PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 7/38

A1

(11) International Publication Number: **WO 95/27939**

(43) International Publication Date: 19 October 1995 (19.10.95)

(21) International Application Number: PCT/US95/04354

(22) International Filing Date: 6 April 1995 (06.04.95)

(30) Priority Data:
224,452 7 April 1994 (07.04.94) US

(71) Applicant: MEDIA VISION, INC. [US/US]; 47300 Bayside Parkway, Fremont, CA 94538 (US).

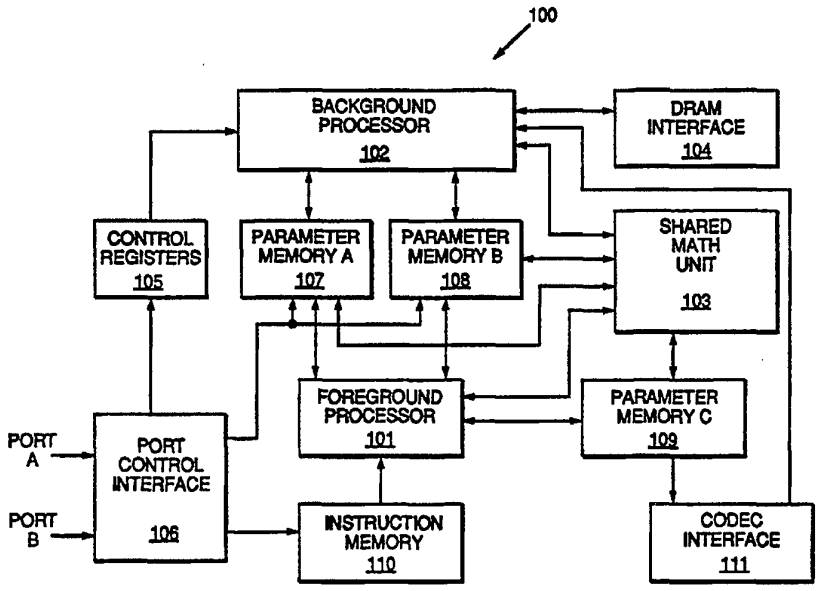
(72) Inventors: COLVIN, Bryan, J., Sr.; 4182 Cherry Avenue, San Jose, CA 95118 (US). GOCHNAUER, Daniel, B.; 18825 Allendale Avenue, Saratoga, CA 95070 (US). COOK, Perry, R.; 1095 Middlefield Road, Palo Alto, CA 94301 (US).

(74) Agents: GUILLOT, Robert, O. et al.; Bronson, Bronson & McKinnon, Suite 600, Ten Almaden Boulevard, San Jose, CA 95113 (US).

(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SI, SK, TJ, TT, UA, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, 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), ARIPO patent (KE, MW, SD, SZ, UG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: MUSICAL INSTRUMENT SIMULATION PROCESSOR



(57) Abstract

A self-contained fully programmable digital signal processor (100) has two processors (101, 102) sharing, in parallel interleave fashion, a math unit (103) such as a multiply-and-accumulate circuit. A background processor (102) controls an external dram and preprocesses the information for a foreground processor (101). On-chip sram (107, 110) stores program parameters for both the foreground and background processors and facilitate information transfer between the foreground and background processors. The sram is time-multiplexed to permit access by the foreground processor, the background processor, and external devices without the expense of multiport sram. Flip-flops maintain data signals to the math unit while the sram is being accessed. The foreground processor has a custom instruction set that optimizes the implementation of complex music synthesis filter structures. An on-chip white noise generator quickly provides pseudorandom data for some of the instructions.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

MUSICAL INSTRUMENT SIMULATION PROCESSOR

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to digital signal processors, to methods for generating digital sound signals, and to using parallel processors to execute, without pipeline delays, sound synthesis models that simulate the sounds of musical instruments.

Description of Related Art

A digital synthesizer typically generates a series of digital values which represent sound amplitudes at a series of discrete sampling times. Feeding the series of values through a digital-to-analog converter (DAC) or a coder-decoder (CODEC) to an amplifier and then to speakers produces sound.

Synthesizers use many synthesis methods to produce sounds that emulate the sounds of musical instruments. One of the most accurate methods for emulating a musical instrument is playing a recording of the instrument. This is called sample synthesis and is commonly used in drum machines. However, sample synthesis cannot practically mimic every musical instrument because some instruments produce many different sounds, and storing digital recordings of every sound requires too much memory. Accordingly, synthesis models have been developed which use computational power to reduce the required recorded information while still producing accurate emulations.

Attack Decay Sustain Release (ADSR) curves are used in many synthesis models. ADSR curves are amplitude envelopes which control the volume and duration of notes. For example, a synthesizer may generate a series of steady state sound amplitude

values and then multiply each sound amplitude value by a corresponding value from an ADSR curve. The duration of the note depends on how quickly the ADSR curve goes to zero.

Wave table synthesis models musical instruments using two circular sound tables. One table represent sound harmonics during the attack. The other table represents the steady state. Two ADSR curves provide envelopes for the tables. For musical instruments that don't have a steady state, a third ADSR curve can be used to control filter parameters.

FM synthesis uses two or more ADSR curves that control sine wave generators which are frequency modulated to create a large spectrum of harmonics. This technique allows a continuously changing spectrum of harmonics to follow the ADSR curves. It also uses no memory. A drawback of this technique is the difficulty in determining model parameters which provide a good emulation of a desired instrument.

Wave guide synthesis emulates a musical instrument using models based on the physics of the instrument. The wave guide models, being based on a physical structure, are more intuitive for many developers of music instrument emulations. The theory of lossless wave guides simplifies calculations needed to make modeling of many musical instruments achievable. The Karplus Strong algorithm (Plucked String model) is a predecessor of wave guide synthesis models. U.S. patent No. 4,984,276 by Julius O. Smith, issued January 8, 1991, provides an example application of wave guide techniques and is incorporated by reference herein in its entirety.

Synthesizers typically employ digital signal processors (DSPs) that execute software which implements synthesis models such as those described above. DSPs typically include math units such as multipliers and summers which are fed data and model parameters from memory. Data is often pipelined into the math unit, for example by decoding an instruction

and fetching data for the next cycle of the math circuitry before the current cycle is complete. To avoid delays with a pipelined system, the new data is fed into the pipeline before the previous cycle is finished. If data required for the next cycle depends on the results of the current cycle, then the data is not ready when required and operation of the math unit is delayed until the required data travels through the pipeline. Accordingly, pipeline delays make DSPs slower because the math units have periods of inactivity. When executing synthesis models such as wave guide models, iterative operations are often required and pipeline delays can significantly decrease effective performance. Accordingly, efficient synthesizer architectures are needed which executed synthesis models without experiencing pipeline delays.

SUMMARY OF THE INVENTION

The current invention provides a DSP that is fast, relatively inexpensive, and well suited to implementing musical instrument simulations models such as wave guide models. One embodiment of the present invention is a DSP that includes first and second processors which share a math unit. The two processors alternate controlling data input to the shared math unit, so that the shared math unit alternates between performing an arithmetic operation for the first processor and performing an arithmetic operation for the second processor. Results of the math unit processing for the first processor are stored while the math unit is processing data for the second processor, so that the results of the operation for the first processor are made ready for the math unit when the math unit begins the next operation for the first processor. Alternately performing operations for the first and second processors allows the math unit to keep operating without pipeline delays even when programs executed by one or both of the processors require iterative operations.

Typically, the first processor (often referred to as the foreground processor) executes a program which generates a sound amplitude value. The second processor (often referred to as the background processor) preprocesses data from an external memory such as a DRAM, and stores the preprocessed data in a memory for use by the foreground processor. The external memory is often used for storing look-up table values.

In another embodiment, two processors control a multiplexer which supplies data to a math unit. Typically, the math unit is a multiplier or a combined multiply-and-accumulate circuit. Data is fed from the multiplexer through a set of flip-flops to the math unit. While the math unit is processing data for one of the two processors, the other processor selects the data supplied to input leads of the flip-flops. Math unit processing is not disturbed because signals from the flip-flops are not changed until the flip-flops' clock is triggered. Upon completion of processing by the math unit, new data from the multiplexer is loaded into the flip-flop set in response to a clock signal. The math unit begins operating for the second processor, and the first processor can change the data supplied through the multiplexer.

Typically, one or more memories provide input data to the multiplexer. If the access time of a memory is less that half the processing time of the math unit, the memory can be accessed more than once during each operation by the math unit, even when the memory has a single data port. For example, during two consecutive operations by the math unit, the first processor can access the memory at least once, the second processor can access the memory at least once, and an external device can access the memory at least once. The flip-flop set maintains correct data for the math unit while the memories are accessed. In addition to the memories, a hardware white noise generator can be

connected to the multiplexer to supply pseudorandom data.

A second set of flip-flops is often employed to store output data from the math unit. The second flip-flop set temporarily stores the output data from the math unit so that the output data can be moved or stored in a desired location while the math unit is processing new data. For example, the math unit output data in the second flip-flop set can be written to one or many memory locations that are accessible by an interface for a CODEC or a DAC. Data can also be routed back as input to the multiplexer, so that the output data is available for further manipulation by the math unit.

In still another embodiment of the invention, a digital signal processor includes a foreground processor and a background processor which perform different functions. The foreground processor executes a program to create a digital representation of a sound amplitude and is connected to a first memory which stores parameters used by the foreground processor. The background processor is operably connected to the first memory and to a second memory which stores data, particularly look-up table values. Typically, the second memory is implemented using DRAM and may be provided on one or more integrated circuit separate from the integrated circuit containing the foreground and background processors.

Look-up table values in the second memory can represent any function and are commonly used for delay lines, wave tables, and ADSR curves. The background processor preprocesses data from the second memory. Typical preprocessing performed by the background processor includes operations such as interpolating between look-up table values or changing an offset within a look-up table representing an ADSR curve or other function. Special incrementing algorithms can control the rate at which an ADSR curve or other look-up table is sampled.

For performing interpolation, the background processor typically includes a third memory for storing interpolation coefficients. The interpolation coefficient are mathematically derived constants which the background processor multiplies by look-up table values and then sums to derive an interpolated value. An exemplary derivation of interpolation coefficients for performing a cubic polynomial interpolation is disclosed below. The interpolation coefficients can be stored in ROM.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a digital signal processor having a foreground and a background processor in accordance with an embodiment of the present invention.

Fig. 2 is a block diagram of circuit blocks which control data flow through a math unit in a digital signal processor in accordance with the present invention.

Fig. 3 is a group of timing diagrams showing an example of how a port control interface, a foreground processor, and a background processor share control of a memory and a multiply-and-accumulate block in a digital signal processor in accordance with the present invention.

Fig. 4 is a memory map for an embodiment of the present invention.

Fig. 5 is a block diagram of a background processor and related elements in part of a digital signal processor in accordance with an embodiment of the present invention.

Fig. 6 is a flow diagram of an FIR filter performed as a task of a background processor in accordance with an embodiment of the present invention.

Fig. 7 shows plots of four types of ADSR curves.

Fig. 8 represents two pages of DRAM memory which store portions of the same look-up table and shows data

which is repeated in both pages of DRAM to speed up access to data during interpolation.

Fig. 9 contains timing diagrams which illustrate the execution of background tasks and foreground instructions in a digital signal processor in accordance with an embodiment of the present invention.

Similar or identical items in different figures have the same reference symbols.

DETAILED DESCRIPTION

Fig. 1 shows a block diagram of a digital signal processor (DSP) 100 in accordance with the present invention. DSP 100 includes parallel processors 101 and 102 which operate in a parallel interleave fashion and share a math unit 103. Processors 101 and 102 contain conventional processing circuitry such as decoders for decoding instructions and control circuits for generating control signal to implement instructions.

Processors 101 and 102 are sometimes referred to as foreground processor 101 and background processor 102 to indicate the different functions of processors 101 and 102. Background processor 102 preprocesses information such as look-up table data from a DRAM (not shown). Foreground processor 101 processes information from background processor 102 and parameter memories 107 and 108 and then writes values representing sound amplitudes to a parameter memory 109. The sound amplitude values are typically accessed by a CODEC (coder-decoder) or a DAC (digital-to-analog converter) through a CODEC interface 111. The CODEC or DAC converts the digital sound amplitude values into analog sound signals.

Typically, an external device such as a personal computer writes data and instructions to DSP 100 through a port control interface 106. Data to be processed by DSP 100 is written into parameter memories 107 and 108. Instructions for foreground processor 101 are written into an instruction memory 110. Parameter

memories 107 and 108 and instruction memory 110 are typically implemented as static random access memory (SRAM).

An external DRAM (not shown in Fig. 1) is controlled by background processor 102 through a DRAM interface 104. DRAM interface 104 typically uses a timing generator for refresh cycles and access to the DRAM. The timing generator may be an external oscillator or an on chip ring oscillator time generator such as described in U.S. patent application entitled "Timing Generator", attorney docket number M-2235-US, by Bryan J. Colvin and Masao Shindo which was co-filed with the present application and is incorporated by reference herein its entirety.

The external device also writes control values to control registers 105. The control values include configuration data such as used by CODEC interface 111. CODEC interface 111 is configurable to provide digital signals to one or more DAC (not shown) or both provide and accept signals from one or more different CODECs (not shown) at a programmable sampling rate. Such CODEC interfaces are known in the art and not further described here.

Control registers 105 also store mode values which indicate tasks for background processor 102. Together with parameter values in memories 107 and 108, the mode values act as a program for background processor 102 and determine how background processor 102 processes data. In one mode, background processor 102 uses DRAM interface 104 to read data points from a look-up table in the DRAM and then uses shared math unit 103 to calculate an interpolated value between the data points. The interpolated value is written into parameter memory 107 or 108 for later use by foreground processor 101. Other operating modes employed in a specific embodiment of the invention are disclosed below.

Foreground processor 101 operates according to a program stored in instruction memory 110. The program

generates sound amplitude values according to one or more sound synthesis models. Generated values are stored in parameter memory 109. Foreground processor 101 manipulates data from a number of sources including parameter memories 107 and 108 which may contain data that has been preprocessed by background processor 102, parameter memory 109 which typically contains the results of previous foreground processor operations, a white noise generator (not shown in Fig. 1), and other sources disclosed below.

Memory 109 may be an SRAM or a set of registers and typically includes (a) storage locations dedicated for output sound amplitude values and (b) general purpose storage locations used during calculation of sound amplitude values. Multiple output sound amplitude values can be generated for multiple DACs or CODECs. For example, two output sound amplitudes may be generated for producing stereo sound. Values stored in the locations dedicated to output are accessible through CODEC interface 111 at intervals determined by the programmable sampling rate. Sampling rate control values stored in control registers 105 determine the sampling rate and can be set through port control interface 106. A programmable clock circuit (not shown) provides a clock signal having a frequency determined by the sampling rate control values.

Fig. 2 shows a block diagram of a data selection circuit which includes parameter memories 107 to 109, multiplexers 209 to 211, and interconnecting circuit blocks for implementing data flow through shared math unit 103 in a DSP in accordance with the present invention. In this embodiment, shared math unit 103 includes a multiplier-accumulator (MAC) 215 which adds a digital value Z to the product of digital values X and Y.

Multiplexer 209 and a circuit block 213 provide value X. Multiplexer 209 selects an output value X' from a set including values MA and MB from parameter memories 107 and 108 respectively, values C_x and C_z from

memory 109, and a value BPX from the background processor. Block 213 transforms the value X′ according to a control signal provided by either the foreground or background processor. Examples of transforms that can be selected are no-change so that X equals X′, a 2′s complement so that X equals -X′, and a pan so that X equals (1-X′). The 2′s complement transformation causes MAC 215 to subtract the product of the values X′ and Y from the value Z. Other transformations can be implemented depending on the desired functions and instruction set of the DSP.

Multiplexer 210 provides the value Y which is selected from a set including the value MB from parameter memory 108, a pseudorandom value from a hardware white noise generator 205, a value C_y from memory 109, a value from a circuit block 206 (FORCE1), and a value from a circuit block 207 (ADSRB). Circuit block 206 provides a value representing the number one. When value Y represents the number one, MAC 215 performs a sum of value X plus value Z. Circuit block 207 provides an incremental increase or a percentage decrease for an ADSR curve. The function of circuit block 207 is disclosed more fully below in regard to the foreground processor instruction set in Appendix A.

White noise generator 205 is a hardware random number generator which provides a pseudorandom series of digital values. Software random number generators are commonly employed in wave guide synthesis models of musical instruments. A hardware white noise generator facilitates implementation of synthesis models by permitting a single program instruction which generates, multiplies, and accumulates a random number. Any known or yet to be developed random number generator may be used as white noise generator 205.

Multiplexer 211 provides the value Z which MAC 215 adds to the product of the values X and Y. Value Z is selected from a value BPZ from the background processor, value C_z from memory 109, values MA and MB from parameter memories 107 and 108 respectively, and a

value representing zero from circuit block 208 (FORCE0). By selecting the value representing zero, the output value from MAC 215 is the product of values X and Y.

Memory 109 may simultaneously provide three values C_x, C_y, and C_z and therefore has multiple data ports. Parameter memories 107 and 108 each provide at most a single value and are single data port memories. Single data port memories require less space in an integrated circuit, are less expensive than multi-port memories, and therefore reduce the cost of the DSP in accordance with the present invention when compared to a DSP with multi-port memories.

Typically, a binary representation of value Z contains more bits than binary representations of values X and Y because value Z is used for accumulations where overflows and round-off error are critical. Circuit blocks 212 convert values from memories 107 and 108 to the proper binary representation for value Z. For fixed point representations, conversion is typically performed by a logical shift and a sign extension. Similar conversion of digital representation is provided by circuit blocks 204 and 218 which reduce the number of bits used in parameter memory 109 to the number of bits appropriate for values X or Y. Blocks 218 perform a truncation of towards zero for both positive and negative values from memory 109. Block 204, in addition to changing the representation of value C_Z, may change the magnitude of value C_Z during generation of an ADSR curve. During an attack phase, block 204 provides a value representing 1 or 0.5 depending on whether a faster or a slower attack is desired. During the decay and release phases, block 204 checks whether value C_Z will cause a discernable decrease in an ADSR curve value and if the value C_Z will not, block 204 provides a value which will cause a minimal decrease in the ADSR curve value.

