C-Cube Microsystems reserves the right to change any products described herein at any time and without notice. C-Cube Microsystems assumes no responsibility or liability arising from the use of the products described herein, except as expressly agreed to in writing by C-Cube Microsystems. The use and purchase of this product does not convey a licence under any patent rights, copyrights, trademark rights, or any other intellectual property rights of C-Cube Microsystems.

**Trademark Acknowledgment:**

C-Cube and CL450 are trademarks of C-Cube Microsystems.
The corporate logo is a registered trademark of C-Cube Microsystems.
Motorola is a registered trademark of Motorola, Inc.
Intel is a registered trademark of Intel Corporation.
© C-Cube Microsystems 1994
All rights reserved

C-Cube Microsystems
1778 McCarthy Boulevard
Milpitas, CA 95035
Telephone (408) 944-6300
Fax (408) 944-6314

**Customer Comments and Feedback:**

If you have any comments about this document, send them to the C-Cube Technical Publications Department at the address listed above, or send e-mail to:

techpubs@c-cube.com

C-Cube Part # 90-1450-101
This manual is the primary source of technical information for the operation and programming of the C-Cube CL450 MPEG Video Decoder.

This manual is intended for:

- System designers and managers who are evaluating the CL450 for possible use in a system.
- Design, software and system engineers developing a video decoding system using the CL450 for whom a comprehensive programming background as well as a detailed understanding of MPEG compression is assumed.


As an aid to developing systems and applications based on the CL450, readers may also wish to obtain C-Cube’s *CL450 MPEG Video Decoder Sample Kit*, which includes a sample development board, microcode and user’s guide.

This manual is divided into three main sections:

- Section I, General Information, including an introduction to the CL450 and the MPEG standard that it implements.
- Section II, Hardware Interface, including signal descriptions, operational information for the main interfaces of the CL450, registers, and detailed electrical and mechanical specifications.
Section III, Software Interface, including an overview of the fundamental mechanisms used to communicate between the host system and the CL450’s microapplication, and an alphabetical listing of all macro commands and interrupts available to the programmer. Note: the microapplication information in this manual applies to CL450 versions 2.00 through 2.FF only.

Please note the following notation examples and conventions used in this manual:

<table>
<thead>
<tr>
<th>Notation Examples</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ox1c3</td>
<td>“Ox” prefix indicates a hexadecimal number.</td>
</tr>
<tr>
<td>11011₂</td>
<td>“2” subscript indicates a binary number.</td>
</tr>
<tr>
<td>IMEM</td>
<td>Four-letter mnemonics indicate on-chip memories, starting with a letter to indicate function of memory, and ending with “MEM.” For example, IMEM is the CL450’s instruction memory.</td>
</tr>
<tr>
<td>HOST_control</td>
<td>This format indicates a register name. The first part of a register name is a group specifier, given in all upper-case. The second part (separated from the first by an underscore and given in all lower-case) is a register specifier, indicating the function performed by the register within the group. In the examples shown, HOST_control and CMEM_control are separate registers, even though both have the same register specifier, “control.”</td>
</tr>
<tr>
<td>CMEM_control</td>
<td></td>
</tr>
<tr>
<td>VIE, Res</td>
<td>Italicized acronyms or abbreviations (initial or all caps) indicate bit field names within registers or data words.</td>
</tr>
<tr>
<td>HMEM[3]</td>
<td>Square bracket notation similar to C language array sub scripting indicates words within memories, and bits within words and registers. HMEM[3], for example, is the 16-bit word at address 3 within HMEM, while CPU_control[0] is bit 0 of the CPU_control register. Similarly, HMEM[0][15] is the most significant bit of the word at HMEM address 0.</td>
</tr>
<tr>
<td>CPU_control[0]</td>
<td>Ranges of bits are indicated by numbers separated by a colon such as the three bits HMEM[3][2:0]. Ranges of words within a memory are indicated by numbers separated by a dash such as the eight words HMEM[0-7].</td>
</tr>
<tr>
<td>HMEM[0][15]</td>
<td></td>
</tr>
<tr>
<td>HMEM[3][2:0]</td>
<td></td>
</tr>
<tr>
<td>RESERVED or Res</td>
<td>Indicates bit fields within registers which are not defined. RESERVED bit fields may return any value when read and must be written with 0 (or 1 if so specified). Writing the incorrect value to a RESERVED CL450 register bit will cause indeterminate behavior. In addition, all CL450 registers which are not explicitly given names are also RESERVED, and accessing these registers may cause indeterminate results on current or future CL450 implementations.</td>
</tr>
<tr>
<td>leftBorder</td>
<td>Bold-face type represents macro command arguments.</td>
</tr>
<tr>
<td>return();</td>
<td>C-style syntax presented in courier typeface represents program pseudocode and equations.</td>
</tr>
<tr>
<td>vbxv_delay</td>
<td>Names presented in courier represent names of items within the MPEG bitstream taken from the MPEG standard.</td>
</tr>
</tbody>
</table>

1. In some cases, this current reference scheme departs from the method used by the first edition of the CL450 MPEG Video Decoder User’s Manual which, for example, referred to CMEM as both the “Coded Data FIFO” and “C-FIFO.”
Section I. General Information

1 Introduction
1.1 General Description 1-1
1.2 The CL450 Product Family 1-1
  1.2.1 CL450 Features 1-2
  1.2.2 Microapplication Features 1-3
1.3 Functional Description 1-4
1.4 Data Organization 1-5
  1.4.1 CL450 Registers 1-5
  1.4.2 CL450 Memories 1-6
1.5 Address Size 1-6
1.6 Programming Overview 1-7
1.7 Typical Applications 1-8

2 MPEG Overview
2.1 MPEG Decoding 2-2
  2.1.1 MPEG Stream Structure 2-2
  2.1.2 General Decoding Process 2-2
  2.1.3 Video Stream Data Hierarchy 2-3
2.2 Inter-picture Coding
  2.2.1 Picture Types
  2.2.2 Video Stream Composition
  2.2.3 Motion Compensation
2.3 Intra-picture (Transform) Coding
2.4 Synchronization
  2.4.1 System Clock References
  2.4.2 Presentation Time Stamps

Section II. Hardware Interface

3 Signal Descriptions
  3.1 Host Interface
    3.1.1 Data Transfer Signals
    3.1.2 DMA Signals
    3.1.3 Interrupt Signals
    3.1.4 Timing, Control, and Status Signals
  3.2 DRAM Interface
  3.3 Video Interface
  3.4 Miscellaneous

4 Host Interface
  4.1 Memory Access
    4.1.1 Register Access
    4.1.2 Local DRAM Access
    4.1.3 CMEM Access
    4.1.4 DMA Operation
    4.1.5 Interrupt Vector Operation
  4.2 Local DRAM or Register Read
  4.3 Local DRAM or Register Write
  4.4 CMEM Write Timing
  4.5 CMEM DMA Write Timing
  4.6 CMEM DMA Write Application Guidelines
  4.7 CMEM Level
  4.8 Interrupt Cycle Timing
    4.8.1 Polled Interrupts
    4.8.2 Vectored Interrupts

5 Local DRAM Interface
  5.1 General Description
    5.1.1 Amount of DRAM
8.5 DRAM Interface Register
8.6 Video Interface Registers
  8.6.1 Direct-access Video Registers
  8.6.2 Indirect-access Video Registers

Section III. Software Interface

9 Microapplication Overview
  9.1 Process Configurations
    9.1.1 Command Process
    9.1.2 Bitstream Transfer Process
    9.1.3 Decode Process
    9.1.4 Display Process
  9.2 Synchronization
  9.3 Interrupts

10 Initialization
  10.1 Registers
    10.1.1 Default Settings
    10.1.2 Loading Sequence
  10.2 Microapplication
    10.2.1 Default Settings
    10.2.2 Loading Sequence
    10.2.3 Halting

11 Macro Commands
  11.1 Writing Macro Commands
  11.2 Command States
    11.2.1 IDLE State
    11.2.2 PLAY-SETUP State
  11.3 Command FIFO
  11.4 Command Latency
  11.5 Macro Command Groups
    11.5.1 Set-type Commands
    11.5.2 Play-type Commands
    11.5.3 Control Commands
  11.6 Macro Command Reference
    AccessSCR()  11-11
    DisplayStill()  11-14
    FlushBitstream()  11-16
    InquireBufferFullness()  11-19
NewPacket() 11-20
11.6.1 length Argument 11-22
11.6.2 timeStamp Arguments 11-22
11.6.3 Loading and Removing NewPacket Commands 11-23
Pause() 11-25
Play() 11-26
Reset() 11-27
Scan() 11-28
SetBlank() 11-29
SetBorder() 11-30
SetColorMode() 11-34
SetInterruptMask() 11-36
SetThreshold() 11-38
SetVideoFormat() 11-40
SetWindow() 11-42
SingleStep() 11-45
SlowMotion() 11-46

12 Interrupts
12.1 Interrupt Types 12-2
12.1.1 Display-time Interrupt 12-3
12.1.2 Decode-time Interrupts 12-6
12.1.3 VSYNC Interrupts 12-6
12.2 Interrupt Handshaking 12-7
12.2.1 Microapplication Behavior 12-8
12.2.2 Host Behavior 12-11
12.3 Interrupt Listing 12-14
END-D 12-15
END-V 12-16
ERR 12-17
GOP 12-19
PIC-D 12-20
PIC-V 12-21
RDY 12-22
SCN 12-24
SEQ-D 12-25
SEQ-V 12-26
UND 12-27
12.4 Interrupt Examples 12-28
12.4.1 Interrupt Example 1: RDY, UND, ERR 12-29
12.4.2 Interrupt Example 2: PIC-V, SCN 12-34
12.5 Servicing Interrupts 12-37
13 Audio/Video Synchronization
  13.1 MPEG Conditions and Constraints 13-1
  13.2 CL450 Synchronization Mechanism 13-2
    13.2.1 CL450 SCR Counter 13-2
    13.2.2 CL450 Presentation Time Stamps 13-2
    13.2.3 CL450 Synchronization 13-3
  13.3 Updating the CL450 SCR Counter 13-4
    13.3.1 SCR Source and Drift 13-4
    13.3.2 Automatic SCR Modifications 13-5
  13.4 Using SCR Masters for Synchronization 13-5
    13.4.1 Synchronizing to the Bitstream 13-5
    13.4.2 Synchronizing to the Audio Decoder 13-6
    13.4.3 Synchronizing from the CL450's SCR 13-6
    13.4.4 Synchronizing from the CL450 and VSYNC 13-7
  13.5Transferring Coded Data to the CL450 13-7
    13.5.1 Unpacketed Data Transfer 13-8
    13.5.2 Packeted Data Transfer 13-8
    13.5.3 Unpacketed/Packeted Data Transfer without PTSs 13-10
    13.5.4 Packeted Data Transfer with one PTS 13-10
    13.5.5 Packeted Data with Periodic PTSs 13-11

14 Host Access of DRAM Variables
  14.1 Types of Variables 14-1
    14.1.1 Sequence Variable Group 14-2
    14.1.2 Picture Variable Group 14-4
  14.2 Access Synchronization Using Semaphores 14-5
    14.2.1 Reading DRAM-Resident Variables 14-7
    14.2.2 Writing DRAM-Resident Variables 14-7
  14.3 Timing Restrictions 14-8

15 CL450 Host-independent Operations
  15.1 Frame Rate Conversion 15-1
  15.2 Error Recovery and Concealment 15-3
    15.2.1 During Picture Decode 15-3
    15.2.2 During Header Decode 15-4
    15.2.3 During Bitstream Underflow 15-4
  15.3 Operating Restrictions 15-5
    15.3.1 Bitstream 15-5
    15.3.2 Output Window and Timing 15-5

Appendix A CL450 DRAM-Variable Allocation
### Appendix B  Microapplication Executable File Format

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 Syntax Conventions</td>
<td>B-1</td>
</tr>
<tr>
<td>B.2 File Structure</td>
<td>B-2</td>
</tr>
<tr>
<td>B.3 File Header Structure</td>
<td>B-2</td>
</tr>
<tr>
<td>B.4 Code Segment Structure</td>
<td>B-5</td>
</tr>
</tbody>
</table>

### Appendix C  CL450 Microapplication Distribution Disk

### Appendix D  CL450 Troubleshooting

### Index
Figures

1-1 Block Diagram of the CL450 1-4
1-2 Typical CL450 System Application 1-8
2-1 General MPEG Decoding System 2-3
2-2 MPEG Data Hierarchy 2-3
2-3 Location of Luminance and Chrominance Values 2-4
2-4 Macroblock Composition 2-5
2-5 Forward Prediction 2-6
2-6 Bidirectional Prediction 2-7
2-7 Typical Display Order of Picture Types 2-8
2-8 Video Stream versus Display Ordering 2-8
2-9 Transform Coding Operations 2-11
2-10 SCR Flow in MPEG System 2-12
3-1 Bus Connection Diagram 3-2
3-2 Data Transfer Signals 3-3
3-3 DMA Signals 3-5
3-4 Interrupt Signals 3-6
3-5 Timing, Control, and Status Signals 3-7
3-6 System Timer Block Diagram 3-9
3-7 DRAM Interface Signals 3-10
3-8 Video Interface Signals 3-12
3-9 Pixel Bus Definition, RGB Format 3-12
### Tables

1-1 Address Size of CL450 Memories
3-1 Meaning of UDS and LDS
3-2 Internal Module Selected by A[20:19]
4-1 CL450 Memory Access Address Bits
4-2 LDS and UDS Decoded Values
4-3 CMEM Level Control Bits
7-1 Absolute Maximum Ratings
7-2 Operating Conditions
7-3 DC Characteristics
7-4 Timing Characteristics - Local DRAM or Register Access
7-5 Timing Characteristics - CMEM Access
7-6 Timing Characteristics - Vectored Interrupt Cycle with Auto Clear
7-7 Timing Characteristics - Local DRAM Bus
7-8 Timing Characteristics - GCLK and RESET
7-9 Timing Characteristics - SCLK Input
7-10 Timing Characteristics - Video Bus Inputs
7-11 Host Bus Interface Pins
7-12 DRAM Bus Interface Pins
7-13 Video Bus Interface Pins
7-14 Power and Miscellaneous Pins
7-15 Plastic Quad Flat Pack Physical Dimensions
7-26
8-1 Direct-access Registers (listed by category) 8-3
8-2 Indirect-access (Video) Registers (listed by category) 8-4
8-3 Direct-access Registers (described) 8-5
8-4 Indirect-access (Video) Registers (described) 8-5
8-5 CMEM Empty Status Bits 8-10
8-6 CFLEVEL Assertion Control 8-10
8-7 Interrupt Mode Values 8-12
8-8 Video Register IDs 8-25
9-1 Macro Command Summary 9-4
9-2 CL450 Interrupt Summary 9-9
10-1 Automatic Register Defaults 10-3
10-2 Macro Commands for Loading Registers 10-5
10-3 Macro Command Defaults 10-6
10-4 MPEG Decoding Defaults, Sequence Layer 10-7
10-5 MPEG Decoding Defaults, GOP Layer 10-7
10-6 Host/CL450 Interface Defaults 10-8
10-7 DRAM-Resident Variable Defaults 10-8
11-1 HMEM Address Allocation 11-2
11-2 Macro Command Summary 11-11
11-3 Filter Argument Encoding 11-18
11-4 SetBorder() Argument Restrictions: leftBorder and topBorder 11-32
11-5 Video Border Sizes 11-32
11-6 Mask Bit Assignments 11-36
11-7 Video Format Summary 11-40
11-8 SetWindow() Argument Restrictions 11-44
11-9 Picture Decode Play Rate while in SLOW State 11-47
12-1 CL450 Interrupt Summary 12-2
14-1 PIC_SEM / SEQ_SEM DRAM Location Encoding 14-5
14-2 Maximum DTACK Pulse Width 14-9
15-1 Picture Display Rates: Play Speed vs. Nominal Speed 15-2
A-1 DRAM Addresses for Host Scratch Storage A-1
A-2 DRAM Addresses for Sequence Variable Group A-2
A-3 DRAM Addresses for INTRA_Q Matrix A-3
A-4 DRAM Addresses for NON_INTRA_Q Matrix A-3
A-5 DRAM Addresses for Picture Variable Group A-3
A-6 DRAM Addresses for Microapplication Version Information A-4
B-1Executable File Syntax Symbols B-2
Chapter 1
Introduction

The C-Cube CL450™ is the world’s first single-chip MPEG video decoder. It is designed to provide full-motion video capability for cost-sensitive consumer electronics products. The CL450 decompresses constrained-parameters MPEG bitstreams (typically SIF resolution) in real time. SIF resolution is 352 x 240 pixels at 30 Hz or 352 x 288 pixels at 25 Hz, with typical compressed data rates of 1.2 to 3 Mbits per second.

The CL450 interpolates decompressed pictures horizontally (typically to 704 pixels per scan line) before outputting them. The CL450 outputs decompressed pictures multiple times to increase the frame rate from 24, 25, or 30 Hz (coded frame rate) to 50 or 60 Hz (display frame rate). The CL450 can change the display position of decompressed pictures relative to its HSYNC and VSYNC inputs.

The CL450 is a fully-integrated MPEG decoding engine whose functionality is an intertwined combination of hardware and microapplication set provided by C-Cube, as explained in the software interface section of this manual, Section III.
1.2.1 CL450 Features

The basic CL450 product includes these features:

- Fully complies with all requirements of the MPEG standard (ISO CD 11172)
- Performs real-time decoding of SIF-resolution bitstreams (352 x 240 pixels at 30 Hz or 352 x 288 pixels at 25 Hz)
- Performs real-time horizontal pixel interpolation and frame duplication to produce output formats of 704 x 240 pixels at 60 Hz or 704 x 288 pixels at 50 Hz
- Provides either RGB or YCbCr video output using an on-chip color-space converter
- Supports NTSC and PAL video output timing formats
- Interfaces to 680x0 processors and DRAM with no external logic
- Requires only 4 Mbits of 80-ns DRAM to decode SIF-resolution MPEG bitstreams
- Provides hardware and microapplication support for audio/video synchronization
- Decodes Huffman variable-length codes at a peak rate of 4 bits per clock (160 Mbits/second at 40 MHz)
- Allows the active display window to be positioned relative to HSYNC and VSYNC inputs with one-pixel and one-HSYNC accuracy, respectively
- Displays all or part of decompressed pictures; displayed section can be selected to one-pixel accuracy (within the decoded picture) and can be changed every coded frame for panning motion video
- Supports host access of local DRAM including byte writes
- Supports both programmed I/O and DMA transfers of compressed bitstreams from the host
- Is fabricated in a CMOS process
- Is supplied in a 160-pin plastic quad flat-pack (PQFP)
1.2.2 Microapplication Features
The CL450 microapplication includes these features:

- 18 macro commands, allowing the host to control interactive playback, CL450 configuration, and audio/video synchronization:
  - AccessSCR()
  - DisplayStill()
  - FlushBitstream()
  - InquireBufferFullness()
  - NewPacket()
  - Pause()
  - Play()
  - Reset()
  - Scan()
  - SetBlank()
  - SetBorder()
  - SetColorMode()
  - SetInterruptMask()
  - SetThreshold()
  - SetVideoFormat()
  - SetWindow()
  - SingleStep()
  - SlowMotion()
- 11 interrupts, providing the host feedback on bitstream transition, display, and decoding processes
- Bit rate up to 5.0 Mbits per second
- Transcoding between NTSC and PAL input and output frame rates
- System memory expansion capability
- Support for 24-Hz film format input
- Display window positioning within decoded picture to single-pixel accuracy
- Flagged and unflagged error concealment
- MPEG double vertical-resolution still pictures
1.3 Functional Description

Figure 1-1 shows a block diagram of the CL450, which has three interfaces:

- **The host interface**: Connects directly to 680x0 processors with no external logic. It can also be easily connected to 80x86 processors.
- **The DRAM interface**: Reads from and writes to the local DRAM with no external logic.
- **The video interface**: Outputs pixel data to a video monitor or other video processing device.
The host computer supplies compressed (coded) data to the CL450 via the host interface. The CL450 buffers up to 16 coded data words in an internal coded data FIFO, called CMEM, from which the data words are read by the DRAM controller and written into the bitstream buffer in the local 4-Mbit DRAM.

The on-chip central processing unit (CPU) interprets commands issued by the host processor, works with the Huffman decoding engine to perform the decompression process, and writes decompressed pixel data into the frame buffer in the local DRAM.

The video display unit reads decompressed pixel data from the frame buffer, sends it through the color-space converter if necessary, and outputs the pixel data on the video bus.

Though the CL450 can be used with many different processors, it has been optimized for use with the Motorola 68070 processor.

### 1.4.1 CL450 Registers

Each of the 34 registers of the CL450 is 16 bits wide and is accessed by word operations which cover the entire 16-bit width. The least significant bit of each address is bit 0, and the most significant bit is bit 15.

Even though CL450 register addresses may be expressed as byte addresses, the CL450 does not support single-byte access to its internal registers; instead, all CL450 register accesses must be made using 16-bit operations (i.e., 16 bits at a time) as shown below.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>Word 0</td>
<td>LSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Word 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Word 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1.4.2 CL450 Memories

All on-chip memory transactions on the CL450 must be 16 bits (as shown in the previous figure), while the local (off-chip) DRAM may be accessed by the host through the CL450 using single-byte access.

When byte-addressable access to local DRAM is performed, the higher-order byte has an even address twice (2x) that of the word (shown below), and the low-order byte has an odd address that is one count higher than twice the word address (2x + 1). The UDS (Upper Data Strobe) and the LDS (Lower Data Strobe) signals distinguish byte operations from word operations as explained in Chapter 4.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 0</td>
<td>Word 0</td>
<td></td>
<td>Byte 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 2</td>
<td>Word 1</td>
<td></td>
<td>Byte 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte FFFFE</td>
<td>Word 7FFFF</td>
<td></td>
<td>Byte FFFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The size of CL450 addresses within a memory (on-chip memories or the off-chip DRAM) are given in units which are the same as the word size of the RAM in question. For example, the off-chip DRAM is configured as 256K or 512K words of 16 bits each, with DRAM addresses therefore being within the range 0 to 0x3ffff or 0x7ffff words and each distinct address containing a 16-bit value.

The word size of the CL450 memories are shown in Table 1-1.

1-5 C-Cube Microsystems
Table 1-1  Address Size of CL450 Memories

<table>
<thead>
<tr>
<th>Memory</th>
<th>Address Unit Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-chip registers</td>
<td>16 bit</td>
</tr>
<tr>
<td>Off-chip (local) DRAM</td>
<td>16 bit</td>
</tr>
<tr>
<td>HMEM</td>
<td>16 bit</td>
</tr>
<tr>
<td>IMEM</td>
<td>32 bit</td>
</tr>
<tr>
<td>TMEM</td>
<td>24 bit</td>
</tr>
<tr>
<td>CMEM</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

Note: Writing host software to interact with the CL450 on byte-addressable processors may necessitate multiplying the given DRAM and register addresses by 2.

The CL450 performs its higher-level functions by executing a microapplication. The microapplication must be loaded into the CL450 by a software driver before the CL450 is fully operational. C-Cube provides C-language source code for a loader, which users may modify and compile on their target system.

Application programs access the CL450 in two ways:

- **Registers**: Some of the CL450’s internal registers are accessed by application programs to set key operating parameters during initialization and operation. Other CL450 registers are used to load command IDs and arguments for the macro commands described below and to determine the status of microapplication execution. Chapter 8, Registers, describes the use and format of the CL450’s registers.

- **Macro commands**: Use of these commands is the primary method for communicating between the host software and the microapplication executing on the CL450’s CPU. Section 1.2.2 on page 1-3 shows the macro commands implemented in the CL450 microapplication set. Section III of this manual, Software Interface, describes the syntax and use of each macro command.
Typical Applications

1.7 Typical Applications

The CL450 MPEG decoder is designed for low-cost consumer applications such as:

- CD-I systems
- Video games
- Interactive multimedia systems
- Point-of-sale/information kiosks
- Interactive TV

Figure 1-2 shows the CL450 in a typical system application.
This chapter presents an overview of the Moving Picture Experts Group (MPEG) standard that is implemented by the CL450. The standard is officially known as ISO/IEC Draft Standard *Coded representation of picture, audio and multimedia/hypermedia information*, CD 11172, December 6, 1991. It is more commonly referred to as the *MPEG standard*.

MPEG addresses the compression and decompression of video and audio signals and the synchronization of audio and video signals during playback of decompressed MPEG data. The MPEG video algorithm can compress video signals to an average of about 1/2 to 1 bit per coded pixel. At a compressed data rate of 1.2 Mbits per second, a coded resolution of 352 x 240 at 30 Hz is often used, and the resulting video quality is comparable to VHS.
This section explains the general structure of an MPEG stream and introduces some basic concepts used in the rest of the chapter.

### 2.1.1 MPEG Stream Structure

In its most general form, an MPEG stream is made up of two layers:

- The system layer contains timing and other information needed to demultiplex the elementary audio and video streams and to synchronize audio and video during playback.
- The compression layers include the elementary audio and video streams.

### 2.1.2 General Decoding Process

Figure 2-1 shows a generalized decoding system using the CL450 as the decoder for the elementary video stream.

The system decoder extracts the timing information from the MPEG system stream and sends it to the other system components. (Section 2.4, Synchronization, has more information about the use of timing information for audio and video synchronization.) The system decoder also demultiplexes the elementary video and audio streams from the system stream and sends each to the appropriate decoder. In many applications, the system decoder function is implemented as a software program on the host computer.

The video decoder decompresses the elementary video stream as specified in Part 2 of the MPEG standard. (See Section 2.2, Inter-picture Coding, and Section 2.3, Intra-picture Coding, for more information about video compression.) The CL450 performs the video decoding function.

The audio decoder decompresses the audio stream as specified in Part 3 of the MPEG standard.
2.1.3 Video Stream Data Hierarchy
The MPEG standard defines a hierarchy of data structures in the video stream as shown schematically in Figure 2-2.
Video Sequence

A sequence begins with a sequence header (and may contain additional sequence headers), includes one or more groups of pictures, and ends with an end-of-sequence code.

Group of Pictures (GOP)

A header followed by a series of one or more pictures; intended to allow random access into the sequence.

Picture

The primary coding unit of a video sequence. A picture consists of three rectangular matrices representing luminance (Y) and two chrominance (Cb and Cr) values. The Y matrix has an even number of rows and columns. The Cb and Cr matrices are each one-half the size of the Y matrix in both directions, horizontal and vertical.

Figure 2-3 shows the relative x-y locations of the luminance and chrominance components. Note that for every four luminance values, there are two associated chrominance values: one Cb value and one Cr value. (The location of the Cb and Cr values is the same, so only one circle is shown in the figure.)

![Figure 2-3 Location of Luminance and Chrominance Values](image)
**Slice**

One or more “contiguous” macroblocks. The order of the macroblocks within a slice is from left to right and top to bottom.

Slices are important in the handling of errors. If the bitstream contains an error, decoders may skip to the start of the next slice. Having more slices in the bitstream allows better error concealment but uses bits that could otherwise be used to improve picture quality.

**Macroblock**

A 16-pixel by 16-line section of luminance components and the corresponding 8-pixel by 8-line sections of the two chrominance components. See Figure 2-3 for the spatial location of luminance and chrominance components. A macroblock contains four Y blocks, one Cb block and one Cr block as shown in Figure 2-4. The numbers correspond to the ordering of the blocks in the data stream, with block 1 first.

![Macroblock Composition](image)

**Block**

A block is an 8-pixel by 8-line set of values of a luminance or a chrominance component. Note that a luminance block corresponds to one-fourth as large a portion of the displayed image as does a chrominance block.
Much of the information in a picture within a video sequence is similar to information in a previous or subsequent picture. The MPEG standard takes advantage of this temporal redundancy by representing some pictures in terms of their differences from other (reference) pictures, or what is known as *inter-picture coding*. This section describes the types of coded pictures and explains the techniques used in this process.

### 2.2.1 Picture Types

The MPEG standard specifically defines three types of pictures used for inter-coding: intra, predicted, and bidirectional.

#### Intra Pictures

Intra pictures, or I-pictures, are coded using only information present in the picture itself. I-pictures provide potential random access points into the compressed video data. I-pictures use only transform coding (as explained in Section 2.3 on page 2-10) and provide moderate compression. I-pictures typically use about two bits per coded pixel.

#### Predicted Pictures

Predicted pictures, or P-pictures, are coded with respect to the nearest previous I- or P-picture. This technique is called *forward prediction* and is illustrated in Figure 2-5.

Like I-pictures, P-pictures serve as a prediction reference for B-pictures and future P-pictures. However, P-pictures use *motion compensation* (see Section 2.2.3) to provide more compression than is possible with I-pictures. Also unlike I-pictures, P-pictures can propagate coding errors because P-pictures are predicted from previous reference (I- or P-) pictures.

![Forward Prediction Diagram](image)
**Bidirectional Pictures**

Bidirectional pictures, or B-pictures, are pictures that use both a past and future picture as a reference. This technique is called *bidirectional prediction* and is illustrated in Figure 2-6. B-pictures provide the most compression and do not propagate errors because they are never used as a reference. Bidirectional prediction also decreases the effect of noise by averaging two pictures.

![Bidirectional Prediction](image)

**Figure 2-6  Bidirectional Prediction**

### 2.2.2 Video Stream Composition

The MPEG standard allows the encoder to choose the frequency and location of I-pictures. This choice is based on the application’s need for random accessibility and the location of scene cuts in the video sequence. In applications where random access is important, I-pictures are typically used two times a second.

The encoder also chooses the number of B-pictures between any pair of reference (I- or P-) pictures. This choice is based on factors such as the amount of memory in the encoder and the characteristics of the material being coded. For a large class of scenes, a workable arrangement is to have two bidirectional pictures separating successive reference pictures. A typical arrangement of I-, P-, and B-pictures is shown in Figure 2-7 in the order in which they are displayed.

![Inter-picture Coding](image)
Inter-picture Coding

The MPEG encoder reorders pictures in the video stream to present the pictures to the decoder in the most efficient sequence. In particular, the reference pictures needed to reconstruct B-pictures are sent before the associated B-pictures. Figure 2-8 demonstrates this ordering for the first section of the example shown above.

![Figure 2-7 Typical Display Order of Picture Types](image)

**Figure 2-7 Typical Display Order of Picture Types**

**Figure 2-8 Video Stream versus Display Ordering**

### 2.2.3 Motion Compensation

*Motion compensation* is a technique for enhancing the compression of P- and B-pictures by eliminating temporal redundancy. Motion compensation typically improves compression by about a factor of three compared to intra-picture coding. Motion compensation is performed at the macroblock level.
When a macroblock is compressed by motion compensation, the compressed file contains this information:

- One or two spatial vectors between the reference macroblock area and the macroblock being coded (motion vectors)
- The content differences between the reference macroblock area and the macroblock being coded (error terms)

However, not all information in a picture can be predicted from a previous picture.

Consider a scene in which a door opens: The visual details of the room behind the door cannot be predicted from a previous frame in which the door was closed. When a case such as this arises—i.e., a macroblock in a P-picture cannot be efficiently represented by motion compensation—the picture is coded in the same way as a macroblock in an I-picture using transform coding techniques (see Section 2.3, Intra-picture Coding).

The difference between B- and P-picture coding is that macroblocks in a P-picture are coded using the previous reference (I- or P-picture) only, while macroblocks in a B-picture are coded using any combination of a previous and/or future reference picture.

Four codings are therefore possible for each macroblock in a B-picture:

- Intra coding: no motion compensation
- Forward prediction: the previous reference picture is used as a reference
- Backward prediction: the next reference picture is used as a reference
- Bidirectional prediction: two reference pictures are used, the previous reference picture and the next reference picture

Backward prediction can be used to predict uncovered areas that do not appear in previous pictures.
The MPEG transform, or intra-picture, coding algorithm includes these steps:

- Discrete cosine transform (DCT)
- Quantization
- Run-length encoding

Both image blocks and prediction-error blocks have high spatial redundancy. To reduce this redundancy, the MPEG algorithm transforms 8 x 8 blocks of pixels or 8 x 8 blocks of error terms from the spatial domain to the frequency domain with the Discrete Cosine Transform (DCT).

Next, the algorithm quantizes the frequency coefficients. Quantization is the process of approximating each frequency coefficient as one of a limited number of allowed values. The encoder chooses a quantization matrix that determines how each frequency coefficient in the 8 x 8 block is quantized. Human perception of quantization error is lower for high spatial frequencies, so high frequencies are typically quantized more coarsely (i.e., with fewer allowed values) than low frequencies.

The combination of DCT and quantization results in many of the frequency coefficients being zero, especially the coefficients for high spatial frequencies. To take maximum advantage of this, the coefficients are organized in a zigzag order to produce long runs of zeros (see Figure 2-9). The coefficients are then converted to a series of run-amplitude pairs, each pair indicating a number of zero coefficients and the amplitude of a non-zero coefficient. These run-amplitude pairs are then coded with a variable-length code, which uses shorter codes for commonly occurring pairs and longer codes for less common pairs.

Some blocks of pixels need to be coded more accurately than others. For example, blocks with smooth intensity gradients need accurate coding to avoid visible block boundaries. To deal with this inequality between blocks, the MPEG algorithm allows the amount of quantization to be modified for each macroblock of pixels. This mechanism can also be used to provide smooth adaptation to a particular bit rate.
The MPEG standard provides a timing mechanism that allows for synchronization of audio and video. The standard includes two parameters in the system layer of a bitstream which are used by the CL450: the system clock reference (SCR) and the presentation time stamp (PTS).

The MPEG-specified “system clock” runs at 90 kHz and generates $7.8 \times 10^9$ clock ticks in a 24-hour day. System clock reference and presentation time stamp values are coded in MPEG bitstreams using 33 bits, which can represent any clock period in a 24-hour period.

### 2.4.1 System Clock References

A system clock reference is a snapshot of the encoder system clock which is placed into the system layer of the bitstream, as shown in Figure 2-10. During decoding, these values are used to update the clock counter(s) in the system decoder before being sent to the audio and video decoders. This is typically done in a system in which MPEG data is delivered at a fixed bit rate, and effectively locks the decoder’s time base with that of the encoder and the bitstream transport mechanism.

The CL450 contains an on-chip counter which it uses as its clock for performing synchronization. The host may update the CL450’s clock with SCR information from the bitstream (in fixed bit-rate systems) or from some other accurate, decoder-wide time base.
2.4.2 Presentation Time Stamps

Presentation time stamps are samples of the encoder system clock that are associated with video or audio presentation units. A presentation unit is a decoded video picture or a decoded audio sequence. The PTS represents the time at which the video picture is to be displayed or the starting playback time for the audio time sequence.

Elementary video or audio decoders use the PTS values associated with each presentation unit to adjust their decoding rate so that each presentation unit is presented (displayed or played) at the correct time as measured by the decoder’s system clock.
This chapter describes the signals that comprise the external physical interface to the CL450. The information presented for each signal includes the signal mnemonic and name, type (input, output, or bidirectional), and description. For information about the functional operation of the CL450, including functional waveforms, see Chapters 4, 5, and 6. For timing information, see Chapter 7.

This chapter is divided into three sections that correspond to the components that interface to the CL450:

- 3.1: Host Interface
- 3.2: DRAM Interface
- 3.3: Video Interface

Figure 3-1 shows a diagram of the CL450 with the host, DRAM and video signals grouped together.
The host interface signals divide logically into these functional groups:

- **Data transfer signals**: These signals comprise the address and data buses and the control signals used for data transfer handshaking.

- **DMA signals**: The CL450 uses these signals to implement the SCC68070 DMA protocol. This is a subset of that used by the 68000 family of DMA controllers.

- **Interrupt signals**: These signals provide a request-acknowledge handshake used by the CL450 to request vectored interrupt service from the host.

- **Timing, control, and status signals**: These signals include the clocks and reset signals, and the CMEM status signal.
3.1.1 Data Transfer Signals

These signals are used to communicate between the CL450 and the host processor. Figure 3-2 shows how the data transfer signals of the CL450 connect to the host processor. The various modes of transferring data are discussed in Sections 4.2 through 4.5.

![Data Transfer Signals Diagram]

*These bits pertain exclusively to the 680XO example: the left-over bits of a 24-bit address bus. Other bits may be chosen for other processors.

**If the host processor is a 680XO, we recommend qualifying the AS input of the CL450 with UDS or LDS so that set-up time can be added on R/W.

**Figure 3-2 Data Transfer Signals**

**UDS — Upper Data Strobe** **Input**

**LDS — Lower Data Strobe** **Input**

The host processor uses UDS and LDS to indicate the byte validity and transfer size of D[15:0] as shown in Table 3-1. The CL450’s local DRAM (accessed when A[20] is 0) is the only CL450 resource which supports byte-wide accesses; at all other times UDS must equal LDS.
Table 3-1  Meaning of UDS and LDS

<table>
<thead>
<tr>
<th>UDS</th>
<th>LDS</th>
<th>Valid</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>High</td>
<td>D[15:0] not valid</td>
<td>none</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>D[15:8] (DRAM only)</td>
<td>byte</td>
</tr>
<tr>
<td>High</td>
<td>Low</td>
<td>D[7:0] (DRAM only)</td>
<td>byte</td>
</tr>
<tr>
<td>Low</td>
<td>Low</td>
<td>D[15:0]</td>
<td>word</td>
</tr>
</tbody>
</table>

\( \overline{AS} \) — Address Strobe  

The host processor asserts \( \overline{AS} \) (active low) to select the CL450 for a non-DMA read or write operation. In a 68070-based system, \( \overline{AS} \) is derived by the combination of the host signal \( \overline{AS} \) and the address bits above A[20].

A[20:1] — Host Address Bus  

A[20:1] are the address lines that the host processor uses to address the CL450’s CMEM, internal registers, and the local DRAM memory. A[20:19] selects which of the CL450 internal modules will be addressed as shown in Table 3-2.

Table 3-2  Internal Module Selected by A[20:19]

<table>
<thead>
<tr>
<th>A20</th>
<th>A19</th>
<th>Internal Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>DRAM bank 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>DRAM bank 1 (optional)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Internal registers</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>CMEM (write only)</td>
</tr>
</tbody>
</table>

A[0] and the address bits above A[20] are not connected to the CL450. Instead of A[0], the host processor uses UDS and LDS to indicate which byte is valid on the data bus. (Note that UDS and LDS must both be asserted for host access to the CL450 when A[20] = 1.) Address bits above A[20] can be used to generate \( \overline{AS} \) to indicate that the CL450 is selected.

D[15:0] — Host Data Bus  

D[15:0] comprises the 16-bit bidirectional host data bus. The host processor uses D[15:0] to write data to the CL450’s CMEM, internal registers, and local DRAM; and to pass data to the CL450 for DMA writes. The CL450 uses D[15:0] to send requested data to the host processor.
R/W — Read/Write

The host processor asserts R/W (high) to initiate a read operation, and deasserts R/W (low) to initiate a write operation.

DTACK — Data Transfer Acknowledge

Open-Drain Output

The CL450 asserts DTACK (active low) when it is ready to receive or output data on D[15:0]. When the CL450 responds to a read request, it holds DTACK deasserted (high) until the requested data is ready. When the CL450 responds to a write request, it asserts DTACK when it has received and latched the write data.

DTACK is an open-drain signal that allows it to be wire-ORed with other components on the host bus. It requires a pull-up resistor of no less than 470 ohms.

3.1.2 DMA Signals

The CL450 operates as a DMA slave only. An external DMA controller performs the DMA transfer of bitstream data to CMEM according to the SCC68070 DMA protocol, which is a subset protocol of the 68000 family of DMA controllers.

Figure 3-3 shows how the CL450 DMA signals connect to an external DMA controller. DMA data transfers are discussed in full detail in Section 4.7.

Note: Be sure to consult the diagram shown in Section 4.6 before connecting the DMAREQ line.
DMAREQ — DMA Request

The CL450 asserts DMAREQ (active low) to request a DMA transfer. The CL450 asserts DMAREQ whenever CMEM has at least one free space, and the DMA enable flag (DE) in the CMEM_dmactrl register has been set correctly. (See the description of the DE bit in the CMEM_dmactrl register in Section 8.3.1, CMEM Registers, for more information about enabling DMA.)

DMAACK — DMA Acknowledge

The external DMA controller asserts DMAACK (active low) in response to a DMA request from the CL450 on DMAREQ.

DTC — DMA Transfer Complete

The external DMA controller asserts DTC (active low) when a valid data word is present on D[15:0]. The CL450 latches the data word on the falling (active) edge of DTC.

DONE — DMA Done

The external DMA controller asserts DONE (active low) to indicate that the DMA sequence is complete. When DONE is asserted and CMEM is not full, the CL450 resets DE in the CMEM_dmactrl register and deasserts DMAREQ.

3.1.3 Interrupt Signals

The CL450 uses these signals to indicate an interrupt to the host. The CL450 can be programmed to implement an optional request-acknowledge handshake for vectored interrupts. Figure 3-4 shows how the CL450 interrupt signals connect to a host or interrupt processor. Section 4.8, Interrupt Cycle Timing, discusses interrupt processing in more detail.

![Diagram showing interrupt signals]

**Figure 3-4** Interrupt Signals
**INT — Interrupt Request**

Open-Drain Output

The CL450 asserts **INT** (active low) to request an interrupt from the host processor. **INT** is an open-drain signal. This allows it to be wire-ORed with other components on the host bus. It requires a pullup resistor of no less than 470 ohms.

**INTACK — Interrupt Acknowledge**

Input

The host processor asserts **INTACK** (active low) in response to **INT** and begins the vectored interrupt sequence. The **INTACK** pin is ignored unless vectored interrupts are enabled (see the **VIE** bit description in the HOST_control register, page 8-12, and the HOST_intvecw register, page 8-13). **INTACK** is also used in conjunction with the TEST pin to 3-state the output pins for diagnostic purposes.

### 3.1.4 Timing, Control, and Status Signals

These signals provide the clock inputs, reset control, and CMEM status. Figure 3-5 shows these signals. The timing characteristics of these signals is described in Section 7.2.

![Timing Generator Diagram](image)
Host Interface

**GCLK — Global Clock**

The CL450 uses GCLK to clock the internal processor and the DRAM controller. For proper operation of the CL450, this signal must have a nominal frequency of 40 MHz. All local DRAM accesses are synchronized to this clock. HCLK and VCLK are limited to half the frequency of GCLK.

*Note: The use of a precision 40.000 MHz crystal for GCLK is strongly recommended. Other statements in this manual regarding the performance and speed of the CL450 hardware and microapplication are based on a 40-MHz GCLK frequency.*

**SCLK — System Clock**

SCLK is an optional external clock input for the CL450 system timer (SCR counter). This is typically a 90-kHz signal, although higher frequencies can be used with internal prescaler values greater than 1.

The system timer (Figure 3-6) is a 33-bit counter that is used by the host processor to synchronize video generated in the CL450 with other outside devices. These devices could include other CL450s, audio decoders, and other display devices within a multimedia system.

The CL450 system timer contains a 9-bit prescaler that can be used to divide the selected source clock prior to incrementing the system timer. The source clock can be either the SCLK signal or (typically) the GCLK signal.

In a typical system application using a 40-MHz GCLK, the divider should be set to divide GCLK by 444. If an external 90-kHz signal is used for the SCLK frequency, the divider can be set to divide by one. When the GCLK signal is used as a clock source, the SCLK pin must be pulled either *HIGH* or *LOW*.

*Note: The divider is hardware-programmable and may therefore be written to a value other than 444 if the microapplication is not running; otherwise, the microapplication fixes the divider at a value of 444.*

The frequency of SCLK must be less than half the frequency of GCLK, and both the SCLK-*LOW* and SCLK-*HIGH* periods must be greater than the GCLK clock period.
Host Interface

Figure 3-6  System Timer Block Diagram

HCLK — Host Clock  
Input  
The clock signal HCLK (host clock) is the host bus interface reference signal. It is asynchronous with respect to both GCLK and VCLK. All host bus transfers to and from the CL450 are internally synchronized to GCLK. The CL450 supports a host processor with an HCLK of up to 20.0 MHz.

CFLEVEL — CMEM Level Status  
Output  
The CL450 asserts CFLEVEL (active high) when CMEM becomes as full or more full than the threshold selected by the IQE to 4QE bits in the CMEM_dmactrl register (see Chapter 8).

RESET — Hardware Reset  
Input  
An external device asserts RESET (active low) to force the CL450 to execute a hardware reset. To be fully recognized, RESET must be asserted for at least 50 GCLK cycles. After a reset, the CL450 registers are in an indeterminate state and a complete re-initialization of the CL450 should be performed.

TEST — Test  
Input  
When TEST (active high) and INTACK (active low) are asserted simultaneously, the CL450 three-states all outputs for performance of system diagnostics. For normal operation, TEST must be held LOW (deasserted).
The CL450’s internal DRAM controller generates the addressing and control signals to control up to one Mbyte of local DRAM. A minimum DRAM size of 512 Kbytes is required for MPEG decoding. This DRAM is typically configured as one or two banks of 256K by 16 bits.

Figure 3-7 shows the signals that comprise the CL450’s DRAM interface. The signal descriptions are presented following the figure. See Section 5.2, Memory Bus Interface, for more information about DRAM memory architecture.

**Figure 3-7**  DRAM Interface Signals

**MA[9:0] — Memory Address Bus**

The CL450 multiplexes the row and column addresses on these signals to address up to one Mbyte of DRAM. See Section 5.2, Memory Bus Interface, for more information about how the address signals are used with different DRAM components and DRAM array sizes.

**MD[15:0] — Memory Data Bus**

These signals comprise the memory data bus by which data is transferred between the CL450 and the local DRAM array. The direction of the data transfer is determined by the state of **WE**.
**RAS[1:0] — Row Address Strobe Outputs**
The CL450 asserts these signals to latch the row address into the local DRAM array. RAS[1] (active low) latches the row address for bank 1, and RAS[0] (active low) latches the row address for bank 0.

**UCAS — Upper Column Address Strobe Output**
**LCAS — Lower Column Address Strobe Output**
The CL450 asserts these signals to latch the column address into the local DRAM array. UCAS (active low) latches the column address for the upper memory data byte, MD[15:8], and LCAS (active low) latches the address for the lower byte, MD[7:0]. For local DRAM accesses performed by the CL450 for the host, UCAS is generated in response to the UDS input, and LCAS is generated in response to the LDS input.

**UCASIN — Upper Data Latch Enable Inputs**
**LCASIN — Lower Data Latch Enable Inputs**
When the CL450 reads data from the local DRAM array, the data on MD[15:0] is latched into the CL450 on the rising edge of the two CASIN signals. UCASIN latches data coming from the high data byte, MD[15:8], and LCASIN latches data coming from the low data byte, MD[7:0]. Typically, these are connected to the CAS pin(s) of the local DRAM array.

**WE — Write Enable Output**
The CL450 asserts WE (active low) to request a write operation (data transfer from CL450 to DRAM). The CL450 deasserts WE to request a read operation (DRAM to CL450).

The CL450’s video interface outputs pixel data to the video display subsystem in RGB or YCbCr format. Figure 3-8 shows the signals in the CL450’s video interface. Operation of the video interface is discussed in Chapter 6.

*Note: CL450 video output is compatible with MPEG, CCIR 601, and CD-I players and therefore uses a digital output range of 16-235 as opposed to 0-255. (See Section 6.1, Digital Video Standards, for more information.)*
Video Interface

**Figure 3-8** Video Interface Signals

**PD[23:0] — Pixel Data Bus Outputs**

The CL450 transmits pixel data to the video display subsystem using these signals. The definition of the signal lines differs for RGB and YCbCr formats as shown in Figure 3-9 and Figure 3-10. The format used is determined by the microapplication and can be changed using the SetColorMode() macro command described in Chapter 11.

