MUSIC SEMICONDUCTORS

APPLICATION NOTE AN - N2

Using The CAMSim™ Workbench Simulator

by Ray Parry and Mike Clausen

1.0 INTRODUCTION

The CAMSim Workbench is a software program that simulates the functional operation of the MUSIC's Mu9C1640 CacheCAM and MU9C1480 LANCAM Content-addressable Memory devices. CAMSim allows the user to execute the instruction set manually or by using a data file and to monitor the changes in the internal CAM memory array and registers, as well as the resulting data and match outputs.

What You Need to Get Started

- A copy of the CAMSim Workbench (Available on a 3.5" diskette)
- This Application note
- Either the *MU9C1640 CacheCAM Data Sheet* or the *MU9C1480 LANCAM Handbook*

CAMSim runs under Windows 3.1 on a 386 or 486 Compatible computer with a minimum of 2MB RAM and a mouse.

Disk Contents

README.TXT
SETUP.EXE
SETUP.INF
INIT.DAT*
INCRE.DAT*
INPUT.DAT*
RESET.DAT*
STATUS.DAT*
TRANSVAR.CWB
FLAGS.CWB
IRX.CWB
REG.CWB
VECTORX.CWB
CWB.EXE
CWB.HLP
METER.DLL

(* indicates sample files)

2.0 INSTALLATION

The installation of the CAMSim under Windows 3.1 is performed automatically by the Setup program on the distribution disk, with all the files stored to a directory named CAMSim by default. Since the files on the distribution disk are not compressed, CAMSim can also be installed by copying all the files directly to a directory on a hard disk and executing it from Windows File Manager. Finally, the program will run directly from the floppy if desired. The disk is not copy protected, and a backup copy should be made and the original disk stored for safe-keeping.

Installation Under Windows 3.1

Step 1: Turn on the computer. Select Windows by either having it called in your autoexec.bat file on boot-up, or by typing "win" from your DOS prompt.

Step 2: When the Windows Program Manager screen is displayed, insert the CAMSim Workbench distribution disk into the floppy drive. With your mouse, click and drag on File and release on Run. At the Run dialog box, type in "A:\SETUP" (assuming that the CAMSim disk is in the A: drive), and hit <return>, or click on the OK box.

Step 3: A CAMSim Workbench Dialog box will appear to inform you that the Setup program will create a directory called C:\CAMSIM and install the simulator to it unless you type in a different directory of your own choosing. Enter <return> or click on the OK box when you are satisfied with the directory name, and the Setup program will proceed with installation.

Step 4: A thermometer will keep you informed of the installation progress, and if it is successful, a dialog box will appear to inform you of that. Hit <return> or click on the OK box to return to the Program Manager.

Step 5: You'll see that the Setup program created a Program Group called CAMSim Workbench with the MUSIC icon in it. You can move the CAMSim icon to another Program Group, such as Main or Applications by clicking and dragging it with the mouse (and then selecting and deleting the CAMSim Workbench Program Group) or, by selecting Window Tile, arrange the screen to incorporate the CAMSim Workbench as part of the Windows display. The icon is a pointer to where the CAMSim Workbench files are stored in the directory structure. The CAMSim Workbench is now installed to operate under Windows.

3.0 USING THE LANCAM WORKBENCH

To use the CAMSim Workbench, double-click on the CAMSim Workbench icon, and then click on the OK box in the copyright dialog box that will appear. The Workbench will appear with the following Pull-Down menus: File, Edit, Search, Window, Commands, Run, and Help. To display the simulator itself, click on the Run menu and pull down to CAMSim Simulator, wAhich will bring up the simulator screen with all the CacheCAM/LANCAM registers and memory locations

displayed and filled with "X"s to indicate that the values are unknown. The simulator operation is controlled by the six buttons to the right: File..., Run..., Pause, Next, Execute, and Restart. Clicking on the Maximize (^) button in the upper right corner of the CAMSim window displays the Comparand and Mask registers at the bottom of the screen.

The Menus

The File menu allows you to open a New edit window for a data file (See data file preparation). It also allows you to Open an existing data file edit window for revising or generating vectors. Vector generation for use with the simulator is done on an open data file edit window. Multiple data files can be open at the same time, but only the currently active file is available for vector generation. You need to Save your data file with a .dat extension for future use.