The foreground processor or background processor, depending on which of the processors is using MAC 215,

generates select signals for multiplexers 209-211. The foreground processor decodes instructions and provides select signals to multiplexers 209-211 and address signals to memories 107-109 typically while MAC 215 performs a multiply-and-accumulate cycle for the background processor. The background processor decodes a mode value and parameters from parameter memories 107 and 108 to generate appropriate values BPX, BPY, and BPZ and select signals for multiplexers 209 to 211 while MAC 215 performs a multiply-and-accumulate cycle for the foreground processor. The timing of the select and address signals are disclosed below.

A set of flip-flops 214 stores one set of values X, Y, and Z while a next set of values X, Y, and Z selected by the foreground or background processor propagates through the data selection circuit including multiplexers 209 to 211. New values X, Y, and Z are stored in flip-flop set 214 and provided to MAC 215 when a timing signal MAC_CLK is asserted. As is well known, signal MAC_CLK is asserted "high" or asserted "low" depending whether flip-flop set 214 is triggered on the leading or trailing edge of signal MAC_CLK. A flip-flop set 216 is also clocked by signal MAC_CLK and stores an output value (Z+X*Y) from MAC 215 when MAC 215 begins processing the next values X, Y, and Z. The numbers of flip-flops in flip-flop sets 214 and 216 depend on the numbers of bits used to represent values X, Y, and Z. In one embodiment, each multiplicand value X or Y contains sixteen bits, and the addend value Z contains thirty-two bits. In such an embodiment, flip-flop set 214 stores 64 bits of information, and flip-flop set 216 stores a 33-bit value (the sum of two 32-bit numbers).

Circuit block 217 (FPDP) tests the output value stored by flip-flop set 216 and sets flags to indicate, for example, that the output value is zero, negative, overflows, or underflows the capacity of the storage location in memory 109 where the output value is to be stored. In the case of an overflow or underflow,

circuit block 217 provides either a saturated value (the most positive or the most negative value that can be represented in the target register) or truncates the most significant bit of the output value from MAC 215. Whether the value provided by block 217 is saturated or truncated depends on a configuration control value SATF. The value from block 217 is directed to a location in memory 109 indicated by an address signal from the foreground processor or is directed to the background processor on bus B_WRITE_DATA. In addition to the output value from flip-flop set 216, values DA, DB, MA, and MB are provided to circuit block 217 and are written to memory 109 during move instructions disclosed below.

Multiplexer 203 implements move instructions which move values into memories 107 and 108. Values C_x, C_y, and C_z from memory 109, a value BP from the background processor, or a value MW from the port control interface, can be routed through multiplexer 203 into one or both of memories 107 and 108. Flip-flop sets 201 and 202 provide delayed values DA and DB from memories 107 and 108. The delayed values DA and DB change every time signal MAC_CLK is asserted and indicate the previous values MA and MB provided by parameter memories 107 and 108 respectively. Delayed values DA and DB can be moved back into memories 107 and 108 through multiplexer 203 or into memory 109 through circuit block 217.

In Fig. 2, memories 107 to 109 are sufficiently fast to provide valid data signals in less than about half the time required for MAC 215 to execute a multiply-and-accumulate cycle, and parameter memories 107 to 109 can be accessed twice during each multiply-and-accumulate cycle of MAC 215. In accordance with the present invention, control of parameter memories 107 and 108 and control of MAC 215 are time division multiplexed. Time division multiplexing of memories 107 and 108 permits use of less expensive single port memories even though memories 107 and 108 are accessed

by the foreground processor, the background processor, and the port control interface.

Fig. 3 shows timing diagrams indicating control of parameter memories 107 and 108 and control of MAC 215 for typical instructions executed by the foreground processor. In this embodiment, the foreground processor has sole control of memory 109, except at the end of sampling periods, when the CODEC interface may have access. In Fig. 3, FP indicates the foreground processor has control, BP indicates the background processor has control, and PC indicates that the port control interface has control. Timing of memory control can be varied according to particular instructions executed by the foreground processor. Example instructions and timing variations are disclosed below.

Each foreground instruction is completed in one instruction cycle time such as instruction cycle 310. The instruction cycle is twice as long as a multiply-and-accumulate cycle (MAC cycle) for MAC 215 and four times as long as the memory access time for memories 107 to 109. Instruction cycle 310 includes four memory access periods 330, 332, 334, and 336.

Considering Figs. 2 and 3 together, during memory access period 330, the foreground processor controls parameter memories 107 and 108, and MAC 215 processes data previously selected by the background processor. Depending on the instruction being executed, the foreground processor generates address signals for parameter memories 107 to 109 and/or select signals for multiplexers 209 to 211. Parameter memories 107 to 109 and/or circuit blocks 205 to 208 provide valid data values which propagate through multiplexers 209 to 211 to input leads of flip-flop set 214. MAC cycle 322 is not disturbed because flip-flop set 214 maintains previous values X, Y, and Z on the input leads of MAC 215.

At the time T1, signal MAC_CLK is asserted and flip-flop sets 201, 202, 214, and 216 are triggered.

Flip flop set 214 stores new values X, Y, and Z selected by the foreground processor and asserts the new values X, Y, and Z to MAC 215. MAC 215 begins MAC cycle 320 for the foreground processor. Also at time T1, flip-flop set 216 stores the output value from previous MAC cycle 322 of MAC 215. The output value from flip-flop set 216 is directed by circuit block 217 to the background processor via bus B_WRITE_DATA.

For some instructions such as move instructions, flip-flops sets 201 and 202 store the selected values DA and DB at time T1. Values DA and DB can be written into any of memories 107, 108, or 109 during memory access period 334, the next time that the foreground processor has control of memories 107 and 108.

During access period 332, the background processor controls memories 107 and 108. The action of the background processor depends on the background processor task and on the processing sequence for the task. The sequence for processing of each task typically requires more than one instruction cycle. In the initial step of most tasks, the background processor generates read addresses for memories 107 and 108 and reads parameter values from memories 107 and 108. The parameter values read from memories 107 and 108 and a mode value from the control registers determine the task and subsequent processing sequence of the background processor. In the last step of a typical processing sequence, the background processor generates write addresses for memories 107 and/or 108 and writes values to memories 107 and/or 108. Execution of background tasks can be pipelined so that two or more background tasks are executed at once.

During MAC cycle 320, the background processor controls multiplexers 209-211 and therefore controls data flow to flip-flop set 214 even though the foreground processor controls memories 107-109 during access period 334. The background processor directly provides data signals BPX, BPY, and BPZ so that new data is ready for MAC 215 at time T3. Accordingly, MAC

215 operates without pipeline delay. The operation of the background processor is disclosed in more detail below.

During memory access period 334, the foreground processor controls memories 107-109. For instructions that move data into parameter memory 107 or 108 from memory 109 or from flip-flop sets 201 or 202, the foreground processor generates write addresses for memories 107 and 108 and select signals for multiplexer 203 so that selected values are written.

At time T3, flip-flop sets 214 and 216 are triggered again. Flip-flop set 216 captures the output value from MAC 215. The output value from flip-flop set 216 can be stored at a write address provided to memory 109 by the foreground processor during the memory access period 336 because the foreground processor does not share access of memory 109 with the background processor or the port control interface. The port control interface controls memories 107 and 108 during memory access period 336, and provides select signals to multiplexer 203 for writing data MW to memories 107 and 108.

A new instruction cycle 315 begins at time T4 and proceeds in a manner similar to instruction cycle 310.

Fig. 4 shows a memory map of parameter memories 107 to 109 and instruction memory 110 of Fig. 1 in an exemplary embodiment in accordance with the invention. Parameter memories 107 and 108 and instruction memory 110 each contain 512 16-bit words of data or instructions and are partitioned into thirty-two slots of sixteen words as shown by memory map 400. Memory maps 407, 408, and 410 show a single slot from parameter memory 107, parameter memory 108, and instruction memory 110 respectively. Each slot in instruction memory 110 corresponds to a slot in parameter memory 107 and a slot in parameter memory 108. Typically, each collection of corresponding slots operates together as program and parameters for a

distinct voice or emulation, but a number of slots may be combined to represent a more complex emulation.

Control registers 105 in Fig. 1 contain 32 sets of mode values which correspond to the 32 slots in memories 107, 108, and 110. As disclosed below, the mode values are combined with parameters from corresponding slots in memories 107 and 108 to determine which tasks are executed for that slot by background processor 102.

Memory 109 contains sixteen locations C0-C15 which are global to all slots. Eight locations C0-C7 are 24-bit locations and accessible through CODEC interface 111. The 24-bit locations are paired into four sets for four stereo CODEC channels. Each set contains left and right sound amplitude values. The remaining eight locations C8-C15 are 32-bit general purpose storage that may be used for intermediate calculations, accumulation, and passing values from one slot to another.

In accordance with the exemplary embodiment of the invention, memories 107 and 108 store 16-bit fixed point representations of values between -2 and just less than 2. The 16-bit representations have a sign bit, an integer bit, and 14 bits representing a fractional part. Memory 109 contains fixed point binary representations of numbers between -16 and just less than 16. The fixed point binary representations contain a sign bit, a 4-bit integer part, and either a 19-bit or a 27-bit fractional part depending on whether the storage location is twenty-four or thirty-two bits. The sound amplitudes provided through CODEC interface 111 are restricted to values between -2 and 2.

A complete instruction set for foreground processor 101 of the exemplary embodiment as shown in Fig. 1 is provided in the Appendix A. Each 16-bit foreground processor instruction contains up to three operands i, j, and k. The operands are indices which identify data in parameter memories 107 to 109. In Appendix A, the instruction words are to the left of

one or more equations which describe the operation performed by execution of the instruction. In Appendix A, subscripted quantities A, B, and C are data values from parameter memories 107, 108, and 109 respectively.

Operands which identify a data word from parameter memories 107 and 108 contain up to four bits and are offsets relative to a slot pointer which identifies the slot that foreground 101 processor is executing. Foreground processor 101 updates the slot pointer as slots are completed. (Background processor 102 keeps a separate slot pointer so that foreground and background processor 101 and 102 can execute different slots.) Data in memory 109 is also identified with 4-bit operands, but the operands are global, i.e. independent of the slot pointer.

There are twelve foreground processor instructions that include three operands. The three operand instructions are executed in a single MAC cycle using the timing described above in reference to Fig. 3. For example, during memory access period 330, the foreground processor provides addresses and select signals to the memories 107-109 and multiplexers 209-211 according to the values on the right side of the equation describing the instruction. The result of MAC cycle 320 is written into memory 109 between times T3 and T4. For the instruction 1001-i-j-k, a single value $C_i+A_j*B_k$ is calculated and written to two addresses C_i and C_{i+1}. Values C_i, A_j, and B_k are read during the first memory access period of the foreground processor, for example during period 330 in Fig. 3 and new values C_i and C_{i+1} are written after MAC has completes a multiply-and-accumulate cycle, for example during period 336 in Fig. 3.

Group A of the instructions in Appendix A includes two operand instructions. Group A instructions are executed in a single MAC cycle with the above disclosed timing. In some of the instruction, a value WN from white noise generator 205 is multiplied by a value from parameter memory 107 or 108. In group A, instructions

1011-0000-j-0i through 1011-0010-j-1i are for multiplications of 32-bit multiplicands. One 32-bit multiplicand is a value C. The other 32-bit multiplicand has sixteen most significant bits in an A value and sixteen least significant bits in a value B. (For these instruction, operand i is a 3-bit value.) Since MAC 215 in the exemplary embodiment multiplies two 16-bit values, multiplication of 32-bit values requires more that one MAC cycle. Values CH and CL indicate the sixteen most significant and sixteen least significant bits respectively in the value C. Value NF is a sign for signed multiplication. The function "abs(...)" is the absolute value.

Group B of the instructions in Appendix A includes two operand instructions for addition and subtraction. Group B instructions are executed in a single MAC cycle using the timing described above. The foreground processor causes block 206 (FORCE1) to provide a one as value Y so that MAC 215 performs an addition or substraction. For subtractions, foreground processor causes circuit block 213 (NEG. PAN) to perform a 2's complement.

Group C of the instructions in Appendix A are two operand instructions that are useful in constructing filters. The instruction combines a multiply and a move. During the foreground processor's first memory access period of an instruction cycle, for example during memory access period 330 in Fig. 3, the foreground processor generates address signals for parameter memories 107 to 109 according to the first equation describing the instructions in Appendix A. At time T1, flip-flop set 214 latches input values for MAC 215. Flip-flop sets 201 and 202 also latch the values from memories 107 and 108.

If the instruction moves a value from memory 109 or from one of flip-flop sets 201 and 202, then during the second memory access period of the foreground processor, time interval 334 in Fig. 3, the foreground processor generates appropriate select and address

signals for writing the moved value into the appropriate memory 107 to 109. During the time interval 336, the foreground processor generates appropriate select and address signals for circuit block 217 and memory 109, so that the result of the MAC cycle is written into memory 109.

Special timing occurs for instructions such as 1101-0000-j-i which multiply and move data between memory 107 and memory 108 when the moved data is not supplied to MAC 215. In this special case, one of the memories 107 or 108 is the source of the value moved, and the other of the memories 108 or 107 is the destination of the value moved. During memory access period 330, the foreground processor generates address signals for input values to MAC 215 according to the right side of the first equation for the instruction. The value to be moved between memory 107 and memory 108 is not read during period 330 and not stored in flip-flop set 201 or 202. During memory access period 334, the foreground processor grants control of the destination memory 107 or 108 to the port control interface, and generates address signals for reading the value to be moved from the source memory 108 or 107. The read value is stored into one of flip-flop sets 201 or 202 at time T3. During memory access period 336, the foreground processor takes control of the destination memory 107 or 108 from the port control interface and generates address and select signals for writing the value to the destination memory 107 or 108, but the port control interface keeps control of the source memory 108 or 107. Accordingly, for this special case the foreground processor and the port control interface swap memory accesses to one of the memories 107 or 108, but both the foreground processor and the port control interface retain the usual number of accesses to memories 107 and 108 for the instruction cycle.

Group D of the instructions in Appendix A contains two operand move instructions. The instructions are

completed in a single instruction cycle. Up to four moves can be executed in a single instruction cycle, one move into each memory 107 and 108, during each access period in which the foreground processor has control of memories 107 and 108. 32-bit values C can be moved into or from two 16-bit values A and B, where value A contains the 16 most significant bits and value B contains the 16 least significant bit. Such combined values A and B are indicated by AB in Appendix A. Move instructions 1110-0011-j-i through 1110-1010-j-i move and shift a value C. The shift can be performed by providing shift capabilities in multiplexer 203 of Fig. 2.

Group E of the instructions in Appendix A contains two parameter instructions which provide special functions for ADSR curves and decay curves. The functions performed by the instructions depend on control values in control register 105 which indicate states "key on", "key off", "alt on", and "alt off" for each slot. Functions ADRS0_K, ADRS1_K, and ADRS2_K respond to key on and key off states. Functions ADRS0_A, ADRS1_A, and ADRS2_A respond to alt on and alt off states.

For each function ADRS0_K, ADRS1_K, or ADRS2_K, when key on is initially indicated, bits 7 and 15 of a value A which an argument of the function are zero indicating an attack state. Functions ADSR0_K, ADRS1_K, and ADRS2_K generate values appropriate for an attack phase of an ADSR curve by incrementing a value C by a fixed increase. The fixed increase is given by bits 0-6 of the value A times 1/8 for ADRS0_K and times 4 for ADRS1_K and ADRS2_K. Each time the instruction is repeated incrementing continues until the value C reaches 1, and then bit 7 of the value A is set indicating a decay phase of the ADSR curve. In the decay phase, each time the instruction implementing one of the functions ADSR0_K, ADRS1_K, or ADRS2_K is executed the value C is decrease by a fraction of the value C where the fraction is indicated by bits 8-15 of

the value B. For ADSR0_K and ADSR1_K, bits 8-15 of the value B are shifted to the right three bits (divided by eight) to determine the fraction. The decay continues until the value C reaches a level indicated by bits 14-12 of the value A then bit 15 of the value A is set to indicate a sustain phase of the ADSR curve. In the sustain phase of the ADSR curve, the instructions leave the value C is unchanged until a key off state arises and then bit 7 of the value A is cleared indicating a release phase of the ADSR curve. During the release phase, the value C is decreased by a fraction of the value C where the fraction is indicated by bits 7-0 of the value B.

Functions ADSR0_A, ADRS1_A, and ADRS2_A perform in the same manner as functions ADSR0_K, ADRS1_K, and ADRS2_K except that functions ADSR0_A, ADRS1_A, and ADRS2_A respond to the conditions alt on and alt off.

The functions DECAY_A and DECAY_K implement long exponential decay function and are used with long table modes. The instruction does nothing unless the background processor mode which uses the value B as a parameter is a long table mode, and the background processor is accessing data in the last page of the long table. (Long table background mode is disclosed below.) On the last page, the instructions decrease a value C by a fraction indicated by the value B if the appropriate key off or alt off condition is set.

Group F of the instructions in Appendix A contains one operand and no operand instructions used mostly for conditional branching and manipulation of status flags. The instructions only permit forward branching so that no program loops are possible. Without loops, the maximum time required to execute all of the instructions in all of the slots can be limited to less than the sampling period.

Group G of the instructions in Appendix A contains one operand special move instructions which are useful for controlling background tasks that are described in detail below. Functions MOVH8 and MOVH10 move the most

significant 8 and 10 bit respectively from a value C to a value A.

