TEXAS INSTRUMENTS

TAS3108/TAS3108IA Audio DSP Instruction Set

Reference Guide

July 2007 ***DAV Digital Audio/Speaker***

SLEU067A

TAS3108/TAS3108IA Audio DSP Instruction Set

Reference Guide

Literature Number: SLEU067A
September 2006–Revised July 2007

TEXAS INSTRUMENTS

Contents

List of Figures

List of Tables

TAS3108/TAS3108IA Audio DSP Instruction Set

1 Introduction

This document describes the TAS3108/TAS3108IA architecture and includes information on programming the internal audio DSP using assembly language instructions. For electrical, performance, mechanical, and other data on TAS3108/TAS3108IA, see the TAS3108/TAS3108IA Audio DSP data sheet, TI literature number SLES152.

The TAS3108 is composed of two processing architectures. The 8051 microprocessor is responsible for coordinating all of the control and interface operations of the TAS3108. The audio DSP performs the majority of the audio processing. Development of the arithmetic code running in the audio DSP is performed with the audio DSP assembler or other software tool set as specified by Texas Instruments. Software developed for the 8051 is normally done in C-code, using the commercially available Keil development tools.

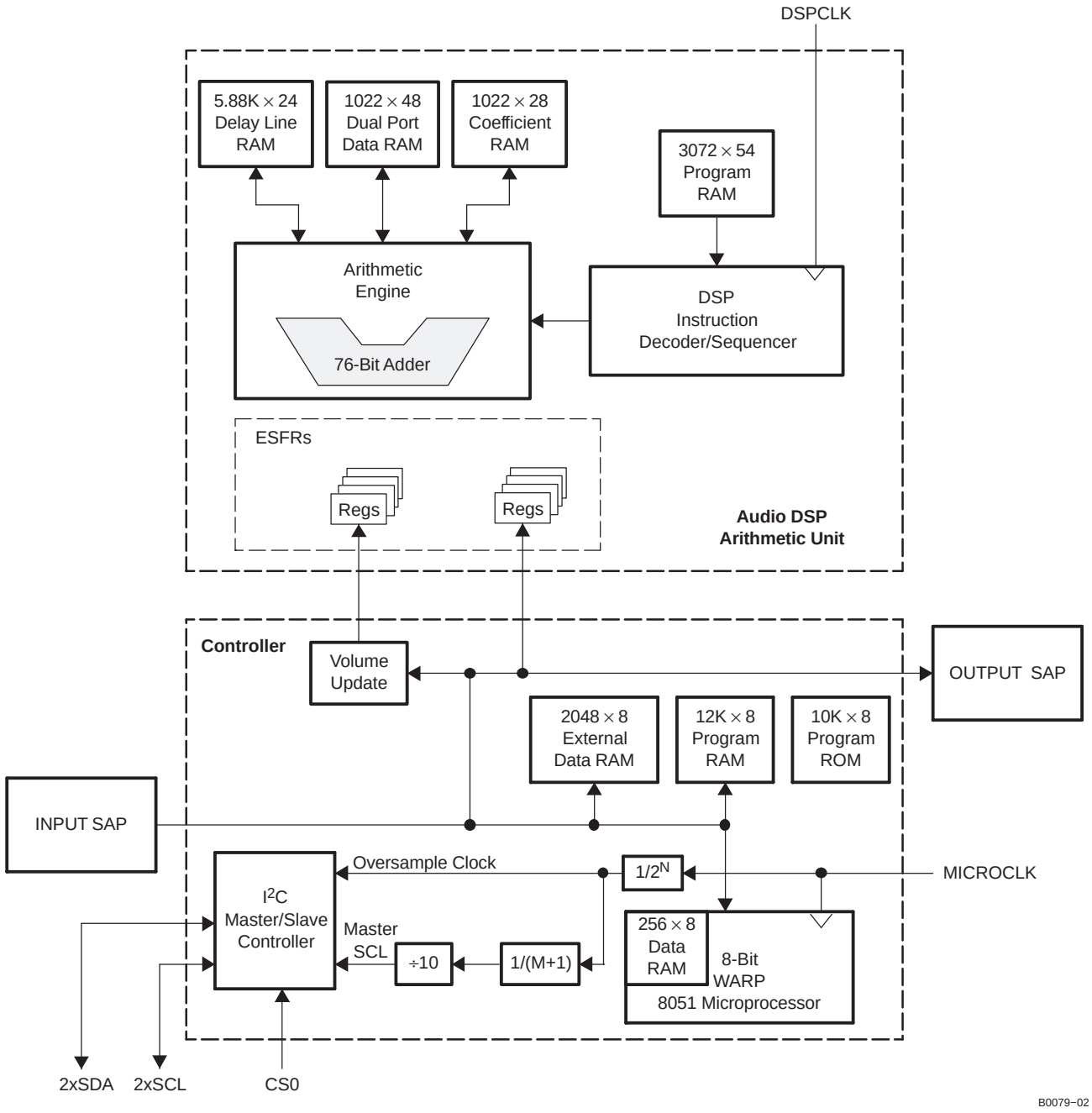


Figure 1. TAS3108 Functional Block Diagram

2 8051 8-Bit Microprocessor

The 8051 is a high-performance version of the popular 8051 8-bit microprocessor. Whereas the standard part operates with a 12-clock-period machine cycle, the 8051 embedded in the TAS3108 uses a two-clock-period machine cycle to facilitate particularly fast and/or low-power embedded solutions.

The ratio of two clock cycles to one machine cycle is constant across the instruction set and across all addressing modes, to maintain instruction execution time compatibility with other devices. The design is software-compatible with industry standard discrete devices, having all their core features and the additional features corresponding to standard 8051/8031/80C51BH/80C31BH/87C51 parts.

The 8051 can address 12K bytes of program memory, 256 bytes of internal data memory, and 2048 bytes of external data memory.

The 8051 also features an expanded 14-source interrupt controller, which supports the five standard internal interrupt sources and nine uncommitted interrupt/acknowledge signal pairs for optional external interrupt source registers. The 8051 features three 16-bit timer/counters.

The 8051 addresses three separate memory stores: program memory, internal data memory (used for storing register bank, stack and scratch pad data), and external memory.

All three memories are logically independent of each other and are accessed through their own interface signals. External data memory can only be accessed by software using dedicated MOVX instructions.

The special function registers (SFRs), including the stack pointer, are incorporated within the core. Additional SFRs can be added to the design using the internal data memory interface together with the dedicated external special function register (ESFR) read address bus (SFRSA), write strobe (SFRWE), and data input port (ESFRDI).

The whole 256 bytes of internal data memory address space is accessible using indirect addressing instructions (including stack operations), but only the lower 128 bytes are accessible using direct addressing. The upper 128 bytes of direct-address data memory space are used to access SFRs.

2.1 Special Function Registers (SFRs)

All I/O and timer/counter operations for the 8051 are accessed via SFRs. These registers occupy direct internal data memory space locations in the range from 80h to FFh. Their names and addresses are given in the *TAS3108/TAS3108IA Microprocessor Reference Guide*, TI literature number SLEU076.

2.2 External Special Function Registers (ESFRs)

External data memory occupies a 64K address space. This space contains the ESFRs at addresses between 80h and FFh. The ESFR permit access and control of the hardware features and internal interfaces of the TAS3108. These include interfaces between the 8051 microprocessor to:

- I2C controller
- Delay memory
- Volume Update Block (VUB)
- DSP
- GPIO
- Treble/bass

DSP-to-microprocessor ESFR interfaces are shown in Table 1. See the *TAS3108/TAS3108IA Microprocessor Reference Guide* for a complete list of the ESFRs, along with detailed information regarding their use.

Table 1. DSP and VUB ESFR Map

ESFR	Mapped_to	# Bits	Direction	Connecting Block	Register Type	Description
A6	vol_mode_i_t	2	Out	Volume Update Block	2-bit asynchronous rstz positive-edge triggered	Volume slew rate 0 = 2048 1 = 4096 2 = 8192
A7	volume_index_i_t	3	Out	Volume Update Block	3-bit asynchronous rstz positive-edge triggered	Audio channel select 0 = Ch1 1 = Ch2 ... 7 = Ch8

Table 1. DSP and VUB ESFR Map (continued)

ESFR	Mapped_to	# Bits	Direction	Connecting Block	Register Type	Description
A9	volume_data_i_t	8	Out	Volume Update Block	8-bit asynchronous rstz positive-edge triggered	Volume coefficient (28 bits)
AA	volume_data_i_t	8	Out	Volume Update Block	8-bit asynchronous rstz positive-edge triggered	
AB	volume_data_i_t	8	Out	Volume Update Block	8-bit asynchronous rstz positive-edge triggered	
AC	volume_data_i_t	4	Out	Volume Update Block	4-bit asynchronous rstz positive-edge triggered	
AD	To_micro_i[7:0]	8	In	DSP	NO REG – direct input	Data bus from DSP to the microprocessor
AE	To_micro_i[15:8]	8	In	DSP	NO REG – direct input	
AF	To_micro_i[23:16]	8	In	DSP	NO REG – direct input	
B1	To_micro_i[31:24]	8	In	DSP	NO REG – direct input	
B2	To_micro_i[39:32]	8	In	DSP	NO REG – direct input	
D6	To_micro_i[47:40]	8	In	DSP	NO REG – direct input	
D7	To_micro_i[53:48]	8	In	DSP	NO REG – direct input	
B3	Data_to_DSP_o[7:0]	8	Out	DSP	8-bit asynchronous rstz positive-edge triggered Reset low	54-bit data bus from microprocessor to the DSP program memory
B4	Data_to_DSP_o[15:8]	8	Out	DSP	8-bit asynchronous rstz positive-edge triggered	
B5	Data_to_DSP_o[23:16]	8	Out	DSP	8-bit asynchronous rstz positive-edge triggered	
B6	Data_to_DSP_o[31:24]	8	Out	DSP	8-bit asynchronous rstz positive-edge triggered Reset low	
B7	Data_to_DSP_o[39:32]	8	Out	DSP	8-bit asynchronous rstz positive-edge triggered Reset low	
B9	Data_to_DSP_o[47:40]	8	Out	DSP	8-bit asynchronous rstz positive-edge triggered Reset low	
BA	Data_to_DSP_o[53:48]	6	Out	DSP	8-bit asynchronous rstz positive-edge triggered Reset low	
BB	micro_addr_o[7:0]	8	Out	DSP	8-bit asynchronous rstz positive-edge triggered Reset low	Microprocessor uses these 16 bits to set DSP RAM and Micro I addresses.
BC	micro_addr_o[13:8]	8	Out	DSP	8-bit asynchronous rstz positive-edge triggered Reset low	Microprocessor uses these 16 bits to set DSP RAM and Micro I addresses. Bit 10 of the address selects between audio DSP coefficient and audio DSP data memory.
BD	Mode0_o	1	Out	DSP	1-bit asynchronous rstz positive-edge triggered Reset low	Miscellaneous signal for microprocessor-DSP communication. This is not a bit-addressable register, but contains bit data. The firmware must read in the data, mask the change, and write it back out.
BE	Mode3_o	1	Out	DSP	1-bit asynchronous rstz positive-edge triggered, Reset low	
BF	Mode4_o	1	Out	DSP	1-bit asynchronous rstz positive-edge triggered Reset low	
C1	C1 Mode5_o	1	Out	DSP	1-bit asynchronous rstz positive-edge triggered Reset low	Miscellaneous signal for microprocessor-DSP communication. This is not a bit-addressable register, but contains bit data. The firmware must read in the data, mask the change, and write it back out.

3 Arithmetic Processor Instructions

3.1 Overview

The arithmetic processor is a fixed-point computational engine consisting of an arithmetic unit and DATA and COEF memory blocks. The architecture is optimized for programming digital audio processing blocks such as IIR filters, FIR filters, volume control, tone controls, mixers, DRC, loudness, etc. The primary features are:

- Two-pipe parallel processing architecture
  - 48-bit data path with 76-bit Accumulator
  - Hardware single-cycle multiplier (28-bit $\times$ 48-bit)
  - Three 48-bit general-purpose data registers
  - One 28-bit coefficient register
  - Adder with 48-bit and 76-bit inputs
  - Shift right, Shift left
  - Bimodal clip
  - Log_2/$Alog_2$
  - Absolute value
  - Negation
  - Magnitude truncation
- Read/read/write single-cycle memory access
- Data input is 48-bit twos-complement multiplexed in from SAP immediately following FSYNC pulse (at LRCLK or f_S sample rate).
- Separate control for writing to delay memory
- Separate coefficient memory (28-bit) and data memory (48-bit)
- Linear feedback shift register (LFSR) in the instruction register functions as a random number generator.
- COEF RAM, DATA RAM, LFSR seed, program counter START value, and memory pointers are all mapped into the same memory space for convenient addressing by the microprocessor.

3.1.1 Data Format

The following shows the data-word structure of the arithmetic unit. Eight bits of overhead or guard bits are provided at the upper end of the 48-bit word, and eight bits of computational precision or noise bits are provided at the lower end of the 48-bit word. The incoming digital audio words are all positioned with the most significant bit abutting the 8-bit overhead/guard boundary. The sign bit, bit 39, indicates that all incoming audio samples are treated as signed data samples.

The arithmetic engine is a 48-bit (25.23 format) processor consisting of a general-purpose 76-bit arithmetic logic unit and function-specific arithmetic blocks. Multiply operations (excluding the function-specific arithmetic blocks) always involve 48-bit words and 28-bit coefficients (usually I2C programmable coefficients). If a group of products is to be added together, the 76-bit product of each multiplication is applied to a 76-bit adder, where a DSP-like multiply-accumulate (MAC) operation takes place. Biquad filter computations use the MAC operation to maintain precision in the intermediate computational stages.

To maximize the linear range of the 76-bit ALU, saturation logic is not used. In MAC computations, intermediate overflows are permitted, and it is assumed that subsequent terms in the computation flow correct the overflow condition.

The memory banks include a dual-port DATA RAM for storing intermediate results, a COEF RAM, and a fixed program ROM.

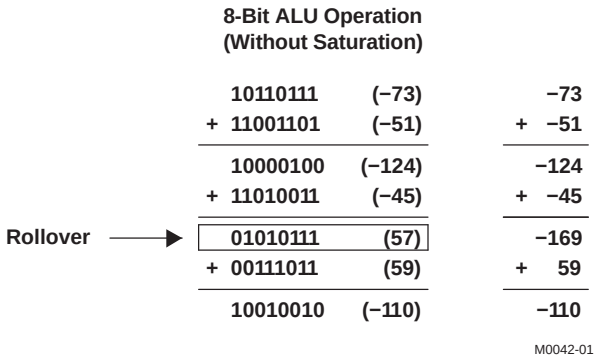


Figure 2. ALU Operation With Intermediate Overflow

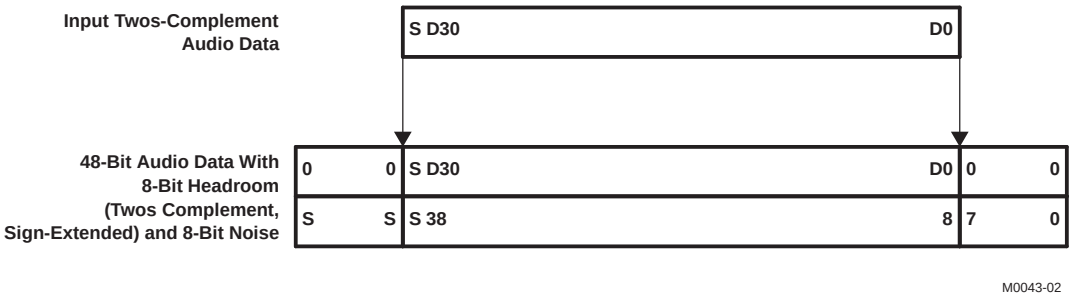


Figure 3. Alignment of Input Data With Data and Coefficient Memory

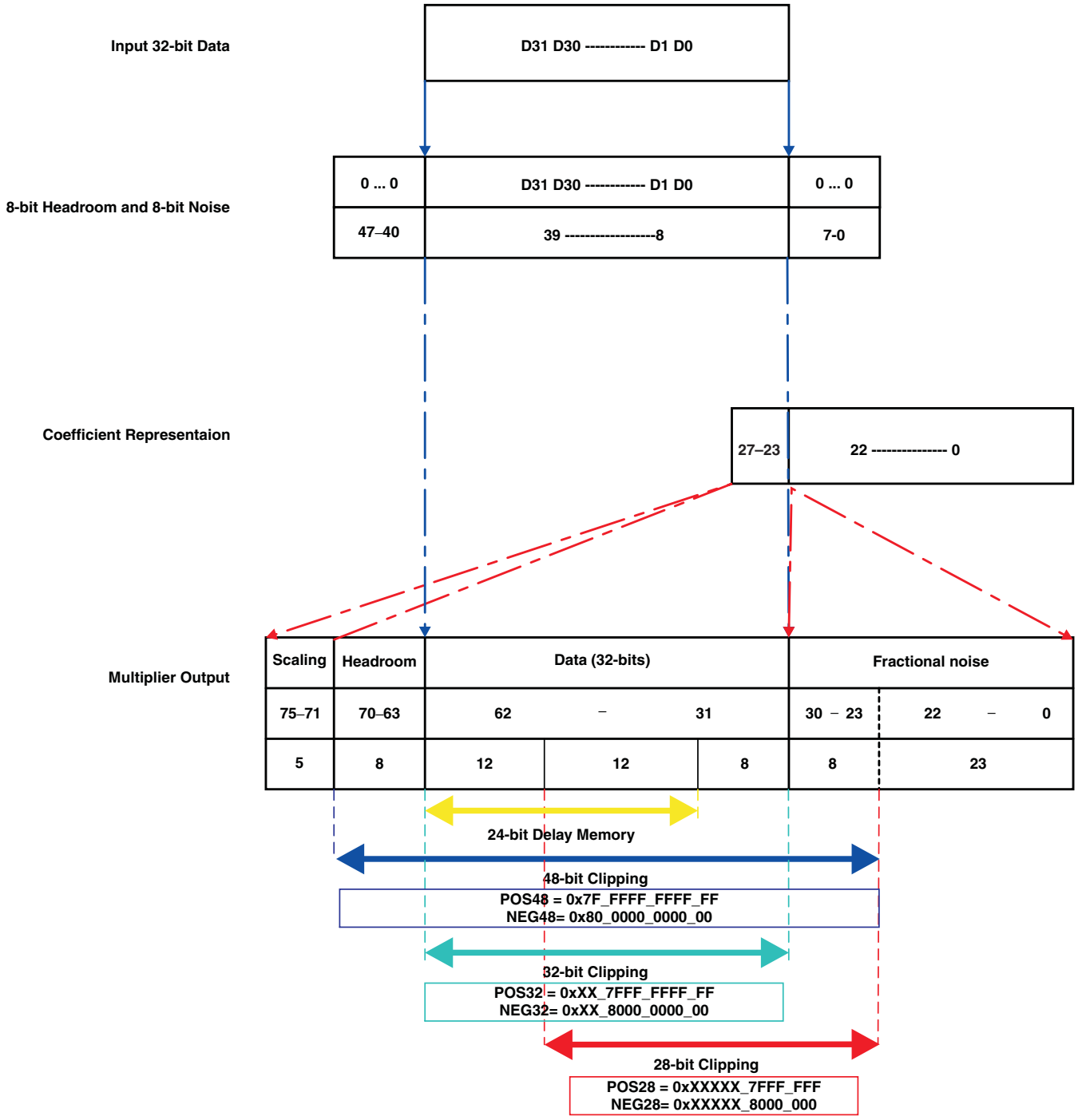


Figure 4. DSP Multiplier and Clipping Bit Alignment

3.2 Number Formats in TAS3108/TAS3108IA

The DAP is a 48-bit signed fixed-point arithmetic processing machine. The data is in twos-complement format, the most significant bit of the 48-bit data path is the sign bit, and the lower 47 bits are data bits. Mixer gain operations are implemented by multiplying a 48-bit signed data value by a 28-bit signed gain coefficient. The 76-bit signed output product is then truncated to a signed 48-bit number. Add operations are implemented by adding a 48-bit signed offset coefficient to a 48-bit signed data value. In most cases,