When RGB format is used, each 24-bit word on PD[23:0] contains eight bits each of blue, green, and red. The highest bit of each component is most significant; for example, BLUE[7] corresponds to PD[23].

```
<table>
<thead>
<tr>
<th></th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLUE[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GREEN[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RED[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Figure 3-9** Pixel Bus Definition, RGB Format

When YCbCr format is selected, each 24-bit word on PD[23:0] contains eight bits of auxiliary register data, eight bits of luminance (Y), and eight bits of one of the chrominance components (Cb or Cr). The Cb and Cr coefficients alternate on successive VCLK periods.

The AUX[7:0] signals have no predefined purpose and can be used for whatever the designer wishes. For example, they could be used to load the color map in a RAMDAC. The bits are controlled by writing the desired data to the VID_selaux register (see Section 8.6.2).
**HSYNC — Horizontal Synchronization**  
Input  
The CL450 begins counting the left border for a new horizontal line on the second VCLK after the rising (inactive) edge of HSYNC (active low). HSYNC must be synchronous to VCLK as shown in Section 6.3, Video Synchronization.

**VSYNC — Vertical Synchronization**  
Input  
The CL450 begins outputting the top border of a new field on the first HSYNC after the rising edge of VSYNC. VSYNC is asynchronous with respect to VCLK, and is active high. See Section 6.3, Video Synchronization, for more information on the relationship of VSYNC to HSYNC.

**VOE — Video Output Enable**  
Input  
VOE must be asserted (active low) to enable the CL450 to drive the pixel bus, PD[23:0]. When VOE is deasserted, the CL450 holds the pixel bus in a high-impedance state.

**VCLK — Video Clock**  
Input  
The CL450 outputs one pixel on PD[23:0] for each cycle of VCLK. VCLK cannot run faster than one-half the frequency of GCLK, although at frequencies below 15 MHz it does not need to be synchronous with GCLK.

**Reserved — Future Expansion**  
Input  
C-Cube has reserved these pins for possible future use. They should be either pulled HIGH or LOW but should not be allowed to float.
The host interface on the CL450 is designed to interface to a variety of general-purpose microprocessors. While it is optimized for connection to members of the 680X0 microprocessor family, it also connects easily to the 80X86 family and other host processors.

The host processor can directly access any CL450 register or local DRAM location by reading and writing to specific addresses. Compressed video data transfers are performed by writing the data into the CMEM on-chip memory using either processor writes or DMA transfers. The CL450 host bus interface supports both vectored and polled interrupts.

This chapter is organized into the following sections which describe how the host interface is used:

- 4.1: Overview
- 4.2: Local DRAM or Register Read
- 4.3: Local DRAM or Register Write
4.4: CMEM Write Timing
4.5: CMEM DMA Write Timing
4.6: CMEM DMA Write Application Guidelines
4.7: CMEM Level
4.8: Interrupt Cycle Timing

Figure 4-1 shows the pinout diagram of the CL450 host bus interface. The host interface signals include a 20-bit address bus, a 16-bit data bus, and the necessary control signals for performing data transfers and interrupt handling.

Note: Be sure to consult the diagram shown in Section 4.6 before connecting the DMAREQ and CFLEVEL lines.

![Host Interface Block Diagram](4-2.png)
The host processor uses memory reads and writes to address the CL450’s local DRAM, the internal registers, and CMEM. The host processor indicates that a memory access is being performed by putting the address for the desired CL450 location on the address bus and asserting the $\overline{AS}$ (Address Strobe) pin. Then the CL450’s internal address decoder determines which resource to access by decoding address lines $A[20:19]$ and the $R/W$ line as shown in Table 4-1.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Yes</td>
<td>DRAM Bank 0 Access (Read/Write)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>Yes</td>
<td>DRAM Bank 1 Access (Read/Write)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>No</td>
<td>Register Access (Read/Write)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>No</td>
<td>CMEM Access (Write Only)</td>
</tr>
</tbody>
</table>

The $\overline{AS}$ (Address Strobe) signal on the CL450 performs like a chip select. In a typical system application, $\overline{AS}$ is a function of the address lines greater than $A[20]$ and a signal indicating that a valid address has been placed on the address bus. ($\overline{AS}$ is the signal used in the 68070 systems.)

Byte-wide memory accesses are performed by asserting either the $\text{LDS}$ (Lower Data Strobe) or $\text{UDS}$ (Upper Data Strobe) pin when $\overline{AS}$ is asserted as shown in Table 4-2. Word-wide (16-bit) accesses are performed by asserting both $\text{LDS}$ and $\text{UDS}$ simultaneously. Whenever the CL450 is the target of a byte-wide read ($R/W=1$), only the drivers for the byte being read are driven. The data read on the byte not selected is indeterminate. The local DRAM is the only CL450 resource which can be addressed using byte-wide accesses.

<table>
<thead>
<tr>
<th>$\text{LDS}$</th>
<th>$\text{UDS}$</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>H</td>
<td>Data Not Valid</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>DRAM only: $D[15:8]$ Valid ($A[0] = 0$)</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>DRAM only: $D[7:0]$ Valid ($A[0] = 1$)</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>$D[15:0]$ Valid ($A[0] = 0$)</td>
</tr>
</tbody>
</table>
Overview

Byte accesses are provided to make the CL450 DRAM available as system memory for the host when the CL450 is not being used. The \textit{LDS} and \textit{UDS} inputs to the CL450 may be tied together if this feature is not required.

\textit{Note:} In systems based on the 68000, 68010, and 68070 processors, the \textit{LDS} and \textit{UDS} signals will be asserted in roughly the same cycle as \textit{AS}; but when a write is being performed, \textit{LDS} and \textit{UDS} will be delayed one or two cycles from the assertion of \textit{AS} to allow setup time for the data.

4.1.1 Register Access

When address lines \( A[20:19] = 10_2 \), the CL450 addresses its internal registers. The lower seven address lines, \( A[7:1] \), determine which register is being addressed, and the \( R/W \) signal determines whether the operation is a read or a write. (The upper eleven address lines, \( A[18:8] \), are in a “don’t care” state at this time; the only time they matter is during a DRAM read or write, at which time they form part of the local DRAM address.) Internal register operations should always be full-word accesses as indicated by the assertion of both \textit{LDS} (Lower Data Strobe) and \textit{UDS} (Upper Data Strobe). Two \textit{UDS}-synchronous state machines inside the host interface—one using \textit{HCLK}, the other using \textit{GCLK}—control the data transfer as shown in Figure 4-2. Inputs to the two state machines are registered to synchronize them to the local clock.

4.1.2 Local DRAM Access

When address line \( A[20] \) is \text{LOW}, the CL450 addresses the local DRAM array. The DRAM array is divided into two banks typically organized as 256K addresses by 16 bits. Address line \( A[19] \) selects whether Bank 0 or Bank 1 is being addressed, and address lines \( A[18:1] \) select the location to be accessed. The local DRAM array can be accessed using either byte-wide or word-wide reads and writes. The same two \textit{UDS}-synchronous state machines inside the host interface that are used for register read/writes are also used for local DRAM read/writes. The DRAM controller drives \textit{GSEL} (see Figure 4-2) to complete the read or write operation.
Note: Address bits A[18:8] are not shown on this diagram because they control DRAM addresses only.
4.1.3 CMEM Access
When address lines A[20:19] = 112, the CL450 addresses CMEM for writes; A[20:19] must be 112 before AS goes LOW. Writing a word to any address within the address range puts data into CMEM as shown in Figure 4-3.

For CMEM writes, the input data is registered on the falling edge of the data strobes (LDS/UDS), and an internal CMEM write request is generated on the rising edge of the data strobes. The data is written into CMEM on the first rising edge of GCLK following the activation of the write request. The CMEM address counters in CMEM_status are incremented at the end of the write.

Note: CMEM writes are asynchronous with respect to both HCLK and GCLK; the state machines used for CL450 register read/write and direct DRAM read/write are not used in either DMA or CMEM operations. Also, CMEM is a true dual-port memory; therefore, internal read and host write operations can occur in the same GCLK cycle. HCLK has no effect on CMEM writes.

4.1.4 DMA Operation
DMA writes are very similar to CMEM writes except that the DMAACK and DTC pins control the operation as shown in Figure 4-3. For DMA operation to work correctly, AS and R/W must both = 1. (HCLK, D[15:0] and A[20:1] do not affect DMA operation.) The input data is registered on the falling edge of DTC, and an internal CMEM write request is generated on the rising edge of DTC. The data is written into CMEM on the first rising edge of GCLK following the activation of the write request. The CMEM address counters in CMEM_status are incremented at the end of the write.

4.1.5 Interrupt Vector Operation
For interrupt vector operation to occur, HOST_control bit 14 (Vectored Interrupt Enable, VIE) must be 1. For a vectored interrupt operation, host address A[3:1] must be set up before INTACK goes low and hold until INTACK goes high. In addition, INT from the CL450 must be low. (Asserting INTACK while INT is inactive will cause indeterminate behavior.) All of these operations are asynchronous with respect to GCLK and, like the other access modes described in Sections 4.1.3 and 4.1.4, do not affect the GDATA bus shown in Figure 4-3.
Figure 4-3  CMEM Block Diagram
The host processor accesses the internal registers of the CL450 and the CL450 local DRAM array by reading them as if they were memory locations. Because the resource being read might not be immediately available, the CL450 can insert wait states by delaying assertion of the DTACK signal until the read can be completed.

Register and DRAM reads have a minimum delay of two GCLK cycles. Maximum DRAM latency is dependent on the operation of the microapplication.

Figure 4-4 shows the sequence of signals for a local DRAM or internal register read. The circled numbers in the figure refer to the steps below.

**Figure 4-4 Local DRAM or Register Read Timing**

1. The host processor sets the R/W line to the proper state for the read operation (HIGH).
2. The host processor outputs the desired address on address lines A[20:1].
3. After the R/W line and the address lines have settled, the host
processor asserts the control lines AS, LDS, and UDS to indicate to the CL450 that a host bus read access is in progress. The data bus (D[15:0]) should be in a high-impedance state at this time.

4. The CL450 holds DTACK deasserted (high) until it can respond to the read request. Because the CL450 is performing arbitration of its internal data buses during memory or register accesses, DTACK assertion is delayed by at least one HCLK cycle. When the data is available, the CL450 asserts DTACK.

5. The requested data becomes available on the data bus at the same time that DTACK is asserted.

6. Once the host processor has read the data, it deasserts AS, LDS, and UDS. This completes the bus cycle.

7. When the CL450 detects the deassertion of AS, LDS, and UDS, it releases the DTACK signal and the data bus.

Figure 4-5 shows the sequence of signals for a local DRAM or internal register write. The circled numbers in the figure refer to the steps below.

4.3 Local DRAM or Register Write
Local DRAM or Register Write

1. The host processor sets the R/W line to the proper state for a write operation (LOW).
2. The host processor outputs the desired address on address lines A[20:1].
3. After the address lines have settled, the host processor asserts AS. This indicates to the CL450 that a memory transfer is in progress.
4. The host processor then puts the data to be written to the CL450 on the data bus.
5. UDS and LDS are asserted. (The host processor asserts LDS or UDS if a byte-wide write is taking place, or both if it is a word-wide write.)
6. The CL450 generates a DTACK signal after UDS and LDS are asserted.
7. The CL450 latches the data within 2 GCLK cycles after DTACK goes low.
8. The host processor releases AS.
9. The CL450 responds by releasing DTACK. This completes the write cycle.

Note: For register reads, the DTACK output of the CL450 provides a handshaking response to indicate to the host CPU that the CL450 is ready to terminate the current read or write operation.

For local DRAM reads, the local DRAM is selected by host A[20], and host bits A[19:1] are the DRAM word address. However, DTACK will not be asserted as quickly because the DRAM controller must acknowledge host interface request signals.

The state diagrams for DTACK generation logic are shown in Figure 4-6. Note that the states are drawn as event-driven and are not keyed to a particular clock. In the case of both reads and writes, if AS is deasserted before DTACK is asserted, DTACK will not be asserted. On a read operation, DTACK can be delayed due to internal DRAM arbitration.
While the CL450 is decompressing video, the bitstream buffer in the CL450's local DRAM must be supplied with a constant stream of MPEG video data. To speed up the transfer of compressed video data, CMEM (a coded data FIFO) has been included in the input data path. Write operations to CMEM can be performed with no added wait-states. The CL450 also supports no-wait-state Direct Memory Access (DMA) transfers of compressed video data into CMEM. The CL450 automatically transfers the compressed data from CMEM into the local DRAM.
bitstream buffer whenever space is available. Writes to CMEM must always be 16-bits wide. Byte writes are not supported.

The host processor can control the transfer of data to CMEM using one of the following two methods:

- **Polling**: The host CPU monitors either the CFLEVEL signal or the empty status bits in the CMEM_dmactrl register; it writes data when CMEM has space.

- **DMA**: The CL450 asserts the DMA Request signal (DMAREQ) whenever there is space in CMEM. The host DMA controller responds by acknowledging the request and transferring a word of data to the CL450. This method is desirable because the host does not need to be involved in the transfer of each data word or polling CMEM’s fullness, leaving it free to perform other tasks.

Direct CMEM write and DMA CMEM write operations are done by a GCLK-synchronous state machine in the host interface (see Figure 4-3), and the inherent speed of this interface is much faster than the maximum average transfer speed of 5.0 Mbits per second. The data is registered on the falling edge of UDS and LDS, and three GCLK cycles after the rising edge of AS the data is written into CMEM. UDS and LDS must not go LOW again during these 3 GCLK periods, and R/W must be LOW when UDS/LDS go LOW with a set-up time of T9 (Table 7-4).

For back-to-back transactions, AS HIGH from one cycle must precede UDS/LDS LOW in the following cycle by the sum of T129 and T130 (see Table 7-5). For back-to-back CMEM writes, it would be best to leave R/W LOW. (The timing diagrams that follow are for a 68070 in which the normal state of R/W is high.) AS must stay HIGH for at least two GCLK cycles, and AS LOW must set up a short time before UDS and LDS go LOW (T129).

In direct DMA mode, DMAACK has the same function as AS in direct CMEM writes, and DTC has the same function as UDS and LDS in direct CMEM writes. This means that DMAACK LOW time can be very short and that DMAACK HIGH time must be two GCLK periods.
Figure 4-7 shows the timing diagram for CMEM writes, and Figure 4-8 shows the timing diagram for DMA transfers to CMEM. The circled numbers in each figure refer to the steps that follow each figure.

2. The host processor sets the R/W line to the proper state for the write operation (LOW).
3. After the address lines have had time to settle, the host processor asserts AS. This assertion indicates to the CL450 that a memory transfer is in progress.
4. The host processor then puts the data to be written to the CL450 on the data bus.
5. After allowing time for the data bus to become valid, the host processor asserts both LDS and UDS. CMEM writes must always be 16-bits wide.
6. The CL450 asserts the DTACK signal after the assertion of AS (LOW). Because no arbitration is necessary, no wait states are necessary for CMEM writes.
7. When the host processor recognizes the DTACK signal, it releases AS, LDS and UDS.

8. The CL450 responds by releasing DTACK. This action completes the write cycle.

The CL450 also supports DMA transfers of data into CMEM using the DMAACK, DTC and DONE control inputs. DMA transfers minimize the burden that the CL450 puts on the host processor.

The host processor starts a DMA transaction by configuring the host DMA controller (to set up the DMA channel) and writing a 1 to the DMA enable bit (DE) in the CL450 CMEM_dmactrl register. The act of setting the DE bit while CMEM is at or below the threshold in CMEM_dmactrl will enable the CL450’s internal DMA logic. Subsequently, when CMEM fullness is below the threshold, the CL450 asserts the DMAREQ (DMA Request) signal.

When the host detects DMAREQ active, it asserts the DMAACK (DMA Acknowledge) signal and begins a block write to CMEM. Each time that a word is transferred, the DMA controller checks to see if another DMAREQ pulse has been generated. If a DMAREQ pulse has been generated, the DMA controller generates another DMAACK pulse and writes a word of data.

When CMEM reaches threshold before all of the data has been transferred, the CL450 releases the DMAREQ signal until space becomes available again. This continues until the transfer of data is complete.

When the transfer is complete, the host may assert the DONE signal, which stops the transaction and clears the DMA enable bit (DE) in the CMEM_dmactrl register if the DONE pulse occurred when CMEM was not full. The host can abort the transaction at any point by clearing this bit.

Note: The CL450 supports a host processor with an HCLK frequency of up to 20.0 MHz, or one-half the GCLK frequency, whichever is less. DMA writes can provide two bytes every six host processor clock cycles.
Figure 4-8  CMEM DMA Write Timing

Figure 4-8 shows the sequence of signals during a typical DMA transfer. During this transfer, four words of data are transferred into CMEM, and the host terminates the transfer by asserting DONE. The circled numbers in the figure refer to the steps below.

1. The CL450 asserts DMAREQ, indicating that space is available in CMEM. This request is asynchronous, and may occur during another CL450 access cycle. In this example, it occurs during a word read from the CL450.

2. After completing the current operation, the host DMA controller responds by asserting DMAACK and starting a word transfer.
3. The DMA controller drives the address bus and strobes the system $\overline{AS}$ (not the CL450’s $\overline{AS}$ input), causing the host memory subsystem to drive the data onto the bus.

4. When the data is ready, the system memory responds by asserting DTACK. Note that the host system is driving the address bus, R/W, $\overline{AS}$, and DTACK during a DMA cycle, but that the CL450 is not using them. The CL450’s $\overline{AS}$ and R/W input signals must remain $HIGH$ during a DMA transfer. (In single-address mode, the CL450’s $\overline{AS}$ and R/W signals will stay high without special logic due to the reads from system memory.)

5. The host DMA controller asserts Data Transaction Complete (DTC) to indicate to the CL450 that the word is available. The CL450 stores the data on the falling edge of DTC.

6. The host DMA controller completes the cycle by releasing DMAACK. At this point, the first word has been transferred into CMEM.

7. Because the DMAREQ signal is still asserted, a second word of data is transferred into CMEM using timing identical to that of the first (repeat steps 2 through 6).

8. The transfer of the third word into CMEM is identical to that of the second.

9. During the transfer of the last word, the DMA controller in the host system asserts the DONE signal.

10. When the CL450 detects the active DONE signal (CMEM is assumed to not be full at this time), it clears the DMA enable bit ($DE$) in the CMEM_dmactrl register and releases the DMAREQ signal.

11. The DMA controller releases DMAACK and DTC. This completes the DMA transfer.

When designing a DMA controller to be used with the CL450, the following guidelines should be followed:

- Both R/W and $\overline{AS}$ must remain $HIGH$ throughout a DMA transfer. If the 68070 dual-address DMA transfer mode is used, external logic must be provided to ensure that these conditions are met.
The behavior of both the DMAREQ and DONE signals is conditional on the fullness of internal CMEM. In systems in which the DONE signal does not have to be used, the deterministic operation of DMAREQ can be assured by correct operation of the host software. Alternately, the host may program the CL450 to output the CFLEVEL signal so that it can be used to condition the DMAREQ signal before it is supplied to the host.

For the CL450 to produce pulses on the DMAREQ signal, DMA operation must be enabled by the host writing the DMA Enable (DE) bit while CMEM is not full. This condition can be ensured by the host software polling the CMEM_dmactrl register prior to writing the DMA Enable bit in it.

Alternately, if the DMAREQ signal is conditioned before being sent to the host, it can be ensured in hardware that CMEM never becomes full at all. To accomplish this, a circuit similar to the following should be used:

For this circuit to operate correctly, CFLEVEL must be programmed to become active if the IQ bit (in CMEM_dmactrl) becomes 1. This action can be accomplished by writing a value of 10000000X2 to CMEM_dmactrl. Doing so reduces the effective size of CMEM from 16 to 12 words, so that the actual CMEM will never become completely full. Limiting the size of CMEM in this way ensures that writing a 1 to the DMA Enable bit (DE) in CMEM_dmactrl always activates the CL450's internal DMA logic, and that a pulse on the DONE input signal will always cause DE to be cleared.

A DMA transfer allows a word of data to be written into CMEM every six HCLK cycles. Using direct memory writes, a word can be transferred every four HCLK cycles (8.75 Mbytes per second). If extremely high (instantaneous) data transfer rates are required,
external hardware could be designed to take advantage of this. An example would be a state machine that loads the data from an external FIFO directly into the CL450’s CMEM.

- AS has a minimum pulse width of 40ns, and UDS/LDS have a minimum pulse width of 5ns (not tested).
- AS can be de-asserted on the same cycle where UDS/LDS get asserted if DTACK is asserted.
- During direct host writes to CMEM, data is registered on the falling edge of UDS/LDS.
- No timing restriction exists on how fast back-to-back write transactions can occur during CMEM data transfers; however, UDS and LDS should not go LOW until data has been written into CMEM on the third GCLK cycle after AS goes HIGH.
- DTACK is combinatorial during direct CMEM writes and vectored interrupts but synchronous otherwise.

CFLEVEL (CMEM level) is an output signal that the CL450 generates to inform the host processor of the fullness of CMEM. The host programs control bits in the CMEM_dmactrl register to set the point at which the CFLEVEL signal is asserted by the CL450. Table 4-3 shows how these bits are programmed. This signal can either be polled by the host processor or used to generate an external interrupt.

<table>
<thead>
<tr>
<th>Table 4-3</th>
<th>CMEM Level Control Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 4</td>
<td>Bit 3 Bit 2 Bit 1</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0</td>
<td>0 0 1</td>
</tr>
</tbody>
</table>

1. During normal operation, CMEM is not necessarily drained to empty.

Note: The CMEM level must always be polled before data is written to it; otherwise, data may be lost. Typically, the CFLEVEL pin is set to 3/4 empty. The host polls for this condition and sends 12 words before polling again.
The CL450 supports both polled and vectored interrupts using the 680X0 vectored interrupt sequence.

**Note:** 
*TNT* is an open-drain output so that it can be wire-ORed with other interrupt signals in the system. It should be pulled up to $V_{CC}$ with a resistor no smaller that 470 ohms.

### 4.8.1 Polled Interrupts

Polled interrupt operation is straightforward. The CL450 microapplication generates an interrupt by writing the INT bit in the HOST_control register with zero, which asserts Interrupt Request (INT). (The interrupt bit in the HOST_control register directly connects to the INT output pin.) Typically, the host first services the interrupt and then clears it by setting the interrupt bit in the HOST_control register. This action automatically deasserts INT.

**Note:** The interrupt must also be logically cleared at some point by clearing the Interrupt Status location maintained by the microapplication in HMEM (see Section 14.2, Interrupt Status Location).

### 4.8.2 Vectored Interrupts

The vectored interrupt sequence is more complex. The microapplication sets the INT bit in the HOST_control register, asserting INT. The host responds by asserting Interrupt Acknowledge (INTACK) and broadcasting the interrupt priority on address lines A[3:1].

The CL450 compares the priority of this interrupt to the priority previously written into the IPID bits of the HOST_intvecw register by the host. If the priorities match, the CL450 outputs the Interrupt Vector programmed into the IVect bits of the HOST_intvecw register on D[7:0] and signals its validity by asserting the DTACK signal. If the IPID and IVect bit fields are not written, the initial value of IPID is 0 and IVect = 0xf.

The host processor receives the Interrupt Vector and ends the cycle by releasing the INTACK signal. This action automatically clears the INT bit in the HOST_control register if the AIC (Auto Interrupt Clear) bit is 1. If AIC is not 1, the host must clear INT by writing to the HOST_control
register. For more information on interrupts and interrupt servicing, see Section 14.2.2, Interrupt Control Registers.

*Note: The CL450 requires $\overline{AS}$ to be inactive (HIGH) during an interrupt acknowledge cycle. This differs from the 680X0 bus protocol in which $\overline{AS}$ is active (LOW) during an interrupt acknowledge cycle. The 680X0 system designer should therefore use $\overline{INTACK}$ in the $\overline{AS}$ generation combinatorial logic to make sure that $\overline{INTACK}$ and $\overline{AS}$ are mutually exclusive.*

Alternatively, the CL450 can be mapped into an address range that has at least one zero in the upper address bits ($A[23:21]$). By doing this, the decoder for $\overline{AS}$ will not activate during an interrupt acknowledge cycle since the upper address bits must be all ones during this time.

*This problem does not exist when using non-vectored interrupts.*

Figure 4-9 shows the timing for a vectored interrupt cycle with automatic interrupt clearing.
1. The CL450 requests an interrupt by asserting the INT signal.
2. The host system broadcasts the Interrupt Priority ID on address lines A[3:1].
3. The host system asserts the INTACK signal to acknowledge the interrupt.
4. The CL450 determines that the Interrupt Priority ID matches IPID, and drives the DTACK signal to indicate to the host processor that the vector interrupt address is valid.
5. The CL450 drives the vector interrupt address byte (IVect) on D[7:0].
6. The host releases the INTACK line. (The host latches the interrupt address on the rising edge of INTACK.) This action signals the CL450 that the interrupt acknowledge cycle is complete.
7. The CL450 completes the cycle by releasing the interrupt request line (if AIC=1), DTACK, and the address and data buses.
Interrupt Cycle Timing
This chapter describes the local DRAM interface bus and how it is used. It details all of the signals necessary to connect the CL450 to a DRAM array and shows several example applications. The sections in this chapter are:

- 5.1: General Description
- 5.2: Memory Bus Interface
- 5.3: Memory Bus Timing
In addition to the CL450’s internal registers, the host has direct access to the CL450’s external DRAM buffer. Typically, DRAM is initialized by the host with the CL450’s microapplication (as described in Section 10.2.2, Loading Sequence). After initialization, the host may access variables stored in DRAM (see Appendix A). The microapplication does not support any other host access to DRAM.

Note: Access to off-chip DRAM is the only operation in which byte-wide rather than 16-bit access to the CL450 is possible. All other transactions with the CL450 must be 16 bits.

5.1.1 Amount of DRAM
The CL450 can be connected to either four Mbits or eight Mbits of local dynamic RAM. However, only four Mbits of DRAM are needed. If a second four Mbits of DRAM are added, it may be used by the system but it will be ignored by the microapplication.

The DRAM array is used to store:

- Compressed video data waiting to be decoded by the bitstream buffer
- The decoded video frame that is currently being displayed
- Future and past decoded reference frames
- CL450 microapplication instructions
- Bitstream header parameters
- The Command FIFO

5.1.2 Type and Organization of DRAM
The DRAM is organized as one or two banks of memory, each 256K locations deep by 16 bits wide, using 80ns fast-page-mode DRAMS. The CL450 can decode any MPEG constrained-parameter bitstream with a single bank of DRAM. However, designing the system so that it can be expanded to support the full amount of memory could allow the system to accommodate larger bitstream buffers and/or future enhanced microapplications.

Note: DRAMs which are “write per bit” or have a write mask should not be used with the CL450. More specifically, the DRAM used must ignore the state of the WE signal dur-
ing a CAS-before-RAS refresh cycle. (See the next section for an explanation of these signals.) Aside from this restriction, any normal 256K x 16 fast-page-mode DRAM can be used, and either 8/10 or 9/9 RAS/CAS may be used.

Although this chapter primarily addresses the connections between the DRAM controller and external memory, it is also useful in understanding some of the other functions that the controller performs. For example, the DRAM controller acts as an intelligent peripheral to the CL450's internal CPU. It can be programmed in the microapplication to:

- Extract data from CMEM and place it in DRAM
- Move data from DRAM to the MPEG decoding engine
- Move data from the MPEG decoding engine back into the DRAM
- Move a decoded video frame from the DRAM to the video display unit
- Arbitrate requests from the host interface for access to the local DRAM

Once the CL450 CPU has instructed the controller to perform one or more of these functions, no more intervention is required until the process is complete. The CL450 CPU is thus free to perform other important functions.

The memory interface on the CL450 consists of a 10-bit multiplexed address bus, a 16-bit data bus, and the control lines necessary for reading and writing the DRAM: Write Enable (WE), Column and Row Address Strobes (LCAS, UCAS and RAS[1:0]), and the data latch enable signals (LCASIN and UCASIN). The high output drive of the CL450 allows it to directly drive the DRAMs without external buffers or logic. This lowers the system parts count and allows slower DRAMs to be used.

Memory arrays designed to use 256K x 4 DRAMs should only use nine of the ten address lines, MA[8:0]. Memory arrays designed to use 256K x 16 DRAMs should use all ten bits, MA[9:0], because some 256K x 16 DRAMs have asymmetrical row and column addresses (a ten-bit row address and an eight-bit column address). The CL450 supports both of
these memory organizations by duplicating column address bit nine in the tenth location of the row address as shown in Figure 5-1.

<table>
<thead>
<tr>
<th>Row Address</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Column Address</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

*Figure 5-1 DRAM Address Bus Configuration*

### 5.2.1 Memory Interface Connections

Figure 5-2 shows how the memory bus interface is connected when using 256K x 16 DRAMs, and Figure 5-3 show how the interface is connected when using 256K x 4 DRAMs.

The host processor addresses the local DRAM whenever address line A[20] is LOW. A[19] is used to select Bank 0 or Bank 1 through assertion of the RAS[0] or RAS[1] signal, respectively. Host address signals A[18:1] map directly into the multiplexed row and column DRAM addresses. The host bus interface signal R/W maps directly to the DRAM interface WE signal.

The host bus interface supports byte-wide accesses to the local DRAM by using the signals LDS and UDS to select LCAS and UCAS, respectively. LCASIN and UCASIN are used to latch the data coming back from the DRAMs.
Figure 5-2  Local DRAM Implementation with 256K x 16 DRAMs
Figure 5-3  Local DRAM Implementation with 256K x 4 DRAMs
5.2.2 Memory Design Guidelines

When designing DRAM arrays for use with the CL450, the following guidelines should be followed:

- When using 256K x 16 DRAMs, connect all 10 address lines to the DRAMs. This allows DRAMs with either ten row and eight column address lines or nine row and nine column address lines to be used. The 10th address bit on 256K x 16 DRAMs is located on pin 15 of SOJ packages, pin 17 of TSOP packages and pin 25 of ZIP packages. This pin has no effect on DRAMs with nine row and nine column addresses.

- The LCAS and UCAS signals should be routed from the DRAM farthest from the CL450 to the LCASIN and UCASIN inputs. This gives the CASIN signal a delay path similar to the path of the DRAM read data to the CL450.

- Series damping resistors may need to be placed between the CL450 and the DRAMs to reduce overshoot and undershoot on the address lines and control signals. This problem can occur when there is a large array with long traces, such as when eight 256K x 4 DRAMs are used. The resistors should be connected as close to the CL450 as possible. The best value for the resistors depends on the board characteristics. The resistors are typically between 20 and 40 ohms with 33 ohms being most common.

- If the host bus byte-wide write capability is not used, only one of the CAS signals needs to be connected to the DRAMs. Either LCAS or UCAS can be used but must be connected to both the LCASIN and the UCASIN inputs. This allows 256K x 16 DRAMs with a single CAS input to be used.

Note: The elimination of all possible DRAM noise is critical to proper DRAM layout for the CL450. Therefore, it is important that you keep the DRAM very close to the CL450, use terminations, and have a very “clean” ground and V_{CC} plane below the DRAM and those portions of the CL450 that connect to it. If possible, do not pass any other signals or clocks anywhere under the DRAMs or their traces.
The DRAM interface on the CL450 is designed to work with 80ns fast-page-mode DRAMs. All DRAM accesses are synchronized to GCLK. Fast-page-mode DRAMs allow shorter read and write cycle periods within a DRAM row. During a set of fast-page-mode accesses, a row address is strobed in using the RAS line, followed by several column addresses strobed in by CAS. Releasing the RAS and CAS lines simultaneously completes the set of cycles.

Refresh is accomplished by asserting the CAS signal before the RAS signal (opposite of normal operation). This sequence starts a refresh cycle within the DRAM using the DRAM’s internal address counter.

Refresh is performed at regular intervals determined by the value programmed into the DRAM_refcnt register (see page 8-23). Typical DRAM specifications require that all pages be refreshed each 8ms. The CL450 meets this specification by refreshing one page at a time.

5.3.1 DRAM Page-Mode Read Timing

Figure 5-4 shows the timing for a page-mode read. The circled numbers in the figure refer to the steps on the next page.

![Page-Mode Read Timing Diagram]

**Figure 5-4 Page-Mode Read Timing**
1. The CL450 starts a read cycle by three-stating the memory data bus, MD[15:0], and setting the write enable (WE) line HIGH.
2. The CL450 then drives the row address onto the memory address (MA) bus.
3. The CL450 asserts the RAS line (LOW) to latch the row address into the DRAM.
4. The CL450 then outputs the column address of the first word to be read on MA.
5. The CL450 latches the column address into the DRAM by asserting the CAS line(s) (LOW).
6. The DRAMs output the data from the selected address.
7. The CL450 deasserts CAS.
8. The data is loaded into the input latch of the CL450 on the rising edge of CASIN. For a single-location read, the CL450 ends the cycle by deasserting RAS at this time.
9. For a page-mode read cycle, the CL450 outputs the column address of the next word to be read on MA[9:0].
10. The CL450 latches the address into the DRAM by asserting the CAS line.
11. The DRAMs output the data from the selected address.
12. The data is loaded into the input latch of the CL450 on the rising edge of CASIN.
13. The CL450 deasserts RAS and CAS to complete the cycle.

5.3.2 DRAM Page-Mode Write Timing
Figure 5-5 shows the timing for a page-mode write. The circled numbers in the figure refer to the steps that follow.
1. The CL450 starts a write cycle by asserting the write enable (WE) line LOW.
2. The CL450 then drives the row address on the memory address (MA) bus.
3. The CL450 asserts the RAS line (LOW) to latch the row address into the DRAM.
4. The CL450 then outputs the column address of the first word to be written on the MA bus, and outputs the data to be written into the first address on the MD bus.
5. The CL450 asserts the CAS line (LOW) to write the data into the selected location of the DRAM.
6. For a single-location write, the write operation is terminated by driving the RAS and CAS lines inactive (HIGH) at this point. For a page-mode write, the address and data for the next word to be written are output on the MA and MD buses, respectively.
7. The CL450 asserts the CAS line to write the data into the selected location of the DRAM.
8. When the last word is written, the write operation is terminated by driving the RAS and CAS lines HIGH.
5.3.3 DRAM Refresh Timing

Figure 5-6 shows the timing for a memory refresh cycle. The circled numbers in the figure refer to the steps below.

1. The CL450 initiates a DRAM refresh cycle by asserting the CAS line while the RAS line is inactive.
2. The CL450 asserts the RAS line to enable the refresh at the address pointed to by the DRAM’s internal address counter.
3. The CL450 completes the refresh cycle by releasing RAS and CAS.

Note: The CL450 WE signal does not necessarily go inactive during DRAM refreshes, so use DRAMs that do not acknowledge the state of WE during DRAM refresh timing (i.e., when CAS is asserted while the RAS line is inactive).
Memory Bus Timing
This chapter describes the operation of the video display unit. It assumes that the reader is familiar with the way that television pictures are received, synchronized, and displayed.

The purpose of the video display unit is to output the decompressed video data in a form that can be either displayed on a television or video monitor, or mixed with computer-generated graphics to provide a video signal within a graphics window.

To perform these functions, the video display horizontally interpolates decompressed video, optionally converts pixel data from YCbCr to RGB color spaces, synchronizes it to the video clock, and outputs it to a display device.

The sections in this chapter are:

- 6.1: Digital Video Standards
- 6.2: Video Display Unit
- 6.3: Video Synchronization
6.1 Digital Video Standards

There are two primary standards for analog video transmission used in the world today:

- **NTSC (National Television Systems Committee) standard**: Used by North America, Japan, Korea, and Taiwan.
- **PAL (Phase Alternating Line) standard**: Used by Europe, China, and most of Africa.

The NTSC video standard specifies a scanning system of 525 horizontal lines per frame. Each frame consists of two interlaced fields of 262.5 horizontal lines. The field rate is 59.94 Hz.

The start of the frame is synchronized with a vertical sync pulse followed by an interval of 42 lines with no video, called *vertical blanking*. This leaves 480 lines in a frame for picture information (240 in a field).

The horizontal lines are synchronized with a horizontal synchronization pulse followed by an interval with no video, called *horizontal blanking*. Horizontal synchronization pulses occur at a rate of 15,734 Hz.

PAL video operates in much the same way, with these distinctions:

- The field rate is 50 Hz.
- There are 625 lines in a frame and 312.5 lines in a field.
- The horizontal sync pulses occur at a rate of 15,626 Hz.
- There are 576 lines of active video in a PAL frame and 288 lines in a field.

The SIF (source input format) specification reflects these standards by defining two *digital* video formats: 352 pixels x 240 lines x 30 Hz for compatibility with NTSC, and 352 pixels x 288 lines x 25 Hz for compatibility with PAL. Because the SIF frame rates are half those of the standard field rates, the CL450 displays each decoded SIF picture nominally for two field periods.
The actual video information may be represented using two data formats:

- **YCbCr data**: Consists of a Y value for the luminance, or brightness, of a pixel and two color values, Cb and Cr, for the chrominance. The Cb and Cr values represent color difference signals that can be converted into red, green, and blue values.

- **RGB (red, green, blue)**: The data output has a digital value that corresponds to the amplitude of the red, green, and blue signals. This digital value is converted to an analog signal before being displayed.

In accordance with the MPEG standard, CL450 compressed video data is always stored in YCbCr format, with Y values ranging from 16 to 235 as specified by MPEG. If RGB-encoded output video is needed, the YCbCr data is converted by the CL450 using a matrix which preserves the 220 discrete levels encoded by MPEG (as required by the CD-I standard), in which 16-16-16 RGB indicates the color black, while 235-235-235 indicates the color white.

*Note: Your DAC should be set to the 16-235 values. If your DAC expects black at 0-0-0 and white at 255-255-255, then your output colors will not be as brilliant as expected.*

*On the CL450 development board or similar designs using the DAC in which the output of the CL450 is being stored into memory, the following steps may be taken to convert from 16-235 to the full, dynamic range of 0-255:*

1. Subtract 16 to shift the RGB range to 0-219.
2. Multiply by 256/220 (1.1634) to expand the range to 256 levels.

*RGB will now equal (RGB-16) x (1.1634).*

The CL450's video display unit accepts decompressed YCbCr data from the local DRAM and outputs it as either horizontally interpolated YCbCr or RGB pixels. The pixels are clocked out by the VCLK (video clock) signal and are synchronized to external devices using the...
Video Synchronization

**HSYNC** (Horizontal Sync) and **VSYNC** (Vertical Sync) input signals. Figure 6-1 shows the video bus connections.

**Figure 6-1** Typical Video Bus Connections

The CL450 does not generate video synchronization signals, but rather is designed to be synchronized with external horizontal and vertical sync signals. This allows it to be easily integrated with other video components in a multimedia system.

In a typical video display, the video information is surrounded by a colored border. The size and color of the border are programmed using the microapplication’s SetBorder() command described in Chapter 11.

If the CL450 is directly driving the video generation logic, the border can be programmed to emulate the horizontal and vertical blanking intervals by selecting the appropriate border widths and setting the border color to black.
If the CL450 is being used to display video within a computer graphics window, the borders can be used to position the video window horizontally and vertically within the graphics frame. The border color is then programmed to a value that can easily be multiplexed with the computer-generated video color values.

A video field example is shown in Figure 6-2. This example field has a 15-line top border, 240 lines of displayed video, and a 12-line bottom border. Horizontally it has a 13-pixel left border, 320 pixels of displayed video, and a 15-pixel right border.

![Figure 6-2 Example Video Frame](image)

The display of the first of 15 lines of the top border of Figure 6-2 begins on the rising edge of the vertical synchronization pulse (VSYNC) received from the system video controller as shown in Figure 6-3. The circled numbers of Figure 6-3 refer to the steps below.
Video Synchronization

Figure 6-3  Vertical Timing

1. The video field begins when the CL450 receives the first rising edge of HSYNC after the rising edge of the vertical sync (VSYNC) pulse. (VSYNC can occur asynchronously to VCLK.)

2. At the end of the vertical border time, the CL450 starts to output active video lines on the pixel bus: in this example, 240 lines. Video lines start when the rising (inactive) edge of HSYNC is received.

3. The CL450 continues to output video until the last line has been output. At this point, it outputs the border color again until the end of the video field.

4. A rising edge on VSYNC starts a new field.

Figure 6-4 shows the beginning and end of a typical horizontal line. This example line has a border at the left side of the field that is 13 VCLKs wide, 320 active pixels, and a 15-pixel-wide border at the right side of the field. The circled numbers refer to the steps below.
1. A horizontal line begins when the CL450 receives an inactive (rising) edge on the horizontal sync signal (HSYNC).

2. At the end of the left border time, the CL450 starts to output active video on the pixel bus. In our example, this will continue for 320 VCLK pulse widths.

3. The CL450 continues to output video until the last pixel has been transmitted. At that point, it will output the border color again until the end of the horizontal line.

4. An inactive edge on HSYNC causes a new horizontal line to be displayed.

To be reliably recognized, HSYNC falling edges must occur either less than 2 μs or more than 20 μs after the rising edge of VSYNC. If this restriction is not met, the first HSYNC signal after VSYNC might be ignored, causing the visible video window to randomly shift up and down by one line.

HSYNC edges which occur less than 2 μs after the VSYNC edge are considered to have been received before the VSYNC edge, while
HSYNC edges which occur more than 20 $\mu$s after the rising edge of VSYNC are considered as being received after the VSYNC edge.

![Figure 6-5 VSYNC Timing Restriction](image)

The video display unit outputs pixel values on the 24-bit wide pixel bus, PD[23:0]. The pixels can be output as either YCbCr or RGB values. The display mode is selected by the RGB bit in the VID_selmode register as described in Chapter 8.

In YCbCr mode, luminance values (Y) are output on PD[15:8] and chrominance (Cb and Cr) values are output on PD[7:0]. While luminance values are stored for every pixel, chrominance values Cb and Cr are only stored for every alternate pixel as shown in Figure 2-3. The reason for this is that the eye is far less sensitive to changes in color (chrominance) than it is to changes in brightness (luminance). The MPEG standard takes advantage of that fact to increase the data compression factor in this way. Because only a quarter as many Cb and Cr values (each) are stored as Y values, Cb and Cr are output by the CL450 in alternate VCLK periods (as shown in Figure 6-6) and identical CbCr information is output for every pair of scan lines displayed.

*Note: The CL450 may begin output with a Cb or a Cr value. The determination as to whether the CL450 begins output with a decoded or interpolated value is, however, somewhat more complicated. That is, if the xOffset parameter of the SetWindow() command (see page 11-41) is an even number, the CL450 begins output with decoded (real Y) values; however, if the xOffset parameter is odd, the CL450 begins output with an interpolated value.*
In YCbCr mode, PD[23:16] outputs the 8-bit value in the VID_selaux register. Since these pins have no predefined use, they can be used as general-purpose programmable outputs.

Decompressed video data is always stored in YCbCr format in the CL450’s local DRAM. If RGB outputs are required, the CL450 can be programmed to convert the data to RGB. The equation used to perform the conversion is shown in the description of the VID_sela and VID_selb registers in Chapter 8. Figure 6-7 shows the PD[23:0] outputs when the CL450 is configured in RGB mode.

Note: CL450 RGB mode is set to be compatible with CD-I players and therefore conforms to an output range of 16-235 as opposed to a range of 0-255 values (see page 6-3).
The VOE signal is used to enable or disable the pixel bus outputs. When VOE is active (LOW), the PD[23:0] signals are driven. When VOE is HIGH, the outputs are three-stated as shown in Figure 6-8.

![Figure 6-8 Video Output Enable Signal](image)

In a typical video-within-a-window system, PD[23:0] outputs form a three-state bus along with the outputs of the system video controller. This bus is connected to the inputs to the video DACs. The system video controller disables its outputs and enables the CL450 outputs during the time that the CL450 window is being displayed. An example is shown in Figure 6-9.

![Figure 6-9 Multiplexing Video Outputs Using VOE](image)
This chapter describes the CL450’s electrical and mechanical characteristics. The chapter is divided into three sections:

- 7.1: Operating Conditions
- 7.2: AC Timing Characteristics
- 7.3: Package Specifications

*Note: These specifications represent an improvement over previously published specifications.*
This section specifies the electrical characteristics of the CL450.

### Table 7-1 Absolute Maximum Ratings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Voltage (V&lt;sub&gt;DD&lt;/sub&gt;)</td>
<td>-0.5 to 6.5 V</td>
</tr>
<tr>
<td>Input Voltage</td>
<td>-1.0 to (V&lt;sub&gt;DD&lt;/sub&gt; +1.0)</td>
</tr>
<tr>
<td>Output Voltage</td>
<td>-0.5 to (V&lt;sub&gt;DD&lt;/sub&gt; +0.5)</td>
</tr>
<tr>
<td>Storage temperature range</td>
<td>-65°C to 150°C</td>
</tr>
<tr>
<td>Operating temperature range (case)</td>
<td>0°C to 90°C</td>
</tr>
<tr>
<td>Reflow Soldering temperature</td>
<td>240°C for 5 Seconds Maximum</td>
</tr>
</tbody>
</table>

1. Exposure to stresses beyond those listed in this table may result in device unreliability, permanent damage, or both.

### Table 7-2 Operating Conditions

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Commercial Unit</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>V&lt;sub&gt;DD&lt;/sub&gt; Supply Voltage</td>
<td>V</td>
<td>4.75</td>
<td>5.25</td>
<td>V</td>
</tr>
<tr>
<td>T&lt;sub&gt;CASE&lt;/sub&gt; Operating Temperature</td>
<td>°C</td>
<td>0</td>
<td>85</td>
<td>°C</td>
</tr>
</tbody>
</table>

### Table 7-3 DC Characteristics

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Test Conditions</th>
<th>Commercial Unit</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>V&lt;sub&gt;IH&lt;/sub&gt; High-level input voltage</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt; = MAX</td>
<td>V</td>
<td>2.4</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>V&lt;sub&gt;IL&lt;/sub&gt; Low-level input voltage</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt; = MIN</td>
<td>V</td>
<td>0.8</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>V&lt;sub&gt;OH&lt;/sub&gt; High-level output voltage</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt; = MIN, I&lt;sub&gt;OH&lt;/sub&gt; = -2.0 mA</td>
<td>V</td>
<td>2.8</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>V&lt;sub&gt;OL&lt;/sub&gt; Low-level output voltage</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt; = MIN, I&lt;sub&gt;OL&lt;/sub&gt; = 8.0 mA</td>
<td>V</td>
<td>0.5</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>I&lt;sub&gt;H&lt;/sub&gt; High-level input current</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt; = MAX, V&lt;sub&gt;IN&lt;/sub&gt; = V&lt;sub&gt;DD&lt;/sub&gt;</td>
<td>I</td>
<td>10</td>
<td></td>
<td></td>
<td>μA</td>
</tr>
<tr>
<td>I&lt;sub&gt;L&lt;/sub&gt; Low-level input current</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt; = MAX, V&lt;sub&gt;IN&lt;/sub&gt; = 0 V</td>
<td>I</td>
<td>-10</td>
<td>-10</td>
<td>+10</td>
<td>μA</td>
</tr>
<tr>
<td>I&lt;sub&gt;QZ&lt;/sub&gt; Output leakage current</td>
<td>Hi-Z output driven to 0V and 5.25 V</td>
<td>I</td>
<td>-10</td>
<td>-10</td>
<td>+10</td>
<td>μA</td>
</tr>
<tr>
<td>I&lt;sub&gt;DD0&lt;/sub&gt; Supply Current</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt; = MAX, GCLK = 40MHz</td>
<td>I</td>
<td>320</td>
<td>400</td>
<td></td>
<td>mA</td>
</tr>
<tr>
<td>@ T&lt;sub&gt;J&lt;/sub&gt;=0°C</td>
<td>V&lt;sub&gt;IN&lt;/sub&gt; = 0 or V&lt;sub&gt;DD&lt;/sub&gt;, C&lt;sub&gt;L&lt;/sub&gt;=50pF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I&lt;sub&gt;DD85&lt;/sub&gt; Supply Current</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt; = MAX, GCLK = 40MHz</td>
<td>I</td>
<td>300</td>
<td>350</td>
<td></td>
<td>mA</td>
</tr>
<tr>
<td>@ T&lt;sub&gt;J&lt;/sub&gt;=85°C</td>
<td>V&lt;sub&gt;IN&lt;/sub&gt; = 0 or V&lt;sub&gt;DD&lt;/sub&gt;, C&lt;sub&gt;L&lt;/sub&gt;=50pF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C&lt;sub&gt;IN&lt;/sub&gt; Input Capacitance</td>
<td></td>
<td>C</td>
<td>10</td>
<td></td>
<td></td>
<td>pF</td>
</tr>
<tr>
<td>C&lt;sub&gt;OUT&lt;/sub&gt; Output Capacitance</td>
<td></td>
<td>C</td>
<td>12</td>
<td></td>
<td></td>
<td>pF</td>
</tr>
<tr>
<td>C&lt;sub&gt;I/O&lt;/sub&gt; Output Capacitance</td>
<td></td>
<td>C</td>
<td>12</td>
<td></td>
<td></td>
<td>pF</td>
</tr>
</tbody>
</table>

1. Not 100% tested, guaranteed by design characterization
This section describes the AC timing characteristics of the CL450. The timing characteristics are divided into related groups and depicted with one or more timing diagrams and a table of timing values. The groups are:

- **Host Bus Memory and Register Timing**
  - Figure 7-1, Host Bus Local DRAM Read
  - Figure 7-2, Host Bus Register Read
  - Figure 7-3, Host Bus Local DRAM Write
  - Figure 7-4, Host Bus Register Write
  - Table 7-4, Local DRAM or Register Access

- **Host Bus CMEM Timing**
  - Figure 7-5, CMEM Write
  - Figure 7-6, CMEM DMA Write
  - Table 7-5, CMEM Access

- **Host Bus Vectored Interrupt Cycle Timing**
  - Figure 7-7, Vectored Interrupt Cycle with Auto Clear
  - Table 7-6, Vectored Interrupt Cycle with Auto Clear

- **Local DRAM Bus Timing**
  - Figure 7-8, Local DRAM Bus
  - Figure 7-9, Local DRAM CAS-before-RAS Refresh
  - Table 7-7, Local DRAM Bus

- **GCLK, SCLK and RESET Timing**
  - Figure 7-10, GCLK
  - Figure 7-11, RESET
  - Table 7-8, GCLK and RESET
  - Figure 7-12, SCLK Input
  - Table 7-9, SCLK Input

- **Video Bus Timing**
  - Figure 7-13, Video Bus Inputs
  - Figure 7-14, VCLK
  - Table 7-10, Video Bus Inputs
7.2.1 Host Bus Memory and Register Timing
Host requests for access to the CL450’s local DRAM and registers are not always honored immediately since higher priority operations within the CL450 may delay access. In such cases, the CL450 inserts wait states by holding off the assertion of DTACK. At least one wait state is inserted in every local DRAM or register access. Since register and local DRAM accesses are made primarily during initialization of the CL450, and rarely during operation, the impact of these delays on system performance are minimal.

Note: Please see Motorola's MC68070 Technical Summary for an explanation of cycle time naming conventions S0, S0, etc.

Figure 7-1 Timing Diagram - Host Bus Local DRAM Read
1. CMEM does not support byte-wide writes.

Note: Please see Motorola’s MC68070 Technical Summary for an explanation of cycle time naming conventions SB, S0, etc.

**Figure 7-2  Timing Diagram - Host Bus Register Read**
AC Timing Characteristics

Note: Please see Motorola’s MC68070 Technical Summary for an explanation of cycle time naming conventions SB, S0, etc.

Figure 7-3  Timing Diagram - Host Bus Local DRAM Write
The CL450's registers do not support byte-wide writes.

Note: Please see Motorola's MC68070 Technical Summary for an explanation of cycle time naming conventions SB, S0, etc.

Figure 7-4 Host Bus Register Write
### Table 7-4  Timing Characteristics - Local DRAM or Register Access

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>HCLK Clock Period</td>
<td>2 (T31)</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T2</td>
<td>HCLK HIGH Pulse Width</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T3</td>
<td>HCLK LOW Pulse Width</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T4</td>
<td>A[20:19] valid setup to AS LOW</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T5</td>
<td>A[20:19] valid hold past AS HIGH</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T6</td>
<td>D[15:0] setup to AS + UDS + LDS LOW (write)</td>
<td>15</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T7</td>
<td>D[15:0] hold past AS + UDS + LDS HIGH (write)</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T8</td>
<td>AS + UDS + LDS HIGH to DTACK HIGH (write)</td>
<td>2</td>
<td>25</td>
<td>ns</td>
</tr>
<tr>
<td>T9</td>
<td>R/W setup to UDS, LDS LOW (write)</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T10</td>
<td>R/W hold past AS HIGH (write)</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T12</td>
<td>DTACK low to D[15:0] valid (read)</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T13</td>
<td>D[15:0] turn-off time from AS + UDS + LDS HIGH (read)</td>
<td>2</td>
<td>15</td>
<td>ns</td>
</tr>
<tr>
<td>T14</td>
<td>AS HIGH to DTACK HIGH (read)</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T15</td>
<td>R/W setup to AS LOW</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T16</td>
<td>R/W hold past AS HIGH (read)</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T23</td>
<td>AS + UDS + LDS LOW to D[15:0] driven</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T48</td>
<td>HCLK HIGH to DTACK LOW</td>
<td></td>
<td></td>
<td>25 ns</td>
</tr>
<tr>
<td>T49</td>
<td>R/W hold past HCLK HIGH (write)</td>
<td>2 (T1) + 10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T105</td>
<td>A[18:1] hold time from DTACK LOW</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T106</td>
<td>A[18:1] setup time to DTACK LOW</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T107</td>
<td>A[7:1] hold time from AS LOW</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T108</td>
<td>A[18:1] hold time from DTACK LOW</td>
<td>3 (T31) + 10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T122</td>
<td>A[7:1] setup time to AS LOW</td>
<td>TBD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V. Output load capacitance = 50 pF.
3. Not 100% tested, guaranteed by design characterization.
4. Time at which output achieves an open circuit condition; not referenced to an output voltage level.
7.2.2 Host Bus CMEM Timing
CMEM logic has been optimized to minimize data transfer times. Direct writes and DMA transfers to CMEM are synchronized within the CL450 so no host bus wait states are necessary.

Note: Please see Motorola’s MC68070 Technical Summary for an explanation of cycle time naming conventions SB, SO, etc.

Figure 7-5 Timing Diagram - CMEM Write
AC Timing Characteristics

Note: DMAREQ will only be asserted by the CL450 after the DMA Enable bit (DE, see page 8-11) has been correctly set by the host. Once set, any number of falling edges may be produced on DMAREQ until the bit is cleared.

Figure 7-6    Timing Diagram - CMEM DMA Write
### Table 7-5  Timing Characteristics - CMEM Access

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>T11</td>
<td>AS to DTACK LOW (CMEM write)</td>
<td>TBD</td>
<td>30</td>
<td>ns</td>
</tr>
<tr>
<td>T17</td>
<td>D[15:0] setup to DTC LOW</td>
<td>5</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T18</td>
<td>D[15:0] hold from DTC LOW</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T19</td>
<td>DTC LOW Pulse Width</td>
<td>15</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T20</td>
<td>DMAACK hold to DTC LOW</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T21</td>
<td>DMAACK to DMAREQ HIGH (to end request)</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T22</td>
<td>DONE low to DMAREQ HIGH¹</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T36</td>
<td>DTACK LOW to AS HIGH</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T47</td>
<td>AS HIGH to HCLK HIGH</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T109</td>
<td>D[15:0] hold time from AS + UDS + LDS LOW</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T110</td>
<td>DMAACK HIGH time</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T111</td>
<td>DONE pulse width</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T112</td>
<td>DMA cycle time</td>
<td>TBD(T31)</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T113</td>
<td>AS setup to DMAACK LOW</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T114</td>
<td>AS hold to DMAACK HIGH</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T115</td>
<td>R/W HIGH to DMAACK LOW</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T116</td>
<td>R/W hold to DMAACK HIGH</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T129</td>
<td>AS setup to UDS/LDS LOW (CMEM write)</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T130</td>
<td>AS HIGH pulse width</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T131</td>
<td>DMAACK setup to DTC LOW</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

1. Note: DONE causes DMAREQ to be come inactive only if CMEM is not full when the DONE pulse occurs.
7.2.3 Host Bus Vectored Interrupt Cycle Timing

The CL450 supports the 680X0 Vectored Interrupt Cycle. Shown is the Vectored Interrupt Cycle with the CL450’s Auto Interrupt Clear enabled (AIC in HOST_control equal to 1, see page 8-10).

Figure 7-7  Timing Diagram - Vectored Interrupt Cycle with Auto Clear

Note: INTACK should be tied HIGH if not used. When used, INTACK must not be asserted unless INT is asserted.
Table 7-6  Timing Characteristics - Vectored Interrupt Cycle with Auto Clear

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>T24</td>
<td>INTACK HIGH to INT HIGH (release)</td>
<td>30</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T25</td>
<td>A[3:1] setup to INTACK LOW</td>
<td>10</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T26</td>
<td>INTACK LOW to DTACK LOW (vectored interrupt)</td>
<td>30</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T27</td>
<td>INTACK LOW to D[7:0] valid (vectored interrupt)</td>
<td>TBD</td>
<td>15</td>
<td>ns</td>
</tr>
<tr>
<td>T28</td>
<td>D[7:0] turn-off from INTACK HIGH (vectored interrupt)</td>
<td>TBD</td>
<td>15</td>
<td>ns</td>
</tr>
<tr>
<td>T29</td>
<td>INTACK HIGH to DTACK HIGH (vectored interrupt)</td>
<td>25</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T30</td>
<td>A[3:1] hold past INTACK HIGH</td>
<td>0</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T119</td>
<td>INTACK pulse width</td>
<td>TBD</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T120</td>
<td>AS setup time to INTACK LOW</td>
<td>TBD</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>T121</td>
<td>AS hold time from INTACK HIGH</td>
<td>TBD</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V. Output load capacitance = 50 pF.
3. Not 100% tested, guaranteed by design characterization.
4. Time at which output achieves an open circuit condition; not referenced to an output voltage level.

7.2.4 Local DRAM Bus Timing

Local DRAM bus timing is specified differently than the other timing parameters. The DRAM interface times are specified at 0.8 Volts and 2.4 Volts, instead of at 1.5 Volts. This allows the designer to easily cross reference CL450 timing values to DRAM specifications. The Timing Characteristics table also gives the industry-standard DRAM timing parameter names as well as the CL450 parameter names.
Figure 7-8  Timing Diagram - Local DRAM Bus Timing
## Table 7-7  Timing Characteristics - Local DRAM Bus

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>T31</td>
<td>GCLK Period (tCYC)</td>
<td></td>
<td>25</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T61</td>
<td>GCLK HIGH to RAS HIGH</td>
<td>tASR</td>
<td>tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T62</td>
<td>GCLK HIGH to CAS HIGH</td>
<td>tOS</td>
<td>tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T63</td>
<td>GCLK HIGH to Memory Data HIGH or LOW (DRAM write)</td>
<td>tDH</td>
<td>tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T64</td>
<td>GCLK HIGH to WE HIGH or LOW</td>
<td>tRSH</td>
<td>tCYC - 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T65</td>
<td>GCLK HIGH to Memory Address HIGH or LOW</td>
<td>tRMH</td>
<td>tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T66</td>
<td>Read Data Setup Time before CASIN HIGH</td>
<td></td>
<td>5</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T67</td>
<td>Read Data Hold Time after CASIN HIGH</td>
<td></td>
<td>5</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T68</td>
<td>Row Address Setup Time</td>
<td>tASR</td>
<td>tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T69</td>
<td>Write Data Setup Time before CAS LOW</td>
<td>tOS</td>
<td>tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T70</td>
<td>Write Data Hold Time after CAS LOW</td>
<td>tDH</td>
<td>tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T71</td>
<td>RAS Hold Time after CAS LOW</td>
<td>tRSH</td>
<td>tCYC - 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T72</td>
<td>Read Command Hold Time from CAS</td>
<td>tRMH</td>
<td>tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T73</td>
<td>Read Command Setup Time to CAS LOW</td>
<td>tRCS</td>
<td>5*tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T74</td>
<td>Column Address Setup Time to CAS LOW</td>
<td>tASC</td>
<td>tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T75</td>
<td>Column Address Hold Time from CAS LOW</td>
<td>tCAN</td>
<td>tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T76</td>
<td>Row Address Hold Time from RAS LOW</td>
<td>tRAN</td>
<td>2*tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T77</td>
<td>CAS LOW Time</td>
<td>tCAS</td>
<td>tCYC - 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T78</td>
<td>CAS HIGH Time</td>
<td>tCP</td>
<td>tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T79</td>
<td>RAS HIGH Time</td>
<td>tRP</td>
<td>3*tCYC - 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T80</td>
<td>RAS LOW Time</td>
<td>tRAS</td>
<td>4*tCYC - 15</td>
<td>TBD</td>
<td></td>
</tr>
<tr>
<td>T81</td>
<td>Write Command Setup Time to CAS LOW</td>
<td>tWCS</td>
<td>2*tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T82</td>
<td>Write Command Hold Time from CAS LOW</td>
<td>tWCH</td>
<td>2*tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T83</td>
<td>CAS HIGH to Memory Data Driven</td>
<td>tRD</td>
<td>3*tCYC - 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T84</td>
<td>Column Address to CAS HIGH</td>
<td>tAA + T66</td>
<td>2*tCYC - 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T85</td>
<td>RAS to CAS delay</td>
<td>tRCD</td>
<td>3*tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T86</td>
<td>CAS Setup Time to RAS (Memory Refresh Cycle)</td>
<td>tCSR</td>
<td>tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T87</td>
<td>CAS Hold Time from RAS (Memory Refresh Cycle)</td>
<td>tCHR</td>
<td>3*tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T88</td>
<td>RAS HIGH to CAS LOW delay (Memory Refresh Cycle)</td>
<td>tRPC</td>
<td>2*tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T89</td>
<td>Column Address to RAS HIGH</td>
<td>tRAL</td>
<td>2*tCYC - 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T90</td>
<td>RAS LOW to CAS HIGH</td>
<td>tRAC + T66</td>
<td>4*tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T91</td>
<td>RAS Hold Time from CAS Precharge</td>
<td>tRHCP</td>
<td>2*tCYC - 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T93</td>
<td>Read Command Hold Time from RAS</td>
<td>tRHH+RCH</td>
<td>tCYC - 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T94</td>
<td>GCLK to MD three-state (write)</td>
<td></td>
<td>TBD</td>
<td>TBD</td>
<td>ns</td>
</tr>
</tbody>
</table>

1. Inputs switch between 0.0V and 3.5V at 1V/\text{ns} and measurements are made at 0.8V and 2.4V. Output load capacitance = 50 \text{pF}.
2. Not 100% tested, guaranteed by design characterization.
3. MD[15:0] driven only when next cycle will be a write.

7-16  C-Cube Microsystems
Table 7-7 Timing Characteristics - Local DRAM Bus (cont.)

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>T95</td>
<td>CAS HIGH to MD three-state (read)</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
<td>ns</td>
</tr>
<tr>
<td>T96</td>
<td>CASlN LOW Time</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
<td>ns</td>
</tr>
<tr>
<td>T97</td>
<td>CASlN HIGH TIME</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
<td>ns</td>
</tr>
<tr>
<td>T98</td>
<td>CASlN HIGH to CAS LOW</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
<td>ns</td>
</tr>
<tr>
<td>T123</td>
<td>GCLK HIGH to CAS LOW</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
<td>ns</td>
</tr>
<tr>
<td>T124</td>
<td>GCLK HIGH to RAS LOW</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
<td>ns</td>
</tr>
</tbody>
</table>

7.2.5 GCLK, SCLK, and RESET Timing

Figure 7-10 GCLK Timing Diagram

Figure 7-11 RESET Timing Diagram

Table 7-8 Timing Characteristics - GCLK and RESET

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>T31</td>
<td>GCLK frequency</td>
<td>40</td>
<td></td>
<td>MHz</td>
</tr>
<tr>
<td>T31</td>
<td>GCLK Period (tCYC)</td>
<td>25</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T32</td>
<td>GCLK HIGH Pulse Width</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T33</td>
<td>GCLK LOW Pulse Width</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T125</td>
<td>RESET Pulse Width</td>
<td>50(T31)</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V.
AC Timing Characteristics

Figure 7-12 Timing Diagram - SCLK Input

Table 7-9 Timing Characteristics - SCLK Input

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>T51</td>
<td>SCLK Period (^2)</td>
<td>2(T31)</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T52</td>
<td>SCLK HIGH Pulse Width (^2)</td>
<td>T31</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T53</td>
<td>SCLK LOW Pulse Width (^2)</td>
<td>T31</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V.
2. Not 100% tested, guaranteed by design characterization

7.2.6 Video Bus Timing

Figure 7-13 Timing Diagram - Video Bus Inputs
Table 7-10  Timing Characteristics - Video Bus Inputs

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>T38</td>
<td>VCLK Period(^2)</td>
<td>2(T31)+17</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T39</td>
<td>VCLK HIGH Pulse Width</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T40</td>
<td>VCLK LOW Pulse Width</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T41</td>
<td>HSYNC setup to VCLK HIGH</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T42</td>
<td>HSYNC hold from VCLK HIGH</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T43</td>
<td>PD valid from VCLK HIGH</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T44</td>
<td>VOE LOW to PD[23:0] driven</td>
<td>TBD</td>
<td>20</td>
<td>ns</td>
</tr>
<tr>
<td>T45</td>
<td>PD [23:0] turn-off time from VOE HIGH(^3,4)</td>
<td>TBD</td>
<td>20</td>
<td>ns</td>
</tr>
<tr>
<td>T46</td>
<td>PD hold time from VCLK HIGH</td>
<td>TBD</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>T126</td>
<td>Alternate VCLK period</td>
<td>2(T31)</td>
<td>2(T31)</td>
<td>ns</td>
</tr>
<tr>
<td>T127</td>
<td>VCLK setup to GCLK(^5)</td>
<td>TBD</td>
<td></td>
<td>TBD</td>
</tr>
<tr>
<td>T128</td>
<td>VCLK hold from GCLK(^5)</td>
<td>TBD</td>
<td></td>
<td>TBD</td>
</tr>
</tbody>
</table>

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V. Output load capacitance = 50pF.
2. Note: T126 may be substituted for T38 if additional timing constraints in Figure 7-14 (T127 and T128) are met.
3. Not 100% tested, guaranteed by design characterization.
4. Time at which output achieves an open circuit condition; not referenced to an output voltage level.
5. Must be met only when T126 is used in place of T38.
The CL450 is packaged in a plastic quad flat pack (PQFP). The package has an internal heat spreader and a copper-alloy lead frame to improve thermal conductivity. This section includes:

- The CL450 Pinout Diagram
- Tables of CL450 pin connections
  - Host Bus Interface Pins
  - DRAM Bus Interface Pins
  - Video Bus Interface Pins
  - Power and Miscellaneous Pins
- Package Physical Dimensions

The CL450 is shipped in a drypack with desiccant and a humidity monitor. Do not use the parts if the humidity indicator indicates that the humidity is greater than 30% at the initial opening of the drypack. To avoid cracking the plastic during soldering, the parts should be soldered within three days after breaking the drypack seal.

Note the following conventions used in this section:

- "NO CONNECT" pins are not connected internally.
- "RESERVED" pins are reserved for future use. They should be pulled either HIGH or LOW.
Figure 7-15  CL450 PQFP Pinout Diagram
### 7.3.7 Pin List

#### Table 7-11 Host Bus Interface Pins

<table>
<thead>
<tr>
<th>Function</th>
<th>Group</th>
<th>I/O</th>
<th>Pin #</th>
<th>Function</th>
<th>Group</th>
<th>I/O</th>
<th>Pin #</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCLK</td>
<td>Host</td>
<td>I</td>
<td>102</td>
<td>A2</td>
<td>Host</td>
<td>I</td>
<td>66</td>
</tr>
<tr>
<td>HCLK</td>
<td>Host</td>
<td>I</td>
<td>76</td>
<td>A1</td>
<td>Host</td>
<td>I</td>
<td>67</td>
</tr>
<tr>
<td>SCLK</td>
<td>Host</td>
<td>I</td>
<td>123</td>
<td>R/W</td>
<td>Host</td>
<td>I</td>
<td>73</td>
</tr>
<tr>
<td>INT</td>
<td>Host</td>
<td>O</td>
<td>108</td>
<td>D15</td>
<td>Host</td>
<td>I/O</td>
<td>106</td>
</tr>
<tr>
<td>INTACK</td>
<td>Host</td>
<td>I</td>
<td>109</td>
<td>D14</td>
<td>Host</td>
<td>I/O</td>
<td>105</td>
</tr>
<tr>
<td>CFLEVEL</td>
<td>Host</td>
<td>O</td>
<td>78</td>
<td>D[13</td>
<td>Host</td>
<td>I/O</td>
<td>104</td>
</tr>
<tr>
<td>RESET</td>
<td>Host</td>
<td>I</td>
<td>71</td>
<td>D12</td>
<td>Host</td>
<td>I/O</td>
<td>103</td>
</tr>
<tr>
<td>TEST</td>
<td>Host</td>
<td>I</td>
<td>124</td>
<td>D11</td>
<td>Host</td>
<td>I/O</td>
<td>99</td>
</tr>
<tr>
<td>AS</td>
<td>Host</td>
<td>I</td>
<td>74</td>
<td>D10</td>
<td>Host</td>
<td>I/O</td>
<td>98</td>
</tr>
<tr>
<td>A20</td>
<td>Host</td>
<td>I</td>
<td>70</td>
<td>D9</td>
<td>Host</td>
<td>I/O</td>
<td>97</td>
</tr>
<tr>
<td>A19</td>
<td>Host</td>
<td>I</td>
<td>69</td>
<td>D8</td>
<td>Host</td>
<td>I/O</td>
<td>96</td>
</tr>
<tr>
<td>A18</td>
<td>Host</td>
<td>I</td>
<td>48</td>
<td>D7</td>
<td>Host</td>
<td>I/O</td>
<td>93</td>
</tr>
<tr>
<td>A17</td>
<td>Host</td>
<td>I</td>
<td>49</td>
<td>D6</td>
<td>Host</td>
<td>I/O</td>
<td>92</td>
</tr>
<tr>
<td>A16</td>
<td>Host</td>
<td>I</td>
<td>50</td>
<td>D5</td>
<td>Host</td>
<td>I/O</td>
<td>91</td>
</tr>
<tr>
<td>A15</td>
<td>Host</td>
<td>I</td>
<td>51</td>
<td>D4</td>
<td>Host</td>
<td>I/O</td>
<td>90</td>
</tr>
<tr>
<td>A14</td>
<td>Host</td>
<td>I</td>
<td>52</td>
<td>D3</td>
<td>Host</td>
<td>I/O</td>
<td>87</td>
</tr>
<tr>
<td>A13</td>
<td>Host</td>
<td>I</td>
<td>53</td>
<td>D2</td>
<td>Host</td>
<td>I/O</td>
<td>86</td>
</tr>
<tr>
<td>A12</td>
<td>Host</td>
<td>I</td>
<td>55</td>
<td>D1</td>
<td>Host</td>
<td>I/O</td>
<td>85</td>
</tr>
<tr>
<td>A11</td>
<td>Host</td>
<td>I</td>
<td>56</td>
<td>D0</td>
<td>Host</td>
<td>I/O</td>
<td>84</td>
</tr>
<tr>
<td>A10</td>
<td>Host</td>
<td>I</td>
<td>57</td>
<td>DTACK</td>
<td>Host</td>
<td>O</td>
<td>77</td>
</tr>
<tr>
<td>A9</td>
<td>Host</td>
<td>I</td>
<td>58</td>
<td>LDS</td>
<td>Host</td>
<td>I</td>
<td>68</td>
</tr>
<tr>
<td>A8</td>
<td>Host</td>
<td>I</td>
<td>59</td>
<td>UDS</td>
<td>Host</td>
<td>I</td>
<td>100</td>
</tr>
<tr>
<td>A7</td>
<td>Host</td>
<td>I</td>
<td>60</td>
<td>DMAREQ</td>
<td>Host</td>
<td>O</td>
<td>72</td>
</tr>
<tr>
<td>A6</td>
<td>Host</td>
<td>I</td>
<td>62</td>
<td>DMAACK</td>
<td>Host</td>
<td>I</td>
<td>118</td>
</tr>
<tr>
<td>A5</td>
<td>Host</td>
<td>I</td>
<td>63</td>
<td>DTC</td>
<td>Host</td>
<td>I</td>
<td>94</td>
</tr>
<tr>
<td>A4</td>
<td>Host</td>
<td>I</td>
<td>64</td>
<td>DONE</td>
<td>Host</td>
<td>I</td>
<td>117</td>
</tr>
<tr>
<td>A3</td>
<td>Host</td>
<td>I</td>
<td>65</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Open-Drain Output.
### Table 7-12  DRAM Bus Interface Pins

<table>
<thead>
<tr>
<th>Function</th>
<th>Group</th>
<th>I/O</th>
<th>Pin #</th>
<th>Function</th>
<th>Group</th>
<th>I/O</th>
<th>Pin #</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA9</td>
<td>DRAM</td>
<td>0</td>
<td>3</td>
<td>MD8</td>
<td>DRAM</td>
<td>I/O</td>
<td>28</td>
</tr>
<tr>
<td>MA8</td>
<td>DRAM</td>
<td>0</td>
<td>4</td>
<td>MD7</td>
<td>DRAM</td>
<td>I/O</td>
<td>26</td>
</tr>
<tr>
<td>MA7</td>
<td>DRAM</td>
<td>0</td>
<td>5</td>
<td>MD6</td>
<td>DRAM</td>
<td>I/O</td>
<td>25</td>
</tr>
<tr>
<td>MA6</td>
<td>DRAM</td>
<td>0</td>
<td>7</td>
<td>MD5</td>
<td>DRAM</td>
<td>I/O</td>
<td>24</td>
</tr>
<tr>
<td>MA5</td>
<td>DRAM</td>
<td>0</td>
<td>8</td>
<td>MD4</td>
<td>DRAM</td>
<td>I/O</td>
<td>23</td>
</tr>
<tr>
<td>MA4</td>
<td>DRAM</td>
<td>0</td>
<td>9</td>
<td>MD3</td>
<td>DRAM</td>
<td>I/O</td>
<td>20</td>
</tr>
<tr>
<td>MA3</td>
<td>DRAM</td>
<td>0</td>
<td>10</td>
<td>MD2</td>
<td>DRAM</td>
<td>I/O</td>
<td>19</td>
</tr>
<tr>
<td>MA2</td>
<td>DRAM</td>
<td>0</td>
<td>12</td>
<td>MD1</td>
<td>DRAM</td>
<td>I/O</td>
<td>18</td>
</tr>
<tr>
<td>MA1</td>
<td>DRAM</td>
<td>0</td>
<td>13</td>
<td>MD0</td>
<td>DRAM</td>
<td>I/O</td>
<td>17</td>
</tr>
<tr>
<td>MA0</td>
<td>DRAM</td>
<td>0</td>
<td>14</td>
<td>RAS[0]</td>
<td>DRAM</td>
<td>0</td>
<td>46</td>
</tr>
<tr>
<td>MD15</td>
<td>DRAM</td>
<td>I/O</td>
<td>37</td>
<td>RAS[1]</td>
<td>DRAM</td>
<td>0</td>
<td>45</td>
</tr>
<tr>
<td>MD14</td>
<td>DRAM</td>
<td>I/O</td>
<td>36</td>
<td>LCAS</td>
<td>DRAM</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>MD13</td>
<td>DRAM</td>
<td>I/O</td>
<td>35</td>
<td>UCAS</td>
<td>DRAM</td>
<td>0</td>
<td>43</td>
</tr>
<tr>
<td>MD12</td>
<td>DRAM</td>
<td>I/O</td>
<td>34</td>
<td>LCASIN</td>
<td>DRAM</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>MD11</td>
<td>DRAM</td>
<td>I/O</td>
<td>31</td>
<td>UCASIN</td>
<td>DRAM</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>MD10</td>
<td>DRAM</td>
<td>I/O</td>
<td>30</td>
<td>WE</td>
<td>DRAM</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>MD9</td>
<td>DRAM</td>
<td>I/O</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 7-13 Video Bus Interface Pins

<table>
<thead>
<tr>
<th>Function</th>
<th>Group</th>
<th>I/O</th>
<th>Pin #</th>
<th>Function</th>
<th>Group</th>
<th>I/O</th>
<th>Pin #</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD23</td>
<td>Video</td>
<td>0</td>
<td>158</td>
<td>PD9</td>
<td>Video</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>PD22</td>
<td>Video</td>
<td>0</td>
<td>156</td>
<td>PD8</td>
<td>Video</td>
<td>0</td>
<td>138</td>
</tr>
<tr>
<td>PD21</td>
<td>Video</td>
<td>0</td>
<td>155</td>
<td>PD7</td>
<td>Video</td>
<td>0</td>
<td>136</td>
</tr>
<tr>
<td>PD20</td>
<td>Video</td>
<td>0</td>
<td>154</td>
<td>PD6</td>
<td>Video</td>
<td>0</td>
<td>133</td>
</tr>
<tr>
<td>PD19</td>
<td>Video</td>
<td>0</td>
<td>153</td>
<td>PD5</td>
<td>Video</td>
<td>0</td>
<td>132</td>
</tr>
<tr>
<td>PD18</td>
<td>Video</td>
<td>0</td>
<td>151</td>
<td>PD4</td>
<td>Video</td>
<td>0</td>
<td>131</td>
</tr>
<tr>
<td>PD17</td>
<td>Video</td>
<td>0</td>
<td>150</td>
<td>PD3</td>
<td>Video</td>
<td>0</td>
<td>130</td>
</tr>
<tr>
<td>PD16</td>
<td>Video</td>
<td>0</td>
<td>149</td>
<td>PD2</td>
<td>Video</td>
<td>0</td>
<td>128</td>
</tr>
<tr>
<td>PD15</td>
<td>Video</td>
<td>0</td>
<td>148</td>
<td>PD1</td>
<td>Video</td>
<td>0</td>
<td>127</td>
</tr>
<tr>
<td>PD14</td>
<td>Video</td>
<td>0</td>
<td>146</td>
<td>PD0</td>
<td>Video</td>
<td>0</td>
<td>126</td>
</tr>
<tr>
<td>PD13</td>
<td>Video</td>
<td>0</td>
<td>145</td>
<td>HSYNC</td>
<td>Video</td>
<td>I</td>
<td>134</td>
</tr>
<tr>
<td>PD12</td>
<td>Video</td>
<td>0</td>
<td>144</td>
<td>VOE</td>
<td>Video</td>
<td>I</td>
<td>143</td>
</tr>
<tr>
<td>PD11</td>
<td>Video</td>
<td>0</td>
<td>142</td>
<td>VCLK</td>
<td>Video</td>
<td>I</td>
<td>137</td>
</tr>
<tr>
<td>PD10</td>
<td>Video</td>
<td>0</td>
<td>140</td>
<td>VSYNC</td>
<td>Video</td>
<td>I</td>
<td>110</td>
</tr>
</tbody>
</table>

### Table 7-14 Power and Miscellaneous Pins

<table>
<thead>
<tr>
<th>Function</th>
<th>Group</th>
<th>I/O</th>
<th>Pin #</th>
<th>Function</th>
<th>Group</th>
<th>I/O</th>
<th>Pin #</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>6</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>32</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>16</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>47</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>27</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>61</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>38</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>75</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>54</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>89</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>68</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>101</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>83</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>112</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>95</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>125</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>107</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>135</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>116</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>147</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>129</td>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>157</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>141</td>
<td>NO CONNECT</td>
<td>Misc</td>
<td>-</td>
<td>1 - 2</td>
</tr>
<tr>
<td>VCC</td>
<td>Power</td>
<td>PWR</td>
<td>152</td>
<td>NO CONNECT</td>
<td>Misc</td>
<td>-</td>
<td>39 - 42</td>
</tr>
<tr>
<td>RESERVED</td>
<td>Misc</td>
<td>I</td>
<td>111</td>
<td>NO CONNECT</td>
<td>Misc</td>
<td>-</td>
<td>79 - 82</td>
</tr>
<tr>
<td>RESERVED</td>
<td>Misc</td>
<td>I</td>
<td>113 - 115</td>
<td>NO CONNECT</td>
<td>Misc</td>
<td>-</td>
<td>119 - 122</td>
</tr>
<tr>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>11</td>
<td>NO CONNECT</td>
<td>Misc</td>
<td>-</td>
<td>159 - 160</td>
</tr>
<tr>
<td>VSS</td>
<td>Power</td>
<td>GND</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7-24 C-Cube Microsystems
7.3.8 Package Drawings

Figure 7-16  Plastic Quad Flat Pack Physical Dimensions
### Table 7-15  Plastic Quad Flat Pack Physical Dimensions

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>INCHES</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.146 MAX.</td>
<td>3.7 MAX.</td>
</tr>
<tr>
<td>A1</td>
<td>0 MIN.</td>
<td>0 MIN.</td>
</tr>
<tr>
<td>D</td>
<td>1.10 ±0.008</td>
<td>28.0 ±0.20</td>
</tr>
<tr>
<td>E</td>
<td>1.259 ±0.016</td>
<td>32.0 ±0.40</td>
</tr>
<tr>
<td>E1</td>
<td>0.012 TYP.</td>
<td>0.30 TYP.</td>
</tr>
<tr>
<td>E2</td>
<td>0.0256 TYP.</td>
<td>0.65 TYP.</td>
</tr>
<tr>
<td>E3</td>
<td>0.006 TYP.</td>
<td>0.15 TYP.</td>
</tr>
<tr>
<td>E4</td>
<td>0.031 ±0.008</td>
<td>0.80 ±0.20</td>
</tr>
<tr>
<td>H</td>
<td>0.157 MAX.</td>
<td>4.0 MAX.</td>
</tr>
<tr>
<td>Z</td>
<td>0.052 TYP.</td>
<td>1.33 TYP.</td>
</tr>
</tbody>
</table>
This chapter describes each of the CL450 hardware registers. You should be familiar with the relation of these registers to the CL450’s external signals as described in Chapter 3, Signal Descriptions.

The sections in this chapter are:

- 8.1: CL450 Register Categories
- 8.2: CL450 Register Summary
- 8.3: Host Interface Registers
- 8.4: Internal CPU Registers
- 8.5: DRAM Interface Register
- 8.6: Video Interface Registers
8.1 CL450 Register Categories

The CL450 contains 34 registers. Typically, these registers perform low-level operations or act as part of the protocol for communicating with the microapplication executing on the CL450’s internal CPU. For purposes of this reference, they are divided into three categories:

- **Normal**: These registers are used routinely to configure/communicate with the CL450 and its microapplication, although not all applications use all of these registers.

- **Diagnostic**: These registers are used *only* when attempting to characterize a problem as discussed in Appendix E, Troubleshooting. They are accessed by the host using an internal data bus which is shared with the CL450’s internal CPU. Because host use of this bus may conflict with CL450 operations, diagnostic register accesses should be minimized to prevent degradation of CL450 performance.

- **Internal**: These registers are used exclusively by the CL450’s internal CPU. They must not be accessed by the host any time the microapplication is executing; otherwise, indeterminate behavior can occur.

- **Initialization**: These registers should only be accessed by the host prior to the beginning of microapplication execution as part of the chip initialization procedure.

8.1.1 Direct-access Registers

Of the CL450’s 34 registers, 26 can be directly accessed using a single transfer across the CL450’s host interface and are listed in Table 8-1. To access these registers, perform a host read or write with address bits A[20:19] equal to 10\(^2\) and address bits A[7:1] set to the register address.

*Note: Register address values in this chapter are given in byte-wide address space. This is done for the convenience of programmers working with byte-addressable processors.*

*The least significant bit of each address value corresponds to address bit A[0] (which is not supplied to the CL450), and the remaining register address bits are driven to the CL450 on address lines A[7:1].*
Even though register addresses are expressed as byte addresses, the CL450 does not support single-byte access to internal registers; all registers must be accessed 16 bits at a time.

Register address values not listed in Table 8-1 are reserved. Attempting to read or write these register addresses—or to read a write-only register, or to write a read-only register—causes indeterminate results.

### Table 8-1 Direct-access Registers (listed by category)

<table>
<thead>
<tr>
<th>Category</th>
<th>Register Name</th>
<th>Address (byte-wide)</th>
<th>Category</th>
<th>Register Name</th>
<th>Address (byte-wide)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CMEM_control</td>
<td>0x80</td>
<td>Diagnostic</td>
<td>CPU_int</td>
<td>0x54</td>
</tr>
<tr>
<td>Normal</td>
<td>CMEM_dmactrl</td>
<td>0x84</td>
<td></td>
<td>CPU_intenzb</td>
<td>0x26</td>
</tr>
<tr>
<td></td>
<td>CPU_control</td>
<td>0x20</td>
<td></td>
<td>CPU_taddr</td>
<td>0x38</td>
</tr>
<tr>
<td></td>
<td>CPU_tmem</td>
<td>0x46</td>
<td></td>
<td>CPU_tmem</td>
<td>0x46</td>
</tr>
<tr>
<td></td>
<td>CPU_imem</td>
<td>0x42</td>
<td>CMEM_data</td>
<td>0x02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPU_pc</td>
<td>0x22</td>
<td></td>
<td>HOST_scr0</td>
<td>0x92</td>
</tr>
<tr>
<td>Normal</td>
<td>DRAM_refcnt</td>
<td>0xAC</td>
<td>Internal</td>
<td>HOST_scr1</td>
<td>0x94</td>
</tr>
<tr>
<td></td>
<td>HOST_control</td>
<td>0x90</td>
<td></td>
<td>HOST_scr2</td>
<td>0x96</td>
</tr>
<tr>
<td></td>
<td>HOST_intvecr</td>
<td>0x9C</td>
<td></td>
<td>VID_chroma</td>
<td>0xA</td>
</tr>
<tr>
<td></td>
<td>HOST_intvecw</td>
<td>0x98</td>
<td></td>
<td>VID_y</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>HOST_newcmd</td>
<td>0x56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HOST_raddr</td>
<td>0x88</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HOST_rdata</td>
<td>0x8C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VID_control</td>
<td>0xEC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VID_regdata</td>
<td>0xEE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that Table 8-1 gives addresses within an 8-bit address space; that is, the values given should be divided by two before being presented to the CL450 using address pins A[7:1]. This restriction applies because CL450 registers must be accessed using 16-bit operations.
8.1.2 Indirect-access (Video) Registers

The remaining eight registers, listed in Table 8-2, are accessed indirectly by specifying the desired register number in bits 4:1 of the VID_control register (the VRID field), and then reading or writing the VID_regdata register. Note that the “VRID Value” of Table 8-2 replaces the “Address” column of Table 8-1 but that the tables are otherwise identical.

The procedure for accessing an indirect video register is described in Section 8.6.2, Indirect-access Video Registers.

Table 8-2 Indirect-access (Video) Registers (listed by category)

<table>
<thead>
<tr>
<th>Category</th>
<th>Register Name</th>
<th>VRID Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>VID_selaux</td>
<td>0xC</td>
</tr>
<tr>
<td></td>
<td>VID_sela</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>VID_selactive</td>
<td>0x8</td>
</tr>
<tr>
<td></td>
<td>VID_selb</td>
<td>0x1</td>
</tr>
<tr>
<td>Internal</td>
<td>VID_selbor</td>
<td>0x9</td>
</tr>
<tr>
<td></td>
<td>VID_selGB</td>
<td>0xB</td>
</tr>
<tr>
<td></td>
<td>VID_selmode</td>
<td>0x7</td>
</tr>
<tr>
<td></td>
<td>VID_selR</td>
<td>0xA</td>
</tr>
</tbody>
</table>

Note: All register addresses and VRID values not shown in Table 8-1 and Table 8-2 are RESERVED, and attempting to access a register at a reserved location will cause indeterminate results. Also, not all registers may be accessed at all times during CL450 operation. The limitations on when registers may be accessed are discussed in Chapter 10.

Table 8-3 and Table 8-4 give a descriptive summary of the same CL450 registers that were listed in Tables 8-1 and 8-2, respectively. Note that the registers in Table 8-3 that begin “CMEM_” are part of the host interface module, and that 8 of the 12 video registers are listed separately in Table 8-4 because they are addressed indirectly using the VID_control register as a pointer.
Table 8-3  Direct-access Registers (described)

<table>
<thead>
<tr>
<th>Register</th>
<th>Addr.</th>
<th>Used For:</th>
<th>R/W</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMEM_control</td>
<td>0x80</td>
<td>Internal reset, byte swap (normal/initialization)</td>
<td>R/W</td>
<td>8-8</td>
</tr>
<tr>
<td>CMEM_data</td>
<td>0x02</td>
<td>CMEM read port (internal)</td>
<td>R</td>
<td>8-9</td>
</tr>
<tr>
<td>CMEM_dmactrl</td>
<td>0x84</td>
<td>CMEM status, CFLEVEL assertion, DMA enable (normal)</td>
<td>R/W</td>
<td>8-9</td>
</tr>
<tr>
<td>CMEM_status</td>
<td>0x82</td>
<td>CMEM counters (internal)</td>
<td>R</td>
<td>8-11</td>
</tr>
<tr>
<td>CPU_control</td>
<td>0x20</td>
<td>Run enable bit (normal)</td>
<td>R/W</td>
<td>8-19</td>
</tr>
<tr>
<td>CPU_iaddr</td>
<td>0x3E</td>
<td>IMEM write address (normal/initialization)</td>
<td>R/W</td>
<td>8-21</td>
</tr>
<tr>
<td>CPU_intm</td>
<td>0x42</td>
<td>IMEM write data (normal/initialization)</td>
<td>W</td>
<td>8-21</td>
</tr>
<tr>
<td>CPU_intn</td>
<td>0x54</td>
<td>Internal interrupt status (diagnostic)</td>
<td>R/W</td>
<td>8-19</td>
</tr>
<tr>
<td>CPU_pc</td>
<td>0x22</td>
<td>Program counter (normal/initialization)</td>
<td>R/W</td>
<td>8-19</td>
</tr>
<tr>
<td>CPU_taddr</td>
<td>0x3E</td>
<td>TMEM address (diagnostic)</td>
<td>R/W</td>
<td>8-22</td>
</tr>
<tr>
<td>CPU_tmem</td>
<td>0x46</td>
<td>TMEM data (diagnostic)</td>
<td>R</td>
<td>8-22</td>
</tr>
<tr>
<td>DRAM_refcnt</td>
<td>0xAC</td>
<td>Refresh clock count (normal)</td>
<td>R/W</td>
<td>8-23</td>
</tr>
<tr>
<td>HOST_control</td>
<td>0x90</td>
<td>Interrupt control (normal)</td>
<td>R/W</td>
<td>8-12</td>
</tr>
<tr>
<td>HOST_intvecr</td>
<td>0x9C</td>
<td>Read /Veet and /PID (normal)</td>
<td>R</td>
<td>8-13</td>
</tr>
<tr>
<td>HOST_intvecw</td>
<td>0x98</td>
<td>Write /Veet and /PID (normal)</td>
<td>W</td>
<td>8-13</td>
</tr>
<tr>
<td>HOST_newcmd</td>
<td>0x56</td>
<td>New command bit (normal)</td>
<td>R/W</td>
<td>8-14</td>
</tr>
<tr>
<td>HOST_raddr</td>
<td>0x88</td>
<td>Pointer to command/status register (normal)</td>
<td>R/W</td>
<td>8-13</td>
</tr>
<tr>
<td>HOST_rdata</td>
<td>0x8C</td>
<td>Data port to command/status register (normal)</td>
<td>R/W</td>
<td>8-14</td>
</tr>
<tr>
<td>HOST_scr0</td>
<td>0x92</td>
<td>SCR – lower portion (internal)</td>
<td>R/W</td>
<td>8-17</td>
</tr>
<tr>
<td>HOST_scr1</td>
<td>0x94</td>
<td>SCR – middle portion (internal)</td>
<td>R/W</td>
<td>8-16</td>
</tr>
<tr>
<td>HOST_scr2</td>
<td>0x96</td>
<td>SCR – upper portion (internal)</td>
<td>R/W</td>
<td>8-16</td>
</tr>
<tr>
<td>VID_control</td>
<td>0xEC</td>
<td>Pointer to indirect video register (normal/initialization)</td>
<td>R/W</td>
<td>8-25</td>
</tr>
<tr>
<td>VID_regdata</td>
<td>0xEE</td>
<td>Data port for indirect video registers (normal/initialization)</td>
<td>W</td>
<td>8-25</td>
</tr>
<tr>
<td>VID_chroma</td>
<td>0x0A</td>
<td>Chrominance data port (internal)</td>
<td>W</td>
<td>8-25</td>
</tr>
<tr>
<td>VID_y</td>
<td>0x00</td>
<td>Luminance data port (internal)</td>
<td>W</td>
<td>8-26</td>
</tr>
</tbody>
</table>

Table 8-4  Indirect-access (Video) Registers (described)

<table>
<thead>
<tr>
<th>Register</th>
<th>VRID</th>
<th>Used For:</th>
<th>R/W</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>VID_sela</td>
<td>0x0</td>
<td>Conversion coefficients (internal)</td>
<td>W</td>
<td>8-26</td>
</tr>
<tr>
<td>VID_selseed</td>
<td>0x8</td>
<td>Width of active region (internal)</td>
<td>W</td>
<td>8-28</td>
</tr>
<tr>
<td>VID_selaux</td>
<td>0xC</td>
<td>Pixel data high byte (normal/initialization)</td>
<td>W</td>
<td>8-29</td>
</tr>
<tr>
<td>VID_selb</td>
<td>0x1</td>
<td>Conversion coefficients (internal)</td>
<td>W</td>
<td>8-26</td>
</tr>
<tr>
<td>VID_selbor</td>
<td>0x9</td>
<td>Left border size (internal)</td>
<td>W</td>
<td>8-28</td>
</tr>
<tr>
<td>VID_selGB</td>
<td>0xB</td>
<td>Border green, blue values (internal)</td>
<td>W</td>
<td>8-29</td>
</tr>
<tr>
<td>VID_selmode</td>
<td>0x7</td>
<td>RGB or YCbCr mode (internal)</td>
<td>W</td>
<td>8-27</td>
</tr>
<tr>
<td>VID_selR</td>
<td>0xA</td>
<td>Border red value (internal)</td>
<td>W</td>
<td>8-28</td>
</tr>
</tbody>
</table>
Figure 8-1 shows a conceptual model of the CL450’s four logical register interfaces. This model does not correspond exactly to the physical structure of the device; instead, it has been simplified to show the programmer’s interface.

The sections that follow describe the registers contained in each CL450 interface module. In the detailed definitions given for each register, note that bits marked Res are reserved. Reserved bits should be written as zeros. Reading from reserved bits gives undefined data.

This section describes the registers in the host interface module. Figure 8-2 shows the internal structure of the host interface module.

Note: Registers accessed by the host that are not contained on the host interface module are accessed through the host interface logic that connects across the CL450 internal bus.
Figure 8-2  Internal Structure of the Host Interface Module
8.3.1 CMEM Registers

The host processor uses these registers to access CMEM for initialization, control, monitoring, and diagnostic purposes.

<table>
<thead>
<tr>
<th>CMEM_control (initialization)</th>
<th>0x80</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>Res</td>
<td>Rst</td>
</tr>
</tbody>
</table>

**Rst**  
Internal Reset (bit 6)  
R/W

The host processor writes a 1 to this bit to cause the CL450 to perform a software reset. A software reset initializes the CL450 in the same way as the hardware reset described in Chapter 4, with the exception that the host interface remains active to allow the host processor to clear this bit to 0. The host processor must clear this bit before initiating another software reset; Rst is not cleared automatically. After any reset, the CL450 must be re-initialized by the host.

**BS**  
Byte Swap (bit 5)  
R/W

When BS is 1, words of data written to the CL450’s CMEM are byte-swapped. (Data written to other internal registers in the CL450 are not swapped regardless of the setting of this bit.)

The byte-swapping capability is provided to ensure compatibility between the CL450, which uses the Motorola style of byte ordering, and host processors that use the Intel style. In the Motorola style, a word address points to the more significant byte (MSB); the less significant byte (LSB) is located at the next higher address. In the Intel style, the byte ordering is reversed. Figure 8-3 shows the memory organization for the two styles.

**Figure 8-3**  
Motorola and Intel Byte Ordering
**CR**  
**CMEM Request (bit 2)**  
When *CRE* (below) is 1, the CL450 sets *CR* to 1 when CMEM contains four or more entries, and clears *CR* to zero when CMEM contains one or no entries. The CL450's internal DRAM controller transfers data from CMEM to the bitstream buffer in the local DRAM when *CR* is 1.

**CRE**  
**CMEM Request Enable (bit 1)**  
When *CRE* is set to 1, CMEM requests a DRAM transfer to the bitstream buffer using *CR* as described above. The microapplication sets this bit to 1 during initialization. When *CRE* is set to 0, the DRAM controller is instructed not to consume any data.

**CRst**  
**CMEM Reset (bit 0)**  
When a 1 is written to *CRst*, the counters in the CMEM_status register are reset to 0, and any data stored in CMEM is deleted. *CRst* must be 0 for normal operation.

---

<table>
<thead>
<tr>
<th>(internal)</th>
<th>0x02</th>
<th>CMEM_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x84</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CData**  
**CMEM Data (bits 15:0)**  
This 16-bit field is the read port from CMEM. The CMEM Read Counter (*CRCnt*) in the CMEM_status register points to the read address in CMEM.

---

1Q – 4Q  
**Empty Status (bits 8:5)**  
These four bits indicate the empty status of CMEM as shown in Table 8-5. When using programmed accesses to write data to CMEM, the application program must check this register or the CFLEVEL pin before writing to ensure that CMEM does not overflow. Note that each of bits [8:5] has a corresponding bit in the range [4:1] which is used as described below.
Table 8-5  

<table>
<thead>
<tr>
<th></th>
<th>1Q</th>
<th>2Q</th>
<th>3Q</th>
<th>4Q</th>
<th>Empty Entries in CMEM</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0–3</td>
<td>Full/nearly full</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4–7</td>
<td>1/4+ empty</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8–11</td>
<td>1/2+ empty</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>12–15</td>
<td>3/4+ empty</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>Empty</td>
</tr>
</tbody>
</table>

1Q – 4QE  Empty Status Enables (bits 4:1)  R/W

These four bits enable the assertion of the CL450’s output signal CFLEVEL based on the state of the empty status bits 1Q through 4Q described above. Table 8-6 shows how these bits control the assertion of CFLEVEL.

Table 8-6  CFLEVEL Assertion Control

<table>
<thead>
<tr>
<th></th>
<th>1Q</th>
<th>2Q</th>
<th>3Q</th>
<th>4Q</th>
<th>CFLEVEL Asserted Upon Assertion of Bit:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td></td>
<td>2Q</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td></td>
<td>3Q</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>4Q</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>CFLEVEL Not Asserted</td>
</tr>
</tbody>
</table>

1. Note that this bit will not necessarily become 1 during normal operations.

DE  DMA Enable (bit 0)  R/W

When this bit is written with 1 by the host and CMEM is not full, the CL450’s DMA capability is enabled. Writing a 1 to this bit if CMEM is full will not enable DMA. Writing a 0 to this bit always disables DMA, regardless of the CMEM fullness level. The value read from this bit will always be the last value written (or 0 if cleared by the DONE pin), and does not necessarily reflect whether or not DMA is enabled.

When DMA is enabled, the CL450 generates a DMA request by asserting DMAREQ (active low) whenever CMEM has at least one empty space. Note that this bit is automatically cleared to 0 by the CL450 when the DONE pin is asserted by the host and CMEM is not full, as described in Section 4.5, CMEM Write Timing.
15 13 12 8 7 4 3 0

Res CDCtr CWCtr CRCtr

**CDCtr** Difference Counter (bits 12:8)  R
This five-bit field contains the count of the current number of words in CMEM (from 0 to 16). CDCtr is incremented when a word is written to CMEM (from the host using programmed access or DMA) and decremented when a word is read from CMEM.

**CWCtr** Write Counter (bits 7:4)  R
This four-bit field contains the CMEM write pointer. CWCtr is incremented when a word is written to CMEM from the host using programmed access or DMA.

**CRCtr** Read Counter (bits 3:0)  R
This four-bit field contains the CMEM read pointer. CRCtr is incremented when a word is read from CMEM to the bitstream buffer in the CL450’s local DRAM.

### 8.3.2 Interrupt Control Registers
The HOST_control, HOST_intvecr, and HOST_intvecw registers control the CL450’s generation of external interrupts. The HOST_control register determines the interrupt mode and the state of the INT pin, and the other two registers provide access to the interrupt vector used when INTACK is asserted (LOW).

The interrupt modes supported by the CL450 are:

- Non-vectored interrupt
- Vectored interrupt with no auto clear
- Vectored interrupt with auto clear

At device initialization, the host is responsible for ensuring that the AIC and VIE bits in the HOST_control register have the correct values for the desired operations (see Table 8-7) before the CL450 microapplication attempts to issue the first interrupt to the host. Subsequent writes to the register to reset Int must use a read-modify-write operation to preserve the value of the other bits in the HOST_control register.
Note: Whether the CL450 produces interrupts to the host, and what events cause these interrupts, is controlled by the CL450 microapplication. See Chapter 12, Interrupts, for a description of the communications protocol between the host and the microapplication for selecting and servicing interrupts.

This bit controls the external interrupt signal INT. When INT is 0, the CL450 will assert INT, driving it LOW. When INT is 1, the CL450 does not drive INT, which will be pulled HIGH (inactive) by an external pullup resistor.

When auto clear is enabled (AIC is 1), INT is set to 1 by the CL450 after the host acknowledges the interrupt by asserting INTACK. When auto clear is disabled (AIC is 0), the host processor must explicitly write INT by a read-modify-write operation.
HOST_intvecr and HOST_intvecw allow the host to access the interrupt vector and interrupt priority ID stored within the CL450. The host processor writes to HOST_intvecw and reads from HOST_intvecr. The information transferred is used only during the interrupt acknowledge cycle, which occurs when INTACK is asserted and VIE in HOST_control is 1.

**IPID**  
Interrupt Priority ID (bits 10:8)  
During the interrupt acknowledge cycle, the CL450 compares bits A[3:1] of the host address bus with this three-bit field. If they match, the CL450 outputs the interrupt vector contained in IVect on D[7:0]. IPID is forced to the value 0 by hardware reset. The host driver should set IPID to match the CL450’s interrupt priority in the system hardware. IPID should be set once at initialization.

**IVect**  
Interrupt Vector (bits 7:0)  
The CL450 outputs the value in this eight-bit field during the interrupt acknowledge cycle if the interrupt priority ID, IPID, matches A[3:1]. IVect is forced to the value 0xF by a hardware reset.

### 8.3.3 Command/Status Registers

These registers—HOST_raddr, HOST_rdata, and HOST_newcmd—allow the host processor to load commands and arguments into the CL450 and to read the status of the microapplication. (See Chapter 11 for information about the CL450 macro commands.)

**HAddr**  
Register Address (bits 3:0)  
R/W  
This four-bit field points to one of the 16 locations of HMEM register space. The HMEM location which is selected by this
field may be accessed by the host reading or writing the HOST_rdata register. After a write operation takes place, HAddr is automatically incremented by one. For read operations, the host processor must write the address each time before reading HOST_rdata. The arrangement of information in HMEM (under microapplication control) is shown in section 11.1, Writing Macro Commands.

**HOST_rdata**

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>HData</td>
<td>16</td>
<td>Host Data (bits 15:0)</td>
<td>0x8C</td>
</tr>
</tbody>
</table>

This 16-bit field accesses the location in the register file currently pointed to by HAddr in the HOST_raddr register.

**HOST_newcmd**

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>1</td>
<td>New Command (bit 0)</td>
<td>0x56</td>
</tr>
</tbody>
</table>

The host processor sets Cmd to 1 after writing the opcode and arguments for a new command to HMEM. The internal CPU resets Cmd to 0 after reading the opcode and arguments of the new command. The host processor must not write to the command locations in HMEM if Cmd is set to 1.

**Command Write Data Flow**

Figure 8-4 shows the general flow for a command write. The circled numbers in the figure refer to the steps below. The syntax of the command words and the arguments are described in Chapter 11, Macro Commands.
The sequence for writing commands to the CL450 is as follows:

1. The host processor reads the new-command bit, $Cmd$, in HOST_newcmd (address $0x56$). If $Cmd$ is 1, the CL450 is accessing the command locations in HMEM and the host processor should not access them. When $Cmd$ is 0, the host processor can proceed with writing a new command.

2. The host processor sets HOST_raddr (address $0x88$) to the HMEM address of the first location used for storing commands.

3. The host processor performs successive write operations to HOST_rdata (address $0x8C$) to write the command word and the arguments (if any).

4. After the command word and all arguments have been written, the host processor sets $Cmd$ to 1.

5. The CL450’s CPU reads the contents of the command registers.

6. The CL450’s CPU clears $Cmd$ to 0 to indicate that it has completed rendering the command and that the host processor can initiate another command write operation.
8.3.4 System Clock Reference Registers
These registers provide access to the CL450’s internal system clock counter. The system clock counter is 33-bits wide (MPEG bitstreams use 33-bit clock values) and is used when synchronizing the decompressed video with other data, in particular decompressed audio.

Typically, the host processor extracts system clock values (SCRs or PTSs) from the bitstream and updates the system timer with these values through use of the AccessSCR() macro command. (See page 11-13 for a description of how the AccessSCR() command is used to write or read the CL450’s internal counter.) The CL450 then compares the presentation time stamps (PTSs) associated with frames in the bitstream with the system clock counter to decide whether to display, skip, or re-display each frame.

The system clock counter is typically clocked by a 90 kHz signal. This signal can be derived either from GCLK or from an external reference on the SCLK input signal. The source of the 90 kHz signal is programmed by the CS bit in the HOST_scr2 register. Regardless of the source, the input signal is divided down by the divisor (Div) programmed in HOST_scr2 to create the 90 kHz signal.

Note: The host must not access these registers while the microapplication is executing (to prevent access collisions between the host and the microapplication). The microapplication provides facilities for the host to update the system clock counter but forces CS and Div to 1 and 444, respectively.

HOST_scr2 (internal) 0x96

<table>
<thead>
<tr>
<th>15</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Res</td>
<td>CS</td>
<td>Div</td>
<td></td>
<td></td>
<td>SysClkHigh</td>
</tr>
</tbody>
</table>

HOST_scr1 (internal) 0x94

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Res</td>
<td>SysClkMid</td>
</tr>
</tbody>
</table>
This bit determines the source of the signal which will be used to generate the 90 kHz clock. When CS is 1, the clock is derived from GCLK. When CS is 0, the 90 kHz signal is derived from an external reference connected to the input signal SCLK.

This nine-bit field contains the divisor used to derive the 90 kHz input for the system clock counter from the source selected by CS. For example, to derive the system clock from a 40.000 MHz GCLK, the value of Div is given by

\[
Div = \text{int}\left(\frac{40.000 \times 10^6}{90 \times 10^3}\right) = 444
\]

When a 90 kHz signal is applied to SCLK and CS is set to 0, Div should be 1 (divide by 1).

Note: The divisor may only be written to a value other than 444 if the microapplication is not executing; otherwise, the microapplication fixes the divisor at a value of 444.

These three fields taken together comprise a 33-bit value for the CL450’s internal system timer.

When these registers are read, HOST_scr0 must be read first. When the HOST_scr0 register is read, the CL450 loads the higher-order 18 bits into HOST_scr1 and HOST_scr2. The system clock continues to count while the registers are being read.

These registers should be written in order starting with HOST_scr2. The internal timer is loaded with the new value when the last register is written. When the internal timer counts past its maximum value of \(2^{33} - 1\), the timer rolls over to 0.
This next section describes the registers that allow application programs to communicate with the CL450’s internal CPU. Figure 8-5 shows the registers in the CPU.

8.4.1 CPU Execution Registers
These registers are used to enable the CPU and to monitor the execution of the internal microapplication.

*Note: The bits in the CPU_int and CPU_intenb registers and the \(LE\) bit in the CPU_pc register control interrupts from CL450 on-chip logic to the CL450 CPU and are unrelated to the interrupts which the CL450 issues to the host.*
**Internal CPU Registers**

(normal)  

```
<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Res</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CEn</td>
</tr>
</tbody>
</table>
```

*CEn*  
**CPU Run Enable (bit 0) R/W**

When the host processor sets this bit to 1, the internal CPU is enabled and begins executing instructions. During a software or hardware reset, the CL450 clears *CEn* to 0, halting the CL450’s CPU.

(initialization)  

```
<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Res</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IE</td>
</tr>
</tbody>
</table>
```

*IE*  
**Interrupt Enable (bit 9) R/W**

When *IE* is 1, the internal CPU interrupts are enabled. When *IE* is 0, the internal CPU interrupts are disabled. Once the CL450 is initialized and the microapplication is started, the microapplication controls this bit. (Note that interrupts to the CL450’s CPU are not related to the interrupts that the microapplication issues to the host.)

```
<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Res</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PC</td>
</tr>
</tbody>
</table>
```

*PC*  
**Program Counter (bits 8:0) R/W**

*PC* is a nine-bit field that holds the current contents of the program counter used by the internal CPU. When a new value is written to *PC*, the internal CPU executes the instruction at that address at the next instruction cycle.

(diagnostic)  

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Res</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

*NCS*  
**New Command Interrupt Status (bit 4) R**

The internal CPU sets *NCS* to 1 to indicate that the new command interrupt is active. *NCS* is cleared to 0 when the interrupt service routine is completed.

*VSS*  
**VSYNC Interrupt Status (bit 1) R/W**

The CL450 sets *VSS* to 1 on the rising edge of VSYNC. This action interrupts the internal CPU, which executes an interrupt service routine that clears *VSS*. 

Registers  8-19
**CPU_intenb**

(disciplinary)

<table>
<thead>
<tr>
<th>15</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Res</td>
<td>NCE</td>
<td>Res</td>
<td>VSE</td>
<td>Res</td>
<td></td>
</tr>
</tbody>
</table>

**NCE** New Command Interrupt Enable (4) R/W

The microapplication sets NCE to 1 to enable the new command interrupt.

**VSE** VSYNC Interrupt Enable (bit 1) R/W

The microapplication sets VSE to 1 to enable the vertical sync interrupt.

### 8.4.2 IMEM Access Registers

The IMEM access registers, CPU_iaddr and CPU_imem, provide a mechanism by which the host processor can write 32-bit microapplication words into the internal instruction memory (IMEM) to initialize the CL450. IMEM can hold up to 512 32-bit microapplication words. The procedure for writing to IMEM is shown in Figure 8-6. The circled numbers refer to the steps that follow.

![Figure 8-6 IMEM Write Data Flow](image)

**Figure 8-6 IMEM Write Data Flow**
The sequence of events for writing to the IMEM is:

1. Ensure that the CPU enable bit, CEn, is 0.
2. The host processor loads the IMEM address to be written into CPU_iaddr bits 9:1. Note that this requires the address to be shifted left one bit (multiplied by 2) before the write.
3. The host processor writes successive pairs of 16-bit words to the CPU_imem register. (Each pair of words makes up a single 32-bit instruction.) The CL450 automatically increments CPU_iaddr with every other write to CPU_imem.

(initialization) 0x3E

<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Res</td>
<td>WAdd</td>
<td></td>
<td>Res</td>
<td></td>
</tr>
</tbody>
</table>

WAdd  Write Address (bits 9:1)  R/W
This nine-bit field contains the address in the IMEM to which the write operation is to be performed. WAdd is automatically post-incremented by the CL450 when 32-bit data is written to CPU_imem (two 16-bit writes).

(initialization) 0x42

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WData</td>
<td></td>
</tr>
</tbody>
</table>

WData  Write Data (bits 15:0)  W
The host processor writes the IMEM data to this 16-bit field. The data written to WData is transferred internally to the IMEM address pointed to by CPU_iaddr.

Note: Writes to this register must be done in pairs, with the first word written containing the most significant 16 bits of each instruction. If writes are not performed in pairs, the auto-increment feature of the CPU_iaddr register and the contents of IMEM will be left in an unknown state.
8.4.3 TMEM Access Registers
These two registers, CPU_taddr and CPU_tmem, provide a mechanism by which the host processor can access the CL450’s “temporary” memory (TMEM) that is normally used only by the internal CPU. The host processor only accesses these registers for diagnostic purposes.

To avoid interfering with the normal operation of the microapplication, consult with a C-Cube Microsystems technical support specialist before using these registers.

CPU_taddr  
(diagnostic)  
0x38

<table>
<thead>
<tr>
<th>15</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>TAddr</td>
</tr>
</tbody>
</table>

TAddr   TMEM Address (bits 6:0)   R/W
This seven-bit field contains the address in TMEM for read and write operations. After a read or write operation is completed, the value of TAddr is incremented by one.

CPU_tmem  
(diagnostic)  
0x46

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TData</td>
</tr>
</tbody>
</table>

TData   TMEM Data (bits 15:0)   R/W
This 16-bit field is used to read data from and write data to TMEM.

8.5 DRAM Interface Register

This section describes the register used by application programs to set the refresh time for the DRAM controller. Figure 8-7 shows the DRAM interface.
RefCnt

Refresh Clock Count (bits 11:0)

This 12-bit field sets the number of GCLK periods between DRAM refresh cycles. The CL450 refreshes a single page at a time rather than refreshing every page of DRAM in a single operation. RefCnt should be set to a value that allows the CL450 to refresh all of the pages (typically 512) within the refresh time required by the DRAMs in use (typically 8ms).

RefCnt should only be written once after power-up. The value is calculated from the formula

\[
\text{RefCnt} = \text{floor}\left(\frac{\left(\text{RP} - 512 \times \text{GCLK}\right)}{(\text{DR} + 1)}\right)
\]

where \(\text{RP}\) is the DRAM refresh period (\(t_{\text{REF}}\)), \(\text{GCLK}\) is the GCLK period, and \(\text{DR}\) is the number of rows in each DRAM.

In this formula, “512 x GCLK” represents the maximum time that the CL450 DRAM controller stays in page mode. DRAM refresh cycles do not interrupt page mode cycles, so the refresh period must be shortened to compensate.

“\(\text{DR} + 1\)” represents the number of refresh cycles that must be done in a refresh period because the DRAM controller can delay one refresh request if other requests are pending. When there are two banks of local DRAM, both banks are refreshed at the same time.

For a typical system with a local DRAM of 512 rows with a refresh period of 8ms and a GLCK period of 25ns, the value of RefCnt should be 622.
8.6.1 Direct-access Video Registers
These registers are directly accessed using a register read or register write operation.
**VRID** Video Register ID (bits 4:0)  R/W
This five-bit field points to one of the indirect video registers. To access one of the indirect video registers, the host processor loads the video register ID into VRID, then performs a read from or a write to the direct register VID_regdata. The video register IDs are listed in numerical order in Table 8-8. VRIDs not listed are reserved and should not be used.

<table>
<thead>
<tr>
<th>VRID</th>
<th>Video Register</th>
<th>VRID</th>
<th>Video Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VID_sela</td>
<td>9</td>
<td>VID_selbor</td>
</tr>
<tr>
<td>1</td>
<td>VID_selb</td>
<td>A</td>
<td>VID_seIR</td>
</tr>
<tr>
<td>7</td>
<td>VID_selmode</td>
<td>B</td>
<td>VID_selGB</td>
</tr>
<tr>
<td>8</td>
<td>VID_selactive</td>
<td>C</td>
<td>VID_selaux</td>
</tr>
</tbody>
</table>

**VRData** Video Register Data (bits 15:0)  W
The host processor uses this 16-bit field to read from and write to the indirect video register pointed to by VRID in VID_control.

**ChromaData** Chroma Data (bits 15:0)  W
Diagnostic programs may use this 16-bit field to write chrominance (CbCr) data to the internal video FIFO (VMEM).
8.6.2 Indirect-access Video Registers

These registers are accessed by first writing the appropriate video register ID (VRID) to the VID_control register, then performing the desired operation by reading from or writing to VID_regdata. Application programs should only access these registers when the CL450's CPU is disabled to avoid interfering with the video display.

**VID_y**

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VID_y</td>
<td></td>
<td>0x00</td>
</tr>
<tr>
<td>(internal)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**YData**  
Y Data (bits 15:0)

Diagnostic programs may use this 16-bit field to write luminance (Y) data to the internal video FIFO (VMEM).

**VID_sela**

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VID_sela</td>
<td></td>
<td>VRID = 0x0 [Indirect]</td>
</tr>
<tr>
<td>(internal)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**VID_selb**

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VID_selb</td>
<td></td>
<td>VRID = 0x1 [Indirect]</td>
</tr>
<tr>
<td>(internal)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**K3 - K0** Conversion Coefficients

These four 8-bit fields set the coefficients for the YCbCr-to-RGB conversion performed by the internal color-space converter. The equations for the color-space conversion are:

- \( \text{Red} = Y + D \times Cr \)
- \( \text{Blue} = Y + B \times Cb \)
- \( \text{Green} = Y + A \times Cb + C \times Cr \)

where \( A, B, C, \) and \( D \) are 10-bit two's-complement coefficients whose binary representations are given in Figure 8-9 (note the location of the binary point).
To set the CL450 for CCIR 601 chromaticity as specified in the MPEG standard, the values of the coefficients should be set as shown below:

\[
\begin{align*}
K0 &= 0xA8 \\
K1 &= 0xC6 \\
K2 &= 0x49 \\
K3 &= 0x67
\end{align*}
\]

**Figure 8-9**  Binary Representations of Conversion Coefficients

When this bit is 1, the CL450 operates in RGB mode. When this bit is 0, the CL450 operates in YCbCr mode.

\[
\begin{array}{c|c}
\text{Res} & \text{RGB} \\
\hline
15 & 1 \\
\end{array}
\]

**VID_selmode**

**VRID** = 0x7 [Indirect]

**RGB**  
RGB Mode Select (bit 0)  
When this bit is 1, the CL450 operates in RGB mode. When this bit is 0, the CL450 operates in YCbCr mode.
### Video Interface Registers

#### VID_selactive

**VID_selactive** (internal)  
**VRID** = 0x8 [Indirect]

<table>
<thead>
<tr>
<th>15</th>
<th>11</th>
<th>10</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Res</td>
<td>VWID</td>
<td>OxF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**VWID** Width of active region (bits 10:4)  
**VWID** determines the width of the active region. The width in pixels of the active region is set to the following value after horizontal interpolation (i.e., in CCIR 601 resolution):

\[
width = 8 \times VWID + 1
\]

#### VID_selbor

**VID_selbor** (internal)  
**VRID** = 0x9 [Indirect]

<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Res</td>
<td>BorLeft</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BorLeft** Left Border Size (bits 9:0)  
This 10-bit field contains the number of border pixels output on each line from the rising (inactive) edge of HSYNC to the first pixel of the active window. The number of left border pixels must be greater than or equal to 10 to ensure correct operation.

#### VID_selR

**VID_selR** (internal)  
**VRID** = 0xA [Indirect]

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Res</td>
<td>BorRd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BorRd** Border, Red Component (bits 7:0)  
This eight-bit field sets the red component of the border (in RGB operation) and the Cr component (in YCbCr operation).
### Video Interface Registers

**VRID = 0xB [Indirect]**

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BorGrn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BorBl</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BorGrn**  
Border, Green Component (bits 15:8)  
This eight-bit field sets the green component of the border (in RGB operation) and the Y component (in YCbCr operation).

**BorBl**  
Border, Blue Component (bits 7:0)  
This eight-bit field sets the blue component of the border (in RGB operation) and the Cb component (in YCbCr operation).

### (initialization)

**VRID = 0xC [Indirect]**

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Res</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PixHData</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PixHData**  
Pixel Data High Byte (bits 7:0)  
When the CL450 is set for YCbCr operation, the upper eight bits of the output pixel data bus, PD[23:16], do not contain pixel data. In this case, this eight-bit value is output on PD[23:16].
Video Interface Registers
Microapplication Overview

The CL450 performs its higher-level functions by executing a microapplication on an internal CPU. The microapplication is supplied by C-Cube Microsystems with the hardware and is considered an integral part of the product.

The microapplication must be loaded into the CL450 by a software driver; it needs to be located in specific sections of the local DRAM and on-chip IMEM. Additional information on loading and executing this code is contained on the distribution disk that is provided with the CL450.

The operation of the CL450 microapplication and the host processor can be described by a set of “process configurations”—each a named group of software processes, some executing within the host processor and some executing within the CL450’s microapplication. Five different process configurations occur at different times in a CL450 system. These process configurations are shown in Figure 9-1 and Figure 9-2.
Microcode-load Configuration *(See Chapter 10.)*

Initialization Configuration *(See Chapter 10.)*

Host Interface

DRAM Interface

CL450

Registers

IMEM

HMEM

Microcode Load Process

Host

DRAM

Variables

Idle Configuration

Pause Configuration

Decoding Configuration

(See Figure 9-2.)
Within each process configuration shown in Figure 9-1 and Figure 9-2, shaded ovals indicate individual but concurrently-running software processes executed by the host or the CL450, rectangles indicate hardware resources, and arrows indicate logical data transfers. (Physical buses used for data transfers are not indicated.) The large arrows pointing to separate process configurations in Figure 9-1 indicate possible microapplication progress over time in a CL450-based system.

A total of six separate software processes are shown (shaded ovals) in the five process configurations of Figure 9-1 and Figure 9-2. They are Microcode-load, Initialization, Command, Bitstream Input, Decode, and Display. The first two of these processes are described in Chapter 10 (Initialization). The remainder of these processes execute simultaneously when the system is in the Decoding process configuration shown in Figure 9-2 and are described in more detail following this figure.

![Figure 9-2 CL450 Decoding Process Configuration](image-url)
9.1.1 Command Process

The primary method by which the microapplication executing on the CL450's CPU receives direction from the host software is by means of macro commands, which are function codes and parameter values written into HMEM by the host (see Table 9-1, Macro Command Summary). Once the function code and parameters are written, the microapplication takes these values and either immediately executes the operation corresponding to the function code or stores the complete macro command into the Command FIFO (see Figure 11-3) for later execution.

Table 9-1   Macro Command Summary

<table>
<thead>
<tr>
<th>Category</th>
<th>Priority</th>
<th>Effect on Command State</th>
<th>Name</th>
<th>Description</th>
<th>Function Code (0x)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set-type</td>
<td>Low</td>
<td>No</td>
<td>SetBlank()</td>
<td>Blanks/unblanks output window</td>
<td>030f</td>
<td>11-29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetBorder()</td>
<td>Sets output window location</td>
<td>0407</td>
<td>11-30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetColorMode()</td>
<td>Enables/disables color-space converter</td>
<td>0111</td>
<td>11-34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetInterruptMask()</td>
<td>Enables/disables interrupts to host</td>
<td>0104</td>
<td>11-36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetThreshold()</td>
<td>Specifies bitstream buffer emptiness</td>
<td>0103</td>
<td>11-38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetVideoFormat()</td>
<td>Configures output resolution and timing</td>
<td>0105</td>
<td>11-40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetWindow()</td>
<td>Sets output window size and contents</td>
<td>0406</td>
<td>11-42</td>
</tr>
<tr>
<td>Play-type</td>
<td>Low</td>
<td>Yes</td>
<td>DisplayStill()</td>
<td>Decodes/displays single still picture(^1)</td>
<td>000c</td>
<td>11-14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pause()</td>
<td>Keeps last picture on display</td>
<td>000e</td>
<td>11-25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Play()</td>
<td>Decodes and displays at normal rate</td>
<td>000d</td>
<td>11-26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Scan()</td>
<td>Decodes and displays next single I-picture</td>
<td>000a</td>
<td>11-28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SingleStep()</td>
<td>Decodes and stores next single picture</td>
<td>000b</td>
<td>11-45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SlowMotion()</td>
<td>Decodes and displays at slower rate</td>
<td>0109</td>
<td>11-46</td>
</tr>
<tr>
<td>Control</td>
<td>High(^2)</td>
<td>No(^3)</td>
<td>AccessSCR()</td>
<td>Reads or writes internal SCR counter</td>
<td>8312</td>
<td>11-12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FlushBitstream()</td>
<td>Discards contents of bitstream buffer</td>
<td>8102</td>
<td>11-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>InquireBufferFullness()</td>
<td>Measures data in bitstream buffer</td>
<td>8001</td>
<td>11-19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NewPacket()</td>
<td>Manages bitstream data</td>
<td>0408</td>
<td>11-20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reset()</td>
<td>Reinitializes CL450 and its microcode</td>
<td>8000</td>
<td>11-27</td>
</tr>
</tbody>
</table>

1. Allows display of a still image with double vertical resolution, but requires use of a specific subset of the MPEG syntax.
2. Except for NewPacket().
3. Except for Reset().

Note: If a macro command is issued and HMEM[0] contains a value other than one of the values listed in the Function Code column above, indeterminate behavior will occur.
9.1.2 Bitstream Transfer Process

MPEG coded bitstream data is transferred by the host system to the CL450 for decoding. The CL450 accepts MPEG elementary video streams only. System streams and elementary audio streams must be handled by the host prior to data arrival at the CL450.

Data can be transferred from the host to the CL450 using either (1) direct host writes to the input of the CL450’s CMEM or (2) DMA. In both cases, the mode is selected by the host writing to the CMEM_dmactrl register, and the behavior of the CL450 (macro commands, interrupts, etc.) is the same for either method chosen.

Once inside the CL450, data is buffered in CMEM and then transferred as a burst to the 47,104-byte bitstream buffer located in the local DRAM. (The bitstream buffer is called the rate buffer in the MPEG standard.)

If the bitstream buffer is full, the CL450 does not transfer data from CMEM, so the host must ensure that the CMEM does not overflow. For direct host transfers (non-DMA), the host processor must monitor the fullness of CMEM. For DMA transfers, the DMA controller automatically holds off transfers when CMEM is full.

In addition to sending video information to the CL450, the host processor also extracts system clock references (SCRs) and presentation time stamps (PTSs) from the system layer of the MPEG bitstream. It uses these to provide the CL450 with the information required to synchronize the video and audio decoding as described in Section 9.2.

DMA Operation

When DMA is used, the CL450 acts as a DMA slave and, once correctly enabled, asserts the DMAREQ pin any time it can accept another 16-bit DMA write. In this mode, the host does not have to monitor the fullness of CMEM or the bitstream buffer portion of DRAM unless the host needs this information to configure its DMA controller. The hardware protocol and timing used for DMA transfers is given in Chapter 4.

Programmed Access

During programmed access, the host writes 16-bit words of coded data into CMEM by addressing any location in a 256K-word area of the
Process Configurations

CL450’s address space. These write operations put coded data into CMEM, a temporary holding FIFO with sixteen 16-bit locations between the host and the DRAM-resident bitstream buffer.

When directly writing CMEM, the host must not only configure CMEM_dmactrl correctly during initialization but also continually poll CMEM to prevent overflow, since a high enough instantaneous transfer rate can cause CMEM to fill up even if the bitstream buffer is not full.

Note: There is no data rate for which the host can avoid polling CMEM fullness; if the host does not ensure that CMEM can accept data before the data is written, CMEM overflow and data corruption will result. Because of this, the host must poll the fullness of CMEM while performing a data transfer.

Pseudocode for a typical transfer loop is shown in Figure 9-3. Note that only the coded data transfer operation is shown. Other non-transfer operations (not shown) need to be performed for most applications.

```c
#define BLOCK < size of burst> /* typically 3/4 empty */

void SendDataToCL450(void)
{
    int c;

    initialize CMEM_dmactrl; /* see section 3.1 for the
                              * proper timing of this write
                              * with respect to other CL450
                              * initialization */

    while (still more data to transfer){
        get CMEM fullness; /* see below */
        if (room in CMEM for BLOCK words)
            for (c=0; c < BLOCK; c++)
                write word of data to CMEM;
    }
}
```

Figure 9-3 Programmed Transfer of Coded Data to the CL450
The fullness of CMEM may be determined by one of the following two methods:

- **Using the CFLEVEL pin**: This pin can be programmed by the CMEM_dmactrl register to go active at any one of four different CMEM fullness levels. This pin can either be polled by the host (if it is accessible through a host I/O port) or wired directly to a device which is providing data to the CL450 instead of the host processor. The “entirely empty” selection in CMEM_dmactrl (bit 1) should not be used because of the existence of operating modes in which CMEM never becomes entirely empty.

- **Performing a register read of CMEM_dmactrl**: Note that while CMEM_status contains information similar to CMEM_dmactrl, CMEM_status is an internal register and should therefore not be used to determine CMEM fullness while the microapplication is running.

### 9.1.3 Decode Process

The decode process is the process by which the CL450 decompresses the input bitstream using the MPEG decoding algorithm and places the decompressed frames in the picture buffers in the CL450’s local DRAM.

The decoding process reconstructs spatial frequency coefficients from the variable-length Huffman codes, dequantizes them, and applies the inverse discrete-cosine transform. It uses the motion vectors in the bitstream to select a predictor from the previously decompressed frames stored in the local DRAM.

The decoding process pauses if the bitstream buffer is empty or if no space is available for writing the decoded frame. Such a condition could occur if the display of the previous frame was not completed before the decoding process was ready to begin writing to its frame buffer.

### 9.1.4 Display Process

The display process in the CL450’s microapplication handles the transfer of decoded pictures from the local DRAM picture buffers through the video bus from where data is passed to the display monitor. The CL450 supports clipping and positioning of the decoded video data anywhere on the display.
Images coded at 24, 25 or 30 Hz can be displayed at field rates of 50 or 60 Hz.

I- and P-pictures are fully buffered before being displayed. B-pictures are written to and displayed from the frame buffer as they are being decoded, provided the presentation time stamp requirements are met as described in Chapter 13, Audio/Video Synchronization.

The CL450 can synchronize its decoding operations with an external time reference such as an audio decoder. Synchronization is performed using the MPEG constructs of system clock references (SCRs) and presentation time stamps (PTSs)—both measured in terms of a 90-kHz time reference. When performing synchronization, the CL450 compares the PTS value associated with each picture—either supplied by the host via the NewPacket() macro command or synthesized by the microapplication—with the current value of the CL450’s internal SCR counter.

If the presentation time stamp differs from the internal clock by more than the allowed jitter tolerance of 3000 clock periods (1/30th of a second), the display rate is adjusted as follows:

- If the presentation time stamp for a B-picture is less than the SCR by more than 3000, the picture is skipped.
- If the presentation time stamp for a picture is greater than the SCR by more than 3000, the picture is repeatedly displayed.

The CL450 provides an interrupt output pin (INT) to the host processor which allows the microapplication to alert the host when certain events occur. The host determines the hardware interrupt protocol by writing the HOST_control register. The host then selects the interrupt events of which it wishes to be informed by using the SetInterruptMask() macro command (see page 11-36) to assign mask bits as shown in Figure 9-4.

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Res</td>
<td>SCN</td>
<td>RDY</td>
<td>SEQ-D</td>
<td>UND</td>
<td>Res</td>
<td>PIC-D</td>
<td>END-D</td>
<td>END-V</td>
<td>SEQ-V</td>
<td>GOP</td>
<td>PIC-V</td>
<td>ERR</td>
</tr>
</tbody>
</table>

Figure 9-4  Mask Bit Allocation
There are 11 logical interrupts which can be produced by the CL450 microapplication. Each belongs to one of three categories (based on when they are reported to the host) and is referenced by name and the event which causes it in Table 9-2. The interrupts and their categories are described in Chapter 12, Interrupts.

Table 9-2  
CL450 Interrupt Summary

<table>
<thead>
<tr>
<th>Category</th>
<th>Interrupt Name</th>
<th>Event Name</th>
<th>Event Description</th>
<th>Mask Bit</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode-time</td>
<td>END-D</td>
<td>sequence_end_code found</td>
<td>found</td>
<td>5</td>
<td>12-15</td>
</tr>
<tr>
<td></td>
<td>ERR</td>
<td>Bitstream data error</td>
<td></td>
<td>0</td>
<td>12-17</td>
</tr>
<tr>
<td></td>
<td>PIC-D</td>
<td>New picture decoded</td>
<td></td>
<td>6</td>
<td>12-20</td>
</tr>
<tr>
<td></td>
<td>SEQ-D</td>
<td>sequence_header_code found</td>
<td>found</td>
<td>9</td>
<td>12-25</td>
</tr>
<tr>
<td></td>
<td>SCN</td>
<td>Picture decode complete in Scan()</td>
<td></td>
<td>11</td>
<td>12-24</td>
</tr>
<tr>
<td></td>
<td>UND</td>
<td>Bitstream buffer underflow error</td>
<td></td>
<td>8</td>
<td>12-27</td>
</tr>
<tr>
<td>VSYNC</td>
<td>END-V</td>
<td>Last picture display before sequence_end_code</td>
<td></td>
<td>4</td>
<td>12-16</td>
</tr>
<tr>
<td></td>
<td>GOP</td>
<td>First I-picture display after group_start_code</td>
<td></td>
<td>2</td>
<td>12-19</td>
</tr>
<tr>
<td></td>
<td>PIC-V</td>
<td>New picture display</td>
<td></td>
<td>1</td>
<td>12-21</td>
</tr>
<tr>
<td></td>
<td>SEQ-V</td>
<td>First I-picture display after sequence_header_code</td>
<td></td>
<td>3</td>
<td>12-26</td>
</tr>
<tr>
<td>Display-time</td>
<td>RDY</td>
<td>Ready for data</td>
<td></td>
<td>10</td>
<td>12-22</td>
</tr>
</tbody>
</table>

Note: The CL450’s internal CPU can also receive interrupts from on-chip sources. The host has no control over these interrupts and they should not be confused with the host interrupts generated by the CL450.
The host processor executes an initialization sequence for the CL450. During the initialization sequence, the host:

1. Loads the microapplication into the CL450 DRAM.
2. Loads bootstrap code into the CL450’s instruction memory (IMEM).
3. Sets the program counter in the CPU_pc register to point to the starting address of the bootstrap code.
4. Enables the CL450’s CPU by setting the run enable bit, $CEn$ (CPU_control[0]), to 1.

Once enabled, the CL450’s CPU bootstraps itself by accessing internal instruction memory (IMEM) and executing code that was written by the host processor; it acts essentially as a microcontroller with on-chip peripherals used to aid MPEG decoding.
Note: Enabling the CPU is different than sending the Play() command. The enable bit, CEn, turns on the CL450’s internal processor, while the Play() command is a specific macro command that runs a portion of code in the DRAM to read and decompress an MPEG bitstream. The Play() command cannot be executed before the microapplication is running; the microapplication, once enabled, will not decode bitstreams until the Play() command is executed. Other macro commands include those that change the border color and display size, and all are described in Chapter 11.

When the CL450’s microapplication is first started, the contents of several registers are written with default parameters. Similarly, when the Reset() macro command is issued, a subset of these registers is re-initialized. Because of these automatic defaults, some registers cannot be effectively written before microapplication initialization.

10.1.1 Default Settings
Table 10-1 summarizes the registers that are modified when microapplication is loaded and when the Reset() command is executed (uninitialized registers are not included). Default values are given in binary, with “1” and “0” representing bits which are set and cleared, “-” representing bits which are unmodified, “?” representing bits which are given indeterminate but acceptable values, and “R” indicating RESERVED bits. The space between each group of four bits is for ease of reading only—these groupings do not indicate register fields.

Note: If the host performs read-modify-write operations on registers in Table 10-1, the host must ensure that RESERVED bits are written with the values specified in Chapter 8, Registers, regardless of the value read for these bits.
Table 10-1  Automatic Register Defaults

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Use</th>
<th>Initialization</th>
<th>Reset()</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST_newcmd</td>
<td>Normal</td>
<td>RRRR RRRR RRRR RRRR</td>
<td>RRRR RRRR RRRR RRRR</td>
</tr>
<tr>
<td>CMEM_control</td>
<td>Normal</td>
<td>RRRR RRRR R--R RR10</td>
<td>RRRR RRRR R--R RR10</td>
</tr>
<tr>
<td>HOST_control</td>
<td>Normal</td>
<td>00RR RRRR 1RRR RRR1</td>
<td>00RR RRRR 1RRR RRR1</td>
</tr>
<tr>
<td>VID_control</td>
<td>Normal</td>
<td>RRRR RRRR RR?? ???R</td>
<td>No effect</td>
</tr>
<tr>
<td>VID_sela</td>
<td>Normal</td>
<td>1010 1000 1100 0110</td>
<td>No effect</td>
</tr>
<tr>
<td>VID_selb</td>
<td>Normal</td>
<td>0100 1001 0110 0111</td>
<td>No effect</td>
</tr>
<tr>
<td>VID_selmode</td>
<td>Internal</td>
<td>RRRR RRRR RRRR RRR1</td>
<td>No effect</td>
</tr>
<tr>
<td>VID_selective</td>
<td>Internal</td>
<td>R101 0111 ?? ??</td>
<td>No effect</td>
</tr>
<tr>
<td>VID_selbor</td>
<td>Internal</td>
<td>RRRR RRR0 0011 1100</td>
<td>No effect</td>
</tr>
<tr>
<td>VID_selIR</td>
<td>Internal</td>
<td>RRRR RRRR 0000 0000</td>
<td>No effect</td>
</tr>
<tr>
<td>VID_selGB</td>
<td>Internal</td>
<td>0000 0000 0000 0000</td>
<td>No effect</td>
</tr>
<tr>
<td>HOST_scr2</td>
<td>Internal</td>
<td>RR1 1101 1110 0000</td>
<td>No effect</td>
</tr>
<tr>
<td>HOST_scr1</td>
<td>Internal</td>
<td>R000 0000 0000 0000</td>
<td>No effect</td>
</tr>
<tr>
<td>HOST_scr0</td>
<td>Internal</td>
<td>R000 0000 0000 0000</td>
<td>No effect</td>
</tr>
</tbody>
</table>

1. "R" indicates RESERVED bits, "-" indicates bits which are unmodified by the microapplication, "??" indicates bits written with variable values, and "1" and "0" indicate fixed values written by the microapplication.

In addition to initialization of the above registers, the following operations are performed on initialization or Reset():

- All 16 locations of HMEM are cleared to 0.
- CMEM is reset via the CRst bit of CMEM_control.
- MPEG default parameters are restored.

10.1.2 Loading Sequence

In general, not all CL450 registers will be used by the host in all applications. Some of the registers can be accessed by the host at any time, while some may only be accessed when microapplication is not executing. In addition, there are some registers which the microapplication automatically writes with default values upon microapplication startup (shown in Table 10-1), negating any writes by the host which occurred previously. Because of these factors, the sequence with which registers are written is critical to the proper operation of the CL450.
Registers

Specifically, the following four registers are the only registers that may be accessed at any time by the CPU without any interference from the microapplication:

- CMEM_dmactrl
- HOST_intvecw
- HOST_intvecr
- DRAM_refcnt

In general, the host can access most Normal registers at any time (although there may be undesirable side effects if an Internal register is accessed). However, there are three classes of exceptions:

- **Indirect video registers** (see Table 8-4 on page 8-6): These registers can only be accessed by the host after a selection has been made using VID_control. In addition, the CL450 microapplication also uses VID_control to access these registers. Therefore, host access (read or write) to an indirect video register is prohibited while microapplication is executing, and any access attempts will produce indeterminate results.

- **Registers for accessing HMEM** (HOST_raddr and HOST_rdata): These registers are reserved for the exclusive use of the host. Because the CL450 microapplication does not use these registers, the host can use them at any time, making HMEM unlike the other on-chip memories. However, while HMEM may be read at any time, HMEM writes performed while the microapplication is executing must occur in accordance with the protocols given in Section 11.2, Processing Macro Commands; and Section 12.2, Interrupt Status Location.

- **Registers for accessing other on-chip memories** (IMEM, TMEM, etc.): These registers are shared by the CL450 and by the host. For this reason, indeterminate results will occur if on-chip memories other than HMEM are accessed while the microapplication is executing.

The following text outlines the sequence in which registers should be written by the host relative to CL450 start-up if they are to be written at all:

1. The VID_selaux register must be written prior to starting the execution of the microapplication because it will be inaccessible afterwards.
2. The CMEM_control register may be written prior to starting the execution of the microapplication, although its value will be modified by microapplication initialization.

3. The microapplication should be loaded and started by following the procedure given in Section 10.2.2. The following registers should be used only as part of the microapplication loading process:
   - CPU_control
   - CPU_pc
   - CPU_iaddr, CPU_imem

4. Once the microapplication has started execution, the following registers may be used by the host:
   - HOST_newcmd, as specified by Section 11.1, Writing Macro Commands
   - CMEM_control
   - HOST_raddr, HOST_rdata
   - HOST_control, as specified by Section 12.3, Handshaking Protocol

5. Finally, several macro commands can be used to implicitly load values into registers. These commands and their corresponding registers are listed in Table 10-2 below.

<table>
<thead>
<tr>
<th>Macro Command Name(s)</th>
<th>Register(s) Loaded</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetColorMode()</td>
<td>VID_selmode</td>
</tr>
<tr>
<td>SetBorder(), SetWindow()</td>
<td>VID_selactive, VID_selbor¹</td>
</tr>
<tr>
<td>SetBorder()</td>
<td>VID_selR, VID_selGB</td>
</tr>
</tbody>
</table>

¹Note that these registers may also be modified in response to new MPEG sequence-layer information. See Section 10.2.1, Default Settings; Section 14.2.2, Writing DRAM-resident Variables; and the descriptions of the SetBorder() and SetWindow() macro commands contained in Chapter 11.

The CL450 microapplication is logically contained in a series of "segments," each of which is a block of numeric constants (constant data or microapplication) that the host must load into either the internal IMEM (instruction memory) or the external DRAM.
All the segments of the CL450 microapplication are distributed in a single microcode executable file. (The format of this file is defined in Appendix B.) In addition to microapplication segments, the executable file also contains an information header which includes the initial program counter value for the CL450’s CPU.

### 10.2.1 Default Settings

All of the programmable parameters used by the CL450 microapplication have defaults. This section lists these defaults, their functional relationship and the event(s) which causes the CL450 to change from the default setting or return to it.

Table 10-3 lists the macro commands which can be used to change settings within the CL450 and the default values for these settings. Note that these defaults are set on microapplication initialization only and are not restored by execution of the Reset() macro command.

**Table 10-3 Macro Command Defaults**

<table>
<thead>
<tr>
<th>Macro Command Name</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetThreshold()</td>
<td>threshold = 4096 bytes</td>
</tr>
<tr>
<td>SetInterruptMask()</td>
<td>mask = 0; no interrupts enabled</td>
</tr>
<tr>
<td>SetVideoFormat()</td>
<td>format = 4 (NTSC):</td>
</tr>
<tr>
<td></td>
<td>□ 60Hz nominal VSYNC frequency</td>
</tr>
<tr>
<td></td>
<td>□ 30Hz nominal picture rate</td>
</tr>
<tr>
<td></td>
<td>□ 352 pixel (SIF) horizontal active display</td>
</tr>
<tr>
<td></td>
<td>□ 240 pixel (SIF) vertical active display</td>
</tr>
<tr>
<td>SetWindow()</td>
<td>xOffset = yOffset = 0</td>
</tr>
<tr>
<td></td>
<td>width &amp; height from sequence-layer parameters</td>
</tr>
<tr>
<td></td>
<td>(horizontal_size and vertical_size)</td>
</tr>
<tr>
<td>SetBorder()</td>
<td>leftBorder = topBorder = auto-center based on sequence layer parameters</td>
</tr>
<tr>
<td></td>
<td>border color is “sub-black” (R=G=B=Y=0)</td>
</tr>
<tr>
<td>SetBlank()</td>
<td>state = 0; video display not blanked</td>
</tr>
<tr>
<td>SetColorMode()</td>
<td>mode = 1 (RGB output)</td>
</tr>
</tbody>
</table>
In addition to the default parameters which can be specified by macro commands, the CL450 contains default values for MPEG bitstream parameters as shown in Table 10-4 and Table 10-5. The decoder is restored to these defaults following microapplication initialization, the execution of the Reset() command, and detection of an MPEG sequence_end_code in the bitstream. These default values are used for decoding unless replaced by the host (see Section 14.2.2, Writing DRAM-resident Variables) or decoded from the bitstream.

In general, the default values are not used for decoding if the decode operation starts at the beginning of a legal MPEG bitstream. However, when performing random-access within a bitstream, the defaults are used if no other values are provided. Note that only parameters which are required for decoding have default values.

Table 10-4 MPEG Decoding Defaults, Sequence Layer

<table>
<thead>
<tr>
<th>Bitstream Parameter</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal_size</td>
<td>352 pixels</td>
</tr>
<tr>
<td>vertical_size</td>
<td>240 pixels</td>
</tr>
<tr>
<td>picture_rate</td>
<td>4 (30Hz)</td>
</tr>
<tr>
<td>load_intra_quantizer_matrix</td>
<td>0 (use default from MPEG standard)</td>
</tr>
<tr>
<td>load_non_intra_quantizer_matrix</td>
<td>0 (use default from MPEG standard)</td>
</tr>
</tbody>
</table>

Table 10-5 MPEG Decoding Defaults, GOP Layer

<table>
<thead>
<tr>
<th>Bitstream Parameter</th>
<th>Default Setting¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>closed_gop</td>
<td>0</td>
</tr>
<tr>
<td>broken_link</td>
<td>1</td>
</tr>
</tbody>
</table>

¹ The CL450 makes no assumptions about whether the current GOP follows the last GOP and waits for an I-picture before beginning to decode.

Table 10-6 summarizes the bits in the CMEM_control, HOST_control and CMEM_dmactrl registers which control communication between the host and the CL450, plus their reset values, if any. For a complete list of all register bits which have default values, see Table 10-1.

Note: Some of the register bits which control interaction with the host have no defaults.
Table 10-6 Host/CL450 Interface Defaults

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Bit(s)</th>
<th>Bit Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMEM_control</td>
<td>5</td>
<td>BS</td>
<td>unknown</td>
<td>Determines whether or not bytes are swapped when 16-bit words enter CMEM.</td>
</tr>
<tr>
<td></td>
<td>4:1</td>
<td>1QE to 4QE</td>
<td>unknown</td>
<td>These bits determine when the CL450 will assert the CFLEVEL pin.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>DE</td>
<td>unknown</td>
<td>This bit (DMA Enable) is used to select whether bitstream data is provided to CMEM using DMA or programmed access (see Section 9.3, Bitstream Transfer Process). Note that this bit is cleared automatically if the DONE pin is asserted.</td>
</tr>
<tr>
<td>HOST_control</td>
<td>15</td>
<td>AIC</td>
<td>unknown</td>
<td>Auto Interrupt Clear; determines whether bit 7 (Int) is set (INT pin inactive) automatically when the INTACK pin is asserted.</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>VIE</td>
<td>0</td>
<td>Vectored Interrupt Enabled; determines if the current interrupt vector (written by the host through the HOST_intvecw register) is driven onto the data bus when the INTACK pin is asserted.</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Int</td>
<td>0</td>
<td>INT pin inactive.</td>
</tr>
</tbody>
</table>

1. The host should initialize these bits following hardware reset; they are unmodified by microapplication initialization or execution of the ResetO macro command.

Finally, several of the DRAM-resident variables also have default values (see Chapter 14). Those variables which have defined defaults are listed in Table 10-7 below. DRAM-resident variables not listed in the table do not have defaults. The DRAM-resident variables are restored to their defaults on both microapplication initialization and Reset().

Table 10-7 DRAM-Resident Variable Defaults

<table>
<thead>
<tr>
<th>Group</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence group</td>
<td>SEQ_SEM</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SEQ_CONTROL</td>
<td>0</td>
</tr>
<tr>
<td>picture group</td>
<td>PIC_SEM</td>
<td>0</td>
</tr>
</tbody>
</table>
10.2.2 Loading Sequence

The microapplication loading sequence is as follows:

1. Typically, microapplication loading starts with the host parsing the microcode executable file and writing the appropriate segments into DRAM.

2. Some segments are also written to IMEM using the CPU_iaddr and CPU_imem registers.

   Note: The host must ensure that segment values are written to IMEM in an order such that the last value written goes to an IMEM address other than the initial program counter value. The segments in the executable file are arranged such that, if the host writes IMEM locations in the same order the data appears in the file, this restriction will always be met.

3. Once both DRAM and IMEM have been completely loaded, the initial program counter value should be written to the CPU_pc register.

   Note: When the initial value is written to CPU_pc, the IE bit (bit 9) must be 0. This bit is the internal master interrupt enable for the CL450’s CPU and is not related to interrupts from the CL450 to the host.

4. After the CPU_pc register has been initialized, the host should write the value 1 to the CPU_control register to enable the CL450’s CPU.

After the on-chip CPU is enabled, the host must wait until the internal initialization is complete before issuing the first macro command.

One way to determine if initialization is complete is to write a non-zero value to HMEM location 15 prior to enabling the CPU. Then the contents of this HMEM location can be polled. Once this location becomes 0, the CL450 has completed its internal initialization and is ready to accept macro commands. This method can also be used to determine when the execution of the Reset() macro command is complete (see Reset() on page 11-28).
10.2.3 Halting

Microapplication execution may be halted from the host either by:

- Resetting the CL450
- Writing a 0 to the CPU_control register

These two alternatives will produce different results on the display depending on whether or not the CL450 was displaying a picture at the time the CPU was halted.

If the CL450 is reset, then the last-specified border color will be displayed continually until the microapplication is reloaded and bitstream decoding is resumed.

If the host writes a 0 to CPU_control, then the display depends on what the CL450 was doing when halted. If the output window was blanked or the CL450 was in vertical border time, the visual result will be the same as if the CL450 was reset. If CPU_control was cleared while the CL450 was displaying an active scan line, then that scan line will be repeatedly output (regardless of vertical timing) until the CL450 is reset or the microapplication re-loaded.

*Note: Once halted, it is not possible to resume microapplication execution without performing the entire microapplication loading sequence described above.*

When the microapplication is loaded, there are four DRAM locations (see Table A-6 on page A-3) which contain information describing the microapplication version. This information is duplicated in the microapplication executable file header but is loaded into DRAM to ensure that the microapplication version present in a running system can be determined.
The host software and the CL450’s microapplication use 18 macro commands as their primary method of communication. Macro commands are function codes and parameter values written into the dual-ported HMEM by the host. Each command has a separate function code and may have 0 to 4 parameters.

Once the function code and parameters are written, the microapplication acts on them according to their priority:

- **High-priority**: These commands start execution as soon as the CL450’s microapplication detects their presence and complete execution before the HOST_newcmd semaphore is cleared.
- **Low-priority**: These commands are stored by the CL450 in the Command FIFO (see Section 11.4, Command Latency) and executed by the CL450 in the order that they were received (see Section 11.4, Command Latency).
Macro commands are written into HMEM by the host. HMEM is a 16-word-by-16-bit memory within the CL450. It is accessed indirectly by writing the desired HMEM address (0 through 0xf) to HOST_raddr and then reading or writing the HOST_rdata register.

When HOST_rdata is read, the contents of the currently selected HMEM location are returned to the host. When HOST_rdata is written by the host, the currently selected HMEM location is written with the same data.

*Note: Indeterminate behavior occurs if the host reads HOST_rdata more than once without re-writing HOST_raddr.*

HMEM locations are allocated for (1) macro command use and (2) CL450 status information. Table 11-1 indicates the standard usage for each HMEM location.

<table>
<thead>
<tr>
<th>HOST_raddr (0x)</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Macro Command, Function Code</td>
</tr>
<tr>
<td>1</td>
<td>Macro Command, Argument 1</td>
</tr>
<tr>
<td>2</td>
<td>Macro Command, Argument 2</td>
</tr>
<tr>
<td>3</td>
<td>Macro Command, Argument 3</td>
</tr>
<tr>
<td>4</td>
<td>Macro Command, Argument 4</td>
</tr>
<tr>
<td>5</td>
<td>RESERVED</td>
</tr>
<tr>
<td>6</td>
<td>RESERVED</td>
</tr>
<tr>
<td>7</td>
<td>RESERVED</td>
</tr>
<tr>
<td>8</td>
<td>RESERVED</td>
</tr>
<tr>
<td>9</td>
<td>RESERVED</td>
</tr>
<tr>
<td>a</td>
<td>Interrupt Status</td>
</tr>
<tr>
<td>b</td>
<td>Buffer Fullness Status</td>
</tr>
<tr>
<td>c</td>
<td>RESERVED</td>
</tr>
<tr>
<td>d</td>
<td>RESERVED</td>
</tr>
<tr>
<td>e</td>
<td>RESERVED</td>
</tr>
<tr>
<td>f</td>
<td>RESERVED</td>
</tr>
</tbody>
</table>

The host should not use the reserved HMEM locations for scratch storage because the CL450 may write to them at any time. Also, the host
should generally avoid writing to reserved locations within HMEM to facilitate upgrading to future microapplication versions.

Because HMEM is dual-ported and contains quantities other than the current macro command (see InquireBufferFullness() on page 11-19 and Section 12.2, Interrupt Handshaking), it uses a semaphore to assure that only one of either the host or the CL450’s CPU accesses the HMEM locations used for macro commands at any one time. This semaphore is located in HOST_newcmd[0].

Any time the semaphore is 0, the host may read and write the macro command locations of HMEM (HMEM addresses 0 through 4). Once an entire new macro command has been written, the host should write a 1 to the semaphore. Once this has occurred, the host may not write to HMEM[4-0] or HOST_newcmd again until the CL450 has cleared the semaphore (i.e., HOST_newcmd[0] = 0).

To ensure that a deadlock does not occur if the CL450 fails to acknowledge a command, the host may use a time-out counter when polling HOST_newcmd.

The mechanism for issuing a macro command is shown in the pseudo-code example of Figure 11-1.

```c
ERROR_RETURN SendNewMacroCommand(arguments) {
    write function code to HMEM[0];
    write arguments (if any) to HMEM[4-1], as appropriate;
    HOST_newcmd= 1; /* tell microcode new macro
                    * command is ready */
    while ( (HOST_newcmd&1 == 1) &&
            (haven’t timed-out yet) ) { /* wait for command to be
                                         * accepted */
        ;
    }
    if (timed-out)
        return ERROR;
    else
        return NO_ERROR;
}
```

Figure 11-1   Issuing a Macro Command to the CL450
In Figure 11-1, note that the time-out processing could also be moved to the beginning of the function without violating the semaphore protocol: the host must not write to HMEM[4-0] when HOST_newcmd[0] is 1. In general, the host software will have better performance if the polling loop is moved up because the microapplication’s acceptance of the new macro command will occur in parallel with the host’s next processing task.

The function shown provides advantages in the diagnostic process because it returns an error immediately upon determining that the CL450 has stopped responding to commands. If the CL450 fails to accept a macro command, it indicates an incorrect microapplication loading procedure or an unrecoverable failure during operation. In either case, the microapplication must be reloaded.

The CL450 operates in one of eight possible internal command processing states which affect how future commands are interpreted and how the decode and display processes operate.

Several command states correspond to a macro command which can cause a command state transition. The macro commands which can explicitly change the command state are the Reset() command and all of the Play-type commands. In addition to macro commands, several internal operations can also cause a command state transition.

The command states and their relationships are shown in Figure 11-2. Within this figure, the eight command states are inscribed within ovals. The solid arrows indicate state transitions which occur due to the completion of internal processing, while shaded arrows indicate transitions caused by the execution of a macro command.

The behavior of the IDLE and PLAY-SETUP states are described next, while the remainder of the command states are described with their corresponding macro commands listed in the second half of this chapter.
Execution of Play-type macro command...

Start

Completion of initialization

IDLE

Execution of any Play-type macro command

PLAY-SETUP

Reset() (from any state)

CL450 unable to accept bitstream data

CL450 ready for bitstream data

Note: Play() transitions are based on the command that caused the departure from the IDLE state.

State transition caused by a macro command

State transition caused by an external event

Execution of picture decode operation

First picture decode after PLAY-SETUP complete; transition to state corresponding to last Play-type command

Figure 11-2 Command State Transition Diagram
11.2.1 IDLE State
The CL450 microapplication is in the IDLE state immediately following initialization or execution of the Reset() command and stays in the IDLE state until a Play-type macro command is executed. While in the IDLE state, the CL450’s output window is blanked (the screen is filled with the current border color) and the CL450 does not accept bitstream data. Because of this, Display-time interrupts are not produced (see Section 12.1.1, Display-time Interrupt), and the host must not send coded data to the CL450. However, the host may issue any macro command while the CL450 is in the IDLE state, although some arguments to the Set-type commands will not have an immediate effect.

11.2.2 PLAY-SETUP State
The PLAY-SETUP state is a transition state between IDLE and the remaining states. The CL450 microapplication spends only a short time in this state, during which it performs internal housekeeping to set up for reception of bitstream information.

Once the operations associated with the PLAY-SETUP state are complete, the microapplication immediately and always transitions to the PLAY command state, regardless of which Play-type macro command caused the transition from IDLE. Because of this, the first decoded picture after the microapplication is initialized will always be decoded in the PLAY state. Once this picture is decoded, the microapplication effectively “re-executes” the Play-type command and then transitions to the state corresponding to the macro command which caused the original transition from IDLE to PLAY-SETUP. This occurs even when the first Play-type command is Pause(), SingleStep(), Scan(), Display-Still(), etc. In addition to causing an “extra” picture to be decoded before entering the desired state, this also causes the execution of subsequent low-priority macro commands to be deferred one extra picture decoding time. Because all Play-type commands executed in the IDLE state are interpreted in the same way that a two-command sequence beginning with the Play() command would be, two picture decodes must complete (one for the implicit Play() command and one for the actual command issued) before other commands from the Command FIFO are read and executed.
Once the microapplication has left the PLAY-SETUP state, the CL450 begins decoding incoming bitstream data as it is received. The output window is subsequently unblanked once enough of the bitstream has been decoded for display to begin.

Note: Unblanking of the output window only occurs if the display has not been explicitly blanked by the host (see the SetBlank() macro command). If the display has been blanked by the host, then the output window is unblanked only when another SetBlank() is executed with the appropriate argument.

The CL450 microapplication maintains a Command FIFO in DRAM. The Command FIFO occupies 630 words of DRAM and is used to store all low-priority (HMEM[0][15] == 0) macro commands issued by the host. Each macro command occupies five words of storage (loaded from HMEM[4-0]), so that the Command FIFO can accommodate up to 126 entries.

Typically, the number of pending commands in the Command FIFO is significantly less than 126. However, NewPacket() commands remain in the Command FIFO until sometime after the corresponding portion of the bitstream has been decoded, regardless of whether the other macro commands ahead of and behind them have been executed, which increases the number of entries required.

Note: The CL450 microapplication cannot prevent the Command FIFO from overflowing. The host must ensure that the maximum number of possible entries in the Command FIFO is not exceeded.

The Command FIFO is maintained as a circular buffer with one write pointer and two read pointers (“command read” and “packet read”) as shown in Figure 11-3. Each time a new low-priority macro command is issued by the host, it is written into the circular buffer and the write pointer is incremented (wrapping if necessary). Each time a macro command other than NewPacket() is extracted and executed, the “command read pointer” is incremented. In addition, if a NewPacket() command is encountered when looking for another macro command, the command read pointer is incremented past the NewPacket(), effectively skipping it.
Figure 11-3  CL450 Command FIFO Example
Each time a new picture_start_code is detected during decode, the microapplication searches through the Command FIFO using the "packet read" pointer and looks for the NewPacket() command corresponding to the packet which contains the start code of the picture which is about to be decoded. While this is done, the packet read pointer is incremented to skip macro commands other than NewPacket().

All low-priority macro commands are placed into the Command FIFO before the CL450 clears HOST_newcmd[0]. The delay before macro command execution is determined by the current command processing state. The microapplication operates according to the following rules:

- NewPacket() commands have no impact on when other low-priority macro commands are executed.
- When in the IDLE or PAUSE command states (see Section 11.4, Command Latency, and the Pause() command on page 11-25), macro commands are executed as they are received unless the host issues commands faster than they can be processed, in which case they accumulate in the Command FIFO. Command execution continues uninterrupted until a Play-type macro command is executed, which changes the command state from IDLE or PAUSE to PLAY-SETUP or a decoding state, respectively.
- When not in the IDLE or PAUSE command states, macro commands which have accumulated in the Command FIFO are executed each time the CL450 completes decoding a picture. Note that this rate is dependent on the output frame rate, the picture_rate sequence parameter, audio/video synchronization, and the last speed parameter if in the SLOW state. If a Play-type command is not encountered, the microapplication continues to execute commands until all commands in the Command FIFO have been consumed or until it is time to begin decoding the next picture, whichever is later.

The host can issue more macro commands than the CL450 is able to process without falling behind in picture decoding. In general, the host:

- Must ensure that the Command FIFO does not overflow.
- May issue NewPacket() and any high-priority commands as frequently as desired.
- May issue one SetBorder() and one SetWindow() command per frame period (nominally every other VSYNC).

Other low-priority commands should be issued only when necessary.
11.5 Macro Command Groups

Note that CL450 macro commands are divided into three functional categories:

- Set-type
- Play-type
- Control

Each command group has distinct properties, which are described below, in Table 11-2, and in Section 11.4.

11.5.1 Set-type Commands
The CL450 has seven Set-type macro commands, all of which:

- Are low priority
- Never affect the command state
- May be issued regardless of the current command state
- Have no effect on the decoding process, except for SetVideoFormat(), which helps determine if video sequences are transcoded (see SetVideoFormat on page 11-40)

11.5.2 Play-type Commands
The CL450 has six Play-type macro commands, each of which causes the:

- Current command state to change
- Macro command processing to be suspended
- Microcode to transition into the PLAY-SETUP state and the CL450 to be configured to accept bitstream data (when issued in the IDLE state)

11.5.3 Control Commands
The CL450 has five Control macro commands (the least homogeneous of the three groups), each of which:

- Can be issued regardless of the current command state
- Is high priority, except for NewPacket()
- Has no effect on the command state, except for Reset()
### Table 11-2 Macro Command Summary

<table>
<thead>
<tr>
<th>Category</th>
<th>Priority</th>
<th>Effect on Name</th>
<th>Description</th>
<th>Function Code (0x)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set-type</td>
<td>Low</td>
<td>No</td>
<td>SetBlank() Blanks/unblanks output window</td>
<td>030f</td>
<td>11-29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetBorder() Sets output window location</td>
<td>0407</td>
<td>11-30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetColorMode() Enables/disables color-space converter</td>
<td>0111</td>
<td>11-34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetInterruptMask() Enables/disables interrupts to host</td>
<td>0104</td>
<td>11-36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetThreshold() Specifies bitstream buffer emptiness</td>
<td>0103</td>
<td>11-38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetVideoFormat() Configures output resolution and timing</td>
<td>0105</td>
<td>11-40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SetWindow() Sets output window size and contents</td>
<td>0406</td>
<td>11-42</td>
</tr>
<tr>
<td>Play-type</td>
<td>Low</td>
<td>Yes</td>
<td>DisplayStill() Decodes/displays single still picture</td>
<td>000c</td>
<td>11-14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pause() Keeps last picture on display</td>
<td>000e</td>
<td>11-25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Play() Decodes and displays pictures</td>
<td>000d</td>
<td>11-26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Scan() Decodes and displays next single P-picture</td>
<td>000a</td>
<td>11-28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SingleStep() Decodes and stores next P-picture</td>
<td>000b</td>
<td>11-45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SlowMotion() Decodes and displays at slower rate</td>
<td>0109</td>
<td>11-46</td>
</tr>
<tr>
<td>Control</td>
<td>High¹</td>
<td>No²</td>
<td>AccessSCR() Reads or writes internal SCR counter</td>
<td>8312</td>
<td>11-12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FlushBitstream() Discards contents of bitstream buffer</td>
<td>8102</td>
<td>11-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>InquireBufferFullness() Calculates fullness of bitstream buffer</td>
<td>8001</td>
<td>11-19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Newpacket() Manages bitstream data</td>
<td>0408</td>
<td>11-20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reset() Reinitializes CL450 and its microcode</td>
<td>8000</td>
<td>11-27</td>
</tr>
</tbody>
</table>

1. Except for NewPacket()
2. Except for Reset()

**Note:** If a macro command is issued and HMEM[0] contains a value other than one of the values listed in the Function Code column above, indeterminate behavior occurs.

All CL450 macro commands are listed alphabetically in the pages that follow.
The AccessSCR() macro command is issued by the host to write or read the current value of the CL450’s internal SCR (System Clock Reference) counter. The microapplication uses the contents of the SCR counter when performing audio/video synchronization. (For a full description of the CL450’s synchronization mechanism, see Chapter 13, Audio/video Synchronization.)

The R/W flag bit (HMEM[1][15]) determines whether the command performs a read or a write, as follows:

- **R/W flag bit = 1**: The microapplication reads the SCR counter and places the results in HMEM[1], HMEM[2] and HMEM[3]. When a read operation is performed, the read completes before the microapplication clears HOST_newcmd[0]. The current counter value is written into HMEM in the format given above. However, the value written to the R/W flag bit and the bits to which the host is required to write 0’s is random, and the host should mask the values read back from HMEM before using them.
- \(R/W\) flag = 0: When the host issues this command, the microapplication writes the new SCR value from HMEM into the CL450’s counter. Note that the contents of the SCR counter have no effect on the CL450’s behavior unless audio/video synchronization is being performed.

As the host processor extracts system clock references from the system layer of a bitstream, it updates the SCR counter by using an AccessSCR() macro command. The microapplication then updates the counter through the three HOST_scr registers shown in Figure 11-4: HOST_scr2, HOST_scr1, and HOST_scr0.

![AccessSCR() Arguments Block Diagram](image)
DisplayStill()

<table>
<thead>
<tr>
<th>Format</th>
<th>DisplayStill()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Low</td>
</tr>
<tr>
<td>Category</td>
<td>Play-type</td>
</tr>
</tbody>
</table>

Syntax:

- HMEM[0]  DisplayStill  0x000c
- HMEM[1]  0x0000
- HMEM[2]  0x0000
- HMEM[3]  0x0000
- HMEM[4]  0x0000

The DisplayStill() macro command is used to decode and display a high-vertical-resolution still picture. When this command is executed, the microapplication enters the STILL state, possibly passing through the PLAY-SETUP and PLAY states.

While in the STILL state, a single still picture from the incoming bitstream is decoded and posted for display.

*Note: Audio/video synchronization has no effect on the timing of picture decoding which occurs in the STILL state regardless of whether or not NewPacket() commands are being used.*

Once decoding is complete, the microapplication automatically changes to the PAUSE state (see the description of the Pause() command, page 11-25).

A still picture must be constructed as follows:
sequence_header() /* optional */
group_of_pictures() /* optional */
    picture() /* must be a single I-picture, supplies * image for upper field display */
    picture() /* must be a single I-picture, supplies * image for lower field display */
sequence_end_code /* optional */

**Figure 11-5  Still Picture Bitstream Format**

The header and picture information within a high-vertical-resolution still picture is decoded normally as two separate images, one containing all of the scan lines for the upper display field and the other containing the lower scan lines. These two images are then displayed during the even and odd display fields (as appropriate) while the microapplication remains in the PAUSE state.

Typically, both images in a still picture contain as many scan lines as a SIF image. Note that the $y_{Offset}$ argument to the SetWindow() macro command (see page 11-42) is interpreted as an offset both from the beginning of the even field to the first even scan line to be displayed and from the beginning of the odd field to the first odd scan line displayed.

Note that the microapplication clears the CL450's internal SCR counter to 0 each time a picture pair is decoded in the STILL state (see Section 13.3.2, Automatic SCR Modifications).
FlushBitstream()

Format: FlushBitstream(filter)
Priority: High
Category: Control
Syntax:

HMEM[0]  FlushBitstream  0x8102
HMEM[1]  filter
bits[15:3] = 0
bit[2] = restore default sequence parameters flag
bits[1:0] = filter parameter
HMEM[2]  0x0000
HMEM[3]  0x0000
HMEM[4]  0x0000

The FlushBitstream() macro command causes the CL450 to:

- Discard the current contents of the bitstream buffer (including any data currently in CMEM or the VLC Decoder) and any unprocessed NewPacket() commands in the Command FIFO (see page 11-9).
- Allow the host to specify the conditions under which picture decoding and display should resume when new bitstream information is received by the CL450.

In all cases, the CL450 continues to display the most-recently decoded picture after FlushBitstream() has been issued.

When using this command, the host must ensure that the bitstream data which is sent following the command has the correct relationship with subsequent NewPacket() commands, if there are any. In particular, the following facts must be considered:

- When FlushBitstream() is executed, all unprocessed NewPacket() commands are discarded, even if the CL450 has not yet received the bitstream data that corresponds to those packets.
- The FlushBitstream() command takes a finite amount of time to execute. If bitstream data transmission to the CL450 is not halted
prior to issuing this command, it is impossible to determine what data will be flushed and what data will be considered to have arrived after the command executed.

- Because FlushBitstream() is a high-priority command, all operations required to clear the bitstream buffer are completed before the CL450 clears HOST_newcmd[0].

- Because FlushBitstream() is a high-priority command, it may be executed while in the middle of a picture decode operation. If this occurs, the CL450 may spuriously report a bitstream underflow (see page 12-26 for a discussion of the buffer underflow interrupt, UND). Also, because the on-going picture-decode operation must be terminated when the bitstream is flushed, the video display may tear if the picture being decoded is a B-picture.

- When bitstream transmission resumes, the host must issue the NewPacket() command(s) which correspond to the beginning of the new bitstream data before the CL450 begins to receive it. This requirement is essentially the same as the requirement for beginning to send a bitstream to the CL450 following initialization or Reset().

After the bitstream and pending NewPacket() commands have been flushed and the CL450 has started to receive new bitstream data, the decoding process resumes. How the decoding process behaves is controlled by the filter argument of the FlushBitstream() command. The filter argument contains two fields:

- **Bit [2]** - This bit determines whether or not the sequence-layer parameters should be restored to their defaults prior to resuming decode. (See Table 10-4 for a definition of the sequence defaults.) Typically, this bit would be set to 1 (causing defaults to be restored) if decoding is to resume in a new video bitstream or in a different sequence within the current bitstream.

If FlushBitstream() is being used to skip a section within a bitstream, filter[2] is usually set to 0. Note however that the quantization matrices in the sequence layer, if present, may change each time a sequence header is incorporated into the bitstream.

When a bitstream is being accessed randomly, either following execution of FlushBitstream() or immediately following initialization or Reset(), the host must ensure that the correct quantization matrices are provided to the CL450 if the portion of the bitstream containing them is not decoded (see Section 14.2.2, Writing...
FlushBitstream() DRAM-Resident Variables). Note that the GOP-layer defaults (Table 2-4) are always restored when FlushBitstream() is issued.

- Bits[1:0] - This field is used to specify the point in the bitstream at which the CL450 should resume decoding. The field is encoded as shown in Table 11-3.

<table>
<thead>
<tr>
<th>filter[1:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00_2</td>
<td>Resume decode/display with next I-picture</td>
</tr>
<tr>
<td>01_2</td>
<td>Resume decode/display with first I-picture after the next GOP header</td>
</tr>
<tr>
<td>10_2</td>
<td>Resume decode/display only after the next sequence header</td>
</tr>
<tr>
<td>11_2</td>
<td>Resume decode/display at the earliest possible time</td>
</tr>
</tbody>
</table>

If a filter[1:0] value of 0 or 1 is selected, the host must provide the appropriate sequence-layer parameters (either by setting filter[2] or as described in Section 14.2.2, Writing DRAM-Resident Variables).

Alternately, if a filter[1:0] value of 2 is selected, the value of filter[2] is irrelevant because picture decoding will not resume until a sequence header is found within the bitstream, typically replacing the previous sequence parameters.
InquireBufferFullness()

**Format:** InquireBufferFullness()
**Priority:** High
**Category:** Control

**Syntax:**
- HMEM[0] = InquireBufferFullness
- HMEM[1] = 0x0000
- HMEM[2] = 0x0000
- HMEM[3] = 0x0000
- HMEM[4] = 0x0000

When the host issues the `InquireBufferFullness()` macro command, the CL450 calculates the amount of data currently stored in the bitstream buffer and updates the Buffer Fullness Status location, HMEM[0xb].

Because this is a high-priority command, the microapplication completes updating HMEM *before* clearing HOST_newcmd[0]. Therefore, the host may read the updated data as soon as the microapplication is capable of accepting a new command. Note that HMEM[0xb] will also be periodically updated by the CL450 approximately every eight video lines (see Section 12.1.1, Display-time Interrupt, for exact timing).

The value placed in HMEM[0xb] should be treated as accurate to ±32 words. The uncertainty implicit in this range of values is due to the CL450 hardware moving blocks of data between DRAM, CMEM and the VLC Decoder’s working buffer without intervention from the CL450’s CPU. Values computed by the microapplication are based on an internal snapshot that may be slightly out of date before the buffer fullness computation is complete.
NewPacket()

<table>
<thead>
<tr>
<th>Format:</th>
<th>NewPacket (length, timeStamp2, timeStamp1, timeStamp0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Low</td>
</tr>
<tr>
<td>Category:</td>
<td>Control</td>
</tr>
</tbody>
</table>

**Syntax:**

- **HMEM[0]**
  - NewPacket
  - 0x0408

- **HMEM[1]**
  - length
  - bits[15:0] = packet length, in bytes (bit 0 always 0)

- **HMEM[2]**
  - timeStamp2
  - bits[14:3] = 0
  - bits[2:0] = PTS[32:30] (or "don't care" if V/d is 0)

- **HMEM[3]**
  - timeStamp1
  - bit[15] = 0
  - bits[14:0] = PTS[29:15] (or "don't care" if V/d is 0)

- **HMEM[4]**
  - timeStamp0
  - bit[15] = 0
  - bits[14:0] = PTS[14:0] (or "don't care" if V/d is 0)

The `NewPacket()` command is used by the host to:

- Supply MPEG system-layer bitstream information to the CL450.
- Allow the CL450 microapplication to associate this information with the correct portions of the elementary video bitstream it is receiving as they pass through the circular bitstream buffer located in DRAM.

`NewPacket()` commands do not “execute” in the same sense as other macro commands; instead, they are queued until the corresponding packet data has been decoded. That is, if a `NewPacket()` command is given immediately preceding a `SetBorder()` command, the `SetBorder()` command does not have to wait until the `NewPacket()` command “executes.”

The microapplication assumes that the use of `NewPacket()` is modal when not in the IDLE command state. `NewPacket()` is modal because the host must either (1) issue no `NewPacket()` commands at all or (2) issue `NewPacket()` commands corresponding to all of the bitstream data.

Following execution of the `Reset()` macro command or CL450 initialization, the host may choose whether or not `NewPacket()` commands are used by either issuing or not issuing a `NewPacket()` command prior to transmitting its first word of data to the CL450 as shown in Figure 11-6.
NewPacket() is considered an optional command from the point of view of the host. However, the following features are unavailable if NewPacket() is not used:

- **The RDY (ready for data) interrupt**: If RDY is not disabled, the microapplication may still produce interrupts to the host, but with indeterminate timing. (See page 12-22.)

- **Audio/video synchronization**: See Chapter 13, Audio/Video Synchronization.

If the NewPacket() macro command is used, then the following items are true:

- Any number of NewPacket() commands may be issued ahead of the corresponding data arriving at the CL450, as long as the Command FIFO does not overflow (see Section 11.4, Command Latency).

- The NewPacket() command which corresponds to a packet of data must be accepted by the CL450 (HOST_newcmd[0] returned to 0) before the first word of data from that packet enters CMEM for every packet.

- The SCR counter must be initialized with the correct value before the first NewPacket() command that has HMEM[2][Vld] equal to 1 is issued.
NewPacket()

11.6.1 length Argument
Each NewPacket() command issued corresponds to a specific portion of the elementary video bitstream, the size of which is specified by the length argument. Note that this value is in units of bytes and must be an even number because transfers from the host to CMEM must always be performed one word (two bytes) at a time.

If the NewPacket() command is used, bitstream data received by the CL450 must be preceded by the corresponding NewPacket() command. This means that the sum of the length arguments of all NewPacket() commands which have been issued since the command state last changed from IDLE must be greater than or equal to the number of bytes of bitstream data that the CL450 has received. If at any time this condition is not met, the behavior of the CL450 is indeterminate until the command state next becomes IDLE.

While it is typically most convenient for the host to create NewPacket() macro commands using information from the packets in the system-level portion of an MPEG bitstream, the host can also add or remove packet boundaries. In particular, if a host is playing an MPEG elementary video stream and wishes to use the RDY interrupt, the host will have to create NewPacket() commands. In any case, if the host constructs its own packets, the following criteria must still be met:

- Packets sent to the CL450 must always have an even length.
- Packet length arguments must be less than 64K bytes (because the length argument is only 16 bits).
- PTSs (if provided) must be encoded consistent with the MPEG requirements for system-level packets and satisfy the buffering constraints of a System Target Decoder (STD).

11.6.2 timeStamp Arguments
The remaining three arguments to NewPacket() make up the PTS associated with the data corresponding to the NewPacket() command. Note that the PTS is optional for each particular NewPacket() command which is issued. Although, if the CL450 is to perform audio/video synchronization (see Chapter 13), at least one PTS must be supplied. The host indicates that a valid PTS is provided with a NewPacket() command by setting the Vld bit (HMEM[2][15]) to 1.
NewPacket()

Note that, as in the MPEG standard, the PTS is broken into three fields, the first two being 15 bits long, the third field being three bits long. For more information on how the CL450 uses PTSs, see Chapter 13.

11.6.3 Loading and Removing NewPacket Commands
NewPacket() commands are placed in the Command FIFO as they are issued by the host. The CL450 buffers the commands in the Command FIFO and then applies the information found in each NewPacket() command to the number of coded data bytes specified by the length portion of the command.

NewPacket() commands are removed from the Command FIFO in groups each time a picture_start_code is encountered while decoding the bitstream.

An example of the relationship between NewPacket() commands stored in the Command FIFO and the corresponding data in the bitstream buffer is shown in Figure 11-7.
NewPacket()

Figure 11-7   Packet Representation in Command FIFO vs. Bitstream Buffer

1. Bitstream decoded but NewPacket() not yet discarded
2. Bitstream partially decoded
3. and 4. Bitstream resident in buffer; waiting for decode
5. Packet has not been entirely received by the CL450
### Pause()

**Format:** Pause()

**Priority:** Low

**Category:** Play-type

**Syntax:**
- HMEM[0]  
  - Pause
  - 0x000e
- HMEM[1]  
  - 0x0000
- HMEM[2]  
  - 0x0000
- HMEM[3]  
  - 0x0000
- HMEM[4]  
  - 0x0000

When the `Pause()` macro command is executed, the microapplication enters the PAUSE state, passing through the PLAY-SETUP and PLAY states if necessary. In the PAUSE state, the last picture is kept on the display, and the CL450 continues to process macro commands as if it were in the IDLE state (i.e., without waiting for a picture decode to complete).

While in the PAUSE state, no bitstream decoding is performed. Because of this, the host must ensure that the bitstream buffer does not overflow either by (1) ending the transmission of coded data to the CL450 or by (2) periodically executing the `FlushBitstream()` macro command (see page 11-16). Upon execution of another Play-type macro command, decoding will resume with the next unused bit from the bitstream buffer.

Pictures are displayed while in the PAUSE state in the manner appropriate to how they were decoded. Still pictures decoded while in the STILL state are displayed with data from two separate DRAM picture buffers in the even and odd display fields, while pictures decoded in other states are displayed with data from one DRAM picture buffer displayed in both the even and odd display fields.
When the Play() macro command is executed, the microapplication enters the PLAY state (passing through PLAY-SETUP if necessary). In this state, the CL450 decodes and displays pictures at the rate prescribed by the picture_rate parameter contained within the bitstream, and uses the time stamp information transmitted by the NewPacket() command, if any, to synchronize the video display to the SCR.

While the microapplication is in the PLAY state, bitstream data is consumed at the normal rate. For systems in which the CL450 is supplied with bitstream data at a fixed rate, the host does not need to allow for bitstream buffer overflow, assuming that the bitstream is being transferred at the rate for which it was encoded. Note that the host must still prevent CMEM overflow if programmed access (rather than DMA) is being used (see Section 9.2, Bitstream Transfer Process).

All audio/video synchronization options may be used with the Play() command (see Chapter 13).

The Play() macro command should only be executed when the microapplication is not in the PLAY command state.
Reset() is a high-priority macro command which is used to re-initialize the CL450 and its microapplication. When this command is executed:

- The contents of the bitstream buffer, the Command FIFO, and the picture buffers are lost.
- The video display process is re-initialized and the output window blanked (screen is filled with current border color).
- The default settings given in Tables 10-1 and 10-3 are restored.

This is the only high-priority macro command which has an effect on the command processing state.

The Reset() macro command is typically used only to recover from error conditions. When suspending and resuming decode operations, or when changing from decoding one bitstream to another, some combination of the FlushBitstream() (page 11-16), Pause() (page 11-25), and SetBlank() (page 11-29) commands should be used. Unlike the Reset() command, these commands allow the CL450 to continue to receive bitstream data and to continue to display the last picture decoded.

Note that, unlike the other high-priority commands, the Reset() macro command has *not* been completely executed when HOST_newcmd[0] is cleared (see Section 10.2.2, Loading Sequence). Also, the Reset() command must not be executed while the host has a DRAM semaphore (SEQSEM or PICSEM) allocated.
Scan()

Format: Scan()
Priority: Low
Category: Play-type
Syntax:

When the Scan() macro command is executed, the microapplication enters the SCAN state, possibly passing through the PLAY-SETUP and PLAY states. While in the SCAN state, a single I-picture from the incoming bitstream is decoded and stored in the CL450’s DRAM.

Note: Whether or not NewPacket() commands are being used, audio/video synchronization has no effect on the timing of picture decoding which occurs in the SCAN state.

Once decoding is complete, the microapplication will automatically transition to the PAUSE state (see the Pause() command, page 11-27) and issue the SCN interrupt, if enabled (see page 12-24).

Coded data passed to the CL450 before an I-picture is either decoded normally (sequence and GOP header information) or discarded (P- and B-pictures). Once the first I-picture has been decoded, the microapplication stops processing the bitstream until some other macro command causes the microapplication to leave the PAUSE state.

The decoded I-picture is stored in one of the picture buffers in the CL450’s DRAM and is immediately posted for display; it remains in DRAM until the next Play-type macro command is executed.
SetBlank() macro command is used to blank and unblank the output window. The state argument determines whether the output window should be blanked (1) or unblanked (0). If the output window is unblanked, then the microapplication resumes displaying decoded pictures. If the output window is to be blanked, the output window is filled with the border color.

Note that blanking the output window has no effect on the border color, and the decoding process continues as though the output window were not blanked.

Once a SetBlank(1) has been executed, the display is blanked after the first VSYNC following execution of the command. After a SetBlank(0) has been executed, the display is unblanked after the first VSYNC after the command is executed.
SetBorder()

Format: SetBorder(leftBorder, topBorder, rBorder, gbBorder)
Priority: Low
Category: Set-type
Syntax: 

| HMEM[0] | SetBorder | 0x0407 |
| HMEM[1] | leftBorder | bits[15:10] = 0 |
| | | bits[9:0] = distance from active HSYNC to left edge of output window, in VCLK periods (min. 10) |
| | | bits[14:0] = distance from active VSYNC to top edge of output window, in VSYNC periods |
| | | bits[7:0] = Red or Cr (=Cb) component of border color |
| | | bits[7:0] = Blue or Cb (=Cr) component of border color |

The SetBorder() macro command is used to set:

- The color of the border
- The size of the top and left borders around the output window

Together with the SetWindow() command and the HSYNC and VSYNC input signals, the SetBorder() command is also used to determine the right and bottom borders as well as the size of the active window. The new values for both commands take effect on the active edge of VSYNC following command execution.

Figure 11-8 shows the relationship of the SetBorder() command’s four arguments to the arguments of the SetWindow() macro command (xOffset, yOffset, width, height) and the configuration of the CL450’s display output.

rBorder and gbBorder are 16-bit integers that determine the border color. The border color specified is encoded either in the RGB color space (if the CL450’s color space converter is enabled; see page 11-34).
or the YCbCr color space. Note that both chrominance values must be the same (Cb = Cr) if the color space converter is not used.

The \texttt{leftBorder} and \texttt{topBorder} arguments are encoded specially, so that a value of 0 in either causes the CL450 to automatically set the corresponding parameter such that the output window is centered within the video display.

Generally, auto-centering should not be used if one extent of the decoded picture is greater than the maximum output window size. If it is, then auto-centering causes the output window to be expanded to its maximum size in the corresponding direction and the corresponding offset (\texttt{xOffset} or \texttt{yOffset}) to be set by the \texttt{SetWindow()} command. \texttt{xOffset} will be computed so the horizontal center of the decoded image is displayed, and \texttt{yOffset} will be forced to 0.
Auto-centering is performed using the current display resolution set by the SetVideoFormat() macro command (see page 11-40). The microapplication also assumes that the correct `leftBorder` value to center a maximum-width output window is 60, and that the correct `topBorder` value to center a maximum-height output window is 18. If these values are not correct for the VSYNC and HSYNC timing used by the video display, auto-centering cannot be used.

In general, the host must ensure that the dimensions of the display window do not exceed the dimensions of the decoded pictures, and that the display window is not positioned on the screen in such a way that the CL450 does not attempt to complete a B-picture display before B-picture decoding is guaranteed to complete (see Section 15.3.2.)

The units and minimum/maximum values of the `leftBorder` and `topBorder` arguments are shown in Table 11-4.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Units</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Equal to 0 Means:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>leftBorder</code></td>
<td>1 display pixel</td>
<td>10</td>
<td>0x3ff</td>
<td>Enable horizontal auto-centering</td>
</tr>
<tr>
<td><code>topBorder</code></td>
<td>1 source pixel</td>
<td>6</td>
<td>0x7fff</td>
<td>Enable vertical auto-centering</td>
</tr>
</tbody>
</table>

The size of the bottom and right border areas are determined by the corresponding video timing signal, opposite border size, and corresponding output window size as shown in Table 11-5 below. Note that the video “Borders” defined in the table are the sum of the border and blanking times in the timing of the CL450’s output. The actual size of the border seen on the video display varies depending on the relationship between the individual monitor’s timing and display area.

<table>
<thead>
<tr>
<th>Border</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Border</td>
<td><code>leftBorder</code></td>
</tr>
<tr>
<td>Right Border</td>
<td><code>HSYNC</code> period - (<code>leftBorder</code> + <code>width</code>)</td>
</tr>
<tr>
<td>Top Border</td>
<td><code>topBorder</code></td>
</tr>
<tr>
<td>Bottom Border</td>
<td><code>VSYNC</code> period - (<code>topBorder</code> + <code>height</code>)</td>
</tr>
</tbody>
</table>
There are circumstances in which the CL450 does not decode all of the picture area specified by the picture_height and picture_width parameters; however, the following rules always apply:

- The full width of the coded picture is always decoded.
- Only the first 330 or 396 macroblocks (for NTSC or PAL video output formats, respectively) of any picture are decoded, regardless of the picture size.

Note that when vertical auto-centering is used, the limited decoded picture height is accounted for, but not necessarily the picture decoding time requirements.

For additional information on the timing of the SetBorder() macro command, see the description of the SetWindow() macro command on page 11-42.
SetColorMode()

Format:  SetColorMode(mode)
Priority: Low
Category: Set-type
Syntax:  HMEM[0]  SetColorMode  0x0111
         HMEM[1]  mode
         bits[15:1] = 0
         bit[0] = 1 (RGB) or 0 (YCbCr)
         HMEM[2]  0x0000
         HMEM[3]  0x0000
         HMEM[4]  0x0000

The SetColorMode() macro command is used to enable and disable the
CL450’s color-space converter; it configures the CL450 for either RGB
or YCbCr video output. (The color-space converter operates using the
method and coefficients included in the description of the VID_sela and
VID_selb registers in Section 8.5.2, Indirect Video Registers.)

The **mode** argument to SetColorMode() is a 16-bit parameter whose
only allowed values are as follows:

- **mode 0**: Puts the CL450 into YCbCr mode by disabling the color-
space converter. Decoded pixel values are output with a Y com-
ponent value output on PD[15:8] every VCLK, Cb and Cr are alter-
nately output on PD[7:0], and the contents of VID_selaux are
output on PD[23:16].

- **mode 1**: Puts the CL450 into RGB mode by enabling the color-
space converter. Converts each decoded pixel into R, G, and B
components, which are output on the PD[7:0], PD[15:8], and
PD[23:16] pins, respectively.

This command affects the encoding of the border color selected using
the SetBorder() macro command (see page 11-30); the Cr and Cb com-
ponents of the border color must be equal if the color-space converter is
disabled.
The SetColorMode() command must be used to enable and disable the CL450’s color-space converter because the microapplication initializes the VID_selmode, VID_sela, and VID_selb registers when execution begins, and because these registers cannot be accessed by the host while the microapplication is executing.
SetInterruptMask() 

Format: SetInterruptMask(mask)
Priority: Low
Category: Set-type
Syntax:

<table>
<thead>
<tr>
<th>HMEM[0]</th>
<th>SetInterruptMask</th>
<th>0x0104</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMEM[1]</td>
<td>mask</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bits[15:12] = 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bits[11:0] = interrupt enable bits</td>
<td></td>
</tr>
<tr>
<td>HMEM[2]</td>
<td></td>
<td>0x0000</td>
</tr>
<tr>
<td>HMEM[3]</td>
<td></td>
<td>0x0000</td>
</tr>
<tr>
<td>HMEM[4]</td>
<td></td>
<td>0x0000</td>
</tr>
</tbody>
</table>

The SetInterruptMask() macro command is used to enable and disable CL450 interrupts to the host. Each bit in the argument to this command is either reserved or corresponds to a logical interrupt which the CL450 can produce. The assignment of mask bits is given in Table 11-6.

### Table 11-6 Mask Bit Assignments

<table>
<thead>
<tr>
<th>Mask Bit</th>
<th>Interrupt Name</th>
<th>Category</th>
<th>Event</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>RDY</td>
<td>Display-time</td>
<td>Ready for data</td>
<td>12-22</td>
</tr>
<tr>
<td>5</td>
<td>END-D</td>
<td>Decode-time</td>
<td>sequence_end_code found</td>
<td>12-15</td>
</tr>
<tr>
<td>0</td>
<td>ERR</td>
<td>Decode-time</td>
<td>Bitstream data error</td>
<td>12-17</td>
</tr>
<tr>
<td>6</td>
<td>PIC-D</td>
<td></td>
<td>New picture decoded</td>
<td>12-20</td>
</tr>
<tr>
<td>9</td>
<td>SEQ-D</td>
<td></td>
<td>sequence_header_code found</td>
<td>12-25</td>
</tr>
<tr>
<td>11</td>
<td>SCN</td>
<td></td>
<td>Picture decode complete in SCAN state</td>
<td>12-24</td>
</tr>
<tr>
<td>8</td>
<td>UND</td>
<td></td>
<td>Bitstream buffer underflow error</td>
<td>12-27</td>
</tr>
<tr>
<td>4</td>
<td>END-V</td>
<td></td>
<td>Last picture display before sequence_end_code</td>
<td>12-16</td>
</tr>
<tr>
<td>2</td>
<td>GOP</td>
<td>VSYNC</td>
<td>First I-picture display after group_start_code</td>
<td>12-19</td>
</tr>
<tr>
<td>1</td>
<td>PIC-V</td>
<td></td>
<td>New picture display</td>
<td>12-21</td>
</tr>
<tr>
<td>3</td>
<td>SEQ-V</td>
<td></td>
<td>First I-picture display after sequence_header_code</td>
<td>12-26</td>
</tr>
</tbody>
</table>

1. Interrupt bits 7 and 12-15 are reserved and must be written with zero only.
Argument bits which are reserved must always be 0 when the SetInterruptMask() command is issued or indeterminate behavior will result.

A new mask value does not take effect until the SetInterruptMask() command is executed (not accepted), and controls events only at the time the microapplication internally recognizes interrupt events. Several interrupts are delayed between the time when the interrupt event is recognized and the time when the interrupt is issued to the host. (See the rules governing command latency in Section 11.4.)

In the case in which the host is enabling new interrupts with SetInterruptMask(), interrupt events which occur between the time the command is accepted and the time it is executed are not recognized, and therefore no corresponding interrupts are produced.

In the case in which the host disables a previously enabled interrupt, the microapplication issues interrupts for any events which are recognized before SetInterruptMask() is executed, even those which are not issued immediately after being recognized. This case is discussed more fully in Chapter 12, Interrupts.
The `SetThreshold()` macro command is used to specify the minimum number of bytes that must be empty in the DRAM bitstream buffer to cause the CL450 to issue the RDY interrupt (see page 12-22). The bitstream buffer threshold is checked when a new value for Buffer Fullness Status (HMEM[0xb]) is automatically computed.

The `level` argument is a 16-bit integer that specifies the number of empty bytes that must be exceeded to cause the generation of the RDY interrupt. `Level` must be a value (in bytes) less than the size of the bitstream buffer. The default value of `level` is 4096 bytes, so that the CL450 will produce RDY interrupts as long as there are more than 4096 empty bytes in the bitstream buffer and the other conditions necessary for the production of the RDY interrupt remain true.

The computation of the contents of the bitstream buffer which is compared against `level` is accurate to ±32 bytes (see page 11-19). However, the fullness of the bitstream buffer is only checked following approximately every eight active scan lines displayed, so the bitstream buffer may become considerably more “empty” before an interrupt is produced. (For a more detailed description of the timing with which buffer fullness is recomputed, see Section 12.1.1, Display-time Interrupt.)
SetThreshold()

Typically, a *level* value which is slightly less than the intended burst transfer size (possibly a multiple of the nominal average packet size) would be used. Such values range from below 2048 to below 10240, but *level* must be less than the size of the bitstream buffer and greater than 255.

Note that the internal variable which is used to generate the RDY interrupt is changed when this command is *executed*, not issued. If *level* is changed dynamically, the host must be aware of the latency between the CL450 accepting a SetThreshold() command and the time at which it is executed. Also, once the command is executed, it affects the *future* generation of the RDY interrupt. If an RDY interrupt is already pending when this command is issued, it will not be cleared by the CL450.
The SetVideoFormat() macro command is used to configure the output resolution and timing of the CL450's video bus. Values 3 and 4 of the format argument correspond to PAL and NTSC resolution and timing values, respectively. The quantities which are set by this command are given in Table 11-7.

### Table 11-7  Video Format Summary

<table>
<thead>
<tr>
<th>Parameter</th>
<th>3 (PAL) Value</th>
<th>4 (NTSC) Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Horizontal Resolution</td>
<td>352 pixels</td>
<td>352 pixels</td>
</tr>
<tr>
<td>Maximum Vertical Resolution</td>
<td>288 pixels</td>
<td>240 pixels</td>
</tr>
<tr>
<td>Nominal VSYNC Frequency</td>
<td>50 Hz</td>
<td>60 Hz</td>
</tr>
</tbody>
</table>

1. All resolutions in this table are given in MPEG source pixels, not the unsampled (interpolated) output pixels.

The resolution values are used by the CL450 when SetWindow() and SetBorder() commands are executed (see page 11-42 and page 11-30, respectively) or when new sequence-layer parameters are processed (see Section 14.1.1, Sequence Variable Group). The VSYNC frequency is used in conjunction with the picture_rate sequence-layer parameter to determine if frame rate conversion must be performed (see Section 15.1, Frame Rate Conversion).
The new VSYNC frequency takes effect immediately after the command is executed. However, if auto-sizing (see SetWindow() on page 11-42) or auto-centering (see SetBorder() on page 11-30) are being used, then SetBorder() and/or SetWindow() must be explicitly executed by the host, or new sequence header information processed, for the new display resolution to take effect.
SetWindow()

<table>
<thead>
<tr>
<th>Format:</th>
<th>SetWindow (xOffset, yOffset, width, height)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Low</td>
</tr>
<tr>
<td>Category:</td>
<td>Set-type</td>
</tr>
<tr>
<td>Syntax:</td>
<td></td>
</tr>
<tr>
<td>HMEM[0]</td>
<td>SetWindow</td>
</tr>
<tr>
<td></td>
<td>0x0406</td>
</tr>
<tr>
<td>HMEM[1]</td>
<td>xOffset</td>
</tr>
<tr>
<td></td>
<td>bit[15] = 0</td>
</tr>
<tr>
<td></td>
<td>bits[14:0] = horizontal offset of output window</td>
</tr>
<tr>
<td></td>
<td>start from picture start</td>
</tr>
<tr>
<td>HMEM[2]</td>
<td>yOffset</td>
</tr>
<tr>
<td></td>
<td>bit[15] = 0</td>
</tr>
<tr>
<td></td>
<td>bits[14:0] = vertical offset of output window</td>
</tr>
<tr>
<td></td>
<td>start from picture start</td>
</tr>
<tr>
<td>HMEM[3]</td>
<td>width</td>
</tr>
<tr>
<td></td>
<td>bit[15] = 0</td>
</tr>
<tr>
<td></td>
<td>bits[14:0] = horizontal size of output window</td>
</tr>
<tr>
<td>HMEM[4]</td>
<td>height</td>
</tr>
<tr>
<td></td>
<td>bit[15] = 0</td>
</tr>
<tr>
<td></td>
<td>bits[14:0] = vertical size of output window</td>
</tr>
</tbody>
</table>

1. See Table 11-8 for argument units.

The SetWindow() macro command is used in conjunction with SetBorder() (see page 11-30) to determine the portion of the decoded image that the CL450 displays in the video window.

Figure 11-9 shows the relationship of this command's four arguments to the configuration of the CL450's display output. Note that the quantities leftBorder, topBorder, rBorder, and gbBorder shown in the figure are the arguments of the SetBorder() command.

The width and height arguments are encoded specially, so that a value of 0 in either causes the CL450 to automatically set the corresponding parameter from the _size fields in the sequence-layer parameters (i.e., the output window becomes the full size of the decompressed picture).
The host must ensure that the combination of the arguments to this command does not cause areas outside of the decoded picture to be displayed. If the host provides arguments which specify that part of the output window falls outside of the decoded picture, then indeterminate data will appear in that portion of the output window in the video display. To avoid this occurrence, use argument values less than or equal to the maximums given in Table 11-8.

The arguments to SetWindow() take effect upon the first active VSYNC edge after the command has executed.
SetWindow()

Table 11-8 SetWindow() Argument Restrictions

<table>
<thead>
<tr>
<th>Argument</th>
<th>Unit</th>
<th>Recommended Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>xOffset</td>
<td>1 source pixel</td>
<td>horizontal_size - 1</td>
</tr>
<tr>
<td>yOffset</td>
<td>1 source pixel</td>
<td>vertical_size - 1</td>
</tr>
<tr>
<td>width</td>
<td>1 display pixel</td>
<td>Smallest of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>□ 2 * (horizontal_size - xOffset)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>□ 704</td>
</tr>
<tr>
<td>height</td>
<td>1 source pixel</td>
<td>Smallest of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>□ vertical_size - yOffset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>□ Maximum Vertical Resolution, see Table 11-7</td>
</tr>
</tbody>
</table>

Note that the arguments to the SetWindow() command use units of one source pixel for offsets within the decoded picture and for all vertical units, and one display pixel (half a source pixel) for the horizontal dimension of the video display, as shown in Table 11-8. Pixel dimensioning is done this way because each pixel in the decoded picture (source pixel) is horizontally interpolated upon display to provide two display pixels. Thus, the xOffset within the picture is in source pixels (pre-interpolation), while the left display border and the width of the output window are in display pixels (post-interpolation).
When the SingleStep() macro command is executed, the microapplication enters the STEP state, possibly passing through the PLAY-SETUP and PLAY states. While in the STEP state, a single picture from the incoming bitstream is decoded and posted for display.

Note: Audio/video synchronization has no effect on the timing of picture decoding which occurs in the STEP state, regardless of whether or not NewPacket() commands are being used.

Once decoding is complete, the microapplication automatically transitions to the PAUSE state (see the Pause() command, page 11-25).

Note that the picture most recently decoded is not necessarily the one which is displayed while in the PAUSE state. Because of the picture reordering which is performed after I- and P-pictures are posted for display, the picture which the SingleStep() command caused to be decoded can be stored within the CL450's DRAM, and another picture (earlier in the display order) can be displayed instead. This situation is particularly apparent when SingleStep() is issued while in the IDLE command state. In this case (for most bitstreams), two pictures must be decoded before the output window is unblanked and the first picture decoded is displayed. Typically, one picture will be decoded in the PLAY state, and the other one in the STILL state.
SlowMotion()
AccessSCR() macro command and/or the FlushBitstream() macro command may be used to correct the value of the SCR counter before transitioning back to the PLAY state.

When in the SLOW state, every decoded picture is displayed for two times the speed argument field times. This function overrides both the microapplication frame rate transcoding mechanism and audio/video synchronization.

Normally, the number of fields for which each decoded picture is displayed varies from 1 to 3 and is determined dynamically by the ratio of the picture_rate coded in the bitstream and the current video output format, controlled by the SetVideoFormat() macro command. While in the SLOW state, this comparison is disregarded, and the nominal value of 2 field periods per picture for normal rate display is assumed. Because of this, the actual play rate while in the SLOW state is derived from multiplying the speed argument times one of the values from Table 11-9.

<table>
<thead>
<tr>
<th>Format</th>
<th>picture_rate</th>
<th>Scale Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTSC (30 Hz)</td>
<td>30Hz / 29.97Hz</td>
<td>1.00</td>
</tr>
<tr>
<td>NTSC (30 Hz)</td>
<td>25 Hz</td>
<td>.83</td>
</tr>
<tr>
<td>NTSC (30 Hz)</td>
<td>24 Hz / 23.97 Hz</td>
<td>0.80</td>
</tr>
<tr>
<td>PAL (25 Hz)</td>
<td>30 Hz / 29.97 Hz</td>
<td>1.20</td>
</tr>
<tr>
<td>PAL (25 Hz)</td>
<td>25 Hz</td>
<td>1.00</td>
</tr>
<tr>
<td>PAL (25 Hz)</td>
<td>24 Hz / 23.97 Hz</td>
<td>0.96</td>
</tr>
</tbody>
</table>

For example, requesting a speed of 4 using SlowMotion() when displaying a bitstream coded at 25 Hz on a 30 Hz display will actually play at 0.83 x 4, or 3.32 times slower than normal, rather than 4.00 times slower.
12
Interrupts

The 11 logical host interrupts which the CL450 microapplication generates are produced within the microapplication by polling internal conditions while the decoding and display processes execute. Because of this, the times at which interrupts can be produced (the domain for interrupts) is restricted. For example, the CL450 microapplication will not produce new interrupts in any of the following circumstances:

- The microapplication is not executing (CPU_control[0] == 0).
- The microapplication is in the IDLE state.
- The SetInterruptMask() macro command has executed with a mask argument of 0.
- The Interrupt Status location of HMEM is non-zero.

>Note: Although new interrupts (INT pin transitioning to low) cannot be generated during any of these conditions, it is possible for the INT pin to remain active during these conditions. Once the INT pin is active, it will remain active until forced inactive by the host (see Section 12.2, Interrupt Handshaking).
12.1 Interrupt Types

SetInterruptMask(), used to enable and disable CL450 interrupts to the host, is a low-priority macro command. Rules given in Chapter 11, Macro Commands, explain how to determine the timing delay before a low-priority command is executed. Note that new mask bits won’t be effective until SetInterruptMask() is executed, not just accepted.

Interrupts are divided into the following three separate classes based on when they are reported to the host:

- **Display-time Interrupt**
- **Decode-time Interrupts**
- **VSYNC Interrupts**

Each interrupt is referenced by the event which causes it in Table 12-1 below, and is described in detail in Section 12.3.

Table 12-1 CL450 Interrupt Summary

<table>
<thead>
<tr>
<th>Category</th>
<th>Interrupt Name</th>
<th>Event</th>
<th>Mask Bit</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display-time</td>
<td>RDY</td>
<td>Ready for data</td>
<td>10</td>
<td>12-22</td>
</tr>
<tr>
<td></td>
<td>END-D</td>
<td>sequence_end_code found</td>
<td>5</td>
<td>12-15</td>
</tr>
<tr>
<td></td>
<td>ERR</td>
<td>Bitstream data error</td>
<td>0</td>
<td>12-17</td>
</tr>
<tr>
<td>Decode-time</td>
<td>PIC-D</td>
<td>New picture decoded</td>
<td>6</td>
<td>12-20</td>
</tr>
<tr>
<td></td>
<td>SEQ-D</td>
<td>sequence_header_code found</td>
<td>9</td>
<td>12-25</td>
</tr>
<tr>
<td></td>
<td>SCN</td>
<td>Picture decode complete in SCAN state</td>
<td>11</td>
<td>12-24</td>
</tr>
<tr>
<td></td>
<td>UND</td>
<td>Bitstream buffer underflow error</td>
<td>8</td>
<td>12-27</td>
</tr>
<tr>
<td>VSYNC</td>
<td>END-V</td>
<td>Last picture display before sequence_end_code</td>
<td>4</td>
<td>12-16</td>
</tr>
<tr>
<td></td>
<td>GOP</td>
<td>First I-picture display after group_start_code</td>
<td>2</td>
<td>12-19</td>
</tr>
<tr>
<td></td>
<td>PIC-V</td>
<td>New picture display</td>
<td>1</td>
<td>12-21</td>
</tr>
<tr>
<td></td>
<td>SEQ-V</td>
<td>First I-picture display after sequence_header_code</td>
<td>3</td>
<td>12-26</td>
</tr>
</tbody>
</table>

1. Interrupt bits 7 and 12-15 are undefined and must be set to zero only.

Note: The CL450’s internal CPU can also receive interrupts from on-chip sources. The host has no control over these interrupts, and they should not be confused with the host interrupts generated by the CL450.
12.1.1 Display-time Interrupt

The single Display-time interrupt may be issued any time the microap­
plication is in a command processing state other than IDLE and PLAY­SETUP. In all cases, the event which causes a Display-time interrupt to
be issued is the active (falling) edge of the HSYNC input to the CL450.
Which HSYNCs can produce Display-time interrupts is determined dif­
ferently in each of the three different sections of the field display (VSYNC) period: the top border, vertical active region, and bottom bor­
der. Figure 12-1 shows these three sections in typical NTSC vs. PAL
display environments.

Within the top border, Display-time interrupts may be generated roughly every eighth HSYNC. Within the vertical active region, Display-time interrupts may be generated at the end of each block row displayed, as shown in Figure 12-2. That is, after the display of the last scan line of a decoded block row, the microapplication checks to see if the Display­
time conditions are true. This check is done on block row boundaries,
which means after the last (eighth, or seventh if counting from 0) scan
line of one block row and before the first scan line of the next within the
decoded picture.
Interrupt Types

**Top Border**

The internal counter used to determine when Display-time interrupts are to be produced is initialized at the active edge of VSYNC with the `topBorder` value. Subsequently, each time an active HSYNC occurs, the counter is decremented, and if the least-significant three bits of the result are 0, then a Display-time interrupt may be produced. Thus the first possible time a Display-time interrupt can be produced in the top border will be

\[(\text{topBorder} \& 7) + 1\]

active HSYNC edges after active VSYNC. The last possible Display-time interrupt in the top border will always occur on the HSYNC edge which marks the end of the last top border scan line and the beginning of the first scan line of the vertical active region.

The block row boundary within the decoded picture which causes the first possible interrupt is dependent only on `yOffset` of the `SetWindow()` macro command.
Because of selected `yOffset`, the block row boundaries of the decoded picture are not necessarily the same as scan line numbers within the video display. In fact, the output window could begin in the middle of a decoded picture block row as in the case of `yOffset`=12 of Figure 12-2, which causes the first possible interrupt to be delayed from the top of the output window by “8-(yOffset%8)” (4) scan lines. (Note that scan line numbers start at 0.)

**Bottom Border**

Within the bottom border, the number and location of opportunities for issuing Display-time interrupts depends on the **height** of the active display region and the current video display **mode**. As in the top border region, an internal counter is used to determine when and how many Display-time interrupts may be issued during the bottom border. This counter is initialized on the active HSYNC edge which separates the last scan line of the vertical active period from the first bottom border scan line, with the value:

\[ 255 - (\text{topBorder} + \text{height}) \]

If the initial value is less than or equal to 0, then no Display-time interrupts will be generated during bottom border. Otherwise, the counter is decremented on subsequent active HSYNC edges, and interrupts may be produced each time the least significant three bits are 0. The last Display-time interrupt during bottom border will occur when the counter decrements to 0, after which no Display-time interrupts will be generated until an active VSYNC edge is detected and the sequence begins again.

*Note: For proper operation of the video display, the active VSYNC edge must occur after this counter decrements to 0. This creates an effective minimum field period requirement.*

When the video display is blanked, either because the first picture after leaving the IDLE state has not yet been displayed or due to the SetBlank() macro command, the entire field period is treated as if it were the bottom border and the sum of the **height** and **topBorder** parameters was 15. In this case, there will be exactly 30 possibilities for Display-time interrupts to be issued (assuming a normal field period is used), and
the first opportunity will occur 8 active HSYNC edges after each active VSYNC edge.

12.1.2 Decode-time Interrupts
Unlike the Display-time interrupt, which the microapplication periodically polls for, Decode-time interrupts are issued every time the appropriate bitstream element is decoded, if the interrupt is enabled. Thus, Decode-time interrupts are synchronous with picture decoding (which occurs in advance of picture display) and they may be issued as soon as the microapplication leaves the PLAY-SETUP state, even if the output window is not yet unblanked.

Note: There may be a delay between the time that the bitstream construct which could cause a Decode-time interrupt event enters the CL450's host bus and the time when the interrupt occurs. This delay is due to the bitstream being stored in the CL450’s internal bitstream buffer before decoding.

12.1.3 VSYNC Interrupts
VSYNC interrupts are issued following the active (rising) edge of VSYNC which begins the top border time of a new display field. VSYNC interrupts are only issued at the beginning of those fields which are the first display of a newly-decoded picture. VSYNC interrupts are not issued at the start of field displays for pictures which have already been displayed once, including the second field display of a still picture, even though the pixel values being displayed have not been displayed before.

Figure 12-3 shows an example of the relationship between the VSYNC pulses which divide field periods, the pictures which are displayed, and the times when VSYNC interrupts may be produced. Each box shown represents the display of a video field. The boxes are labeled using the standard MPEG nomenclature in which “B16” indicates that the picture is type B (of I, P, and B) and number 16 (in display order) in a bitstream. In addition, each box also contains a label of the form “f_n” in which n indicates the number of times that a field containing that picture has already been displayed.

Figure 12-3 shows a bitstream with a picture_rate of 25 Hz being displayed at a nominal picture rate of 30 Hz (field rate 60 Hz), with picture B16 therefore displayed for three field periods due to pull-down.
VSYNC interrupts are not produced while the output window is blanked because newly decoded pictures are not being displayed.

![VSYNC Interrupt Generation](image)

**Figure 12-3  VSYNC Interrupt Generation**

All interrupts cause the INT pin to be asserted. To determine which interrupt(s) has occurred, the Interrupt Status location in HMEM is written by the CL450 at the same time the INT pin is asserted.

Each bit in Interrupt Status corresponds to a different logical interrupt source. These bits are allocated in the same order used for the mask argument to the SetInterruptMask() macro command, which is given in Table 12-1, CL450 Interrupt Summary.

The host may read the Interrupt Status location at any time to determine which interrupts, if any, are pending. The host may also write to this location to clear pending interrupts. However, writes to Interrupt Status must only occur as specified in this section.

The same handshaking protocol is used for all CL450 interrupts, regardless of the logical interrupt source. When the CL450 issues an interrupt to the host, it makes the INT pin active (low) and sets one or more bits in the Interrupt Status location of HMEM.

Because there is no semaphore to protect Interrupt Status from simultaneous access by both the host and the microapplication, the protocol for issuing interrupts is structured such that simultaneous access cannot occur for the following two reasons:

- The microapplication only writes Interrupt Status when it is 0.
- The host is only allowed to write Interrupt Status when it is not 0.

Once the host writes a 0 to Interrupt Status, the host cannot write to Interrupt Status again (even a 0) until the microapplication sets one or
more bits. If the host violates this protocol, then interrupts may be lost or spuriously created.

Excessive latency by the host in clearing the active bit(s) within Interrupt Status could also prevent the microapplication from issuing subsequent interrupts as distinct events.

### 12.2.1 Microapplication Behavior

When an interrupt *can* be produced, the microapplication polls Interrupt Status. If Interrupt Status is 0, then all pending interrupts, if any, are posted to Interrupt Status and the INT pin is activated, if appropriate. If Interrupt Status is non-zero, then the new interrupts, if any, will be ORed into the microapplication’s internal variable for storing pending interrupts. However, the microapplication will not re-check Interrupt Status until the next occasion in which it might post an interrupt, even if interrupts have been queued internally.

Interrupt Status is checked and queued interrupts potentially issued every time the timing for Display-time, VSYNC, or Decode-time interrupts is satisfied.

*Note:* If HMEM[Interrupt Status] is non-zero for an extended period, then multiple interrupt events of the same type may be ORed into the microapplication’s internal variable for storing new interrupts, causing events to be lost.

For example, assume that when the microapplication is in the PLAY state and in the midst of decoding and displaying a video sequence, the microapplication then decodes a new sequence_header_code. When this occurs, the current interrupt mask is checked to determine if the corresponding interrupt (SEQ-D) is enabled. If it is, the microapplication checks the current state of Interrupt Status. If Interrupt Status is 0, then the SEQ-D interrupt and any other already queued interrupts will be posted to Interrupt Status. If Interrupt Status is non-zero, then the SEQ-D interrupt is queued internally, and the next occasion when the microapplication checks Interrupt Status will depend on the other interrupt sources. If SEQ-D was disabled in mask, but there were queued interrupts, the queued interrupts would still be posted.
The microapplication’s half of the interrupt handshaking protocol is shown in the pseudocode example of the “MaybeIssueInt” function of Figure 12-4.

```c
void MaybeIssueInt(unsigned short newInts)
{
    static unsigned short pendingInts= 0;

    pendingInts|= newInts;
    if (HMEM[Interrupt Status] == 0)
    {
        HMEM[Interrupt Status]= pendingInts;
        INT pin= LOW;
        pendingInts= 0;
    }
}
```

**Figure 12-4  CL450 Interrupt Posting Procedure (pseudocode)**

The decision flow of the CL450 posting procedure of Figure 12-4 is shown from a block diagram viewpoint in Figure 12-5. Every time the microapplication reaches a point where it could issue a Display-time, Decode-time, or VSYNC interrupt, it performs equivalent operations.
Interrupt Handshaking

Figure 12-5  CL450 Interrupt Posting Procedure (block diagram)
12.2.2 Host Behavior

Each time the microapplication issues one or more new interrupts to the host, it always sets the corresponding bit(s) in Interrupt Status and makes the INT pin active. The microapplication performs these actions inseparably. However, the times at which the host may clear the Interrupt Status location of HMEM and make the INT pin inactive may vary greatly from each other.

For example, in a system in which the automatic interrupt clear feature is used (enabled by HOST_control[15]), the INT pin will become inactive as soon as the hardware interrupt acknowledge cycle occurs. But even though the hardware interrupt acknowledge has occurred, the microapplication will not issue another interrupt until the host clears Interrupt Status.

Alternately, since many hardware interrupt controllers cannot be reconfigured to receive a new interrupt until the interrupt signal has become inactive, the host may clear Interrupt Status (telling the CL450 that a new interrupt can be produced) but not deactivate INT or re-initialize the host’s interrupt controller. In this case, the microapplication might issue another interrupt which would not be recognized by the host processor, possibly resulting in lost interrupts or a deadlock in which it appears that no interrupts are being generated.

Because the host typically needs to (1) clear Interrupt Status as soon as possible to minimize the interrupt latency added to the system by the host and (2) service multiple logical interrupts received with the same physical interrupt, it follows an interrupt handling procedure somewhat more complicated than that of the microapplication.

Figure 12-6, for example, shows sample pseudocode for a host interrupt service routine for the CL450, which minimizes the amount of elapsed time between an interrupt being issued by the microapplication and Interrupt Status being cleared. However, once entered, this routine holds the processor and does not allow any other interrupts to be serviced or any non-interrupt code to execute until all CL450 interrupts, even those issued while earlier interrupts are being processed, have been processed.
Interrupt Handshaking

interrupt_service_routine_type isr(void)
{
  static unsigned int unservicedInts= 0;
  unsigned int newInts;
  unsigned short temp;
  do {

    if (INT pin == LOW){
      /* is new int pending? Or are
       * we servicing another logical
       * source from last physical int?
       */
      temp= HOST_control;
      temp&= OxCO01;
      temp|= Ox80;
      HOST_control= temp;

      newInts= HMEM[Interrupt Status];
      HMEM[Interrupt Status]= 0;
    }
    /* At this point the microcode is
     * able to issue another interrupt
     */

    if (newInts & unservicedInts)
      handle error condition;
    /* we’ve received a new int and
     * still haven’t serviced the last
     * one of the same type
     */

    unservicedInts|= newInts;
    switch (unservicedInts){
      /* might be a series of if-else’s instead. Arrange cases so
       * that highest priority interrupt is checked first and
       * process one interrupt only for each pass through the switch.
       */
      /* For each pass: */
      perform interrupt-specific processing;
      clear corresponding bit in unservicedInts;
    }
  } while (unservicedInts != 0 ||
           INT pin == LOW);
  /* got multiple logical ints */
  /* CL450 issued new physical int
   * while processing last int */

  /* no more interrupts to process */
  configure host’s hardware interrupt controller to receive new ints;
}

Figure 12-6  Host Interrupt Handler Example 1
If the behavior shown in Figure 12-6 is undesirable (perhaps because there are higher-priority interrupt sources in the system), then an approach which takes advantage of the CL450's internal interrupt queuing can be used, as shown in Figure 12-7. This routine processes only one logical interrupt each time it is entered and uses Interrupt Status to hold unprocessed logical interrupts.

Note: Figure 12-6 and Figure 12-7 should be treated as examples only; the optimal interrupt handling procedure for a given CL450-based system will probably be different from either of these.

In addition to initializing the interrupt controller so a new CL450 interrupt can be received (if the interrupt controller is edge sensitive), the host must use the HOST_control register to cause an inactive pulse on the CL450’s INT pin (simulating a new physical interrupt) immediately before leaving the routine. However, with this method, it cannot be determined if more than one of each logical interrupt has occurred before being serviced.

In general, the host must ensure that all interrupts produced by the CL450 are produced when the rest of the system is ready to handle them. This responsibility not only includes configuration of the host interrupt controller, but writing proper values to HOST_control[15:14] and HOST_intvecw.

Note: It is impossible to tell which values will be used if the host writes to one of these registers “simultaneously” with the microapplication issuing a new interrupt. In general, these registers should either be configured correctly as part of system initialization, or SetInterruptMask() should be used to disable all CL450 interrupts while the registers are being modified.
Interrupt Listing

INTERRUPT_SERVICE_ROUTINE_TYPE isr(void)
{
    unsigned int newInts;
    unsigned short temp;
    newInts = HMEM[Interrupt Status];
    switch (newInts){
        /* might be a series of if-else’s instead. Arrange cases so
         * that highest priority interrupt is checked first and
         * process one interrupt only for each pass through the switch.
         */
        /* For each pass: */
        perform interrupt-specific processing;
        clear corresponding bit in newInts;
    }
    HMEM[Interrupt Status]= newInts;  /* put back ones we didn’t process */
    configure host’s hardware interrupt controller to receive new ints;
    if (!newInts){
        temp= HOST_control;  /* TNT pin = HIGH */
        temp&= 0xC001;
        temp|= 0x80;
        HOST_control= temp;
    }
    else
        maybe create TNT pulse;  /* see text */

Figure 12-7  Host Interrupt Handler Example 2

12.3  Interrupt Listing

This section gives a detailed description of the CL450 Display-time, Decode-time, and VSYNC interrupts listed together in alphabetical order.
An END-D interrupt event occurs if both of the following are true:

- A `sequence_end_code` is detected in the coded data stream while decoding is being performed.
- The END-D interrupt is enabled (mask[5] of the last SetInterrupt-Mask() macro command executed equals 1).

END-D interrupts are issued to the host immediately after the `sequence_end_code` is decoded.

When an END-D event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an END-D interrupt or places an END-D interrupt in the pending interrupt queue.

This interrupt can be used to warn the host of (1) a transition between video sequences, or (2) the end of a video stream (if the host knows how many sequences are in the stream). In particular, each time a `sequence_end_code` is decoded, the microapplication returns the sequence parameters in DRAM to the default values unless SeqNoDef is 1 (see Section 14.1.1, Sequence Variable Group).

Note: The END-V interrupt is also caused by detection of a `sequence_end_code` but is not issued until the VSYNC prior to the display of the last picture (in display order) found in the bitstream before the `sequence_end_code`. (See Section 12.1.3, VSYNC Interrupts, and END-V on page 12-16.)
The END-V interrupt event occurs at the rising edge of VSYNC prior to the first display field of the last picture (in display order) decoded prior to the detection of a sequence_end_code. Each time an END-V event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an END-V interrupt or places an END-V interrupt in the pending interrupt queue.

An END-V interrupt event occurs if all of the following are true:

- The timing for VSYNC interrupts is satisfied (see Section 12.1.3, VSYNC Interrupts).
- A sequence_end_code has been decoded.
- The picture which will be displayed in the following field time is the last picture (in display order) in the sequence terminated by the sequence_end_code.
- The END-V interrupt is enabled (mask[4] of the last SetInterrupt-Mask() macro command executed equals 1).

Note: The END-D interrupt is also caused by detection of a sequence_end_code, but is issued as soon as the decode process detects it. (See Section 12.1.2, Decode-time Interrupts, and END-D on page 12-15.)
The ERR interrupt (bitstream data error) event occurs each time the decoding process detects one of the following errors in the coded data:

- An MPEG sequence_error_code
- Invalid variable-length codes
- Marker bit's with 0 values
- Header fields with illegal values

The decoding process does not detect dropped or incorrect bits unless they cause one of these detectable conditions, in which case this interrupt may be used by the host to take the appropriate action (possibly applying high-level error concealment techniques), if any.

When an ERR event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an ERR interrupt or places an ERR interrupt in the pending interrupt queue.

An ERR interrupt event occurs if both of the following are true:

- An error is detected in the coded data stream while decoding is being performed.
- The ERR interrupt is enabled (mask[0] of the last SetInterrupt-Mask() macro command executed equals 1).

Not all data errors can be detected, and the portion of the bitstream being decoded when an error is detected will not necessarily be the portion containing the error. For example, a dropped bit may cause an erroneous VLC (variable-length code) to be detected considerably later in the bitstream.

Typically, the microapplication’s internal error concealment operates by continuing to display the nearest (in display order) correctly-decoded reference frame until it finds a portion of the bitstream that it can decode. When this is occurring, undecodable portions of the bitstream (typically B-pictures and sometimes P-pictures) are discarded at a rapid rate. Because of this, the bitstream buffer can underflow temporarily, which may cause an additional interrupt.

In a system using audio/video synchronization, the bitstream buffer eventually returns to the correct fullness level because the synchronization mechanism causes the microapplication to wait until the proper system time before decoding the next picture. Similarly, in a fixed bit-
rate system, the CL450’s decode process will underflow until the data source provides valid data, at which point the CL450 will become resynchronized.

If pictures are discarded due to error recovery in a variable bit-rate system in which synchronization is not being performed, the microapplication resumes decoding and display as soon as a decodable picture is found, which effectively reduces the playing time of the current video sequence by the number of pictures discarded. If this reduction is undesirable, the host should use the ERR interrupt and generate appropriate Pause() commands.
Name: First I-picture display after `group_start_code`

Category: VSYNC

Mask Bit: 2

The GOP interrupt event occurs at the rising edge of VSYNC prior to the first display field of the first I-picture decoded following the detection of a `group_start_code`. Each time a GOP event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues a GOP interrupt or places a GOP interrupt in the pending interrupt queue.

The microapplication uses an internal flag to produce the GOP interrupt. This flag is cleared each time the microapplication reaches the IDLE state, the FlushBitstream() command is executed, or a GOP interrupt event occurs. The flag is set each time a `group_start_code` is decoded.

A GOP interrupt event occurs if all of the following are true:

- The timing for VSYNC interrupts is satisfied (see Section 12.1.3, VSYNC Interrupts).
- The picture to be displayed in the following field time is an I-picture which has not been previously displayed.
- The GOP interrupt flag is set.
- The GOP interrupt is enabled (mask[2] of the last SetInterrupt-Mask() macro command executed equals 1).

A group of pictures can be decoded and displayed without a GOP interrupt event occurring (if it contains only P- and B-pictures), but GOP interrupt events cannot occur more often than `group_start_codes`. 
The PIC-D interrupt (new picture decoded) event occurs each time all of the following are true:

- A coded picture is detected in the coded data stream and completely decoded without error.
- The PIC-D interrupt is enabled (mask[6] of the last SetInterruptMask() macro command executed equals 1).

Each time a PIC-D event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues a PIC-D interrupt or places a PIC-D interrupt in the pending interrupt queue.

The PIC-D interrupt is closely related to the PIC-V interrupt because both are produced by the processing of newly coded pictures. However, while the PIC-D and other Decode-time interrupts are issued to the host immediately after the entire picture is decoded, the PIC-V interrupt is produced by the detection of a picture_start_code and is not issued until the VSYNC prior to the display of the picture found in the bitstream after the picture_start_code. (See Section 12.1.3, VSYNC Interrupts, and PIC-V on page 12-21.)

The order in which PIC-D and PIC-V interrupts are issued depends on the type of picture being decoded. If an I- or P-picture is decoded, then the entire picture is completely decoded before display starts, and PIC-D precedes PIC-V. If a B-picture is being decoded, then display will begin after only half of the picture is decoded, and PIC-V precedes PIC-D by approximately one field time. In addition, bitstream errors may cause PIC-V interrupts to be produced without a corresponding PIC-D (if an error occurs while decoding a B-picture) or vice versa (if an error occurs which causes a completely decoded reference frame never to be displayed).
Event: New picture display  
Category: VSYNC  
Mask Bit: 1

The PIC-V interrupt (new picture display) event occurs at the rising edge of VSYNC prior to the first display field of a newly-decoded picture. Each time a PIC-V event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues a PIC-V interrupt or places a PIC-V interrupt in the pending interrupt queue.

A PIC-V interrupt event occurs if all of the following are true:

- The timing for VSYNC interrupts is satisfied (see Section 12.1.3, VSYNC Interrupts).
- The picture to be displayed in the following field time has not been previously displayed.
- The PIC-V interrupt is enabled (mask[1] of the last SetInterrupt-Mask() macro command executed equals 1).

Note that the second field of a picture decoded using the DisplayStill() command (page 11-14) is not considered a “newly-decoded picture,” even though the data from that field has not been previously displayed. In other words, only one PIC-V interrupt will be produced per Display-Still() command.
The RDY interrupt (ready for data) event occurs when the number of empty bytes in the bitstream buffer in DRAM is greater than a given threshold value. It can be used by the host:

- To indicate that more data needs to be sent to the CL450.
- With a larger threshold, as a warning that bitstream buffer underflow is about to occur.

Each time a RDY event occurs, the microapplication either issues a RDY interrupt or places a RDY interrupt in the pending interrupt queue (See Figure 12-4).

A RDY interrupt event occurs when the following conditions are true:

- The timing for the Display-time interrupt is satisfied (see Section 12.1.1, Display-time Interrupt).
- The number of free bytes in the bitstream buffer meets or exceeds the threshold argument to the most recently executed SetThreshold() macro command.
- The number of data bytes which have been delivered to the CL450 since the last occurrence of the IDLE state or execution of the FlushBitstream() command is greater than or equal to the sum of all the length arguments to all the NewPacket() commands which have been accepted in the same interval minus 6. That is:

\[
\text{total bytes received} \geq \left( \sum \text{length} \right) - 6
\]

- At least one NewPacket() macro command has been issued.
- The RDY interrupt is enabled (mask[10] of the last SetInterrupt-Mask() macro command is set equal to 1).

The interlock between NewPacket() commands and the production of RDY interrupts is provided to minimize the interrupt traffic to the host. The microapplication assumes that if the host has issued a NewPacket() command, it knows that the corresponding data must be sent to the CL450 and the host does not have to be interrupted. The -6 bias is provided so that the interrupt will be produced slightly before the current data transfer completes.

*Note: For the second condition listed, the RDY interrupt is sensitive to the buffer emptiness being above threshold (level*
Event: Ready for data
Category: Display-time
Mask Bit: 10

sensitive), not becoming above threshold (not edge sensitive). This means that, once an RDY interrupt is produced, new RDY interrupts are produced each time a Display-time interrupt can be issued until the host takes steps to make one of the other required conditions false.

Typical host systems which use the RDY interrupt issue one or more NewPacket() macro commands within the interrupt handler that responds to the RDY interrupt. This procedure prevents redundant RDY interrupts from being produced but allows the actual transfer of data (if DMA is not used) to occur outside the interrupt handler, improving the overall interrupt latency of the system.

If the RDY interrupt is to be used by the host to control data transfers to the CL450, the host must ensure that the latency between the times when the microapplication polls to determine if an RDY interrupt should be produced does not cause bitstream buffer underflow.
The SCN interrupt (picture decode complete in SCAN state, see page 11-28) event occurs each time the decoding process finishes decoding an I-picture in the SCAN state. This interrupt event also marks the automatic transition between the SCAN and PAUSE states.

Each time an SCN event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an SCN interrupt or places an SCN interrupt in the pending interrupt queue.

An SCN interrupt event occurs if all of the following are true:

- The microapplication is in the SCAN state.
- Decoding of an I-picture has completed.
- The SCN interrupt is enabled (mask[11] of the last SetInterrupt-Mask() macro command executed equals 1).

This interrupt can be used to inform the host when a new picture is available to be displayed.
The SEQ-D interrupt (sequence_header_code found) event occurs each time the decoding process detects a sequence_header_code in the coded data. Each time a SEQ-D event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an SEQ-D interrupt or places an SEQ-D interrupt in the pending interrupt queue. SEQ-D interrupts are issued to the host immediately after the code is decoded.

An SEQ-D interrupt event occurs if all of the following are true:

- A sequence_header_code is detected in the coded data stream while decoding is being performed.
- The SEQ-D interrupt is enabled (mask[9] of the last SetInterruptMask() macro command executed equals 1).

This interrupt can be used:

- To warn the host of a transition into a new video sequence.
- At the beginning of decode of the first sequence.
- Any time that the decode parameters may have changed.

When a sequence_header_code is decoded, the microapplication writes the sequence parameters in DRAM with the values contained in the bitstream unless SeqWP is 1. (See Section 14.1.1 on page 14-2, Sequence Variable Group.)

Note: The SEQ-V interrupt is also caused by detection of a sequence_header_code, but is not issued until the VSYNC prior to the display of the first picture (in display order) found in the bitstream after the sequence_header_code. (See Section 12.1.3, VSYNC Interrupts, and SEQ-V on page 12-25.)
The SEQ-V interrupt (first I-picture display after `sequence_header_code`) event occurs at the rising edge of VSYNC prior to the first display field of the first I-picture decoded following the detection of a `sequence_header_code`. Each time a SEQ-V event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an SEQ-V interrupt or places an SEQ-V interrupt in the pending interrupt queue.

The microapplication uses an internal flag to produce the SEQ-V interrupt. The flag is set each time a `sequence_header_code` is decoded. This flag is cleared when the microapplication reaches the IDLE state, the `FlushBitstream()` command is executed, a `sequence_end_code` is decoded, and each time an SEQ-V interrupt event occurs.

An SEQ-V interrupt event occurs if all of the following are true:

- The timing for VSYNC interrupts is satisfied (see Section 12.1.3, VSYNC Interrupts).
- The picture to be displayed in the following field time is an I-picture which has not been previously displayed.
- The SEQ-V interrupt flag is set.
- The SEQ-V interrupt is enabled (`mask[3]` of the last `SetInterruptMask()` macro command executed equals 1).

*Note: The SEQ-D interrupt is also caused by detection of a `sequence_header_code`, but is issued as soon as the decode process detects it. (See Section 12.1.2, Decode-time Interrupts, and SEQ-D on page 12-25.)*
The UND interrupt (bitstream buffer underflow) event occurs when the number of valid bytes in the bitstream buffer in DRAM is 0. This interrupt can be used by the host to detect a bitstream buffer underflow error condition. Each time a UND event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues a UND interrupt or places a UND interrupt in the pending interrupt queue.

A UND interrupt event occurs if all of the following are true:

- The number of valid words in the bitstream buffer is 0, and a new word is needed by the decoder process.
- The UND interrupt is enabled (mask[8] of the last SetInterruptMask() macro command executed equals 1).

*Note: The UND interrupt is sensitive to the buffer’s emptiness being 0, not becoming 0. This means that, once a UND interrupt is produced, new UND interrupts will be produced continuously until the host takes steps to make the bitstream buffer non-empty.*

Unlike the RDY interrupt, the UND interrupt does not depend on the NewPacket() command, and production of the UND interrupt cannot be suppressed by issuing NewPacket() commands. UND interrupt production can only be stopped by (1) transferring new data to the CL450 or (2) disabling the UND interrupt. And, as described in Section 15.2.3, During Bitstream Underflow, the microapplication is unlikely to execute a SetInterruptMask() macro command if the bitstream buffer is empty since low-priority macro commands may only be executed after an entire picture has completed decoding.

This section contains two examples of how the CL450’s interrupts behave in relation to one another and the host’s actions. Each example
This section contains two examples of how the CL450’s interrupts behave in relation to one another and the host’s actions. Each example consists of introductory text explaining the cases shown, a numbered list of events which occur in the example, and a corresponding figure illustrating how those events relate to each other in time sequence.

While reading the examples, remember that the situations and the example host actions shown have been chosen to best illustrate the behavior of the CL450. These situations are not necessarily typical, and the host’s responses and actions may not be optimal in an actual system.

The examples show some, but not necessarily all, of the macro commands which the host issues during the period shown. In general, only those macro commands whose timing is important to the interrupts in the example are shown. For example, in a typical system the host would be issuing many more NewPacket() commands than are shown, and there are several other commands (SetBorder, SetWindow, InquireBufferFullness, etc.) which would not substantively affect the examples if the host issued them at times not explicitly shown. However, all macro commands which can affect the microapplication’s command processing state or the interrupts which are enabled are explicitly shown.

The figures in each example are arranged similar to timing diagrams, showing the timing relationship between changes in the CL450’s state and output. All of the figures show the same state elements, although not all elements affect the outcome of each example.

Figure 12-8 is a key to some of the items in the example figures. In particular, miniature versions of the display-field picture from Figure 12-2 are used to represent the CL450’s video output. However, because the example figures are organized with time passing from left to right, the video display has been rotated so the top of the display field appears on the left and scan lines run bottom-to-top within the field.
12.4.1 Interrupt Example 1: RDY, UND, ERR

This example shows how the RDY interrupt is produced, how a bitstream data error causes the ERR interrupt and triggers the microapplication’s error concealment, and how this in turn can cause both the RDY and UND interrupts.

At the beginning of this example, the CL450 is in the middle of decoding and displaying a bitstream and only the RDY, UND, and ERR interrupts are enabled. (see Figure 12-9).

1. The bitstream buffer’s fullness increases until the host pauses the incoming data supply, which happens after the host sends all the data for all the NewPacket() commands that have been issued. This pause satisfies one of the requirements for the creation of the RDY interrupt.

2. As the microapplication continues to decode, the number of empty bytes exceeds the threshold value. However, because the microapplication polls the fullness level according to the constraints for Display-time interrupts, the RDY interrupt is not yet produced.
Interrupt Examples

Figure 12-9 Interrupt Example 1: RDY, UND, ERR
3. After the first block-row from the picture I\textsubscript{15} is displayed, the microapplication recomputes the buffer fullness, and discovers that there are more empty bytes than \textit{threshold}. Because the other interrupt requirements have also been satisfied, the microapplication recognizes an RDY interrupt event. Because HMEM[0xa] (Interrupt Status) is 0, the RDY interrupt can be issued to the host. Therefore the microapplication makes the INT pin active and sets HMEM[0xa][10] equal to 1. No other bits are set because no other interrupts are pending.

4. The host receives the interrupt and begins its interrupt service process by deactivating the INT pin. While the host may now be able to receive more interrupts, the microapplication cannot issue another interrupt because Interrupt Status is still non-zero.

5. For whatever reason, the host does not issue a NewPacket() command to begin transfer of more data before the end of display of the second block row of I\textsubscript{15} occurs. Therefore, the conditions necessary for the RDY interrupt all become true again, so a second RDY interrupt event occurs. Because the host has also not yet cleared Interrupt Status, the RDY interrupt cannot be issued (even though INT is inactive) and is queued internally instead.

6. The host then issues a NewPacket() command. While this does not directly add more data to the bitstream buffer, it does make one of the conditions required for the RDY interrupt false.

7. The host then clears Interrupt Status. The host must do this after issuing the NewPacket() command if a new RDY interrupt can be issued between these two actions.

8. Finally, coded data begins to be received by the CL450 again and the bitstream buffer fullness begins to rise.

9. When the end of display of the third block row of I\textsubscript{15} occurs, even though the number of empty bytes in the bitstream buffer is less than \textit{threshold}, the microapplication issues a new (second) RDY interrupt to the host. This interrupt is issued because the internal interrupt queue is examined each time a Display-time interrupt could be produced, and because there is a queued RDY interrupt from the end of the previous block row. This interrupt would not have been queued if the host had responded more quickly to the original RDY interrupt.
10. The host receives the interrupt and determines that it is a remnant from the previous RDY interrupt (possibly by examining Buffer Fullness Status). Therefore, it deactivates the INT pin and clears Interrupt Status. Note that these operations should be done in that order because the microapplication will not modify the state of the INT pin unless Interrupt Status is 0.

11. The decode process discovers an error in the bitstream while decoding picture B_{16}. This error causes the microapplication to perform two actions. First, the microapplication attempts error recovery (see Section 15.2, Error Recovery and Concealment). Second, because the ERR interrupt is enabled, a ERR interrupt event occurs; and because Interrupt Status is 0, the interrupt is issued directly to the host, activating the INT pin and writing Interrupt Status with 1.

12. The host receives the ERR interrupt. In response, the INT pin is made inactive and Interrupt Status is cleared. The host also sets an internal flag because it will have to interpret future RDY and UND interrupts differently than if the microapplication were not performing error recovery.

13. The host issues and the CL450 accepts a SetInterruptMask() macro command. This command is issued with a mask argument of 0x503, leaving RDY, UND and ERR enabled, and adding PIC-V. The host enables the PIC-V interrupt so that it will be informed when the first new picture is displayed, which roughly corresponds to the end of error recovery. Note that, because SetInterruptMask() is a low priority command, the microapplication does not execute it until item number 18, below.

14. The next time the timing requirements for Display-time interrupts are satisfied (in this example, the end of the first block row of the third field time in which picture I_{15} is displayed), the microapplication issues both a UND and an RDY interrupt (assuming that an adequate number of bitstream words has been transferred to satisfy the RDY condition).

15. The host receives the interrupts from the CL450. It ignores the UND interrupt, because it knows that the microapplication is in error recovery. It uses the RDY interrupt as it normally would. The INT pin is deactivated and Interrupt Status cleared.
16. In this region, the microapplication will issue UND and/or RDY interrupts for every block row displayed. The host should continue to ignore the UND interrupts, and supply data as quickly as possible. Some or all of the RDY interrupts may be suppressed by issuing NewPacket() commands one or more in advance of sending data to the CL450.

17. The decode process finds a picture_start_code for a picture that it has the information to decode. At this point, it resumes normal decoding operation.

18. Microcode executes the SetInterruptMask() command issued at item number 13, above. It is only at this point that the PIC-V interrupt is actually enabled.

19. The VSYNC prior to the first display of the newly-decoded picture P[21] occurs. This creates a PIC-V interrupt event. The PIC-V interrupt is immediately issued to the host because (in this example) Interrupt Status is 0 when VSYNC occurs. The microapplication therefore activates INT and sets Interrupt Status[1].

20. The host receives the PIC-V interrupt. It clears Interrupt Status and deactivates the INT pin. In addition, the host clears its internal flag, indicating that the microapplication is performing error recovery. Subsequently, if the host receives a UND interrupt, it will be indicative of a system error condition. The host may also subsequently issue another SetInterruptMask() macro command to disable the PIC-V interrupt if the host has no other use for it.
12.4.2 Interrupt Example 2: PIC-V, SCN

This example shows how the PIC-V interrupt is produced, how the Scan() macro command might be used, and how the SCN interrupt is produced.

At the beginning of this example, the CL450 is in the middle of decoding and displaying a bitstream in the PLAY state, and only the PIC-V and SCN interrupts are enabled. (See Figure 12-10.)

1. The active edge of VSYNC preceding the first display of picture B_{15} occurs. This causes a PIC-V interrupt event and, because Interrupt Status was previously 0, the microapplication issues the interrupt immediately. Therefore, the INT pin is activated and Interrupt Status[1] is set.

2. The host issues and the CL450 accepts the Scan() command.

3. When the decoding for picture B_{15} completes, the microapplication processes accumulated low-priority macro commands. The previously-accepted Scan() command is executed, causing the microapplication to transition from the PLAY state to the SCAN state.

4. The host finally receives and processes the PIC-V interrupt issued in item 1. The host deactivates INT and clears Interrupt Status.

5. The microapplication finishes its internal SCAN initialization and begins to process the portion of the bitstream ahead of the I-picture at a very high rate. MPEG header information is still decoded and processed and only P- and B-pictures are discarded completely.

6. When the decoding process encounters a picture_start_code for an I-picture (in this example, I_{38}), it stops discarding bitstream data and begins normal picture decoding.

7. When the decoding of I_{38} is complete, the microapplication automatically transitions to the PAUSE state (and stops consuming bitstream data). In addition, the completion of decode causes an SCN interrupt event. Because Interrupt Status is 0, the interrupt is issued immediately and the INT pin is activated and Interrupt Status[11] set.
Interrupt Examples

Figure 12-10  Interrupt Example 2: PIC-V, SCN
8. The host then receives the SCN interrupt. In response, the host’s interrupt handler sends a message to a portion of non-interrupt software (which will respond later), deactivates the \texttt{INT} pin, and clears Interrupt Status.

9. After the interrupt handler exits, the host issues a \texttt{FlushBitstream()} command, which the microapplication immediately executes because it is high priority. This macro command discards all of the bitstream data (and associated NewPacket() macro commands) which followed the I-picture and have been transmitted to the CL450 since the decoding of I_{38} completed (while the microapplication was in the \texttt{PAUSE} state). After this command has executed, the host begins transmitting bitstream information and NewPacket() macro commands in preparation for the next Scan() command.

10. When the next VSYNC occurs, the microapplication begins the display of I_{38}. Because this is the first display field of a new picture, it also causes a PIC-V interrupt event. The PIC-V interrupt is issued immediately (because Interrupt Status is 0), so the microapplication activates \texttt{INT} and sets Interrupt Status[1].

11. The host (continuing the non-interrupt processing which began in item # 9) then issues a new Scan() command, which the microapplication accepts and places in the Command FIFO.

12. Because the microapplication is in the \texttt{PAUSE} state and no other low priority macro commands are pending, the Scan() command issued in item #11 is executed almost immediately. This command causes the microapplication to transition back to the SCAN state and begin the rapid consumption of bitstream data.

13. The host finally receives the PIC-V interrupt (issued in item #10) which indicates that display of I_{38} has begun. In response, the \texttt{INT} pin is deactivated and Interrupt Status cleared.

14. When the decoding process encounters a \texttt{picture-start_code} for an I-picture (in this case I_{61}), it stops discarding bitstream data and begins normal picture decoding.

15. When decoding of I_{61} is complete, the microapplication transitions back to the \texttt{PAUSE} state and an SCN interrupt event occurs. The interrupt is issued immediately; \texttt{INT} is activated and Interrupt Status[11] is set to 1.
16. The host receives the SCN interrupt and begins its interrupt processing by deactivating INT (rather than delaying processing as in item #8).

17. While the microapplication still thinks the SCN interrupt is pending (Interrupt Status[11] = 1 even though the host has deactivated INT), the active edge of VSYNC occurs which begins the first field display of I_{61}. This edge causes a PIC-V interrupt event. However, because Interrupt Status is non-zero, the interrupt is queued within the CL450 to be issued in the future.

18. The host finally finishes its interrupt processing and clears Interrupt Status.

19. At the end of display of the first block row from I_{61}, the microapplication checks for Display-time interrupts. Although RDY is not enabled, the bitstream buffer fullness is still recomputed (and written to Buffer Fullness Status, HMEM[0xb]), and the microapplication checks for queued interrupts. The PIC-V interrupt event which occurred in item #17 is in the queue, and since Interrupt Status is now 0, the PIC-V interrupt is issued: INT is activated and Interrupt Status[2] set.

20. The host receives the PIC-V interrupt, deactivates INT, and clears Interrupt Status.

21. Finally, the host issues FlushBitstream() to clean out any data in the bitstream buffer which was not consumed while I_{61} was being decoded or which was received since the decoding of I_{61} completed.

The following are the sequences for servicing non-vectored and vectored interrupts. Note that these sequences assume that the host processes all CL450 logical interrupts simultaneously if they were received with the same physical interrupt. For other service mechanisms, see Section 12.2.2.

The sequence for servicing a non-vectored interrupt is:

1. The CL450 asserts external signal INT low by setting Int to 0 in the HOST_control register.

2. The host reads the interrupt type from the Interrupt Status register.
3. The host executes an interrupt service routine (ISR) located at a pre-determined host internal address.
4. The host sets the \( \overline{Int} \) bit to 1 in the HOST_control register.
5. The host writes 0 to the Interrupt Status register.

The sequence for servicing a vectored interrupt is:

1. The CL450 asserts external signal \( \overline{INT} \) low by setting \( \overline{int} \) to 0 in the HOST_control register.
2. The host acknowledges the interrupt by asserting \( \overline{INTACK} \). If the \( \text{Ale} \) bit in HOST_control is 1, the CL450 sets \( \overline{int} \) to 1 in the HOST_control register.
3. The host broadcasts the interrupt priority ID (IPID) on \( A[3:1] \).
4. If the IPID on \( A[3:1] \) matches the value of \( IPID \) in the HOST_intvecr register, the CL450 responds by driving the interrupt vector \( I Vect \) onto \( D[7:0] \).
5. The host executes the interrupt service routine (ISR) pointed to by the interrupt vector. As part of the ISR, the host reads the interrupt type from the Interrupt Status location in HMEM.
6. If the \( \overline{Int} \) bit was not set in item #3 above, then the host sets \( \overline{Int} \) to 1 in the HOST_control register.
7. The host writes 0 to the Interrupt Status register.
The MPEG standard includes information and restrictions intended to allow elementary audio and video decoders to be synchronized to:

- Each other (so that audio data is played when the appropriate pictures are on the video display and vice versa)
- The bitstream source (to prevent bitstream buffer underflows or overflows in both decoders)

MPEG synchronization is primarily based on the following two pieces of information which are present in the system level of an MPEG bitstream:

- *System clock references (SCRs)*, which serve as a time base for elementary decoders.
- *Presentation time stamps (PTSs)*, which indicate the time relative to the SCR at which a picture should be displayed on the screen.
The CL450 provides support for each of these MPEG constructs, respectively, through use of its on-chip SCR (System Clock Reference) counter, and its NewPacket() and AccessSCR() macro commands.

13.2.1 CL450 SCR Counter
The CL450 implements an on-chip 33-bit counter which it uses as a local copy of the system clock (see Section 13.3). This counter must be initialized by the host, after which the CL450 will increment it at a nominal frequency of 90 KHz. The host may also periodically read or write this counter to keep it in step with the decoder global SCR, if any, using the AccessSCR() macro command.

It is recommended that the host update the CL450’s SCR (if the CL450 is not the SCR master for the system) at least every 0.7 seconds, which is the minimum rate at which the MPEG standard requires that SCR update values be placed in a system bitstream.

13.2.2 CL450 Presentation Time Stamps
The CL450 receives PTS information from the host in the three timeS­ tamp arguments of the NewPacket() macro command. Each time the host issues a NewPacket() command, it may place a PTS in the timeS­ tamp arguments. Each PTS received by the CL450 is then associated with a picture_start_code in the corresponding packet according to the requirements given in the MPEG standard.

PTS values given to the CL450 may be created by the host or extracted from the packet headers in the system bitstream. All PTS values present in an MPEG system stream do not have to be passed to the CL450, but for smooth synchronization it is recommended that valid PTS values be given to the CL450 at least once every 0.7 seconds.

Figure 13-1 illustrates the association between the parameters of the NewPacket() command and picture_start_codes within the bit­ stream. Note that the words “no PTS” are used to indicate the timeS­ tamp arguments to NewPacket() when the Vld bit (timeStamp2[15]) is 0, and “PTS” is used to represent the time­ Stamp arguments if Vld is 1.
13.2.3 CL450 Synchronization

The CL450 performs synchronization as follows: Each time a picture_start_code is detected in the bitstream, the microapplication determines if there is a valid PTS associated with that start code. There are three possible outcomes:

- No valid PTSs have been received from the host since the last time the microapplication was in the IDLE state, the FlushBitstream() command was executed, or a data error was detected. In this case, the microapplication takes no further action for synchronization and behaves as described in Section 13.5.3 on page 13-10 (Unpacketed/Packeted Data Transfer without PTSs).

- This picture does not have a valid PTS, either because none was supplied in the associated NewPacket() command or because this is not the first picture_start_code in this packet. In this case, the microapplication synthesizes a PTS for this picture based on the PTS for the previous picture and the current picture_rate parameter.

- This picture has a valid PTS which was supplied by the host via the NewPacket() command.
Once the microapplication has a PTS for the picture, it samples the value of the on-chip SCR counter. The value read is then biased (because this operation is being performed before the picture is decoded rather than at the instant of "presentation") and compared with the picture's PTS. If the PTS and the biased SCR match to within ±3000 SCR counts, then the picture is decoded and posted for display.

If the PTS is more than 3000 counts greater than the SCR, then the CL450 has been decoding too quickly or there has been a discontinuity in the PTS sequence. In either of these cases, the microapplication posts the last picture displayed for display again, waits a frame time, and then repeats the synchronization operation. In this manner, pictures are repeated as many times as necessary to allow the SCR to catch up to the decoding and display processes.

Alternately, if the PTS is more than 3000 counts less than the SCR, then the CL450 has been decoding too slowly or there has been a discontinuity in the bitstream. In either of these cases—if and only if the next picture to be decoded is a B-picture—the CL450 discards the picture about to be decoded and advances to the next portion of the bitstream.

Skipping the decode (and therefore the display) of a picture allows the CL450's decode process to catch up to the "actual time" as defined by the SCR counter. Only B-pictures are skipped to ensure that the CL450 always has the necessary reference frames decoded for normal decoding to continue following the skip. If the next picture to be decoded is an I- or P-picture, the PTS/SCR mismatch is ignored and will be resolved the next time a B-picture is encountered.

The host may initialize or update the value in the CL450's SCR counter by issuing the AccessSCR() macro command (see page 11-13). (The AccessSCR() command may also be used to read the current state of the SCR counter.) The counter is updated in the interval between when the host sets HOST_newcmd[0] and when the microapplication clears HOST_newcmd[0].

### 13.3.1 SCR Source and Drift

The SCR counter is incremented based on the CL450's GCLK input, which is nominally 40 MHz. To generate the 90-KHz frequency at
which the SCR should be incremented, the CL450 divides GCLK by 444, which produces a frequency of 90,090 Hz. Because this frequency is not exactly 90 KHz, the CL450's SCR counter will run fast by roughly 0.1% relative to an ideal SCR. Not counting any inaccuracies in the GCLK frequency, this difference causes a drift of +63 SCR counts in 0.7 seconds. Since typical applications update the SCR counter at least every 0.7 seconds, and 63 is much less than the built-in jitter tolerance of 3000 SCR counts, this difference has no effect on the CL450's synchronization.

**13.3.2 Automatic SCR Modifications**

Immediately before the microapplication transitions from the STILL command state to the PAUSE state upon completing the decode of a still picture (see page 11-15), the microapplication clears the SCR counter to 0. The host should account for this and initialize the CL450 with the correct SCR before issuing a Play() command (which will resume decoding with synchronization).

Systems that perform audio/video synchronization must have one master time-base within the decoder (the “real” SCR) to which local SCRs, such as the SCR counter within the CL450, are locked. The following sections describe several alternatives for the master SCR.

**13.4.1 Synchronizing to the Bitstream**

The SCR values which are embedded in the pack headers in an MPEG system bitstream are intended to indicate the time (relative to the decoder’s System Clock Reference) at which the SCR fields are received by the system-level decoder. In systems where the bitstream is delivered to the decoder with the timing indicated by the SCR fields (either because the bitstream is coming from a fixed bit-rate source or because some other system element is explicitly timing the delivery of MPEG packs), the SCR values can be used to update all of the SCR counters in the decoder.

*Note: The host, which serves as the system-level decoder and demultiplexer in systems containing the CL450, must minimize the timing skew introduced when the*
CL450's SCR and any other SCRs in the system are updated.

13.4.2 Synchronizing to the Audio Decoder
The CL450 (and the system-level decoder) can also be synchronized to the audio decoder. If the audio decoder contains its own SCR counter, then this counter can be initialized based on the first SCR contained in the bitstream. Subsequently, the audio decoder will synchronize its own operation to its SCR, and the host can update its SCR (if any) and the CL450’s SCR with the contents of the audio decoder’s SCR. Again, it is important that a minimal amount of SCR skew be introduced when updating the CL450.

Alternately, if the audio decoder does not perform synchronization and/or does not contain an SCR counter, SCR updates may be synthesized by the host based on the amount of audio data consumed by the audio decoder. Because MPEG-coded audio data is decoded at a fixed bit rate, the number of bits of coded data consumed by the audio decoder can be used to accurately calculate the amount of time elapsed since audio decoding began (relative to the decoder’s clock source). This duration may be combined with an audio PTS to synthesize a value representing the System Clock Reference at the time the audio decoder’s data consumption was sampled.

These methods effectively lock the entire decoder subsystem to the audio decoder’s output clock so that any instability in that clock will be reflected in the audio and video outputs.

*Note: Locking the decoder subsystem to the audio clock may cause bitstream underflow or overflow if drift occurs with bitstream delivery relative to the decoder.*

13.4.3 Synchronizing from the CL450’s SCR
The CL450 can also be used as the master SCR for the decoder. Assuming that the audio decoder performs synchronization and accepts SCR updates, the CL450’s SCR can be initialized using data from the bitstream and then periodically sampled to provide updates for the host, if necessary, and the audio decoder.
Note: The CL450's SCR runs 0.1% fast (plus any instability in the GCLK source), so the host might wish to periodically adjust the CL450's SCR to correct this inaccuracy.

13.4.4 Synchronizing from the CL450 and VSYNC

The CL450 can be used as an SCR master without using the CL450’s internal synchronization mechanism. In this case, the NewPacket() commands sent to the CL450 by the host, if any, are never given validated PTS values. Instead, the host uses the time_code and temporal_reference information written into the CL450’s DRAM variable area when pictures are decoded to compute the amount of time which has elapsed since video decoding began. This timing information can then be combined with SCR or PTS information from the bitstream to create SCR updates for the host and/or audio decoder.

Two significant differences between this method and using the CL450’s SCR as the SCR master are:

- This method locks the decoder subsystem to the CL450’s VSYNC input via the CL450’s transcoding algorithm, rather than GCLK.
- If a bitstream error occurs which causes a large portion of the bitstream to be decoded, both the CL450 and the audio decoder will immediately jump forward in time together. In the alternate case where the CL450 is synchronizing to its SCR, the next picture from the bitstream which can be decoded would be delayed until the correct display time, as specified by the PTS.

Data may be transferred to the CL450 either with or without the host issuing the NewPacket() macro command. However, most systems use NewPacket() because of the microapplication features (especially audio/video synchronization) which are available only if the NewPacket() command is used.

The host must decide whether or not the NewPacket() command will be used before the first word of coded data is transmitted to the CL450. Typically, the engineer writing the software controlling the host will make this decision based on the type of application and bitstreams to be decoded.
However, there may be systems which will dynamically change between decoding bitstreams with and without using NewPacket(). For these systems, it is critical to remember that the microapplication is modal (when not in the IDLE state); it assumes that if it receives one NewPacket() command, NewPacket()’s are associated with all coded data.

13.5.1 Unpacketed Data Transfer
MPEG coded data can be transferred to the CL450, correctly decoded, and be displayed without using the NewPacket() macro command. However, as stated in the description of this command (page 11-21), some microapplication functions cannot be performed without the additional system-level information provided by the NewPacket() command.

13.5.2 Packeted Data Transfer
When NewPacket() is used, the microapplication associates each NewPacket() command with a section of the coded data stream. The command’s length parameter specifies the amount of coded data.

To associate data and NewPacket() commands, the microapplication simply starts counting coded data bytes and adding length parameters starting with the first word of coded data and the first NewPacket() command received, respectively. The NewPacket() command associated with a section of the bitstream must be accepted by the microapplication before the first word of data from that section is transmitted to the CL450.

When using NewPacket(), bitstream data must be associated with a NewPacket() command. However, the host may issue any number of NewPacket() commands ahead of the arrival of bitstream data at the CL450, provided that the Command FIFO does not overflow (see Section 11.2.2, Command FIFO).

Figure 13-2 shows an example of how NewPacket() commands and bitstream data could be sent to the CL450. Note that the times marked as NewPacket() commands are the times when the CL450 accepts the macro command (microcode clears HOST_newcmd[0] to 0), not the earlier time when the host issued the macro command (host sets HOST_newcmd[0] to 1).
Items are represented in the figure as follows:

- The horizontal dimension is the time axis.
- The bitstream data is divided into numbered sections to show the portions which correspond to the similarly-numbered NewPacket() commands. Note that the horizontal size of a marked section of bitstream data has nothing to do with the size of that section in bytes but instead corresponds to the time within which it is transmitted to the CL450.
- Stars indicate the acceptance of NewPacket() commands by the microapplication.
- Arrows indicate the amount of time between the acceptance of a NewPacket() command and the reception of the associated data.

![Bitstream Data Transmission Diagram](image)

**Figure 13-2** Bitstream/NewPacket() Association
13.5.3 Unpacketed/Packeted Data Transfer without PTSs
When unpacketed data is sent to the CL450 or when NewPacket() is used but the Vld bit in timeStamp2 is never 1, the microapplication does not have information to perform audio/video synchronization. In this case, the rate at which pictures are displayed (and therefore bitstream data consumed) is controlled solely by the nominal VSYNC frequency specified by the host (see SetVideoFormat() on page 11-40) and by the current picture_rate parameter, as described in Section 15.1.

In general, if the VSYNC frequency is stable and the bitstream has been properly encoded and multiplexed and no bitstream errors are encountered, very little drift occurs between the CL450’s video display and the output of an audio decoder operating from the same bitstream (depending on the transcoding case). However, there are several important reasons why it is recommended that audio/video synchronization be performed:

- If a bitstream error occurs, the CL450 will probably skip decoding one or more pictures. If synchronization is not being used, the decoding and display processes will jump ahead by an amount of time equal to the number of pictures skipped (potentially an entire GOP even for a single-bit error) times the nominal frame rate. This is sufficient to destroy “lip sync” with an associated audio decoder as well as potentially causing buffer fullness problems depending on how the system bitstream is demultiplexed and transferred to the CL450.
- The quality achieved by letting both the CL450 and the audio decoder run independently is dependent on the stability of audio decoder timing.
- Either the CL450 or the audio decoder could suffer from bitstream buffer underflow or overflow even if no errors occur.

13.5.4 Packeted Data Transfer with one PTS
The microapplication treats specially the first NewPacket() with a valid PTS (Vld bit equal to 1). Until the first picture with an associated valid PTS is decoded, the microapplication does not perform audio/video synchronization and behaves as described above. Once a valid PTS is received, the microapplication assumes that the SCR counter contains a valid value.
In addition, the microapplication will subsequently synthesize PTS values for pictures which do not have PTSs supplied by the host, based on the last host-supplied PTS value.

If the host supplies only a single valid PTS, then all pictures after the picture corresponding to the valid PTS will still be synchronized. Because the host does not provide any additional PTS values, the microapplication synthesizes PTSs for the rest of the pictures, causing them to be displayed with the exact timing specified by the picture_rate parameter.

This approach is recommended for systems in which elementary video streams are being played (so the host does not have a system stream to retrieve PTS values from) unless the host wants the improved error recovery behavior that synchronization can provide. Note that the "synchronized" display rate will be 0.1% fast unless the host periodically updates the SCR counter.

13.5.5 Packeted Data with Periodic PTSs
This is the typical use for audio/video synchronization, and it occurs in two basic steps:

1. PTS values are extracted from the system bitstream by the host as part of the system demultiplexing function and given to the microapplication as arguments to the NewPacket() command.

2. The bitstream is reassembled without adding or deleting any bytes, and the reassembled coded data is transferred to the CL450 as 16-bit words.

Generally, the SCR counter is periodically updated to prevent the CL450 from synchronizing to an SCR value which is divergent from the system SCR, and the host must still ensure that the SCR counter is initialized before synchronization begins.

It is also important when demultiplexing an MPEG system bitstream to create CL450 NewPacket() commands to provide for the fact that (1) MPEG packets may be an odd number of bytes, and (2) the length argument to the NewPacket() command must be an even number of bytes. To do this, system demultiplexers may need to adjust packet boundaries, logically moving one byte of data between two adjacent packets. This
adjustment affects the length parameters of the two successive New-Packet() commands but not the demultiplexed bitstream data.

Note: It is particularly important when reassigning a byte from one packet to the next or previous packet to ensure that, if that byte is the first byte of a picture_start_code with an associated PTS, the PTS is moved to the correct packet as well.

How this problem is resolved depends on the particular application area. For some applications, system-level packets are guaranteed to be an even number of bytes, eliminating the problem altogether. In other applications, the frequency of video PTS values in the system bitstream is significantly higher than minimum, or slightly less quality is required in audio/video synchronization. In these cases, the PTS values associated with modified-length packets can simply be discarded and not sent to the CL450.
The host may access several variable locations in DRAM while the microapplication is executing. These areas include:

- **Scratch storage**: 16 words of DRAM are allocated exclusively for this purpose. The host may read and write these locations at any time, provided that the restrictions in this section are met.

- **Semaphores for host/microcode communication**

- **Bitstream parameters**

The precise organization of these locations is given in Appendix A, CL450 DRAM-variable Allocation.

The DRAM locations containing bitstream parameters are divided into the following two groups, each of which uses a semaphore to arbitrate between the CL450 and the host for access to the variables:

- **The Sequence Variable Group**: Contains information from the **sequence_header** of an MPEG bitstream and is accessed using the **SEQ_SEM** semaphore location.
The Picture Variable Group: Contains information from the GOP and picture headers and is accessed using the PIC_SEM semaphore.

14.1.1 Sequence Variable Group
The sequence group contains the variables shown in Figure 14-1. Note that each entry occupies a separate word address, and that these addresses are specified in Appendix A.

All bits within the 16-bit DRAM words which do not correspond to a bit within the sequence header are 0 and must be written to 0 if these values are written by the host (see Section 14.2.2).

The microapplication can write to all the locations in the sequence variable group. However, writes to locations containing bitstream information (locations other than SEQ_SEM and SEQ_CONTROL) are controlled by the SeqWP and SeqNoDef bits of SEQ_CONTROL. The microapplication sets NewSeq (in SEQ_CONTROL) each time a sequence_header_code or sequence_end_code is decoded, and clears NewSeq each time new information is read and digested (triggered by decoding a group_start_code).

Note: When the microapplication writes to SEQ_CONTROL, the values of SeqWP and SeqNoDef will be retained. The host should never clear NewSeq.
SEQ_SEM  /* semaphore controlling access to variable group */
SEQ_CONTROL  /* bit 0 NewSeq: new sequence information - need
*   update
* bit 1 SeqWP: write-protect sequence information
*   from bitstream
* bit 2 SeqNoDef: write-protect sequence information
*   from default restoration
*/

HORIZONTAL_SIZE
VERTICAL_SIZE
PICTURE_RATE
FLAGS  /* bit 0 == load_intra_quantizer_matrix
* bit 1 == load_non_intra_quantizer_matrix */
INTRA_Q[64]  /* 64-element intra_quantizer_matrix */
NON_INTRA_Q[64]  /* 64-element non_intra_quantizer_matrix */

Figure 14-1  DRAM-Resident Sequence Variable Group Variables

Also note that the the elements of the INTRA_Q and NON_INTRA_Q matrices are not stored in DRAM in the same order they are stored in the bitstream. In particular:

- Each matrix entry occupies a 16-bit word, not a byte, and the most-significant byte for all entries must be 0.
- Matrix entries are stored in row-major order, not in zig-zag order.

Figure 14-2 below shows the word-address offset from the beginning of each matrix for each matrix element.

+0x0  +0x1  +0x2  +0x3  +0x4  +0x5  +0x6  +0x7
+0x8  +0x9  +0xa  +0xb  +0xc  +0xd  +0xe  +0xf
+0x10 +0x11 +0x12 +0x13 +0x14 +0x15 +0x16 +0x17
+0x18 +0x19 +0x1a +0x1b +0x1c +0x1d +0x1e +0x1f
+0x20 +0x21 +0x22 +0x23 +0x24 +0x25 +0x26 +0x27
+0x28 +0x29 +0x2a +0x2b +0x2c +0x2d +0x2e +0x2f
+0x30 +0x31 +0x32 +0x33 +0x34 +0x35 +0x36 +0x37
+0x38 +0x39 +0x3a +0x3b +0x3c +0x3d +0x3e +0x3f

Figure 14-2  INTRA_Q and NON_INTRA_Q Matrix Ordering
The contents of the sequence variable group are invalid from the time the microapplication is initialized or the Reset() command is executed until the microapplication leaves the PLAY-SETUP state.

When initialization or Reset() is complete, SEQ_SEM and SEQ_CONTROL are both 0. Once initialized, the host may allocate SEQ_SEM and write to the sequence variable group at any time.

While in the PLAY-SETUP state, the microapplication writes default values to the sequence variable group and sets the NewSeq bit, unless the SeqNoDef bit is 1.

The contents of INTRA_Q and NON_INTRA_Q are invalid any time the corresponding bits of FLAGS are 0.

14.1.2 Picture Variable Group
The picture group contains the variables listed in Figure 14-3, which includes information from both GOP and picture headers. Note that each entry occupies a separate word address and that each address is specified in Appendix A. All bits within the 16-bit DRAM words which do not correspond to a bit within the MPEG syntax are 0.

```
PIC_SEM     /* semaphore controlling access to variable group */
TIME_CODE_0  /* time_code: */
  * [10:6] time_code_hours
  * [5:0] time_code_minutes
  */
TIME_CODE_1  /* time_code: */
  * [5:0] time_code_pictures
  */
TEMPORAL_REFERENCE
```

**Figure 14-3  DRAM-Resident Picture Variable Group Variables**

*Note: The microapplication writes values into these locations at the time the picture and GOP headers are decoded.*
The contents of the picture variable group are invalid from the time the microapplication is initialized or the Reset() macro command is executed until the corresponding header information has been decoded.

Because neither the CL450 nor the host has a “test-and-set” method for accessing DRAM locations, both the host and the microapplication must use a method in which a value is written to a semaphore and then the location is polled to determine if the value written is replaced.

The semaphores for both variable groups are independent, and the host should allocate only one at a time. Each semaphore may be in one of three states, encoded according to Table 14-1.

<table>
<thead>
<tr>
<th>Semaphore Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unallocated; neither host nor microapplication may access variables, but either may attempt to allocate the semaphore</td>
</tr>
<tr>
<td>0x4242</td>
<td>Allocated to host; microapplication may not access variables</td>
</tr>
<tr>
<td>other</td>
<td>Allocated to microapplication; host may not access variables</td>
</tr>
</tbody>
</table>

When attempting to allocate the semaphore, the host must perform operations equivalent to the pseudocode shown in Figure 14-4.
Access Synchronization Using Semaphores

```c
#define HOST_CODE 0x4242
/* "semAddr" (the argument to the function defined below) is one of
 * the following: */
#define SEQ_SEM <see Appendix A>
#define PIC_SEM <see Appendix A>

void AllocateSemaphore(unsigned long semAddr)
{
    do {
        while (CL450DRAM[semAddr] != 0)
            /* make sure microcode doesn’t
             * already have semaphore
             * allocated */
        CL450DRAM[semAddr] = HOST_CODE; /* try to allocate semaphore for
             * host */
        wait 0.5\mu s or more;
    } while (CL450DRAM[semAddr] != HOST_CODE);
    /* try again if we didn’t win */
}

Figure 14-4 Allocate Semaphore Function Pseudocode

Once allocated, either semaphore may be released simply by writing a
0 to the correct DRAM location.

Note: The total time between allocating the semaphore and
releasing the semaphore should not exceed 100\mu s; other­
wise, too much CL450 bandwidth is wasted in the case where
the CL450 has to wait for the host to release the semaphore
before decoding can continue. Assuming the bitstream buffer
does not underflow while the CL450 is performing decoding
with a semaphore allocated, the CL450 will not hold a sema­
phore for longer than 5\mu s.
The sequence header information (accessed by SEQ_SEM) is accessed by the CL450 each time a sequence_header_code, group_start_code, or sequence_end_code is found in the bitstream. Similarly, the CL450 accesses the information controlled by PIC_SEM each time a group_start_code or picture_start_code is encountered in the bitstream.

If the host can guarantee that it will not access a group of variables at the same time the microapplication does, then the maximum time that the semaphore is allocated can be ignored. However, the host should never access the DRAM variables without allocating the corresponding semaphore.

### 14.2.1 Reading DRAM-Resident Variables

Both variable groups may be read by the host any time the appropriate semaphore has been allocated (provided that the timing restrictions in Section 14.1 are met), although there are times when the data in the variables will be stale or invalid. The procedure for reading from either group of variables is the same and is outlined in the following pseudocode:

```c
#define SEQ_SEM <see Appendix A>
#define PIC_SEM <see Appendix A>

/* “semaphore address” is either SEQ_SEM or PIC_SEM */
AllocateSemaphore(semaphore address); /* function in Figure 14-4 */
read from DRAM address(es) containing desired variable(s);
CL450DRAM[semaphore address] = 0; /* release semaphore to allow
* CL450 to access variables */
```

**Figure 14-5** DRAM Variable Read Pseudocode

### 14.2.2 Writing DRAM-Resident Variables

The host may write values to the sequence variable group in DRAM. This can be done to provide sequence-layer parameters to the picture decode process when random-accessing a bitstream in a location which does not contain a sequence header.
Timing Restrictions

Each time the microapplication decodes a group_start_code, the value of SEQ_CONTROL is read. If the NewSeq bit is set, the microapplication reads in the values from the sequence variable group and uses them to update internal variables which control the decoding and display processes.

Note that the NewSeq bit is also used internally by the microapplication and is automatically set when new values are decoded from the bitstream (a sequence_header_code is decoded and SeqWP is 0) or when the default values have been restored (a sequence_end_code is decoded and SeqNoDef is 0).

The procedure for writing to the sequence variable group is shown in Figure 14-6.

```c
#define SEQ_SEM <see Appendix A>

AllocateSemaphore(SEQ_SEM); /* function in Figure 14-4 */
write to DRAM address(es) containing desired variable(s);
set SEQ_CONTROL[NewSeq];
CL450DRAM[SEQ_SEM] = 0;
/* release semaphore to allow */
/* CL450 to access variables */
```

Figure 14-6    DRAM Variable Write Pseudocode

Note that the new values written do not take effect until the microapplication decodes the next GOP header in the bitstream, at which point it observes that NewSeq is 1 and reads in the new sequence information.

14.3

Timing Restrictions

The microapplication does not support host access to DRAM locations other than those listed in Appendix A while microapplication is executing; attempting to access other locations will cause indeterminate results.

For accessible locations, the following is true:

- The maximum cycle time for host access to DRAM varies according to the current state of microcode execution.
- Some worst-case cycle times are too long to permit access to the CL450’s DRAM in some host environments.
There are three different maximum cycle times for host access to DRAM, depending on the operations the microcode is currently performing. Each of these maximum cycle times is determined by the maximum low time for the DTACK pin, which is used by the CL450 to extend access cycles.

The three maximum DTACK-low times and their corresponding operations are given in Table 14-2.

<table>
<thead>
<tr>
<th>Operating Mode</th>
<th>Maximum DTACK Pulse (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No microapplication executing (CPU_control[0] == 0)</td>
<td>TBD</td>
</tr>
<tr>
<td>Microapplication executing: output window blanked or in vertical border time</td>
<td>TBD</td>
</tr>
<tr>
<td>Microapplication executing: in vertical active display time</td>
<td>TBD</td>
</tr>
</tbody>
</table>
The CL450 microapplication uses two parameters to determine how many times each coded picture is displayed:

- *The display’s nominal field rate*: Supplied by the SetVideoFormat() macro command (see Chapter 11).

- *The coded picture rate (the sequence-layer parameter picture_rate)*: Supplied by default from the bitstream sequence header (see Table 10-4), or from the host via the DRAM variable PICTURE_RATE (see Figure 14-1).

Note that the display field rate for NTSC is treated as exactly 30 Hz. Similarly, picture_rates of 29.97 and 23.976 are treated as 30 Hz, and 24 Hz, respectively.

The CL450 compares both field rate and picture_rate parameters to each other, with the following three possible outcomes:

- When both the nominal field rate and the picture_rate are equal (30 Hz or 25 Hz), the display microapplication displays each
decoded picture for two field times (VSYNC periods). Note that audio/video synchronization continues to operate, possibly causing some pictures to be displayed for multiple frame times but always an even number of field times.

- When the picture_rate is higher than the nominal field rate (for example, a bitstream coded at 30 Hz displayed in a 25 Hz system), the display microapplication causes some frames to be displayed for only one field time rather than two.

- When the picture_rate is slower than the nominal field rate (for example, a 25-Hz bitstream display with a 60-Hz VSYNC frequency), the display microapplication causes some frames to be displayed for an extra field time (three fields per frame rather than two).

Field repeating and skipping is done at the required frequency for the nominal picture_rate to be correct. Note that, unlike synchronization, these rate conversion operations take place at the field level rather than the frame level.

For purposes of rate conversion, the CL450 microapplication treats 25 Hz (either input picture_rate or display frame rate) as 24 Hz. Because of this, bitstreams coded at 25 Hz and displayed at 30 Hz, or bitstreams coded at 24 Hz and displayed at 25 Hz, will play at a slightly incorrect speed. Table 15-1 indicates the cases in which the actual play speed does not match the nominal speed, and the nominal number of synchronization events per second if A/V synchronization is being performed.

<table>
<thead>
<tr>
<th>Format</th>
<th>picture_rate</th>
<th>Rate Error (%)</th>
<th>Average Synchronizations per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTSC (30 Hz)</td>
<td>25 Hz</td>
<td>4.2</td>
<td>Slow 1.25</td>
</tr>
<tr>
<td>PAL (25 Hz)</td>
<td>24 Hz</td>
<td>4.0</td>
<td>Fast 1.00</td>
</tr>
</tbody>
</table>

Conceptually, the synchronization and rate conversion processes operate entirely independently. The synchronization process decides how many times a particular picture should be displayed, and the pull-down process decides for each picture time how many fields that picture should be displayed.
should be repeated. Extra fields can be added to the second frame display of a picture as easily as the first, and if a picture is repeatedly displayed for enough frame times, it will have extra fields added to its display repeatedly.

*Note:* When performing this kind of rate conversion, whether the first field displayed for a picture is odd or even depends solely on what has occurred before and is very difficult to predict.

The CL450 handles signaled and unsignaled errors the same way. That is, when the CL450 detects an illegal (non-MPEG) construct (unsigned error) or a sequence_error_code (signaled error) in the bitstream, it assumes that a transmission or storage error has occurred and that a portion of the bitstream is corrupt. (A signaled error is denoted by a `sequence_error_code` hex value of 0x000001B4.)

In general, the CL450's error-recovery strategy assumes that a relatively small section of the bitstream has been corrupted. For example, if an error is detected while decoding a B-picture, the CL450 assumes that it may decode the next B-picture correctly. If, instead, the error resulted in an entire reference frame being lost, then all pictures decoded until an I-picture and a reference picture have been found and decoded will be in error. Similarly, if an error causes a sequence header to be lost, then subsequent decode operations may be conducted with incorrect dequantization matrices, resulting in incorrect images.

In practice, this error-recovery strategy allows the CL450 to resume decode and display as soon as possible. A strict algorithm which avoided all possibility of displaying an erroneous picture would very often have to wait until a new sequence header was found before resuming decode, which could be an arbitrarily long period.

### 15.2.1 During Picture Decode

If the error occurs while a picture is being decoded, the microapplication abandons that picture and begins scanning through the bitstream to find the next picture for which it has enough information to decode. Because of the forward and backward prediction linkages between pictures, the kind of picture searched for depends on the picture which was being decoded when the error was detected. For example:
If a B-picture was being decoded, then the microapplication resumes decoding and display with the next picture of any type found in the bitstream. Because B-pictures contain no information used for future decoding, decoding can resume with the next B-, P-, or I-picture found.

If a P- or an I-picture was being decoded when the error was detected, decoding does not resume until the next I-picture is found. Intervening P- and B-pictures are discarded because both kinds could contain motion vectors referring to the incompletely decoded reference picture.

While the decode process is searching for the next picture to decode, sequence and GOP headers are decoded normally if encountered.

15.2.2 During Header Decode
If an error is encountered while decoding header information, the CL450 behaves as described above for errors detected in B-pictures. When an error occurs in a header, the corresponding variable area in DRAM may be corrupted. In particular, it is impossible to determine which values are from the new, partially-decoded header and which are old. It is also impossible to tell if the actual error occurred significantly before it was detected.

15.2.3 During Bitstream Underflow
Because bitstream buffer underflow ordinarily results only from catastrophic failure at the system level, the CL450 microapplication does not necessarily behave as otherwise indicated if bitstream buffer underflow occurs. In general, any time that the microapplication is in a command processing state in which bitstream decoding occurs and the bitstream buffer underflows, microapplication execution stalls in the middle of the decode operation. The most noticeable result is that low-priority macro commands no longer are executed because the criteria for execution (given in Section 11.4, Command Latency) are based on the completion of picture decoding operations.

Functions implemented by the CL450’s internal interrupt handlers (including video display, Display-time interrupt production, macro command acceptance, and high-priority macro command execution) continue even if the bitstream buffer has underflowed.
Note: The host could create behavior similar to a bitstream buffer underflow by retaining control of the DRAM variables for longer than the maximum specified time (see Section 14.1, Types of Variables). This action blocks the decoding process and results in similar functional behavior.

The CL450 is designed to work over a particular set of input data and parameters. The contents of the rest of this document is true only within this operating domain. The most significant restrictions on the operating domain are given in this section.

15.3.1 Bitstream
The CL450 is intended for use with “constrained parameters” video bitstreams only. In general, only those bitstreams which have (or could have) the MPEG constrained_parameters_flag set are consistently decoded correctly. The only exceptions to this are:

- Still pictures meeting the requirements of the MPEG standard for bitstreams which can be decoded by an STD and which contain individual field pictures which do not exceed the requirements for a constrained parameters bitstream
- Bitstreams with coded data rates up to 5.0 Mbits/second but meeting all other requirements for the constrained_parameters_flag.

The bitstream’s picture_rate parameter must have a value between 0001_2 (23.976 Hz) and 0101_2 (30.0 Hz), inclusive.

The CL450 also cannot decode bitstreams consisting of D-pictures (DC intra-coded pictures, indicated by a picture_coding_type of 100_2).

15.3.2 Output Window and Timing
Because the CL450 microapplication begins the display of B-pictures before they have been completely decoded, there are some otherwise-legal combinations of coded picture size and SetBorder() and SetWindow() parameters which may produce incorrect visual results.
Typically, the microapplication will have decoded approximately 55% of the macroblocks in a B-picture before the first possible time a scan line from the picture can be displayed. This condition is shown in Figure 15-1, which assumes use of the tallest possible output window located at the top of the video display area (minimum topBorder). If the output window starts further down the display screen, a larger portion of the picture is decoded before display of the picture begins, assuming that display of the picture begins with the first scan line of the picture.

Any combination of SetWindow() and SetBorder() parameters which cause scan lines to be displayed at or below their position on the screen shown in Figure 15-1 will function correctly. However, combinations of windowing parameters which cause scan lines to be moved up relative to their default positions may cause visual glitches.

The host may compute the sections of the picture that will be completely decoded at a given time by assuming that:

- 45% of the macroblocks in the picture remain to be decoded at the end of the top border before a B-picture is displayed.
- Macroblocks are decoded at a constant rate through the remainder of the picture.
- All macroblocks are decoded 16 scan lines before the beginning of the bottom border.

For example, assume that an NTSC-resolution (15-macroblock high) picture is being decoded. In this case, 8.25 macroblock rows are completely decoded when the display of each B-picture begins (55% of 15). During picture display, macroblock rows are decoded at a rate of 0.03 macroblocks per scan line (15-8.25 / 240-16). Thus, when display scan line 150 occurs (counting from the top of the default output window), 12.75 macroblock rows will be decoded (150 x 0.03 + 8.25). Note that this number should always be truncated because only completely decoded macroblock rows can be displayed without glitches.
The method just described is a conservative way of determining (1) which scan lines are available for display, and (2) which macroblock rows will always display correctly (based on the macroblock rows which this method indicates are completely decoded). However, this method will also show that the bottom macroblock row of a picture can never be displayed above its default location on the screen. This result is too conservative for many applications, so experimentation with the actual system and bitstream are recommended.
Operating Restrictions

These restrictions are also present when decoding and displaying pictures with extreme aspect ratios (very tall and narrow, or very short and wide). In these cases, the default window configuration does not cause all of the picture to be displayed.

To calculate when portions of odd shaped pictures can be displayed, you can (1) assume that 55% of the picture is decoded before display begins, (2) compute the number of macroblocks per second (not scan lines) that are decoded in the NTSC case, and then (3) multiply this value by your HSYNC period to determine the number of macroblocks decoded per scan line. This should provide a conservative estimate of when macroblock rows from the picture may be displayed.
This appendix contains the DRAM addresses for each of the host-accessible DRAM variables, starting with the host scratch storage listed in Table A-1.

Table A-1  DRAM Addresses for Host Scratch Storage

<table>
<thead>
<tr>
<th>DRAM Address</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>HOST_WORD_0</td>
</tr>
<tr>
<td>0x1</td>
<td>HOST_WORD_1</td>
</tr>
<tr>
<td>0x2</td>
<td>HOST_WORD_2</td>
</tr>
<tr>
<td>0x3</td>
<td>HOST_WORD_3</td>
</tr>
<tr>
<td>0x4</td>
<td>HOST_WORD_4</td>
</tr>
<tr>
<td>0x5</td>
<td>HOST_WORD_5</td>
</tr>
<tr>
<td>0x6</td>
<td>HOST_WORD_6</td>
</tr>
<tr>
<td>0x7</td>
<td>HOST_WORD_7</td>
</tr>
<tr>
<td>0x8</td>
<td>HOST_WORD_8</td>
</tr>
</tbody>
</table>
Table A-2 lists the locations of the scalar variables in the sequence group.

<table>
<thead>
<tr>
<th>DRAM Address</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x9</td>
<td>HOST_WORD_9</td>
</tr>
<tr>
<td>0xa</td>
<td>HOST_WORD_A</td>
</tr>
<tr>
<td>0xb</td>
<td>HOST_WORD_B</td>
</tr>
<tr>
<td>0xc</td>
<td>HOST_WORD_C</td>
</tr>
<tr>
<td>0xfd</td>
<td>HOST_WORD_D</td>
</tr>
<tr>
<td>0xe</td>
<td>HOST_WORD_E</td>
</tr>
<tr>
<td>0xf</td>
<td>HOST_WORD_F</td>
</tr>
</tbody>
</table>

Table A-2  DRAM Addresses for Sequence Variable Group

<table>
<thead>
<tr>
<th>DRAM Address</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10</td>
<td>SEQ_SEM</td>
</tr>
<tr>
<td>0x11</td>
<td>SEQ_CONTROL</td>
</tr>
<tr>
<td>0x12</td>
<td>HORIZONTAL_SIZE</td>
</tr>
<tr>
<td>0x13</td>
<td>VERTICAL_SIZE</td>
</tr>
<tr>
<td>0x14</td>
<td>PICTURE_RATE</td>
</tr>
<tr>
<td>0x15</td>
<td>FLAGS</td>
</tr>
</tbody>
</table>

In Table A-3 and Table A-4, the INTRA_Q and NON_INTRA_Q matrix element addresses are shown arranged in the same physical order as the corresponding default matrices in the MPEG standard (given with the definitions of the load_intra_quantizer_matrix and load_non_intra_quantizer_matrixflags). Matrix element (0,0) is located in the upper left, and the matrices are stored in row-major format within DRAM.
Table A-3  DRAM Addresses for INTRA_Q Matrix

<table>
<thead>
<tr>
<th>DRAM Address</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ox20</td>
<td></td>
</tr>
<tr>
<td>Ox21</td>
<td></td>
</tr>
<tr>
<td>Ox22</td>
<td></td>
</tr>
<tr>
<td>Ox23</td>
<td></td>
</tr>
<tr>
<td>Ox24</td>
<td></td>
</tr>
<tr>
<td>Ox25</td>
<td></td>
</tr>
<tr>
<td>Ox26</td>
<td></td>
</tr>
<tr>
<td>Ox27</td>
<td></td>
</tr>
<tr>
<td>Ox28</td>
<td></td>
</tr>
<tr>
<td>Ox29</td>
<td></td>
</tr>
<tr>
<td>Ox2a</td>
<td></td>
</tr>
<tr>
<td>Ox2b</td>
<td></td>
</tr>
<tr>
<td>Ox2c</td>
<td></td>
</tr>
<tr>
<td>Ox2d</td>
<td></td>
</tr>
<tr>
<td>Ox2e</td>
<td></td>
</tr>
<tr>
<td>Ox2f</td>
<td></td>
</tr>
<tr>
<td>Ox30</td>
<td></td>
</tr>
<tr>
<td>Ox31</td>
<td></td>
</tr>
<tr>
<td>Ox32</td>
<td></td>
</tr>
<tr>
<td>Ox33</td>
<td></td>
</tr>
<tr>
<td>Ox34</td>
<td></td>
</tr>
<tr>
<td>Ox35</td>
<td></td>
</tr>
<tr>
<td>Ox36</td>
<td></td>
</tr>
<tr>
<td>Ox37</td>
<td></td>
</tr>
<tr>
<td>Ox38</td>
<td></td>
</tr>
<tr>
<td>Ox39</td>
<td></td>
</tr>
<tr>
<td>Ox3a</td>
<td></td>
</tr>
<tr>
<td>Ox3b</td>
<td></td>
</tr>
<tr>
<td>Ox3c</td>
<td></td>
</tr>
<tr>
<td>Ox3d</td>
<td></td>
</tr>
<tr>
<td>Ox3e</td>
<td></td>
</tr>
<tr>
<td>Ox3f</td>
<td></td>
</tr>
<tr>
<td>Ox40</td>
<td></td>
</tr>
<tr>
<td>Ox41</td>
<td></td>
</tr>
<tr>
<td>Ox42</td>
<td></td>
</tr>
<tr>
<td>Ox43</td>
<td></td>
</tr>
<tr>
<td>Ox44</td>
<td></td>
</tr>
<tr>
<td>Ox45</td>
<td></td>
</tr>
<tr>
<td>Ox46</td>
<td></td>
</tr>
<tr>
<td>Ox47</td>
<td></td>
</tr>
</tbody>
</table>

Table A-4  DRAM Addresses for NON_INTRA_Q Matrix

<table>
<thead>
<tr>
<th>DRAM Address</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ox60</td>
<td></td>
</tr>
<tr>
<td>Ox61</td>
<td></td>
</tr>
<tr>
<td>Ox62</td>
<td></td>
</tr>
<tr>
<td>Ox63</td>
<td></td>
</tr>
<tr>
<td>Ox64</td>
<td></td>
</tr>
<tr>
<td>Ox65</td>
<td></td>
</tr>
<tr>
<td>Ox66</td>
<td></td>
</tr>
<tr>
<td>Ox67</td>
<td></td>
</tr>
<tr>
<td>Ox68</td>
<td></td>
</tr>
<tr>
<td>Ox69</td>
<td></td>
</tr>
<tr>
<td>Ox6a</td>
<td></td>
</tr>
<tr>
<td>Ox6b</td>
<td></td>
</tr>
<tr>
<td>Ox6c</td>
<td></td>
</tr>
<tr>
<td>Ox6d</td>
<td></td>
</tr>
<tr>
<td>Ox6e</td>
<td></td>
</tr>
<tr>
<td>Ox6f</td>
<td></td>
</tr>
<tr>
<td>Ox70</td>
<td></td>
</tr>
<tr>
<td>Ox71</td>
<td></td>
</tr>
<tr>
<td>Ox72</td>
<td></td>
</tr>
<tr>
<td>Ox73</td>
<td></td>
</tr>
<tr>
<td>Ox74</td>
<td></td>
</tr>
<tr>
<td>Ox75</td>
<td></td>
</tr>
<tr>
<td>Ox76</td>
<td></td>
</tr>
<tr>
<td>Ox77</td>
<td></td>
</tr>
<tr>
<td>Ox78</td>
<td></td>
</tr>
<tr>
<td>Ox79</td>
<td></td>
</tr>
<tr>
<td>Ox7a</td>
<td></td>
</tr>
<tr>
<td>Ox7b</td>
<td></td>
</tr>
<tr>
<td>Ox7c</td>
<td></td>
</tr>
<tr>
<td>Ox7d</td>
<td></td>
</tr>
<tr>
<td>Ox7e</td>
<td></td>
</tr>
<tr>
<td>Ox7f</td>
<td></td>
</tr>
<tr>
<td>Ox80</td>
<td></td>
</tr>
<tr>
<td>Ox81</td>
<td></td>
</tr>
<tr>
<td>Ox82</td>
<td></td>
</tr>
<tr>
<td>Ox83</td>
<td></td>
</tr>
<tr>
<td>Ox84</td>
<td></td>
</tr>
<tr>
<td>Ox85</td>
<td></td>
</tr>
<tr>
<td>Ox86</td>
<td></td>
</tr>
<tr>
<td>Ox87</td>
<td></td>
</tr>
<tr>
<td>Ox88</td>
<td></td>
</tr>
<tr>
<td>Ox89</td>
<td></td>
</tr>
<tr>
<td>Ox8a</td>
<td></td>
</tr>
<tr>
<td>Ox8b</td>
<td></td>
</tr>
<tr>
<td>Ox8c</td>
<td></td>
</tr>
<tr>
<td>Ox8d</td>
<td></td>
</tr>
<tr>
<td>Ox8e</td>
<td></td>
</tr>
<tr>
<td>Ox8f</td>
<td></td>
</tr>
<tr>
<td>Ox90</td>
<td></td>
</tr>
<tr>
<td>Ox91</td>
<td></td>
</tr>
<tr>
<td>Ox92</td>
<td></td>
</tr>
<tr>
<td>Ox93</td>
<td></td>
</tr>
<tr>
<td>Ox94</td>
<td></td>
</tr>
<tr>
<td>Ox95</td>
<td></td>
</tr>
<tr>
<td>Ox96</td>
<td></td>
</tr>
<tr>
<td>Ox97</td>
<td></td>
</tr>
<tr>
<td>Ox98</td>
<td></td>
</tr>
<tr>
<td>Ox99</td>
<td></td>
</tr>
<tr>
<td>Ox9a</td>
<td></td>
</tr>
<tr>
<td>Ox9b</td>
<td></td>
</tr>
<tr>
<td>Ox9c</td>
<td></td>
</tr>
<tr>
<td>Ox9d</td>
<td></td>
</tr>
<tr>
<td>Ox9e</td>
<td></td>
</tr>
<tr>
<td>Ox9f</td>
<td></td>
</tr>
</tbody>
</table>

Table A-5 lists the DRAM word locations of variables in the picture group.

Table A-5  DRAM Addresses for Picture Variable Group

<table>
<thead>
<tr>
<th>DRAM Address</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ox16</td>
<td>PIC_SEM</td>
</tr>
<tr>
<td>Ox17</td>
<td>TIME_CODE_0</td>
</tr>
<tr>
<td>Ox18</td>
<td>TIME_CODE_1</td>
</tr>
<tr>
<td>Ox19</td>
<td>TEMPORAL_REFERENCE</td>
</tr>
</tbody>
</table>

Table A-6 indicates the location in DRAM for the encoding of the microapplication-executable header information.
### Table A-6  DRAM Addresses for Microapplication Version Information

<table>
<thead>
<tr>
<th>DRAM Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xa0[15:8]</td>
<td>Microapplication version number, high byte (REV_HI). This contains the major version (should be 0x02 for the microapplication covered by this document).</td>
</tr>
<tr>
<td>0xa0[7:0]</td>
<td>Microapplication version number, low byte (REV_LO). This contains the minor version number.</td>
</tr>
<tr>
<td>0xa1[15]</td>
<td>This bit is used to denote experimental microapplication executables. If this bit is set, it is indicative of an intermediate development version or a version with internal debugging functionality.</td>
</tr>
<tr>
<td>0xa1[14:8]</td>
<td>Extensions ID. Should be 0x00. Other values indicate that the microapplication is an experimental (debugging) revision and/or contains functions not specified in this document.</td>
</tr>
<tr>
<td>0xa1[7:0]</td>
<td>Product ID. Should be 0x02.</td>
</tr>
<tr>
<td>0xa2</td>
<td>Initial value for CPU_pc register.</td>
</tr>
<tr>
<td>0xa3</td>
<td>RESERVED</td>
</tr>
</tbody>
</table>
Appendix B
Microapplication Executable File Format

This appendix describes the format of the CL450 executable file and applies to microapplication versions 2.00 through 2.FF only.

The CL450 microapplication executable is distributed in a single file with a "BIN" extension. Once a microapplication executable file has been copied from the distribution disk, its name can be changed.

The syntax for the file is represented with the same conventions used in the compressed bitstream syntax of the MPEG standard. In addition to the data-type symbols used in the MPEG standard, the new symbols shown in Table B-1 will be used.

B.1 Syntax Conventions
It is assumed that the file is read one byte at a time, and that bit ordering within the file bytes is preserved when the executable file is read.

The microapplication executable consists of a single header, followed by one or more code segments. The top-level syntax of the file is as follows:

```c
microcode_executable()
{
    file_header;
    for (c=0; c < num_segments; c++)
        code_segment();
}
```

The file_header includes the total number of code segments in the executable file (num_segments), information about the microapplication, and information needed to initialize the CL450 on-chip CPU’s instruction memory (IMEM) and program counter.

The code_segments contain the CL450 executable code, and must be loaded into the CL450's local DRAM by the host. In addition, one or more of the code segments must also be loaded into IMEM to serve as a bootstrap for the CL450's CPU.

The file header contains 10 entries as shown below. Each entry is described following the file_header syntax shown on the next page.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubyte</td>
<td>Unsigned integer, occupying a single byte</td>
</tr>
<tr>
<td>uilsbf</td>
<td>Unsigned integer, least-significant byte first</td>
</tr>
<tr>
<td>silsbf</td>
<td>Signed integer, least-significant byte first</td>
</tr>
<tr>
<td>strfbf</td>
<td>String, null-terminated, first (left-most) byte first</td>
</tr>
</tbody>
</table>
file_header() 
{
    magic_number 16 uilsbf
    rev_hi 8 ubyte
    rev_lo 8 ubyte
    productID 16 uilsbf
    init_PC 16 uilsbf
    imem_dram 16 uilsbf
    exe_length 32 ubyte
    num_segments 16 uilsbf
    RESERVED 32 uilsbf
    comment_string 80.8 strfbf
}

**magic_number** — This entry in the file_header contains the value 0xC3C3 for all valid CL450 microapplication files. If a file does not begin with this value, then it is not a valid "BIN" microapplication executable.

**rev_hi** — This 8-bit value gives the major version number of the microapplication executable contained in the file. For microapplication executables and executable files covered by this document, the value is 0x02.

**rev_lo** — This 8-bit value gives the minor version number of the microapplication executable. Typically, larger values indicate later microapplication releases.

**productID** — This value is used to encode the feature set of the microapplication executable. If productID[15] is 1, then the executable is an "experimental" microapplication release and should be regarded with caution. Experimental microapplication releases are typically made as part of the microapplication support and system debugging process and will not match the microapplication functional specification in one or more ways. Experimental microapplications should not be used except under the direction of C-Cube Microsystems technical support.
File Header Structure

The least significant bits of productID are used to encode the feature set implemented by the microapplication. For a microapplication with a major version number (rev_hi) of 0x02, a productID with the lower bits equal to 2 indicates that the feature set described by this document is implemented. Previous microapplication releases with major version numbers of 0x01 used the values 0 and 1 to indicate the discontinued "CL450 Baseline" and "CL450i" feature sets.

Finally, the high-order bits of productID (bit 14 down to the bits described above) are reserved for the encoding of customer-specific microapplication modifications.

**init_PC** — This entry contains the value the host should write to the CL450's CPU_pc register during the microapplication loading process prior to enabling CPU execution.

**imem_dram** — This value is a byte address in the CL450's DRAM at which the DRAM and IMEM address spaces are conceptually overlapped. All data from the code_segments which is written to the DRAM range from byte address "imem_dram" through byte address "imem_dram+2044" inclusive must also be written by the host to the corresponding IMEM address. IMEM addresses can be computed from DRAM addresses within the "overlapped" range as follows:

\[
\text{IMEMaddress} = \frac{(\text{DRAMaddress} - \text{imem_dram})}{4};
\]

Note that IMEM may only be written 32-bits at a time (requiring two sequential 16-bit write operations from the host).

**exe_length** — This entry gives the length in bytes of the complete executable file, including the header.

**num_segments** — This entry gives the number of code_segments contained in the file. If the end of the last code segment in the file and the actual end of the file do not coincide exactly, the executable file has been corrupted and should not be used.
RESERVED — This 32-bit quantity is reserved for future use and executable file readers should ignore its contents.

**comment_string** — This file_header:entries:magic_number string contains ancillary information about the microapplication contained in the file. Typically this includes at least the microapplication copyright notice.

Each code segment contains two entries and an array of values, as shown below. Each entry is described following the code_segment syntax:

```c
code_segment()
{
    seg_length; 32 ulsbf
    seg_address; 32 ubyte
    for (c=0;c < (seg_length/4);c++){
        instruction_word_1 ; 16 ulsbf
        instruction_word_0 ; 16 ulsbf
    }
}
```

**seg_length** — This entry gives the length of the segment in bytes. Each CL450 microinstruction is 32-bits long, occupying four bytes.

**seg_address** — The entry gives the DRAM byte address at which this segment should begin loading. The loaded segment should occupy DRAM byte addresses "seg_address" through "seg_address + seg_length - 1" inclusive. Note that if part or all of the DRAM area occupied by a segment coincides with the "DRAM/IMEM overlap" region (see imem_dram, above), then that part of the segment must be written to the appropriate portion of IMEM as well as DRAM.

**instruction_word_1** and **instruction_word_0** — These values are the actual microapplication instructions. They should be loaded into the CL450's local DRAM (and possibly IMEM). The first value from the file (instruction_word_1) is the most significant half of each CL450 instruction, and should be written to the CL450 first. When writing code_segment data to the CL450's DRAM, the first word of data from the
Code Segment Structure

executable file (instruction_word_1) goes in the lower DRAM word address, and instruction_word_0 goes in the higher DRAM word address. When writing code_segment data to IMEM, instruction_word_1 is written to the CPU_imem register first, followed by instruction_word_0.
This appendix describes the C-Cube Microsystems microapplication distribution disk, which contains the following files:

- \*DRAM450.BIN: This file contains the CL450 microapplication executable in the format described in Appendix B.

- \*450DEMO.EXE: This file is a demonstration program, which runs under Microsoft Windows™ and is compiled for execution on the CL450 development board. When executed, the microapplication executable located in the same directory in the file “DRAM-450.BIN” is loaded, and the MPEG elementary video bitstream located in the same directory in the file “MPEG.NFL” is decoded and displayed.

- \*MPEG.NFL: This file contains an MPEG elementary video bitstream.

- \*README.DOC: This file contains a complete description of the contents of the subdirectories, as well as release notes and errata (if any) for the particular microapplication version contained on the disk.
\KERMIT.INI: This application contains a configuration file read by 450DEMO.EXE (by the C3VIO library) at program start-up for information on the configuration of the CL450 development boards. The version of this file supplied with the microapplication corresponds with the factory settings of the development board.

The C-Cube Microsystems microapplication distribution disk also contains the following directories:

- **\450DEMO**: This directory contains the source, header, and make files used to compile 450DEMO.EXE.
- **\C3LIB**: This directory contains the source, header, and make files for an example library of CL450 I/O routines. This library may be built either for DOS or Windows applications, although a version compiled for DOS is supplied and is required to build 450DEMO.EXE. By changing the routines in this library, it should be possible to port 450DEMO.EXE to any other CL450-based subsystem.
This appendix contains information which should be collected before reporting a suspected bug in the CL450 microapplication to C-Cube Microsystems customer support:

1. Determine the version number of your microapplication by examining the appropriate DRAM location (see Table A-6) while the microapplication is loaded.

2. Determine the value of CPU_control[0] after the failure is observed.

3. If CPU_control[0] is 0, then either the host or the microapplication halted the CL450’s CPU. Capture the contents of CPU_pc. If CPU_control[0] is 1, then the CL450’s CPU is still running. Read the contents of CPU_pc and attempt to determine if the microapplication is looping within one or more areas of IMEM. Report the IMEM addresses (CPU_pc values) for these areas.

4. Capture the contents of CPU_intenable and CPU_int.

5. Capture the contents of DRAM addresses 0x0 through 0x360.
6. Capture the contents of TMEM addresses 0x0 through 0x7f. Note that reading TMEM contents from the host only retrieves the least-significant 16 bits from each 24-bit TMEM word. This is sufficient for an initial bug report, but you may be asked to run a special microapplication which reads and dumps the full TMEM contents.

7. Capture the contents of HMEM addresses 0x0 through 0xf. When reporting the content of a CL450 register, it is important to get all 16 bits read from the CL450, not just the bits which are defined.

Most of the items listed above may change while the microapplication is executing. Because of this, values should be captured twice. First, perform the operations which cause the suspected bug and capture the values while the microapplication is still executing. Second, reset and restart the system and execute until the bug occurs again. Then clear CPU_control[0] and capture all of the values. This will ensure that the microapplication is not changing the state while it is being captured.

The registers which must be used to read from TMEM (CPU_taddr and CPU_tmem) are used by the microapplication. Because of this, reading TMEM while the microapplication is executing will not always be successful. Sometimes the microapplication will interfere with the results of reading TMEM, and sometimes the act of reading TMEM will cause the microapplication to malfunction.