Functions MOVADL and MOVADH are typically used with background tasks which use a page size selector. Some background tasks use a value A as a page size selector which selects the location of a physical page in a DRAM and the size of a logical page for storing a look-up table or delay line. These background tasks use a value B as a pointer within the physical page. Physical memory pages in the DRAM may for example contain 512 words. When the logical page is smaller than the physical page, more than one look-up table or delay line can be store in a single physical page of DRAM. For example, if value A selects a logical page size of 64 words, then value B may point to any of eight 64-word logical pages in a 512-word physical page. The most significant bits of the value B indicate a logical page, and less significant bits indicate a location within the logical page. MOVADL and MOVADH move bits from a value C to selected bits of a value B, either to the bits which indicate a location within a logical page or the bits which indicate the logical page. A value A (the page size selector) determines the bits of the value B changed. MOVADH instructions change just the bits indicating the logical page and are useful for creating a family of curves which can be selected algorithmically. MOVADL instructions preserve the bits which indicate the logical page and change the bits which indicate a location with in the logical page.

Groups H and I of the instructions in Appendix A contain instructions for special functions. Functions UNPACKS and UNPACKU move four nibbles from a signed or unsigned 32-bit value C into two value A and two values B. Functions PACKU and PACKS moves nibbles from two values A and two values B into one 32-bit value C. Functions RSPP, SPPA, SPPR, and SPPL are for jumping between instruction slots according to addresses given by a value C. Functions FAR A and FAR B are for

accessing parameter of other slots. Functions FKF, FKO, FAF, and FAO set states "key off", "key on", "alt off" and "alt on" respectively. Functions I and FAR I are for accessing instructions in the current and in another slot. ZF indicates the zero flag. MW indicates a background processor mode word.

Fig. 5 is a block diagram showing greater detail of background processor 102. The tasks executed by background processor 102 are determined by mode and control values stored in control registers 105 and by parameters stored in memories 107 and 108. Typically, the mode values and control values are set by an external device such as a personal computer connected to the DSP through port control interface 106. Parameter values in memories 107 and 108 are set or changed by the external device, foreground processor 101, or background processor 102.

One set of control values in control registers 105 indicates which slots are enabled and which slots are disabled. Foreground processor 101 executes the instructions for each enabled slot such as instructions 410 in Fig. 4, and background processor 102 executes background tasks for each enabled slot as indicated by mode values in control registers 105 and parameter values in memories 107 and 108. The number of slots enabled should not exceed the maximum number of slots that can be executed within a single sampling period. Foreground processor 101 must be fast enough to execute all of the instructions in all enabled slots before an amplitude value is required by the CODEC interface. Overhead such as refresh cycles for attached DRAM must also be handled during the sampling period. For a DSP with 35.4 nS MAC cycle time, memory access periods less than about 17.7 nS for memories 107 to 109, and a 283 nS DRAM refresh cycle, 32 slots can be enabled for sampling frequency less than 27.19 KHz. At a 44.1 KHz sampling rate, up to nineteen slots may be enabled.

For disabled slots, background processor 102 either skips the slot and begins execution of the next

slot or executes a block transfer from parameter memories 107 and 108 to DRAM 502. Block transfers permit an external device connected to port control interface 106 to access DRAM 502. To transfer data to DRAM 502, the external device disables a slot and during memory access periods of the port control interface 106, writes a DRAM address and data to locations in memories 107 and 108 corresponding to the disabled slot. The external device then sets a flag in control registers 105 to indicate that the disabled slot contains data to be written to DRAM 502. When background processor 102 reaches the disabled slot, background processor 102 transfers the data from memories 107 and 108 to DRAM 502, and foreground processor 101 executes no operations. For the external device to read from DRAM 502, the external device writes a DRAM address to memory 107 or 108 and sets a second flag for the disabled slot. Background processor 102 transfers data from DRAM 502 to memory 107 and 108 where port control interface 106 can read the data in subsequent memory access periods.

Typically, DRAM 502 is provided on one or more separate integrated circuits while the remaining circuits shown in Fig. 5 are situated together on a single monolithic integrated circuit. DRAM 502 may include multiple banks of paged memory for data such as look-up tables, ADSR curves, delay lines, and any desired wave tables. ADSR curves, delay lines, and wave tables are generically referred to herein as look-up tables.

Look-up tables in DRAM 502 provide output values that represent values y_n of a function Y for a range of discrete values n. The value n is indicated by the address provided to DRAM 502. In some emulations, function values Y(n+x) are needed for fractional values x, and interpolation between look-up table values y_n and y_{n+1} is required. The background processor may perform interpolations. One interpolation technique approximates the function Y as a polynomial for a range

having values between n and n+1. When the values n and n+1 are inserted into the polynomial, the polynomial yields values y_n and y_{n+1} respectively. For example, a cubic polynomial approximates the function Y(n+x) as

$$Y(n+x) = K_3x^3 + K_2x^2 + K_1x + K_0 \quad \text{(eq. 1)}$$

Using linear algebra, it can be shown that eq. 1 yields values y_{n-1}, y_n, y_{n+1}, and y_{n+2} for x equal -1, 0, 1, and 2 respectively if the values K_3, K_2, K_1, and K_0 are as shown in eqs. 2-5.

$$K_3 = (y_{n+2} - 3y_{n+1} + 3y_n - y_{n-1})/6 \quad \text{(eq. 2)}$$

$$K_2 = (3y_{n+1} - 6y_n + 3y_{n-1})/6 \quad \text{(eq. 3)}$$

$$K_1 = (-y_{n+2} + 6y_{n+1} - 3y_n - 2y_{n-1})/6 \quad \text{(eq. 4)}$$

$$K_0 = y_n \quad \text{(eq. 5)}$$

Combining eqs. 2-5 with eq. 1 yields

$$Y(n+x) = y_{n-1}\cdot(-x^3+3x-2)/6 + y_n\cdot(3x^3-6x^2-3x+6)/6 + y_{n+1}\cdot(-3x^3+3x^2+6x)/6 + y_{n+2}\cdot(x^3-x)/6 \quad \text{(eq. 6)}$$

Y(n+x) from eq. 6 is a sum of four terms, each term being the product of one look-up table values y_{n-1}, y_n, y_{n+1}, or y_{n+2} and a coefficient that depends on the fraction x.

Background processor 102 of Fig. 5 can interpolate by approximating look-up table functions with cubic polynomials. The coefficients from eq. 6 for desired values of fraction x are stored in a table in an interpolation ROM 501. In one embodiment of the invention, coefficient values are stored in interpolation ROM 501 for fractions x from 1/128 to 127/128 in steps of 1/128. Appendix B shows a table of coefficient values referred to as "Lagrange Data" for a step size of 1/128. Interpolated values y(n+x) between any two consecutive look-up table values y_n and y_{n+1} are determine in four multiply-and-accumulate cycles using eq. 6.

Once the background processor determines that interpolation is required, address generator 507 generates a DRAM read address for the point y_{n-1}. The address generated depends on the mode values in control register 105 and on parameter values in memories 107

and 108 as disclosed in more detail below. Address generator 507 supplies an address for the first of four consecutive location in DRAM 502 to be written into FIFO buffer 503. The address generator 507 also provides an address in interpolation ROM 501 for the interpolation coefficients corresponding to the fractional value x. In four multiply-and-accumulate operation of shared MAC 215, values BPX, BPY, and BPZ are provided from interpolation ROM 501, FIFO 503, and an accumulator 509 respectively. The output value from MAC 215 is stored into accumulator 509 until the last multiply-and-accumulate cycle provides the desired interpolated value. The interpolated value is stored in memory 107 or 108 for later use by foreground processor 101.

A problem with the interpolation disclosed above arises when interpolation is used to provide values in a wave table representing a higher frequency sound. In such situations, interpolating a series of values from a wave table is equivalent to a filter operation. The Lagrange Data coefficients provide very smooth interpolation of points but in a filter operation, tend to introduce higher frequency components in the results which cause aliasing. A second table coefficient values entitled "Minimum Sidelobe Data" shown in Appendix B performs a filter operation which reduces the aliasing. The Minimum Sidelobe Data coefficient values may be used in place of the Lagrange Data coefficients where aliasing may be a problem.

In accordance with the exemplary embodiment of the invention, parameter memories 107 and 108 and instruction memory 110 are divided into slots as shown in Fig. 4. Each slot in memory 107 contains eight words A_0-A_8 used by background processor 102, and each slot in memory 108 contains eight words B_0-B_8 used by background processor 102. Control registers 105 contain one 16-bit mode value for each slot. Background processor 102 executes four background tasks per enabled slot. Each background task is determined

by a nibble from the mode value and two parameters from each of the corresponding parameter slots. If the nibbles in the mode value are indexed by an integer n between 0 and 3, a task decoder 506 reads nibble n and parameters A_{2n}, A_{2n+1}, B_{2n}, and B_{2n+1} to determine a task to be executed. Once the task is determined, a state controller 504 and pipeline timing circuit 505 controls the sequence of operations for completion of the task. The operations for the tasks are pipelined as described below in regard to Fig. 9.

The exemplary embodiment of the invention has sixteen background tasks indicated by mode nibbles. Nibble 0000 indicates no background task. The first mode (0001) is referred to as delay line mode and is primarily used for wave guide synthesis. Delay line mode provides an interpolated value from a delay line in DRAM 502 and optionally writes a value to the delay line.

Background processor 102 maintains an 18-bit master write pointer used when determining where data is read or written in DRAM 502. The nine most significant bits of the master write pointer are referred to as the absolute pointer and the nine least significant bits of the master write pointer are referred to as the write index pointer. The write index pointer is decremented after each sample period and used for determining addresses that should change every sampling period. The absolute pointer is set every time a background task is executed in absolute mode and is used to define a base for a series of background tasks.

For the delay line mode, parameter A_{2n} indicates the write address in DRAM 502 if data is written. Bit 15 of A_{2n} indicates whether DRAM access is in absolute or relative mode. Bit 14 of A_{2n} is set if data is to be written. Bits 11-13 of A_{2n} indicate a logic page size which determines the amount of memory used to generate the delay line. Page sizes range from 512 words to four words in powers of two. Bits 9 and 10 of A_{2n}

indicate which of four DRAM circuit contains the desired address. In relative mode, an address indicated by the bits 0-8 of A_{2n} is added to the write index pointer, the sum is wrapped around a logical page boundary to provide the nine least significant bits of the write address. The absolute pointer provides the nine most significant bits of the write address. In absolute mode, the absolute pointer is set to the value given by bits 0-8 of A_{2n}, and the write address has nine most significant bits given by bits 0-8 of A_{2n} and nine least significant bits given by the write index pointer wrapped around a page boundary.

Parameter A_{2n+1} indicates the value to write. Parameter B_{2n} indicates the delay line length (an offset relative to the write address for reading from the delay line). Offsets beyond a logical page boundary wrap around. The delay line length includes a fractional part for interpolation. At the end of the delay line mode background task, an interpolated result is written to the parameter B_{2n+1}.

FIR filter mode is mode nibble 0010 binary and implements a two tap finite impulse response (FIR) filter illustrated in Fig. 6. The FIR filter performs the sum of three products $B_{2n+1}*A_{2n+1}$ + DRAM0*A_{2n} + DRAM1*B_{2n} and writes the result into parameter B_{2n+1}. The values DRAM0 and DRAM1 are read from a page of DRAM at an address given by a control value in control registers 105. The page is divided in sections. Each section correspond to slot and contains values DRAM0 and DRAM1 for FIR filter mode tasks.

Read only look-up table mode has mode value nibble 0011 binary and interpolates a value from a look-up table contained in one or more pages of memory. Parameter A_{2n} indicates the starting page and the number of pages in the look-up table. Parameter B_{2n} indicates an offset relative to the start of the look-up table, including a fractional part for interpolation if the look-up table is one page or less. One bit of

parameter A_{2n} may be used to indicate ir wrap around or truncation occurs when the offset provides an address past a boundary of the look-up table. The result (interpolated or otherwise) is written into B_{2n+1} upon completion of the task.

Mode nibbles 0100 through 0111 indicate modes for generating four types of ADSR curves. Plots of the four ADSR curves as a function of time are shown in Fig. 7. Each type of ADSR curve has a look-up table stored in DRAM and a sampling rate that is given by a variable increment that is added to an offset every sampling period. Mode 0100 is a single shot ADSR curve which when initiated by a key-on state 710, runs at a constant sampling rate through the ADSR look-up table once and thereafter returns a value zero. Mode 0101 is a drum roll type ADSR curve which starting with a key-on state 720 repeatedly runs at constant sampling rate through the ADSR look-up table until a key-off state 722 occurs. Mode 0110 is a piano ADSR curve which starts with a key-on state 730 and fast sampling rate but switches to a slow sampling rate after a fixed time 722. The fast sampling rate permits more points in the look-up during the critical attack portion of a note and relatively fewer points thereafter. When a key-off state occurs, the sampling rate of the piano ADSR curve returns to the fast sampling rate and runs through the remaining points of the look-up table as shown by a faster decay curve 734. If the key-off state does not occur, the slow sampling rate is maintained and a slower decay curve 736 is provided. Faster and slower decay curves 734 and 736 use the same look-up table but sample through the values at different rates. Mode 0111 is an organ ADSR which starts with a key-on condition 740 and after a fixed time 742, stays at a fixed point in the look-up table, until a key off state 744 occurs then sampling continues.

For each of the modes 0100-0111, parameter A_{2n} indicates a speed shift factor, a starting physical page, and a logical page size for the ADSR look-up

table. Parameter B_{2n} indicates an offset within the ADSR table including a fractional part for interpolation. Parameter A_{2n+1} indicates a time step constant and the least significant bits (LSBs) of the offset in the ADSR look-up table. The LSBs from the parameter A_{2n+1} gives offset resolution finer than the interpolation capabilities of the background processor. The added accuracy in offset value may be necessary to avoid round-off or truncation error during changes of the offset. Each sampling period, the background processor increments the offset by an amount given by the time step constant shifted by the speed shift factor. For long duration notes, the increment can be small, perhaps only a change only in the LSBs provided by A_{2n+1}. An interpolated result from the ADSR table is written into parameter B_{2n+1}.

Mode nibble 1000 binary indicates Read-Only Wave Table mode. In this mode, parameter A_{2n} indicates an absolute address and size of a wave table and indicates how often the offset within the wave table should be changed, for example once every sampling period or once every two sampling period. Parameter B_{2n} indicates an offset including a fractional part for interpolation. The offset wraps around if the offset is past an end of the wave table. Parameter A_{2n+1} is a signed step rate which is added to the offset when the background processor changes the offset. An interpolated result is written to parameter B_{2n+1}.

Mode 1001 binary is long table mode which is used when a look-up table is contained in more than one page of DRAM as shown in Fig. 8. Parameter A_{2n} indicates a starting page number and the number of pages containing the look-up table. Parameter A_{2n} also indicates whether offset should wrap around from one end of the look-up table to the other or be truncated. Parameter B_{2n} indicates a current page index and offset within the page. Parameter A_{2n+1} indicates a step rate for changing

the offset and an interpolation fraction for interpolating between values in the look-up table.

As described above, four values y_{n-1}, y_n, y_{n+1}, and y_{n+2} are read from a look-up table for interpolation. Because access to four data values in two or more physical pages takes more time than access to four value in a single page, the data values in the last four locations of each page are repeated in the first four locations of the next page. For example, memory locations 820 in DRAM Page 1 contain the same data as locations 810 in DRAM Page 0. The very last page has final values copied to the beginning of the first page in look-up table when wrap around enabled.

Modes 1010 and 1011 are long table modes with 2-to-1 and 4-to-1 compression, and operate in substantially the same manner as long table mode 1001. The primary difference of the three modes is long table mode operates on a look-up table containing 16-bit word values, long table modes with 2-to-1 compression operates on a look-up table containing byte values, and long table modes with 4-to-1 compression operates on a look-up table containing nibble values.

Mode 1100 is a long read-write delay line mode that both reads and writes to a delay line that extends over several pages of DRAM. Parameter A_{2n} indicates a starting page and the number of pages containing the delay line. Parameter B_{2n} indicates the delay line length as a page index and an offset within the page. No fractional offset is provided and therefore no interpolation is done in this mode. For large look-up tables, the accuracy provided by a large number of data points makes interpolation less important. Parameter A_{2n+1} stores data to be written at an address indicated by a write index pointer maintained by background processor 102. The value read from the delay line is written to parameter B_{2n+1}.

Mode 1101 is long read only mode and is the same as mode 1100 except that no data is written.

Mode 1110 is either a sample record mode or a line input mode. Bits in parameter A_{2n} distinguish the sample record mode from the line input mode. Sample record mode records or stores into DRAM sound amplitude values generated by the DSP. Parameter A_{2n} contains a starting address and a data size for writing of data into DRAM. Data may be written in word, byte, or nibble sizes. Parameter B_{2n} contains a sample counter which indicates a current page index and an offset in the current page for writing data. The offset is incremented by the background processor as data is written. Parameter A_{2n+1} contains the data to be written. Parameter B_{2n+1} contains an index for writing byte or nibble values and flags for stopping and starting recording.

Line input mode moves data from the CODEC interface into parameter memory. Up to four stereo CODECs or analog-to-digital converters (ADCs) can be connected to the CODEC interface. For recording of a sound, CODECs or ADCs write sound amplitude values to registers in the CODEC interface. Four pairs of registers are provided in the CODEC interface to store four pairs of values from the CODECs or ADCs, each pair of values being a left value and a right value as are common for stereo sound. Typically, the sound amplitude values are changed once every sampling period. In line input mode, parameter A_{2n} contains a flag which indicates line input mode rather than sample record mode and contains a code which indicates which of the four pairs values are transferred. The background processor transfers the left value of the pair indicated to parameter A_{2n+1} and the right value of the pair indicated to parameter B_{2n}. Parameter B_{2n+1} is not used in this mode.