if the addition results in overflowing the 48-bit signed number format, saturation logic is used. This means that if the summation results in a positive number that is greater than 0x7FFF FFFF FFFF (the spaces are used to ease the reading of the hexadecimal number), the number is set to 0x7FFF FFFF FFFF. If the summation results in a negative number that is less than 0x8000 0000 0000, the number is set to 0x8000 0000 0000.

3.2.1 28-Bit 5.23 Number Format

All mixer gain coefficients are 28-bit coefficients using a 5.23 number format. Numbers formatted as 5.23 numbers have 5 bits to the left of the binary point and 23 bits to the right of the binary point. This is shown in Figure 5.

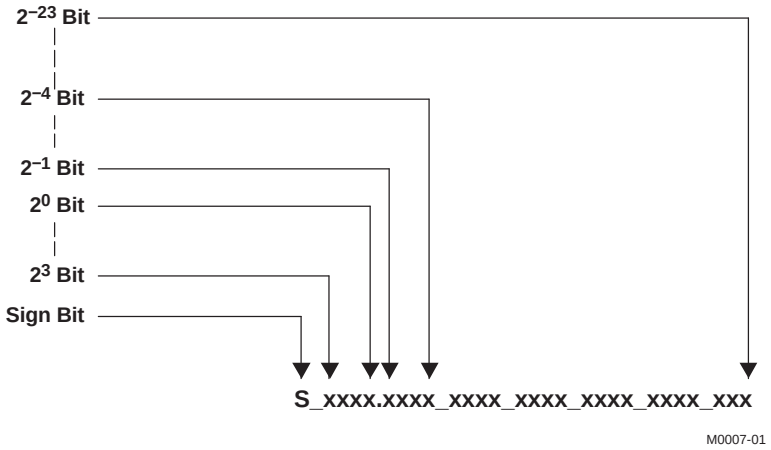


Figure 5. 5.23 Format

The decimal value of a 5.23 format number can be found by following the weighting shown in Figure 6. If the most significant bit is logic 0, the number is a positive number, and the weighting shown yields the correct number. If the most significant bit is a logic 1, then the number is a negative number. In this case, every bit must be inverted, a 1 added to the result, and then the weighting shown in Figure 6 applied to obtain the magnitude of the negative number.

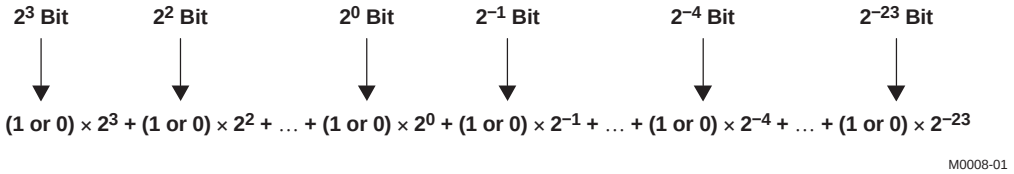


Figure 6. Conversion of a 5.23 Format Number to Decimal

Gain coefficients, entered via the I2C bus, must be entered as 32-bit binary numbers. The format of the 32-bit number (4-byte or 8-digit hexadecimal number) is shown in Figure 7.

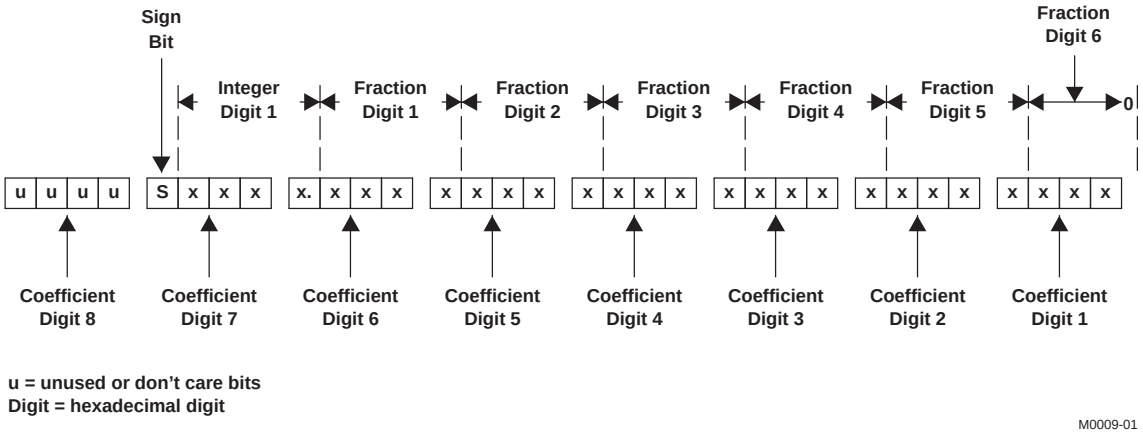


Figure 7. Alignment of 5.23 Coefficient in 32-Bit I2C Word

As Figure 7 shows, the hexadecimal (hex) value of the integer part of the gain coefficient cannot be concatenated with the hex value of the fractional part of the gain coefficient to form the 32-bit I2C coefficient. The reason is that the 28-bit coefficient contains five bits of integer, and thus the integer part of the coefficient occupies all of one hex digit and the most significant bit of the second hex digit. In the same way, the fractional part occupies the lower three bits of the second hex digit, and then occupies the other five hex digits (with the eighth digit being the zero-valued most-significant hex digit).

3.2.2 48-Bit 25.23 Number Format

All level adjustment and threshold coefficients are 48-bit coefficients using a 25.23 number format. Numbers formatted as 25.23 numbers have 25 bits to the left of the decimal point and 23 bits to the right of the decimal point. This is shown in Figure 8.

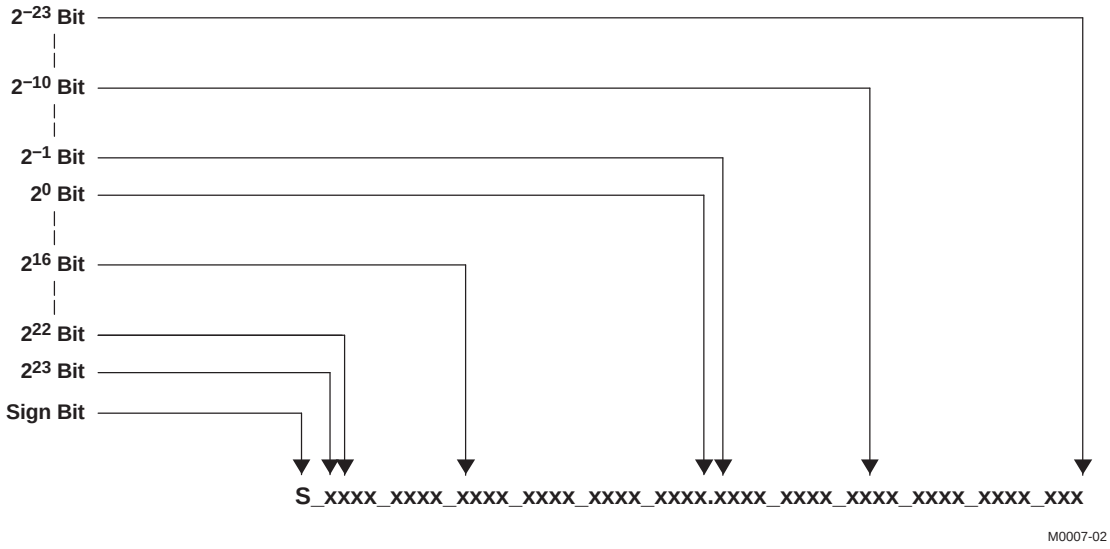


Figure 8. 25.23 Format

Figure 9 shows the derivation of the decimal value of a 48-bit 25.23 format number.

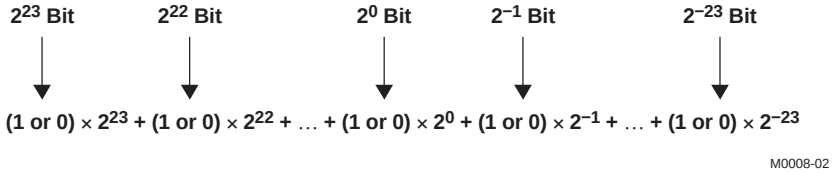


Figure 9. Conversion of a 25.23 Format Number to Decimal

Two 32-bit words must be sent over the I2C bus to download a level or threshold coefficient into the TAS3108/TAS3108IA. The alignment of the 48-bit, 25.23 formatted coefficient in the 8-byte (two 32-bit words) I2C word is shown in Figure 10.

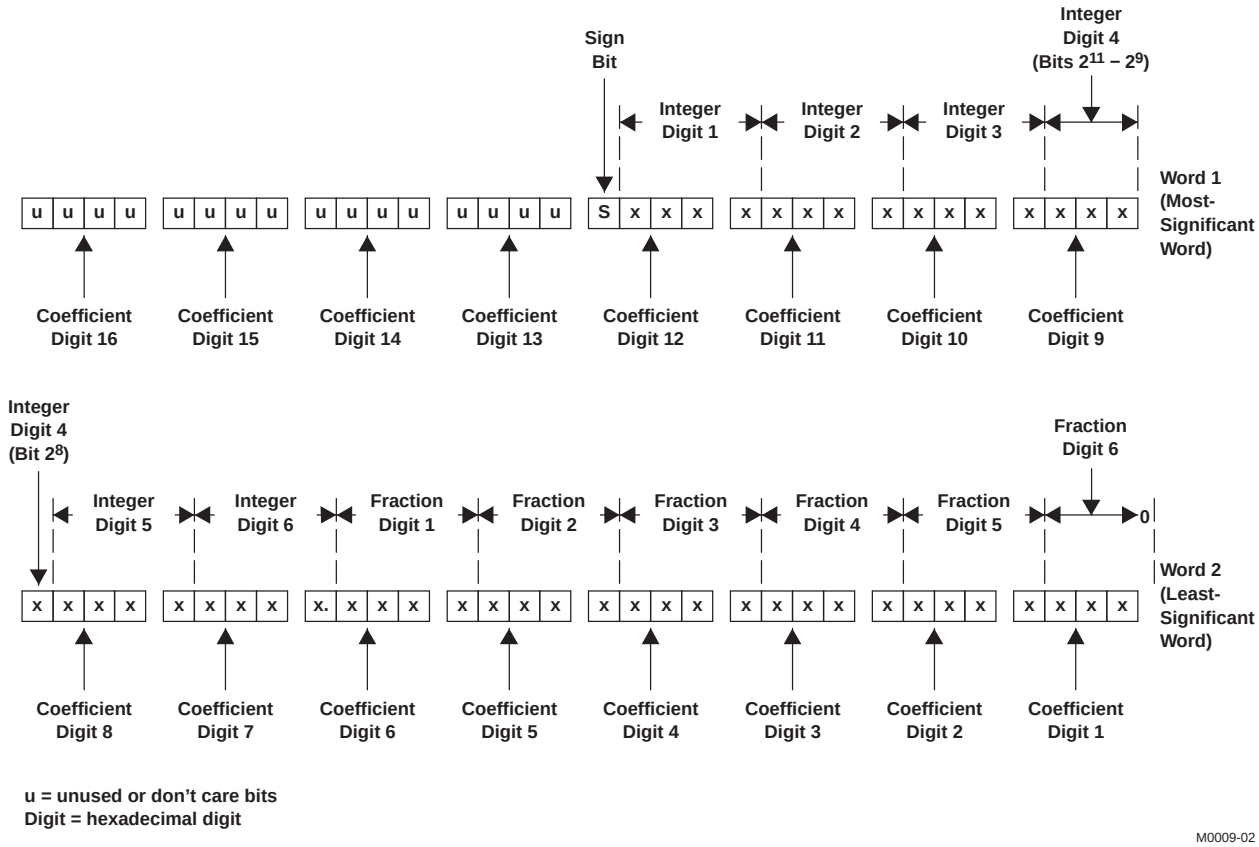


Figure 10. Alignment of 25.23 Coefficient in Two 32-Bit I2C Words

3.2.3 TAS3108/TAS3108IA Digital Audio Scaling

The TAS3108/TAS3108IA digital audio processing is designed so that noise produced by filter operations is maintained below the smallest signal amplitude of interest, as shown in Figure 11. This low noise level is achieved by increasing the precision of the signal representation substantially above the number of bits that are absolutely necessary to represent the input signal.

Additional precision, in the form of overflow bits, is used to permit the value of intermediate calculations to exceed the input precision without clipping. The TAS3108/TAS3108IA DAP achieves both of these important performance capabilities by using a high-performance digital audio processing architecture with a 48-bit data path (source from DATA RAM), 28-bit filter coefficients (sourced from COEF RAM), and a 76-bit Accumulator.

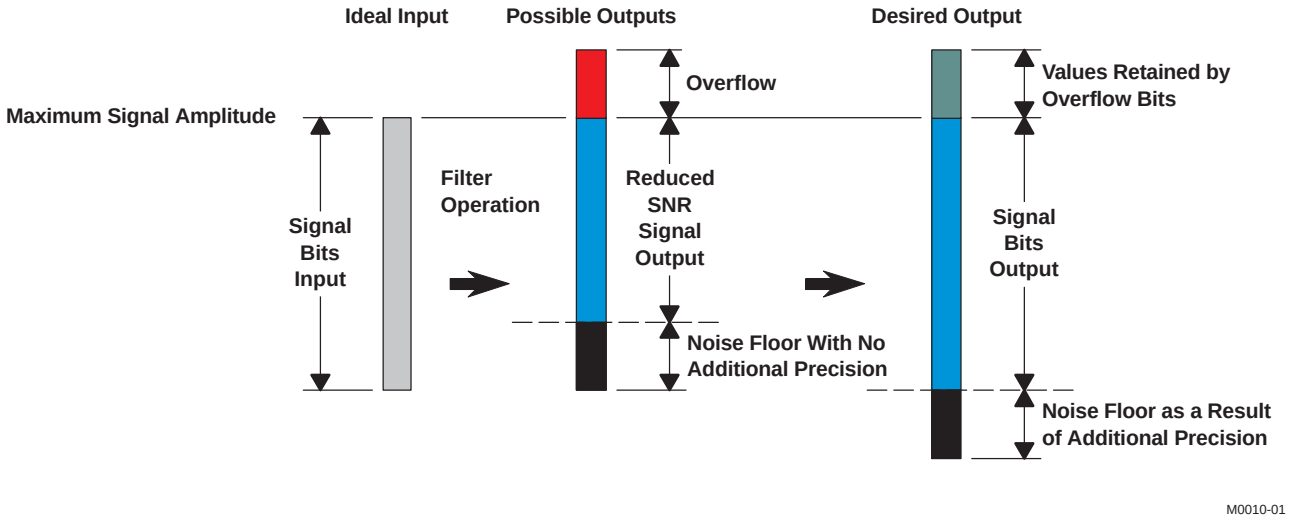


Figure 11. TAS3108/TAS3108IA Digital Audio Scaling

3.3 DSP Program Instruction 54-Bit Format

Each DSP instruction controls five independent operations, which can simultaneously load two operands from DATA memory and COEF (coefficient) memory, store the result into DATA or COEF memory, and perform two parallel arithmetic operations.