The Edit menu provides the standard Windows edit tools to use with your data file edit window.

The Search menu provides the standard Find and Replace edit tools to use with your data file edit window.

The Window menu allows you to select between your current open windows.

The Commands menu provides the CacheCAM/LANCAM command set for insertion into the simulator window or into edit windows.

The Run menu allows you to bring up the CAMSim Simulator, or to Generate Vectors from an active data file edit window. It also permits selecting Logging, which will generate a .log file of a simulation.

The Help menu will give you access to the Windows help library. Refer to the CacheCAM or the LANCAM Handbook for more information.

The Registers

At the top of the screen are displayed the internal registers of the device: the 16-bit Page Address, Device Select, Address, Instruction, Control, and Segment Contol registers, the 10-bit Next Free Address register and the 32-bit Status register. When the Window Maximize button is pressed, the bottom of the screen displays the Comparand register and the two Mask registers, which are 64 bits in width. Each bit can be displayed as an "X", indicating an "unknown" state of the bit at power-up, or a "1" or "0" after having been set by loading information into the register.

The CAM Memory Array

The Memory Array Display ("CAM") at the left side of the screen is composed of 1024 entries, displayed as 4 groups of 4-digit hexadecimal words representing the 64 bits of stored information. Each group represents a 16-bit segment of the memory, with the Most Significant word at the left, and the Least Significant word at the right. Each memory entry is followed by three bits, denoted as "s e m." The first bit in this group is the "Skip" bit, which is a "0" if the entry is not skipped. The second bit is the "Empty" bit, which is a "1" if the entry is empty. The third bit represents the internal "Match" line for that entry, and is a "1" if that particular entry was a match during an automatic or forced compare operation. At power-up, each bit in memory is an "X" to indicate its unknown state. When loaded with data, the memory digits become hexadecimal characters (0-F), and the "s e m" bits are "0" or "1."

The Input Display

The Inputs to the device are displayed as two boxes for (Command) Type and (Command or) Data Bus value, and as three Input Control Flags (/MI, /FI, /EC) for setting the two Flag inputs and the Enable Compare input, over to the right of the Command input boxes. The Command Type is the left-most box, and can be either "Command_write," "Command_read," "Data_write," "Data_read" or "Dummy." This box is loaded in one of five ways: manually clicking on the box and typing in the Command type; pulling down the Commands menu and selecting a command, which will automatically fill in the Command type as a "Command-write"; pulling down the Commands menu and selecting User Defined, which allows you to select a Command type; clicking and dragging on the arrow box next to the Command Type input box; or by reading the Command type from a vector file.

The Command or Data Bus value box is where the actual instruction, address, or data value is input. Again, input to this box is by a variety of methods: manually clicking on the box and typing in the desired instruction; pulling down the Commands menu and selecting an instruction, which will automatically fill in the Command or Data Bus value box; pulling down the Commands menu and selecting User Defined, which allows you to select a Command type and then replacing the "0x0000" that automatically appears in the Data Bus value box with the desired value; or by reading the Instruction or Data Bus value from a vector file. Input data is represented in Hexadecimal form, with the "0x" being a prefix denoting Hexadecimal.

Check boxes are provided to change the default settings for the /MI, /FI, or /EC pins. When an instruction is selected from the Commands menu, these values are automatically set to /MI=HIGH, /FI=LOW, and /EC=HIGH or LOW. It is important to check that the proper value of /EC is loaded with each instruction. Normally, /EC is HIGH except when you want to do a compare on data and then needs to be over-ridden to LOW on the last data segment input to the Comparand register to allow the daisy chain to reflect the match output. In these boxes, a HIGH is indicated by an "X" and a LOW by a blank box. Clicking on the box will reverse the setting.

The Output Display

On the lower left of the screen are two boxes, one labeled /MF and the other /FF. These two display the outputs of the /MF and /FF pins. If an "X" is displayed in a box, the output is HIGH. If it's blank, the output is LOW. If these levels go LOW, they are also stored in the History Log, and on the .log file, if that is enabled. To the right of these boxes, under the input Data Bus are 16 bits representing the output data bus. In the CacheCAM and LANCAM, the input and output data busses are the same bi-directional three-state bus. When data is read out, it will be displayed here in binary form, as "1"s, "0"s, or as "+"s when the

bus is high impedance or Three-state. This data will also be shown in the History Log in Hexadecimal form. If the output was Three-state, that will be displayed as "0xxxxx."

