and contracted dynamically, without interfering with the other delay lines.

Audio effects processor 50 also makes it easy to modulate delay line addresses, since addresses are stored in memory 72 in the TRAM address buffer 84 and can be accessed just like any other arithmetic operand. The length of a delay line is modulated by computing new addresses for the read and/or the write pointer of the delay line, although it is most typical to change only the read pointer.

The length of a delay line is the distance in samples from the write address to the read address, so to properly modulate a delay line the new computed delay length must be summed with the delay line's write address before storing it in the TRAM address buffer. Since the write address stored in the buffer is eventually used again as an offset from the delay base address counter to get the physical TRAM address, the modulated delay length really represents an additional offset from an offset from the delay base address counter. Since the delay line's write address is already available in the TRAM address buffer, the read offset can be computed with an instruction of the form:

$$\text{Read Address} = \text{Write Address} + (\text{Mod Function} \times \text{Max Delay Length})$$

where Max Delay Length is the declared maximum length of the delay line, and the Mod Function is a control signal that varies from 0 to 1.0.

An example of a recall hazard from a TRAM read operation is shown in Fig. 8A. Fig. 8A shows two TRAM read operations, one from address offset 100 and the other from offset 110. The Base Address Counter, which decrements by 1 each sample period, is added to the TRAM offsets stored in the TRAM address buffer during every access to form a physical TRAM memory address. For the sake of simplicity, this example assumes that the TRAM and instruction execution engines are sequencing at the same rate through the buffer and microprogram memories, respectively.

The fetch operation occurs as the TRAM engine encounters the TRAM read offset 100. It initiates a memory fetch from physical address 101 and after a TRAM access delay time the data from 101 appears in the data buffer. It is subsequently read from the buffer by the execution of a microinstruction. In this case, the fetch preceded the read, and the microinstruction received its data from TRAM address 101, offset from the base address of 1 by 100, just as it expected.

Some number of instruction cycles later, a microinstruction seeking data from offset 110 reads its data from the buffer location nominally filled with data from 110. In this case, however, the read operation precedes the TRAM fetch, which won't occur for another few cycles, since the TRAM offset 110 is stored in the address buffer a few locations past the microinstruction word. Instead of receiving data from TRAM address 111 (the offset 110 plus the current Base Address Count of 1), the microinstruction receives data that was fetched during the previous sample period. At that time the Base Address Counter was 2, so the datum in the buffer that is read is actually from TRAM address 112, delayed exactly one sample period too long.

This off-by-one error occurs whenever microinstruction reads precede TRAM buffer fetches. A 1-bit address offset (align bit 83a) along with align control circuit 87 provides a solution to this problem by canceling out the off-by-one error whenever microinstruction reads precede TRAM engine fetches. Using the same example as before, this solution is illustrated in Fig. 8B. As shown in Fig. 8B, the TRAM buffer structure has been augmented to include a field containing address align bit 83a, which is paired 1-to-1 with the TRAM addresses. Align control circuit 87 examines align bit 83a and provides an input to address generator 88 that is used in determining the physical TRAM address.

A problem symmetrical to read hazards can exist for TRAM write operations. This situation is illustrated in Fig. 8C. When microinstructions write results to TRAM, the result data is buffered just as for read operations, waiting for the TRAM engine to flush the result out to memory. There is no problem if microinstructions that write to the TRAM are executed before the flush operation takes place. The difficulty arises when the TRAM offset and data are stored in buffer locations preceding the microinstruction word. In these cases the TRAM engine is actually flushing data that was written by the microcode during the previous sample period; the effective result is that the delay line includes one sample period of delay beyond its nominal length.

To address this issue, sound memory engine 74 ensures that during the subsequent sample period that the data is flushed out to exactly where it should have ended up. If the base address is counting down, this means "1" is added to the offset to compute the physical address to which we should have flushed the data during the previous sample period. Again, and as shown in Fig. 8D, align bit 83a can be used by align control circuit 87 to cancel this off-by-one error for write operations.

The difference in this case is that the align bit signifies a "+1" offset going into the TRAM address engine instead of the "-1" used for read operations. This is easily accomplished by taking the read/write bits associated with each address (not shown) into account.

One strategy for setting the align bits is to use a microcode loader (software driver), which keeps track of the location of all offsets in the TRAM address buffer as well as all instructions which use the TRAM data corresponding to those offsets. This "loader" may be running either at microcode download time or might really just be a "locator" which only needs to run at microcode build time. With information and knowledge about the relative execution rate of the microcode and TRAM engine sequencers, the loader can determine which TRAM data will be read from the buffer before it is actually fetched from memory or written after it has been flushed. It will flag such situations by setting the align bit corresponding to the TRAM offset associated with that data. The align bit is then fed into TRAM address engine 88 and used as an extra offset of ± 1 when forming the physical TRAM address.

Another approach is to use hardware in sound memory engine 74 to automatically generate the align bit offsets. The microprogram sequencer restarts on the same cycle during which the TRAM engine sequencer is restarted at the beginning of the buffer and the base address count is incremented. At this time, the align bit is cleared for every read offset, and set for every write offset. As the microcode accesses the TRAM data buffer for read or write operations, the align bit for the corresponding offset is set for every read offset and cleared for every write offset. Thus, "misaligned" TRAM fetch or flush operations will automatically have used the correct sense of the align bit. This approach is less preferred because other measures must be taken to ensure that it is not possible to slip a microinstruction access to the TRAM data buffer into the pipeline delay between address usage and data fetch. Without such assurance, the microinstruction access may not have the opportunity to update the alignment bit even though it was accessing last sample period's buffer data, and the physical address may never be correct.

<u>Initialization of TRAM Data and Address Buffers 84 and 85</u>

When delay lines are used, TRAM read and write operations start as soon as power is received by effects processor 50. Write operations do not pose a problem, but unless TRAMs 76 and 78 are properly initialized, read operations will initially produce noise. Thus, it is important to initialize the TRAMs. In one embodiment, these memory spaces are initialized by writing 0s to each location before executing a program in sound processor 50. This can be a time consuming task, however. Thus, in other preferred embodiments, dedicated hardware circuits are employed to perform such an initialization routine. As would be understood by a person of skill in the art, a variety of such circuits can be designed and employed in the present invention.

Other Embodiments

The above is a full, detailed description of one specific embodiment of the present invention. It should be understood that many specifics of the invention described above can be changed without departing from the concept of the invention. For example, word size, address field length, memory size, number of control bits 83 used and other implementation-specific items can vary in other embodiments. Also, while effects processor 50 was described as an audio effects processor that performs operations on audio signals in a sound processing system, a person of ordinary skill in the art will realize that effects processor 50 can also process radar signals, seismic signals or any other time series data in any signal processing system.