The DSP program counter (PC) is automatically set to 0 on startup and on completion of an LRCLK (f_S) frame. The PC is advanced by the DSP clock, whose frequency is dependent on the sample rate (f_S) and the serial data mode. For a standard I2S input at f_S = 48 kHz and MCLK = 12.288 MHz, the DSP clock runs about 135 MHz. This means that there are 2816 DSP cycles per sample available for processing.

Table 2 shows the format of the control data generated for a single 54-bit DSP instruction word.

Table 2. 54-Bit Audio DSP Instruction Word

Function	ALU 1st Stage	ALU 2nd Stage	DATA Memory Load		COEF Memory Load		Memory Store	
Name	ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3
Bit Field	$b_{53} – b_{49}$	$b_{48} – b_{42}$	$b_{41} – b_{37}$	$b_{36} – b_{27}$	$b_{26} – b_{24}$	$b_{23} – b_{14}$	$b_{13} – b_{10}$	$b_9 – b_0$

3.4 TAS3108/TAS3108IA DSP Processing Architecture

Figure 12 shows the TAS3108/TAS3108IA DSP audio processing architecture. The blocks with bold lines are registers that are clocked by the internal DSP clock. The maximum frequency of the DSP clock is 135 MHz. For more information, see the TAS3108/TAS3108IA Audio DSP data sheet, TI literature number SLES152.

The register elements in this architecture are:

- Data RAM – Normally used as temporary storage for audio data
- COEF RAM – Normally contains filter or gain coefficients that are multiplied by audio data
- Register B – Storage for data to be transformed by bit shifting, negation, absolute value, or pass through
- Register L – Storage for data to be transformed by log2, alog2, negation, absolute value, or pass through
- Register MD –Storage for 48-bit data to be multiplied
- Register MC – Storage for 28-bit data to be multiplied
- Register BR – Storage for result of operations on data in Register B
- Register LR – Storage for result of operations on data in Register L
- Register MR – Storage for result of multiplication of data contained in registers MD and MC
- ACC – Accumulator register
- Register DI – Data input from serial audio port (SAP), DATA RAM memory mapped so that lower three LSBs decode the input channels as shown in Figure 12
- Register DO1, DO2, ..., DO8 – An 8-register block for outputting data to the output serial audio port
- Register VOL – Used for implementing a volume control that uses the internal hardware Volume Update Block
- Register LFS – Lower two LSBs provide the programmer with a way to dither the audio (used in conjunction with Register B)
- Register DLYI – Input register used as an interface to the Delay RAM which provides a mechanism for delaying the audio data
- Register DLYO – Delay RAM output register used to store delayed audio data

A number of memory locations are reserved for specific purposes:

- 0x301 – LFSR
- 0x2BC – LFSR seed
- 0x2BD – PC START
- 0x2BE – DREG0 (to microprocessor)

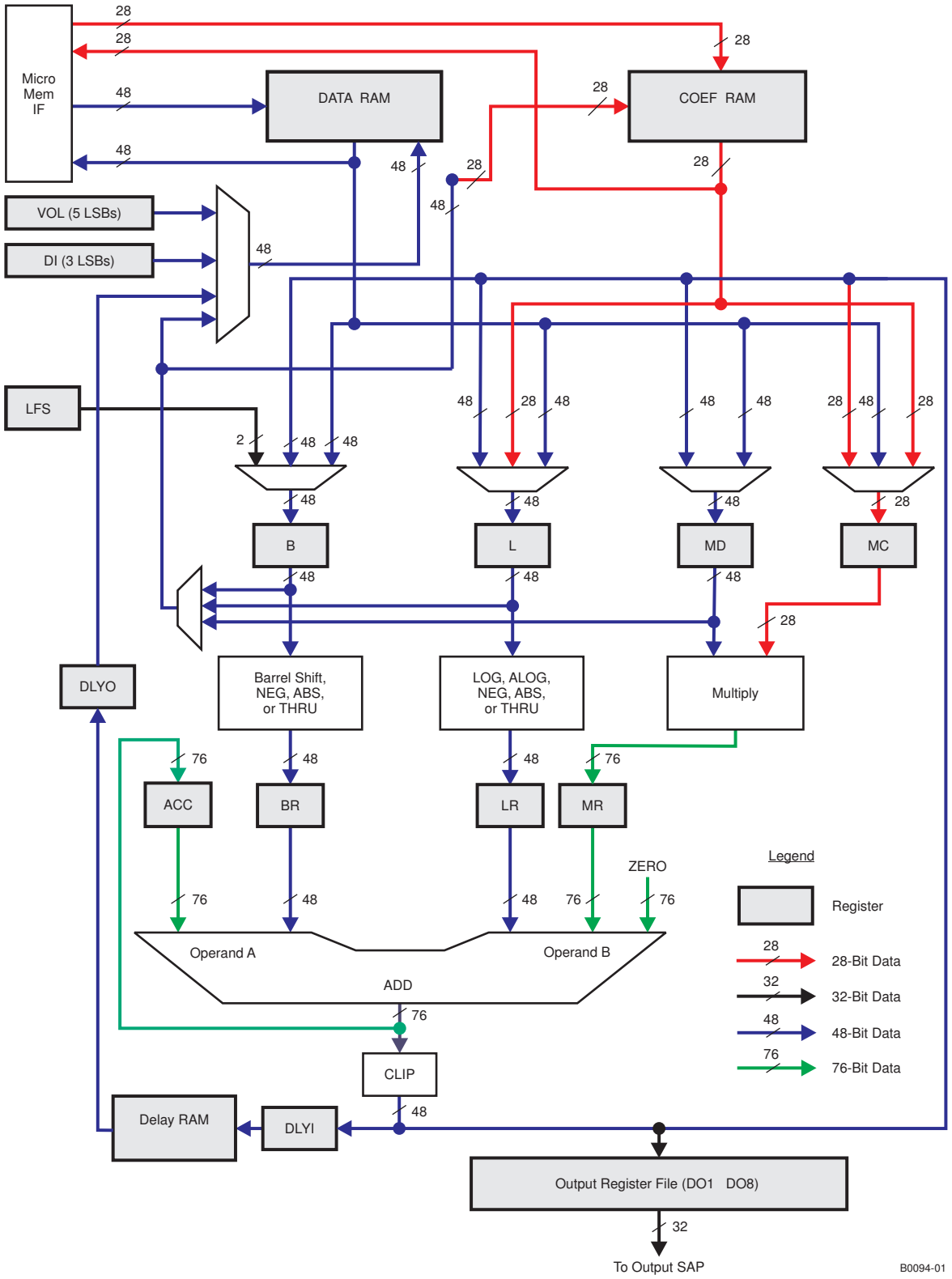


Figure 12. TAS3108/TAS3108IA DSP Audio Processing Architecture

3.4.1 Audio Processing Flow

TAS3108/TAS3108IA audio processing is normally done on a sample-by-sample basis at the f_S rate. Processing flow follows this general scheme, as shown in Figure 13:

1. Input audio data from input SAP (serial audio port)
2. Process audio data
3. Output audio data to output SAP
4. Real-time control from user is provided by I2C using internal 8051 microcode

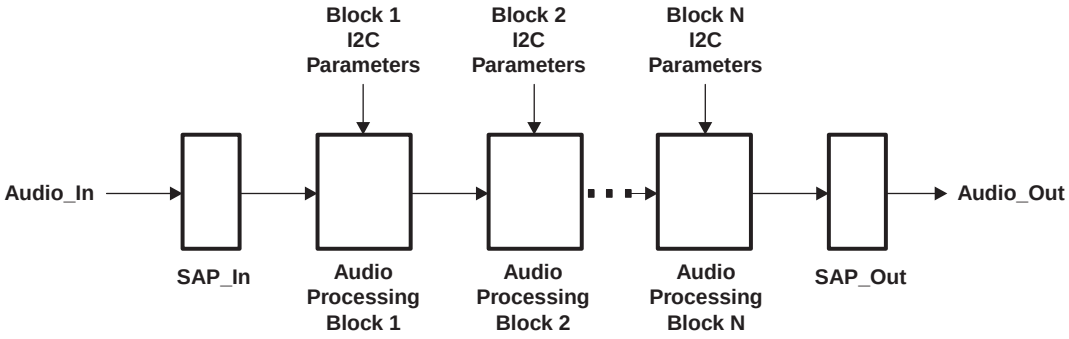


Figure 13. Audio Processing Flow

These next few sections explain how this is done from the programmer's point of view.

3.4.2 Input Audio Data

TAS3108/TAS3108IA provides four digital audio data input pins (SDIN1, SDIN2, SDIN3, and SDIN4). For most applications, these are stereo serial audio inputs. The audio data is connected to these pins and into the input serial audio port. Note that the lower three bits of the memory location selects the input channel as shown. Register DI represents the input SAP 8-channel data input, which is stored in DATA RAM for further processing. Note that the variable name convention for 48-bit data stored in DATA RAM is to place "_D" at the end of the variable name.

Table 3. Input SAP Channel Memory Map

Input Audio Channel	Data RAM Memory Location (Variable Name)	DSP Instruction to Store SAP Data to Data RAM
SDIN1-Left	0x000 (SDIN1L_D)	ST(DI,DATA,SDIN1L_D)
SDIN1-Right	0x001 (SDIN1R_D)	ST(DI,DATA,SDIN1R_D)
SDIN2-Left	0x002 (SDIN2L_D)	ST(DI,DATA,SDIN2L_D)
SDIN2-Right	0x003 (SDIN2R_D)	ST(DI,DATA,SDIN2R_D)
SDIN3-Left	0x004 (SDIN3L_D)	ST(DI,DATA,SDIN3L_D)
SDIN3-Right	0x005 (SDIN3R_D)	ST(DI,DATA,SDIN3R_D)
SDIN4-Left	0x006 (SDIN4L_D)	ST(DI,DATA,SDIN4L_D)
SDIN4-Right	0x007 (SDIN4R_D)	ST(DI,DATA,SDIN4R_D)

3.4.3 Audio Data Processing

Following the audio input, the audio data is normally processed by a series of processing blocks programmed in TAS3108 DSP assembly language. For the non-programmer, a high-level TAS3108 drag-and-drop audio tool is available from Texas Instruments. These processing blocks are normally common audio blocks such as IIR filters, volume control, tone control, DRC, loudness, delay, mixers, and many others. Because the TAS3108/TAS3108IA is fully programmable, the processing that can be done is only limited by the programmer's imagination and the available hardware resources (instruction, data, and coefficient memory).

A simple example of an audio processing block is shown in Figure 14. This example is a 4-input mixer that provides real-time control of the mixer gains by way of the I2C parameters GAIN1, GAIN2, GAIN3, and GAIN4.

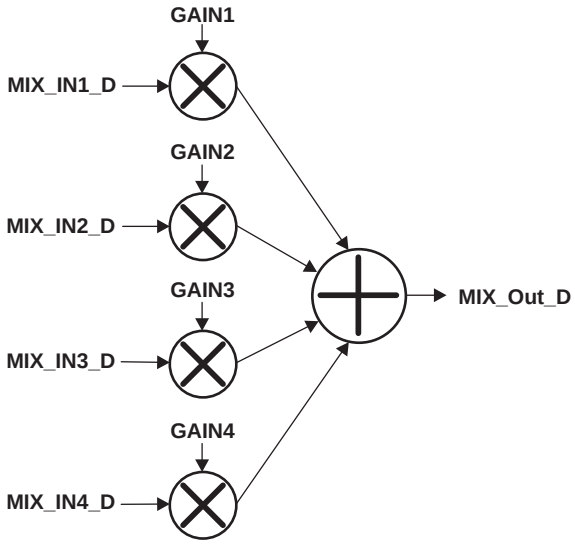


Figure 14. 4-Input Mixer Example Processing Block

An example of TAS3108 assembly code to implement this 4-input mixer is shown in Table 4. The inputs to this block (MIX_IN1_D, MIX_IN2_D, MIX_IN3_D, and MIX_IN4_D) have had the previous block outputs stored to them prior to this example block. Likewise, data output from this block (MIX_OUT_D) is available for further processing as input to the next block.

Table 4. 4-Input Mixer Example Code Snippet

ALU1	ALU2	Data RAM Load to Register	COEF RAM Load to Register	RAM Store
NOP |	CLRACC |	LD(MIX_IN1_D,MD) |	LD(GAIN1,MC) |	NOP
NOP |	NOP |	LD(MIX_IN2_D,MD) |	LD(GAIN2,MC) |	NOP
NOP |	ADD(ACC,MR,NONE,ACC) |	LD(MIX_IN3_D,MD) |	LD(GAIN3,MC) |	NOP
NOP |	ADD(ACC,MR,NONE,ACC) |	LD(MIX_IN4_D,MD) |	LD(GAIN4,MC) |	NOP
NOP |	ADD(ACC,MR,NONE,ACC) |	NOP |	NOP |	NOP
NOP |	ADD(ACC,MR,NONE,B) |	NOP |	NOP |	NOP
NOP |	NOP |	NOP |	NOP |	ST(B,DATA,MIX_OUT_D)

3.4.4 Output Audio Data

Four output pins (SDOUT1, SDOUT2, SDOUT3, and SDOUT4) are provided to output the data from the TAS3108. Following the audio processing blocks, the audio data is output from the DSP. Note that there are no constraints on which memory locations are used for the output data variables. Table 5 shows optimized input/output SAP assembly code.

Table 5. Optimized Input/Output SAP Assembly Code

ALU1	ALU2	Data RAM Load to Register	COEF RAM Load to Register	RAM Store
NOP |	CLRACC |	LD(SDOUT1L_D,L) |	NOP |	NOP
THRU(L) |	NOP |	LD(SDOUT1R_D,L) |	NOP |	ST(DI,DATA,SDIN1L_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO1) |	LD(SDOUT2L_D,L) |	NOP |	ST(DI,DATA,SDIN1R_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO2) |	LD(SDOUT2R_D,L) |	NOP |	ST(DI,DATA,SDIN2L_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO3) |	LD(SDOUT3L_D,L) |	NOP |	ST(DI,DATA,SDIN2R_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO4) |	LD(SDOUT3R_D,L) |	NOP |	ST(DI,DATA,SDIN3L_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO5) |	LD(SDOUT4L_D,L) |	NOP |	ST(DI,DATA,SDIN3R_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO6) |	LD(SDOUT4R_D,L) |	NOP |	ST(DI,DATA,SDIN4L_D)
THRU(L) |	ADD(ACC,LR,CLP32,DO7) |	NOP |	NOP |	ST(DI,DATA,SDIN4R_D)
NOP |	ADD(ACC,LR,CLP32,DO8) |	NOP |	NOP |	NOP

3.4.5 Real-Time Control of Audio Processing

Real-time control of audio processing is provided by the I2C control interface. This I2C interface is facilitated by a TAS3108 hardware block that eventually interfaces with the 8051 microprocessor. This is accomplished through the I2C slave interface, in which the external system controller is the master, and the TAS3108 is the slave. Legal slave addresses are shown in Table 6.

Table 6. TAS3108 I2C Slave Addresses

Base Address	CS0	R/$\overline{W}$	Slave Address
0110 10	0	0	0x68
0110 10	0	1	0x69
0110 10	1	0	0x6A
0110 10	1	1	0x6B

The smallest I2C transaction in a command is four bytes. The 8051 microprocessor is normally programmed so that the maximum number of bytes in an I2C command is 20 bytes. This facilitates sending five filter coefficients at a time since one of the main audio components is an IIR biquad filter that requires five filter coefficients (four bytes per coefficient). In the case of the 4-input mixer component shown above, the four gain values GAIN1, GAIN2, GAIN3, and GAIN4 can be sent from the system controller to the TAS3108 with one I2C write containing 16 bytes, where each gain value is a 28-bit coefficient.

3.4.6 Audio Processing Features of the DSP Architecture

Special features of the processing architecture include:

1. One-cycle multiply
2. 76-bit add block
3. Multiply-accumulate
4. Approximate LOG_2 and $ALOG_2$ functions
5. S-Curve volume processing

3.4.6.1 One-Cycle Multiply

There is no explicit multiply instruction. The multiply operation is implicitly performed every DSP clock and is controlled by the data loaded into registers MC and MD. The output of each multiply is stored in Register MR each clock cycle. So, if the contents of MD and MC are not changed, Register MR contains the same value after each clock cycle. The inputs to the multiply block are 25.23 data (MD) and 5.23 data (MC), with the results being 30.46 data (MR).

3.4.6.2 76-Bit Addition Block

The 76-bit add block has two input operands A and B. Operand A input can be from the 76-bit Accumulator Register (ACC) or 48-bit Register BR. Operand B input can be from the 48-bit Register LR or 76-bit Register MR. There are three automatic transformations done as the input data is converted from 48-bits to 76-bits:

1. Original 48-bit adder input data from BR or LR
2. Concatenate 28 zeroes in LSB
3. Arithmetic shift right by five bits

Figure 15 shows these internal operations.