The History Log and ".log" Files

On the right of the screen under the Push buttons is a small box with scrolling capability. In this box are stored the last 20 instructions given the device along with any data output during reads as well as the /MF and /FF symbols which, if recorded, indicate that those flags went LOW during the given operation. The most recent instruction is at the top of the list, and using the scroll box will allow you to view the preceeding instructions. If you want to log the entire command sequence, selecting Logging from the Run menu will open a dialog box requesting a filename with the extension ".log." Type in a filename.log and the simulator will store the entire list of instructions and outputs from a lengthy command sequence. Also stored in the file are screen updates which include all changes to the registers. Additionally, all changes to the memory array that are visible on the display are recorded. The filename.log file can be viewed by opening an edit window and typing in filename.log. It can also be viewed from DOS edit by typing "EDIT filename.log."

The Push Buttons

On the right side of the screen are six push buttons that control that operation of the simulator. These are activated by clicking on them with the mouse.

File... The File button brings up a window that asks you to select a vector file for running on the simulator. It displays the list of .vec files currently in the directory. You can either type the name in, or double-click on the file name you want from the list displayed. A .vec file must be selected for the simulator to run prior to pressing the Run... button.

Run... The Run... button executes the vector file loaded above. The execution of instructions will proceed automatically unless halted by encountering an error or a Stop instruction in the original data file; pressing the Pause button; or reaching the end of file. If interrupted by an error, or an instruction that can't be processed, a dialog box will display an error message and ask whether you want to continue by pressing OK or Cancel. Clicking on Cancel will stop the execution so you can examine the history of instructions in the History Log and the present state of the simulated device registers and outputs. You can then continue the execution step by step using the Next and Execute buttons, or in a continuous stream by clicking on the Run... button. Clicking on OK will continue with the instruction execution.

Pause The Pause button stops the execution of instructions started by the Run... button. This allows you to look at the history of instructions in the History Log, the state of the simulated device registers and outputs, and if desired, to proceed one instruction at a time by using the Next and Execute buttons. Pressing the Run... button will continue the instruction execution in a continuous stream.

Next The Next button reads in the next instruction and needs to be followed by pressing the Execute button, which will allow you to single step through an instruction list in a vector file.

Execute The Execute button causes an instruction to be executed in the simulator. The instruction can come from either a vector file loaded into the simulator with the Next button, or from entering it into the input data boxes, either manually or from the Commands menu.

Restart The Restart button returns the simulator to its power-up state, where all the registers and memory locations are represented by "X"s, indicating unknown states.

4.0 A SAMPLE SIMULATION

The simulator has two ways to load instructions and data to exercise the device: manually one instruction at a time, or through input files that allow you to give the simulated device a series of instructions that can either be executed one at a time, or in a continuous string. To get a feel for how the simulator performs, open one of the data files by clicking and pulling down File Open. The dialog box will display a list of the available data files in the C:\camsim directory. You can either double-click on one of the files listed, change directories by double-clicking on one in the directory column, or typing in a path and file (with the extension .dat). Double-clicking on "init.dat" will bring up an Edit window with the "init.dat" file displayed. Each line displays an instruction to be issued to the LANCAM with the Command Type (e.g., "CW"), the Command Code or Data (e.g., "TCO_DS" or "0X0000"), followed by three input signals. You can scroll through the list of instructions by clicking and dragging on the scroll box or clicking in the scroll bar. Note that spaces are used in an instruction to separate the various parts. Note also that For-loop sections have a different format.

The simulator uses vector files to input instructions that are generated from these data files. To generate the "init.vec" file from the "init.dat" file, click and pull-down the Run Generate Vectors menu. A dialog box appears giving you a running status on the generation of the vector file from the open data file. Click on the OK box when "Success" appears next to "Status." You can close the data file if you want by clicking and dragging on the File Save menu, or you can keep it active and switch to the simulator window by clicking on it. At this point, you need to select the "init.vec" file to load the LANCAM by clicking on the File... button, which brings up the File Open dialog box. Again, the vector file is selected by double-clicking on it, changing directories, or typing it in directly. Click on "init.vec" to load that file into the simulator.