Many other equivalent or alternative methods of implementing audio effects processor 50 will be apparent to those skilled in the art. For example, it is possible for sound memory engine 74 to implement table lookup operations using non-circular indexed addressing in internal and external TRAM memories 76 and 78. Tables of data can be loaded into TRAM by host processor 18 and then accessed like static arrays whose indices are computed by instruction execution unit 70. The sound memory engine can distinguish delay line accesses from table-lookup accesses by the setting of additional TRAM control bits associated with each address. Table-based addressing can be useful for implementing non-linear functions, wavetables or excitation functions for synthesis, block-based coefficient storage, or for many other applications. Note that although "table lookup" implies a read operation, table-based addressing applies equally to write operations, so it is possible to have audio effects processor 50 compute table values in real time instead of performing such calculations with the host.

Table-based addressing is distinguished from delay line addressing by the use of a static table base address register in place of the constantly decrementing delay base address counter. In the same way the delay line accesses are always made with offsets relative to the delay base address counter, table lookups are always made relative to a table base address register. The table base address register is initialized by the host and mapped into the address space of memory 72.

If necessary, the table base address can be read (but not written) from DSP programs like any other instruction operand.

Initialization of the table base address register is typically performed by host 18 at system startup time. The address stored there partitions physical TRAM into a circular region and a table-lookup region. All delay line addresses are computed modulo the table base address instead of modulo the physical memory size, so that delay line operations automatically take place within the memory region from 0-[Table Base]. In this way the table base address register also serves in effect as a "delay limit" register.

In the same way that circular delay addressing treats the TRAM address buffer elements as offsets from the delay base address counter, table-based addressing uses these addresses as offsets relative to the table base address register. To compute the physical TRAM address, the TRAM address buffer contents are added to the table base address register. By calculating a table index with the proper scaling and storing it as an offset in the TRAM address buffer, DSP programs can perform computed table lookups.

All programs loaded into audio effects processor 50 use the same table base address register, whose value is typically established by the host at system initialization time. If more than one program requires table lookup storage, the individual tables belonging to different programs are partitioned from within the global table lookup region. DSP programs can declare their own private table offset registers using ordinary GPRs whose contents represent an offset from the table-based address register, and whose location and content will be properly relocated by the program loader. Then these private table offset register GPRs are summed with the computed table index before performing the table look-up operation in TRAM memory.

Typically table-lookup offsets will create addresses which are inside the table lookup region of the TRAM, but since the sound memory engine performs no address limiting or trapping it is possible for specialized applications to generate addresses outside this region. As for delay line addressing, the operand data path alignment of the TRAM address buffer guarantees that all TRAM table offsets are positive quantities, so to generate addresses below the table lookup region the program must generate addresses which wrap around the entire TRAM address space (2^{20} locations in one embodiment) and back into the delay line region.

In addition to implementing table look-up operations, other variations of the present invention are also possible. For example, while sound processing engine 40, sound processor 50 and interface unit 60 were all described as being part of a single chip sound processor, the functionality of these elements can be performed from separate ICs in other embodiments. Similarly, it is possible to develop an architecture in which instruction execution unit 70 and sound memory engine 74 are implemented on separate chips. Also, sound processor 30 can be included directly on a computer motherboard rather than a dedicated multimedia card 32, and in another data is not compressed and expanded when it is transferred between memory 72 and TRAM memories 76 and 78. These equivalents and other alternatives are intended to be included within the scope of the present invention.

WHAT IS CLAIMED IS:

1. A signal processor comprising:

(a) a first memory including a first address space;

(b) an instruction execution unit that processes signal processing instructions including instructions to read from and write to said first address space; and

(c) a signal memory engine that operates independent of said instruction execution unit to read from and write to said first address space and perform address arithmetic and memory accesses to a second memory as necessary to implement signal data storage.

2. The signal processor of claim 1 wherein said instruction execution unit executes instructions at a first execution rate and wherein said signal memory engine executes operations at a second execution rate different than said first execution rate.

3. The signal processor of claim 1 wherein said first address space includes a first memory location and wherein said signal memory engine and instruction execution unit can independently access said first memory location in a non-fixed temporal relationship.

4. The signal processor of claim 1 wherein said signal memory engine performs arithmetic and executes memory accesses to said second memory based on values stored in said first address space.

5. The signal processor of claim 1 wherein said signal data storage comprises audio data storage.

6. The signal processor of claim 5 further comprising a sound processing engine that synthesizes sounds.

7. The signal processor of claim 6 wherein output from said sound processing engine representing synthesized sounds is stored in a second address space of said first memory and wherein instructions processed by said instruction execution unit read from and write to said second address space.

8. The sound processor of claim 1 further comprising an instruction memory coupled to said instruction execution unit, said instruction memory comprising a microprogram containing a specified set of signal processing instructions to be executed by said instruction execution unit.

9. The signal processor of claim 8 wherein said instruction execution unit executes a single pass through its entire microprogram each sample period in an order in which the instructions appear in said microprogram without branches, loops or subroutine calls.

10. The signal processor of claim 9 wherein said instruction execution unit provides for conditional execution of instructions by ignoring execution results of selected instructions based upon values of a condition code for said selected instructions.

11. The signal processor of claim 8 wherein said instruction memory is a different memory than said first memory.

12. The signal process of claim 1 wherein said first memory is a randomly addressable memory.

13. The signal processor of claim 12 wherein said randomly addressable memory comprises a plurality of separate, physical RAM memories.

14. The signal processor of claim 12 wherein said first memory further includes a second address space for storing values representative of sound input and sound output.

15. The signal processor of claim 12 wherein said first memory further includes a third address space for storing frequently used coefficients, variables and intermediate values.

16. The signal processor of claim 15 wherein said instruction execution unit uses a uniform addressing scheme to access said first, second and third address spaces of said first memory.

17. The signal processor of claim 5 wherein said audio data storage is accessed using circular addressing to implement audio delay lines.

18. The sound processor of claim 5 wherein said audio data storage is accessed using table-based addressing.

19. A single chip sound processor comprising:

(a) a first memory including a first address space and a second address space;

(b) a second memory that stores sound processing instructions;

(c) a sound processing engine that synthesizes sounds and outputs data representative of said synthesized sounds to said second address space;

(d) an instruction execution unit configured to execute said sound processing instructions stored in said second memory, including instructions to reads from and write to said first and second address spaces; and

(e) a sound memory engine that operates independent of said instruction execution unit to read from and write to said first address space and perform address arithmetic and memory accesses to a second memory as necessary to implement audio data storage.

20. The sound processor of claim 19 wherein said instruction execution unit executes instructions at a first execution rate and wherein said sound

memory engine executes operations at a second execution rate different than said first execution rate.

21. The sound processor of claim 19 wherein said first address space includes a first memory location and wherein said sound memory engine and instruction execution unit independently access said first memory location in a non-fixed temporal relationship.