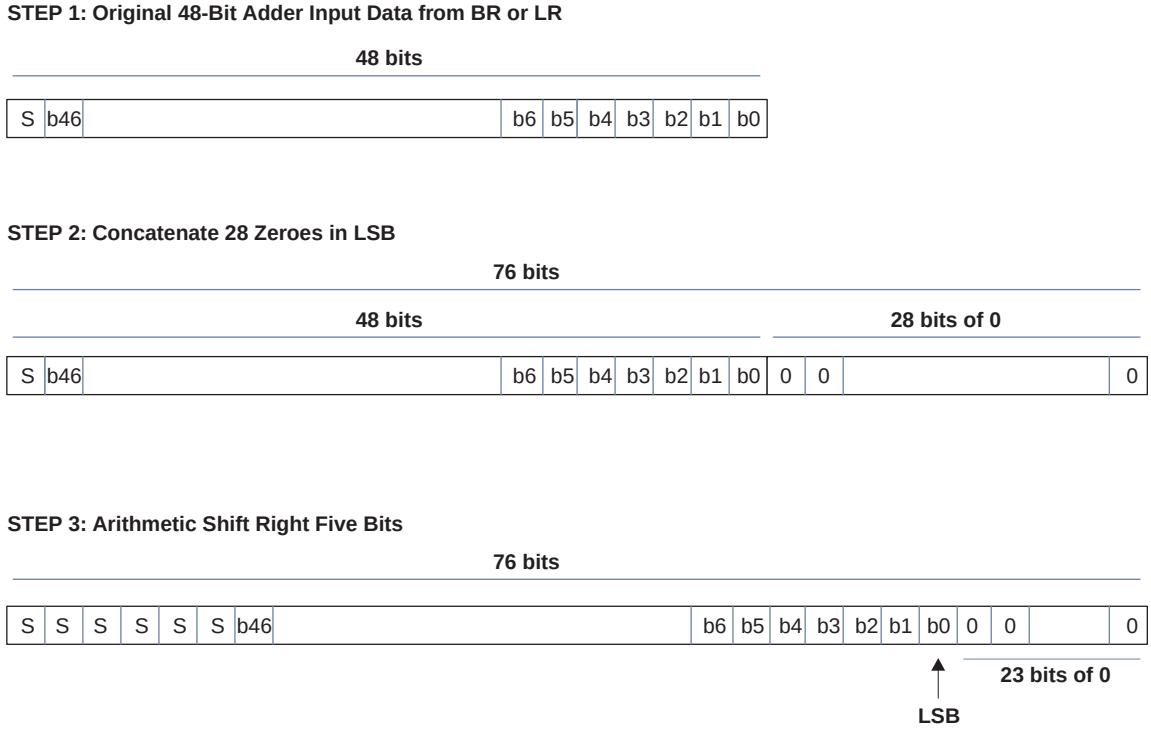
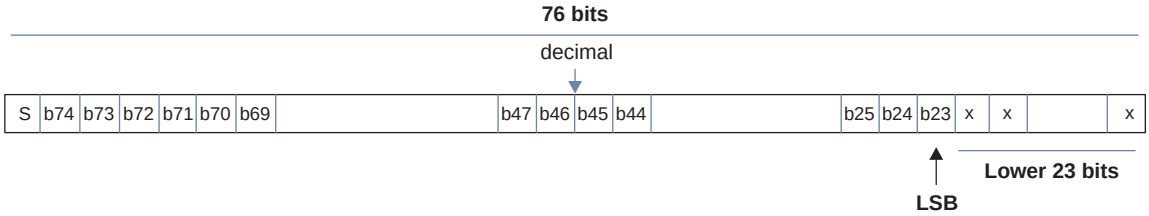


Figure 15. Conversion of 48-Bit Data to 76-Bit Data for Addition

When the data is output from the ADD block, there is a clip algorithm by which the data is transformed from 76-bit data to 48-bit data. Figure 16 shows this clip algorithm.

ADD Output Algorithm to Get to 48-Bit Space



if S = 1 and (if SUM of lower 23 bits ^= 0)
then b23 = b23 + 1;

if S = 1 and (if SUM of lower 23 bits = 0)
then "do nothing";

if S = 0
then "do nothing";

if S = 1 and any adjacent lower bits = 0
then output MAX NEGATIVE 25.23 number;

if S = 0 and any adjacent lower bits = 1
then output MAX POSITIVE 25.23 number;

Figure 16. Output Clipping Algorithm

3.4.6.3 Multiply-Accumulate

Figure 17 shows how a common multiply-accumulate operation can be performed using the TAS3108 assembly language.

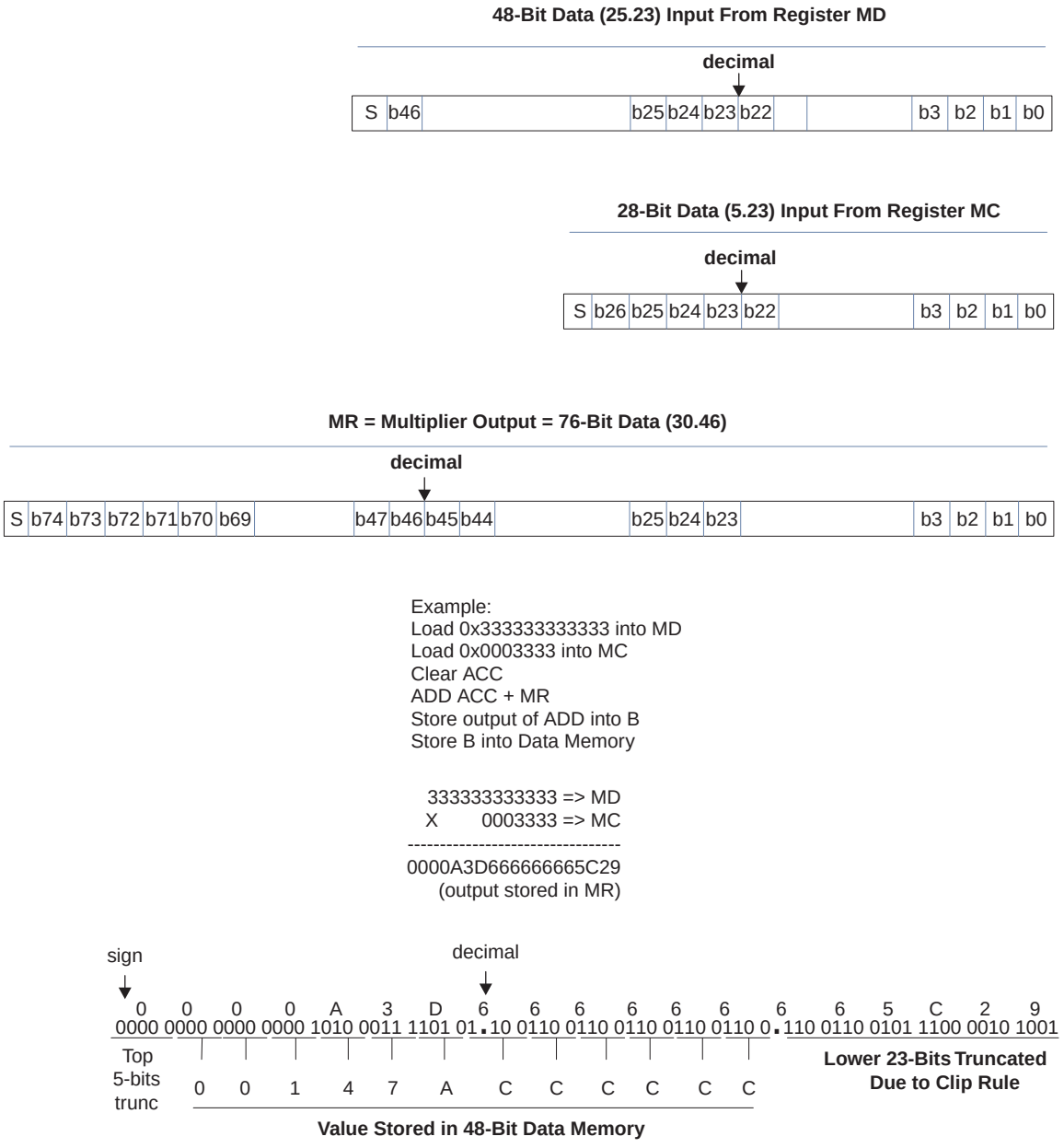


Figure 17. Multiply-Accumulate

3.4.6.4 LOG$_2$ and ALOG$_2$ Operations

LOG and ALOG operations are normally used to scale data so that processing can be performed unclipped (see Figure 18).

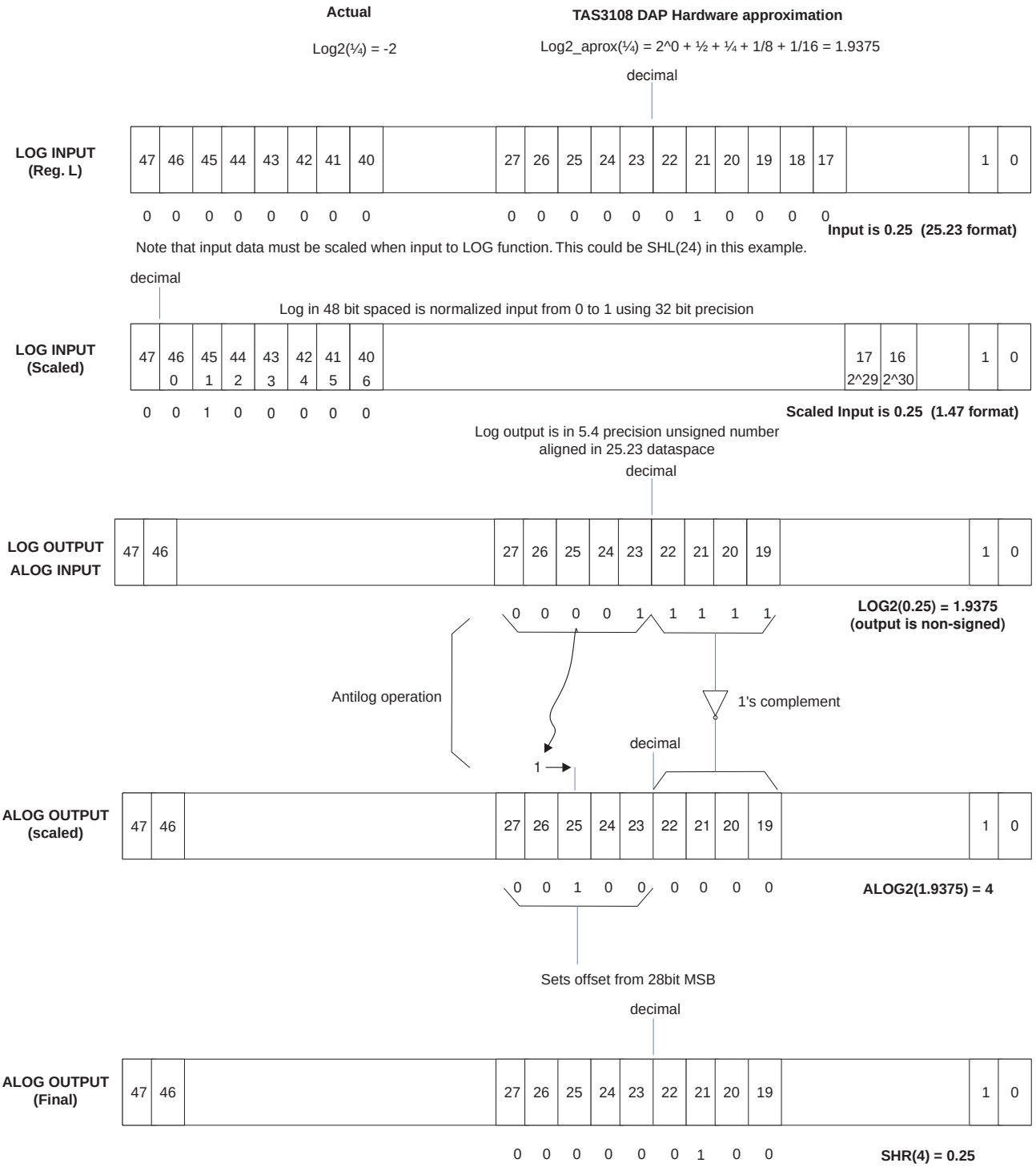


Figure 18. LOG, ALOG Operations

3.4.7 S-Curve Volume Processing

A special feature of the TAS3108 is the built-in S-Curve volume function. This function allows the volume to smoothly change from the initial volume to the target volume. After the S-Curve volume is implemented, the user sends an I2C command (via the system controller) to the TAS3108 that specifies the desired or target volume. The TAS3108 processes this volume command and slowly ramps the volume to the target value. Along with the volume command, the user can specify (usually during initialization) the volume slew rate. For f_S = 48 kHz, the volume slew rate (VSR) = 1024; for f_S = 96 kHz, VSR = 2048, and for f_S = 192 kHz, VSR = 4096. This maintains the volume slew rate at about 42.6 ms. As an example, consider the case where f_S = 48 kHz. In this case, SlewRate is 1024, so 2 × SlewRate = 2048. The total time to ramp on the S-Curve is 2048 × (1 ÷ 48,000) or 42.6 ms (see Figure 19).

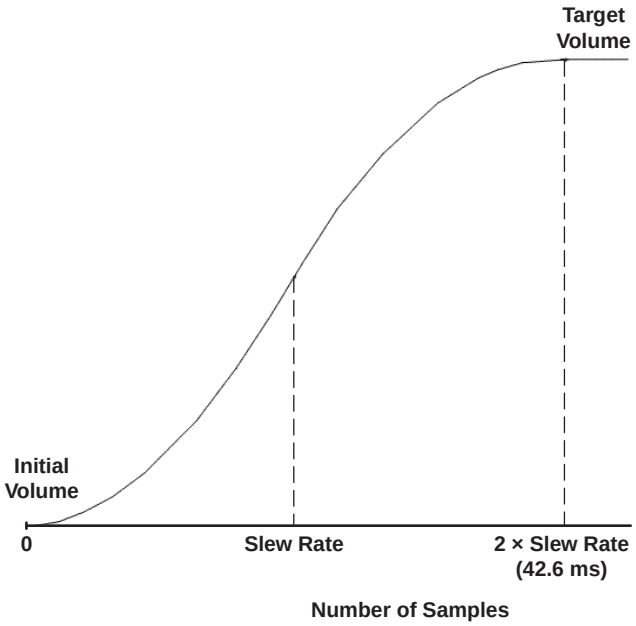


Figure 19. S-Curve Volume Plot

Volume processing is composed of five major blocks (as shown in Figure 20) :

- 8051 Microprocessor
- ESFRs
- Volume Update Block (VUB)
- VOL Register
- TAS3108 DSP Assembly Code

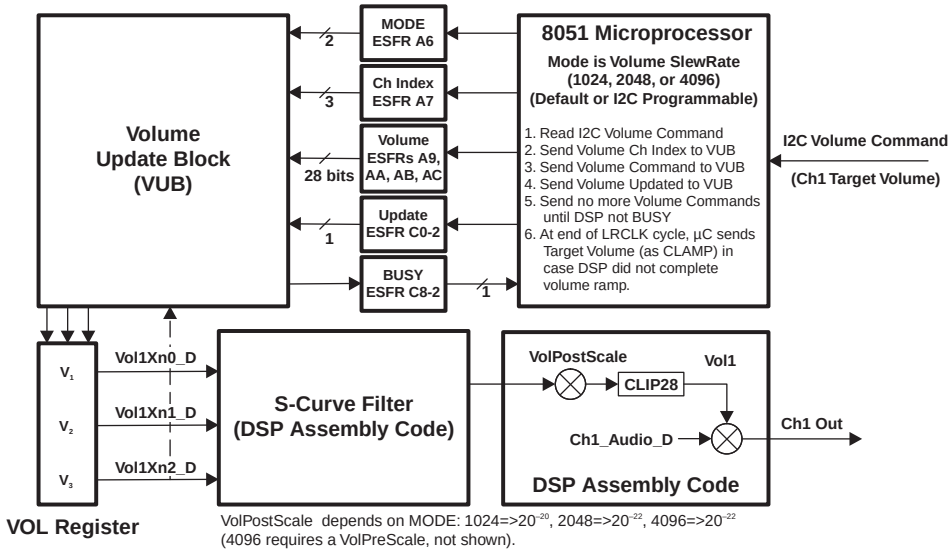


Figure 20. S-Curve Volume Processing

Processing for each block follows:

1. 8051 Microprocessor: The 8051 microprocessor receives an I2C volume command. It then sends volume index to the VUB. This volume index represents the channel whose volume is to be updated.
2. ESFRs: Transfers data between microprocessor and VUB
3. Volume Update Block (VUB): Calculates three slew-rate related filter inputs (Vol1Xn0_D, Vol1Xn1_D, and Vol1Xn2_D) and transfers same to VOL register
4. VOL Register: Provides S-Curve filter inputs used by the DSP Assembly Code
5. DSP Assembly Code: Implements the S-Curve filter transfer function and ultimately applies the target volume to the channel audio stream

3.4.7.1 S-Curve Volume Processing - 8051 Microprocessor Code

For the built-in S-Curve volume processing, the 8051 microprocessor code performs the following functions:

1. Send the Mode command to the VUB – The Mode command is the 2-bit volume SlewRate, which is usually set during initialization. There are three available slew rates: 1024 (48-kHz sample rate) , 2048 (96-kHz sample rate), and 4096 (192-kHz sample rate).
2. Read I2C target volume command – This is a standard TAS3108 28-bit coefficient taking up 4 bytes for the I2C transaction. This command typically comes form the system controller.
3. Send the volume channel index to the VUB – This is a 3-bit command that indicates which channel the volume command is for.
4. Send the target volume command to the VUB – This is the 28-bit volume coefficient that is used to determine the three inputs to the S-Curve filter (Vol1Xn0_D, Vol1Xn1_D, and Vol1Xn2_D).
5. Send volume update command to VUB – This is a one-shot command that begins the S-Curve calculation process.
6. The microprocessor monitors the Busy command from the VUB to wait to send more volume commands. The microprocessor can only send volume commands when the VUB is not busy.
7. At the end of the LRCLK cycle, the microprocessor sends the target volume for final volume applied to channel audio (in case the the volume ramp did not reach its final value).

3.4.7.2 S-Curve Volume ESFRs, Volume Update Block, and VOL Register

The ESFRs associated with the S-Curve volume are used to transfer data from the microprocessor to the VUB.