Mixer mode (1111 binary) performs two multiplications. Data is not read or written to DRAM, so that during this background task refresh cycles for the DRAM can be executed. Parameter A_{2n} indicates a

slot pointer, a parameter index for a multiplicand A, and a parameter index for a multiplicand B. Each parameter index has a code bit which indicates whether the parameter is in the current slot or the slot indicated by the slot pointer. The product of A and B is store in A_{2n+1}. Parameter B_{2n} indicates a slot pointer and two parameter indices for multiplicands A′ and B′ which are either in the current slot or the slot pointed to by the slot pointer. The product of A′ and B′ is store in B_{2n+1}.

Referring again to Fig. 5, for enabled slots, background processor 102 performs background tasks as indicated by mode values from control registers 105 and parameters from parameter memories 107 and 108. There are four background processor mode values for each slot indicating four background tasks. Each task requires up to four multiply-and-accumulate operations. Accordingly, background processor 102 executes up to sixteen multiply-and-accumulate operations per slot which is exactly the same as the maximum number of instruction executed per slot by foreground processor 101.

Typically, foreground processor 101 processes a slot after background processor 102 has completed all of the background tasks for the slot. For example, at the beginning of a sampling period, background processor 102 starts processing background tasks for slot 0 and foreground processor 101 is idle. Foreground processor 101 starts processing slot 0 after background processor 102 has completed slot 0 and written preprocessed data into parameter memories 107 and 108, so that although background processor 102 and foreground processor 101 share math unit 103 in parallel interleave fashion, foreground and background processors 101 and 102 do not simultaneously process the same slot.

Fig. 9 shows timing diagrams indicating an example of the operation of background processor 102 in accordance with the present invention as shown in Fig.

5. Each sampling period, a sample clock asserts a signal START_BP which starts operation background processor 102. Background processor starts decoding and executing background tasks starting with slot 0 (or the first enabled slot). Execution of operations to complete background tasks are pipelined and controlled by timing signals generated by a pipeline timing circuit 505.

Decoding by task decoder 506 begins with reading of mode values from control registers 105 and parameter values from memories 107 and 108. Background processor 102 has access to memories 107 and 108 once per instruction cycle of foreground processor 101. Accordingly, background 102 can read two parameters per instruction cycle, one from each or memories 107 and 108.

A signal RA0B0 is asserted low during instruction cycles in which background processor 102 reads even indexed parameters A_{2n} and B_{2n} from memories 107 and 108. For example, during instruction cycle 901, background processor 102 reads parameters A_0 and B_0 from slot zero. A signal RA1B1 is asserted low during instruction cycles in which background processor 102 reads odd indexed parameters A_{2n+1} and B_{2n+1} from memories 107 and 108. For example, during instruction cycle 902, background processor 102 reads parameters A_1 and B_1 from slot zero.

After parameters A_0, B_0, A_1, and B_1 and the mode nibble are read, task decoder 506 determines the task to execute. Fig. 9 illustrates the example of a delay line mode background task described above. The delay line mode background task requires one write to DRAM 502, four reads from DRAM 502, and four multiply-and-accumulates operations. A row address signal RAS is asserted to DRAM 502 at time 951 after task decoder 506 and DRAM controller 510 have determined a row address (or physical page) for memory 502. For delay line mode, the physical page is determined from parameter A_0, the write index pointer, and the absolute pointer as

described above. While signal RAS remains asserted, a column address signal CAS is asserted five times, once for a write and four times for reads from the same page in DRAM 502. Data read from DRAM 502 goes into FIFO buffer 503.

Execution of background tasks is pipe lined. While the signal RAS is assert, background processor 102 continues to access memories 107 and 108 and begins reading parameters for the next background task. During instruction cycle 905, signal RA0B0 is asserted, and background processor 102 reads parameters A_2 and B_2 from slot zero. During instruction cycle 906, signal RA1B1 is asserted, and background processor 102 reads parameters A_3 and B_3 from slot zero. Accordingly, during instruction cycles 905 and 906, background processor 102 is reading parameters for the second background task of slot 0 and is accessing DRAM 502 for the first background task of slot 0. During instruction cycle 909, background processor 102 is controlling a multiply-and-accumulate operation for the first task of slot 0, accessing DRAM 502 for the second task of slot 0, and reading parameters for a third task of slot 0.

Each background task has four opportunities to use shared MAC 215. Signals MK0, MK1, MK2, and MK4 are asserted low if the background task actually uses shared MAC 215 during the first, second, third, or fourth opportunity, respectively. For interpolation, each of the four opportunities is used. Signal MK0 is asserted low during instruction cycle 906, and MAC 215 multiplies a value from FIFO buffer 503 (the first value read from DRAM 502 after time 951 when signal RAS was asserted) by a first interpolation coefficient from interpolation ROM 501. During instruction cycles 907, 908, and 909 successive values from FIFO buffer 503 are multiplied by corresponding values from interpolation ROM 501 and the results are accumulated. After four multiplications, the accumulated results is the desired interpolated value.

Signals WA0B0 and WA1B1 are asserted low during instruction cycles when background processor 102 writes to memories 107 and 108. For the first task of slot 0, signal WA1B1 is asserted low during instruction cycle 911, and background processor 102 writes the desired interpolated value to parameter B_1. In the embodiment described above, interpolated values are always written to odd parameters, typically B_{2n+1}. Values written to even number parameters do not require interpolation or shared MAC 215. Accordingly, assertion of signal WA0B0 during instruction cycle 904 corresponds to the first task of slot 0 and even though the last write operation for the first task of slot 0 does occur until instruction cycle 911. This timing is maintained for all background tasks regardless of the number of multiply-and-accumulate operations actually employed by a particular background task.

The second, third, and fourth task of slot zero proceed in the same manner as described above, and background processor 102 writes final results to memories 107 and 108 during instruction cycles 915, 919, and 923 respectively. Reading parameter values for slot 1 begins with instruction cycle 917 which is before the third and fourth task of slot 0 are complete.

After instruction cycle 923, all the background tasks for slot 0 are completed and slot 0 is ready to be processed by foreground processor 101. A signal START_FP is asserted low to commence processing by foreground processor 101. Foreground processor 101 executes the instructions in instruction memory 110 while background processor 102 continues processing tasks for another slot. Because the time required for background processor 102 to complete a slot equals the time required for foreground processor 101 to complete a slot, background processor 102 completes each slot before foreground processor 101 begins the slot.

Processing continues in this fashion until all enabled slots have been processed by both foreground

processor 101 and background processor 102. Foreground processor 101 then transfers eight sound amplitude values from parameter memory 109 to CODEC interface 111 where one or more DAC or CODEC can access the sound amplitudes. Foreground processor 101 and background processor 102 are then idle until a sample rate clock causes signal START_BP to be asserted again, and background processor 102 begins again with slot 0. In some embodiments, background processor 102 begins before the CODEC interface has read all the sound amplitude values. This is possible because background processor 102 does not disturb the sound amplitude values.

Although the present invention has been described with reference to particular embodiments, the description is only an example of the invention's application and should not be taken as a limitation. Accordingly, various modifications, adaptations, substitutions and combinations of different features of the specific embodiments can be practiced without departing from the scope of the invention set forth in the appended claims.

APPENDIX A

Foreground Processor Instruction Set:

These instructions use a multi-operand structure to enhance random access use of all memories. There are no time hits with respect to pipe-line delays. All instructions are one cycle execution time.

Three operand instructions:

There are 12 different types of three operand instruction. The remaining instructions require 2 operands or less and use an additional 4 bits for defining instruction types.

0000-kkkk-jjjj-iiii $C_i=A_j*B_k$; MUL
0001-kkkk-jjjj-iiii $C_i=A_j*C_k$; Note 1
0010-kkkk-jjjj-iiii $C_i=B_j*C_k$; Note 1
0011-kkkk-jjjj-iiii $C_i=C_j*C_k$; Note 1
0100-kkkk-jjjj-iiii $C_i=C_i+A_j*B_k$; MAC
0101-kkkk-jjjj-iiii $C_i=C_i+A_j*C_k$; Note 2
0110-kkkk-jjjj-iiii $C_i=C_i+B_j*C_k$; Note 2
0111-kkkk-jjjj-iiii $C_i=C_i+C_j*C_k$; Note 2
1000-kkkk-0jjj-iiii $C_i=C_i+C_k*(1-A_{j+8})$; j {0..7} PAN
1000-kkkk-1jjj-iiii $C_i=C_i+C_k*(1-B_{j+8})$; j {0..7}
1001-kkkk-jjjj-iiii $C_{i+1}=C_i=C_i+A_j*B_k$; Biquad
1010-kkkk-jjjj-iiii $C_i=C_k-B_j$; (Scattering junction.)
1011 Expanded Instruction Group A
1100 Expanded Instruction Group B
1101 Expanded Instruction Group C
1110 Expanded Instruction Group D
1111 Expanded Instruction Group E

Note 1: C values are pre-truncated.
Note 2: A or B values are pre-expanded.

Group A

1011-0000-jjjj-0iii NF=(C_{i+7}{31}^A_j{15})*(-2)+1,
C_{i+8}=abs(CL_{i+7}*Aj)*NF

1011-0000-jjjj-1iii NF=(C_{i+9}{31}^A_j{15})*(-2)+1,
C_{i+8}=abs(CL_{i+9}*Aj)*NF

1011-0001-jjjj-0iii C_{i+8}=abs(B_j*CH_{i+7})*NF

1011-0001-jjjj-1iii C_{i+8}=abs(B_j*CH_{i+9})*NF

1011-0010-jjjj-0iii C_{i+8}=(C_{i+8}>>16)+A_j*CH_{i+7} ;
Note: Sign extended

1011-0010-jjjj-1iii C_{i+8}=(C_{i+8}>>16)+ A_j*CH_{i+9} ;
Note: Sign extended

1011-0011-jjjj-iiii C_i=B_j-C_i

1011-0100-0jjj-iiii C_{i-1}=A_{j+8}+B_{j+8}*C_i ; f(x)=m(x)+b

1011-0100-1jjj-iiii C_{i+1}=A_{j+8}+B_{j+8}*C_i

1011-0101-00jj-iiii C_i=B1+A_{j+8}*C_i

1011-0101-01jj-iiii C_i=B3+A_{j+8}*C_i

1011-0101-10jj-iiii C_i=B5+A_{j+8}*C_i

1011-0101-11jj-iiii C_i=B7+A_{j+8}*C_i

1011-0110-jjjj-iiii C_i=A_j+B1*C_i ; Scale output of BP

1011-0111-jjjj-iiii C_i=A_j+B3*C_i

1011-1000-jjjj-iiii C_i=A_j+B5*C_i

1011-1001-jjjj-iiii C_i=A_j+B7*C_i

1011-1010-jjjj-iiii C_i=A_j*WN ; White Noise

1011-1011-jjjj-iiii C_i=B_j*WN

1011-1100-jjjj-iiii C_i=C_i+A_j*WN

1011-1101-jjjj-iiii C_i=C_i+B_j*WN

1011-1110-jjjj-iiii C_i=C_j*WN

1011-1111-jjjj-iiii C_i=C_i+C_j*WN

Group B: (Adds and Subtracts)

1100-Ins[0..15]-Pi[0..15]-Pj[0..15]
1100-0000-jjjj-iiii $C_i = C_i + C_j$
1100-0001-jjjj-iiii $C_i = C_i + A_j$
1100-0010-jjjj-iiii $C_i = C_i + B_j$
1100-0011-0jjj-0iii $C_{i+8} = A_{j+8} + B1$; i, j={ 0..7 }
1100-0011-0jjj-1iii $C_{i+8} = A_{j+8} + B3$
1100-0011-1jjj-0iii $C_{i+8} = A_{j+8} + B5$
1100-0011-1jjj-1iii $C_{i+8} = A_{j+8} + B7$
1100-0100-jjjj-iiii $C_{i-1} = C_i + C_j$; i-1 wraps around
1100-0101-jjjj-iiii $C_{i-1} = C_i + A_j$
1100-0110-jjjj-iiii $C_{i-1} = C_i + B_j$
1100-0111-jjjj-iiii $C_i = A_j - C_i$
1100-1000-jjjj-iiii $C_{i-1} = C_j - C_i$
1100-1001-jjjj-iiii $C_{i-1} = C_i - C_j$
1100-1010-jjjj-iiii $C_i = C_i - C_j$
1100-1011-jjjj-iiii $C_i = C_j - C_i$
1100-1100-jjjj-iiii $C_{i-1} = A_j - C_i$
1100-1101-jjjj-iiii $C_{i-1} = B_j - C_i$
1100-1110-jjjj-iiii $C_{i-1} = C_i - A_j$
1100-1111-jjjj-iiii $C_i = C_i - A_j$

Note: saturation occurs if value C greater than 15.99 or less than -15.99.

Group C: (Filters)

These instructions include an additional move after a multiply operation and may be used for filters.

1101-Ins[0..15]-Pi..15]-Pj0..15]

1101-0000-jjjj-iiii $C_i=C_i+A_j*B_j$, $A_j=B_{j-1}$;
Filter with Post Move

1101-0001-jjjj-iiii $C_i=C_i+B_j*A_j$, $B_j=A_{j-1}$

1101-0010-jjjj-iiii $C_i=A_j*B_j$, $A_j=B_{j-1}$

1101-0011-jjjj-iiii $C_i=B_j*A_j$, $B_j=A_{j-1}$

1101-0100-jjjj-iiii $C_i=C_i+A_j*B_j$, $A_j=C_{i-1}$

1101-0101-jjjj-iiii $C_i=C_i+B_j*A_j$, $B_j=C_{i-1}$

1101-0110-jjjj-iiii $C_{i+1}=C_i-A_j*B_j$, $C_{i-1}=A_j$; Lattice Filter, i+1 and i-1 wrap around.

1101-0111-jjjj-iiii $C_{i+1}=C_i-A_j*B_j$, $C_{i-1}=B_j$

1101-1000-jjjj-iiii $C_i=C_i+A_j*B_j$, $A_j=B1$

1101-1001-jjjj-iiii $C_i=C_i+A_j*B_j$, $A_j=B3$

1101-1010-jjjj-iiii $C_i=C_i+A_j*B_j$, $A_j=B5$

1101-1011-jjjj-iiii $C_i=C_i+A_j*B_j$, $A_j=B7$

1101-1100-jjjj-iiii $C_i=A_j*B_j$, $A_j=B1$

1101-1101-jjjj-iiii $C_i=A_j*B_j$, $A_j=B3$

1101-1110-jjjj-iiii $C_i=A_j*B_j$, $A_j=B5$

1101-1111-jjjj-iiii $C_i=A_j*B_j$, $A_j=B7$

Group D: (Moves)

1110-0000-jjjj-iiii Bi=Aj
1110-0001-jjjj-iiii Bi=Bj
1110-0010-jjjj-0iii A_{2i+1}=Aj
1110-0010-jjjj-1iii A_{2i+1}=Bj
1110-0011-jjjj-iiii Aj=Ci*4; shift by 2 bits
1110-0100-jjjj-iiii Bj=Ci*4
1110-0101-jjjj-iiii Aj=Ci*8 ; Shift by 3 bits
1110-0110-jjjj-iiii Bj=Ci*8
1110-0111-jjjj-iiii $A_j=C_i$; (truncates toward 0)
1110-1000-jjjj-iiii $B_j=C_i$; (truncates toward 0)
1110-1001-jjjj-iiii $C_i=A_j$; (expands)
1110-1010-jjjj-iiii $C_i=B_j$; (expands)
1110-1011-0jjj-0iii $C_{i+8}=AB_{j+8}$; i, j {0..7}
1110-1011-1jjj-0iii $AB_{j+8}=C_{i+8}$; Save state of C_{i+8}
1110-1011-0jjj-1iii $C_{2i}=AB_{2j}$, $C_{2i+1}=AB_{2j+1}$
1110-1011-1jjj-1iii $AB_{2j}=C_{2i}$, $AB_{2j+1}=C_{2i+1}$
1110-1100-00jj-00ii $A_{4j}=C_{4i}$, $B_{4j+1}=C_{4i+1}$, $A_{4j+2}=C_{4i+2}$, $B_{4j+3}=C_{4i+3}$
1110-1100-00jj-01ii $B_{4j}=C_{4i}$, $A_{4j+1}=C_{4i+1}$, $B_{4j+2}=C_{4i+2}$, $A_{4j+3}=C_{4i+3}$
1110-1100-00jj-10ii $A_{4j}=C_{4i-1}$, $B_{4j+1}=C_{4i}$, $A_{4j+2}=C_{4i+1}$, $B_{4j+3}=C_{4i+3}$
1110-1100-00jj-11ii $B_{4j}=C_{4i-1}$, $A_{4j+1}=C_{4i}$, $B_{4j+2}=C_{4i+1}$, $A_{4j+3}=C_{4i+3}$
1110-1100-01jj-000i $A_{j+8}=C_{4i+8}$, $B_{j+8}=C_{4i+9}$, $A_{j+12}=C_{4i+10}$, $B_{j+12}=C_{4i+11}$
1110-1100-01jj-001i A1=C_{4i+8} , $B_{j+8}=C_{4i+9}$, $A_{j+12}=C_{4i+10}$, $B_{j+12}=C_{4i+11}$
1110-1100-01jj-010i A1=C_{4i+8}, $B_{2j}=C_{4i+9}$, A3=C_{4i+10} , $B_{j+12}=C_{4i+11}$
1110-1100-01jj-011i A1=C_{4i+8}, $B_{2j}=C_{4i+9}$, A3=C_{4i+10} , $B_{2j+2}=C_{4i+11}$; if j=3, 2j+2 > 0
1110-1100-0100-1iii C_{i+8}=B1+A_{i+8} , A_{i+8}=B1 ; i = { 0-7 }
1110-1100-0101-1iii C_{i+8}=B3+A_{i+8} , A_{i+8}=B3
1110-1100-0110-1iii C_{i+8}=B5+A_{i+8} , A_{i+8}=B5
1110-1100-0111-1iii C_{i+8}=B7+A_{i+8} , A_{i+8}=B7
1110-1100-1kii-0011 $B_{0+8K}=C_{i+12}$, $B_{2+8K}=C_{i+13}$; K={0,1}, i={0-3}, Overflow wraps to C12, example: B8=C15 , B10=C12
1110-1100-1kii-0110 $B_{2+8K}=C_{i+12}$, $B_{4+8K}=C_{i+13}$
1110-1100-1kii-1100 $B_{4+8K}=C_{i+12}$, $B_{6+8K}=C_{i+13}$

1110-1100-1kii-1001 $B_{0+8K}=C_{i+12}$, $B_{6+8K}=C_{i+13}$

1110-1100-1kii-0101 $B_{0+8K}=C_{i+12}$, $B_{4+8K}=C_{i+13}$

1110-1100-1kii-1010 $B_{2+8K}=C_{i+12}$, $B_{6+8K}=C_{i+13}$

1110-1100-1kii-0010 $A_{1+8K}=C_{i+12}$, $A_{3+8K}=C_{i+13}$

1110-1100-1kii-0111 $A_{3+8K}=C_{i+12}$, $A_{5+8K}=C_{i+13}$

1110-1100-1kii-1101 $A_{5+8K}=C_{i+12}$, $A_{7+8K}=C_{i+13}$

1110-1100-1kii-1000 $A_{1+8K}=C_{i+12}$, $A_{7+8K}=C_{i+13}$

1110-1100-1kii-0100 $A_{1+8K}=C_{i+12}$, $A_{5+8K}=C_{i+13}$

1110-1100-1kii-1011 $A_{3+8K}=C_{i+12}$, $A_{7+8K}=C_{i+13}$

1110-1100-1kii-0000 $B_{0+8K}=C_{i+12}$, $A_{1+8K}=C_{i+13}$

1110-1100-1kii-0001 $B_{2+8K}=C_{i+12}$, $A_{3+8K}=C_{i+13}$

1110-1100-1kii-1110 $B_{4+8K}=C_{i+12}$, $A_{5+8K}=C_{i+13}$

1110-1100-1kii-1111 $B_{6+8K}=C_{i+12}$, $A_{7+8K}=C_{i+13}$

1110-1101 1-OP group I

1110-1110 1-OP group F

1110-1111 1-OP group G

Group E (Envelope Instructions)

Many of the Group E Instruction depend on functions which are described on the following pages.