22. A computer processing system comprising:

(a) a bus;

(b) a host processor communicatively-coupled to said bus;

(c) a second memory; and

(d) a sound processor communicatively-coupled to said bus, said sound processor comprising:

(i) a first memory including a first address space;

(ii) an instruction execution unit that processes sound processing instructions including instructions to read from and write to said first address space; and

(iii) a sound memory engine that operates independent of said instruction execution unit to read from and write to said first address space and perform address arithmetic and memory accesses to a second memory as necessary to implement audio data storage; and

(iv) a host interface unit that interfaces said second processor to said bus.

23. The computer processing system of claim 22 wherein said host processor can read from and write to said first address space.

24. The method of implementing an audio delay line in a sound processor, said method comprising the steps of:

(a) executing a sound processing instruction in an instruction execution unit of said sound processor to store data representative of a sound to be delayed in a first memory;

(b) retrieving, independent of step (a), said data from said first memory with a sound memory engine of said sound processor and writing said data to a second memory; and

(c) performing address arithmetic and subsequent memory accesses to said first and second memories with said sound processor to implement the audio delay line.

25. The method of claim 24 further comprising the step of executing an instruction in said sound processor to write data stored in said first memory by said sound memory engine to a different location in said first memory for output from said sound processor.

26. A method of operating a computer processor, said method comprising the steps of:

(a) synthesizing an audio signal with a sound synthesis engine of said computer processor;

(b) outputting said audio signal to a sound effects engine of said computer processor;

(c) storing said outputted audio signal in a first location of said memory coupled to said sound effects engine;

(d) executing a first sound processing instruction in an instruction execution unit of said sound effects engine to transfer said stored audio signal to a second location of said memory, said second location being paired with an address to a location in a sound delay memory; and

(e) performing, independent of step (d), address arithmetic and memory accesses to said sound delay memory with a sound memory engine independent of step (d) to implement an audio data storage.

27. The method of claim 26 wherein step (e) comprises the step of, after storing said audio signal in said sound delay memory for at least one sample period, writing said audio signal to a third memory location of said first memory.

28. The method of claim 27 further comprising the step of executing a second sound processing instruction in said instruction execution unit to write data stored in said third location of said first memory by said sound processing engine to a fourth location of said first memory for output from said sound effects engine.