At this point, you can choose to step through the instruction list in the "init.dat" ("init.vec") file one line at a time or in a continuous string. To step through, click on the Next button. This loads the first instruction into the Input Type and Data Bus fields. Clicking on the Execute button causes this instruction (command_write SPS_CR) to load into the Instruction Register as all zeroes (the Op Code for an SPS_CR is "0x0000," where the 0x prefix indicates a Hexadecimal number; without the 0x prefix, the number would be considered decimal). Clicking the Next button again brings up the next instruction (command_write TCO_DS) and clicking the Execute button shows the change to the Instruction register. Note that both

instructions are listed in a window in the lower right that keeps a record of the last 20 instructions. Logging can also be turned on to keep a record of all the instructions issued and data output, as well as other useful information.

At this point, to complete executing the file in a continuous string, click on the Run... button. As you can see, the instructions load the first 50 entries of the CAM memory with data. After loading the data, different data is loaded into the Comparand register, and when the compare happens, no match occurs, resulting in the Warning dialog box, that the Data Move instruction is ignored. This is because no match occured with /EC taken LOW. If you now click on Cancel, the simulator is paused so that you can review the instruction list up to this point. If you click on OK, the execution of the instruction list continues as before. A Warning box appears when the last instruction is executed. Clicking on OK returns you to the simulator, ready for additional instructions to be entered manually or by another file.

5.0 DATA FILE PREPARATION

To simplify running longer routines, data files can be prepared by using any editor, including the Windows editor. Clicking and dragging on File New will bring up the Edit Windows dialog box, with a new, unnamed file ready for entering instructions. The instruction format for these files is:

Command_type Command/Data Flags (MI FI EC)

with each field separated by one or more spaces. A typical instruction might look like this:

cw SPD_M@[AR],V MI_HI FI_LO EC_HI

Review Appendix A for a sample of a properly formatted data file for Windows. These files are stored as filename.dat. To be turned into vector files for use by the simulator, click and drag down to Generate Vectors on the Run menu, and when the dialog box appears, double-click on the filename you have assigned. A vector file will be generated having the same filename but with a .vec extension.

Command Type

Command_types correspond to the type of cycle you want the simulated device to execute, and correspond to the levels of the input pins /CM and /W.

Command_type	/CM	/W	Abbreviation
Command_write	LO	LO	CW or cw
Command_read	LO	HI	CR or cr
Data_write	HI	LO	DW or dw
Data_read	HI	HI	DR or dr

For an instruction line, one of the above needs to start the line. Abbreviations can be used. If using the Windows editor, clicking and dragging on the Commands User Defined menu, will bring up another menu where the first four are available. Moving the mouse to one of them and then releasing the button will insert that command_type into the edit window at the cursor location.

The Command_or_Data field will be filled with "0x0000" which you can replace with either the Instruction you desire for a Command_write, or the data you want for a Data_write. For Command_reads or Data_reads, the input value in this field is ignored (but a value must be in the field).

Command or Data

The next field contains the command to be written to the simulated device. The allowable command set is listed in the CacheCAM or LANCAM documentation, and is also available on the pull-down Commands menu. It is important that the data file commands be written exactly as shown in Appendix B for proper parsing of the instruction by the vector generation program. Clicking and dragging to the command you want from the menu bar will insert it in the edit window at the cursor location to save you having to type each one. Also inserted will be the Command type, and the three input Flags in their typical settings. These Flag settings may need to be changed as appropriate, but the order they are in must be as in the examples.

Input Control Flags

The Input Control Flags can be either typed into the data file by hand, or inserted in a default form when you click and drag to a command from the Command menu. Check to see that the flag conditions are the ones you want, particularly the state of EC.

MI = The state of the Match Input pin. MI_HI means that either this is a single chip with the /MI pin tied HIGH, or if it's in a string of chips, that no chip ahead of it in the daisy-chain has a match, thus selecting this chip for detecting Highest-priority Matches. If MI were LOW (MI_LO), it would indicate that a prior chip has a match indicated, and that chip would receive Highest-priority Match operations. Normally, you would set MI_HI.