1111-0000-jjjj-0iii C_{i+8}=ADSR0_K(C_{i+8}, A_j, B_j) ;
Slow Attack, Slow Release

1111-0000-jjjj-1iii C_{i+8}=ADSR0_A(C_{i+8},A_j,B_j)

1111-0001-jjjj-0iii C_{i+8}=ADSR1_K(C_{i+8}, A_j, B_j) ;
Fast Attack, Slow Release

1111-0001-jjjj-1iii C_{i+8}=ADSR1_A(C_{i+8},A_j,B_j)

1111-0010-jjjj-0iii C_{i+8}=ADSR2_K(C_{i+8}, A_j, B_j) ;
Fast Attack, Fast Release

1111-0010-jjjj-1iii C_{i+8}=ADSR2_A(C_{i+8},A_j,B_j)

1111-0011-jjjj-0iii C_{i+8}=DECAY_K(C_{i+8},B_j) ; i={ 0..7 }

1111-0011-jjjj-1iii C_{i+8}=DECAY_A(C_{i+8},Bj)

1111-0100-jjjj-0iii C_{i+8}=ADR0_K(C_{i+8}, A_j, B_j) ; ADR is like ADSR but ADR has no sustain.

1111-0100-jjjj-1iii C_{i+8}=ADR0_A(C_{i+8},A_j,B_j)

1111-0101-jjjj-0iii C_{i+8}=ADR1_K(C_{i+8}, A_j, B_j)

1111-0101-jjjj-1iii C_{i+8}=ADR1_A(C_{i+8},A_j,B_j)

1111-0110-jjjj-0iii C_{i+8}=ADR2_K(C_{i+8}, A_j, B_j)

1111-0110-jjjj-1iii C_{i+8}=ADR2_A(C_{i+8},A_j,B_j)

1111-0111-kkjj-00ii B_{2k+8}=C_{i+12}, C_{i+12}=B1*A_{2j+8}

1111-0111-kkjj-01ii B_{2k+8}=C_{i+12}, C_{i+12}=B3*A_{2j+8}

1111-0111-kkjj-10ii B_{2k+8}=C_{i+12}, C_{i+12}=B5*A_{2j+8}

1111-0111-kkjj-11ii B_{2k+8}=C_{i+12}, C_{i+12}=B7*A_{2j+8}

1111-1000-jjjj-iiii C_i=A_j*B1, A1=C_{i-1}

1111-1001-jjjj-iiii C_i=A_j*B3, A3=C_{i-1}

1111-1010-jjjj-iiii C_i=A_j*B5, A5=C_{i-1}

1111-1011-jjjj-iiii C_i=A_j*B7, A7=C_{i-1}

1111-1100-jjjj-iiii C_i += C_{i-1}, A_j=C_{i-1}

1111-1101-jjjj-iiii C_i += C_{i-1}, B_j=C_{i-1}

1111-1110-jjjj-00ii C_{i+12}=Aj*B1, B0=C_{i+8}

1111-1110-jjjj-01ii C_{i+12}=Aj*B3, B2=C_{i+8}

1111-1110-jjjj-10ii C_{i+12}=Aj*B5, B4=C_{i+8}

1111-1110-jjjj-11ii C_{i+12}=Aj*B7, B6=C_{i+8}

1111-1111 1-OP Group H

ADSR and ADR Envelope Instructions:

The envelope instructions perform different operations depending on control values and whether the envelope is in an attack, a decay, a sustain, or a release phase.

Attack:

After Key On Edge, the attack phase begins, and the instruction performs

Ci=Ci+Aj[6..0]*4 for ADSR2 and ADSR1, or

Ci=Ci+Aj[6..0]*(1/8) for ADSR0 (slow attack).

Decay:

If Ci reaches 1 or higher, the decay phase begins, and Aj[7] is set. During the decay phase (Key On and Aj[7] set), the instruction performs

Ci=Ci-Ci*Bj[15..8] for ADSR2 (note: i=j) or

Ci=Ci-(Ci/8)*Bj[15..8] for ADSR 1 and ADSR0.

At -75dB, the decay speeds up to the fast rate.

Sustain:

If Ci <= Aj[14..8] the sustain phase begins, Aj[15] is set, and otherwise the instruction performs NOP. A sustain phase does not apply to ADR.

Release:

If Key Off, the release phase begins, and the instruction performs

Ci=Ci-Ci*Bj[7..0]; Aj7=0 for ADSR2 or

Ci=Ci-(Ci/8)*Bj[7..0] for ADSR 1 and ADSR0.

At -75dB, the release speeds up to Fast rate.

ADR0 through ADR2 instructions work the same as ADSR0 through ADSR2 instructions except that there is no sustain point.

Note: C_i will be lost unless saved into and out of an A or B parameter. This means that the Foreground Envelope Generator requires two overhead instructions (get state and save state). C_i must be saved before it is modified. Below is an example program which uses an envelope instruction:

```
C15=AB15;                  Get previous value.
C15=ADSR2(C15, A14,B14); C15 is the new Envelope value.
*AB=*C, *C=*C*A9;          Store and use Envelope value.
```

Decay Instruction:

The decay instruction augments the Long Table modes. The last page may be looped in these modes. This instruction creates an exponentially decaying envelope during that last page as it repeats. This instruction requires 2 additional instructions to save and restore the C register. The instruction functions as follows:

Default Condition: NOP

Last Page Detected: Ci=Ci-(Ci/8)*Bj ;

Example Program:

```
C14=AB15 ;                 Get previous envelope data.
C14=DECAY(C14, B7);        C14 is the envelope Data.
AB15=C14 ;                 Save new envelope data.
```

Note: if the Decay flag is not true, B7 would simply be copied to C14, otherwise C14=C14*B7.

Estimation of Exponential Decays:
This uses -80dB as the target level.

$0.0001 = (\text{Fraction})^n$

Where n=Number of samples to achieve the targeted decay. Fraction equals [1-B] for fast modes and [1-B/8] for slow modes. For ADSR functions, B may take on a maximum value of 7.8E-3 for fast modes; this produces a fraction equal to 0.992. In slow mode, B may take on an effective value of 3.8E-6 producing the fraction equal to 0.999996. The decay time is as follows:

$$\text{Decay} = \frac{-4}{SR \times LOG_{10}(\text{Fraction})}$$

Where SR is defined as the sample rate.

One Operand Instructions: Group F

```
1110-1110-0000-0nnn SKIP n if KEY ON ; Note: n={0..7},
                    If n=0 then skip 8.
1110-1110-0000-1nnn SKIP n if not KEY ON
1110-1110-0001-0nnn SKIP n if ALT KEY ON
1110-1110-0001-1nnn SKIP n if not ALT KEY ON
1110-1110-0010-0nnn SKIP n if KEY ON EDGE
1110-1110-0010-1nnn SKIP n if not KEY ON EDGE
1110-1110-0011-0nnn SKIP n if ALT KEY ON EDGE
1110-1110-0011-1nnn SKIP n if not ALT KEY ON EDGE
1110-1110-0100-0nnn SKIP n if DECAY FLAG
1110-1110-0100-1nnn SKIP n if not DECAY FLAG
1110-1110-0101-0nnn SKIP n if KEY OFF EDGE
1110-1110-0101-1nnn SKIP n if not KEY OFF EDGE
1110-1110-0110-0nnn SKIP n if ALT KEY OFF EDGE
1110-1110-0110-1nnn SKIP n if not ALT KEY OFF EDGE
1110-1110-0111-0nnn SKIP n if > EFFECTIVE ZERO
1110-1110-0111-1nnn SKIP n if < or = EFFECTIVE ZERO
1110-1110-1000-0nnn SKIP n if > or = EFFECTIVE ZERO
1110-1110-1000-1nnn SKIP n if LESS THAN EFFECTIVE ZERO
1110-1110-1001-0nnn SKIP n if EFFECTIVE ZERO
1110-1110-1001-1nnn SKIP n if not EFFECTIVE ZERO
1110-1110-1010-0nnn SKIP n if OVERFLOW or SAT
                    (C-Reg > (+-)15.999)
1110-1110-1010-1nnn SKIP n if not OVERFLOW or SAT
1110-1110-1011-0nnn SKIP n if GEA2 ; (> or = 2)
1110-1110-1011-1nnn SKIP n if not GEA2 ; (not > or = 2)
1110-1110-1100-0nnn SKIP n if GEA1 ; (> or = 1)
1110-1110-1100-1nnn SKIP n if not GEA1 ; (not > or = 1)
1110-1110-1101-0nnn SKIP if ABS ZERO
1110-1110-1101-1nnn SKIP if not ABS ZERO
1110-1110-1110-nnnn SKIP always (n=0 if set to zero)
1110-1110-1111-0000 NOP ; No Operation.
1110-1110-1111-0001 DSRS ; Disable all Sample Record
                    modes next Sample
1110-1110-1111-0010 SF ; Save arithmetic Flag
1110-1110-1111-0011 GF ; Get arithmetic Flag
1110-1110-1111-0100 FDA2 ; Force DRQ-A2
```

```
                        (2 DMA Reads or writes)
    1110-1110-1111-0101 FDA4 ; Force DRQ-A4 (4 DMA Reads)
    1110-1110-1111-0110 FDB2 ; Force DRQ-B2
                        (2 DMA Reads or writes)
    1110-1110-1111-0111 FDB4 ; Force DRQ-B4 (4 DMA Reads)
    1110-1110-1111-1000 JMP[C8] ; execute instructions in
                        new slot.
    1110-1110-1111-1001 RET ; return to normal slot.
    1110-1110-1111-1010 GDS ; Get DMA Status.
                        Zero Flag='1' means data not ready.
    1110-1110-1111-1011 FDR ; Get DMA Error.
                        Zero Flag='1' means no error.
    1110-1110-1111-1100 FKI ; Force Key Interrupt of
                        Current Slot
    1110-1110-1111-1101 FAI ; Force AltKey Interrupt of
                        Current Slot
    1110-1110-1111-1110 IKZ ; Interrupt if KEY OFF and
                        effective zero
    1110-1110-1111-1111 IAZ ; Interrupt if ALTKEY OFF and
                        effective zero
```

Group G

1110-1111-0000-0iii A1=MOVH8(A1, C_{i+8}) ;
Long Table Step Rate
1110-1111-0001-0iii A3=MOVH8(A3, C_{i+8})
1110-1111-0010-0iii A5=MOVH8(A5, C_{i+8})
1110-1111-0011-0iii A7=MOVH8(A7, C_{i+8})
1110-1111-0100-0iii A1=MOVH10(A1, C_{i+8}) ; ADSR Step Rate
1110-1111-0101-0iii A3 =MOVH10(A3, C_{i+8})
1110-1111-0110-0iii A5=MOVH10(A5, C_{i+8})
1110-1111-0111-0iii A7=MOVH10(A7, C_{i+8})
1110-1111-1000-0iii B0=MOVADL(A0 , B0, C_{i+8}) ; Delay Line
1110-1111-1001-0iii B2=MOVADL(A2 , B2, C_{i+8})
1110-1111-1010-0iii B4=MOVADL(A4 , B4, C_{i+8})
1110-1111-1011-0iii B6=MOVADL(A6 , B6, C_{i+8})
1110-1111-1100-0iii B0=MOVADH(A0 , B0, C_{i+8}) ;
Look Up Table
1110-1111-1101-0iii B2=MOVADH(A2 , B2, C_{i+8})
1110-1111-1110-0iii B4=MOVADH(A4 , B4, C_{i+8})
1110-1111-1111-0iii B6=MOVADH(A6 , B6, C_{i+8})
1110-1111-0000-1iii A1=MOVH8(A1, C_{i+8}) , B2=C_{i+9}
1110-1111-0001-1iii A3=MOVH8(A3, C_{i+8}) , B4=C_{i+9}
1110-1111-0010-1iii A5=MOVH8(A5, C_{i+8}) , B6=C_{i+9}
1110-1111-0011-1iii A7=MOVH8(A7, C_{i+8}) , B0=C_{i+9}
1110-1111-0100-1iii A1=MOVH10(A1, C_{i+8}) , B2=C_{i+9}
1110-1111-0101-1iii A3=MOVH10(A3, C_{i+8}) , B4=C_{i+9}
1110-1111-0110-1iii A5=MOVH10(A5, C_{i+8}) , B6=C_{i+9}
1110-1111-0111-1iii A7=MOVH10(A7, C_{i+8}) , B0=C_{i+9}
1110-1111-1000-1iii B0=MOVADL(A0 , B0, C_{i+8}) , A1=C_{i+9}
1110-1111-1001-1iii B2=MOVADL(A2 , B2, C_{i+8}) , A3=C_{i+9}
1110-1111-1010-1iii B4=MOVADL(A4 , B4, C_{i+8}) , A5=C_{i+9}
1110-1111-1011-1iii B6=MOVADL(A6 , B6, C_{i+8}) , A7=C_{i+9}
1110-1111-1100-1iii B0=MOVADH(A0 , B0, C_{i+8}) , A3=C_{i+9}
1110-1111-1101-1iii B2=MOVADH(A2 , B2, C_{i+8}) , A5=C_{i+9}
1110-1111-1110-1iii B4=MOVADH(A4 , B4, C_{i+8}) , A7=C_{i+9}
1110-1111-1111-1iii B6=MOVADH(A6 , B6, C_{i+8}) , A1=C_{i+9}

Group H

```
1111-1111-0000-00ii UNPACKS(&A14,&B14,&A15,&B15,C[i+12]) ;
                    Align A,B[15-12]
1111-1111-0000-01ii UNPACKU(&A14,&B14,&A15,&B15,C[i+12]) ;
                    Align A,B[14-11]
1111-1111-0000-10ii C[i+12]=PACKU(A14,B14,A15,B15)
1111-1111-0000-11ii C[i+12]=PACKS(A14,B14,A15,B15)
1111-1111-0001-iiii B14=Ai | B15 ; Bit-wise OR
1111-1111-0010-iiii A14=Ai & B15 ; Bit-wise AND
1111-1111-0011-iiii B14=Ai ^ B15 ; Bit-wise XOR
1111-1111-0100-0000 RSPP ; Return to Initial Slot
1111-1111-0100-0001 SPPA[C8] ; Set Slot Parameter
                    Pointer Absolute
1111-1111-0100-0010 SPPR[C8] ; Set Slot Parameter
                    Pointer Relative to Initial Slot
1111-1111-0100-0011 SPPL[C8] ; Set Slot Parameter
                    Pointer Relative to Current Slot
1111-1111-0100-0100 FAR A[C9]=FAR A[C8] ; Relative to
                    Relative Moves
1111-1111-0100-0101 FAR A[C9]=FAR B[C8]
1111-1111-0100-0110 FAR B[C9]=FAR A[C8]
1111-1111-0100-0111 FAR B[C9]=FAR B[C8]
1111-1111-0100-1000 A[C9]=A[C8] ; Near Relative to
                    Relative Moves
1111-1111-0100-1001 A[C9]=B[C8]
1111-1111-0100-1010 B[C9]=A[C8]
1111-1111-0100-1011 B[C9]=B[C8]
1111-1111-0100-1100 FKF[C8] ; force KEY OFF
1111-1111-0100-1101 FKO[C8] ; force KEY ON
1111-1111-0100-1110 FAF[C8] ; force ALTKEY OFF
1111-1111-0100-1111 FAO[C8] ; force ALTKEY ON
1111-1111-0101-0iii C[i+8]=GSN ; C={0,0,0,0,0,0,SN[4-0],0,
                    ..., 0}, SN=Current Slot Number
1111-1111-0101-1iii C[i+8]=WP ; C={0,0,0,0,wp16,wp15,...,
                    wp1,wp0}-rev2
1111-1111-0110-0iii Reserved
1111-1111-0110-10ii B[i+12] =FAR I[C8] ; Get Instruction
                    Word
```

1111-1111-0110-1111 B_{i+12}=I[C8] ; Get Instruction Word
1111-1111-0111-nnnn ZF=B15(n) ; n is the nth bit of B15 where n=0 is the LSB Bit.
1111-1111-1000-iiii Bi=MW[C8] ; This gets current Mode Word
1111-1111-1001-iiii MW[C8]=Bi ; This changes the Background task.
1111-1111-1010-iiii *AB=*C, *C=*C*C_i ; Index* = last i of last instruction
1111-1111-1011-iiii *AB=*C, *C=*C*A_i ;
1111-1111-1100-iiii *AB=*C, *C=*C*B_i ; Envelope use and state save
1111-1111-1101-iiii *AB=*C, *C=*C*C_i+C_{i+1} ; Index * defined as previous instruction i
1111-1111-1110-iiii *AB=*C, *C=*C*Ai+Bi ;
1111-1111-1111-iiii *AB=*C, *C=*C*Bi+Ai ;

Relative Addressing:
Parameter in square brackets such as [C8] are relative addresses which permit accessing parameters and instructions from any slot. Only [C8] and [C9] may be used as Relative Addresses. A relative address contains five bits S0 to S4 which indicate a slot number and four bits P0 to P3 which indicate a parameter within the slot. Additionally, the relative address contains two bits R0 and R1 which indicate the addressing mode. The two bits R1 and R0 function as follows:
R1=0 R0=0 Absolute Pointer to Slot Space.
R1=0 R0=1 Local to Current Slot. S4-S0 are ignored.
R1=1 R0=0 Relative to Parameter Pointer.
R1=1 R0=1 Relative to Current Slot.
The relative address is defined as the sum of the slot pointer S4-S0 and either the current slot or the parameter pointer. The sum output wraps around.