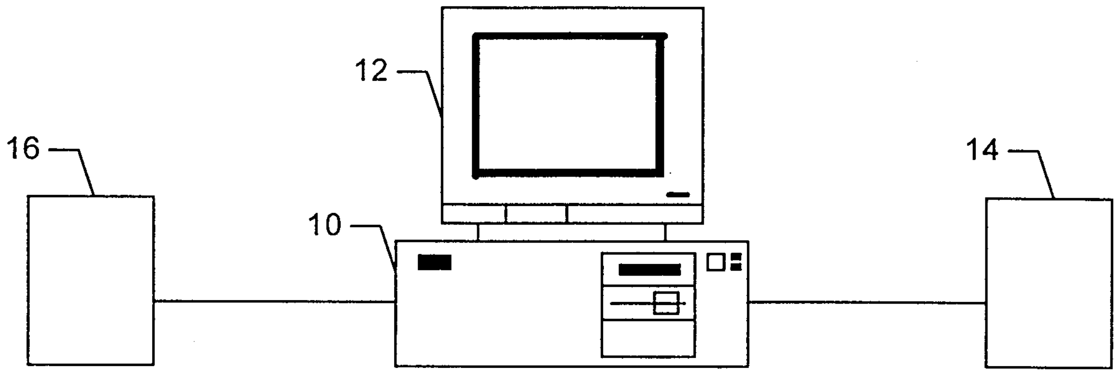

12
16
14
10

Fig. 1A

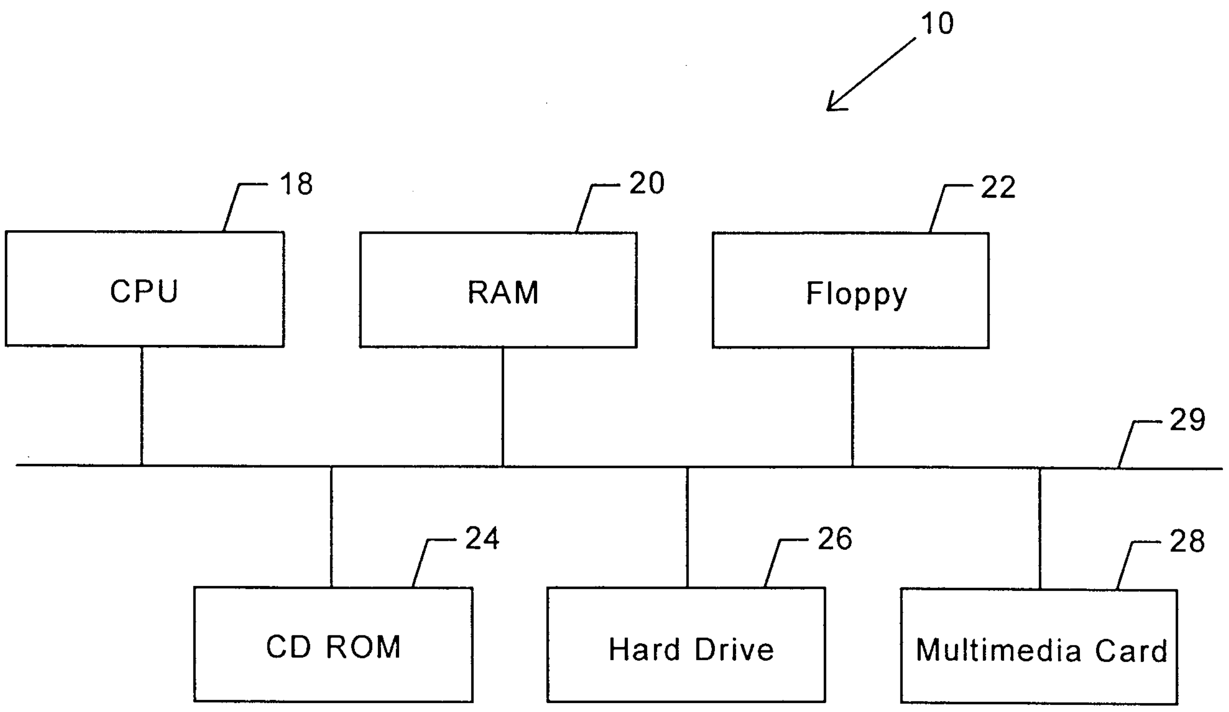
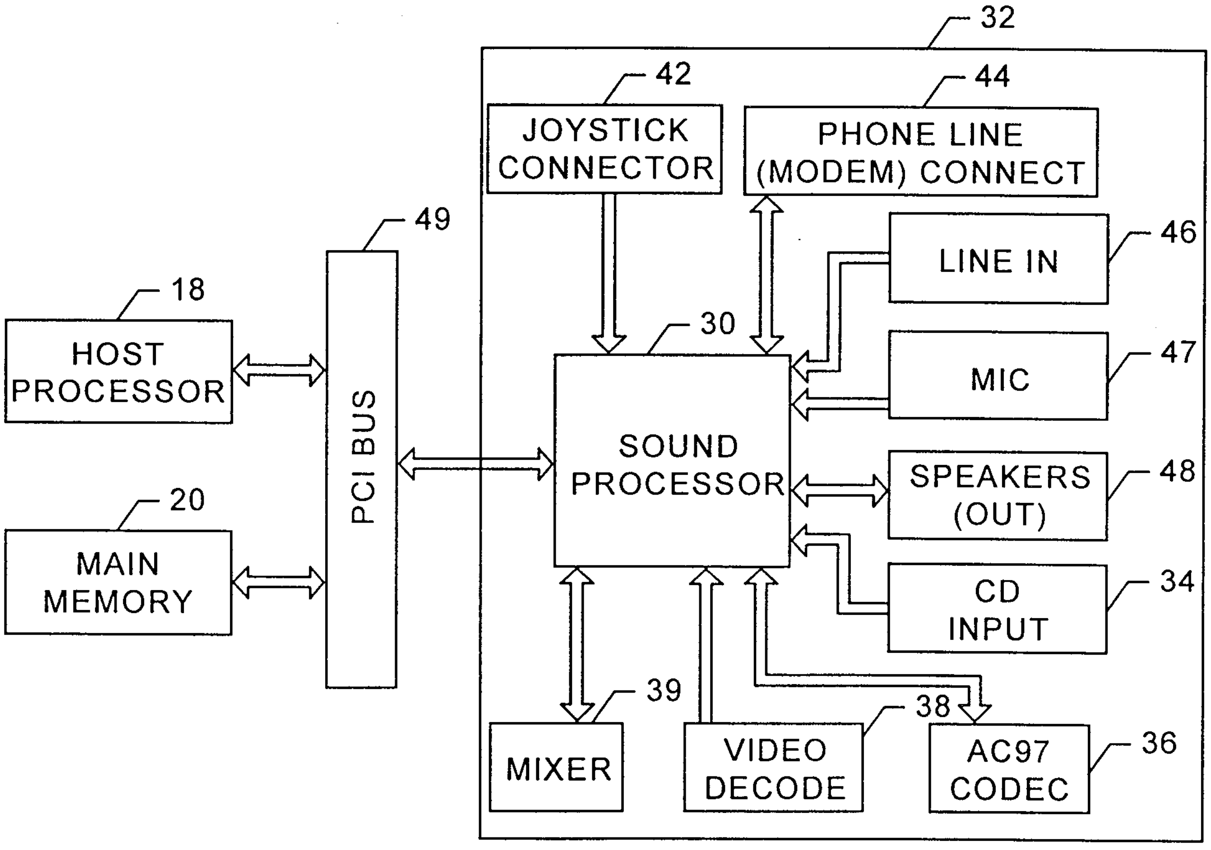
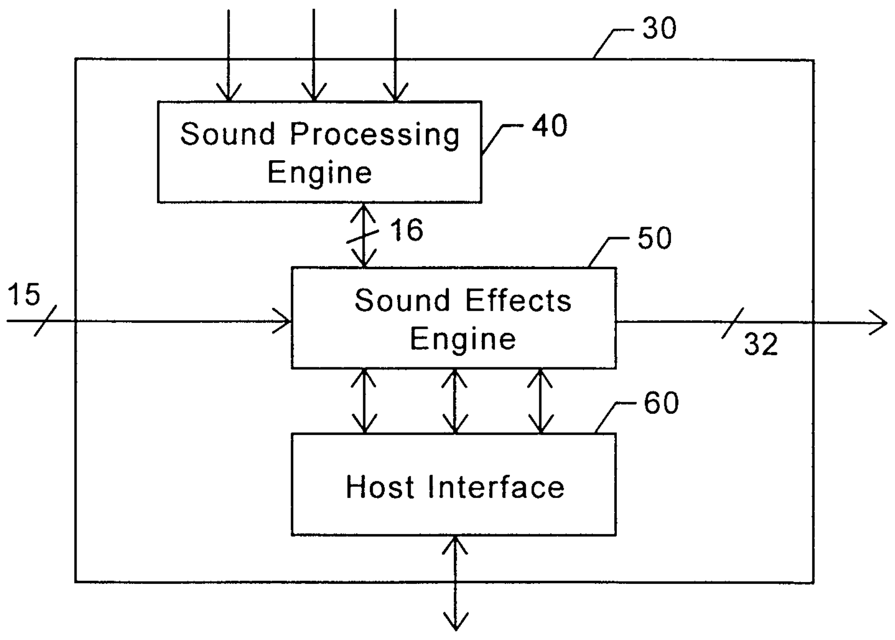

10
18
CPU
20
RAM
22
Floppy
29
24
CD ROM
26
Hard Drive
28
Multimedia Card

Fig. 1B



32
42
JOYSTICK CONNECTOR
44
PHONE LINE (MODEM) CONNECT
49
46
LINE IN
18
HOST PROCESSOR
30
47
MIC
PCI BUS
SOUND PROCESSOR
48
SPEAKERS (OUT)
20
MAIN MEMORY
34
CD INPUT
39
38
MIXER
VIDEO DECODE
AC97 CODEC
36