The S-Curve VUB is a hardware block that takes slew rate, channel index, volume command, and update command as inputs and calculates S-Curve filter inputs Vol1Xn0_D, Vol1Xn1_D, and Vol1Xn2_D. These filter inputs are then provided to the VOL Register for access by the DSP assembly code.

The VOL Register is a mechanism the DSP programmer can use to get the three S-Curve filter input values in the S-Curve volume assembly code.

3.4.7.3 S-Curve Filter Block

The S-Curve filter block is implemented by the DSP assembly code.

Table 7. S-Curve Volume Assembly Code (One Channel)

```
-- Coefficient memory definitions
.coef Vol1 =0x00000000
.coef VolPreScale =0x00800000
.coef VolPostScale =0x00000008
.coef VOLT =0x00800000
.coef NEG2 =0x0F000000
.coef POS2 =0x01000000
-- Data memory definitions
.data VolIn1_D
.data VolOut1_D
-- The volume in data must start with a Data Memory Location with the lower five bits equal to zero (here, start at address 32)
.data Vol1Xn0_D =0x000000000000 @32
.data Vol1Xn1_D =0x000000000000 @32+1
.data Vol1Xn2_D =0x000000000000 @32+2
-- Note for eight channels of volume we must continue to address 32+23 (3 locations per volume)
.data Vol1Yn1_D =0x000000000000 @32+24
.data Vol1Yn2_D =0x000000000000 @32+25
-- Note for eight channels of volume we must continue to address 32+39 (2 locations per volume)
.data Vol1Clamp_D =0x000000000000
.data VolClamp_D =0x000000000000
.data VOLO_D =0x000000000000
-- Begin Volume Calculation. This code can be anywhere in the code space. However, it cannot occur during an LRCLK transition.
NOP |          CLRACC |                    LD(VolClamp_D, B) |  NOP |                  ST(VOL, DATA, Vol1Xn0_D)
NEG (B) |      ADD(ACC, ZERO, NONE, L) |    NOP |                NOP |                  ST(VOL, DATA, Vol1Xn1_D)
THRU (L) |     NOP |                        NOP |                NOP |                  ST(VOL, DATA, Vol1Xn2_D)
NOP |          COMP (BR, LR)               NOP |                NOP |                  NOP
NOP |          NOP |                       LD (Vol1Clamp_D, MD) |  NOP |                  NOP
NOP |          NOP |                       LNC (Vol1Yn1_D, MD) |  NOP |                  NOP
NOP |          NOP |                       LNC (Vol1Yn2_D, MD) |  NOP |                  ST(BD, DATA, Vol1Yn1_D)
NOP |          NOP |                       NOP |                NOP |                  ST(BD, DATA, Vol1Yn2_D)
NOP |          NOP |                       LD(Vol1Xn0_D, MD) |  LD(VolPreScale, MC) |  NOP
NOP |          CLRACC |                    LD(Vol1Xn1_D, MD) |  NOP |                  NOP
NOP |          ADD( ACC, MR, NONE, B) |    LD(Vol1Xn2_D, MD) |  NOP |                  NOP
THRU (B) |     ADD(ACC, MR, NONE, MD) |    LD(Vol1Yn2_D, L) |  LD (NEG2 MC) |         NOP
Neg(L) |       ADD(BR, MR, NONE, ACC) |    LD(Vol1Yn1_D, MD) |  LD (POS2, MC) |        NOP
NOP |          ADD(ACC MR, NONE, ACC) |    LD(Vol1Xn1_D, B) |  NOP |                  NOP
```

Table 7. S-Curve Volume Assembly Code (One Channel) (continued)

NOP |	ADD(ACC, LR, NONE, ACC) |	NOP |	NOP |	ST (B, DATA, Vol1Yn2_D)
NOP |	ADD(ACC, MR, NONE, MDB |	NOP |	LD (VolPostScale, MC) |	NOP
NOP |	NOP |	NOP |	NOP |	ST (B, DATA, Vol1Yn2_D)
NOP |	ADD(ACC, MR, NONE, MD) |	LD (VOLO_D, B) |	LD (VOLT, MC) |	NOP
THRU(B) |	NOP |	NOP |	NOP |	NOP
NOP |	ADD(BR, MR, NONE, ACC) |	NOP |	NOP |	NOP
NOP |	ADD(ACC, ZERO, CLP28, B) |	NOP |	NOP |	NOP
NOP |	NOP |	NOP |	NOP |	ST (B, COEF, Vol1)
-- Multiply Volume Coefficient (Vol1) by the Audio Data (VolIn1_D) and write the result to the volume out variable (Vol1Out1_D)				
NOP |	CLRACC |	LD (VolIn1_D, MD) |	LD (Vol1, MC) |	NOP
NOP |	NOP |	NOP |	NOP |	NOP
NOP |	ADD(ACC, MR, NONE, B) |	NOP |	NOP |	NOP
NOP |	NOP |	NOP |	NOP |	ST (B, DATA, VolOut1_D)

3.5 DSP Assembly Language Instruction Summary

Table 8 summarizes the TAS3108/TAS3108IA assembly language instructions.

Table 8. Audio DSP Assembly Instruction Summary

Instruction	Function	Description
ALU1 Instructions		
ABS (r)	Absolute value	r = B or L, ABS(B) –> BR, ABS(L) –> LR
NEG (r)	Invert and add 1	r = B or L, NEG(B) –> BR, NEG(L) –> LR
LOG2	Base two logarithm	LOG2 of contents of L –> LR
ALOG2	Base two antilogarithm	ALOG2 of contents of L –> LR
SHR (bits)	Logical shift right	SHR (bits) –> BR, bits = 1, 2, 3, or 4
SHL (bits)	Shift left	SHRL(bits) –> BR, bits = 1, 2, 3, 4, or 20
CLRACC	Clear Accumulator	All zeroes –> ACC
THRU (r)	Copy first pipe register r to second pipe register	r = B or L, THRU(B) –> BR, THRU(L) –> LR
JMP	Jump absolute	Used in conjunction with PCADDR
BOC	Branch on A_CP_B = 1	Used in conjunction with PCADDR
BNC	Branch no A_CP_B = 0	Used in conjunction with PCADDR
STOP	Stop program counter	PC
Label		Parameter of PCADDR: PCADDR(*Label*)
NOP	No operation	
ALU2 Instructions		
CLRACC	Clear Accumulator	All zeroes –> ACC
ADD (a,b,c,d)	Add op a + op b with clip c to destination d	a + b –> d (d = DOx, ACC, B, L, MC, MD, DLYI, MDB (with CLRACC)), c = clip parameter (NONE, CLP28, CLP32)
COMP (r1,r2)	Compare	r1 = ACC or BR and r2 = LR or MR
NOP	No operation	
MOP1(AD1) Instructions - Load from DATA RAM to Register		
LD (addr,r)	Load	DATA RAM addr –> r (B, L, BL, MD, or MC)
LDC (addr,r)	Load on A_CP_B = 1	DATA RAM addr –> r (B, L, or MD)
LNC (addr,r)	Load no A_CP_B = 0	DATA RAM addr –> r (B, L, or MD)
NOP	No operation	
MOP2(AD2) Instructions - Load from COEF RAM to Register		
LD (addr,r)	Load	COEF RAM addr –> r (MC or L)
LDC (addr,r)	Load on A_CP_B = 1	COEF RAM addr –> r (MC)
LNC (addr,r)	Load no A_CP_B = 0	COEF RAM addr –> r (MC)
NOP	No operation	
MOP3(AD3) Instructions - Store from Register to DATA or COEF RAM		
ST (r,DATA,addr)	Store	r = B, L, MD, DI, DLYO, or VOL; addr = destination address in DATA RAM
ST (r,COEF,addr)	Store	r = B, L, MD; addr = destination address in COEF RAM
PCADDR	Assign *Label*	Used in Conjunction with JMP, BOC, BNC
NOP	No operation	

3.5.1 ALU1 Instruction Summary

The ALU1 field enables instructions shown in Table 9.

Table 9. ALU1 Instruction Summary

Instruction	Result of Operation	Description
ABS (Register)	ABS(B) –> BR, ABS(L) –> LR	Twos complement absolute value of contents of register stored in the next pipe register.
NEG (Register)	NEG(B) –> BR, NEG(L) –> LR	Twos complement of contents of register stored in the next pipe register.
LOG2	LOG2 of contents of L -> LR	Base 2 log of contents of Register L stored in Register LR
ALOG2	ALOG2 of contents of L -> LR	Base 2 anti-log of contents of Register L stored in Register LR
SHR (bits)	SHR (bits) –> BR	Logical shift right, where bits = 1, 2, 3, or 4
SHL (bits)	SHL (bits) –> BR	Shift left, where bits = 1, 2, 3, 4, or 20
CLRACC	All zeroes –> Accumulator	Clear the Accumulator
THRU (B)	THRU (B) –> BR	Contents of B copied to BR
THRU (L)	THRU (L) –> LR	Contents of L copied to LR
JMP	Absolute jump to label specified in PCADDR instruction	Jump absolute
BOC	If A_CP_B = 1, branch to label specified in PCADDR instruction	Branch on compare bit A_CP_B = 1
BNC	If A_CP_B = 0, branch to label specified in PCADDR instruction	Branch on compare bit A_CP_B = 0
STOP	Stop PC	
NOP	No operation	

3.5.2 Program Counter (PC) Operations

When a PC operation is performed, it uses the whole instruction word. No other operations are allowed on that cycle. Note the following limitations/characteristics of PC operations:

- It requires three cycles to clear the instruction pipe and, hence, three NOP instructions must follow a PC operation. A NOP should be added after these instructions to allow MOP3 AD3 to propagate to memory interface.
- When a PC instruction (JMP, BOC, BNC) is performed, it uses the whole instruction word. No other operations are allowed on that cycle. Note that JMP, BOC, and BNC can occur anywhere in program space but are limited to target (location of PC following branch) the lower 1024 locations of instruction memory.
- PCADDR should always be used in conjunction with a PC operation. PCADDR is a MOP3 command.
- PC ← PCADDR(Label)
- A conditional JUMP or BRANCH is based on the condition of the A_CP_B bit. The A_CP_B bit is set within the COMP function, and it takes two clocks to set the A_CP_B bit.
- The PC position within the JMP command is equal to the PC position of Label1. The instruction PCADDR (Label1) is replaced with the PC location of Label1.

See Table 10 for a list of available PC related operations.

Table 10. Program Counter (PC) Operations

Instruction	Description
JMP	Unconditional jump to label specified in PCADDR instruction
BOC	Branch to label if A_CP_B = 1
BNC	Branch to label if A_CP_B = 0
STOP	Stop program counter (PC)

See Table 11 for a code example using the JMP instruction.

Table 11. Code Example Using JMP Instruction

JMP |	NOP |	NOP |	NOP |	PCADDR (Label1)
NOP |	NOP |	NOP |	NOP |	NOP
NOP |	NOP |	NOP |	NOP |	NOP
NOP |	NOP |	NOP |	NOP |	NOP
		•		
		• Other Code		
		•		
Label1:				
NOP |	NOP |	NOP |	NOP |	NOP

See Table 12 for a code example using the BOC instruction.

Table 12. Code Example Using BOC Instruction

NOP |	COMP (BR, LR) |	NOP |	NOP |	NOP
NOP |	NOP |	NOP |	NOP |	NOP
BOC |	NOP |	NOP |	NOP |	PCADDR (Label1)
NOP |	NOP |	NOP |	NOP |	NOP
NOP |	NOP |	NOP |	NOP |	NOP
NOP |	NOP |	NOP |	NOP |	NOP
		•		
		• Other Code		
		•		
Label1:				
NOP |	NOP |	NOP |	NOP |	NOP

3.5.3 ALU2 Instructions

ALU2 instructions include:

- ADD instructions
- COMP instructions
- CLRACC (load all zeroes into Accumulator)
- NOP (no operation)

Section 3.5.3.1 and Section 3.5.3.2 explain the ADD and COMP instructions.

3.5.3.1 ADD Instructions

The syntax of the ADD instruction is ADD (Reg1, Reg2, Clip, Dest).

The result of the ADD instruction is CLIP (Reg1 + Reg3) –> Dest.

Table 13 summarizes the ADD instruction with these special considerations:

- Whenever Dest is a register other than the Accumulator, magnitude truncation logic is used.
- NONE is the default clip to 48 bits.
- CLP32 is a dynamic range clip to 32 bits.
- CLP28 is a 5.23 coefficient clip. Bits above 28 are clipped.
- If the result destination is MDB, the result is placed in Register MD and Register B, and the Accumulator is cleared.

Table 13. ADD Instruction With Special Considerations

Instruction	Destination
ADD(BR, MR, NONE, Destination)	Destination = B, L, MD, MC, ACC, or DLYI
ADD(BR, MR, CLP32, Destination)	Destination = DO1, DO2, DO3, DO4, DO5, DO6, DO7, or DO8
ADD(BR, LR, NONE, Destination)	Destination = B, L, MD, or MC
ADD(BR, LR, CLP32, Destination)	Destination = DO1, DO2, DO3, DO4, DO5, DO6, DO7, or DO8
ADD(BR, ZERO, NONE, Destination)	Destination = B, L, MD, or MC
ADD(BR, ZERO, CLP32, B)	B is only valid destination
ADD(ACC, LR, NONE, Destination)	Destination = B, L, MD, MC, ACC, or DLYI
ADD(ACC, LR, CLP32, Destination)	Destination = DO1, DO2, DO3, DO4, DO5, DO6, DO7, or DO8
ADD(ACC, MR, NONE, MD)	Destination = B, L, MC, ACC, DLYI, or MDB (with simultaneous CLRACC)
ADD(ACC, MR, CLP32, Destination)	Destination = DO1, DO2, DO3, DO4, DO5, DO6, DO7, or DO8
ADD(ACC, ZERO, NONE, MD)	Destination = B, L, or MD
ADD(ACC, ZERO, CLP28, Destination)	Destination = B or MC

3.5.3.2 COMP Instructions

COMP(Operand_A, Operand_B) Where Operand_A = ACC or BR and Operand_B = LR or MR

The COMP function compares the sign of the contents of Operand_A to the sign of the contents of Operand_B. In this example, the value of Operand_A can come from the Accumulator or Register BR. The value of Operand_B can come from Register LR or Register MR. As Figure 21 shows, the signs of Operand_A and Operand_B are used as the mux select to determine which input is transferred to A_CP_B. Note that it takes two clocks for A_CP_B to be set.

The Figure 21 shows the hardware implementation of the COMP function.

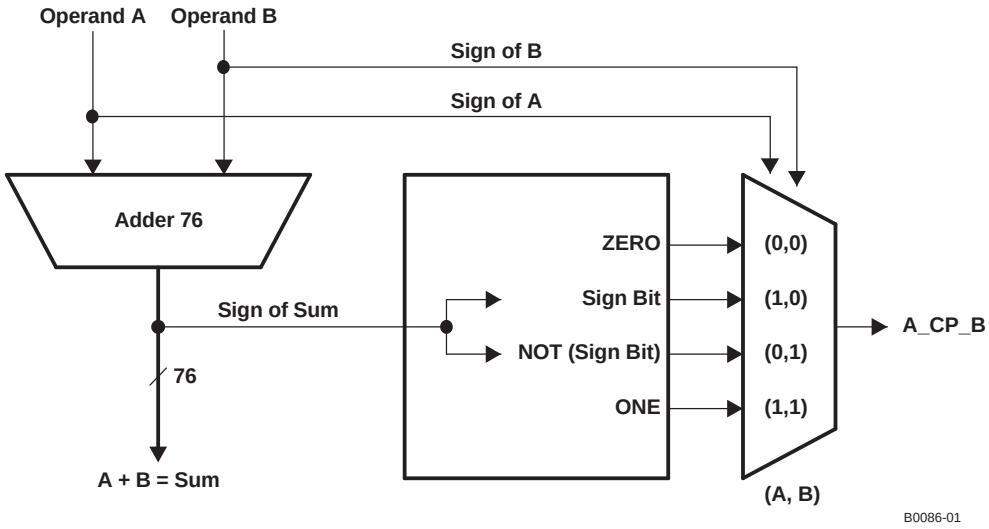


Figure 21. COMP Function

SeeTable 14 below for a summary of the values of A_CP_B.

Table 14. A_CP_B Description

A_CP_B Description
If (0,0) // A and B are Positive Then { A_CP_B = 0 }

If (1,0) // B is Negative and C is Positive
Then
If (|A| = |B|) Then { A_CP_B = 0 }
else A_CP_B = "Sign of Sum"

If (0,1) // A is Positive and B is Negative
Then
If (|A| =|C|) Then { A_CP_B = 1 }
else A_CP_B = NOT "Sign of Sum"

If (1,1) // A and B are Negative Then {A_CP_B = 1}

Further evaluating the results of A_CP_B leads to the rules summarized in Table 15 .

Table 15. The Six Rules for A_CP_B

1) If Operand_A and Operand_B are Positive Then A_CP_B = 0
2) If Operand_A and Operand_B are Negative Then A_CP_B = 1
3) If Operand_A is Negative and Operand_B is Positive and |Operand_A| = |Operand_B| Then A_CP_B = 0
4) If Operand_A is Positive and Operand_B is Negative and |Operand_A| = |Operand_B| Then A_CP_B = 1
5) If |Operand_A| > |Operand_B| Then A_CP_B = 1 //Operand_A and Operand_B signs are different

Table 15. The Six Rules for A_CP_B (continued)

6) If |Operand_A| < |Operand_B| Then A_CP_B = 0 //Operand_A and Operand_B signs are different

Some example test data using the COMP function is provided in Table 16 .