Group I

```
1110-1101-0000-iiii FAR B[C8]=Ai ; Relative Store Far
                    from Local
1110-1101-0001-iiii FAR B[C8]=Bi
1110-1101-0010-iiii FAR A[C8]=Ai
1110-1101-0011-iiii FAR A[C8]=Bi
1110-1101-0100-iiii Ai=FAR B[C8] ; Relative Store Local
                    from Far
1110-1101-0101-iiii Bi=FAR B[C8]
1110-1101-0110-iiii Ai=FAR A[C8]
1110-1101-0111-iiii Bi=FAR A[C8]
1110-1101-1000-iiii B[C8]=Ai ; Relative Store Local to
                    Local
1110-1101-1001-iiii B[C8]=Bi
1110-1101-1010-iiii A[C8]=Ai
1110-1101-1011-iiii A[C8]=Bi
1110-1101-1100-iiii Ai=B[C8] ; Relative Store Local to
                    Local
1110-1101-1101-iiii Bi=B[C8]
1110-1101-1110-iiii Ai=A[C8]
1110-1101-1111-iiii Bi=A[C8]
```

APPENDIX B

Lagrange Data

0080	ffd6	7fbf	0080	ffeb
0081	ff83	7f3b	0182	ffc1
0082	ff31	7eb3	0286	ff96
0083	fee2	7e27	038b	ff6b
0084	fe95	7d98	0493	ff41
0085	fe49	7d04	059c	ff16
0086	fdff	7c6d	06a8	feec
0087	fdb8	7bd2	07b4	fec2
0088	fd72	7b34	08c3	fe97
0089	fd2e	7a92	09d3	fe6d
008a	fcec	79ec	0ae5	fe44
008b	fcab	7943	0bf8	fe1a
008c	fc6d	7896	0d0c	fdf0
008d	fc30	77e6	0e23	fdc7
008e	fbf5	7733	0f3a	fd9e
008f	fbbc	767c	1053	fd75
0090	fb85	75c2	116d	fd4c
0091	fb4f	7505	1288	fd24
0092	fb1c	7444	13a4	fcfc
0093	fae9	7381	14c2	fcd4
0094	fab9	72ba	15e0	fcac
0095	fa8a	71f1	1700	fc85
0096	fa5d	7124	1821	fc5e
0097	fa32	7054	1942	fc38
0098	fa08	6f82	1a65	fc11
0099	f9e0	6ead	1b88	fbec
009a	f9b9	6dd4	1cac	fbc6
009b	f994	6cf9	1dd1	fba1
009c	f970	6c1c	1ef7	fb7d
009d	f94f	6b3c	201d	fb59
009e	f92e	6a59	2144	fb35
009f	f90f	6973	226c	fb12
00a0	f8f2	688b	2394	faef
00a1	f8d6	67a1	24bc	facd
00a2	f8bc	66b4	25e5	faab
00a3	f8a3	65c5	270e	fa8a
00a4	f88b	64d3	2838	fa6a
00a5	f875	63df	2962	fa4a
00a6	f861	62e9	2a8c	fa2a
00a7	f84e	61f0	2bb6	fa0c
00a8	f83c	60f6	2ce1	f9ed
00a9	f82b	5ff9	2e0b	f9d0
00aa	f81c	5efb	2f36	f9b3
00ab	f80e	5dfa	3061	f997
00ac	f802	5cf7	318b	f97b
00ad	f7f7	5bf3	32b6	f960
00ae	f7ed	5aed	33e0	f946
00af	f7e4	59e4	350a	f92d
00b0	f7dd	58da	3634	f914
00b1	f7d7	57cf	375e	f8fc
00b2	f7d2	56c1	3888	f8e5
00b3	f7ce	55b2	39b1	f8cf
00b4	f7cc	54a1	3ad9	f8b9
00b5	f7cb	538f	3c01	f8a5
00b6	f7cb	527c	3d29	f891
00b7	f7cc	5167	3e50	f87e
00b8	f7ce	5050	3f77	f86c
00b9	f7d1	4f38	409d	f85a
00ba	f7d5	4e1f	41c2	f84a
00bb	f7db	4d05	42e6	f83a
00bc	f7e1	4be9	440a	f82c
00bd	f7e9	4acc	452c	f81e
00be	f7f2	49ae	464e	f812
00bf	f7fb	488f	476f	f806
00c0	f806	476f	488f	f7fb
00c1	f812	464e	49ae	f7f2
00c2	f81e	452c	4acc	f7e9
00c3	f82c	440a	4be9	f7e1
00c4	f83a	42e6	4d05	f7db
00c5	f84a	41c2	4e1f	f7d5
00c6	f85a	409d	4f38	f7d1
00c7	f86c	3f77	5050	f7ce
00c8	f87e	3e50	5167	f7cc
00c9	f891	3d29	527c	f7cb
00ca	f8a5	3c01	538f	f7cb
00cb	f8b9	3ad9	54a1	f7cc
00cc	f8cf	39b1	55b2	f7ce
00cd	f8e5	3888	56c1	f7d2
00ce	f8fc	375e	57cf	f7d7
00cf	f914	3634	58da	f7dd
00d0	f92d	350a	59e4	f7e4
00d1	f946	33e0	5aed	f7ed
00d2	f960	32b6	5bf3	f7f7
00d3	f97b	318b	5cf7	f802
00d4	f997	3061	5dfa	f80e
00d5	f9b3	2f36	5efb	f81c
00d6	f9d0	2e0b	5ff9	f82b
00d7	f9ed	2ce1	60f6	f83c
00d8	fa0c	2bb6	61f0	f84e
00d9	fa2a	2a8c	62e9	f861
00da	fa4a	2962	63df	f875
00db	fa6a	2838	64d3	f88b
00dc	fa8a	270e	65c5	f8a3
00dd	faab	25e5	66b4	f8bc
00de	facd	24bc	67a1	f8d6
00df	faef	2394	688b	f8f2
00e0	fb12	226c	6973	f90f
00e1	fb35	2144	6a59	f92e
00e2	fb59	201d	6b3c	f94f
00e3	fb7d	1ef7	6c1c	f970
00e4	fba1	1dd1	6cf9	f994
00e5	fbc6	1cac	6dd4	f9b9
00e6	fbec	1b88	6ead	f9e0
00e7	fc11	1a65	6f82	fa08
00e8	fc38	1942	7054	fa32
00e9	fc5e	1821	7124	fa5d
00ea	fc85	1700	71f1	fa8a
00eb	fcac	15e0	72ba	fab9
00ec	fcd4	14c2	7381	fae9
00ed	fcfc	13a4	7444	fb1c
00ee	fd24	1288	7505	fb4f
00ef	fd4c	116d	75c2	fb85
00f0	fd75	1053	767c	fbbc
00f1	fd9e	0f3a	7733	fbf5
00f2	fdc7	0e23	77e6	fc30
00f3	fdf0	0d0c	7896	fc6d
00f4	fe1a	0bf8	7943	fcab
00f5	fe44	0ae5	79ec	fcec
00f6	fe6d	09d3	7a92	fd2e
00f7	fe97	08c3	7b34	fd72
00f8	fec2	07b4	7bd2	fdb8
00f9	feec	06a8	7c6d	fdff
00fa	ff16	059c	7d04	fe49
00fb	ff41	0493	7d98	fe95
00fc	ff6b	038b	7e27	fee2
00fd	ff96	0286	7eb3	ff31
00fe	ffc1	0182	7f3b	ff83
00ff	ffeb	0080	7fbf	ffd6

Minimum Sidelobe Data

0000	0f0f	62a8	0f9f	fea9
0001	0e82	62a2	1031	fea0
0002	0df7	6298	10c6	fe96
0003	0d6f	6288	115e	fe8d
0004	0ce9	6274	11f7	fe83
0005	0c66	625b	1293	fe7a
0006	0be5	623d	1332	fe70
0007	0b67	621a	13d3	fe66
0008	0aeb	61f3	1477	fe5d
0009	0a72	61c7	151d	fe53
000a	09fc	6196	15c5	fe49
000b	0988	6160	166f	fe3f
000c	0916	6126	171c	fe35
000d	08a7	60e7	17cb	fe2b
000e	083a	60a3	187d	fe21
000f	07d0	605b	1930	fe18
0010	0769	600f	19e6	fe0e
0011	0704	5fbe	1a9e	fe04
0012	06a1	5f68	1b58	fdfa
0013	0640	5f0e	1c14	fdf1
0014	05e3	5eaf	1cd3	fde7
0015	0587	5e4d	1d93	fdde
0016	052e	5de6	1e55	fdd4
0017	04d7	5d7a	1f19	fdcb
0018	0483	5d0b	1fdf	fdc2
0019	0431	5c97	20a7	fdb9
001a	03e1	5c20	2171	fdb0
001b	0393	5ba4	223c	fda7
001c	0348	5b24	2309	fd9f
001d	02ff	5aa1	23d8	fd97
001e	02b8	5a19	24a8	fd8f
001f	0273	598e	257a	fd87
0020	0231	58ff	264d	fd80
0021	01f0	586c	2722	fd78
0022	01b2	57d6	27f8	fd72
0023	0176	573d	28cf	fd6b
0024	013b	569f	29a8	fd65
0025	0103	55ff	2a81	fd5f
0026	00cd	555b	2b5c	fd5a
0027	0099	54b4	2c38	fd55
0028	0066	540a	2d15	fd50
0029	0036	535d	2df2	fd4c
002a	0007	52ad	2ed1	fd48
002b	ffdb	51fa	2fb0	fd45
002c	ffb0	5144	3090	fd43
002d	ff87	508b	3170	fd41
002e	ff60	4fd0	3251	fd3f
002f	ff3a	4f12	3332	fd3f
0030	ff16	4e52	3414	fd3e
0031	fef3	4d8f	34f6	fd3f
0032	fed2	4cca	35d8	fd40
0033	feb3	4c03	36bb	fd42
0034	fe95	4b3a	379d	fd45
0035	fe79	4a6e	387f	fd48
0036	fe5e	49a1	3962	fd4c
0037	fe45	48d1	3a44	fd51
0038	fe2d	4800	3b25	fd57
0039	fe16	472e	3c07	fd5e
003a	fe01	4659	3ce7	fd66
003b	fded	4583	3dc8	fd6f
003c	fdda	44ac	3ea7	fd78
003d	fdc9	43d3	3f86	fd83
003e	fdb8	42f9	4064	fd8f
003f	fda9	421e	4142	fd9b

0040	fd9b	4142	421e	fda9
0041	fd8f	4064	42f9	fdb8
0042	fd83	3f86	43d3	fdc9
0043	fd78	3ea7	44ac	fdda
0044	fd6f	3dc8	4583	fded
0045	fd66	3ce7	4659	fe01
0046	fd5e	3c07	472e	fe16
0047	fd57	3b25	4800	fe2d
0048	fd51	3a44	48d1	fe45
0049	fd4c	3962	49a1	fe5e
004a	fd48	387f	4a6e	fe79
004b	fd45	379d	4b3a	fe95
004c	fd42	36bb	4c03	feb3
004d	fd40	35d8	4cca	fed2
004e	fd3f	34f6	4d8f	fef3
004f	fd3e	3414	4e52	ff16
0050	fd3f	3332	4f12	ff3a
0051	fd3f	3251	4fd0	ff60
0052	fd41	3170	508b	ff87
0053	fd43	3090	5144	ffb0
0054	fd45	2fb0	51fa	ffdb
0055	fd48	2ed1	52ad	0007
0056	fd4c	2df2	535d	0036
0057	fd50	2d15	540a	0066
0058	fd55	2c38	54b4	0099
0059	fd5a	2b5c	555b	00cd
005a	fd5f	2a81	55ff	0103
005b	fd65	29a8	569f	013b
005c	fd6b	28cf	573d	0176
005d	fd72	27f8	57d6	01b2
005e	fd78	2722	586c	01f0
005f	fd80	264d	58ff	0231
0060	fd87	257a	598e	0273
0061	fd8f	24a8	5a19	02b8
0062	fd97	23d8	5aa1	02ff
0063	fd9f	2309	5b24	0348
0064	fda7	223c	5ba4	0393
0065	fdb0	2171	5c20	03e1
0066	fdb9	20a7	5c97	0431
0067	fdc2	1fdf	5d0b	0483
0068	fdcb	1f19	5d7a	04d7
0069	fdd4	1e55	5de6	052e
006a	fdde	1d93	5e4d	0587
006b	fde7	1cd3	5eaf	05e3
006c	fdf1	1c14	5f0e	0640
006d	fdfa	1b58	5f68	06a1
006e	fe04	1a9e	5fbe	0704
006f	fe0e	19e6	600f	0769
0070	fe18	1930	605b	07d0
0071	fe21	187d	60a3	083a
0072	fe2b	17cb	60e7	08a7
0073	fe35	171c	6126	0916
0074	fe3f	166f	6160	0988
0075	fe49	15c5	6196	09fc
0076	fe53	151d	61c7	0a72
0077	fe5d	1477	61f3	0aeb
0078	fe66	13d3	621a	0b67
0079	fe70	1332	623d	0be5
007a	fe7a	1293	625b	0c66
007b	fe83	11f7	6274	0ce9
007c	fe8d	115e	6288	0d6f
007d	fe96	10c6	6298	0df7
007e	fea0	1031	62a2	0e82
007f	fea9	0f9f	62a8	0f0f

We claim:

1. A digital signal processor comprising:

a first processor;

a second processor;

a math unit which periodically accepts input signals representing data, performs an arithmetic operation on the data, and provides a digital value representing a result of the arithmetic operation; and

a data selection circuit which provides input data to the math unit, wherein the first and second processors alternate controlling the data selection circuit to select the input data provided to the math unit.

2. The digital signal processor of claim 1, further comprising:

a first memory operably connected to the first and second processors;

a second memory operably connected to the second processor, the second memory storing look-up table values, wherein the second processor controls processing of the look-up table values from the second memory and controls writing of results of the processing to the first memory.

3. The digital signal processor of claim 2, wherein:

the math unit performs a multiply-and-accumulate operation;

the second processor comprises a non-volatile memory containing interpolation coefficients; and

the second processors controls processing look-up table values by causing the data selection circuit to select a look-up table value from the second memory and an interpolation coefficient from the non-volatile memory as the input data provided to the math unit.

4. The digital signal processor of claim 2, further comprising a CODEC interface circuit, wherein the first processor executes a program for generating a sound amplitude value and provides the sound amplitude value to the CODEC interface.

5. The digital signal processor of claim 4, further comprising a third memory operably connected to the first processor, wherein the third memory contains the program executed by the first processor to generate the sound amplitude value.

6. The digital signal processor of claim 5, further comprising a timing circuit which periodically asserts a start signal which causes the first processor to begin processing the program, wherein the time between successive assertions of the start signal is sufficient for the first processor to complete execution of the program and generate the sound amplitude value.

7. The digital signal processor of claim 5, wherein:

each of the first and the third memories are partitioned into slots;

each slot in the third memory has a corresponding slot in the first memory; and

in each slot of the third memory, an instruction in the slot has a parameter which is given by a value in a corresponding slot of the first memory.

8. The digital signal processor of claim 7, further comprising a timing circuit which periodically asserts a start signal which causes the first processor to begin processing the program, wherein the time between successive assertions of the start signal is sufficient for the first processor to complete

execution of the program and generate the sound amplitude value.

9. The digital signal processor of claim 8, wherein the second processor controls writing to a slot in the first memory before the first processor executes an instruction in a corresponding slot in the third memory.

10. A digital signal processor comprising:

a foreground processor which executes a program to create a digital representation of sound;

a first memory operably connected to the foreground processor, the first memory storing parameters used by the foreground processor;

a second memory which stores look-up table values; and

a background processor operably connected to the first and second memories, the background processor operating in parallel with the foreground processor, wherein the background processor processes look-up table values and writes processed values to the first memory for use by the foreground processor.