FI = The state of the Full Input pin. FI_LO means that either this is a single chip with the /FI pin tied LOW, or if it's in a string of chips, that all the chips ahead of it in the daisy-chain are full, thus selecting this chip for Next Free Address operations. If FI were HIGH (FI_HI), that would indicate that a prior chip has empty locations, and that chip would receive Next Free Address operations. Normally, you would set FI_LO.

EC = The state of the /EC input pin. This pin enables the /MF output pin to reflect the presence of the detection of an internal match. If /EC is HIGH (EC_HI), then /MF only reflects the state of /MI, and any internal match detected on this chip is not shown on the /MF pin. If /EC is LOW (EC_LO), then /MF will go LOW if an internal match is detected. /EC also controls the daisy-chain locking mechanism, whereby only the Highest-priority chip can respond after a match is detected. Thus, normally, you only take /EC LOW on the last data segment loaded into the Comparand register, so that the results of the automatic compare can be shown on the /MF pin. If you are not using hardware flags in your application, but only reading the status register for the results of a compare, you might wish to keep /EC HIGH.

A NOP instruction allows /EC to be taken HIGH after a "no-match," for example, to unlock the daisy chain without altering any register or memory contents.

Other Instructions

The following additional instructions are similar to those used in "C," and will help in data file preparation and documentation.

! or /* These indicate comment lines. Comments must be on separate lines from instructions.

#define This defines a constant to be used in the routine. Each constant must be separately defined. The format is as follows:

```
#define MEMSIZE
```

variable or, **var** This defines variables, one per line, to be used within for...loops, as in:

```
var pattern
FOR pattern = 0 LT 10
dw  pattern MI_HI FI_LO EC_HI
LOOP
```

for...loop For...loops are like Do loops or For...next statements. Here, "i" does not need to be pre-defined as a variable, since it's not used within the loop. The format for a For...loop is as follows: (each item in the "for" statement must be separated by spaces)

```
for   i = n  condition m
instructions
loop
```

where i is any variable, defined or not, n is a start constant, m is a finish constant , and condition is one of: LT (Less Than), GT (Greater Than), LE (Less Than or Equal), and GE (Greater Than or Equal). For...loops cannot be nested.

repeat *n* This is used to repeat a command *n* times, as in:

```
repeat 4 dw 0x0000 MI_HI FI_LO EC_HI
```

which will do four data writes of all zeroes. These can be combined to fill the CAM memory by doing:

```
for i = 0 LE 1023
repeat 4 dw 0x0000 MI_HI FI_LO EC_HI
loop
```

stop This inserts a breakpoint in the file simulation. The simulation will halt and a dialog box will appear giving you the choice to continue by pressing "OK," or to stop by pressing "Cancel."

6.0 ERROR MESSAGES

"Error, couldn't parse xxxxxxxxx."

During vector generation, the parser had difficulty with a specific entry in the filename.dat file. Check the command against the listing in Appendix B.

"Bits n and m in the control register are set illegally." or "Control register bits n and m are set illegally."

You need to check your setting of the Control Register during a TCO_CT command for proper selection.

"The xxxxxxxxxx instruction is ignored."

This would indicate that the simulator expected a different next instruction, such as a value following a two-step command, or that a no-match was found after a Compare, so that the chip can't respond until you unlock it by taking /EC HIGH.

"There is no match."

The command issued or command sequence was not recognized by the simulator as proper or expected under the existing conditions.

"Warning, a Data Read or Data Write cycle was interrupted before the Segment count was complete."

This indicates that the Segment counter had not finished the incrementing the segment count before it was interrupted. This is allowable, but it means that it will resume counting where it left off at the next DW or DR cycle, rather than resetting itself, with erroneous segment reading or writing likely.

"No more vectors in this file."

You have reached the end of the filename.vec input file to the simulator. You may now execute instructions step-by-step, load another vector file for execution from this point, or start over by clicking on the Restart button.

"Need to open a file for reading first."

The simulator needs a filename.vec input file to be selected for execution. Click on the File... button and select a .vec file by double-clicking on it, or typing its name into the entry box and hitting return, before clicking on Run....

"Conversion to decimal overflow." or "Conversion to binary with negative number. Integers must be less than 32767."