Fig. 2



30
Sound Processing Engine
40
16
50
15
Sound Effects Engine
32
60
Host Interface

Fig. 3

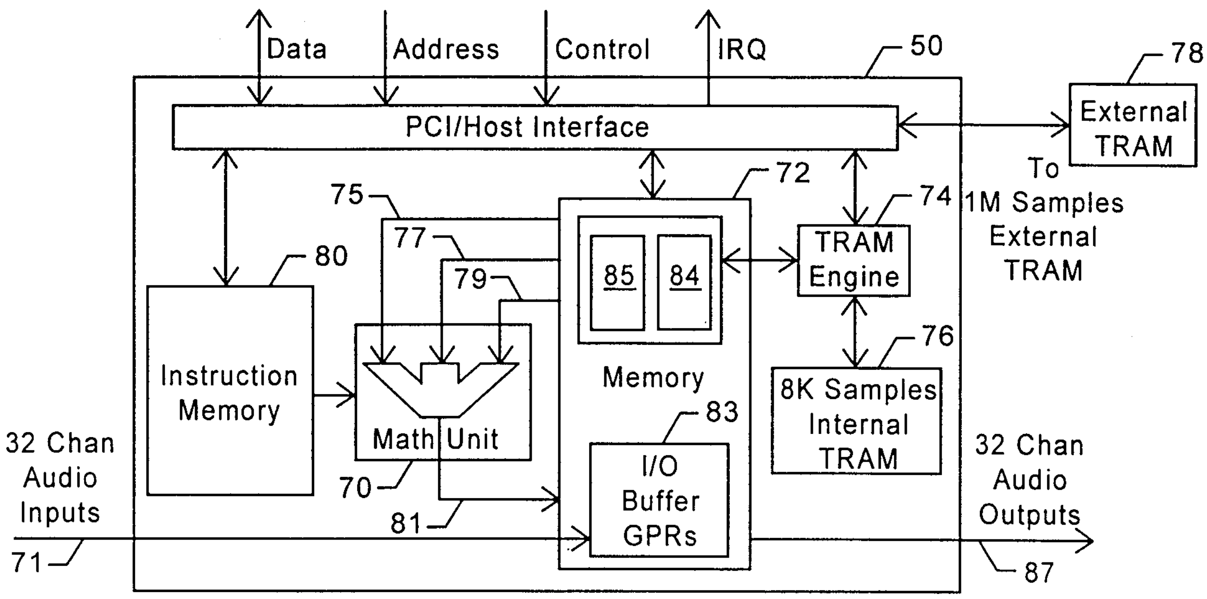

Data
Address
Control
IRQ
50
78
External TRAM
PCI/Host Interface
To 1M Samples External TRAM
72
74
75
77
79
80
85
84
TRAM Engine
Instruction Memory
76
Memory
8K Samples Internal TRAM
83
Math Unit
32 Chan Audio Inputs
I/O Buffer GPRs
32 Chan Audio Outputs
70
81
71
87

Fig. 4

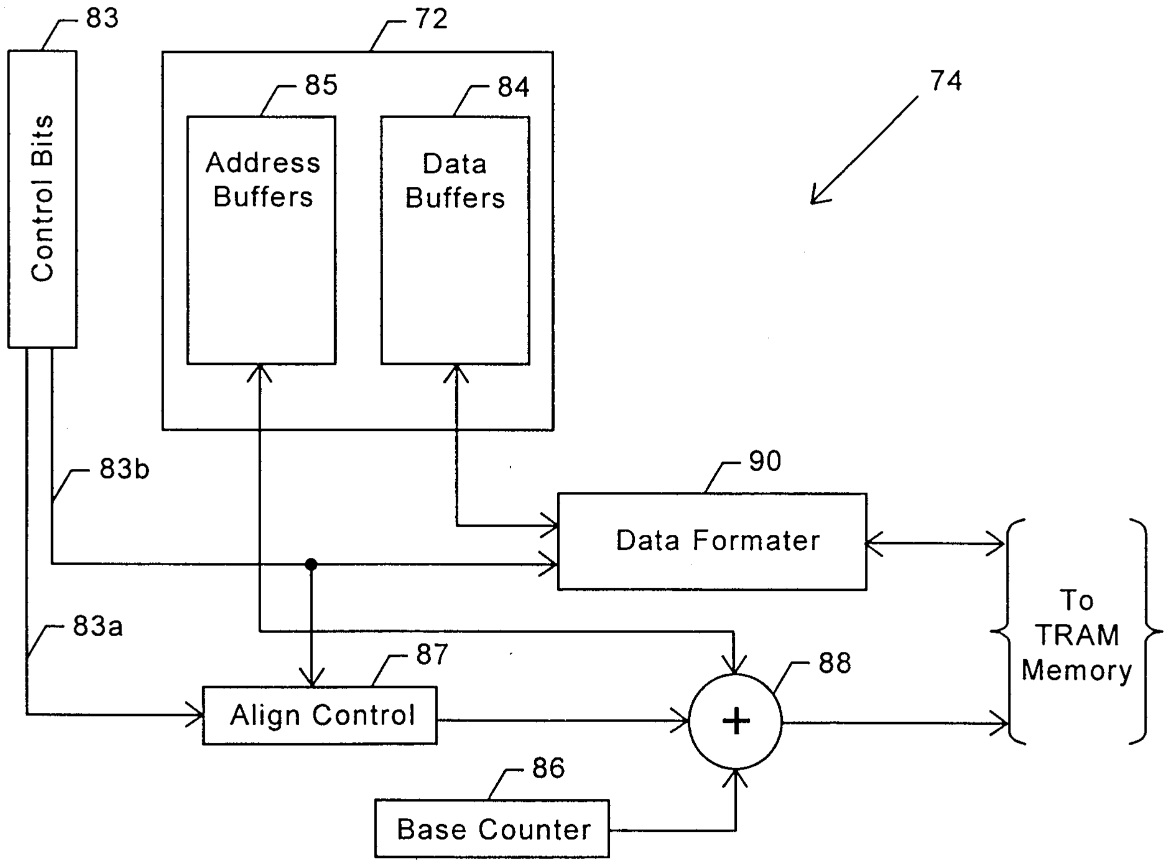

83
72
85
84
74
Control Bits
Address Buffers
Data Buffers
83b
90
Data Formater
83a
87
88
To TRAM Memory
Align Control
+
86
Base Counter

Fig. 5

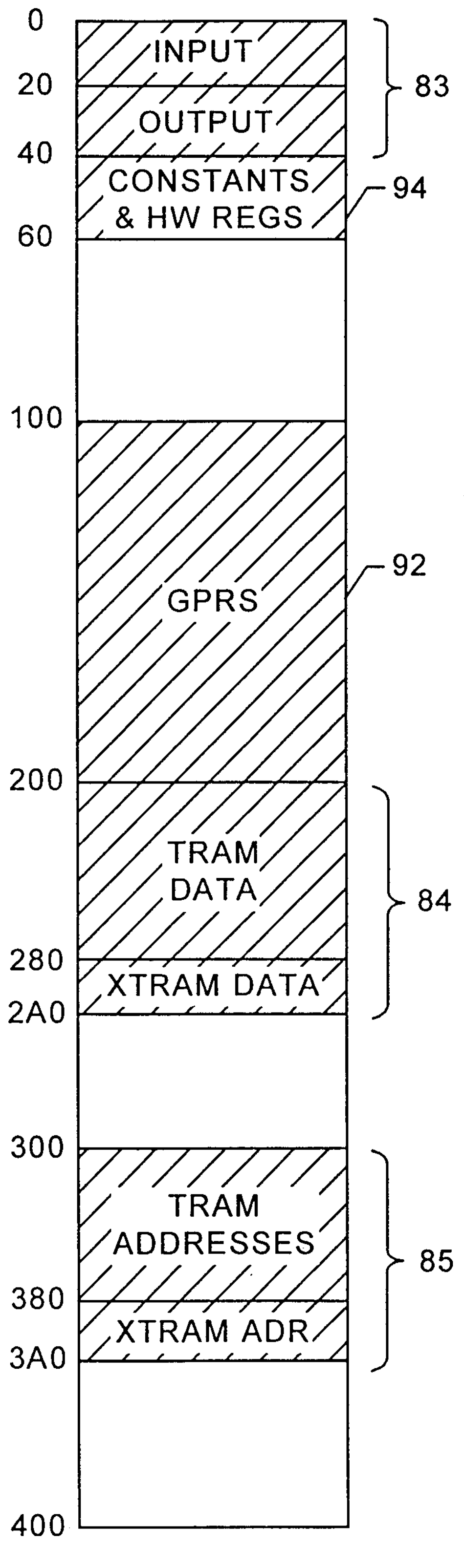

0
20
40
60
100
200
280
2A0
300
380
3A0
400
INPUT
OUTPUT
CONSTANTS & HW REGS
GPRS
TRAM DATA
XTRAM DATA
TRAM ADDRESSES
XTRAM ADR
83
94
92
84
85

Fig. 6

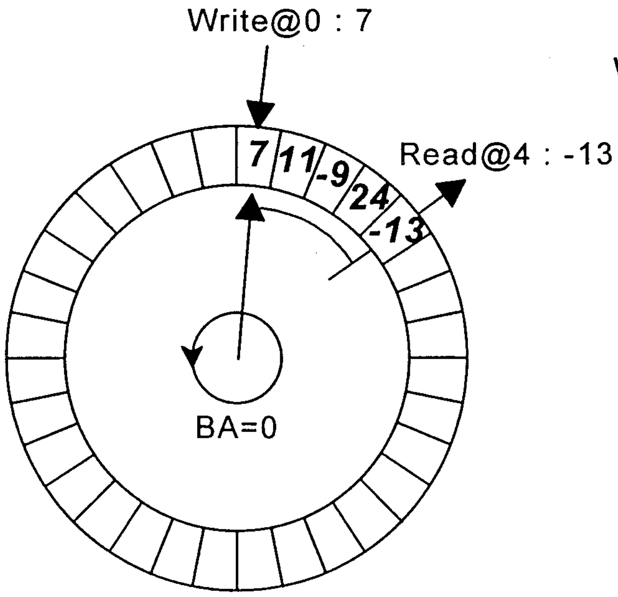


Fig. 7A

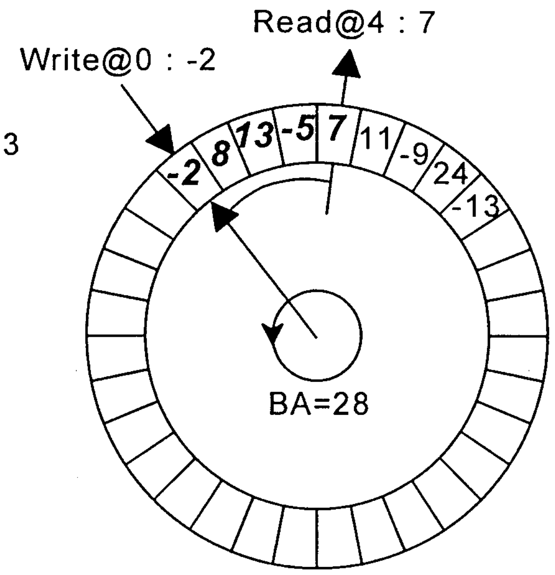


Fig. 7B

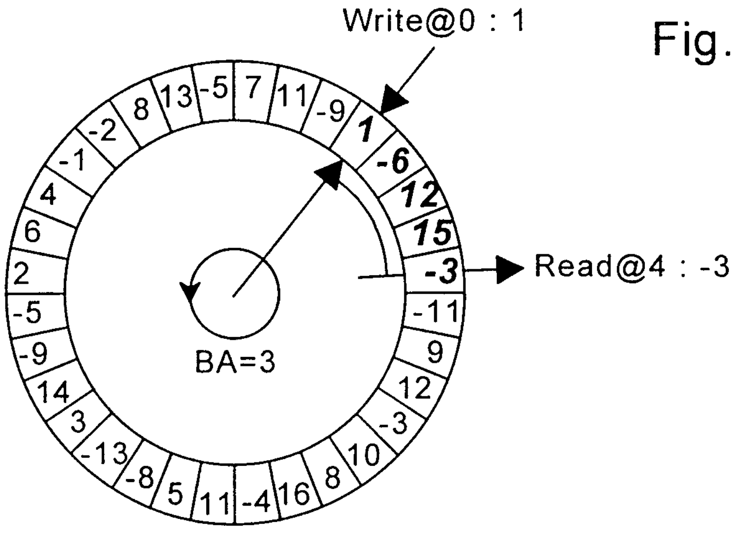


Fig. 7C

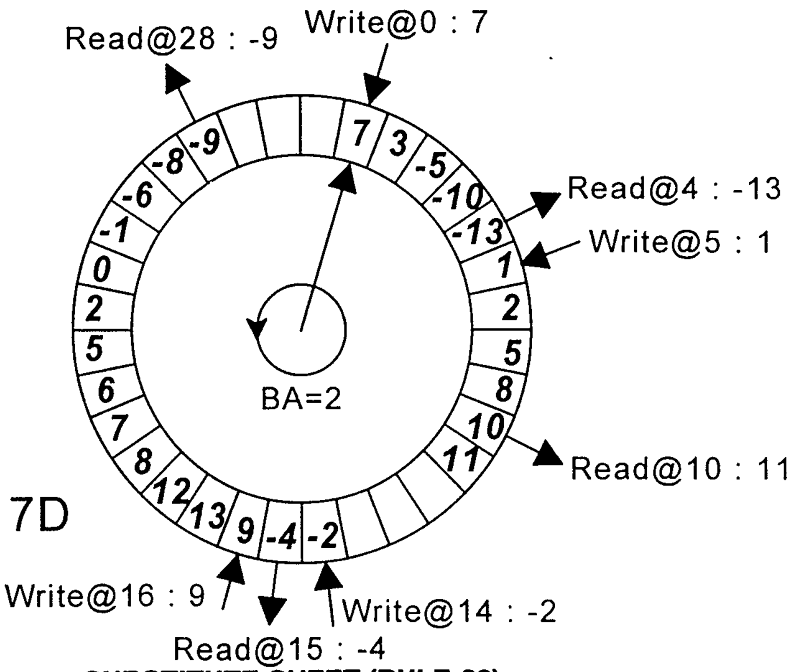


Fig. 7D

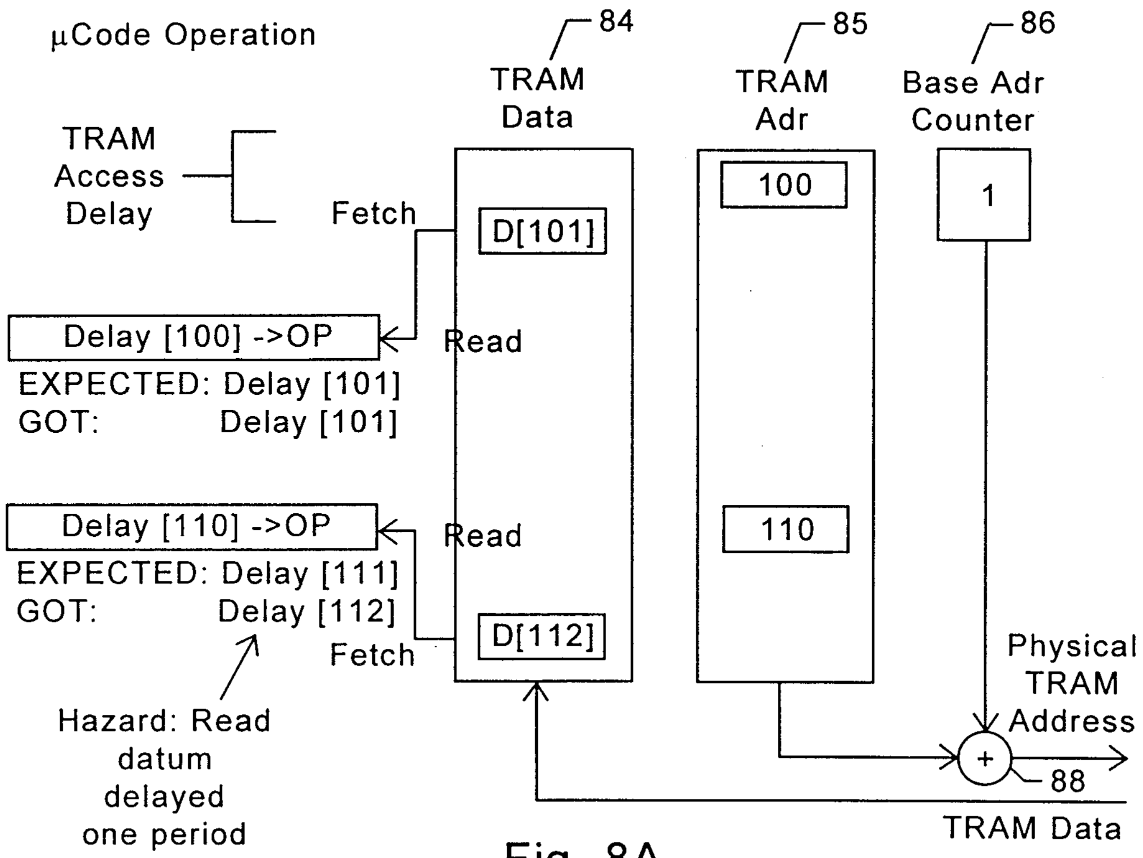

µCode Operation
84
TRAM Data
85
TRAM Adr
86
Base Adr Counter
TRAM Access Delay
Fetch
D[101]
100
1
Delay [100] ->OP
Read
EXPECTED: Delay [101]
GOT: Delay [101]
Delay [110] ->OP
Read
110
EXPECTED: Delay [111]
GOT: Delay [112]
D[112]
Fetch
Physical TRAM Address
Hazard: Read datum delayed one period
+
88
TRAM Data

Fig. 8A

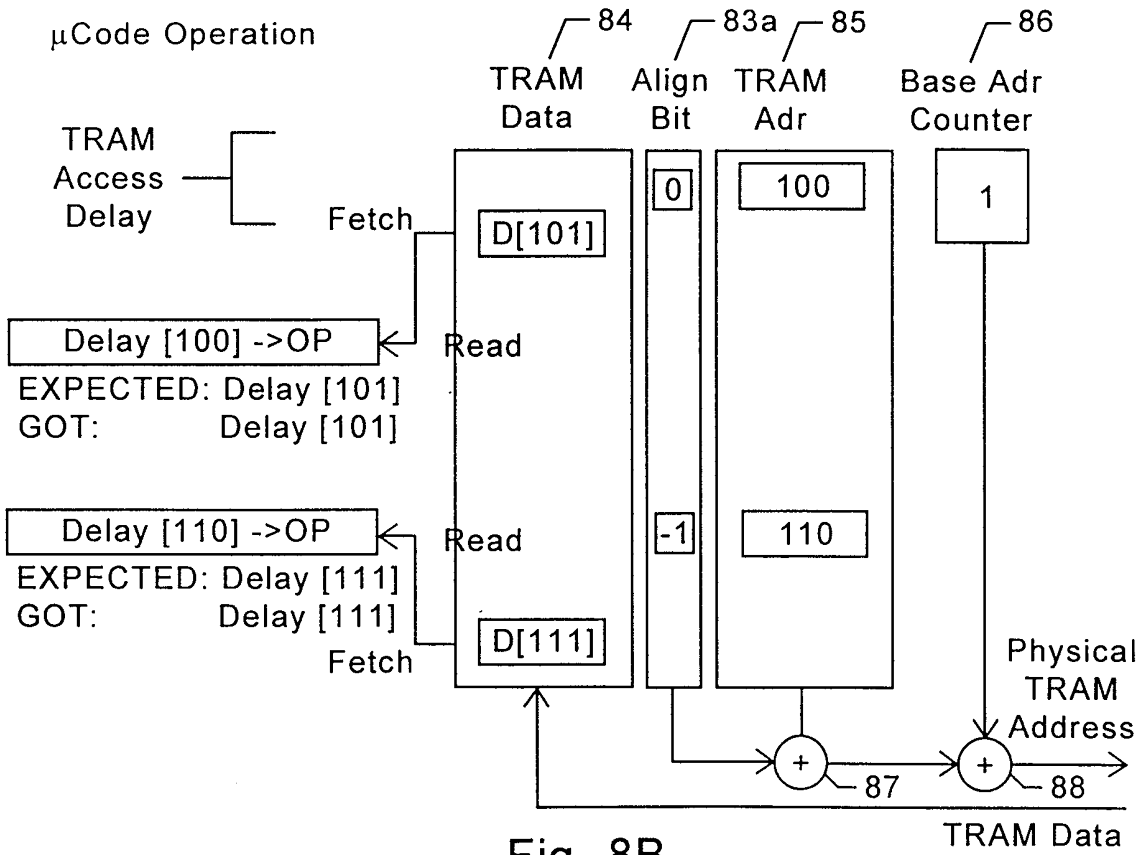

µCode Operation
84
83a
85
86
TRAM Data
Align Bit
TRAM Adr
Base Adr Counter
TRAM Access Delay
0
100
1
Fetch
D[101]
Delay [100] ->OP
Read
EXPECTED: Delay [101]
GOT: Delay [101]
Delay [110] ->OP
Read
-1
110
EXPECTED: Delay [111]
GOT: Delay [111]
D[111]
Fetch
Physical TRAM Address
+
87
+
88
TRAM Data