11. The digital signal processor of claim 10, wherein the background processor processes the look-up table values by generating a value interpolated from the look-up table values.

12. The digital signal processor of claim 11, further comprising a third memory for storing interpolation coefficients, wherein the background processor processes the look-up table values by performing a sum of the products of a look-up table value from the second memory and an interpolation coefficient from the third memory.

13. The digital signal processor of claim 12, wherein:

the foreground processor, the background processor, and the first and third memories are formed in a first integrated circuit; and

the second memory comprises a dynamic random access memory which includes a second integrated circuit.

14. The digital signal processor of claim 13, wherein:

the first memory comprises a static random access memory having a single data port; and

the third memory comprises a read-only memory.

15. The digital signal processor of claim 10, wherein the function represented by the look-up table is a delay line.

16. The digital signal processor of claim 10, wherein the function represented by the look-up table is an ADSR curve.

17. A digital signal processor comprising:

a first processor;

a second processor;

a multiplexer operably connected to the first and second processors so that the first processor can select output signals of the multiplexer and the second processor can select the output signals of the multiplexer;

a flip-flop set operably connected to the multiplexer so that in response to assertion of a clock signal, the flip-flop set stores the output signals from the multiplexer and assert data signals which are determined by the signals stored; and

a math unit operably coupled to the flip-flop set to receive the data signals asserted by the flip-flop set,

wherein the clock signal has a period between assertions which is sufficient for the math unit to process the data signals received and generate valid output signals, and the first and second processors alternate selecting the output signals of the multiplexer so that the math unit alternates between receiving data signals selected by the first processor and receiving data signals selected by the second processor.

18. The digital signal processor of claim 17, further comprising a memory coupled to the multiplexer so that the memory supplies input signals to the multiplexer.

19. The digital signal processor of claim 18, wherein:

the memory comprises a single data port memory having an address bus coupled to the first and second processors;

the first processor provides address signals to the memory when the first processor selects the output signals of the multiplexer; and

the second processor provides address signals to the memory when the first processor is not providing address signals to the memory.

20. The digital signal processor of claim 19, wherein:

the memory has an access time which is less than about half the period of the clock signal; and

each processor is permitted at least one access to the memory during any two consecutive periods of the clock signal.

21. The digital signal processor of claim 20, further comprising an interface circuit coupled to the memory, wherein the interface circuit is permitted at least one access to the memory during any two consecutive periods of the clock signal.

22. The digital signal processor of claim 17, further comprising a white noise generator coupled the multiplexer so that the white noise generator supplies to the multiplexer input signals representing a pseudorandom number.

23. The digital signal processor of claim 17, further comprising:

a memory;

a CODEC interface circuit operably connected to the memory to enable external access of the memory through the CODEC interface;

a second flip-flop set coupled to the math unit, wherein the second flip-flop set periodically stores output signals from the math unit; and

means for connecting the second flip-flop set to the memory and writing data from the second flip-flop set to the memory.

24. The digital signal processor of claim 23, wherein the memory has a data port operably connected to the multiplexer so that the memory provides input signals to the multiplexer.

25. The digital signal processor of claim 17, wherein the math unit comprises a multiply-and-accumulate circuit which processes data signals representing three values A, B, and C and generates output signals representing a sum of the value C with the a product of the values A and B.

26. A programmable digital signal processor, comprising:

a control circuit which generates control signals for executing a plurality of different types of instructions; and

a hardware white noise generator operably connected to the control circuit, wherein the white noise generator produces a pseudorandom sequence of digital signals in response to control signals for executing instructions of a first type.

27. A method for generating a series of digital values representing sound amplitudes, comprising the steps of:

executing a series of instructions which generates a sound amplitude value from parameters stored in a first memory, wherein execution of an instruction in the series causes a math unit to perform an arithmetic operation on a parameter from the first memory; and

executing a series of tasks which generates parameters for use by the series of instructions, wherein execution of a task from the series of tasks comprises the steps of:

reading a value from a second memory;

asserting to the math unit a signal indicating the value read from the second memory;

performing, with the math unit, an arithmetic operation on the value from the second memory to generate a resultant value, wherein the performance of the arithmetic operation on the value from the second memory begins immediately after completion of the arithmetic operation on the parameter so that the math unit operates continuously and without pipeline delays; and

writing the resultant value to the first memory for use by the series of instructions.

28. The method of claim 27, further comprising the steps of:

partitioning the series of instructions, the first memory, and the series of tasks into a plurality of slots, wherein each slot in series of instructions corresponds to a slot in the first memory and a slot in the series of tasks, each slot in the series of instructions uses a parameter from a corresponding slot of the first memory, and each slot in the series of tasks writes a value to a corresponding slot in the first memory; and

for each slot in the series of instructions, executing tasks in a corresponding slot of the series of tasks before executing instructions in the slot of the series of instructions.

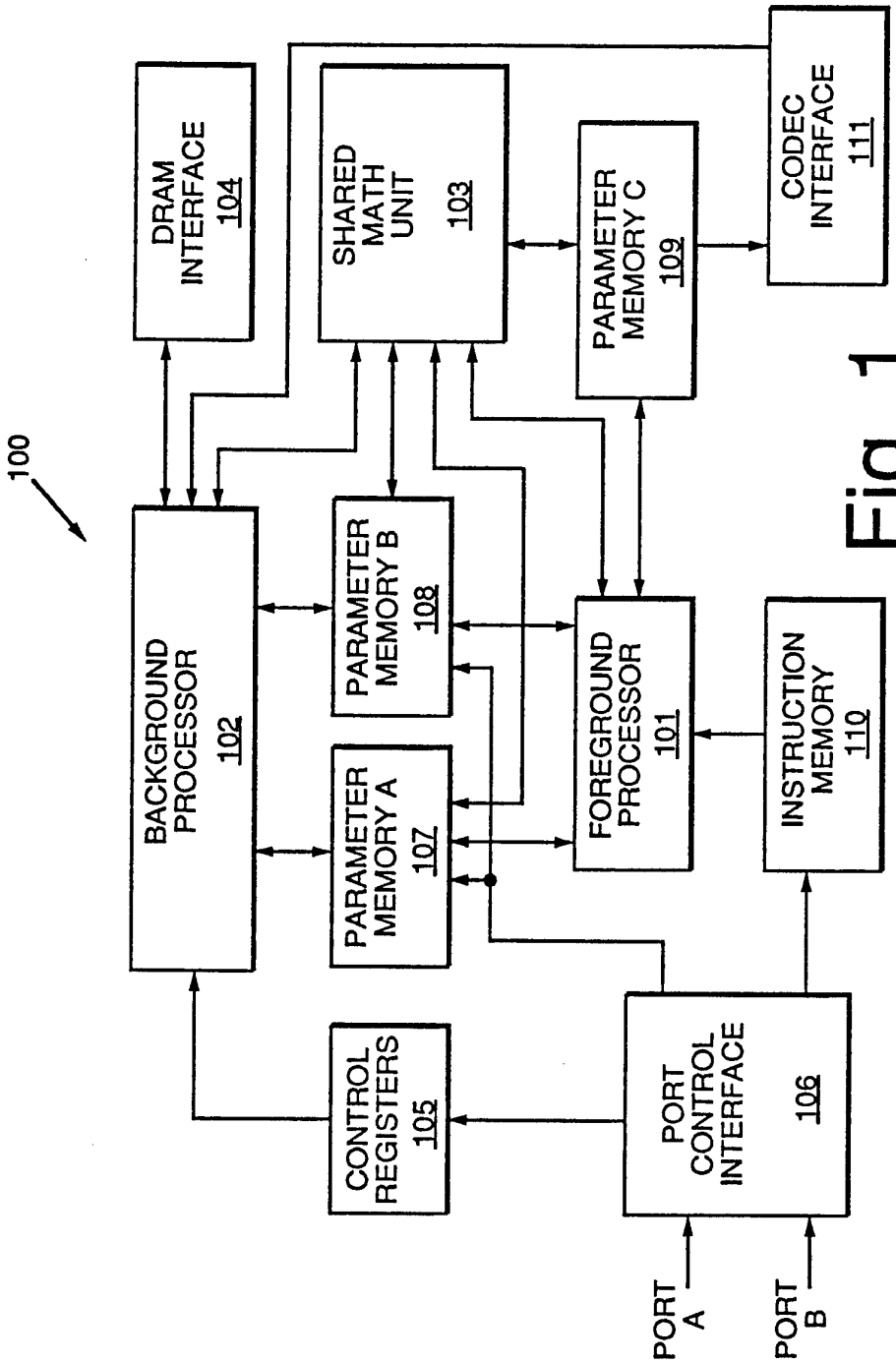

100
DRAM INTERFACE 104
SHARED MATH UNIT 103
PARAMETER MEMORY C 109
CODEC INTERFACE 111
BACKGROUND PROCESSOR 102
PARAMETER MEMORY B 108
PARAMETER MEMORY A 107
FOREGROUND PROCESSOR 101
INSTRUCTION MEMORY 110
CONTROL REGISTERS 105
PORT CONTROL INTERFACE 106
PORT A
PORT B

Fig. 1

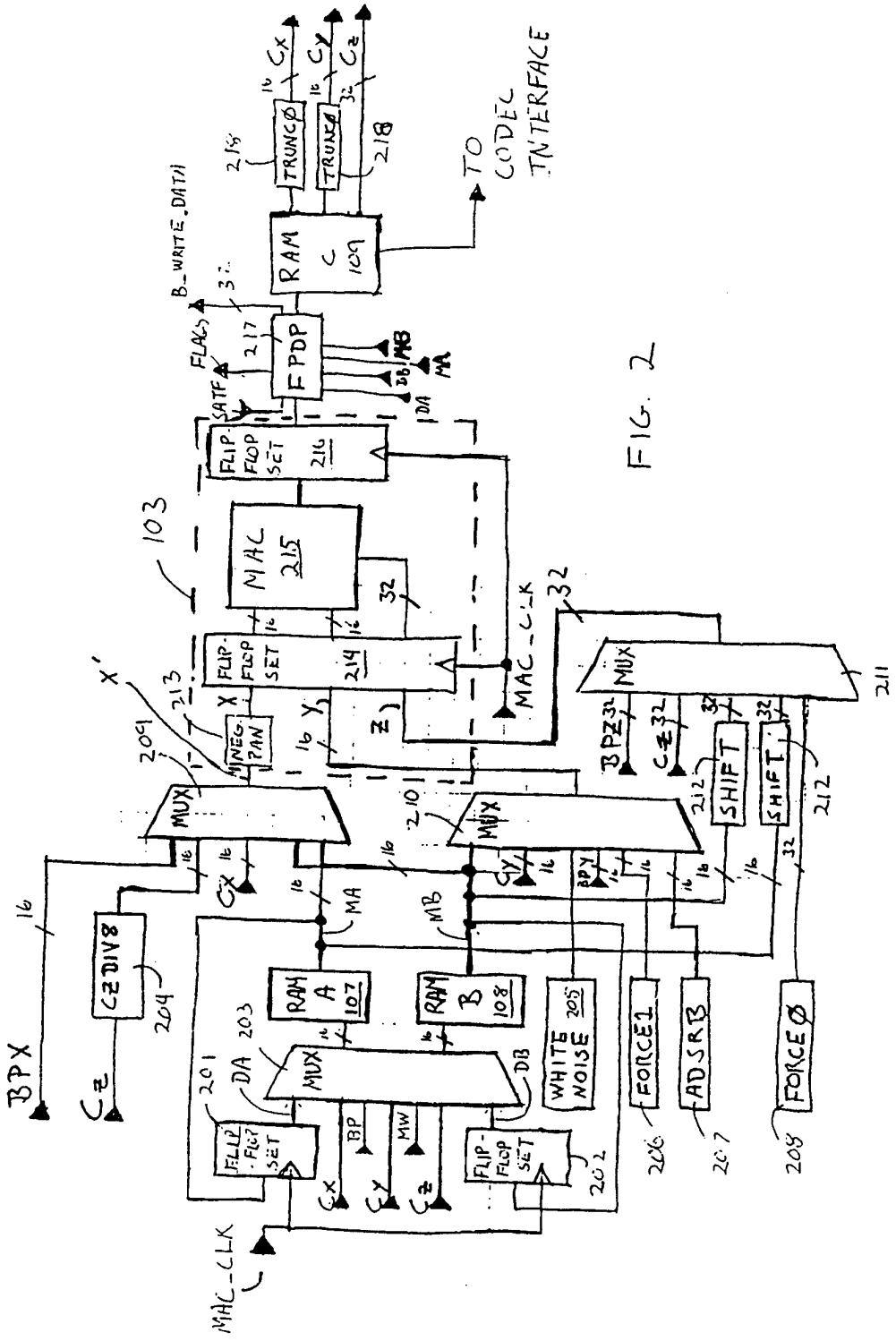

TO CODEC INTERFACE
B_WRITE_DATA
RAM C 109
TRUNC Ø
TRUNC 4
FPDP
FLAGS
SATF
FLIP-FLOP SET
MAC 215
MAC_CLK
MUX
NEG PAN
RAM A 107
RAM B 108
WHITE NOISE 205
FORCE 1
ADSR 3
FORCE Ø
SHIFT
BPX
BPZ
BPY
C2 DIV 8
FLIP-FLOP SET
MAC_CLK

