# The MUSIC_CAM™ Device Models

## 1.0 INTRODUCTION

The MUSIC_CAM models are behavioral VHDL or Verilog functional models of individual devices in the MUSIC Semiconductors LANCAM® family. VHDL model packages include a CAM model, a workbench model, and a CAM stimulus driver model. These three models combine to provide the user with the means to develop CAM instruction sequences that can be tested against the model from the provided stimulus driver or some other system level device. The Verilog and VHDL device models may be used as components for board simulations or as reference models while developing custom ASICs to drive the MUSIC CAMs. (Note that the VHDL workbench model and CAM stimulus driver model can be supplied with the Verilog model upon special request.)

The MUSIC_CAM models have been designed to perform setup and hold timing checks (except 0 setup times) as well as generate the correct timing for device output signals. Timing verification may be enabled or disabled by setting a single variable, allowing CAM instruction sets to be tested before timing is considered.

### What You Need to Get Started

To receive a MUSIC_CAM model, your company must sign a software license with MUSIC Semiconductors. The model will then be shipped as source code on a 3.5" HD floppy disk in UNIX tar format. (Note that other media and formats may be available on special request and should be specified when placing an order.)

To run a MUSIC_CAM model, you must have the files contained on the MUSIC_CAM disk and a Verilog or IEEE 1076 VHDL simulator. The disk files include the MUSIC_CAM model, workbench and stimulus driver models (VHDL versions only), and some sample instruction sequence files.

The release notes supplied with the model list the files contained in that version. All files are part of the MUSIC_CAM model unless otherwise specified.

## 2.0 INSTALLATION

To install the MUSIC_CAM model, copy the contents of the MUSIC_CAM disk to the area where the simulation will be compiled. Then follow the compilation order outlined in the file *compile.fil* to create the model executables. In general, the model can be run at one of two simulation levels.

The first level is the mcam (MUSIC_CAM entity) level. This is the MUSIC_CAM model itself. When this level is used, it is assumed that the user will define a model or circuit to drive the MUSIC_CAM model.

The second level (supplied only with the VHDL versions) is the mcambnch (MUSIC_CAM_BENCH entity) level. This level is intended to test the operation of the MUSIC_CAM model as well as the development of instruction sequences for driving LANCAM devices. Two MUSIC_CAM_BENCH versions are supplied, one with a single instantiation of the MUSIC_CAM model, and the other with two instantiations connected in a daisy chain. In this environment the mcamtest (MUSIC_CAM_TEST entity) model reads instructions and data out of a file named *opcodes.fil* and creates the appropriate control signals for the connected MUSIC_CAM model(s). The mnemonics that are defined for *opcodes.fil* are contained in Appendix A of this document.

## 3.0 USING THE CAM MODEL

To use the MUSIC_CAM models in conjunction with other models or circuitry, see your VHDL or Verilog simulator manuals. The models provided by MUSIC are behavioral level models and should be usable in any Verilog or IEEE 1076 VHDL simulator.

The user will typically create a symbol description of the MUSIC model that can be wired together with other circuitry. A number of options are set by the entity statement in the *mcam.vhd file.* These option selections may be changed by using generics when the MUSIC_CAM model is instantiated in the next level of the hierarchy.

MUSIC Semiconductors, the MUSIC logo, LANCAM, and the phrase “MUSIC Semiconductors” are registered trademarks of MUSIC Semiconductors. MUSIC_CAM and MUSIC are trademarks of MUSIC Semiconductors.

Some VHDL simulation environments require a configuration statement. For these cases, a configuration statement that is commented out has been provided in the *mcam file*. The *mcambnch file* contains an active configuration statement (i.e., not commented out).

## Device Cycle Time

The device cycle time of the device being simulated is set by the "DEVICE" integer generic in the *mcam.vhd* entity statement. Comments there will guide the user to the value required to set the "DEVICE" generic to the desired CAM cycle time. Each version of the MUSIC_CAM model will contain the timing information for all available cycle times available at the model release date.