Table 16. COMP Test Data

Operand_A	Operand_B	sum	A_CP_B	Rule
1	1	2	0	1
-1	1	0	0	3
1	-1	0	1	4
-1	-1	-2	1	2
2	1	3	0	1
-2	1	-1	1	5
2	-1	1	1	5
-2	-1	-3	1	2
1	2	3	0	1
-1	2	1	0	6
1	-2	-1	0	6
-1	-2	3	1	2
0	1	1	0	1
1	0	1	0	1
0	-1	-1	0	6
-1	0	-1	1	5
0	0	0	0	1

See Table 17 for an example of one way to use the COMP instruction. Load Y into MD if X is negative, else load Z into MD. Inverting X using the NEG command puts –X into BR. If X is negative, then it becomes positive and A_CP_B = 0. If X was positive, inverting it makes it negative and A_CP_B = 1.

Table 17. COMP Code Example 1

ALU1	ALU2	MOP1	MOP2	MOP3	Comment
NOP |	NOP |	LD(X_D,B) |	LD(ZERO,L) |	NOP	Load X_D into B
NEG(B) |	NOP |	NOP |	NOP |	NOP	– (B) –> (BR)
THRU(L) |	NOP \	NOP |	NOP |	NOP	(L) –> (LR)
NOP |	COMP(BR,LR) |	NOP |	NOP |	NOP	Compare (BR) with (LR)
NOP |	NOP |	NOP |	NOP |	NOP	
NOP |	NOP |	LNC(Y_D,MD) |	NOP |	NOP	If A_CP_B = 0, Load Y_D into MD
NOP |	NOP |	LDC(Z_D,MD) |	NOP |	NOP	If A_CP_B = 1, Load Z_D into MD

3.5.3.3 CLIP Parameter

The CLIP only applies when performing an ADD operation. Whenever the DEST for the ADD operation is a register other then the Accumulator, magnitude truncation logic is implemented.

The CLIP gives control over how much clipping occurs. Two types of clipping can occur, and they are referred to as CLP32 and CLP28. If no clip is specified in the ADD function, the default clip is 48 bits.

The Accumulator can process all 76 bits; therefore, no magnitude truncation occurs.

3.5.4 MOP1, MOP2, and MOP3 Instructions

Memory transfer operations to/from DATA RAM and COEF RAM are done in MOP1, MOP2, and MOP3 instruction fields.

MOP1 instructions allow the programmer to load data from DATA RAM to various registers. An example is LD (D_addr, Register) where D_addr is the address (via the variable name) of data stored in DATA RAM and Register is a valid destination for this instruction.

MOP2 instructions allow the programmer to load data from COEF RAM to various registers. An example is LD (C_addr, Register) where C_addr is the address (via the variable name) of data stored in COEF RAM and Register is a valid destination for this instruction.

MOP3 instructions allow the programmer to store data from various register to either DATA RAM or COEF RAM. An example is ST (Register, DATA, D_addr) where D_addr is the address (via the variable name) of data to be stored in DATA RAM and Register is a valid source for this instruction

Table 18 shows the details of MOP1 instructions. Note that the programming convention for data variable names is that they end in "_D".

Table 18. MOP1 Instructions

Instruction	Description
LD (DataVariable_D, Destination)	Load DataVariable_D into Destination register (B, L, BL, MD, or MC)
LDC (DataVariable_D, Destination)	Load DataVariable_D into Destination register (B, L, or MD) only if A_CP_B = 1
LNC (DataVariable_D, Destination)	Load DataVariable_D into Destination register (B, L, or MD) only if A_CP_B = 0
NOP	No operation

Table 19 shows a MOP1 code example.

Table 19. MOP1 Code Example

```
.data Var1_D
NOP |   NOP |   LD (Var_D, B)    NOP |   NOP    -- Transfer Var1_D from DATA RAM to Register B
NOP |   NOP |   LD (Var_D, BL)   NOP |   NOP    -- Transfer Var1_D from DATA RAM to Register B and Register L
```

Table 20 shows the details of MOP2 instructions.

Table 20. MOP 2 Instructions

Instruction	Description
LD (CoefVariable, Destination)	Load CoefVariable into Destination register (L or MC)
LDC (CoefVariable, Destination)	Load CoefVariable into Destination register (MC) only if A_CP_B = 1
LNC (CoefVariable, Destination)	Load CoefVariable into Destination register (MC) only if A_CP_B = 0
NOP	No operation

Table 21 shows a MOP2 code example.

Table 21. MOP2 Code Example

```
.coef Var1
NOP |      NOP |      NOP |      LD (Var1, MC) |          NOP      -- Transfer Var from COEF RAM to Register MC
```

Table 22 shows the details of MOP3 instructions.

Table 22. MOP3 Instructions

Instruction	Description
ST (SourceRegister, DATA, DataVariable_D)	Store contents of SourceRegister into DATA RAM DataVariable_D, where SourceRegister = B, L, MD, DLYO, DI, or VOL
ST (SourceRegsiter, COEF, CoefVariable)	Store contents of SourceRegister into COEF RAM CoefVariable, where SourceRegister = B, L, or MD
PCADDR (Label)	Used in conjunction with JMP, BOC, or BNC
NOP	No operation

Table 23 shows a MOP3 code example.

Table 23. MOP3 Code Example

```
.coef Var1
.data Var1_D
NOP |      NOP |      NOP |      NOP |      ST (B, COEF, Var1)      -- Store contents of B into COEF RAM variable Var1 (with clip 28)
NOP |      NOP |      NOP |      NOP |      ST (B, COEF, Var1_D)    -- Store contents of B into DATA RAM variable Var1_D
```

The LD command with Register BL as the destination allows loading data into simultaneously into Register B and Register L.

- It is illegal to store in memory and load from the same memory location on the next cycle. There must be at least one cycle between (see example code).
- It is illegal to write to Register MC from DATA memory and simultaneously write to Register L from COEF memory.

LFSR, PC, PC START, and LFSR seed are memory mapped.

Note that LFSR is memory mapped to 0x301 (decimal 769). DSP reads directly into B.

From the programming perspective, a store (ST) command to DLYO, DI, or VOL is seen as writing to a register. It is implemented as a multiplexer, which provides DATA RAM from various sources. The ST commands have the appropriate op codes to manipulate the multiplexer to allow DATA RAM storage of information coming from the delay memory, SAP block, VUB, or from the microprocessor. Default multiplexer settings (if MOP3 = NOP) is to propagate the microprocessor into the audio DSP. This is necessary for the sneak writes from the microprocessor.

Table 24. Example Code to Access Newly Written Data

ALU1	ALU2	MOP1	MOP2	MOP3	Comment
NOP |	NOP |	NOP |	NOP |	ST(B,DATA,VAR1_D)	Store VAR1_D into DATA RAM
NOP |	NOP |	NOP |	NOP |	NOP	Wait one cycle (note that other instructions can be inserted in place of NOPs)
NOP |	NOP |	LD(VAR1_D,MD) |	NOP |	NOP	Load VAR1_D from DATA RAM

3.6 Delay Memory Implementation

Write the audio data to the DLYI (Delay Input) Register and select which delay pointer to use. The delay data can be retrieved from DLYO (Delay Output) Register **13 clock cycles** later. The example code in Table 25 shows one delay implementation. The data in **bold** is the only code directly related to delay. Note that for code optimization, the NOPs can contain other code.

Table 25. Delay Implementation Sample Code

PC	ALU1	ALU2	MOP1	MOP2	MOP3
0	CLRACC |	NOP |	LD (AUDIO_IN_D:B:NONE) |	NOP |	NOP
1	THRU(B) |	NOP |	NOP |	NOP |	NOP
2	NOP |	NOP |	NOP |	NOP |	NOP
0	**ADD(ACC,BR,NONE,DLYI) |**	NOP |	NOP |	NOP |	**DLYPTR(0)**
1	NOP |	NOP |	NOP |	NOP |	NOP
2	NOP |	NOP |	NOP |	NOP |	NOP
3	NOP |	NOP |	NOP |	NOP |	NOP
4	NOP |	NOP |	NOP |	NOP |	NOP
5	NOP |	NOP |	NOP |	NOP |	NOP
6	NOP |	NOP |	NOP |	NOP |	NOP
7	NOP |	NOP |	NOP |	NOP |	NOP
8	NOP |	NOP |	NOP |	NOP |	NOP
9	NOP |	NOP |	NOP |	NOP |	NOP
10	NOP |	NOP |	NOP |	NOP |	NOP
11	NOP |	NOP |	NOP |	NOP |	NOP
12	NOP |	NOP |	NOP |	NOP |	NOP
13	NOP |	NOP |	NOP |	NOP |	**ST(DLYO,DATA,AUDIO_OUT_D)**

Appendix A TAS3108/TAS3108IA DSP Instructions

A.1 ALU1 Instructions

The ALU1 instructions are:

- ABS (B) – Twos-complement absolute value of contents of Register B stored in Register BR
- ABS (L) – Twos-complement absolute value of contents of Register L stored in Register LR
- ALOG2– Inverse $\log_2$ of contents of Register L stored in Register LR
- BNC – On A_CP_B = 0 branch to PC
- BOC – On A_CP_B = 1 branch to PC
- CLRACC – Clear the Accumulator
- JMP – Unconditional jump to PC
- LOG2 – Base 2 log of contents of Register L stored in Register LR
- NEG (B) – Twos complement of contents in Register B stored in Register BR
- NEG (L) – Twos complement of contents in Register L stored in Register LR
- NOP – No operation
- SHL (1), SHL (2), SHL (4), SHL (3), SHL (20) – Contents of Register B shifted left the number of indicated bits and stored in Register BR
- SHR (1), SHR (2), SHR (3) and SHR (4) – Contents of Register B shifted right the number of indicated bits and stored in Register BR
- STOP – Halt the PC
- THRU (B) – Copy contents of Register B into Register BR
- THRU (L) – Copy contents of Register L into Register LR

ABS (reg) Absolute Value of B or L stored in next pipe register

Syntax ABS (B)
ABS (L)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU1 Opfield**
ABS (B)	00001
ABS (L)	01110

Description ABS (B)

Twos-complement absolute value of 48-bit contents of Register B stored in Register BR on next clock cycle

ABS (L)

Twos-complement absolute value of 48-bit contents of Register L stored in Register LR on next clock cycle

Example 1 **ABS (B)** | NOP | NOP | NOP | NOP

Note: NOPs added for clarity. Normally other processing occurs in place of NOP.

Register	**Before Instruction**	**1 Cycle After Instruction**
B	FFFF FE80 0000 (-3.0)	FFFF FE80 0000 (-3.0)
BR	xxxx xxxx xxxx (DC)	0000 0180 0000 (3.0)

Example 2 **ABS (L)** | NOP | NOP | NOP | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
L	0000 0180 0000 (3.0)	0000 0180 0000 (3.0)
LR	xxxx xxxx xxxx (DC)	0000 0180 0000 (3.0)

Comment These are the only two absolute value functions available.

ALOG2 Inverse Log 2 of contents of L stored in LR

Syntax ALOG2

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU1 Opfield
ALOG2	00101

Description Approximation of inverse base 2 log of contents of Register L stored in Register LR on next clock cycle.

Input is in base 2 log-space (5.4 precision unsigned number aligned in 48 bits as a 25.23 number).

Input range: 0.0000 (0x 7FFF FFFF FFFF) to 31.9375 (0x0000 0000 0000)

Output is in 4×-scaled linear space (5.4 precision unsigned number aligned in 48 bits as a 25.23 number).

Output range: 0 to approximately 16

Example 1 **ALOG2** | NOP | NOP | NOP | NOP

Note: NOPs added for clarity. Normally other processing occurs in place of NOP.

Register	Before Instruction	1 Cycle After Instruction
L	0000 00F8 0000 (1.9375)	0000 00F8 0000 (1.9375)
LR	XXXX XXXX XXXX	0000 0200 0000 (4.0)

Example 2 **ALOG2** | NOP | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
L	0000 0078 0000 (2.9375)	0000 0078 0000 (2.9375)
LR	xxxx xxxx xxxx	0000 0400 0000 (8.0)

Comment This inverse log 2 function is only available for contents of Register L. See Section 3.4.6.4 for a description of LOG2 and ALOG2 and the scaling required to use.

BNC Branch on A_CP_B = 0 to *label*

Syntax BNC | NOP | NOP | NOP | PCADDR (*label*)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU1 Opfield**
BNC	10100

Description If A_CP_B = 0, then PC = PC Address specified by *label*, else PC = PC + 1

Notes:

1. The BNC instruction must be followed by three cycles of NOPs since it takes that many cycles to clear the instruction pipe.

2. When BNC is performed, no other instructions (except NOP) are allowed on that cycle.

3. The PCADDR instruction is always used with the BNC instruction.

Example 1 **BNC** | NOP | NOP | NOP | PCADDR(PMRK_A)

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

<other processing>

PMRK_A | NOP | NOP | NOP | NOP

Note: The branch label can be anywhere in the code outside the four lines of code specified for the BNC instruction.

Register	**Before Instruction**	**3 Cycles After Instruction**
PC	PC	If A_CP_B = 0, PC = PMRK_A If A_CP_B = 1, PC = PC + 1

Comment

BOC Branch on A_CP_B = 1 to *label*

Syntax BOC | NOP | NOP | NOP | PCADDR (*label*)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU1 Opfield
BOC	10011

Description If A_CP_B = 1, then PC = PC address specified by *label*, else PC = PC + 1

Notes:

1. The BOC instruction must be followed by three cycles of NOPs, since it takes that many cycles to clear the instruction pipe.

2. When BOC is performed, no other instructions (except NOP) are allowed on that cycle.

3. The PCADDR instruction is always used with the BOC instruction.

Example 1 **BOC** | NOP | NOP | NOP | PCADDR(PMRK_A)

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

<other processing>

PMRK_A | NOP | NOP | NOP | NOP

Note: The branch label can be anywhere in the code outside the four lines of code specified for the BOC instruction.

Register	Before Instruction	3 Cycles After Instruction
PC	PC	If A_CP_B = 1, PC = PMRK_A If A_CP_B = 0, PC = PC + 1

Comment

CLRACC Clear the Accumulator

Syntax CLRACC

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU1 Opfield**
CLRACC	01111

Description The cycle following the CLRACC instruction, the 76-bit Accumulator will contain all zeroes.

Example **CLRACC** | NOP | NOP | NOP | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
ACC	xxx xxxx xxxx xxxx xxxx	000 0000 0000 0000 0000

Comment There is also a CLRACC instruction available in the Pipe Operation 2 (ALU2) Opfield.

LOG2 Log 2 of contents of L stored in LR

Syntax LOG2

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU1 Opfield
LOG2	00100

Description Approximation of base 2 log of contents of Register L stored in Register LR on next clock cycle.

Input is in linear-space (25.23 precision unsigned number aligned in 48 bits).

Input range: 0.0000 (0x 0000 0000 0000 0000) to approximately 1.0 (0x7FFF FFFF FFFF)

Output is in base 2 log-space (5.4 precision unsigned number aligned in 48 bits as 25.23 data).

Output range: 31.9375 to 0.0

Example 1 **LOG2** | NOP | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
L	2000 0000 0000 (0.25)	2000 0000 0000 (0.25)
LR	xxxx xxxx xxxx	0000 00F8 0000 (1.9375)

Example 2

Register	Before Instruction	1 Cycle After Instruction
L	0080 0000 0000 (0.00390625)	0080 0000 0000 (0.00390625)
LR	xxxx xxxx xxxx	0000 03F8 0000 (7.9375)

Comment This log 2 function is only available for contents of Register L. See Section 3.4.6.4 for a description of LOG2 and ALOG2 and the scaling required to use.

JMP Unconditional jump to PC

Syntax JMP | NOP | NOP | NOP | PCADDR (*label*)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU1 Opfield**
JMP	10010

Description Unconditional jump to PC address specified by *label*

Notes:

1. The JMP instruction must be followed by three cycles of NOPs, since it takes that many cycles to clear the instruction pipe.
2. When JMP is performed, no other instructions (except NOP) are allowed on that cycle.
3. The PCADDR instruction is always used with the JMP instruction.

Example 1 **JMP** | NOP | NOP | NOP | PCADDR(PMRK_A)

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

<other processing>

PMRK_A | NOP | NOP | NOP | NOP

Note: The branch label can be anywhere in the code outside the four lines of code specified for the JMP instruction.

Register	**Before Instruction**	**3 Cycles After Instruction**
PC	PC	PC = PMRK_A

Comment

NEG Twos complement of contents of Register B or L

Syntax NEG (B)
NEG (L)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU1 Opfield
NEG (B)	00010
NEG (L)	00011

Description NEG (B)

Twos complement of 48-bit contents of Register B stored in Register BR on next clock cycle.

NEG (L)

Twos complement of 48-bit contents of Register L stored in Register LR on next clock cycle.

Example 1 **NEG (B)** | NOP | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
B	FFFF FF80 0000 (-1.0)	FFFF FF80 0000 (-1.0)
BR	xxxx xxxx xxxx	0000 0080 0000 (1.0)

Example 2 **NEG (L)** | NOP | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
L	FFFF FE80 0000 (-3.0)	FFFF FE80 0000 (-3.0)
LR	xxxx xxxx xxxx	0000 0180 0000 (3.0)

Comment These are the only twos-complement functions available.

NOP No operation

Syntax NOP

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU1 Opfield**
NOP	00000

Description The cycle following the NOP instruction, the state of all registers remains unchanged.

Example 1 **NOP** | NOP | NOP | NOP | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
All	xxx xxxx xxxx xxxx xxxx	Unchanged

Comment The NOP instruction is mandatory when used with PC instructions and can also be used to improve readability of code.

SHL (bits) Contents of Register B shifted left the number of indicated bits and stored in Register BR on next cycle.