FIG. 2

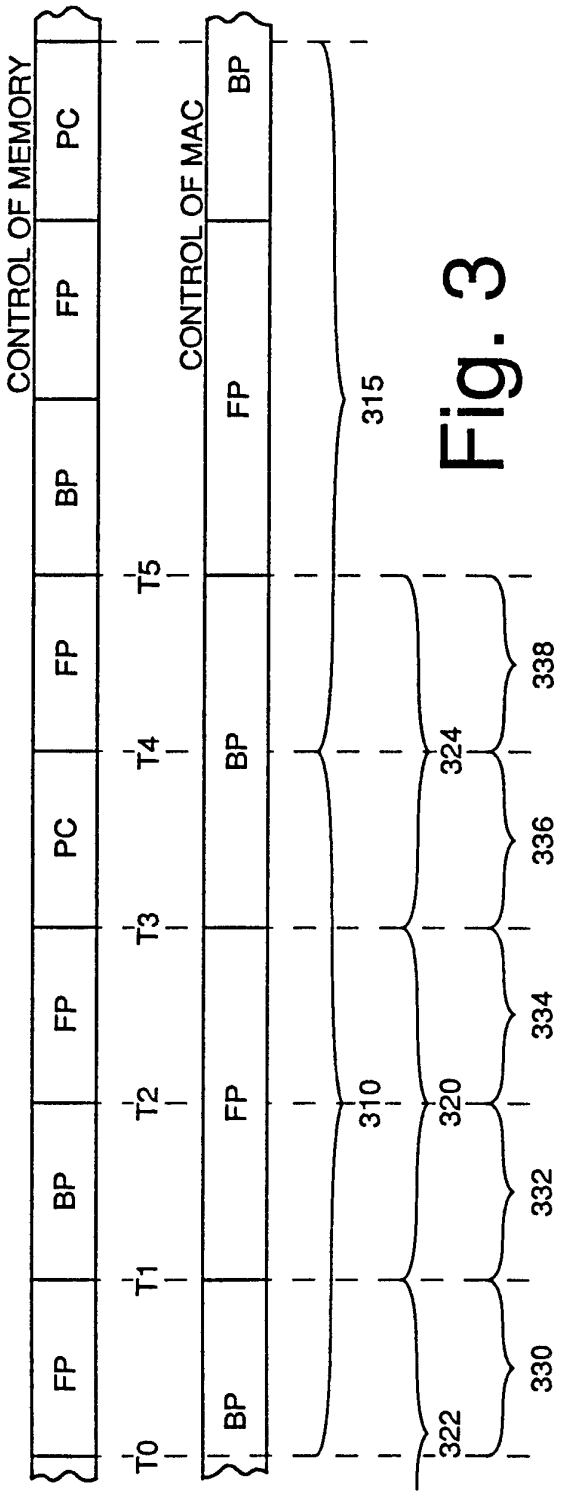

CONTROL OF MEMORY
CONTROL OF MAC
FP
BP
PC
T0
T1
T2
T3
T4
T5
315
310
320
322
324
330
332
334
336
338

Fig. 3

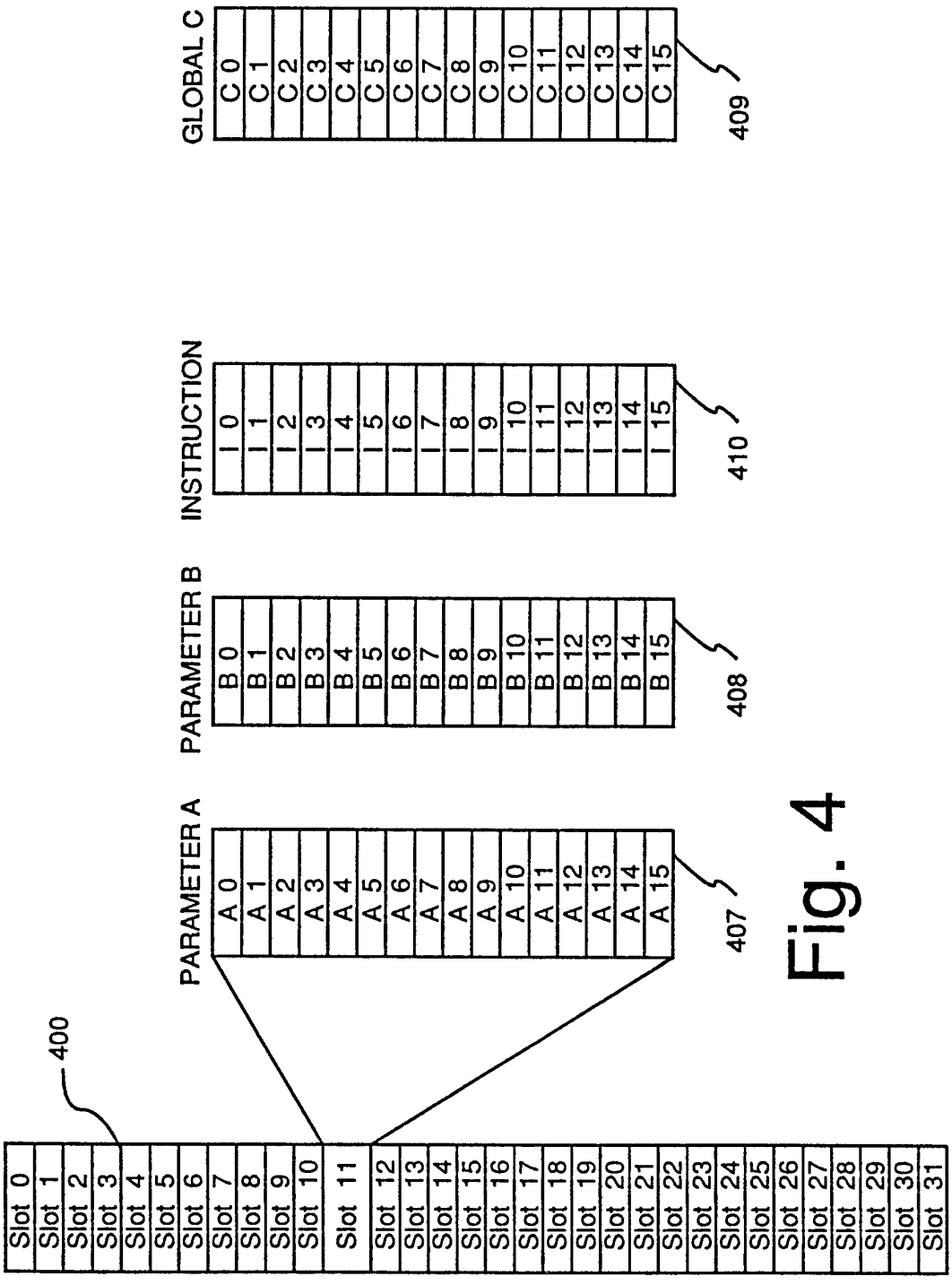

GLOBAL C
C 0
C 1
C 2
C 3
C 4
C 5
C 6
C 7
C 8
C 9
C 10
C 11
C 12
C 13
C 14
C 15
409
INSTRUCTION
I 0
I 1
I 2
I 3
I 4
I 5
I 6
I 7
I 8
I 9
I 10
I 11
I 12
I 13
I 14
I 15
410
PARAMETER B
B 0
B 1
B 2
B 3
B 4
B 5
B 6
B 7
B 8
B 9
B 10
B 11
B 12
B 13
B 14
B 15
408
PARAMETER A
A 0
A 1
A 2
A 3
A 4
A 5
A 6
A 7
A 8
A 9
A 10
A 11
A 12
A 13
A 14
A 15
407
400
Slot 0
Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
Slot 7
Slot 8
Slot 9
Slot 10
Slot 11
Slot 12
Slot 13
Slot 14
Slot 15
Slot 16
Slot 17
Slot 18
Slot 19
Slot 20
Slot 21
Slot 22
Slot 23
Slot 24
Slot 25
Slot 26
Slot 27
Slot 28
Slot 29
Slot 30
Slot 31

Fig. 4

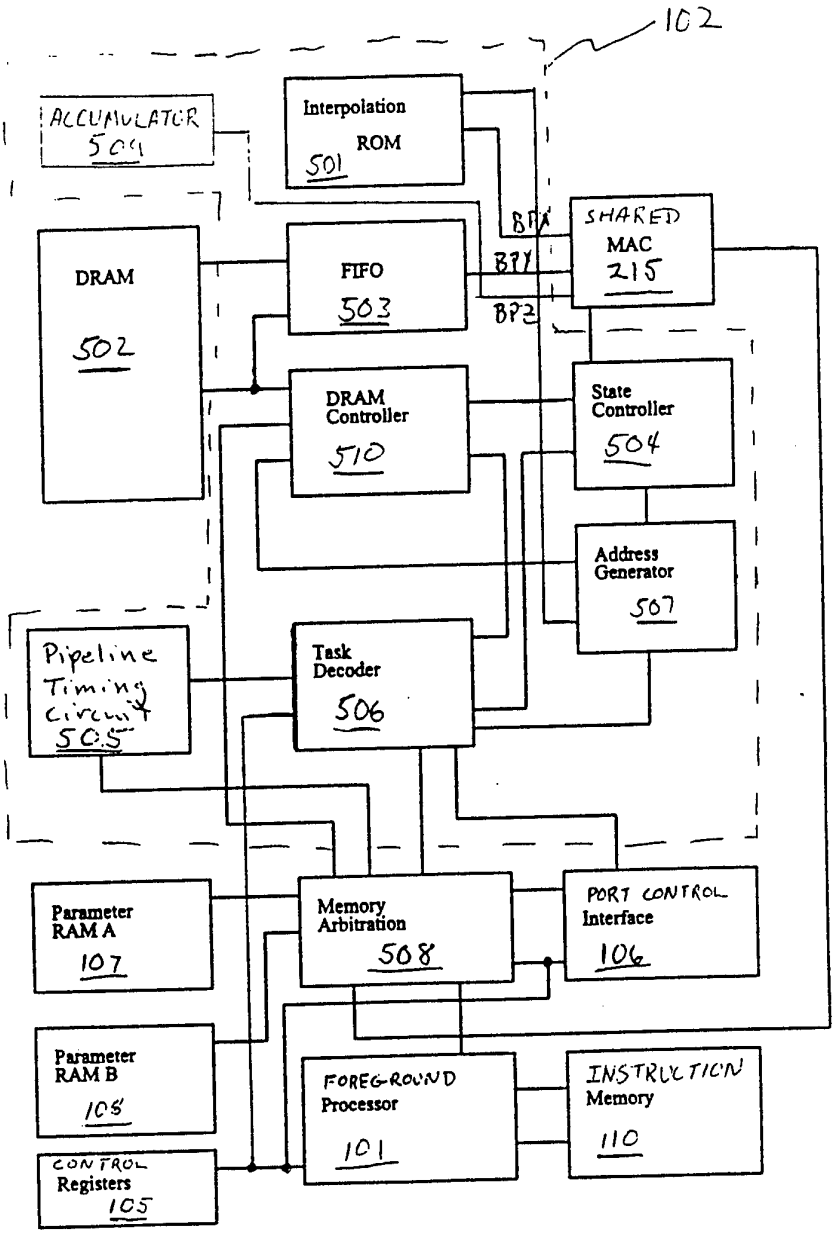

102
ACCUMULATOR
509
Interpolation
ROM
501
SHARED
MAC
215
DRAM
502
FIFO
503
DRAM
Controller
510
State
Controller
504
Address
Generator
507
Pipeline
Timing
Circuit
505
Task
Decoder
506
Parameter
RAM A
107
Memory
Arbitration
508
PORT CONTROL
Interface
106
Parameter
RAM B
108
FOREGROUND
Processor
101
INSTRUCTION
Memory
110
CONTROL
Registers
105

FIG. 5

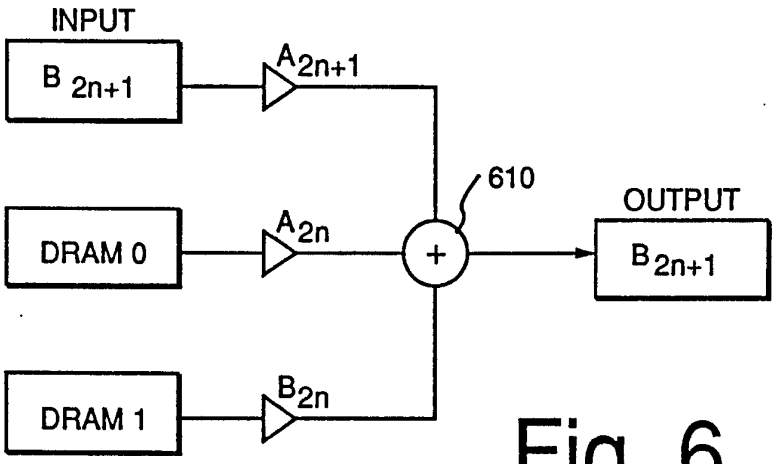


Fig. 6

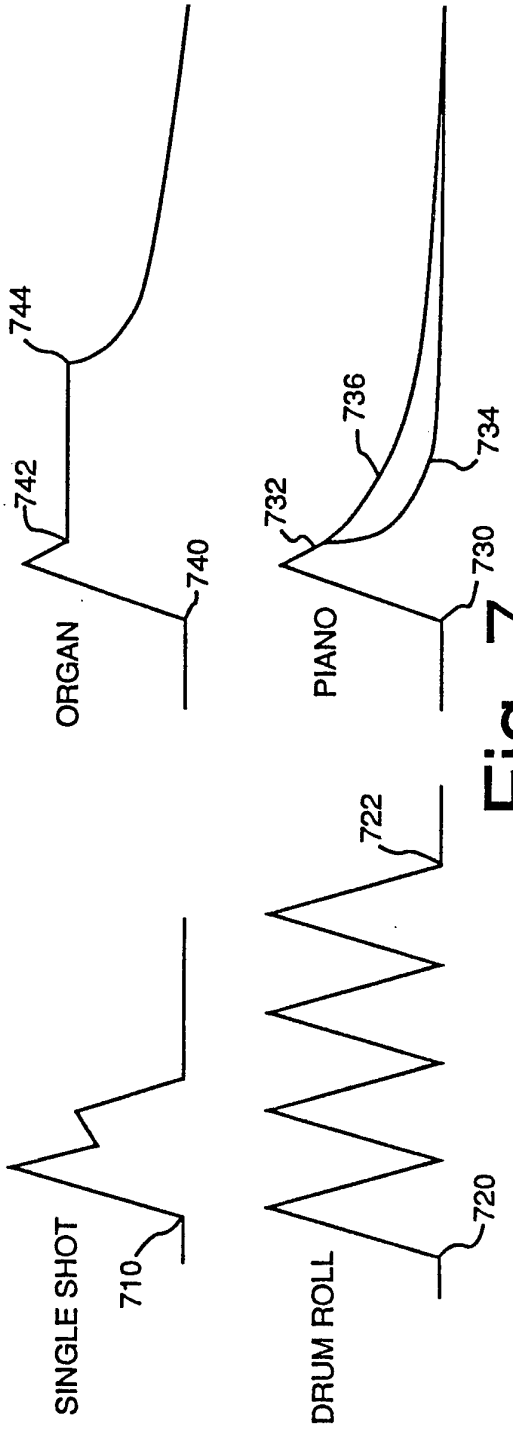

SINGLE SHOT
710
DRUM ROLL
720
722
ORGAN
740
742
744
PIANO
730
732
734
736

Fig. 7

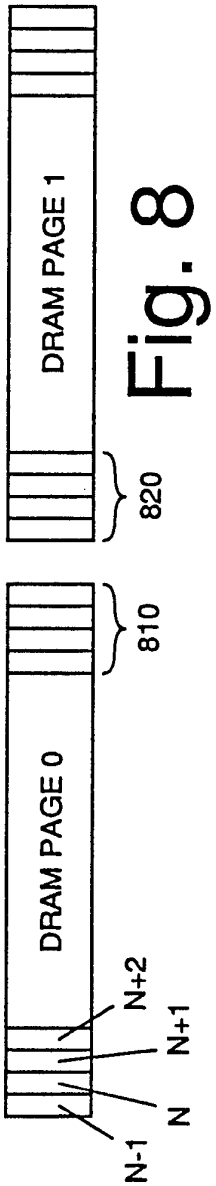

DRAM PAGE 0
DRAM PAGE 1
810
820
N-1
N
N+1
N+2

Fig. 8

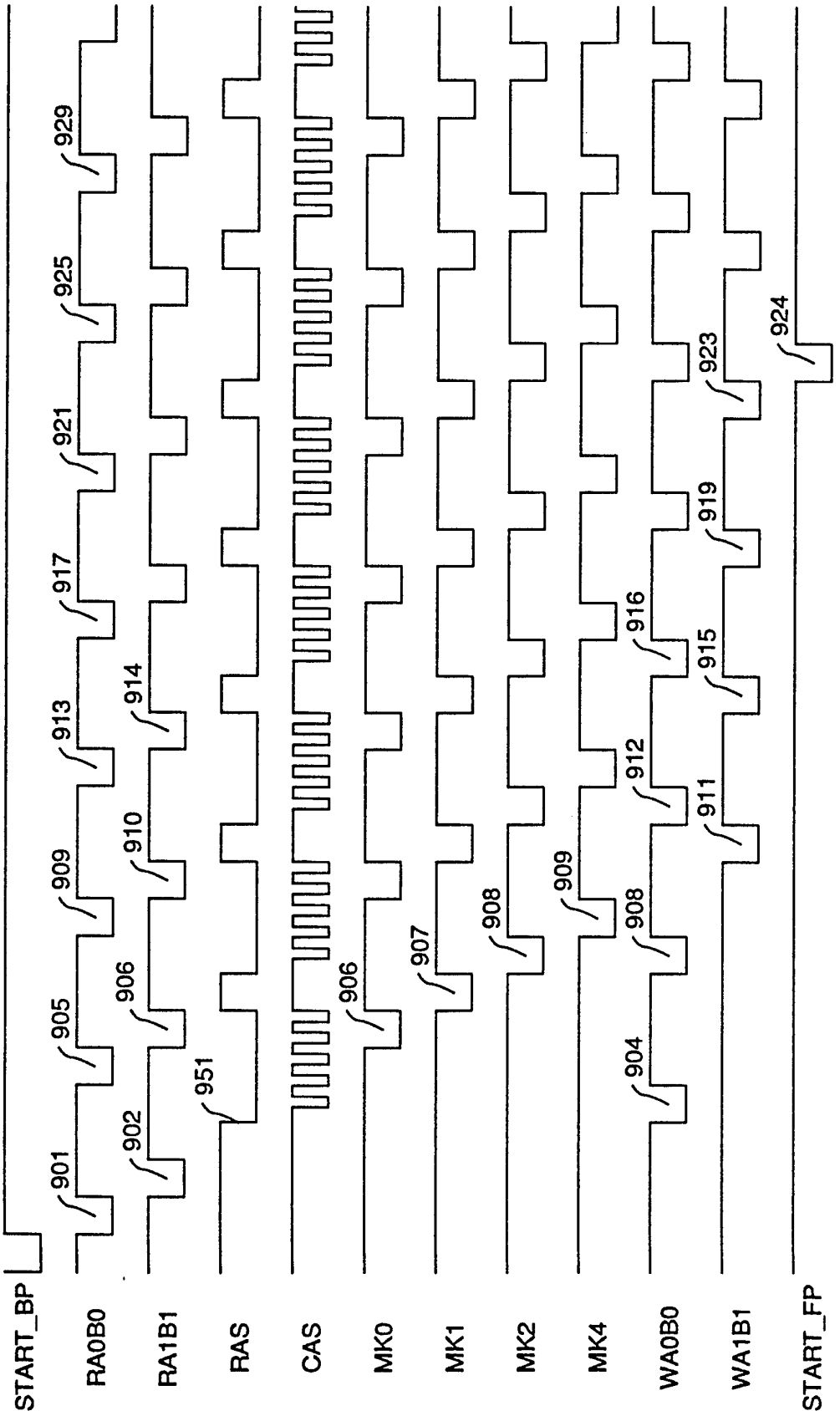

START_BP
RA0B0
RA1B1
RAS
CAS
MK0
MK1
MK2
MK4
WA0B0
WA1B1
START_FP
901
902
951
905
906
906
907
908
909
904
908
909
910
911
912
913
914
915
916
917
919
921
923
924
925
929

Fig. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/04354

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 7/38
US CL :364/736, 723, 717

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 364/736, 723, 717;84/602

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

NONE

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X - Y	US, A, 4,490,807 (CHEVILLAT ET ALL) 25 DECEMBER 1984, SEE ABSTRACT AND FIG. 2	1-9 ------ 17-22 & 23 25
X - Y	US, A, 4,736,333 (MEAD ET AL) 05 APRIL 1988, SEE COLUMN 6, LINES 6-17 AND FIG. 9	26 - 22
Y	US, A, 5,005,150 (DENT ET AL) 02 APRIL 1991, SEE COLUMN 4-7 AND FIGS. 4A AND 4B	17-22 &23-25
A	US, A, 4,327,419 (DEUTSCH ET AL) 27 APRIL 1982, SEE THE ENTIRE DOCUMENT	1-28

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 17 MAY 1995	Date of mailing of the international search report 07AUG 1995
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer D. H. MALZAHN Telephone No. (703) 305-9762

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/04354

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 4,507,728 (SAKAMOTO ET AL) 26 MARCH 1985, SEE THE ENTIRE DOCUMENT	1-28
A	US, A, 4,901,230 (CHEN ET AL) 13 FEBRUARY 1990, SEE THE ENTIRE DOCUMENT	1-28
A	US, A 4,953,437 (STARKEY) 04 SEPTEMBER 1990, SEE THE ENTIRE DOCUMENT	1-28
A	US, A, 4,957,552 (IWASE) 18 SEPTEMBER 1990, SEE THE ENTIRE DOCUMENT	1-28
A	US, A , 5,204,828 (KOHN) 20 APRIL 1993, SEE THE ENTIRE DOCUMENT	1-28
A	US, A, 5,241,492 (GIRARDEAU, JR) 31 AUGUST 1993, SEE THE ENTIRE DOCUMENT	1-28
A	US, A, 5,260,508 (BRUTI ET AL) 09 NOVEMBER 1993, SEE THE ENTIRE DOCUMENT	1-28