## Input Timing Checks

The "TIME_CHECKS" generic is a boolean value that controls the checking of input timing violations. Setting it TRUE will check input signals for set-up and hold timing and report violations as a WARNING.

## Output Timing

Minimum, typical, or maximum output timing may be selected by setting the "DELAY_SETTING" generic to an integer value of 0 (minimum), 1 (typical), or 2 (maximum). The defined constants "min," "typ," and "max" may also be used within the *mcam.vhd file.*

## Match Flag Output Waveforms

The LANCAM match output flags have a period of time where their output may not be valid. The MUSIC_CAM model outputs an 'X' to indicate these time periods when the "NO_XOUT" boolean generic is set to FALSE. The actual match flag waveform is output when "NO_XOUT" is TRUE.

## Signals

Some signal names in the MUSIC_CAM model are different from those used on the actual devices. These differences are necessary because of the '/' character used in the device signal names. In general, the model signal names are the same with the '/' removed and a 'B' appended to indicate "bar" or "active LOW." Table 1 shows this mapping for the MUSIC MU9C1480A LANCAM. Other devices follow this same pattern.

| Model Signal Name | Device Signal Name |
|---|---|
| EB | /E |
| WB | /W |
| CMP | /CM |
| ECB | /EC |
| MIB | /MI |
| FIB | /FI |
| DQ[15:0] | DQ[15:0] |
| MFB | /MF |
| FFB | /FF |

**Table 1: MU9C1480A Signal Names Mapping**

## Registers

The MUSIC_CAM model intentionally does not display the register values since this would imply seeing the insides of the device. However, this information can be obtained through read instructions or by looking inside the CAM model. To assist the user, the paths to the register values are given in Table 2 for the mcam level and Table 3 for the mcambnch level. Registers MR1, MR2, and CR are Mask Registers 1 and 2 and the Comparand register, respectively.

## Debug Switches

To assist the user in understanding the operation of the model, additional debug switches have been incorporated. These switches allow the user to enable the recording of debug statements in a file. For example, the user may choose to see what has been placed into the CAM. This can be performed by setting the DBGMEM flag to TRUE in the *mcampack.vhd file.* As a result of the TRUE value, the next simulation using the VHDL code will result in the generation of a file, *mem.fil*, which will contain information relating to any accesses to the CAM portion of the device. In a similar manner, the generation of files relating to internal registers, Comparand and Mask registers, and output flags can be selected individually or all at once. The file output formats and the flags to set are documented in *mcampack.vhd.*

```
\N4\REG.DEVICE_SELECT_REGV
\N4\REG.PAGE_ADDRESS_REGV
\N4\REG.STATUS_REGV
\N4\REG.NEXT_FREE_REGV
\N4\REG.SEGMENT_REGV
\N4\REG.CONTROL_REGV
\N4\REG.ADDRESS_REGV
\N9\DPATH.CR
\N9\DPATH.MR1
\N9\DPATH.MR2
```

**Table 2: mcam Simulation Level Paths to Registers**

| |
|---|
| \MCAM\N4\REG.DEVICE_SELECT_REGV |
| \MCAM\N4\REG.PAGE_ADDRESS_REGV |
| \MCAM\N4\REG.STATUS_REGV |
| \MCAM\N4\REG.NEXT_FREE_REGV |
| \MCAM\N4\REG.SEGMENT_REGV |
| \MCAM\N4\REG.CONTROL_REGV |
| \MCAM\N4\REG.ADDRESS_REGV |
| \MCAM\N9\DPATH.CR |
| \MCAM\N9\DPATH.MR1 |
| \MCAM\N9\DPATH.MR2 |

**Table 3: mcambnch Simulation Level Paths to Registers**

## 4.0 USING THE WORKBENCH

The VHDL MUSIC_CAM packages include a workbench for code development. The workbench takes an input file named *opcodes.fil* containing instruction mnemonics from Appendix A and generates the proper signal values and timing to initiate those instructions in the CAM. As each instruction is processed, the line number, the instruction, and any value read from the CAM will be displayed. Your simulator will allow you to observe the values of the control signals and the data bus simultaneously. To see how the workbench functions, copy one of the demo files to *opcodes.fil* and start a MUSIC_CAM_BENCH level simulation.