Constants and variables must be positive integers less than 32767.

"Not enough memory to run CAMSim. Try closing other applications."

You need 2 Megabytes of RAM or greater to run Windows and the simulator at the same time.

7.0 WHO TO CALL FOR HELP

For assistance in running the simulator, or information on the MUSIC Semiconductors' MU9C1480 LANCAM or MU9C1640 CacheCAM, please call the MUSIC sales office near you.

Appendix A, Listing of Sample File "init.dat"

```
CW 0x0000           MI_HI   FI_LO   EC_HI   ! Initializes internal State machine
CW TCO_DS           MI_HI   FI_LO   EC_HI   ! Targets Device Select register
CW 0xFFFF           MI_HI   FI_LO   EC_HI   ! Selects all chips
CW TCO_CT           MI_HI   FI_LO   EC_HI   ! Targets Control register
CW 0x0000           MI_HI   FI_LO   EC_HI   ! Resets chip
CW TCO_PA           MI_HI   FI_LO   EC_HI   ! Targets Page Address register
CW 0xABCD           MI_HI   FI_LO   EC_HI   ! Sets chip for address 'ABCD'
CW TCO_CT           MI_HI   FI_LO   EC_HI   ! Targets Control register
CW 0x8040           MI_HI   FI_LO   EC_HI   ! Sets 48 bits CAM
CW TCO_SC           MI_HI   FI_LO   EC_HI   ! Targets Segment Control register
CW 0x18C0           MI_HI   FI_LO   EC_HI   ! Sets for 4 segment writes and reads
CW SPD_M@[AR],V     MI_HI   FI_LO   EC_HI   ! Sets Pers. Dest. to Mem @ Add. reg
FOR i=0 LT 50                               ! This For...loop fills the first 50 locations
REPEAT 4 DW 0x1010  MI_HI   FI_LO   EC_HI   ! in the memory array with a '1010' pattern.
LOOP                                        ! 'Repeat 4' fills all four segments
CW TCO_SC           MI_HI   FI_LO   EC_HI   ! Targets Segment Counter register
CW 0x39C9           MI_HI   FI_LO   EC_HI   ! Sets for 3 segment writes and reads
CW TCO_DS           MI_HI   FI_LO   EC_HI   ! Targets Device Select register
CW 0xABCD           MI_HI   FI_LO   EC_HI   ! Selects chip with address 'ABCD'
CW SPD_CR           MI_HI   FI_LO   EC_HI   ! Sets Pers. Dest. to Comparand reg.
DW 0x0000           MI_HI   FI_LO   EC_HI   ! Writes 3 segments of network data
DW 0x0000           MI_HI   FI_LO   EC_HI   ! of all zeroes, takin g /EC LOW on
DW 0x0000           MI_HI   FI_LO   EC_LO   ! last segment to lock daisy chain.
CW MOV_HM,CR,V      MI_HI   FI_LO   EC_HI   ! If a match (which there isn't), move
                                            ! the Comparand to the matching address
CW MOV_NF,CR,V      MI_HI   FI_LO   EC_HI   ! Move the Comparand to the Next Free address
DW 0x1010           MI_HI   FI_LO   EC_HI   ! Writes 3 segments of network data
DW 0x1010           MI_HI   FI_LO   EC_HI   ! of 1010 to the Comparand reg.
DW 0x1010           MI_HI   FI_LO   EC_LO   ! taking /EC LOW on last segment.
CW MOV_HM,CR,V      MI_HI   FI_LO   EC_HI   ! Moves the Comparand to the Highest-priority
                                            ! match address.
CW MOV_NF,CR,V      MI_HI   FI_LO   EC_HI   ! Moves the Comparand to the Next Free address
CR 0x0000           MI_HI   FI_LO   EC_HI   ! Reads Status register bits 15-0
CR 0x0000           MI_HI   FI_LO   EC_HI   ! Reads Status register bits 31-16
```

Note: Comments were put adjacent to instructions for clarity, rather than on a separate line as the simulator requires.