Fig. 8B

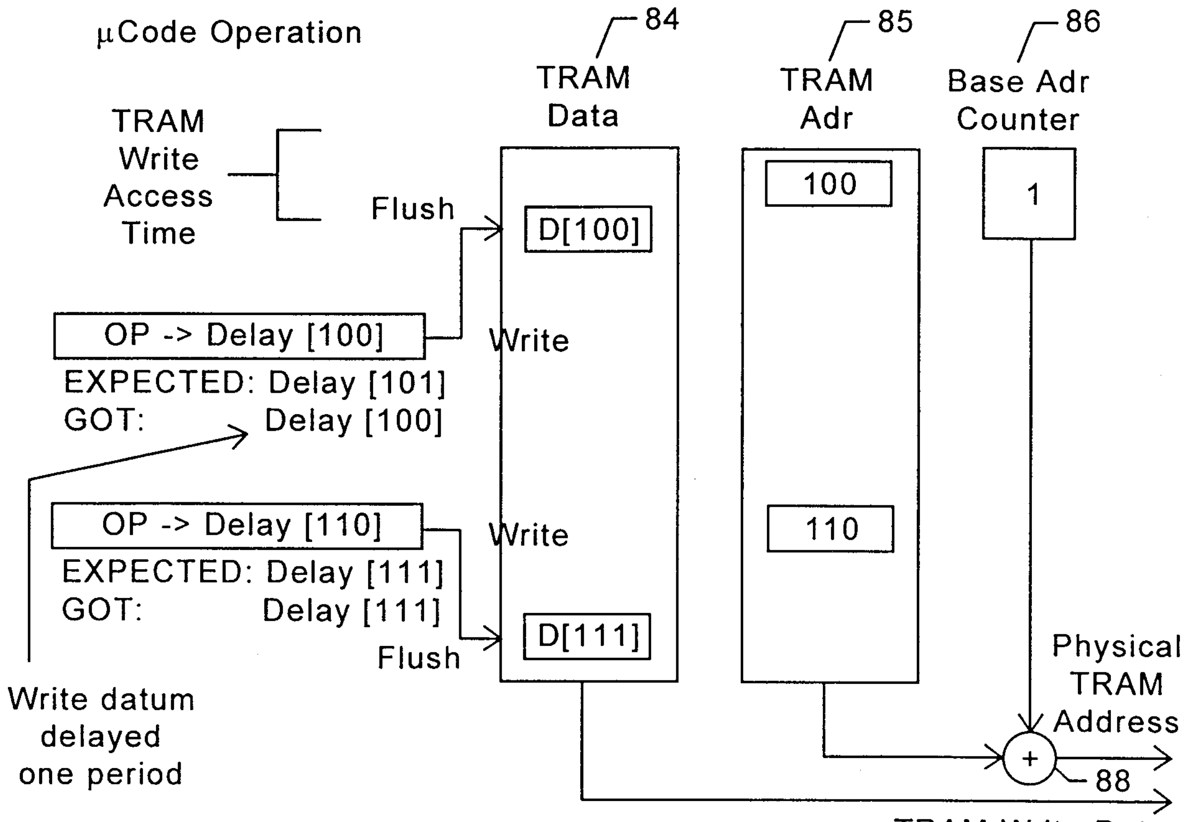

μCode Operation
84
TRAM Data
85
TRAM Adr
86
Base Adr Counter
TRAM Write Access Time
Flush
D[100]
100
1
OP -> Delay [100]
Write
EXPECTED: Delay [101]
GOT: Delay [100]
OP -> Delay [110]
Write
110
EXPECTED: Delay [111]
GOT: Delay [111]
D[111]
Flush
Physical TRAM Address
Write datum delayed one period
+
88
TRAM Write Data

Fig. 8C

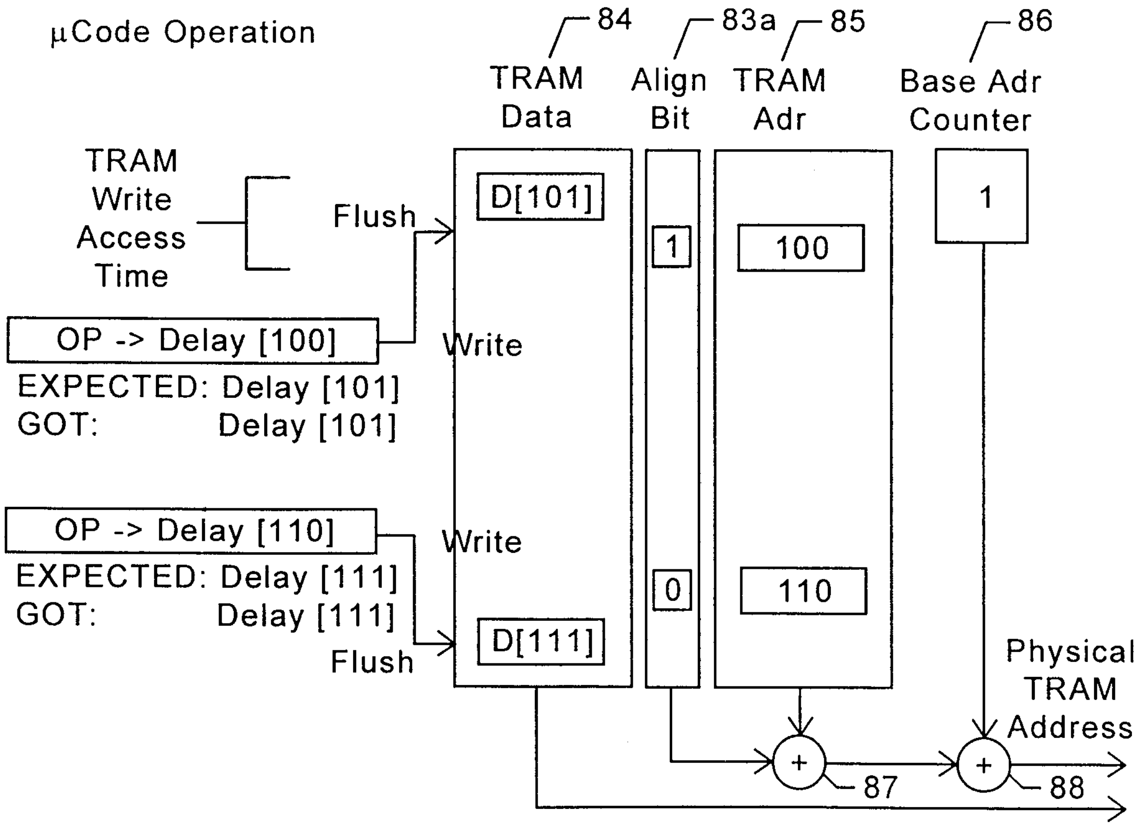

μCode Operation
84
83a
85
86
TRAM Data
Align Bit
TRAM Adr
Base Adr Counter
TRAM Write Access Time
D[101]
Flush
1
100
1
OP -> Delay [100]
Write
EXPECTED: Delay [101]
GOT: Delay [101]
OP -> Delay [110]
Write
0
110
EXPECTED: Delay [111]
GOT: Delay [111]
D[111]
Flush
Physical TRAM Address
+
87
+
88
TRAM Write Data

Fig. 8D