Finally, a description of the host operations and/or bitstream which cause the suspected bug to occur should be supplied. It is possible that C-Cube Microsystems will request a copy of the bitstream to duplicate the problem.
AC characteristics, 7-3 to 7-19
AccessSCR(). See Macro commands
Address Bus. See Host Address Bus
Address size
registers, 1-5
other memories, 1-6
Address Strobe, 3-4
AIC bit, 4-19, 7-12, 8-11, 8-12, 12-38
AS (address strobe) 3-2, 3-3, 3-4, 4-2, 4-3, 4-5, 4-6, 4-7, 4-8, 4-9, 4-10, 4-11, 4-12, 4-13, 4-14, 4-15, 4-16, 4-18, 4-20, 4-21, 7-4, 7-5, 7-12
Audio decoder
in MPEG standard, 2-2
synchronizing to, 13-6
Audio streams
See bitstream transfer process
Audio/video synchronization, 13-1 to 13-12
Auto interrupt clear. See AIC bit

B
Bank 0, 5-4
Bank 1, 5-4

Bitstream
demultiplexing, 13-11
NewPacket() association, 13-9
operating restrictions, 15-5
Bitstream buffer 4-12, 5-2, 9-5, 9-6, 11-20, 11-27, 12-6, 12-10, 12-17, 12-22, 12-23, 12-27, 12-29, 12-31
and DMA operation, 9-5
and programmed access, 9-6
flushing, 11-16
fullness reading, 11-19
setting threshold, 11-38
Bitstream data
example using NewPacket(), 13-8
Bitstream data transmission 13-9
Bitstream parameters
accessing, 14-1
default settings, 10-6
picture variable group, 14-1, 14-4
sequence variable group, 14-1
Bitstream transfer process
DMA operation, 9-5
programmed access, 9-5
pseudocode example, 9-6
Bitstream underflow. See error recovery
and concealment
Block, 2-5
Block row boundaries, 12-2
Bootstrap. See initialization
Border positioning, 6-4
Border
blue component bits. See BorBl bits
green component bits. See BorGrn bits
left size. See BorLeft bits
red component bits. See BorRd bits
BorBl bits, 8-29
BorGrn bits, 8-29
BorLeft bits, 8-28
BorRd bits, 8-28
B-pictures. See Picture type (bidirectional)
broken_link, 10-7
BS bit, 8-8
Buffer fullness status, 11-2
Byte addressing
with other memories, 1-6
with registers, 1-5
Byte swap bit. See BS, 8-8

C
CAS, 5-8, 7-14, 7-15
CAS-before-RAS, 5-3, 7-15
CASIN, 5-8, 7-14
CDctr bit, 8-11
CEn bit, 8-19, 8-21, 10-1
CFLEVEL, 3-2, 3-7, 3-9, 4-2, 4-7, 4-12, 4-17, 4-18, 8-7, 8-9, 9-7
CFLEVEL assertion control, 8-10
ChromaData bits, 8-25
CL450
AC characteristics, 7-3
bitstream transfer process, 9-5
block diagram of, 1-4
coded data transfer, 13-7
command process, 9-4
command states, 11-4
command writing sequence, 8-15
deadlock avoidance, 11-3
decode process, 9-7
display process, 9-7
DRAM interface, 3-9
electrical specs, 7-1 to 7-19
error recovery and concealment, 15-3
executable file format, B-1
features, 1-2
functional description, 1-4
general description, 1-1
host interface, 4-1
initialization, 10-1 to 10-10
interrupts, 12-1 to 12-38, 9-8
macro command failure, 11-4
macro command summary, 9-4
macro commands, 11-1 to 11-47
microcode distribution disk, C-1
microcode features, 1-2, 9-1 to 9-10
packaging specs, 7-20 to 7-26
picture display rate. See frame rate conversion
pinout, 7-21
process configurations, 9-3
product family, 1-1
programming overview, 1-7
registers, 1-7, 8-1 to 8-29
restrictions, 15-5
SCR counter, 13-2
synchronizing from, 13-3, 13-7
typical applications, 1-8
typical system application, 1-8
used as SCR master, 13-7
CL450 microapplication. See microapplication
Clock timing, 7-17
Clock source bit. See CS bit
closed_gop, 10-7
Cmd bit, 8-14, 8-15
CMEM 1-5, 1-7, 3-2, 3-4, 3-5, 3-7, 4-1, 4-6, 4-7, 4-11, 4-12, 4-13, 4-14, 4-15, 4-17, 4-18, 5-3, 8-7, 8-8, 8-9, 9-5, 9-6, 9-7, 103, 11-16, 11-20,
back-to-back transactions, 4-12
CFLEVEL signal, 4-18
determining fullness of, 9-6
diagram 4-4-7
fullness polling, 9-6
how accessed, 4-6
polled and DMA writes, 4-12
pseudocode example, 9-6
timing, 7-9
when accessed, 4-6
write timing, 7-9
writing to, 4-11, 9-6
CMEM registers, 8-8
CMEM request. See CR bit
CMEM request enable. See CRE bit
CMEM reset. See CRst bit
CMEM_control. See registers
CMEM_data register. See registers
CMEM_dmactrl.
"entirely empty" selection, 9-7
determining fullness, 9-7
See also registers
CMEM_dmactrl register. See registers
Coded Data FIFO. See CMEM
Coded picture rate. See picture_rate 15-1
Color-space converter, 1-5, 11-30, 11-34
Command FIFO, 5-2, 9-4, 11-1, 11-7, 11-9, 11-23, 11-27, 12-36
command read pointer, 11-7
timing, 7-9
Prescaler,
when accessed, 4-6
write timing, 7-9
writing to, 4-11, 9-6
CMEM registers, 8-8
CMEM request. See CR bit
CMEM request enable. See CRE bit
CMEM reset. See CRst bit
CMEM_control. See registers
CMEM_data register. See registers
CMEM_dmactrl.
"entirely empty" selection, 9-7
determining fullness, 9-7
See also registers
CMEM_dmactrl register. See registers
Coded Data FIFO. See CMEM
Coded picture rate. See picture_rate 15-1
Color-space converter, 1-5, 11-30, 11-34
Command FIFO, 5-2, 9-4, 11-1, 11-7, 11-9, 11-23, 11-27, 12-36
command read pointer, 11-7
timing, 7-9
Prescaler,
when accessed, 4-6
write timing, 7-9
writing to, 4-11, 9-6
CMEM registers, 8-8
CMEM request. See CR bit
CMEM request enable. See CRE bit
CMEM reset. See CRst bit
CMEM_control. See registers
CMEM_data register. See registers
CMEM_dmactrl.
"entirely empty" selection, 9-7
determining fullness, 9-7
See also registers
CMEM_dmactrl register. See registers
Coded Data FIFO. See CMEM
Coded picture rate. See picture_rate 15-1
Color-space converter, 1-5, 11-30, 11-34
Command FIFO, 5-2, 9-4, 11-1, 11-7, 11-9, 11-23, 11-27, 12-36
command read pointer, 11-7
timing, 7-9
Prescaler,
when accessed, 4-6
write timing, 7-9
writing to, 4-11, 9-6
CMEM registers, 8-8
CMEM request. See CR bit
CMEM request enable. See CRE bit
CMEM reset. See CRst bit
CMEM_control. See registers
CMEM_data register. See registers
CMEM_dmactrl.
"entirely empty" selection, 9-7
determining fullness, 9-7
See also registers
CMEM_dmactrl register. See registers
Coded Data FIFO. See CMEM
Coded picture rate. See picture_rate 15-1
Color-space converter, 1-5, 11-30, 11-34
Command FIFO, 5-2, 9-4, 11-1, 11-7, 11-9, 11-23, 11-27, 12-36
command read pointer, 11-7
timing, 7-9
Prescaler,
with periodic PTSs, 13-11
without PTSs, 13-10
Data transfer acknowledge, 3-5
Data transfer signals
A[20:1], 3-3, 3-4
AS (Address Strobe), 3-4
D[15:0] 3-2, 3-3, 3-4, 4-2, 4-5, 4-6, 4-7, 4-8, 4-9, 4-13, 4-15, 7-4, 7-5, 8-7, 7-4, 7-21, 7-12
DTACK (Data transfer acknowledge) 3-2, 3-3, 3-5, 4-2, 4-5, 4-8, 4-9, 4-10, 4-11, 4-13, 4-15, 4-18, 4-19, 4-21, 7-4, 7-5, 7-9, 7-10, 7-12, 14-9
generation logic, 4-10
state logic diagram, 4-11
LDS (Lower Data Strobe) 3-2, 3-3
R/W (Read/Write), 3-4, 3-5
UDS (Upper Data Strobe), 3-3
dct. See discrete cosine transform, 2-10
de bit, 4-14, 4-16, 8-10
debug. See registers, 8-2
de decode process, 9-7
de decoder
audio, 2-2
system, 2-2
video, 2-2
decode-time interrupt, 12-10, 12-14
decoding process configuration, 9-3
decoding defaults. See bitstream parameter (default settings)
default values
bitstream parameters, 10-7
DRAM resident variables, 10-8
registers, 10-8
demonstration program, C-1
demultiplexing system bitstream, 13-11
difference counter bits. See CDCT bits
direct-access registers. See registers
discrete cosine transform, 2-10
display mode
RGB, 6-9
ycbcr, 6-9
display process, 9-7
display time stamps. See Presentation time stamps, 13-2
display still(). see macro commands
display-time (See Interrupts) 12-2, 12-10, 12-14
div bits, 8-16, 8-17
DMA transfers, 1-2, 3-2, 4-6, 4-11, 4-12, 4-13, 4-14, 4-15, 4-17, 8-7, 9-5, 11-26, 12-23
t Femite controller, 3-5
signals, 3-5
DMA acknowledge, 3-6
DMA done, 3-6
DMA enable bit. See DE bit
DMA request, 3-6
DMA signals, 3-5
DMAACK (DMA Acknowledge) 3-2, 3-5, 3-6, 4-2, 4-6, 4-7, 4-12, 4-14, 4-15, 7-10
DMAREQ (DMA Request), 3-2, 3-5, 3-6, 4-2, 4-7, 4-12, 4-14, 4-15, 4-17, 7-10, 8-10
DONE (DMA Done), 3-2, 3-5, 3-6
DTC (Data transfer complete), 3-6, 3-2, 3-5, 3-6, 4-2, 4-6, 4-7, 4-12, 4-14, 4-15
DMA transaction, 4-14
dMA transfer complete, 3-6
dMA transfers, 4-1
DMAACK. See DMA signals
DMAREQ. See DMA signals
DONE. See DMA Signals
DRAM command FIFO. See command FIFO
DRAM (local), 1-7, 3-4, 3-8, 3-10, 3-11, 4-1, 4-8, 4-10, 4-11, 5-1, 5-2, 5-3, 5-4, 5-7, 5-8, 6-3, 6-9, 7-4, 7-13, 7-14, 8-6, 8-23, 9-7, 10-1, 10-2, 10-9, 10-10, 11-19, 11-20, 11-25, 11-45, 12-27, 14-1, 14-5, 14-7, 14-8

Index-4 C-Cube Microsystems
256 x 16, 5-7
256 x 4, 5-6
accessing variables, 14-1 to 14-10
amount recommended, 5-2
design guidelines, 5-7
eliminating noise from, 5-7
example implementation, 5-5
host access, 5-2
interface registers, 8-22
latency, 4-8
matrix order, A-2
microcode-programmable features 5-3
page-mode read timing, 5-8
page-mode write timing, 5-9
parts to use, 5-8
proper layout, 5-7
read timing, 4-8
refresh performed, 5-8
semaphore access, 14-5
timing showing CAS and RAS, 7-15
type and organization, 5-2
uses of, 5-2
when accessed, 4-4
write timing, 4-9
DRAM addresses
host scratch storage, A-1
INTRA_Q matrix, A-3
microcode version, A-4
NON_INTRA_Q matrix, A-3
picture variable group, A-3
sequence variable group, A-2
DRAM bank 0, 3-4
DRAM bank 1, 3-4
DRAM bus timing, 7-14
DRAM interface, 3-9
DRAM Interface signals
LCAS (Lower column address strobe) 3-11
LCASIN (Lower data latch enable), 3-11
MA (Memory address bus), 3-2, 3-9, 3-10, 5-5, 5-6, 5-8, 7-14
MD (Memory data bus), 3-2, 3-9, 3-10, 5-5, 5-6, 7-14
RAS (Row Address Strobe), 3-2, 3-10, 5-4, 5-5, 5-6, 5-6,
UCAS (Upper column address strobe), 3-11
UCASIN (Upper data latch enable) 3-11
WE (write enable), 3-11
DRAM variables
reading, 14-7
timing restrictions, 14-8
writing, 14-7
DRAM variables. See Chapter 14
DRAM_refcnt register. See registers (direct-access)
DRAM-resident variables default values, 10-8
DTACK. See Data transfer signals
DTC. See DMA signals
dual-address, 4-16

E
Empty status bits. See IQ-4Q, 8-9
Empty status enable bits. See IQE-4QE, 8-10
END-D. See interrupts (listed)
ERR. See Interrupts (listed)
Error recovery and concealment
during bitstream underflow, 15-4
during header decode, 15-4
during picture decode, 15-3
signaled vs. unsignaled, 15-3

F
FIFO. See Command FIFO or CMEM
file_header
described, B-2
entries,
comment_string, B-5
exe_length, B-4
imem_dram, B-4
init_PC, B-4
instruction_word 0, B-5
instruction_word 1, B-5
magic_number, B-3
num_segments, B-4
productID, B-3
RESERVED, B-5
rev_hi, B-3
rev_lo, B-3
seg_address, B-5
seg_length, B-5

filter argument, 11-16, 11-17
FLAGS, 14-3
FlushBitstream(). See Macro commands
format argument, 11-40
Forward prediction, 2-2-6
Frame rate conversion, 15-1

G

gbBorder, 11-28, 11-30
GCLK, 3-2, 3-7, 3-8, 3-9, 3-13, 4-4, 4-5, 4-6, 4-7, 4-8, 4-11, 4-12, 4-14, 5-8, 7-14, 7-15, 7-17, 7-19, 8-7, 8-16, 8-17, 8-23
timing, 7-17
GCLK. See Timing signals, 3-8
GDATA, 4-5, 4-7
GDATA bus, 4-6
GOP. See group of pictures
GOP header, 11-28, 14-4
group of pictures (GOP) 2-4
group_of_pictures, 11-15
group_start_code, 12-19, 14-8
GRP Interrupt. See Interrupts (listed)
GSEL, 4-4, 4-5, 4-7

H

Halting microcode, 10-10
Handshaking protocol. See interrupts
HCLK, 3-2, 3-7, 3-8, 3-9, 4-2, 4-4, 4-5, 4-6, 4-8, 4-9, 4-13, 4-14, 4-15, 4-17, 4-21, 7-4, 7-5, 7-6
See also timing signals
HData bits, 8-14
height argument, 11-42, 12-3, 12-5
HMEM 4-19, 8-15, 9-4, 10-4, 10-9, 11-1, 11-2, 11-3, 11-4, 11-7, 11-12, 11-13, 11-19, 12-1, 12-7, 12-10, 12-38
access registers, 10-4
address allocation, 11-2
interrupt status location, 12-7
macro command use, 11-3
scratch storage use, 11-3
semaphore use, 11-3
Horizontal blanking, 6-2
Horizontal synchronization. See HSYNC
Horizontal timing (synchronization), 6-6
HORIZONTAL_SIZE, 14-3
horizontal_size, 10-7, 11-44
Host
access of DRAM variables, 14-1
overflow responsibilities, 11-9
time-out counter use, 11-3
Host address bus, 3-4
Host data bits. See HData
Host data bus, 3-4
Host Interface
memory access, 4-3
pinout diagram, 4-2
Host interface
description, 4-1
diagram, 4-5
registers, 8-6
Host processor
initialization sequence, 10-1
writing bootstrap, 10-2
HOST_control. See registers
HOST_intvecw. See registers
HOST_newcmd. See registers
HOST_raddr register. See registers
HOST_rdata. See registers

Index-6 C-Cube Microsystems
HOST_scr0. See registers
HOST_scr1. See registers
HOST_scr2. See registers
HSYNC 1-1, 3-1, 3-2, 3-12, 3-13, 6-4, 6-5, 6-6, 6-7, 6-8, 7-18, 11-28, 11-32, 12-3, 12-5

I

IDLE command state, 11-4, 11-6, 11-9, 11-20, 11-45, 13-3
Idle process configuration 9-9-2
IDLE. See command states
IE bit, 8-18, 8-19
Images
coding vs. display frequency, 9-8
IMEM 1-7, 8-18, 8-20, 9-1, 10-1, 10-4, 10-5, 10-9
access registers, 8-20
registers, 10-4
used in bootstrap, 10-1
write data flow, 8-20
Indirect register access. See registers (indirect)
Indirect video registers. See registers (indirect)
Initialization, 10-1 to 10-10
determining completeness, 10-9
operations performed, 10-3
process configuration, 9-2
registers, 10-2
Inputs, 1-1
InquireBufferFullness(). See Macro commands 11-19
INT, 3-2, 3-6, 3-7, 4-2, 4-6, 4-7, 4-19, 4-21, 7-12, 8-7, 9-8, 12-1, 12-7, 12-8, 12-10, 12-36, 12-37
 interrupts, 4-19, 8-11, 8-12, 12-38
INT bit, 8-19, 8-11, 8-12, 12-38
INTACK, 3-2, 3-6, 3-7, 3-9, 4-2, 4-6, 4-7, 4-19, 4-20, 4-21, 7-12, 8-7, 8-12, 8-13, 12-38
Interface (video), 3-10
Internal registers. See Registers (internal)
Internal Reset. See Rst bit
Inter-picture decoding. See Picture type
Interrupt (vectored) timing, 7-12
Interrupt priority ID. See IPID bit
Interrupt signals
INT (interrupt request), 3-7
INTACK (Interrupt Acknowledge), 3-7
Interrupt status, 4-19, 11-2, 12-7, 12-8, 12-9, 12-11, 12-32, 12-33, 12-36
Interrupt Status Location
cleared by Host, 12-11
host's responsibilities, 12-11
Interrupt vector operation, 4-6
Interrupts, 3-6, 9-8, 12-1 to 12-38
CL450 internal queuing, 12-13
type
control registers, 8-11
types
examples, 12-28
handshaking protocol, 12-7
listed
END-D, 12-15
END-V, 12-16
ERR, 12-17, 12-29
GRP, 12-19
PIC-D, 12-20
PIC-V, 12-21
RDY, 12-22
SCN, 11-28, 12-24, 12-37
SEQ-D, 12-25
SEQ-V, 12-26
UND, 12-27
PIC-V, SCN example, 12-34
pooled, 4-19
posting procedure, 12-9
RDY, UND, ERR example, 12-29
servicing, 12-37
summary table, 12-2
Interrupt enable bit. See IE bit
INTRA_Q, 14-4
Intra-picture (Transform) coding, 2-10
I-pictures. See Picture type
IPID bit, 4-19, 8-13
IVect bits 4-19, 8-13

J
Jitter tolerance, 9-8

K
K3-K0 bits, 8-26

L
LCAS, 3-2, 3-10, 3-11, 5-3, 5-5, 5-6, 5-7
LCASIN, 3-2, 3-11, 5-3, 5-4, 5-5, 5-6, 5-7
LDS, 1-6, 3-2, 3-3, 3-4, 4-2, 4-4, 4-6, 4-8, 4-9, 4-13, 4-15, 4-21, 5-4
decoded values, 4-3
shown with Write timing, 4-10
See also data transfer signals
Left border size bits. See BorLeft
LeftBorder argument, 11-31, 11-32
length argument, 11-22, 2-22, 13-11, 13-8
level argument, 11-36, 11-38
load_intra_quantizer_matrix, 10-7
load_non_intra_quantizer_matrix, 10-7
locations, 11-2
Lower column address strobe. See LCAS.
Lower data latch enable. See LCASIN

M
MA. See DRAM interface signals
Macro commands
default settings, 10-6
discussed, 11-1 to 11-47
effect on command state, 11-11
for loading registers, 10-5
function codes 11-1, 11-11

latency, 11-9
listed
AccessSCR(), 11-12
DisplayStill(), 11-14
FlushBitstream(), 11-16
InquireBufferFullness(), 11-19
NewPacket(), 11-20
Pause(), 11-25
Play(), 11-26
Reset(), 11-27
Scan(), 11-28
SetBlank(), 11-29
SetBorder(), 6-4, 11-30
SetColorMode(), 11-34
SetInterruptMask(), 11-36
SetThreshold(), 11-38
SetVideoFormat(), 11-40
SetWindow(), 6-8, 11-42
SingleStep(), 11-45
SlowMotion(), 11-46
overflow of, 11-9
parameter values, 11-1
polling HOST_newcmd, 11-3
priority, 11-1
summary, 9-4
types, 11-10
writing, 11-2
Macroblock, 2-5
marker_bits, 12-17
mask argument, 12-8, 12-15, 12-21, 12-24, 12-32
mask bit assignment, 9-8, 11-36, 11-38
MD, 5-8, 7-14
Memory
design guidelines, 5-7
refresh cycle, 5-11
Memory access
byte-wide vs. word-wide, 4-3
CMEM, 4-6
DMA operation, 4-6
interrupt vectors, 4-6
local DRAM, 4-4
registers, 4-4
Memory address bus. See DRAM interface signals
Memory bus
address lines, 5-3
control lines, 5-3
Memory data bus, 3-10
Microapplication (CL450)
bitstream transfer process, 9-5
code segment structure, B-5
command process, 9-4
decode process, 9-7
decoding process configuration, 9-3
default settings, 10-6
display process, 9-7
distribution disk, C-1
executable file, 10-5
file structure, B-2
frame rate conversion, 15-1
halting, 10-10
Idle process configuration, 9-2
initialization process configuration, 9-2
loading process configuration, 9-2
loading registers, 10-5
loading sequence, 10-9
Pause process configuration, 9-2
relationship to hardware, 9-3
segment blocks, 10-5
synchronization, 9-8
Miscellaneous signals, 3-13
mode argument, 11-34, 12-5
Motion compensation, 2-6, 2-8
MPEG
bitstream demultiplexing, 13-11
decoding defaults. See bitstream parameters (default settings) 10-7
decoding process, 2-2
defined, 2-1
standard, iii
stream structure, 2-2

N
NCE bit, 8-20
NCS bit, 8-19
New command bit. See Cmd bit
New command interrupt status bit. See NCS bit, 8-19
New command interrupt enable bit See NCE bit, 8-20
NewPacket() 9-8, 11-16, 13-7, 13-11
and RDY interrupt production, 12-22
bitstream association, 13-3, 13-9
Command FIFO vs. Bitstream buffer 11-24
timeStamp arguments, 11-22
when to use, 13-7
NewPacket(). See Macro commands
NewSeq bit, 14-2, 14-8
nominal field rate, 15-1
NON_INTRA_Q, 14-4
Normal. See registers (normal)
NTSC, 1-2, 1-3, 1-8, 6-2, 15-1

O
Open-drain output, 7-13
Output window. See window, 11-7

P
Packaging drawings, 7-25
Packet. See NewPacket()
Packet read pointer, 11-9
Packeted data transfer. See data transfer
PAL, 1-2, 1-3, 1-8, 6-2
PAUSE command state, 11-9, 11-15, 11-25, 11-28, 12-34, 13-5
Pause process configuration, 9-2
Pause(). See Macro commands, 11-25
PC bits, 8-19
PIC_SEM, 108
PIC_SEM location encoding, 14-5
PIC-D Interrupt. See Interrupts (listed)
Picture, 2-4, 11-15
Picture display rate. See frame rate conversion, 15-1
Picture group (of DRAM variables), 14-4
Picture headers, 14-4
Picture type
  bidirectional, 2-7, 9-8
  intra, 2-6, 9-8, 11-18, 11-28
  predicted, 2-6
Picture variable group
  See bitstream parameters, 14-4
picture_coding_type, 15-5
picture_height parameter, 11-33
PICTURE_RATE sequence variable, 14-3
picture_rate parameter
picture_start_code, 11-9, 11-23, 12-20, 12-33, 12-34, 12-36, 13-2, 13-3, 13-12, 14-7
picture_width parameter, 11-33
PIC-V interrupt. See Interrupts (listed)
Pinout of CL450, 7-21
Pixel bus
  described, 3-12, 6-8
  disabling outputs, 6-10
Pixel data high byte bits. See PixHData bits
PixHData bits, 8-29
PLAY command state, 11-28, 11-45, 11-47
Play(). See Macro commands
PLAY-SETUP command state, 11-4 11-6, 11-9, 11-26, 11-28, 11-46
Play-type commands 11-4, 11-10
Powerup. See initialization, 10-1
P-pictures, 9-8
P-pictures. See Picture types (predicted)
  2-6
PQFP, 7-20
Presentation time stamps (PTSs), 2-11, 2-12, 8-16, 9-8, 11-22, 11-23, 13-2, 13-3, 13-4, 13-6, 13-12
Process configurations, 9-1, 9-3
Program counter bits. See PC bits
Programmed access, 9-5, 11-26
  pseudocode example, 9-6
Programmed I/O, 1-2
PTS. See Presentation time stamp
R
R/W, 3-2, 3-3, 3-5, 4-2, 4-4, 4-5, 4-6, 4-7, 4-8, 4-9, 4-12, 4-13, 4-15, 4-16, 4-21, 5-4, 7-4, 7-5, 7-12
RAS, 3-11, 5-3, 5-8, 7-14, 7-15
RAS/CAS, 5-3
rate buffer, 9-5
rBorder, 11-28, 11-30
RDY interrupt. See Interrupts (listed)
Read counter bits. See CRCtr bits
Read/write, 3-5
RefCnt bits, 8-23
refresh clock counter bits. See RefCnt bits
Refresh memory timing, 5-11
Refresh on DRAM, 5-8
Register (internal)
  read timing, 4-8
  write timing, 4-9
Registers
  at startup, 10-2
  byte-wide vs. 16-bit, 8-3
  CMEM_status, See direct-access
  CPU_control, See direct-access
  CPU_iaddr, See direct-access
  CPU_imem, See direct-access
  CPU_int, See direct-access
  CPU_intenb, See direct-access
  CPU_pc, See direct-access
  CPU_taddr, See direct-access
  CPU_tmem, See direct-access
Data organization, 1-5
Debug, 8-2
default parameters, 10-2
default settings, 10-2
default values, 10-7
direct-access (listed)

CMEM_control, 8-3, 8-5, 8-7, 8-8
CMEM_data, 8-3, 8-5, 8-7, 8-9
CMEM_dmactrl, 4-12, 4-14, 4-16, 4-17, 4-18, 8-3, 8-5, 8-7, 8-9
CMEM_status, 4-6, 8-3, 8-5, 8-7, 8-11
CPU_control, 8-3, 8-5, 8-19, 8-20
CPU_iaddr, 8-3, 8-5, 8-20, 8-21
CPU_imem, 8-3, 8-5, 8-20, 8-21
CPU_int, 8-3, 8-5, 8-18, 8-19
CPU_intenb, 8-3, 8-5, 8-18, 8-20
CPU_pc, 8-3, 8-5, 8-19
CPU_taddr, 8-3, 8-5, 8-22
CPU_tmem, 8-3, 8-5, 8-22
DRAM_refcnt, 5-8, 8-3, 8-5, 8-23
HOST_control, 3-7, 4-6, 7-12, 8-7, 8-11, 8-12
HOST_intvecr, 8-7, 8-11, 8-13
HOST_intvecw, 3-7, 4-19, 8-3, 8-5, 8-7, 8-11, 8-13
HOST_newcmd, 8-3, 8-5, 8-7, 8-14
HOST_raddr, 8-3, 8-5, 8-7, 8-13, 8-15
HOST_rdata, 8-3, 8-5, 8-7, 8-14
HOST_scr0, 3-9, 8-3, 8-5, 8-7, 8-17
HOST_scr1, 3-9, 8-3, 8-5, 8-16
HOST_scr2, 3-9, 8-3, 8-5, 8-7, 8-16
VID_chroma, 8-3, 8-5, 8-25
VID_control, 8-3, 8-4, 8-5, 8-23, 8-25
VID_regdata, 8-3, 8-5, 8-25
VID_y, 8-26
DRAM interface, 8-22

DRAM_refcnt. See direct-access for IMEM, TMEM, 10-4
for loading microcode, 10-5
host writing sequence, 10-4
IMEM access, 8-20

indirect-access (listed)

VID_sela register, 8-4, 8-5, 8-26
VID_selactive register, 8-4, 8-28
VID_seaux register, 3-12, 8-29
VID_selb register, 6-9, 8-4, 8-26
VID_sebor register, 8-4, 8-28
VID_selGB register, 8-4, 8-29
VID_selmode register, 8-4, 8-27
VID_selR register, 8-4, 8-28

indirect video. See indirect-access initialization, 10-2
initialized by ResetO(), 10-2
internal (summary), 8-5
internal CPU, 8-2, 8-18
loaded by macro commands, 10-5
loading sequence, 10-3
microcode-independent, 10-3
normal category, 8-2
system clock reference, 8-16
TMEM access, 8-22
video (indirect), 8-26
video interface, 8-23
when accessed, 10-3
type
diagnostic, 8-2
initialization, 8-2
internal, 8-2
normal, 8-2, 10-4

release notes, C-1
Reserved signals, 3-3-13
RESET, 3-2, 3-7, 3-9, 4-5, 7-17
Reset(), See Macro commands
Restrictions (CL450)
bitstream, 15-5
output window and timing, 15-5
RGB bit, 8-27
RGB (red, green, blue) 1-2, 3-11, 3-12, 6-
RGB mode select bit. See RGB bit
Row address strobe, 3-10
Rst bit 8-8
Run-length encoding, 2-10

S
Scan lines, 12-2
Scan(). See Macro commands
SCLK, 3-2, 3-7, 3-8, 3-9, 4-5, 7-17, 7-18, 8-7, 8-16
Also see timing signals
SCN interrupt. See Interrupts (listed)
SCR, 2-11, 3-8, 8-16, 9-8, 11-12, 11-13, 11-15, 11-47, 13-1, 13-4, 13-6
See also memory bus, pixel bus
SCR counter 13-2, 13-11
automatic modifications, 13-5
source and drift, 13-4
synchronizing to bitstream, 13-5
updating, 13-4
SCR synchronization, 13-6
Scratch storage (DRAM), 14-1
Semaphore allocation, 14-1, 14-6, 14-7
SEQ_CONTROL, 108, 14-2, 14-3, 14-4, 14-8
SEQ_SEM, 108, 14-2, 14-3, 14-4
SEQ_SEM location encoding, 14-5
SEQ-D interrupt. See Interrupts (listed)
SeqNoDef bit, 14-2, 14-4
Sequence group (of DRAM variables) 14-14-2
Sequence variable group. See bitstream parameters, 14-2
sequence_end_code, 11-15, 12-10, 12-15, 12-16, 12-26, 14-2, 14-7, 14-8
sequence_error_code, 12-17, 15-3
sequence_header, 11-15, 14-1
sequence_header_code, 12-8, 12-25, 12-26, 14-2, 14-7
SEQ-V interrupt. See Interrupts (listed)
SeqWP bit, 14-2
SetBlank(). See Macro commands
SetBorder() command
used to synchronize video, 6-4
SetBorder(). See Macro commands
SetColorMode(). See Macro commands
SetColorMode(). See Macro commands
SetInterruptMask(). See Macro commands
SetThreshold(). See Macro commands
Set-type macro commands, 11-10
SetVideoFormat(). See Macro commands
SetWindow(). See Macro commands
SIF (source input format), 6-2
SIF resolution, 1-1, 1-2
Signals
data transfer, 3-3
described, 3-1
DRAM interface, 3-9
host interface, 3-2
timing, control, and status, 3-7
video interface, 3-11
SingleStep(). See Macro commands
Slice, 2-5
SLOW, 11-9, 11-46, 11-47
SlowMotion(). See Macro commands
SOJ, 5-7
source input format. See SIF, 6-2
speed argument, 11-9, 11-46
Status signals, 3-7
STEP state, 11-45
STILL state, 11-15, 11-25, 11-45, 13-5
Synchronization,
display time stamps, 9-8
frame rate conversion, 15-1
from the CL450 and VSYNC, 13-7
from the CL450's SCR, 13-6
horizontal, 6-6
of audio and video, 2-11
system clock references, 9-8
using PTSs (presentation time stamps)
using SCRs (system clock references) 2-11
using semaphores, 14-5
vertical pixels, 6-5
video, 6-4
Synchronizion
to audio decoder 13- 13-6
to bitstream, 13-5
SysClkHigh, 8-17, 11-13
SysClkLow, 8-17, 11-13
SysClkMid, 8-17, 11-13
System clock
 encoding for, 2-11
 speed, 2-11
System clock divisor bits. See Div bits
System clock references. See SCRs
System decoder, 2-2
System layer, 2-2
System streams
 See Bitstream transfer process
System target decoder (STD), 11-22
System timer. See SCR counter

T
TAddr bits, 8-22
TData bits, 8-22
temporal reference, 13-7
TEST pin, 3-2, 3-7, 3-9
Test signal, 3-9
threshold argument, 12-22, 12-31
time stamps. See presentation time stamps
time_code, 13-7
timeStamp argument, 13-2
timestamp0, 11-12
timestamp1, 11-12
timestamp2, 11-12, 13-10
Timing
clock and control signal, 7-17
CMEM, 7-9

DRAM CAS and RAS, 7-14
DRAM restrictions, 14-8
GCLK (global clock), 3-8, 7-17
HCLK (host clock), 3-9
Local DRAM, 7-4
 bus, 7-14
 CAS and RAS, 7-15
 register read, 7-4
 register write, 7-6
 Reset, 7-17
 SCLK Input, 7-17
 signals, 3-7
SCLK (system clock), 3-8
 vectored interrupt w/ auto clear, 7-12
video bus input, 7-18
TMEM, 1-7, 8-18, 10-4, 10-5
 access registers, 8-22, 10-4
 address bits. See TAddr bits, 8-22
data bits. See TData bits, 8-22
topBorder, 11-31, 11-32, 12-3, 12-4, 15-6
Transform coding. See intra-picture cod­
ing, 2-10
Troubleshooting, D-1

U
UCAS, 3-2, 3-11, 5-3, 5-5, 5-6, 5-7
UCASIN, 3-2, 3-11, 5-3, 5-4, 5-5, 5-6, 5-7
UDS, 1-6, 3-2, 3-3, 3-4, 4-2, 4-3, 4-5, 4-8,
 4-9, 4-13, 4-15, 4-21, 5-4
UDS/LDS 4-5, 4-7, 4-11, 4-18, 7-4, 7-5, 7-12
UND interrupt. See Interrupts (listed)
Unpacketed data transfer. See data transfer
 (unpacketed)
Upper column address strobe. See UCAS
Upper Data Latch Enable. See UCASIN

V
VCC, 4-19, 5-7
VCLK, 3-2, 3-8, 3-12, 3-13, 6-3, 6-4, 6-7,
 6-8, 6-9, 7-18, 7-19
VCLK. See Video clock, 3-13
Vectored interrupt
    with automatic interrupt clearing 4-20
Vectored interrupt enable. See VIE
Vertical blanking, 6-2
Vertical synchronization. See VSYNC
Vertical timing (synchronization), 6-5
VERTICAL_SIZE, 14-3
vertical_size, 10-7, 11-44
VID_chroma. See registers (direct)
VID_control. See registers (direct)
VID_regdata. See registers (direct)
VID_selective. See registers (indirect)
VID_sela. See registers (indirect)
VID_selaux. See registers (indirect)
VID_selb. See registers (indirect)
VID_selbore. See registers (indirect)
VID_selGB. See registers (indirect)
VID_selmode. See registers (indirect)
VID_selR. See registers (indirect)
VID_y. See registers (direct)
Video
    bus timing, 7-18
    field example, 6-5
    multiplexing outputs, 6-10
    synchronization, 6-4
Video clock, 3-13
Video decoder, 2-2
Video display interface, 6-1 to 6-10
Video display unit, 6-3
Video interface, 3-10, 8-23
Video interface signals
    HSYNC (horizontal synchronization) 3-13
    P[23:0] 3-1, 3-2, 3-12, 6-6, 6-7, 6-8, 6-10, 7-18, 6-4, 6-10
    VCLK (video clock), 3-13
    VOE (video output enable), 3-13
    VSYNC (vertical synchronization), 3-13
Video output enable. See VOE
Video refresher, 6-2
Video register data bits. See VRData bits
Video register ID bits. See VRID bits
Video register IDs, 8-25
Video registers
direct, 8-24 to 8-25
indirect, 8-26 to 8-29
Video sequence, 2-4
Video stream
    block, 2-5
    composition, 2-6, 2-7
    defined, 2-3
    display order, 2-7
    group of pictures, 2-4
    macroblock, 2-5
    picture, 2-4
    slice, 2-5
    video sequence, 2-4
Video streams
    See bitstream data transfer
VIE bit, 3-7, 4-6, 8-11, 8-12
VLC Decoder, 11-16, 11-19
Vld bit, 11-20, 11-22, 13-2, 13-10
VMEM, 8-26
VOE signal, 3-12, 3-13, 6-4, 6-10, 7-18
VRData bits, 8-25
VRID bit field, 8-4, 8-25
VSE bit, 8-20
VSS bit, 8-19
VSYNC, 1-1, 3-1, 3-2, 3-12, 6-3, 6-4, 6-5, 6-6, 6-7, 6-8, 11-9, 11-28, 11-32, 11-43, 12-3, 12-5, 12-6, 12-8, 12-34, 12-37
control of top border, 6-5
synchronizing from, 13-7
timing restriction, 6-7
VSYNC frequency, 13-10
VSYNC interrupt 12-10, 12-14
VSYNC interrupt enable bit. See VSE bit
VSYNC interrupt status bit. See VSS bit
VWID bits, 8-28
W

WAdd bits, 8-21
WData bits, 8-21
WE, 3-2, 3-10, 3-11, 5-2, 5-3, 5-4, 5-5, 5-6, 5-8, 7-14
width argument, 11-42
Window blanking/unblanking, 11-7
Write address bits. See WAdd bits
Write counter bits. See CWCtr bits
Write enable. See WE
"Write per bit" DRAMs, 5-2
Write pointer, 11-7
Writing commands to the CL450, 8-15
Write data bits. See WData bits

X

xOffset argument, 6-8, 11-31, 11-42

Y

Y data bits. See YData bits
YCbCr, 1-2, 3-11, 6-1, 6-8, 11-34
conversion coefficients, 8-26
data, 6-3
display mode, 6-9
format, 3-12
YData bits, 8-26
yOffset argument, 11-15, 11-31, 11-42, 12-4, 12-5

Z

ZIP packages, 5-7
C-Cube Microsystems is always working to improve the quality of our documentation. If you have comments or suggestions about this document, please send us a marked-up copy of the page or pages or send us an e-mail message. We will acknowledge all comments received. Our address is:

Technical Publications Department
C-Cube Microsystems
1778 McCarthy Boulevard
Milpitas, CA 95035

phone: (408) 944-6300
fax: (408) 944-6314

e-mail: techpubs@c-cube.com
## North American Representatives

<table>
<thead>
<tr>
<th>State</th>
<th>Name</th>
<th>Address</th>
<th>Phone</th>
<th>Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>M2I</td>
<td>1910 Sparkman Avenue, Huntsville, AL 35816</td>
<td>205-830-0498</td>
<td>205-837-7049</td>
</tr>
<tr>
<td>Arkansas</td>
<td>TL Marketing</td>
<td>14850 Quorum Dr., Suite 100, Dallas, TX 75240</td>
<td>214-490-9300</td>
<td>214-960-6075</td>
</tr>
<tr>
<td>California</td>
<td>Bager Electronics</td>
<td>519 Encinitas Blvd, Encinitas, CA 92024</td>
<td>619-632-8816</td>
<td>619-632-8810</td>
</tr>
<tr>
<td></td>
<td>Bager Electronics</td>
<td>17220 Newhope Street, Suite 209, Fountain Valley, CA 92708</td>
<td>714-957-3367</td>
<td>714-546-2654</td>
</tr>
<tr>
<td></td>
<td>Norcomp</td>
<td>2140 Professional Drive, #200, Roseville, CA 95661</td>
<td>916-782-8070</td>
<td>916-782-8073</td>
</tr>
<tr>
<td></td>
<td>Norcomp</td>
<td>1267 Oakmead Parkway, Sunnyvale, CA 94086</td>
<td>408-733-7707</td>
<td>408-774-1947</td>
</tr>
<tr>
<td>Colorado</td>
<td>Promotech Sales, Inc.</td>
<td>2901 S. Colorado Blvd, Denver, CO 80222</td>
<td>303-692-8484</td>
<td>303-692-8416</td>
</tr>
<tr>
<td>Georgia</td>
<td>M2I</td>
<td>3000 Northwoods Parkway #110, Norcross, GA 30071</td>
<td>404-447-6124</td>
<td>404-447-0422</td>
</tr>
<tr>
<td>Hawaii</td>
<td>Bager Electronics</td>
<td>17220 Newhope Street, Suite 209, Fountain Valley, CA 92708</td>
<td>714-957-3367</td>
<td>714-546-2654</td>
</tr>
<tr>
<td>Illinois</td>
<td>Beta Technology Sales, Inc.</td>
<td>1009 Hawthorn Drive, Itasca, IL 60143</td>
<td>708-250-9586</td>
<td>708-250-9592</td>
</tr>
<tr>
<td>Iowa</td>
<td>Cahill, Schmitz &amp; Howe</td>
<td>226 Sussex Drive, N.E., Cedar Rapids, IA 52402</td>
<td>319-377-8219</td>
<td>319-377-0958</td>
</tr>
<tr>
<td>Louisiana</td>
<td>TL Marketing</td>
<td>14850 Quorum Dr., Suite 100, Dallas, TX 75240</td>
<td>214-490-9300</td>
<td>214-960-6075</td>
</tr>
<tr>
<td>Massachusetts</td>
<td>Advanced Technical Services</td>
<td>348 Park Street, Suite 102, North Reading, MA 01864</td>
<td>508-664-0888</td>
<td>508-664-5503</td>
</tr>
<tr>
<td>Minnesota</td>
<td>Cahill, Schmitz &amp; Cahill, Inc.</td>
<td>315 N. Pierce Street, St. Paul, MN 55104</td>
<td>612-646-7217</td>
<td>612-646-4484</td>
</tr>
<tr>
<td>Mississippi</td>
<td>M2I</td>
<td>1910 Sparkman Avenue, Huntsville, AL 35816</td>
<td>205-830-0498</td>
<td>205-837-7049</td>
</tr>
<tr>
<td>Montana</td>
<td>Promotech Sales, Inc.</td>
<td>2901 S. Colorado Blvd, Denver, CO 80222</td>
<td>303-692-8484</td>
<td>303-692-8416</td>
</tr>
<tr>
<td>Nevada</td>
<td>Clark County Only</td>
<td>Bager Electronics</td>
<td>6324 Varie!, Suite 314, Woodland Hills, CA 91367</td>
<td>818-712-0011</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>Norcomp, Inc.</td>
<td>2140 Professional Drive, #200, Roseville, CA 95661</td>
<td>916-782-8070</td>
</tr>
<tr>
<td>New Jersey</td>
<td>Parallax</td>
<td>734 Walt Whitman Road, Melville, NY 11747</td>
<td>516-351-1000</td>
<td>516-351-1606</td>
</tr>
</tbody>
</table>
# North American Representatives (cont.)

<table>
<thead>
<tr>
<th>State</th>
<th>Contact Information</th>
</tr>
</thead>
</table>
| **New York**| Empire Technical Associates  
              29 Fennell Street, Suite A  
              Skaneateles, NY 13152  
              phone: 315-685-5703  
              fax: 315-685-5979  
              Empire Technical Associates  
              349 W. Commercial Street  
              Suite 2920  
              East Rochester, NY 14445  
              phone: 716-381-8500  
              fax: 716-381-0911  
              Parallax  
              734 Walt Whitman Road  
              Melville, NY 11747  
              phone: 516-351-1000  
              fax: 516-351-1606 |
| **North Carolina**| M2I  
              1200 Trinity Road  
              Raleigh, NC 27607  
              phone: 919-851-0010  
              fax: 919-851-6620 |
| **North Dakota**| Cahill, Schmitz & Cahill, Inc.  
              315 N. Pierce Street  
              St. Paul, MN 55104  
              phone: 612-646-7217  
              fax: 612-646-4484 |
| **Oklahoma**| TL Marketing  
              14850 Quorum Dr., Suite 100  
              Dallas, TX 75240  
              phone: 214-490-9300  
              fax: 214-960-6075  
              TL Marketing  
              8100 Shoal Creek, Suite 250  
              Austin, TX 78758  
              phone: 512-371-7272  
              fax: 512-371-0727  
              TL Marketing  
              14343 Tory Chase Blvd. Suite 1  
              Houston, TX 77014  
              phone: 713-587-8100  
              fax: 713-580-7517 |
| **South Carolina**| M2I  
              1200 Trinity Road  
              Raleigh, NC 27607  
              phone: 919-851-0010  
              fax: 919-851-6620 |
| **South Dakota**| Cahill, Schmitz & Cahill, Inc.  
              315 N. Pierce Street  
              St. Paul, MN 55104  
              phone: 612-646-7217  
              fax: 612-646-4484  
              Tennessee  
              M2I  
              3000 Northwoods Parkway #110  
              Norcross, GA 30071  
              phone: 404-447-6124  
              fax: 404-447-0422  
              Texas  
              TL Marketing  
              14850 Quorum Dr., Suite 100  
              Dallas, TX 75240  
              phone: 214-490-9300  
              fax: 214-960-6075  
              TL Marketing  
              8100 Shoal Creek, Suite 250  
              Austin, TX 78758  
              phone: 512-371-7272  
              fax: 512-371-0727  
              TL Marketing  
              14343 Tory Chase Blvd. Suite 1  
              Houston, TX 77014  
              phone: 713-587-8100  
              fax: 713-580-7517 |
| **Wisconsin**| Western  
              Cahill, Schmitz & Cahill, Inc.  
              315 N. Pierce Street  
              St. Paul, MN 55104  
              phone: 612-646-7217  
              fax: 612-646-4484  
              Eastern  
              Beta Technology Sales, Inc.  
              9401 W. Beloit Road, Suite 409  
              Milwaukee, WI 53227  
              phone: 414-543-6609  
              fax: 414-543-9288 |
| **Canada**| Electrosource  
              6875 Royal Oak  
              Burnaby, BC  
              phone: 604-435-2533  
              fax: 604-435-2538  
              Electrosource  
              230 Galaxy Blvd.  
              Rexdale, ONT  
              Canada M9W 5R8  
              phone: 416-675-4490  
              fax: 416-675-6871  
              Electrosource  
              340 March Road, Suite 503  
              Kanata, ONT  
              Canada K2K 2E4  
              phone: 613-592-3214  
              fax: 613-592-4256  
              Electrosource  
              6600 Trans Canada Highway, Suite 420  
              Pointe Claire, Quebec  
              H9R 4S2  
              phone: 514-630-7486  
              fax: 514-630-7421 |
| **Others (Not Listed)**| Contact nearest office of C-Cube Microsystems |

C-Cube Microsystems
# International Representatives and Distributors

<table>
<thead>
<tr>
<th>Country</th>
<th>Representative/Distributor</th>
<th>Address</th>
<th>Phone</th>
<th>Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>NEWTEK (Rep/Dist.)</td>
<td>8, rue de l'Esterel, SILIC 583, 94663 Rungis Cedex</td>
<td>(33) 1-46.87.22.00</td>
<td>(33) 1-46.87.80.49</td>
</tr>
<tr>
<td>Hong Kong</td>
<td>MEMEC Asia Pacific (Rep/Dist.)</td>
<td>Unit No 2520-2525, Tower 1, Metroplaza, Hing Fong Road, Kwai Fong, N.T., phone: (852) 410-2780, fax: (852) 418-1600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singapore</td>
<td>Serial System Marketing (Rep/Dist.)</td>
<td>11 Jalan Mesin, Standard Industrial Bldg, #06-00, Singapore 1336, phone: (65) 280-0200, fax: (65) 286-6723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>United Kingdom</td>
<td>Kudos Thame Ltd. (Rep/Dist.)</td>
<td>55 Suttons Park, London Rd., Reading, BERKS RG6 1AZ, phone: (44) 734-351010, fax: (44) 734-351030</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>Metronik GmbH (Rep/Dist.)</td>
<td>Leonhardsweg 2, 8025 Unterhaching, phone: (49) 89-61108-0, fax: (49) 89-6116858</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Australia</td>
<td>ZATEK Components Pty Ltd.  (Rep/Dist.)</td>
<td>Suite 8, 1059 Victoria Rd., West Ryde 2114, Sydney, phone: (61) 2-874-0122, fax: (61) 2-874-6171</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Japan</td>
<td>MEMEC Asia Pacific (Rep/Dist.)</td>
<td>4th Floor, Jae Woong Bldg 176-11 Nonhyun-Dong, Kangnam-ku, Seoul, phone: (82) 2-518-8181, fax: (82) 2-518-9419</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Republic of China</td>
<td>MEMEC Asia Pacific (Rep)</td>
<td>14F-1, 171 Section 58, Min Sheng East Road, Hai Hwa Building, Taipei, Taiwan, R.O.C., phone: (886) 2 760-2028, fax: (886) 2 765-1488</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

# C-Cube Microsystems Sales Offices

<table>
<thead>
<tr>
<th>Office Type</th>
<th>Address</th>
<th>Phone</th>
<th>Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home Office</td>
<td>C-Cube Microsystems, 1778 McCarthy Boulevard, Milpitas, CA 95035</td>
<td>408-944-6300</td>
<td>408-944-6314</td>
</tr>
<tr>
<td>Eastern Area Office</td>
<td>C-Cube Microsystems, One Kendall Square, Suite 220, Cambridge, MA 02139</td>
<td>617-621-7180</td>
<td>617-621-7179</td>
</tr>
<tr>
<td>Southwestern Area Office</td>
<td>C-Cube Microsystems, 453 Bristol Avenue, Cardiff, CA 92007</td>
<td>619-632-0864</td>
<td>619-632-0864</td>
</tr>
<tr>
<td>European Office</td>
<td>C-Cube Microsystems, 44 Dartford Road, Sevenoaks, Kent, UK TN 133 TQ</td>
<td>(44) 732 743 256</td>
<td>(44) 732 450 151</td>
</tr>
</tbody>
</table>