Set Persistent Source
SPS_CR = 0x0000;
SPS_MR1 = 0x0001;
SPS_MR2 = 0x0002;
SPS_M@[AR] = 0x0004;
SPS_M@aaaH = 0x0804;
SPS_M@HM = 0x0005;

Set Persistent Destination
SPD_CR = 0x0100;
SPD_CR[MR1] = 0x0140;
SPD_CR[MR2] = 0x0180;
SPD_MR1 = 0x0108;
SPD_MR2 = 0x0110;
SPD_M@[AR],V = 0x0124;
SPD_M@[AR][MR1],V = 0x0164;
SPD_M@[AR][MR2],V = 0x01A4;
SPD_M@[AR],E = 0x0125;
SPD_M@[AR][MR1],E = 0x0165;
SPD_M@[AR][MR2],E = 0x01A5;
SPD_M@[AR],S = 0x0126;
SPD_M@[AR][MR1],S = 0x0166;
SPD_M@[AR][MR2],S = 0x01A6;
SPD_M@[AR],R = 0x0127;
SPD_M@[AR][MR1],R = 0x0167;
SPD_M@[AR][MR2],R = 0x01A7;
SPD_M@aaaH,V = 0x0924;
SPD_M@aaaH[MR1],V = 0x0964;
SPD_M@aaaH[MR2],V = 0x09A4;
SPD_M@aaaH,E = 0x0925;
SPD_M@aaaH[MR1],E = 0x0965;
SPD_M@aaaH[MR2],E = 0x09A5;
SPD_M@aaaH,S = 0x0926;
SPD_M@aaaH[MR1],S = 0x0966;
SPD_M@aaaH[MR2],S = 0x09A6;
SPD_M@aaaH,R = 0x0927;
SPD_M@aaaH[MR1],R = 0x0967;
SPD_M@aaaH[MR2],R = 0x09A7;
SPD_M@HM,V = 0x012C;
SPD_M@HM[MR1],V = 0x016C;
SPD_M@HM[MR2],V = 0x01AC;
SPD_M@HM,E = 0x012D;
SPD_M@HM[MR1],E = 0x016D;
SPD_M@HM[MR2],E = 0x01AD;
SPD_M@HM,S = 0x012E;
SPD_M@HM[MR1],S = 0x016E;
SPD_M@HM[MR2],S = 0x01AE;
SPD_M@HM,R = 0x012F;
SPD_M@HM[MR1],R = 0x016F;
SPD_M@HM[MR2],R = 0x01AF;
SPD_M@NF,V = 0x0134;
SPD_M@NF[MR1],V = 0x0174;
SPD_M@NF[MR2],V = 0x01B4;
SPD_M@NF,E = 0x0135;
SPD_M@NF[MR1],E = 0x0175;
SPD_M@NF[MR2],E = 0x01B5;
SPD_M@NF,S = 0x0136;
SPD_M@NF[MR1],S = 0x0176;
SPD_M@NF[MR2],S = 0x01B6;
SPD_M@NF,R = 0x0137;
SPD_M@NF[MR1],R = 0x0177;
SPD_M@NF[MR2],R = 0x01B7;

Temporary Command Override
TCO_CT = 0x0200;
TCO_PA = 0x0208;
TCO_SC = 0x0210;
TCO_NF = 0x0218;
TCO_AR = 0x0220;
TCO_DS = 0x0228;
TCO_PS = 0x0230;
TCO_PD = 0x0238;