The workbench has the option of generating EB cycle timing based on the specifications for 70, 90, or 120 ns cycle time devices. In addition, it can generate 100 ns cycles (75 ns EB LOW, 25 ns EB HIGH) that may be used with models set to simulate device cycle times of 70 or 90 ns, allowing CAM cycles to be generated on nicely spaced boundaries for easy interpretation of simulation results. The desired EB cycle time may be set in the generic statement contained in the *mcamtbnchvhd* file entity statement, otherwise the default cycle time will be use as specified in the *mcamtest.vhd* file.

The workbench is capable of supplying variable EB cycle lengths to the device model based on the entries in *opcodes.fil*. The details for doing this are discussed in Section 6. Note that specifying 100 ns cycles disables this feature.

Command read and data read cycles report the value output onto the DQ bus output by the LANCAM model. This value is compared to the value in the data field of all read cycles, and a warning is generated and a 50 ns pulse is output on the /READERR signal if there is a mismatch. The /READERR pulse may be used by most simulators as a breakpoint to temporarily stop the simulation execution. See Commands and Data in Section 6 for additional information on data formats and special values.

## 5.0 USING THE SAMPLE SIMULATIONS

Demo files that may be simulated at the mcambnch level are provided on the disk to assist the user in understanding the operation of the model. To make use of one of the sample stimulus files, the user must copy one to *opcodes.fil*, then start the simulation. The workbench code will read the instruction mnemonics from the file and transmit the correct signals to the MUSIC_CAM model.

## 6.0 STIMULUS FILE PREPARATION

To simplify running longer routines, stimulus files can be prepared by using any editor. The file must be named *opcodes.fil* when the simulation is run. The format of the stimulus file is:

Cycle_type[Length[EC_flag]] Cmd/Data [Cmd_Val] [Flags];

Each field must be separated by one or more white space characters. White space characters are spaces, commas (‘,’), and parentheses (‘(’ and ‘)’). Below are a few examples of typical instructions.

```
cw  SPD_AR_V  MI_HI FI_LO EC_HI;
cws2(TCOW_SC,0x18C0);
drlec(0x1EF35ab7);
```

Each line in *opcodes.fil* must be terminated with a semicolon (‘;’), including REPEATs and comments, and there must be no blank lines at the end of the file. Demo instruction files have been included on the disk to illustrate the proper file format. The data files can be stored as any name, but they must be renamed to *opcodes.fil* to be run by the model.

The fields, white space characters, and comment indicators have been selected to allow the stimulus file to be prepared in the form of a C or C++ source code file, which allows instruction sequences to be transported between the model and development hardware. In the workbench environment, C++ comments ("//") and "#DEFINE" and "#INCLUDE" statements are ignored by the parser.

The CAM cycle types may be specified in a way that invokes a C function call. In a typical situation, the functions are included in a header file that is specific to the development hardware configuration. A header file is available for the CAMLAB Development Kit, and a similar header file is available for WidePort LANCAMs. In addition to the function calls for the various cycle types, these header files contain the #DEFINE statements that translate the instruction mnemonics in Appendix A to the op-code values required by the device. The functions in these files may be modified by the user to meet specific hardware configurations. For more information, please contact MUSIC Semiconductors applications support.

## Cycle Types

The Cycle_type sets the levels of the CMB and WB input pins. One of the two-character abbreviations in Table 4 must be used to start every instruction line. The characters 'c', 'd', 'r', and 'w' in the Cycle_type field are the only characters that are not case sensitive, but they must be both upper case or both lower case.

## Cycle Lengths

The third character on the line is optional, but if it is 's', 'm', or 'l', the cycle length will be set to short, medium, or long, respectively. These characters must be lower case. If it is any other character or white space, the cycle length is set to long.

| Cycle_type | CMB | WB | 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 |

**Table 4: Cycle_type Abbreviations**

## ECB Control

The fourth character on the line is optional, but if it is 'e', the ECB input will be set LOW for that CAM cycle. This character must be lower case. Once an 'e' is found in the fourth character position, the EC_LO and EC_HI input control flags are disabled, and any other character or whitespace at that position will cause ECB to be set HIGH. For more information on the ECB pin, please see Input Control Flags.

## Commands and Data

The next field is "Cmd/Data," which contains the command or data to be written to the CAM. The allowable command set is printed in Appendix A. It is important that the data file commands be written exactly as shown in Appendix A for proper parsing of the instructions by the workbench.

For WidePort LANCAMs, TCOW commands and commands including "aaa" require a hexadecimal value for "Cmd_Val" in the command write cycle. This data is provided by separating the command and data with a space or a comma (',').

Data values and data for WidePort LANCAM TCOW and "aaa" cycles are given as hexadecimal values. The parser ignores the first two characters, so that the value may include the usual "0x" before the value. The value must be four hex characters (16-bits) for LANCAMs and WidePort LANCAM TCOW and "aaa" cycles, and eight hex characters (32-bits) for WidePort LANCAM read and data write cycles.

Two special values are available for use during read cycles. The values 0xZZZZ for LANCAM simulations and 0xZZZZZZZZ for WidePort LANCAM simulations may be used to check for a high impedance condition on the DQ bus. This would be expected, for example, when performing a read cycle with the daisy chain locked (ECB LOW at the start of the previous cycle) when no match is present in cascaded LANCAMs. See the device data sheet or LANCAM handbook for more details. The values 0xXXXX or 0xXXXXXXXX may be used to disable the output value checking, including the /READERR signal generation.

### Input Control Flags

Input Control Flags may be provided with each command line and must be in capital letters. The order of the flag inputs is not critical, but they must be of the form signal_HI or signal_LO. The latest release makes the input flags persistent; i.e., the flag state is held over from the previous cycle, so they are not required on every line.

**MI =** The state of the Match Input pin. MI_HI means that either this is a single device with the MIB pin tied HIGH; or, if it is in a string of devices, that no device ahead of it in the daisy-chain has a match, thus selecting this device for detecting Highest-priority matches. If MI were LOW (MI_LO), it would indicate that a prior device has a match indicated, and that device would receive Highest-priority matches. If MI were LOW (MI_LO), it would indicate that a prior device has a match indicated, and that device 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 device with the FIB pin tied LOW; or, if it is in a string of devices, that all the devices ahead of it in the daisy-chain are full, thus selecting this device for Next Free Address operations. If FI were HIGH (FI_HI), that would indicate that a prior device has empty locations, and that device would receive Next Free Address operations. Normally, you would set FI_LO.

**EC =** The state of the ECB input pin. This pin enables the LANCAM's MFB output pin to reflect the detection of an internal match. If ECB is HIGH (EC_HI), then MFB only reflects the state of MIB, and any internal match detected in this device is not shown on the MFB pin. If ECB is LOW (EC_LO), then MFB will go LOW if an internal match is detected. ECB also controls the daisy-chain locking mechanism, whereby only the Highest-priority device can respond after a match is detected. Normally, you only take ECB LOW on the last data segment loaded into the Comparand register, so that the results of the automatic compare can be output by the MFB pin for the next device in a daisy chain and to get the system match indication.

A no-operation instruction (NOP, NOP_1, NOP_2, or NOP_3) allows ECB to be taken HIGH after a "no-match," for example, to unlock the daisy chain without altering any register or memory contents. This is only required for LANCAM versions that do not have the 'A' suffix. See the device data sheet for more information.

The EC_LO and EC_HI flag indicators will be disabled once an 'e' is used in the fourth character position (see ECB Control above), but this function has been retained for compatibility with code sequences for the original workbench.

### Comments

'!', '#', and "//" indicate a comment line. The comment designator must be the first character(s) in the line and the comment must be terminated with a semicolon (';'). Comments may also be added after the semicolon of an instruction line, as the parser ignores everything after the first one in the line.

### Repeat

"REPEAT n" is used to repeat a command n times, as in:

```
REPEAT 4 dw 0x0000 MI_HI FI_LO EC_HI ;
```

which will perform four data writes of all zeroes. The word "REPEAT" must be all upper-case letters. A REPEAT command line must be terminated with a semicolon (;).

## 7.0 ERROR MESSAGES

Error and warning messages are generated either by the device model or by the workbench. Some messages may apply only to certain LANCAM device configurations, and others may be modified for the specific device being modeled.

### Device Model Errors

The user may encounter several messages in the MUSIC_CAM model. These messages are generated using a VHDL assertion statement, so the message appearance may vary with each simulator. However, a summary of the basic messages that could appear and the severity of the message are given below. It is assumed that the simulator will provide the time that the assertion statement is encountered.

*Ignored Data Read.*
*Ignored Data Write.*
*Ignored read cycle.*
*Ignored TCO_PA or SFF instruction.*
*Ignored instruction - DS not equal to PA.*
*Ignored cycle - DS not equal to PA.*
*Severity: "Warning"*

These messages will appear and operation will continue with the next command if, during a command or data access, the device ignores the instruction due to reasons outlined in the device documentation.

*Ignored instruction - no match.*
*Ignored instruction - match in higher priority device.*
*Ignored instruction - match flag disabled.*
*Ignored cycle - no match.*
*Ignored cycle - match in higher priority device.*
*Ignored cycle - match flag disabled.*
*Severity: "Warning"*

The messages above will appear if an instruction requires a match to be present and no match is indicated, the match is in a higher priority device, or the match flag is disabled.

*Ignored instruction - device full or Full Flag not enabled.*
*NFA write ignored - not device with next free location.*
*Severity: "Warning"*

One of these messages will appear if the a MOV_NF (Move to Next Free Address) instruction is encountered when the device is full, the Full flag is disabled in the Control register, or the FFB input is not LOW.

*CMB hold time violation.*
*ECB hold time violation.*
*WB hold time violation.*
*DQ hold time violation.*
*RESETB LOW time violation.*
*Severity: "Error"*

The preceding messages may appear as a result of a timing violation with timing checks enabled. If an error message does occur, most instructions will still be performed. However, there are instances where the instruction will not complete as expected and may result in incorrect or incomplete operations being performed.

*EB low pulse width violation. TCO_CT reset (MEDIUM)*
*EB low pulse width violation. TCO CMND READ (MEDIUM)*
*EB low pulse width violation. WRITE TCO_CT (LONG)*
*EB low pulse width violation. COMMAND WRITE (SHORT)*
*EB low pulse width violation. (SHORT)*
*EB low pulse width violation. LAST REG WRITE (LONG)*
*EB low pulse width violation. MEMORY write (MEDIUM)*
*EB low pulse width violation. (MEDIUM)*
*EB low pulse width violation — VBC. (LONG)*
*EB low pulse width violation. (LONG)*
*Severity: "Error"*

The preceding messages may appear as a result of an EB LOW pulse width violation with timing checks enabled. If possible, the message describes the reason for the violation.

*EB high pulse width violation.*
*Severity: "Error"*

This message will appear when a violation of the HIGH pulse width has occurred with timing checks enabled.

*EB has become unknown or Z.*
*Severity: "Warning"*

This message will result whenever the EB signal goes to a state other than '1' or '0.'

### Workbench Errors

MUSIC_CAM_BENCH will notify the user if it encounters any errors in *opcodes.fil*. These error statements are explained below. The 'n' designates the line number in *opcodes.fil* where the error was detected.

*Could not decode command on line : n*
*Could not decode line : n*

These error messages are issued when the line being processed has some kind of syntax error that prevents it from being parsed correctly.

*Unexpected end of line : n*

This error is issued during the parsing of *opcodes.fil* when no command, data or flag field is found in a command write cycle.

*Could not get repeat value line : n*

This error is displayed when a syntax error prevents a repeat line from being parsed correctly.

*WARNING: put in XXXX because of bad hex value on line: n*

This warning is given anytime the data value for the instruction has an illegal hex value.

*Incorrect coding for op_type line : n*

This error will be issued when the cycle type is other than one of the abbreviated values in Table 4.

*WARNING: DQ Output did not match expected value!!!*

This warning will appear if the data value specified for a read cycle did not match the value actually output by the CAM(s), and the /READERR signal will go to a '1' for 50 ns.

## 8.0 WHO TO CALL FOR HELP

For assistance in running the device models, or information on the MUSIC Semiconductors' CAM products, please call your local MUSIC sales office.

## APPENDIX A: COMMAND LIST WITH OP-CODES

### Set Persistent Source

```
SPS_CR          = 0x0000;
SPS_MR1         = 0x0001;
SPS_MR2         = 0x0002;
SPS_AR          = 0x0004;
SPS_aaa         = 0x0804;
SPS_HM          = 0x0005;
```

### Set Persistent Destination

```
SPD_CR          = 0x0100;
SPD_CRMR1       = 0x0140;
SPD_CRMR2       = 0x0180;
SPD_MR1         = 0x0108;
SPD_MR2         = 0x0110;
SPD_AR_V        = 0x0124;
SPD_ARMR1_V     = 0x0164;
SPD_ARMR2_V     = 0x01A4;
SPD_AR_E        = 0x0125;
SPD_ARMR1_E     = 0x0165;
SPD_ARMR2_E     = 0x01A5;
SPD_AR_S        = 0x0126;
SPD_ARMR1_S     = 0x0166;
SPD_ARMR2_S     = 0x01A6;
SPD_AR_R        = 0x0127;
SPD_ARMR1_R     = 0x0167;
SPD_ARMR2_R     = 0x01A7;
SPD_aaa_V       = 0x0924;
SPD_aaaMR1_V    = 0x0964;
SPD_aaaMR2_V    = 0x09A4;
SPD_aaa_E       = 0x0925;
SPD_aaaMR1_E    = 0x0965;
SPD_aaaMR2_E    = 0x09A5;
SPD_aaa_S       = 0x0926;
SPD_aaaMR1_S    = 0x0966;
SPD_aaaMR2_S    = 0x09A6;
SPD_aaa_R       = 0x0927;
SPD_aaaMR1_R    = 0x0967;
SPD_aaaMR2_R    = 0x09A7;
SPD_HM_V        = 0x012C;
SPD_HMMR1_V     = 0x016C;
SPD_HMMR2_V     = 0x01AC;
SPD_HM_E        = 0x012D;
SPD_HMMR1_E     = 0x016D;
SPD_HMMR2_E     = 0x01AD;
SPD_HM_S        = 0x012E;
SPD_HMMR1_S     = 0x016E;
SPD_HMMR2_S     = 0x01AE;
SPD_HM_R        = 0x012F;
SPD_HMMR1_R     = 0x016F;
SPD_HMMR2_R     = 0x01AF;
SPD_NF_V        = 0x0134;
SPD_NFMR1_V     = 0x0174;
SPD_NFMR2_V     = 0x01B4;
SPD_NF_E        = 0x01B4;
SPD_NFMR1_E     = 0x01B4;
```

### Move

```
MOV_CR_MR1      = 0x0301;
MOV_CR_MR2      = 0x0302;
MOV_CR_AR       = 0x0304;
MOV_CR_ARMR1    = 0x0344;
MOV_CR_ARMR2    = 0x0384;
MOV_CR_aaa      = 0x0B04;
MOV_CR_aaaMR1   = 0x0B44;
MOV_CR_aaaMR2   = 0x0B84;
MOV_CR_HM       = 0x0305;
MOV_CR_HMMR1    = 0x0345;
MOV_CR_HMMR2    = 0x0385;
MOV_MR1_CR      = 0x0308;
MOV_MR1_MR2     = 0x030A;
MOV_MR1_AR      = 0x030C;
MOV_MR1_aaa     = 0x0B0C;
MOV_MR1_HM      = 0x030D;
MOV_MR2_CR      = 0x0310;
MOV_MR2_MR1     = 0x0311;
MOV_MR2_AR      = 0x0314;
MOV_MR2_aaa     = 0x0B14;
MOV_MR2_HM      = 0x0315;
MOV_AR_CR       = 0x0320;
MOV_AR_CRMR1    = 0x0360;
MOV_AR_CRMR2    = 0x03A0;
MOV_AR_MR1      = 0x0321;
MOV_AR_MR2      = 0x0322;
MOV_AR_CR_V     = 0x0324;
MOV_AR_CRMR1_V  = 0x0364;
MOV_AR_CRMR2_V  = 0x03A4;
MOV_AR_MR1_V    = 0x0325;
MOV_AR_MR2_V    = 0x0326;
MOV_aaa_CR      = 0x0B20;
MOV_aaa_CRMR1   = 0x0B60;
MOV_aaa_CRMR2   = 0x0BA0;
MOV_aaa_MR1     = 0x0B21;
MOV_aaa_MR2     = 0x0B22;
MOV_aaa_CR_V    = 0x0B24;
MOV_aaa_CRMR1_V = 0x0B64;
MOV_aaa_CRMR2_V = 0x0BA4;
MOV_aaa_MR1_V   = 0x0B25;
MOV_aaa_MR2_V   = 0x0B26;
MOV_HM_CR       = 0x0328;
MOV_HM_CRMR1    = 0x0368;
MOV_HM_CRMR2    = 0x03A8;
MOV_HM_MR1      = 0x0329;
MOV_HM_MR2      = 0x032A;
MOV_HM_CR_V     = 0x032C;
MOV_HM_CRMR1_V  = 0x036C;
MOV_HM_CRMR2_V  = 0x03AC;
MOV_HM_MR1_V    = 0x032D;
MOV_HM_MR2_V    = 0x032E;
MOV_NF_CR       = 0x0330;
MOV_NF_CRMR1    = 0x0370;
MOV_NF_CRMR2    = 0x03B0;
```

## APPENDIX A: COMMAND LIST WITH OP-CODES (CONTINUED)

### Set Persistent Source

```
SPD_NFMR2_E     = 0x01B5;
SPD_NF_S        = 0x0136;
SPD_NFMR1_S     = 0x0176;
SPD_NFMR2_S     = 0x01B6;
SPD_NF_R        = 0x0137;
SPD_NFMR1_R     = 0x0177;
SPD_NFMR2_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;
```

### Validity Bit Control

```
VBC_AR_V        = 0x0424;
VBC_AR_E        = 0x0425;
VBC_AR_S        = 0x0426;
VBC_AR_R        = 0x0424;
VBC_aaa_V       = 0x0C24;
VBC_aaa_E       = 0x0C25;
VBC_aaa_S       = 0x0C26;
VBC_aaaH_R      = 0x0C27;
VBC_HM_V        = 0x042C;
VBC_HM_E        = 0x042D;
VBC_HM_S        = 0x042E;
VBC_HM_R        = 0x042F;
VBC_ALM_V       = 0x043C;
VBC_ALM_E       = 0x043D;
VBC_ALM_S       = 0x043E;
VBC_ALM_R       = 0x043F;
```

### Move

```
MOV_NF_MR1      = 0x0331;
MOV_NF_MR2      = 0x0332;
MOV_NF_CR_V     = 0x0334;
MOV_NF_CRMR1_V  = 0x0374;
MOV_NF_CRMR2_V  = 0x03B4;
MOV_NF_MR1_V    = 0x0335;
MOV_NF_MR2_V    = 0x0336;
```

### Compare

```
CMP_V           = 0x0504;
CMP_E           = 0x0505;
CMP_S           = 0x0506;
CMP_R           = 0x0507;
```

### Special Instructions (LANCAM “A” family only)

```
SFT_CR_R        = 0x0600;
SFT_CR_L        = 0x0601;
SFT_MR1_R       = 0x0610;
SFT_MR2_L       = 0x0611;
SFR             = 0x0618;
SBR             = 0x0619;
RSC             = 0x061A;
```

### Set Full Flag

```
SFF             = 0x0700;
```

### No Operation

```
NOP
NOP_1           = 0x0300;
NOP_2           = 0x0300;
NOP_3           = 0x0309;
                = 0x0312;
```

**APPENDIX B: Sample Stimulus File (Excerpt From LANCAM Bridge Demo File)**

```
#include  <stdio.h>;
#define COMPARE;
#define NO_VHDL;
#define DEBUG;
#include “1480.h”;
!void main ();
!{;
// **************;
// Initialize CAM;
// **************;
// Clear Power-up anomalies;
crl (zzzzzz);
// Set Device Select for Broadcast;
cws (TCO_DS);
cws (0xFFFF);
// Reset All CAMs;
cws (TCO_CT);
// Initialize Page Address Registers;
cws (TCO_PA);
cws (0X0000);
cws (SFF);
cws (TCO_PA);
cws (0X0001);
// Reset All Full Flags;
cws (TCO_CT);
cwl (0X0000);
cws (TCO_DS);
cws (0x0000);    /* select device zero  */
crm (0x07FF);    /* read status */
//;
// **************;
// Configure CAM;
// **************;
cws (TCO_SC);
cws (0x1800);    /* 2 dest segs, 1 src segment */
cws (SPD_MR1);    /* time stamp mask */
dws (0x7FFF);
dws (0xFFFF);
dws (0xFFFF);
dws (0xFFFF);
cws (SPD_CR);
cws (SPS_HM);
cws (TCO_CT);
cwl (0x8040);    /* 48 CAM, 16 RAM */
//;
// First packet:  to B from A;
// ********************************;
// DA filter routine with no match;
// ********************************;
dws (0x000);
dws (0xBBBB);
dws (0xBBBB);
dwlec (0xBBBB);
drlec (ZZZZZZ);
cws (NOP);    /* clear /EC for next command */
// ********************************;
// SA filter routine with no match;
// ********************************;
dws (0x000A);
dws (0xAAAA);
dws (0xAAAA);
drlec (0XAAAA);
drlec (ZZZZZZ);
cws (NOP);
cwl (MOV_NF_CR_V);
//;
// Second packet:  back to A from B;
// ********************************;
// DA filter routine with match;
// ********************************;;
dws (0x0000);
dws (0xAAAA);
dws (0xAAAA);
dwlec (0XAAAA);
drlec (0x000A);
cwl (MOV_HM_CRMR1);    /* update time stamp */
// ********************************;
// SA filter routine with no match
// ********************************;
dws (0x000B);
dws (0xBBBB);
dws (0xBBBB);
dwlec (0xBBBB);
drlec (ZZZZZZ);
cws (NOP);
cwl (MOV_NF_CR_V);
//;
```

NOTES

**NOTES**

**MUSIC Semiconductors Agent or Distributor:**

MUSIC Semiconductors reserves the right to make changes to its products and specifications at any time in order to improve on performance, manufacturability, or reliability. Information furnished by MUSIC is believed to be accurate, but no responsibility is assumed by MUSIC Semiconductors for the use of said information, nor for any infringement of patents or of other third party rights which may result from said use. No license is granted by implication or otherwise under any patent or patent rights of any MUSIC company.
©Copyright 1998, MUSIC Semiconductors

MUSIC SEMICONDUCTORS

**http://www.music-ic.com**
**email: info@music-ic.com**

**USA Headquarters**
MUSIC Semiconductors
254 B Mountain Avenue
Hackettstown, New Jersey 07840
USA
Tel: 908/979-1010
Fax: 908/979-1035
USA Only: 800/933-1550 Tech. Support
888/226-6874 Product Info.

**Asian Headquarters**
MUSIC Semiconductors
Special Export Processing Zone 1
Carmelray Industrial Park
Canlubang, Calamba, Laguna
Philippines
Tel: +63 49 549 1480
Fax: +63 49 549 1023
Sales Tel/Fax: +632 723 62 15

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