Syntax SHL (*bits*)

Where *bits* are 1, or 2, or 3, or 4, or 20

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU1 Opfield
SHL (1)	01010
SHL (2)	00111
SHL (3)	01100
SHL (4)	01101
SHL (20)	10110

Description SHL (*bits*)

Contents of Register B are shifted left the number of bits indicated and stored in Register BR.

Processing Register BR = Register B * 2 exp *bits*

Example 1 **SHL (1)** | NOP | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
B	0000 0080 0000	0000 0080 0000
BR	xxxx xxxx xxxx	0000 0100 0000

Example 2 **SHL (20)** | NOP | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
B	0000 0000 0008	0000 0000 0008
BR	xxxx xxxx xxxx	0000 0008 0000

Comment Shift left operations can only be done on the contents of Register B.

SHR (bits) Contents of Register B shifted right the number of indicated bits and stored in Register BR on next cycle

Syntax SHR (*bits*)

Where *bits* are 1, 2, 3, or 4

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU1 Opfield
SHR (1)	00110
SHR (2)	00111
SHR (3)	01000
SHR (4)	01001

Description SHR (*bits*)

Contents of Register B are shifted right the number of bits indicated and stored in Register BR.

Processing Register BR = (Register B) * 2 exp -*bits*

Example 1 **SHR (3)** | NOP | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
B	0000 0080 0000	0000 0080 0000
BR	xxxx xxxx xxxx	0000 0010 0000

Example 2 **SHR (4)** | NOP | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
B	0000 0080 0000	0000 0080 0000
BR	xxxx xxxx xxxx	0000 0008 0000

Comment Shift right operations can only be done on the contents of Register B.

STOP Stop the Program Counter

Syntax STOP | NOP | NOP | NOP | NOP

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU1 Opfield**
STOP	10101

Description Halt the Program Counter. Every DSP assembly program must have at least one STOP instruction (usually at the end of the program).

Note:

If an LRCLK error occurs at the same time as execution of the STOP instruction, the PC may miss the STOP instruction and erroneously continue. To guard against this, the following example shows a defensive programming technique to ensure that the STOP instruction is correctly executed.

Example

PMRK_D | NOP | NOP | NOP | NOP

STOP | NOP | NOP | NOP | PCADDR(PMRK_D)

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

NOP | NOP | NOP | NOP | NOP

JMP | NOP | NOP | NOP | PCADDR(PMRK_D)

Note: The branch label can be anywhere in the code outside the four lines of code specified for the JMP instruction.

Comment

THRU(reg) Copy contents of Register reg to regR

Syntax THRU (B)
THRU (L)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU1 Opfield**
THRU (B)	10000
THRU (L)	10001

Description THRU (B)
Copy the 48-bit contents of Register B to Register BR on next clock cycle.

THRU (L)
Copy the 48-bit contents of Register L to Register LR on next clock cycle.

Example 1 **THRU (B)** | NOP | NOP | NOP | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
B	FFFF FF80 0000	FFFF FF80 0000
BR	xxxx xxxx xxxx	FFFF FF80 0000

Example 2 **THRU (L)** | NOP | NOP | NOP | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
L	00AB CDEF 1234	00AB CDEF 1234
LR	xxxx xxxx xxxx	00AB CDEF 1234

Comment

A.2 ALU2 Instructions

The ALU2 Instructions are:

- ADD(ACC,LR,CLP32,*reg*) – Add the Accumulator to Register LR and store the result in Register *reg* with 32-bit clip, where *reg* = DO1, DO2, ..., or DO8.

- ADD(ACC,LR,NONE,*reg*) – Add the Accumulator to Register LR and store the result in Register *reg*, where *reg* = ACC, B, DLYI, L, MC, or MD.
- ADD(ACC,MR,CLP32,*reg*) – Add the Accumulator to Register MR and store the result in Register *reg* with 32-bit clip, where *reg* = DO1, DO2, ..., or DO8.
- ADD(ACC,MR,NONE,*reg*) – Add the Accumulator to Register MR and store the result in Register *reg*, where *reg* = MDB, ACC, B, DLYI, L, MC, or MD.
- ADD(ACC,ZERO,CLP28,*reg*) – Add the Accumulator to Register LR and store the result in Register *reg* with 28-bit clip, where *reg* = B, L, or MC.
- ADD(ACC,ZERO,NONE,*reg*) – Add the Accumulator to Register LR and store the result in Register *reg*, where *reg* = B.
- ADD(BR,LR,CLP32,*reg*) – Add Register BR to Register LR and store the result in Register *reg* with 32-bit clip, where *reg* = DO1, DO2, ..., or DO8.
- ADD(BR,LR,NONE,*reg*) – Add Register BR to Register LR and store the result in Register *reg*, where *reg* = B, L, MC, MD, or DLYI.
- ADD(BR,MR,CLP32,*reg*) – Add Register BR to Register LR and store the result in Register *reg* with 32-bit clip, where *reg* = DO1, DO2, ..., or DO8.
- ADD(BR,MR,NONE,*reg*) – Add Register BR to Register LR and store the result in Register *reg*, where *reg* = ACC, B, DLYI, L, MC, and MD.
- ADD(BR,ZERO,CLP32,B) **–** Add Register BR to Register ZERO and store the result in Register B with 32-bit clip.
- ADD(BR,ZERO,NONE,*reg*) – Add Register BR to Register LR and store the result in Register *reg*, where *reg* = B, L, MC, or MD.
- CLRACC – Clear the Accumulator
- COMP(ACC,LR) – Compare Accumulator with Register LR
- COMP(ACC,MR) – Compare Accumulator with Register MR
- COMP(BR,LR) – Compare Register BR with Register LR
- COMP(BR,MR) – Compare Register BR with Register MR
- NOP – No operation

ADD(ACC,LR,CLP32,*reg*) Add the Accumulator to Register LR and store result in Register *reg* with 32-bit clip, where *reg* = DO1, DO2,..., or DO8.

Syntax ADD(ACC,LR,CLP32,DO1)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
ADD(ACC,LR,CLP32,DO1)	011 1001
ADD(ACC,LR,CLP32,DO2)	011 1010
ADD(ACC,LR,CLP32,DO3)	011 1011
ADD(ACC,LR,CLP32,DO4)	011 1100
ADD(ACC,LR,CLP32,DO5)	110 1111
ADD(ACC,LR,CLP32,DO6)	110 0010
ADD(ACC,LR,CLP32,DO7)	110 0011
ADD(ACC,LR,CLP32,DO8)	111 0011

Description DO1 = Clip32 (ACC + LR)

Contents of Accumulator added to contents of Register LR and stored in Register DO1. The result is clipped to 32 bits.

Example NOP | **ADD(ACC,LR,CLP32,DO8)** | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
ACC	000 0000 0000 0000 0000	000 0000 0000 0000 0000
LR	0000 0000 000A	0000 0000 000A
DO8	xxxx xxxx	0000 000A

Comment The eight DSP output registers DO1–DO8 send their 32-bit output to the output serial audio port (SAP).

ADD(ACC,LR,NONE,*reg*) Add the Accumulator to Register LR and store result in Register *reg* where *reg* = ACC, B, DLYI, L, MC, or MD.

Syntax ADD(ACC,LR,NONE,ACC)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
ADD(ACC,LR,NONE,ACC)	001 1101
ADD(ACC,LR,NONE,B)	001 1010
ADD(ACC,LR,NONE,DLYI)	001 1111
ADD(ACC,LR,NONE,L)	001 1011
ADD(ACC,LR,NONE,MC)	001 1100
ADD(ACC,LR,NONE,MD)	001 1001

Description Clip48(ACC + LR) --> dest

Contents of Accumulator added to contents of Register LR and stored in dest register. The result is clipped to 48 bits.

Example 1 NOP | **ADD(ACC,LR,NONE,ACC)** | NOP | NOP | NOP

Note: Destination is the Accumulator (a 76-bit register).

Register	Before Instruction	1 Cycle After Instruction
ACC	000 0000 0000 0000 0000	000 0000 0000 0600 0000
LR	0000 0000 000C	0000 0000 000C

Example 2 NOP | **ADD(ACC,LR,NONE,L)** | NOP | NOP | NOP

Note: Destination is Register L (a 48-bit register).

Register	Before Instruction	1 Cycle After Instruction
ACC	000 0000 0000 0000 0000	000 0000 0000 0000 0000
LR	0000 1234 5678	0000 1234 5678
L	xxxx xxxx xxxx	0000 1234 5678

Comment

ADD(ACC,MR,CLP32,*reg*) Add the Accumulator to Register MR and store result in Register *reg* with 32-bit Clip where *reg* = DO1, DO2, ..., or DO8.

Syntax ADD(ACC,MR,CLP32,DO1)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
ADD(ACC,MR,CLP32,DO1)	011 0101
ADD(ACC,MR,CLP32,DO2)	011 0110
ADD(ACC,MR,CLP32,DO3)	011 0111
ADD(ACC,MR,CLP32,DO4)	011 1000
ADD(ACC,MR,CLP32,DO5)	110 1110
ADD(ACC,MR,CLP32,DO6)	110 0000
ADD(ACC,MR,CLP32,DO7)	110 0001
ADD(ACC,MR,CLP32,DO8)	111 0010

Description DO1 = Clip32 (ACC + MR)

Contents of Accumulator added to contents of Register MR and stored in Register DO1. The result is clipped to 32 bits.

Example NOP | **ADD(ACC,MR,CLP32,DO6)** | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
ACC	000 0000 0000 0000 0000	000 0000 0000 0000 0000
MR	0000 0000 00AB	0000 0000 00AB
DO6	xxxx xxxx	0000 00AB

Comment The eight DSP output registers DO1–DO8 send their 32-bit output to the output serial audio port (SAP).

ADD(ACC,MR,NONE,*reg*) Add the Accumulator to Register MR and store result in Register *reg*, where *reg* = ACC, B, DLYI, L, MC, MD, or MDB.

Syntax ADD(ACC,MR,NONE,ACC)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
ADD(ACC,MR,NONE,ACC)	001 0101
ADD(ACC,MR,NONE,B)	001 0010
ADD(ACC,MR,NONE,DLYI)	001 0111
ADD(ACC,MR,NONE,L)	001 0011
ADD(ACC,MR,NONE,MC)	001 0100
ADD(ACC,MR,NONE,MD)	001 0001
ADD(ACC,MR,NONE,MDB)	111 0100

Description Clip48(ACC + MR) --> dest

Contents of Accumulator added to contents of Register MR and stored in dest register. Note that when MDB is the destination, the result is stored simultaneously to MD and B registers while clearing the Accumulator.

Example 1 NOP | **ADD(ACC,MR,NONE,ACC)** | NOP | NOP | NOP

Note: Destination is the Accumulator (a 76-bit register).

Register	Before Instruction	1 Cycle After Instruction
ACC	000 0000 0000 0000 0000	000 0000 0000 0600 0000
MR	0000 0000 000C	0000 0000 000C

Example 2 NOP | **ADD(ACC,MR,NONE,L)** | NOP | NOP | NOP

Note: Destination is Register L (a 48-bit register).

Register	Before Instruction	1 Cycle After Instruction
ACC	000 0000 0000 0000 0000	000 0000 0000 0000 0000
LR	0000 0000 1234	0000 0000 1234
L	xxxx xxxx xxxx	0000 0000 1234

Comment

ADD(ACC,ZERO,CLP28,*reg*) Add the Accumulator to Register ZERO and store result in Register *reg* with 28-bit clip, where *reg* = B, L, or MC.

Syntax ADD(ACC,ZERO,CLP28,B)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU2 Opfield**
ADD(ACC,ZERO,CLP28,B)	110 1010
ADD(ACC,ZERO,CLP28,MC)	110 1000
ADD(ACC,ZERO,CLP28,L)	110 1011

Description

Contents of Accumulator added to ZERO Register (all zeroes) and stored in the destination register. The result is clipped to 28 bits.

Example 1 NOP | **ADD(ACC,ZERO,CLP28,B)** | NOP | NOP | NOP

Note: Destination is Register B (a 48-bit register).

Register	**Before Instruction**	**1 Cycle After Instruction**
ACC	FFF FFFF FFFF FF80 0000	FFF FFFF FFFF FF80 0000
ZERO	000 0000 0000 0000 0000	000 0000 0000 0000 0000
B	FFFF FFFF FFFF	FFFF FFFF FFFF

Example 2 NOP | **ADD(ACC,ZERO,CLP28,MC)** | NOP | NOP | NOP

Note: Destination is Register MC (a 28-bit register).

Register	**Before Instruction**	**1 Cycle After Instruction**
ACC	FFF FFFF FFFF FF80 0000	FFF FFFF FFFF FF80 0000
ZERO	000 0000 0000 0000 0000	000 0000 0000 0000 0000
MC	FFF FFFF	FFF FFFF

Comment

ADD(ACC,ZERO,NONE,*reg*) Add the Accumulator to Register ZERO and store result in Register *reg*, where *reg* = B.

Syntax ADD(ACC,ZERO,NONE,B)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
ADD(ACC,ZERO,NONE,B)	101 0000

Description Contents of Accumulator added to Register ZERO (all zeroes) and stored in the destination register.

Example 1 NOP | **ADD(ACC,ZERO,NONE,B)** | NOP | NOP | NOP

Note: Destination is Register B (a 48-bit register).

Register	Before Instruction	1 Cycle After Instruction
ACC	000 FFFF FFFF FF80 0000	FFF FFFF FFFF FF80 0000
ZERO	000 0000 0000 0000 0000	000 0000 0000 0000 0000
B	FFFF FFFF FFFF	0FFF FFFF FFF8

Example 2 NOP | **ADD(ACC,ZERO,NONE,MD)** | NOP | NOP | NOP

Note: Destination is Register MD (a 48-bit register).

Register	Before Instruction	1 Cycle After Instruction
ACC	000 FFFF FFFF FF80 0000	FFF FFFF FFFF FF80 0000
ZERO	000 0000 0000 0000 0000	000 0000 0000 0000 0000
MD	FFFF FFFF FFFF	0FFF FFFF FFF8

Comment

ADD(BR,LR,CLP32,*reg*) Add Register BR to Register LR and store result in Register *reg* with 32-bit clip, where *reg* = DO1, DO2, ..., or DO8.

Syntax ADD(BR,LR,CLP32,DO1)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
ADD(BR,LR,CLP32,DO1)	011 0001
ADD(BR,LR,CLP32,DO2)	011 0010
ADD(BR,LR,CLP32,DO3)	011 0011
ADD(BR,LR,CLP32,DO4)	011 0100
ADD(BR,LR,CLP32,DO5)	110 1101
ADD(BR,LR,CLP32,DO6)	101 1110
ADD(BR,LR,CLP32,DO7)	101 1111
ADD(BR,LR,CLP32,DO8)	111 0001

Description outreg = Clip32 (BR + LR)

Contents of Register BR added to contents of Register LR and stored in Register outreg. The result is clipped to 32 bits.

Example NOP | **ADD(BR,LR,CLP32,DO4)** | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
BR	0000 0000 001C	0000 0000 001C
LR	0000 0000 0123	0000 0000 0123
DO4	FFFF FFFF	0000 013F

Comment The eight DSP output registers DO1–DO8 send their 32-bit output to the output serial audio port (SAP).

ADD(BR,LR,NONE,*reg*) Add Register BR to Register LR and store result in Register *reg*, where *reg* = B, L, MC, MD, or DLYI.

Syntax ADD(BR,LR,NONE,*reg*)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
ADD(BR,LR,NONE,B)	000 1010
ADD(BR,LR,NONE,L)	000 1011
ADD(BR,LR,NONE,MC)	000 1100
ADD(BR,LR,NONE,MD)	000 1001
ADD(BR,LR,NONE,DLYI)	000 1111

Description Clip48(BR + LR) --> dest

Contents of Register BR added to contents of Register LR and stored in dest register. The result is clipped to 48 bits.

Example 1 NOP | **ADD(BR,LR,NONE,B)** | NOP | NOP | NOP

Note: Destination is the Accumulator (a 48-bit register).

Register	Before Instruction	1 Cycle After Instruction
BR	0000 0000 001C	0000 0000 001C
LR	0000 0000 0123	0000 0000 0123
B	xxxx xxxx xxxx	0000 0000 013F

Example 2 NOP | **ADD(BR,LR,NONE,MC)** | NOP | NOP | NOP

Note: Destination is Register L (a 28-bit register).

Register	Before Instruction	1 Cycle After Instruction
BR	0000 0000 001C	0000 0000 001C
LR	0000 0000 0123	0000 0000 0123
MC	xxx xxxx	000 013F

Comment

ADD(BR,MR,CLP32,*reg*) Add Register BR to Register MR and store result in Register *reg* with 32-bit clip, where *reg* = DO1, DO2, ..., DO8.

Syntax ADD(BR,MR,CLP32,DO1)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU2 Opfield**
ADD(BR,MR,CLP32,DO1)	010 1101
ADD(BR,MR,CLP32,DO2)	010 1110
ADD(BR,MR,CLP32,DO3)	010 1111
ADD(BR,MR,CLP32,DO4)	010 1000
ADD(BR,MR,CLP32,DO5)	110 1100
ADD(BR,MR,CLP32,DO6)	101 1100
ADD(BR,MR,CLP32,DO7)	101 1101
ADD(BR,MR,CLP32,DO8)	111 0000

Description DO1 = Clip32 (BR + MR)

Contents of Register BR added to contents of Register MR and stored in Register DO1. The result is clipped to 32 bits.

Example NOP | **ADD(BR,MR,CLP32,DO3)** | NOP | NOP | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
BR	0000 0000 001C	0000 0000 001C
MR	0000 0000 0123	0000 0000 0123
DO3	xxxx xxxx	0000 013F

Comment The eight DSP output registers DO1–DO8 send their 32-bit output to the output serial audio port (SAP).

ADD(BR,MR,NONE,*reg*) Add Register BR to Register MR and store result in Register *reg*, where *reg* = ACC, B, DLYI, L, MC, or MD.

Syntax ADD(BR,MR,NONE,ACC)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
ADD(BR,MR,NONE,ACC)	000 0101
ADD(BR,MR,NONE,B)	000 0010
ADD(BR,MR,NONE,DLYI)	000 0111
ADD(BR,MR,NONE,L)	000 0011
ADD(BR,MR,NONE,MC)	000 0100
ADD(BR,MR,NONE,MD)	000 0001

Description Clip48(BR + LR) --> dest

Contents of Accumulator added to contents of Register LR and stored in dest register. The result is clipped to 48 bits.

Example 1 NOP | **ADD(BR,MR,NONE,ACC)** | NOP | NOP | NOP

Note: Destination is the Accumulator (a 76-bit register).

Register	Before Instruction	1 Cycle After Instruction
BR	FFF FFFF FFFF FF80 0000	FFF FFFF FFFF FF80 0000
LR	FFFF FFFF FFFF	FFFF FFFF FFFF
ACC	FFF FFFF FFFF FF80 0000	FFF FFFF FFFF FF80 0000

Example 2 NOP | **ADD(BR,MR,NONE,L)** | NOP | NOP | NOP

Note: Destination is Register L (a 48-bit register).

Register	Before Instruction	1 Cycle After Instruction
BR	FFFF FFFF FFFF	FFFF FFFF FFFF
LR	FFFF FFFF FFFF	FFFF FFFF FFFF
L	FFFF FFFF FFFF	FFFF FFFF FFFF

Comment

ADD(BR,ZERO,CLP32,B) Add Register BR to Register ZERO and store result in Register B with 32-bit clip.

Syntax ADD(BR,ZERO,CLP32,B)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU2 Opfield**
ADD(BR,ZERO,CLP32,B)	101 1011

Description

Contents of Register BR added to ZERO Register (all zeroes) and stored in Register B. The result is clipped to 32 bits.

Example 1 NOP | **ADD(BR,ZERO,CLP32,B)** | NOP | NOP | NOP

Note: Destination is Register B (a 48-bit register).

Register	**Before Instruction**	**1 Cycle After Instruction**
BR	FFFF FFFF FFFF	FFFF FFFF FFFF
ZERO	000 0000 0000 0000 0000	000 0000 0000 0000 0000
B	FFFF FFFF FFFF	FFFF FFFF FFFF

Comment

ADD(BR,ZERO,NONE,*reg*) Add the Register BR to Register ZERO and store result in Register *reg*, where *reg* = B, L, MC, or MD.

Syntax ADD(BR,ZERO,NONE,B)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
ADD(BR,ZERO,NONE,B)	101 1000
ADD(BR,ZERO,NONE,L)	101 1001
ADD(BR,ZERO,NONE,MC)	101 1010
ADD(BR,ZERO,NONE,MD)	101 0111

Description Clip48(BR + MR) --> dest

Contents of Register BR added to contents of Register LR and stored in dest register. The result is clipped to 48 bits.

Example 1 NOP | **ADD(BR,ZERO,NONE,B)** | NOP | NOP | NOP

Note: Destination is the Accumulator (a 76-bit register).

Register	Before Instruction	1 Cycle After Instruction
BR	FFFF FFFF FFFF	FFFF FFFF FFFF
ZERO	0000 0000 0000	0000 0000 0000
B	FFFF FFFF FFFF	FFFF FFFF FFFF

Example 2 NOP | **ADD(BR,ZERO,NONE,MC)** | NOP | NOP | NOP

Note: Destination is Register MC (a 28-bit register).

Register	Before Instruction	1 Cycle After Instruction
BR	FFFF FFFF FFFF	FFFF FFFF FFFF
ZERO	0000 0000 0000	0000 0000 0000
MC	FFF FFFF	FFF FFFF

Comment

CLRACC Clear the Accumulator

Syntax CLRACC

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU2 Opfield**
CLRACC	101 1101

Description The cycle following the CLRACC instruction, the 76-bit Accumulator will contain all zeroes.

Example NOP | **CLRACC** | NOP | NOP | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
ACC	xxx xxxx xxxx xxxx xxxx	000 0000 0000 0000 0000

Comment There is also a CLRACC instruction available in the ALU1 Opfield.

COMP(reg1,reg2) Compare Register reg1 with Register reg2 (reg1 = ACC | BR; reg2 = LR | MR)

Syntax COMP(ACC,LR)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	ALU2 Opfield
COMP(ACC,LR)	010 0000
COMP(ACC,MR)	001 1000
COMP(BR,LR)	001 0000
COMP(BR,MR)	000 1000

Description

Example NOP | **COMP(ACC,LR)** | NOP | NOP | NOP

Register	Before Instruction	1 Cycle After Instruction
ACC	xxx xxxx xxxx xxxx xxxx	xxx xxxx xxxx xxxx xxxx
LR	xxxx xxxx xxxx	xxxx xxxx xxxx

Comment See Section 3.5.3.2 for a detailed description of the COMP instruction.

NOP No operation

Syntax NOP

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	**ALU2**	MOP1	AD1	MOP2	AD2	MOP3	AD3

Instruction	**ALU1 Opfield**
NOP	000 0000

Description The cycle following the NOP instruction, the state of all registers remains unchanged.

Example 1 NOP | **NOP** | NOP | NOP | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
All	xxx xxxx xxxx xxxx xxxx	Unchanged

Comment The NOP instruction is mandatory when used with PC instructions and can also be used to improve readability of code.

A.3 Memory Opcode 1 (MOP1) Instructions

MOP1 instructions provide a mechanism that allows the TAS3108/TASS3108IA programmer to load 48-bit data from data memory to registers B, L, MD, BL, or MC (28-bit register). Note that a load to BL is a simultaneous load to Register B and Register L.

The MOP1 instructions are:

- LD(D_addr,*reg*) – Unconditional load from D_addr to *reg*, where *reg* = B, L, BL, MC, or MD.
- LDC(D_addr,*reg*) – Conditional load from D_addr *reg*, where *reg* = B, L, or MD.
- LNC(D_addr,*reg*) – Conditional load from D_addr *reg*, where *reg* = B, L, or MD
- NOP – No operation

.

LD(D_addr,*reg*) Load contents of data memory address D_addr into Register *reg* (B, L, BL, MC, or MD).

Syntax LD(D_addr, B)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	**MOP1**	**AD1**	MOP2	AD2	MOP3	AD3

Instruction	**MOP1 Opfield**	**AD1 (10-Bit Data Memory Address)**
LD(D_addr,B)	0 0010	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LD(D_addr,L)	0 0011	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LD(D_addr,BL)	1 1100	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LD(D_addr,MC)	1 1111	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LD(D_addr,MD)	0 0001	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$

Description LD(D_addr,B)

Contents of data memory D_addr are loaded into destination register.

Example NOP | NOP | **LD(D_addr, B)** | NOP | NOP

Register/Data Memory Address	**Before Instruction**	**1 Cycle After Instruction**
(D_addr)	1234 5678 9ABC	1234 5678 9ABC
B	FFFF FFFF FFFF	1234 5678 9ABC

Comment

LDC(D_addr,*reg*) or **LNC(D_addr,*reg*)** Conditional load of contents of data memory address D_addr into Register *reg* (B, L, or MD).

Syntax LDC(D_addr, B)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	**MOP1**	**AD1**	MOP2	AD2	MOP3	AD3

Instruction	**MOP1 Opfield**	**AD1 (10-Bit Data Memory Address)**
LDC(D_addr,B)	0 0101	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LDC(D_addr,L)	0 0110	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LDC(D_addr,MD)	1 0100	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LNC(D_addr,B)	0 1000	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LNC(D_addr,L)	0 1001	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LNC(D_addr,MD)	0 0111	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$

Description Conditional load of data memory address D_addr into Register *reg* (B, L, or MD).

Example

```
NOP | NOP | LD(X_D, B) | NOP | NOP
NEG(B) | NOP | LD(ZERO_D, L) | NOP | NOP
THRU(L) | NOP | NOP | NOP | NOP
NOP | COMP(BR,LR)| NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | LNC(Y_D, MD) | NOP | NOP
NOP | NOP | LDC(Z_D, MD) | NOP | NOP
```

Comment See Section 3.5.3.2 for a description of the COMP instruction.

NOP No operation

Syntax NOP

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	**MOP1**	AD1	MOP2	AD2	MOP3	AD3

Instruction	**MOP1 Opfield**
NOP	0 0000

Description The cycle following the NOP instruction, the state of all registers remains unchanged.

Example 1 NOP | NOP | **NOP** | NOP | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
All	xxx xxxx xxxx xxxx xxxx	Unchanged

Comment The NOP instruction is mandatory when used with PC instructions and can also be used to improve readability of code.

A.4 Memory Opcode 2 (MOP2) Instructions

MOP2 instructions provide a mechanism that allows the TAS3108/TASS3108IA programmer to load 28-bit data from coefficient memory to Register MC and Register L.

The MOP2 instructions are:

- LD(C_addr,*reg*) – Unconditional load from C_addr to *reg*, where *reg* = MC or L
- LDC(C_addr,MC) – Conditional load from C_addr to MC
- LNC(C_addr,MC) – Conditional load from C_addr to MC
- NOP – No operation

LD(C_addr,*reg*) Load contents of coefficient memory address C_addr into Register *reg* (MC or L).

Syntax LD(C_addr, MC)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	**MOP2**	**AD2**	MOP3	AD3

Instruction	**MOP2 Opfield**	**AD2 (10-Bit Data Memory Address)**
LD(C_addr,MC)	001	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LD(C_addr,L)	110	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$

Description Contents of data memory C_addr are loaded into destination register.

Example NOP | NOP | NOP | **LD(VAR1,MC)** | NOP

Register/Data Memory Address	**Before Instruction**	**1 Cycle After Instruction**
(C_addr)	678 9ABC	678 9ABC
MC	FFF FFFF	678 9ABC

Comment

LDC(C_addr,MC) or **LNC(C_addr,MC)** Conditional load of contents of coefficient memory address C_addr into Register MC

Syntax LDC(D_addr, B)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	**MOP1**	**AD1**	MOP2	AD2	MOP3	AD3

Instruction	**MOP1 Opfield**	**AD1 (10-Bit Data Memory Address)**
LDC(C_addr,MC)	010	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
LNC(C_addr,MC)	111	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$

Description Conditional load of data memory address D_addr into Register MC

Example

NOP | NOP | LD(X_D, B) | NOP | NOP
NEG(B) | NOP | LD(ZERO_D, L) | NOP | NOP
THRU(L) | NOP | NOP | NOP | NOP
NOP | COMP(BR,LR)| NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | LNC(Y_D, MD) | NOP | NOP
NOP | NOP | LDC(Z_D, MD) | NOP | NOP

Comment See Section 3.5.3.2 for a description of the COMP instruction.

NOP No operation

Syntax NOP

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	**MOP1**	AD1	MOP2	AD2	MOP3	AD3

Instruction	**MOP1 Opfield**
NOP	000

Description The cycle following the NOP instruction, the state of all registers remains unchanged.

Example 1 NOP | NOP | NOP | **NOP** | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
All	xxx xxxx xxxx xxxx xxxx	Unchanged

Comment The NOP instruction is mandatory when used with PC instructions and can also be used to improve readability of code.

A.5 Memory Opcode 3 (MOP3) Instructions

MOP3 instructions provide a mechanism that allows the TAS3108/TASS3108IA programmer to save data from registers MD, B, L, DLYO, DI, and VOL to data and coefficient memory.

MOP3 instructions are:

- ST(*reg*,DATA,D_addr) – Store data from *reg* to DATA RAM address D_addr, where *reg* = MD, L, B, DLYO, DI
- ST(*reg*,COEF,C_addr) – Store data from *reg* to coefficient RAM address C_addr, where *reg* = MD, B, or L
- DLYPTR(*ptr*) – A special instruction used to indicate which Delay Memory pointer (*ptr*) is used for the current audio delay function
- NOP – No operation
- PCADDR(*label*) – Always used with branch instructions BOC, BNC, and JMP

ST(*reg*,DATA,D_addr) Store contents of Register *reg* (L, B, or MD) into data memory address D_addr.

Syntax ST(L,DATA,D_addr)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	**MOP3**	**AD3**

Instruction	**MOP3 Opfield**	**AD3 (10-Bit Data Memory Address)**
ST(L,DATA,D_addr)	0011	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
ST(MD,DATA,D_addr)	0001	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
ST(B,DATA,D_addr)	0010	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$

Description Contents of source register is loaded into Data memory D_addr.

Example NOP | NOP | NOP | NOP | **ST(B,DATA,VAR1_D)**

Register/Data Memory Address	**Before Instruction**	**1 Cycle After Instruction**
(VAR1_D)	XXXX XXXX XXXX	1234 5678 9123
B	1234 5678 9123	1234 5678 9123

Comment

ST(DI,DATA,D_addr) Store contents of Register DI (Data In) into data memory address D_addr.

Syntax ST(DI,DATA,D_addr)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	**MOP3**	**AD3**

Instruction	**MOP3 Opfield**	**AD3 (10-bit Data Memory Address)**
ST(DI,DATA,D_addr)	0110	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$

Description Register DI (Data In) is a special register used to access audio data received by the input SAP. The input SAP can receive up to eight channels. The ST(DI,DATA,D_addr) instruction copies data from the input SAP into data memory. Note that the lower three LSBs of the data memory address indicate which input channel is copied to memory. The three LSBs mapping is SDIN1-L (000), SDIN1-R (001), SDIN2-L (010), SDIN2-R (011), SDIN3-L (100), SDIN3-R (101), SDIN4-L (110), and SDIN4-R (111).

Example

NOP | NOP | NOP | NOP | **ST(DI,DATA,SDIN1L_D)**

NOP | NOP | NOP | NOP | **ST(DI,DATA,SDIN1R_D)**

.....

NOP | NOP | NOP | NOP | **ST(DI,DATA,SDIN4R_D)**

Register / Memory Address	**Before Instruction**	**1 Cycle After Instruction**
(SDIN1L_D)	XXXX XXXX XXXX	SDIN1-L (Ch 1) Audio Data
(SDIN1R_D)	XXXX XXXX XXXX	SDIN1-R (Ch 2) Audio Data
.....	.....	.....
(SDIN4R_D)	XXXX XXXX XXXX	SDIN4-R (Ch 8) Audio Data

Comment

ST(DLYO,DATA,D_addr) Store contents of Register DLYO (Delay Memory Out) into data memory address D_addr.

Syntax ST(DLYO,DATA,D_addr)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	**MOP3**	**AD3**

Instruction	**MOP3 Opfield**	**AD3 (10-Bit Data Memory Address)**
ST(DLYO,DATA,D_addr)	0100	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$

Description Register DLYO (Delay Memory Out) is a special register used to output delayed audio samples from the delay memory. There must be 13 cycles between the delay pointer instruction (DLYPTR) and the ST(DLYO,DATA,D_addr) instruction. These 13 cycles can contain NOPs or processing as required.

Example

NOP | CLRACC | LD(DelayIn1_D;L;NONE) | NOP | NOP
THRU(L) | NOP | NOP | NOP | NOP
NOP | ADD(ACC;LR;NONE;DLYI) | NOP | NOP | DLYPTR(0)
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | **ST(DLYO,DATA,DelayOut1_D)**

Comment

ST(*reg*,COEF,C_addr) Store lower 28-bits of Register *reg* (L, B or MD) into coefficient memory address C_addr.

Syntax ST(L,COEF,C_addr)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	**MOP3**	**AD3**

Instruction	MOP3 Opfield	AD3 (10-Bit Data Memory Address)
ST(L,COEF,C_addr)	1001	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
ST(MD,COEF,C_addr)	0111	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$
ST(B,COEF,C_addr)	1000	$a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$

Description Contents of source register is loaded into coefficient memory C_addr.

Example NOP | NOP | NOP | NOP | **ST(L,COEF,VAR1)**

Register/Coefficient Memory Address	Before Instruction	1 Cycle After Instruction
(VAR1)	XXX XXXX	678 9123
L	0000 0678 9123	0000 0678 9123

Comment

DLYPTR(*ptr*) Access delay memory pointer *ptr*

Syntax DLYPTR(0)

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	**MOP3**	**AD3**

Instruction	MOP3 Opfield AD3 (10-Bit Delay Memory Pointer ptr)	
DLYPTR(ptr)	0000	$p_9p_8p_7p_6p_5p_4p_3p_2p_1p_0$

Description DLYPTR(ptr) is a special instruction used to indicate which delay memory pointer is used for the current audio delay function.

Example

```
NOP | CLRACC | LD(DelayIn1_D;L;NONE) | NOP | NOP
THRU(L) | NOP | NOP | NOP | NOP
NOP | ADD(ACC;LR;NONE;DLYI) | NOP | NOP | DLYPTR(0)
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | NOP
NOP | NOP | NOP | NOP | ST(DLYO,DATA,DelayOut1_D)
```

Comment

NOP No operation

Syntax NOP

Opcode

53 49	48 42	41 37	36 27	26 24	23 14	13 10	9 0
ALU1	ALU2	MOP1	AD1	MOP2	AD2	**MOP3**	**AD3**

Instruction	**MOP3 Opfield**
NOP	0000

Description The cycle following the NOP instruction, the state of all registers remains unchanged.

Example 1 NOP | NOP | NOP | **NOP** | NOP

Register	**Before Instruction**	**1 Cycle After Instruction**
All	xxx xxxx xxxx xxxx xxxx	Unchanged

Comment The NOP instruction is mandatory when used with PC instructions and can also be used to improve readability of code.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265