Move
MOV_CR,MR1 = 0x0301;
MOV_CR,MR2 = 0x0302;
MOV_CR,[AR] = 0x0304;
MOV_CR,[AR][MR1] = 0x0344;
MOV_CR,[AR][MR2] = 0x0384;
MOV_CR,aaaH = 0x0B04;
MOV_CR,aaaH[MR1] = 0x0B44;
MOV_CR,aaaH[MR2] = 0x0B84;
MOV_CR,HM = 0x0305;
MOV_CR,HM[MR1] = 0x0345;
MOV_CR,HM[MR2] = 0x0385;
MOV_MR1,CR = 0x0308;
MOV_MR1,MR2 = 0x030A;
MOV_MR1,[AR] = 0x030C;
MOV_MR1,aaaH = 0x0B0C;
MOV_MR1,HM = 0x030D;
MOV_MR2,CR = 0x0310;
MOV_MR2,MR1 = 0x0311;
MOV_MR2,[AR] = 0x0314;
MOV_MR2,aaaH = 0x0B14;
MOV_MR2,HM = 0x0315;
MOV_[AR],CR = 0x0320;
MOV_[AR],CR[MR1] = 0x0360;
MOV_[AR],CR[MR2] = 0x03A0;
MOV_[AR],MR1 = 0x0321;
MOV_[AR],MR2 = 0x0322;
MOV_[AR],CR,V = 0x0324;
MOV_[AR],CR[MR1],V = 0x0364;
MOV_[AR],CR[MR2],V = 0x03A4;
MOV_[AR],MR1,V = 0x0325;
MOV_[AR],MR2,V = 0x0326;
MOV_aaaH,CR = 0x0B20;
MOV_aaaH,CR[MR1] = 0x0B60;
MOV_aaaH,CR[MR2] = 0x0BA0;
MOV_aaaH,MR1 = 0x0B21;
MOV_aaaH,MR2 = 0x0B22;
MOV_aaaH,CR,V = 0x0B24;
MOV_aaaH,CR[MR1],V = 0x0B64;
MOV_aaaH,CR[MR2],V = 0x0BA4;
MOV_aaaH,MR1,V = 0x0B25;
MOV_aaaH,MR2,V = 0x0B26;
MOV_HM,CR = 0x0328;
MOV_HM,CR[MR1] = 0x0368;
MOV_HM,CR[MR2] = 0x03A8;
MOV_HM,MR1 = 0x0329;
MOV_HM,MR2 = 0x032A;

MOV_HM,CR,V	= 0x032C;	VBC_aaaH,R	= 0x0C27;
MOV_HM,CR[MR1],V	= 0x036C;	VBC_HM,V	= 0x042C;
MOV_HM,CR[MR2],V	= 0x03AC;	VBC_HM,E	= 0x042D;
MOV_HM,MR1,V	= 0x032D;	VBC_HM,S	= 0x042E;
MOV_HM,MR2,V	= 0x032E;	VBC_HM,R	= 0x042F;
MOV_NF,CR	= 0x0330;	VBC_ALM,V	= 0x043C;
MOV_NF,CR[MR1]	= 0x0370;	VBC_ALM,E	= 0x043D;
MOV_NF,CR[MR2]	= 0x03B0;	VBC_ALM,S	= 0x043E;
MOV_NF,MR1	= 0x0331;	VBC_ALM,R	= 0x043F;
MOV_NF,MR2	= 0x0332;		
MOV_NF,CR,V	= 0x0334;	Compare	
MOV_NF,CR[MR1],V	= 0x0374;	CMP_V	= 0x0504;
MOV_NF,CR[MR2],V	= 0x03B4;	CMP_E	= 0x0505;
MOV_NF,MR1,V	= 0x0335;	CMP_S	= 0x0506;
MOV_NF,MR2,V	= 0x0336;	CMP_R	= 0x0507;
Validity Bit Control		Set Full Flag	
VBC_[AR],V	= 0x0424;	SFF	= 0x0700;
VBC_[AR],E	= 0x0425;		
VBC_[AR],S	= 0x0426;	No Operation	
VBC_[AR],R	= 0x0427;	NOP_1	= 0x0300;
VBC_aaaH,V	= 0x0C24;	NOP_2	= 0x0309;
VBC_aaaH,E	= 0x0C25;	NOP_3	= 0x0312;
VBC_aaaH,S	= 0x0C26;		

Appendix B, Command List with Op Codes, Continued

MUSIC Semiconductors®

USA Headquarters
MUSIC Semiconductors
254 B Mountain Avenue
Hackettstown, New Jersey 07840
USA
Tel: 908/ 979-1010
Fax: 908/ 979-1035

Asian Headquarters
MUSIC Semiconductors
Special Export Processing Zone
Carmelray Industrial Park
Canlubang, Calamba, Laguna
Philippines
Tel: +63 92 549 1480
Fax: +63 92 549 1024
Sales Tel/Fax: +632 723 62 15

European Headquarters
MUSIC Semiconductors
Torenstraat 28
6471 JX Eygelshoven
The Netherlands
Tel: +31-45-5462177
Fax: +31-45-5463663

http://www.music.com

MUSIC Semiconductors' agent or distributor: